4. Operatori şi expresii.

Un *operator* este un simbol care arată ce operaţii se execută asupra unor operanzi (termeni).

Un *operand* este o constantă, o variabilă, un nume de funcţie sau o subexpresie a cărei valoare este prelucrată direct de operator sau suportă în prealabil o conversie de tip.

Operatorii, după numărul de operanzi asupra cărora se aplică pot fi: *unari, binari* şi *ternari.*

În C există 45 de operatori diferiţi dispuşi pe 15 *niveluri de prioritate.*

În funcţie de tipul operanzilor asupra cărora se aplică, operatorii pot fi: aritmetici, relaţionali, binari, logici, etc.

Operatorii sunt împărţiţi în *clase de precedenţă* (sau de *prioritate*). În fiecare clasă de precedenţă este stabilită o *regulă de asociativitate*, care indică ordinea de aplicare a operatorilor din clasa respectivă: de la stânga la dreapta sau de la dreapta la stânga.

O expresie este o combinaţie de operanzi, separaţi între ei prin operatori; prin *evaluarea* unei expresii se obţine o *valoare rezultat*.Tipul valorii rezultat depinde de tipul operanzilor şi a operatorilor folosiţi.

Evaluarea unei expresii poate avea *efecte laterale,* manifestate prin modificarea valorii unor variabile.

4.1. Conversii de tip.

Valorile pot fi convertite de la un tip la altul. Conversia poate fi implicită sau realizată în mod explicit de către programator.

4.1.1. Conversii implicite de tip.

Conversiile implicite au loc atunci când este necesar ca operatorii şi argumentele funcţiilor să corespundă cu valorile aşteptate pentru acestea.
Acestea pot fi sintetizate prin tabelul:

Tabel 4.1. Conversii implicite de tip

Tip	Tip la care se converteşte implicit
`char`	**`int, short int, long int`**
`int`	**`char`** (cu trunchiere) **`short int`** (cu trunchiere) **`long int`** (cu extensia semnului)
`short int`	ca şi **`int`**
`long int`	ca şi **`int`**
`float`	**`double`** **`int, short int, long int`**
`double`	**`float`** **`int, short int, long int`**

4.1.2. Conversii aritmetice.

Când un operator binar se aplică între doi operanzi de tip diferit, are loc o *conversie implicită* a tipului unuia dintre ei, şi anume, operandul de tip "mai restrâns" este convertit la tipul "mai larg" al celuilalt operand. Astfel în expresia **`f + i`**, operandul **`int`** este convertit în **`float`**.

Operatorii aritmetici convertesc automat operanzii la un anumit tip, dacă operanzii sunt de tip diferit. Se aplică următoarele reguli:

- operanzii **char** şi **short int** se convertesc în **int**; operanzii **float** se convertesc în **double**.
- dacă unul din operanzi este **double** restul operanzilor se convertesc în **double** iar rezultatul este tot **double**.
- dacă unul din operanzi este **long** restul operanzilor se convertesc în **long** , iar rezultatul este tot **long**.
- dacă unul din operanzi este **unsigned** restul operanzilor se convertesc în **unsigned** , iar rezultatul este tot **unsigned**.
- dacă nu se aplică ultimele 3 reguli, atunci operanzii vor fi de tip **int** şi rezultatul de asemeni de tip **int.**

```
double    ←float
  ↑
long
  ↑
unsigned
  ↑
 int      ←char, short
```

Astfel `n = c - '0'` în care **c** reprezintă un caracter cifră calculează valoarea întreagă a acestui caracter.

Conversii implicite se produc şi în cazul operaţiei de atribuire, în sensul că valoarea din partea dreaptă este convertită la tipul variabilei acceptoare din stânga.

Astfel pentru declaraţiile:

```
int i;
float f;
double d;
char c;
```

sunt permise atribuirile:

```
i=f;  /* cu trunchierea partii fractionare */
f=i;
d=f;
f=d;
c=i;
i=c;
```

4.1.3. Conversiile de tip explicite (cast).

Conversiile explicite de tip (numite şi cast) pot fi forţate în orice expresie folosind un operator unar (*cast*) într-o construcţie de forma:

```
(tip) expresie
```

în care expresia este convertită la tipul numit.

Operatorul *cast* are aceeaşi precedenţă cu a unui operator unar.
Astfel funcţia **sqrt()** din biblioteca **<math.h>** cere un argument **double**, deci va fi apelată cu un cast: **sqrt((double) n).**

Apelurile cu argumente de alt tip vor fi convertite în mod automat la tipul **double: x=sqrt(2)** va converti constanta **2** în **2.0**.

4.2. Operatorii aritmetici.

Operatorii aritmetici binari sunt: **+, -, *, /** şi **%** (modul = restul impărţirii întregi).
Prioritatea operatorilor aritmetici este:

+, - unari
***, /, %** binari
+, - binari

Regula de asociativitate este de la stânga la dreapta (la priorităţi egale operatorii sunt evaluaţi de la stânga la dreapta).

Tabel 4.2. Operatori multiplicativi

operator	descriere	tip operanzi	tip rezultat	precedenţă
*	înmulţire	aritmetic	int,unsigned, long,double	3
/	împărţire	aritmetic	int,unsigned, long,double	3
%	rest împărţire întreagă	întreg	int,unsigned, long	3

Tabel 4.3. Operatori aditivi

operator	descriere	tip operanzi	tip rezultat	precedenţă
+	adunare	aritmetici, pointer şi întreg	int,unsigned,long double pointer	4
-	scădere	aritmetici, pointer şi întreg doi pointeri	int,unsigned,long double pointer int	4

4.3. Operatorii de atribuire.

Operaţia de atribuire modifică valoarea asociată unei variabile (partea stângă) la valoarea unei expresii (partea dreaptă). Valoarea transmisă din partea dreaptă este convertită implicit la tipul părţii stângi.

Atribuiri de forma: **a = a op b** se scriu mai compact **a op= b** în care **op=** poartă numele de *operator de atribuire*, op putând fi un operator aritmetic **(+, -,*,/,%)** sau binar (**>>, <<, &, ^, |**).

O *atribuire multiplă* are forma **v1=v2=...=vn=expresie** şi este asociativă la dreapta.

O operaţie de atribuire terminată prin punct-virgulă (terminatorul de instrucţiune) se transformă într-o *instrucţiune de atribuire*.

4.4. Operatorii relaţionali.

Operatorii relaţionali sunt: **>, >=, <, <=** , care au toţi aceeaşi prioritate (precedenţă).
Cu prioritate mai mică sunt: **==, !=.**

Operatorii relaţionali au prioritate mai mică decât operatorii aritmetici. Putem deci scrie
a < b -1 în loc de **a < (b -1)**

Exemple: **`car >= 'a' && car <= 'z'`**

Tabel 4.4. Operatori relaţionali

operator	descriere	tip operanzi	tip rezultat	precedenţă
<	mai mic	aritmetic sau pointer	**int**	6
>	mai mare	aritmetic sau pointer	**int**	6
<=	mai mic sau egal	aritmetic sau pointer	**int**	6
>=	mai mare sau egal	aritmetic sau pointer	**int**	6
==	egal	aritmetic sau pointer	**int**	7
!=	neegal	aritmetic sau pointer	**int**	7

4.5. Operatorii booleeni.

Există următorii operatori logici:

! – NEGATIE (operator unar)
&& – ŞI logic (operatori binari)
|| – SAU logic

Exemple:

```
i<n-1 && (c=getchar()) != '\n' && c != EOF
```

nu necesită paranteze suplimentare deoarece operatorii logici sunt mai puţin prioritari decât cei relaţionali.

```
bisect= an % 4 == 0 && an % 100 != 0 || an % 400 == 0;
estecifra= c >= '0' && c <= '9'
```

Condiţia **`x == 0`** este echivalentă cu **`!x`**
`x != 0 && y != 0 && z != 0` este echivalentă cu **`x && y && z`**

Tabel 4.5. Operatori booleeni (logici)

operator	descriere	Tip operanzi	Tip rezultat	precedenţă	asociativitate
!	negaţie	aritmetic sau pointer	**int**	2	DS
&&	ŞI logic	aritmetic sau pointer	**int**	11	SD
||	SAU logic	aritmetic sau pointer	**int**	12	SD

4.6. Operatorii binari (la nivel de biţi).

În C există 6 operaţii de manipulare a biţilor aplicate asupra unor operanzi întregi (**`char, short, int, long`**) cu sau fără semn:

& – ŞI
| – SAU inclusiv
^ – SAU exclusiv
<< – deplasare stânga
>> – deplasare dreapta
~ – complement fată de 1 (inversare)

Operatorul ŞI se foloseste pentru *operaţia de mascare* a unor biţi.

```
n=n & 0177;  /* pune pe 0 bitii din pozitia 8 in sus */
```

Operatorul SAU pune pe 1 biţii specificaţi printr-o mască:

```
x=x | MASCA; /* pune pe 1 bitii care sunt 1 in MASCA */
```

Deplasarea dreapta a unui întreg cu semn este aritmetică, iar a unui întreg fără semn este logică. De exemplu:

```
x |= 1 << 7; /* pune pe 1 bitul 0 din octetul x */
x &= ~(1 << 7); /* pune pe 0 bitul 0 din octetul x */
```

Tabel 4.6. Operatori binari (pe biţi)

operator	descriere	Tip operand	tip rezultat	precedenţă
<<	deplasare stânga	întreg	ca operandul stâng	5
>>	deplasare dreapta	întreg	ca operandul stâng	5
&	ŞI pe biţi	întreg	**int,long,unsigned**	8
^	SAU exclusiv pe biţi	întreg	**int,long,unsigned**	9
|	SAU inclusiv pe biţi	întreg	**int,long,unsigned**	10

4.7. Operatorul condiţional.

Decizia
```
if (a > b)
   max = a;
else
   max = b;
```

poate fi reprezentată prin expresia condiţională:

```
max = a > b ? a : b
```

În general **ex1 ? ex2 : ex3** determină evaluarea **ex1**; dacă aceasta nu este 0 atunci valoarea expresiei condiţionale devine **ex2,** altfel **ex3**.

4.8. Operatorul secvenţă.

Este reprezentat prin **,** şi se foloseste în situaţiile în care sintaxa impune prezenţa unei singure expresii, dar prelucrarea presupune prezenţa şi evaluarea mai multor expresii.

Exemplu: **a < b ? (t=a, a=b, b=t) : a**

4.9. Operatori unari

a) Operatorul `sizeof`.

Aplicat asupra unei variabile furnizează numărul de octeţi necesari stocării variabilei respective. Poate fi aplicat şi asupra unui tip sau asupra tipului unei expresii:

```
sizeof variabila
sizeof tip
sizeof expresie
```

`sizeof` este un operator cu efect la compilare.

Tabel 4.7. Operatorul sizeof

operator	descriere	tip operand	tip rezultat	precedenţă
sizeof	necesar de memorie	variabilă sau tip	**unsigned**	2

b) Operatorii de incrementare /decrementare.

Tabel 4.8. Operatori de incrementare / decrementare

operator	descriere	tip operand	tip rezultat	precedenţă
++	preincrementare	aritmetic sau pointer	**int,long,double, unsigned,pointer**	2
++	postincrementare	aritmetic sau pointer	la fel	2
--	predecrementare	aritmetic sau pointer	la fel	2
--	postdecrementare	aritmetic sau pointer	la fel	2

```
int a, b=5;
a = b++;  /* a=5 */
a = ++b;  /* a=7 */
```

c) Operatori de adresare indirectă / determinare adresă

& entitate - obţine adresa unei entităţi,

**** pointer*** - pentru adresare indirectă - adică memorează adresa unei entităţi printr-o valoare a unui pointer

Tabel 4.9. Operatori unari

operator	descriere	tip operand	tip rezultat	precedenţă
*	indirectare	pointer la **T**	**T**	2
&	adresare	**T**	pointer la **T**	2
~	negaţie	aritmetic	**int,long,double**	2
!	negaţie logică	aritmetic sau pointer	int	2

d) Operatori de acces :

- la elementele unui tablou
- la câmpurile unei structuri sau unei uniuni
- indirect prin intermediul unui pointer la câmpurile unei structuri sau unei uniuni

Operatorii de acces sunt:

- **[]** *indexare* folosit în expresii de forma **tablou[indice]**
- **.** *selecţie directă* - pentru adresarea unui câmp dintr-o structură sau uniune sub forma: **struct.selector**
- → *selecţie indirectă* - pentru accesul la un câmp dintr-o structură sau uniune, a cărei adresă este memorată într-un pointer
  pointer -> selector este echivalent cu **(* pointer) . selector**

Toţi aceşti operatori de acces, împreună cu operatorul de apel de funcţie **()** au cea mai ridicată prioritate, şi anume 1.

Tabel 4.10. Operatori de acces

operator	descriere	exemple	precedenţă
()	apel de funcţie	**sqrt(x),printf("salut\n")**	1
[]	indexare tablou	**x[i], a[i][j]**	1
.	selector structură	**student.nastere.an**	1
->	selector indirect structură	**pstud->nume**	1

Tabel 4.11. Ordinea evaluarii operanzilor.

precedenţă	operatori	simbol	asociativitate
1	apel funcţie / selecţie	**() [] . ->**	SD
2	unari	*** & - ! ~ ++ -- sizeof**	DS
3	multiplicativi	*** / %**	SD
4	aditivi	**+ -**	SD
5	deplasări	**<< >>**	SD
6	relaţionali	**< > <= >=**	SD
7	egalitate / neegalitate	**== !=**	SD
8	ŞI pe biţi	**&**	SD
9	SAU exclusiv pe biţi	**^**	SD
10	SAU inclusiv pe biţi	**|**	SD
11	ŞI logic	**&&**	SD
12	SAU logic	**||**	SD
13	condiţional	**?:**	DS
14	atribuire	**= op=**	DS
15	virgula	**,**	SD

5. Instrucţiuni.

5.1. Instrucţiunea expresie.

O instrucţiune expresie se obţine punând terminatorul de instrucţiune (punct-virgula) după o expresie:

```
expresie;
```

Exemple:

```
a++;
scanf(…);
max=a>b ? a : b;
```

Exemplul 1: *Un număr real, introdus de la tastatură reprezintă măsura unui unghi exprimată în radiani. Să se scrie un program pentru conversia unghiului în grade, minute şi secunde sexagesimale.*

```
#include <stdio.h>
#define PI 3.14159265
int main(){
  float rad, gfr, mfr;
  int g, m, s;
  printf("Introduceti numarul de radiani: ");
  scanf("%f", &rad);
  g=gfr=rad*180/PI;
  m=mfr=(gfr-g)*60;
  s=(mfr-m)*60;
  printf("%5.2f radiani=%4d grade %02d min %02d sec\n",
       rad, g, m, s);
  return 0;
}
```

5.2. Instrucţiunea compusă (blocul).

Forma generală:

```
{
  declaratii_si_definitii;
  instructiuni;
}
```

Se foloseşte în situaţiile în care sintaxa impune o singură instrucţiune, dar codificarea impune prezenţa unei secvenţe de instrucţiuni. Blocul de instrucţiuni conteaza ca o singură instrucţiune.

5.3. Instrucţiunea vidă.

Forma generală: `;`

Sintaxa impune prezenţa unei instrucţiuni, dar logica problemei nu necesită nici o prelucrare. In acest mod se introduc unele relaxări în sintaxă.

5.4. Instrucţiunea `if`.

Forma generală:

```
if (expresie)
```

```
  instructiune1;
else
  instructiune2;
```

Se evaluează expresia; dacă este diferită de 0 se execută **instrucţiune1** altfel **instrucţiune2**

O formă simplificată are instrucţiune2 vidă:

```
if (expresie)
  instructiune;
```

În problemele de clasificare se întâlnesc decizii de forma:

```
if (expr1)
  instr1;
else if (expr2)
  instr2;
...
else
  instrn;
```

De exemplu dorim să contorizăm caracterele citite pe categorii: litere mari, litere mici, cifre, linii şi altele:

```
if  (c == '\n')
  linii++;
else if (c>='a' &&  c<='z')
  lmici++;
else if (c>='A' && c<='Z')
  lmari++;
else if (c>='0' && c<='9')
  cifre++;
else
  altele++;
```

Exemplul 2 *Să se scrie un program pentru rezolvarea cu discuţie a ecuaţiei de grad 2:* $ax^2+bx+c=0$ *folosind operatorul condiţional*.

```
#include <stdio.h>
#include <math.h>
int main(){
 float a, b, c, d;
 printf("Introduceti coeficientii ecuatiei: a,b,c\n");
 scanf("%f %f %f", &a,&b,&c);
 a? d=b*b-4*a*c, d>=0? printf("x1=%f\tx2=%f\n",(-b- sqrt(d))/2/a,
                                (-b+sqrt(d))/2/a):
                       printf("x1=%f+i*%f\tx2=%f-i*%f\n",-b/2/a,
                                sqrt(-d)/2/a,-b/2/a,sqrt(-d)/2/a)):
    b? printf("x=%f\n",-b/2/a): c? printf("0 solutii\n"):
                                   printf("identitate\n");
 return 0;
}
```

Exemplul 3: *Data curentă se exprimă prin* ***an, luna*** *şi* ***zi****. Să se scrie un program care determină data zilei de mâine.*

```
#include <stdio.h>
int bisect(int a){
return a%4==0 && a%100!=0 || a%400==0;
}

int ultima(int a, int l){
 if (l==2)
   return (28+bisect(a));
 else if (l==4||l==6||l==9||l==11)
   return 30;
 else
   return 31;
}

int main()
{int a, l, z;
 printf("Introduceti data curenta: an,luna,zi\n");
 scanf("%d%d%d",&a,&l,&z);
 printf("azi: zi:%02d luna:%02d an:%4d\n", z,l,a);
 if (z < ultima(a,l))
   z++;
 else
   {z=1;
    if (l < 12)
      l++;
    else
      {l=1;
       a++;
      }
   }
 printf("maine: zi:%02d luna:%02d, an:%4d\n", z,l,a);
 return 0;
}
```

5.5. Instrucţiunea `switch`.

Criteriul de selecţie într-o problemă de clasificare îl poate constitui un selector care ia valori întregi. Forma generală:

```
switch (expresie){
  case val1:  secventa1;
  case val2:  secventa2;
        . . .
  default:    secventa s;
}
```

Se evaluează expresia selectoare; dacă valoarea ei este egală cu una din constantele cazurilor, se alege secvenţa de prelucrare corespunzătoare, după care se continuă cu secvenţele de prelucrare ale cazurilor următoare.

Dacă valoarea expresiei selectoare nu este egală cu nici una din constantele cazurilor, se alege secvenţa corespunzătoare etichetei default.

Pentru ca prelucrările corespunzătoare cazurilor să fie disjuncte se termină fiecare secventă de prelucrare prin `break`. De exemplu:

```
y=x;
switch (n)
{ case 5: y*=x;
  case 4: y*=x;
  case 3: y*=x;
  case 2: y*=x;
}
```

calculează x^n, unde `n` ia valori de la 1 la 5.

Exemplul 4 *Scrieti o functie pentru determinarea ultimei zile din lună.*

```
int ultima(int a, int l)
{ switch (l) {
     case 1: case  3: case  5: case 7:
     case 8: case 10: case 12: return 31;
     case 4: case  6: case  9: case 11: return 30;
     case 2: return (28 + bisect(a));
  }
}
```

5.6. Instrucţiunea `while`.

Este ciclul cu test iniţial; se repetă instrucţiunea componentă cât timp expresia are valoarea adevărat (diferit de 0).

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

Exemplu 5:*Copiaţi fişierul standard de intrare* ***stdin*** *la ieşirea standard* ***stdout***

```
/*copierea intrarii la iesire*/
{ int c;
  c = getchar();
  while (c != EOF)
  { putchar(c);
    c = getchar();
  }

/* varianta simplificata */
{ int c;
  while ((c=getchar()) != EOF)
    putchar(c);
}
```

Exemplul 6: *Să se calculeze cel mai mare divizor comun şi cel mai mic multiplu comun a 2 numere folosind algoritmul lui Euclid cu scăderi. (cât timp numerele diferă se înlocuieşte cel mai mare dintre ele prin diferenţa lor).*

```
#include <stdio.h>
int main(){
unsigned long a, b, ca, cb;
  printf("Introduceti cele doua numere\n");
  scanf("%lu %lu", &a, &b);
  ca=a; cb=b;
  while (a!=b)
    if(a > b)
      a-=b;
    else
      b-=a;
  printf("cmmdc(%lu,%lu)=%lu\ncmmmc(%lu,%lu)=%lu\n",
     ca,cb,a,ca,cb,ca*cb/a);
  return 0;
}
```

5.7. Instrucţiunea do...while.

Reprezintă ciclul cu test final;repetarea instrucţiunii are loc cât timp expresia este diferită de 0.

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

Corpul buclei este format dintr-o singură instrucţiune. Repetarea se face cel puţin o dată.

```
/* citirea unui raspuns */
{ char  opt;
  printf("Continuam ? D / N");
  do
  scanf("%c", &opt);
  while (opt == 'D' || opt == 'd');
```

Exemplul 7: *Să se stabilească dacă un număr este sau nu palindrom (are aceeaşi reprezentare citit de la stânga sau de la dreapta).*

```
#include <stdio.h>
int main(){
unsigned long n, c, r=0;
  scanf("%lu", &n);
  c=n;
  do{ r=10*r+n%10;
      n/=10;
  }while (n);
  printf("%lu %s este palindrom\n",c, (c==r)? "":"nu");
  return 0;
}
```

5.8. Instrucţiunea `for`.

Reprezintă o altă formă a ciclului cu test iniţial.

```
for (exp_init; exp_test; exp_modif)
  instructiune;
```

este echivalentă cu:

```
exp_init;
while  (exp_test){
  instructiune;
  exp_modif;
}
```

Exemplul 8: *Să se stabilească dacă un număr întreg* n *este sau nu prim.*

Vom încerca toţi divizorii posibili (de la 2 la √n). Dacă nu găsim nici un divizor, numărul este prim. Continuarea ciclului pentru testarea posibililor divizori este determinată de două condiţii:

- să mai existe divizori netestaţi
- candidaţii deja testaţi să nu fi fost divizori.

La ieşirea din ciclu se determină motivul pentru care s- a părăsit ciclul:

- s-au testat toţi candidaţii şi nu s-a găsit nici un divizor, deci numărul este prim
- un candidat a fost găsit divizor, deci numărul este neprim.

Ciclul de testare a candidaţilor are forma:

```
for (d=2; d*d <= n && n % d != 0; d++)
                      ;
```

Programul poate fi îmbunătăţit prin evitarea testării candidaţilor pari (cu excepţia lui 2).

```
#include <stdio.h>
int main() {
  unsigned long n, d;
  scanf("%lu",&n);
  for(d=2; d*d <= n && n%d; (d=2)? d=3: d+=2)
                          ;
  printf("numarul %lu este %sprim\n",n,!(n%d)? "ne":"");
  return 0;
}
```

5.9. Instrucţiunea `continue`.

Plasarea acestei instrucţiuni în corpul unui ciclu are ca efect terminarea iteraţiei curente şi trecerea la iteraţia următoare: **continue;**

Exemplul 9: *O secvenţă de numere întregi este terminată prin 0. Să se calculeze suma termenilor pozitivi din secvenţă.*

```
#include <stdio.h>
int main() {
  int n, suma;
  for(suma=0,scanf("%d",&n); n; scanf("%d",&n){
    if(n < 0) continue;
    suma += n;
  }
```

```
  printf("suma pozitivi = %d\n", suma);
  return 0;
}
```

5.10. Instrucţiunea `break`.

Are ca efect ieşirea dintr-o instrucţiune de ciclare sau dintr-o instrucţiune **`switch`**, pentru a face alternativele disjuncte (în caz contrar dintr-o alternativă se trece în următoarea). Permite implementarea unor cicluri cu mai multe ieşiri plasate oriunde în interiorul ciclului.

O structură repetitivă foarte generală cu mai multe ieşiri plasate oriunde este:

```
while (1) {
 . . .
  if(expresie1) break;
 . . .
  if(expresien) break;
 . . .
}
```

5.11. Instrucţiunea `goto`.

Realizează saltul la o etichetă. Este o instrucţiune nestructurată şi se evită.

```
goto eticheta;
```

5.12. Instrucţiunea `return`.

Orice funcţie nedeclarată **`void`** va trebui să întoarcă un rezultat. Tipul acestui rezultat este specificat în antetul funcţiei. Transmiterea acestui rezultat este realizată de o instrucţiune **`return`** inclusă în corpul funcţiei.

```
return expresie;
```

Exemplul 10: *Calculul factorialului.*

```
long factorial( int p)
{ int i;
  long f;
  for ( f = 1,i=2; i <= n; i++)
         f *= i;
  return f;
}
```

Dacă expresia întoarsă este de alt tip decât cel al funcţiei, atunci se face conversia la tipul funcţiei.

5.13. Probleme rezolvate.

1. Să se obţină reprezentarea ca fracţie zecimală a numărului **`m/n`**. Eventuala perioadă se afişează între paranteze.

Reprezentarea zecimală a fracţiei ordinare se obţine prin simularea împărţirii cifră cu cifră. In prealabil se simplifică fracţia şi se separă partea întreagă.
Lungimea părţii neperiodice a fracţiei zecimale reprezintă maximul dintre multiplicităţile cifrelor 2 şi 5 din descompunerea numitorului.

Dacă fracţia zecimală are şi parte periodică, atunci descompunerea numitorului conţine şi alţi factori primi în afară de 2 şi 5. O condiţie mai simplă care stabileşte dacă există parte periodică este ca restul parţial rămas după obţinerea cifrelor din partea neperiodică sa fie diferit de 0.
Operaţia de împărţire se încheie în momentul în care apare un rest parţial egal cu primul rest parţial din partea periodică. La citirea datelor (`m` si `n`) se asigură verificarea n != 0.

```
citire m si n si validare n != 0
simplificarea fractiei cu cmmdc
separarea partii intregi si afisarea ei
determinarea lungimii partii neperiodice
simularea impartirii pe lungimea partii neperiodice
if (exista parte periodica)
    salveaza primul rest partial din partea periodica
    afisare paranteza deschisa pentru partea periodica
    do
      calcul cifra din partea periodica si afisare
      obtinerea urmatorului rest partial
    while (restul partial != primul rest partial)
    afisare paranteza inchisa pentru partea periodica
```

In efectuarea împărţirii, o cifră a câtului se obţine prin împărţirea întreagă a restului parţial cu numitorul.
Următorul rest parţial se obţine ca restul împărţirii cu numitorul a restului parţial curent, completat în ultima poziţie cu un zero (înmulţit cu 10). Primul rest parţial din partea neperiodică este numărătorul înmulţit cu 10.
Simularea împărţirii pe lungimea părţii neperiodice se exprimă prin:

```
rest_partial = 10 * m
for (i = 1 ; i<=lungime_parte_neperiodica; i++)
       printf("%d", rest_partial / n);
       rest_partial = rest_partial % n * 10
```

Programul complet este:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int   m, n,          /* numarator si numitor fractie */
          rp,            /* restul partial */
          lfn,           /* lungimea fractiei neperiodice */
          m2, m5,        /* multiplicitati 2 si 5 in numitor */
          c, d, r, i;

    /*citire date */
    scanf("%d%d", &m, &n);
    while (n==0)   /* fortare n!=0 */
       scanf("%d", &n);
    /* simplificarea fractiei cu cmmdc calculat cu algoritmul lui
       Euclid */
    c = m;
    d = n;
```

```
    do {                /* algoritmul lui Euclid */
         r = c % d;
         c = d;
         d = r;
    } while(r);
    m /= c;             /* simplificare fractie */
    n /= c;
    /* separare parte intreaga */
    printf("%6d / %6d = %6d.", m,  n,  m / n);
    m %=  n;
    /* lungimea fractiei neperiodice */
    m2 = 0; c = n;      /* multiplicitate 2 */
    while (c % 2 == 0){
        m2++;
        c /= 2;
    }
    m5 = 0; c = n;      /* multiplicitate 5 */
    while (c % 5 == 0){
        m5++;
        c /= 5;
    }
    /* lfn = max ( m2, m5 ) */
    lfn = m2;
    if(m5 > lfn)
        lfn = m5;
    /* efectuarea impartirii pentru partea neperiodica */
    rp = 10 * m;                /* primul rest partial */
    for(i=0; i<lfn; i++){
        printf("%1d", rp / n);  /* cifra din partea neperiodica */
        rp %=  n * 10;          /* urmatorul rest partial */
    }
    /* efectuarea impartirii pentru partea periodica */
    if(rp){             /* exista parte periodica */
        printf("(");
        c = rp;         /* salvare primul rest partial */
        do{
            printf("%1d", c / n);  /* cifra din partea periodica */
            c %= n * 10;           /* urmatorul rest partial */
        } while(c!=rp);
        printf(")");
    }
    printf("\n");
    return 0;
}
```

5.14. Probleme propuse.

1. De pe mediul de intrare se citeşte un număr real **rad** reprezentând un unghi exprimat în radiani. Să se convertească în **grade, minute** şi **secunde** centesimale.

2. Un maratonist porneşte în cursă la un moment de timp exprimat prin **ora, minutul** şi

secunda startului. Se cunoaşte de asemeni timpul necesar sportivului pentru parcurgerea traseului. Să se determine momentul terminării cursei de către sportiv.

3. Să se stabilească codomeniul D al valorilor funcţiei:
f : [x1, x2] → D, f(x) = a.x^2+b.x+c, a, b, c ∈ R, a ≠ 0.
Se cunosc **a, b, c, x1, x2**.

4. Cunoscând data curentă exprimată prin trei numere întregi reprezentând anul, luna, ziua precum şi data naşterii unei persoane exprimată în acelaşi mod, să se calculeze vârsta persoanei exprimată în ani, luni şi zile. Se consideră în mod simplificator că toate lunile au 30 de zile.

5. Un punct în plan este dat prin coordonatele lui **(x, y)**. Să se stabilească poziţia lui prin indicarea cadranului (1, 2, 3 sau 4) în care este plasat. Pentru un punct situat pe una din semiaxe se vor preciza cadranele separate de semiaxa respectivă (de exemplu 2-3).

6. Se citesc trei numere reale pozitive ordonate crescător. Să se verifice dacă acestea pot să reprezinte laturile unui triunghi şi în caz afirmativ să se stabilească natura triunghiului: isoscel, echilateral, dreptunghic sau oarecare şi să se calculeze aria sa.

7. Cunoscând data curentă şi data naşterii unei persoane exprimată fiecare sub forma unui triplet **(an,luna,zi)** să se afle vârsta persoanei în ani împliniţi.

8. De pe mediul de intrare se citeşte un unghi exprimat în **grade,minute,secunde**. Să se convertească în radiani.

9. Un număr întreg S reprezintă o durată de timp exprimată în secunde . Să se convertească în **zile, ore, minute** şi **secunde** utilizând în program cât mai puţine variabile.

10. Trei valori reale sunt citite în variabilele **a, b, c.** Să se facă schimbările necesare astfel încât valorile din **a, b, c** să apară în ordine crescătoare.

11. Să se scrie algoritmul pentru rezolvarea cu discuţie a ecuaţiei de gradul 1: **a*x+b=0**, cu valorile lui **a** şi **b** citite de pe mediul de intrare.

12. Să se scrie algoritmul pentru rezolvarea cu discuţie a ecuaţiei de gradul 2: **a*x^2 + b*x + c = 0**. Se dau pe mediul de intrare coeficienţii **a, b, c** care pot avea orice valori reale reprezentabile în memoria calculatorului.

13. Se consideră sistemul de ecuaţii:

a*x + b*y = c
m*x + n*y = p

dat prin valorile coeficienţilor **a, b, c, m, n, p**. Să se rezolve sistemul cu discuţie.

14. Să se scrie algoritmul pentru "casierul automat" care citeşte de pe mediul de intrare suma (intreagă) datorată de un client şi calculează "restul" pe care acesta îl primeşte în număr minim de bancnote şi monezi de **100000, 50000, 10000, 5000, 1000, 500, 100, 50, 25,**

10, 5, 3 şi **1** leu considerând că suma plătită este cel mai mic multiplu de **100000** mai mare decât suma datorată.

15. Să se calculeze data revenirii pe pământ a unei rachete, exprimată prin **an, lună, zi, oră, minut, secundă**, cunoscând momentul lansării exprimat în acelaşi mod şi durata de zbor exprimată în secunde.

16. Un număr perfect este un număr egal cu suma divizorilor săi, printre care este considerată valoarea 1 dar nu şi numărul.
Să se găsească toate numerele perfecte mai mici sau egale cu un număr **k** dat pe mediul de intrare, şi să se afişeze fiecare număr astfel determinat, urmat de suma divizorilor lui. De exemplu numărul **6** are divizorii **1, 2, 3, 6**. El este număr perfect deoarece: **6 = 1 + 2 + 3.**

17. Dându-se trei numere întregi reprezentând data unei zile **(an,lună,zi)**, să se stabilească a câta zi din an este aceasta.

18. Se dau pe mediul de intrare un număr necunoscut de numere nenule terminate cu o valoare nulă. Să se stabilească dacă acestea:

- formează un şir strict crescător;
- formează un şir crescător;
- formează un şir strict descrescător;
- formează un şir descrescător;
- sunt identice;
- nu sunt ordonate.

19. Dându-se un număr întreg **n**, să se afişeze toţi factorii primi ai acestuia precum şi ordinele lor de multiplicitate.

20. Abaterea medie pătratică a rezultatelor obţinute prin determinări experimentale se poate calcula cu formula:

$$\mathbf{sigma} = \sqrt{\frac{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}{N(N-1)}}$$

aplicabilă numai dacă s-au făcut cel puţin 2 măsurători.

Dându-se pe mediul de intrare **N (N≤25)** şi rezultatele celor **N** determinări să se calculeze abaterea medie pătratică.

21. Să se calculeze $\mathbf{S} = \sum_{k=1}^{n} k!$ când se cunoaşte **n**

22. Să se scrie algoritmul pentru rezolvarea a **n** ecuaţii de gradul 2. Se citesc de pe mediul de intrare valoarea lui **n** şi **n** tripleţi **(a,b,c)** reprezentând coeficienţii ecuaţiilor.
Se recomandă realizarea unui program care să utilizeze cât mai puţine variabile.

23. Dându-se notele obţinute de o grupă de n studenţi la o disciplină, să se stabilească câţi dintre ei au promovat (nu se vor utiliza decât variabile simple).

24. Se dau pe mediul de intrare notele obţinute de către studenţii unei grupe la un examen, precedate de numărul studenţilor. Să se determine dacă grupa este sau nu integralistă, precum şi procentajul de note foarte bune (8..10).

25. Să se determine cel mai mare **(max)** precum şi cel mai mic **(min)** element dintr-un şir a_0 , a_1 , ... a_{n-1} .
Se dau pe mediul de intrare **n** precum şi cele **n** elemente ale şirului, care sunt citite pe rând într-o aceeaşi variabilă **a**.

27. De pe mediul de intrare se citeşte un număr real **b** şi un şir de valori reale pozitive terminate printr-o valoare negativă (care nu face parte din şir).
Să se stabilească elementul din şir cel mai apropiat de **b**. Se va preciza şi poziţia acestuia. Elementele şirului vor fi păstrate pe rând în aceeaşi variabilă **a**.

28. Să se calculeze x^n pentru **x** (real) şi **n** (intreg) daţi, folosind un număr cât mai mic de înmulţiri de numere reale.

29. De pe mediul de intrare se citesc **n** valori întregi pozitive. Pentru fiecare element să se indice cel mai mare pătrat perfect mai mic sau egal cu el.

30. De pe mediul de intrare se citeşte o listă de numere întregi pozitive terminate cu un număr negativ ce marchează sfârşitul listei. Să se scrie în dreptul fiecărei valori numărul prim cel mai apropiat mai mic sau egal cu numărul dat.

31. Să se rezolve ecuaţia **f(x) = 0** cu precizia **epsilon** dată, ştiind că are o rădăcină în intervalul **{a,b}** precizat. Se va utiliza metoda înjumătăţirii intervalului ("bisecţie").
Indicaţie: Se împarte intervalul **{a,b}** în două jumătăţi egale; fie m mijlocul intervalului. Dacă la capetele intervalului **{a,m}** funcţia are semne contrare soluţia se va căuta în acest interval, altfel se va considera intervalul **{m,b}.** Se consideră determinată soluţia dacă mărimea intervalului a devenit inferioară lui **epsilon** sau valoarea **|f(m)| < epsilon**..

32. Dându-se un număr întreg **n** să se afle cifrele reprezentării sale în baza **10** începând cu cifra cea mai semnificativă.

33. Să se afle cifrele reprezentării în baza **b** (începând cu cea mai semnificativă) a unui număr a dat în baza **10**.
Indicaţie:

$$a = c_n\, b^n + c_{n-1} b^{n-1} + \ldots + c_0$$

$$c_n = a \,/\, b^n$$

$$a \,\%\, b^n = c_{n-1}\, b^{n-1} + \ldots + c_0$$

34. Dându-se numărul real **a (0 < a < 1)** să se determine primele **n** cifre ale reprezentării lui într-o bază **b** dată.

35. De pe mediul de intrare se citesc **n** cifre constituind reprezentarea unui număr într-o bază **b1 < 10**, începând cu cea mai puţin semnificativă. Să se obţină şi să se afişeze cifrele reprezentării aceluiaşi număr într-o altă bază **b2 < 10.**

36. Să se verifice dacă un numar întreg citit de pe mediul de intrare este palindrom, adică se citeşte la fel de la stânga la dreapta şi de la dreapta la stânga (numărul este identic cu răsturnatul său). Un astfel de număr este **4517154**. Nu se vor folosi tablouri de variabile pentru păstrarea cifrelor numărului.

37. Să se determine toate numerele prime mai mici sau egale cu un număr **k** dat pe mediul de intrare.Pentru a verifica dacă un număr **x** este prim se va incerca divizibilitatea lui cu **2,3,4,...{√x }**. Numărul este prim dacă nu se divide cu niciunul dintre aceste numere şi este neprim dacă are cel puţin un divizor printre ele.

38. Printre numerele mai mici sau egale cu un număr **n** dat pe mediul de intrare să se găsească cel care are cei mai mulţi divizori.

39. Se consideră funcţia **f(x) = ln (2.x^2 + 1)**. Să se scrie un program pentru tabelarea pe intervalul **[-10, 10]** cu următorii paşi:

0.1 pentru **| x | ≤ 1.0**
0.5 pentru **1.0 < | x | ≤ 5.0**
1.0 pentru **5.0 < | x | ≤ 10.0**

40. Se dă un număr întreg pozitiv reprezentat în baza 10. Să se determine şi să se afişeze cifrele reprezentării sale în baza 16. Se precizează că în baza 16 cifrele utilizate sunt **0,...9,A,B,C,D,E,F**. Prin convenţie la sfârşitul numerelor reprezentate în baza 16 (hexazecimal) se scrie litera **H**. Dacă cifra cea mai semnificativă a reprezentării sale este A..F atunci se va afişa ca primă cifră un 0. Exemplu:

175 = 0AFH
(10) (16)

0AFH = 10 * 16^1 + 15 * 16^0

41. De pe mediul de intrare se citesc cifrele reprezentării unui număr întreg în baza 16 terminate cu caracterul H (cifrele hexazecimale sunt **0,...,9, A, B, C, D, E, F**). Să se calculeze şi să se afişeze reprezentarea numărului în baza 10.

42. Dându-se un număr întreg **n** să se afişeze reprezentarea sa cu cifre romane impunând regula ca o cifră să nu poată fi urmată de o alta cu valoare strict mai mare decât ea. Numărul 99 se va reprezenta în aceste condiţii ca **LXXXXVIIII** şi nu ca **XCIX.**

43. Se dau două numere întregi, primul reprezentând un an şi al doilea, numărul de zile scurse din

acel an. Să se determine data (luna şi ziua).

44. Să se calculeze coeficienţii binomiali C_n^1 , C_n^2 , ... C_n^p în care **n** şi **p** sunt intregi pozitivi daţi **(p <= n)**, ştiind că există următoarea relaţie de recurenţă:

$C_n^k = (n-k+1)/k * C_n^{k-1}$ pornind cu $C_n^0 = 1$

45. Se consideră polinomul: $p_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_n$
Să se calculeze valoarea polinomului într-un punct x dat, dacă valorile coeficienţilor lui **x** se citesc pe rând, în aceeaşi variabilă **a** :
a) în ordinea descrescătoare a puterilor lui **x** (adică în ordinea $a_0, a_1, \ldots a_n$)
b) în ordinea crescătoare a puterilor lui **x**, (adică în ordinea $a_n, a_{n-1}, \ldots a_0$)

46. Pentru a, b şi n daţi (**a,b ∈ R, n ∈ Z**) să se calculeze **x** şi **y** astfel ca: $x+i*y=(a+i*b)^n$ fără a folosi formula lui Moivre

47. Să se calculeze şi să se afişeze valorile integralei:

$$I_k(x) = \int_0^x u^k e^u du$$

pentru **k=1,2,...,n**, în care n şi x sunt daţi, cunoscând că:
$I_k(x) = [x^k - A_k^1 x^{k-1} + A_k^2 x^{k-2} - \ldots (-1)^k A_k^k] e^x$ unde :
$A_k^p = k(k-1) \ldots (k-p+1)$

48. Să se calculeze pentru **n** dat, f_n termenul de rangul **n** din şirul lui Fibonacci, cunoscând relaţia de recurenţă:

$f_p = f_{p-1} + f_{p-2}$ pentru $p > 2$ şi $f_0 = 1$, $f_1 = 1$

49. Şirul $\{x_n\}$ generat cu relaţia de recurenţă $x_n = (x_{n-1} + a/x_{n-1})/2$ pornind cu $x_0 = a/2$ este convergent pentru $a > 0$ şi are ca limită $\sqrt{a}$. Pentru **a** oarecare, dat, să se construiasca un algoritm care calculează $\sqrt{a}$ ca limită a acestui şir,cu o precizie **eps** dată.

50. Şirurile $\{u_n\}$ şi $\{v_n\}$ generate cu relaţiile de recurenţă:
$u_n = (u_{n-1} + v_{n-1})/2$ şi $v_n = \sqrt{u_{n-1} v_{n-1}}$ pornind cu $u_0 = 1/|a|$, $v_0 = 1/|b|$, unde $a \neq 0$, $b \neq 0$ au o aceeaşi limită comună, valoarea integralei eliptice:

$$I = \frac{2}{\pi} \int_0^{\pi/2} \frac{dx}{\sqrt{a^2 \cos^2 x + b^2 \sin^2 x}}$$

Să se calculeze această integrală pentru **a** şi **b** daţi, ca limită comună a celor două şiruri, determinată aproximativ cu precizia **eps**, în momentul în care distanţa între termenii celor două şiruri devine inferioară lui **eps**, adică $|u_n - v_n| < eps$.

51. Pentru calculul lui $lg_2 x$ se generează şirurile $\{a_n\}$, $\{b_n\}$ şi $\{c_n\}$

$$a_n = \begin{cases} a_{n-1}^2 & \text{daca } a_{n-1}^2 < 2 \\ \frac{a_{n-1}^2}{2} & \text{daca } a_{n-1}^2 \geq 2 \end{cases} \quad \text{pornind cu } a_0 = x$$

cu relaţiile de recurenţă:

$$a_n = \begin{cases} a_{n-1}^2 & \text{dacă } a_{n-1}^2 < 2 \\ \frac{a_{n-1}^2}{2} & \text{dacă } a_{n-1}^2 \geq 2 \end{cases} \quad \text{pornind cu } a_0 = x$$

$b_n = b_{n-1} / 2$ pornind cu $b_0=1$

$$c_n = \begin{cases} c_{n-1} & \text{daca } a_{n-1}^2 < 2 \\ c_{n-1} + b_n & \text{daca } a_{n-1}^2 \geq 2 \end{cases} \quad \text{pornind cu } c_0 = 0$$

Se ştie că pentru $1 < x < 2$, $\lim c_n = \lg_2 x$.
Dacă $x \notin (1,2)$ se aduce argumentul în acest interval folosind relaţiile:
$\lg_2 x = - \lg_2 (1/x)$ pentru $x < 1$
$\lg_2 x = k+\lg_2 (x/2^k)$ pentru $x \geq 2^k$

52. Dezvoltarea în serie:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

este rapid convergentă pentru x mic.Pentru x oarecare,acesta se descompune sub forma:
$x = i + f$ in care:
i = partea intreagă a lui x
f = partea fracţionară a lui x.
Rezultă $e^x = e^i * e^f$ cu:

$$e^i = \underbrace{e \ e \ \cdots \ e}_{i} \quad \text{pentru} \quad i > 0$$

$$e^i = \underbrace{\frac{1}{e} \ \frac{1}{e} \ \cdots \ \frac{1}{e}}_{i} \quad \text{pentru} \quad i < 0$$

Pentru x dat, să se calculeze e^x cu o precizie eps dată.

53. Să se obţină reprezentarea ca fracţie zecimală a numărului m / n. Eventuala perioadă se afişează între paranteze.

54. Să se determine valoarea n pentru care:

$$S = \sum_{k=1}^{n} \frac{2}{\sqrt{4n^2 - k}}$$

satisface conditia $|S - \pi/3| < \varepsilon$, in care eps este dat.Se stie că.

$$\lim_{k\to\infty} \quad S = \frac{\pi}{3}$$

55. Să se calculeze funcţia Bessel de speţa I-a **J_n (x)** ştiind că există relaţia de recurenţă:

J_p (x) = (2p-2)/x*J_{p-1}(x) - J_{p-2}(x)

$$J_0(x) = \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{x}{2}\right)^{2k}}{(k!)^2}$$

$$J_1(x) = \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{x}{2}\right)^{2k+1}}{k!\,(k+1)!}$$

Calculele se fac cu precizia **eps** (**x, n** şi **eps** se dau pe mediul de intrare).

56. Pentru n dat să se calculeze suma:

$$S = \frac{1}{2} + \frac{1*3}{2*4} + \cdots + \frac{1*3\cdots(2n-1)}{2*4*\cdots 2n}$$

57. Să se calculeze π cu o precizie cunoscută ε ştiind că:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

58. Să se calculeze **sin(x)** şi **cos(x)** cu precizia dată eps utilizând dezvoltările în serie de puteri:

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

59. Fie şirurile **{a_n},{b_n},{c_n}** generate cu relaţiile de recurenţă:
a_n **= (b_{n-1}+c_{n-1})/2** **b_n = (c_{n-1}+a_{n-1})/2** **c_n = (a_{n-1}+b_{n-1})/2** cu
a_0 = alfa ; **b_0 = beta ;** **c_0 = gama**
alfa, beta,gama date. Ştiind că cele trei şiruri sunt convergente şi au o limită comună, să se calculeze cu o precizie **eps** dată această limită.

60. Să se calculeze prin dezvoltare in serie, cu precizia **eps** dată, **sin(x) :**

$$\mathbf{sin(x)} = \mathbf{x} - \frac{\mathbf{x}^3}{3!} + \frac{\mathbf{x}^5}{5!} - \frac{\mathbf{x}^7}{7!} + \cdots$$

Deoarece seria este rapid convergentă când argumentul se află în primul cadran, se va face reducerea sa la primul cadran utilizând următoarele relaţii:

sin(x) = - sin(-x) **dacă x < 0**

sin(x) = sin(x-2πn) **dacă x > 2πn**

sin(x) = - sin(x-π) **dacă x > π**

sin(x) = sin(π-x) **dacă x > π/2**