Emanuela Cerchez (n. 12.04.1968, la Iaşi) este absolventă a Facultăţii de Matematică, secţia Informatică (1990), şi a Seminarului pedagogic postuniversitar (1997), profesoară de informatică (grad didactic I), membră în Comisia Naţională de Informatică. Autoarea a mai publicat la Editura Polirom: *Internet. Manual pentru liceu* (2000, avizat MEN), *Informatica. Manual pentru clasa a X-a* (coautor Marinel-Paul Şerban, 2000, avizat MEN), *PC. Pas cu pas* (coautor Marinel-Paul Şerban, 2001), *Informatica pentru gimnaziu* (coautor Marinel-Paul Şerban, 2002, avizat MEC), *Informatica. Culegere de probleme pentru liceu* (2002).

Marinel-Paul Şerban (n. 29.06.1950, la Arad) este absolvent al Facultăţii de Matematică-Mecanică, Universitatea din Timişoara (1973), specializare postuniversitară în informatică (1974), profesor de informatică (grad didactic I), membru în Comisia Naţională de Informatică. De acelaşi autor, la Editura Polirom au apărut: *Informatica. Manual pentru clasa a X-a* (coautoare Emanuela Cerchez, 2000, avizat MEN), *PC. Pas cu pas* (coautoare Emanuela Cerchez, 2001), *Informatica pentru gimnaziu* (coautoare Emanuela Cerchez, 2002, avizat MEC).
Autorii au o bogată experienţă în pregătirea de performanţă a elevilor: susţin cursuri la Centrul de Pregătire a Tinerilor Capabili de Performanţă din Iaşi, desfăşoară probleme pentru olimpiadele şi concursurile naţionale şi judeţene de informatică, susţinând activitatea de pregătire a lotului naţional de informatică, precum şi programul de pregătire de performanţă în informatică *.campion.*

www.polirom.ro

Editura Polirom
Iaşi, B-dul Carol I nr. 4, P.O. BOX 266, 700506
Bucureşti, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33; O.P. 37, P.O. BOX 1-728, 030174

Descrierea CIP a Bibliotecii Naţionale a României:

CERCHEZ, EMANUELA
Programarea în limbajul C/C++ / Emanuela Cerchez, Marinel Şerban. - Iaşi: Polirom, 2005
ISBN: 937-681-868-3

I. Şerban, Marinel

004.43 C
004.43 C++

Printed in ROMANIA

Emanuela Cerchez, Marinel Şerban

Programarea în limbajul C/C++ pentru liceu

POLIROM
2005

Cuprins

1. Elemente de bază ale limbajului C/C++

1.1. *Noțiuni introductive*

Un limbaj de programare reprezintă un mijloc de comunicare între programator și calculator. Pentru a defini cu rigurozitate un limbaj de programare, trebuie să descriem trei aspecte ale limbajului:

- *sintaxa* – reprezintă totalitatea regulilor pe baza cărora se obțin elementele limbajului;
- *semantica* – definește semnificația construcțiilor sintactic corecte (a elementelor limbajului);
- *pragmatica* – definește modul de utilizare a elementelor limbajului.

Observați că aceste trei aspecte se studiază și în cazul unui limbaj natural (de exemplu, limba română). Dar spre deosebire de limba română, unde, dacă facem o greșeală gramaticală sau utilizăm greșit o expresie într-un context, sunt șanse destul de mari ca interlocutorul să ne înțeleagă, atunci când comunicăm cu un calculator, trebuie să respectăm cu exactitate regulile de comunicare. Calculatorul nu face presupuneri, nu ghicește ceea ce ați fi dorit să-i spuneți. Dacă nu vă „exprimați" corect, nu veți obține de la calculator decât mesaje de eroare.

Descrierea unui algoritm într-un limbaj de programare se numește *program*.

Evoluția limbajelor de programare

Calculatorul, la nivel intim, nu „cunoaște" decât un singur mod de comunicare – limbajul procesorului cu care este dotat, denumit *cod-mașină*. Programarea în acest limbaj este dificilă și necesită cunoștințe detaliate despre procesorul respectiv. Ca urmare, s-au dezvoltat mai întâi limbajele de asamblare (care necesitau în continuare cunoștințe legate de procesorul calculatorului, dar erau mai ușor de utilizat) și apoi limbaje de programare de nivel înalt (care nu necesită cunoștințe detaliate referitoare la structura calculatorului pe care se va executa programul și utilizează notații asemănătoare limbajului matematic sau limbajului natural).

De-a lungul timpului s-au dezvoltat extrem de multe limbaje de programare, dar puține dintre acestea au reușit să se impună în timp și ca arie de utilizare. Să menționăm câteva repere istorice în evoluția limbajelor de programare:

– 1955 – limbajul **FORTRAN** (**FOR**mula **TRAN**slation), destinat aplicațiilor tehnico-științifice cu caracter numeric;

- 1960 – limbajul **ALGOL** (**ALGO**rithmic **L**anguage), primul limbaj definit riguros, cu o sintaxă complet formalizată; conceptele introduse de colectivul coordonat de *Peter Naur* sunt utilizate şi azi de proiectanţii de limbaje de programare;
- 1960 – prima versiune a limbajului **COBOL** (**CO**mmon **B**usiness **O**riented **L**anguage), destinat aplicaţiilor economice;
- 1971 – *Niklaus Wirth* a conceput un limbaj care să-i ajute pe studenţi să-şi însuşească rapid şi mai ales corect principiile „artei programării"; în onoarea matematicianului francez *Blaise Pascal*, el a numit acest limbaj **PASCAL**;
- 1972 – *Brian Kernighan* şi *Dennis Ritchie* au conceput un limbaj cu destinaţie universală, denumit **C**;
- 1980 – *Bjarne Stroustrup* publică specificaţiile limbajului **C++**, o extensie a limbajului C destinată programării orientate pe obiect;
- 1995 – *James Gostling* publică specificaţiile limbajului *Java*, un limbaj orientat pe obiect, cu sintaxă şi principii asemănătoare cu ale limbajului C++. Java are ca prim obiectiv portabilitatea (el este independent de maşina pe care lucrează).

Desigur, sunt numai câteva dintre reperele istorice ale dezvoltării limbajelor de programare. Permanent se dezvoltă limbaje de programare specifice diferitelor domenii de programare (de exemplu, limbaje destinate inteligenţei artificiale, limbaje grafice, limbaje de programare pentru Internet etc.).

În această carte vom studia limbajul de programare C, precizând eventualele diferenţe faţă de limbajul C++. Am ales acest limbaj datorită faptului că ocupă o poziţie importantă în ofertele pentru locuri de muncă, dar şi pentru că poate fi învăţat gradual de începătorii în programare. Prin intermediul limbajului C/C++, ne vom însuşi sistematic principiile programării structurate şi, ulterior, principiile programării orientate pe obiect.

Pentru a specifica faptul că un element al limbajului este disponibil atât în C, cât şi în C++, vom nota C/C++.

1.2. *Setul de caractere*

Setul de caractere utilizat pentru scrierea programelor C/C++ este setul de caractere al codului ASCII[1].

1.3. *Identificatori*

Identificatorii, întâlniţi şi sub denumirea *nume*, au rolul de a denumi elemente ale programului: constante, variabile, funcţii etc.

Din punct de vedere sintactic, un identificator este constituit dintr-o succesiune de litere, cifre sau caracterul '_' (liniuţa de subliniere – *underscore*), primul caracter fiind obligatoriu literă sau liniuţa de subliniere.

1. Vezi anexa 1.

Exemple

Corecte:	Incorecte:	
`Program`	`Nume Prenume`	(conţine caracterul spaţiu)
`Nume_Prenume_elev1`	`a+b`	(conţine caracterul +)
`_unu`	`2b`	(începe cu o cifră)

Observaţii

1. Un identificator poate avea orice lungime, dar sunt luate în consideraţie numai primele 31 de caractere.
2. Atenţie! Limbajul C/C++ este *case-sensitive*, adică face diferenţa între literele mici şi literele mari. Prin urmare, identificatorii `NMx` şi `Nmx` sunt diferiţi!
3. Este recomandat să utilizati identificatori sugestivi, astfel încât sa nu fie necesar, pe cât posibil, să apelaţi la declaratii sau comentarii pentru a înţelege scopul în care este folosit un identificator. Este util sa construim identificatorii prin compunerea mai multor cuvinte sau rădăcini ale acestora. În acest caz, iniţiala fiecărui cuvânt care intră în compunerea identificatorului poate fi majuscula (de exemplu, `NrMaxCuvinte`, `NrTelefon` etc).

1.4. *Cuvinte rezervate*

Cuvintele rezervate, denumite uneori şi *cuvinte-cheie* (*keywords*), sunt identificatori speciali, cu înţeles predefinit, care pot fi utilizaţi numai în construcţiile sintactice în care sunt specificaţi.

Exemple

`if` – cuvânt-cheie care descrie începutul unei instrucţiuni alternative;
`while` – cuvânt-cheie care descrie începutul unei instrucţiuni repetitive.

Observaţii

1. În limbajul C/C++, toate cuvintele rezervate[2] se scriu numai cu litere mici.
2. În programele pe care le vom prezenta în carte, vom scrie îngroşat cuvintele rezervate.

1.5. *Comentarii*

Comentariile sunt texte care vor fi ignorate de compilator[3], dar au rolul de a explicita pentru programatori anumite secvenţe de program.

2. Lista completă a cuvintelor rezervate ale limbajului C/C++ se găseşte în anexa 2.
3. Compilatorul este un program care are rolul de a traduce textul programului scris în limbajul de programare (numit cod sursă) în limbaj-maşină, obţinând cod-obiect.

Din punct de vedere sintactic, un comentariu este:
- o succesiune de caractere încadrată între `/*` şi `*/`; aceste comentarii pot fi formate din mai multe linii;
- o succesiune de caractere care începe cu `//` şi se termină la sfârşitul liniei.

Exemple

```
/* acesta este un comentariu care poate fi scris pe mai
multe linii */
// acest comentariu se termina la sfarsitul liniei
/*Legea lui Schyer: Daca instructiunile programului si
comentariile sunt in dezacord, atunci probabil amandoua sunt
eronate. */
```

Observaţie

În activitatea de programare, documentarea programelor este esenţială. Documentarea unui produs *software* se face în două forme: documentarea externă (manual de referinţă, rapoarte, articole etc) şi documentarea internă (comentarii inserate în program). Pentru documentarea internă a programelor, este indicat ca fiecare secvenţă de program să fie însoţită de un comentariu referitor la scopul acesteia. De asemenea, tipurile de date şi variabilele folosite trebuie să fie însoţite de comentarii referitoare la semnificaţia lor. Cu cât un program este mai detaliat comentat, cu atât este mai uşor de citit, de înţeles, de modificat, de întreţinut atât de către programatorul însuşi, cât şi de colaboratorii acestuia.

1.6. *Separatori*

Separatorii au rolul de a separa unităţile sintactice.

Ca separatori „universali" se utilizează caracterele albe (spaţiu `' '`, TAB `'\t'`, sfârşit de linie[4] – *newline* – `'\n'`) şi comentariile.

Unele construcţii sintactice utilizează şi *separatori specifici* (de exemplu, într-o declaraţie de variabile, variabilele sunt separate prin caracterul virgulă `','`) sau *delimitatori* (de exemplu, caracterul `';'` delimitează sfârşitul unei instrucţiuni sau al unei declaraţii; caracterul apostrof delimitează o constantă caracter, ghilimelele delimitează constantele şir de caractere).

4. Sfârşitul de linie este marcat prin acţionarea tastei ENTER, care (dacă lucrăm sub sistemul de operare Windows) produce combinaţia de caractere CR+LF (CR – Carriage Return – poziţionare la începutul rândului, cod ASCII 13; LF – Line Feed – trecerea la rândul următor, cod ASCII 10). Dacă lucrăm sub sistemul de operare Linux, marcajului de sfârşit de linie (caracterul *newline*) îi corespunde numai codul ASCII 10.

1.7. *Structura generală a unui program C/C++*

Privit în ansamblu, un program C/C++ este constituit dintr-o succesiune de module, denumite *funcţii*. Una dintre aceste funcţii este funcţia principală, denumită `main()`. Aceasta este o funcţie specială, care trebuie să apară obligatoriu o singură dată în orice program C/C++, deoarece execuţia oricărui program începe cu funcţia `main()`.

Pentru început, vom elabora programe constituite numai din funcţia `main()`, urmând ca ulterior să elaborăm propriile noastre funcţii.

Cel mai simplu program C/C++ arată astfel:

```
void main()
{ }
```

Evident, acest program nu face nimic! Observaţi că el este constituit numai din funcţia `main()`.

Definirea oricărei funcţii este constituită din *antetul funcţiei* şi *corpul funcţiei*. *Antetul* funcţiei conţine numele funcţiei, tipul rezultatului pe care îl calculează funcţia şi o listă de parametri prin care funcţia comunică cu exteriorul ei, încadrată între paranteze rotunde:

```
tip_rezultat nume(lista_parametri)
```

Observaţi că în cazul funcţiei `main()` pe care am descris-o mai sus, lista parametrilor este vidă, iar funcţia nu întoarce nici un rezultat (acest lucru este indicat prin specificarea tipului `void`).

Rezultatul returnat de funcţia `main()` este preluat de sistemul de operare şi de obicei oferă indicaţii despre modul de funcţionare a programului. Când execuţia unui program se termină cu succes, în mod uzual, programul returnează la încheierea execuţiei sale valoarea `0`.

Cel mai simplu program C/C++ care nu face nimic, dar se termină cu succes, arată astfel:

```
int main()
{return 0;}
```

Instrucţiunea `return` este utilizată pentru a încheia execuţia unei funcţii şi a returna valoarea expresiei specificate în instrucţiunea `return` ca valoare a funcţiei. Dacă funcţia este `main()`, se încheie o dată cu funcţia `main()` şi execuţia programului.

Formatul general al instrucţiunii `return` este:

```
return expresie;
```

În caz că apar erori pe parcursul execuţiei programului (de exemplu, utilizatorul programului a introdus valori negative pentru o variabilă număr natural sau un şir de caractere acolo unde ar fi trebuit un număr), programul trebuie să se termine returnând sistemului de operare un cod de eroare diferit de `0`, de obicei specific erorii care a

apărut pe parcursul execuției programului. De exemplu, dacă pentru vârsta unui pacient a fost introdus un număr care nu se încadrează în intervalul [0, 150], prin convenție programul ar putea returna codul de eroare 1.

Corpul funcției este încadrat între acolade. În corpul funcției se vor scrie declarațiile și instrucțiunile care trebuie să fie executate în această funcție. În exemplul nostru, corpul funcției este vid, deci funcția nu face nimic.

1.8. *Tipuri de date standard*

Prin *date* înțelegem, în general, tot ceea ce este prelucrat de un calculator. Fiecare dată are un anumit *tip*.

Un *tip de date* definește mulțimea valorilor pe care le pot lua datele de tipul respectiv, modul de reprezentare a acestora în memorie, precum și operațiile care se pot efectua cu datele respective.

Orice limbaj de programare dispune de un set de tipuri de date predefinite, denumite și *tipuri de date standard*[5].

Mulțimea valorilor unui anumit tip de date reprezintă *constantele* tipului respectiv.

Tipul de date `int`

Valorile datelor de tip `int` sunt numere întregi, cuprinse în intervalul `[-32768,32767]`, reprezentate în memorie pe 2 octeți[6] (*bytes*), în cod complementar[7]. Tipul de date `int` suportă modificatorii de tip `unsigned` (datele sunt numere naturale) și `long` (modifică dimensiunea reprezentării). Se obțin astfel următoarele tipuri de date întregi:

Tip	Valori	Reprezentare
`int`	`[-32768,32767]`	2 octeți, cu semn
`unsigned int`	`[0,65535]`	2 octeți, fără semn
`long int`	`[-2147483648, 2147483647]`	4 octeți, cu semn
`unsigned long int`	`[0, 4294967295]`	4 octeți, fără semn

5. Tipurile de date predefinite depind de implementarea limbajului pe care o utilizați. În carte vom prezenta implementarea Borland C++ 3.1. Dacă veți utiliza alt mediu de programare (de exemplu, GNU C pentru Linux), veți observa că există și alte tipuri de date predefinite, precum și diferențe de implementare pentru tipurile de date prezentate în carte. De exemplu, în GNU C există și tipul long long int pentru reprezentarea numerelor întregi pe 64 biți, iar tipul int este reprezentat pe 4 octeți, în loc de 2 octeți.
6. În anexa 4 găsiți informații despre organizarea logică a memoriei.
7. În anexa 5 găsiți informații despre modalitățile de reprezentare a datelor în memorie.

Atunci când un tip de date nu este precizat, implicit este considerat `int`. Prin urmare, putem specifica numai modificatorul `long`, sau `unsigned`, sau `unsigned long`, tipul fiind implicit `int`.

Constantele întregi sunt numere întregi din intervalul corespunzător tipului. Ele pot fi precizate în baza `10` (folosind notația uzuală), în baza[8] `8` (în acest caz, constanta este precedată de un `0` nesemnificativ) sau în baza `16` (caz în care constanta are prefixul `0x` sau `0X`).

Exemple

`123`	constantă zecimală de tip `int`
`-12345678`	constantă zecimală de tip `long int`
`01234`	constantă octală
`0x1a0`	constantă hexazecimală
`0XFFFF`	constantă hexazecimală

Tipul constantei este determinat implicit de valoarea ei (`int` dacă se găsește în intervalul `[-32768,32767]`, `long int` dacă depășește intervalul tipului `int`, `unsigned long int` în rest). Dacă dorim să specificăm explicit tipul unei constante, putem să utilizăm un sufix corespunzător tipului dorit:

Sufix	Tip constantă
`u, U`	`unsigned int`
`l, L`	`long int`
`ul, Ul, uL, UL`	`unsigned long int`

Exemple

`1` constanta 1, reprezentată pe 2 octeți, cu semn (`int`)
`1u` constanta 1, reprezentată pe 2 octeți, fără semn (`unsigned int`)
`1L` constanta 1, reprezentată pe 4 octeți, cu semn (`long int`)
`1ul` constanta 1, reprezentată pe 4 octeți, fără semn (`unsigned long int`)

Tipul `char`

Tipul `char` este, de asemenea, un tip întreg, care se reprezintă pe un octet, cu semn. Acest tip suportă un singur modificator – `unsigned`:

Tip	Valori	Reprezentare
`char`	`[-128,127]`	1 octet, cu semn
`unsigned char`	`[0,255]`	1 octet, fără semn

8. În anexa 3 găsiți informații despre baze de numerație și conversii între baze de numerație.

Constantele de tip `char` (`unsigned char`) pot fi numere întregi din intervalul specificat sau caracterele care au codurile ASCII în intervalul specificat. Valorile de tip `char` par a avea o natură dublă – caractere ASCII şi numere întregi. Pentru calculator nu este însă nimic ambiguu, deoarece el reţine orice caracter cu ajutorul valorii numerice asociate (codul său ASCII). De exemplu, caracterul `'A'` şi constanta 65 au pentru calculator aceeaşi semnificaţie.

Caracterele grafice (coduri ASCII de la `32` la `127`) se pot specifica încadrând caracterul respectiv între apostrofuri. De exemplu, `'a'`, `'9'`, `' '`, `'*'`.

Caracterele negrafice (dar şi cele grafice, dacă dorim) se pot specifica încadrând între apostrofuri o secvenţă de evitare (secvenţă *escape*). Secvenţele *escape* sunt formate din caracterul \ (*backslash*), urmat de codul ASCII al caracterului (exprimat în baza 8 sau exprimat în baza 16, precedat de `x`).

Exemple

Secvenţă *escape*	Caracter
`'\65'`	`'5'` (codul exprimat în baza 8)
`'\x35'`	`'5'` (codul exprimat în baza 16)
`'\5'`	caracterul ♣ (codul exprimat in baza 8)
`'\356'`	caracterul ∈ (codul exprimat in baza 8)

Unele caractere negrafice, mai des utilizate, au asociate secvenţe *escape* speciale, constituite din \ (*backslash*) şi o literă sugestivă:

Exemple

Secvenţă *escape*	Caracter
`'\b'`	Caracterul *backspace* (deplasează cursorul pe ecran cu o poziţie la stânga).
`'\t'`	Caracterul tab orizontal.
`'\n'`	Caracterul *newline* (determină trecerea cursorului la linie nouă).
`'\a'`	Caracterul *alarm* (generează un sunet).
`'\\'`	Caracterul *backslash*.
`'\''`	Caracterul apostrof.
`'\"'`	Caracterul ghilimele.

Constantele şir de caractere sunt constituite dintr-o succesiune de caractere încadrată între ghilimele.

Exemple

```
"Acesta este un sir."
"Prima linie \n A doua linie."
```


Tipuri reale

Tipurile reale ale limbajului C/C++ sunt `float` şi `double`. Tipul `double` acceptă şi modificatorul `long`:

Tip	Valori	Reprezentare
`float`	$[3.4*10^{-38}, 3.4*10^{+38}] \cup [-3.4*10^{+38}, -3.4*10^{-38}]$	4 octeţi, în virgulă mobilă
`double`	$[1.7*10^{-308}, 1.7*10^{+308}] \cup [-1.7*10^{+308}, -1.7*10^{-308}]$	8 octeţi, în virgulă mobilă
`long double`	$[3.4*10^{-4932}, 1.1*10^{+4932}] \cup [-3.4*10^{+4932}, -1.1*10^{-4932}]$	10 octeţi, în virgulă mobilă

Constantele reale care se pot reprezenta în memoria calculatorului sunt numere raţionale din intervalele specificate. Constantele reale pot fi specificate în notaţia uzuală sau în format exponenţial (ştiinţific). În forma uzuală se precizează partea întreagă şi partea zecimală a numărului, separate prin marca zecimală din sistemul englezesc (caracterul punct). În format exponenţial se poate preciza, în plus, şi un exponent al lui 10, precedat de litera `e` sau `E`. Valoarea numărului se obţine înmulţind numărul cu 10 la puterea specificată.

Exemple

```
-12.3
 3.14
 1.2e-10      (în format uzual 0.00000000012)   = 1.2 ·10^-10
 2.67E+8      (în format uzual 267000000)       = 2.67·10^8
```

Tipul `void`

Tipul `void` este un tip special, pentru care mulţimea valorilor este vidă. Acest tip se utilizează atunci când este necesar să specificăm absenţa oricărei valori (de exemplu, pentru tipul funcţiilor care nu întorc nici un rezultat, cum a fost cazul funcţiei `main`).

Observaţie

În limbajul C/C++ nu există un tip de date special pentru valori logice. Valoarea `0` este asociată valorii de adevăr `fals`, orice valoare diferită de `0` fiind asociată valorii de adevăr `adevarat`.

1.9. *Variabile*

O variabilă este o dată care îşi poate modifica valoarea pe parcursul execuţiei programului.

În limbajul C/C++, înainte de a utiliza o variabilă, trebuie să o declarăm. La declarare, trebuie să specificăm numele variabilei, tipul acesteia şi, eventual, o valoare iniţială pe care dorim să o atribuim variabilei.

Formatul general al unei declaraţii de variabile este:

```
tip nume_var1[=expresie1] [, nume_var2[=expresie2]...];
```

Observaţii

1. Prin `tip` specificăm tipul variabilelor care se declară.
2. Prin `nume_var1`, `nume_var2`, ... specificăm numele variabilelor care se declară (acestea sunt identificatori).
3. Se pot declara simultan mai multe variabile de acelaşi tip, separând numele lor prin virgulă.
4. La declarare, putem atribui variabilei o valoare iniţială, specificând după numele variabilei caracterul `'='` şi o expresie de iniţializare. Expresia trebuie să fie evaluabilă în momentul declarării.
5. Parantezele `[]` utilizate în descrierea formatului general au semnificaţia că elementul încadrat între paranteze este opţional (poate să apară sau nu într-o declaraţie de variabile).

Exemple

```
int a, b=3, c=2+4;
char z;
float x=b*2.5, y;
```

Am declarat trei variabile `a`, `b`, şi `c` de tip `int`, o variabilă `z` de tip `char` şi două variabile `x` şi `y` de tip `float`. Variabilei `b` i-am atribuit valoarea iniţială `3`, variabilei `c` i-am atribuit valoarea `6`, iar variabilei `x` i-am atribuit valoarea `7.5`. Variabilelor `a`, `y` şi `z` nu le-am atribuit nici o valoare iniţială la declarare.

Declaraţia unei variabile trebuie să preceadă orice referire la variabila respectivă şi poate fi plasată în interiorul unei funcţii (în cazul nostru, al funcţiei `main()`) sau în exteriorul oricărei funcţii (în cazul nostru, în exteriorul funcţiei `main()`). Dacă declaraţia este plasată în interiorul unei funcţii, variabila se numeşte *locală* funcţiei, altfel se numeşte *globală*.

Există diferenţe majore între variabilele locale şi cele globale, pe care le vom studia amănunţit ulterior, când vom şti să proiectăm propriile noastre funcţii. La nivel de iniţiere este util să ştim că *variabilele globale sunt automat iniţializate cu 0*. Cele locale nu sunt iniţializate.

Referitor la poziţia declaraţiilor de variabile în cadrul unei funcţii, există diferenţe între limbajul C şi limbajul C++. În limbajul C++ putem plasa declaraţii de variabile oriunde în corpul unei funcţii. În limbajul C, declaraţiile de variabile trebuie să fie plasate la începutul corpului funcţiei, înaintea oricărei instrucţiuni.

1.10. *Preprocesare*

Preprocesorul este un program lansat în execuţie automat înainte de compilare[9]. El execută toate directivele preprocesor incluse în program, efectuând substituţii de texte.

Toate directivele preprocesor încep cu caracterul `#`. De exemplu, `#include`, `#define`, `#if`, `#undef`, `#line` etc. Studiul acestor directive depăşeşte nivelul de iniţiere în programare, prin urmare, le vom discuta detaliat mai târziu. Dar pentru a elabora programe, trebuie să învăţăm directivele `#include` şi `#define`.

Directiva `#include` este utilizată pentru a include într-un program un fişier antet standard sau creat de utilizator.

Un *fişier antet (header)* conţine declaraţiile funcţiilor, constantelor, variabilelor şi tipurilor definite într-o bibliotecă.

Fişierul antet este specific bibliotecii[10] pe care dorim să o utilizăm şi are întotdeauna extensia `h`.

Pentru a include într-un program un fişier antet standard:

```
#include <nume_fisier_antet.h>
```

Exemple

```
#include <math.h>
```
Include fişierul antet al bibliotecii de funcţii matematice.

```
#include <iostream.h>
```
Include un fişier antet al bibliotecii limbajului C++ cu funcţii de intrare/ieşire.

```
#include <stdio.h>
```
Include un fişier antet al bibliotecii limbajului C cu funcţii de intrare/ieşire.

Pentru a include un fişier antet creat de utilizator, sintaxa este:

```
#include "nume_fisier_antet.h"
```

9. Pentru descrierea etapelor de execuţie a programelor, vezi anexa 6.
10. Fişierele antet sunt specifice mediului de programare pe care îl utilizaţi! De exemplu, dacă elaboraţi un program sub mediul de programare Borland C++ 3.1 şi doriţi să funcţioneze şi sub sistemul de operare Linux, cu compilatoarele GNU, trebuie să fiţi foarte atenţi, pentru că nu toate fişierele antet existente în Borland există şi pentru GNU sau ele nu au neapărat acelaşi conţinut.

Directiva `#define`, într-un format simplu, permite definirea unei constante simbolice:

```
#define identificator_constanta valoare
```

Ca efect, preprocesorul va substitui în program orice apariție a identificatorului de constantă cu valoarea acesteia.

Exemple

```
#define PI  3.1415
#define NrMaxElevi 35
```

Se recomandă utilizarea constantelor simbolice atunci când dorim să asociem o denumire mai sugestivă unei valori, ceea ce conduce la creșterea lizibilității programului. De asemenea, prin utilizarea constantelor simbolice, programul devine mai ușor de modificat. De exemplu, dacă am scris un program care să funcționeze pentru clase cu maximum 35 de elevi, atunci când numărul maxim de elevi în clasă se modifică, nu trebuie să parcurgem întreg programul și să schimbăm fiecare apariție a valorii modificate, este suficient să schimbăm valoarea constantei simbolice din definiția ei.

Observație

Fișierele antet conțin și definiții de constante simbolice. De exemplu, în fișierul antet `values.h` sunt definite constantele simbolice `MAXINT` (care are ca valoare `32767`, cel mai mare număr de tip `int`) și `MAXLONG` (care are ca valoare `2147483647`, cel mai mare număr de tip `long int`). În fișierul antet `conio.h` sunt definite constante simbolice pentru culori (de exemplu, `BLACK` cu valoarea `0`, `BLUE` cu valoarea `1`, `GREEN` cu valoarea `2`, `RED` cu valoarea `4` etc).

1.11. *Utilizarea funcțiilor din bibliotecile standard*

Setul de instrucțiuni al limbajului C/C++ este relativ restrâns, dar suficient pentru o implementare eficientă și concisă a algoritmilor.

Există unele operații care sunt frecvent utilizate (cum ar fi, de exemplu, citirea, scrierea, extragerea radicalului, calculul modulului, ștergerea ecranului etc) pentru care nu există instrucțiuni specifice. Ele pot fi implementate cu ajutorul instrucțiunilor existente în limbaj, dar... ar fi absurd să le implementăm de fiecare dată, în fiecare program pe care îl elaborăm. Din acest motiv, s-au constituit biblioteci de funcții, care conțin colecții de funcții de utilitate generală grupate pe categorii. Pentru a le utiliza, este suficient să includem în program fișierul antet al bibliotecii și să apelăm funcția care ne este necesară.

Pentru a utiliza funcțiile dintr-o bibliotecă trebuie să includem la începutul programului fișierul antet (*header*) care conține declarațiile funcțiilor din biblioteca respectivă.

Pentru a apela o funcție trebuie să cunoaștem numele și formatul (prototipul) acesteia. În biblioteci există sute și sute de funcții, este improbabil să le putem cunoaște pe toate. Sigur că, după un timp, le vom reține pe cele pe care le vom utiliza frecvent. În rest, singura noastră soluție este să apelăm la *Help* (sistemul de autodocumentare al mediului de programare). În *Help* vom găsi prototipul tuturor funcțiilor din bibliotecile standard și explicații despre modul de funcționare a acestora. Prototipul unei funcții ne informează despre numele funcției, tipul valorii calculate de funcție și despre parametrii funcției. Cu alte cuvinte, are structura unui antet de funcție. Parametrii unei funcții au o semnificație similară argumentelor funcțiilor învățate la matematică. Atunci când utilizăm o funcție (în limbaj informatic, o apelăm), trebuie să specificăm valorile efective ale parametrilor pentru care apelăm funcția. Formatul unui apel de funcție este:

```
nume_functie(lista_valori_parametri)
```

Valorile parametrilor de la apel trebuie să corespundă ca număr, ordine și tip cu parametrii specificați în prototipul funcției.

Exemple

1. În fișierul antet `math.h`, care conține funcții matematice, este declarată funcția de extragere a radicalului `sqrt()`. Prototipul funcției `sqrt()` este:

```
double sqrt(double x);
```

Efect: calculează radicalul valorii `x`, transmise ca parametru. De exemplu, dacă dorim să calculăm $\sqrt{3}$, apelăm `sqrt(3)`.

2. În fișierul antet `conio.h` este declarată o funcție care are ca efect ștergerea ferestrei curente (mai exact, fereastra curentă, care implicit coincide cu întreg ecranul, este colorată în culoarea de fundal – implicit negru):

```
void clrscr(void);
```

Dacă dorim să schimbăm culoarea de fundal, putem apela funcția `textbackground()`, declarată tot în `conio.h`.

```
void textbackground(int culoare);
```

De exemplu, pentru a schimba culoarea ecranului în verde, vom apela funcția `textbackground()` pentru a stabili culoarea de fundal, apoi funcția `clrscr()` pentru a colora ecranul în culoarea de fundal:

```
textbackground(GREEN);
clrscr();
```

1.12. *Citirea/scrierea datelor*

Prin *citirea datelor* vom înțelege operația prin care una sau mai multe variabile primesc valori prin introducerea lor de la tastatură sau prin extragerea lor de pe un suport de memorare extern[11].

Prin *scrierea datelor* vom înțelege operația prin care rezultatele obținute în urma prelucrării datelor de intrare sunt fie afișate pe ecranul monitorului, fie stocate pe un suport extern de memorare.

Aceste operații sunt denumite frecvent și operații de intrare/ieșire.

În acest capitol vom prezenta doar citirea datelor de la tastatură și afișarea datelor pe ecran.

În limbajul C/C++ nu există instrucțiuni specializate pentru citirea/scrierea datelor. Aceste operații se realizează prin intermediul unor funcții existente în bibliotecile[12] standard ale limbajului.

Din exemplele precedente se poate deduce că operațiile de intrare/ieșire diferă în limbajul C++ față de limbajul C. De data aceasta diferențele sunt majore, deoarece operațiile de intrare/ieșire din C++ sunt proiectate din perspectiva programării orientate pe obiect.

1.13. *Citiri și scrieri în limbajul C++*

Conceptul central în operațiile de intrare/ieșire în limbajul C++ este fluxul de intrare/ieșire (denumirea originală fiind *stream*). Simplificând, putem privi un *stream* ca pe o succesiune de caractere. Dacă *stream*-ul este de intrare, secvența de caractere „curge" dinspre exterior (în cazul nostru, dinspre tastatură) către memoria calculatorului. Dacă *stream*-ul este de ieșire, secvența de caractere „curge" dinspre memoria calculatorului către exterior (în cazul nostru, ecranul monitorului).

În fișierul antet `iostream.h` sunt declarate două fluxuri standard: un flux de intrare de la tastatură, denumit `cin` (*console input*) și un flux de ieșire către ecran, denumit `cout` (*console output*).

Când dorim să citim date de la tastatură, le vom extrage din fluxul de intrare, folosind operatorul de extragere `'>>'`:

```
cin >> nume_variabila;
```

11. Operațiile de citire/scriere a datelor de pe/pe suport extern se vor studia în capitolul „Fișiere".
12. O bibliotecă este o colecție de funcții elaborate de firmă care a dezvoltat mediul de programare pe care îl utilizăm (o vom denumi bibliotecă standard) sau de către alți programatori.

Ca urmare, se va citi de la tastatură o succesiune de caractere, care va fi convertită într-o dată de tipul variabilei specificate și apoi atribuită variabilei.

Exemplu

```
#include <iostream.h>
int main()
{int x, y;
 cin>>x;
 cin>>y;
 return 0; }
```

Am declarat două variabile de tip `int` `x` și `y` și am citit de la tastatură valorile lor. Dacă presupunem că tastăm secvența de caractere

```
10 20<Enter>
```

în variabila `x` vom citi valoarea `10`, iar în variabila `y`, valoarea `20`.

Operatorul de extragere (îl vom numi și operator de citire) îl putem utiliza și înlănțuit. Mai exact, dacă dorim să citim succesiv mai multe variabile putem scrie:

```
cin >> nume_var_1 >> nume_var_2 >> ... >> nume_var_n;
```

De exemplu, am fi putut citi valorile variabilelor `x` și `y` astfel:

```
cin >> x >> y;
```

Observații

1. La citirea cu ajutorul operatorului `'>>'`, valorile numerice care se citesc trebuie introduse de la tastatură, separate prin caractere albe.
2. Caracterele albe sunt ignorate de operatorul de citire `'>>'`.

Dacă dorim să scriem date pe ecran, vom utiliza operatorul `'<<'` de inserție în fluxul de ieșire (denumit și operator de ieșire sau de scriere):

```
cout << expresie;
```

Ca efect, se evaluează expresia, apoi valoarea ei este convertită într-o succesiune de caractere care va fi afișată pe ecran.

Și operatorul de ieșire poate fi utilizat înlănțuit, atunci când dorim să scriem mai multe date:

```
cout << expresie_1 << expresie_2 << ... << expresie_n;
```

Exemplu

```
#include <iostream.h>
int a;
int main()
{int x;
 cout<<"x="; cin>>x;
 cout<<"Valoarea lui x este "<<x<<'\n';
 cout<<"Valoarea lui a este "<<a<<'\n';
 return 0;}
```

În primul rând va fi afişat pe ecran mesajul

```
x=
```

apoi vom tasta valoarea lui x, care se va citi după acţionarea tastei *Enter*. După care se va afişa mesajul `Valoarea lui x este` , urmat de valoarea citită a lui x. De exemplu, dacă vom tasta 5 ca valoare a lui x, pe ecran vom obţine:

```
x=5
Valoarea lui x este 5
Valoarea lui a este 0
```

Variabila a este globală, deci automat a fost iniţializată cu 0. Observaţi că am scris şi caracterul `'\n'` (*newline*) pentru a trece pe linia următoare. Unii programatori preferă să utilizeze un simbol mai sugestiv care să determine trecerea la linie nouă. Acest simbol se numeşte *manipulator* şi este `endl` (*endline*). Deci am putea scrie în mod echivalent:

```
cout<<"Valoarea lui x este "<<x<<endl;
```

1.14. *Citiri şi scrieri în limbajul C*

Citirile şi scrierile în limbajul C se realizează prin intermediul unor funcţii specifice.

Citirea datelor cu format specificat

Funcţia `scanf()` permite citirea datelor sub controlul unui format specificat. Această funcţie este declarată în fişierul antet `stdio.h` şi are următorul format:

```
int scanf(format, adr_var1, adr_var2, ..., adr_varn);
```

Efect

Funcţia parcurge succesiunea de caractere introdusă de la tastatură şi extrage valorile care trebuie citite conform formatului specificat. Valorile citite sunt memorate în ordine în variabilele specificate prin adresa lor în lista de parametri ai funcţiei `scanf ()`. Adresa unei variabile se obţine cu ajutorul operatorului de referenţiere (&). Acest operator este unar (are un singur operand, care trebuie să fie o variabilă): `&variabila`.

Funcţia `scanf()` returnează numărul de valori citite corect. În cazul unei erori, citirea se întrerupe în poziţia în care a fost întâlnită eroarea.

Observaţie

Dacă în lista de parametri ai funcţiei `scanf()` specificaţi doar numele variabilei, nu adresa acesteia, funcţia va citi valoarea corespunzătoare acestei variabile, dar la ieşirea din funcţie valoarea citită nu este atribuită variabilei respective.

Parametrul `format` este un şir de caractere care poate conţine specificatori de format, caractere albe şi alte caractere.

Caracterele albe vor fi ignorate. Celelalte caractere (care nu fac parte dintr-un specificator de format) trebuie să fie prezente la intrare în poziţiile corespunzătoare.

Specificatorii de format au următoarea sintaxă:

```
% [*] [lg] [l|L] litera_tip
```

Observaţi că orice specificator de format începe cu caracterul `%`, conţine obligatoriu o literă ce indică tipul valorii care se citeşte şi, eventual, alte elemente opţionale. Litera ce indică tipul poate fi, de exemplu:

Literă_tip	Semnificaţie
d	Se citeşte un număr întreg scris în baza 10, care va fi memorat într-o variabilă de tip `int`.
o	Se citeşte un număr întreg scris în baza 8, care va fi memorat într-o variabilă de tip `int`.
u	Se citeşte un număr întreg scris în baza 10, care va fi memorat într-o variabilă de tip `unsigned int`.
x	Se citeşte un număr întreg scris în baza 16, care va fi memorat într-o variabilă de tip `int`.
f, **e** sau **g**	Se citeşte un număr real scris în baza 10, care va fi memorat într-o variabilă de tip `float`.
c	Se citeşte un caracter (în acest caz, caracterele albe prezente la intrare nu se ignoră).
s	Se citeşte un şir de caractere (şirul începe cu următorul caracter care nu este alb şi continuă până la primul caracter alb întâlnit sau până la epuizarea dimensiunii maxime `lg` din specificatorul de format).

Opţional, unele litere tip pot fi precedate de litera `l` sau de litera `L`. Literele `d`, `o` şi `x` pot fi precedate de litera `l`, caz în care valoarea citită va fi convertită la tipul `long int`. Litera `u` poate fi precedată de litera `l`, caz în care valoarea citită va fi convertită la tipul `unsigned long int`. Literele `f`, `e`, `g` pot fi precedate de litera `l` (caz în care valoarea citită este convertită la tipul `double`) sau de litera `L` (caz în care valoarea citită este convertită la tipul `long double`).

Opţional, poate fi specificată şi `lg` – lungimea maximă a zonei din care se citeşte valoarea. Mai exact, funcţia `scanf()` va citi maxim `lg` caractere, până la întâlnirea unui caracter alb sau a unui caracter neconvertibil în tipul specificat de `litera_tip`.

Caracterul opţional `'*'` specifică faptul că la intrare este prezentă o dată de tipul specificat de acest specificator de format, dar ea nu va fi atribuită nici uneia dintre variabilele specificate în lista de parametri ai funcţiei `scanf()`.

Observaţie

Caracterul `%` are o semnificaţie specială în parametrul format, el indică începutul unui specificator de format. Dacă dorim să specificăm în parametrul format că la intrare trebuie să apară caracterul `%`, vom utiliza construcţia sintactică `%%`.

Exemple

1. Să considerăm următoarele declaraţii de variabile:

```
int a; unsigned long b; double x;
```

Să presupunem că de la tastatură sunt introduse caracterele: `312 -4.5 100000`. Apelând funcţia:

```
scanf("%d %lf %lu", &a, &x, &b);
```

variabilei `a` i se atribuie valoarea `312`, variabilei `x` i se atribuie valoarea `-4.5`, iar variabilei `b` i se atribuie valoarea `100000`.

2. Să considerăm următoarea declaraţie de variabile:

```
char c1, c2, c3;
```

Să presupunem că de la tastatură sunt introduse caracterele: `312`. Apelând funcţia:

```
scanf("%c%c%c", &c1, &c2, &c3);
```

variabilei `c1` i se atribuie caracterul `'3'`, variabilei `c2` i se atribuie caracterul `'1'`, iar variabilei `c3` i se atribuie caracterul `'2'`.

3. Să considerăm următoarele declaraţii de variabile:

```
char c1, c2; unsigned a;
```

Să presupunem că de la tastatură se introduce: `312 xd`. Apelând funcţia:

```
scanf("%u%c%c", &a, &c1, &c2);
```

variabilei `a` i se atribuie valoarea `312`, variabilei `c1` i se atribuie caracterul `' '` (spaţiu), iar variabilei `c2` i se atribuie caracterul `'x'`.

Apelând funcţia:

```
scanf("%u %c %c", &a, &c1, &c2);
```

variabilei `a` i se atribuie valoarea `312`, variabilei `c1` i se atribuie caracterul `'x'`, iar variabilei `c2` i se atribuie caracterul `'d'`.

4. Să considerăm următoarea declaraţie de variabile:

```
int a, b, c;
```

Să presupunem că de la tastatură se introduce: `3=1+2`. Apelând funcţia:

```
scanf("%d=%d+%d", &a, &b, &c);
```

variabilei `a` i se atribuie valoarea `3`, variabilei `b` i se atribuie valoarea `1`, iar variabilei `c` i se atribuie valoarea `2`.

Apelând funcţia

```
scanf("%d=%*d+%d", &a, &b);
```


variabilei `a` i se atribuie valoarea `3`, variabilei `b` i se atribuie valoarea `2`. Valoarea `1` a fost citită, dar nu a fost atribuită nici unei variabile.

5. Să presupunem că de la tastatură se introduce o dată în formatul `zz-ll-aa`. Pentru a citi ziua, luna şi anul putem apela funcţia:

```
scanf("%d-%d-%d",&zi, &luna, &an);
```

Citirea caracterelor

Pentru citirea caracterelor au fost elaborate funcţii speciale. De exemplu:

1. Funcţia `getch()` este declarată în fişierul antet `conio.h`. Formatul funcţiei:

```
int getch(void);
```

Efect

Se citeşte de la tastatură un caracter. Funcţia returnează codul ASCII al caracterului citit. Caracterul tastat nu este afişat pe ecran.

2. Funcţia `getche()` este declarată în fişierul antet `conio.h`. Formatul funcţiei:

```
int getche(void);
```

Efect

Se citeşte de la tastatură un caracter. Funcţia returnează codul ASCII al caracterului citit. Caracterul tastat este afişat pe ecran.

Afişarea datelor cu format

Afişarea datelor pe ecran cu un format specificat se realizează apelând funcţia `printf()`. Această funcţie este declarată în fişierul antet `stdio.h` şi are următoarea sintaxă:

```
int printf(format, expresie1, expresie2, ..., expresien);
```

Efect

Se evaluează expresiile şi se scriu în ordine valorile acestora, în forma specificată de parametrul `format`.

În caz de succes, funcţia returnează numărul de caractere afişate.

Parametrul `format` este un şir de caractere care poate conţine specificatori de format şi, eventual, alte caractere. În parametrul format trebuie să existe câte un specificator de format pentru fiecare expresie din lista de parametri (specificatorul 1 corespunde expresiei 1, specificatorul 2, corespunde expresiei 2 etc).

Celelalte caractere din parametrul format vor fi afişate pe ecran în poziţiile corespunzătoare.

Un specificator de format are următoarea sintaxă:

```
% [ind] [lg] [.prec] [l|L] litera_tip
```

Observați că un specificator de format începe cu caracterul `%`, trebuie să conțină obligatoriu o literă care să indice tipul expresiei corespunzătoare și, eventual, alte elemente opționale. Litera care indică tipul poate fi, de exemplu:

Literă_tip	Semnificație
`d`	Expresia se convertește din tipul `int` și se afișează în baza 10.
`o`	Expresia se convertește din tipul `int` și se afișează în baza 8.
`u`	Expresia se convertește din tipul `unsigned` și se afișează în baza 10.
`x, X`	Expresia se convertește din tipul `int` și se afișează în baza 16 (literele care reprezintă 10, 11, 12, 13, 14, 15 sunt majuscule pentru `X` sau minuscule pentru `x`).
`f`	Expresia se convertește din tipul `float` și se afișează în formatul `parte_intreaga.parte_zecimala`.
`e, E`	Expresia se convertește din tipul `float` și se afișează în format exponențial (științific) `cifra.parte_zecimalae±exponent`. Pentru `E`, litera care precedă exponentul este `E`.
`g`	Se aplică una dintre conversiile corespunzătoare literei `f` sau literei `e` (cea care se reprezintă cu număr minim de caractere).
`c`	Expresia are ca valoare codul ASCII al unui caracter și se afișează caracterul corespunzător.
`s`	Expresia este un șir de caractere care va fi afișat pe ecran.

Opțional, înainte de `litera_tip` poate să apară litera `l` sau litera `L`. Dacă litera `l` apare înainte de `d`, `o`, `x`, `X` sau `u`, conversia se realizează din tipul `long int` (respectiv `unsigned long int`). Dacă litera `l` apare înainte de `f`, `e`, `E` sau `g`, conversia se realizează din tipul `double`. Litera `L` poate să preceadă doar literele `f`, `e`, `E` sau `g` (caz în care conversia se realizează din tipul `long double`).

Opțional, poate fi specificat `lg` – numărul minim de caractere pe care se va realiza afișarea. Dacă numărul de caractere necesar pentru a afișa valoarea expresiei depășește `lg`, se vor utiliza atâtea caractere câte sunt necesare. Dacă numărul de caractere necesare pentru a afișa valoarea expresiei este mai mic decât `lg`, se vor utiliza `lg` caractere (restul zonei pe care se realizează afișarea completându-se cu spații). Modul de aliniere implicit este la dreapta (deci completarea cu spații se realizează în stânga). Dacă se specifică indicatorul – (minus), atunci alinierea se va face la stânga (și completarea cu spații se va realiza la dreapta).

Pentru expresiile de tip real sau șir de caractere se poate specifica și precizia `prec`. Pentru valorile reale, aceasta indică numărul de cifre de la partea zecimală care vor fi afișate (implicit 6 zecimale). Dacă precizia este mai mică decât numărul de zecimale

ale numărului real afișat, atunci valoarea afișată va fi rotunjită (dacă prima zecimală care nu se afișează este mai mare sau egală cu 5, atunci ultima zecimală afișată va fi mai mare cu 1).

Pentru șiruri de caractere, precizia indică numărul de caractere din șir care se afișează (primele `prec`).

Exemple

Să considerăm următoarele declarații de variabile:

```
int a=-1, b=0567, d=0xf01a;
char c='x';
float x=-123.147, y=0.00008;
```

Să urmărim efectul următoarelor apeluri ale funcției `printf()`:

Apel	Pe ecran se afișează
`printf("a=%d sau a=%u\n", a, a);`	`a=-1 sau a=65535`

`printf("x=%f sau\n x=%e sau\n x=%g\n", x, x, x);`	`x=-123.147000 sau`
	`x=-1.231470e+02 sau`
	`x=-123.147`

`printf("y=%f sau\n y=%e sau\n y=%g\n", y, y, y);`	`y=0.000080 sau`
	`y=8.000000e-05`
	`sau y=8e-05`

`printf("b=%d sau b=%o sau b=%x \n", b, b, b);`	`b=375 sau b=567 sau b=177`
`printf("c=%c sau c=%d\n", c, c);`	`c=x sau c=120`
`printf("d=%d sau d=%x sau\n d=%u\n", d, d, d);`	`d=-4070 sau d=f01a sau` `d=61466`
`printf("x=%.2f\n", x);`	`x=-123.15`
`printf("b=%6dxb=%-6dx\n", b, b);`	`b= 375xb=375 x`

1.15. *Expresii*

O *expresie* este constituită dintr-o succesiune de operanzi conectați prin operatori. Un *operand* poate fi o constantă, o variabilă, un apel de funcție sau o expresie încadrată între paranteze rotunde.

Operatorii desemnează operațiile care se execută asupra operanzilor și pot fi grupați pe categorii, în funcție de tipul operațiilor desemnate. Pentru început, vom învăța operatorii aritmetici, de incrementare și decrementare, relaționali, de egalitate, logici, logici pe biți, de atribuire, condiționali, de determinare a dimensiunii și de conversie explicită.

Din punctul de vedere al priorității, operatorii pot fi grupați în 16 clase de prioritate, numerotate de la 1 la 16, 1 fiind prioritatea maximă. Pentru fiecare operator pe care îl vom prezenta vom indica clasa de prioritate.

Operatorii limbajului C/C++ sunt *unari* (se aplică unui singur operand) sau *binari* (necesită doi operanzi). Toți operatorii unari au clasa de prioritate 2.

Operatori aritmetici

Operatorii aritmetici desemnează operații aritmetice uzuale, cum sunt: `'*'` (înmulțirea), `'/'` (împărțirea), `'%'` (restul împărțirii întregi), `'+'` (adunarea), `'-'` (scăderea) și semnul algebric (operatorii unari `'+'`, `'-'`).

Operator	Denumire	Tip	Prioritate
`'+', '-'`	semn algebric	unari	2
`'*', '/', '%'`	multiplicativi	binari	4
`'+', '-'`	aditivi	binari	5

Observații

Operatorul `'%'` nu poate fi aplicat decât operanzilor întregi.

Operatorul `'/'` poate fi aplicat atât operanzilor întregi, cât și operanzilor reali, dar funcționează diferit pentru operanzii întregi, față de operanzii reali. Dacă cei doi operanzi sunt numere întregi, operatorul `'/'` furnizează câtul împărțirii întregi. Dacă cel puțin unul din cei doi operanzi este un număr real, operatorul `'/'` furnizează rezultatul împărțirii reale.

Exemple

Să considerăm următoarele declarații de variabile:

```
int a=3, b=5;
float x=2.5;
```

Expresie	Valoare	Observații
`b%2`	`1`	Restul împărțirii întregi a lui `b` la `2`.
`x%2`	–	Eroare! Operatorul `'%'` se aplică numai operanzilor întregi.
`a/2`	`1`	Câtul împărțirii întregi a lui `a` la `2`.
`x/2`	`1.25`	Câtul împărțirii reale a lui `x` la `2`.
`(a+b)/2`	`4`	Media aritmetică dintre `a` și `b`.

Operatori de incrementare/decrementare

Operatorul de incrementare este `'++'`. Operatorul de decrementare este `'--'`. Sunt operatori unari care au ca efect mărirea (respectiv micșorarea) valorii operandului cu 1. Acești operatori se pot utiliza în formă prefixată (înaintea operandului), caz în care se efectuează mai întâi incrementarea/decrementarea și apoi se

utilizează valoarea operandului, sau în formă postfixată (după operand), caz în care se utilizează mai întâi valoarea operandului și apoi se efectuează incrementarea/decrementarea. Acești operatori pot fi aplicați numai variabilelor simple.

Exemple

Să considerăm următoarele declarații de variabile:

```
int a=3, b=5;
float x=2.5;
```

Expresie	Valoare	Observații
`a++`	`4`	Se mărește valoarea variabilei `a` cu `1`.
`--x`	`1.5`	Se micșorează valoarea variabilei `x` cu `1`.
`int c=--a;`	`2`	Variabila declarată `c` primește valoarea `2`.
`int c=a--;`	`3`	Variabila declarată `c` primește valoarea `3`.
`(a+b)++`	–	Eroare! operatorii de incrementare/decrementare nu pot fi aplicați unei expresii.

Operatori relaționali

Operatorii relaționali sunt operatori binari și desemnează relația de ordine în care se găsesc cei doi operanzi: `'<'` (mai mic), `'>'` (mai mare), `'<='` (mai mic sau egal), `'>='` (mai mare sau egal). Rezultatul aplicării unui operator relațional este `1` dacă cei doi operanzi sunt în relația indicată de operator, și `0`, altfel. Grupa de prioritate a operatorilor relaționali este 7.

Exemple

Expresie	Valoare	Observații
`3>5`	`0`	Operanzii nu sunt în relația >.
`7<=3+12`	`1`	Operanzii sunt în relația <=.

Operatori de egalitate

Operatorii de egalitate sunt operatori binari și desemnează relația de egalitate (==) sau inegalitate (`!=`) în care se găsesc cei doi operanzi. Rezultatul aplicării unui operator de egalitate este `1`, dacă cei doi operanzi sunt în relația indicată de operator, și `0`, altfel. Grupa de prioritate a operatorilor relaționali este 8.

Exemple

Expresie	Valoare	Observații
`3==2+1`	`1`	Operanzii sunt în relația `'=='`.
`3!=2+1`	`0`	Operanzii nu sunt în relația `'!= '`.

Operatori logici globali

Există trei operatori logici globali:

Operator	Denumire	Tip	Prioritate
!	negația logică (not)	unar	2
&&	conjuncție logică (și)	binar	12
||	disjuncție logică (sau)	binar	13

În limbajul C/C++, valoarea logică `fals` este asociată valorii `0`, orice valoare diferită de `0` având semnificația `adevarat`. Prin urmare, efectul operatorilor logici globali, așa cum știm de la logică matematică, este:

x	0	≠0
!x	1	0

&&	0	≠0
0	0	0
≠0	0	1

||	0	≠0
0	0	1
≠0	1	1

Exemple

Expresie	Valoare
`!(x%2)`	`1`, dacă `x` este par. `0`, dacă `x` este impar.
`(x>=a)&&(x<=b)`	`1`, dacă `x` este în intervalul `[a,b]`. `0`, dacă `x` nu este în intervalul `[a,b]`.
`(x<a)||(x>b)`	`0`, dacă `x` este în intervalul `[a,b]`. `1`, dacă `x` nu este în intervalul `[a,b]`.

Operatori logici pe biți

Operatorii logici pe biți se aplică numai operanzilor întregi și au ca efect aplicarea operațiilor logice cunoscute (negație, conjuncție, disjuncție și disjuncție exclusivă) **bit cu bit**. Operatorii logici pe biți sunt:

Operator	Denumire	Tip	Prioritate
~	complementariere (negația pe biți)	unar	2
<<, >>	deplasare la stânga, deplasare la dreapta	binari	6
&	conjuncție logică pe biți	binar	9
^	disjuncție exclusivă pe biți	binar	10
|	disjuncție logică pe biți	binar	11

Efectul operatorilor pe biți este:

x	0	1
~x	1	0

&	0	1
0	0	0
1	0	1

^	0	1
0	0	1
1	1	0

|	0	1
0	0	1
1	1	1

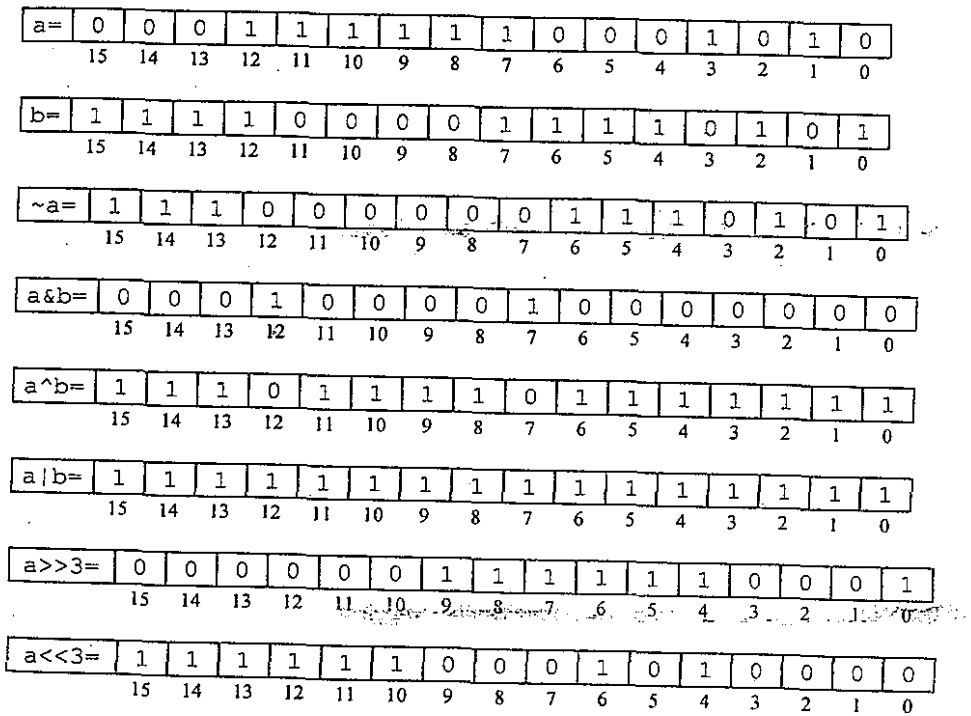
Operatorii de deplasare au ca efect deplasarea reprezentării binare a primului operand spre stânga (<<) sau spre dreapta (>>). Numărul de poziții care se deplasează este specificat de cel de-al doilea operand. La deplasarea la stânga, pozițiile rămase libere în dreapta se completează cu `0`. La deplasarea la dreapta, pozițiile rămase libere în stânga se completează cu `0` (dacă operandul stâng este un întreg pozitiv) sau cu `1` (dacă operandul este întreg negativ).

Exemple

Să considerăm următoarele declarații de variabile:

```
int n=3, a=0X1F8A, b=0XF0F5;
```

Pentru a determina cu ușurință reprezentările în memorie ale variabilelor `a` și `b`, am exprimat valorile lor în hexazecimal. Reprezentarea în memorie a acestor variabile este:



Dacă a ar fi fost declarat de tip unsigned, prin deplasare la dreapta, s-ar obține același rezultat, deoarece valoarea lui a este pozitivă (bitul semn este 0). Valoarea lui b este negativă (bitul 15 – bitul semn – este 1), prin deplasare la dreapta se propagă semnul, deci se completează cu 1.

b>>3=	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Observații

1. Expresia x<<n are ca efect înmulțirea operandului x cu 2^n. Expresia x>>n are ca efect împărțirea întreagă a operandului x cu 2^n.
2. Operațiile care se efectuează pe biți (deci acționează direct asupra reprezentării interne a operanzilor) sunt foarte performante (se execută foarte rapid).

Operatori de atribuire

Operatorii de atribuire sunt operatori binari care permit modificarea valorii unei variabile. Există un operator de atribuire simplu (=) și 10 operatori de atribuire compuși cu ajutorul operatorului '=' și al unui alt operator binar (aritmetic sau logic pe biți), după cum urmează:

```
variabila = expresie
```

Efect: se evaluează expresia, apoi se atribuie variabilei valoarea expresiei. Valoarea expresiei de atribuire este egală cu valoarea atribuită variabilei.

```
variabila operator= expresie
```

unde operatorul poate fi din mulțimea {*, /, %, +, -, <<, >>, &, |, ^}.

Efect: echivalent cu al unei atribuiri de tipul

```
variabila = variabila operator expresie
```

Grupa de prioritate a operatorilor de atribuire este 15.

Observație

Expresia poate fi la rândul ei o expresie de atribuire, cu alte cuvinte, operatorii de atribuire se pot utiliza înlănțuit:

```
variabila_1=variabila_2=...=variabila_n=expresie
```

Exemple

Să considerăm următoarele declarații de variabile:

```
int a=10, b;
long c=100001;
float x, y=5;
```


Expresie	Valoare	Observații
a+=2	12	La valoarea variabilei a se adună 2.
x=(a+c)/2	50005	Variabilei x i se atribuie câtul împărțirii întregi a lui a+c la 2.
b=a*=2	20	Valoarea variabilei a se înmulțește cu 2, apoi se atribuie această valoare și variabilei b.
a=y/2	2	Variabila a primește partea întreagă a valorii de tip float y/2.

Operatori condiționali

Operatorii condiționali sunt '?' și ':' și se utilizează numai împreună. Formatul unei expresii condiționale este:

```
expresie_1 ? expresie_2 : expresie_3
```

Efect: se evaluează expresie_1. Dacă expresie_1 are o valoare nenulă, atunci valoarea expresiei condiționale este egală cu valoarea expresie_2. Dacă expresie_1 are valoarea 0, atunci valoarea expresiei condiționale este egală cu valoarea expresie_3. Grupa de prioritate a operatorilor condiționali este 14.

Exemple

Expresie	Valoare
x>y ? x : y	maximul dintre x și y
x>=0 ? x : -x	modulul lui x

Operatorul de referențiere (adresă)

Este un operator unar care permite determinarea adresei zonei de memorie în care este stocată o variabilă:

```
&variabila
```

Operatorul de conversie explicită

Este un operator unar care permite conversia explicită (forțată) a tipului unei expresii la un tip specificat:

```
(tip) expresie
```

Exemple

Să considerăm următoarele declarații de variabile:

```
int a=10, b=15;
long x;
```

Expresie	Valoare	Observaţii
`((float)a+b)/2`	`12.5`	Media aritmetică a variabilelor `a` şi `b` (în absenţa conversiei de tip, se calcula câtul împărţirii întregi a lui `a+b` la 2, deci 12).
`x=(long)a*10000`	`100000`	În absenţa conversiei de tip se calcula valoarea expresiei `a*10000` ca valoare de tip `int`; cum `100000` depăşeşte `32767`, s-ar fi obţinut valoarea eronată `-31072`.

Operatorul de determinare a dimensiunii

Este un operator unar care determină dimensiunea exprimată în număr de octeţi a zonei de memorie necesare pentru stocarea expresiei specificate ca operand sau a unei date de tipul specificat ca operand:

```
sizeof (expresie)
sizeof (tip)
```

Exemple

Expresie	Valoare	Observaţii
`int v;` `sizeof(v)`	2	O variabilă de tip `int` se memorează pe 2 octeţi.
`sizeof(float)`	4	O dată de tip `float` se memorează pe 4 octeţi.

Operatorul virgulă

Operatorul virgulă permite compunerea mai multor expresii, astfel încât să fie tratate din punct de vedere sintactic ca o singură expresie.

```
expresie_1, expresie_2, ..., expresie_n
```

Se evaluează în ordinea de la stânga la dreapta cele `n` expresii, valoarea întregii expresii fiind egală cu valoarea `expresie_n`.

Utilizarea expresiilor compuse cu operatorul virgulă este necesară atunci când sintaxa permite evaluarea unei singure expresii (de exemplu, în instrucţiunea `for`, pe care o vom învăţa în capitolul următor), dar este necesar să fie evaluate mai multe expresii. Prioritatea operatorului virgulă este minimă (16).

Exemplu

Expresie	Valoare	Observaţii
`int i, a, b;` `i=0,b=i+2,a=b*2`	4	Variabila `i` primeşte valoarea `0`, variabilei `b` i se atribuie valoarea `2`, apoi variabilei `a` i se atribuie valoarea `4`.

Evaluarea expresiilor

Evaluarea unei expresii presupune calculul valorii expresiei, prin înlocuirea în expresie a fiecărei variabile cu valoarea ei şi a fiecărei funcţii cu valoarea returnată de funcţia respectivă şi efectuarea operaţiilor specificate de operatori. În timpul evaluării expresiei se ţine cont de existenţa parantezelor, de asociativitate şi de prioritatea operatorilor:

- se evaluează în primul rând expresiile din paranteze, începând cu parantezele cele mai interioare;
- în cadrul unei expresii fără paranteze, se efectuează operaţiile în ordinea priorităţii operatorilor;
- dacă într-o expresie apare o succesiune de operatori cu priorităţi egale, se ţine cont de asociativitatea operatorilor. În limbajul C/C++, operatorii se asociază de la stânga la dreapta, cu excepţia operatorilor unari, condiţionali şi de atribuire, care se asociază de la dreapta la stânga.

Dacă toţi operanzii care intervin într-o expresie au acelaşi tip, tipul expresiei coincide cu tipul operanzilor. În cazul în care operanzii nu au acelaşi tip, pe parcursul evaluării expresiei se realizează automat o serie de conversii implicite. Pe scurt, regula pe care se bazează conversiile implicite este: *operandul care are un domeniu de valori mai restrâns este convertit la tipul operandului care are mulţimea valorilor mai amplă*. De exemplu: toţi operanzii de tip `char` se convertesc la `int`; dacă unul dintre operanzi este `long double`, celălalt va fi convertit de asemenea la `long double`; dacă unul dintre operanzi este `long`, iar celălalt este de tip `unsigned` sau `int`, el va fi convertit automat la `long`. Dacă un operand este de tip `int`, iar celălalt este de tip `unsigned`, conversia se face către `unsigned`.

Exemple

Expresie	Valoare	Observaţii
`int a=3; float b=6;` `(a+b)/2`	`4.5`	Se converteşte tipul variabilei `a` la `float`.
`int a=5;` `!a%2`	`0`	Se aplică operatorul unar `'!'` variabilei `a`, apoi se execută operaţia `'%'`.

`int a=30000, b=30000;`
`long c;`
`c=a+b;`
	`-5536`	Se adună `a` cu `b`; suma depăşeşte valoarea maximă pentru tipul `int`; valoarea obţinută în urma depăşirii (-5536) este atribuită variabilei `c`.

`int a=30000, b=30000;`
`long c;`
`c=(long)a+b;`
	`60000`	Se converteşte tipul variabilei `a` la `long`, apoi se adună `a` cu `b`, obţinând un rezultat de tip `long`, care este atribuit variabilei `c`.

Tabelul priorității operatorilor

Grupa	Operatori	Asociativitate
2	! ~ + - ++ -- (tip) sizeof &	dreapta → stânga
4	* / %	stânga → dreapta
5	+ -	stânga → dreapta
6	<< >>	stânga → dreapta
7	< > <= >=	stânga → dreapta
8	== !=	stânga → dreapta
9	&	stânga → dreapta
10	^	stânga → dreapta
11	|	stânga → dreapta
12	&&	stânga → dreapta
13	||	stânga → dreapta
14	? :	dreapta → stânga
15	= *= /= %= += -= &= ^= |= <<= >>=	dreapta → stânga
16	,	stânga → dreapta

1.16. *Aplicații*

Disc

Fie `r` un număr real, citit de la tastatură, care reprezintă lungimea razei unui cerc. Să se scrie un program care să calculeze și să afișeze aria și perimetrul discului de rază `r`.

Soluție

Vom citi valoarea razei în variabila `r`. Vom calcula aria discului după formula πr^2, iar perimetrul, după formula $2\pi r$. Numărul π este un număr irațional. Calculatorul nu poate memora numere iraționale (care au o infinitate de zecimale), ci doar aproximări ale acestora. În fișierul antet `math.h` este definită o constantă numită `M_PI` care reprezintă o aproximare a numărului irațional π.

Prezentăm atât programul C, cât și programul C++ (diferențele constau, evident, în modul de citire și afișare).

Varianta C:

```
#include <stdio.h>
#include <math.h>
int main(void)
{double r;
 scanf("%lf",&r);
 printf("Aria=%.2lf Perimetrul=%.2lf\n",M_PI*r*r,2*M_PI*r);
 return 0;}
```

Varianta C++:

```
#include <iostream.h>
#include <math.h>
int main()
{double r;
 cin>>r;
 cout<<"Aria="<<M_PI*r*r<< " Perimetrul="<< 2*M_PI*r;
 return 0; }
```

Compus chimic

Un grup de cercetători studiază un compus chimic descoperit pe planeta Marte. În urma analizelor efectuate, au dedus că o moleculă din acest compus este formată din `nC` atomi de carbon, `nO` atomi de oxigen și `nH` atomi de hidrogen. Știind că masa atomului de carbon este `12`, masa atomului de oxigen este `16`, iar masa atomului de hidrogen este `1`, să se scrie un program care să calculeze și să afișeze masa moleculară a acestui compus.

Soluție

Varianta C

```
#include <stdio.h>
int main(void)
{ unsigned nC, nO, nH, m;
  scanf("%u%u%u", &nC, &nO, &nH);
  printf("Masa moleculara a compusului este %u \n",
          12*nC+16*nO+nH);
  return 0; }
```

Varianta C++

```
#include <iostream.h>
int main()
{ unsigned nC, nO, nH, m;
  cin>>nC>>nO>>nH;
  cout<<"Masa moleculara a compusului este " <<
       12*nC+16*nO+nH << endl;
  return 0; }
```

Triunghi

Fie x un număr natural format din 5 cifre ($x_4x_3x_2x_1x_0$). Să se afişeze un triunghi format din cifrele numărului x astfel:

- pe prima linie (în vârful triunghiului) se va afla cifra din mijloc (x_2);
- pe linia a doua se vor afla cifrele $x_3x_2x_1$;
- pe a treia linie se vor afla toate cifrele lui x.

De exemplu, dacă x=15289, triunghiul va arăta astfel:

```
  2
 528
15289
```

Soluţie

Problema constă în „spargerea" numărului x în cifre. În acest scop, am numerotat cifrele numărului x de la dreapta la stânga începând cu 0, astfel încât numărul cifrei să corespundă puterii corespunzătoare bazei (în cazul nostru, baza 10): x_0 este cifra unităţilor (deci corespunde lui 10^0), x_1 este cifra zecilor (deci corespunde puterii 10^1), x_2 este cifra sutelor (corespunde lui 10^2) ş.a.m.d.

Devine astfel evident că, pentru a extrage cifrele numărului x, trebuie să efectuăm împărţiri la 10. Pentru a obţine ultima cifră din x, vom împărţi pe x la 10 şi vom reţine restul în x_0. Eliminăm apoi ultima cifră din x (împărţind pe x la 10, x devine $x_4x_3x_2x_1$). Acum x_1 a devenit cifra unităţilor şi continuăm extragerea cifrelor numărului x în acelaşi mod. Programul în limbajul C este:

```
#include <stdio.h>
int main(void)
{unsigned long x;
 unsigned x0, x1, x2, x3;
 scanf("%lu",&x);
 x0=x%10; /* retin cifra unitatilor */
 x=x/10;  /* elimin cifra unitatilor */
 x1=x%10; /* retin cifra zecilor */
 x=x/10;  /* elimin cifra zecilor */
 x2=x%10; /* retin cifra sutelor */
 x=x/10;  /* elimin cifra sutelor */
 x3=x%10; /* retin cifra miilor */
 x=x/10;  /* in x ramane cifra zecilor de mii */
 printf("  %u\n", x2);
 printf(" %u%u%u\n", x3,x2,x1);
 printf("%lu%u%u%u%u\n",x,x3,x2,x1,x0);
 return 0; }
```

Observaţie

Variabila x în care citim numărul dat este de tip `unsigned long`, deoarece un număr de 5 cifre poate depăşi 65535.

Exerciţiu

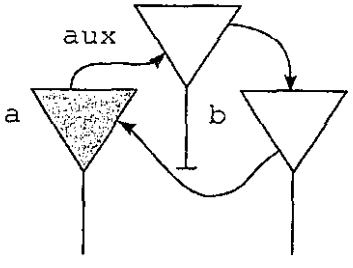
Modificaţi algoritmul precedent astfel încât să afişeze un triunghi format din cifrele unui număr de 6 cifre. De exemplu, pentru 123456, algoritmul va afişa:

```
  34
 2345
123456
```

Schimb

Se introduc de la tastatură numerele reale a şi b. Să se interschimbe valorile variabilelor a şi b, apoi să se afişeze.

Soluţie



Să ne imaginăm că variabila a este un pahar cu vin alb, iar variabila b un pahar cu vin roşu. Trebuie să schimbăm conţinuturile celor două pahare. Singura soluţie fără „pierderi" este de a utiliza un pahar auxiliar (variabila aux). Vom turna vinul alb în paharul auxiliar (atribuim variabilei aux valoarea variabilei a). Acum paharul de vin alb este gol, turnăm în el vinul roşu (atribuim variabilei a valoarea variabilei b). Paharul de vin roşu (variabila b) a devenit disponibil, turnăm în el vinul alb din paharul auxiliar (atribuim variabilei b valoarea variabilei aux).

Această metodă este denumită sugestiv *regula celor trei pahare*, deşi analogia nu este absolut perfectă (când atribuim unei variabile valoarea altei variabile, această valoare se va găsi după atribuire în ambele, în timp ce atunci când turnăm conţinutul unui pahar de vin în alt pahar de vin...).

```
#include <stdio.h>
int main(void)
{
 float a, b, aux;
 scanf("%f%f", &a, &b);       /* 1 */
 aux=a;                       /* 2 */
 a=b;                         /* 3 */
 b=aux;                       /* 4 */
 printf("a=%f b=%f\n",a,b);   /* 5 */
 return 0;
}
```

Pentru a înţelege mai bine acest algoritm, să urmărim execuţia lui pentru un set particular de date de intrare. În acest scop, am numerotat instrucţiunile de la 1 la 5. De exemplu, să presupunem că de la tastatură se introduc valorile 3 şi 7.

Linia	a	b	aux	Explicație
1	3	7	?	Se citesc de la tastatură valorile 3 și 7 și se atribuie în ordine variabilelor a și b. Variabila aux are deocamdată o valoare necunoscută.
2	3	7	3	Se copiază valoarea variabilei a în variabila aux.
3	7	7	3	Variabilei a i se atribuie valoarea variabilei b.
4	7	3	3	Variabilei b i se atribuie valoarea variabilei aux (în care am „salvat" valoarea inițială a variabilei a).
5	7	3	3	Scriem valorile variabilelor a și b însoțite de mesaje explicative (pe ecran va apărea a=7 b=3).

Leul și iepurașul

Un iepuraș zglobiu ieși din pădure și începu să alerge pe câmpie cu o viteză constantă de v1 m/s. După un timp t0, apare la marginea pădurii un leu. Leul zări iepurașul și începu să alerge după el cu o viteză constantă de v2 m/s. Scrieți un program care să afișeze după câte secunde prinde leul iepurele sau valoarea -1 dacă leul nu prinde iepurele.

Soluție

Evident, dacă viteza leului este mai mică sau egală cu viteza iepurașului, leul nu va prinde iepurele. În caz contrar, să notăm cu x distanța de la marginea pădurii până în punctul în care leul prinde iepurele și cu t timpul după care prinde leul iepurele. Leul străbate distanța x în t secunde (fiindcă viteza leului este v2 m/s, deducem că x=v2*t). Iepurele străbate distanța x în t+t0 secunde (fiindcă viteza iepurelui este v1 m/s, deducem că x=v1*(t+t0)). Din aceste două ecuații cu două necunoscute putem afla t. Programul în limbajul C++ este:

```
#include <iostream.h>
int main()
{ float v1, v2, t;
  cout << "Viteza iepurelui= ";              cin >> v1;
  cout << "Viteza leului= ";                 cin >> v2;
  cout << "Timpul dupa care apare leul= "; cin >> t0;
  cout << (v2>v1?v1*t0/(v2-v1):-1)<<endl;  return 0;}
```

Dacă am executa programul pe următorul exemplu:

```
Viteza iepurelui= 4
Viteza leului= 5
Timpul dupa care apare leul= 2
```

pe ecran se va afișa:

```
8
```


Bazin

Un bazin se umple cu apă cu ajutorul a două robinete. Dacă lăsăm primul robinet deschis timp de h1 ore și al doilea timp de h2 ore, în bazin vor fi x litri de apă.

Dacă lăsăm primul robinet deschis timp de h1+1 ore și al doilea timp de h2-1 ore, în bazin vor fi y litri de apă. Scrieți un program care să determine câți litri de apă curg prin fiecare robinet într-o oră.

Soluție

Să notăm cu a cantitatea de apă care curge într-o oră prin primul robinet și cu b cantitatea de apă care curge într-o oră prin cel de-al doilea robinet. Deducem relațiile: x=h1*a+h2*b și y=(h1+1)*a+(h2-1)*b.

Dacă scădem cele două relații, obținem y-x=a-b, deci a=(y-x)+b. Înlocuim a în prima relație și obținem x=h1*(y-x)+h1*b+h2*b, deci b=(x-h1*(y-x))/(h2+h1).

```
#include <iostream.h>
int main()
{int x, y, h1, h2, b;
 cout<<"h1="; cin>>h1;
 cout<<"h2="; cin>>h2;
 cout<<"x="; cin>>x;
 cout<<"y="; cin>>y;
 cout<< "b="<<(b=(x-h1*(y-x))/(h2+h1))<<endl;
 cout<<"a= "<<(y-x)+b<<endl; return 0;}
```

De exemplu, pentru următoarele date de intrare:

```
h1=4
h2=7
x=47
y=45
```

programul va afișa:

```
b=5
a=3
```

1.17. *Probleme propuse*

1. Ce va afișa următorul program, dacă se citesc valorile 17 și 8?

```
#include <stdio.h>
int main(void)
{
 unsigned a, b;
 scanf("%u %u", &a, &b);
 a=a+b; b=a-b; a=a-b;
 printf("a=%u b=%u\n", a, b);
 return 0;
}
```

2. Ce va afişa următorul program, dacă se citeşte valoarea `1234`?

```
#include <stdio.h>
int main(void)
{unsigned a, b, c;
 scanf("%u", &a);
 b=a%100; a=a/100; c=b*100+a;
 printf("c=%u\n", c);
 return 0; }
```

3. Ce valoare va avea variabila `a` la sfârşitul executării următoarei secvenţe de program?

```
int a=13, b=4;
b=a+b/2; a=a-b/2*a;
```

4. Fie `a`, `b`, `c` şi `d` patru variabile reale. Care dintre următoarele instrucţiuni atribuie variabilei `d` media aritmetică a valorilor variabilelor `a`, `b` şi `c`?

a. `d=(a+b+c)/2;`
b. `d=a/3+b/3+c/3;`
c. `d=a+b+c/3;`
d. `d=(a+b+c)/4-1;`

5. Care dintre următoarele declaraţii de variabile sunt eronate sintactic şi de ce?

a. `float a=b=0;`
b. `double z=x/2;`
c. `char d=120;`
d. `int k;i;`
e. `long a=0, b=a+3;`
f. `unsigned float a;`

6. Care dintre următoarele expresii sunt adevărate dacă şi numai dacă numărul întreg `x` este impar negativ?

(variantă Bacalaureat, 2000)

a. `(x%2==1) && (x<0)`
b. `(x%2!=0) || (x<0)`
c. `!((x%2==0) || (x>=0))`
d. `!((x%2==0) && (x>=0))`
e. `x%2=1 && x<0`

7. Care dintre următoarele expresii sunt adevărate dacă şi numai dacă valorile variabilelor `x` şi `y` sunt numere naturale consecutive?

(variantă Bacalaureat, 2000)

a. `x-y==1`
b. `(x==1) && (y==2)`
c. `(x-y==1) && (y-x==1)`
d. `y==x±1`
e. `(x-y==1) || (y-x==1)`

8. Să considerăm următoarele declaraţii de variabile:

```
int x=0XF0A8, v=0XFFFD;
unsigned y=0XF0A8, z=1;
float a=2.5, b=4;
char c='8';
```


Evaluaţi următoarele expresii:

a. `a+z/2`
b. `(v+z)/2`
c. `(v+1)/2`
d. `c-'0'`
e. `-1 > z`
f. `c>='0' && c <= '9'?"da":"nu"`
g. `x>>3&z<<4`
h. `y>>10|z`

9. În contextul declaraţiilor de variabile de la exerciţiul precedent, care sunt erorile din următoarele expresii?

a. `(a+z)%2`
b. `a+++x`
c. `b*b-4ac`
d. `x=(a>>2)`
e. `c=a?"da":"nu"`
f. `x&&=10`

10. În condiţiile următoarelor declaraţii şi iniţializări de variabile:

```
unsigned char x=250, z=x+7, a='8';
```

Care este valoarea expresiei: `z|(a-'0')`?

a. Expresia nu se poate evalua, deoarece este eronată sintactic.
b. `1`
c. `265`
d. `true`
e. `0`
f. `9`

11. Se consideră `a` şi `b` două variabile de tip `int`. Variabilei `a` i se atribuie valoarea `1`, iar variabilei `b`, valoarea `32767`. Care este valoarea expresiei `a+b`?

a. `32768`
b. `-1`
c. `-32768`
d. nu se poate calcula valoarea acestei expresii
e. `0`

12. Se consideră `x`, `y` şi `z` trei variabile întregi. Care dintre următoarele expresii are valoarea diferită de `0` dacă şi numai dacă `y=max(x,y,z)`?

a. `x>z?y>=x?1:0:y>=z?1:0`
b. `!(y<x || y<z)`
c. `!(y<x && y<z)`
d. `x>z && y>x || z>x && y>z`
e. `y>x && y>z`

13. Fie `x`, `a`, `r` trei numere naturale. Care dintre următoarele expresii au valoarea diferită de `0` dacă şi numai dacă `x` este impar situat la distanţă cel mult egală cu `r` faţă de `a`?

a. `x%2==1 && !(x>a+r && x<a-r)`
b. `x%2 && x<=a+r && x>a-r`
c. `!(x%2==0 || x>a+r || x<a-r)`
d. `x%2=1 &&a+r>=x && a-r<=x`

14. Care dintre următoarele expresii au valoarea diferită de `0` dacă şi numai dacă variabilele întregi `a`, `b` şi `c` conţin trei valori distincte două câte două?

a. `a!=b!=c!=a`
b. `!(a==b && b==c && a==c)`
c. `(a!=b) &&(b!=c)`
d. `!(a==b)&&!(b==c || a==c)`

15. Pentru a atribui variabilei reale `x` rezultatul expresiei $\frac{2ab - c^2}{0.25}$, unde `a`, `b` şi `c` desemnează variabile reale, se utilizează instrucţiunea:

(variantă Bacalaureat, 2001)

a. `x=(2*a*b)-(c*c)/0.25;`
c. `x=(2*a*b)-(c*c)*4;`
b. `x=2*a*b-c*c/0.25;`
d. `x=(2*a*b-c*c)*4;`

16. Fie `c` o variabilă de tip `char`. Scrieţi o expresie care, în cazul în care valoarea variabilei `c` este o literă mică, atribuie variabilei `c` majuscula corespunzătoare.

17. Fie `x` un număr natural. Scrieţi o expresie care să utilizeze operatori logici pe biţi astfel încât:

a. să înmulţească valoarea variabilei `x` cu 2^n ($0 \le n \le 15$);
b. să împartă valoarea variabilei `x` cu 2^n ($0 \le n \le 15$);
c. să aibă valoarea `1` dacă şi numai dacă `x` este impar;
d. să aibă valoarea `0` dacă şi numai dacă bitul `n` ($0 \le n \le 15$) din `x` este `0`;
e. să anuleze bitul `n` ($0 \le n \le 15$) din `x`;
f. să seteze bitul `n` ($0 \le n \le 15$) din `x` (să-i atribuie valoarea `1`).

18. Scrieţi un program care citeşte de la tastatură lungimea laturii unui triunghi echilateral şi afişează pe ecran lungimea înălţimii triunghiului şi aria sa, pe linii diferite, însoţite de mesaje explicative.

19. Fie `x1`, `x2`, `x3`, `x4`, `x5` cinci valori reale. Scrieţi un program care să folosească o singură variabilă suplimentară pentru a permuta circular valorile celor cinci variabile (adică în final `x1` să aibă valoarea iniţială a variabilei `x2`, `x2` valoarea iniţială a variabilei `x3`, `x3` valoarea iniţială a variabilei `x4`, `x4` valoarea iniţială a variabilei `x5`, iar `x5` valoarea iniţială a variabilei `x1`).

20. O broască ţestoasă parcurge o distanţă de `D` kilometri în `H` ore. Să se scrie un program care să calculeze şi să afişeze viteza cu care se deplasează broasca ţestoasă (exprimată în metri/secundă).

21. Doi colegi (Vasilică şi Ionică) pleacă simultan din oraşele în care locuiesc, unul către celălalt. Ştiind că distanţa dintre cele două oraşe este `D`, că Vasilică merge cu viteza `v1`, iar Ionică merge cu viteza `v2` (`D`, `v1`, `v2` sunt numere reale), scrieţi un program care calculează după cât timp se întâlnesc cei doi colegi şi la ce distanţă de oraşul în care locuieşte Vasilică.

22. Fie A şi B două puncte în plan, specificate prin coordonatele lor carteziene. Să se scrie un program care să calculeze şi să afişeze lungimea segmentului AB.

23. Scrieţi un program care citeşte de la tastatură un număr real `x` şi afişează pe ecran valoarea funcţiei:

$$f(x) = \begin{cases} \sqrt{|2-x^2|}, & \text{dacă } x \le 10 \\ \frac{-2x+1}{3}, & \text{dacă } x > 10 \end{cases}.$$

24. Consultaţi sistemul *Help* al mediului de programare pentru a afla care sunt funcţiile care permit generarea numerelor aleatoare. Aceste funcţii sunt declarate în fişierul antet `stdlib.h`. Scrieţi un program care să citească două numere naturale nenule `a` şi `b` şi care să genereze şi să afişeze:

a. un număr natural aleator mai mic decât `a`;
b. un număr întreg aleator din intervalul [`-a`, `b`];
c. un număr natural aleator din intervalul [`a`, `b`];
d. un număr real aleator din intervalul [`-a`, `b`].

2. Instrucțiunile limbajului C/C++

2.1. *Instrucțiunea expresie*

```
expresie;
```

Efect

Se evaluează expresia.

Exemple

```
i++;        //se incrementeaza valoarea variabilei i
p*=2;       //se inmulteste cu 2 valoarea variabilei p
clrscr();   //se sterge ecranul
```

Observații

1. Ultima expresie este constituită dintr-un apel de funcție. Pentru a apela funcția standard `clrscr()`, trebuie să includem la începutul programului fișierul antet `conio.h`, în care se găsește prototipul (declarația) funcției:

```
void clrscr(void);
```

ceea ce înseamnă că funcția nu are nici un parametru și nu întoarce nici un rezultat.
2. Este permis ca expresia să fie vidă. Instrucțiunea devine `;` (instrucțiunea vidă).
3. Este permisă utilizarea oricărei expresii sintactic corecte, chiar și atunci când instrucțiunea nu generează nici un efect, de exemplu: `2+5;`
4. Orice instrucțiune din limbajul C/C++ se termină cu caracterul `';'`.

2.2. *Instrucțiunea compusă*

```
{
 declaratii
 instructiune_1
 instructiune_2
   ...
 instructiune_n
}
```

Efect

Se execută în ordine instrucțiunile specificate.

Observații

1. Utilizarea instrucțiunilor compuse este necesară atunci când sintaxa permite executarea unei singure instrucțiuni, dar este necesară efectuarea mai multor operații (de exemplu, într-o instrucțiune `if`, `for`, `while` sau `do-while`).
2. Declarațiile care apar într-o instrucțiune compusă sunt **locale** instrucțiunii. Mai exact, ele sunt valabile numai în corpul instrucțiunii compuse, din momentul declarării lor până la sfârșitul instrucțiunii.

2.3. *Instrucțiunea* `if`

```
if (expresie) instructiune_1
    else instructiune_2
```

Efect

Se evaluează expresia. Dacă valoarea expresiei este diferită de `0`, se execută `instructiune_1`, altfel se execută `instructiune_2`.

Exemplu

```
if (a) cout << a << " este nenul" << endl;
    else cout << a << " este nul" << endl;
```

Observații

1. Expresia se încadrează obligatoriu între paranteze rotunde.
2. Dacă `instructiune_2` este vidă, ramura `else` poate să lipsească, obținându-se o formă simplificată a instrucțiunii `if`:

```
if (expresie) instructiune
```

Exemplu

Să calculăm în variabila `max` maximul dintre `a` și `b`:

```
max=a;
if (max<b) max=b;
```

2.4. *Instrucțiunea* `switch`

```
switch     (expresie)
{ case     expresie-constanta1: secventa-instructiuni1
  case     expresie-constanta2: secventa-instructiuni2
    ...
  case     expresie-constantan: secventa-instructiunin
  default:                      secventa-instructiunin+1
}
```

Efect

Se evaluează expresia.

Se compară succesiv valoarea expresiei cu valorile expresiilor constante care etichetează alternativele `case`. Dacă se întâlneşte o alternativă `case` etichetată cu valoarea expresiei, se execută secvenţa de instrucţiuni corespunzătoare şi toate secvenţele de instrucţiuni care urmează, până la întâlnirea instrucţiunii `break` sau până la întâlnirea acoladei închise, care marchează sfârşitul instrucţiunii `switch`. Dacă nici una dintre valorile etichetelor alternativelor `case` nu coincide cu valoarea expresiei, se execută secvenţa de instrucţiuni de pe ramura `default`[13].

Exemplu

În funcţie de valoarea variabilei de tip `char c` (`'+'`, `'-'`, `'*'` sau `'/'`), vom efectua operaţia corespunzătoare între variabilele `x` şi `y`. Dacă variabila `c` are valoarea `'~'` sau `'!'`, vom da mesajul `"Nu e operator binar!"`, iar dacă `c` are orice altă valoare, vom da mesajul `"Eroare"`.

```
switch (c)
{
     case '*':   x *= y; break;
     case '/':   x /= y; break;
     case '+':   x += y; break;
     case '-':   x -= y; break;
     case '~':
     case '!':   cout << "Nu e operator binar!"; break;
     default :   cout << "Eroare!";
}
```

Observaţii

1. Expresia se încadrează obligatoriu între paranteze rotunde.
2. Pe fiecare alternativă `case` este permisă executarea mai multor instrucţiuni.
3. Dacă $\texttt{secventa-instructiuni}_{n+1}$ este vidă, ramura `default` poate lipsi (similar ramurii `else`).
4. Instrucţiunea `switch` este o generalizare a instrucţiunii `if`. Spre deosebire de `if`, care permite selectarea unei alternative din maximum două posibile, `switch` permite selectarea unei alternative din maximum `n+1` posibile. O altă diferenţă majoră constă în faptul că în `if` se execută instrucţiunea corespunzătoare valorii expresiei şi atât, în timp ce în `switch` se execută şi toate secvenţele de instrucţiuni ale alternativelor `case` următoare.

13. În limba engleză, *default* înseamnă lipsă. Deci este alternativa care se alege în lipsă de altceva. Noi o vom numi alternativă implicită.

2.5. *Instrucţiunea* `break`

```
break;
```

Efect

Determină ieşirea necondiţionată din instrucţiunea `switch`, `while`, `for` sau `do-while` în care apare.

2.6. *Instrucţiunea* `while`

```
while (expresie)
  instructiune
```

Efect

Pas 1: se evaluează expresia;

Pas 2: dacă valoarea expresiei este `0`, se iese din instrucţiunea `while`;
dacă valoarea expresiei este diferită de `0`, se execută instrucţiunea, apoi se revine la Pas 1.

Observaţii

1. Instrucţiunea se execută repetat cât timp valoarea expresiei este nenulă. Pentru ca ciclul să nu fie infinit, este obligatoriu ca instrucţiunea care se execută să modifice cel puţin una dintre variabilele care intervin în expresie, astfel încât aceasta să poată lua valoarea `0`, sau să conţină o operaţie de ieşire necondiţionată din ciclu (de exemplu, `break;`).
2. Dacă expresia are de la început valoarea `0`, instrucţiunea nu se execută nici măcar o dată.
3. Sintaxa permite executarea în `while` a unei singure instrucţiuni, prin urmare, atunci când este necesară efectuarea mai multor operaţii, acestea se grupează într-o singură instrucţiune compusă.

Exemplu

Să calculăm răsturnatul numărului natural `n`. Răsturnatul unui număr se obţine considerând cifrele numărului în ordine inversă (de la dreapta la stânga).

```
r=0;                  // in r vom obtine rasturnatul lui n
while (n)
  { r=r*10+n%10;      //"lipesc" ultima cifra din n la r
    n/=10;            //elimin ultima cifra din n
  }
```

Să urmărim pas cu pas execuţia acestei secvenţe de instrucţiuni, pentru n=237:

Linia	Efect	n	r
1	Se iniţializează variabila r cu 0.	237	0
2	Se testează dacă n≠0, apoi se execută instrucţiunea compusă.	237	0
3	Se adaugă la sfârşitul lui r ultima cifră din n.	237	7
4	Se elimină ultima cifră din n.	23	7
2	Se testează dacă n≠0, apoi se execută instrucţiunea compusă.	23	7
3	Se adaugă la sfârşitul lui r ultima cifră din n.	23	73
4	Se elimină ultima cifră din n.	2	73
2	Se testează dacă n≠0, apoi se execută instrucţiunea compusă.	2	73
3	Se adaugă la sfârşitul lui r ultima cifră din n.	2	732
4	Se elimină ultima cifră din n.	0	732
2	Se testează dacă n≠0; se iese din while.	0	732

2.7. *Instrucţiunea* `do-while`

```
do
  instructiune
while (expresie);
```

Efect

Pas 1: se execută instrucţiunea;
Pas 2: se evaluează expresia;
Pas 3: dacă valoarea expresiei este 0, se iese din instrucţiunea repetitivă;
dacă valoarea expresiei este diferită de 0, se revine la Pas 1.

Observaţie

Spre deosebire de `while`, instrucţiunea `do-while` execută instrucţiunea specificată cel puţin o dată, chiar dacă de la început valoarea expresiei este 0, deoarece evaluarea expresiei se face după execuţia instrucţiunii.

În afară de momentul evaluării expresiei (înainte sau după executarea instrucţiunii), alte diferenţe între cele două instrucţiuni repetitive nu există. Prin urmare, preferăm să utilizăm instrucţiunea `while` când este necesar să testăm o condiţie înainte de efectuarea unor prelucrări. Preferăm să utilizăm `do-while` când condiţia depinde de la început de prelucrările din ciclu şi, prin urmare, este necesar să o testăm după executarea instrucţiunii.

Exemplu

Să numărăm cifrele numărului natural memorat în variabila n:


```
nr=0;              //in nr vom obtine numarul de cifre
do
  {n/=10;          //elimin ultima cifra din n
   nr++;}          //o numar
while (n); //actiunea se repeta cat timp n mai are cifre
```

Dacă în locul instrucţiunii `do-while` am fi utilizat o instrucţiune `while`, algoritmul nu ar fi funcţionat şi pentru n=0, deoarece testând condiţia la început, nu s-ar fi executat nici o dată instrucţiunea din ciclu, deci valoarea variabilei `nr` ar fi rămas 0 (greşit! numărul 0 are o cifră!).

2.8. *Instrucţiunea* `for`

```
for (expresie_initializare; expresie_continuare; expresie_reinitializare)
 instructiune
```

Efect

Pas 1: se evaluează expresia de iniţializare;
Pas 2: se evaluează expresia de continuare;
Pas 3: dacă valoarea expresiei de continuare este 0, se iese din instrucţiunea repetitivă `for`;
dacă valoarea expresiei de continuare este diferită de 0:
– se execută instrucţiunea;
– se evaluează expresia de reiniţializare;
– se revine la Pas 2.

Observaţii

Instrucţiunea `for` este tot o instrucţiune repetitivă, ca şi `while` şi `do-while`. Nu este o instrucţiune strict necesară, ea poate fi simulată cu instrucţiunea `while` astfel:

```
expresie_initializare;
while (expresie_continuare)
      {  instructiune
         expresie_reinitializare;  }
```

Totuşi, majoritatea programatorilor preferă să utilizeze instrucţiunea `for`, deoarece este un instrument puternic, flexibil, care ne permite să exprimăm foarte concis prelucrări de natură repetitivă.

Exemplu

Fie n un număr natural (n<10) citit de la tastatură. Să se calculeze:

n!=1*2*...*n (n factorial). Prin definiţie, 0!=1.

Observăm că pentru orice n>0 are loc relaţia:

n!=1*2*...*(n-1)*n=(1*2*...*(n-1))*n=(n-1)!*n

Prin urmare, pentru a calcula valoarea factorialului pentru n este suficient să înmulțim cu n valoarea factorialului pentru n-1. Putem calcula n! din aproape în aproape, utilizând instrucțiunea while astfel:

```
f=i=1;
while (i<=n)
      { f*=i;
        i++; }
```

Aceleași prelucrări pot fi descrise mult mai concis, utilizând instrucțiunea for astfel:

```
for (f=i=1; i<=n; f*=i++);
```

Observați că instrucțiunea care se execută în for este instrucțiunea vidă. Toate prelucrările au fost descrise cu ajutorul celor trei expresii care intervin în for.

Observație

Oricare dintre cele trei expresii care intervin în for poate să fie vidă. Dar și în acest caz, caracterul separator ';' trebuie să apară.

Exemplu

```
for ( ; ; )
    { cin >> n;
      if (n>0) break;  }
```

Observați că toate cele trei expresii sunt vide. Instrucțiunea for citește în mod repetat un număr de la tastatură, până când numărul citit este pozitiv (atunci se iese din for, datorită instrucțiunii break;).

Exercițiu

Scrieți o instrucțiune do-while echivalentă, care să nu utilizeze break;.

2.9. *Aplicații*

Modul

Se introduce de la tastatură un număr întreg x. Scrieți un program care calculează și afișează modulul numărului x.

Soluție

Funcția modul este definită astfel: $|x| = \begin{cases} x, \text{ dacă } x \geq 0 \\ -x, \text{ dacă } x < 0 \end{cases}$

```
#include <stdio.h>
int main(void)
{int x, m;
 scanf("%d", &x);
 if (x<0) m=-x;
    else m=x;
 printf("Modulul este %d\n", m);
 return 0; }
```


Observație

Observați că nu am testat decât condiția x<0. Evident, este suficient. Dacă am pus întrebarea *moneda a căzut cu stema în sus?* și ni s-a răspuns *nu!*, este inutil să mai întrebăm dacă a căzut cu cealaltă față în sus.

Paritate

Se introduce de la tastatură un număr întreg x. Scrieți un program care testează dacă x este un număr par.

Soluție

Numărul x este par dacă este divizibil cu 2 (adică restul împărțirii lui x la 2 este 0). Programul în limbajul C este:

```
#include <stdio.h>
int main(void)
{int x;
 scanf("%d", &x);
 if (x%2) printf("%d este impar\n", x);
    else  printf("%d este par\n", x);
 return 0; }
```

Ecuația de gradul al II-lea

Fie o ecuație de gradul al II-lea cu coeficienți reali $ax^2+bx+c=0$ ($a \neq 0$). Scrieți un program care să rezolve ecuația în mulțimea numerelor reale.

Soluție

Se calculează mai întâi discriminantul ecuației. Dacă acesta este negativ, ecuația nu are rădăcini reale. Dacă acesta este egal cu 0, ecuația are două rădăcini reale egale (confundate). Dacă discriminantul este mai mare decât 0, ecuația are două rădăcini reale distincte.

```
#include <stdio.h>
#include <math.h>
int main(void)
{float a, b, c, d;
 scanf("%f %f %f", &a, &b, &c);
 d=b*b-4*a*c;                    /* calculam discriminantul */
 if (d<0) printf("Ecuatia nu are radacini reale\n");
     else
     if (!d) printf("x1=x2=%.2f\n", -b/(2*a));
        else   /*d este >0 */
        printf("x1=%.2f x2=%.2f\n", (-b+sqrt(d))/(2*a),
                (-b-sqrt(d))/(2*a));
 return 0; }
```

Exercițiu

Modificați programul pentru cazul în care ecuația are gradul cel mult II (caz în care a poate fi 0).

Sumă

Orice sumă de bani S (S>7) poate fi plătită numai cu monede de 3 lei și de 5 lei. Dat fiind S>7, scrieți un program care să determine o modalitate de plată a sumei S numai cu monede de 3 lei și de 5 lei.

Soluție

Problema cere, de fapt, să găsim două numere naturale x și y, astfel încât S să poată fi scris sub forma S=3*x+5*y.

Vom analiza următoarele trei cazuri:

1. S%3 (restul împărțirii lui S la 3) este 0. În acest caz, x=S/3, iar y=0.
2. S%3 este 1. Deoarece S>7, vom considera x=S/3-3 și y=2, pentru că S=3*S/3+1=3*(S/3-3)+3*3+1=3*(S/3-3)+10=3*(S/3-3)+5*2.
3. S%3 este 2. În acest caz vom considera x=S/3-1 și y=1, deoarece S=3*S/3+2=3*(S/3-1)+3+2=3*(S/3-1)+5.

```
#include <stdio.h>
int main(void)
{int S, x, y, r;
 scanf("%d", &S);
 r=S%3;
 switch (r)
        {case 0: x=S/3;   y=0; break;
         case 1: x=S/3-3; y=2; break;
         case 2: x=S/3-1; y=1;          }
 printf("3*%d+5*%d\n",x,y);
 return 0; }
```

Aria triunghiului

Se dau trei numere reale x, y, z. Să se verifice dacă aceste numere pot fi laturile unui triunghi. Dacă da, programul se va termina cu succes și va afișa aria triunghiului. Dacă nu, programul va returna un cod de eroare specific erorii apărute.

Soluție

Condițiile ca x, y și z să constituie laturile unui triunghi sunt:

a. x>0, y>0 și z>0 (în caz contrar, vom returna codul de eroare 1);
b. x+y>z, x+z>y și y+z>x (în caz contrar, vom returna codul de eroare 2).

Cunoscând laturile, putem calcula aria triunghiului utilizând formula lui *Heron*:

$$aria = \sqrt{p \cdot (p-x) \cdot (p-y) \cdot (p-z)},$$

unde cu p am notat semiperimetrul.

Prin urmare, este necesară utilizarea funcției radical. Această funcție este deja implementată în bibliotecile standard ale limbajului C++ și se numește sqrt(). Pentru a o putea utiliza, este suficient să includem în program fișierul antet math.h, în care se găsește declarația funcției:

```
double sqrt(double x);
```

ceea ce înseamnă că funcția are un singur parametru de tip double și întoarce ca rezultat o valoare de tip double (radicalul numărului transmis ca parametru).

```
#include <math.h>
#include <iostream.h>
int main ()
{ double x, y, z, p, aria;
  cin >> x >> y >> z;
  if (x<=0 || y<=0 || z<=0)     return 1;
  if (z≥x+y || y≥x+z || x≥y+z) return 2;
  p=(x+y+z)/2;                          //semiperimetrul
  aria=sqrt(p*(p-x)*(p-y)*(p-z));  //aria
  cout << aria << endl;
  return 0;
}
```

Secvență

Se consideră următorul program în limbajul C:

```
#include <stdio.h>
int main(void)
{int n, a, b, nr, i;
 scanf("%d", &n);              /* 1 */
 scanf("%d", &a);              /* 2 */
 i=1; nr=0;                    /* 3 */
 while (i<n)                   /* 4 */
       { scanf("%d", &b);      /* 5 */
         i++;                  /* 6 */
         if (b%a==0) nr++;     /* 7 */
         a=b;        }         /* 8 */
printf("nr=%d\n",nr);}         /* 9 */
```

a. Ce se va afișa dacă se citesc valorile 6, 4, 8, 7, 21, 3, 6?
b. Introduceți o succesiune de cel puțin trei valori, astfel încât programul să afișeze valoarea 0.
c. Care este efectul acestui program?

Soluție

a. Se cere să verificăm efectul programului pentru un set particular de date de intrare. În acest caz, vom executa instrucțiunile algoritmului pas cu pas, urmărind valorile variabilelor care intervin în algoritm. Pentru simplitate, am numerotat liniile de la 1 la 9.

Linia	n	a	b	nr	i	Explicație
1	6	?	?	?	?	Se citește valoarea 6 și se atribuie variabilei n. Celelalte variabile au valori necunoscute.
2	6	4	?	?	?	Se citește valoarea 4 și se atribuie variabilei a.
3	6	4	?	0	1	Se inițializează variabilele nr și i cu valorile 0, respectiv 1.
4	6	4	?	0	1	Se testează dacă valoarea variabilei i (1) este mai mică decât valoarea variabilei n (6). Deoarece expresia i<n are valoarea adevarat, se continuă execuția algoritmului cu instrucțiunea 5.
5	6	4	8	0	1	Se citește valoarea 8 și se atribuie variabilei b.
6	6	4	8	0	2	Se mărește valoarea variabilei i cu 1.
7	6	4	8	1	2	Testăm dacă restul împărțirii variabilei b la variabila a este 0 (adică dacă a divide b). Deoarece 4 divide 8, valoarea variabilei nr se mărește cu 1.
8	6	8	8	1	2	Variabilei a i se atribuie valoarea variabilei b, apoi se revine la instrucțiunea de pe linia 4.
4	6	8	8	1	2	Se testează dacă valoarea variabilei i (2) este mai mică decât valoarea variabilei n (6). Deoarece expresia i<n are valoarea adevarat, se continuă execuția algoritmului cu instrucțiunea de pe linia 5.
5	6	8	7	1	2	Se citește valoarea 7 și se atribuie variabilei b.
6	6	8	7	1	3	Se mărește valoarea variabilei i cu 1.
7	6	8	7	1	3	Testăm dacă restul împărțirii variabilei b la variabila a este 0 (adică dacă a divide b). Deoarece 8 nu divide 7, valoarea variabilei nr nu se mărește cu 1.
8	6	7	7	1	3	Variabilei a i se atribuie valoarea variabilei b, apoi se revine la instrucțiunea de pe linia 4.
4	6	7	7	1	3	Se testează dacă valoarea variabilei i (3) este mai mică decât valoarea variabilei n (6). Deoarece expresia i<n are valoarea adevarat, se continuă execuția algoritmului cu instrucțiunea de pe linia 5.
5	6	7	21	1	3	Se citește valoarea 21 și se atribuie variabilei b.
6	6	7	21	1	4	Se mărește valoarea variabilei i cu 1.
7	6	7	21	2	4	Testăm dacă restul împărțirii variabilei b la variabila a este 0 (adică dacă a divide b). Deoarece 7 divide 21, valoarea variabilei nr se mărește cu 1.

8	6	21	21	2	4	Variabilei a i se atribuie valoarea variabilei b, apoi se revine la instrucțiunea de pe linia 4.
4	6	21	21	2	4	Se testează dacă valoarea variabilei i (4) este mai mică decât valoarea variabilei n (6). Deoarece expresia i<n are valoarea adevarat, se continuă execuția algoritmpului cu instrucțiunea de pe linia 5.
5	6	21	3	2	4	Se citește valoarea 3 și se atribuie variabilei b.
6	6	21	3	2	5	Se mărește valoarea variabilei i cu 1.
7	6	21	3	2	5	Testăm dacă restul împărțirii variabilei b la variabila a este 0 (adică dacă a divide b). Deoarece 21 nu divide 3, valoarea variabilei nr nu se mărește cu 1.
8	6	3	3	2	5	Variabilei a i se atribuie valoarea variabilei b, apoi se revine la instrucțiunea de pe linia 4.
4	6	3	3	2	5	Se testează dacă valoarea variabilei i (5) este mai mică decât valoarea variabilei n (6). Deoarece expresia i<n are valoarea adevarat, se continuă execuția algoritmului cu instrucțiunea de pe linia 5.
5	6	3	6	2	5	Se citește valoarea 6 și se atribuie variabilei b.
6	6	3	6	2	6	Se mărește valoarea variabilei i cu 1.
7	6	3	6	3	6	Testăm dacă restul împărțirii variabilei b la variabila a este 0 (adică dacă a divide b). Deoarece 3 divide 6, valoarea variabilei nr se mărește cu 1.
8	6	6	6	3	6	Variabilei a i se atribuie valoarea variabilei b, apoi se revine la instrucțiunea de pe linia 4.
4	6	6	6	3	6	Se testează dacă valoarea variabilei i (6) este mai mică decât valoarea variabilei n (6). Deoarece expresia i<n are valoarea fals, se iese din ciclu și se continuă execuția algoritmului cu instrucțiunea de pe linia 9.
9	6	6	6	3	6	Se afișează valoarea variabilei nr (3).

b. Vom alege o secvență de exact trei valori. Observăm că prima valoare din secvență este egală cu numărul de valori care urmează, deci va trebui să fie 2. Pentru ca valoarea variabilei nr să rămână 0 (așa cum este inițial), va trebui să alegem două valori astfel încât condiția din instrucțiunea 7 să nu fie îndeplinită (adică prima valoare să nu dividă pe cea de-a doua). Prin urmare, o soluție posibilă este 2 5 7.

c. Urmărind algoritmul pas cu pas, deducem că se citește mai întâi o valoare n, apoi se citesc succesiv n numere naturale. Permanent avem memorate în variabilele a și b două numere consecutiv citite dintre cele n. La fiecare pas se testează dacă primul număr (cel memorat în variabila a) îl divide pe cel de-al doilea (cel memorat în variabila b) și dacă da, valoarea variabilei nr este incrementată (mărită cu 1). Prin urmare, algoritmul numără în variabila nr câte perechi de elemente

consecutive din secvența citită au proprietatea că primul element din pereche îl divide pe cel de-al doilea element din pereche.

Calcul

Se consideră următorul program C:

```
#include <stdio.h>
int main(void)
{int n, x, i;
 scanf("%d%d",&n,&x);          /* 1 */
 for (i=1; i<=n; i++)          /* 2 */
       x=x*x;                  /* 3 */
 printf("x=%d",x);             /* 4 */
 return 0; }
```

a. Ce se va afișa pe ecran pentru `n=3` și `x=2`?
b. Scrieți un program echivalent, care să utilizeze instrucțiunea repetitivă `while`.
c. Care este efectul acestui program?

Soluție

a. Pentru a determina efectul algoritmului pentru `n=3` și `x=2`, vom urmări pas cu pas execuția algoritmului, ilustrând în tabelul următor valorile variabilelor care intervin în acest algoritm.

Linia	n	x	i	Explicație
v	3	2	?	Se citește valoarea 3 și se atribuie variabilei n, apoi se citește valoarea 2 și se atribuie variabilei x. Valoarea variabilei i este deocamdată nedefinită.
2	3	2	1	Se inițializează variabila i cu 1. Deoarece valoarea variabilei i ≤ valoarea variabilei n, se continuă cu instrucțiunea de pe linia 3.
3	3	4	1	Se modifică valoarea variabilei x, se revine la linia 2.
2	3	4	2	Se mărește valoarea variabilei i cu 1. Deoarece valoarea variabilei i ≤ valoarea variabilei n, se continuă cu instrucțiunea de pe linia 3.
3	3	16	2	Se modifică valoarea variabilei x, se revine la linia 2.
2	3	16	3	Se mărește valoarea variabilei i cu 1. Deoarece valoarea variabilei i ≤ valoarea variabilei n, se continuă cu instrucțiunea de pe linia 3.
3	3	256	3	Se modifică valoarea variabilei x, se revine la linia 2.
2	3	256	4	Se mărește valoarea variabilei i cu 1. Deoarece valoarea variabilei i > valoarea variabilei n, se iese din ciclu și se continuă cu instrucțiunea de pe linia 4.
4	3	256	4	Se afișează valoarea 256.

b.

```
#include <stdio.h>
int main(void)
{int n, x, i;
 scanf("%d%d",&n,&x);
 i=1;
 while (i<=n)
       { x=x*x; i++;}
 printf("x=%d",x);
 return 0; }
```

Observați că înainte de a intra în ciclu am inițializat valoarea variabilei `i` cu `1`, iar în ciclu am inclus o instrucțiune care să incrementeze la fiecare pas valoarea variabilei `i`.

c. Pentru a determina efectul general al acestui program vom aplica un raționament inductiv. Deoarece valoarea variabilei x se modifică la fiecare pas, pentru a fi ușor de urmărit evoluția acestei variabile, vom nota cu `x0` valoarea inițială a variabilei x.

n	x	Explicație
0	$x0$	Dacă pentru n se citește valoarea 0, instrucțiunea `for` nu va fi executată nici o dată, deci x rămâne cu valoarea sa citită inițial (x0).
1	$x0^2$	Dacă pentru n se citește valoarea 1, instrucțiunea `for` va fi executată o singură dată, deci x va avea valoarea sa $x0^2$.
2	$x0^4$	Dacă pentru n se citește valoarea 2, instrucțiunea `for` va fi executată de două ori; după prima execuție, x va avea valoarea sa $x0^2$, iar după cea de a doua, $x0^4$.
3	$x0^8$	Dacă pentru n se citește valoarea 3, instrucțiunea `for` va fi executată de trei ori; după prima execuție, x va avea valoarea sa $x0^2$, după cea de-a doua, $x0^4$, iar după cea de-a treia, $x0^8$.

Urmărind execuția programului pentru câteva valori ale lui n, putem formula propoziția inductivă:

„După n pași, variabila x va avea valoarea $x0^{2^n}$".

Să demonstrăm propoziția: „După n+1 pași, variabila x va avea valoarea $x0^{2^{n+1}}$".

Observăm că la pasul n+1 variabilei x i se atribuie valoarea variabilei x de la pasul n înmulțită cu ea însăși, adică $x0^{2^n} * x0^{2^n} = x0^{2^n + 2^n} = x0^{2^{n+1}}$.

Numărare

Se consideră următorul program în limbajul C:

```
#include <stdio.h>
int main(void)
{int a, b, nr, i;
 scanf ("%d%d", &a, &b);      /* 1 */
 nr=0;                        /* 2 */
 for (i=a; i<=b; i++)         /* 3 */
     if (i%2==0) nr++;        /* 4 */
 printf("%d", nr);            /* 5 */
 return 0;}
```

a. Ce se va afişa pe ecran pentru a=5 şi b=8?
b. Scrieţi un program echivalent, care să utilizeze instrucţiunea do-while.
c. Care este efectul acestui program?
d. Scrieţi un program echivalent mai eficient.

Soluţie

a. Vom urmări pas cu pas execuţia acestui program:

Linia	a	b	nr	i	Explicaţie
1	5	8	?	?	Se citesc valorile 5 şi 8 şi se atribuie în ordine variabilelor a şi b. Variabilele nr şi i au deocamdată o valoare nedefinită.
2	5	8	0	?	Se iniţializează variabila nr cu 0.
3	5	8	0	5	Se iniţializează variabila i cu valoarea variabilei a (5). Deoarece valoarea variabilei i este ≤ valoarea variabilei b, se continuă cu instrucţiunea de pe linia 4.
4	5	8	0	5	Se testează dacă restul împărţirii variabilei i la 2 este 0 (adică dacă i este număr par). Deoarece i este impar, valoarea variabilei nr rămâne nemodificată. Se revine la linia 3.
3	5	8	0	6	Se măreşte valoarea variabilei i cu 1. Deoarece valoarea variabilei i este ≤ valoarea variabilei b, se continuă cu instrucţiunea de pe linia 4.
4	5	8	1	6	Se testează dacă restul împărţirii variabilei i la 2 este 0 (adică dacă i este număr par). Deoarece i este par, valoarea variabilei nr se măreşte cu 1. Se revine la linia 3.
3	5	8	1	7	Se măreşte valoarea variabilei i cu 1. Deoarece valoarea variabilei i este ≤ valoarea variabilei b, se continuă cu instrucţiunea de pe linia 4.
4	5	8	1	7	Se testează dacă restul împărţirii variabilei i la 2 este 0 (adică dacă i este număr par). Deoarece i este impar, valoarea variabilei nr rămâne nemodificată. Se revine la linia 3.

3	5	8	1	8	Se măreşte valoarea variabilei i cu 1. Deoarece valoarea variabilei i este ≤ valoarea variabilei b, se continuă cu instrucţiunea de pe linia 4.
4	5	8	2	8	Se testează dacă restul împărţirii variabilei i la 2 este 0 (adică dacă i este număr par). Deoarece i este par, valoarea variabilei nr se măreşte cu 1. Se revine la linia 3.
3	5	8	2	9	Se măreşte valoarea variabilei i cu 1. Deoarece valoarea variabilei i este > valoarea variabilei b, se iese din instrucţiunea repetitivă şi se continuă cu instrucţiunea de pe linia 5.
3	5	8	2	9	Se afişează valoarea variabilei nr (2).

b. Va trebui să simulăm instrucţiunea for cu ajutorul instrucţiunii do while:

```
#include <stdio.h>
int main(void)
{int a, b, nr, i;
 scanf ("%d%d", &a, &b);
 nr=0; i=a;
 if (i<=b)
    do
      { if (i%2==0) nr++;
        i++;}
    while (i<=b);
 printf("%d", nr);
 return 0;}
```

Observaţi că a fost necesar să testăm separat, înainte de a intra în instrucţiunea repetitivă do while, dacă i<=b. Dacă nu am fi pus acest test şi dacă iniţial am fi citit o valoare pară pentru variabila a strict mai mare decât valoarea pentru variabila b, atunci acest-program ar fi afişat (incorect) valoarea 1, în loc de 0. Acest lucru se datorează faptului că instrucţiunea compusă subordonată instrucţiunii do while s-ar fi executat o dată şi s-ar fi mărit valoarea variabilei nr cu 1.

c. Observăm că variabila i primeşte în ordine toate valorile naturale din intervalul [a, b] (spunem că „variabila i parcurge intervalul [a, b]"). Pentru fiecare număr natural din intervalul [a, b], se testează dacă el este par şi, dacă da, se măreşte valoarea variabilei nr cu 1. Prin urmare, în variabila nr numărăm valorile pare din intervalul [a, b].
d. Două programe sunt echivalente dacă pentru orice set de date de intrare produc aceleaşi date de ieşire.

Un program echivalent, dar mai eficient de a număra valorile pare dintr-un interval dat [a, b], nu necesită parcurgerea tuturor numerelor din interval. În primul rând, observăm că în intervalul [a, b] sunt exact L=b-a+1 numere naturale. În al doilea rând, observăm că aproximativ jumătate din valorile din intervalul [a, b] sunt pare şi restul sunt impare. Am spus aproximativ jumătate deoarece trebuie să fim atenţi la paritatea capetelor intervalului. Mai exact, va trebui să analizăm patru cazuri posibile:

a	b	nr	Explicație
par	impar	L/2	Se formează exact L/2 perechi de numere naturale consecutive (par, impar).
impar	par	L/2	Se formează exact L/2 perechi de numere naturale consecutive (impar, par).
par	par	1+L/2	Se formează exact L/2 perechi de numere naturale consecutive (par, impar), la care se mai adaugă b, care este, de asemenea, par.
impar	impar	L/2	Se formează exact L/2 perechi de numere naturale consecutive (impar, par), b fiind impar, nu va fi numărat.

Programul eficient de a număra valorile pare din intervalul [a, b] este:

```
#include <stdio.h>
int main(void)
{int a, b, nr, i;
 scanf ("%d%d", &a, &b);
 nr=0;
 if (a<=b)
    {nr=(b-a+1)/2;
     if (a%2==0 && b%2==0) nr++;   }
 printf("%d", nr);
 return 0;}
```

Putere

Fie n un număr natural şi x un număr real. Scrieţi un program care calculează pe x^n.

Soluţie

$x^n=x*x*...*x=(x*x*...*x)*x=x^{n-1}*x$

(primul grup: de n ori; al doilea grup: de n-1 ori)

Evident, $x^0=1$.

Vom utiliza o variabilă p în care vom calcula puterea. La fiecare pas i (i variind de la 1 la n) vom înmulţi valoarea variabilei p cu x.

```
#include <stdio.h>
int main(void)
{int n, i;
 float x, p=1;
 scanf("%d %f", &n, &x);
 for (i=1; i<=n; i++) p*=x;
 printf("%.2f", p);
 return 0;}
```


Observaţie

Acest program efectuează exact n înmulţiri.

Numere

Numerele întregi mai mari decât 1 sunt aşezate pe 5 coloane, câte 4 pe linie, ca în exemplul următor:

2	3	4	5	
	9	8	7	6
10	11	12	13	
	17	16	15	14
...				

Coloanele sunt numerotate de la 1 la 5. Scrieţi un program care să citească de la tastatură un număr natural mai mare decât 1 şi care să afişeze coloana pe care este aşezat numărul citit.

Soluţie

Observăm că în coloanele 1, 3 şi 5 sunt numai numere pare. În prima coloană sunt plasate numerele 2, 8, 18,... deci sunt numere de forma 8k+2. În coloana 5 sunt plasate numerele 6, 14, 22,... deci numere de forma 8k+6. În coloana 3 sunt plasate numerele 4, 8, 12, 16,... deci numere de forma 8k+4.

În coloanele 2 şi 4 sunt plasate numere impare. Multiplii lui 8 sunt întotdeauna pe coloana 3 pe o linie pară, deci în coloana 2 pe liniile pare sunt numere de forma 8k+1. Pe coloana 2, pe liniile impare sunt numere de forma 8k+3. Prin urmare, în coloana 4, pe liniile pare sunt numere de forma 8k+7, iar pe liniile impare, numere de forma 8k+5.

```
#include <iostream.h>
int main()
{int x;
 cout<<"x="; cin>>x;
 switch (x%8)
       {
        case 1: cout<<2<<endl; break;
        case 2: cout<<1<<endl; break;
        case 3: cout<<2<<endl; break;
        case 4: cout<<3<<endl; break;
        case 5: cout<<4<<endl; break;
        case 6: cout<<5<<endl; break;
        case 7: cout<<4<<endl; break;
        case 0: cout<<3<<endl; break;
        }
return 0;}
```

Coduri şi caractere

Scrieţi un program care să afişeze toate caracterele din codul ASCII, precum şi codul lor. Se va scrie mai întâi caracterul, apoi codul ASCII corespunzător separat printr-un spaţiu, câte două grupe cod-caracter pe o linie, separate prin tab.

Soluţie

O primă idee ar fi să parcurgem succesiv toate codurile ASCII, (numere naturale de la 0 la 255). Pentru fiecare cod, afişăm caracterul corespunzător, conform formatului specificat în problemă.

Pentru a obţine pe ecran caracterul corespunzător codului ASCII este necesar să utilizăm operatorul de forţare a tipului.

```
#include <iostream.h>
int main()
{int cod;
 for (cod=0; cod <=255; cod++)
    {cout<<(unsigned char)cod<<' '<< cod;
     cout<<'\t'; cod++;
     cout<<(unsigned char)cod<<' '<< cod;
     cout<<endl;     }
return 0; }
```

Dacă vom executa acest program, vom avea nişte surprize, datorate unora dintre primele 26 de caractere, care sunt caractere speciale. Mai exact:

Cod	Explicaţie	Corecţie
7	Nu generează nici un efect pe ecran, doar emite un sunet.	scriem \a în locul acestui caracter
8	Caracterul *backspace* care determină deplasarea cursorului ecran cu o poziţie la stânga.	scriem \b în locul acestui caracter
9	Caracterul tab orizontal determină deplasarea cursorului cu 8 poziţii la dreapta.	scriem \t în locul acestui caracter
10	Determină trecerea cursorului pe linia următoare în aceeaşi coloană.	scriem \l în locul acestui caracter
11	Caracterul tab vertical determină deplasarea cursorului în jos.	scriem \v în locul acestui caracter
12	Când este transmis către imprimantă se sare la începutul paginii următoare.	scriem \f în locul acestui caracter
13	Determină trecerea cursorului la începutul liniei curente.	scriem \r în locul acestui caracter
26	Reprezintă caracterul EOF (*end of file*) care reprezintă marcajul de sfârşit de fişier.	scriem eof în locul acestui caracter
27	Caracterul *escape*.	scriem spaţiu în locul acestui caracter

Prin urmare va trebui să tratăm separat aceste cazuri particulare.

```
#include <iostream.h>
int main()
{int cod;
 for (cod=0; cod <=255; cod++)
     {switch (cod)
           {case 7:  cout<<"\\a   " ;  break;
            case 8:  cout<<"\\b   " ;  break;
            case 9:  cout<<" \\t   " ; break;
            case 10: cout<<"\\l   " ;  break;
            case 11: cout<<" \\v   "; break;
            case 12: cout<<"\\f   ";  break;
            case 13: cout<<" \\r   "; break;
            case 26: cout<<"eof ";    break;
            case 27: cout<<"      ";  break;
            default: cout<<(unsigned char)cod<<"    ";
           }
      cout<< cod;
      if (cod %2) cout<<'\n';
          else cout<<'\t';
     }
return 0;}
```

Exerciţiu

Programul precedent afişează pe ecran 128 de linii, prin urmare noi vom putea analiza doar ultimele 25, celelalte defilând cu viteză. Modificaţi programul precedent astfel încât să pagineze rezultatele afişate (după umplerea unui ecran, să afişeze un mesaj de tipul „Apăsaţi o tastă pentru a continua", apoi să aştepte acţionarea unei taste).

Secvenţă simetrică

Fie n un număr natural nenul. Să se afişeze pe ecran o secvenţă de lungime n, construită după procedeul ilustrat în exemplele următoare.

n=7

1	2	3	4	3	2	1

n=8

1	2	3	4	4	3	2	1

Soluție

În primul rând, observăm că secvența începe cu numerele de la 1 la n/2 în ordine crescătoare. Dacă n este impar, urmează numerele de la 1 la (n+1)/2 în ordine descrescătoare. Dacă n este par, urmează numerele de la 1 la n/2 în ordine descrescătoare.

```
#include <iostream.h>
int main()
{int n, i;
 cout<<"n= "; cin>>n;
 for (i=1; i<=n/2; i++)     cout<<i<<' ';
 for (i=(n+n%2)/2; i; i--) cout<<i<<' ';
 cout<<endl; return 0; }
```

Maxim

I. Fie a și b două numere reale, citite de la tastatură. Scrieți un program care determină și afișează pe ecran cea mai mare valoare citită.

Soluție

Compar valorile celor două variabile citite și o afișez pe cea mai mare:

```
#include <stdio.h>
int main(void)
{double a, b;
 scanf("%lf %lf", &a, &b);
 if (a<b) printf("max=%.2lf\n", b);
    else printf("max=%.2lf\n", a);
 return 0; }
```

O altă soluție, care are avantajul că o vom putea generaliza pentru oricâte elemente, este de a reține maximul într-o variabilă (să o numim max). Inițializăm variabila max cu a (inițial putem considera că a este „cel mai bun"). Apoi comparăm maximul cu valoarea variabilei b și dacă max este mai mic, îl schimbăm.

```
#include <stdio.h>
int main(void)
{double a, b, max;
 scanf("%lf %lf", &a, &b);
 max=a;
 if (max<b) max=b;
 printf("max=%.2lf\n", max);
 return 0; }
```

II. Fie a, b și c trei numere reale, citite de la tastatură. Scrieți un program care determină și afișează cea mai mare valoare citită.

Soluție

Ideea utilizată în cea de-a doua variantă a programului precedent o putem utiliza și pentru trei variabile:


```
#include <stdio.h>
int main(void)
{double a, b, c, max;      /* maximul dintre a, b si c */
 scanf("%lf %lf %lf", &a, &b, &c);
 max=a;                    /* initializare */
 if (max<b) max=b;         /* compar cu b */
 if (max<c) max=c;         /* compar cu c */
 printf("max=%.2lf\n", max);
 return 0; }
```

III. Se citește de la tastatură un număr natural nenul n, apoi se citesc succesiv n valori reale. Scrieți un program care determină și afișează cea mai mare valoare reală citită.

Soluție

O primă problemă este declararea datelor de intrare: se dau n valori reale, numai că de data aceasta n nu este constant, ci variabil (citit de la tastatură). Cum să declarăm un număr variabil de variabile? Nu putem și nici nu este necesar, pentru că nu este nevoie să citim fiecare număr într-o variabilă separată! Vom citi numerele succesiv în aceeași variabilă (să o numim a) și imediat ce le-am citit le vom compara cu elementul maxim.

Pentru a intui mai bine algoritmul de calcul al maximului, să ne imaginăm un șir de elevi care intră în clasă unul câte unul. Am dori ca pe măsură ce intră să reținem care este cel mai înalt. Evident, primul care intră poate fi considerat cel mai înalt (de altfel, este singurul pe care l-am văzut până acum). Când intră pe ușă un nou elev, compar înălțimea lui cu înălțimea celui pe care eu îl consider cel mai înalt de până acum. Dacă noul elev este mai înalt, reținînălțimea lui ca etalon pentru viitoarele comparații (el este cel mai înalt de până acum). Prin urmare, noi reținem permanent maximul dintre cei intrați deja.

Această idee o vom utiliza în continuare: în variabila max vom reține la fiecare moment cea mai mare valoare citită până la momentul respectiv.

```
#include <stdio.h>
int main(void)
{double a, max;  /* max - maximul dintre valorile citite */
 int n, i;
 /* i - numarul de valori citite pana la un moment dat*/
 scanf("%d", &n);
 scanf("%lf", &max);
 /* initializam max cu primul element citit */
 for (i=1; i<n; i++)
    {scanf("%lf", &a); /* citesc o noua valoare */
     if (max<a) max=a;} /* compar cu max valoarea citita*/
 printf("max=%.2lf\n", max);
 return 0; }
```

Observaţii

1. În instrucţiunea `for` a fost necesară efectuarea mai multor operaţii (citirea unei valori şi compararea acesteia cu valoarea maximă de până acum). Prin urmare, am grupat instrucţiunile corespunzătoare într-o singură instrucţiune compusă.
2. Pentru a calcula maximul a `n` numere a fost necesară executarea instrucţiunii compuse subordonate instrucţiunii `for` de `n-1` ori.

Exerciţii

1. Rescrieţi programul precedent utilizând instrucţiunea repetitivă `while`.
2. Modificaţi programul precedent astfel încât să calculeze cel mai mic dintre elementele citite.
3. Modificaţi programul precedent astfel încât să calculeze cele mai mari două elemente dintre cele citite.

Media aritmetică

Se citeşte de la tastatură un număr natural nenul `n`, apoi se citesc succesiv `n` valori reale. Scrieţi un program care determină şi afişează media aritmetică a valorilor strict pozitive.

Soluţie

Media aritmetică se calculează împărţind suma valorilor strict pozitive la numărul lor.

În mod asemănător cu problema precedentă, vom citi succesiv cele `n` valori întregi în aceeaşi variabilă `a`. După fiecare citire, vom verifica dacă valoarea citită este strict pozitivă şi, în caz afirmativ, o vom număra şi o vom adăuga la sumă.

```
#include <stdio.h>
int main(void)
{double a, s=0;
 int n, i=0, nr=0;
 /*in i retinem cate valori am citit pana la un moment dat;
   in nr calculam cate valori pozitive am citit; in s
   calculam suma valorilor strict pozitive citite */
 scanf("%d", &n);
 while (i<n) /* cat timp nu am citit toate cele n valori */
    {scanf("%lf",&a);  /* citesc o noua valoare */
     i++;              /* numar valoarea citita */
     if (a>0)          /* valoarea este strict pozitiva */
        {s+=a;         /* o adun la suma */
         nr++;}        /* o numar */
    }
 if (nr) printf("media artimetica=%.2lf\n", s/nr);
    else printf("media aritmetica nu se poate calcula\n");
 return 0; }
```


Suma cifrelor

Se citeşte de la tastatură un număr natural `n`. Să se calculeze suma cifrelor lui `n`.

Soluţie

Cel mai uşor se obţine ultima cifră din număr (cifra unităţilor): `n%10` (restul împărţirii lui `n` la `10`). După ce utilizăm ultima cifră (o adăugăm la sumă), nu mai avem nevoie de ea şi o eliminăm din număr (`n=n/10`). Cifra zecilor a devenit acum ultima cifră. Repetăm procedeul, cât timp numărul mai are cifre.

```
#include <stdio.h>
int main(void)
{int n, s=0;
 scanf("%d", &n);
 while (n)              /* cat timp mai exista cifre in n */
       { s+=n%10;       /* adun la s ultima cifra a lui n */
         n/=10; }       /* elimin ultima cifra a lui n*/
 printf("suma cifrelor=%d\n", s);
 return 0; }
```

Observaţie

Iniţializarea unei variabile în care trebuie să calculăm o sumă se face întotdeauna cu 0 (elementul neutru la adunare). În mod similar, iniţializarea unei variabile în care trebuie să calculăm un produs se face întotdeauna cu `1` (elementul neutru la înmulţire).

Exerciţii

1. Urmăriţi pas cu pas execuţia acestui program pentru `n=5672`.
2. Modificaţi programul astfel încât să determine numărul de cifre pare ale lui `n`.

Perechi

Fie `n` un număr natural nenul. Să se genereze toate perechile (`a`, `b`), cu proprietatea că `a | b`, unde `a` şi `b` sunt numere naturale nenule mai mici decât `n`.

Soluţie

În primul rând, observăm că pentru ca `a` să dividă `b` trebuie ca $a \le b$. O primă idee ar fi ca `a` să parcurgă toate numerele naturale nenule mai mici decât `n`, apoi pentru fiecare valoare `a`, `b` să parcurgă toate numerele mai mari sau egale cu `a` şi mai mici decât `n`, formând astfel toate perechile de numere (`a`, `b`) cu $a \le b < n$. Pentru fiecare pereche vom testa dacă `a | b`.

```
#include <stdio.h>
int main(void)
{int n, a, b;
 scanf("%d", &n);
 for (a=1; a<n; a++)
     for (b=a; b<n; b++)
         if (b%a==0) printf("(%d, %d) ", a, b);
 return 0; }
```

În acest program instrucțiunea `if` este executată de `n*(n-1)/2` ori, deoarece:
- când `a=1`, a doua instrucțiune `for` se execută de `n-1` ori;
- când `a=2`, a doua instrucțiune `for` se execută de `n-2` ori;
- când `a=3`, a doua instrucțiune `for` se execută de `n-3` ori;

(...)
- când `a=n-1`, a doua instrucțiune `for` se execută o dată;

În total: `1+2+3+...+n-1=n*(n-1)/2` executări ale instrucțiunii `if`.

O idee mai bună ar fi ca, având `a` fixat, să nu mai parcurgem cu variabila `b` toate numerele naturale de la `a` la `n-1`, ci doar multiplii lui `a`:

```
#include <stdio.h>
int main(void)
{int n, a, m;
 scanf("%d", &n);
 for (a=1; a<n; a++)
     for (m=1; a*m<n; m++)
         printf("(%d, %d) ", a, a*m);
 return 0; }
```

În acest mod, programul a devenit mai eficient, obținând direct perechile corecte.

Divizori

Dat fiind `n` un număr natural, să se determine toți divizorii naturali ai lui `n`.

Soluție

În primul rând, observăm că orice număr natural `n` are ca divizori pe `1` și pe `n` (aceștia se numesc divizori improprii). În afară de divizorii improprii, cel mai mic divizor ar putea fi `2`, iar cel mai mare `n/2`. Prin urmare, pentru a afla divizorii proprii ai lui `n`, vom parcurge toate numerele naturale din intervalul `[2, n/2]` și vom testa pentru fiecare număr natural `x` din acest interval dacă divide sau nu pe `n` (adică dacă restul împărțirii lui `n` la `x` este `0`).

```
#include <stdio.h>
int main(void)
{int n, x;
 scanf("%d", &n);
 printf("1 ");
 for (x=2; x<=n/2; x++)     /* parcurg intervalul [2,n/2] */
     if (n%x==0)            /* x divide n? */
         printf("%d ", x); /* x este divizor, il scriu */
 printf("%d\n", n);
 return 0; }
```

Și aici se poate da o soluție mai bună utilizând faptul că „dacă `d` este divizor al lui `n`, atunci și `n/d` este divizor a lui `n`". Procedând în acest mod, `x` va trebui să parcurgă doar intervalul `[2, √n ]`.


```
#include <stdio.h>
#include <math.h>
int main(void)
{int n, x, rad;
 scanf("%d", &n);
 rad=sqrt(n);        /*calculam radicalul o singura data */
 printf("1 ");
 for (x=2; x<=rad; x++)  /* parcurgem intervalul [2, √n ] */
   if (n%x==0)           /* x divide n? */
     printf("%d %d ", x, n/x);  /*x si n/x sunt divizori */
 printf("%d\n", n);
 return 0; }
```

Observăm că nu este necesar să calculăm radicalul la fiecare evaluare a expresiei din instrucțiunea `for`. Este suficient să îl calculăm o dată și să îl memorăm într-o variabilă pentru a-l utiliza ori de câte ori este necesar.

Număr prim

Fie `n` un număr natural, `n>1`, citit de la tastatură. Scrieți un program care verifică dacă numărul `n` este prim.

Soluție

Pentru a testa dacă numărul `n` este prim, vom verifica dacă are sau nu divizori proprii (divizori diferiți de `1` și de `n`). Pentru aceasta vom parcurge intervalul numerelor naturale de la `2` la $\sqrt{n}^2$, verificând pentru fiecare număr dacă este sau nu divizor al lui `n`. Pentru a reține rezultatul testului, vom utiliza o variabilă denumită `prim` cu semnificația „`prim` este `1`, dacă `n` este prim, și `0`, dacă `n` nu este prim". Inițial, variabilei `prim` îi atribuim valoarea `1` („prezumția de nevinovăție").

Pe parcursul verificării, dacă găsim un divizor, variabilei `prim` îi atribuim valoarea `0` („dovada" că `n` nu este prim a fost găsită).

Verificarea continuă cât timp avem *unde* căuta (nu am ajuns la $\sqrt{n}$) și avem *de ce* căuta (nu am găsit încă „dovada vinovăției" – un divizor al lui `n`).

```
#include <stdio.h>
#include <math.h>
int main(void)
{int n, d=2, prim=1;
 scanf("%d", &n);
 while (d<=sqrt(n) && prim)
       if (n%d==0) prim=0;     /* am gasit un divizor */
         else d++;             /* cautam mai departe */
  if (prim) printf("%d este prim", n);
    else printf("%d nu este prim", n);
 return 0; }
```

Am putea îmbunătăți performanțele acestui program observând că singurul număr prim par este 2. Vom testa în primul rând dacă n este par (caz în care, exceptându-l pe 2, nu este prim), apoi vom verifica numai numerele impare.

```
#include <stdio.h>
#include <math.h>
int main(void)
{int n, d, prim=1, rad;
 scanf("%d", &n);
 if (n>2)
    if (n%2==0) prim =0;
       else
        for (d=3, rad=sqrt(n); d<=rad && prim; d+=2)
           if (n%d==0) prim=0;    /* am gasit un divizor */
  if (prim) printf("%d este prim", n);
     else printf("%d nu este prim", n);
 return 0; }
```

Exerciții

1. Urmăriți pas cu pas execuția acestui program pentru n=13, n=8 și n=49.
2. Modificați programul astfel încât să afișeze toate numerele naturale prime ≤ n.

Descompunere în factori primi

Fie n un număr natural (n>1), citit de la tastatură. Scrieți un program care să afișeze descompunerea numărului natural n în factori primi.

De exemplu, pentru n=720, programul va afișa 2^4*3^2*5^1, ceea ce înseamnă că divizorii primi sunt 2, 3 și 5, având respectiv multiplicitățile 4, 2 și 1.

Soluție

Vom aplica algoritmul învățat la matematică: considerăm toate numerele naturale începând cu 2. Pentru fiecare număr, verificăm dacă este divizor al lui n. În caz afirmativ, calculăm multiplicitatea acestui divizor în n, împărțind succesiv pe n la divizor și calculând numărul de împărțiri efectuate, după care afișăm atât divizorul găsit, cât și multiplicitatea sa. Procedeul se repetă cât timp n mai are divizori (n>1).

```
#include <stdio.h>
int main(void)
{int n, d, m;
 scanf("%d", &n);
 for (d=2; n>1; d++)
     {/* calculez multiplicitatea lui d in n */
       for (m=0; n%d==0; m++, n/=d);
       if (m) /* d este divizor */
         printf("Divizorul %d, multiplicitatea %d\n",d,m);
     }
 return 0; }
```


Exerciții

1. Urmăriți pas cu pas execuția acestui program pentru n=2500.
2. Observați că nu am testat nicăieri dacă d este prim. Nu este necesar, deoarece datorită faptului că atunci când găsim un divizor al lui n îl împărțim pe n la divizor ori de câte ori este posibil, orice divizor nou găsit este prim. Demonstrați această afirmație prin reducere la absurd.
3. Modificați programul astfel încât să afișeze numai cel mai mare divizor prim.
4. Modificați programul astfel încât să calculeze numărul total de divizori ai lui n.

Cel mai mare divizor comun

Fie n și m două numere naturale, citite de la tastatură. Scrieți un program care să calculeze și să afișeze c.m.m.d.c. (n, m).

Soluție

Vom aplica algoritmul lui Euclid, învățat la matematică. Cât timp m≠0, împărțim pe n la m și calculăm restul; la pasul următor, împărțitorul devine deîmpărțit, iar restul devine împărțitor. Ultimul rest≠0 este c.m.m.d.c.

De exemplu, pentru a=495 și b=720, *cel mai mare divizor comun* prin algoritmul lui Euclid se obține astfel:

Deîmpărțit		Împărțitor	·	Rest	
495	=	720	· 0 +	495	
720	=	495	· 1 +	225	
495	=	225	· 2 +	45	
225	=	45	· 5 +	0	⇒ c.m.m.d.c.(495,720)=45

```
#include <stdio.h>
int main(void)
{int n, m, rest;
 scanf("%d %d", &n, &m);
 while (m)
       { rest=n%m;      /* calculez restul                */
         n=m;           /* impartitorul devine deimpartit */
         m=rest; }      /* restul devine impartitor       */
 printf("cmmdc=%d", n);/* ultimul rest diferit de 0      */
 return 0; }
```

Exerciții

1. Urmăriți pas cu pas execuția acestui program pentru n=121 și m=4950.
2. Observați că nu am testat nicăieri dacă inițial n>m. Nu este necesar, deoarece, dacă inițial n ar fi mai mic decât m, după prima executare a instrucțiunii compuse subordonate instrucțiunii while, valorile variabilelor n și m vor fi interschimbate.

3. Modificați acest program astfel încât să afișeze etapele de determinare a c.m.m.d.c. ca în exemplul de mai sus.
4. Scrieți un program echivalent care să utilizeze instrucțiunea repetitivă `do while`.
5. Modificați acest program astfel încât să simplifice fracția `n/m` și să afișeze fracția ireductibilă obținută.
6. Modificați acest program astfel încât să verifice dacă numerele `n` și `m` sunt prime între ele.
7. Modificați acest program astfel încât să calculeze `c.m.m.m.c.(n,m)` (cel mai mic multiplu comun al numerelor `n` și `m`).
8. Modificați acest program astfel încât să determine $\Phi(n)$ (indicatorul lui Euler[14]). Prin definiție, $\Phi(n)$ este numărul de numere naturale mai mici decât `n`, relativ prime cu `n`. Două numere sunt relativ prime dacă cel mai mare divizor comun al lor este `1`.

Gard

Doi copii vopsesc un gard alcătuit din `n` scânduri (numerotate de la `1` la `n`, $n \leq 100000$) astfel: primul ia o cutie de vopsea roșie cu care vopsește scândurile cu numărul `p`, `2p`, `3p` etc. Al doilea procedează la fel, începe de la același capăt al gardului, dar ia o cutie de vopsea albastră și vopsește din `q` în `q` scânduri. Astfel, când vor termina de vopsit, gardul va avea multe scânduri nevopsite, unele scânduri vopsite în roșu, altele în albastru, iar altele în violet (cele care au fost vopsite și cu roșu și cu albastru).

Cunoscând numerele `n`, `p` și `q`, scrieți un program care să afișeze:
- câte scânduri rămân nevopsite;
- câte scânduri sunt vopsite în roșu;
- câte scânduri sunt vopsite în albastru;
- câte scânduri sunt vopsite în violet.

De exemplu, pentru `n=25, p=4, q=6` se vor afișa valorile `17 4 2 2`.

(Olimpiada Județeană de Informatică pentru Gimnaziu 2003, clasa a VI-a)

Soluție

O primă idee ar fi să parcurgem succesiv toate scândurile (adică toate numerele naturale de la `1` la `n`) și să verificăm pentru fiecare scândură următoarele cazuri:
- dacă numărul scândurii este divizibil cu `p` și este divizibil și cu `q`, atunci scândura va fi violet (și mărim cu `1` numărul de scânduri de culoare violet);
- dacă numărul scândurii este divizibil numai cu `p`, atunci scândura va fi roșie (și mărim cu `1` numărul de scânduri de culoare roșie);

14. Leonhard Euler (1707-1783) a fost unul dintre cei mai prolifici matematicieni din istorie. A scris 866 de lucrări și a câștigat de 12 ori Premiul Academiei Franceze.

- dacă numărul scândurii este divizibil numai cu `q`, atunci scândura va fi albastră (și mărim cu `1` numărul de scânduri de culoare albastră);
- dacă nici una dintre condițiile precedente nu a fost îndeplinită, atunci scândura va rămâne nevopsită (și mărim cu `1` numărul de scânduri nevopsite).

```
#include <iostream.h>
int main()
{long int n, i, p, q, nec, ros, alb, vio;
 cout<<"Numarul de scanduri n= "; cin>>n;
 cout<<"p, q = "; cin>>p>>q;
 nec=ros=alb=vio=0;
 for (i=1; i<=n; i++)
     if (i%p==0 && i%q==0) vio++;
        else
        if (i%p==0) ros++;
           else
           if (i%q==0) alb++;
              else nec++;
cout<<nec<<' '<<ros<<' '<<alb<<' '<<vio<<endl;
return 0;}
```

O idee mai eficientă ar fi să observăm că scândurile vopsite în violet (deci cele ale căror numere trebuie să fie divizibile cu `p` și cu `q`) trebuie, de fapt, să fie divizibile cu cel mai mic multiplu comun al numerelor `p` și `q`. Să notăm cu `m c.m.m.m.c.(p,q)`. Observăm că numărul scândurilor violet este `n/m`. Numărul scândurilor roșii este `n/p`, din care eliminăm pe cele care au fost vopsite în violet, iar numărul scândurilor albastre este `n/q`, din care eliminăm pe cele care au fost vopsite în violet.

```
#include <iostream.h>
int main()
{long int n, m, p, q, a, b, r, nec, ros, alb, vio;
 cout<<"Numarul de scanduri n= "; cin>>n;
 cout<<"p, q = "; cin>>p>>q;
 a=p; b=q;          //calculam m=cmmmc(p,q)
 while (b) {r=a%b; a=b; b=r;}
 m=p*q/a;
 vio=n/m;
 ros=n/p - vio;
 alb=n/q - vio;
 nec=n-ros-alb-vio;
 cout<<nec<<' '<<ros<<' '<<alb<<' '<<vio<<endl;
 return 0;}
```

Exponent

Fie `n` un număr natural și `p` un număr prim. Să se determine cel mai mare număr natural `k` astfel încât p^k divide `n`.

Soluție

Problema cere practic să determinăm multiplicitatea factorului prim p în n.

```
#include <iostream.h>
int main ( void )
{int n, p, k=0;
 cout<<"n= "; cin>>n;
 cout<<"p= "; cin>>p;
 for (k=0; n%p==0; k++, n/=p);
 cout<<"k= "<<k<<endl;
 return 0; }
```

Termen Fibonacci

Fie șirul Fibonacci[15] 1, 1, 2, 3, 5, 8, 13, 21, 34,...

Observați că primii doi termeni sunt egali cu 1, fiecare termen următor fiind egal cu suma celor doi termeni care îl precedă.

Fie n un număr natural citit de la tastatură. Scrieți un program care afișează cel de-al n-lea termen din șirul Fibonacci.

Soluție

Observăm că pentru a calcula un termen al șirului Fibonacci avem nevoie de doi termeni precedenți. Vom utiliza trei variabile, f0, f1, și f2. În f0 și f1 vom reține doi termeni Fibonacci consecutivi, iar în f2 vom calcula termenul următor. Variabilele f0 și f1 vor fi inițializate cu 1 (primii doi termeni din șirul Fibonacci). Următorii termeni (de la al doilea până la al n-lea) îi vom calcula succesiv. La fiecare pas i, în f2 vom calcula al i-lea termen din șirul Fibonacci (f2=f1+f0), apoi ne pregătim pentru pasul următor (f0=f1, f1=f2):

```
#include <stdio.h>
int main(void)
{
int n, i, f0=1, f1=1, f2=1;
scanf("%d", &n);
for (i=2; i<=n; i++)
      { f2=f1+f0;   /* calculez termenul curent */
        f0=f1;      /* ma deplasez in sir       */
        f1=f2; }
printf("Cel de-al %d-lea termen Fibonacci este %d\n",n,f2);
return 0; }
```

15. Acest șir a fost introdus în 1202 de către Leonardus filius Bonaccii Pisanus (Leonardo, fiul lui Bonacci Pisanul) pentru rezolvarea următoarei probleme: Să se determine câte perechi de iepuri se vor naște într-un an dintr-o singură pereche de iepuri, știind că fiecare pereche de iepuri dă naștere lunar la o nouă pereche de iepuri, o pereche devine fertilă în decurs de o lună, iar iepurii nu mor pe parcursul anului. Șirul Fibonacci are multe aplicații, în variate domenii. De exemplu, numerele Fibonacci sunt utilizate în probleme de sortare și căutare sau în strategii de joc.

Exerciții

1. Urmăriți pas cu pas execuția acestui program pentru n=10.
2. Modificați algoritmul astfel încât să afișeze cel de-al n-lea termen din șirul Lucas[16]. Primii termeni din șirul Lucas sunt 1, 3, 4, 7, 11, 18, 29, 47,...
3. Modificați algoritmul astfel încât să calculeze suma primilor n termeni din șirul Fibonacci.
4. Modificați algoritmul astfel încât să calculeze cel mai mare termen din șirul Fibonacci mai mic sau egal cu n.

Existență

Se citesc de la tastatură două numere naturale nenule n și p, apoi se citesc succesiv n valori întregi. Scrieți un program care să verifice dacă printre cele n valori citite există multipli ai lui p.

Soluție

Vom citi succesiv cele n valori întregi în aceeași variabilă a. După fiecare citire vom verifica dacă valoarea citită este divizibilă cu p și, în caz afirmativ, vom reține faptul că există un multiplu al lui p. Rezultatul verificării îl vom reține în variabila exista. Inițial, valoarea acestei variabile este 0 (deoarece încă nu am descoperit nici un multiplu al lui p). Dacă pe parcursul citirii, vom descoperi un multiplu al lui p, variabila exista va primi valoarea 1.

```
#include <stdio.h>
int main(void)
{int n, p, a, exista=0, i;
 scanf("%d %d", &n, &p);
 for (i=1; i<=n; i++)
     {scanf("%d",&a); /* citesc un nou termen      */
      if (a%p==0)     /* am descoperit un multiplu */
         exista=1; }
 if (exista) printf("Exista un multiplu al lui %d", p);
   else printf("Nu exista un multiplu al lui %d", p);
 return 0; }
```

Exercițiu

1. Urmăriți pas cu pas execuția acestui program pentru șirul de valori 6, 3, 7, 4, 1, 12, 4, 5.
2. Modificați programul astfel încât să calculeze numărul de multipli ai lui p existenți în șir.

16. François-Edouard-Anatole Lucas (1842-1891) matematician francez cu importante contribuții în teoria numerelor.

Număr divizori primi

Se citeşte de la tastatură un număr natural nenul n, apoi se citesc succesiv n valori întregi. Scrieţi un program care să verifice dacă oricare număr dintre cele n citite are un număr impar de divizori primi.

Soluţie

Vom citi succesiv cele n valori în aceeaşi variabilă x. Pentru fiecare valoare citită, vom determina numărul de divizori primi (folosind un algoritm asemănător cu descompunerea în factori primi, numai că în loc să afişăm divizorii primi şi multiplicităţile acestora, vom număra divizorii).

Rezultatul verificării îl vom reţine în variabila oricare. Iniţial, această variabilă are valoarea 1 (deoarece încă nu am găsit un număr care să nu aibă un număr impar de divizori primi). Dacă, pe parcursul citirii, vom întâlni un număr cu număr par de divizori primi, variabilei oricare îi vom atribui valoarea 0.

```
#include <stdio.h>
int main()
{unsigned n, x, d, nr, i, oricare=1;
 scanf("%u", &n);
 for (i=1; i<=n; i++)
    {scanf("%u",&x);       /* citesc o noua valoare     */
     nr=0;
     /*numarul de divizori primi ai lui x este initial 0 */
      d=2;              /*determin divizorii primi ai lui x */
      while (x>1)
          {if (x%d==0)        /* am descoperit un divizor */
              {nr++;
               /* numar acest divizor prim, apoi
               il elimin din x, prin impartiri repetate  */
               while (x%d==0) x/=d;  }
           d++; }             /* caut divizori mai departe */
      if (nr%2==0) oricare=0;} /* pentru ca am gasit o
                    valoare cu numar par de divizori primi */
if (oricare) printf("DA");
   else printf("NU");
return 0; }
```

Vocale

Se citeşte de la tastatură o propoziţie scrisă cu litere mici, terminată cu '.'. Scrieţi un program care calculează şi afişează numărul de vocale din propoziţie.

Soluţie

Citim succesiv caracterele din care este constituită propoziţia, până la sfârşit, în aceeaşi variabilă (să o numim c). Pentru fiecare caracter citit, verificăm dacă este vocală ('a', 'e', 'i', 'o' sau 'u') şi dacă da, îl numărăm.


```
#include <stdio.h>
int main(void)
{unsigned nr=0;                          /* numarul de vocale */
 char c;/*in c vom citi succesiv caracterele propozitiei*/
 do
  {c=getchar();                           /*citim un caracter */
   if (c=='a' || c=='e' || c=='i' || c=='o' || c=='u')
       nr++;}                        /* c este vocala, o numar */
 while (c!='.');
 printf("Numarul de vocale este %u\n",nr);
 return 0;}
```

Expresie

Fie n un număr natural şi x un număr real, citite de la tastatură. Să se calculeze valoarea expresiei[17]:

$$E(n,x)=1+\frac{x}{1!}+\frac{x^2}{2!}+...+\frac{x^n}{n!}$$

Soluţie

Expresia este o sumă formată din termeni care au aceeaşi formă. Mai exact, termenul i din sumă ($i=\overline{0,n}$) este o fracţie cu numitorul egal cu i! şi numărătorul egal cu x^i. Să urmărim calculul acestei expresii pentru n=3:

Pas	Factorial	Putere	Expresie
0	0!=1	$x^0=1$	1/1=1
1	1!=0!*1-1	$x^1= x^0*x$	1+x/1=1+x
2	2!=1!*2=2	$x^2= x^1*x$	$1+x/1+x^2/2$
3	3!=2!*3=6	$x^3= x^2*x$	$1+x+x^2/2+x^3/6$

Observăm că la fiecare pas i trebuie să calculăm i! (înmulţind factorialul de la pasul precedent cu i), x^i (înmulţind puterea de la pasul precedent cu x) şi să adăugăm un nou termen la expresie ($x^i/i!$).

```
#include <iostream.h>
int main()
{ unsigned long n, i, f;  float s, x, p;
  cout << "n, x= "; cin >> n >> x;
  for (f=s=p=i=1; i<=n; i++)
      { f*=i;         //actualizez factorialul
```

17. Această expresie este remarcabilă, deoarece poate fi utilizată pentru calculul aproximativ al lui ex (funcţia exponenţială).

```
        p*=x;          //actualizez puterea
        s+=p/f;    }   //adun termenul i la suma
    cout << "Expresia:" << s << endl;
    return 0; }
```

Radical

Fie x un număr real. Să se calculeze $\sqrt[3]{x}$ (radical de ordin 3 din x), folosind relația de recurență:

$r_1=1$

$r_i=(2*r_{i-1} + x/(r_{i-1}*r_{i-1}))/3$

Rezultatul trebuie să fie obținut cu o precizie mai bună decât 10^{-4}.

Soluție

Observăm că pentru a calcula termenul i din acest șir recurent este necesar numai termenul i-1. Prin urmare, vom utiliza două variabile r1 (în care reținem termenul calculat la pasul precedent) și r2 (în care reținem termenul pe care urmează să îl calculăm).

Pentru a obține precizia cerută în problemă, vom calcula termeni din șir până când diferența în valoare absolută dintre doi termeni consecutivi este mai mică decât precizia specificată. Pentru a calcula diferența în valoare absolută dintre doi termeni consecutivi, vom utiliza funcția fabs() din biblioteca de funcții matematice (această funcție calculează modulul unui număr de tip double specificat ca parametru).

```
#include <iostream.h>
#include <math.h>
#define EPS 0.0001
int main()
{ double x, r1, r2=1;
  cout<<"x= "; cin>>x;
  do
   {r1=r2;
    r2=(2*r1+x/(r1*r1))/3;}
  while (fabs (r2-r1)>=EPS);
  cout<< r2<<endl;
  return 0;}
```

Relație

Fie x un număr natural nenul. Să se determine toate perechile de numere naturale m, n care satisfac relația $m^2*(n+1)=x$.

Soluție

Observăm că pentru a fi satisfăcută această relație trebuie ca n+1 și m^2 să fie divizori ai lui x. Prin urmare, vom considera toate pătratele perfecte care divid pe x.


```
#include <iostream.h>
int main()
{int x, i;
 cout<<"x= "; cin>>x;
 for (i=1; i*i<=x; i++)
    if (x % (i*i)==0)
       cout<<"m= "<<i<<' '<<"n= "<<(x/(i*i)-1)<<endl;
 return 0; }
```

Sumă de două numere prime

Colegul de bancă a făcut la ora de matematică o afirmație interesantă: „orice număr natural n>3 se poate scrie ca sumă de două numere prime". El nu a demonstrat afirmația, așa că nu prea vă vine să credeți. Cum nu sunteți prea bun la matematică, v-ați gândit la o altă soluție: „inducția informatică". Prin urmare, scrieți un program care să citească VMax număr natural și care să verifice dacă pentru orice număr natural ≤Vmax este adevărată această proprietate.

Soluție

Parcurgem numerele naturale din intervalul [4,VMax] și pentru fiecare număr încercăm să determinăm o soluție. Pentru oricare număr n din intervalul specificat, considerăm toate perechile de forma (a, n-a), a≤n-a, până când găsim o pereche în care ambele valori sunt prime sau până când am epuizat toate perechile.

```
#include <iostream.h>
#include <math.h>
int main()
{int n, a, rad, d, prim, exista, VMax, oricare=1;
 cout <<"VMax= "; cin>>VMax;
 for (n=4; n<=VMax && oricare; n++)
     {for (exista=0, a=2; a<=n/2 && !exista; a++)
        {//testam daca a este prim
         prim=1;
         if (a>2 && a%2==0) prim=0;
            else
         for (rad=sqrt(a), d=3; prim&&d<=rad; d+=2)
             if (a%d==0) prim=0;
         if (prim)
            {//testam daca n-a este prim
             prim=1;
             if (n-a>2 && (n-a)%2==0) prim=0;
                else
                for (rad=sqrt(n-a),d=3; prim&&d<=rad; d+=2)
                   if ((n-a)%d==0) prim=0;
             if (prim) exista=1;} //am gasit o varianta
        }
      if (!exista) oricare=0; }
```

```
 if (oricare) cout<<"Proprietatea este verificata\n";
     else
     {cout<<"Proprietatea nu este verificata\n";
      cout<<"Pentru n= "<<n<<" nu exista solutie\n";}
 return 0;
}
```

Progres

Tatăl lui Gigel este foarte atent la performanțele școlare ale fiului său. El își notează toate notele obținute de fiul său, apoi realizează tot felul de statistici. Astăzi analizează șirul notelor și vrea să determine lungimea celei mai lungi secvențe crescătoare, secvență de note care corespunde unei perioade de progres școlar. Cum notele sunt deja foarte multe, el are nevoie de un program care să citească numărul de note și secvența notelor și să afișeze lungimea celei mai lungi secvențe crescătoare de note.

Soluție

Vom citi succesiv notele. Pentru a putea compara două note consecutive, vom utiliza două variabile: în variabila `a` rețin nota precedentă, iar în variabila `b` rețin nota curentă. La fiecare nouă citire, compar nota curentă cu cea precedentă. Dacă nota curentă este cel puțin egală cu nota precedentă, secvența curentă de note este crescătoare, deci măresc lungimea ei (memorată în variabila `lg`). Dacă nota curentă este mai mică decât nota precedentă, deducem că s-a terminat o secvență crescătoare, compar lungimea ei cu lungimea maximă (memorată în variabila `lgmax`) și reinițializez variabila `lg` cu `1` (nota curentă este prima notă dintr-o altă secvență).

```
#include <iostream.h>
int main()
{int n, i, a=0, b, lg=0, lgmax=0;
 cout<<"nr de note = "; cin>>n;
 cout<<"Introduceti notele: ";
 for (i=1; i<=n; i++)
     {cin>>b; //citesc o nota
      if (a<=b) lg++; //maresc lungimea secventei curente
          else        //s-a terminat o secventa crescatoare
           {if (lgmax<lg) //compar lg cu lungimea maxima
               lgmax=lg;
            lg=1;}        //incepe o noua secventa
     a=b; //nota curenta devine nota precedenta
    }
 //compar si lungimea ultimei secvente
 if (lgmax<lg) lgmax=lg;
 cout<<"Lungimea maxima = "<<lgmax<<endl;
 return 0;}
```


Tabla adunării

Scrieți un program care să îl ajute pe un copil de clasa I să învețe tabla adunării până la 100. Programul va genera aleator două numere naturale mai mici decât 100, le va afișa pe ecran și îl va întreba pe copil cu cât este egală suma lor. Dacă răspunsul copilului este corect, îl felicită. Dacă răspunsul copilului este greșit, îl întreabă dacă dorește să mai încerce sau dacă dorește să afle răspunsul corect. Programul repetă testarea până copilul se plictisește și spune că nu mai vrea!

Soluție

Programul va genera aleator două numere întregi mai mici decât 100. Pentru aceasta, va trebuie să inițializăm generatorul de numere aleatoare, cu ajutorul funcției `randomize()` declarate în fișierul antet `stdlib.h`:

```
void randomize(void);
```

ceea ce înseamnă că funcția nu are parametri și nu returnează nici un rezultat.

Pentru a genera un număr aleator, vom utiliza funcția `random()`, declarată în fișierul antet `stdlib.h`:

```
int random(int x);
```

Funcția returnează o valoare aleatoare de tip `int`, cuprinsă între `0` și `x-1`.

După generarea aleatoare a celor două numere și afișarea lor pe ecran, programul va citi răspunsul copilului și îl va compara cu suma celor două numere:

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
int main ()
{ char raspuns;
  int x, y, suma;
  cout<<"Sa invatam tabla adunarii! Esti pregatit? (D/N)";
  raspuns=getche();
  if (raspuns =='N' || raspuns =='n')
      cout<<"\nMai gandeste-te! Eu zic sa incercam!\n";
  randomize();//initializez generatorul de numere aleatoare
  do
    {x=random(100);y=random(100);//generez aleator numerele
     cout<<'\n'<<x<<'+'<<y<<"=?\b"; raspuns = 'D';
     do
       {cin >> suma;     //citeste raspunsul elevului
        if (suma==x+y) cout << "\n Excelent raspuns!!!";
           else
           {cout <<"\nNu! Mai incerci o data?(D/N) ";
             raspuns=getche();}
       }
     while (suma!=x+y && (raspuns=='D'|| raspuns =='d'));
     if (raspuns=='N'|| raspuns =='n')
```

```
        cout <<"\nRaspunsul corect era "<<x+y<<endl;
    cout <<"\nContinuam? (D/N) "; raspuns=getche();
  }
while (raspuns=='D'||raspuns =='d');
cout <<"La revedere!!!"<<endl;
return 0;}
```

Exercițiu

Utilizând funcții din fișierul antet `conio.h`, faceți acest program cât mai atrăgător (scrieți mesajele colorat pe o culoare de fond vioaie, realizați și puțină animație). Dacă doriți, puteți utiliza și mici semnale sonore (cu ajutorul funcțiilor `sound()` și `nosound()` din fișierul antet `dos.h`).

Semne

Fie `n` un număr natural nenul citit de la tastatură. Scrieți un program care să determine un număr natural `k` și o combinație de $\overline{1,k}$ semne + sau – (mai exact o succesiune $x_1, x_2, ..., x_k$ unde $x_i \in \{-1, 1\}$, $\forall i=\overline{1,k}$) astfel încât să aibă loc relația: $n=x_1*1^2+x_2*2^2+...+x_k*k^2$. Programul va afișa pe ecran o succesiune de `k` semne + sau – care să îndeplinească relația de mai sus. De exemplu, pentru `n=3`, o soluție este `-+---`. Pentru `n=2`, o soluție este `---+`.

Soluție

Analizând problema pentru câteva cazuri particulare, observăm că succesiunea de semne `+--+` produce întotdeauna valoarea `4`, deoarece:

$k^2-(k+1)^2-(k+2)^2+(k+3)^2=4, \ \forall k>0.$

Prin urmare, orice număr `n` divizibil cu `4` se poate scrie ca o succesiune de `n/4` secvențe `+--+`. Dacă numărul nu este divizibil cu `4`, vom scrie în prealabil o secvență de semne egală cu valoarea restului.

```
#include <iostream.h>
int main()
{ int n;
  cout << "n= "; cin >> n;
  switch (n%4)
  { case 1: cout << "+"; break;
    case 2: cout << "---+"; break;
    case 3: cout << "-+--+";  }
  for (int i=0; i<n/4; i++) cout << "+--+";
  return 0; }
```

Excursie

La Olimpiada Națională de Informatică, organizatorii vor să realizeze o excursie în care să participe toți cei `n` concurenți. Fiecare concurent are asociat în mod unic

un număr între `1` și `n`. Sponsorii au pus la dispoziție trei mijloace de transport impunând însă o condiție: *suma numerelor de ordine ale participanților din fiecare mijloc de transport să fie aceeași*, în caz contrar, excursia nu va avea loc.

Scrieți un program care să distribuie pe cei `n` concurenți în cele trei mijloace de transport.

(Concursul Interjudețean „Urmașii lui Moisil", Iași, 2001)

Soluție

În primul rând, observăm că suma valorilor din fiecare mașină trebuie să fie `n*(n+1)/6` (pentru că `1+2+3+...+n=n*(n+1)/2`).

În al doilea rând, observăm că există cazuri pentru care nu există soluție (pentru `n<5` sau pentru acele valori ale lui `n` pentru care suma `n*(n+1)/2` nu este divizibilă cu 3, pentru că trebuie să obținem 3 grupe de sumă egală).

Analizăm separat cazurile `n=5, 6, 8, 9` și observăm că pentru fiecare dintre acestea există soluție:

n	n=5	n=6	n=8	n=9
Grupe	(5) (2,3) (1,4)	(2,5) (3,4) (1,6)	(1,2,3,6) (5,7) (4,8)	(1,2,3,4,5) (7,8) (6,9)

Pentru orice altă valoare a lui `n` pentru care există soluție, construim soluția plecând de la unul dintre aceste 4 cazuri elementare, adăugând grupe de câte 6 elemente consecutive, distribuite astfel: `(a+1, a+6)`, `(a+2, a+5)`, `(a+3, a+4)`.

```
#include <iostream.h>
int main()
{long int n, Start, x, i;
 cin>>n;
 if (n<5||n*(n+1)%3) {cout<<"Nu exista solutie"; return 0;}
 if (!((n-9)%6)){Start=10; x=(n-9)/6;}
 if (!((n-8)%6)){Start= 9; x=(n-8)/6;}
 if (!((n-6)%6)){Start= 7; x=(n-6)/6;}
 if (!((n-5)%6)){Start= 6; x=(n-5)/6;}
 switch (Start)                                    //masina 1
  {case 10: cout<<"1 2 3 4 5 "; break;
   case  9: cout<<"1 2 3 6 "; break;
   case  7: cout<<"3 4 "; break;
   case  6: cout<<5<<' '; break;                  }
 for (i=0;i<x;i++) cout<<Start+6*i<<' '<<Start+5+6*i<<' ';
 cout<<endl;
 switch (Start)                                    //masina 2
   {case 10: cout<<7<<' '<<8<<' '; break;
    case  9: cout<<5<<' '<<7<<' '; break;
    case  7: cout<<2<<' '<<5<<' '; break;
    case  6: cout<<2<<' '<<3<<' '; break;         }
for (i=0;i<x;i++) cout<<Start+1+6*i<<' '<<Start+4+6*i<<' ';
cout<<endl;
```

```
switch (Start)                                          //masina 3
 {case 10: cout<<6<<' '<<9<<' '; break;
  case  9: cout<<4<<' '<<8<<' '; break;
  case  7: cout<<1<<' '<<6<<' '; break;
  case  6: cout<<1<<' '<<4<<' '; break;          }
for (i=0;i<x;i++) cout<<Start+2+6*i<<' '<<Start+3+6*i<<' ';
cout<<endl;
return 0; }
```

2.10. *Probleme propuse*

1. Fie x un număr real. Scrieți un program C care să calculeze valoarea funcției: $F(x)=\max\{|2x-1|, 9-x^2\}$.

2. Care este efectul următoarei secvențe de instrucțiuni?

```
int a, b, c, x;
a=3; b=5; c=7;
if (a-b/2<0) x=1; else if (a+b-c/2<b) x=2; else if
(a%b+c>b) x=3; else x←4;
cout<<x;
```

3. Ce valoare inițială ar putea avea variabila x, astfel încât la sfârșitul execuției următoarei secvențe de instrucțiuni variabila y să aibă valoarea 2?

```
if (x>3) if (x<7) if (x%2==0) y=1; else y=2; else y=3;
else y=4;
```

4. Fie a și b două numere întregi. Scrieți un program care să verifice dacă a și b sunt numere consecutive.

5. Fie ecuația cu coeficienți reali $ax^2+bx+c=0$ ($a\neq 0$). Scrieți un program care, fără a calcula rădăcinile ecuației, să determine natura și semnul acestora.

6. Fie x un număr natural de trei cifre. Scrieți un program care să elimine una dintre cifrele numărului astfel încât numărul de două cifre rămas să fie maxim.

7. Scrieți un program care să citească 3 caractere și să determine câte caractere distincte s-au citit.

8. Fie a și b două unghiuri, ale căror măsuri sunt exprimate în grade, minute și secunde. Să se scrie un program care să calculeze și să afișeze măsura sumei celor două unghiuri.

9. Se citesc de la tastatură a, b și c, trei numere reale pozitive. Scrieți un program care să verifice dacă numerele citite pot constitui laturile unui triunghi dreptunghic. În caz afirmativ, calculați și afișați aria triunghiului.

10. Fie x și y două numere reale, citite de la tastatură. Scrieți un program care calculează și afișează valoarea funcției:

$$f(x,y)=\begin{cases}\dfrac{x+y}{5xy}, \text{dacă } x,y>0\\ \min(x,y), \text{dacă } x=0 \text{ sau } y=0\\ \left(\dfrac{1}{x}+\dfrac{1}{y}\right)\left(\dfrac{1}{x}+\dfrac{1}{y}+x^2+y^2\right), \text{altfel}\end{cases}$$

11. Scrieți un program care citește de la tastatură trei numere întregi strict pozitive a, b și c, numere cu cel mult trei cifre fiecare. Valoarea variabilei a reprezintă distanța în km dintre orașul A și orașul B, b distanța în km dintre orașul B și orașul C, iar c reprezintă distanța în km dintre orașul C și orașul A. Știind că un călător își planifică o vizită a celor trei orașe pornind din oricare din orașele A sau B și ajungând, în final, în oricare din orașele B sau C cu trecere prin cel de-al treilea oraș, să se determine un traseu de lungime minimă care respectă aceste condiții. Programul va afișa cele trei litere corespunzătoare celor trei orașe, în ordinea în care sunt vizitate. Se va alege o metodă cât mai eficientă din punctul de vedere al gestionării memoriei. De exemplu, pentru a=58, b=140, c=125, se va afișa BAC.

(Bacalaureat 2002, sesiune specială)

12. Să se scrie un program care să rezolve sistemul de două ecuații de gradul I, cu două necunoscute și coeficienți reali:

$$\begin{cases}a_1x+b_1y=c_1\\ a_2x+b_2y=c_2\end{cases}$$

13. Un elev este declarat promovat la Bacalaureat dacă la fiecare dintre cele cinci probe de examen a luat cel puțin nota 5, iar media sa generală este cel puțin 6. Date fiind cele 5 note pe care elevul le-a obținut la bacalaureat, scrieți un program care să verifice dacă elevul a promovat sau nu examenul de Bacalaureat.

14. Ionel are H1 cm, Gigel are H2 cm, iar Dănuț are H3 cm. Scrieți un program care să afișeze numele celor trei copii în ordinea crescătoare a înălțimii.

15. Care este efectul următoarelor secvențe de instrucțiuni?

a.

```
int x=0, a=6, b=4, c=5, d=1;
if (a>b) if (b>c) if (c>d) x=1; else x=2;
      else x=3; else if (a>c) x=4; else x=5;
cout  << x;
```

b.

```
int n=5;
while (--n) cout << n;
```

c.

```
int n=0;
while (--n) cout << n;
```

d.
```
int n=20;
for (int i=0; i<n; i++)
    if (!(i%5)) cout<<i;
```

e.
```
for (int i=0; i<5; i++)
    for (int j=i+1; j<5; j++)
        cout<<i<<j<<' ';
```

f.
```
int n=2;
do if (n%2) cout << n++;
while (++n % 2);
```

g.
```
int n=-5;
do if (n++) break; while (1);
cout << n;
```

h.
```
int n=8, i=0, j=1;
for (int k=0; k++ < n; j=i+j, i=j-i)
    cout << j << ' ';
```

16. Care este valoarea variabilei întregi `a` pentru care următoarea secvență de program va afișa exact un caracter `'A'`?

(variantă Bacalaureat, 2000)

```
x=5;
do   {cout << 'A'; x++; } while (x>a);
```

a. 12 **b.** 5 **c.** 6 **d.** 4 **e.** 1

17. Fie `a` și `b` două numere naturale, a≤b. Care dintre următoarele secvențe de instrucțiuni calculează în variabila `nr` numărul de valori pare din intervalul `[a,b]`?

a.
```
nr=0;
for (i=a; i<=b; i++)
    if (!i%2) nr++;
```

b.
```
nr=0;
while (b>=a)
   if (b%2==0) nr++; b--;
```

c.
```
nr=(b-a+1)/2+(b-a+1)%2;
```

d.
```
nr=(b-a+1)/2;
if (a%2 && b%2) nr--;
```

e.
```
nr=(b-a+1)/2;
if (a%2==0&&b%2==0) nr++;
```

f.
```
for (nr=i=0; i<=b-a;i++)
     if ((i+a)%2==0) nr++;
```

18. Se consideră următoarea secvență de instrucțiuni:

```
while(i%2)
     {cout<<i%2;
     i/=2;}
```


Care dintre valorile următoare pot fi considerate valoare inițială pentru variabila `i`, astfel încât pe ecran să se afișeze `111`?

a. 7 **b.** 128 **c.** 15 **d.** 135 **e.** 117 **f.** 39

19. Câte caractere # afișează secvența următoare?

```
int a, b, c, n=5;
for (a=1; a<=n; a++)
     for (b=1; b<=n; b+=2)
          for (c=b+1; c<=n; c++)
               cout<<'#';
```

20. Care dintre următoarele secvențe de instrucțiuni atribuie variabilei de tip `int x` valoarea 2^n, cu `n` număr natural (n<16), variabila auxiliară `i` fiind de tip `int`?

a.
```
x=1;
for (i=1; i<=n; i++) x *= n;
```

b.
```
x=1;
for (i=0; i<n; i++) x *= 2;
```

c.
```
x=1;
for (i=1; i<=n; i++) x <<= 1;
```

d.
```
x=2;
for (i=1; i<=n; i++) x*=i;
```

e.
```
x=1; i=0;
while (i<n) x*=2; i++;
```

f.
```
x=1;
x << n;
```

21. Ce valoare inițială ar trebui să aibă variabila întreagă `x` astfel încât după execuția următoarei secvențe pe ecran să fie afișată valoarea 2?

```
y=0;
while (x)
 {y+=x%10; x=x/10;}
if (y<5) if (y>2) if (y>4) cout<<1; else cout<<2; else
cout<<3; else cout<<4;
```

a. 111 **b.** 21 **c.** 113 **d.** 3 **e.** 2 **f.** 312

22. Fie `n` și `i` două variabile întregi. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni?

```
for (n=9, i=0; i<=n; i++) {i++; n--;}
cout<<n<<i;
```

a. 910 **b.** 99 **c.** 58 **d.** 67 **e.** 57 **f.** 56

23. Secvența următoare trebuie să testeze dacă numărul natural `x` este pătrat perfect.

```
for (ok=1, d=2; x>1 && ok; )
    {for (m=0; x%d==0; m++, x/=d);
     if  (...) ok=0; else d++;     }
if (ok) cout<<"Este patrat perfect.\n";
   else cout<<"Nu este patrat perfect.\n";
```

Cum trebuie completată secvența care lipsește (marcată cu . . .)?

a. m==0 **b.** m==1 **c.** m%2==0 **d.** m%2

24. Completați secvențele lipsă (marcate cu ...[1] și ...[2]) din algoritmul următor, astfel încât să se afișeze numărul de numere naturale din intervalul [x,y] care sunt prime cu y.

```
int x, y, z, i, a, b;
cin>>x>>y;
for (z=0, i=x; i<=y; i++)
 {a=i; b=y;
 while (...[1])
 if (a>b) a=a-b; else b=b-a;
 if (...[2]) z++; }
cout << z;
```

a.	b.	c.	d.
1. a!=y	1. a>y	1. a!=b	1. b==0
2. a==1	2. a==y	2. a==1	2. a!=1

25. Știind că valoarea inițială a variabilei întregi i este mai mare decât 10, stabiliți care este valoarea expresiei abs(3-i) la sfârșitul executării următoarei instrucțiuni.

(variantă Bacalaureat, 2002)

```
while (i>4) i--;
```

a. -1 **b.** 0 **c.** o valoare nedeterminată
d. 1 **e.** 2 **f.** o valoare mai mare decât 2.

26. Se consideră următoarea secvență de instrucțiuni:

```
int a, b, c=0;
cin>>a>>b;
while (a && b)
      {if (a%10>b%10) c=c*10+a%10;
          else  c=c*10+b%10;
       a/=10;  b/=10;                }
```

a. Ce valoare va avea variabila c după execuția acestei secvențe dacă valorile citite pentru variabilele a și b sunt 14624 și, respectiv, 8632 ?
b. Dacă se citește valoarea 7382 pentru variabila a, ce valoare am putea introduce pentru variabila b astfel încât valoarea calculată în variabila c să fie 2938?
c. Dați exemplu de două numere naturale de cel puțin două cifre, care ar putea fi introduse pentru variabilele a și b, astfel încât valoarea calculată în variabila c să fie 1?

27. Se consideră următoarea secvență de instrucțiuni:

```
int n, d=2;
cin>>n;
while (n>1)
      if (n%d==0) n/=d;
         else d++;
cout<<d;
```


a. Ce valoare va fi afișată dacă valoarea citită pentru variabila n este 720?
b. Dați exemplu de valori (cel puțin două) care ar putea fi introduse pentru variabila n, astfel încât valoarea afișată să fie 11.
c. Care este efectul acestei secvențe de instrucțiuni?

28. Se consideră următoarea secvență de instrucțiuni:

```
int n, b, x=0, p=1;
cin>>n>>b;                    /* 2≤b≤9 */
while (n)
      x+=p*(n %10), p*=b, n/=10;
cout<<x;
```

a. Ce valoare va fi afișată dacă se citesc valorile 1032 și 4?
b. Ce valoare trebuie să introducem pentru variabila b astfel încât pentru n=407 să se afișeze valoarea 331?
c. Care este efectul acestei secvențe de instrucțiuni?

29. Se consideră următoarea secvență de instrucțiuni:

```
int x, a, b, i;
cin>>a>>b;
i=a; x=0;
while (i<=b) { x+=i;i++; }
cout<<x;
```

a. Ce valoare va fi afișată pe ecran dacă se introduc valorile 1 și 10?
b. Ce valori ar putea fi introduse pentru a și b astfel încât programul să afișeze valoarea 22?
c. Dați exemplu de valori distincte care ar putea fi introduse astfel încât programul să afișeze valoarea 0.
d. Scrieți un program echivalent mai eficient.

30. Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi u valoarea primei cifre a numărului natural reprezentat de variabila x ?

a. `u=x/10;`
b. `u=x; while (u>=10) u=u%10;`
c. `u=x%10;`
d. `while (x>=10) x=x/10; u=x;`

31. Care dintre următoarele secvențe afișează cel mai mare divizor propriu al numărului natural neprim memorat în variabila x?

a.
```
d=2;
while (x%d==0) d++;
cout<<x/d;
```

b.
```
d=2;
do d++; while (x%d);
cout<<x/d;
```

c.
```
d=2;
while (x%d) d++;
cout<<x/d;
```

d.
```
d=x/2;
while (x%d) d--;
cout<<d;
```

32. Ce valoare inițială ar trebui să aibă variabila x astfel încât după execuția următoarei secvențe de instrucțiuni să se afișeze valoarea 640?

```
d=2;
while (d<50) { x=x*d; d=d*d;}
cout<<x;
```

33. Fie n un număr natural nenul și p un număr prim. Următoarea secvență de instrucțiuni trebuie să determine multiplicitatea lui p în n. Ce greșeli conține această secvență?

```
int n, p, m;
cin>>n>>p;
while (n%p==0) m++; n/=p;
cout<<m;
```

34. *Concurs*

La un concurs de matematică se acordă câte 5 puncte pentru fiecare problemă rezolvată corect și se scad 3 puncte pentru fiecare problemă rezolvată greșit. Un elev a obținut x puncte. Scrieți un program care să determine câte probleme a rezolvat corect și câte probleme a greșit elevul. Dacă există mai multe soluții, o veți afișa pe cea cu număr minim de probleme greșite.

35. *Numere prietene*

Două numere naturale a și b se numesc prietene dacă a este egal cu suma divizorilor lui b (exclusiv b), iar b este egal cu suma divizorilor lui a (exclusiv a). De exemplu, a=220 și b=284 sunt prietene. Scrieți un program care să determine primele trei perechi de numere prietene, cu a<b.

36. *Expresii*

Fie n un număr natural și x un număr real, citite de la tastatură. Să se calculeze valoarea expresiei[18]:

a. $E_1(n,x)=\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\ldots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}$

b. $E_2(n,x)=1-\frac{x^2}{2!}-\frac{x^4}{4!}-\ldots+(-1)^n\frac{x^{2n}}{(2n)!}$

37. *Prime*

Fie x un număr natural, x>2. Scrieți un program care să determine cel mai mare număr prim mai mic decât x și cel mai mic număr prim mai mare decât x.

18. Aceste expresii sunt de asemenea remarcabile, deoarece (așa cum veți demonstra la analiză matematică) pot fi utilizate pentru calculul aproximativ al lui sin(x), respectiv cos(x).

38. *Sumă și produs*

Se citește de la tastatură un număr natural nenul n, apoi se citesc succesiv n valori întregi. Scrieți un program care determină și afișează suma valorilor pare citite și produsul valorilor nenule.

39. *Palindrom prim*

Se citește de la tastatură un număr natural nenul n, apoi se citesc succesiv n valori naturale. Să se verifice dacă printre valorile citite există un palindrom prim.

40. *Cifre nenule*

Se citește un număr natural nenul n, apoi se citesc succesiv n valori întregi. Să se verifice dacă oricare dintre cele n valori citite are exact 3 cifre nenule.

41. *Cifra 0*

Scrieți un program care să citească de la tastatură o succesiune de valori naturale, până la citirea valorii –1, și care să determine de câte ori apare cifra 0 în scrierea numerelor citite.

42. *Cifra de control*

Fie n un număr natural, citit de la tastatură. Scrieți un program care calculează și afișează cifra de control a lui n. Cifra de control a unui număr natural se obține calculând suma cifrelor numărului, apoi suma cifrelor sumei, ș.a.m.d. până la obținerea unei singure cifre.

De exemplu, pentru n=293 calculăm suma cifrelor 2+9+3=14. Cum suma nu este formată dintr-o singură cifră, repetăm procedeul: 1+4=5. Deci 5 este cifra de control a lui 293.

43. *Cifre*

Fie N un număr natural nenul (N≤1000000000). Scriind în ordine, unul după altul, toate numerele naturale de la 1 la N, obținem o secvență de cifre. De exemplu, pentru N=22, obținem:

1234567891011121314151617181920212 2

Scrieți un program eficient care să determine numărul de cifre din secvența formată din numerele de la 1 la N.

44. *Pătrate perfecte*

Se citește de la tastatură un număr natural n. Scrieți un program eficient care să afișeze toate pătratele perfecte mai mici decât n.

45. *Numere piramidale*

Numerele piramidale se definesc ca fiind sumele parțiale ale șirului pătratelor perfecte 1, 4, 9, 16, 25... De exemplu, primele 5 numere piramidale sunt 1, 5, 14, 30, 55. Scrieți un program care să afișeze primele n numere piramidale.

46. *Fracţii*

Fie n un număr natural nenul. Scrieţi un program care să determine numărul de fracţii ireductibile care au numitorul şi numărătorul din mulţimea {1, 2, ..., n}. De exemplu, pentru n=3, se pot forma fracţiile ireductibile { $\frac{1}{1}$, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{2}{1}$, $\frac{2}{3}$, $\frac{3}{1}$, $\frac{3}{2}$ }, deci programul va afişa valoarea 7.

47. *Şir*

Să considerăm următorul şir definit prin recurenţă astfel $s_0=1$, $s_1=2$, $s_2=1$, iar $s_n=s_{n-1}+2*s_{n-2}-s_{n-3}$ (pentru $n\geq 3$). Scrieţi un program care să afişeze al n-lea termen din acest şir.

48. *Medie semestrială*

Se citeşte n, numărul de note obţinute de un elev la informatică pe parcursul semestrului, apoi se citesc cele n note. Se citeşte apoi nota obţinută de elev la teză. Scrieţi un program care să calculeze media semestrială a elevului.

49. *C.m.m.d.c.*

Se citeşte n, un număr natural nenul, apoi se citesc n valori naturale. Scrieţi un program care să calculeze cel mai mare divizor comun al celor n numere citite.

50. *Număr*

Se citeşte de la tastatură o succesiune de caractere, terminată cu spaţiu (' '). Scrieţi un program care să verifice dacă succesiunea de caractere citită poate fi considerată un număr natural scris în baza 10. Modificaţi programul astfel încât să testeze dacă succesiunea de caractere citită poate fi considerată un număr real.

51. *Cuvinte*

Se citeşte de la tastatură o propoziţie constituită din cuvinte separate prin unul sau mai multe spaţii. Sfârşitul propoziţiei este marcat de întâlnirea caracterului '.'. Scrieţi un program care determină şi afişează numărul de cuvinte din propoziţie.

52. *Progresie aritmetică*

Se citesc două numere naturale x şi r, care reprezintă primul termen, respectiv raţia unei progresii aritmetice, apoi se citeşte un număr natural n. Scrieţi un program care să afişeze primii n termeni ai acestei progresii aritmetice.

Fie n un număr natural nenul. Scrieţi un program care să determine cel mai mic număr natural care se împarte exact la toate numerele naturale mai mici sau egale cu n.

53. *Număr maxim de divizori primi*

Se citeşte de la tastatură un număr natural n, apoi se citesc succesiv n numere naturale. Să se determine un număr dintre cele n citite, care are un număr maxim de divizori primi.

54. *Palindroame prime*

Să se scrie un program care să citească un număr natural n ($n\leq 1000000000$) şi care să determine eficient toate palindroamele prime mai mici decât n.

55. *Sumă şi produs*

Fie n un număr natural. Să se determine toate numerele naturale mai mici decât n cu proprietatea că sunt divizibile atât prin suma, cât şi prin produsul cifrelor lor.

56. *Zerouri*

Se citesc de la tastatură n ($n\leq 1000000$) numere naturale (<30000). Să se determine numărul de zerouri în care se termină produsul numerelor citite.

57. *Problema bibliotecarului*

Bibliotecarul de la şcoala ta a descoperit o carte în care paginile ce ar fi trebuit să fie numerotate cu un număr prim aveau numărul de pagină mâzgălit. Ştiind că numerotarea paginilor cărţii începe de la 3, scrieţi un program care citeşte de la tastatură pe n numărul de pagini din carte şi afişează pe ecran numărul de pagini nemâzgălite, precum şi numărul de cifre pe care trebuie să le scrie bibliotecarul, pentru a renumerota paginile mâzgălite.

58. *Numere bine ordonate*

Se numesc numere „bine ordonate" acele numere care au cifrele în ordine strict crescătoare sau strict descrescătoare. Să se scrie un program care determină toate numerele „bine ordonate crescător" cu trei cifre. Modificaţi programul pentru a obţine şi toate numerele „bine ordonate descrescător" cu trei cifre. Câte soluţii există?

59. *Relaţie*

Fie n un număr natural. Să se afle numerele naturale de trei cifre xyz care au proprietatea că $n/(x^2+y^2+z^2)$ este număr natural.

60. *Cifre romane*

Se citeşte de la tastatură un număr natural $n<5000$. Scrieţi un program care afişează numărul n în scriere cu cifre romane.

Indicaţie: I=1; V=5; X=10; L=50; C=100; D=500; M=1000; IV=4; IX=9; XL=40; XC=90; CD=400; CM=900.

61. *Plata sumei*

Se citesc de la tastatură două numere naturale S şi x ($0<S\leq 10000$, $0<x\leq 100$). S reprezintă o sumă pe care trebuie să o plătim, utilizând un număr minim de bancnote. Bancnotele au ca valori numai puteri ale lui x ($1, x, x^2, x^3, \ldots$) şi presupunem că dispunem de un număr suficient de mare de bancnote. Scrieţi un program care afişează pe ecran o modalitate de a plăti suma, utilizând un număr minim de bancnote. De exemplu, pentru S=107 şi x=5, veţi afişa:

```
4 bancnote cu valoarea 25
1 bancnote cu valoarea 5
2 bancnote cu valoarea 1
```

62. *Euro-dolar*

Ion are 100 de euro şi un prieten care l-a ajutat să obţină ratele de schimb euro-dolar pentru următoarele n zile (n≤100). Mai exact, pentru fiecare dintre cele n zile, prietenul îi comunică două numere D şi E (ceea ce înseamnă că, în ziua respectivă, cu 100 de euro se pot cumpăra D dolari, iar cu E dolari se pot cumpăra 100 euro). Scrieţi un program care să determine suma maximă (exprimată în euro) pe care Ion o poate acumula după n zile, făcând tranzacţii euro-dolar.

De exemplu, să considerăm că n=8 şi ratele de schimb pentru cele 8 zile sunt:

```
218 219
228 231
227 235
205 213
230 232
239 239
251 258
205 213
```

Pentru a acumula o sumă maximă, Ion va realiza următoarele tranzacţii:

```
Ziua 2 ... schimba 100.0000 Euro in 228.0000 USD
Ziua 4 ... schimba 228.0000 USD in 107.0422 Euro
Ziua 7 ... schimba 107.0422 Euro in 268.6760 USD
Ziua 8 ... schimba 268.6760 USD in 126.1389 Euro
```

63. *Şef*

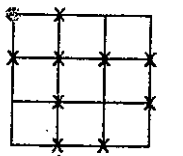
O firmă face angajări. Fiecărei persoane angajate îi corespunde un număr natural nenul a cărui reprezentare binară ocupă 16 biţi. Acest număr reprezintă codul persoanei în ierarhia firmei. Astfel, fiind date două numere a şi b, spunem că a este „şeful" lui b dacă pentru orice poziţie binară 0 a lui a, poziţia corespunzătoare din b este 0. Evident, orice persoană este propriul ei şef. Se citeşte n, un număr natural care reprezintă codul unei persoane. Se cere să se afişeze numărul şefilor săi. De exemplu, pentru n=255 veţi afişa 256.

(Olimpiada Judeţeană de Informatică, Timiş, 2000)

64. *Ceas cu cuc*

Bunica mea are un ceas cu cuc care cântă la ore fixe (de un număr de ori egal cu ora) şi la „şi jumătate" (o singură dată). Eu am plecat de acasă exact după ce cucul a cântat de oră fixă. Când m-am întors, cucul tocmai cântase de „şi jumătate" şi bunica supărată mi-a spus: „De când ai plecat, cucul a cântat de x ori!". Fiindcă nu înţeleg de ce bunica mea e aşa de supărată, scrieţi un program care să determine cât timp am lipsit de acasă!

65. *Problema pădurarului neamţ*



Să ne imaginăm o pădure nemţească ca pe o reţea ortogonală[19] de n x n puncte (n≤100), astfel încât distanţa dintre oricare două puncte vecine pe orizontală sau verticală este egală cu 1. În toate punctele se află pomi, cu excepţia unui singur punct de coordonate x0, y0 în care se află pădurarul. Presupunând că pomii sunt punctiformi, pădurarul vede doar o parte din pomi (pe orice linie dreaptă care trece prin poziţia sa pădurarul vede cel mult 2 pomi). Scrieţi un program care citeşte de la tastatură n, x0 şi y0 şi afişează numărul de pomi pe care îi vede pădurarul. De exemplu, pentru n=4, x0=0, y0=0, pădurarul vede 9 pomi.

66. *Numai cu 0 şi 1*

Fie n un număr natural (n≤30000) şi b o bază (2≤b≤10). Să se scrie un program care afişează în baza 10 toate numerele mai mici sau egale cu n, care scrise în baza b folosesc numai cifrele 0 şi 1. De exemplu, pentru n=64 şi b=4, programul va afişa 0, 1, 4, 5, 16, 17, 20, 21, 64.

19. Vă puteţi imagina o reţea ortogonală ca pe o foaie de matematică. Punctele sunt la intersecţiile liniilor orizontale cu cele verticale.

3. Fişiere

3.1. *Noţiuni introductive*

Un *fişier* este o colecţie de date de acelaşi tip, memorate pe suport extern (hard-disc, dischetă, CD etc).

Avantajele utilizării fişierelor sunt o consecinţă a proprietăţilor memoriilor externe. Fişierele permit memorarea unui volum mare de date persistente (care nu se „pierd" la încheierea execuţiei programului sau la închiderea calculatorului).

Până acum am lucrat numai cu date citite de la tastatură, memorate în variabile simple. Pe parcursul execuţiei unui program, se alocă memorie în RAM pentru variabilele utilizate în programul respectiv. Memoria alocată se eliberează fie pe parcursul execuţiei programului, fie la terminarea execuţiei acestuia. În orice caz, după terminarea programului, valorile variabilelor „s-au pierdut", deoarece zona de memorie în care au fost memorate s-a eliberat şi, dacă doream să executăm încă o dată programul cu aceleaşi date de intrare, trebuia să le introducem din nou de la tastatură. Chiar şi pentru programe cu un volum mic de date de intrare, aşa cum am elaborat noi până acum, acest aspect reprezintă un inconvenient. Dar imaginaţi-vă ce s-ar întâmpla la o firmă care, de exemplu, la fiecare rulare a programului de salarii ar trebui să introducă toate datele despre angajaţii firmei sau la o bibliotecă la care, de exemplu, orice cerere de împrumut s-ar solda cu introducerea tuturor cărţilor existente în bibliotecă! Exemplele pot continua, dar ideea principală rămâne că în orice situaţie practică fişierele sunt indispensabile.

Fişierele pot fi clasificate după conţinutul lor astfel:
- fişiere *text*: conţin o secvenţă de caractere ASCII structurate pe linii;
- fişiere *binare*: conţin o secvenţă de octeţi, fără o structură predefinită.

Noi vom studia modul de utilizare a fişierelor text.

3.2. *Fişiere text în limbajul C++*

Declararea fişierelor

Operaţiile de intrare/ieşire în limbajul C++ au utilizat până acum *stream*-urile `cin` şi `cout`, care erau automat asociate tastaturii, respectiv ecranului. Pentru ca un program să poată citi informaţii dintr-un fişier, respectiv să scrie informaţii într-un fişier, trebuie să asociem fişierul respectiv unui *stream* de intrare/ieşire.

În fişierul antet `fstream.h` sunt declarate clasele `ifstream`, `ofstream` şi `fstream`[20]. Pentru a utiliza într-un program operaţii de intrare/ieşire folosind fişiere trebuie să declarăm variabile de tipurile `ifstream`, `ofstream` sau `fstream`. Mai exact:

`ifstream` – declarare *stream* de intrare (numai pentru operaţii de citire);
`ofstream` – declarare *stream* de ieşire (numai pentru operaţii de scriere);
`fstream` – declarare *stream* de intrare/ieşire (în care se pot realiza atât operaţii de citire, cât şi operaţii de scriere, în funcţie de modul specificat la deschidere).

Exemple

```
ifstream f1; //am declarat un stream de intrare denumit f1
ofstream f2; //am declarat un stream de iesire denumit f2
fstream  f3; //am declarat un stream denumit f3.
```

Observaţii

1. Declaraţiile precedente respectă sintaxa declaraţiilor de variabile (`ifstream`/`ofstream`/`fstream` reprezintă tipul, iar `f1`, `f2`, `f3` reprezintă numele variabilelor). Tipul fiind o clasă, variabilele `f1`, `f2`, `f3` sunt obiecte.
2. Nu intrăm în amănunte referitoare la programarea orientată pe obiect, dar trebuie să precizăm că o clasă este asemănătoare tipului `struct`, dar clasa conţine atât date, cât şi funcţiile necesare prelucrării acestora. Datele şi funcţiile incluse într-o clasă se numesc membri. Variabilele de tip clasă se numesc obiecte. Pentru a ne referi la un membru al unui obiect utilizăm operatorul punct, denumit operator de selecţie directă: `nume_obiect.nume_membru`.

Deschiderea fişierelor

Pentru a utiliza *stream*-ul declarat trebuie să îl deschidem. La deschidere, se asociază *stream*-ului un fişier fizic (existent pe suport extern) şi, eventual, se precizează modul de deschidere, care determină operaţiile permise cu fişierul respectiv. Deschiderea se poate realiza în două moduri:

20. La nivel de iniţiere, vom trata aceste clase ca un tip al limbajului C++.

- după declarare, prin apelul funcției membră `open`, precizând ca parametri un șir de caractere, care reprezintă specificatorul fișierului fizic, conform sintaxei sistemului de operare utilizat, și (eventual) modul de deschidere;
- la declarare, specificând după numele *stream*-ului care se declară numai parametrii corespunzători funcției `open`, încadrați între paranteze rotunde.

La deschidere, este obligatoriu să specificăm modul de deschidere dacă fișierul este declarat de tip `fstream`. Pentru tipul `ifstream` se utilizează implicit modul `ios::in`, iar pentru tipul `ofstream` se utilizează implicit modul `ios::out`.

Exemplul 1

```
ifstream f1;
f1.open("date.in");
```

Am declarat un *stream* de intrare denumit `f1`, apoi l-am deschis cu ajutorul funcției `open`, asociindu-l fișierului `date.in`. Deoarece nu am precizat discul logic pe care se află acest fișier sau calea către directorul în care se află fișierul, se deduce că fișierul se află pe hard-disc în directorul curent. Același efect l-am fi obținut prin deschiderea fișierului la declarare, astfel:

```
ifstream f1("date.in");
```

Exemplul 2

```
ofstream f2;
f2.open("c:\\ema\\exemple\\date.out");
```

Am declarat un *stream* de ieșire, denumit `f2`, pe care l-am asociat fișierului fizic `date.out`, care se află pe discul `c:`, în directorul `exemple`, care este un subdirector al directorului `ema`[21]. În acest exemplu nu am specificat nici un mod de deschidere. În acest caz se utilizează modul de deschidere implicit (se creează un fișier vid cu numele specificat, în care se vor realiza doar operații de scriere).

Exemplul 3

```
fstream f3;
f3.open("nr.in", ios::in);
```

Am declarat un *stream* denumit `f3`, pe care la deschidere l-am asociat fișierului `nr.in`. Deoarece tipul `fstream` nu are asociat un mod de deschidere implicit, a fost necesar să precizăm modul de deschidere. În acest caz, modul de deschidere este `ios::in`, ceea ce înseamnă că fișierul a fost deschis ca fișier de intrare (numai pentru operații de citire).

Exemplul 4

```
fstream f4;
f4.open("nr.out", ios::out);
```

21. Observați că în șirul de caractere care reprezintă specificatorul de fișier a fost necesar să dublăm caracterul special *backslash*!

Am declarat un *stream* denumit `f4`, pe care la deschidere l-am asociat fișierului `nr.out`. Modul de deschidere specificat este `ios::out`, ceea ce semnifică deschiderea fișierului ca fișier de ieșire (se creează un fișier vid cu numele specificat, în care se vor realiza doar operații de scriere).

Exemplul 5

```
fstream f5;
f4.open("nr.ad", ios::app);
```

Am declarat un *stream* denumit `f5`, pe care la deschidere l-am asociat fișierului `nr.ad`. Modul de deschidere specificat este `ios::app`, ceea ce înseamnă că deschidem fișierul ca fișier de ieșire, pentru adăugare de informații la sfârșit[22].

Observație

Dacă după operația de deschidere a fișierului variabila corespunzătoare are valoarea `NULL`, deducem că operația de deschidere a eșuat (de exemplu, dacă dorim să deschidem un fișier de intrare care nu există). Este indicat să testăm succesul operației de deschidere înainte de a efectua orice altă operație cu fișierul.

Citirea datelor dintr-un fișier

După deschiderea fișierului ca fișier de intrare se pot realiza operații de citire. În acest scop se poate utiliza operatorul de citire >>, sau pot fi utilizate funcții-membru specifice.

Una dintre particularitățile citirii cu operatorul >> este faptul că ignoră caracterele albe. Prin urmare, dacă intenționăm să citim toate caracterele din fișier (inclusiv spațiu, tab, enter), acest mod de citire este inadecvat.

În acest scop putem utiliza funcția-membru `get()`. Funcția `get()` are mai multe forme de apel. De exemplu, pentru citirea caracterelor (`char` sau `unsigned char`) se poate utiliza astfel:

```
ifstream f1("text.in");
char c;
f1.get(c);
```

Am citit primul caracter din fișier, indiferent dacă este sau nu caracter alb. Evident, în mod similar, funcția `get()` se poate utiliza pentru citirea unui caracter de la tastatură: `cin.get(c)`.

Scrierea datelor într-un fișier

După deschiderea fișierului ca fișier de ieșire, se pot realiza operații de scriere utilizând operatorul de scriere <<.

22. app – append (adăugare).

Exemplu

```
ofstream fout("date.out");
fout<<"Salut";
```

Operații de test

Detectarea sfârșitului de fișier

În multe aplicații este necesar să citim toate datele existente într-un fișier, fără să avem informații prealabile despre numărul acestora.

La deschiderea unui fișier se creează un *pointer de fișier*, care indică poziția curentă în *stream*. Orice operație de citire determină deplasarea pointerului de citire în *stream*-ul de intrare, respectiv orice operație de scriere determină deplasarea pointerului de scriere în *stream*-ul de ieșire.

Pentru a testa dacă pointerul de fișier a ajuns la sfârșitul fișierului putem utiliza funcția membră[23] `eof()`. Funcția `eof()` returnează valoarea `0` dacă nu am ajuns la sfârșitul fișierului, respectiv o valoare diferită de `0` dacă am ajuns la sfârșit.

Exemplu

Vom citi și vom număra toate caracterele existente într-un fișier, până la sfârșitul acestuia.

```
ifstream fin("text.in");
char c;
long int nr=0;
while (!fin.eof()) {fin.get(c); nr++;}
cout<<"Fisierul contine "<<nr<<" caractere\n";
```

Funcția membru `get()` returnează pointerul de fișier (poziția curentă în *stream*). Prin urmare, secvența de program ar putea fi rescrisă și astfel:

```
ifstream fin("text.in");
char c;
long int nr=0;
while (fin.get(c)) nr++;
cout<<"Fisierul contine "<<nr<<" caractere\n";
```

Testarea reușitei unei operații de intrare/ieșire

În aplicațiile didactice, de cele mai multe ori presupunem că datele de intrare sunt corecte. În condiții practice însă trebuie să verificăm reușita oricărei operații de intrare/ieșire și, mai mult, să verificăm și corectitudinea datelor citite.

Pentru a testa reușita unei operații de intrare/ieșire se poate utiliza, de exemplu, funcția membră `good()`. Funcția `good()` returnează o valoare diferită de `0` dacă operația precedentă de intrare/ieșire s-a efectuat cu succes și `0` în caz contrar.

23. Denumirea *eof* provine de la *end of file* – sfârșit de fișier.

Există și alte funcții de test asemănătoare: `bad()` (care returnează o valoare diferită de `0` dacă s-a efectuat o operație incorectă) sau `fail()` (care returnează o valoare diferită de `0` dacă a apărut o eroare în operațiile de intrare/ieșire până în acel moment).

Anticiparea următorului caracter la intrare

În anumite situații este necesară citirea unor date până la întâlnirea unui anumit caracter, cu excepția acestuia. Pentru a afla anticipat ce caracter urmează în *stream*-ul de intrare (fără a-l extrage din *stream*) putem utiliza funcția membră `peek()`, care returnează următorul caracter din *stream*.

Închiderea unui fișier

După realizarea tuturor operațiilor cu un fișier, acesta trebuie închis. Închiderea unui fișier se realizează prin apelarea funcției membre `close()`.

Exemplu

```
f1.close();
```

Operația de închidere este obligatorie, în special pentru fișierele de ieșire. *În cazul în care nu închideți fișierul de ieșire, există mari șanse să pierdeți informații!*

3.3. *Fișiere text în limbajul C*

Declararea fișierelor

Operațiile de intrare/ieșire în limbajul C au fost realizate până acum utilizând instrucțiunile `scanf()` și `printf()`, care implicit preiau informații de la tastatură și, respectiv, le transmit spre ecran. Pentru ca un program să poată prelucra informații dintr-un fișier, acesta trebuie declarat, indicându-se numele lui precum și operațiile permise asupra conținutului fișierului respectiv.

În fișierul antet `stdio.h` sunt declarate, printre altele, și funcțiile, constantele, tipurile de date și variabilele globale cu ajutorul cărora pot fi prelucrate informațiile conținute în fișiere. Pentru a utiliza într-un program operații de intrare/ieșire folosind fișiere trebuie să declarăm una sau mai multe variabile de tip `FILE *`. Declarația unui fișier se face astfel:

```
FILE * fisier;
```

Observație

Declarația precedentă respectă sintaxa declarațiilor de variabile (`FILE *` reprezintă tipul, iar `fisier` reprezintă numele variabilei).

Deschiderea fişierelor

Desigur, pentru a avea acces la informaţiile din fişier trebuie să fie cunoscut fişierul fizic de pe suport (specificatorul de fişier conform sintaxei sistemului de operare utilizat), ca şi modul în care vor fi prelucrate informaţiile din fişier. Acest lucru se realizează utilizând funcţia `fopen`.

```
fisier = fopen(const char *nume, const char *mod)
```

unde `nume` este o constantă de tip şir de caractere care reprezintă specificatorul fişierului fizic pe suport, iar `mod` este o constantă şir de caractere care indică modul în care vor fi accesate informaţiile din fişier.

Exemple

```
FILE *f1, *f2, *f3;   //am declarat trei fisiere f1, f2, f3
f1 = fopen("date.in", "r");
   //am deschis fisierul de intrare (r - read) f1
f2 = fopen("date.out", "w");
   //am deschis fisierul de iesire (w - write) f2
f3 = fopen("date.txt", "a");
   //am deschis fisierul f3 pentru adaugare de
   //informatii (a - append)
```

Declararea şi deschiderea unui fişier se pot face şi simultan:

Exemplu 1

```
FILE *f1 = fopen("date.in", "r");
```

Am declarat fişierul de intrare denumit `f1`, apoi l-am deschis cu ajutorul funcţiei `fopen`, asociindu-l fişierului `date.in`. Deoarece în toate exemplele anterioare nu am precizat discul logic pe care se află fişierele sau calea către directorul în care se află acestea, se deduce că fişierele se află pe hard-disc în directorul curent.

Exemplul 2

```
FILE *f2;
f2 = fopen("c:\\ema\\exemple\\date.out", "w");
```

Am declarat un fişierul de ieşire, denumit `f2`, pe care l-am asociat fişierului fizic `date.out`, care se află pe discul `c:`, în directorul `exemple`, care este un subdirector al directorului `ema`.

Observaţie

Dacă, după operaţia de deschidere a fişierului, variabila corespunzătoare are valoarea `NULL`, deducem că operaţia de deschidere a eşuat (de exemplu, dacă dorim să deschidem un fişier de intrare care nu există). Este indicat să testăm succesul operaţiei de deschidere înainte de a efectua orice altă operaţie cu fişierul.

Exemplul 3

```
#include <stdio.h>
int main()
{ FILE *numef;
  if (!(numef = fopen("nr.in", "r")))
     printf("EROARE la deschidere\n");
     else printf("Deschidere corecta a fisierului\n");
  return 0; }
```

Testarea erorii la deschiderea fişierului se poate face şi astfel:

```
if ((numef = fopen("nr.in", "r")) == NULL)
```

Citirea datelor dintr-un fişier

După deschiderea fişierului ca fişier de intrare se pot realiza operaţii de citire. În acest scop se utilizează funcţia `fscanf`.

```
int fscanf(fisier, format, adr_var1, ..., adr_varn);
```

Observaţi că prototipul funcţiei `fscanf()` este asemănător cu prototipul funcţiei `scanf()`. Funcţia `fscanf()` are în plus, ca prim parametru, fişierul din care se face citirea. În rest, semnificaţia parametrilor, rezultatului şi efectul funcţiei sunt aceleaşi ca în cazul funcţiei `scanf()`.

Scrierea datelor într-un fişier

După deschiderea fişierului ca fişier de ieşire, se pot realiza operaţii de scriere utilizând funcţia `fprintf()`.

```
int fprintf(fisier, format, expresie1, ..., expresien);
```

Observaţi că prototipul funcţiei `fprintf()` este asemănător cu prototipul funcţiei `printf()`. Funcţia `fprintf()` are în plus, ca prim parametru, fişierul în care se face afişarea. În rest, semnificaţia parametrilor, rezultatului şi efectul funcţiei sunt aceleaşi ca în cazul funcţiei `printf()`.

Închiderea fişierelor

După terminarea tuturor operaţiilor care implică lucrul cu fişiere, acestea trebuie închise. Funcţia care realizează acest lucru este:

```
fclose(fisier);
```

Exemplul 1

Să scriem un program care citeşte din fişierul de intrare `nr.in` un număr natural `n` (`0<n≤100`) şi apoi `n` valori naturale şi afişează în fişierul de ieşire `nr.out` numerele citite descompuse în factori primi, pe fiecare linie fiind scris câte un număr urmat de perechi de forma `(d,p)`, indicând divizorul şi puterea lui.

```
#include <stdio.h>

int main()
{ int n, i, d, p, er;
  unsigned long x;
  FILE *fin = fopen("nr.in", "r");
  if (!fin)
    {printf("Eroare la deschiderea fisierului nr.in\n");
     return 1;}
  FILE *fout = fopen("nr.out", "w");
  er = fscanf(fin, "%d", &n); //citesc n din fisierul fin
  if (!er || n>100 || n<0)
    {printf("Numarul de valori este incorect\n");
     return 2;}
  for (i=0; i<n; i++)     //citesc cele n valori
    { er = fscanf(fin, "%lu", &x);
      if (!er)
        { printf("Eroare la citirea valorilor\n");
          return 3;}
      fprintf(fout, "%10lu ", x);
      d=2;
      while (x>1)
        { p=0;
          while (x%d==0)
            { p++; x/=d; }
          if (p) fprintf(fout, "(%d,%d)", d, p);
          d++;}
      fprintf(fout, "\n");
    }
  fclose(fout);
  return 0;
}
```

Exemplul 2

Scrieți un program care creează o copie a fișierului `intrare.txt` cu numele `iesire.txt`.

```
#include <stdio.h>

int main()
{
FILE *fisin, *fisout;
char x;
if (!(fisin = fopen("intrare.txt", "r")))
   {printf("EROARE la deschiderea fisierului de intrare");
     return 1;}
if (!(fisout = fopen("iesire.txt", "w")))
   {printf("EROARE la deschiderea fisierului de iesire");
    return 2;}
```



```
while (!feof(fisin))
      if (fscanf(fisin, "%c", &x) != EOF)
         fprintf(fisout, "%c", x);
 fclose(fisin);
 fclose(fisout);}
 return 0; }
```

3.4. *Probleme propuse*

1. Ce va conține fișierul `test.out` pentru `n=31` și `a=3`? Care credeți că este efectul programului?

```
#include <fstream.h>
#include <iomanip.h>
int main()
{
ofstream fout("test.out");
int a, n, i, j;
cout<<"n (28<=n<=31), a (1<=a<=7) = "; cin >> n >> a;
for (i=1; i<= 7; i++)
     {j=i-a+1;
      if (j>0)  fout<<setw(4)<<j;  else fout << "    ";
      for (j += 7; j <= n; j+=7) fout<<setw(4)<<j;
      fout << endl;}
fout.close();
return 0;
}
```

2. Fișierul `nr.in` conține numere întregi, scrise pe mai multe linii, numerele de pe aceeași linie fiind separate prin spații. Scrieți un program care să numere câte valori întregi sunt pe fiecare dintre liniile fișierului.
3. Scrieți un program care să testeze dacă două fișiere ale căror nume se citesc de la tastatură sunt identice.
4. În fișierul `NR.TXT` se află numere naturale din intervalul `[0,5000]`, separate prin spații. Să se creeze fișierul `PARE.TXT` care să conțină doar valorile pare din fișierul `NR.TXT`, câte o valoare pe linie.

   (variantă Bacalaureat, 2000)

5. Scrieți un program care să numere câte linii conține fișierul text `text.in`.

4. Tipuri structurate de date

Pentru început, am scris programe care nu prelucrează decât date simple (de exemplu, numere întregi, numere reale, caractere). În realitate, un program trebuie să prelucreze volume mari de date şi pentru ca prelucrarea să se realizeze eficient este *necesară* organizarea acestor date în structuri. De exemplu, să presupunem că dorim să ordonăm elevii din şcoală în ordine alfabetică. Pentru aceasta este nevoie să reţinem „undeva" (într-o structură de date) numele şi prenumele elevilor şcolii şi abia apoi îi putem ordona.

O structură de date reprezintă un ansamblu (o colecţie) de date organizate după anumite reguli, reguli care depind de tipul de structură.

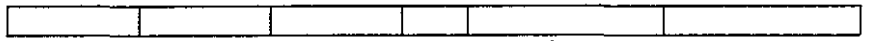
4.1. *Tablouri*

Un tablou este o colecţie de date *de acelaşi tip*, memorate într-o zonă de *memorie contiguă*, reunite sub un nume comun (numele tabloului).

Declararea unei variabile de tip tablou:

```
tip nume[NrE];
```

Am declarat un tablou format din `NrE` elemente de tipul `tip`. `NrE` indică numărul de elemente din tablou şi trebuie să fie obligatoriu o expresie constantă.



Deoarece elementele unui tablou sunt memorate în ordine, unul după altul, într-o zonă contiguă, pentru a ne referi la un element al unui tablou putem specifica numele tabloului din care face parte elementul şi poziţia sa în tablou, prin numărul său de ordine (numerotarea începe de la `0`).

```
nume[indice]
```

Am specificat elementul `indice` al tabloului `nume` (`indice` reprezintă numărul de ordine al elementului în tablou, cuprins între `0` şi `NrE-1`). Parantezele pătrate (`[]`) constituie *operatorul de indexare*. Operatorul de indexare are prioritate maximă (mai mare decât a operatorilor unari).

Exemple

1. Să declarăm un tablou cu `10` elemente de tip `int`:

```
int a[10];
```

Elementele tabloului sunt `a[0], a[1], a[2], ..., a[9]`.

2. Să declarăm un tablou cu `100` de elemente de tip `float`:

```
float b[100];
```

Observaţii

1. La întâlnirea unei declaraţii de variabilă tablou, compilatorul verifică dacă dimensiunea zonei de memorie necesară pentru memorarea tabloului nu depăşeşte memoria disponibilă. Dimensiunea zonei de memorie necesară unui tablou se calculează înmulţind numărul de elemente cu numărul de octeţi necesari pentru memorarea unui element, adică `NrE*sizeof(tip)`.
2. Ca în cazul oricărei declaraţii de variabile, putem iniţializa elementele unei variabile tablou, chiar de la declarare.

```
tip nume[NrE]={val0, val1, ..., valk};
```

Ca urmare, se vor atribui în ordine elementelor tabloului valorile din lista de iniţializare (`k≤NrE`). Dacă tabloul este integral iniţializat la declarare, nu este necesar să mai specificăm dimensiunea sa, fiind considerată egală cu numărul de valori din lista de iniţializare. De exemplu:

```
int a[]={12,20,30,5};
```

Am declarat un tablou cu `4` elemente de tip `int` iniţializate astfel:

12	20	30	5
a[0]	a[1]	a[2]	a[3]

3. Un astfel de tablou, pentru care la declarare este specificată o singură dimensiune, iar poziţia unui element este specificată utilizând un singur indice, se numeşte *tablou unidimensional* sau *vector*.
4. Elementele unui tablou pot fi de orice tip al limbajului. Prin urmare... elementele unui tablou pot fi de tip tablou! Declarare:

```
tip nume[Nr1][Nr2];
```

Am declarat un tablou cu `Nr2` elemente, fiecare element fiind un tablou cu `Nr1` elemente de tipul specificat. Un astfel de tablou, pentru care la declarare trebuie să specificăm două dimensiuni, iar poziţia unui element este specificată utilizând doi indici, se numeşte *tablou bidimensional* sau *matrice*.

Putem să ne imaginăm un tablou bidimensional ca pe o tablă de şah. Poziţia unui element pe tablă este identificată prin doi indici: linia şi coloana. Prin analogie cu tabla de şah, şi la informatică primul indice utilizat în referirea unui element este denumit „indice de linie"; iar cel de al doilea indice este denumit „indice de coloană".

De exemplu, să declarăm o matrice cu două linii şi trei coloane cu elemente întregi, pe care o vom iniţializa la declarare:

```
int a[2][3]={{1,2,3},{4,5,6}};
```

a	coloana 0	coloana 1	coloana 2
linia 0	1	2	3
linia 1	4	5	6

Pentru a ne referi la un element al unei matrice, specificăm numele matricei, indicele de linie şi indicele de coloană astfel:

```
nume[indice_linie][indice_coloana]
```

De exemplu, pentru a ne referi la elementul de pe linia 1, coloana 2 din matricea a, vom scrie `a[1][2]`.

5. Elementele unui tablou bidimensional pot fi de orice tip, inclusiv... tablou! Se obţin astfel tablouri multidimensionale. Să declarăm, de exemplu, un tablou tridimensional (vă imaginaţi un paralelipiped cu dimensiunile 10, 5, 40):

```
int a[10][5][40];
```

4.2. *Prelucrări elementare pe vectori*

Citirea unui vector

Citirea unui vector se realizează citind elementele vectorului, unul câte unul. Să considerăm ca exemplu un vector a cu n elemente (n≤100) de tip int.

```
int a[100], n, i;          //declarare vector
cout << "n= "; cin >>n;    //citesc numarul de componente
for (i=0; i<n; i++)        //citesc succesiv componentele
     {cout <<"a["<<i<<"]="; cin>>a[i];}
```

Afişarea unui vector

Afişarea unui vector se realizează afişând pe rând componentele vectorului.

```
for (i=0; i<n; i++)        //afisez succesiv componentele
     cout << a[i] <<' ';
```

Copierea unui vector

Să presupunem că dorim să copiem vectorul a într-un alt vector b. Copierea unui vector nu se poate face printr-o atribuire de forma b=a (veţi obţine un mesaj de eroare). Copierea unui vector se realizează element cu element:

```
for (i=0; i<n; i++) b[i]=a[i];
```

Determinarea elementului maxim/minim dintr-un vector

Pentru a determina cel mai mare element din vector, vom considera o variabilă (să o numim max) în care vom reţine la fiecare pas maximul dintre elementele analizate. Iniţializăm variabila max cu un element din vector (de exemplu, cu

a[0]). Parcurgem apoi vectorul, comparând fiecare element din vector cu max şi actualizând eventual maximul după comparare.

```
max=a[0];           //initializez maximul cu primul element
for (i=1; i<n; i++) //compar succesiv componentele cu max
     if (max<a[i]) max=a[i];          //actualizez max
```

Exerciţii

1. Modificaţi secvenţa de instrucţiuni precedentă, astfel încât să determine cel mai mic element din vector.
2. Modificaţi secvenţa de instrucţiuni precedentă, astfel încât să determine maximul, după iniţializarea lui max cu ultimul element din vector.

Media aritmetică a elementelor strict pozitive

Se consideră un tablou de 10 numere întregi. Scrieţi un program care citeşte de la tastatură cele 10 componente ale vectorului şi afişează pe ecran media aritmetică a valorilor strict pozitive din vector, cu două zecimale.

(variantă Bacalaureat, 2000)

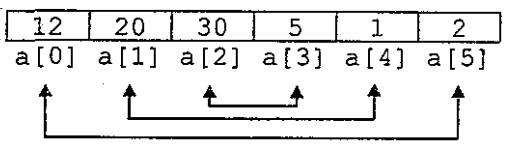
Soluţie

După citire, se parcurge vectorul, verificând pentru fiecare element dacă este sau nu strict pozitiv. Când găsim un element strict pozitiv, îl numărăm şi îl adunăm la suma elementelor strict pozitive:

```
#include <iomanip.h>
void main()
{ float a[10], s=0;
  int i, nr=0;
  for (i=0; i<10;i++) cin >> a[i];
  for (i=0; i<10; i++)
      if (a[i]>0) s+=a[i], nr++;
  if (nr) cout << setprecision(2) << s/nr;
      else cout << "Nu exista elemente strict pozitive";}
```

Inversarea ordinii elementelor din vector

Pentru a inversa ordinea elementelor dintr-un vector trebuie ca primul element să fie interschimbat cu ultimul, al doilea element cu penultimul ş.a.m.d. Mai exact, trebuie să parcurgem vectorul până la mijloc şi să interschimbăm elementele simetrice faţă de mijloc. De exemplu:



Problema care rămâne este determinarea simetricului.

Poziția elementului	Poziția simetricului	Observație
0	n-1	0+n-1=n-1
1	n-2	1+n-2=n-1
2	n-3	2+n-3=n-1

...

Observăm că atunci când poziția elementului crește cu 1, poziția simetricului scade cu 1, deci suma dintre poziția elementului și poziția simetricului este constantă (n-1). Deducem că simetricul elementului a[i] este a[n-1-i].

```
for (i=0; i<n/2; i++)
    {aux=a[i]; a[i]=a[n-1-i]; a[n-i-1]=aux;}
```

Verificarea unei proprietăți

Frecvent apar probleme în care se cere să se verifice dacă toate elementele unui vector au o anumită proprietate *P* sau dacă există în vector un element care are proprietatea *P*. Pentru a descrie o secvență de instrucțiuni cât mai generală, considerăm că P(x) are valoarea 1, dacă x are proprietatea respectivă, și 0, altfel.

a. Pentru a verifica dacă toate elementele unui vector au proprietatea *P*, vom verifica succesiv fiecare element din vector. Pentru a reține rezultatul verificării, vom considera o variabilă întreagă ok, care va avea valoarea 1, dacă toate elementele vectorului au proprietatea *P*, și 0, altfel. Inițial variabilei ok îi atribuim valoarea 1 (ipoteza optimistă, „prezumția de nevinovăție" – din moment ce nu am găsit încă nici un element care să conteste acest lucru, putem presupune că toate elementele vectorului au proprietatea *P*). Dacă, parcurgând vectorul, vom găsi un element care nu are proprietatea respectivă, variabilei ok îi vom atribui valoarea 0:

```
for (ok=1, i=0; i<n && ok; i++)
//parcurg vectorul pana la sfarsit sau pana la intalnirea
//unui element care nu are proprietatea P
     if ( !P(a[i]) ) ok=0; //a[i] nu are proprietatea P
```

b. Pentru a verifica dacă există un element în vector cu proprietatea *P*, vom parcurge vectorul (până la sfârșit sau până la întâlnirea unui element cu proprietatea *P*). Rezultatul verificării îl vom reține în variabila întreagă gasit. Inițial (deoarece nu am găsit încă nici un element cu proprietatea *P*) îi atribuim variabilei gasit valoarea 0. Dacă vom găsi în vector un element cu proprietatea *P*, îi vom atribui variabilei gasit valoarea 1:

```
for (gasit=i=0; i<n && !gasit; i++)
     if ( P(a[i]) ) gasit=1; //a[i] are proprietatea P
```


Căutarea unui element într-un vector

Fie a un vector cu n (n≤100) componente întregi și x o valoare întreagă. Verificați dacă x apare sau nu în vectorul a.

Practic, trebuie să verificăm dacă există în vectorul a un element cu proprietatea că este egal cu x. Particularizând secvența de verificare de la punctul b, obținem:

```
for (gasit=i=0; i<n && !gasit; i++)
     if (a[i]==x) gasit=1;  //am gasit valoarea x in vector
```

Această metodă de căutare, în care testăm succesiv elementele vectorului, se numește *căutare secvențială*. În cazul cel mai defavorabil (când x nu se găsește în vector sau este plasat pe ultima poziție), căutarea secvențială efectuează n comparații. O soluție mai eficientă nu există, în cazul general.

În realitate ne confruntăm frecvent cu problema căutării unui element într-o mulțime și frecvent mulțimea respectivă este ordonată (de exemplu, căutarea unui cuvânt în dicționar, a unui număr în cartea de telefon etc.). În astfel de situații nu aplicăm o căutare secvențială (de exemplu, pentru a găsi în cartea de telefon numărul lui Popescu Ion nu începem să căutăm de la litera 'A').

Cum procedăm? Deschidem cartea de telefon la întâmplare. Verificăm dacă am dat peste Popescu Ion. Dacă nu, verificăm dacă Popescu Ion este în prima parte a cărții sau în cea de a doua și continuăm căutarea numai în porțiunea respectivă. Procedeul se repetă până când găsim sau până când nu mai avem unde căuta.

Aceeași idee poate fi aplicată și pentru căutarea unui element într-un vector ordonat. Cum pentru calculator este dificil să lucreze „la întâmplare", vom lucra prin înjumătățiri succesive: mai întâi comparăm elementul căutat cu elementul din mijloc. Dacă este egal, am găsit, am terminat. Dacă nu este egal, verificăm dacă elementul căutat este mai mare decât elementul din mijloc (în acest caz, căutăm mai departe numai în cea de-a doua jumătate). Dacă elementul căutat este mai mic decât elementul din mijloc, căutăm mai departe numai în prima jumătate.

```
for (st=0, dr=n-1, gasit=0; !gasit && st<=dr;)
//cautam pe x de la pozitia st la pozitia dr; cautarea
//continua pana gasim sau pana nu mai avem unde cauta
    { mijloc=(st+dr)/2;                  //calculam mijlocul
      if (a[mijloc]==x) gasit =1;        //am gasit pe x
         else
         if (a[mijloc]<x) st=mijloc+1; //caut in stanga
             else dr=mijloc-1;}         //caut in dreapta
if (gasit) cout<<x<<" se gaseste pe pozitia "<<mijloc;
    else cout <<x<<" nu se afla in vector ";
```

Acest algoritm de căutare se numește *căutare binară*. Denumirea este sugestivă: la fiecare pas, alegem una din cele două alternative posibile: caut la stânga elementului din mijloc sau în dreapta lui.

Exerciţiu

Să considerăm un vector cu n=7 elemente ordonate crescător. Câte comparaţii execută algoritmul de căutare binară în cazul cel mai defavorabil (când elementul căutat nu se află în vector sau este depistat abia la ultima comparaţie)? Dar pentru n=10 elemente? Puteţi estima în general (în funcţie de n) numărul de comparaţii necesare în cazul cel mai defavorabil? Ce concluzie trageţi, comparând algoritmul de căutare binară cu algoritmul de căutare secvenţială?

Sortare

Fie n ($n \in \mathbf{N}^*$) elemente $a_0, a_1, \ldots, a_{n-1}$ dintr-o mulţime total ordonată. Ordonaţi crescător elementele $a_0, a_1, \ldots, a_{n-1}$.

Problema ordonării unor elemente (cunoscută şi sub denumirea de sortare) este frecvent întâlnită în practică şi din acest motiv a fost studiată intens. Ca urmare, au fost elaboraţi numeroşi algoritmi de sortare. Cum, de obicei, numărul de elemente care trebuie să fie ordonate este mare, s-a studiat şi eficienţa acestor algoritmi, în scopul elaborării unor algoritmi de sortare performanţi. Pentru început, vom studia algoritmi de sortare simpli, nu performanţi, urmând ca ulterior să învăţăm să evaluăm eficienţa acestor algoritmi şi să elaborăm algoritmi eficienţi.

Sortare prin selecţie

Sortarea prin selecţie are două variante: sortarea prin selecţia elementului maxim şi sortarea prin selecţia elementului minim. În ambele variante, ideea de bază este aceeaşi: se selectează cel mai mare element din vector (sau cel mai mic) şi se plasează pe ultima poziţie în vector (respectiv, pe prima poziţie). Apoi se calculează cel mai mare dintre elementele rămase şi se plasează pe penultima poziţie în vector (sau cel mai mic dintre elementele rămase şi se plasează pe a doua poziţie) ş.a.m.d. Acest procedeu se repetă de n-1 ori.

```
for (dr=n-1; dr>0; dr--)    //calculez maximul de la 0 la dr
    {for (max=a[0],pozmax=0,i=1; i<=dr; i++)
          if (a[i] > max)  max=a[i], pozmax=i;
     a[pozmax]=a[dr];       //plasez maximul pe pozitia dr
     a[dr] = max; }
```

Observaţie

La fiecare iteraţie a ciclului `for` exterior este calculat $\max\{a_0, a_1, \ldots, a_{dr}\}$ şi este plasat pe poziţia `dr`, elementele de la `dr+1` la `n-1` fiind deja plasate pe poziţiile lor definitive. Pentru a calcula $\max\{a_0, a_1, \ldots, a_{dr}\}$ sunt necesare `dr` comparaţii. Deci, în total se execută `1+2+...+n-1=n*(n-1)/2` comparaţii, indiferent de ordinea iniţială a elementelor vectorului.

Exerciţii

1. Modificaţi algoritmul precedent, astfel încât să realizeze ordonarea descrescătoare a elementelor vectorului.
2. Modificaţi algoritmul precedent, astfel încât să realizeze ordonarea prin selecţia minimului.

*Sortare prin compararea vecinilor (*bubblesort*)*

O altă metodă de sortare este de a parcurge vectorul, comparând fiecare două elemente vecine şi (dacă este cazul) interschimbându-le. Cum într-o singură trecere nu se poate realiza sortarea vectorului, acest procedeu se repetă până când vectorul devine sortat (la ultima trecere nu am mai efectuat nici o interschimbare).

```
do
  {schimb=0;          //initial nu am facut nici o schimbare
   for (i=0; i<n-1; i++)                  //parcurg vectorul
        if (a[i]>a[i+1])      //compar doua elemente vecine
           { //nu sunt in ordine, le interschimb
             aux=a[i]; a[i]=a[i+1]; a[i+1]=aux;
             schimb=1;}       //retin ca am facut schimbari
  }    //procedeul se repeta cat timp se executa schimbari
while (schimb);
```

Observaţii

1. Metoda este denumită şi *bubblesort* (metoda bulelor) deoarece la fiecare nouă trecere prin vector elementele cu valori mari migrează către sfârşitul vectorului („urcă" în vector, aşa cum bulele de aer se ridică la suprafaţă când fierbe apa).
2. În cazul cel mai defavorabil (când valorile sunt ordonate descrescător), această metodă execută aproximativ n^2 operaţii.

Exerciţii

1. Modificaţi algoritmul precedent, astfel încât să realizeze ordonarea descrescătoare a elementelor vectorului.
2. Observăm că, dacă la parcurgerea curentă ultima interschimbare a fost efectuată la poziţia x, în oricare dintre parcurgerile ulterioare nu se vor efectua interschimbări după poziţia x. Îmbunătăţiţi algoritmul precedent utilizând această observaţie.

Sortare prin inserţie

Metoda de sortare prin inserţie este, de asemenea, o metodă simplă, pe care o utilizăm adesea când ordonăm cărţile la jocuri de cărţi: de fiecare dată când tragem o carte, o plasăm pe poziţia sa corectă, astfel încât în mână cărţile să fie ordonate.

Utilizând această idee, sortăm vectorul astfel: parcurgem vectorul, element cu element; la fiecare pas `i`, căutăm poziţia corectă a elementului curent `a[i]`, astfel încât secvenţa `a[0], a[1], ..., a[i]` să fie ordonată:

```
for (i=1; i<n; i++)
    {v=a[i];                    //caut pozitia lui v
     for (poz=i; poz && a[poz-1]>v; poz--)
         a[poz]=a[poz-1];   //mut la dreapta elementele > v
     a[poz] = v;  }      //poz este pozitia corecta pentru v
```

Exerciţiu

Modificaţi algoritmul precedent, astfel încât să realizeze ordonarea descrescătoare a elementelor vectorului.

Sortare prin numărarea aparițiilor

În anumite probleme, elementele vectorului au ca valori numere naturale dintr-un interval [0,Max) de dimensiune redusă (sau au valori care pot fi asociate numerelor naturale dintr-un astfel de interval). Prin dimensiune redusă înţelegem că se poate declara un vector V cu Max componente întregi.

În acest caz particular, cea mai bună metodă este sortarea prin numărarea aparițiilor: se numără pentru fiecare valoare din intervalul [0, Max) de câte ori apare în vector.

```
#include <iostream.h>
#define Max 1000
#define NrMax 10000

int n, V[Max], a[NrMax];
int main()
{int x, i, j, nr;
 cout<<"n="; cin>>n;
 for (i=0; i<n; i++)
     {cin>>x;  //citesc o noua valoare
     V[x]++;} //marim numarul de aparitii ale valorii x
//plasam in ordine elementele in vectorul a
for (nr=i=0; i<Max; i++)
    for (j=0; j<V[i]; j++)
        a[nr++]=i;
for (i=0; i<n; i++) cout<<a[i]<<' ';
cout<<endl; return 0;}
```

Observaţie

Numărul de operaţii efectuate de acest algoritm de sortare este de ordinul lui n+Max.

Interclasare

Fie a un vector cu n elemente şi b un vector cu m elemente, ordonaţi crescător. Să se construiască un al treilea vector, c, care să conţină atât elementele vectorului a, cât şi elementele vectorului b, în ordine crescătoare.

O soluţie (ineficientă, care nu ţine cont de faptul că a şi b sunt deja ordonaţi) este să copiem în vectorul c elementele din a şi din b şi apoi să sortăm vectorul c.

O altă idee este de a parcurge simultan cei doi vectori, comparând la fiecare pas elementul curent din a cu elementul curent din b. Cel mai mic dintre cele două elemente va fi copiat în vectorul c, avansând doar în vectorul din care am copiat. Când am epuizat elementele din unul din cei doi vectori, copiem elementele rămase în celălalt, deoarece nu mai avem cu ce le compara.

```
for (i=j=k=0; i<n && j<m;)
     if (a[i]<b[j])    //copiez in c elementul cel mai mic
        c[k++]=a[i++];                          //copiez din a
        else c[k++]=b[j++];                     //copiez din b
//copiez eventualele elemente ramase in a
for (; i<n; i++)
    c[k++]=a[i];
//copiez eventualele elemente ramase in b
for (; j<m; j++)
    c[k++]=b[j];
```

Exerciţiu

Estimaţi în funcţie de n şi m numărul de comparaţii necesare pentru interclasare, în cazul cel mai defavorabil.

4.3. *Prelucrări elementare cu matrice*

Citirea unei matrice

Citirea unei matrice se realizează citind elementele matricei, linie cu linie. Să considerăm ca exemplu o matrice a cu n linii şi m coloane (n,m≤100), cu elemente de tip int.

```
int a[100][100], n, m, i, j;          //declarare matrice
cout << "n, m= "; cin >> n >> m;
for (i=0; i<n; i++)          //parcurg matricea linie cu linie
    for (j=0; j<m; j++)    //citesc elementele liniei i
        {cout <<"a["<< i << "][" <<j << "]="; cin>>a[i][j];}
```

Afişarea unei matrice

Afişarea unei matrice se realizează afişând elementele matricei, linie cu linie:

```
for (i=0; i<n; i++)      //parcurg matricea linie cu linie
   {for (j=0; j<m; j++) //afisez componentele de pe linia i
      cout <<a[i][j]<<' ';
    cout<<endl;}         //trec la linia urmatoare
```

Parcurgerea unei matrice pe linii

Unele probleme impun parcurgerea unei matrice linie cu linie şi prelucrarea separată a elementelor de pe fiecare linie în parte. De exemplu, să calculăm suma elementelor de pe fiecare linie a unei matrice:

```
for (i=0; i<n; i++)
    {for (s=0, j=0; j<m; j++) s+=a[i][j];
     cout<<"Suma de pe linia "<<i<<" : " << s <<'\n';}
```

Parcurgerea unei matrice pe coloane

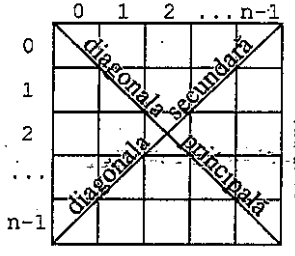
Există probleme care impun parcurgerea unei matrice pe coloane şi prelucrarea separată a elementelor de pe fiecare coloană în parte. De exemplu, să calculăm suma elementelor de pe fiecare coloană a unei matrice:

```
for (j=0; j<m; i++)
    {for (s=0, i=0; i<n; j++) s+=a[i][j];
     cout<<"Suma de pe coloana "<<j<<" :" <<s<<endl;}
```

4.4. *Prelucrări elementare specifice matricelor pătratice*

O matrice se numeşte *pătratică* dacă numărul de linii este egal cu numărul de coloane ale matricei.

Datorită acestei particularităţi, se pot realiza prelucrări specifice matricelor pătratice, de exemplu: parcurgerea diagonalelor sau a elementelor de sub sau de deasupra unei diagonale.



Parcurgerea diagonalelor

O matrice pătratică are două diagonale: diagonala principală (care uneşte primul element al matricei, `a[0][0]`, cu ultimul element, `a[n-1][n-1]`) şi diagonala secundară (care uneşte ultimul element de pe prima linie, `a[0][n-1]`, cu primul element de pe ultima linie, `a[0][n-1]`).

Prin urmare, elementele de pe diagonala principală sunt `a[0][0]`, `a[1][1]`, `a[2][2]`, ..., `a[n-1][n-1]`.

Pe diagonala principală indicele de linie este egal cu indicele de coloană.

Afişarea elementelor de pe diagonala principală se poate realiza astfel:

```
for (i=0; i<n; i++) cout <<a[i][i]<<' ';
```


Elementele de pe diagonala secundară sunt `a[0][n-1]`, `a[1][n-2]`, `a[2][n-3]`, ..., `a[n-1][0]`. Observăm că pe diagonala secundară când indicele de linie creşte cu 1, indicele de coloană scade cu 1.

Pe diagonala secundară suma dintre indicele de linie şi indicele de coloană este constantă:

```
linie+coloana=constant=n-1=>coloana=n-1-linie
```

Afişarea elementelor de pe diagonala secundară se poate realiza astfel:

```
for (i=0; i<n; i++) cout <<a[i][n-i-1]<<' ';
```

Parcurgerea elementelor de sub diagonala principală

Să presupunem că dorim să calculăm suma elementelor de sub diagonala principală (exclusiv). Observăm că, pe linia `i` (`i` variind de la `1` la `n-1`), indicele de coloană variază între `0` şi `i-1` (deoarece pe linia `i` elementul `a[i][i]` este deja pe diagonala principală).

```
for (s=0, i=1; i<n; i++)
    for (j=0; j<i; j++) s+=a[i][j];
```

Exerciţii

1. Modificaţi secvenţa de instrucţiuni precedentă astfel încât să calculeze suma elementelor de deasupra diagonalei principale.
2. Modificaţi secvenţa de instrucţiuni precedentă astfel încât să calculeze suma elementelor de sub diagonala secundară.
3. Modificaţi secvenţa de instrucţiuni precedentă astfel încât să calculeze suma elementelor de deasupra diagonalei secundare.

4.5. *Şiruri de caractere*

Un şir de caractere este o succesiune de caractere terminată cu caracterul `NULL`. Caracterul `NULL` este caracterul care are codul ASCII `0`.

În limbajul C/C++ nu există un tip special pentru şirurile de caractere, acestea pot fi implementate ca vectori de caractere. Prin urmare, declararea unei variabile `nume`, ce reprezintă un şir de `Lg` caractere, se poate realiza astfel:

```
char nume[Lg];
```

Putem iniţializa o variabilă şir de caractere chiar de la declarare, cu o constantă şir (în acest caz, lungimea şirului `Lg` poate lipsi, fiind determinată automat).

Exemplu

```
char s1[]="Mama are mere", s2[50]="Vasile nu";
```

Pentru şirul s1, lungimea a fost determinată automat: 13 octeţi (câte unul pentru fiecare dintre cele 13 caractere din şir) + 1 octet suplimentar pentru marcajul de sfârşit de şir, deci în total 14 octeţi. Pentru s2, lungimea a fost specificată (50 de octeţi), primii 9 octeţi fiind iniţializaţi cu cele 9 caractere ale şirului "Vasile nu", iar cel de-al zecelea octet fiind iniţializat cu marcajul de sfârşit al şirului.

Şirurile de caractere pot fi prelucrate la nivel de caracter (pot fi parcurse caracter cu caracter, ca un vector de caractere) sau pot fi prelucrate la nivel de structură (cu ajutorul funcţiilor existente în bibliotecile limbajului).

Observaţie

Din modul de reprezentare a unui şir de caractere deducem că o constantă caracter (de exemplu, 'a') nu este echivalentă cu o constantă şir de caractere (de exemplu cu "a"). Constanta 'a' este stocată pe un singur octet care conţine codul ASCII al caracterului. Constanta "a" este stocată pe doi octeţi (primul conţine codul ASCII al caracterului, iar al doilea, marcajul de sfârşit de şir – NULL).

4.6. *Citirea şi afişarea şirurilor de caractere*

Să presupunem că am declarat un şir s de maxim 100 de caractere astfel:

```
char s[100];
```

Citirea unui şir de caractere în limbajul C++

Citirea şirului s se poate face utilizând operatorul uzual de citire >>:

```
cin >> s;
```

În acest caz, se vor citi în şirul s toate caracterele până la primul spaţiu. De exemplu, dacă fluxul de intrare conţine caracterele Mama are mere, după citire, şirul s va fi Mama.

Atenţie! Dacă numărul de caractere până la primul spaţiu este mai mare decât lungimea declarată a şirului, citirea se face în continuare şi puteţi distruge astfel date sau cod de program. Efectul este indefinit, dar cu siguranţă este *eroare*!

Acest mod de funcţionare a operatorului >> poate fi inadecvat în multe situaţii. Din acest motiv, în clasa istream (pe care o utilizăm când realizăm citiri de la tastatură) există două funcţii speciale pentru citirea caracterelor şi a şirurilor de caractere: get() şi getline(). Aceste două funcţii sunt incluse în clasa istream (se numesc funcţii membre) şi pentru a le putea utiliza trebuie să specificăm fluxul de intrare din care citim şi numele funcţiei, separate prin *operatorul de selecţie* '.': cin.get() sau cin.getline().

Funcţia getline() are următorul format:

```
getline (char * s, int n, char c='\n');
```


Primul parametru este întotdeauna şirul în care se citeşte. Al doilea parametru reprezintă numărul maxim de caractere care se pot citi. Al treilea parametru, dacă apare, este un caracter delimitator (dacă nu apare, acest caracter este considerat implicit ca fiind '\n'). Ca efect, funcţia citeşte caractere de la intrare până la întâlnirea caracterului delimitator sau până ce a citit exact n-1 caractere.

De exemplu, dacă dorim să citim în şirul s o linie de la tastatură, apelăm funcţia getline() cu doi parametri (deoarece marcajul de sfârşit de linie este '\n'):

```
cin.getline(s,100);
```

Dacă dorim să citim de la tastatură toate caracterele până la întâlnirea caracterului '.', vom apela funcţia getline(), cu trei parametri, astfel:

```
cin.getline(s, 100, '.');
```

Funcţia get() are mai multe forme de apel. Dacă dorim să o utilizăm pentru citirea şirurilor de caractere, apelul este similar cu apelul funcţiei getline(). Şi efectul este asemănător. Diferenţa constă în faptul că getline() extrage din fluxul de intrare caracterul delimitator, în timp ce get() nu îl extrage.

Putem apela funcţia get() fără nici un parametru (caz în care returnează un caracter citit de la intrare) sau cu un parametru de tip char (caz în care parametrul va avea, după citire, valoarea caracterului citit de la intrare).

Observaţie

La citire, marcajul de sfârşit de şir NULL este automat plasat la sfârşitul şirului.

Citirea unui şir de caractere în limbajul C

Un şir de caractere poate fi citit cu ajutorul funcţiei de citire scanf(), utilizând specificatorul de format %s. De exemplu:

```
char s[100];
scanf("%s", s);
```

Observaţi că nu am transmis funcţiei scanf() adresa şirului s (&s). Acest lucru se datorează faptului că numele oricărui vector (deci, implicit, şi al unui şir de caractere) este adresa primului element din vector.

Vor fi citite caracterele care încep cu primul caracter diferit de un caracter alb (spaţiu, tab sau *newline*) până la întâlnirea unui caracter alb. Dacă numărul de caractere citite este mai mare decât lungimea declarată a şirului, veţi obţine eroare[24] la execuţia programului.

24. Eroarea depinde de sistemul de operare şi de mediul de programare uitilizat. De exemplu, dacă lucraţi sub sistemul de operare Linux, veţi obţine mesajul de eroare Segmentation Fault. Acest mesaj de eroare se obţine ori de câte ori încercaţi să accesaţi o zonă de memorie nealocată. Dacă utilizaţi mediul de programare Borland, comportamentul programului este imprevizibil, dar cu maximă probabilitate aplicaţia se va bloca.

Dacă doriți să limitați numărul de caractere care se citesc, puteți indica în specificatorul de format numărul maxim de caractere. De exemplu:

```
scanf ("%9s", s);
```

Observați că, pentru a citi corect un șir de 10 caractere, în specificatorul de format am precizat lungimea maximă 9, deoarece este necesar un octet suplimentar pentru a memora marcajul de sfârșit de șir (caracterul `NULL`).

Funcția `scanf()` nu permite citirea șirurilor de caractere care conțin caractere albe. Pentru a citi șiruri de caractere care conțin spații sau caractere tab se poate utiliza funcția `gets()`, declarată în fișierul antet `stdio.h`. Prototipul funcției `gets()` este:

```
char *gets(char *s);
```

Funcția citește în șirul `s` caracterele introduse de la tastatură, până la întâlnirea caracterului marcaj de sfârșit de linie (*newline*). Caracterul *newline* este citit și convertit automat în caracterul `NULL`.

Observați că, în prototipul funcției `gets()`, faptul că funcția are ca parametru un șir de caractere și returnează ca rezultat un șir de caractere este specificat într-o formă neobișnuită. Practic, se indică drept parametru/rezultat al funcției un pointer cu tipul de bază `char` (adresa de memorie a unui caracter). În secțiunea următoare vom studia tipul pointer, precum și legătura dintre pointeri și tablouri. De exemplu, pentru a citi un șir `s` care, eventual, poate conține și caractere tab sau spații apelăm funcția `gets()` astfel:

```
gets(s);
```

Afișarea unui șir de caractere în limbajul C++

Afișarea unui șir de caractere se face cu ajutorul operatorului de scriere <<:

```
cout << s;
```

Se vor scrie caracterele până la întâlnirea primului marcaj de sfârșit de șir (`NULL`). De exemplu, dacă `s` este "`mama are \n mere`" se va afișa:

```
mama are
 mere
```

Dacă `s` este "`mama are \n mere \0 de aici nu mai scrie`" efectul va fi același (scrie până la '`\0`', caracterul `NULL`).

Afișarea unui șir de caractere în limbajul C

Afișarea unui șir de caractere se poate face cu ajutorul funcției `printf()`, utilizând specificatorul de format `%s`. De exemplu:

```
char s[100];
printf("%s", s);
```

În fișierul antet `stdio.h` este declarată o funcție specială pentru afișarea șirurilor de caractere, funcția `puts()`. Prototipul acestei funcții este:


```
int puts(const char *s);
```

Funcția afișează caracterele șirului `s` până la întâlnirea caracterului `NULL` și apoi afișează un caracter *newline* (de trecere la linie nouă). În caz de afișare cu succes, funcția returnează o valoare pozitivă; în caz contrar, returnează valoarea `-1`. Prin urmare, apelul:

```
puts(s);
```

este echivalent cu:

```
printf("%s\n", s);
```

4.7. *Tipul pointer*

Un *pointer* este o dată care are ca valori adrese de memorie.

Facilitățile oferite de limbajul C/C++ pentru lucrul cu pointeri reprezintă unul dintre cele mai importante atuuri ale limbajului. Pointerii sunt utilizați cu precădere în următoarele situații: lucrul cu tablouri, transferul parametrilor funcțiilor, accesul direct la memorie și alocarea dinamică a memoriei. Pentru a înțelege mai bine modul de lucru cu tablouri și cu șiruri de caractere, ne interesează pointerii de date (pointeri care conțin adresa zonei de memorie a unei variabile).

Declararea unui pointer de date

Declararea unui pointer de date are formatul general:

```
tip * variabila_pointer;
```

unde `tip` reprezintă *tipul de bază* al pointerului, care indică tipul datelor memorate la adresa conținută în `variabila_pointer`.

Exemple

```
int * p1;
```

`p1` este un pointer cu tipul de bază `int`, deci va conține adresa unei zone de memorie la care este memorat un număr întreg (de tip `int`).

```
char * s;
```

`s` este un pointer cu tipul de bază `char`, deci va conține adresa unei zone de memorie la care este memorat un caracter.

Observații

1. Declarația precedentă este asemănătoare cu declarația unei variabile uzuale, faptul că variabila declarată este un pointer fiind indicat de prezența asteriscului.
2. Tipul de bază al pointerului poate fi orice tip al limbajului, inclusiv tipul `void`. Un pointer cu tipul de bază `void` (`void *`) se numește pointer generic.
3. Indicarea tipului de bază al pointerului este esențială. Adresa unei zone de memorie este de fapt adresa primului octet din zona respectivă (o valoare numerică). Indicând tipul de bază al pointerului, se poate determina, în primul rând, dimensiunea zonei de memorie a cărei adresă este memorată în pointer (de exemplu, 1

octet pentru `char *`, 4 octeți pentru `long int *` etc). Dimensiunea zonei de memorie nu este suficientă, pentru a afla valoarea memorată într-o anumită zonă de memorie trebuie să știm și codul utilizat pentru reprezentarea datelor în acea zonă. De exemplu, într-o zonă de 4 octeți s-ar putea afla un număr natural de tip `unsigned long int`, un număr întreg de tip `long int`, un număr real de tip `float` sau un șir de 4 caractere. Cum decodificăm o secvență de octeți dacă nu știm codul utilizat pentru reprezentarea datelor în acea secvență? Prin urmare, tipul de bază al pointerului ne indică atât dimensiunea zonei a cărei adresă de început este memorată în pointer, cât și codul utilizat pentru reprezentarea datelor în acea zonă.

4. Există o constantă pointer specială, denumită `NULL`. Semnificația acestei constante este „pointerul nu conține adresa nici unei zone de memorie". Valoarea acestei constante este `0`.

Operații cu pointeri

1. Operația de referențiere

Este operația prin care putem obține adresa de memorie a unei variabile. Rezultatul acestei operații este un pointer (care are ca valoare adresa primului octet al zonei de memorie în care este stocată variabila). Referențierea unei variabile se realizează cu ajutorul operatorului unar `&` (numit operatorul de referențiere sau operatorul adresă):

```
&variabila
```

2. Operația de dereferențiere

Este operația prin care putem accesa valoarea memorată într-o zonă de memorie a cărei adresă de început este memorată într-un pointer. Dereferențierea unui pointer se realizează utilizând operatorul unar * (numit operator de dereferențiere):

```
*variabila_pointer
```

Această expresie are ca valoare conținutul zonei de memorie a cărei adresă este memorată în `variabila_pointer`.

Exemplu

```
int i=10, *p;
p=&i;
```

Referențiere – atribuim pointerului `p` adresa variabilei `i`.

```
cout << *p;
```

Dereferențiere – afișez conținutul zonei de memorie a cărei adresă este memorată în `p` (valoarea `10`).

3. Incrementare/decrementare

Asupra unui pointer se poate aplica operatorul de incrementare `'++'` și operatorul de decrementare `'--'`. Ca efect, se mărește/micșorează adresa memorată în pointer

cu numărul de octeți necesari pentru a memora o dată de tipul de bază al pointerului (`sizeof(tip)`).

Exemple

```
long int * p;
p++;
```

Adresa memorată în `p` se mărește cu `4=sizeof(long int)`.

```
char * p;
p--;
```

Adresa memorată în `p` se micșorează cu `1=sizeof(char)`.

Efectul este natural: pointerul indică următorul (++) sau precedentul (--) element de tipul său de bază.

4. Adunarea/scăderea dintre un pointer și un număr întreg

Se poate realiza adunarea (+) dintre un pointer și un întreg sau scăderea dintre un pointer și un întreg: `p+n` sau `p-n`. Ca efect, adresa memorată în pointer se mărește (+) sau se micșorează (-) cu `n*sizeof(tip)`.

Deci, `p+n` ar indica al `n`-lea element de tipul de bază al pointerului care urmează după adresa `p`; `p-n` ar indica al `n`-lea element de tipul de bază al pointerului care precedă elementul de la adresa `p`.

5. Scăderea dintre doi pointeri

Se poate realiza scăderea între doi pointeri numai dacă aceștia au același tip de bază. Rezultatul este o valoare întreagă care reprezintă diferența dintre adrese, raportată la `sizeof(tip)`.

6. Comparații

Asupra pointerilor care au același tip de bază se pot aplica operatorii relaționali și de egalitate, cu semnificația uzuală.

Exemplu

```
if (p==NULL) ...
```

7. Afișarea unui pointer

Pentru a afișa valoarea unui pointer în limbajul C se utilizează funcția `printf()` cu specificatorul de format `%p`. De exemplu:

```
int a, *q=&a;
printf("%p", q);
```

Pe ecran va fi afișată adresa variabilei `a`, în baza 16.

Afișarea unui pointer în limbajul C++ se face cu ajutorul operatorului uzual de afișare. De exemplu:

```
cout<<q;
```

4.8. *Legătura dintre pointeri şi tablouri*

Numele unui tablou este un *pointer constant* care are ca valoare adresa primului element din tablou.

Să considerăm următoarea declaraţie:

```
tip v[LgMax];
```

Următoarele expresii sunt echivalente:

`v`	`&v[0]`	adresa primului element din tablou
`v+i`	`&v[i]`	adresa elementului de pe poziţia `i` din tablou
`*v`	`v[0]`	primul element din tablou
`*(v+i)`	`v[i]`	elementul de pe poziţia `i` din tablou
`v++`	`v[1]`	elementul de pe poziţia `1`

Operaţiile cu pointeri reprezintă o alternativă pentru adresarea elementelor unui tablou.

Exemplu

Să considerăm `s` un şir de maxim 100 de caractere.

```
char s[100];
```

Să determinăm lungimea efectivă a şirului de caractere `s`. Determinarea lungimii unui şir se poate face parcurgând şirul până la întâlnirea marcajului de sfârşit de şir şi numărând caracterele parcurse.

Pentru a parcurge şirul putem utiliza variabila întreagă `lungime` (care indică poziţia curentă în şir, iar la sfârşit lungimea şirului):

```
int lungime;
for (lungime=0; s[lungime]; lungime++);
```

Şirul poate fi parcurs şi cu ajutorul unui pointer `p` care iniţial va avea ca valoare adresa de început a şirului. Pointerul va fi incrementat cât timp valoarea lui nu este adresa zonei de memorie care conţine caracterul `NULL`. Lungimea şirului este diferenţa dintre valoarea pointerului `p` (care indică la sfârşit adresa caracterului `NULL`) şi valoarea pointerului `s` (care indică adresa primului caracter din şir).

```
char *p;
for (p=s; *p; p++);
lungime=p-s;
```

Observaţii

1. Acum putem explica de ce atribuirea `a=b` produce mesajul de eroare *„Lvalue required"* atunci când încercăm să atribuim unui tablou `a` tabloul `b`: membrul stâng al acestei atribuiri este un pointer constant. Această atribuire este tentativa de a modifica un pointer constant (încercăm să atribuim pointerului constant `a` valoarea pointerului `b`).

2. O matrice este un vector de vectori, deci numele său este un pointer la prima linie din matrice. Să considerăm următoarea declaraţie:

```
tip v[LgMax][LgMax];
```

Următoarele expresii sunt echivalente:

`v`	`&v[0]`		adresa primei linii din matrice
`v+i`	`&v[i]`		adresa liniei `i` din matrice
`*v`	`v[0]`	`&v[0][0]`	adresa primului element de pe linia 0
`*(v+i)`	`v[i]`	`&v[i][0]`	adresa primului element de pe linia `i`
`*(*(v+i)+j)`		`v[i][j]`	elementul de pe linia `i` şi coloana `j`

Utilizarea funcţiilor standard de lucru cu şiruri de caractere

În fişierul antet `string.h` sunt declarate numeroase funcţii de lucru cu şiruri de caractere.

Determinarea lungimii efective a unui şir de caractere

În secţiunea precedentă am determinat lungimea efectivă a unui şir de caractere parcurgând şirul până la întâlnirea caracterului `NULL`. O variantă mai simplă este de a apela funcţia `strlen()`. Prototipul acestei funcţii este:

```
unsigned strlen(const char *s);
```

Apelul `strlen(s)` returnează lungimea şirului `s`.

Copierea unui şir de caractere într-un alt şir de caractere

Copierea unui şir de caractere (denumit `sursa`) într-un alt şir de caractere (denumit `destinatia`) se poate realiza cu ajutorul funcţiei `strcpy()`. Prototipul funcţiei `strcpy()` este:

```
char * strcpy(char *destinatia, const char *sursa);
```

Apelul `strcpy(destinatie, sursa)` copiază şirul `sursa`, caracter cu caracter, în şirul `destinatie`, până după copierea caracterului `NULL` şi returnează adresa de început a şirului `destinatie`.

Efectul acestui apel este echivalent cu următoarea secvenţă de instrucţiuni:

```
int i;
for (i=0; sursa[i]; i++) //parcurg sirul sursa
     //copiez caracterul curent in sirul destinatie
     destinatie[i]=sursa[i];
destinatie[i]=NULL; //plasez la sfarsit caracterul NULL
```

Exerciţiu

Scrieţi o secvenţă echivalentă care să utilizeze pointeri pentru parcurgerea şirurilor.

Observaţie

Spaţiul de memorie alocat şirului `destinatie` trebuie să fie suficient de mare pentru a memora caracterele şirului `sursa`.

Copierea unui prefix al unui şir de caractere

Pentru a copia primele `n` caractere din şirul de caractere `sursa` în şirul de caractere `destinatie` se poate utiliza funcţia `strncpy()`. Prototipul funcţiei `strncpy()` este:

```
char * strncpy(char *destinatie, char *sursa, unsigned n);
```

Apelul `strncpy(destinatie,sursa,n)` are ca efect copierea primelor maxim `n` caractere din şirul `sursa` în şirul `destinatie` şi returnează adresa de început a şirului `destinatie`. Dacă lungimea şirului `sursa` este mai mică decât `n`, se copiază întreg şirul `sursa` în şirul `destinatie` şi la sfârşitul şirului `destinatie` se plasează caracterul `NULL`. Dacă lungimea şirului `sursa` este mai mare sau egală cu `n`, atunci la sfârşitul şirului `destinatie` nu va fi plasat caracterul `NULL`.

Efectul acestui apel este echivalent cu următoarea secvenţă de instrucţiuni:

```
int i=-1;
do
  {i++; destinatie[i]=sursa[i];}
while (sursa[i] && i<n-1);
```

Concatenarea a două şiruri de caractere

Pentru a concatena şirul de caractere `destinatie` cu şirul de caractere `sursa` se utilizează funcţia `strcat()`. Prototipul funcţiei `strcat()` este:

```
char * strcat(char *destinatie, const char *sursa);
```

Apelul `strcat(destinatie,sursa)` copiază caracterele şirului `sursa` la sfârşitul şirului `destinatie` şi returnează adresa de început a şirului obţinut după concatenare (`destinatie`).

Efectul acestui apel este echivalent cu următoarea secvenţă de instrucţiuni:

```
int i, lgd=strlen(destinatie);
for (i=0; sursa[i]; i++)
     destinatie[i+lgd]=sursa[i];
destinatie[lgd+i]=NULL;
```

Observaţie

Zona de memorie alocată şirului `destinatie` trebuie să fie suficient de mare pentru a reţine rezultatul concatenării.

Concatenarea unui şir de caractere cu un prefix al altui şir de caractere

Pentru a concatena primele `n` caractere din şirul de caractere `sursa` la şirul de caractere `destinatie` se poate utiliza funcţia `strncat()`. Prototipul funcţiei `strncat()` este:


```
char *strncat(char *destinatie, char *sursa, unsigned n);
```

Apelul `strncat(destinatie,sursa,n)` are ca efect copierea primelor maxim `n` caractere din şirul `sursa` la sfârşitul şirului `destinatie` şi returnează adresa de început a şirului `destinatie`. Dacă lungimea şirului `sursa` este mai mică decât `n`, se concatenează întreg şirul `sursa` la şirul `destinatie`. La sfârşitul şirului obţinut în urma concatenării se plasează caracterul `NULL`.

Efectul acestui apel este echivalent cu următoarea secvenţă de instrucţiuni:

```
int i, lgd=strlen(destinatie);
for (i=0; sursa[i] && i<n; i++)
     destinatie[i+lgd]=sursa[i];
destinatie[i+lgd]=NULL;
```

Compararea a două şiruri de caractere

Pentru a compara două şiruri de caractere se poate utiliza funcţia `strcmp()`. Prototipul acestei funcţii este:

```
int strcmp(const char *s1, const char *s2);
```

Apelul `strcmp(s1,s2)` compară din punct de vedere lexicografic şirul de caractere `s1` cu şirul de caractere `s2` şi returnează o valoare:

`<` 0, dacă `s1<s2` (în ordine lexicografică, ordinea cuvintelor din dicţionar);
`==`0, dacă `s1==s2` (`s1` şi `s2` sunt egale);
`>` 0, dacă `s1>s2` (în ordine lexicografică).

Observaţie

Spunem că şirul `s1` precedă din punct de vedere lexicografic şirul `s2` dacă există un număr natural `n≤strlen(s1)`, astfel încât `s1[i]=s2[i]` pentru orice `0≤i<n` şi `s1[n]<s2[n]`.

Efectul acestui apel este echivalent cu următoarea secvenţă de instrucţiuni:

```
int ok, i; //variabila ok va retine rezultatul testului
for (i=0; s1[i] && s2[i] && s1[i]==s2[i]; i++);
if (!s1[i] && !s2[i]) ok=0;
   else
   if (s1[i]<s2[i]) ok=-1;
      else ok=1;
```

Observaţie

Utilizând funcţia `strcmp()`, la comparare se face distincţie între literele mari şi literele mici. Dacă dorim o comparare care să nu facă distincţie între majuscule şi minuscule (*case insensitive*), putem utiliza funcţia `stricmp()`:

```
int stricmp(const char *s1, const char *s2);
```

Căutarea primei apariții a unui caracter într-un șir

Pentru a căuta prima apariție a unui caracter într-un șir se poate utiliza funcția `strchr()`. Prototipul acestei funcții este:

```
char * strchr(const char *s, int c);
```

Apelul `strchr(s,c)` caută prima apariție a caracterului `c` în șirul `s` și returnează un pointer care are ca valoare adresa primului caracter din șir egal cu `c` (dacă un astfel de caracter există) sau `NULL` (dacă în `s` caracterul `c` nu apare).

Efectul acestui apel este echivalent cu următoarea secvență de instrucțiuni:

```
char *p;
for (p=s; *p && *p!=c; p++);
if (!*p) p=NULL;
```

Exercițiu

Rescrieți această secvență de instrucțiuni utilizând indici pentru parcurgerea șirului `s`.

Căutarea primei apariții a unui șir într-un alt șir

Pentru a căuta prima apariție a unui șir într-un alt șir se poate utiliza funcția `strstr()`. Prototipul acestei funcții este:

```
char *strstr(const char *s1, const char *s2);
```

Apelul `strstr(s1,s2)` determină prima apariție a șirului `s2` în șirul `s1` și returnează un pointer către începutul primei apariții a lui `s2` în `s1` sau `NULL` (dacă `s2` nu apare în `s1`).

Efectul acestui apel este echivalent cu următoarea secvență de instrucțiuni:

```
int i, j, lg1=strlen(s1), lg2=strlen(s2);
char *p=NULL;
for (i=0; i<=lg1-lg2; i++)
    {
     for (j=0; s2[j] && s2[j]==s1[i+j]; j++);
     if (!s2[j]) {p=s1+i; break;}
    }
```

Transformări între litere mici și litere mari

Pentru a transforma toate literele mari dintr-un șir în litere mici se poate utiliza funcția `strlwr()`. Prototipul acestei funcții este:

```
char *strlwr(char *s);
```

Pentru a transforma toate literele mici dintr-un șir în litere mari se poate utiliza funcția `strupr()`. Prototipul acestei funcții este:

```
char *strupr(char *s);
```


Exercițiu

Scrieți câte o instrucțiune care să aibă un efect echivalent cu funcția `strlwr()`, respectiv cu `strupr()`.

Separarea unităților lexicale dintr-un șir de caractere

Numim unitate lexicală o secvență de caractere din șir delimitată de separatori. Separatorii depind de problema concretă și sunt memorați într-un șir de caractere (pe care l-am denumit `separatori`).

Funcția `strtok()` permite extragerea succesivă a unităților lexicale dintr-un șir de caractere. Prototipul acestei funcții este:

```
char *strtok(char *s, const char *separatori);
```

Referitor la modul de apel, această funcție este mai specială. Prima dată vom apela funcția sub forma:

```
strtok(s,separatori);
```

După primul apel, funcția va returna adresa de început a primei unități lexicale din șirul `s`. După acest prim apel, șirul `s` este modificat (primul separator de după prima unitate lexicală este înlocuit de caracterul `NULL`. Dacă șirul `s` este vid sau conține numai separatori, atunci funcția returnează `NULL`.

Pentru a extrage succesiv și celelalte unități lexicale din șirul `s` putem apela din nou funcția `strtok()`, dar de data aceasta primul parametru trebuie să fie `NULL` (altfel, funcția returnează tot prima unitate lexicală din `s`):

```
strtok(NULL, separatori);
```

Acest apel returnează adresa de început a următoarei unități lexicale din șirul `s` (sau `NULL`, dacă în șirul `s` nu mai există unități lexicale). După fiecare apel, primul separator de după unitatea lexicală identificată este înlocuit de caracterul `NULL`.

Exemplu

Să considerăm următoarele șiruri:

```
char s[]=" Vai, vai!  Ce necaz!!!!\nUnde e pisica?";
char sep[]=" ,!?\n;.:";                    //sirul de separatori
```

La primul apel:

```
char *p=strtok(s,sep);
```

variabilei `p` i se atribuie adresa de început a primei unități lexicale (`Vai`), iar caracterul separator `' '` de pe poziția 4 este înlocuit de caracterul `NULL`. Dacă vom afișa șirul a cărui adresă de început este memorată în `p`:

```
cout<<p;
```

vom obține pe ecran prima unitate lexicală:

```
Vai
```

La al doilea apel:

```
p=strtok(NULL, sep);
```

variabilei `p` i se atribuie adresa de început a celei de-a doua unități lexicale (`vai`), iar caracterul separator `'!'` este înlocuit de `NULL`.

La al treilea apel:

```
p=strtok(NULL, sep);
```

variabilei `p` i se atribuie adresa de început a celei de-a treia unități lexicale (`Ce`), iar caracterul separator `' '` este înlocuit de `NULL`.

Pentru a extrage toate unitățile lexicale din șirul `s` separate de caractere din șirul de separatori `sep` putem utiliza următoarea secvență de instrucțiuni:

```
int i=0;
p=strtok(s, sep);
while (p)
      {i++;
       cout<<"Unitatea lexicala nr. "<<i<<" este ";
       cout<<p<<endl;
       p=strtok(NULL, sep);}
```

În urma execuției acestei secvențe de instrucțiuni pentru șirurile declarate anterior, vom obține pe ecran:

```
Unitatea lexicala nr. 1 este Vai
Unitatea lexicala nr. 2 este vai
Unitatea lexicala nr. 3 este Ce
Unitatea lexicala nr. 4 este necaz
Unitatea lexicala nr. 5 este Unde
Unitatea lexicala nr. 6 este e
Unitatea lexicala nr. 7 este pisica
```

Observație

De la un apel la altul, șirul de separatori se poate modifica. De exemplu, să considerăm că în șirul s este memorată o relație de forma:

```
<rezultat> = <operand_1> <operator> <operand_2>
```

unde operatorul poate fi `'+'`, `'-'`, `'*'` sau `'/'`.

Pentru a extrage cele trei unități lexicale (`rezultat`, `operand_1` și `operand_2`) putem proceda astfel:

```
p=strtok(s,"=");
cout<<"Rezultat:"<<p<<endl;
p=strtok(NULL,"+-*/ ");
cout<<"Operand 1:"<<p<<endl;
p=strtok(NULL,"+-*/ ");
cout<<"Operand 2:"<<p<<endl;
```


Funcții de conversie

În fișierul antet `stdlib.h` sunt declarate funcții care permit conversia unui număr întreg într-un șir de caractere:

```
char *itoa(int nr, char *sir, int baza);
char *ltoa(long nr, char *sir, int baza);
char *ultoa(unsigned long nr, char *sir, int baza);
```

Funcția `itoa()` convertește un număr de tip `int`, funcția `ltoa()` convertește un număr de tip `long`, iar funcția `ultoa()` convertește un număr de tip `unsigned long`. Șirul de caractere `sir` va conține reprezentarea numărului `nr` în baza de numerație `baza` (`2≤baza≤36`).

Exercițiu

În fișierul antet `stdlib.h` sunt declarate funcții care realizează și conversia în sens invers (din șir de caractere în valoare numerică) – `atoi()`, `atol()`, `atof()`, `strtod()`, `_strtold()`, `strtol()`, `strtoul()`. Studiați aceste funcții utilizând sistemul de autodocumentare (*Help*).

4.9. *Tipul de date* `struct`

Tablourile sunt structuri de date care conțin date de *același tip*. Totuși, apare frecvent necesitatea de a utiliza structuri de date care să conțină date *de tipuri diferite*, reunite sub un nume comun. O astfel de structură este cunoscută sub denumirea de *articol* (sau *înregistrare* sau *structură*) și este implementată în limbajul C/C++ cu ajutorul tipului de date `struct`. Datele înglobate într-un articol sunt denumite *câmpuri* sau *membri*.

Deoarece câmpurile nu mai sunt obligatoriu de același tip, ele nu mai pot fi referite prin intermediul indicilor. Din acest motiv, fiecare câmp are un nume distinct, care este utilizat pentru a referi câmpul respectiv.

Declararea unui tip `struct`

Declararea unui tip `struct` are următoarea sintaxă:

```
struct [nume_tip]
{lista_declaratii_campuri} [lista_variabile];
```

Efect

Am declarat un tip `struct` cu numele `nume_tip` care conține câmpurile declarate în `lista_declaratii_campuri` și o listă de variabile având tipul `struct` definit.

Observații

1. Lista de variabile este opțională. Nu este obligatoriu ca atunci când definim tipul să declarăm și variabile de tipul respectiv. Acest lucru se poate realiza și ulterior, astfel:

```
struct nume_tip lista_variabile;
```

sau (valabil numai în C++, nu şi în C):

```
nume_tip lista_variabile;
```

2. Numele tipului `struct` este opţional, deci puteţi declara şi tipuri anonime. Dar în acest caz este obligatoriu ca lista de variabile să apară (tipul fiind anonim, nu îl mai putem utiliza în declaraţii de variabile ulterioare).

O declaraţie de câmpuri are sintaxa unei declaraţii de variabile uzuale:

```
tip nume_camp1, nume_camp2, ..., nume_campk;
```

Exemple

1. Să declarăm un tip `struct` denumit `Punct` care să reprezinte un punct în planul cartezian:

```
struct Punct
       {double x, y;};
```

Un articol de tip `Punct` conţine două câmpuri de tip `double`: `x` (abscisa punctului) şi `y` (ordonata punctului). În acest caz, câmpurile sunt de acelaşi tip. Nu am declarat şi variabile de tip `Punct`. Acest lucru se poate realiza ulterior. De exemplu, declarăm `P1`, şi `P2` două variabile de tip `Punct`:

```
struct Punct P1, P2;
```

sau (valabil numai în C++, nu şi în C):

```
Punct P1, P2;
```

2. Să declarăm o structură care să conţină informaţii despre un elev: numele, mediile la cele 14 discipline de învăţământ, media generală, adresa şi numărul de telefon.

```
struct Elev
      {char nume[30], adr[60];
       float medii[14], mg;
       unsigned long tel; } Clasa[30];
```

Observaţi că la definirea tipului `Elev` am declarat şi o variabilă (`Clasa`) care este un vector cu 30 de componente de tip `Elev`.

3. Să declarăm o structură care să reţină date calendaristice:

```
struct Data
       {int zi, an;
        char luna[15];};
```

Referirea la un câmp al unei structuri

Referirea la un câmp al unui articol se realizează specificând numele structurii şi numele câmpului respectiv, separate prin *operatorul de selecţie directă* `'.'`:

```
variabila_structura.camp
```


Exemple

1. Pentru a ne referi la abscisa punctului `P1`, vom scrie `P1.x`.
2. Pentru a ne referi la numele primului elev memorat în vectorul `Clasa`, vom scrie `Clasa[0].nume`.

Iniţializarea unei structuri

Ca oricare alte variabile, la declararea unei variabile de tip `struct` aceasta poate fi iniţializată. La iniţializare se precizează, în ordine, valorile ce vor fi atribuite câmpurilor structurii. Valorile sunt separate prin virgulă şi încadrate între paranteze acolade. Valorile sunt atribuite în ordine câmpurilor structurii şi trebuie să corespundă ca tip cu acestea. Numărul de valori precizate la iniţializare poate fi egal cu numărul de câmpuri ale structurii sau mai mic (caz în care numai primele câmpuri vor fi iniţializate).

De exemplu, să declarăm o variabilă de tipul `struct Data` definit anterior şi să o iniţializăm cu 31-12-2004.

```
struct Data azi={31, 2004,"Decembrie"};
```

Citirea şi afişarea structurilor

Pentru a citi/afişa o structură, trebuie să citim/afişăm separat fiecare câmp al structurii.

Operaţii cu structuri

Singura operaţie care se poate realiza la nivel de structură este atribuirea. Mai exact, unei variabile de tip `struct` i se poate atribui o altă variabilă de acelaşi tip `struct`. De exemplu, putem declara o variabilă de tip `struct Data` denumită `ieri` şi să o iniţializăm cu valoarea variabilei `azi`:

```
struct Data ieri;
ieri=azi;
```

4.10. *Asocierea unui nume pentru un tip de date*

Tipurile de date predefinite ale limbajului (de exemplu, `int`, `char`, `float` etc.) se identifică printr-un nume. În secţiunea precedentă, am văzut şi că atunci când definim un tip `struct` îi putem asocia un nume.

Programatorul poate asocia unui tip de date un nume, indiferent dacă acesta este un tip predefinit sau definit de el, utilizând construcţia `typedef`:

```
typedef descriere_tip nume_tip;
```

Exemple

1. Să asociem tipului predefinit `float` numele (mai sugestiv pentru noi) `REAL`:

```
typedef float REAL;
```

2. Să asociem tipului `int *` (pointer cu tipul de bază `int`) numele `PInt`:

```
typedef int * PInt;
```

3. Să asociem numele `Polinom` unui tip articol care să conțină gradul polinomului și coeficienții:

```
typedef struct
        {int grad;
         double coef[100];} Polinom;
```

4. Să asociem un nume tipului tablou cu `100` de componente de tip `int`:

```
typedef int Vector[100];
```

4.11. *Aplicații*

Temperaturi

Se citesc de la tastatură temperaturile înregistrate la Institutul meteorologic în fiecare dintre zilele lunii ianuarie. Scrieți un program care să numere în câte zile ale lunii au fost temperaturi strict pozitive și de câte ori s-a schimbat semnul temperaturilor (s-a înregistrat o trecere de la temperaturi pozitive la temperaturi negative sau invers) în decursul lunii ianuarie.

Soluție

Vom memora temperaturile din luna ianuarie într-un vector `T` cu `31` de elemente. Vom parcurge vectorul de la primul până la ultimul element, testând pentru fiecare element din vector dacă este o temperatură pozitivă. În caz afirmativ, numărăm încă o zi cu temperaturi pozitive (în acest scop utilizăm variabila `nrp`).

Pentru a determina numărul de schimbări de semn din vectorul `T` este suficient ca în timpul parcurgerii să verificăm dacă există în vector două elemente situate pe poziții consecutive și de semne contrare. Schimbările de semn vor fi numărate în variabila `nrs`.

```
#include <iostream.h>
#define NMax 31
int main()
{int T[NMax], nrp, nrs=0, i;
 cout<<"Introduceti temperaturile: ";
 for (i=0; i<31; i++) cin>>T[i];
 nrp=T[0]>0?1:0;
 for (i=1; i<31; i++)
     {if (T[i]>0) nrp++;
      if (T[i-1]*T[i]<0) nrs++;     }
 cout<<"nr de zile cu temperaturi pozitive "<<nrp<<endl;
 cout<<"nr de schimbari de semn "<<nrs<<endl;
 return 0; }
```


Inserare medii

Se citește de la tastatură un număr natural `n` (`1<n≤50`) și elementele unui vector cu `n` componente reale. Să se insereze între oricare două elemente consecutive ale vectorului media lor aritmetică.

Soluție

O primă idee este de a citi elementele vectorului și apoi de a insera între oricare două elemente vecine media lor. Pentru a insera media, trebuie să-i „facem loc", adică trebuie să deplasăm toate elementele care urmează cu o poziție la dreapta. Nu uitați că, prin inserare, numărul de elemente din vector crește!

```
#include <iostream.h>
int main ()
{float a[100];
 int n, i, j;
 cout <<"n="; cin >> n;
 for (i=0; i<n; i++) {cout <<"a["<<i<<"]="; cin>>a[i];}
 for (i=1; i<n; i+=2) //inserez media dintre a[i] si a[i-1]
   {//mut elementele de la i la n-1 cu o pozitie la dreapta
    for (j=n; j>i; j--) a[j]=a[j-1];
    a[i]=(a[i]+a[i-1])/2;       //plasez media pe pozitia i
    n++; }          //maresc numarul de elemente din vector
 for (i=0; i<n; i++) cout <<a[i]<<' ';
 cout<<endl; return 0;}
```

O soluție mai simplă ar fi fost de a plasa în vector elementele citite numai pe poziții de indice par, astfel încât pozițiile de indice impar să rămână disponibile pentru medii, evitând astfel deplasările succesive:

```
#include <iostream.h>
int main ()
{
 float a[100];
 int n, i;
 cout <<"n="; cin >> n;
 for (i=0; i<n; i++)
     {cout <<"a["<<i<<"]=";
      cin>>a[2*i]; }
 for (i=1; i<n; i++)
     a[2*i-1]=(a[2*i]+a[2*i-2])/2;
 for (i=0; i<2*n-1; i++) cout <<a[i]<<' ';
 cout<<endl;  return 0; }
```

Combinare vectori

Fie `n` un număr natural, `n≤20`, și `A`, `B` doi vectori cu câte `n` componente întregi. Scrieți un program care să combine cei doi vectori conform relațiilor următoare:

`A[i]=A[i]+B[i]`
`B[i]=A[i]*B[i]`, pentru orice `i=0, 1, 2, ..., n-1`.

Soluție

O primă idee este de a parcurge vectorii simultan și de a actualiza valorile elementelor vectorilor conform relațiilor de mai sus:

```
#include <iostream.h>
int main()
{int n, A[20], B[20], i;
 cout<<"n= "; cin>>n;
 cout<<"Elementele vectorului A: ";
 for (i=0; i<n; i++) cin >> A[i];
 cout<<"Elementele vectorului B: ";
 for (i=0; i<n; i++) cin >> B[i];
 for (i=0; i<n; i++)
     {A[i]=A[i]+B[i];
      B[i]=A[i]*B[i];  }
 cout<<"Vectorul A: ";
 for (i=0; i<n; i++) cout<<A[i]<<' '; cout<<endl;
 cout<<"Vectorul B: ";
 for (i=0; i<n; i++) cout<<B[i]<<' '; cout<<endl;
 return 0; }
```

Executând acest program, observăm că rezultatele sunt *greșite*. Să urmărim execuția acestui program, pas cu pas, pentru `n=2`, `A={1, 2}`, `B={3, 4}`.

i	A	B	Explicație
0	{4, 2}	{12, 4}	Se calculează `A[0]=A[0]+B[0]=1+3=4`. Apoi se calculează `B[0]=A[0]*B[0]=4*3=12`. Observăm că valoarea lui `A[0]` a fost calculată corect, dar valoarea lui `B[0]` este greșită, pentru că în calcul se utilizează noua valoare a lui `A[0]`.
1	{4, 6}	{12, 24}	Se calculează `A[1]=A[1]+B[1]=2+6=6`. Apoi se calculează `B[1]=A[1]*B[1]=6*4=24`. Din nou, valoarea lui `A[1]` a fost calculată corect, dar valoarea lui `B[1]` este greșită.

Pentru a calcula corect și valorile elementelor din vectorul `B`, salvăm valoarea inițială a elementului `A[i]` într-o variabilă auxiliară.

Programul corect este:

```
#include <iostream.h>
int main()
{int n, A[20], B[20], i, aux;
 cout<<"n= "; cin>>n;
```



```
 cout<<"Elementele vectorului A: ";
 for (i=0; i<n; i++) cin >> A[i];
 cout<<"Elementele vectorului B: ";
 for (i=0; i<n; i++) cin >> B[i];
 for (i=0; i<n; i++)
     {aux=A[i];
      A[i]=A[i]+B[i];
      B[i]=aux*B[i];     }
 cout<<"Vectorul A: ";
 for (i=0; i<n; i++) cout<<A[i]<<' '; cout<<endl;
 cout<<"Vectorul B: ";
 for (i=0; i<n; i++) cout<<B[i]<<' '; cout<<endl;
 return 0; }
```

Ciurul lui Eratostene

Fie `n` un număr natural (`n≤10000`). Să se genereze toate numerele prime mai mici decât `n`.

Soluție

O primă idee ar fi să parcurgem toate numerele naturale din intervalul [2, n], pentru fiecare număr să verificăm dacă este prim și să afișăm numerele prime determinate.

O idee mai eficientă provine din Antichitate și poartă numele de ciurul lui Eratostene[25]. Ideea este de a „pune în ciur" toate numerele mai mici decât `n`, apoi de a „cerne" aceste numere până rămân în ciur numai numerele prime. Mai întâi „cernem" (eliminăm din ciur) toți multiplii lui 2, apoi cernem multiplii lui 3 ș.a.m.d.

Vom reprezenta ciurul ca un vector cu `10000` de componente care pot fi `0` sau `1`, cu semnificația `ciur[i]=1`, dacă numărul `i` este în ciur, și `0`, altfel.

Vom parcurge vectorul ciur de la `2` (cel mai mic număr prim), până la $\sqrt{n}$.

Dacă `ciur[i]` este `1`, deducem că `i` este prim, dar toți multiplii lui nu vor fi (deci eliminăm din ciur toți multiplii lui `i`).

În final, în vectorul `ciur` vor avea valoarea `1` doar componentele de pe poziții numere prime.

```
#include <iostream.h>
#define NMax 10000
int main()
{int ciur[NMax], n, i, j;
```

25. Eratostene (276-196 î.e.n.) a fost un important matematician și filosof al Antichității. Deși puține dintre lucrările lui s-au păstrat, a rămas celebru prin metoda rapidă de determinare a numerelor prime și prin faptul că a fost primul care a estimat cu acuratețe diametrul Pământului.

```
cout<<"n= "; cin>>n;
//initial toate numerele sunt in ciur
for (i=2; i<n; i++) ciur[i]=1;
for (i=2; i*i<=n; i++)
    if (ciur[i])//i este prim
       //elimin toti multiplii lui i
       for (j=2; j*i<n; j++)
           ciur[i*j]=0;
for (i=2; i<n; i++)
    if (ciur[i]) cout<<i<<' ';
return 0; }
```

Copii

La ora de educație fizică, profesorul a cerut elevilor să se alinieze. Fiind un profesor pasionat de informatică, a observat imediat că există situații în care un copil de înălțime maximă este așezat lângă un copil de înălțime minimă.

Scrieți un program care să determine câte perechi de elevi alăturați îndeplinesc această condiție.

Soluție

Vom reține înălțimile copiilor, în ordinea în care s-au aliniat aceștia, în vectorul `h`. Determinăm mai întâi înălțimea celui mai înalt copil și înălțimea celui mai scund copil. Parcurgem apoi vectorul, identificând toate perechile de copii alăturați în care primul copil are înălțimea maximă și al doilea, înălțimea minimă sau invers.

```
#include <iostream.h>
#define NMax 40
int main()
{
 int h[NMax], n, i, nr, min, max;
 //citire
 cout<<"n= "; cin>>n;
 for (i=0; i<n; i++) cin>>h[i];
 //determinam inaltimea maxima si inaltimea minima
 max=min=h[0];
 for (i=1; i<n; i++)
     if (max<h[i]) max=h[i];
        else
        if (min>h[i]) min=h[i];
 //determinam numarul de perechi
 nr=0;
 for (i=1; i<n; i++)
     if (h[i]==max && h[i-1]==min ||
         h[i]==min && h[i-1]==max)
         nr++;
 cout<<nr<<endl;
 return 0; }
```


Exercițiu

Modificați programul precedent astfel încât să realinieze elevii pentru a obține un număr maxim de perechi de elevi alăturați cu proprietatea din enunț.

Perechi

Construiți un algoritm eficient care să determine toate perechile de numere naturale `a`, `b`, cu `a≤b`, având proprietatea că nu au nici o cifră comună și suma lor este egală cu `S`. Valoarea `S` este un număr natural citit de la tastatură (`S<100000000`). Fiecare pereche se va scrie pe un rând al ecranului, cu un spațiu între elementele ce compun perechea.

De exemplu, pentru `S=14`, se vor afișa (nu neapărat în această ordine) perechile:

```
3 11
6 8
4 10
5 9
0 14
```

(variantă Bacalaureat, august 2001)

Soluție

O primă idee este de a genera toate perechile de numere mai mici sau egale cu `S` și pentru fiecare pereche să verificăm proprietatea cerută.

Pentru a testa dacă numerele `a` și `b` au cifre distincte, vom determina mulțimea cifrelor numărului a, apoi vom determina mulțimea cifrelor numărului b, apoi vom verifica dacă cele două mulțimi au elemente comune.

Pentru a determina mulțimea cifrelor numărului `a` vom considera un vector `cifrea` cu `10` componente (câte una pentru fiecare cifră din baza `10`), având semnificația `cifrea[i]=1`, dacă cifra `i` apare în numărul `a`, și `0`, în caz contrar. Inițial, toate componentele vectorului au valoarea `0`. Vom extrage succesiv cifrele numărului `a`, reținând pentru fiecare cifră extrasă valoarea `1` pe poziția corespunzătoare cifrei din vectorul `cifrea`. În mod analog procedăm și pentru numărul `b`, reținând mulțimea cifrelor în vectorul `cifreb`. După ce am construit cele două mulțimi de cifre, le vom parcurge simultan verificând că nu există nici o cifră care să apară atât în `a` cât și în `b` (adică să nu existe nici o poziție `k` astfel încât `cifrea[k]` și `cifreb[k]` să fie simultan `1`).

```
#include <iostream.h>
int main()
{long int s, a, b, aux;
 int k, cifrea[10], cifreb[10];
 cout<<"s= "; cin>>s;
 for (a=0; a<s; a++)
      for (b=a+1; b<=s; b++)
          if (a+b==s)
```

```
            {//determinam multimea cifrelor lui a
            for (k=0; k<10; k++) cifrea[k]=0;
            aux=a;
            do {cifrea[aux%10]=1; aux/=10;} while (aux);
             //determinam multimea cifrelor lui b
            for (k=0; k<10; k++) cifreb[k]=0;
            aux=b;
            do {cifreb[aux%10]=1; aux/=10;} while (aux);
             //verificam daca cifrele sunt distincte
            for (k=0;k<10 && !(cifrea[k]&&cifreb[k]);k++);
            if (k==10) cout<<a<<' '<<b<<endl;
            }
 return 0;}
```

Algoritmul de mai sus execută `s*(s-1)/2` operații, operațiile fiind relativ complexe (determinare de mulțimi de cifre și verificare). Valoarea maximă a variabilei `s` fiind foarte mare, timpul de execuție a acestui program va fi extrem de mare.

Algoritmul precedent poate fi transformat cu ușurință într-un algoritm liniar (care execută `s/2` operații), dacă observăm că valoarea lui `b` este unic determinată de valoarea lui `a` (`b=s-a`).

```
#include <iostream.h>
int main()
{long int s, a, b, aux;
 int k, cifrea[10], cifreb[10];
 cout<<"s= "; cin>>s;
 for (a=0; a<s/2; a++)
     {b=s-a;
      //determinam multimea cifrelor lui a
      for (k=0; k<10; k++) cifrea[k]=0;
     aux=a;
      do {cifrea[aux%10]=1; aux/=10;} while (aux);
      //determinam multimea cifrelor lui b
      for (k=0; k<10; k++) cifreb[k]=0;
      aux=b;
      do {cifreb[aux%10]=1; aux/=10;} while (aux);
      //verificam daca cifrele sunt distincte
      for (k=0; k<10 && !(cifrea[k] && cifreb[k]); k++);
      if (k==10) cout<<a<<' '<<b<<endl;
      }
 return 0;}
```

Observați că `a` variază numai până la `s/2` pentru a nu genera de două ori aceeași pereche (de exemplu, și perechea `0 14`, și perechea `14 0`).

Repetiție

Se citește de la tastatură o succesiune de litere mici terminată cu caracterul punct. Să se afișeze pe ecran literele care se repetă.

Soluție

Vom citi succesiv caractere de la tastatură până la întâlnirea caracterului punct. Caracterele citite vor fi memorate în vectorul `s`, iar în variabila `n` vom reține numărul lor.

O primă idee ar fi să parcurgem vectorul `s` și pentru fiecare caracter din vector să verificăm dacă mai apare sau nu în vector pe o altă poziție și, dacă da, să îl afișăm.

```
#include <iostream.h>
#define NMax 100
int main()
{char s[NMax], c;
 int n, i, j, serepeta=1;
 n=0; //numarul de litere din secventa
 do
  {cin>>c;
   if (c!='.') s[n++]=c;    }
 while (c!='.');
 for (i=0; i<n; i++)
     {serepeta=0;
      for (j=0; j<i && !serepeta; j++)
          if (s[i]==s[j]) {cout<<s[i]; serepeta=1;}
      }
 cout<<endl; return 0; }
```

Acest program este *greșit* deoarece, în cazul în care există litere care se repetă de mai mult de două ori, ele vor fi afișate pe ecran de mai multe ori. Deci este obligatoriu să știm dacă o literă a mai fost sau nu afișată.

Ar trebui să utilizăm un vector `uz` cu 26 de componente (câte una pentru fiecare literă mică din alfabet), cu semnificația `uz[i]=1`, dacă a `i`-a literă din alfabet a mai fost afișată, și 0, în caz contrar.

Ideea vectorului `uz` ar putea fi exploatată mai eficient. Am putea chiar de la citirea caracterelor să numărăm în `uz` de câte ori apare fiecare literă din alfabet în șir. Apoi să parcurgem vectorul `uz` și să afișăm numai literele pentru care numărul de apariții `>1`. Cu această idee am transformat un program greșit și ineficient (fiindcă executa aproximativ `n*(n-1)` operații) într-un program corect și eficient (care execută `n+26` operații).

```
#include <iostream.h>
int uz[26];
int main()
{char c;
 int i;
 do
  {cin>>c;
   if (c!='.') //numar o aparitie a literei c
      uz[c-'a']++;    }
 while (c!='.');
```

```
for (i=0; i<26; i++)
     if (uz[i]>1) cout<<(char)(i+'a');
cout<<endl; return 0; }
```

Eliminare

Se citeşte de la tastatură o secvenţă formată din maxim 100 de litere mici, până la întâlnirea caracterului punct. Dacă există în secvenţa citită litere alăturate egale, aceste litere vor fi eliminate. Dacă în urma eliminării se obţin din nou litere alăturate egale, se elimină şi acestea, eliminările efectuându-se până când în secvenţă nu există litere alăturate egale.

De exemplu, să presupunem că am citit secvenţa `aacbooobcaxa`. La o primă parcurgere a secvenţei, identificăm subsecvenţa `aa` şi subsecvenţa `ooo` formată din litere consecutive egale. După eliminarea acestora, secvenţa devine `cbbcaxa`. S-a format o nouă secvenţă de litere consecutive egale, `bb`. După eliminarea acestei secvenţe, obţinem `ccaxa`. S-a format o nouă secvenţă de litere consecutive egale, `cc`, pe care o eliminăm şi obţinem `axa`. În acest moment procesul de eliminare se opreşte, deoarece nu mai există litere consecutive egale.

Soluţie

Vom citi succesiv caractere de la tastatură până la întâlnirea caracterului punct. Caracterele citite vor fi memorate în vectorul `s`, iar în variabila `n` vom reţine numărul lor. Vom parcurge vectorul `s` pentru a identifica o pereche de caractere egale situate pe poziţii consecutive. Dacă o astfel de pereche a fost identificată, reţinem poziţia de început a unei secvenţe ce urmează a fi eliminată şi determinăm sfârşitul secvenţei. Eliminăm apoi secvenţa identificată, prin deplasarea la stânga a tuturor elementelor din vectorul `s` situate după poziţia de sfârşit a secvenţei care se elimină, urmată de micşorarea numărului de elemente din vector cu lungimea secvenţei eliminate. Procedeul de eliminare se repetă cât timp în vectorul `s` se produc schimbări.

```
#include <iostream.h>
#define NMax 100
int main()
{char s[NMax], c;
 int n, i, j, k, schimb=1;
 n=0; //numarul de litere din secventa
 do
  {cin>>c;
   if (c!='.') s[n++]=c;        }
 while (c!='.');
 while (schimb)
      {for (schimb=i=0; i<n-1; )
          if (s[i]==s[i+1])
             //incepe o secventa de litere egale
```



```
             {//determin sfarsitul acestei secvente
              for (j=i+1;j<n && s[j]==s[i]; j++);
              //o elimin
              for (k=j; k<n; k++) s[i+k-j]=s[k];
              //actualizez numarul de elemente din secventa
              n-=j-i;
              schimb=1;          }
            else i++; //trec mai departe
      }
for (i=0; i<n; i++) cout<<s[i];
cout<<endl; return 0; }
```

Permutare ciclică

Fie `a` un vector cu `n` componente întregi şi `p` un număr natural. Numim permutare ciclică dreapta cu `p` paşi operaţia prin care toate elementele vectorului sunt deplasate cu `p` poziţii spre dreapta, ultimele `p` elemente din vector fiind „împinse" circular pe primele `p` poziţii.

De exemplu, să considerăm că vectorul `a` conţine numerele naturale de la 0 la 7. După o deplasare ciclică dreapta cu 3 poziţii, elementelor vectorului `a` vor fi 5, 6, 7, 0, 1, 2, 3, 4.

Scrieţi un program care să execute o permutare ciclică dreapta cu `p` paşi a vectorului `a`.

Soluţie

O primă idee ar fi să executăm `p` permutări ciclice cu o poziţie spre dreapta.

```
#include <iostream.h>
int main()
{int a[100], n, p, i, aux, j;
 cout<<"n, p= "; cin>>n>>p;
 cout<<"elementele vectorului: ";
 for (i=0; i<n; i++) cin>>a[i];
 for (i=1; i<=p; i++)
     {aux=a[n-1];
      for (j=n-1; j>0; j--) a[j]=a[j-1];
      a[0]=aux;      }
 for (i=0; i<n; i++) cout<<a[i]<<' '; cout<<endl;
 return 0; }
```

Acest algoritm efectuează `p` parcurgeri ale vectorului, deci `p*n` operaţii. O metodă rapidă de permutare ciclică dreapta cu `p` paşi este:

- inversăm elementele vectorul din segmentul `0..n-p-1`;
- inversăm elementele vectorul din segmentul `n-p..n-1`;
- inversăm întregul vector.

```
#include <iostream.h>
int main()
{int a[100], n, p, i, aux, j;
 cout<<"n, p= "; cin>>n>>p;
 cout<<"elementele vectorului: ";
 for (i=0; i<n; i++) cin>>a[i];
 //inversam elementele vectorului de la 0 la n-p-1
 for (i=0, j=n-p-1; i<j; i++,j--)
     { aux=a[i]; a[i]=a[j]; a[j]=aux;}
 //inversam elementele vectorului de la n-p la n-1
 for (i=n-p, j=n-1; i<j; i++,j--)
     { aux=a[i];a[i]=a[j]; a[j]=aux;}
 //inversam intreg vectorul
 for (i=0, j=n-1; i<j; i++,j--)
     { aux=a[i]; a[i]=a[j]; a[j]=aux;}
 for (i=0; i<n; i++) cout<<a[i]<<' '; cout<<endl;
 return 0; }
```

Exerciţiu

Modificaţi programul precedent astfel încât să execute o permutare ciclică stânga cu p paşi. Numim permutare ciclică stânga cu p paşi operaţia prin care toate elementele vectorului sunt deplasate cu p poziţii spre stânga, primele p elemente din vector fiind „împinse" circular pe ultimele p poziţii.

Generarea mulţimii

Să considerăm mulţimea A constituită numai din elemente obţinute după următoarele reguli:

1. 1∈A.
2. Dacă x∈A, atunci 2*x+1∈A.
3. Dacă x∈A, atunci 3*x+1∈A.

Scrieţi un program care să genereze cele mai mici n elemente ale mulţimii A.

Soluţie

Vom genera în ordine crescătoare elementele mulţimii într-un vector A. Vom iniţializa primul element al vectorului A cu 1. Plecând de la 1 putem genera elementele 2*1+1=3 şi 3*1+1=4.

Deci, după primul pas, elementele vectorului A sunt {1, 3, 4}. Am marcat îngroşat ultimele elemente generate.

Aplicând regulile 2 şi 3 pentru ultimele elemente generate, obţinem elementele 2*3+1, 3*3+1, 2*4+1 şi 3*4+1. Prin urmare, componenţa vectorului A la pasul 2 devine {1, 3, 4, 7, 10, 9, 13}.

La pasul următor aplicăm regulile 2 şi 3 ultimelor elemente generate (cele marcate îngroşat) şi obţinem elementele 2*7+1, 3*7+1, 2*10+1, 3*10+1, 2*9+1, 3*9+1, 2*13+1, 3*13+1. Vectorul A conţine {1, 3, 4, 7, 10, 9, 13, **15, 22, 21, 31, 19, 28, 27, 40**}.

La pasul x aplicăm regulile 2 şi 3 ultimelor elemente generate la pasul x-1. Prin urmare, la pasul x se obţin 2^x elemente noi. Procedeul se repetă până când în vector obţinem n elemente. Problema este că, generând elementele în acest mod, este posibil să obţinem elemente care se repetă, prin urmare, ar trebui ca la fiecare element nou generat să verificăm dacă nu cumva există deja în vector. În al doilea rând, elementele generate nu sunt în ordine crescătoare. Deci la sfârşit trebuie să sortăm vectorul şi apoi să îl afişăm.

Algoritmul prezentat mai sus este relativ dificil de implementat şi, în plus, este şi neeficient (execută aproximativ n^2 operaţii).

O idee mai bună ar fi să generăm de la început elementele vectorului în ordine crescătoare. Pentru aceasta, la pasul x nu vom mai plasa în vector 2^x elemente, ci vom plasa un singur element (cel mai mic dintre cele care urmează să fie generate). Pentru a determina cel mai mic element dintre cele care urmează a fi generate nu este necesar să le determinăm pe toate, apoi să calculăm minimul. Este suficient să reţinem permanent două elemente: următorul element generat pe baza regulii 2 şi următorul element generat pe baza regulii 3. Minimul va fi selectat dintre aceste două elemente. Pentru aceasta vor fi necesari doi indici (poz2 şi poz3) cu semnificaţia poz2 = poziţia următorului element pentru care se aplică regula 2, iar poz3 = poziţia următorului element pentru care se aplică regula 3.

```
#include <iostream.h>
int main()
{int n, poz2, poz3, A[1000], i, min;
 cout<<"n= "; cin>>n;
 A[0]=1; poz2=poz3=0;
 for (i=1; i<n; i++)
     {min=2*A[poz2]+1;
      if (min>3*A[poz3]+1) min=3*A[poz3]+1;
      A[i]=min;
      if (min==2*A[poz2]+1) poz2++;
      if (min==3*A[poz3]+1) poz3++;     }
for (i=0; i<n; i++) cout<<A[i]<<' '; cout<<endl;
return 0; }
```

Să urmărim execuţia acestui algoritm pentru n=7:

Pas	poz2	poz3	Vectorul A	Explicaţie
0	0	0	{1}	Iniţializăm vectorul A cu 1.
1	1	0	{1, 3}	Determinăm minimul dintre 2*A[poz2]+1=3 şi 3*A[poz3]+1=4. Minimul este 3, îl plasăm în vectorul A şi incrementăm poz2.

2	1	1	{1, 3, 4}	Determinăm minimul dintre 2*A[poz2]+1=7 şi 3*A[poz3]+1=4. Minimul este 4, îl plasăm în vectorul A şi incrementăm poz3.
3	2	1	{1, 3, 4, 7}	Determinăm minimul dintre 2*A[poz2]+1=7 şi 3*A[poz3]+1=9. Minimul este 7, îl plasăm în vectorul A şi incrementăm poz2.
4	3	1	{1, 3, 4, 7, 9}	Determinăm minimul dintre 2*A[poz2]+1=9 şi 3*A[poz3]+1=10. Minimul este 9, îl plasăm în vectorul A şi incrementăm poz2.
5	3	2	{1, 3, 4, 7, 9, 10}	Determinăm minimul dintre 2*A[poz2]+1=13 şi 3*A[poz3]+1=10. Minimul este 10, îl plasăm în vectorul A şi incrementăm poz3.
6	4	2	{1, 3, 4, 7, 9, 10, 13}	Determinăm minimul dintre 2*A[poz2]+1=13 şi 3*A[poz3]+1=25. Minimul este 13, îl plasăm în vectorul A şi incrementăm poz2.

Pentru n=7, nu am detectat cazul în care 2*A[poz2]+1=3*A[poz3]+1. În acest caz, elementul se plasează în vector o singură dată, dar se incrementează atât poz2, cât şi poz3.

Schema lui Horner

Fie $P(X)=p_0X^n+p_1X^{n-1}+...+p_{n-1}X+p_n$ un polinom de gradul n ($n \le 100$) cu coeficienţi reali. Scrieţi un program care să calculeze câtul şi restul împărţirii polinomului P(X) la (X-a), unde a este un număr real dat.

Soluţie

Vom reţine coeficienţii polinomului într-un vector, în ordinea descrescătoare a gradelor. Vom obţine coeficienţii câtului şi restul în acelaşi vector, parcurgând[26] vectorul începând cu poziţia 1 (primul coeficient rămâne neschimbat) şi adăugând la fiecare coeficient precedentul său înmulţit cu a.

De exemplu, că considerăm polinomul de gradul 3 $P(X)=X^3-3X^2+1$ şi a=2. Să urmărim pas cu pas execuţia acestui algoritm:

Pas	Vectorul P	Explicaţie
0	1, -3, 0, 1	Iniţializăm vectorul cu coeficienţii polinomului în ordinea descrescătoare a gradelor.
1	1, -1, 0, 1	P[1]=P[0]*a+P[1]=1*2-3=-1
2	1, -1, -2, 1	P[2]=P[1]*a+P[2]=-1*2+0=-2
3	1, -1, -2, -3	P[3]=P[2]*a+P[3]=-2*2+1=-3

26. Metoda este cunoscută de la matematică.

Observăm că P[3] conţine restul împărţirii polinomului la polinomul (X-2), iar (1, -1, -2) sunt coeficienţii câtului.

```
#include <iostream.h>
int main ()
{ float P[100], a;
  int n, i;
  cout <<"n="; cin >> n;                              //citire
  for (i=0; i<=n; i++) {cout <<"P"<<i<<"="; cin>>P[i];}
  cout <<"a="; cin >> a;
  for (i=1; i<=n; i++)          //aplic schema lui Horner
      P[i]=P[i-1]*a+P[i];
  cout <<"Coeficientii catului sunt ";          //afisare
  for (i=0; i<n; i++)  cout <<P[i]<<" ";
  cout <<"\nRestul este "<<P[n]<<endl;
  return 0;}
```

Subsecvenţă de sumă maximă

Fie $x=(x_0, x_1, ..., x_{n-1})$ o secvenţă de numere întregi, dintre care cel puţin un element este pozitiv. Să se determine o subsecvenţă de elemente consecutive $x_i, x_{i+1}, ..., x_j$, astfel încât suma elementelor din subsecvenţă să fie maximă. De exemplu, pentru x=(1, 10, 3, -5, 4, 2, -100, 30, 15, 25, -10, 40, 1000), subsecvenţa de sumă maximă începe la poziţia 7, are lungimea 6, iar suma elementelor este 1100.

Soluţia 1

Considerăm toate subsecvenţele de elemente consecutive, calculăm suma elementelor din subsecvenţă şi reţinem poziţia de început şi lungimea subsecvenţei de sumă maximă.

```
#include <iostream.h>
int main ()
{int x[50], n, i, st, dr, poz, Lg, Sum, SMax;
 cout <<"n="; cin >> n;
 for (i=0; i<n; i++) {cout <<"x["<<i<<"]="; cin>>x[i];}
 for (SMax=x[0], st=0; st<n; st++)
     for (dr=st; dr<n; dr++)
         {for (Sum=0,i=st; i<=dr; i++) Sum += x[i];
              if (SMax<Sum) SMax=Sum, Lg=dr-st+1, poz=st; }
cout <<"Poz= "<<poz<<" ;Lg= "<<Lg<< " ; Smax= "<<SMax;
return 0;}
```

Datorită celor trei cicluri for imbricate, algoritmul realizează un număr de operaţii comparabil (ca ordin de mărime) cu n^3.

Soluția 2

Pentru a calcula suma elementelor din subsecvența de la `st` la `dr`, vom folosi suma deja calculată a elementelor de la `st` la `dr-1`. Secvența de instrucțiuni care rezolvă cerințele problemei devine:

```
for (SMax=x[0], st=0; st<n; st++)
     for (Sum=0,dr=st; dr<n; dr++)
          {Sum += x[dr];
           if (SMax < Sum) SMax=Sum, poz=st, Lg=dr-st+1;}
```

În acest caz, numărul de operații efectuate de algoritm este comparabil (ca ordin de mărime) cu n^2. Spunem că algoritmul este *pătratic*.

Soluția 3

O dată cu parcurgerea vectorului `x` de la stânga la dreapta, se va memora suma maximă calculată până la momentul curent în variabila `SMax`. În variabila `Sum` este reținută suma elementelor subsecvenței curente. Elementul `x[i]` va fi înglobat în subsecvența curentă dacă `Sum+x[i]≥0`. Dacă `Sum+x[i]<0`, înglobarea lui `x[i]` în orice subsecvență ar conduce la o subsecvență cu suma elementelor mai mică decât cea maximă până la momentul curent, prin urmare, `x[i]` trebuie ignorat, iar noua subsecvență curentă va începe la poziția următoare. Mai exact, la pasul `i`: `Sum=max(Sum, 0)+x[i]; SMax=max(SMax, Sum)`. Secvența de instrucțiuni care rezolvă cerințele problemei devine:

```
for (SMax=Sum=x[0],st=poz=0, Lg=i=1; i<n; i++)
     if (Sum < 0)     //noua subsecventa incepe pe pozitia i
         Sum = x[i], st=i;
         else         //inglobez x[i] in subsecventa curenta
         {Sum += x[i];
          if (SMax<Sum) //subsecventa curenta este maximala
             SMax=Sum, poz=st, Lg=i-st+1;}
```

În acest caz, numărul de operații efectuate de algoritm este comparabil (ca ordin de mărime) cu `n`. Spunem că algoritmul a devenit *liniar*.

Cărți

Vasile joacă un joc foarte interesant. El are un pachet de `N` (`1≤N≤20000`) cărți de joc (numerotate distinct de la `1` la `N`). Cărțile din pachet sunt amestecate.

Vasile se uită la fiecare carte din pachet începând cu prima, până ajunge la cartea cu numărul `1`, pe care o scoate din pachet. Apoi caută cartea cu numărul `2`, cartea cu numărul `3` ș.a.m.d. De fiecare dată începe căutarea de unde a rămas (de la cartea care urmează după ultima carte scoasă din pachet). De fiecare dată când ajunge la sfârșitul pachetului, Vasile bate din palme și reia căutarea de la începutul pachetului. Când ultima carte din pachet este eliminată, jocul se termină.

Scrieți un program care să determine de câte ori bate din palme Vasile în timpul unui joc.

De exemplu, pentru `N=3` și ordinea cărților `2, 1, 3` programul va afișa `1`. Pentru `N=7` și ordinea cărților `3, 6, 7, 1, 5, 4, 2` programul va afișa `3`. Programul va citi datele din fișierul `carti.in` și va afișa rezultatul în fișierul `carti.out`.

Soluție

O primă idee ar fi să reținem într-un vector cărțile din pachet în ordinea în care acestea apar în pachet și să simulăm operațiile pe care le face Vasile, numărând de câte ori luăm de la capăt parcurgerea pachetului de cărți.

Această idee este mult prea costisitoare ca timp (în cazul cel mai defavorabil, adică atunci când cărțile sunt sortate descrescător, numărul de operații este comparabil ca ordin de mărime cu n^2).

O altă idee ar fi ca pentru fiecare carte `x` (de la `1` la `N`) să reținem în vectorul `poz` poziția sa în pachetul de cărți. Numărul de bătăi din palme este egal cu numărul de inversiuni din vectorul `poz` (numărul de elemente pentru care `poz[i]>poz[i+1]`). În acest mod, soluția a devenit liniară (numărul de operații este comparabil ca ordin de mărime cu `n`), esențial pentru valori mari ale lui `n`.

```
#include <fstream.h>
#define MAX_N 20000
int main()
{ int i, n, x, poz[MAX_N], rez=0;
  ifstream fin("carti.in");
  fin>>n;
  for (i = 0; i<n; i++)
      {fin>>x; poz[x-1]=i;  }
  fin.close();
  for (i = 0; i<n-1; i++)
      if (poz[i] > poz[i+1]) rez++;
  ofstream fout("carti.out");
  fout<<rez<<endl;
  fout.close();
  return 0; }
```

Marcare

Numerele de la `1` la `N` sunt așezate în ordine crescătoare pe circumferința unui cerc, astfel încât `N` ajunge lângă `1`. Începând cu numărul `S` (`1≤S≤N`) se marchează numerele din `K` în `K`, în ordinea lor crescătoare, până când un număr este marcat de două ori (`1≤N, K≤1000`). Scrieți un program care să determine numerele marcate în ordinea marcării lor, precum și câte numere au rămas nemarcate.

De exemplu, pentru `N=8`, `S=2` și `K=5`, programul va afișa:

```
Numerele marcate in ordinea marcarii lor: 2 7 4 1 6 3 8 5
Numarul de valori nemarcate: 0
```

Soluție

Problema se rezolvă matematic: la pasul `i` se marchează numărul `x=(s+i*k)%n+n*((s+i*k)%n==0)`.

Vom memora toate valorile marcate într-o mulțime. Pentru a testa eficient dacă un număr aparține mulțimii de numere marcate, vom implementa mulțimea valorilor marcate prin vector caracteristic.

Fie `M` o mulțime de numere naturale ≤`N`. Vectorul caracteristic corespunzător mulțimii `M` este un vector cu `N+1` elemente având componentele `0` sau `1`: `M[i]=1`, dacă numărul `i` aparține mulțimii `M`, și `0`, în caz contrar.

```
#include <iostream.h>
#define NMax 1001
int M[NMax];
int main()
{int n, s, k, i;
 cout<<"n, s, k= "; cin>>n>>s>>k;
 cout<<"Numerele marcate in ordinea marcarii lor: ";
 for (i=0; M[(s+i*k)%n + n*((s+i*k)%n==0)]==0; i++)
      {//scriu elementul care urmeaza sa fie marcat
       cout<<(s+i*k)%n + n*((s+i*k)%n==0)<<' ';
       //il retin in multimea de elemente marcate
       M[(s+i*k)%n + n*((s+i*k)%n==0)]=1;}
 cout<<endl<<"Numarul de valori nemarcate: "<<n-i<<endl;
 return 0; }
```

Observație

Am declarat vectorul `M` global, pentru a fi automat inițializat cu `0` (ceea ce corespunde mulțimii vide).

Depozit

Considerăm un depozit cu `n` (n≤1000) camere, care conțin respectiv cantitățile de marfă C_1, C_2, ..., C_n (numere naturale). Scrieți un program care să determine un grup de camere cu proprietatea că suma cantităților de marfă pe care le conțin se poate împărți exact la cele `n` camioane care o transportă.

(Olimpiada Națională de Informatică, Iași, 1993)

Soluție

Problema este particulară: solicită determinarea unei submulțimi a unei mulțimi cu `n` elemente, divizibilă tot cu `n`. Vom construi sumele `S1`, `S2`, ..., `Sn` și resturile pe care acestea le dau prin împărțire la `n` astfel:

```
S1=C1 ⇒ R1=S1 % n
S2=C1+C2     ⇒ R2=S2 % n
...
Si=C1+C2+...+Ci    ⇒ Ri=Si % n
...
Sn=C1+C2+...+Cn    ⇒ Rn=Sn % n
```


Cazul I: Există un rest `Ri=0`. În acest caz, suma `Si` este divizibilă cu `n`, prin urmare camerele solicitate sunt `1, 2, ..., i`.

Cazul II: Toate resturile sunt diferite de `0`. Prin urmare `R1, R2, ..., Rn` sunt `n` resturi care iau valori în mulțimea `{1, 2, ..., n-1}`. În mod obligatoriu există cel puțin două resturi egale: Ri=Rj (`i<j`), adică `Si` și `Sj` produc același rest la împărțirea cu `n` ⇒ suma `Sj-Si` este divizibilă cu `n`, deci camerele solicitate sunt `i+1, i+2, ..., j`.

Observații

1. Soluția nu este unică. Procedeul prezentat produce o soluție posibilă.
2. Rezolvarea se bazează pe *Principiul cutiei al lui Dirichlet*[27].

```
#include <iostream.h>
int main ()
{ int SR[100], n, i, j, k, c;
  cout <<"n="; cin >> n;
  SR[0]=0;                    //calculez sumele de la citire
  for (i=1; i<=n; i++)
      {cout<<"C"<<i<<"="; cin>>c;
       SR[i]=SR[i-1]+c;}
  for (i=1; i<=n; i++) SR[i]%=n;        //calculez resturile
  cout << "Solutia este " << endl;
  for (i=1; i<=n; i++) //caut un rest = 0
      if (!SR[i])
         {for (k=1;k<=i; k++) cout <<k<<' '; return 0;}
  //cazul II, caut doua resturi egale
  for (i=1; i<n; i++)
      for (j=i+1; j<=n; j++)
          if (SR[i]==SR[j])
             {for (k=i+1;k<=j;k++) cout<<k<<' ';return 0;}
  return 0;}
```

Problema celebrității

Numim celebritate o persoană care este cunoscută de toată lumea, dar nu cunoaște pe nimeni. Cunoscând relațiile dintr-o comunitate de `n` persoane (n≤50), se pune problema de a identifica în comunitate o celebritate, dacă aceasta există.

Soluție 1

Reprezentăm relațiile din cadrul comunității ca o matrice pătratică cu `n` linii:

`a[i][j]=1`, dacă persoana `i` cunoaște pe persoana `j`, și `0`, în caz contrar.

27. Matematicianul Peter Gustav Lejeune Dirichlet a enunțat următorul principiu: Se consideră n obiecte care trebuie plasate în p cutii, unde n>p*k, k∈N*. Oricum am plasa obiectele, există o cutie care va conține k+1 obiecte.

O primă soluție ar fi să calculăm pentru fiecare persoană p din grup numărul de persoane pe care p le cunoaște (out) și numărul de persoane care cunosc persoana p (in). Dacă găsim o persoană pentru care out=1 și in=n, aceasta va fi celebritatea căutată (am presupus că orice persoană se cunoaște pe sine însăși).

```
#include <iostream.h>
int main()
{int a[100][100], n, i, j, in, out, p, celebritate=-1;
 cout <<"n="; cin >> n;
 cout << "Introduceti matricea relatiilor!\n";
 for (i=0; i<n; i++)
     for (j=0; j<n; j++) cin>>a[i][j];
 for (p=0; p<n; p++)
     {for(in=out=j=0; j<n; j++)
          in+=a[j][p], out+=a[p][j];
     if (in==n && out==1) celebritate=p;}
if (celebritate<0) cout<<"În grup nu exista celebritati !";
    else cout<<celebritate+1<<" este o celebritate. ";
return 0;}
```

Se poate observa cu ușurință că algoritmul este *pătratic* (numărul de operații efectuate de algoritm este comparabil ca ordin de mărime cu n^2).

Soluție 2

Putem îmbunătăți algoritmul făcând observația că, atunci când testăm relațiile dintre două persoane x și y, apar următoarele posibilități :

a[x,y]=0, și în acest caz y nu are nici o șansă să fie celebritate, sau

a[x,y]=1, și în acest caz x nu poate fi celebritate.

Deci la un test eliminăm o persoană care nu are șanse să fie celebritate. Efectuând succesiv n-1 teste, în final vom avea o singură persoană candidat la celebritate. Rămâne să calculăm numărul de persoane cunoscute și numărul de persoane care îl cunosc pe acest candidat, singura celebritate posibilă. Secvența de determinare a celebrității devine:

```
for (celebritate=0, i=1; i<n; i++)
    if (a[celebritate][i]) celebritate=i;
for (out=in=i=0; i<n; i++)
    in+=a[i][celebritate], out+=a[celebritate][i];
```

În acest caz, algoritmul a devenit *liniar* (numărul de operații efectuate de algoritm este comparabil ca ordin de mărime cu n).

Cadrane

Fie n un număr natural impar (n≤100). Să se construiască o matrice pătratică având n linii și n coloane după modelul din exemplul următor:

2	2	0	1	1
2	2	0	1	1
0	0	0	0	0
3	3	0	4	4
3	3	0	4	4

Soluție

Numărul n fiind impar, există o linie și o coloană „de mijloc". Dacă vom numerota liniile și coloanele de la 0 la n-1, linia, respectiv coloana din mijloc, are numărul n/2. Observăm din exemplu că toate elementele de pe linia, respectiv coloana „de mijloc", au valoarea 0.

Putem să ne imaginăm că elementul din mijlocul matricei (cel de coordonate n/2, n/2) ar fi centrul unui sistem de coordonate, linia n/2 fiind axa Ox, iar coloana n/2 fiind axa Oy. Observăm din exemplu că celelalte elemente ale matricei au valori egale cu numărul cadranului în care se află.

```
#include <iostream.h>
#define NMax 100
int main()
{int a[NMax][NMax], n, i, j;
 cout<<"n= "; cin>>n;
 //completam linia n/2 cu 0
 for (i=0; i<n; i++) a[n/2][i]=0;
 //completam coloana n/2 cu 0
 for (i=0; i<n; i++) a[i][n/2]=0;
 //completam cadranul 1
 for (i=0; i<n/2; i++)
     for (j=n/2+1; j<n; j++) a[i][j]=1;
 //completam cadranul 2
 for (i=0; i<n/2; i++)
     for (j=0; j<n/2; j++) a[i][j]=2;
 //completam cadranul 3
 for (i=n/2+1; i<n; i++)
     for (j=0; j<n/2; j++) a[i][j]=3;
 //completam cadranul 4
 for (i=n/2+1; i<n; i++)
     for (j=n/2+1; j<n; j++) a[i][j]=4;
 //afisam matricea
 for (i=0; i<n; i++)
     {for (j=0; j<n; j++) cout<<a[i][j]<<' ';
      cout<<endl;     }
 return 0; }
```

Matrice

Fie n un număr natural impar (n≤100). Să se construiască o matrice pătratică având n linii și n coloane după modelul din exemplul următor:

0	1	1	1	0
2	0	1	0	4
2	2	0	4	4
2	0	3	0	4
0	3	3	3	0

Soluție

Din exemplu, observăm că toate elementele situate pe diagonala principală sau pe diagonala secundară sunt 0. Cele două diagonale împart matricea în patru zone. Zona 1 este constituită din elemente situate deasupra diagonalei principale și deasupra diagonalei secundare. Toate elementele din zona 1 au valoarea 1. Zona 2 este constituită din elemente situate sub diagonala principală și deasupra diagonalei secundare. Toate elementele din zona 2 au valoarea 2. Zona 3 este constituită din elemente situate sub diagonala principală și sub diagonala secundară. Toate elementele din zona 3 au valoarea 3. Zona 4 este constituită din elemente situate deasupra diagonalei principale și sub diagonala secundară. Toate elementele din zona 4 au valoarea 4.

```
#include <iostream.h>
#define NMax 100
int main()
{int a[NMax][NMax], n, i, j;
 cout<<"n= "; cin>>n;
 //completam diagonala principala cu 0
 for (i=0; i<n; i++) a[i][i]=0;
 //completam diagonala secundara cu 0
 for (i=0; i<n; i++) a[i][n-i-1]=0;
 //completam zona 1
 for (i=0; i<n/2; i++)
     for (j=i+1; j<n-i-1; j++) a[i][j]=1;
 //completam zona 2
 for (j=0; j<n/2; j++)
     for (i=j+1; i<n-j-1; i++) a[i][j]=2;
 //completam zona 3
 for (i=n/2+1; i<n; i++)
     for (j=n-i; j<i; j++) a[i][j]=3;
 //completam cadranul 4
 for (j=n/2+1; j<n; j++)
     for (i=n-j; i<j; i++) a[i][j]=4;
 //afisam matricea
 for (i=0; i<n; i++)
     {for (j=0; j<n; j++)  cout<<a[i][j]<<' ';
     cout<<endl;     }
 return 0; }
```


Situație școlară

Profesorul diriginte dorește ca la sfârșitul anului școlar părinții să aibă o imagine clară asupra situației școlare a copilului lor, dar și a întregii clase.

Din catalogul clasei (memorat în fișierul `catalog.in`), dirigintele preia următoarele informații: numărul de elevi, numărul de obiecte de studiu, media anuală a fiecărui elev pentru fiecare obiect de studiu. Dacă un elev nu studiază o anumită disciplină (de exemplu, este scutit la educație fizică), media lui anuală la obiectul respectiv va fi 0 și nu va fi luată în calcul pentru determinarea mediei generale a elevului.

Profesorul diriginte trebuie să determine și să afișeze în fișierul `situatie.out` următoarele informații:

1. Media generală a fiecărui elev promovat (care nu are nici o medie mai mică decât 5).
2. Media generală a clasei.
3. Media generală a clasei pentru fiecare obiect de studiu.
4. Elevii premianți (identificați prin numărul lor de ordine din catalog).

Regulamentul de stabilire a premiilor este următorul:

- obține premiul I elevul cu media cea mai mare din clasă, medie care nu trebuie să fie mai mică de 9.50; dacă există mai mulți elevi cu medie maximă, toți obțin premiul I.
- obține premiul al II-lea elevul (sau elevii, în caz de egalitate) cu cea mai mare medie din clasă (exceptând elevii care au obținut premiul I), medie care nu trebuie să fie mai mică decât 9.00.
- obține premiul al III-lea elevul (sau elevii, în caz de egalitate) cu cea mai mare medie din clasă (exceptând elevii care au obținut premiul I și premiul al II-lea), medie care nu trebuie să fie mai mică decât 8.50.

Soluție

Vom memora datele din catalog într-o matrice `C` cu `n` linii, numerotate de la `1` la `n` (unde `n` este numărul de elevi din clasă), și `m` coloane, numerotate de la `1` la `m` (unde `m` este numărul de obiecte de studiu). Elementul de pe linia `i` și coloana `j` din matricea `C` reprezintă media anuală pe care elevul `i` a obținut-o la obiectul `j`.

Observați că linia `0` și coloana `0` din matrice nu sunt utilizate deocamdată. Vom utiliza coloana `0` pentru a reține mediile generale ale elevilor (`C[i][0]`=media generală a elevului `i`), iar linia `0` o vom utiliza pentru a reține mediile generale pe obiecte (`C[0][j]`=media clasei la obiectul `j`). Rămâne disponibil elementul de pe linia `0` și coloana `0` (`C[0][0]`) pentru a reține media generală a clasei.

Pentru a determina media generală a fiecărui elev trebuie să parcurgem fiecare linie a matricei și să calculăm media aritmetică a elementelor diferite de `0` de pe linia

respectivă. Pe parcursul acestui calcul verificăm şi dacă elevul este promovat. Dacă vom descoperi o media mai mică decât 5, media generală a acestui elev va rămâne 0.

Pentru a determina media clasei pentru fiecare obiect trebuie să parcurgem fiecare coloană a matricei şi să calculăm media aritmetică a elementelor diferite de 0 de pe linia respectivă.

Pentru a calcula media generală a clasei este suficient să facem media mediilor generale ale elevilor promovaţi (deci media elementelor nenule de pe coloana 0).

Pentru a determina premianţii, determinăm max1, max2 şi max3 (cele mai mari trei medii generale distincte), punând condiţia suplimentară ca max1>=9.50 (în caz contrar, nu se acordă premiul I), max2>=9.00 (în caz contrar, nu se acordă premiul al II-lea), max3>=8.50 (în caz contrar, nu se acordă premiul al III-lea).

Dacă se acordă premiul I, vom parcurge toţi elevii şi vom afişa pe cei care au media generală egală cu max1. În acelaşi mod procedăm şi cu premiile al II-lea şi al III-lea.

```
#include <fstream.h>
#include <iomanip.h>
int n, m;
double C[50][30];
int main()
{int i, j, nr, corigent;
 double s, max1, max2, max3;
 ifstream fin("Catalog.in");
 ofstream fout("Situatie.out");
 //citire
 fin>>n>>m;
 for (i=1; i<=n; i++)
     for (j=1; j<=m; j++) fin>>C[i][j];
 fin.close();
 //calculam mediile generale ale elevilor
 for (i=1; i<=n; i++)
     {s=0; nr=0; corigent=0;
      for (j=1; j<=m && !corigent; j++)
          if (C[i][j])
             if (C[i][j]<5) corigent =1;
                 else {s+=C[i][j]; nr++;}
      if (corigent) C[i][0]=0;
          else C[i][0]=s/nr;
     }
 //afisarile se vor face cu doua zecimale
 fout<<setprecision(2);
 //afisam  mediile generale ale elevilor
 fout<<"Mediile generale ale elevilor:\n";
 for (i=1; i<=n; i++)
     fout<<"Media elevului "<<i<<" este "<<C[i][0]<<endl;
 fout<<endl;
```

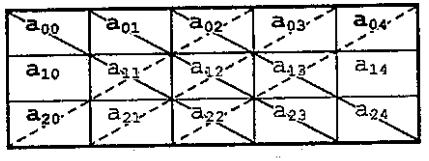


```
//calculam mediile pe obiecte
for (i=1; i<=m; i++)
    {for (s=0, nr=0, j=1; j<=n ; j++)
         if (C[j][i])
            {s+=C[j][i]; nr++;}
     C[0][i]=s/nr;     }
//afisam mediile pe obiecte
fout<<"Mediile clasei pe obiecte:\n";
for (i=1; i<=m; i++)
    fout<<"Media la obiectul "<<i<<" e "<<C[0][i]<< endl;
//calculam media generala clasei
s=0; nr=0;
for (i=1; i<=n; i++)
    if (C[i][0]){s+=C[i][0]; nr++;}
C[0][0]=s/nr;
fout<<"\nMedia generala a casei este "<<C[0][0]<<endl;
//calculam max1, max2, max3
max1=max2=max3=0;
for (i=1; i<=n; i++)
    {if (C[i][0]>max1)
        {max3=max2; max2=max1;max1=C[i][0];}
     if (C[i][0]<max1 && C[i][0]>max2)
        {max3=max2; max2=C[i][0];}
     if (C[i][0]<max2 && C[i][0]>max3) max3=C[i][0];
    }
if (max1>=9.50)
   //se acorda premiul I
   {fout<<"Se acorda Premiul I cu media "<<max1;
    fout<<" elevilor:\n";
    for (i=1; i<=n; i++)
        if (C[i][0]==max1) fout<<i<<' ';
    fout<<endl;  }
   else
   {fout<<"Nu se acorda Premiul I\n";
    max3=max2; max2=max1;}
if (max2>=9)
   //se acorda premiul al II-lea
   {fout<<"Se acorda Premiul al II-lea cu media "<<max2;
    fout<<" elevilor :\n";
    for (i=1; i<=n; i++)
        if (C[i][0]==max2) fout<<i<<' ';
    fout<<endl; }
   else
   {fout<<"Nu se acorda Premiul al II-lea\n";
    max3=max2;  }
if (max3>=8.50)
   //se acorda premiul al III-lea
   {fout<<"Se acorda Premiul al III-lea cu media "<<max3;
    fout<<" elevilor :\n";
```

```
    for (i=1; i<=n; i++)
        if (C[i][0]==max3)  fout<<i<<' ';
    fout<<endl;
   }
   else fout<<"Nu se acorda Premiul al III-lea\n";
fout<<endl<<max1<<' '<<max2<<' '<<max3<<endl;
fout.close(); return 0; }
```

Pseudodiagonale

Fie `A` o matrice cu `n` linii şi `m` coloane (`1≤n<m≤100`) cu componente întregi. Nefiind pătratică, matricea `A` nu are diagonale, dar are pseudodiagonale principale şi secundare, aşa cum este ilustrat în exemplul următor:



În exemplu am figurat o matrice cu 3 linii şi 5 coloane. În această matrice sunt 3 pseudodiagonale principale (trasate cu linie continuă) şi 3 pseudodiagonale secundare (trasate cu linie punctată).

Scrieţi un program care să determine suma elementelor de pe fiecare pseudodiagonală principală şi de pe fiecare pseudodiagonală secundară.

Soluţie

Pentru a rezolva această problemă sunt câteva întrebări la care trebuie să răspundem:

1. Câte pseudodiagonale există într-o matrice cu `n` linii şi `m` coloane, cu `n<m`?
2. Care este poziţia de început şi poziţia de sfârşit pentru fiecare pseudodiagonală?
3. Care este relaţia dintre indicele de linie şi cel de coloană pe fiecare pseudodiagonală?

Pentru a răspunde la aceste întrebări vom urmări într-un tabel variaţia poziţiei de început şi a poziţiei de sfârşit pentru pseudodiagonale:

Pseudodiagonala	Linie început	Coloana început	Linie sfârşit	Coloana sfârşit
0	0	0	n-1	n-1
1	0	1	n-1	n
2	0	2	n-1	n+1
...				...
d	0	d	n-1	n+d-1
...				
ultima	0	m-n	n-1	m-1

Observaţii

1. Fiecare pseudodiagonală începe pe linia `0` şi se termină pe linia `n-1`.
2. Pseudodiagonala `0` începe în coloana `0`, pseudodiagonala `1` începe în coloana `1` etc. Prin urmare, coloana de început a unei pseudodiagonale este egală cu numărul pseudodiagonalei.
3. Interpretând geometric, observăm că o pseudodiagonală principală formează un unghi de 45° cu orizontala, deci este ipotenuza unui triunghi dreptunghic isoscel. Pentru a calcula coloana de sfârşit pentru fiecare pseudodiagonală, punem condiţia de egalitate a catetelor.
4. Ultima pseudodiagonală se va termina în coloana `m-1`, deci (din condiţia de triunghi dreptunghic isoscel) deducem că începe în coloana `n-m`. Am determinat astfel că numărul de pseudodiagonale este `n-m+1`.

Vom parcurge succesiv cele `n-m+1` pseudodiagonale principale, calculând pentru fiecare pseudodiagonală suma elementelor.

Pentru a parcurge pseudodiagonala principală `d`, trebuie să determinăm relaţia care există între indicele de linie şi cel de coloană pe această pseudodiagonală. În acest scop, vom urmări în tabelul următor variaţia indicelui de linie în raport cu cel de coloană pe pseudodiagonala `d`:

Linia	0	1	2	...	i	...	n-1
Coloana	d	d+1	d+2		i+d		n+d-1

Observaţii

Când indicele de linie creşte cu 1, indicele de coloană creşte, de asemenea, cu 1. Pin urmare, diferenţa dintre indicele de coloană şi indicele de linie este constantă: `coloana-linie=d`. Deducem relaţia: `coloana=linie+d`.

Raţionamentul este analog pentru pseudodiagonalele secundare:

Pseudodiagonala	Linie început	Coloana început	Linie sfârşit	Coloana sfârşit
0	0	m-1	n-1	m-n
1	0	m-2	n-1	m-n-1
2	0	m-3	n-1	m-n-2
...				
d	0	m-d-1	n-1	m-n-d
...				
ultima	0	n-1	n-1	0

Observaţii

1. Fiecare pseudodiagonală secundară începe pe linia `0` şi se termină pe linia `n-1`.
2. Pseudodiagonala secundară `0` începe în coloana `m-1`, pseudodiagonala `1` începe în coloana `m-2` etc. Prin urmare, coloana de început a pseudodiagonalei secundare `d` este egală cu `m-d+1`.

3. Interpretând geometric, observăm că o pseudodiagonală secundară formează un unghi de 45° cu orizontala, deci este ipotenuza unui triunghi dreptunghic isoscel. Pentru a calcula coloana de sfârşit pentru fiecare pseudodiagonală, punem condiţia de egalitate a catetelor.
4. Ultima pseudodiagonală se va termina în coloana 0, deci (din condiţia de triunghi dreptunghic isoscel) deducem că începe în coloana n-1.

Vom parcurge succesiv cele n-m+1 pseudodiagonale secundare, calculând pentru fiecare pseudodiagonală secundară suma elementelor. Pentru a parcurge pseudodiagonala secundară d, trebuie să determinăm relaţia care există între indicele de linie şi cel de coloană pe această pseudodiagonală. În acest scop, vom urmări în tabelul următor variaţia indicelui de linie în raport cu cel de coloană pe pseudodiagonala secundară d:

Linia	0	1	2	...	i	...	n-1
Coloana	m-d-1	m-d-2	m-d-3		m-d-i-1		m-n-d

Observaţii

Când indicele de linie creşte cu 1, indicele de coloană scade cu 1. Prin urmare, suma dintre indicele de coloană şi indicele de linie este constantă: coloana+linie=m-d-1. Deducem relaţia: coloana=m-d-linie-1.

```
#include <iostream.h>
int main()
{int n, m, a[100][100], i, j, d, s;
 cout<<"n= "; cin>>n;
 cout<<"m= "; cin>>m;
 cout<<"Introduceti elementele matricei, linie cu linie\n";
 for (i=0; i<n; i++)
     for (j=0; j<m; j++) cin>>a[i][j];
 //parcurgem pseudodiagonalele principale
 for (d=0; d<m-n+1; d++)
     {s=0; //suma elementelor de pe pseudodiagonala d
      for (i=0; i<n; i++)
          s+=a[i][i+d];
      cout<<"Suma elementelor de pe pseudodiagonala
            principala "<<d<<" este "<<s<<endl;
     }
 //parcurgem pseudodiagonalele secundare
 for (d=0; d<m-n+1; d++)
     {s=0; //suma elementelor de pe pseudodiagonala d
      for (i=0; i<n; i++)
          s+=a[i][m-d-i-1];
      cout<<"Suma elementelor de pe pseudodiagonala
            secundara "<<d<<" este "<<s<<endl;
     }
 return 0;}
```


Exerciţiu

Modificaţi programul precedent astfel încât să rezolve problema pentru cazul n>m.

Submulţimi cu sume egale

Fie n un număr natural nenul, n≤50. Scrieţi un program care să determine n submulţimi disjuncte cu n elemente din mulţimea {1, 2, ..., n^2}, submulţimi pentru care suma elementelor este aceeaşi.

Soluţie

Problema pare deosebit de complexă. O primă idee, greu de implementat, este de a genera toate submulţimile de n elemente ale mulţimii {1, 2, ..., n^2}, apoi de a alege n submulţimi disjuncte cu suma elementelor aceeaşi.

Pe lângă faptul că această idee este greu de implementat, este şi inutilizabilă din punct de vedere practic, pentru că numărul de submulţimi este foarte mare.

Soluţia se bazează pe o idee constructivă.

1. Plasăm în ordine elementele de la 1 la n^2 într-o matrice pătratică cu n linii şi n coloane. De exemplu, pentru n=4 obţinem:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

2. Alegem ca primă submulţime elementele de pe diagonala principală {1, 6, 11, 16}. Suma elementelor din submulţime este S=1+6+11+16=34.
3. Pentru a construi cu uşurinţă celelalte n-1 submulţimi cu suma elementelor S, vom extinde matricea, copiind în continuare primele n-1 linii. Se obţine astfel o matrice cu 2*n-1 linii şi n coloane. De exemplu, pentru n=4 obţinem:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
1	2	3	4
5	6	7	8
9	10	11	12

Această matrice are n pseudodiagonale principale. Mulțimile elementelor de pe oricare două pseudodiagonale sunt disjuncte, iar suma elementelor de pe fiecare pseudodiagonală este aceeași. Problema se reduce astfel la o problemă asemănătoare cu precedenta.

```
#include <iostream.h>
int main()
{int n, m, a[101][51], i, j, d, nr=1;
 cout<<"n= "; cin>>n;
 m=2*n-1;
 for (i=0; i<n; i++)
     for (j=0; j<n; j++) a[i][j]=a[i+n][j]=nr++;
 //parcurgem pseudodiagonalele
 for (d=0; d<n; d++)
     {//afisez elementele de pe pseudodiagonala d
      cout<<"Submultimea "<<d+1<<": { ";
      for (j=0; j<n; j++) cout<<a[j+d][j]<<' ';
      cout<<"}\n";      }
 return 0;}
```

Dreptunghi de sumă maximă

Fie A o matrice cu n linii și m coloane cu elemente întregi (n, m≤50). Să se determine o zonă dreptunghiulară în această matrice pentru care suma elementelor din această zonă este maximă. În cazul în care există mai multe soluții, veți afișa o zonă de arie maximă.

Soluție

O zonă dreptunghiulară din matrice este formată din elemente A[i][j] cu proprietatea că x1≤i≤x2 și y1≤j≤y2. Pentru a identifica o zonă dreptunghiulară din matrice este suficient să reținem (x1, y1) (coordonatele colțului din stânga-sus al zonei) și (x2, y2) (coordonatele colțului din dreapta-jos al zonei).

O primă idee ar fi să generăm toate zonele dreptunghiulare și, pentru fiecare zonă, să calculăm suma elementelor. Obținem astfel toate sumele posibile, dar reținem numai suma maximă.

```
#include <iostream.h>
int main()
{int n, m, a[100][100], i, j, x1,y1,x2,y2;
 int smax, x1max, y1max, x2max, y2max, s;
 cout<<"n, m= "; cin>>n>>m;
 cout<<"Introduceti elementele matricei, linie cu linie\n";
 for (i=1; i<=n; i++)
    for (j=1; j<=m; j++) cin>>a[i][j];
 smax=a[0][0]; x1max=x2max=y1max=y2max=0;
 for (x1=1; x1<=n; x1++)
     for (y1=1; y1<=m; y1++)
```



```
         for (x2=x1; x2<=n; x2++)
             for (y2=y1; y2<=m; y2++)
                {
                /* calculez suma elementelor din
                dreptunghiul cu coltul stanga-sus (x1, y1)
                si coltul dreapta-jos (x2, y2) */
                for (s=0, i=x1; i<=x2; i++)
                    for (j=y1; j<=y2; j++) s+=a[i][j];
                if (s>smax)
                   {smax=s; //retin suma maxima
                    x1max=x1; y1max=y1; //si colturile
                    x2max=x2; y2max=y2;}
                   else //aceeasi suma dar arie mai mare?
                   if (s==smax && (x2-x1+1)*(y2-y1+1)>
                      (x2max-x1max+1)*(y2max-y1max+1))
                      {x1max=x1;y1max=y1; //retin colturile
                       x2max=x2; y2max=y2;}
                }
cout<<"Suma maxima: "<<smax<<endl;
cout<<"Coltul stanga-sus: "<<x1max<<", "<<y1max<<endl;
cout<<"Coltul dreapta-jos: "<<x2max<<", "<<y2max<<endl;
return 0; }
```

Acesta este cel mai simplu, dar totodată și cel mai ineficient algoritm la care ne-am fi putut gândi. Numărul de operații este aproximativ $(n*m)^3$.

O idee mai eficientă, dar care necesită spațiu de memorie suplimentar este de a calcula și memora într-o matrice auxiliară sum sumele din toate dreptunghiurile care au colțul din stânga-sus (1, 1) și colțul din dreapta-jos (x, y).

Suma elementelor din orice alt dreptunghi se poate determina pe baza informațiilor din matricea sum. Mai exact, pentru a determina suma elementelor din dreptunghiul cu colțul stânga-sus (x1, y1) și cel din dreapta-jos (x2, y2), formula de calcul este:

```
sum[x2][y2]-sum[x2][y1-1]-sum[x1-1][y2]+sum[x1-1][y1-1]
```

```
#include <iostream.h>
int sum[100][100];
int main()
{int n, m, a[100][100], i, j, x1,y1,x2,y2;
 int smax, x1max, y1max, x2max, y2max, s;
 cout<<"n, m= "; cin>>n>>m;
 cout<<"Introduceti elementele matricei, linie cu linie\n";
 for (i=1; i<=n; i++) for (j=1; j<=m; j++) cin>>a[i][j];
 smax=a[0][0]; x1max=x2max=y1max=y2max=0;
 for (x1=1; x1<=n; x1++)
     for (y1=1; y1<=m; y1++)
    /* calculez suma elementelor din dreptunghiul cu
    coltul stanga-sus (0,0) si coltul dreapta-jos (x1,y1)*/
          for (i=1; i<=x1; i++)
               for (j=1; j<=y1; j++) sum[x1][y1]+=a[i][j];
```

```
for (x1=1; x1<=n; x1++)
    for (y1=1; y1<=m; y1++)
        for (x2=x1; x2<=n; x2++)
            for (y2=y1; y2<=m; y2++)
                {s=sum[x2][y2]-sum[x1-1][y2]-
                  sum[x2][y1-1]+sum[x1-1][y1-1];
                if (s>smax)
                  {smax=s;
                   x1max=x1; y1max=y1;
                   x2max=x2; y2max=y2;}
                  else
                  if (s==smax &&
                     (x2-x1+1)*(y2-y1+1)>
                     (x2max-x1max+1)*(y2max-y1max+1))
                     {x1max=x1; y1max=y1;
                      x2max=x2; y2max=y2;}
                }
cout<<"Suma maxima: "<<smax<<endl;
cout<<"Coltul stanga-sus: "<<x1max<<", "<<y1max<<endl;
cout<<"Coltul dreapta-jos: "<<x2max<<", "<<y2max<<endl;
return 0; }
```

Complexitatea algoritmului în acest caz este de $(n*m)^2$. Există şi algoritmi mai performanţi de rezolvare a acestei probleme, dar îi vom studia ulterior.

Secvenţe

Fie A o secvenţă formată din N valori distincte, cuprinse între 1 şi N, date într-o ordine oarecare. Definim secvenţa B astfel: B[k]=1 dacă şi numai dacă primele k valori din secvenţa A sunt numai numere distincte din mulţimea {1, 2, ..., k} într-o ordine oarecare; în caz contrar, B[k]=0.

Dată fiind secvenţa B şi câteva elemente din secvenţa A, scrieţi un program care să reconstituie secvenţa A.

Date de intrare

Fişierul de intrare secv.in conţine:

- pe prima linie, două numere naturale N şi M;
- pe cea de-a doua linie, elementele secvenţei B, separate prin spaţii;
- pe fiecare dintre următoarele M linii, informaţii despre elementele cunoscute din secvenţa A sub forma a două numere naturale X şi Y cu semnificaţia „al X-lea element din secvenţa A este Y".

Date de ieşire

Fişierul de ieşire secv.out conţine o singură linie pe care sunt afişate elementele secvenţei A separate prin câte un spaţiu. Dacă nu există soluţie, se va afişa în fişierul de ieşire mesajul IMPOSIBIL.

Restricţii

- $1 \le N \le 10000$
- $0 \le M \le N$
- informaţiile din fişierul de intrare nu sunt contradictorii;
- datele de pe aceeaşi linie din fişiere sunt separate prin spaţii;
- soluţia nu este în mod necesar unică.

Exemple

secv.in	secv.out	secv.in	secv.out
8 3 0 0 0 1 0 0 1 1 1 2 5 6 2 7	IMPOSIBIL	7 2 0 0 0 1 0 0 1 1 2 5 6	2 4 3 1 6 7 5

(Olimpiada Municipală de Informatică, clasa a IX-a, Iaşi, 2003)

Soluţie

Vom utiliza un vector sol cu N componente în care încercăm să reconstituim secvenţa A. Chiar de la citire vom plasa în vectorul sol cele M elemente pentru care sunt cunoscute poziţiile. De asemenea, vom utiliza un vector stiu, cu semnificaţia stiu[i]=1, dacă elementul i este deja plasat în vectorul sol, şi 0, în caz contrar. La fiecare pas vom căuta un 1 în vectorul B. Segmentul din secvenţa A care ţine de la precedentul 1 (sau de la începutul vectorului) până la acest 1 trebuie să conţină toate valorile întregi din intervalul dintre aceste două poziţii. O parte dintre aceste valori sunt deja fixate, celelalte le plasăm în orice ordine dorim.

```
#include <fstream.h>
#define INFILE   "secv.in"
#define OUTFILE  "secv.out"
#define MAXN      10000
int b[MAXN+1], sol[MAXN+1], stiu[MAXN+1];
int n, M;
int main ( )
{ int i, temp, temp1, max;
  //citire
  ifstream fin(INFILE);
  fin>>n>>M;
  for (i=1; i<=n; i++) fin>>b[i];
  for (i=0; i<M; i++)
      {fin>>temp>>temp1;
       sol[temp]=temp1; stiu[temp1]=1;}
  fin.close();
  int curent=1, inceput, w,  j, ok=1;
  while (curent<=n && ok)
       {inceput=curent;
        while (!b[curent] && curent<=n) curent++;
```

```
        if (curent>n) ok=0;
       /*secventa de la inceput la curent este o permutare
       a valorilor de la inceput la curent*/
       for (w=inceput, j=curent; j>=inceput; j--)
          if (!stiu[j])
           {while (sol[w]) w++;
            sol[w]=j; }
      curent++;
   }
ofstream fout(OUTFILE);
if (!ok) fout<<"IMPOSIBIL\n";
   else
   {//verific
    for (max=0, i=1; i<=n && ok; i++)
        {if (sol[i]>max) max=sol[i];
         if (b[i] && max!=i) ok=0;
         if (!b[i] && max==i) ok=0;        }
    if (!ok) fout<<"IMPOSIBIL\n";
       else
       {for (i=1; i<n; i++) fout<<sol[i]<<' ';
        fout<<sol[n]<<endl;}
   }
fout.close(); return 0;}
```

Problema spectacolelor

Managerul artistic al unui festival trebuie să selecteze o mulțime cât mai amplă de spectacole ce pot fi jucate în singura sală pe care o are la dispoziție. Știind că i s-au propus $n \leq 100$ spectacole și pentru fiecare spectacol i i-a fost anunțat intervalul în care se poate desfășura $[s_i, f_i)$ (s_i reprezintă ora și minutul de început, iar f_i ora și minutul de final al spectacolului i), scrieți un program care să permită spectatorilor vizionarea unui număr cât mai mare de spectacole. De exemplu, dacă vom citi $n=5$ și următorii timpi:

```
12 30 16 30
15 0 18 0
10 0 18 30
18 0 20 45
12 15 13 0
```

Spectacolele selectate sunt: `5 2 4`.

Soluție

Ordonăm spectacolele crescător după ora de final. Selectăm inițial primul spectacol (deci cel care se termină cel mai devreme). La fiecare pas selectăm primul spectacol neselectat care nu se suprapune cu cele deja selectate (deci care începe după ce se termină ultimul spectacol selectat).


```
#include <iostream.h>
int inceput[100], sfarsit[100], nr[100];
int main()
{int n, i, h, m, schimb, ultim, aux;
 cout << "n= "; cin >> n;                          //citire
 cout<<"Introduceti inceputul si sfarsitul spectacolelor";
 for (i=0; i<n; i++)
       {nr[i]=i+1;            //transform timpul in minute
        cin>>h>>m; inceput[i]=h*60+m;
        cin>>h>>m; sfarsit[i]=h*60+m;}
 //ordonez spectacolele crescator dupa ora de final
 do
  {schimb=0;
   for (i=0; i<n-1; i++)
        if (sfarsit[nr[i]]>sfarsit[nr[i+1]])
           {aux=nr[i];nr[i]=nr[i+1];nr[i+1]=aux; schimb=1;}
  }
 while (schimb);
 cout << "Spectacolele selectate sunt:\n"<<nr[0]<<' ';
 for (ultim=0, i=1; i<n; i++)
      if (inceput[nr[i]]>=sfarsit[nr[ultim]])
          {cout <<nr[i]<<' '; ultim=i;}
 cout<<endl;
 return 0;}
```

Pentru rezolvarea acestei probleme am utilizat o metodă de programare importantă, denumită *Greedy*. În general, metoda *Greedy* se aplică problemelor de optimizare. Specificul acestei metode constă în faptul că se construiește soluția optimă pas cu pas, la fiecare pas fiind selectat (sau „înghițit") în soluție elementul care pare „cel mai bun" la momentul respectiv, în speranța că această alegere locală va conduce la optimul global.

Algoritmii *Greedy* sunt foarte eficienți, dar nu conduc în mod necesar la o soluție optimă. Și nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda *Greedy* rezolvă sau nu o anumită problemă de optimizare. Din acest motiv, orice algoritm *Greedy* trebuie însoțit de o demonstrație a corectitudinii sale[28]. Demonstrația faptului că o anumită problemă are proprietatea alegerii *Greedy* se face de obicei prin inducție matematică.

Problema rucsacului

Un hoț neprevăzător are la dispoziție un singur rucsac cu care poate transporta o greutate maximă $Gmax$. Hoțul are de ales din $n \leq 50$ obiecte și, evident, intenționează să obțină un câștig maxim în urma singurului transport pe care îl poate face. Cunoscând

28. Demonstrația depășește nivelul de cunoștințe al unui elev de clasa a IX-a.

pentru fiecare obiect i greutatea g_i şi câştigul c_i pe care hoţul l-ar obţine transportând obiectul respectiv în întregime, scrieţi un program care să determine o încărcare optimă a rucsacului, în ipoteza că hoţul poate încărca în rucsac orice parte dintr-un obiect. De exemplu, pentru n=5, GMax=100 şi câştigurile-greutăţile următoare:

(1000 120) (500 20) (400 200) (1000 100) (25 1), se va afişa pe ecran:

```
2          100.00%
4          79.00%
5          100.00%
```

Soluţie

Vom reprezenta o soluţie a problemei ca un vector x cu n componente, în care reţinem pentru fiecare obiect fracţiunea încărcată în rucsac de hoţ. Deci vectorul x trebuie să îndeplinească următoarele condiţii:

1. $x_i \in [0, 1], \forall i \in \{1, 2, \ldots, n\}$;
2. $g_1 \cdot x_1 + g_2 \cdot x_2 + \ldots + g_n \cdot x_n \leq Gmax$;
3. $f(x) = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n \cdot x_n$ este maximă.

Ordonăm obiectele descrescător după câştigul pe unitatea de greutate (valoare care constituie o măsură a eficienţei transportării obiectelor). Cât timp este posibil (încap în rucsac), selectăm obiectele în întregime. Completăm rucsacul cu un fragment din următorul obiect ce nu a fost selectat.

```
#include <iostream.h>
int o[50]; //ordinea obiectelor
float c[100], g[100], x[100], Gr, GMax;
int n;
int main()
{int i, schimb, aux;
 cout<<"n= "; cin>> n;                    //citire
 cout<< "GMax= "; cin >> GMax;
 cout<<"Castigul si greutatea pt. fiecare obiect\n";
 for (i=0; i<n; i++)
     {o[i]=i; cin>>c[i]>>g[i];}
 //ordonez obiectele descrescator dupa castigul unitar
 do
  {schimb=0;
   for (i=0; i<n-1; i++)
       if (c[o[i]]/g[o[i]]<c[o[i+1]]/g[o[i+1]])
          {aux=o[i];o[i]=o[i+1]; o[i+1]=aux; schimb=1;}
  } while (schimb);
 for (i=0, Gr=GMax; i<n && Gr>g[o[i]]; i++)
     {x[o[i]]=1; Gr-=g[o[i]];}
 if (i<n) x[o[i]]= Gr/g[o[i]];
 cout << "Obiectele selectate sunt:\n";
 for (i=0; i<n; i++)
     if (x[i]) cout<<i+1<<' '<<x[i] * 100<<'%'<<endl;
 return 0;}
```


Observaţie

Am aplicat tot o strategie de tip *Greedy*. Se poate demonstra corectitudinea acestui algoritm în condiţiile în care putem încărca în rucsac orice parte a unui obiect. Pentru cazul în care hoţul poate încărca un obiect doar în întregime (varianta discretă a problemei rucsacului), algoritmul *Greedy* nu mai funcţionează. De exemplu, pentru n=3, Gmax=10, g=(8, 6, 4), c=(8, 6, 4), algoritmul *Greedy* va obţine soluţia x=(1, 0, 0), pentru care câştigul obţinut în urma transportului este 8. Soluţia optimă este x=(0, 1, 1), pentru care se obţine câştigul 10.

Generare de submulţimi

Fie n un număr natural nenul. Să se genereze toate submulţimile mulţimii {0, 1, ..., n-1}.

Soluţie

Vom reprezenta submulţimile prin vector caracteristic (un vector S cu n componente 0 sau 1, având semnificaţia S[i]=1, dacă i aparţine submulţimii, şi 0, în caz contrar). Problema se reduce astfel la generarea tuturor secvenţelor binare de lungime n. Pentru a rezolva această problemă vom utiliza un algoritm de tip succesor. Mai exact, un astfel de algoritm funcţionează astfel:

Pas 1. Se iniţializează soluţia cu cea mai mică configuraţie posibilă (în cazul nostru, iniţializăm vectorul S cu 0, ceea ce corespunde mulţimii vide).

Pas 2. Cât timp este posibil (există succesor), se execută:

- se afişează configuraţia curentă;
- se generează configuraţia următoare (în cazul nostru, vom considera că elementele vectorului reprezintă cifrele unui număr scris în baza 2, la care vom aduna 1).

```
#include <fstream.h>
#define NMax 100
int main()
{int n, S[NMax], i, gata=0;
 ofstream fout("subm.out");
 cout<<"n= "; cin>>n;
 for (i=0; i<n; i++) S[i]=0; //initializare
 while (!gata)
     {//afisam configuratia curenta
      for (i=0;i<n;i++) if (S[i]) fout<<i<<' '; fout<<endl;
      //generam configuratia urmatoare prin adunare binara
      for (i=0; i<n && S[i]; i++) S[i]=0;
      if (i==n) gata=1; /* este ultima configuratie  */
         else S[i]=1; }
 fout.close();
 return 0; }
```

Generare elemente produs cartezian

Fie n un număr natural (n>1) și L=(l_1, l_2, ..., l_n) o mulțime de n numere naturale nenule. Să se determine în ordine lexicografică[29] toate elementele produsului cartezian {1, 2, ..., l_1}x{1, 2, ..., l_2}x...x{1, 2, ..., l_n}.

De exemplu, pentru n=3 și L=(2, 3, 2), elementele produsului cartezian sunt: (1,1,1), (1,1,2), (1,2,1), (1,2,2), (1,3,1), (1,3,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2), (2,3,1), (2,3,2). Observați că produsul cartezian are $l_1*l_2*...*l_n$ elemente.

Soluție

Vom reprezenta un element al produsului cartezian ca un vector E cu n elemente, unde E[i]∈{1, 2, ..., l_i}.

Pentru a genera toate elementele produsului cartezian în ordine lexicografică, vom aplica tot un algoritm de tip succesor:

Pas 1. Inițializăm vectorul E cu 1 (cel mai mic element al produsului cartezian, din punct de vedere lexicografic).

Pas 2. Cât timp este posibil (mai există succesor),
- afișăm elementul curent;
- generăm elementul următor; în acest scop, căutăm prima componentă (începând din dreapta către stânga) care poate fi mărită (adică E[i]<L[i]); dacă găsim o astfel de componentă, o mărim și repunem pe 1 toate componentele următoare; dacă nu găsim o astfel de componentă, deducem că generarea s-a încheiat, acesta a fost cel mai mare element din punct de vedere lexicografic.

```
#include <fstream.h>
#define NMax 100
int main()
{int n, L[NMax], E[NMax], i, gata=0;
 ofstream fout("pc.out");
 //citire si initializare
 cout<<"n= "; cin>>n;
 for (i=0; i<n; i++) {cin>>L[i]; E[i]=1;}
 while (!gata)
      {//afisam configuratia curenta
       for (i=0; i<n; i++) fout<<E[i]<<' '; fout<<endl;
       //generam configuratia urmatoare
       for (i=n-1; i>=0 && E[i]==L[i]; i--) E[i]=1;
       if (i<0) /* acesta a fost ultima configuratie */
          gata=1;
          else E[i]++; }
fout.close(); return 0;}
```

29. Fie x= (x_1, x_2, ..., x_n) și y= (y_1, y_2, ..., y_m). Spunem că x precedă pe y din punct de vedere lexicografic dacă există k astfel încât $x_i=y_i$, pentru orice i<k și $x_k<y_k$ sau k>n.

Expresie

Se dă un șir de n numere naturale nenule x_1, x_2, ..., x_n și un număr natural m.

Scrieți un program care să verifice dacă valoarea expresiei $\sqrt[m]{x_1x_2...x_n}$ este un număr natural. În caz afirmativ, să se afișeze acest număr descompus în factori primi.

Date de intrare

În fișierul exp.in se află pe prima linie m, pe linia a doua n, iar pe linia a treia numerele x_1, x_2, ..., x_n separate între ele prin câte un spațiu.

Date de ieșire

În fișierul exp.out se va scrie pe prima linie cifra 0, dacă valoarea expresiei nu este un număr natural, respectiv 1, dacă este un număr natural. Dacă valoarea expresiei este un număr natural, pe următoarele linii se vor scrie perechi de forma p e (p este factor prim care apare în descompunere la puterea e≥1). Aceste perechi se vor scrie în ordine crescătoare după primul număr (adică p).

Restricții

- n – număr natural nenul <5000
- x_i – număr natural nenul <30000, i∈{1, 2, ..., n}

Exemple

exp.in	exp.out	exp.in	exp.out
2 4 32 81 100 19	0	2 4 32 81 100 18	1

(Olimpiada Județeană de Informatică, 2004, clasa a IX-a)

Soluție

Numărul de valori fiind mare, nu putem calcula produsul $x_1*x_2*...*x_n$ deoarece ar depăși, cu siguranță, valoarea maximă ce poate fi reprezentată folosind tipurile întregi ale limbajului.

Prin urmare, vom adopta o altă strategie: determinăm descompunerea în factori primi a acestui produs, fără a calcula produsul.

Pentru aceasta, descompunem în factori primi fiecare valoare x_i pe care o citim. Pentru a lucra eficient cu descompunerile în factori primi, vom considera un vector fp cu maxim 30000 de componente (30000 fiind valoarea maximă pe care o poate lua oricare număr dintre cele date). Dacă i este număr prim, fp[i] va reprezenta multiplicitatea factorului prim i în produsul $x_1*x_2*...*x_n$; dacă i nu este număr prim, fp[i] rămâne 0.

Condiția ca expresia din enunțul problemei să fie un număr natural este ca toți factorii primi care intervin în produs să aibă multiplicitățile divizibile cu m (mai exact, toate componentele vectorului fp să fie divizibile cu m).

```
#include <fstream.h>
unsigned int fp[30000];
int main ()
{ifstream fin("exp.in");
 ofstream fout("exp.out");
 int n, m, x, i, d;
 //citire
 fin >> m >> n;
 for (i=0; i<n; i++)
     {
      fin>>x;
      //descompun pe x in factori primi
      for (d=2; x>1; d++)
          while (!(x%d))
                { fp[d]++; x/=d;}
     }
 /* verific daca toti factorii primi au multiplicitatile
    divizibile cu m */
 for (i=2; i<30000 && !(fp[i]%m); i++);
 if (i<30000)
    /* exista cel putin un factor prim pentru care
       multiplicitatea nu este divizibila cu m, deci
       rezultatul expresiei nu este numar natural */
    fout << 0 << '\n';
    else
    {
    fout<<1<<'\n';
    for (i=2;i<30000; i++)
        if (fp[i])
           fout<<i<<' '<<fp[i]/m<<'\n';
    }
fout.close(); return 0; }
```

Reactivi

Într-un laborator de analize chimice se utilizează N reactivi. Se știe că, pentru a evita accidentele sau deprecierea reactivilor, aceștia trebuie să fie stocați în condiții de mediu speciale. Mai exact, pentru fiecare reactiv x se precizează intervalul $[min_x, max_x]$ în care trebuie să se încadreze temperatura de stocare a acestuia. Reactivii vor fi plasați în frigidere. Orice frigider are un dispozitiv cu ajutorul căruia putem stabili temperatura (constantă) ce va fi în interiorul aceluia (exprimată într-un număr întreg de grade Celsius).

Scrieți un program care să determine numărul minim de frigidere necesare pentru stocarea reactivilor chimici.

Date de intrare

Fișierul de intrare react.in conține:
- pe prima linie, numărul natural N, care reprezintă numărul de reactivi;
- pe fiecare dintre următoarele N linii se află min max (două numere întregi separate printr-un spațiu); numerele de pe linia x+1 reprezintă temperatura minimă, respectiv temperatura maximă de stocare a reactivului x.

Date de ieșire

Fișierul de ieșire react.out va conține o singură linie pe care este scris numărul minim de frigidere necesar.

Restricții

- $1 \le N \le 8000$;
- $-100 \le min_x \le max_x \le 100$ (numere întregi, reprezentând grade Celsius), pentru orice x de la 1 la N;
- un frigider poate conține un număr nelimitat de reactivi.

Exemple

react.in	react.out	react.in	react.out	react.in	react.out
3 -10 10 -2 5 20 50	2	4 2 5 5 7 10 20 30 40	3	5 -10 10 10 12 -20 10 7 10 7 8	2

(Olimpiada Județeană de Informatică, 2004, clasa a IX-a)

Soluție

Intervalele de temperatură pot fi considerate segmente de dreaptă. Problema cere, de fapt, determinarea unui număr minim de puncte, astfel încât orice segment să conțină cel puțin unul dintre punctele determinate.

Vom sorta, în primul rând, intervalele de temperatură crescător după temperatura minimă și descrescător după temperatura maximă.

Deschidem un frigider și plasăm primul reactiv în acest frigider. Pentru frigiderul curent reținem temperatura minimă și temperatura maximă (intervalul de temperatură în care poate fi setat).

Parcurgem succesiv reactivii (în ordine) și pentru fiecare reactiv verificăm dacă el poate fi plasat în frigiderul curent (pentru aceasta, trebuie ca intersecția dintre intervalul de temperatură al frigiderului și intervalul de temperatură al reactivului să fie nevidă). Dacă da, plasăm acest reactiv în frigiderul curent (actualizând corespunzător

intervalul de temperatură al frigiderului). În caz contrar, deschidem un nou frigider (intervalul de temperatură al acestui frigider va fi intervalul reactivului plasat în el).

```
#include <fstream.h>
ifstream fin("react.in");
ofstream fout("react.out");
int N;
int ti[8000], tf[8000];
/*ti[i]=temperatura minima pentru reactivul i
  tf[i]=temperatura maxima pentru reactivul i */
int main ()
{int nf, poz, miny, maxx, max, i, j, icx, icy;
 //citire
 fin>>N;
 for (i=0;i<N;i++) fin>>ti[i]>>tf[i];
 fin.close();
 //sortare prin selectia maximului
 for (i=N-1; i>0; i--)
     {for (poz=i, j=0; j<i; j++)
          if (ti[j]>ti[poz] ||
              ti[j]==ti[poz] && tf[j]<tf[poz]) poz=j;
      max=ti[poz];ti[poz]=ti[i]; ti[i]=max;
      max=tf[poz];tf[poz]=tf[i]; tf[i]=max;  }
/* deschidem un frigider si plasam primul reactiv
icx=temperatura minima de functionare a frigiderului curent
icy=temperatura maxima de functionare a frigiderului curent
nf=numarul de frigidere deschise */
 nf=1; icx=ti[0];icy=tf[0];
//parcurg ceilalti reactivi in ordine
 for (i=1; i<N; i++)
     {
/*   verificam   daca   intervalul   curent   icx,   icy   se
intersecteaza
cu intervalul de temperatura al reactivului i */
      miny=icy;if (miny>tf[i]) miny=tf[i];
      maxx=icx;if (maxx<ti[i]) maxx=ti[i];
      if (maxx <= miny)
         //actualizez intervalul de temperatura
         {icx=maxx;icy=miny;}
         else //deschid un nou frigider
         {nf++;
          icx=ti[i];icy=tf[i]; }
     }
 fout<<nf<<'\n';
 fout.close(); return 0;}
```


Furnici

Una dintre cele mai importante funcții din colonia de furnici este cea de ofițer de circulație. Motivul este că în mușuroiul de furnici există un tunel foarte îngust, astfel încât prin tunel nu pot să treacă două furnici una pe lângă alta. Din fericire, în tunel există niște locuri speciale, unde furnicile pot sta și pot aștepta până trec alte furnici. Furnicile pot trece una pe lângă alta fără nici o problemă la capetele tunelului. Ofițerul de circulație cunoaște lungimea tunelului, precum și pozițiile locurilor speciale, în care furnicile pot staționa. În fiecare dimineață, ofițerul de serviciu primește o listă cu toate sosirile în tunel. Misiunea sa este de a organiza traficul astfel încât toate furnicile să poată traversa tunelul. Bineînțeles că acest lucru poate fi realizat în multe moduri, mai rapid sau mai lent. Deoarece atunci când ultima furnica va ieși din tunel ofițerul de serviciu își poate încheia programul, interesul său este de a organiza traficul astfel încât acest lucru să se întâmple cât mai repede. Orice furnică se deplasează cu 1 cm/s. În locurile speciale pot aștepta oricâte furnici, iar dimensiunile unui astfel de loc pot fi neglijate (poate fi considerat punctiform).

Cerință

Scrieți un program care să determine timpul minim necesar pentru ca toate furnicile să traverseze tunelul.

Date de intrare

Fișierul de intrare `furnici.in` conține pe prima linie două numere naturale `D` și `U`, separate printr-un spațiu; `D` reprezintă lungimea tunelului exprimată în cm, iar `U` reprezintă numărul de locuri speciale din tunel. Fiecare dintre următoarele `U` linii conține poziția unui loc special din tunel; poziția este dată prin distanța față de capătul din stânga al tunelului (exprimată în cm); pozițiile sunt date în ordinea de la stânga la dreapta. Pe următoarea linie se află un număr natural `L` care reprezintă numărul de furnici care intră prin capătul din stânga al tunelului. Fiecare dintre următoarele `L` linii conține timpul de sosire a unei furnici la capătul din stânga al tunelului. Pe următoarea linie se află un număr natural `R` care reprezintă numărul de furnici ce intră prin capătul din dreapta al tunelului. Fiecare dintre următoarele `R` linii conține timpul de sosire a unei furnici la capătul din dreapta al tunelului.

Date de ieșire

Fișierul de ieșire `furnici.out` conține o singură linie pe care se află timpul minim necesar pentru ca toate furnicile să traverseze tunelul, exprimat în secunde.

Restricții

- $1 \leq D, U, L, R \leq 10000$;
- $D > U$
- timpii de sosire sunt exprimați în secunde și numere naturale ≤ 2000000.

Exemple

furnici.in	furnici.out	furnici.in	furnici.out
5 1 2 1 3 1 2	8	10 2 4 6 2 0 4 1 0	14

(Concursul „Urmaşii lui Moisil", Iaşi, 2004)

Soluţie

Ideea de rezolvare a problemei se bazează pe următoarea observaţie: „Oricât de multe furnici ar fi, timpul maxim în care *toate* furnicile traversează tunelul depinde doar de ultimele două furnici – cea care intră ultima prin capătul din stânga şi cea ce intră ultima prin capătul din dreapta al tunelului, fiindcă atunci toate celelalte furnici au trecut deja". Deci, pentru a determina timpul minim în care toate furnicile trec prin tunel, trebuie determinat timpul minim în care cele două ultime furnici trec prin tunel. Acesta se adaugă apoi la ultimul timp, cel la care intră în tunel ultima furnică (fie în stânga, fie în dreapta) – de fapt, determinarea se face direct, în timpul calculului.

```
#include <fstream.h>
#define INFILE          "furnici.in"
#define OUTFILE         "furnici.out"
#define MaxF            10000 + 1
#define MaxLoc          10000 + 1
#define Infinit         200000000
long int l, m, n, nr_loc, maxL, maxR, rez=Infinit;
long int loc[MaxLoc];
int main(void)
{ifstream fin(INFILE);
 long int i, timp, st, dr, poz;
 fin>>l>>nr_loc; //lungime tunel si numar locuri speciale
 for (i=1; i<=nr_loc; i++) fin>>loc[i];
 fin>>m;  //cate furnici intra prin stanga
 maxL=0; //ultimul moment la care intra o furnica in stanga
 maxR=0;//ultimul moment la care intra o furnica in dreapta
 for (i=1; i<=m; i++)
     //retin timpul cand ultima furnica intra prin stanga
    {//restul nu intereseaza
      fin>>timp;
      if (timp>maxL) maxL=timp;     }
 fin>>n;           //cate furnici intra prin dreapta
 for (i=1; i<=n; i++)
```



```
     //retin timpul cand ultima furnica intra prin dreapta
    {//restul nu intereseaza
      fin>>timp; if (timp>maxR) maxR=timp;     }
 loc[0]=0;//primul loc special este 0 (intrarea din stanga)
 //ultimul loc special este intrarea din dreapta
 loc[nr_loc+1]=l;
 fin.close ();
/*nu ne intereseaza decat ultimele furnici care intra in
tunel prin stanga si prin dreapta, deci pentru aceste doua
furnici se face calculul timpului */
 for (i=0; i<=nr_loc+1; i++)
     {/* pentru fiecare dintre locurile speciale determin
      timpul maxim cand ajunge ultima furnica din stanga
      in acest loc */
      st=maxL+loc[i];
      /* si timpul maxim cand ajunge ultima furnica din
         dreapta in acest loc */
      dr=maxR-loc[i]+l;
      /* in acest loc se ajunge dupa poz secunde, poz fiind
         maximul dintre furnica stanga si furnica dreapta */
       if (st>dr) poz=st;
          else    poz=dr;
       /*  se adauga eventualul timp de asteptare pana
       ajunge cealalta furnica */
       if (loc[i]>l-loc[i]) poz=poz+loc[i];
          else              poz=poz+l-loc[i];
       /* se retine minimul pana in acest moment */
       if (poz<rez) rez=poz;
     }
 //afisare
 ofstream fout(OUTFILE);
 fout<<rez<<endl;
 fout.close();  return 0; }
```

Apariţii cifră

Se consideră numerele naturale de la 0 la N (N≤999999999) şi o cifră nenulă K. Scrieţi un program care să calculeze câte dintre aceste numere conţin în scrierea lor cel puţin o cifră K. De exemplu, pentru N=13 şi K=1, programul va afişa 5.

Soluţie

O primă idee este de a verifica pentru fiecare număr natural din intervalul [0, N] dacă el conţine sau nu în scrierea sa cifra K. Deoarece valoarea lui N este foarte mare, această soluţie este practic inutilizabilă.

O altă idee ar fi de a determina c[i]= numărul de numere naturale din intervalul [0, 10^i-1] care conţin cifra K.

c[1]= numărul de numere naturale din intervalul [0, 9] care conţin cifra K;

c[2]= numărul de numere naturale din intervalul [0, 99] care conţin cifra K;
c[3]= numărul de numere naturale din intervalul [0, 999] care conţin cifra K;
(...)
c[10]= numărul de numere naturale din intervalul [0, 999999999] care conţin cifra K.

Observăm că, indiferent de valoarea cifrei nenule K, vectorul c va avea aceleaşi valori şi că există o relaţie de recurenţă între c[i] şi c[i-1]:
$c[i]=9*c[i-1]+10^{i-1}$.

Dacă prima cifră este K, atunci toate cele 10^{i-1} numere corespund. Dacă prima cifră este diferită de K, atunci numărul de numere este c[i-1]*9.

Pentru a calcula c[i], trebuie să calculăm în prealabil puterile lui 10 (în vectorul P10).

Pentru a calcula numărul total de cifre (NT) vom determina două puteri consecutive ale lui 10 astfel încât $10^p \le N < 10^{p+1}$. Pentru aceasta vom calcula puterile lui 10, în vectorul P10 şi îl vom parcurge de la sfârşit spre început, până determinăm p. Prima cifră a lui N este N/P10[p]. Sunt trei cazuri posibile (prima cifră a lui N este egală cu K , mai mică sau mai mare decât K). Adunăm la NT numărul de numere corespunzător primei cifre a lui N, eliminăm prima cifră din N şi reluăm procedeul.

```
#include <stdio.h>
#define CIFREMAX 10
long int N, K, NT = 0;
long int C[CIFREMAX], P10[CIFREMAX];
int main(void)
{int i, p, pcifra;
 scanf("%d %d",&N,&K);
 C[0]=0; P10[0]=1;
 for (i=1; i<CIFREMAX; i++)
     {C[i]=9*C[i-1]+P10[i-1]; P10[i]=P10[i-1]*10;}
 for (p = CIFREMAX-1; p+1; p--)
     if (N>=P10[p])
         {pcifra=N/P10[p]; //determin prima cifra a lui N
          if (pcifra>K)
             {NT += P10[p]; NT += (pcifra-1)*C[p];  }
          if (pcifra==K)
             {NT+= pcifra*C[p]; NT+= N%P10[p]+1;
              break;                                 }
          if (pcifra<K)  NT+=pcifra*C[p];
          N%=P10[p]; } //eliminam prima cifra a lui N
printf("%d\n",NT); return 0; }
```

Examen de capacitate

Se citesc de la tastatură numărul de elevi care au participat la examenul de capacitate (n≤1000) şi pentru fiecare elev se citesc numele, prenumele şi cele trei note obţinute la matematică, limba română şi istorie/geografie. Să se calculeze pentru

fiecare elev media obţinută la examen şi să se afişeze numele şi prenumele elevilor în ordine alfabetică şi mediile obţinute de aceştia.

Soluţie

Vom reţine informaţiile despre elevii care au participat la examenul de capacitate într-un vector Ex, fiecare componentă a vectorului fiind o structură cu următoarele câmpuri: np (numele şi prenumele elevului), r (nota la română), m (nota la matematică), ig (nota la istorie sau geografie) şi mg (media generală).

```
#include <iostream.h>
#include <string.h>
int main ()
{ struct Elev {char np[40];
               float r, m, ig, mg;} Ex[1000], aux;
  int n, i, schimb;
  cout <<"n="; cin >> n;
  cout << "Introduceti informatiile despre elevi\n";
  for (i=0; i<n; i++)
      {cout << "Numele si prenumele: ";
       cin.get(); cin.getline(Ex[i].np,40);
       cout <<"Notele (R/M/IG): ";
       cin>>Ex[i].r>>Ex[i].m>>Ex[i].ig;}
  //calculam mediile generale
  for (i=0; i<n; i++)
       Ex[i].mg=(Ex[i].r+Ex[i].m+Ex[i].ig)/3;
  //ordonez elevii alfabetic
   do {schimb=0;
       for (i=0; i<n-1; i++)
           if (strcmp(Ex[i].np, Ex[i+1].np)>0)
              { aux=Ex[i]; Ex[i]=Ex[i+1]; Ex[i+1]=aux;
                schimb=1;}
      } while (schimb);
  //afisez
  for (i=0; i<n; i++) cout<<Ex[i].np<<' '<<Ex[i].mg<<endl;
  return 0;}
```

Observaţii

1. Pentru ordonarea alfabetică a trebuit să comparăm numele elevilor. Deoarece acestea sunt şiruri de caractere, am utilizat funcţia strcmp().
2. Înainte de a citi numele unui elev, am apelat funcţia get() pentru a extrage caracterul '\n' tastat după introducerea valorilor numerice precedente.

Apariţiile unui cuvânt

Se citeşte de la tastatură un cuvânt de maximum 20 de litere şi, începând cu linia următoare, un text terminat cu '\n', constituit din maximum 1000 de caractere. Scrieţi un program care să numere de câte ori apare cuvântul în text.

Soluție

Pentru a căuta cuvântul în text, vom utiliza funcția `strstr()`. Deoarece această funcție nu determină decât prima apariție, o vom apela repetat (după fiecare căutare cu succes, vom reapela `strstr()`, dar specificând ca adresă de început al șirului în care se caută nu începutul textului, ci adresa caracterului care urmează după apariția găsită).

```
#include <iostream.h>
#include <string.h>
int main ()
{char c[20], t[1000], *p; int nr=0;
 cout << "Cuvantul cautat: "; cin.getline(c, 20);
 cout << "Introduceti textul \n"; cin.getline(t,1000);
 p=t;                          //cu pointerul p parcurg textul
 do {
     p=strstr(p,c);                          //caut cuvantul c
     if (p)                                //am gasit o aparitie
        {nr++;                                      // o numar
         p++;}                              //caut mai departe
    } while (p);
 cout << "Nr de aparitii: "<<nr<<endl; return 0;}
```

Propoziție

Se citește de la tastatură un text scris pe mai multe linii, constituit din propoziții terminate cu `'.'`, alcătuite din maximum `500` de caractere. Sfârșitul textului este marcat de întâlnirea secvenței `"#."`. Să se determine cea mai lungă propoziție din text, în cazul în care:

a. Lungimea unei propoziții este exprimată în număr de caractere.
b. Lungimea unei propoziții este exprimată în număr de cuvinte.

Două cuvinte consecutive pot fi separate prin unul sau mai multe caractere din mulțimea `{' ',';',',',':','-',')','('}`.

Soluție

Se citește textul propoziție cu propoziție. Pentru fiecare propoziție citită se determină lungimea și apoi se compară cu lungimea maximă. Dacă lungimea propoziției este mai mare decât lungimea maximă, reținem această propoziție și lungimea sa ca fiind maximă.

```
#include <iostream.h>
#include <string.h>
int main ()
{char p[500], pmax[500];
 int lg, lgmax=0, gata=0;
 do
  {cin.getline(p,500,'.');              //citesc o propozitie
   if (!strcmp(p,"#")) gata=1;          //s-a terminat textul
      else
      {//calculez lungimea propozitiei citite
       lg=strlen(p);
       if (lg>lgmax) //compar lungimea cu lungimea maxima
          { lgmax=lg; strcpy(pmax,p);}}
      }
 while (!gata);
 cout <<"lgmax="<<lgmax<<" pmax="<<pmax; return 0;}
```


În varianta b se modifică numai modalitatea de calcul al lungimii unei propoziții. Pentru aceasta se extrag succesiv cuvintele din propoziție, utilizând funcția `strtok()` și se numără:

```
char copie[500], sep[]=" ;,:-)(", *c;
 lg=0;
 strcpy(copie,p);                          //copiez propozitia
 c=strtok(copie, sep);
 while (c)                                 //am extras un cuvant
       {lg++;                                          //il numar
        c=strtok(NULL, sep);}  //incerc sa extrag urmatorul
```

Observație

A fost necesară copierea propoziției și extragerea cuvintelor din copie, deoarece funcția `strtok()` distruge șirul din care extrage unitățile lexicale.

4.12. *Probleme propuse*

1. Care dintre următoarele secvențe de instrucțiuni determină în variabila reală `min` cel mai mic element din vectorul `a` de `n` numere reale?

(variantă Bacalaureat, 2000)

a.
```
min=0;
for (i=1; i<n; i++)
     if (min<a[i]) min=a[i];
```

b.
```
min=a[0];
for(i=1;i<n;i++)
     if (min<a[i]) min=a[i];
```

c.
```
min=a[0];
for (i=1; i<n; i++)
     if (min>a[i]) min=a[i];
```

d.
```
min=a[n-1];
for (i=1; i<n; i++)
     if(min>a[i-1])min=a[i-1];
```

2. Care dintre următoarele declarații reprezintă un tablou cu `2` linii și `4` coloane cu elemente întregi?

a.
```
longint a[2][4];
```

b.
```
unsigned int a[2][4];
```

c.
```
int a[4][2];
```

d.
```
long a[2][4];
```

3. Fie `A` o matrice cu `n` linii (numerotate de la `0` la `n-1`) şi `m` coloane (numerotate de la `0` la `m-1`) cu elemente întregi. Ce proprietate a elementelor matricei `A` testează următoarea secvenţă de instrucţiuni?

```
for (ok=1, i=0; i<m && ok; i++)
     {for  (ok=j=0; j<n &&. !ok; j++)
            if (a[j][i]) ok=1;     }
if (ok) cout<<"da"; else cout<<"nu";
```

a. Toate elementele matricei `A` sunt nenule.
b. Există o coloană a matricei `A` care conţine cel puţin un element nenul.
c. Există o linie a matricei `A` care are toate elementele nenule.
d. Toate coloanele matricei `A` conţin cel puţin un element nenul.
e. Toate coloanele matricei `A` au toate elementele nenule.
f. Există un element nenul în matricea `A`.

4. Fie `A` o matrice cu `n` linii (numerotate de la `0` la `n-1`) şi `m` coloane (numerotate de la `0` la `m-1`) cu elemente întregi. Ce proprietate a elementelor matricei `A` testează următoarea secvenţă de instrucţiuni?

```
for (ok=1,i=0; i<n && ok; i++)
    for  (j=0, k=m-1; j<k && ok; j++, k--)
          if (a[i][j]!=a[i][k]) ok=0;
if (ok) cout<<"da";  else cout<<"nu";
```

a. Orice linie a matricei `A` are elementele egale.
b. Oricare două elemente ale matricei `A` sunt diferite.
c. Există două elemente diferite în matricea `A`.
d. Oricare două elemente de pe aceeaşi linie a matricei `A` sunt diferite.
e. Oricare două elemente de pe aceeaşi linie a matricei `A`, simetrice faţă de mijlocul liniei, sunt egale.
f. Există o linie a matricei `A` în care elementele situate la egală distanţă faţă de capetele liniei sunt egale.

5. Fie `A` un vector cu `n` elemente întregi. Care dintre următoarele secvenţe de instrucţiuni inversează ordinea elementelor plasate în vector de la poziţia `st` până la poziţia `dr` (`st<dr`)?

a.
```
for (i=st,j=dr;i<j;i++,j--)
     {aux=A[st];
      A[st]=A[dr];
      A[dr]=aux;}
```

b.
```
for (; st<dr; st++, dr--)
       {aux=A[st];
        A[st]=A[dr];
        A[dr]=aux;}
```

c.
```
for(i=st;i<=(dr-st+1)/2;i++)
    {aux=A[i];
     A[i]=A[dr-st+i];
     A[dr-st+i]=aux;}
```

d.
```
for (i=st;i<=(st+dr)/2;i++)
        {aux=A[i];
         A[i]=A[dr+st-i];
         A[dr+st-i]=aux;}
```


6. Ce va afişa pe ecran următorul program?

```
#include <iostream.h>
void main()
{int n=10, i;
 int a[]={1,2,3,4,5,6,7,8,9,10};
 for (i=n/2; i; i--) a[i+n/2-1]=a[i];
 for (i=0; i<n; i++) cout<<a[i];}
```

a. Nu se va afişa nimic, deoarece se obţine eroare la compilare.
b. `1234562345`
c. `1234512345`
d. `1234554321`
e. `1234567891`
f. `1234523456`

7. Se consideră următoarea secvenţă:

```
int a[4][4] = {1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7},n=4,x=0,i;
for (i=0; i<n-1; i++) x+=a[i][n-i-2];
```

Care va fi valoarea variabilei `x` după execuţia acestei secvenţe?

a. `15` **b.** `16` **c.** `18` **d.** `20`
e. secvenţa nu se execută, deoarece se obţine eroare la compilare.

8. Unde ar trebui inserată instrucţiunea `i++;` în secvenţa următoare astfel încât în urma execuţiei secvenţei să fie eliminate toate elementele care erau iniţial pe poziţii pare în vectorul `a` cu `n` elemente întregi, plasate în vector pe poziţiile `1, 2, ..., n`?

```
i=1;
while (i<=n)
    {for (j=i+1; j<=n; j++) a[j-1]=a[j];
     n--;     }
```

a. Nicăieri, deoarece nu este necesar.
b. O dată înainte şi o dată după instrucţiunea `for`.
c. O dată înainte de instrucţiunea `for`.
d. O dată după instrucţiunea `for`.
e. De două ori după instrucţiunea `for`.
f. De două ori înainte de instrucţiunea `for`.

9. Ce va afişa pe ecran următorul program, dacă se citesc de la tastatură valorile:

```
4 0 4 2 3 1 1 2 4 3 2 1 5
#include <iostream.h>
int a[5][5];
void main()
{int nr, i,x,y,z;
 cin>>nr;
 for (i=0; i<nr; i++) {cin>>x>>y>>z; a[x][y]=z;}
 for (nr=i=0; i<5; i++) nr+=a[i][4-i];
 cout<<nr; }
```

a. `11` **b.** `5` **c.** `1` **d.** `3` **e.** `2` **f.** `8`

10. Ce va afişa pe ecran următoarea secvenţă de instrucţiuni?

```
int *p, *q;
int a=3;
float b=7.5;
*p=a; *q=b;
cout<<*p<<' '<<*q;
```

a. 3 7 **b.** 7 3 **c.** 3 7.5 **d.** 7.5 3
e. Nimic, se produce eroare la execuţia programului.

11. Să considerăm următoarele declaraţii de variabile:

```
char s[]="Era Vasile om frumos", *q=strstr(s,"Vasile");
```

a. Care este efectul următoarei atribuiri?

```
*(q+1)='y';
```

b. Dar al atribuirii următoare?

```
*q+1='x';
```

12. Care este efectul următoarei secvenţe?

```
char s[]="Era Vasile om frumos", *q=strchr(s,'a');
char *p=strchr(s,'f');
while (++q < p++)
      {*p='x';
       *q='y';
       p-=2; q+=2;  }
puts(s);
```

13. Care este efectul următoarei secvenţe?

```
char s[]="abracadabra", *q=s, *p=s+strlen(s)-1;
while (p!=s && ++*q)
      {*p=*q;
       p--;q++;  }
puts(s);
```

14. Fie `s` un şir de `n` caractere (`n` par, `n≤100`). Care dintre următoarele secvenţe inversează prima jumătate a şirului `s` cu cea de-a doua jumătate (de exemplu, şirul „`mamatata`" ar deveni „`tatamama`").

a.

```
int i, n=strlen(s), m=n/2-1;
char v;
for (i=0; i<m; i++)
{v=s[i];s[i]=s[m+i];s[m+i]=v;}
```

b.

```
int i, n=strlen(s), m=n/2;
char v;
for (i=0; i<m; i++)
{v=s[i];s[i]=s[m+i];s[m+i]=v;}
```

c.

```
char v[101];
int n=strlen(s),m=n/2;
strcat(v,s+m); s[m]='\n';
strcat(v,s); strcpy(s,v);
```

d.

```
char *v;
int n=strlen(s), m=n/2;
strcat(v,s+m); s[m]=NULL;
strcat(v,s); strcpy(s,v);
```

e.

```
int n=strlen(s),m=n/2;
strncpy (s+n,s,m);
*(s+n+m)=0;
strcpy (s,s+m);
```

f.

```
char v[201]="";
int n=strlen(s),m=n/2;
strcat(v,s+m); s[m]='\0';
strcat(v,s); strcpy(s,v);
```


15. Se consideră următorul program:

```
#include <iostream.h>
char s[100];
int uz[26], i;
int main()
{cin>>s;
 for (i=0; s[i]; i++)
    uz[s[i]-'a']++;
 for (i=0; i<26; i++)
      if (uz[i]) cout<<uz[i];
 return 0;}
```

Ce valoare ar putea fi introdusă de la tastatură pentru variabila `s` pentru a obţine pe ecran `11211`?

a. `Cabana`
b. Nici o valoare, pentru că programul este greşit din punct de vedere sintactic.
c. `viitor`
d. `osmoza`
e. `activi`
f. `baaaaabaaaaaasm`

16. Scrieţi un program care citeşte de la tastatură cele `10` numere reale ce compun vectorul `a` şi apoi cele `8` numere reale ce compun vectorul `b` şi afişează pe ecran câte dintre componentele vectorului `a` sunt strict mai mici decât toate componentele vectorului `b`. De exemplu, dacă `a=(4, 8, 1, 9, 5, 11, 3, 43, 6, 20)` şi `b=(9, 9, 6, 9, 9, 8, 6, 9)`, atunci veţi afişa `4`, deoarece valorile `4`, `1`, `5` şi `3` sunt mai mici decât toate elementele lui `b`.

(variantă Bacalaureat, 2000)

17. Scrieţi un program care citeşte de la tastatură cele `20` de componente reale ale unui vector `a` şi afişează pe ecran câte dintre valorile citite sunt mai mici decât media aritmetică a componentelor vectorului. De exemplu, dacă valorile citite sunt `3.2, 9, -2, 0, 4, -4, -0.5, 7, 1, 0.3, 14, 0, -7, 2, 2, -1, 3, 6, 0, 1`, se va afişa pe ecran valoarea `11`.

(variantă Bacalaureat, 2001)

18. Fie `a` un vector cu `n` (`n≤50`) componente de tip `int`.

a. Verificaţi dacă toate elementele vectorului `a` sunt numere prime.
b. Verificaţi dacă există în vectorul `a` un palindrom (număr care, citit de la stânga la dreapta şi de la dreapta la stânga, este acelaşi).

c. Eliminați din vector toate elementele impare.
d. Inserați în vector după fiecare element egal cu 0 un element cu valoarea 1.
e. Determinați cea mai lungă secvență formată din elemente egale existentă în vector. Afișați lungimea secvenței, poziția de început și valoarea care se repetă.
f. Determinați un element din vector care se repetă de cele mai multe ori.

19. Fie a o matrice cu n linii și m coloane (n, m≤100) cu componente de tip int.

a. Verificați dacă există o coloană în matrice cu toate elementele nule.
b. Calculați și afișați produsul elementelor nenule de pe fiecare linie a matricei.
c. Numărați câte elemente din matrice au ca vecini numai numere pare (două elemente sunt numite vecine dacă sunt adiacente pe verticală sau orizontală).

20. Fie a o matrice pătratică cu n linii (n≤100) cu componente de tip int.

a. Verificați dacă matricea este inferior triunghiulară (toate elementele de deasupra diagonalei principale sunt nule).
b. Verificați dacă matricea este simetrică față de diagonala principală.
c. Calculați suma elementelor impare de sub diagonala secundară.
d. Determinați sumele elementelor de pe chenarele concentrice ale matricei.

De exemplu, dacă n=5 și matricea este:

1	2	9	0	3
9	0	7	6	4
8	7	1	2	0
3	6	2	0	9
1	3	7	0	0

se formează trei chenare concentrice, sumele (dinspre exterior spre interior) fiind: 59, 30, 1.

21. Pe prima linie a fișierului text BAC2000.TXT se găsește o succesiune de cel puțin două și cel mult 2000 de caractere ce pot fi doar litere mari și spații. Scrieți un program care citește de la tastatură un număr natural k (0<k<2000) și stabilește dacă există în fișier vreo literă care apare de exact k ori. Programul afișează pe ecran mesajul Da în cazul în care există cel puțin o literă cu proprietatea menționată și mesajul Nu, în caz contrar. De exemplu, dacă fișierul BAC2000.TXT are următorul conținut:
EXAMEN DE BACALAUREAT LA INFORMATICA
iar de la tastatură se citește numărul 8, atunci, deoarece litera A apare de exact 8 ori, se va afișa pe ecran mesajul Da.

(variantă Bacalaureat, 2000)

22. *Melci*

Se consideră n stâlpi (numerotați de la 1 la n, n<20) de înălțimi $h_1, h_2, \ldots, h_n$ metri. La baza fiecărui stâlp se află câte un melc codificat prin numărul stâlpului. Fiecare melc i urcă ziua p_i metri și coboară noaptea q_i metri ($p_i \geq q_i$). Scrieți un program care să afișeze melcii în ordinea în care ating vârfurile stâlpilor.

De exemplu, să considerăm trei stâlpi cu înălțimile 2, 4, 5. Melcul aflat la baza stâlpului 1 urcă 1 m pe zi și coboară 0 m pe noapte. Melcul aflat la baza stâlpului 2 urcă 4 m pe zi și coboară 1 m pe noapte. Melcul aflat la baza stâlpului 3 urcă 1 m pe zi și coboară 0 m pe noapte.

Melcul 1 ajunge în vârful stâlpului după două zile. Melcul 2 ajunge în vârful stâlpului după o zi. Melcul 3 ajunge în vârful stâlpului după cinci zile. Deci ordinea în care melcii ajung în vârf este 2 1 3.

(Concurs PACO, 1997)

23. *Generare tablou*

Scrieți un program care construiește în memorie un tablou T cu n linii și n coloane, cu elemente numere naturale, astfel încât pe ultima coloană și pe ultima linie a tabloului să se afle numai elemente egale cu 1, iar oricare alt element al tabloului să fie suma dintre elementul aflat imediat sub el și elementul aflat imediat în dreapta lui.

Valoarea lui n (număr natural, n<20) se citește de la tastatură. Tabloul se va afișa pe ecran în mod standard: de la prima la ultima linie; elementele unei linii fiind scrise de la stânga la dreapta, cu spații între elementele fiecărei linii.

De exemplu, pentru n=4, se va afișa tabloul următor:

```
20  10  4  1
10  6   3  1
4   3   2  1
1   1   1  1
```

Se observă că fiecare element al tabloului (în afara elementelor 1) respectă cerința: 20=10+10, 10=6+4, 4=3+1, 10=4+6, 6=3+3, 3=2+1, 4=1+3, 3=1+2 și 2=1+1.

(variantă Bacalaureat, 2002)

24. *Generare tablou*

Scrieți un program care construiește în memorie un tablou T cu n linii și n coloane, cu elementele numere naturale, astfel încât pe diagonala principală să existe numai elemente egale cu 1, elementele de pe cele două „semidiagonale" paralele cu diagonala principală și alăturate diagonalei principale să fie toate egale cu 2, elementele de pe următoarele două „semidiagonale" să fie egale cu 3 etc. Valoarea lui n (număr natural, n<20) se citește de la tastatură.

Tabloul se va afișa pe ecran cu formatul sugerat în exemplul următor. Pentru n=4, se va afișa tabloul:

```
1 2 3 4
2 1 2 3
3 2 1 2
4 3 2 1
```

(variantă Bacalaureat, 2002)

25. *Sumă şi produs de polinoame*

Fie `P` un polinom de grad `n` şi `Q` un polinom de grad `m` (`n, m<100`) cu coeficienţi reali. Scrieţi un program care calculează suma şi produsul polinoamelor.

26. *Problema instructorului de schi*

Un instructor de schi are la dispoziţie `n` perechi de schiuri (`n≤50`) pe care trebuie să le distribuie la `n` elevi începători. Scrieţi un program care să distribuie cele `n` perechi de schiuri astfel încât suma diferenţelor absolute dintre înălţimea elevului şi lungimea schiurilor atribuite să fie minimă.

27. *Mulţimi*

Se citesc de la tastatură două mulţimi `A` şi `B` cu `n`, respectiv `m` elemente numere naturale mai mici decât `1000`. Să se determine intersecţia şi reuniunea celor două mulţimi. Implementaţi mai întâi o mulţime ca un vector în care memoraţi elementele mulţimii, apoi prin vector caracteristic. Realizaţi o comparaţie între eficienţa implementării celor două operaţii cu cele două reprezentări.

28. *Sumă*

Se consideră un număr natural `n` nenul. Să se elaboreze un program care să afişeze şi să contorizeze toate modalităţile de a scrie acest număr ca sumă de minim două numere întregi consecutive. De exemplu, pentru `n=6` veţi obţine 3 modalităţi: `1+2+3`, `0+1+2+3`, `-5-4-3-2-1+0+1+2+3+4+5+6`.

Fişierul de ieşire `sume.out` va conţine pe prima linie numărul `k` de modalităţi de a scrie numărul ca sumă de numere întregi consecutive, iar pe următoarele `k` linii câte două numere întregi `n s`, separate printr-un spaţiu, reprezentând, respectiv, numărul de termeni şi primul termen din suma corespunzătoare liniei.

29. *Zig-zag*

Din fişierul `ZigZag.in` se citesc `N` numere întregi (`N≤1000`). Afişaţi pe prima linie a fişierului `ZigZag.out` cel mai lung MZigZag care se poate construi din numerele citite. Numim MZigZag o secvenţă de numere a_1, a_2, ..., a_m astfel încât $a_1 \le a_2 \ge a_3 \le a_4 \ge a_5 \le \ldots \le a_{m-1} \ge a_m$. De exemplu:

ZigZag.in	ZigZag.out
7 7 5 0 1 4 9 3	0 9 3 7 1 5 4

30. *Relaţii*

Profesorul X a studiat structura relaţiilor dintre elevii săi. Pentru a reprezenta această structură, profesorul a numerotat elevii de la `1` la `n` (`n≤100`) şi a construit o matrice pătratică cu `n` linii, astfel: `a[i][j]=1`, dacă elevul `i` îl agreează pe elevul `j`, şi `0`, în caz contrar. În plus, a considerat că fiecare elev se agreează pe el însuşi. Această matrice a memorat-o în fişierul `Relatii.in`.

a. Determinaţi şi afişaţi pe ecran toate perechile distincte de elevi care se agreează reciproc.
b. Afişaţi elevii care nu sunt agreaţi de nimeni.
c. Afişaţi elevii care nu agreează pe nimeni.
d. Afişaţi elevii în ordinea descrescătoare a popularităţii lor. Popularitatea unui elev este egală cu numărul de elevi care îl agreează.

31. Se citeşte de la tastatură o propoziţie terminată cu `'.'`, constituită din maximum `200` de caractere. Scrieţi un program care înlocuieşte toate literele mici din propoziţie cu litere mari, fără a utiliza funcţii standard.

32. Fie `c` un cuvânt şi `p` o propoziţie. Să se cerceteze dacă propoziţia `p` conţine, în ordine, literele cuvântului `c`. De exemplu, literele cuvântului `Multe` se găsesc, în ordine, în propoziţia `Muntele te cheama`.

33. *Agendă telefonică*

Să se scrie un program care citeşte de la tastatură un număr natural `n` (`n≤200`), apoi numele şi numărul de telefon ale `n` persoane. Programul va crea o agendă telefonică în care va reţine informaţiile citite în ordinea alfabetică a numelor persoanelor. După crearea agendei telefonice, programul va citi de la tastatură numele unei persoane, apoi va căuta persoana în agendă. Dacă persoana va fi găsită, se va afişa pe ecran numărul ei de telefon, altfel se va afişa un mesaj de eroare.

34. Se citeşte de la tastatură un şir `c` de maximum `20` de litere şi, începând cu linia următoare, un text terminat cu `'\n'`, din maximum `1000` de caractere. Scrieţi un program care să şteargă din text toate apariţiile disjuncte ale şirului `c`.

35. *Multiplu*

Fie `n` un număr natural (`n≤1000`). Determinaţi un multiplu al lui `n` a cărui scriere în baza `10` conţine numai cifrele `1` şi `0`.

36. *Planificarea optimală a lucrărilor*

Să se planifice optimal (astfel încât penalizarea totală să fie minimă) executarea a `n` ($n \in \mathbf{N}^*$) lucrări, cunoscând pentru fiecare lucrare `i` termenul de predare t_i şi penalizarea p_i care se plăteşte în cazul în care lucrarea nu este finalizată la termen. Se ştie că pentru execuţia unei lucrări este necesară o unitate de timp, că nu se pot executa două lucrări în acelaşi timp şi toate lucrările trebuie să fie executate. Planificarea constă în specificarea ordinii în care trebuie să fie executate lucrările.

37. *Bac*

Oraşul A este situat pe un mal al Dunării, iar oraşul B pe celălalt mal. Legătura dintre oraşele A şi B se poate realiza numai cu bacul. Compania X a primit autorizaţie de transport fluvial, cu condiţia să asigure şi legătura cu bacul între oraşele A şi B. Primăriile celor două oraşe au stabilit de comun acord programul sosiri-plecări pe care trebuie să îl respecte compania X.

Fiind un tânăr programator dornic de afirmare, vreţi să determinaţi numărul minim de bacuri necesare companiei, astfel încât programul să poată fi respectat.

Date de intrare

Prima linie a fişierului de intrare `bac.in` conţine două numere naturale `K` şi `L`, separate printr-un singur spaţiu (`K` – durata traversării, `L` – durata de îmbarcare, egală cu durata de debarcare, exprimată în minute).

Următoarea linie conţine un număr natural `A` care reprezintă numărul de plecări din oraşul `A`. Fiecare dintre următoarele `A` linii conţine timpul unei plecări din oraşul `A`. Următoarea linie conţine un număr natural `B`, care reprezintă numărul de plecări din oraşul `B`. Fiecare dintre următoarele `B` linii conţine timpul unei plecări din oraşul `B`. Timpii de plecare sunt daţi în ordine cronologică în formatul `HH:MM` (ora şi minutul). Dacă ora sau minutul nu este un număr de două cifre, va fi precedat de un 0.

Date de ieşire

Fişierul de ieşire `bac.out` conţine o singură linie pe care se află numărul minim de bacuri necesare pentru a respecta programul.

Restricţii

- $1 \le K, L \le 1000$
- $1 \le A, B \le 1440$
- timpii sunt cuprinşi între `00:00` şi `23:59`

Exemple

bac.in	bac.out	bac.in	bac.out	bac.in	bac.out
30 15 1 08:00 1 08:00	2	90 30 2 09:00 10:00 4 08:00 11:00 14:00 20:00	3	15 30 2 08:00 12:00 1 08:45	1

38. *Numere*

Să se genereze toate şirurile formate din `n` cifre, fiecare şir generat având următoarele proprietăţi:

– conţine numai cifre din mulţimea {1, 2, 3, 4};

– oricare două cifre alăturate sunt fie ambele pare, fie ambele impare.

Numărul natural `n` ($3 \le n \le 15$) se citeşte de la tastatură. Toate soluţiile vor fi scrise una după alta, cu spaţii între soluţii, fiecare şir fiind scris fără spaţii între cifrele ce-l formează. De exemplu, pentru `n=3`, se afişează (nu neapărat în această ordine) şirurile: `111 113 131 133 311 313 331 333 222 224 242 244 422 424 442 444.`

(variantă Bacalaureat, august 2003)

39. *Adevăr*

Să considerăm un grup format din `N` persoane, numerotate de la `1` la `N`. Aceste persoane pot fi împărţite în două categorii: persoane care spun adevărul (categoria A) şi persoane care mint (categoria F). Toate persoanele din grup au fost chemate la un interviu. Aici, fiecare persoană a nominalizat câtiva membri ai grupului, precizând că aceştia aparţin uneia din cele două categorii.

Scrieţi un program care, pe baza informaţiilor obţinute la interviu, să determine care persoane spun adevărul şi care mint.

Date de intrare

Fişierul de intrare `adevar.in` conţine pe prima linie `N`, numărul de persoane din grup. Următoarele `2N` linii conţin afirmaţiile persoanelor. Mai exact, afirmaţiile persoanei `i` sunt plasate pe liniile `2i` şi `2i+1` sub forma:

`K` A_1 A_2 ... A_K

`L` B_1 B_2 ... B_L

Ceea ce înseamnă că persoana `i` a spus că persoanele A_1 A_2 ... A_K spun adevărul şi că persoanele B_1 B_2 ... B_L mint.

Fiecare persoană face cel puţin o afirmaţie despre o altă persoană şi nici o persoană nu face afirmaţii despre ea însăşi. De asemenea, nici o persoană nu va afirma că o altă persoană spune adevărul, dar şi minte în acelaşi timp.

Dacă o persoană spune adevărul, atunci toate afirmaţiile persoanei respective trebuie să fie adevărate. Dacă o persoană minte, cel puţin una dintre afirmaţiile persoanei respective trebuie să fie falsă.

Date de ieşire

Fişierul de ieşire `adevar.out` conţine `N` linii. Pe linia `i` se precizează dacă persoana `i` spune adevărul sau minte. Dacă persoana `i` spune adevărul, pe linia `i` se va afişa `0`, altfel se va afişa `1`.

Exemple

adevar.in	adevar.out	adevar.in	adevar.out
3 1 3 0 0 1 1 1 1 0	0 1 0	4 1 2 1 3 1 4 0 0 1 0 1 3	0 0 1 0

40. *Schi*

Un concurs de schi se desfăşoară astfel:

- fiecare cursă are două runde;
- în runda 1, N schiori, numerotaţi de la 1 la N, iau startul în această ordine; când primul schior din prima runda (numerotat cu 1) termină cursa, avem timpul de care schiorul a avut nevoie pentru a parcurge traseul;
- pentru fiecare schior ce urmează dispunem de diferenţa de timp dintre timpul său şi timpul schiorului care este în acel moment lider al cursei, deci cu cel mai bun timp;
- doar cei mai buni M schiori (ca timp) se califică pentru runda 2 şi ei vor porni în cursă în ordinea descrescătoare a timpului obţinut în runda 1 (ultimul schior calificat pentru runda 2 va porni primul, în timp ce primul clasat în runda 1 va porni ultimul);
- în runda 2 vom dispune iniţial de timpul total al primului schior care pleacă în cursă (suma dintre timpul realizat în runda 1 şi timpul realizat în runda 2), iar apoi pentru fiecare dintre următorii schiori vom avea diferenţa de timp dintre timpul său total şi timpul total al schiorului cu cel mai bun timp până în acel moment.

Scrieţi un program care va determina învingătorii după terminarea cursei. Presupunem că nu se poate întâmpla ca doi schiori să obţină acelaşi timp, nici după runda 1, nici după terminarea cursei.

Date de intrare/ieşire

Prima linie a fişierului de intrare `schi.in` conţine două valori întregi N şi M separate printr-un singur spaţiu, reprezentând numărul total de schiori, respectiv numărul de schiori calificaţi pentru runda 2. Linia a doua conţine timpul primului schior din runda 1, apoi următoarele N-1 linii conţin diferenţele de timp pentru ceilalţi N-1 schiori din runda 1. Următoarea linie conţine timpul primului schior după runda 2, apoi, pe următoarele M-1 linii, diferenţele de timp pentru ceilalţi M-1 schiori după runda 2.

Prima linie a fişierului de ieşire `schi.out` va conţine numărul de ordine al schiorului clasat pe primul loc (aur), linia a doua, numărul de ordine al schiorului clasat pe locul 2 (argint), iar linia a treia, numărul de ordine al schiorului clasat pe locul 3 (bronz).

Restricţii

- 3 ≤ M ≤ N ≤ 100
- timpul pentru orice cursă este între 10 şi 300 de secunde
- timpii sunt numere reale cu cel mult două cifre zecimale

Exemple

schi.in	schi.out	schi.in	schi.out
3 3 25.13 +1.14 +2.18 45.08 +2.14 +3.11	3 2 1	4 3 29.18 +2.18 +0.05 +1.13 54.22 +1.23 +1.11	4 1 3

41. *Rima*

O poezie constă din N strofe (1≤N≤5), fiecare strofă având patru linii. Fiecare linie este constituită din unul sau mai multe cuvinte separate prin spaţii şi/sau semne de punctuaţie (' : ; . ! ?). Fiecare cuvânt este format din una sau mai multe litere ale alfabetului englez. Lungimea maximă a unei linii este de 100 de caractere.

Vom spune că ultima silabă a unui cuvânt este secvenţa de caractere care începe cu ultima vocală din cuvânt până la sfârşitul cuvântului. În cazul în care cuvântul nu are vocale, ultima silabă este chiar cuvântul întreg.

Două linii rimează dacă au aceeaşi ultimă silabă, ignorând diferenţa dintre literele mari şi mici. Strofele pot avea rimă perfectă, rimă uniformă, rimă încrucişată, rimă imbricată sau rimă albă.

Strofa are rimă perfectă dacă toate cele patru linii rimează (a a a a). Dacă strofa nu are o rimă perfectă, atunci se poate spune că are:

- rimă uniformă, dacă prima şi a doua linie rimează şi, la fel, linia a treia cu a patra (a a b b);
- rimă încrucişată, dacă prima şi a treia linie rimează şi, la fel, linia a doua cu a patra (a b a b);
- rimă imbricată, dacă prima şi a patra linie rimează şi, la fel, linia a doua cu a treia (a b b a);
- rimă albă, dacă nu este nici unul dintre cazurile anterioare.

Scrieți un program care determină pentru fiecare dintre strofele poeziei tipul rimei.

Date de intrare/ieșire

Prima linie a fișierului de intrare `rima.in` conține numărul natural `N` (numărul de strofe din poezie). Următoarele `4N` linii ale fișierului de intrare conțin liniile poeziei. Fișierul de ieșire `rima.out` va conține `N` linii. Pe fiecare linie a fișierului se va scrie unul dintre cuvintele `perfecta`, `uniforma`, `incrucisata`, `imbricata`, `alba`, care vor indica tipul rimei din strofa respectivă.

Exemple

rima.in	rima.out
1 A fost odata ca-n povesti A fost ca niciodata Din rude mari imparatesti O prea frumoasa fata	incrucisata
2 Vreti sa va spun cine sunt eu Un pod cu chip de curcubeu Un curcubeu multicolor Armonizat pe placul tuturor Dar intr-o zi o fata bat-o focul Mi-a-ntors din cale pasul obosit Unde-as fi fost de nu m-as fi oprit Si nu mi-as fi vandut ei tot norocul	uniforma imbricata

5. Stiva și coada

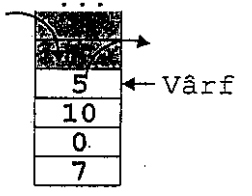
5.1. *Stiva*

Stiva este o structură de date abstractă pentru care atât operația de inserare a unui element în structură, cât și operația de extragere a unui element se realizează la un singur capăt, denumit *vârful* stivei.

Singurul element din stivă la care avem acces direct este cel de la vârf.

Operații caracteristice

Singurele operații care pot fi executate cu o stivă sunt:



- crearea unei stive vide;
- inserarea unui element în stivă (operație denumită în literatura de specialitate PUSH[30]);
- extragerea unui element din stivă (operație denumită în termeni de specialitate POP[31]);
- accesarea elementului de la vârf (operație denumită TOP[32]).

Ca să ne imaginăm mai bine funcționarea unei stive, să ne gândim cum lucrăm cu un teanc de farfurii. Când dorim să punem o farfurie în teanc, o punem deasupra, când dorim să luăm o farfurie din teanc, o luăm tot pe cea de deasupra. Motivul este lesne de înțeles: nu ne-am propus să spargem farfuriile!

Acest mod de funcționare face ca ultimul element inserat în stivă să fie primul extras. Din acest motiv, stiva este definită și ca o structură de date care funcționează după principiul LIFO (*L*ast *I*n *F*irst *O*ut – ultimul intrat primul ieșit).

30. În traducere exactă, PUSH înseamnă „a împinge". Termenul sugerează o imagine plastică: imaginându-ne stiva ca un sac, PUSH ar însemna că împingem înăuntru un element, prin capătul superior (nu prin mijloc sau pe la fundul sacului).
31. În traducere, POP înseamnă „a scoate cu zgomot (de exemplu, dopul etc.), a descărca (pistolul etc.), a ieși repede sau pe neașteptate" (Levițchi, Leon; Bantaș, Andrei – Dicționar englez-român). Imaginea este, de asemenea, sugestivă: POP este operația prin care primul element, cel de deasupra, „sare" din structură (sau din sac, ca să păstrăm analogia).
32. În traducere, TOP înseamnă „vârf".

Care este utilitatea stivelor?

În informatică, stiva joacă un rol fundamental. Pentru a înțelege mecanismele fundamentale ale programării (de exemplu, funcțiile sau recursivitatea) este necesară cunoașterea noțiunii de stivă. Pe scurt, stiva este utilă în situații în care este necesară memorarea unor informații și regăsirea acestora într-o anumită ordine, descrisă de principiul LIFO. Stiva este utilizată atunci când programul trebuie să amâne execuția unor operații, pentru a le executa ulterior, în ordinea inversă a apariției lor. Operația curentă este cea corespunzătoare vârfului stivei, în stivă fiind reținute toate informațiile necesare programului pentru a executa operațiile respective.

Cum implementăm o stivă?

Stiva este o structură de date abstractă, ce poate fi implementată în diferite moduri. De exemplu, putem implementa o stivă ca un vector în care reținem elementele stivei. Pentru ca acest vector să funcționeze ca o stivă, singurele operații permise sunt operațiile caracteristice stivei.

Pentru exemplificare, să prezentăm declarațiile necesare pentru implementarea unei stive cu elemente de tip `int`:

```
#define DimMax 100    //numarul maxim de elemente din stiva
typedef int Stiva[DimMax];
//tipul Stiva implementat ca vector
Stiva S;                    //stiva
int   vf;                   //varful stivei
```

Crearea unei stive vide

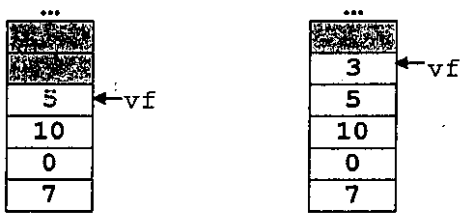
Pentru a crea o stivă vidă inițializăm vârful stivei cu `-1` (vârful stivei indică întotdeauna poziția ultimului element introdus în stivă; elementele sunt memorate în vector începând cu poziția `0`):

```
vf=-1;
```

Inserarea unui element în stivă

Pentru a insera un element `x` în stiva `S` trebuie să verificăm în primul rând dacă „avem loc", deci dacă stiva nu este plină. Dacă stiva este plină, inserarea nu se poate face, altfel vom mări vârful stivei și vom plasa la vârf noul element.

De exemplu, dacă dorim să inserăm elementul `x = 3` în stiva din figura următoare, obținem:



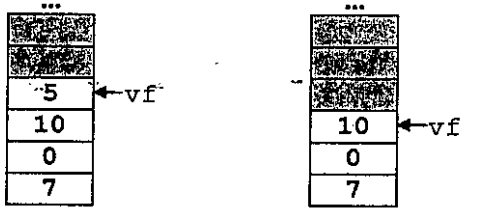
Stiva `S` înainte de inserare Stiva `S` după inserare

```
if (vf == DimMax-1)       //stiva este plina
    cout<<"Eroare - stiva este plina\n";
    else                  //inseram elementul x in stiva S
    S[++vf] = x;
```

Extragerea unui element din stivă

Pentru a extrage un element dintr-o stivă `S` trebuie să verificăm în primul rând dacă există elemente în stivă (deci dacă stiva nu este vidă). Dacă da, reținem elementul de la vârful stivei într-o variabilă (să o notăm `x`), după care micșorăm cu o unitate vârful stivei.

De exemplu, dacă extragem un element din stiva din figura următoare, obținem:



Stiva `S` înainte de extragere Stiva `S` după extragere

```
if (vf<0)                    //stiva este vida
   cout<<"Eroare - stiva este vida\n";
   else                      //extragem elementul de la varf
   x = S[vf--];
```

Accesarea elementului de la vârf

Prin modul său restrictiv de funcționare, stiva permite numai accesarea elementului de la vârf. Dacă dorim să aflăm valoarea unui alt element al stivei, ar trebui să „golim" stiva (deci să extragem succesiv elemente) până la elementul dorit.

Accesarea elementului de la vârf presupune determinarea valorii acestuia, valoare pe care noi o vom reține într-o variabilă denumită `x`.

```
x = S[vf];
```

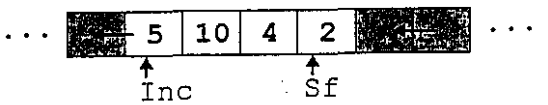
Observații

1. Dezavantajul implementării unei stive ca vector alocat static constă în faptul că indiferent de numărul de elemente existente în stivă, dimensiunea zonei de memorie alocată stivei este aceeași (`DimMax`).
2. Pentru a executa operații cu stiva alocată static este suficient să cunoaștem vârful stivei. Ca să reținem mai ușor modul de funcționare a stivei, ne imaginăm că la inserare vârful stivei urcă, iar la extragere vârful coboară.

5.2. *Coada*

Coada este o structură de date abstractă pentru care operația de inserare a unui element se realizează la un capăt, în timp ce operația de extragere a unui element se realizează la celălalt capăt.

Singurul element din coadă la care avem acces direct este cel de la început.



Operații caracteristice

Singurele operații ce pot fi executate cu o coadă sunt:

- crearea unei cozi vide;
- inserarea unui element în coadă;
- extragerea unui element din coadă;
- accesarea unui element.

Executarea acestor operații asupra unei cozi presupune cunoașterea începutului cozii (să-l notăm `Inc`) și a sfârșitului acesteia (să-l notăm `Sf`).

Modul de funcționare a unei cozi este foarte ușor de intuit: toată lumea a „stat la coadă", măcar o dată. Orice situație în care sunt mai multe cereri de acces la o resursă unică (de exemplu, mai mulți clienți și o singură vânzătoare; o singură pompă de benzină și mai multe mașini, un singur pod și mai multe capre etc.) necesită formarea unei „linii de așteptare". Dacă nu apar alte priorități, cererile sunt satisfăcute în ordinea sosirii.

Datorită faptului că întotdeauna este extras („servit") primul element din coadă, iar inserarea oricărui nou element se face la sfârșit („la coadă"), coada este definită ca o structură de date care funcționează după principiul *FIFO* (*F*irst *I*n *F*irst *O*ut – primul intrat primul ieșit).

Care este utilitatea unei cozi?

Utilitatea structurii de tip coadă reiese din modul său de funcționare – este necesară utilizarea unei cozi atunci când informațiile trebuie prelucrate exact în ordinea în care „au sosit" și ele sunt reținute în coadă până când pot fi prelucrate. În informatică, cozile sunt utilizate frecvent. De exemplu, să considerăm că avem o rețea de calculatoare și o singură imprimantă. Când utilizatorii rețelei vor da comenzi de tipărire, imprimanta nu poate răspunde tuturor comenzilor în același timp (imaginați-vă ce-ar ieși!). Prin urmare, comenzile de tipărire primite sunt înregistrate într-o coadă (*Print Queue* – coadă de tipărire). Imprimanta va tipări documentele pe rând, în ordinea în care au fost înregistrate în coadă. Un alt exemplu: pentru a mări viteza de execuție, microprocesoarele Intel utilizează o coadă de instrucțiuni în care sunt memorate instrucțiunile ce urmează a fi executate.

Cum implementăm o coadă?

Coada este o structură de date abstractă care poate fi implementată în diferite moduri. Ca și în cazul stivei, coada poate fi implementată static, reținând elementele sale într-un vector.

Să considerăm următoarele declarații care reprezintă o coadă cu elemente de tip `int` alocată static:

```
#define DimMax 100    //numarul maxim de elemente din coada
typedef int Coada[DimMax];
//tipul Coada implementat ca vector
Coada C;                        //coada
int  Inc, Sf;                   //inceputul si sfarsitul cozii
```

Elementele cozii sunt memorate în vector de la poziția `Inc` până la poziția `Sf`, deci numărul lor este `Sf-Inc+1`.

Crearea unei cozi vide

Pentru a crea o coadă vidă trebuie să inițializăm variabilele `Inc` și `Sf` astfel:

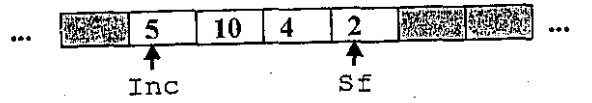
```
Inc = 0;
Sf = -1;
```

Observați că numărul de elemente din coadă este `0`, iar poziția pe care va fi plasat primul element din coadă este `0`.

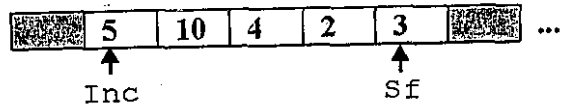
Inserarea unui element în coadă

Pentru a insera un element `x` în coada `C` trebuie să verificăm în primul rând dacă „avem loc", deci dacă variabila `Sf` nu depășește dimensiunea maximă a vectorului. Dacă inserarea se poate face, vom mări valoarea variabilei `Sf` cu o unitate (coada „crește") și vom plasa la sfârșit noul element.

De exemplu, să inserăm elementul x = 3 în coada din figura următoare:



Coada C înainte de inserare



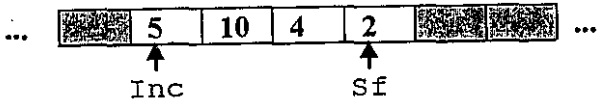
Coada C după inserare

```
if (Sf == DimMax-1)          //nu mai avem loc
   cout<<"Eroare\n";
   else                      //inseram elementul x in coada C
   C[++Sf] = x;
```

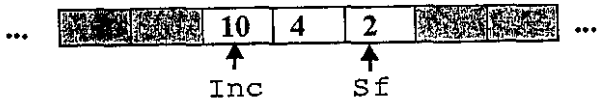
Extragerea unui element din coadă

Pentru a extrage un element dintr-o coadă C trebuie să verificăm în primul rând dacă numărul de elemente din coadă este diferit de 0 (coada nu este vidă). Dacă da, reținem elementul de la începutul cozii într-o variabilă (să o notăm x), după care mărim cu o unitate începutul cozii.

De exemplu, să extragem un element din coada din figura următoare:



Coada C înainte de extragere



Coada C după extragere

```
if (Inc > Sf)                //coada este vida
 cout<<"Eroare \n";
 else                        //extragem primul element
 x = C[Inc++];
```

Accesarea unui element

Singurul element al unei cozi care poate fi accesat direct este primul. Dacă dorim să aflăm valoarea unui alt element, va trebui să extragem succesiv elemente până la cel dorit.

Accesarea primului element are ca scop determinarea valorii acestuia, valoare pe care o vom reține într-o variabilă denumită x.

```
x = C[Inc];
```

Observație

În acest mod de alocare statică a unei cozi observați că, pe măsură ce inserăm și extragem elemente, atât sfârșitul, cât și începutul cozii „migrează" către limita maximă a vectorului. Dezavantajul este că în vector rămân poziții neutilizate (de la 0 până la Inc-1). De exemplu, ar putea să apară o situație în care coada conține un element, dar operația de inserare să nu fie posibilă pentru că Inc=Sf=DimMax-1. O soluție mai bună ar fi de a reutiliza circular spațiul de memorie alocat cozii. Pentru aceasta, următoarea poziție după poziția i poate fi:

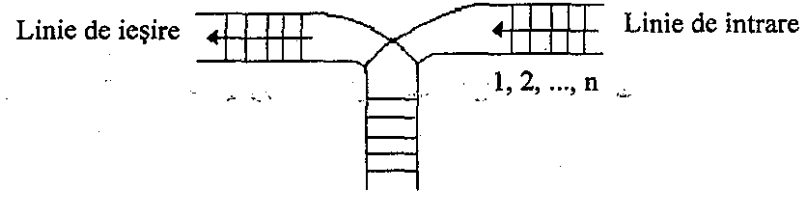
- i+1, dacă i<DimMax -1;
- 0, dacă i=DimMax-1.

Acest lucru se poate scrie concis: (i+1)%DimMax.

5.3. *Aplicații*

Depou

Să considerăm următorul sistem de manevrare a vagoanelor într-un depou:



Depou

Observați că în dreapta sunt vagoane, numerotate crescător, de la 1 la n. În orice moment, din dreapta poate intra un vagon în depou sau un vagon poate părăsi, prin stânga, depoul. Construiți un tren astfel încât vagoanele să fie numerotate în ordine inversă (de la n la 1).

Soluție

Observați că depoul funcționează ca o stivă: întotdeauna ultimul vagon intrat va fi primul vagon ieșit.

Prin urmare, va fi suficient să introducem în depou toate vagoanele, în ordinea sosirii, apoi să le extragem pe toate.

De exemplu, dacă în depou intră vagonul 1, apoi vagonul 2, apoi vagonul 3, vor ieşi în ordine vagoanele 3, 2, 1.

```
#include <iostream.h>
#define NrMaxVagoane  100
int S[NrMaxVagoane], n, Vf = 0;

int main ()
{ int i;
  cout << "Introduceti numarul de vagoane n = "; cin >> n;
  if (n > NrMaxVagoane)
     cout << "Prea multe vagoane! Nu incap in depou! \n";
     else
     {
       //introducem cele n vagoane in depou
       for (i=1; i<=n; i++) S[++Vf] = i;
       //formam trenul, extragand vagoanele din depou
       while (Vf) cout << S[Vf--] << ' ';
      }
  return 0;}
```

Exerciţii

1. Să presupunem acum că doriţi să construiţi un tren mai special: vagoanele cu număr par vor merge la un moment dat în altă direcţie faţă de cele cu număr impar. Din acest motiv, toate vagoanele cu număr par trebuie să fie plasate la începutul trenului (în ordinea crescătoare a numerelor lor), iar cele cu număr impar, la sfârşitul trenului, în ordinea descrescătoare a numerelor lor. Cum faceţi?
2. Calculaţi pentru n=3 şi pentru n=4 care sunt toate permutările de vagoane ce se pot realiza în depou. Există permutări ce nu se pot realiza în acest mod, mai exact utilizând o stivă?

Observaţie

Problema precedentă ilustrează foarte clar o proprietate esenţială a stivei: stiva inversează ordinea elementelor introduse.

Paranteze

Se citeşte pe o linie de la tastatură o succesiune de paranteze rotunde deschise şi închise, până la întâlnirea caracterului '.'. Întâlnirea unei paranteze deschise determină introducerea acesteia într-o coadă. Întâlnirea unei paranteze închise determină extragerea unui element din coadă. Verificaţi dacă parantezele din şirul citit se închid corect şi determinaţi dimensiunea maximă a cozii (numărul maxim de paranteze deschise reţinute în coadă la un moment dat).

Exemple

Şirul de intrare	Rezultate
()).	Parantezele se inchid corect. Dimensiunea maxima a cozii este 2.
((().	Parantezele nu se inchid corect. Dimensiunea maxima a cozii este 3.
()).	Parantezele nu se inchid corect. Dimensiunea maxima a cozii este 1.

Soluţie

Vom citi şirul de intrare caracter cu caracter, până când întâlnim caracterul '.'. Pentru fiecare caracter citit verificăm dacă este paranteză închisă sau deschisă.

Dacă este paranteză deschisă, o inserez în coadă, şi fiindcă dimensiunea cozii creşte, verific dacă dimensiunea curentă este mai mare decât dimensiunea maximă de până acum, caz în care reţin dimensiunea curentă ca maximă.

Dacă este paranteză închisă, trebuie să extrag o paranteză deschisă din coadă. Pentru ca extragerea să fie posibilă trebuie ca în coadă să existe paranteze închise (altfel, tragem concluzia că parantezele nu se închid corect şi nu putem face extragerea).

După citirea tuturor caracterelor din şirul de intrare, verificăm dacă în coadă mai sunt paranteze deschise. În acest caz, deducem că parantezele nu se închid corect (au fost mai multe paranteze deschise decât închise).

```
#include <iostream.h>
#define DimMaxCoada 100
int C[DimMaxCoada];
int Inceput, Sfarsit=-1, LgMax, Corect=1;
//Corect este 1 daca parantezele se inchid corect
char p;
int main ()
{
 do                    //citesc caracterele, unul cate unul
   { p=cin.get();
     if (p == '(')
        { //inserez o paranteza deschisa in coada
          C[++Sfarsit] = '(';
          if (Sfarsit-Inceput+1>LgMax)
             LgMax=Sfarsit-Inceput+1;
        }
        else
        if (p == ')')
           //verific daca pot extrage un element din coada
           if (Inceput <= Sfarsit)
              Inceput++;      //extrag un element din coada
            else Corect=0;//parantezele nu se inchid corect
   }
while (p != '.');
```

```
if (Inceput <= Sfarsit) Corect = 0;
//in coada au mai ramas paranteze deschise
if (Corect) cout << "Parantezele se inchid corect.\n";
    else cout << "Parantezele nu se inchid corect.\n";
cout << "Dimensiunea maxima a cozii este " << LgMax<<endl;
return 0;}
```

Manna-Pnueli

Calculați pentru x întreg, citit de la tastatură, valoarea funcției *Manna-Pnueli*:

$$f : Z \to Z$$

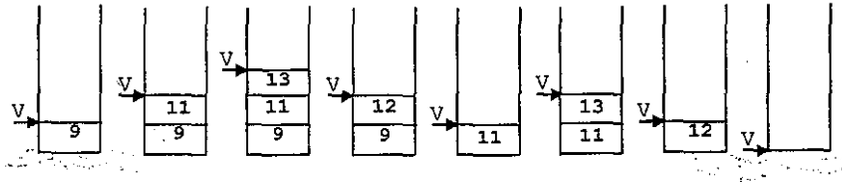
$$f(x)=\begin{cases} x-1, \text{dacă } x \geq 12 \\ f(f(x+2)), \text{dacă } x<12 \end{cases}$$

Soluție

Funcția este definită într-un mod special: în cazul în care argumentul x este mai mare sau egal cu 12, valoarea funcției se poate calcula direct. Dacă argumentul x este mai mic decât 12, valoarea funcției nu se poate calcula direct, ci trebuie calculată valoarea `f(f(x+2))`. Observați că în definiția funcției `f` intervine însăși funcția `f`. O astfel de definiție se numește în matematică *definiție recurentă*[33].

Prin urmare, dacă argumentul x este mai mic decât 12, nu putem calcula imediat valoarea funcției `f`, trebuie să amânăm acest calcul, pe care îl vom putea face numai după ce calculăm `f(f(x+2))`. Dar pentru a calcula `f(f(x+2))` trebuie să calculăm mai întâi `f(x+2)`. Evaluarea funcției presupune amânarea unor operații și executarea lor în ordine inversă. Din acest motiv este necesar să utilizăm o stivă în care să reținem argumentele funcției `f`, în ordinea în care intervin în procesul de evaluare. Când putem calcula direct valoarea funcției pentru argumentul de la vârful stivei, extragem din stivă valorile argumentelor corespunzătoare ultimelor două apeluri și inserăm valoarea calculată a funcției.

De exemplu, să urmărim pentru `x=9` cum sunt memorate în stivă argumentele pentru care trebuie evaluată funcția:



33. La informatică o vom numi ulterior recursivă.

Obținem `f(x)=11`.

```
#include <iostream.h>
#define DimMax 100
int  S[DimMax], x, y, V;
int main()
{cout << " x= "; cin >> x;
 S[V] = x; //initializam stiva cu parametrul primului apel
 while (V >= 0)
       {y = S[V];
        if (y >= 12) //calculam valoarea functiei direct
           {//plasam in stiva valoarea calculata a functiei
           if (--V >= 0) S[V] = y-1;           }
           else
           /* functia nu poate fi calculata direct,
           inseram in stiva si parametrul noului apel */
           S[++V] = y+2;
        }
cout << "f (" << x << ")= " << y-1; return 0;}
```

Caroiaj

Se consideră un caroiaj dreptunghiular cu `m` linii și `n` coloane, în care pe anumite poziții sunt plasate obstacole. În poziția inițială (x_0, y_0) se află plasat un mobil. Să se determine, pentru toate pozițiile în care mobilul poate ajunge, distanța minimă de la poziția inițială a mobilului, măsurată în deplasări elementare. Prin deplasare elementară se înțelege deplasarea mobilului cu o poziție stânga, dreapta, sus sau jos.

Soluție

Reprezentăm caroiajul ca o matrice `A`, în care marcăm obstacolele cu `-1`, iar pozițiile libere cu `-2`. La sfârșitul algoritmului, în matricea `A` vom avea:

`A[i][j]= -1`, dacă poziția `(i,j)` este obstacol;
`A[i][j]= -2`, dacă poziția `(i,j)` nu este accesibilă din (x_0, y_0);
`A[i][j]`= distanța minimă de la (x_0, y_0) la `(i,j)`, altfel.

Pentru a nu testa la fiecare pas dacă nu am ajuns la o margine a caroiajului, vom borda caroiajul cu obstacole (linia și coloana `0`, respectiv linia `n+1` și coloana `m+1` le vom utiliza ca bordură).

Ideea constă în a calcula distanța minimă până la pozițiile în care se poate ajunge dintr-o singură mutare, apoi la pozițiile la care se poate ajunge din două mutări etc. Pentru a reține pozițiile în ordinea distanței lor față de poziția inițială, vom folosi o coadă `C`, în care vom reține, pentru fiecare poziție atinsă, coordonatele și distanța minimă de la poziția inițială.

```
#include <fstream.h>
#include <iomanip.h>
#define DimMax 20
#define DimMaxCoada 400
int dx[4]={-1, 0, 1, 0}, dy[4]={0, 1, 0, -1};
//deplasarile pe linie si coloana pe directiile N,E,S,V
struct Element
  {int l,c;     //pozitia in caroiaj
   unsigned d; //distanta minima minima pana la pozitia l,c
  } C[DimMaxCoada], x, y;
int A[DimMax][DimMax], n, m, x0, y0, i, j, k, IncC, SfC;
int main()
{ifstream fin("careu.in");
 fin >> n >> m;          //citesc dimensiunile caroiajului
 fin >> x0 >> y0;        //citesc pozitia initiala
 //marchez cu -2 pozitiile libere
 for (i=1;i<=n;i++)
     for (j=1;j<=m;j++) A[i][j] = -2;
 while (fin)
       {fin >> i >> j;    //citesc coordonatele obstacolelor
        A[i][j] = -1; }  //marchez obstacolele cu -1
 fin.close();
 //bordez caroiajul cu obstacole
 for (i=1;i<=n;i++) A[i][0]=A[i][m+1]=-1;
 for (i=1;i<=m;i++) A[0][i]=A[n+1][i]=-1;
 //initializez coada
 x.l=x0; x.c=y0; x.d=0; A[x0][y0]=0;      //pozitia initiala
 C[IncC]=x;
 while (IncC <= SfC)                      //parcurg caroiajul
       {
        x = C[IncC++];          //extrag un element din coada
        //ma deplasez in cele patru directii posibile
        for (k=0; k<4; k++)
            { y.l = x.l + dx[k];  y.c = x.c + dy[k];
                    //y - urmatoarea pozitie in directia k
              if (A[y.l][y.c] == -2)
                 {
          //y- pozitie libera cu distanta minima necalculata
                    y.d = x.d + 1; A[y.l][y.c] = y.d;
                    //inserez pozitia y in coada
                    C[++SfC]= y;
                 }
            }
       }
 //afisez solutia
 ofstream fout("careu.out");
 for (i=1;i<=n;i++)
     { for (j=1;j<=m;j++) fout<<setw(3)<<A[i][j];
       fout<<endl; }
 fout.close();
 return 0;}
```


De exemplu, pentru fişierul de intrare:

```
5 5                    //dimensiunile caroiajului
3 3                    //pozitia initiala
1 2                    //obstacole
1 3
2 4
4 3
4 4
2 1
```

obţinem soluţia:

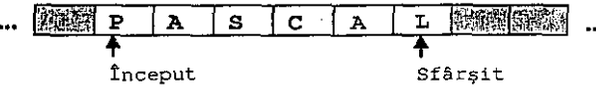
```
-2 -1 -1  5  4
-1  2  1 -1  3
 2  1  0  1  2
 3  2 -1 -1  3
 4  3  4  5  4
```

5.4. *Probleme propuse*

1. Determinaţi valoarea de adevăr a următoarelor propoziţii:
   a. Coada este un vector în care putem adăuga elemente doar la sfârşit şi putem extrage elemente doar de la început.
   b. Extragerea elementelor dintr-o stivă se realizează în ordinea inversă a introducerii lor.
   c. Coada este o structură de date care funcţionează după principiul LIFO.
   d. Afişarea elementelor dintr-o stivă se poate face prin parcurgerea vectorului în care sunt memorate elementele stivei, începând de la vârf.

2. Să considerăm o stivă iniţial vidă, care poate conţine litere. Ilustraţi efectul următoarelor operaţii: inserează `R`, inserează `A`, inserează `M`, inserează `A`, extrage un element, extrage un element, inserează `V` (exprimate în mod echivalent – `PUSH(R); PUSH(A); PUSH(M); PUSH(A); POP; POP; PUSH(V)`).

3. Să considerăm coada din figura următoare:



Ilustraţi efectul următoarelor operaţii: extrage un element, extrage un element, inserează `C`, extrage un element, inserează `U`, inserează `L`.

4. Implementaţi operaţiile elementare de lucru cu o coadă alocată static, utilizând circular spaţiul de memorie.

5. *Simulare*

Multe fenomene fizice, chimice sau biologice nu pot fi observate direct (fie condiţiile nu permit o observaţie directă, fie observarea lor implică probleme practice

sau financiare majore). În astfel de situații se recurge la o simulare – un program care imită evoluția fenomenului respectiv.

Să analizăm un exemplu simplu: simularea activității într-un cabinet stomatologic în care există un singur medic, care lucrează de la 9 la 17. Fiecare pacient care sosește la medic ia loc pe un scaun și așteaptă până când medicul devine disponibil și asistenta îl invită în cabinet. Presupunând că medicul tratează pacienții în ordinea sosirii lor, fără alte priorități (obiective sau subiective), simulați modul de desfășurare a activității! În urma acestei simulări, determinați numărul minim de scaune necesar pentru pacienții care așteaptă, durata medie de așteptare, precum și durata medie a unui tratament.

Pentru simulare, presupunem că am generat aleator datele de intrare în fișierul `'PACIENTI.IN'`. Fișierul conține câte o linie pentru fiecare pacient, pacienții fiind memorați în ordinea sosirii. Pentru fiecare pacient sosit la cabinet va fi reținut timpul la care a sosit (exprimat în număr de minute, calculat de la ora 9, când medicul stomatolog își începe programul) și durata tratamentului (exprimată, de asemenea, în număr de minute).

De exemplu, dacă fișierul de intrare conține:

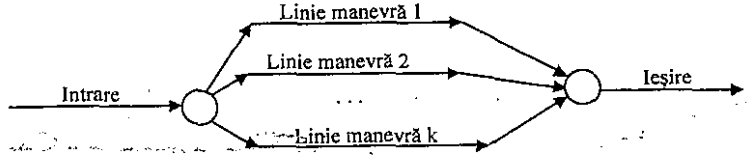
```
10 45
20 30
22 10
```

deducem că la cabinet au sosit trei pacienți:

- primul pacient a sosit la 10 minute de la începerea programului, iar tratamentul lui a durat 45 de minute;
- al doilea pacient a sosit la 20 de minute de la începerea programului, iar tratamentul lui a durat 30 de minute;
- al treilea pacient a sosit la 22 de minute de la începerea programului, tratamentul lui durând 10 minute.

6. *Trenulețe*

Într-un joc cu trenulețul, dintr-o eroare de proiectare, liniile pot fi montate doar într-un singur mod: o linie de intrare din care se desprind într-un nod `k` linii de manevră ce se vor uni în alt nod într-o linie de ieșire, ca în figură:



Jucându-se, un copil așază pe linia de intrare o garnitură de tren cu `n` vagoane numerotate de la `1` la `n`, așezate într-o ordine oarecare. Vagoanele intră câte unul pe liniile de manevră și se deplasează într-un singur sens, de la intrare spre ieșire.

Copilul dorește să obțină garnitura de tren cu care se joacă în ordinea crescătoare a numerelor vagoanelor. Ajutați-l!

Date de intrare

Din fișierul text `tren.in` se citesc de pe prima linie două numere naturale separate printr-un spațiu `n k` (`1≤n, k≤200`), unde `n` reprezintă numărul de vagoane, iar `k` numărul de linii de manevră. De pe cea de-a doua linie se citesc `n` numerele naturale distincte cuprinse între `1` și `n`, separate prin spații, reprezentând ordinea în care intră cele `n` vagoane pe linia de intrare.

Datele de ieșire

În fișierul text `tren.out` se vor afișa mutările executate sub forma:

X_1 V_1 L_1 P_1
X_2 V_2 L_2 P_2
X_3 V_3 L_3 P_3
...
X_p V_p L_p P_p

unde: X_i poate fi caracterul `M` sau `O`, cu semnificația `M` – mutarea se face de pe linia de intrare pe linie de manevră; `O` – mutarea se face de pe linie de manevră pe linia de ieșire; V_i reprezintă numărul vagonului; L_i este linia de manevră pe/de pe care se mută vagonul; P_i este poziția din linie pe care se mută vagonul. Dacă nu există soluție, fișierul de ieșire conține mesajul: `VAGOANELE NU SE POT MUTA`

Exemplu

tren.in	tren.out
5 3	M 1 1 1
1 3 5 4 2	O 1 1 1
	M 3 1 1
	M 5 2 1
	M 4 1 2
	M 2 3 1
	O 2 3 2
	O 3 1 3
	O 4 1 4
	O 5 2 5

7. *Labirint*

Se dă un labirint dreptunghiular de dimensiuni `n` x `m` (`n,m≤100`). Pozițiile din stânga sus și dreapta jos sunt marcate cu `0`, celelalte conțin unul dintre numerele `1, 2, 3, 4`. Scopul este de a parcurge labirintul din colțul stânga sus până la colțul din dreapta jos pe un drum de lungime minimă, pe direcții paralele cu laturile sale. Drumul urmat trebuie să plece din `0` în `1`, apoi din `1` în `2`, din `2` în `3`, din `3` în `4`, din `4` în `1` etc. Se poate ajunge în poziția finală din oricare poziție vecină ei.

Din fişierul de intrare `INPUT.TXT` se vor citi de pe prima linie numerele întregi `n` şi `m`, care reprezintă dimensiunile labirintului, iar de pe următoarele `n` linii câte `m` numere întregi, separate prin spaţiu, reprezentând labirintul. În fişierul de ieşire `OUTPUT.TXT` se va afişa pe prima linie numărul minim de paşi, iar pe cea de-a doua linie, un şir de caractere care reprezintă succesiunea de mişcări de pe cel mai scurt drum din labirint, folosind codificarea: `D` (jos), `U` (sus), `L` (stânga), respectiv `R` (dreapta).

Exemplu

INPUT.TXT	OUTPUT.TXT
5 4 0 1 2 3 3 2 1 4 4 1 2 1 1 4 3 2 2 3 4 0	7 RRRDDDD

(Balcaniada de Informatică, Cipru, 1996)

8. *Romeo şi Julieta*

În ultima ecranizare a celebrei piese shakespeariene, Romeo şi Julieta trăiesc într-un oraş modern, comunică prin e-mail şi chiar învaţă să programeze. Într-o secvenţă tulburătoare sunt prezentate frământările interioare ale celor doi eroi încercând fără succes să scrie un program care să determine un punct optim de întâlnire. Ei au analizat harta oraşului şi au reprezentat-o sub forma unei matrice cu `n` linii şi `m` coloane, în matrice fiind marcate cu spaţiu zonele prin care se poate trece (străzi lipsite de pericole) şi cu `X` zonele prin care nu se poate trece. De asemenea, în matrice au marcat cu `R` locul în care se află locuinţa lui Romeo, iar cu `J` locul în care se află locuinţa Julietei. Ei se pot deplasa numai prin zonele care sunt marcate cu spaţiu, din poziţia curentă în oricare dintre cele opt poziţii învecinate (pe orizontală, verticală sau diagonale). Cum lui Romeo nu îi place să aştepte şi nici să se lase aşteptat n-ar fi tocmai bine, ei au hotărât că trebuie să aleagă un punct de întâlnire în care atât Romeo, cât şi Julieta să poată ajunge în acelaşi timp, plecând de acasă. Fiindcă la întâlniri amândoi vin într-un suflet, ei estimează timpul necesar pentru a ajunge la întâlnire prin numărul de elemente din matrice care constituie drumul cel mai scurt de acasă până la punctul de întâlnire. Şi cum probabil există mai multe puncte de întâlnire posibile, ei vor să îl aleagă pe cel în care timpul necesar pentru a ajunge la punctul de întâlnire este minim.

Scrieţi un program care să determine o poziţie pe hartă la care Romeo şi Julieta pot să ajungă în acelaşi timp. Dacă există mai multe soluţii, programul trebuie să determine o soluţie pentru care timpul este minim.

Date de intrare

Fişierul de intrare `rj.in` conţine:

- pe prima linie, numerele naturale `n m`, care reprezintă numărul de linii şi, respectiv, de coloane ale matricei, separate prin spaţiu (`1<n,m≤100`); liniile şi coloanele sunt numerotate începând cu `1`;
- pe fiecare dintre următoarele `n` linii se află `m` caractere (care pot fi doar `R`, `J`, `X` sau spaţiu) reprezentând matricea.

Date de ieşire

Fişierul de ieşire `rj.out` va conţine o singură linie pe care sunt scrise trei numere naturale separate prin câte un spaţiu `tmin x y`, având semnificaţia:
- `x y` reprezintă punctul de întâlnire (`x` – numărul liniei, `y` – numărul coloanei);
- `tmin` este timpul minim în care Romeo şi Julieta ajung la punctul de întâlnire.

Exemple

rj.in	rj.out	Explicaţie
5 8 XXR XXX X X X J X X X XX XXX XXXX	4 4 4	Traseul lui Romeo poate fi: (1,3), (2,4), (3,4), (4,4). Deci timpul necesar lui Romeo pentru a ajunge de acasă la punctul de întâlnire este 4. Traseul Julietei poate fi: (3,1), (4,2), (4,3), (4,4). Timpul necesar Julietei pentru a ajunge de acasă la punctul de întâlnire este, de asemenea, 4. În plus, 4 este punctul cel mai apropiat de ei cu această proprietate.

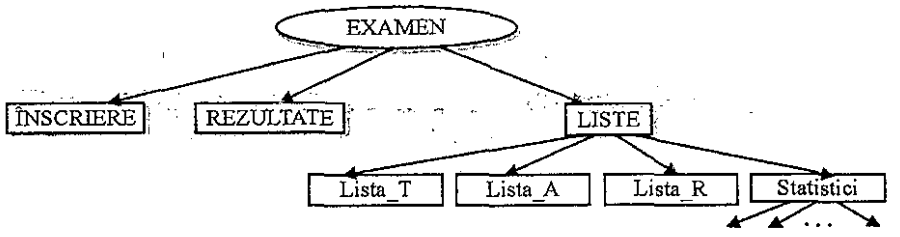
6. Funcţii

Pentru proiectarea unor aplicaţii complexe este necesară descompunerea problemei care trebuie rezolvată în subprobleme relativ independente, pentru fiecare dintre aceste subprobleme scriindu-se module de program mai simple. Cum la orice firmă se lucrează în echipă, modulele de program sunt, de obicei, implementate de mai mulţi programatori. Pentru ca programul să funcţioneze, în final, modulele de program trebuie asamblate. Prin urmare, într-o etapă prealabilă implementării se face o analiză a problemei de rezolvat, se stabilesc subproblemele şi modulele de program care trebuie să rezolve aceste subprobleme, precum şi modalităţile în care acestea trebuie să comunice între ele (interfaţa).

De asemenea, în rezolvarea problemelor apar frecvent operaţii care „se repetă". În aceste situaţii ar fi bine să scriem secvenţa de program corespunzătoare unei astfel de operaţii o singură dată şi să o folosim ori de câte ori este nevoie de ea. În practică, în dezvoltarea unei aplicaţii nu se porneşte de fiecare dată de la zero, ci se utilizează modulele de program deja implementate. Cum ar fi dacă ar trebui să scriem de fiecare dată instrucţiunile care să calculeze radicalul sau care să realizeze comunicarea cu dispozitivele periferice?

Realizarea unui program de complexitate mare impune ca o necesitate organizarea datelor şi a prelucrărilor la care acestea trebuie supuse în subprobleme şi, corespunzător acestora, sub formă de subprograme.

Să analizăm problema prelucrării datelor în cazul unui examen de admitere. Primul nivel de subprobleme ar putea fi: înscrierea candidaţilor, înregistrarea rezultatelor probelor, listarea diferitelor situaţii cerute. Dacă prima şi a doua subproblemă sunt relativ simple, cea de-a treia însă, trebuie, la rândul ei, să fie împărţită în subprobleme: obţinerea listei generale ordonate alfabetic, a listei celor admişi în ordinea descrescătoare a mediilor, a listei celor respinşi ordonate de asemenea alfabetic. În plus, la orice examen sunt solicitate multe alte situaţii statistice. Deci, schematic:



Dacă luăm în consideraţie aplicaţia descrisă mai sus, care este o aplicaţie reală, utilizată la fiecare examen de admitere în liceu sau într-o instituţie de învăţământ superior, se observă faptul că întreaga aplicaţie poate fi împărţită în module şi fiecare modul poate fi programat separat.

6.1. *Subprograme în limbajul C/C++*

În limbajele C şi C++, subprogramele sunt denumite *funcţii*.

Funcţiile reprezintă un element fundamental al limbajului C/C++. Aşa cum ştim, orice program C/C++ este constituit dintr-o succesiune de funcţii, dintre care una este funcţia principală, denumită `main()`. La lansarea în execuţie a unui program C/C++ este apelată funcţia `main()`.

Până acum, în cadrul funcţiei `main()` am apelat numai funcţii standard ale limbajului (de exemplu, `sqrt()`, `clrscr()` etc.). În acest capitol vom învăţa să descriem propriile noastre funcţii. Acestea vor fi apelate din funcţia `main()` sau dintr-o altă funcţie apelată din `main()`.

În concluzie, într-un program C/C++ toate prelucrările sunt organizate ca o ierarhie de apeluri de funcţii, baza acestei ierarhii fiind funcţia `main()`.

Pentru a putea dezvolta şi utiliza funcţii proprii este necesar să cunoaştem cum se *definesc*, cum se *declară* şi cum se *apelează* funcţiile.

6.2. *Definiţia unei funcţii*

Definiţia unei funcţii este constituită dintr-un *antet*, care conţine numele funcţiei, tipul rezultatului returnat de funcţie şi lista parametrilor funcţiei şi *un bloc de instrucţiuni*, care descrie prelucrările efectuate de funcţie.

Formatul general al definiţiei unei funcţii este:

```
tip nume(lista_parametri_formali)
{
    declaratii_variabile_locale
    instructiuni
}
```

unde:

`tip` – reprezintă tipul rezultatului returnat de funcţie;

`nume` – reprezintă numele funcţiei;

`lista_parametri_formali` – este constituită din una sau mai multe declaraţii de parametri formali, separate prin virgulă, sau poate lipsi.

O declaraţie de parametru formal specifică tipul şi numele parametrului, sub forma: `tip nume`.

Parametrii unei funcții constituie o interfață prin intermediul căreia funcția comunică cu exteriorul. Parametrii desemnează date primite de funcție din exterior, date de care depind prelucrările efectuate de funcție.

Parametrii formali sunt denumiți astfel deoarece cu ajutorul lor se descriu în mod *formal* operațiile la care vor fi supuse datele ce vor fi transmise de programul *apelant* spre subprogram.

Observații

Lista de parametri formali poate să fie vidă, dar și în acest caz parantezele trebuie să apară. În standardul ANSI al limbajului C, în cazul în care lista parametrilor formali este vidă, trebuie să se specifice explicit aceasta prin cuvântul rezervat `void`. Deși C++ acceptă o listă vidă de parametri, pentru a păstra compatibilitatea dintre C și C++ este recomandat să utilizăm `void`.

Dacă tipul rezultatului returnat de funcție nu este specificat, implicit este considerat `int`. Dacă funcția nu trebuie să întoarcă nici o valoare, se specifică drept tip al rezultatului `void`.

Exemplul 1

Definim o funcție fără parametri, denumită `scrie`. Funcția afișează un mesaj și nu returnează nici un rezultat:

```
void scrie(void)
{  cout << "Aceasta este prima mea functie C/C++ \n"; }
```

Exemplul 2

Definim o funcție denumită `scrie_suma` care primește ca parametri două numere întregi și afișează suma acestora. Funcția nu returnează nici un rezultat:

```
void scrie_suma(int a, int b)
{  cout << "a+b="<<a+b; }
```

Exemplul 3

Definim o funcție denumită `suma` care primește ca parametri un vector cu maximum `100` de componente întregi și numărul efectiv de elemente din vector. Funcția returnează suma elementelor vectorului:

```
int suma(int a[100], int n)
{ int i, s=0;
  for (i=0; i<n; s+=a[i++]);
  return s; }
```

Observați că funcția utilizează două variabile, `i` și `s`. Deoarece acestea sunt declarate în blocul funcției, ele sunt *variabile locale*.

În blocul funcției intervine instrucțiunea `return`. Această instrucțiune are formatul general:

```
return expresie;
```

Efect

Se evaluează expresia și se încheie execuția funcției, returnând în exterior (mai exact, în funcția apelantă) valoarea expresiei. În cazul în care expresia este vidă, funcția nu returnează nici o valoare.

În concluzie, revenirea dintr-o funcție se face fie după executarea ultimei instrucțiuni (instrucțiunea care precedă acolada închisă a blocului funcției) – caz în care funcția nu returnează nici o valoare, fie la întâlnirea instrucțiunii `return`.

Observație

Pot exista mai multe instrucțiuni `return` în cadrul blocului unei funcții, de exemplu:

```
int max (int a, int b)
{
  if (a>b) return a;
  return b;
}
```

Funcția `max` are doi parametri întregi, `a` și `b`. Dacă `a>b`, atunci funcția returnează valoarea parametrului `a` și execuția funcției se încheie. Dacă nu s-a încheiat execuția funcției (ceea ce echivalează cu `a<=b`), se execută următoarea instrucțiune de după `if`, prin urmare funcția returnează valoarea parametrului `b`. Deci funcția calculează maximul dintre `a` și `b`.

6.3. *Declararea funcțiilor*

Așa cum am văzut în secțiunea precedentă, definiția unei funcții informează compilatorul despre formatul funcției (nume, parametri, tip rezultat) și descrie prelucrările efectuate de funcție.

Declarația unei funcții informează compilatorul despre *existența* funcției și *formatul* acesteia. Mai exact, o declarație de funcție specifică numele funcției, tipul rezultatului returnat de funcție și parametrii acesteia:

```
tip nume(lista_parametri);
```

Observați că o declarație de funcție este constituită din antetul funcției, urmat de `;`, nu de blocul de instrucțiuni al funcției. Spre deosebire de definiție, în declarație lista parametrilor nu conține în mod obligatoriu și numele parametrilor, ci este permisă doar specificarea tipurilor parametrilor.

Exemple

```
void f1(int, int);
```

Funcția `f1()` are doi parametri de tip `int` și nu întoarce nici un rezultat.

```
long f2(int x, int y[20]);
```

Funcția `f2()` are doi parametri: unul de tip `int` și un tablou de 20 de valori de tip `int` și întoarce ca rezultat o valoare de tip `long int`. În această declarație am specificat și numele parametrilor, nu doar tipul acestora.

```
void clrscr();
```

Funcția `clrscr()` este o funcție declarată în fișierul antet `conio.h` și are rolul de a șterge fereastra text curentă și de a plasa cursorul în colțul din stânga sus al ferestrei. Este o funcție care nu are nici un parametru și nu întoarce nici un rezultat.

```
double sqrt(double);
```

Funcția `sqrt()` are un parametru de tip `double` și returnează radicalul valorii primite ca parametru. Este declarată în fișierul antet `math.h`.

```
int rand (void);
```

Funcția `rand()` este declarată în fișierul antet `stdlib.h` și returnează un număr aleator mai mic decât o constantă `RAND_MAX`, definită de asemenea în `stdlib.h`.

Declarația unei funcții se mai numește și *prototip*.

Utilitatea declarațiilor

Definiția unei funcții apare înaintea apelului funcției numai în cazuri particulare. Acest lucru nu este întotdeauna posibil (de exemplu, dacă funcția nu este definită în același fișier sursă sau este o funcție standard, care se găsește în bibliotecile limbajului în formă compilată).

Totuși, când întâlnește un apel de funcție, compilatorul trebuie să verifice validitatea acestuia. Prin urmare, este necesar ca înaintea oricărui apel de funcție să apară fie definiția funcției, fie declarația acesteia.

Un program C/C++ poate conține pentru o funcție o singură definiție, dar oricâte declarații.

Biblioteci de funcții și fișiere antet

În activitatea de programare există anumite operații care se execută frecvent (de exemplu, citiri, scrieri, sortări, operații matematice complexe, operații cu șiruri de caractere etc.). Pentru aceste operații firmele au dezvoltat funcții specifice. Aceste funcții sunt grupate pe categorii, memorate în formă compilată în colecții, denumite biblioteci. Utilizarea bibliotecilor eliberează programatorii de sarcina de a

scrie și testa un volum mare de cod, întrucât funcțiile din biblioteci[34] sunt deja testate și funcționează corect.

Pentru a utiliza funcțiile dintr-o bibliotecă într-un program, acestea trebuie declarate. Din acest motiv, programatorii creează pentru bibliotecile lor fișiere speciale, denumite fișiere antet sau fișiere *header*, care conțin declarații de funcții și, eventual, declarații de variabile externe. Ca urmare, pentru a utiliza funcțiile dintr-o bibliotecă, este suficient să includem printr-o directivă `#include` fișierul *header* corespunzător. Dacă biblioteca este una standard, plasată în subdirectorul `Include` al directorului în care este instalat mediul de programare, directiva de includere a fișierului antet corespunzător are forma:

```
#include <fisier.h>
```

Dacă biblioteca nu este una standard, ci este creată de voi, directiva de includere a fișierului antet corespunzător are forma:

```
#include "fisier.h"
```

6.4. *Apelul funcțiilor*

Apelul unei funcții se poate realiza în două moduri – printr-o *instrucțiune de apel* sau ca *operand într-o expresie*.

Instrucțiunea de apel al unei funcții are următorul format general:

```
nume(lista_parametri_actuali);
```

unde `nume` reprezintă numele funcției. Lista parametrilor actuali este formată dintr-o succesiune de expresii, separate prin virgulă.

Utilizăm instrucțiuni de apel atunci când funcția apelată nu returnează nici o valoare sau atunci când nu dorim să utilizăm valoarea returnată de funcție, ci ne interesează doar efectuarea prelucrărilor descrise de funcție.

În cazul în care dorim să utilizăm valoarea returnată de funcție ca operand într-o expresie, vom apela funcția în cadrul expresiei astfel:

```
nume(lista_parametri_actuali)
```

Observați că în acest caz lipsește caracterul `';'` care marchează sfârșitul instrucțiunii de apel.

La apelul unei funcții, *valorile* parametrilor actuali sunt atribuite, în ordine, parametrilor formali corespunzători.

34. Mediul de programare Borland C++ conține un utilitar special pentru crearea de biblioteci, denumit TLIB. Pentru a afla informații despre modul de utilizare a acestui utilitar este suficient să dați comanda: TLIB ?.

Regula fundamentală: Parametrii actuali trebuie să corespundă cu parametrii formali ca număr, ordine și tip.

Exemplu

Să considerăm următoarele declarații de variabile:

```
int x=4, y=5, m;
```

Putem atribui variabilei m maximul dintre x și y apelând funcția max definită anterior astfel:

```
m=max(x, y);
```

În acest caz, valoarea parametrului actual x înlocuiește parametrul formal a (deci a va avea valoarea 4), iar valoarea parametrului actual y înlocuiește parametrul formal b (deci b va avea valoarea 5).

Pentru a ilustra transferul parametrilor la apel să facem o analogie între un subprogram și o piesă de teatru. Autorul scrie acțiunea piesei în funcție de niște personaje. Personajele care apar în piesă sunt menționate la începutul piesei, însoțite de o scurtă descriere a „tipului" lor (de exemplu, Conu Leonida – pensionar, 60 de ani; Coana Efimița – consoarta lui, 56 de ani; Safta – slujnica lor). În mod similar, autorul unui subprogram descrie acțiunea acestuia în funcție de niște parametri formali. În antetul subprogramului sunt precizați numele și tipul parametrilor formali.

Scriind o piesă, un dramaturg descrie formal acțiunea din piesa de teatru. Abia în momentul în care piesa este pusă în scenă ea „prinde viață". Fiecare personaj din piesă este interpretat de un actor. Prin urmare, în momentul reprezentației, distribuția înlocuiește lista de personaje din piesă. Aceeași piesă poate fi jucată cu mai multe distribuții, însă acțiunea se desfășoară la fel. În mod analog, parametrii formali ai unui subprogram sunt înlocuiți la apel („reprezentația") de parametrii actuali („distribuția"). Evident, trebuie să existe o corespondență între parametrii formali și cei actuali, așa cum trebuie să existe o corespondență între personaj și actor (ar fi destul de dificil să-l distribuim pe Van Damme în rolul Coanei Efimița).

Prin urmare, piesa de teatru, cu descrierea personajelor și a acțiunilor, poate fi asemuită cu definirea unui subprogram cu lista corespunzătoare de parametri formali. Reprezentația piesei, când personajele sunt înlocuite de actori, este similară apelului, când parametrii formali sunt înlocuiți de cei actuali și acțiunile devin „reale".

Apare întrebarea: *unde sunt memorate valorile parametrilor actuali*?

Funcțiile utilizează o zonă de memorie specială, denumită *stivă* (în limba engleză – *stack*). Denumirea se datorează modului de funcționare a acestei zone de memorie – LIFO (ultimele informații memorate sunt primele informații extrase).

La apelarea unei funcții, se alocă spațiu de memorie pe stivă pentru valorile parametrilor actuali (*transfer prin valoare*). La încheierea execuției funcției, zona de memorie alocată pe stivă pentru apel este eliberată.

Ca urmare a acestui mod de funcționare, chiar dacă în blocul funcției sunt modificate valorile parametrilor, la ieșirea din funcție, deoarece memoria alocată pentru parametri se eliberează, valorile modificate *se pierd*.

Exemplu

Să considerăm următoarea funcție care intenționează să interschimbe valorile parametrilor x și y:

```
void schimba(int x, int y)
{
  int aux;
  aux=x; x=y; y=aux;
  cout <<"x="<<x<<" y="<<y<<endl;
}
```

Să presupunem că am declarat variabilele:

```
int a=3, b=4;
```

Secvența de instrucțiuni:

```
schimba(a, b);
cout <<"a="<<a<<" b="<<b<<endl;
```

afișează pe ecran:

```
x=4 y=3
a=3 b=4
```

Prin urmare, deși valorile parametrilor x și y au fost schimbate în blocul funcției, la ieșirea din funcție valorile variabilelor a și b care au fost utilizate ca parametri actuali rămân neschimbate. Acest comportament este absolut normal: valorile parametrilor funcției au fost copiate pe stivă, copiile au fost interschimbate, apoi la ieșirea din funcție zona de memorie alocată pe stivă este eliberată, iar copiile s-au pierdut. Funcția nu poate modifica valorile parametrilor actuali, deoarece nu cunoaște adresa la care ei sunt memorați.

Dacă dorim ca la ieșirea din funcție să se păstreze valorile modificate ale parametrilor, nu vom transmite ca parametri expresii ale căror valori să fie copiate pe stivă, ci vom transmite *adresele* variabilelor ale căror valori dorim să le modificăm.

Să ilustrăm, printr-o nouă analogie, modul de funcționare a celor două modalități de a transmite parametrii: să presupunem că în oraș trăiește un Jack Spintecătorul. Cineva dorește să-mi facă rău, comunicându-i lui Jack o informație despre mine.

Informația transmisă poate fi, de exemplu, adresa mea. În acest caz, sunt pierdut. Jack îmi cunoaște adresa, deci poate ajunge la mine. În mod analog, dacă un subprogram cunoaște adresa unui parametru, modificarea valorii acestui parametru în subprogram se va face la adresa cunoscută și va avea ca efect modificarea valorii parametrului actual.

Dacă însă informaţia care ajunge la Jack este fotografia mea, adică o copie, sunt salvat. Orice ar face Jack fotografiei, originalul rămâne nevătămat. Acelaşi lucru se întâmplă şi în cazul unui parametru transmis prin valoare: în stivă se va face o copie a valorii parametrului, asupra acestei copii se pot face modificări, dar originalul rămâne neschimbat. Mai mult, datorită faptului că stiva este eliberată la revenirea în programul/subprogramul apelant, copia se pierde, deci şi modificările făcute asupra ei.

6.5. *Transferul parametrilor prin referinţă*

În paragraful precedent am văzut că în limbajul C/C++ transferul parametrilor se face prin valoare. Totuşi apare frecvent necesitatea de a modifica valorile parametrilor funcţiei şi de a utiliza în afara funcţiei valorile modificate. În acest caz, este necesar să transmitem funcţiei adrese (*transfer prin referinţă*).

Există două modalităţi de a transmite parametrii prin referinţă: utilizând *pointeri* (soluţie utilizabilă atât în limbajul C, cât şi în C++) sau utilizând *referinţe* (tipul referinţă fiind disponibil numai în limbajul C++).

Utilizarea pointerilor ca parametri

Să reluăm funcţia `schimba()`, care intenţiona să interschimbe valorile parametrilor `x` şi `y`, transmiţând de data aceasta adresele parametrilor care trebuie modificaţi:

```
void schimba(int *x, int *y)
{
  int aux;
  aux=*x; *x=*y; *y=aux;
  cout <<"x="<<*x<<" y="<<*y<<endl;
}
```

De data aceasta, dacă presupunem că am declarat variabilele:

```
int a=3, b=4;
```

Secvenţa de instrucţiuni:

```
schimba(&a, &b);
cout <<"a="<<a<<" b="<<b<<endl;
```

afişează pe ecran:

```
x=4 y=3
a=4 b=3
```

Deci interschimbarea s-a executat cu succes!

Observaţi că de data aceasta parametrii funcţiei nu sunt numere întregi, ci pointeri la întregi. La apelul funcţiei am ţinut cont de tipul parametrilor şi am transmis adresele variabilelor ale căror valori trebuie să fie interschimbate.

Tipul referinţă

Tipul referinţă este o facilitate suplimentară a limbajului C++, introdusă pentru a acoperi o lacună importantă a limbajului C, şi anume transmiterea parametrilor prin referinţă.

O variabilă de tip referinţă conţine adresa unei alte variabile (asemănător unui pointer), dar nu este interpretată ca pointer, ci ca un alt nume asociat variabilei (un alias, un sinonim).

Declararea unei referinţe:

```
tip & variabila_referinta=variabila_referita;
```

Declaraţia de mai sus creează un alt nume (`variabila_referinta`) pentru variabila `variabila_referita` având tipul `tip`.

Se pot crea referinţe către variabile de orice tip. Ulterior în program puteţi utiliza atât variabila, cât şi referinţa către ea (sinonimul).

Exemplu

```
int x=10;
int &rx=x;
```

Am declarat variabila întreagă `x` şi o referinţă pentru `x`, denumită `rx`. De exemplu, dacă dorim să afişăm valoarea variabilei `x`, putem scrie fie:

```
cout<<x;
```

fie:

```
cout<<rx;
```

Efectul va fi acelaşi!

Observaţii

Valoarea atribuită unei referinţe *nu* poate fi schimbată, ea rămâne valabilă pe toată durata execuţiei blocului de program în care este declarată.

Deşi referinţa seamănă cu o variabilă, ea nu este însăşi variabila, ci doar o referire la variabila respectivă (o adresă).

Aparent, referinţele nu sunt grozav de utile. Dar să analizăm situaţia în care utilizăm referinţe drept parametri ai unei funcţii. Să reluăm funcţia `schimba()`, care intenţiona să interschimbe valorile parametrilor `x` şi `y`, transmiţând de data aceasta referinţe către parametrii care trebuie modificaţi:

```
void schimba(int &x, int &y)
{
  int aux;
  aux=x; x=y; y=aux;
  cout <<"x="<<x<<" y="<<y<<endl;
}
```

De data aceasta, dacă presupunem că am declarat variabilele a şi b şi două referinţe către acestea, ra, respectiv rb:

```
int a=3, b=4;
int & ra=a, & rb=b;
```

secvenţa de instrucţiuni:

```
schimba(ra, rb);
cout <<"a="<<a<<" b="<<b<<endl;
```

afişează pe ecran:

```
x=4 y=3
a=4 b=3
```

Deci şi în acest caz interschimbarea s-a executat cu succes!

În concluzie, în cazul utilizării referinţelor drept parametri, beneficiem de toate avantajele pointerilor (posibilitatea de a modifica valorile parametrilor), fără complicaţii de sintaxă.

Evident, orice operaţie cu referinţe se poate realiza şi cu ajutorul pointerilor, afirmaţia reciprocă nefiind valabilă.

Observaţie

Am fi putut apela funcţia schimba() şi sub forma:

```
schimba(a, b);
```

În acest caz, compilatorul va transmite automat adresele variabilelor a şi b, fără a utiliza o variabilă referinţă intermediară. Aceste adrese sunt atribuite la apel parametrilor de tip referinţă corespunzători.

S-ar putea spune că referinţele sunt *pointeri care se dereferenţiază automat*. Pentru compilatorul C++, referinţa este o adresă. Pentru programator, referinţa este doar un alt nume pentru variabilă, deci poate fi interpretată ca valoare a variabilei.

Utilizarea referinţelor este utilă nu numai atunci când un parametru trebuie să îşi modifice valoarea. Transmiterea unui parametru prin valoare presupune copierea valorii parametrului pe stivă (ceea ce uneori poate fi costisitor atât ca spaţiu, cât şi ca timp). În cazul în care variabila transmisă ca parametru este o structură de date voluminoasă, este preferabil să utilizăm o referinţă către variabilă, deoarece pe stivă se va copia doar adresa. Dacă dorim să evităm modificarea valorii unui parametru transmis prin referinţă, putem preceda declaraţia parametrului de specificatorul const, care împiedică modificarea valorii. Orice tentativă de a modifica valoarea unui parametru transmis prin referinţă a cărui declaraţie este precedată de const va fi sancţionată cu eroare la compilare.

De exemplu, dacă modificăm antetul funcţiei schimba() astfel:

```
void schimba (const int & x, int &y);
```

nu putem modifica valoarea parametrului x (atribuirea x=y va genera eroare).

Exemplu

Să urmărim, pas cu pas, pe un exemplu simplu, ce se întâmplă în fiecare moment al execuţiei unui program care conţine un apel de funcţie. Pentru identificarea acţiunilor, numerotăm liniile programului.

```
#include <iostream.h>                                          //1
int x, y = 1;                                                  //2
void F(int & a, int b)                                         //3
{                                                              //4
  cout<<"a="<<a<<"  b="<<b<<"  x="<<x<<"  y="<<y<<endl; //5
  a = 5;                                                       //6
  b = 10;                                                      //7
  cout<<"a="<<a<<"  b="<<b<<"  x="<<x<<"  y="<<y<<endl; //8
}                                                              //9
void main(void)                                                //10
{                                                              //11
  F(x, y);                                                     //12
  cout <<"x="<<x<< "  y="<<y<<endl;                            //13
}                                                              //14
```

Urmărind fiecare instrucţiune executabilă, indicăm în tabelul următor linia care se execută, valorile variabilelor globale x şi y, valorile depuse pe stivă corespunzătoare parametrilor formali a şi b şi ce anume se va afişa pe ecran. Semnul '•' semnifică o adresă, săgeata indicând variabila la care se referă adresa.

Linia	Memorie		Stivă			Ecran
	x	y	a	b	adresa revenire	
2	0	1	–	–	–	
12	0	1	•	1	adresa instrucţiunii de pe linia 13	
5	0	1	•	1	adresa instrucţiunii de pe linia 13	a=0 b=1 x=0 y=1
6	5	1	•	1	adresa instrucţiunii de pe linia 13	a=0 b=1 x=0 y=1
7	5	1	•	10	adresa instrucţiunii de pe linia 13	a=0 b=1 x=0 y=1
8	5	1	•	10	adresa instrucţiunii de pe linia 13	a=0 b=1 x=0 y=1 a=5 b=10 x=5 y=1
13	5	1	–	–	–	a=0 b=1 x=0 y=1 a=5 b=10 x=5 y=1 x=5 y=1

Observaţie

1. Nu uitaţi că numele unui tablou este un pointer către primul element al tabloului! Prin urmare, când utilizaţi vectori ca parametri, la definirea/declararea funcţiei puteţi utiliza pentru declararea parametrului una din construcţiile echivalente:

`tip *` , `tip[]` sau `tip[Max]` (unde `tip` reprezintă tipul componentelor vectorului, iar `Max` numărul maxim de elemente din vector). Nu este obligatorie specificarea numărului de elemente din vector între parantezele pătrate.

2. Când apelați o funcție care are parametri de tip tablou, trebuie să transmiteți doar numele tabloului (acesta este el însuși o adresă!).

Exemplu

Scriem un program care citește un vector `a` cu `n` elemente întregi și un vector `b` cu `m` elemente întregi, apoi afișează pe ecran cei doi vectori citiți.

Varianta C:

```
#include <stdio.h>

void  citire_vector(int *, int *);
void  afisare_vector(int, int *);

int main(void)
{ int n, a[10], m, b[10];
  citire_vector(&n, a);
  citire_vector(&m, b);
  afisare_vector(n, a);
  afisare_vector(m, b);
  return 0;            }

void  citire_vector(int * nr, int x[10])
{int i;
 printf("Nr. elemente = "); scanf("%d", nr);
 printf("Elementele vectorului: ");
 for (i=0; i < *nr; i++) scanf("%d", &x[i]);
}
void  afisare_vector(int nr, int a[10])
{int i;
 printf("Elementele vectorului sunt: ");
 for (i = 0; i < nr; i++) printf("%d ", a[i]);
 printf("\n");
}
```

În exemplul precedent am declarat două funcții: `citire_vector()` și `afisare_vector()`. Funcția `citire_vector()` are doi parametri care sunt pointeri cu tipul de bază `int` (`int *`). Observați că la definirea funcției `citire_vector()` am declarat al doilea parametru (vectorul) în mod echivalent, specificând și numărul maxim de componente din vector (`int x[10]`). Primul parametru al funcției `citire_vector()` este numărul de elemente din vector. Este declarat ca pointer, deoarece funcția `citire_vector()` trebuie să transmită în exterior valoarea citită.

La apelul funcțiilor, observați că am precizat doar numele vectorului care se citește, respectiv se afișează. Apelul este corect, numele vectorului fiind un pointer către primul element din vector.

Varianta C++ va utiliza referințe în loc pe pointeri:

```
#include <iostream.h>

void  citire_vector(int &, int *);
void  afisare_vector(int, int *);

int main()
{ int n, a[10], m, b[10];
  citire_vector (n, a);
  citire_vector (m, b);
  afisare_vector(n, a);
  afisare_vector(m, b);
  return 0;            }

void  citire_vector(int & nr, int x[10])
{
 cout << "Nr. elemente = "; cin >> nr;
 cout << "Elementele vectorului: ";
 for (int i=0; i < nr; i++) cin >> x[i];
}

void  afisare_vector(int nr, int a[10])
{
 cout << "Elementele vectorului sunt: ";
 for (int i = 0; i < nr; i++) cout << a[i]<<"  ";
 cout <<endl;
}
```

Exercițiu

Scrieți un program care citește o matrice pătratică `a` cu `n` x `n` elemente întregi și o matrice pătratică `b` cu `m` x `m` elemente întregi, apoi afișează pe ecran cele două matrice citite.

6.6. *Variabile globale și variabile locale*

Vom diferenția cele două categorii de variabile analizând următoarele caracteristici:

- poziția declarației variabilei;
- clasa de memorare (precizează zona de memorie în care este alocată variabila – în segmentul de date al programului, pe stivă etc.);
- durata de viață (precizează timpul în care variabila are alocată zonă de memorie);

- domeniul de vizibilitate (precizează zonele din program în care variabila respectivă este vizibilă, deci poate fi utilizată).

Variabilele globale

- *poziție*: sunt declarate în exteriorul oricărei funcții;
- *clasa de memorare*: li se alocă memorie în segmentul de date al programului, memoria alocată fiind automat inițializată cu 0;
- *durata de viață*: memoria rămâne alocată până la sfârșitul execuției programului;
- *domeniu de vizibilitate*: sunt vizibile din momentul declarării până la sfârșitul programului, în toate modulele acestuia, chiar și în alte fișiere sursă[35], cu excepția cazurilor de omonimie.

Variabilele locale

- *poziție*: sunt declarate în blocul unei funcții;
- *clasa de memorare*: li se alocă memorie pe stivă, memoria alocată nefiind inițializată automat;
- *durata de viață*: memoria rămâne alocată până la sfârșitul execuției blocului în care este declarată variabila;
- *domeniu de vizibilitate*: sunt vizibile numai în blocul în care sunt declarate.

Regula de omonimie

Deoarece variabilele globale au altă clasă de memorare decât variabilele locale, este permis să declarăm o variabilă locală care să aibă același nume cu al unei variabile globale. O astfel de situație este denumită *omonimie*. Ce se întâmplă în astfel de cazuri?

Regula de omonimie: În interiorul blocului în care sunt declarate, variabilele locale *au prioritate* față de variabilele globale omonime.

Prin urmare, în cazul în care în interiorul unei funcții am declarat o variabilă locală cu același nume ca al unei variabile globale, variabila locală ascunde temporar (pe parcursul execuției blocului în care este declarată) variabila globală cu același nume.

Același lucru se întâmplă și atunci când numele unui parametru formal coincide cu al unei variabile globale: pe durata execuției funcției respective, parametrul are prioritate față de variabila globală omonimă.

35. În secțiunea „Proiecte" veți vedea că un program C/C++ poate fi constituit din mai multe fișiere sursă. Presupunând că am declarat o variabilă globală `v` în fișierul `f1.cpp` și dorim să o utilizăm și în fișierul `f2.cpp` din cadrul aceluiași program, este suficient să declarăm variabila `v` și în fișierul `f2.cpp`, precedând declarația ei de specificatorul extern. Aceasta este o indicație pentru compilator că variabila respectivă există într-un alt fișier sursă.

Operatorul de rezoluție

Limbajul C++ oferă programatorilor posibilitatea de a accesa în cadrul aceluiași bloc atât variabila locală, cât și variabila globală omonimă. Acest lucru este posibil cu ajutorul unui operator unar denumit operatorul de rezoluție:

```
::variabila
```

Această expresie desemnează variabila globală, și nu variabila locală omonimă.

Exemplu

```
int x;                                    //variabila globala
void f(void)
{ int x=5;                                //variabila locala
  cout <<"x local = " <<x << endl;
  cout <<"x global = "<<::x << endl;
}
```

În exemplul precedent am definit o variabilă globală `x`, apoi în cadrul funcției `f()` am definit o variabilă locală omonimă. În cadrul funcției, când ne referim la `x`, are prioritate variabila locală. Pentru a ne referi la variabila globală `x`, am utilizat construcția `::x`. Prin urmare, funcția `f()` va afișa:

```
x local = 5
x global = 0
```

6.7. *Specificatori de clasă de memorare*

În secțiunea precedentă am precizat clasa de memorare a variabilelor locale și a variabilelor globale. Acestea sunt clasele de memorare *implicite*. Dacă dorim, putem specifica în mod explicit clasa de memorare a unei variabile (fie ea locală sau globală), precizând înaintea declarației variabilei un cuvânt-cheie care reprezintă *specificatorul de clasă de memorare*.

Specificatorul `static`

În cazul variabilelor locale, indică faptul că variabila nu este memorată pe stivă, ci în segmentul de date. Prin urmare, ea este inițializată automat cu 0 și are zonă de memorie alocată de la prima execuție a blocului în care este declarată până la sfârșitul programului. Domeniul de vizibilitate nu se modifică (variabila rămâne vizibilă numai în interiorul blocului în care este declarată).

Avantajele utilizării variabilelor locale statice sunt multiple: se alocă o singură dată, nu la fiecare apel al funcției; își păstrează valorile între două apeluri consecutive ale funcției; domeniul de vizibilitate fiind local, nu există riscuri ca valoarea variabilei să fie alterată de alte funcții ale programului, ca în cazul variabilelor globale; dacă spațiul disponibil pe stivă devine o problemă, reprezintă o soluție de alocare a unor variabile locale.

În cazul variabilelor globale, indică faptul că variabila poate fi utilizată numai în cadrul fişierului sursă în care este declarată (deci nu mai are legătură externă, nu mai poate fi utilizată în alte fişiere sursă ale programului). Cu alte cuvinte, variabila devine locală fişierului sursă în care este declarată.

Exemplu

```
void f()
{ static int x=2;
  x++;
  cout << x << endl;}
```

Funcţia `f()` conţine variabila locală statică `x`. La primul apel al funcţiei, variabila `x` este iniţializată cu valoarea 2, apoi valoarea variabilei este incrementată şi se va afişa pe ecran valoarea 3.

La al doilea apel al funcţiei `f()`, variabila `x`, fiind statică, nu mai este nici alocată, nici iniţializată, ci îşi păstrează valoarea de la apelul precedent (3). Această valoare va fi incrementată, apoi se va afişa pe ecran valoarea 4.

Deci la fiecare apel al funcţiei `f()` se va afişa o altă valoare. Dacă nu am fi declarat variabila `x` utilizând specificatorul `static`, funcţia ar fi afişat întotdeauna valoarea 3.

Specificatorul `auto`

Se poate utiliza numai pentru variabile locale şi indică faptul că variabila este alocată pe stivă, ceea ce se întâmplă în mod implicit. Din acest motiv, acest specificator nu prea este utilizat.

Specificatorul `register`

Se utilizează pentru variabile locale şi reprezintă o indicaţie pentru compilator că accesul la această variabilă trebuie să fie cât mai rapid. Prin urmare, dacă este posibil (există regiştri disponibili), variabila va fi memorată în unul dintre regiştrii microprocesorului, nu în memoria RAM. Utilizarea acestui specificator poate fi utilă, de exemplu, pentru o variabilă contor într-o instrucţiune repetitivă.

6.8. *Parametrii funcţiei* `main()`

Funcţia `main()` are un statut special, deoarece ea este automat apelată la lansarea în execuţie a programului.

Până acum am descris funcţii `main()` fără parametri. Uneori este necesar ca sistemul de operare să comunice programului anumite informaţii. Mai exact, la lansarea în execuţie a programului din sistemul de operare în mod linie de comandă, i se pot transmite programului ca informaţii diferite opţiuni sau date iniţiale, ca argumente în linia de comandă. Argumentele din linia de comandă sunt succesiuni de caractere separate prin spaţii sau caractere TAB.

Pentru funcţia `main()` se pot utiliza trei parametri:

```
main(int argc, char * argv[], char * env[]);
```

unde:

`argc` – indică numărul de argumente din linia de comandă (`argc > 0`);

`argv` – este un vector în care sunt reţinute în ordine cele `argc` argumente din linia de comandă, ca şiruri de caractere (`argv[0]` este numele programului);

`env` – este un vector în care sunt reţinute informaţii de mediu[36] ale sistemului de operare (de exemplu, prompterul sistemului, căi implicite de căutare), ultimul element al vectorului fiind `NULL`, pentru a marca sfârşitul listei.

Exemplu

Vom scrie un program care va primi ca argumente în linia de comandă numele utilizatorului şi va afişa un mesaj de salut personalizat:

```
#include <iostream.h>
int main(int argc, char * argv[])
{if (argc==1)
     {cout << "Nu am primit numele utilizatorului";
      return 1;}
 cout <<"Salut, ";
 for (int i=1; i<argc; i++) cout << argv[i] <<' ';
 cout << "! \n";
 return 0;  }
```

Să presupunem că programul executabil se numeşte `buna.exe`. Lansând executabilul în mod linie de comandă astfel:

```
>buna Dan V. Ionescu
```

programul va prelua din linia de comandă patru argumente (`argc = 4`), valorile argumentelor fiind:

```
argv[]={"buna", "Dan", "V.", Ionescu"}
```

Efectul va fi:

```
Salut, Dan V. Ionescu!
```

Exerciţiu

Scrieţi un program care primeşte ca argument în linia de comandă numele unui fişier text şi afişează pe ecran conţinutul fişierului, pagină cu pagină. Mai exact, când ecranul este plin (au fost afişate 24 de linii), afişarea se întrerupe, se afişează pe ultima linie a ecranului mesajul `"Apasati o tasta pentru a continua"`, apoi se aşteaptă ca utilizatorul să acţioneze o tastă; după ce utilizatorul acţionează o tastă, se afişează următoarea pagină ş.a.m.d.

36. Denumirea *env* provine din termenul *environment*, care înseamnă mediu.

6.9. *Caracteristici specifice funcțiilor C++*

Funcții inline

Limbajul C++ încurajează (uneori chiar impune) utilizarea funcțiilor „mici", care realizează un număr redus de operații specifice. Întrucât programatorii sunt preocupați de viteza de execuție a programelor lor, iar apelul unei funcții presupune executarea unor operații suplimentare (de exemplu, alocarea pe stivă a memoriei necesare apelului, copierea valorilor parametrilor, returnarea valorii calculate de funcție), în limbajul C++ a fost introdusă o categorie specială de funcții, denumite funcții *inline*.

Definiția unei funcții *inline* are același format general cu definiția unei funcții C uzuale, cu excepția faptului că este precedată de cuvântul rezervat `inline`.

Exemplu

```
inline max (int a, int b) {return a>b?a:b;}
```

Când compilatorul întâlnește o funcție *inline*, nu generează cod ca pentru o funcție obișnuită, ci înlocuiește apelul funcției cu secvența de instrucțiuni executată de funcția respectivă.

De obicei, pentru funcții mici, acest lucru nu duce la creșterea dimensiunii codului executabil (dacă luăm în calcul și operațiile suplimentare executate la apelul oricărei funcții). În plus, dacă o funcție *inline* este definită, dar nu este apelată niciodată, compilatorul nu generează cod pentru funcția respectivă, în timp ce pentru funcțiile uzuale, indiferent dacă funcția este sau nu apelată, codul este generat.

Funcții cu parametri impliciți

Pentru a corespunde unei varietăți cât mai mari de situații, funcțiile sunt proiectate de obicei cu un număr foarte mare de parametri, chiar dacă pentru unii parametri se utilizează frecvent la apel aceeași valoare.

Pentru ca funcțiile respective să fie mai ușor de utilizat, limbajul C++ a introdus o facilitate suplimentară, prin care programatorul poate declara în antetul funcției valori implicite.

Exemplu

```
void f(int a, int b=1, int c=2)
{cout<<a<<b<<c<<endl;}
```

Funcția `f()` are trei parametri, dintre care au doi au valori implicite (`b` are valoarea implicită `1`, iar `c` are valoarea implicită `2`).

La apelul funcției, se pot omite parametrii actuali corespunzători parametrilor formali cu valori implicite, în acest caz utilizându-se valorile implicite ale acestora. Pentru ca la apel corespondența parametru actual – parametru formal să se execute corect, întotdeauna parametrii cu valori implicite sunt ultimii declarați în lista de parametri a funcției.

De exemplu, funcția `f()` definită mai sus o putem apela astfel:

```
f(9);
```

În acest caz se utilizează valorile implicite ale parametrilor `b` și `c` și efectul este:

```
912
```

```
f(9,8);
```

În acest caz se utilizează valoarea implicită a parametrului `c` și efectul este:

```
982
```

```
f(9,8,7);
```

În acest caz nu se utilizează valorile implicite ale parametrilor și efectul este:

```
987
```

Un exemplu de funcție cu parametri impliciți pe care ați utilizat-o deja este funcția `getline()`, funcție membră a clasei `istream`:

Prototip:

```
istream & getline(signed char*, int, char = '\n');
```

Efect

Funcția citește caracterele de la intrare și le plasează în șirul descris de primul parametru. Operația se termină fie când s-au extras atâtea caractere câte specifică cel de-al doilea parametru, fie la întâlnirea marcajului de sfârșit, specificat de cel de-al treilea parametru.

Observați că cel de-al treilea parametru are ca valoare implicită caracterul *newline* (deci implicit citirea se termină la sfârșitul liniei).

Exercițiu

Descrieți o funcție de ordonare a unui vector. Funcția va primi ca parametri numărul de elemente din vector, vectorul și un parametru suplimentar care indică sensul de ordonare, cu valorile `0` – crescător, `1` – descrescător. În mod implicit, vectorul va fi ordonat crescător.

Supraîncărcarea funcțiilor

Cea mai puternică facilitate introdusă de limbajul C++ referitor la funcții constă în posibilitatea de supraîncărcare[37] a funcțiilor.

Supraîncărcarea oferă programatorului posibilitatea de a defini mai multe funcții cu același nume, dar care diferă prin numărul și/sau tipul parametrilor.

37. Termenul supraîncărcare este o traducere a termenului englezesc *overloading*. În unele cărți puteți întâlni și termenul supradefinire, cu aceeași semnificație.

Exemplu

Definim o funcție `suma()` care calculează suma a două numere întregi:

```
int suma (int a, int b)
{return a+b;}
```

Putem supraîncărca funcția `suma()`, astfel încât să calculeze suma elementelor unui vector:

```
int suma (int * a, int n)
{
  for (int i=0,s=0; i<n; i++) s+=a[i];
  return s;
}
```

Deși nu am discutat explicit despre aceasta, am folosit deja supraîncărcarea funcțiilor. De exemplu, funcția `get()`, funcție membră a clasei `istream`, este o funcție supraîncărcată, ea având următoarele prototipuri:

```
Forma 1: int get();
Forma 2: istream& get(signed char*, int len, char = '\n');
Forma 3: istream& get(unsigned char*, int len, char = '\n');
Forma 4: istream& get(unsigned char&);
Forma 5: istream& get(signed char&);
Forma 6: istream& get(streambuf&, char = '\n');
```

Compilatorul este cel care selectează la apel funcția adecvată în funcție de lista parametrilor actuali. În primul rând, se caută o corespondență exactă (o funcție pentru care tipurile parametrilor formali să coincidă cu cele ale parametrilor actuali ai apelului respectiv). Dacă nu este găsită o corespondență exactă, se fac conversii de tip implicite (de exemplu, tipul `float` este convertit la `double`; `char`, `unsigned char`, `unsigned int` sunt convertite la `int`).

Utilizarea funcțiilor supraîncărcate reprezintă unul dintre cele mai importante mecanisme ale programării orientate pe obiect.

Observație

Un caz particular de funcții sunt operatorii. Cei mai frecvent utilizați operatori supraîncărcați sunt operatorii de deplasare (<< și >>). În clasa `istream`, operatorul >> a fost supraîncărcat pentru a putea fi utilizat ca operator de citire, iar în cadrul clasei `ostream`, a fost supraîncărcat operatorul << pentru a putea fi utilizat ca operator de scriere. Când vom studia programarea orientată pe obiect, vom învăța despre modul de supraîncărcare a operatorilor.

Funcții șablon

În activitatea de programare apar frecvent situații în care trebuie să realizăm aceleași operații, dar pentru date diferite. De exemplu, trebuie să calculăm suma

unui șir de numere întregi, dar și suma unui șir de numere reale. Prin utilizarea funcțiilor supraîncărcate am putea defini două funcții de forma:

```
float suma (float *, int);
int   suma (int *, int);
```

Din păcate, chiar dacă instrucțiunile din blocul funcției coincid, ele trebuie scrise pentru fiecare funcție în parte.

Pentru astfel de situații, când funcțiile execută aceleași operații, dar pentru date de tipuri diferite, limbajul C++ oferă programatorilor posibilitatea de a defini funcții șablon[38].

O funcție șablon este un model cu ajutorul căruia compilatorul C++ generează funcții pentru diferite tipuri de date.

Definiția unei funcții șablon este precedată de cuvântul rezervat `template`, urmat de o listă de tipuri (un fel de parametri formali ai șablonului) încadrată între <>, fiecare tip specificat ca parametru fiind precedat de cuvântul rezervat `class`.

Exemplu

Definim o funcție șablon care să calculeze suma unui șir de numere, indiferent de tipul acestora:

```
template <class T>
T suma (T * a, int n)
{
  T s=0;
  for (int i=0; i<n; i++) s+=a[i];
  return s;
}
```

Observați că șablonul depinde de tipul `T`. La apelul unei funcții `suma()`, compilatorul va substitui `T` cu tipul corespunzător.

```
int   a[] = {1, 2, 3, 4, 5};
float b[] = {2.5, 2.3, 3.5, 1.2};
cout << suma(b, 4) << endl;
cout << suma(a, 5) << endl;
```

La primul apel al funcției `suma()`, `T` va fi înlocuit cu tipul `float`, la al doilea apel, `T` va fi înlocuit cu tipul `int`.

Funcțiile șablon se plasează, de obicei, într-un fișier antet care va fi inclus în program. Astfel, programele devin mai scurte, mai ușor de citit și de înțeles.

38. Termenul șablon este o traducere a termenului englezesc *template*.

Exerciții

1. Creați o funcție șablon care să ordoneze un vector, indiferent de tipul componentelor acestuia.
2. Creați o funcție șablon care să determine elementul maxim dintr-un vector, indiferent de tipul componentelor acestuia.
3. Creați un fișier antet care să conțină funcții șablon pentru operațiile cu stive și cozi implementate static.

6.10. *Proiecte*

O aplicație C/C++ nu este constituită în mod obligatoriu dintr-un singur fișier sursă. Definițiile funcțiilor din care este constituit programul se pot găsi în mai multe fișiere. Acest lucru permite elaborarea aplicațiilor în echipă (fiecare membru al echipei va salva funcțiile elaborate de el în fișiere sursă proprii), precum și reutilizarea anumitor funcții (dacă dorim ca într-o altă aplicație să utilizăm anumite funcții pe care le-am elaborat deja și se găsesc într-un fișier sursă, este suficient să includem în program fișierul sursă respectiv).

În final, după elaborarea tuturor fișierelor sursă, le vom asambla, prin crearea unui *proiect*.

În mediul de programare Borland C++ 3.1, *crearea* unui nou proiect sau *deschiderea* unui proiect existent se realizează cu ajutorul opțiunii *Open Project* din meniul *Project* al mediului de programare. Se va deschide o fereastră în care este vizualizată lista fișierelor sursă din care este constituit proiectul. Pentru *adăugarea* unui fișier sursă la proiectul deschis, se selectează opțiunea *Add Item* din meniul *Project*. *Eliminarea* din proiect a unui fișier sursă se realizează cu ajutorul opțiunii *Delete Item* din meniul *Project*. *Închiderea* unui proiect se realizează cu ajutorul opțiunii *Close Project* din meniul *Project*.

Exemplu

Să construim un proiect simplu, format din două fișiere sursă. Primul fișier sursă se numește `Radical.cpp` și conține definițiile a două funcții proprii de determinare a radicalului: `R2()` – pentru radicalul de ordin 2 și `R3()` – pentru radicalul de ordin 3.

Pentru calculul valorii radicalului de ordin doi vom folosi o metodă cunoscută sub numele de *algoritmul lui Hero*. Această metodă folosește pentru extragerea radicalului de ordin doi din numărul real pozitiv *a*, formula de recurență:

$$x_n = \frac{1}{2}(x_{n-1} + \frac{a}{x_{n-1}}),\ n=1, 2, \ldots$$

Demonstrația matematică arată că, după un anumit număr de pași[39], valoarea calculată a lui x_n dă o aproximație bună pentru $\sqrt{a}$.

Pentru radicalul de ordin trei vom folosi o formulă de recurență similară:

$$x_n = \frac{1}{3}(2x_{n-1} + \frac{a}{x_{n-1}^2}),\ n=1, 2, \ldots$$

Pentru implementarea relațiilor de recurență vom folosi două variabile, `x0` și `x1`. În `x1` calculăm termenul curent, iar în `x0` reținem termenul precedent.

Vom considera, de asemenea, că după zece pași se obține o aproximație suficient de bună pentru calculul valorilor respective cu șase zecimale exacte. Numărul de pași poate fi determinat exact dacă se va considera drept condiție de oprire:

$$|x1-x0|<10^{-6}$$

ceea ce exprimă matematic faptul că primele șase zecimale ale celor două aproximații succesive sunt identice. În tabelul următor se poate observa faptul că în cazul calculării radicalului de ordin 2 s-a ajuns la șase zecimale exacte după cinci pași, iar în cazul calculării radicalului de ordin 3 s-a ajuns la șase zecimale exacte după șapte pași:

Pași	x0	x1
1	8.000000	4.500000
2	4.500000	3.138889
3	3.138889	2.843781
4	2.843781	2.828469
5	2.828469	2.828427
6	2.828427	2.828427
7	2.828427	2.828427
8	2.828427	2.828427
9	2.828427	2.828427
10	2.828427	2.828427

Pași	x0	x1
1	8.000000	5.375000
2	5.375000	3.675635
3	3.675635	2.647804
4	2.647804	2.145565
5	2.145565	2.009652
6	2.009652	2.000046
7	2.000046	2.000000
8	2.000000	2.000000
9	2.000000	2.000000
10	2.000000	2.000000

```
double R2(double a)
{double x0 = a, x1;
 for (int i = 1; i<=10; i++)
     { x1 = 0.5 * (x0 + a / x0);
       x0 = x1; }
 return x1;}
double R3(double a)
{
 double x0 = a, x1;
 for (int i = 1; i<=10; i++)
      { x1 = (1/3.0)*(2*x0 + a/(x0*x0));
        x0 = x1;    }
 return x1;
}
```

39. Vezi tabelul următor.

Al doilea fişier, denumit `F.cpp`, este constituit numai din funcţia `main()`, în cadrul căreia vom apela cele două funcţii:

```
#include <iostream.h>#include "radical.h"int main ()
{ double x;
  cout << "x= "; cin >> x;
  cout << "Radical de ordin 2= " << R2(x) << endl;
  cout << "Radical de ordin 3= " << R3(x) << endl;
  return 0;}
```

Observaţi că în fişierul `F.cpp` includem fişierul antet `Radical.h`, care conţine prototipurile celor două funcţii de determinare a radicalului:

```
double R2(double);
double R3(double);
```

După crearea celor două fişiere sursă şi a fişierului antet:

1. Selectăm opţiunea *Open Project* din meniul *Project*. Va apărea o fereastră de dialog în care vom introduce (în câmpul *Open Project File*) numele proiectului pe care îl creăm.
2. Selectăm opţiunea *Add Item* din meniul *Project*. Va apărea o fereastră de dialog, în cadrul căreia selectăm fişierul `F.cpp`, acţionăm butonul de comandă *Add*, apoi selectăm fişierul `Radical.cpp` şi acţionăm (în final) butonul de comandă *Done*.
3. Selectăm opţiunea *Make* sau *Build All* din meniul *Compile*, pentru a crea un program executabil. Programul executabil va fi un fişier cu numele proiectului şi extensia `.exe`.
4. Lansăm în execuţie programul, acţionând combinaţia de taste *CTRL+F9* sau selectând opţiunea *Run* din meniul *Run*.

6.11. *Aplicaţii*

Expresie

Să se scrie un program care să calculeze valoarea expresiei:

$$E=\begin{cases}\max(\min(|a|-3b, c-2|d|), \min(|a|+3b, c+2|d|)), & \text{daca } a+b>c-d\\ \min(\max(|a|-3b, c-2|d|), \max(|a|+3b, c+2|d|)), & \text{daca } a+b\le c-d\end{cases}$$

Soluţie

Studiind expresia, observăm că pentru a calcula valoarea expresiei se utilizează trei funcţii: calculul valorii absolute (`abs`), determinarea minimului dintre două valori (`min`), determinarea maximului dintre două valori (`max`). Pentru fiecare dintre ele scriem câte o funcţie care returnează exact valoarea cerută. După definirea celor trei funcţii, calculul valorii expresiei devine extrem de simplu: se transcrie în limbaj expresia utilizând funcţiile definite:


```
#include <iostream.h>

float min(float x, float y);
float max(float x, float y);
float abs(float x);
int main ()
{ float a, b, c, d, e;
  cout << "a, b, c, d = "; cin >> a >> b >> c >> d;
  cout << (a+b > c-d)? max(min(abs(a)-3*b, c-2*abs(d)),
          min(abs(a)+3*b, c+2*abs(d))): min(max(abs(a)-3*b,
          c-2*abs(d)), max(abs(a)+3*b, c+2*abs(d)));
  return 0;}
float min(float x, float y)        //x,y - parametri formali
{
  if (y < x) return y;
  return x;
}
float max(float x, float y)
{
  if (y > x) return y;
  return x;
}
float abs(float x)
{
  if (x<0) return -x;
  return x;
}
```

Find & Replace

Fiind date un fişier text şi două şiruri de caractere (`s1` şi `s2`), să se realizeze înlocuirea fiecărei apariţii a şirului `s1` în fişierul dat cu şirul `s2`; fişierul iniţial va rămâne nemodificat, rezultatul prelucrării obţinându-se într-un alt fişier text.

Soluţie

Ideea este de a consulta fişierul sursă caracter cu caracter. Caracterele citite sunt plasate într-un şir auxiliar denumit `buffer`, de lungimea şirului `s1`. Dacă la un moment dat şirul `buffer` este egal cu şirul `s1`, în fişierul modificat se va scrie `s2`, iar din fişierul sursă se citesc caractere pentru reumplerea şirului `buffer`. Dacă şirul `buffer` nu este egal cu `s1`, se extrage primul caracter din `buffer`, se scrie în fişierul modificat, apoi se citeşte un nou caracter din fişierul sursă, care se introduce în `buffer`, la sfârşit.

```
#include <string.h>
#include <fstream.h>
ifstream f;
ofstream g;
char c, s1[200], s2[200], buffer[200], ns[100], nd[100];
int lbuffer, ls1, ls2;

void Citire()
{cout <<"Numele fisierului sursa:"; cin.getline(ns, 100);
cout<<"Numele fisierului modificat:";  cin.getline(nd,100);
cout << "Sirul cautat : "; cin.getline(s1, 200);
cout <<"Sirul cu care se inlocuieste: ";
cin.getline(s2,200); }

void UmpleBuffer()
{ int i = 1;
  char c[2];
  c[1] = buffer[0] = NULL;
  while ((i++ <= lbuffer) && !f.eof())
        { f.get(c[0]);
          strcat(buffer,c);  }
}

void Initializare()
{ f.open(ns); g.open(nd);
  lbuffer = ls1 = strlen(s1); ls2 = strlen(s2);
  //lbuffer - lungimea bufferului de intrare a caracterelor
  UmpleBuffer(); }

void Stanga(char * b, char & stanga, int lg)
{ //muta caracterele din buffer cu o pozitie la stanga
  stanga = b[0];
  for (int i = 1; i<lg; i++) b[i-1] = b[i]; }
int main()
{ Citire();
  Initializare();
  while (!f.eof())
       if (!strcmp(buffer,s1))
      //daca am gasit cuvantul s1, scriem s2 in destinatie}
           { g << s2;
             UmpleBuffer(); }          //reumplem bufferul
           else
          //scriem la destinatie primul caracter din buffer
           { Stanga(buffer, c, lbuffer);
             g << c;
             if (!f.eof())
                f.get(buffer[lbuffer-1]);
                              //citim un caracter in buffer
             else             //nu mai putem umple bufferul
             strncpy(buffer, buffer, lbuffer-1);
           }
```



```
  g << buffer;                //scriem bufferul la destinatie
  f.close(); g.close();
  return 0; }
```

Cel mai mare divizor comun cu descompunere în factori primi

În capitolul precedent am studiat un algoritm pentru calculul celui mai mare divizor comun a două numere naturale (c.m.m.d.c.) utilizând algoritmul lui Euclid. La matematică, în clasele gimnaziale, am învățat și altă metodă de determinare a celui mai mare divizor comun: c.m.m.d.c. a două numere naturale se calculează astfel:

- se descompun cele două numere în factori primi;
- se calculează produsul factorilor primi comuni la puterea cea mai mică.

Să se scrie un program care calculează c.m.m.d.c. în acest mod.

Soluție

Vom considera cele două numere de tip `long int`. Avem nevoie, în primul rând, de doi vectori în care se vor reține factorii primi ai celor două numere și de încă doi vectori în care se vor reține puterile acestor factori. Trebuie, de asemenea, să determinăm numărul de divizori primi ai fiecărui număr.

Funcția `Descompune()` realizează descompunerea în factori primi a parametrului de intrare `X`, reținând factorii în vectorul `DivX`, iar puterea corespunzătoare fiecărui factor în vectorul `OrdX`. Funcția calculează și numărul de factori determinați, prin parametrul `i`.

În funcția de calcul și afișare a soluției (`Afiseaza()`) se parcurg simultan cei doi vectori și în cazul detectării unui factor comun se determină puterea cea mai mică, apoi se înmulțește c.m.m.d.c. cu factorul respectiv ridicat la puterea determinată. Deci, pentru calculul c.m.m.d.c. se folosesc două funcții: prima determină minimul a două valori, cealaltă calculează puterea naturală a unui număr.

```
#include <iostream.h>
#define Dim 20

typedef long Factori[Dim];
typedef int Multiplicitati[Dim];

Factori DivA, DivB;
Multiplicitati OrdA, OrdB;
int CatiA, CatiB;

void Descompune(long, int &, Factori, Multiplicitati);
long Putere (unsigned, unsigned);
long cmmdc();
```

```
int main()
{
  long int a, b;
  cout << "a = "; cin >> a;
  cout << "b = "; cin >> b;
  Descompune(a, CatiA, DivA, OrdA);
  Descompune(b, CatiB, DivB, OrdB);
  cout <<"c.m.m.d.c.("<<a<<", "<<b<<")= "<<cmmdc();
  return 0;}

void Descompune (long x, int & nx, Factori DivX,
                 Multiplicitati OrdX)
{ long  d = 2;
  nx = 0;
  while (x>1)
        {
         if (!(x % d))                    //am gasit un factor
            {
            DivX[++nx] = d;
            while (!(x % d))      //calculez multiplicitatea
                  { x/=d;
                    OrdX[nx]++; }
             }
          d++;
        }
}
long Putere (unsigned x, unsigned y)
//calculeaza x la puterea y
{ unsigned i;
  long P=1;
  for (i = 1; i <= y; i++) P *= x;
  return P; }
long cmmdc()
{
int i = 1, j = 1;
long c = 1;
while ((i <= CatiA) && (j <= CatiB))
      if (DivA[i] == DivB[j])     //am gasit un factor comun
       {          //determin puterea minima apoi calculez
        c*=Putere(DivA[i],OrdA[i]>OrdB[j]?OrdA[i]:OrdB[j]);
        i++; j++;                 //trec la factorii urmatori
        }
        else
        if (DivA[i] < DivB[j]) i++;
           else j++;
return c;
}
```


Incluziune

Se consideră un fișier text cu numele `prim.txt` care conține două linii. Pe prima linie se află elementele mulțimii `A`, iar pe a doua linie sunt scrise elementele mulțimii `B`. Elementele celor două mulțimi sunt din mulțimea

```
['a'..'z','A'..'Z','0'..'9']
```

și sunt separate între ele prin câte un spațiu. Scrieți un program care scrie pe prima linie a fișierului de tip text `doi.txt` cuvântul `'DA'`, dacă mulțimea `A` este inclusă în `B`, respectiv `'NU'` în caz contrar.

(Bacalaureat, iunie 1999)

Soluție

Vom reprezenta o mulțime de caractere prin vectorul caracteristic. Mai exact, o mulțime de caractere va fi un vector cu 256 de componente care pot avea valorile `0` sau `1`. Dacă pe poziția `i` în vectorul caracteristic se găsește valoarea 1, caracterul cu codul ASCII `i` aparține mulțimii. Dacă pe poziția `i` în vectorul caracteristic se găsește valoarea `0`, caracterul cu codul ASCII `i` nu aparține mulțimii.

Pentru a citi o mulțime utilizăm funcția numită `Citeste_Multimea()`, care citește caracterele de pe linia curentă a fișierului de intrare (până la întâlnirea caracterului *newline*) și le reunește la mulțimea specificată ca parametru. Parametrul este obligatoriu, deoarece în programul principal apelăm această funcție de două ori: o dată pentru a citi mulțimea `A` și încă o dată pentru a citi mulțimea `B`. Parametrul trebuie să fie de intrare/ieșire (transmis prin referință). Funcția `Compara()` realizează compararea celor două mulțimi construite. În cadrul funcției `Compara()` apelăm funcția `Inclus()`, care testează dacă mulțimea specificată de primul parametru este inclusă în mulțimea specificată de cel de-al doilea parametru.

```
#include <fstream.h>

typedef int Multime[256];
Multime A, B;
ifstream p("prim.txt");

void Citeste_Multimea(Multime X);
void Compara(Multime X, Multime Y);
int Inclus(Multime X, Multime Y);

int main ()
{
  Citeste_Multimea(A);              //construiesc multimea A
  Citeste_Multimea(B);              //construiesc multimea B
  p.close();
  Compara(A, B);                      //verific incluziunea
  return 0;           }
```

```
void Citeste_Multimea(Multime X)
{ char c;
  do
     { c=p.get();        //citesc un caracter
       X[c]=1;    }     //adaug caracterul c la multime
  while (c!='\n');
  X[' '] = X[10] = 0; //extrag caracterele spatiu si LF
}

void Compara(Multime X, Multime Y)
{ ofstream d("doi.txt");
  if (Inclus(X, Y))           //daca X este inclus in Y...
     d << "DA"<<endl;
     else
     d << "NU"<<endl;
  d.close();                  }

int Inclus(Multime X, Multime Y)
{//intoarce 1 daca X inclus in Y si 0 altfel
 int i;
 for (i='0'; i<='9'; i++)
     if (X[i]>Y[i]) return 0;
 for (i='A'; i<='Z'; i++)
     if (X[i]>Y[i]) return 0;
 for (i='a'; i<='z'; i++)
     if (X[i]>Y[i]) return 0;
 return 1;                    }
```

Observație

Funcția `Citeste_Multimea()` trebuie să modifice valoarea parametrului său. Parametrul său este de tip `Multime`, deci este un vector. Prin urmare, la apelul funcției, pe stivă este memorat un pointer care are ca valoare adresa primului element din vector.

Operații cu numere naturale mari

Să considerăm numere naturale mari (cu maximum 200 de cifre). Nici unul dintre tipurile limbajului nu permite lucrul cu astfel de numere. Prin urmare, trebuie să implementăm propriul nostru tip pentru numere mari și operațiile uzuale cu numere mari (adunare, scădere, înmulțire, împărțire, comparare).

Soluție

Vom reprezenta un număr natural mare ca un vector în care reținem în ordine cifrele sale, începând cu unitățile (astfel, pe poziția 0 se va afla cifra unităților, pe poziția 1, cifra zecilor ș.a.m.d.). Observați că indicele din vector al cifrei coincide cu puterea bazei corespunzătoare cifrei.

```
#define Lg 200
typedef char NrMare[Lg];
```


Citirea unui număr mare

Pentru a citi un număr mare vom citi de la tastatură numărul într-un șir de caractere, apoi vom transforma caracterele cifră în numere și le vom memora în ordine inversă în vectorul cu cifrele numărului mare. Pentru a ușura calculele, vom completa zona de memorie alocată numărului mare (celelalte componente ale vectorului de cifre) cu 0. Funcția `citire()` va avea doi parametri: vectorul în care reținem cifrele numărului mare citit (`NrMare x`) și numărul de cifre ale numărului mare (`int & nx`). Observați că am transmis parametrul `nx` prin referință (pentru ca funcția `citire()` să poată modifica valoarea acestui parametru, valoarea rămânând modificată și după apel). Parametrul `x` nu este necesar să fie transmis prin referință, deoarece `x` este numele unui vector, deci este un pointer constant la primul element din vector.

```
void citire (NrMare x, int &nx)
{ int i;
  char s[Lg+1];
  cin>>s;
  nx=strlen(s); //determin numarul de cifre
  for (i=nx-1; i>=0; i--) x[nx-i-1]=s[i]-'0';
  for (i=nx; i<Lg; i++) x[i]=0;
}
```

Afișarea unui număr mare

Vom parcurge vectorul de cifre de la sfârșit către început, afișând succesiv cifrele. Funcția de afișare va avea doi parametri (vectorul în care reținem cifrele numărului mare citit și numărul de cifre ale numărului mare).

```
void afisare (NrMare x, int nx)
{for (int i=nx-1; i>=0; i--) cout <<(int)x[i];
 cout<<endl; }
```

Suma a două numere mari

Pentru a calcula suma a două numere mari vom parcurge simultan cele două numere, începând de la unități. Vom aduna cele două cifre de pe poziții corespondente și vom lua în calcul și eventuala cifră de transport care s-a obținut de la adunarea precedentă. Reținem în sumă cifra obținută prin adunare și recalculăm transportul. Suma ar putea avea o cifră în plus față de cel mai mare dintre numere (dacă la sfârșitul adunării cifra de transport este nenulă).

Funcția `suma()` va avea ca parametri cele două numere mari care se adună și numărul mare rezultat (vectorul de cifre și numărul de cifre pentru fiecare). Observați că parametrul `ns` (numărul de cifre ale numărului mare obținut în urma adunării) a fost transmis prin referință.

```
void suma (NrMare a, int na, NrMare b, int nb,
           NrMare s, int & ns)                    //s=a+b
{int i, cifra, t=0, max;
```

```
//completam numarul cel mai mic cu zerouri nesemnificative
if (na<nb) {max=nb; for (i=na; i<nb; i++) a[i]=0;}
    else   {max=na; for (i=nb; i<na; i++) b[i]=0;}
for (i=0; i<max; i++)
     {cifra=a[i]+b[i]+t;
      s[i]=cifra%10;
      t=cifra/10;          }
if (t) s[i++]=t;
ns=i; }
```

Diferența a două numere mari

Vom presupune că descăzutul este mai mare sau egal cu scăzătorul. Pentru a calcula diferența a două numere mari vom parcurge descăzutul începând de la unități. Vom scădea din cifra curentă a descăzutului cifra curentă a scăzătorului și vom lua în calcul și eventuala cifră de transport, obținută de la scăderea precedentă. Dacă rezultatul este negativ, împrumutăm 10 de pe poziția următoare (este posibil, dacă descăzutul este mai mare decât scăzătorul), și în acest caz cifra de transport devine -1. Diferența ar putea avea mai puține cifre decât descăzutul, astfel încât la sfârșit calculăm numărul de cifre din diferență, ignorând zerourile nesemnificative.

```
void dif (NrMare a, int na, NrMare b, int nb,
          NrMare d, int & nd)                    //d=a-b
{int i, t=0;
 for (i=0; i<na; i++)
     {d[i]=a[i]-b[i]+t;
      if (d[i]<0)
         {d[i]+=10; t=-1;}
         else t=0;
     }
 i--;
 while (i && !d[i]) i--;
 nd=i+1; }
```

Produsul a două numere mari

Vom inițializa vectorul de cifre ale produsului cu 0. Vom înmulți fiecare cifră a înmulțitorului (b) cu deînmulțitul (a) și vom aduna acest produs parțial obținut la produs (ale cărui cifre sunt memorate în vectorul p). Observați că atunci când înmulțim cifra b[j] cu a, produsul parțial obținut este înmulțit cu 10^j (adică cifrele sunt deplasate cu j poziții la dreapta).

```
void produs (NrMare a, int na, NrMare b, int nb,
             NrMare p, int & np)                 //p=a*b
{int i, j, t, cifra;
 for (i=0; i<Lg; i++) p[i]=0;
 for (i=0; i<nb; i++)
     {for (t=j=0; j<na; j++)
          {cifra=p[i+j]+a[j]*b[i]+t;
           p[i+j]=cifra%10;
           t=cifra/10;}
```



```
      if (t) p[i+j]=t;      }
 np=na+nb-1;
 if (p[np]) np++; }
```

Compararea a două numere mari

Dacă numărul de cifre din a este mai mic decât numărul de cifre din b, atunci a<b (și invers). Dacă numerele a și b au același număr de cifre, atunci parcurgem cele două numere simultan, începând de la cifra cea mai semnificativă până la întâlnirea a două cifre diferite. Dacă am întâlnit două cifre diferite, ele determină ordinea dintre a și b; în caz contrar, numerele sunt egale.

```
int compara(NrMare a, int na, NrMare b, int nb)
/*functia returneaza -1 daca a<b; 0 daca a=b; 1 daca a>b */
{int i;
 if (na<nb) return -1;
 if (nb<na) return 1;
 for (i=na-1; i>=0 && a[i]==b[i]; i--);
 if (i<0) return 0;
 if (a[i]<b[i]) return -1;
 return 1;}
```

Înmulțirea unui număr mare cu o putere a lui 10

Pentru a înmulți un număr mare cu 10^{nr} trebuie să deplasăm cifrele numărului cu nr poziții la dreapta și să completăm în stânga cu zerouri.

```
void p10(NrMare x, int &nx, int nr)
{// inmulteste numarul mare x cu 10^nr
 int i;
 for (i=nx-1; i>=0; i--) x[i+nr]=x[i];
 for (i=0; i<nr; i++) x[i]=0;
 nx+=nr;   }
```

Împărțirea a două numere mari

Dacă cele două numere mari sunt comparabile ca dimensiune, putem simula împărțirea prin scăderi repetate; în caz contrar, această metodă este ineficientă. Vom simula algoritmul de împărțire învățat la matematică:

```
void div(NrMare a, int na, NrMare b, int nb,
         NrMare c, int & nc, NrMare r,int &r)
{// imparte a la b; c este catul; r este restul
  int i;
  nr=0; nc=na;
  for (i=na-1; i>=0; i--)
    { p10(r,nr,1); r[0]=a[i];
      c[i]=0;
      while (compara(b,nb,r,nr)!=1)
            {c[i]++;
             dif(r,nr,b,nb,r,nr);  }
    }
```

```
  while (!c[nc-1] && nc>1) nc--;
  while (!r[nr-1] && nr>1) nr--; }
```

Secvență regulată

Să considerăm secvențe constituite numai din paranteze rotunde și paranteze pătrate, adică din caracterele `()[]`. Prin definiție, o secvență de paranteze este regulată dacă se obține aplicând următoarele reguli:

1. `()` și `[]` sunt secvențe regulate;
2. Dacă `A` este regulată, atunci `(A)` și `[A]` sunt secvențe regulate;
3. Dacă `A` și `B` sunt secvențe regulate, atunci `AB` este secvență regulată.

De exemplu, `()()[]` și `[(())][]` sunt secvențe regulate, în timp ce `][` sau `[(]` sau `([)]` nu sunt secvențe regulate.

Se consideră o secvență de paranteze. La fiecare pas se inserează o paranteză (rotundă sau pătrată, deschisă sau închisă) la începutul sau la sfârșitul secvenței. Scrieți un program care să determine, după fiecare pas, lungimea celei mai scurte subsecvențe regulate (formată din caractere consecutive ale secvenței) care conține paranteza inserată la pasul respectiv.

Date de intrare

Fișierul de intrare `secvreg.in` conține pe prima linie secvența inițială de paranteze (de lungime maximă `10000`). Pe cea de-a doua linie a fișierului de intrare se află un număr natural `N` care reprezintă numărul de pași (`N≤10000`). Fiecare dintre următoarele `N` linii conține un număr natural `A` și un caracter `C` separate printr-un singur spațiu. Dacă `A` este `0`, atunci caracterul `C` este inserat la începutul secvenței; dacă `A` este `1`, atunci caracterul `C` este inserat la sfârșitul secvenței.

Date de ieșire

Fișierul de ieșire `secvreg.out` conține `N` linii. Pe cea de a `i`-a linie va fi afișată lungimea celei mai scurte subsecvențe regulate (formată din caractere consecutive ale secvenței) care conține paranteza inserată la pasul `i`. Dacă nu există o astfel de subsecvență, pe linia respectivă veți afișa `0`.

Exemple

secvreg.in	secvreg.out	secvreg.in	secvreg.out
[] 3 0) 0 (0 (	0 2 6	() 3 1) 0) 0 (	0 0 2

(.campion, 2004)

Soluție

Numim coadă cu dublă prioritate (*dequeue*) o structură de date abstractă care suportă următoarele operații:

- inserarea unui element la sfârșit;
- inserarea unui element la început;
- extragerea elementului de la sfârșit;
- extragerea elementului de la început;
- accesarea elementului de la început;
- accesarea elementului de la sfârșit.

Vom utiliza două cozi cu dublă prioritate:

1. Coada `a` cu elemente de tip `char`, în care inserăm succesiv caracterele din șirul inițial și ulterior cele specificate în operațiile din fișierul de intrare.
2. Coada `b` cu elemente de tip `int`, cu următoarea semnificație:
   `b[i]`= lungimea secvenței regulate care începe cu poziția `i`.

Implementam cozile cu dublă prioritate `a` și `b` ca doi vectori. Pentru a putea realiza operații și la începutul, și la sfârșitul vectorilor, am dublat spațiul de memorie necesar pentru fiecare dintre cei doi vectori și am început inserarea elementelor de la mijloc.

Inițial, inserăm în ordine toate caracterele șirului citit în coada `a`. Pentru a păstra ordinea, fiecare nou caracter va fi inserat la sfârșitul cozii. Când inserăm o paranteză închisă la sfârșitul cozii, iar pe prima poziție în coadă este paranteza deschisă corespunzătoare, deducem că se formează o secvență regulată cu lungimea cu `2` mai mare decât cea deja existentă pe această poziție. Prin urmare, reținem în vectorul `b` această lungime și eliminăm din coada `a` cele două paranteze (practic, fiecare secvență regulată formată este eliminată din `a`, dar în `b` reținem lungimile secvențelor formate). Când inserăm la începutul cozii `a` o paranteză deschisă, iar la sfârșitul cozii `a` există paranteza închisă corespunzătoare, se formează o secvență regulată cu lungimea cu `2` mai mare și executăm aceleași operații ca în cazul precedent.

```
#include <stdio.h>
#define LgMax 10000
#define InFile "secvreg.in"
#define OutFile "secvreg.out"
typedef int CoadaI[LgMax];
typedef char CoadaC[LgMax];
char pereche[100];
CoadaC a;
CoadaI b;
/* inca = inceputul cozii a; sfa = sfarsitul cozii a;
   initial coada a este vida */
int inca=LgMax/2, sfa=LgMax/2-1;
/* incb = inceputul cozii b; sfb = sfarsitul cozii b;
   initial coada b este vida */
int incb=LgMax/2, sfb=LgMax/2-1;
char s[LgMax];

int deschis(char c)
{return c=='(' || c=='[';}
```

```
int la_sfarsit(char c)
/* functia insereaza caracterul c la sfarsitul cozii a si
returneaza lungimea secventei regulate care contine
caracterul inserat */
{ int ret;
  if ( inca<=sfa && !deschis(c) && a[sfa] == pereche[c] )
     {ret=b[sfb]+2;/* retin lungimea secventei regulate*/
      sfa--; sfb--; /* elimin secventa regulata formata */
     b[sfb]+=ret; /*memorez in b lungimea noii secvente*/
     return ret;      }
  a[++sfa]=c; /*adaug la sfarsitul cozii a caracterul c */
  b[++sfb]=0;
  /* cu c se formeaza o secventa regulata de lungime 0 */
  return 0; }

int la_inceput(char c)
/* functia insereaza caracterul c la inceputul cozii a si
returneaza lungimea secventei regulate care contine
caracterul inserat */
{ int ret;
  if (inca<=sfa && deschis(c) && a[inca] == pereche[c] )
     {ret=b[incb]+2; /* retin lungimea secventei regulate*/
     inca++; incb++; /* elimin secventa regulata formata */
     b[incb]+=ret; /* memorez in b lungimea noi secvente */
     return ret;      }
  a[--inca]=c ; /*adaug la inceputul cozii a caracterul c*/
  b[--incb]=0 ;
  /* cu c se formeaza o secventa regulata de lungime 0 */
  return 0; }
int main(void)
{ char * c, ch;
  int n, di, i;
  FILE *fin, *fout;
  fin=fopen(InFile, "rt");
  fscanf(fin,"%s",s);
  fout=fopen(OutFile,"wt");
  pereche['('] = ')'; pereche[')'] = '(';
  pereche['['] = ']'; pereche[']'] = '[';
  /* deocamdata elementele din coada a formeaza o secventa
     regulata de lungime 0 */
  b[++sfb]=0;
  /* inserez succesiv caracterele din sirul citit la
     sfarsitul cozii a */
  for (c=s; *c; ++c ) la_sfarsit( *c );
  /* citesc numarul de operatii */
  fscanf(fin, "%d", &n );
  for (i = 0; i < n; i++ )
      {/* citesc succesiv operatiile */
      fscanf(fin, "%d %c", &di, &ch );
```



```
      if (di) fprintf(fout,"%d\n", la_sfarsit(ch));
         else fprintf(fout,"%d\n", la_inceput(ch));
      }
  fclose(fin);
  fclose(fout);
  return 0; }
```

6.12. *Probleme propuse*

1. Care dintre următoarele propoziții sunt adevărate și care sunt false? Justificați!
   a. Parametrii formali ai unei funcții sunt cei care intervin la apelul funcției.
   b. La apelul unei funcții se specifică numele și tipul parametrilor actuali.
   c. Parametrii actuali trebuie să corespundă cu parametrii formali ca număr, ordine și tip.
   d. La apelul unei funcții, valorile parametrilor actuali sunt copiate pe stivă.
   e. Funcțiile trebuie să aibă cel puțin un parametru formal.

2. Funcția `f` primește ca parametru un număr real și trebuie să determine numărul de cifre de la partea întreagă și numărul de cifre de la partea zecimală a numărului. Care sunt prototipuri corecte pentru funcția `f`? Justificați răspunsul!
   a. `void f(float a, int nr1, int nr2);`
   b. `void f(float & a, int nr1, int nr2);`
   c. `void f(float, int *, int *);`
   d. `void f(float ; int & ; int &);`
   e. `int  f(float, int &);`

3. Să considerăm funcția `Sum(int y)` care returnează suma elementelor plasate într-un vector până la poziția `y` (inclusiv). Care dintre următoarele expresii are valoarea diferită de `0` dacă și numai dacă elementul de pe poziția `i` din vector este strict pozitiv?
   a. `Sum(i)>0 && Sum(i-1)<=0`
   b. `Sum(i)-Sum(i-1)>0`
   c. `Sum(i)>0 && Sum(i+1)<=0`
   d. `Sum(i)>0 && Sum(i-1)>0 && Sum(i+1)>0`

4. Alegeți propozițiile adevărate. Justificați alegerea!
   a. Programul următor va afișa `0363153159`.
   b. Programul următor conține erori de sintaxă.
   c. Programul următor va afișa `0303163169`.
   d. Programul următor va afișa `036315399`.

```
#include <iostream.h>
int x, y;
void F(int, int &);
void main ()
```

```
{ F(x, y); cout << x << y;
  F(y, x); cout << x << y;
  F(x, x); cout << x << y;
  F(y, y); cout << x << y; }
void F(int a, int & b)
{  a+=3;  b+=a; }
```

5. Alegeți varianta corectă. Justificați alegerea!

a. Programul afișează:
```
4122
4412
8244
8124
```

b. Programul afișează:
```
4422
4112
8244
8124
```

c. Programul afișează:
```
4422
4112
4244
8128
```

d. Programul conține erori de sintaxă.

```
#include <iostream.h>
int a=1, b=1, c=1, d=1;
void P(int &, int);
void main ()
{ P(a, b);   cout << a << b << c << d << endl;
  P(c, d);   cout << a << b << c << d << endl; }
void P(int & b, int c)
{  a *= 2; b *= 2; c *= 2; d *= 2;
   cout << a << b << c << d << endl; }
```

6. Funcția următoare trebuie să calculeze numărul de cifre egale cu `0` din scrierea numărului întreg `n`, primit ca parametru. Care sunt erorile din această funcție?

```
int nr0(int n)
{int nr;
 while (nr)
      {if (!nr%10) nr++;
       nr/=10; }
}
```

7. Ce va afișa pe ecran următorul program?

```
#include <iostream.h>
int a=5, b=10, c=20;
void p(int a, int & b)
{a++; b++; c++;
 cout<<a<<b<<c;  }

int main()
{p(a, b); cout<<a<<b<<c;
 p(c, c); cout<<a<<b<<c;
 return 0; }
```

8. Alegeți afirmațiile adevărate. Justificați alegerea!

a. Programul următor conține erori de sintaxă.

b. Programul următor afișează `1.0 2.0`.

c. Programul următor afișează `0 2`.

d. Programul următor afișează `1.0 0.0`.

```
#include <iostream.h>
float a[2];
int i;
void P(int & k, float & x)
{  k = 1; x = 0;  }
int main(void)
{ a[0] = 1; a[1] = 2;
  for (i = 0; i < 2; i++)
        {P(i, a[i]);
        cout << a[0] << " " << a[1] << endl; }
  return 0;      }
```

9. Ce va afișa pe ecran următorul program?

```
#include <iostream.h>
int a[5]={1,2,3,4,5}, n=2;
int P(int, int &, int);
void main()
{ int i;
  P(n, a[1], a[2]);
  for (i=0; i<=n; i++) cout<<a[i];
  P(n, a[0], a[1]);
  for (i=0; i<=n; i++) cout<<a[i];
  return 0; }
void P(int n, int &x, int y)
{n++; x++; y++; }
```

10. Cum trebuie declarat antetul subprogramului `f` astfel încât programul următor să afișeze pe ecran valoarea `147`?

```
#include <iostream.h>
void f...
{a++; b+=a; c+=b;}
void main()
{int a=1, b=2, c=3;
 f(a,b,c);
 cout<<a<<b<<c;}
```

a. `void f(int, int , int );`

b. `void f(int & a, int b, int c);`

c. `void f(int a, int & b , int & c );`

d. `void f(int &a, int &b, int & c);`

e. `void f(int &a , int b, int & c );`

f. Indiferent de antet, aceste valori nu pot fi obținute.

11. Fie a, b, c, d numere reale citite de la tastatură. Să se calculeze valoarea expresiei:

$$E(a,\ b,\ c,\ d) = \begin{cases} \max(|a|-3, |b|+6), & \text{daca } c+2d<=3 \\ \min(|a|-3, |b|+6), & \text{daca } 3<c+2d<7 \\ \min(|a|+|b|, 18), & \text{daca } c+2d\geq 7 \end{cases}$$

12. Fie a şi b două numere complexe citite de la tastatură. Să se evalueze expresia pentru o valoare dată a variabilei complexe z:

$$E(z) = \begin{cases} 3z^2-6az+b, & \text{daca } |z|<1 \\ 6az-b, & \text{daca } |z|>1 \end{cases}$$

13. Scrieţi o funcţie care să determine raza cercului circumscris unui triunghi dat prin coordonatele vârfurilor.

14. Scrieţi o funcţie care să elimine zerourile dintr-un vector de numere întregi transmis ca parametru, fără a modifica ordinea elementelor nenule din vector. Descrieţi şi o funcţie şablon care să realizeze acelaşi lucru, indiferent de tipul elementelor vectorului.

15. Fie două numere naturale n şi m şi o funcţie $f:\{1,\ldots,n\}\to\{1,\ldots,m\}$. Scrieţi câte o funcţie care să testeze injectivitatea, surjectivitatea, respectiv bijectivitatea funcţiei.

16. Concepeţi câte o funcţie pentru următoarele operaţii cu şiruri de caractere:

a. Inserarea unui şir de caractere într-un alt şir de caractere; se transmit ca parametri cele două şiruri şi poziţia i în care se face inserarea.

b. Ştergerea unei secvenţe de caractere dintr-un şir; şirul de caractere, poziţia i de la care începe secvenţa de caractere ce va fi ştearsă şi numărul de caractere care se şterg sunt specificate ca parametri.

17. *Operaţii cu mulţimi*

Fie n un număr natural (n<1024). O metodă eficientă de a reprezenta o submulţime S a mulţimii {0, 1, ..., n} este vectorul caracteristic. Mai exact, asociem fiecărui element x din mulţimea {0, 1, ..., n} un bit. Bitul corespunzător elementului x are valoarea 1, dacă $x\in S$, respectiv valoarea 0, dacă $x\notin S$. Descrieţi funcţii care să implementeze operaţiile elementare cu mulţimi (reuniune, intersecţie, diferenţă, incluziune, apartenenţă) folosind o astfel de reprezentare.

18. *Ciurul lui Eratostene*

Creaţi un proiect în care utilizaţi fişierul cu funcţiile de lucru cu mulţimi elaborat la problema precedentă. Proiectul va avea ca scop generarea tuturor numerelor prime mai mici decât 1024, folosind metoda ciurului lui *Eratostene*.

19. *Traducere*

Să se scrie un program care realizează traducerea „cuvânt cu cuvânt" a unui text pe baza unui dicţionar. Textul care trebuie tradus, traducerea, precum şi dicţionarul sunt fişiere text ale căror nume se citesc de la tastatură. Cuvintele de pe o linie sunt separate prin spaţii şi/sau semne de punctuaţie (',', '.', ';', ':') şi au lungimea maximă de 20 de caractere. Lungimea maximă a unei linii din fişier este de 100 de caractere. În dicţionar, fiecare rând are forma cuvant=traducere.

20. *Jocul vieţii şi al morţii*

Acest joc a fost propus de matematicianul John Conway şi poate fi descris după cum urmează.

Un habitat este un tablou bidimensional cu n linii şi m coloane; în fiecare element al acestui tablou poate fi o celulă. Vecinătatea unei poziţii este constituită din cele 8 poziţii din jurul ei. Numărul de vecini ai unei celule este numărul de celule care se află în vecinătatea ei. Populaţia evoluează în paşi discreţi: la momentul t+1, configuraţia populaţiei este determinată doar de configuraţia de la momentul t. Legile după care evoluează populaţia sunt:

- dacă o celulă are 0 sau 1 vecini, ea moare prin izolare;
- dacă o celulă are mai mult de 3 vecini, ea moare prin sufocare;
- dacă în vecinătatea unei poziţii goale se află 3 celule, atunci în acea poziţie se naşte o celulă;
- în toate celelalte cazuri nu se înregistrează vreo schimbare.

Să se scrie un program care simulează acest joc, vizualizând pe ecran, în mod text, fiecare pas.

21. *Apariţie*

Fie x un număr natural (x<MAXLONG). Se cere să se determine numărul de apariţii ale lui x ca subsecvenţă de cifre în numerele din intervalul dat [a,b] (a şi b fiind numere naturale, a<b<MAXLONG). De exemplu, pentru a=1, b=60, x=5 veţi afişa 16.

22. *Comparaţie*

Scrieţi un program care realizează o comparaţie între metodele de sortare învăţate până acum (*bubblesort*, sortare prin selecţie, sortare prin inserţie). Realizaţi o reprezentare grafică animată, precum şi estimarea timpului de sortare pentru fiecare dintre cele trei metode.

23. *Baze ascunse*

Se dau n numere naturale, $x_1, x_2, \ldots, x_n$, de câte cel mult 9 cifre. Fiecare număr x_i este reprezentat într-o bază b_i ($2\leq b_i\leq 10$), care nu este cunoscută. Să se determine pentru fiecare număr x_i o bază b_i astfel încât intervalul de valori [a,b] în care sunt cuprinse toate cele n numere, scrise în baza 10, să fie de lungime minimă. Prin lungimea unui interval se înţelege diferenţa dintre capetele intervalului în valoare absolută. De exemplu, pentru numerele 35, 27, 100, 13, 19, în fişierul de ieşire veţi afişa:

```
35 in baza 6
27 in baza 8
100 in baza 4
13 in baza 10
19 in baza 10
a=13  b=23
```

(Olimpiada Națională de Informatică, Timișoara, 1997)

24. *Petice*

La grădiniță, Vasilică a învățat să coasă. Doamna educatoare i-a dat petice dreptunghiulare de material (de dimensiuni numere naturale), de culori distincte, și o bucată de pânză albă, de formă dreptunghiulară (de dimensiuni, de asemenea, numere naturale), pe care Vasilică urma să coasă peticele, așezându-le laturile paralel cu laturile bucății de pânză. Doamna educatoare i-a spus că are voie să coasă un petic peste altul, cu condiția ca întotdeauna peticul de deasupra să acopere cel mult un colț al oricărui petic de dedesubt. După ce și-a terminat opera, Vasilică o duce doamnei educatoare și o întreabă care este culoarea ce ocupă cea mai mare suprafață și în ce ordine a cusut el peticele. Scrieți un program care să o ajute pe doamna educatoare să îi răspundă lui Vasilică.

Din fișierul de intrare `petice.in` se citesc de pe prima linie două numere naturale `n` și `m` reprezentând lățimea și, respectiv, lungimea bucății de pânză albă (`n, m<100`). Urmează `n` linii, pe fiecare linie fiind specificate `m` caractere. Al `j`-lea caracter de pe linia `i+1` reprezintă culoarea punctului de coordonate `i, j` de pe pânză, culorile fiind codificate cu litere mici ale alfabetului englez, 'a' reprezentând alb, culoarea pânzei.

Fișierul de ieșire `petice.out` va conține pe prima linie un caracter `cmax` reprezentând culoarea ce ocupă cea mai mare suprafață, iar pe cea de-a doua linie $c_1c_2\ldots c_n$ = culorile peticelor, în ordinea în care au fost cusute pe pânză.

25. *Pepsi*

Într-o cameră există `M` sticle identice, care conțin fiecare câte `K` dl de Pepsi. Sticlele sunt numerotate distinct de la `1` la `M`. Fiind cald, s-a format o coadă din `N` persoane însetate. Persoanele sunt numerotate de la `1` la `N`, în ordinea în care s-au așezat la coadă (deci prima persoana de la coadă are numărul `1`). La un moment dat, o singură persoană poate intra în cameră, poate servi un pahar de `1` dl de Pepsi, din una dintre sticlele existente în cameră, îl bea, apoi pleacă. Există două categorii de persoane: persoane risipitoare (care întotdeauna vor lua un pahar din una dintre cele mai pline sticle) și persoane econoame (care întotdeauna vor lua un pahar din una dintre cele mai goale sticle; evident, sticlele complet goale se ignoră).

Cunoscând volumul de lichid rămas în fiecare dintre cele `M` sticle, scrieți un program care să determine sticla din care a băut fiecare persoană.

Date de intrare/ieșire

Fișierul de intrare `pepsi.in` conține pe prima linie trei numere naturale `N M K`, separate prin spații, reprezentând numărul de persoane așezate la coadă, numărul

de sticle și, respectiv, numărul de decilitri de suc existenți inițial în fiecare dintre cele `M` sticle (`1≤N≤1000, 1≤M≤100, 1≤K≤20`). Cea de-a doua linie conține o succesiune de `N` caractere ce pot fi `R` sau `E`; dacă al `i`-lea caracter din această succesiune este `R`, deducem că persoana `i` este risipitoare, altfel, că este econoamă. Cea de-a treia linie din fișierul de intrare conține `M` numere naturale separate prin spații, reprezentând, în ordine, volumul de lichid rămas în fiecare dintre cele `M` sticle.

Fișierul de ieșire `pepsi.out` conține o singură linie pe care se află `N` numere naturale separate prin spații; al `i`-lea număr reprezintă numărul sticlei din care a băut persoana `i`. Dacă nu există soluție, pe prima linie a fișierului de ieșire se va scrie mesajul `IMPOSIBIL`.

Exemple

pepsi.in	pepsi.out	pepsi.in	pepsi.out
4 3 3 EEEE 3 0 2	2 2 2 3	5 5 2 EERRE 1 2 0 2 0	3 3 5 1 5

26. *Bonus*

Gigel este programator amator, dar web designer profesionist. Acum proiectează un magazin virtual și se confruntă cu o problemă de programare. Pe site sunt prezentate `n` produse, numerotate de la `1` la `n`. În coșul de cumpărături virtual, un client poate alege orice submulțime de produse dintre cele `n`. În funcție de componența coșului de cumpărături, Gigel trebuie să ofere un bonus.

Gigel a observat că se pot forma din cele `n` produse 2^n-1 submulțimi distincte nevide. A ordonat submulțimile lexicografic și a asociat apoi fiecărei submulțimi un număr de ordine de la `1` la 2^n-1.

De exemplu, pentru `n=3` submulțimile nevide în ordine lexicografică sunt:

Submulțimea	{1}	{1, 2}	{1, 2, 3}	{1, 3}	{2}	{2, 3}	{3}
Nr. de ordine	1	2	3	4	5	6	7

Pentru a acorda corect și eficient bonusul, trebuie să fie implementate operațiile:
- dată fiind o submulțime, să determine numărul ei de ordine;
- dat fiind un număr de ordine, să determine submulțimea corespunzătoare.

Date de intrare/ieșire

Fișierul de intrare `bonus.in` conține pe prima linie numărul natural `n` (`1≤n≤30`), iar pe a doua linie, un număr natural `m`, reprezentând numărul de operații ce trebuie executate (`1≤m≤1000`). Pentru fiecare operație urmează în fișier câte două linii; pe prima linie din cele două este scris tipul operației (`1` sau `2`); dacă tipul operației este `1`, atunci pe cea de-a doua linie este scris numărul de elemente din submulțime, urmat de elementele submulțimii, separate prin spații; dacă tipul operației este `2`, pe cea de-a doua linie va fi scris un număr de ordine (adică un număr natural cuprins între `1` și 2^n-1).

Fişierul de ieşire `bonus.out` va conţine `m` linii, câte una pentru fiecare operaţie din fişierul de intrare. Pe fiecare linie va fi scris rezultatul unei operaţii, în ordinea în care operaţiile apar în fişierul de intrare. Dacă operaţia este de tip `1`, în fişierul de ieşire va fi afişat numărul de ordine al submulţimii specificate în operaţie. Dacă operaţia este de tip `2`, în fişierul de ieşire vor fi afişate elementele submulţimii cu numărul de ordine specificat, separate prin spaţii, în ordine crescătoare.

Exemple

bonus.in	bonus.out
3 3 2 3 1 2 2 3 2 7	1 2 3 6 3

(Olimpiada Municipală de Informatică, Iaşi, 2004)

27. *Sumă Fibonacci*

Considerăm şirul lui Fibonacci: `0, 1, 1, 2, 3, 5, 8, 13, ...` Dat fiind un număr natural cu maxim 80 de cifre, scrieţi acest număr sub formă de sumă de termeni distincţi neconsecutivi din şirul Fibonacci, astfel încât numărul de termeni din sumă să fie minim. De exemplu, pentru `n=20` veţi afişa `2+5+13`.

(Olimpiada Naţională de Informatică, Constanţa, 2000)

28. *Pic*

Un grup de cercetători pleacă într-o expediţie în Oceanul Pacific. După o furtună puternică rămân fără apă. Norocul vine pe neaşteptate printr-o ploaie torenţială. Toate cele `n` vase existente pe vapor sunt scoase şi umplute cu apă (`n<70`). Fiecare vas în care se reţine apa de ploaie dispune de un dispozitiv care contorizează picăturile de apă, prin urmare, pentru fiecare vas `i` se cunoaşte numărul de picături din vas x_i ($x_i<10^{22}$). După ce se termină ploaia, apa strânsă este mutată în cele `k` butoaie (toate goale) existente pe vapor (`k<40`). Se cunoaşte capacitatea fiecărui butoi – în butoiul `i` încap v_i ($v_i<10^{22}$) picături, $1 \le i \le k$. Să se determine numărul maxim de butoaie care se pot umple.

(*campion*, 2003)

29. *Maşină*

Vasile a „împrumutat" o maşină şi a plecat în oraş să se distreze. Ghinionul a fost că maşina aparţinea poliţiei şi era dotată cu un echipament care emitea semnale indicând direcţia de mişcare a maşinii. Poliţiştii au luat imediat harta oraşului şi, cunoscând poziţia iniţială a maşinii şi direcţiile de mişcare, vor să determine poziţia finală a maşinii. Harta oraşului este reprezentată ca o matrice cu `R` linii şi `C` coloane, în matrice fiind marcate cu `'.'` (punct) poziţiile prin care maşinile pot

circula, iar cu `'X'` poziţiile prin care maşinile nu pot circula. Poziţia iniţială a maşinii a fost marcată pe hartă cu `'*'`. Maşina se poate deplasa în patru direcţii: la nord (adică de pe linia `i` pe linia `i-1` pe hartă), la sud (adică de pe linia `i` pe linia `i+1` pe hartă), la vest (adică de pe coloana `i` pe coloana `i-1` pe hartă) sau la est (adică de pe coloana `i` pe coloana `i+1` pe hartă).

Scrieţi un program care să ajute poliţiştii să găsească maşina.

Date de intrare/ieşire

Pe prima linie a fişierului de intrare `masina.in` se găsesc două numere naturale separate prin spaţiu `R C` (`1≤R, C≤50`), unde `R` reprezintă numărul de linii, iar `C` numărul de coloane de pe harta oraşului. Fiecare dintre următoarele `R` linii conţine o secvenţă de `C` caractere care pot fi doar `'.'`, `'*'` sau `'X'`, descriind harta oraşului. Următoarea linie, a `(R+2)`-a, conţine un număr natural `N` (`1≤N≤1000`) reprezentând numărul de schimbări de direcţie pe care le-a făcut maşina. Fiecare dintre următoarele `N` linii conţine un şir de caractere ce poate fi `NORD`, `SUD`, `VEST` sau `EST`. Oricare două direcţii consecutive sunt diferite.

Fişierul de ieşire `masina.out` conţine harta oraşului reprezentată ca în fişierul de intrare (`R` linii de câte `C` caractere). Pe hartă vor fi marcate cu `'*'` numai poziţiile finale posibile ale maşinii.

Exemple

masina.in	masina.out
3 4 .... *..X X.X. 2 EST NORD	.**. ...X X.X.

masina.in	masina.out
10 9 ....X X..XX..X. .X.XX.X.. ...XX.... ...XX.... .XXX..XX. ....X. ..XXX.X.. X.X....X. *.....X.. 4 EST NORD EST SUD	........X X..XX.*X. .X.XX.X.. ...XX.... ...XX.*** .XXX..XX* ....X* ..XXX*X.* X.X..*.X* ....**X.*

Anexe

1. Tabela codurilor ASCII

Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter
000	(nul)	022	▬ (syn)	044	,	066	B	088	X	110	n
001	☺ (soh)	023	↨ (etb)	045	-	067	C	089	Y	111	o
002	☻ (stx)	024	↑ (can)	046	.	068	D	090	Z	112	p
003	♥ (etx)	025	↓ (em)	047	/	069	E	091	[	113	q
004	♦ (eot)	026	→ (eof)	048	0	070	F	092	\	114	r
005	♣ (enq)	027	← (esc)	049	1	071	G	093	]	115	s
006	♠ (ack)	028	∟ (fs)	050	2	072	H	094	^	116	t
007	• (bel)	029	↔ (gs)	051	3	073	I	095	_	117	u
008	_ (bs)	030	▲ (rs)	052	4	074	J	096	`	118	v
009	○ (tab)	031	▼ (us)	053	5	075	K	097	a	119	w
010	■ (lf)	032	(spațiu)	054	6	076	L	098	b	120	x
011	♂ (vt)	033	!	055	7	077	M	099	c	121	y
012	♀ (np)	034	"	056	8	078	N	100	d	122	z
013	♪ (cr)	035	#	057	9	079	O	101	e	123	{
014	♫ (so)	036	$	058	:	080	P	102	f	124	|
015	☼ (si)	037	%	059	;	081	Q	103	g	125	}
016	► (dle)	038	&	060	<	082	R	104	h	126	~
017	◄ (dc1)	039	'	061	=	083	S	105	i	127	
018	↕ (dc2)	040	(	062	>	084	T	106	j		
019	‼ (dc3)	041	)	063	?	085	U	107	k		
020	¶ (dc4)	042	*	064	@	086	V	108	l		
021	§ (nak)	043	+	065	A	087	W	109	m		

Codul **ASCII** extins

Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter
128	Ç	149	ò	170	¬	191	┐	212	╘	234	Ω
129	ü	150	û	171	½	192	└	213	╒	235	δ
130	é	151	ù	172	¼	193	┴	214	╓	236	∞
131	â	152	_	173	¡	194	┬	215	╫	237	∅
132	ä	153	Ö	174	«	195	├	216	╪	238	∈
133	à	154	Ü	175	»	196	─	217	┘	239	∩
134	å	155	¢	176	▒	197	┼	218		240	≡
135	ç	156	£	177	▒	198	╞	219	▉	241	±
136	ê	157	¥	178	▓	199	╟	220	▄	242	≥
137	ë	158	_	179	|	200	╚	221	▌	243	≤
138	è	159	č	180	┤	201	╔	222	▐	244	⌠
139	ï	160	á	181	╡	202	╩	223	▀	245	⌡
140	î	161	í	182	╢	203	╦	224	α	246	÷
141	ì	162	ó	183	╖	204	╠	225	ß	247	≈
142	Ä	163	ú	184	╕	205	═	226	Γ	248	°
143	Å	164	ñ	185	╣	206	╬	227	π	249	•
144	É	165	Ñ	186	║	207	╧	228	Σ	250	–
145	æ	166	ª	187	╗	208	╨	229	σ	251	√
146	Æ	167	º	188	╝	209	╤	230	μ	252	ⁿ
147	ô	168	¿	189	╜	210	╥	231	τ	253	²
148	ö	169	_	190	╛	211	╙	232	ϕ	254	■
								233	Θ	255	

2. Cuvintele cheie C/C++

asm	asm	asm	auto	break	case
cdecl	_cdecl	__cdecl	char	class	const
continue	_cs	__cs	default	delete	do
double	_ds	__ds	else	enum	_es
_es	_export	__export	extern	far	_far
_far	_fastcall	__fastcall	float	for	friend
goto	huge	__huge	__huge	if	inline
int	interrupt	__interrupt	__interrupt	_loadds	__loadds
long	near	__near	__near	new	operator
pascal	_pascal	__pascal	private	protected	public
register	return	__saveregs	__saveregs	_seg	__seg
short	signed	sizeof	_ss	__ss	static
struct	switch	template	this	typedef	union
unsigned	virtual	void	volatile	while	

3. Sisteme de numerație

Un *sistem de numerație* este alcătuit dintr-o mulțime finită de simboluri și un set de reguli de reprezentare a numerelor cu ajutorul simbolurilor respective. Numărul de simboluri constituie *baza* sistemului de numerație.

În general, utilizăm sisteme de numerație poziționale, pentru care pozițiile simbolurilor corespund puterilor bazei sistemului de numerație. Mulțimea simbolurilor utilizate pentru reprezentarea numerelor depinde de baza sistemului de numerație. De exemplu, în sistemul de numerație *zecimal* se utilizează zece simboluri: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; în sistemul de numerație *binar* se utilizează două simboluri: 0, 1; în sistemul de numerație *octal* se utilizează opt simboluri: 0, 1, 2, 3, 4, 5, 6, 7; în sistemul de numerație *hexazecimal* se utilizează 16 simboluri: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Semnificația zecimală a simbolurilor A, B, C, D, E, F este, în ordine, 10, 11, 12, 13, 14, 15.

Să notăm cu b baza sistemului de numerație ($b \in N, b>1$).

Regula de reprezentare a numerelor în baza b

Fiecare cifră dintr-un număr valorează de b ori mai mult decât cifra din dreapta sa, deci cifrele numărului, de la dreapta spre stânga, corespund poziționat puterilor bazei sistemului de numerație:

$$x_n x_{n-1} \ldots x_1 x_0, y_1 y_2 \ldots y_p{}_{(b)} = x_n \cdot b^n + x_{n-1} \cdot b^{n-1} + \ldots + x_1 \cdot b + x_0 + y_1 \cdot b^{-1} + \ldots + y_p \cdot b^{-p}$$

Operații de conversie

Conversia numerelor din baza 10 în baza b

Regula de conversie a numerelor naturale

Pentru a converti un număr natural din baza 10 într-o bază b ($b \in N, b>1$) vom face împărțiri succesive la b până când obținem câtul 0. La primul pas, deîmpărțitul este numărul care se convertește; în continuare, la fiecare pas, câtul de la împărțirea precedentă devine deîmpărțit. Reprezentarea numărului în baza b se obține considerând resturile în ordinea inversă obținerii lor.

Exemplu

Pentru a converti $n=1263_{(10)}$ în baza 16 vom face împărțiri succesive la 16:

deîmpărțit	cât	rest
1263	=16*78	+ 15
78	=16* 4	+ 14
4	=16* 0	+ 4

Considerând resturile în ordine inversă, obținem $1263_{(10)}=4EF_{(16)}$.

Regula de conversie a numerelor reale din intervalul (0,1)

Conversia numerelor subunitare din baza 10 într-o bază b ($b \in N, b>1$) se realizează prin înmulțiri succesive cu b, până când se obține o perioadă sau până când partea fracționară a rezultatului este 0. La primul pas, deînmulțitul este numărul care se convertește; în continuare, la fiecare pas, partea fracționară a rezultatului precedent devine deînmulțit. Reprezentarea numărului în baza b se obține considerând cifrele de la partea întreagă a rezultatelor în ordinea obținerii lor.

Exemplu

Să convertim $n=0,3125_{(10)}$ în baza 2:

```
0,3125  x 2 = 0,625
0,625   x 2 = 1,25
0,25    x 2 = 0,5
0,5     x 2 = 1,0
```

Am obținut partea fracționară 0, deci $0,3125_{(10)}=0,0101_{(2)}$.

Regula de conversie a numerelor reale

Pentru a converti un număr real din baza 10 în baza b ($b \in N, b>1$) se convertesc separat partea întreagă și partea fracționară, conform regulilor anterioare.

De exemplu, $1263,3125_{(10)}=10011101111,0101_{(2)}$.

Conversia numerelor din baza b în baza 10

Pentru a converti un număr din baza b ($b \in N, b>1$) în baza 10 utilizăm dezvoltarea după puterile bazei sistemului de numerație și efectuăm calculele.

Exemple

$1AF_{(16)}=1 \cdot 16^2+A \cdot 16+F=431_{(10)}$

$11001,101_{(2)}=2^4+2^3+1+2^{-1}+2^{-3}=25,625_{(10)}$

Conversia numerelor din baza b în baza b'

Conversia numerelor din baza b în baza b' ($b, b' \in N, b, b'>1$) se realizează prin intermediul bazei 10: realizăm conversia din baza b în baza 10, apoi din baza 10 în b'.

Conversii între baze puteri ale lui 2

Datorită faptului că, deocamdată, calculatoarele utilizează informații reprezentate în formă binară, sistemul de numerație binar prezintă pentru informaticieni o importanță deosebită. Calculatoarele par a se descurca foarte ușor cu șiruri de cifre binare, dar oamenii se obișnuiesc mai greu. Frecvent, informaticienii apelează la conversia șirurilor binare în valori octale sau hexazecimale[40]. Operațiile de conversie între bazele 2 și 8, respectiv 2 și 16 se realizează mai simplu decât conversiile între două baze oarecare, datorită faptului că atât 8, cât și 16 sunt puteri ale bazei 2.

Regula de conversie a numerelor din baza 8 (16) în baza 2

Pentru a converti un număr octal (hexazecimal) în binar vom înlocui fiecare cifră a numărului cu reprezentarea sa binară pe 3, respectiv 4 poziții.

Exemple

$168AF_{(16)} = 1\ 0110\ 1000\ 1010\ 1111_{(2)}$

$75210,231_{(8)} = 111\ 101\ 010\ 001\ 000,\ 010\ 011011_{(2)}$

40. De exemplu, informaticienii exprimă uzual adresele de memorie în hexazecimal, nu ca un șir de 32 de cifre binare.

Regula de conversie a numerelor din baza 2 în baza 8 (16)

Pentru a converti un număr binar în baza 8 (16) vom asocia cifrele binare în grupe de câte 3 (respectiv 4) începând de la dreapta spre stânga pentru partea întreagă şi de la stânga la dreapta pentru partea zecimală, completând ultima grupă cu zerouri nesemnificative.

Exemple

1. $110\ 1010\ 1110\ 1010\ 1000\ 0010,\ 1111\ 0000\ 0001\ 11_{(2)} = 6AE82,F01C_{(16)}$
2. $1\ 111\ 001\ 010\ 000\ 110,\ 110\ 101\ 1_{(2)} = 171206,654_{(8)}$

Pentru aceste conversii este util tabelul următor, care conţine reprezentarea binară a valorilor {0, 1, ..., 15}:

Sistem zecimal	Sistem binar	Sistem zecimal	Sistem binar	Sistem zecimal	Sistem binar	Sistem zecimal	Sistem binar
0	0	4	100	8	1000	12	1100
1	1	5	101	9	1001	13	1101
2	10	6	110	10	1010	14	1110
3	11	7	111	11	1011	15	1111

4. Organizarea logică a memoriei

Din punct de vedere logic, memoria poate fi privită ca o succesiune de *celule binare*, fiecare celulă fiind capabilă să reţină o cifră binară (*0* sau *1*).

Cantitatea de informaţie ce poate fi înregistrată într-o celulă binară se numeşte *bit* (*B*inary digi*T* – cifră binară).

Principalele operaţii de lucru cu memoria sunt extragerea informaţiilor din memorie (*citire*) şi transferul informaţiilor în memorie (*scriere*). Pentru realizarea acestor operaţii este necesară localizarea zonei de memorie ce conţine informaţiile (pentru citire), respectiv a zonei de memorie în care urmează a fi stocate informaţiile (pentru scriere). Localizarea unei zone de memorie se realizează prin intermediul unei *adrese*.

Cea mai mică zonă de memorie adresabilă este *locaţia de memorie*, constituită din 8 celule binare consecutive[41].

Cantitatea de informaţie stocată într-o succesiune de 8 celule binare se numeşte *octet* (*byte*).

Numărul total de *bytes* care pot fi înregistraţi în memorie reprezintă *capacitatea memoriei*. Pentru a exprima capacitatea memoriei se folosesc multipli ai *byte*-ului:

1 Kb (kilobyte) = 1024 bytes (2^{10} bytes)
1 Mb (megabyte) = 1024 Kb (2^{20} bytes)
1 Gb (gigabyte) = 1024 Mb (2^{30} bytes)
1 Tb (terabyte) = 1024 Gb (2^{40} bytes).

41. Uzual, termenul *byte* desemnează atât locaţia de memorie, cât şi informaţia conţinută.

5. Reprezentarea datelor în memorie

Oamenii comunică prin intermediul unor simboluri. Din aceste simboluri (de exemplu, literele alfabetului), respectând anumite reguli sintactice, se pot obţine construcţii mai ample (de exemplu, cuvinte, propoziţii, fraze), purtătoare de informaţie. Dar mintea umană este mult mai complexă decât un calculator. Calculatorul nu este capabil să utilizeze decât două simboluri (0 şi 1). Pentru a reuşi să comunice cu un calculator, omul trebuie să convertească datele şi comenzile sale într-o formă pe care calculatorul să o poată înţelege (formă binară). O astfel de operaţie se numeşte *codificare*.

Codificarea caracterelor

Codificarea caracterelor constă în a asocia în mod unic fiecărui caracter (litere, cifre, semne speciale) o valoare numerică.

Cel mai utilizat cod alfanumeric este codul *ASCII* (*A*merican *S*tandard *C*ode for *I*nformation *I*nterchange), prin care se asociază caracterelor un cod format din 7 cifre binare, deci o valoare numerică din mulţimea {0, 1, 2, ..., 127}. De exemplu, codul caracterului 'A' este 65. Pentru reprezentarea unor simboluri suplimentare (caractere specifice altor limbi, simboluri matematice, simboluri grafice), se utilizează codul *ASCII extins*. Codul ASCII extins utilizează 8 biţi pentru codificarea caracterelor, permiţând astfel codificarea a 256 de simboluri distincte.

Codificarea datelor numerice

Modul de codificare a datelor numerice şi dimensiunea zonei de memorie necesare depind de tipul acestora.

1. Codificarea numerelor naturale

Numerele naturale sunt codificate prin reprezentarea lor binară. Dimensiunea reprezentării este fixă, fiind utilizate 8, 16, 32 sau 64 de poziţii binare, în funcţie de domeniul de valori.

2. Codificarea numerelor întregi

Numerele întregi se reprezintă în *cod complementar*. Dimensiunea reprezentării poate fi de 8, 16, 32 sau 64 de poziţii binare, în funcţie de domeniul de valori de reprezentat:

Numerele întregi pozitive sunt deci codificate prin reprezentarea lor binară.

Numerele întregi negative se codifică prin complementarierea codificării valorii lor absolute faţă de 2^x, unde x reprezintă dimensiunea reprezentării.

Observaţi că prima poziţie din reprezentare precizează semnul (primul bit este 0 pentru numere pozitive, respectiv 1 pentru numere negative).

Exemplu

Pentru a reprezenta în cod complementar numărul întreg -125, se reprezintă în binar valoarea sa absolută (125), apoi se calculează complementul faţă de 2^8 (prin scădere binară din 2^8), 8 poziţii binare fiind suficiente pentru a reprezenta numărul -125.

```
1 0 0 0 0 0 0 0-
  0 1 1 1 1 1 0 1
  1 0 0 0 0 0 1 1
```

3. Codificarea numerelor raționale

Numerele raționale se codifică uzual folosind reprezentarea în *virgulă mobilă*, utilizând 32 de biți, 64 de biți sau 80 de biți. Pentru a reprezenta un număr rațional în virgulă mobilă se convertește numărul în binar, apoi se aduce în formă normalizată. Pentru a aduce numărul în formă normalizată se înmulțește numărul cu o putere convenabilă a lui 2 (baza sistemului de numerație), astfel încât partea întreagă a numărului să fie 1. Deci, ca urmare a operației de normalizare, obținem: $N = \pm 1, a_1a_2...a_n \cdot 2^{\pm e}$.

Exemplu

Să considerăm numărul rațional $10,625_{(10)}$. Convertind numărul în baza 2, obținem: $1010,101$. Pentru a aduce numărul în formă normalizată trebuie să deplasăm trei cifre de la partea întreagă la partea fracționară, ceea ce echivalează cu o împărțire cu 2^3. Deci: $10,625_{(10)}=1010,101_{(2)}= 1,010101_{(2)} \cdot 2^3$.

Reprezentarea numerelor raționale în *virgulă mobilă simplă precizie* (32 de biți) are următoarea structură:

- bitul cel mai semnificativ precizează semnul numărului rațional;
- următorii 8 biți reprezintă caracteristica;
- următorii 23 de biți reprezintă partea fracționară a numărului rațional în formă normalizată.

Deoarece exponentul bazei de numerație în forma normalizată poate fi și negativ, pentru a nu mai utiliza o poziție specială pentru semnul exponentului, în reprezentarea în virgulă mobilă se utilizează o mărime denumită *caracteristică* $C=e+127$. Dacă $C \in \{0, 1, ..., 126\}$, exponentul e este negativ; dacă $C \in \{127, ..., 255\}$, exponentul este pozitiv. Dacă partea fracționară a numărului rațional nu necesită 23 de poziții binare, alinierea se face la stânga, completându-se, eventual, restul zonei de memorie cu zerouri nesemnificative.

Probleme speciale apar la reprezentarea numărului rațional `0.0`, care nu poate fi adus în formă normalizată. În acest caz, caracteristica este considerată 0, iar partea fracționară de asemenea 0, semnul fiind + sau −. Deci 0 admite două reprezentări (+0 și −0). Din acest motiv, valoarea −127 nu face parte din domeniul de valori valide ale exponentului, deși ar putea fi reprezentată.

6. Etapele dezvoltării programelor

Dezvoltarea unei aplicații comportă mai multe etape:

1. *Editarea textului programului.* Pentru această operație este necesar un *editor de texte*. Rezultatul este un fișier sursă, care are extensia `.cpp` (pentru C++) sau `.c` (pentru C).
2. *Compilarea programului.* Pentru această operație se lansează în execuție un program special, denumit *compilator*. Compilatorul analizează textul sursă al programului din punct de vedere sintactic, semnalând eventualele erori. Dacă programul nu conține erori sintactice, compilatorul traduce acest program din limbajul de programare utilizat de programator în singurul limbaj „înțeles" de calculator – limbajul-mașină. Rezultatul compilării este un fișier denumit fișier obiect, care are extensia `.obj` (sub Windows/DOS) sau `.o` (sub Linux).

3. *Editarea de legături.* Această operație este realizată de un program special denumit *editor de legături* (sau *link-editor*). Editorul de legături asamblează toate modulele obiect din care este constituită aplicația, rezultând programul executabil (sub Windows/DOS, un fișier cu extensia `.exe`).

Soluții și indicații

1. Elemente de bază ale limbajului C/C++

1. `a=13; b=7`. **2.** `c=3412`. **3.** `-78`. **4. d. 5. a.** La declarare, nu se pot inițializa simultan două variabile. **b.** Variabila `x` nu este definită. **d.** La declarare, variabilele se separă prin virgulă. **f.** Nu există tipul `unsigned float`. **6. c. 7. e. 8. a.** `2.5`; **b.** `32767`; **c.** `-1`; **d.** `8`; **e.** `1`; **f.** `da`; **g.** `16`; **h.** `61`. **9. a.** Operatorul `%` nu se aplică operanzilor reali. **b.** Ambiguitate: se adună `a` cu valoarea incrementată a lui `x` sau se adună `a` cu `x` și apoi se incrementează valoarea lui `a`? **c.** Între operanzii `4`, `a` și `c` (sau `ac`) lipsesc operatorii, deci `4ac` este interpretat ca nume incorect de variabilă. **d.** Operatorii logici pe biți nu se aplică operanzilor reali. **e.** Variabila `c` este de tip `char`, nu i se poate atribui un șir de caractere. **f.** Nu există operatorul `&&=`; **10. f. 11. c. 12. a** și **b. 13. c. 14. d. 15. d. 16.** `c=(c>='a'&&c<='z'? c-'a'+'A':c)` **17. a.** `x<<=n;` **b.** `x>>=n;` **c.** `x&1;` **d.** `x&1<<n;` **e.** `x&=~(1<<n);` **f.** `x|=1<<n`.

18.

```
#include <iostream.h>
#include <math.h>
int main()
{ float l; cout << "Latura="; cin >> l;
  cout << "Inaltimea=" << l*sqrt(3)/2 << endl;
  cout << "Aria=" << l*l*sqrt(3)/4 << endl; return 0;}
```

19. `aux=x1; x1=x2; x2=x3; x3=x4; x4=x5; x5=aux;`

20.

```
/* transform distanta in metri si timpul in secunde */
D=D*1000; H=H*3600;
v=D/H;
```

21. Utilizăm formula vitezei `v=s/t` ⇒ `t=s/v`. Cei doi colegi se întâlnesc după același interval de timp, în care primul a parcurs distanța `d1`, iar cel de-al doilea distanța `D-d1`. Obținem un sistem de ecuații prin rezolvarea căruia determinăm `d1` și `t`:

```
d1=v1*D/(v1+v2); t=d1/v1;
```

22. `L=sqrt((xA-xB)*(xA-xB)+(yA-yB)*(yA-yB));`

23. `cout<<(x<=10?2>x*x?sqrt(2-x*x):sqrt(x*x-2):(-2*x+1)/3);`

2. Instrucțiunile limbajului C/C++

2. Valoarea afișată este `3`. **3.** `5`.

4.

```
int a, b;
cin>>a>>b;
if (a-b==1) || (b-a==1) cout<<"DA";
   else cout<<"NU";
```


5.

```
float a, b, c, s, p, d;
s=-b/a; p=c/a; d=b*b-4*a*c;
if (d<0) cout<<"Ecuatia nu are radacini reale"<<endl;
   else
    {cout"Ecuatia are radacini reale"<<endl;
     if (d==0) cout<<"Radacinile sunt egale"<<endl;
       else cout<<"Radacinile sunt distincte"<<endl;
     if (p<0) cout<<"Radacinile au semne contrare"<<endl;
       else
         {cout<<"Radacinile sunt "<<endl;
          if (s>0) cout<<"pozitive"<<endl;
            else
              if (s<0) cout<<"negative"<<endl;
                else cout<<"nule"<<endl;
         }
    }
```

6.

```
unsigned x, c1, c2, c3;
cin>>x;
c3←x%10; c2←x/10%10; c1←x/100; /* extrag cele 3 cifre ale numarului */
if (c1<c2) cout<<c2<<c3;
   else if (c2<c3) cout<<c1<<c3;
           else cout<<c1<<c2;
```

7.

```
char c1, c2, c3;
cin>>c1>>c2>>c3;
if (c1==c2) && (c2==c3) nr=1;
   else if (c1==c3) || (c1==c2) || (c2==c3) nr=2;
          else nr=3;
 cout<<nr;
```

8.

```
float ag, am, as, bg, bm, bs, cg, cm, cs;
cin>>ag>>am>>as>>bg>>bm>>bs;
cs=as+bs; cm=0; cg=0;
if (cs>60) {cs=cs-60; cm=1;}
cm=cm+am+bm;
if (cm>60) {cm=cm-60; cg=1;}
cg=cg+ag+bg;
if (cg>360) cg=cg-360;
cout<<cg<< "grade "<<cm<<" minute "<<cs<<" secunde";
```

10.

```
{float x, y;
 cin>>x>>y;
 if (x>0 && y>0) cout<<(x+y)/(2*x*y);
   else
     if (x==0 || y==0)
        if (x>y) cout<<x;
           else cout<<y;
        else cout<<(1/x+1/y)*(1/x-1/y+2*x*y+x*x+y*y);}
```

11. Conform restricțiilor problemei există doar 3 variante în care pot fi vizitate cele 3 orașe (`ABC`, `ACB`, `BAC`). Vom calcula lungimea fiecărui traseu posibil și vom afișa traseul cel mai scurt.

```
{unsigned a, b, c;
 cin>>a>>b>>c;
 if (a+b<c+b && a+b<a+c) cout<<"ABC";
   else if (b+c<a+b && b+c<a+c) cout<<"ACB";
          else cout<<"BAC";}
```

Observați că nu am utilizat nici o variabilă suplimentară.

12.
```
float a1, b1, c1, a2, b2, c2, x, y;
cin>>a1>>b1>>c1>>a2>>b2>>c2;
if (a1==0)
   if (b1==0)
     if (c1==0)
       cout<<"x si y pot avea orice valoare reala";
     else cout<<"Sistemul nu are solutii";
   else     /* a1=0, b1≠0 */
     if (a2==0)
       if (c1*b2==c2*b1)
         cout<<"y="<<c1/b1<<" x orice valoare reala";
       else cout<<"Sistemul nu are solutii";
     else /* a2≠0 */
       cout<<"x="<<(c2*b1-c1*b2)/(a2*b1)<<" y="<<c1/b1;
else     /* a1≠0 */
  if (a1*b2==a2*b1)
    if (a1*c2==c1*a2)
      cout<<"x si y pot avea orice valoare reala";
    else
      cout<<"Sistemul nu are solutii";
  else
    {y=(c2*a1-a2*c1)/(b2*a1-a2*b1);
     x=(c1-b1*y)/a1;
     cout<<"x= "<<x<<" y="<<y;}
```

13.
```
{float m1, m2, m3, m4, m5, mg;
 cin>>m1>>m2>>m3>>m4>>m5;
 if (m1<5 || m2<5 || m3<5 || m4<5 || m5<5)
   cout<<"Nepromovat";
   else
     {mg=(m1+m2+m3+m4+m5)/5;
      if (mg<6) cout<<"Nepromovat";
        else cout<<"Promovat";}
}
```

14.
```
{unsigned h1, h2, h3;
 cin>>h1>>h2>>h3;
 if (h1≤h2)
   if (h2≤h3) cout<<"Ionel Gigel Danut"
   else  /* h3<h2 */
     if (h1≤h3) cout<<"Ionel Danut Gigel"
     else /* h3<h1 */
       cout<<"Danut Ionel Gigel"
 else /* h2<h1 */
   if (h1≤h3) cout<<"Gigel Ionel Danut";
   else /*h2<h1, h3<h1 */
     if (h2≤h3) cout<<"Gigel Danut Ionel"
     else /* h3<h2<h1*/
       cout<<"Danut Gigel Ionel";}
```

15. a. 3; **b.** 4321; **c.** Valoarea lui n este scăzută succesiv cu 1 şi afişată, cât timp n≠0. Prin scăderi succesive cu 1 se va ajunge la situaţia în care se va scădea 1 din -32768, obţinându-se +32767, care se va scădea în continuare cu 1 până la 0. Deci, se vor afişa pe ecran 65535 de numere; **d.** 051015; **e.** 01 02 03 04 12 13 14 23 24 34; **f.** ciclul este infinit, afişându-se pe ecran numai numere impare ; **g.** -4; **h.** afişează primele n numere Fibonacci: 1 1 2 3 5 8 13 21. **16. a** şi **c. 17. e** şi **f. 18.** a, d şi f. **19.** 30. **20.** b şi c. **21.** a, b şi d. **22.** c. **23.** d. **24.** c. **25.** d. **26. a.** 4368 **b.** algoritmul extrage succesiv cifrele din numerele a şi b, până când unul dintre numere se termină; la fiecare pas compară ultima cifră din a cu ultima cifră din b, o alege pe cea mai mare şi o adaugă la sfârşitul numărului

construit în variabila c, prin urmare o soluţie posibilă ar fi 8291; **c.** o soluţie posibilă ar fi a=10, b=10 (sau orice putere a lui 10); **27. a.** 5; **b.** orice număr pentru care cel mai mare divizor prim este 11, de exemplu 11 sau 33; **c.** calculează cel mai mare divizor prim al lui n. **28. a.** 78; **b.** 9; **c.** numărul n, considerat a fi scris în baza b, este convertit în baza 10. **29. a.** 55; **b.** a=4 şi b=7; **c.** orice pereche de valori, pentru care a>b; **d.** în variabila x se calculează suma numerelor din intervalul [a,b]; calculul acestei valori se poate realiza direct, astfel:
```
if (a<=b) x=b*(b+1)/2-a*(a-1)/2;
  else x=0;
```
30. d. **31.** c şi d. **32.** 5. **33.** Variabila m nu a fost iniţializată cu 0. Ciclul este infinit deoarece se execută numai instrucţiunea m++; (dacă am fi dorit să se execute şi instrucţiunea n/=p; ar fi trebuit să scriem {m++; n/=p;}, formând astfel o instrucţiune compusă).

35.
```
unsigned a,b,nr,sa,sb,d;
nr=0;     /*numarul de perechi determinate este initial 0*/
a=2;
while (nr<3)             /* nu am obtinut inca 3 perechi */
   {sa=1;               /* calculez suma divizorilor lui a */
    for (d=2; d<=a/2; d++)
      if (a%d==0)              /* d este divizor al lui a */
        sa+=d;                       /* il adunam la suma */
    b=sa;
      /* b trebuie sa fie egal cu suma divizorilor lui a */
    if (b>a)
      {sb=1;           /* calculez suma divizorilor lui b */
       for (d=2; d<=b/2; d++)
         if (b%d==0)                  /*d divizor al lui b*/
           sb=sb+d;                  /* il adunam la suma */
       if (a==sb)/*am obtinut o pereche de numere prietene*/
         {cout<<a<<' '<<b<<endl;              /* o afisez */
          nr++;}                            /* si o numar */
      }
    a++;                             /* caut in continuare */
   }
```

41. Voi citi succesiv valorile date şi pentru fiecare valoare citită voi extrage cifrele numărând în variabila nr0 câte cifre egale cu 0 am obţinut. **42.** Vom aplica repetat algoritmul de calcul al sumei cifrelor unui număr. **44.** Nu vom parcurge toate numerele naturale mai mici decât n testând la fiecare pas dacă numărul curent este sau nu pătrat perfect, ci vom genera direct pătratele perfecte. **46.** Vom construi toate fracţiile care au numitorul şi numărătorul în mulţimea $(1,2,\ldots,n)$ şi le vom număra numai pe cele ireductibile. **47.** În mod similar generării celui de-al n-lea termen al şirului Fibonacci, vom utiliza 4 variabile: s0, s1, s2 reţin 3 termeni consecutivi din şir, iar în s3 vom calcula termenul următor; după care ne pregătim pentru pasul următor, deplasându-ne în şir s0=s1; s1=s2; s2=s3; **48.** Calculăm mai întâi MO media la oral, apoi media semestrială se obţine după formula (3*MO+Teza)/4. **49.** Pentru a calcula c.m.m.d.c. a n valori ne bazăm pe faptul că operaţia c.m.m.d.c. este asociativă, adică $cmmdc(a_1,\ldots,a_{n-1},a_n)=cmmdc(cmmdc(a_1,\ldots,a_{n-1}),a_n)$. Vom utiliza o variabilă d în care vom memora la fiecare pas c.m.m.d.c. al valorilor citite până la momentul respectiv. Variabila d va fi iniţializată cu prima valoare citită. La fiecare pas citim o nouă valoare şi calculăm c.m.m.d.c. dintre valoarea citită şi d prin algoritmul lui Euclid. **50.** Citim succesiv caracterele în aceeaşi variabilă c, până la întâlnirea caracterului spaţiu. Vom utiliza o variabilă logică (Este) care va avea valoarea adevarat dacă secvenţa de caractere poate fi un număr natural şi fals, altfel. La întâlnirea unui caracter diferit de cifră, variabila Este va primi valoarea fals. Singura modificare pe care trebuie să o facem pentru ca algoritmul să verifice dacă secvenţa este un număr întreg se referă la primul caracter din secvenţă (care ar putea fi şi '+' sau '-'). Pentru ca numărul să fie real, acceptăm şi un singur caracter punct.

51. Citesc propoziția caracter cu caracter, memorând succesiv caracterele citite în aceeași variabilă c:

```
char c;
unsigned nr=0;                          /*numarul de cuvinte*/
do
  {do c=getche(); while (c==' '); /*sar peste spatii */
   if (c!='.')                    /*incepe un cuvant */
     {nr++;                       /*il numar  si sar peste cuvant*/
      while (c!=' ' && c!='.') c=getche();}
  } while (c!='.');
cout<<"nr = "<<nr<<endl;
```

52.

```
{int n, x, r, i;
 cout<<"x, r, n= "; cin>>x>>r>>n;
 for (i=0; i<n; i++) cout<<x+i*r<<' '; }
```

53. Trebuie să calculăm c.m.m.m.c. (1, 2, ..., n); Folosim faptul că c.m.m.m.c.(x,y)=x*y/c.m.m.d.c. (x, y) și asociativitatea c.m.m.m.c.(a_1, a_2, ..., a_n) = c.m.m.m.c.(c.m.m.m.c.(a_1, a_2, ..., a_{n-1}), a_n).

```
int n, i, cmmmc=1, r, x, y;
cout<<"n= "; cin>>n;
for (i=2; i<=n; i++)
     {x=i; y=cmmmc;
      while(x) {r=y%x; y=x; x=r;}
      cmmmc=i*cmmmc/y;}
cout<<cmmmc<<endl;
```

54. Se citesc succesiv cele n numere naturale și se determină pentru fiecare număr citit numărul de divizori primi. Se reține numărul cu număr maxim de divizori primi.

```
{int n, nr_max, nr_div_max=0, snr, d, nr, i, nrd;
 cout<<"n= "; cin>>n;
 for (i=1; i<=n; i++)
   {cout<<"nr="; cin>>nr;              //citesc o noua valoare
    for (snr=nr,d=2,nrd=0;nr>1;d++)    //descompun in factori primi
      if (nr%d==0)                     //am gasit un divizor prim
        {nrd++;                        //il numar
         while (nr%d==0) nr/=d;}
    if (nrd>nr_div_max) nr_max=snr, nr_div_max=nrd; }
cout<<"Numarul maxim de divizori="<<nr_div_max<<endl;
cout<<"Numarul= "<< nr_max;}
```

57. Datorită faptului că n este mare, nu putem calcula produsul celor n numere, pentru ca apoi să numărăm câte cifre 0 are la sfârșit. Dar un 0 final nu se poate obține decât prin înmulțirea lui 2 cu 5, vom calcula pe parcursul citirii multiplicitatea lui 2 (m2) și multiplicitatea lui 5 (m5) în produs. Numărul de zerouri finale este egal cu min(m2, m5). **58.** Fiecare număr x cuprins între 3 și n se testează dacă este sau nu prim. Dacă este prim, se calculează câte cifre are x și se adună la numărul total de cifre pe care trebuie să le scrie bibliotecarul. Dacă x nu este prim, se mărește cu 1 numărul de pagini nemâzgălite.

59. Programul conține trei cicluri for imbricate; contorul primului ciclu reprezintă prima cifră (1... 7), al doilea contor cifra a doua (2... 8), iar ultimul cifra a treia (3... 9).

```
int c1, c2, c3;
for (c1=1; c1<8; c1++)                    //prima cifra
    for (c2=c1+1; c2<9; c2++)             //cifra a doua
        for (c3=c2+1; c3<10; c3++)        //cifra a treia
            cout <<c1<<c2<<c3<<' ';
```

Numerele „bine ordonate descrescător":

```
for (c1=9; c1>1; c1--)                    //prima cifra
  for (c2=c1-1; c2>0; c2--)               //cifra a doua
    for (c3=c2-1; c3>=0; c3--)            //cifra a treia
      cout <<c1<<c2<<c3<<' ';
```


61.

```
int n;
cout << "n= "; cin >> n;
while (n>=1000) {cout <<'M';n-=1000;} //scriu miile
if (n>=900) {cout <<"CM";n-=900;}     //cazul 900
if (n>=500) {cout <<'D';n-=500;}      //cazul 500
if (n>=400) {cout <<"CD";n-=400;}     //cazul 400
while (n>=100) {cout <<'C';n-=100;}   //scriu sutele
if (n>=90) {cout <<"XC";n-=90;}       //cazul 90
if (n>=50) {cout <<'L';n-=50;}        //cazul 50
if (n>=40) {cout <<"XL";n-=40;}       //cazul 40
while (n>=10) {cout <<'X';n-=10;}     //scriu zecile
if (n==9) {cout <<"IX";n-=9;}         //cazul 9
if (n>=5) {cout <<'V';n-=5;}          //cazul 5
if (n==4) {cout <<"IV";n-=4;}         //cazul 4
while (n>0) {cout <<'I';n-=1;}        //scriu unitatile
```

62. Calculăm valoarea celei mai mari bancnote pe care o putem utiliza pentru a plăti suma S. Utilizăm numărul maxim de bancnote de valoare maximă, apoi reluăm procedeul pentru restul sumei.

```
int S, x, p=1;
cout << "S= "; cin >> S;  cout << "x= "; cin >> x;
while (p*x<=S) p*=x;          //valoarea celei mai mari bancnote
while (S)                     //mai avem de platit
      {if (S/p)               //pot utiliza bancnote cu valoarea p
         {cout<<S/p<<" bancnote cu valoarea "<<p<<endl;
          S%=p;}              //rest de plata
       p/=x;}                 //trec la bancnote mai mici
```

64. Observăm că dacă un subordonat are un bit cu valoarea 1, orice șef trebuie să aibă bitul corespunzător tot pe valoarea 1. Să notăm cu nr_1 numărul de biți 1 din reprezentarea internă a lui n. Atunci numărul de zerouri este nr_0=16-nr_1. Numărul șefilor lui n este egal cu numărul funcțiilor care se pot defini astfel: f:{1,2,...,nr_0}->{0,1}. Prin urmare, numărul șefilor este 2^{nr_0}.

```
unsigned n, nr_1=0, i;
cout<<"n= "; cin>>n;
//calculez numarul de biti 1 in reprezentarea lui n
for (i=0; i<16; i++)  nr_1+=(n>>i)&1;
cout <<"Nr. sefilor lui "<<n<<" este "<<(1u<<(16-nr_1));
```

66. Se poate demonstra matematic că pădurarul din poziția x0, y0 vede pomul din poziția x, y dacă și numai dacă c.m.m.d.c.(|x-x0|, |y-y0|)=1. Prin urmare, este suficient să numărăm câți pomi respectă această condiție>

```
int n, x0, y0, x, y, a, b, r, c=0;
cout << "n, x0, y0= "; cin >> n >> x0 >> y0;
for (x=0; x<n; x++)
    for (y=0; y<n; y++)
        {a = x>x0?x-x0:x0-x; b = y>y0?y-y0:y0-y;
         while (b) {r=a%b;a=b; b=r;}          //calculez cmmdc
         if (a==1) c++;}                      //pom vizibil, il numar
 cout<<"Numarul pomilor vizibili este "<<c<<endl;
```

67. Consider că reprezentarea internă a numerelor mai mici decât n (o succesiune de 0 și 1) reprezintă un număr în baza b, îl convertesc în baza 10 și îl afișez:

```
int n, i, b, nr=0, x, pb;
cout<<"n, b ="; cin >> n >> b;
cout<<0<<' '<<1<<' ';
for (i=2; ; i++)
  {for (nr=0, x=i, pb=1; x; x>>=1, pb*=b) nr+=(x&1)*pb;
     if (nr<=n) cout<<nr<<' '; else break;}
```

3. Fişiere

1. Programul afişează calendarul unei luni cu n zile, a fiind ziua din săptămână în care începe luna.

2.
```
ifstream f("nr.in"); if (!fin) return 1;
char c;  int nr=0, lin=1, v;
while (!f.eof())
   {f.get(c);
    if (f.good())
      if (c=='\n')
        {//s-a terminat o linie;
         cout<<"Pe linia "<<lin<<" sunt "<<nr<<" valori\n";
         nr=0; lin++;}
      else
        {f.putback(c); //repun caracterul citit in stream-ul de intrare
         f>>v; nr++;}        //citesc o valoare si o numar
   }
f.close();
```

3. Parcurg secvenţial cele două fişiere, cât timp caracterele citite sunt egale:
```
ifstream f("f1.in"), g("f2.in");
char c1, c2;
while (f.get(c1) && g.get(c2) && c1==c2);
if (!f.get(c1)&& !f.get(c2)) cout << "DA";
      else cout <<"NU";
f.close(); g.close();
```

4.
```
ifstream f("nr.txt"); ofstream g("pare.txt");
int v;
while (!f.eof())
       {f>>v; //citesc o valoare
        if (f.good() && v%2==0)
           g<<v<<endl;} //daca este para o scriu in g
f.close();
g.close();
```

5. Numărul de linii coincide cu numărul de caractere *newline* din fişier.

4. Tipuri structurate de date

1. c, d. **2.** b, d. **3.** d. **4.** e. **5.** b, d. **6.** f. **7.** c. **8.** c. **9.** d. **10.** e. **11. a.** Şirul s devine `Era Vysile om frumos`; **b.** Va produce eroare deoarece în membrul drept al unei atribuiri nu poate fi o expresie. **12.** Se va afişa `ErayVayily omxxxumos`. **13.** Se va afişa `bdtcebectdb`. **14.** b, e, f. **15.** d, e, f.

16.
```
double a[10], b[8];
int i, j, nr=0;
for (i=0; i<10; i++) cin >> a[i];                    //citire a
for (i=0; i<8; i++) cin >> b[i];                     //citire b
for (i=0; i<10; i++)                          //parcurg vectorul a
    {//verific daca a[i] este mai mic decat toate elementele din b
      for (j=0; j<8 && a[i]<b[j]; j++);
          if (j==8) nr++;}                           //il numar
cout << nr;
```

O soluţie mai eficientă ar fi să determinăm minimul din vectorul b, apoi să comparăm elementele vectorului a cu minimul.

18. a.
```
for (ok=1, i=0; ok && i<n; i++)
    {                                    //verific daca a[i] este prim
     for (d=2, rad=sqrt(a[i]); a[i]%d && d<=rad; d++);
         if (a[i]%d==0) ok=0;}                   //a[i] nu este prim
if (ok) cout << "Toate elementele vectorului sunt prime";
   else cout << "Nu toate elementele vectorului sunt prime";
```

b.
```
for (exista=i=0; !exista && i<n; i++)
    {                              //verific daca a[i] este palindrom
      x=a[i]; rx=0;          //determin in rx rasturnatul lui a[i]
      while (x) {rx=rx*10+x%10; x/=10;}
      if (rx==a[i]) exista=1; }              //a[i] este palindrom
if (exista) cout << "Exista un palindrom in vector";
    else  cout << "Nu exista nici un palindrom in vector";
```

c.
```
for (i=0; i<n; )
    if (a[i]%2)//a[i] impar
       {//il elimin
         for (j=i+1; j<n; j++) a[j-1]=a[j];
          n--;}
       else i++;
```

d.
```
for (i=0; i<n; i++)
    if (!a[i])
       {for (j=n; j>i; j--) a[j]=a[j-1]; //fac loc
         n++; //creste numarul de elemente
        a[i+1]=1;}
```

e.
```
int i, lg=1, lgmax=0, emax=a[0], inc=0, incmax=0;
for (i=1; i<n; i++)
    if (a[i]==a[i-1])lg++; //marim lungimea secventei curente
       else
       { if (lg>lgmax) //compar cu secventa de lungime maxima
           {lgmax=lg; incmax=inc; emax=a[i-1];}
         lg=1; inc=i;} //incepe o noua secventa
cout<<lgmax<<' '<<incmax<<' '<<emax;
```

f. Calculez numărul de apariţii pentru fiecare element din vector şi determin maximul.

19. a.
```
for (exista=j=0; !exista && j<m; j++)
    {//verific daca coloana j are toate elementele nule
     for (i=0; i<n && !a[i][j]; i++);
     if (i==n) exista =1;}                    //coloana j este nula
if (exista) cout <<"Exista o coloana cu toate elementele nule!\n";
   else cout<<"Nu exista o coloana cu toate elementele nule!\n";
```

b.
```
for (i=0;i<n; i++)
    {//calculez produsul elementelor nenule de pe linia i
     for (p=1,j=0; j<m; j++)
         if (a[i][j]) p*=a[i][j];
      cout<<"Produsul elem. nenule de pe linia "<<i<<"="<< p;}
```

c. Citim elementele matricei începând cu linia 1, coloana 1 (lăsam o „bordură" pentru ca orice element din matrice să aibă 4 vecini. Parcurgem apoi matricea şi pentru fiecare element testăm cei 4 vecini; dacă toţi 4 sunt numere pare, numărăm acest element.
```
#include <fstream.h>
#define NVecini 4
#define DMax 102
int a[DMax][DMax], n, m;
int dl[NVecini]={-1,0,1, 0};
```

```
int dc[NVecini]={ 0,1,0,-1};
int main()
{int i, j, k, nr, total=0;
 ifstream fin("matrice.in");
 fin>>n>>m;
 for (i=1; i<=n; i++) for (j=1; j<=m; j++) fin>>a[i][j];
 fin.close();
 for (i=1; i<=n; i++)
    for (j=1; j<=m; j++)
        {//parcurg vecinii si-i numar pe cei pari
         for (nr=0, k=0; k<NVecini; k++)
             if (a[i+dl[k]][j+dc[k]]%2==0) nr++;
         //daca toti vecinii sunt pari, numar acest element
         if (nr==NVecini) total++;    }
 cout<<total<<endl; return 0;}
```

20. a.

```
//parcurg elementele de deasupra diagonalei principale
for (nule=1,i=0; nule && i<n; i++)
     for (j=i+1; nule && j<n; j++)
          if (a[i][j]) nule=0;
if (nule) cout <<"Matricea este inferior triunghiulara\n";
   else cout <<"Matricea nu este inferior triunghiulara\n";
```

b.

```
//parcurg elementele de sub diagonala principala
for (sim=1, i=0; sim && i<n; i++)
    for (j=0; sim && j<i; j++)
        if (a[i][j]!=a[j][i]) //il compar cu simetricul sau
             sim=0;
if (sim) cout <<"Matricea este simetrica\n";
    else cout <<"Matricea nu este simetrica\n";
```

d. Numărul de chenare concentrice într-o matrice pătratică cu n linii este n/2 pentru n par sau (n+1)/2 pentru cazul n impar (există în mijloc un chenar degenerat, format dintr-un singur element). Chenarul c are colțul stânga-sus (c,c) și colțul dreapta jos (n-1-c, n-1-d). Pentru a calcula suma elementelor de pe un chenar nu este necesar să parcurgem separat fiecare latură, este suficient să parcurgem o singură latură, dar să adunăm patru elemente deodată (câte unul pentru fiecare dintre cele patru laturi). Singura problemă este că în acest mod adunăm colțurile de două ori.

```
for (c=0; c<n/2; c++)
   {for(s=0, i=c; i<n-c; i++)
       s+=a[c][i]+a[i][c]+a[c][n-1-c]+a[n-1-c][i];
    //scad colturile
    s-=a[c][c]+a[n-1-c][c]+a[c][n-1-c]+a[n-1-c][n-1-c];
    cout<<"Suma pentru chenarul "<<c<<" = "<<s<<endl; }
if (n%2) //scriu chenarul din mijloc
   cout<<"Suma pentru chenarul "<<n/2<<" = "<<a[n/2][n/2]<<endl;
```

21. Parcurg fișierul secvențial, calculând în vectorul l frecvența de apariție a fiecărei litere. Verific apoi dacă în vectorul l există o componentă egală cu k.

```
#include <fstream.h>
int l[26], k;
void main()
{ ifstream f("bac2000.txt");
  char c;
  cout<<"k="; cin>>k;
  while (f.get(c))
        if (c!=' ')    //c este litera mare
           l[c-'A']++; //maresc frecventa de aparitie a literei c
  f.close();
  for (int i=0; i<26; i++)
      if (l[i]==k) {cout <<"DA"; return;}
  cout<<"NU"; }
```


23. Vom completa în primul rând ultima linie și ultima coloană cu 1. Apoi vom calcula celelalte elemente în ordine de jos în sus și de la dreapta spre stânga.

```
for (i=0; i<n; i++)  a[i][n-1]=a[n-1][i]=1;
for (i=n-2; i>=0; i--)
    for(j=n-2; j>=0; j--)
       a[i][j]=a[i+1][j]+a[i][j+1];
```

25. Vom reprezenta un polinom ca un vector în care reținem coeficienții săi, în ordinea crescătoare a gradelor. Pentru a fi automat inițializate cu 0, declarăm polinoamele în afara funcției main().

```
#include <iostream.h>
typedef double Polinom[100];
Polinom P, Q, S, M;
int Pgrad, Qgrad, Sgrad, Mgrad; //gradele polinoamelor
int main ()
{int i, gmax, j;
 cout << "\nGradul polinomului P = "; cin >> Pgrad;
 cout<<"\nCoeficientii polinomului P, in ordinea gradelor:";
 for (i=0; i<=Pgrad; i++) cin >> P[i];
 cout << "\nGradul polinomului Q = "; cin >> Qgrad;
 cout<<"\nCoeficientii polinomului Q, in ordinea gradelor:";
 for (i=0; i<=Qgrad; i++) cin >> Q[i];
 gmax = Pgrad>Qgrad ? Pgrad: Qgrad;
 for (i=0; i<=gmax;i++) S[i]=P[i]+Q[i];                    //suma
 for (; gmax && !S[gmax]; gmax--);
 Sgrad=gmax;
 cout<<"\nPolinomul suma are coeficientii ";
 for (i=0; i <= Sgrad; i++) cout << S[i] <<' ';
 Mgrad=Pgrad+Qgrad;                                        //produsul
 for (i=0; i<=Pgrad; i++)
    for (j=0; j<=Qgrad; j++)  M[i+j]+=P[i]*Q[j];
cout<<"\nPolinomul produs are coeficientii ";
for (i=0; i <= Mgrad; i++) cout <<M[i]<<' '; return 0;}
```

26. Ordonăm crescător elevii după înălțime și schiurile după lungime, apoi distribuim schiurile în această ordine.

29. Se sortează crescător elementele vectorului. Dacă numărul de elemente este par, ultimul element se ignoră. Se parcurge prima jumătate a vectorului, afișând alternativ un element din prima jumătate și un element din cea de-a doua. La sfârșit se mai afișează doar mijlocul.

30.

```
#include <fstream.h>
int a[100][100], n, p[100];
int main()
{ ifstream f("relatii.in");
  int o[100], i, j, aux, ok, nr;
  f>>n;              //citesc numarul de elevi si matricea relatiilor
  for (i=0; i<n; i++)
      for (j=0; j<n; j++) f>>a[i][j];
  f.close();
  cout <<"Perechile de elevi care se agreeaza reciproc:"<<endl;
  for (i=0; i<n; i++)
      for (j=i+1; j<n; j++)
          if (a[i][j] && a[j][i]) cout <<i+1<<' '<<j+1<<endl;
  cout<<"Elevii care nu sunt agreati de nimeni:";
  for (i=0; i<n; i++)
      {for(j=0;j<n; j++) //numar cati elevi agreeaza elevul i
         p[i]+=a[j][i];
       if (p[i]==1) cout<<i+1<<' ';}
  cout<<endl;
  cout<<"Elevii care nu agreeaza pe nimeni: ";
  for (i=0; i<n; i++)
      {for(nr=j=0; j<n; j++) //numar elevii agreati de elevul i
         nr+=a[i][j];
```

```
      if (nr==1) cout<<i+1<<' ';}
  cout<<"\nElevii in ordinea descrescatoare a popularitatii:\n";
  //determin ordinea elevilor in functie de popularitate
  for (i=0; i<n; i++) o[i]=i+1;
  do {ok=1;
      for (i=0; i<n-1; i++)
          if (p[o[i]-1]<p[o[i+1]-1])
             {aux=o[i];o[i]=o[i+1];o[i+1]=aux;ok=0;}
      } while (!ok);
  for (i=0; i<n; i++) cout<<o[i]<<' '; cout<<endl; return 0;}
```

31.

```
#include <iostream.h>
int main ()
{char p[200];  int i;
 cin.getline(p,200,'.');                          //citesc propozitia
 for (i=0; p[i]; i++)                            //parcurg propozitia
     if ('a'<=p[i] && p[i]<='z')                   //este litera mica
        p[i]=p[i]-'a'+'A';                  //o convertesc in majuscula
  cout <<p<<endl; return 0;}                       //afisez propozitia
```

32.

```
#include<iostream.h>
#include<string.h>
int main()
{char c[30], p[1000], *poz=p;
 cin.getline(c, sizeof(c)); cin.getline(p, sizeof(p));
 for (int n=strlen(c), i=0; poz && i<n; i++) poz=strchr(poz,c[i]);
 if (poz) cout<<"Literele cuvantului apar in ordine in propozitie";
    else cout<<"Literele cuvantului nu apar in ordine in propozitie"; }
```

33.

```
#include <iostream.h>
#include <string.h>
void main ()
{ int n, i, j,st, dr, gasit, mijloc, rez;
  struct Persoana {char nume[40];
                   unsigned long tel;} Ag[200], v;
  char x[40];
  cout <<"n="; cin >> n;
cout << "Introduceti informatiile\n";//sortam de la citire, prin insertie
for (i=0; i<n; i++)
    {cout << "Numele si prenumele: ";  cin.get();cin.getline(v.nume,40);
     cout << "Telefonul: "; cin>>v.tel;
     //inserez informatiile citite in agenda, pe pozitia corecta
     for (j=i; j && strcmp (Ag[j-1].nume, v.nume)>0; j--)
         Ag[j]=Ag[j-1];
     Ag[j]=v;}
cout <<"Numele persoanei cautate:";  cin.get();cin.getline(x, 40);
for (st=0,dr=n-1, gasit=0; !gasit && st<=dr; )            //cautare binara
    {mijloc=(st+dr)/2;
     rez= strcmp(Ag[mijloc].nume, x);
     if (!rez) gasit=1;
        else if (rez>0) dr=mijloc-1;
                else st=mijloc+1;}
if (gasit) cout <<"Telefon: "<<Ag[mijloc].tel;
   else cout <<"Persoana "<<x<<" nu exista in agenda"; }
```

34.

```
#include <iostream.h>
#include <string.h>
int main ()
{ char t[1000], c[20], *p=t, *rest;
  cin.getline(c,20);  cin.getline(t,1000);
```



```
  do
  { p=strstr(p,c);          //caut c in t, incepand cu pozitia p
    if (p)                  //c apare in t pe pozitia p, il sterg
       { rest=p+strlen(c); *p=NULL;
         strcat(t,rest);}
  } while(p);
  cout <<t<<endl; return 0;}
```

35. Vom utiliza principiul cutiei al lui Dirichlet: construim numerele 1, 11, 111, 1111, ..., 11...11 (n de 1). Sunt n numere distincte, care au toate cifrele egale cu 1. Calculăm resturile împărțirii numerelor la n. Dacă există un număr pentru care restul este egal cu 0, acela este multiplul căutat. Dacă nu, deducem că cele n resturi iau valori în mulțimea {1, 2, ..., n-1}, deci există două resturi egale. Prin urmare, există două astfel de numere, a căror diferență este divizibilă cu n. Diferența lor este multiplul căutat.

36. Vom spune că o lucrare este *restantă* dacă execuția ei se termină după termenul de predare; altfel, o vom numi *în termen*. Observăm că orice planificare poate fi adusă la o formă echivalentă (din punctul de vedere al penalizării totale) în care toate lucrările *în termen* precedă lucrările *restante*. Mai mult, orice planificare poate fi adusă în *formă canonică*, astfel încât lucrările *în termen* să preceadă lucrările *restante*, iar lucrările *în termen* să fie în ordinea crescătoare a termenelor de predare (dacă într-o planificare oarecare L, lucrarea în termen L_i are termenul de predare ulterior termenului de predare a lucrării în termen L_j, i<j, putem inversa în planificare execuția lucrărilor L_i și L_j, acestea rămânând în termen). Deci problema determinării unei planificări optimale se reduce la a determina lucrările în termen în ordinea crescătoare a termenelor de predare. Vom ordona lucrările descrescător după penalizări și vom construi mulțimea L a lucrărilor în termen selectând la fiecare pas prima lucrare (deci cea a cărei întârziere ar produce penalizarea cea mai mare), care poate fi executată în termen până la momentul curent, inserând în planificare lucrarea astfel încât lucrările să fie în ordinea crescătoare a termenelor de predare. Utilizăm următoarele variabile:

```
int  n, nt, nr;
//n - nr. de lucrari;
// nt - nr. de lucrari "in termen";
//nr - numarul de restante
int t[NMax];          //termenele de predare a lucrarilor
double p[NMax];       //penalizarile pentru restante
int o[NMax];          //ordinea lucrarilor in functie de penalizari
int R[LNMax]          //multimea lucrarilor restante
int L[NMax];          //multimea lucrarilor "in termen"
```

După citirea și ordonarea lucrărilor descrescător după penalizări, determinăm mulțimea lucrărilor în termen astfel:

```
nt=1; L[0]=o[0];
for (i=1;i<n; i++)
    {//incerc sa inserez lucrarea o[i] in L
    for (j=nt-1;j>=0 && t[L[j]]>(j+1) && t[L[j]] > t[o[i]]; j--);
     //Lucrarea L[j] nu mai poate fi decalata
     j++;
     //verific daca lucrarea o[i] ar fi "in termen" la momentul j
 if (t[o[i]] > j)
    {
       /* inserez lucrarea o[i] in L pe pozitia j,
          decaland celelalte lucrari */
       for (k=nt++;k>j;k--) L[k]=L[k-1];          //am decalat
       L[j]=o[i];}
       else                                      //lucrare restanta
       {ptotal+=p[o[i]]; R[nr++]=o[i];}
    }
```

38. Metoda de programare utilizată este *Greedy*. Cât timp mai există ore de plecare „nerezolvate": punem în funcțiune un nou bac, sortăm timpii de plecare, alegem timpul minim (acesta va reprezenta prima cursă pentru acest bac); simulăm toate cursele pe care le poate rezolva acest bac.

39. Pentru fiecare persoană trebuie să reținem două liste: una formată din persoanele despre care ea

afirmă că spun adevărul, cealaltă din persoanele despre care afirmă că mint. Pentru aceasta vom utiliza două matrice cu n linii şi n coloane. În matricea A vom reţine pe linia i lista persoanelor despre care persoana i afirmă că spun adevărul. În matricea F vom reţine pe linia i lista persoanelor despre care persoana i afirmă că mint.
Observăm că în matricele A şi F diferă numerele de elemente de pe linii.
Deci vom utiliza şi doi vectori, nr_a şi nr_f, astfel:
nr_a[i] = numărul de persoane despre care persoana i afirmă că spun adevărul
nr_f[i]= numărul de persoane despre care persoana i afirmă că mint.
Pentru a reprezenta o soluţie vom utiliza un vector cu n componente denumit stare, cu semnificaţia: stare[i]=0 dacă persoana i spune adevărul şi 1 dacă persoana i minte. În vectorul stare vom construi succesiv toate configuraţiile de 0 şi 1 până la depistarea unei configuraţii care respectă afirmaţiile memorate în A şi F. Pentru aceasta vom iniţializa vectorul stare cu 0 şi vom aduna 1 în baza 2 până când configuraţia este corectă.

40.

```
#include <fstream.h>
#define MAX_N 100
int main(void)
{int i, j, n, m;
 double timp[MAX_N], best_timp, dif;
  int clasament[MAX_N], temp;
  ifstream fisin("schi.in");
  fisin>>n>>m;
  fisin>>timp[0];   //citesc timpul primului concurent din runda 1
  best_timp = timp[0];   //deocamdata acesta este cel mai bun timp
  for (i = 1; i < n; ++i) //pentru fiecare concurent urmator
    {
     //citesc diferenta de timp fata de cel mai bun timp
     fisin>>dif;
     timp[i] = best_timp + dif;     //calculez timpul efectiv
     if (dif < 0)                   //daca a scos un timp mai bun
       best_timp = timp[i];        //acesta devine cel mai bun timp
    }
  for (i = 0; i < n; ++i)   //fac clasamentul dupa prima runda
    clasament[i] = i;
  for (i = 0; i < n - 1; ++i)
    for (j = i + 1 ;j < n; ++j)
      if (timp[clasament[i]] > timp[clasament[j]])
          {temp = clasament[i];
           clasament[i] = clasament[j];
           clasament[j] = temp;      }
//citesc timpul celui de al m-lea concurent
fisin>>timp[clasament[m-1]];
best_timp = timp[clasament[m - 1]];
//deocamdata acesta este cel mai bun timp total
for (i = m - 2; i >= 0; --i)
  { fisin>>dif;//citesc diferenta de timp fata de cel mai bun timp
    //calculez timpul total efectiv
    timp[clasament[i]]=best_timp+dif;
    if (dif < 0)                  //daca a scos un timp total mai bun
       //acesta devine cel mai bun timp total
       best_timp = timp[clasament[i]];
  }
for (i = 0; i < m - 1; ++i)   //fac clasamentul dupa runda a doua
   for (j = i + 1; j < m; ++j)
      if (timp[clasament[i]] > timp[clasament[j]])
         { temp = clasament[i];
           clasament[i] = clasament[j];
           clasament[j] = temp;          }
```



```
fisin.close();
  ofstream fisout("schi.out");     //afisez primii 3 clasati
  fisout << clasament[0] + 1 << endl;
  fisout << clasament[1] + 1 << endl;
  fisout << clasament[2] + 1 <<endl;
  fisout.close();  return 0; }
```

5. Stiva şi coada

1. a. *Fals* (coada este o structură de date abstractă, nu un vector; ea poate fi implementată static, cu ajutorul unui vector); **b.** *Adevărat*; **c.** *Fals* (coada este o structură de date care funcţionează după principiul FIFO, stiva funcţionează după principiul LIFO); **d.** *Fals* (pentru a afişa elementele unei stive trebuie să extragem succesiv elementele din stivă, deoarece nu avem acces direct decât la elementul din vârful stivei). **4.** Pentru a face distincţia între cazul în care coada este vidă (nu conţine nici un element) şi cazul în care coada este plină (toată zona de memorie alocată cozii este ocupată), vom reţine, pe lângă începutul şi sfârşitul cozii şi Nr, numărul de elemente din coadă.

a. Inserarea elementului x în coada C:

```
if (Nr == DimMaxCoada) {cout << "Eroare "; return;}
     else {Sf=(Sf+1)%DimMaxCoada; C[Sf]=x; Nr++;}
```

b. Extragerea unui element:

```
if (!Nr) {cout << "Eroare "; return;}
 else {x = C[Inc]; Inc = (Inc+1)% DimMaxCoada; Nr--;};
```

5. Deoarece primul pacient intrat va fi primul pacient tratat, funcţionarea cabinetului poate fi simulată cu ajutorul unei cozi, în care vom înregistra pacienţii în ordinea sosirii lor. Când medicul devine liber, un pacient este extras din coadă şi tratat. La fiecare minut vom testa dacă se produce un eveniment (vin pacienţi sau/şi medicul termină tratamentul unui pacient). În funcţie de evenimentul produs se inserează un element în coadă (pacientul nou sosit) sau se afişează timpul la care pleacă pacientul tratat şi se extrage un element din coadă (dacă există). **6.** Liniile de intrare, cele k linii de manevră şi linia de ieşire funcţionează, în fapt, ca structuri de tip coadă, pe care le vom implementa cu ajutorul vectorilor. Pe cele k linii de manevră, care pot conţine câte maxim n vagoane, sunt dispuse, în ordine, vagoanele care intră de pe linia de intrare. Pentru fiecare vagon care intră de pe linia de intrare se verifică dacă acesta poate intra pe una dintre liniile de manevră. Pentru aceasta se verifică pe rând liniile de manevră. Vagonul poate intra pe una dintre ele dacă fie linia de manevră este goală, fie numărul vagonului este cu 1 mai mare decât ultimul vagon deja ajuns pe linia de manevră respectivă. Dacă vagonul se poate muta pe una dintre liniile de manevră, se realizează mutarea urmată imediat de verificarea posibilităţii de mutare a unor vagoane de pe liniile de manevră pe linia de ieşire, adică operaţia de extragere a unor elemente din coadă. **7.** Problema este asemănătoare cu problema caroiajului, fiind cerută obţinerea unui drum de lungime minimă. De data aceasta nu mai există obstacole, trecerea din poziţia curentă într-o poziţie învecinată (sus, jos, stânga, dreapta) realizându-se în funcţie de valorile acestora. Prin urmare, vom utiliza o coadă în care vom memora poziţiile din labirint în ordinea în care acestea sunt parcurse. Iniţial în coadă plasăm poziţia de plecare – (1,1). La fiecare pas extragem un element din coadă, după care inserăm în coadă toate poziţiile vecine cu poziţia elementului extras care respectă condiţia de deplasare. Algoritmul se termină când coada devine vidă sau când am atins destinaţia – poziţia (n,m). Deoarece problema cere şi drumul minim parcurs, vom reţine pentru fiecare poziţie din labirint atinsă în timpul acestei parcurgeri şi direcţia din care a fost atinsă (caracterul D – jos, U – sus, L – stânga, respectiv R – dreapta).
8. Problema este asemănătoare cu problema caroiajului, numai că trebuie să o rezolvăm de două ori: o dată pentru a determina timpul minim în care ajunge Julieta în fiecare poziţie, respectiv cel în care ajunge Romeo. Vom determina o poziţie în care cei doi ajung în acelaşi timp minim.

6. Funcții

1. a. *Fals* (la apel intervin parametrii actuali); b. *Fals* (se specifică doar numele parametrilor actuali); c. *Adevărat*; d. *Adevărat*; e. *Fals* (lista de parametri formali poate să fie vidă).
2. c, e. **3.** b. **4.** a. **5.** b. **6.** Variabila locală `nr` nu este inițializată, funcția nu returnează nici o valoare, pentru `n=0` `nr` este `0`. **7.** `611215112122232351123`. **8.** c. **9.** `133233`. **10.** c.
11. Pentru calculul expresiei se scriu două funcții utilizator, `Min` și `Max`, pentru determinarea minimului, respectiv maximului dintre două numere reale. Calcularea valorii expresiei E se face apoi într-o instrucțiune alternativă imbricată. Pentru determinarea valorii absolute a unui număr real se utilizează funcția `abs`. **12.** Utilizând cunoștințele de la matematică cu privire la operațiile cu numere complexe, se scrie câte o funcție pentru citirea unui număr complex, afișarea unui număr complex, determinarea modulului unui număr complex, suma, diferența, produsul a două numere complexe.
13. Reprezentăm vârfurile triunghiului ca o structură cu două câmpuri – coordonatele `x` și `y` ale punctului respectiv. Pentru calcularea diferitelor elemente vom utiliza trei funcții: distanța dintre două puncte (`Latura`), aria triunghiului (`S`), raza cercului circumscris (`R`).

```
struct Punct {float  x, y;} Pa, Pb, Pc;
float Latura(Punct X, Y)
{  return sqrt((Y.x-X.x)* (Y.x-X.x)+(Y.y-X.y)* (Y.y-X.y));}
float S(float a, float b, float c)
{ float p=(a+b+c)/2;                                  //semiperimetrul
  return sqrt(p*(p-a)*(p-b)*(p-c));  }                //formula lui Heron
float R(Punct Pa, Punct Pb, Punct Pc)
{ float a, b, c;
  a = Latura(Pb, Pc); b = Latura(Pa, Pc); c = Latura(Pa, Pb);
  return a*b*c/(4*S(a, b, c));     }                  //raza cercului circumscris
```

14. a.

```
void elimin0(int a[20], int & n)
{ int nr=0, i;
  //nr este numarul de zerouri gasite pana la un moment dat in vector
  for (i=0; i<n; i++)
      if (!a[i]) nr++;                                //am mai gasit un zero
         else                                         //am gasit un element nenul
         if (nr)                                      //am gasit zerouri
         {//plasez elementul nenul pe pozitia primului zero din vector
            a[i-nr]=a[i];
            a[i]=0; }
n-=nr;  }        //n indica acum numarul elementelor nenule din vector
```

b. Singura modificare în funcția șablon este antetul:

```
template <class T>
void elimin0(T a[20], int & n)
```

15. O soluție ar fi de a reprezenta funcția ca o matrice cu elemente 0 și 1: `Mat[i][j]=1` dacă și numai dacă `f(i)=j`. Acest mod de reprezentare prezintă avantajul de a putea verifica foarte ușor următoarele:

- dacă funcția este bine definită – matricea definește o funcție dacă fiecare element din domeniul de definiție are exact o imagine, adică suma elementelor de pe fiecare linie trebuie să fie egală cu 1;
- dacă funcția este injectivă – funcția este injectivă dacă fiecare element din codomeniu are cel mult o imagine, adică suma elementelor de pe fiecare coloană trebuie să fie cel mult egală cu 1;
- dacă funcția este surjectivă – funcția este surjectivă dacă fiecare element din codomeniu are cel puțin o imagine, adică suma elementelor de pe fiecare coloană trebuie să fie cel puțin egală cu 1.

O altă soluție ar fi de a reprezenta funcția ca un vector în care reținem pentru fiecare element din domeniu imaginea sa. Utilizând această reprezentare, funcția este întotdeauna bine definită. Pentru a testa injectivitatea verificăm dacă oricare două componente ale vectorului sunt diferite. Pentru a testa surjectivitatea verificăm dacă mulțimea componentelor vectorului (imaginea funcției) coincide cu domeniul funcției.

16. a.

```
char* inserare(char* s2, char* s1, int n)
{ int i, lg1=LgSir(s1), lg2=LgSir(s2);
 //deplasez caracterele de la n la lg2 cu lg1 pozitii spre dreapta
 for (i=lg2; i>=n; i--) s2[i+lg1]=s2[i];
 //inserez s1 in locul disponibil
 for(i=0; s1[i]; i++) s2[i+n]=s1[i]; return s2; }
```

b. Deplasez în șirul de caractere dat caracterele de după poziția `n+nr` cu `nr` poziții la stânga.
17. Pentru a reprezenta prin vectorul caracteristic o mulțime de maxim `DimMax` elemente, vom utiliza un vector cu `DimMax/8` componente (putem presupune fără a restrânge generalitatea că `DimMax` este multiplu de 8, altfel îl înlocuim cu cel mai mic multiplu al lui 8, mai mare decât `DimMax`). Elementului cu numărul de ordine `i` ($i \in \{0,1,\ldots,DimMax-1\}$) îi corespunde bitul `i%8` din octetul `i/8`. Descriem principalele operații cu mulțimi, utilizând această reprezentare.

```
#define DimMax 1024
#define Lg (DimMax/8)
typedef unsigned char Multime[Lg];
int in (unsigned x, Multime M)
{   //intoarce 1 daca x apartine multimii M si 0 altfel
    return M[x/8] >> x%8 & 1; }
int inclus (Multime M1, Multime M2)
{   //returneaza 1 daca M1 inclus sau egal cu M2 si 0 altfel
    for (int i=0; i < DimMax; i++)
        if((M1[i/8] >> i%8 & 1)>(M2[i/8] >> i%8 & 1)) return 0;
    return 1;  }
void adauga(unsigned x, Multime M)
{    //insereaza elementul x in multimea M
     M[x/8]|=(1<< x%8);  }
void extrage(unsigned x, Multime M)
{    //elimina elementul x din multimea M
     M[x/8]&=~(1<< x%8);   }
void reuniune (Multime M1, Multime M2, Multime M)
{    //rezultatul reuniunii va fi in M
     for (int i=0; i<Lg;i++) M[i] = M1[i] | M2[i]; }
void intersectie (Multime M1, Multime M2, Multime M)
{   //rezultatul intersectiei va fi in M
    for (int i=0; i<Lg;i++) M[i] = M1[i] & M2[i];   }
void diferenta (Multime M1, Multime M2, Multime M)
{   //rezultatul diferentei M1-M2 va fi in M
    for (int i=0; i<Lg;i++) M[i] = M1[i] & ~M2[i];  }
void diferentaS (Multime M1, Multime M2, Multime M)
{   //rezultatul diferentei simetrice va fi in M
    Multime D1, D2;
    diferenta (M1, M2, D1); diferenta (M2, M1, D2);
    reuniune (D1, D2, M);                            }
void afisare(Multime M)
{    for (unsigned i=0; i<DimMax; i++)
        if (in(i, M)) cout << i<< ' ';
     cout << endl;                                   }
void citire(Multime M)
{    int cate;   unsigned x;
     for (unsigned i=0; i<Lg; i++) M[i]=0;
     cout <<"Cate elemente are multimea ? "; cin>>cate;
     cout <<"Introduceti elementele multimii ";
     for (i=0; i<cate; i++) {cin >> x; adauga(x, M);}
     cout << endl;                                   }
```

18. Vom crea un fişier antet care va conţine declaraţiile funcţiilor, tipurilor şi constantelor simbolice necesare pentru lucrul cu mulţimi (definite în fişierul `multimi.cpp`), denumit `multimi.h`:

```
#define DimMax 256
#define Lg (DimMax/8)
typedef unsigned char Multime[Lg];
int in (unsigned x, Multime & M);
int inclus (const Multime & M1, const Multime & M2);
void adauga(unsigned x, Multime & M);
void extrage(unsigned x, Multime & M);
void reuniune (Multime & M1, Multime & M2, Multime & M);
void intersectie (Multime & M1, Multime & M2, Multime & M);
void diferenta (Multime & M1, Multime & M2, Multime & M);
void diferentaS (Multime & M1, Multime & M2, Multime & M);
void afisare(Multime & M);
void citire(Multime & M);
```

Apoi vom crea un proiect care conţine fişierul `multimi.cpp` şi următorul fişier care generează numerele prime, prin metoda ciurului lui Eratostene:

```
#include <iostream.h>
#include "multimi.h"
Multime P;
void main()
{ unsigned i, k;
  //plasam in ciur toate numerele naturale pana la DimMax:
  for (i=2; i < DimMax;i++) adauga(i, P);
  for (i=2; i < DimMax/2;i++)
       if (in (i, P))                  //i este in ciur, deci este prim
          for (k=2; k*i<DimMax; k++) //elimin din ciur multiplii lui i
              extrage(k*i,P);
  cout << "Numerele prime sunt \n"; afisare(P);                }
```

19. Parcurg fişierul de tradus linie cu linie. Traduc linia citită şi o scriu în fişierul de ieşire. Pentru traducerea unei linii extrag succesiv câte un cuvânt şi îi caut traducerea în dicţionar. Dacă am găsit cuvântul în dicţionar, plasez în linia tradusă traducerea cuvântului, altfel scriu cuvântul original. Separatorii îi copiez în linia tradusă aşa cum se află pe linia de tradus.

```
#include <fstream.h>
#include <string.h>
ifstream f, d; ofstream ft;
char linie [100], liniet[100], nume[40];
void Init();
char * Traduc (char *);
void CautDict(char *, char *);
void main ()
{ Init();
  while (!f.eof())       //parcurg fisierul de tradus linie cu linie
        { f.getline(linie, 100);        //citesc o linie din fisier
          ft<<Traduc(linie)<<endl;  }    //traduc linia si o scriu in ft
  f.close(); ft.close();   }
void Init()
{ cout <<"Fisierul care trebuie tradus "; cin.getline(nume, 40);
  f.open(nume);
  cout <<"Fisierul care contine traducerea "; cin.getline(nume, 40);
  ft.open(nume);
  cout <<"Dictionarul "; cin.getline(nume, 40); }
char * Traduc(char linie[100])
{ char sep[]=" ,.;:";                    //contine separatorii
  char cuvant[20], ctradus[20];
  int i, j, k;
                        //extrag succesiv cuvintele de pe linie
  for (i=j=0; i < strlen(linie);)
      { while (i<strlen(linie) && strchr(sep, linie[i]))
              liniet[j++]=linie[i++];          //copiez separatorii
```



```
        //urmeaza un cuvant, il extrag
        for (k=0; i<strlen(linie) &&!strchr(sep, linie[i]);)
            cuvant[k++]=linie[i++];
        cuvant[k++]=NULL;
        CautDict(cuvant, ctradus);              //traduc cuvantul extras
                                                //copiez cuvantul tradus
        for (k=0; ctradus[k]; liniet[j++]=ctradus[k++]);      }
  liniet[j++]=NULL; return liniet; }
void CautDict(char c[20], char ct[20])
{ char l[42], *p;
  d.open(nume);
  while (!d.eof())
        { d.getline(l, 42);          //citesc o linie din dictionar
          p=l;                      // extrag cuvantul si traducerea sa
          if (!strcmp(strtok(p, "="), c))          //am gasit cuvantul
             {  strcpy(ct,strtok(NULL, "\n"));    //copiez traducerea
                d.close();
                return;  }
        }
  strcpy(ct, c);                                  //cuvantul nu se traduce
  d.close(); }
```

20. Vom reţine configuraţia populaţiei într-o matrice `Pop1`, cu `n` linii şi `m` coloane. La fiecare pas vom apela funcţia `Urmator()`, care va construi, pe baza configuraţiei curente, configuraţia următoare în matricea `Pop2`. La pasul următor, aceasta va deveni „configuraţie curentă" (`Pop2` devine `Pop1`). Funcţia se apelează repetat, până la apăsarea unei taste.

```
int Pop1[100][100], Pop2[100][100], n, m;
void Urmator()
{int i,j, nv;
for (i=1;i<=n; i++)
    { for (j=1; j<=m; j++)
        {                                       //calculez numarul de vecini
          nv=Pop1[i-1][j-1]+Pop1[i-1][j]+Pop1[i-1][j+1]+Pop1[i][j-1]+
             Pop1[i][j+1]+Pop1[i+1][j-1]+Pop1[i+1][j]+Pop1[i+1][j+1];
          switch (nv)
                {case 2: Pop2[i][j] = Pop1[i][j];
                         if (Pop1[i][j]) cout << '*'; else cout<<' ';
                         break;
                 case 3:  Pop2[i][j] = 1; cout <<'*'; break;
                 default: Pop2[i][j] = 0; cout << ' ';        }
        }
      cout << endl;  }
for (i=1; i<=n;i++)   //configuratia obtinuta devine configuratie curenta
    for (j=1;j<=m; j++) Pop1[i][j]=Pop2[i][j]; }
```

21. Funcţia `apare(x,y)` determină de câte ori apare `x` ca subsecvenţă de cifre în `y`. Funcţia va fi apelată în `main()` pentru fiecare număr din intervalul [a, b].

```
unsigned long apare(unsigned long x, unsigned long y)
{                   //determina de cate ori apare x ca subsecventa in y
  char Sx[11], Sy[11], *p, *pSy;
  unsigned long cate = 0;
  ultoa(x, Sx, 10); ultoa(y, Sy, 10); pSy=Sy;
  while ((p=strstr(pSy, Sx))!=NULL)   //x apare in y incepand cu pozitia p
       {  cate++;   pSy=p+1;  }
  return cate; }
```

22. Se va descrie o funcţie pentru fiecare metodă de sortare învăţată. La apelul unei funcţii de sortare se estimează timpul de execuţie şi se realizează o reprezentare grafică (cu funcţii din `graphics.h`).

23. Pentru fiecare număr x_i se determină cifra maximă şi bazele în care poate fi considerat că este reprezentat (`cifra maxima+1,...,10`). Pentru fiecare bază posibilă, se converteşte numărul x_i în

baza 10, obținându-se astfel pentru fiecare x_i o mulțime de valori v_i. Se calculează produsul cartezian $v_1 \times v_2 \times \ldots \times v_n$ și se alege elementul care determină un interval de lungime minimă.

24. Pentru a determina ordinea în care au fost cusute peticele, le vom „descoase" în ordinea inversă coaserii lor. Mai exact, la fiecare pas vom determina un petic care este deasupra (peste el nu este plasat nici un alt petic) și îl vom descoase (vom înlocui culoarea lui cu alb).

25.

```
#include <stdio.h>
#define MaxM 105
#define MaxN 1005
#define Infinit 1000
#define INFILE "pepsi.in"
#define OUTFILE "pepsi.out"
int tip[MaxN];  /*tip[i]=1 daca persoana i este risipitoare; 0 - altfel */
int volum[MaxM];/*volum[i]=volumul existent la un moment dat in sticla i*/
int final[MaxM];/*final[i]=volumul existent in final in sticla i */
int sol[MaxN];
int perm[MaxM], uz[MaxM];
int M, N, K;
void afisare( void )
{int i;
 FILE *fout = fopen( OUTFILE, "wt" );
 for (i = 1; i <= N; i++ ) fprintf( fout, "%d ", perm[sol[i]] );
 fclose(fout);}
void citire(void)
{int i;
 FILE *fin=fopen( INFILE, "rt" );
 char s[MaxN+1];
 fscanf( fin, "%d %d %d", &N, &M, &K );
 for ( i = 1; i <= M; i++ ) volum[i] = K;
 fscanf( fin, "%s", s );
 for( i = 1; i <= N; i++ )
      if( s[i] == 'R' ) tip[i] = 1; else tip[i] = 0;
 for( i = 1; i <= M; i++ ) fscanf( fin,"%d", &final[i] );
 fclose( fin );}
int main( void )
{int i, j, m, poz;
 citire();
 /*servesc persoanele in ordine */
 for (i = 1; i <= N; i++ )
     if (tip[i] == 1)
         {m = -1; /*determin volumul maxim */
         for (j = 1; j <= M; j++ )
              if (volum[j] > m )
                  {m = volum[j]; poz = j;  }
             sol[i] = poz; volum[poz]--;          }
        else
        {   m = Infinit; /*determin volumul minim */
            for ( j = 1; j <= M; j++)
                if ( volum[j] < m  && volum[j] > 0 )
                   {m = volum[j]; poz = j;    }
            sol[i] = poz;  volum[poz]--;      }
 /*in solutia obtinuta este posibil sa fie o alta ordine a sticlelor*/
 for (i = 1; i <= M; i++ )
     {j = 1; /*determin renumerotarea sticlelor */
      while (final[i] != volum[j] || uz[j]) j++;
      perm[j] = i; uz[j] = 1;     }
 afisare(); return 0; }
```

26. Citim succesiv operațiile și pentru fiecare determinăm tipul și apelăm funcția corespunzătoare (determinare număr submulțime `detnr()` sau determinare submulțime `detsub()`).


```
long detnr(void)
{long nro=0, gasit=0;
 int pas;
 for (pas=1; pas<=n && gasit<x; pas++)
      if (!s[pas]) nro+=1l<<n-pas;
         else
         {nro++;gasit++;}
return nro;}
void detsubm(void)
{int pas=0;
 while (nr)
      {pas++;
       if (nr<=1l<<n-pas) {s[pas]=1; nr--;}
         else {nr-=1l<<n-pas;}
      }
}
```

27. Orice număr natural are o scriere unică sub formă de sumă de termeni neconsecutivi ai șirului Fibonacci (demonstrația se poate face prin inducție). Vom determina cel mai mare număr Fibonacci care este mai mic sau egal cu `N`, îl afișăm, apoi îl scădem din `N` și reluăm algoritmul pentru noul `N`. Sunt necesare operații pe numere mari. **28.** Ordonăm butoaiele crescător după capacitate. Calculăm volumul total de apă acumulat în vase, apoi îl distribuim în ordine în butoaie. Atenție, trebuie să utilizați operații cu numere mari.

29.

```
#include <stdio.h>
#define INPUTFILE "masina.in"
#define OUTPUTFILE "masina.out"
#define MAXR 50
#define MAXS 50
#define Culoar '.'
#define Zid 'X'
#define OK '*'
#define MaxDir 1000
int n, r, s, InitL, InitC, LgC, LgAux;
char harta[MAXR][MAXS+1];
typedef struct { int r, s; } Deplasare;
Deplasare Dir[MaxDir];
Deplasare Coada[MAXR*MAXS];
Deplasare AuxC[MAXR*MAXS];
void citire(void)
{int i, j;
 char str[20];
 FILE *f = fopen(INPUTFILE, "rt");
 fscanf(f, "%d%d", &r, &s);
 for (i=0; i<r; i++)
     {fscanf(f, "%s", harta[i]);
      for (j=0; j<s; j++)
          if (harta[i][j]==OK) //retin pozitia initiala a masinii
              {InitL = i; InitC = j;}
     }
 fscanf(f, "%d", &n);
 for (i=0; i<n; i++)
      {fscanf(f, "%s", str);
       if (str[0]=='N') { Dir[i].r = -1; Dir[i].s = 0; }
       if (str[0]=='S') { Dir[i].r = 1; Dir[i].s = 0; }
       if (str[0]=='V') { Dir[i].r = 0; Dir[i].s = -1; }
       if (str[0]=='E') { Dir[i].r = 0; Dir[i].s = 1; }
      }
 fclose(f); }
```

```
void rezolva(void)
{ int p, i, tr, ts;
  LgC = 1;  //initializez coada cu pozitia initiala a masinii
  Coada[0].r = InitL; Coada[0].s = InitC;
  for (p=0; p<n; p++)
      {for (i=0; i<LgC; i++) harta[Coada[i].r][Coada[i].s] = Culoar;
      LgAux = 0;
      for (i=0; i<LgC; i++)
          {tr = Coada[i].r; ts = Coada[i].s;
           while (1)
                { tr += Dir[p].r; ts += Dir[p].s;
                  if (tr<0 || tr>=r || ts<0 || ts>=s) break;
                  if (harta[tr][ts]!=Culoar) break;
                  AuxC[LgAux].r = tr; AuxC[LgAux++].s = ts;
                  harta[tr][ts] = OK; };
          }
      for (i=0; i<LgAux; i++) Coada[i] = AuxC[i];
      LgC = LgAux;    }
}
void afisare(void)
{int i;
 FILE *f = fopen(OUTPUTFILE, "wt");
 for (i=0; i<r; i++) fprintf(f, "%s\n", harta[i]);
 fclose(f);    }
int main()
{     citire(); rezolva();   afisare(); return 0;     }
```

Bibliografie

Andonie, Razvan; Gârbacea, Ilie, *Algoritmi fundamentali. O perspectiva C++*, Editura Libris, Cluj-Napoca, 1995.

Atanasiu, A., *Cum se scrie un algoritm? Simplu*, Editura Agni, Bucureşti, 1993.

Atanasiu, Adrian; Pintea, Rodica, *Culegere de probleme Pascal*, Editura Petrion, Bucuresti, 1996.

Bostan, Gheorghe, *Culegere de probleme de informatică*, Editura Lumina, Chişinău, 1999.

Catrina, Octavian; Cojocaru, Iulian, *Turbo C++*, Editura Teora, Bucureşti, 1993.

Cerchez, Emanuela; Şerban, Marinel, *Informatică. Manual pentru clasa a IX-a*, Editura Didactică şi Pedagogică, Bucureşti, 2004.

Cerchez, Emanuela; Şerban, Marinel, *Informatică. Varianta Pascal. Manual pentru clasa a X-a*, Editura Polirom, Iaşi, 2000.

Cerchez, Emanuela, *Informatică. Culegere de probleme pentru liceu*, Editura Polirom, Iaşi, 2002.

Cormen, Thomas; Leiserson, Charles; Rivest, Ronald, *Introduction to Algorithms*, The Massachusetts Institute of Technology, 1990.

Horowitz, Ellis; Sahni, Sartaj; Anderson-Freed, Susan, *Fundamentals of Data Structures in C*, Computer Science Press, New York, 1993.

Ivaşc, Cornelia; Prună, Mona; Mateescu-Cerchez, Emanuela, *Bazele Informaticii. Caiet de laborator*, Editura Petrion, Bucureşti, 1997.

Jamsa, Kris, *Succes cu C++*, Editura All, Bucureşti, 1997.

Jamsa, Kris, *Totul despre C şi C++*, Editura Teora, Bucureşti, 2001.

Livovschi, Leon; Georgescu, Horia, *Sinteza si analiza algoritmilor*, Editura Stiintifica si Enciclopedica, Bucuresti, 1986.

Lucanu, Dorel, *Bazele proiectarii programelor si algoritmilor*, Editura Universitatii „Al.I. Cuza", Iasi, 1996.

Mitrana, Victor, *Provocarea algoritmilor*, Editura Agni, Bucuresti, 1994.

Negrescu, Liviu, *Limbajul C*, Editura Computer Libris Agora, Cluj-Napoca, 1997.

Niculescu, St.; Cerchez, Em.; Şerban M. ş.a.; *Bacalaureat şi atestat în informatică*, Editura L&S Infomat, Bucureşti, 2000.

Sedgewick, Robert, *Algorithms in C++*, Addison-Wesley Publishing Company, Boston, MA, 1992.

S.N.E.E., Subiecte Bacalaureat 2000-2003.

La Editura POLIROM
au apărut :

Mircea Băduț – *Calculatorul în trei timpi*

Luminița Fînaru, Ioan Brava – *Visual Basic. Primii pași... și următorii*

Ștefan Tanasă, Cristian Olaru, Ștefan Andrei – *JAVA de la 0 la expert*

Mihai Cioată – *ActiveX. Concepte și aplicații*

Mircea Băduț – *AutoCAD-ul în trei timpi. Ghidul proiectării profesionale*

Emanuela Cerchez, Marinel Șerban – *Informatica. Manual pentru clasa a X-a*

Emanuela Cerchez, Marinel Șerban – *PC. Pas cu pas*

Emanuela Cerchez, Marinel Șerban – *Informatica. Culegere de probleme pentru liceu*