LUCRAREA 4

1.Operatori

Operatorii sunt elementele care permit efectuarea diferitelor operaţii între obiecte.
În C++ există două mari tipuri de operatori: unari sau binari.
Un operator unar se aplică unui singur operand iar un operator binar se aplică la doi operanzi.
Operatorul binar se aplică la operandul care îl precede imediat şi la care îl urmează imediat.
Operatorii limbajului C nu pot avea ca operanzi şiruri de caractere.
Operatorii limbajului C pot fi grupaţi în mai multe clase: operatori aritmetici, operatori relaţionali, operatori logici.

1.1 Operatori aritmetici

Operatorii aritmetici se utilizează la efectuarea calculelor cu date de diferite tipuri predefinite. Aceştia sunt:

a) operatori unari + (adunare) şi - (scădere);
b) operatori binari multiplicativi * (înmulţire), / (împărţire) şi % (restul împărţirii întregi - modulo);
c) operatori binari aditivi + şi -.

Operatorii unari sunt mai prioritari decât cei binari, iar operatorii multiplicativi sunt mai prioritari decât cei aditivi.
Operatorul unar + nu are nici un efect.
Operatorul unar – are ca efect negativarea valorii operandului pe care-l precede.
Pentru operatori binari formatul operaţiilor este următorul:
operand1 operator binar *operand2*.

Aplicaţie
Să se scrie un program care citeşte valoarea lui x, calculează valoarea polinomului $a(x) = 3x^{20} - 6x^{16} + 8x^{9} - 7x^{5} + 1$ şi afişează rezultatul. Calculele se fac în virgulă flotantă dublă precizie.

Pentru ridicarea la putere se va apela funcţia pow cu prototipul:
double pow(double x, double y); , definit în fişierul math.h. Rezultatul ridicării la putere va fi tot de tipul double.

```
#include<stdio.h>
#include<math.h>
main()
{
double x;
printf("tastati valoarea lui x = ");
scanf("%lf", &x);
printf("x=%g\ta(x)=%g\n", x, 3*pow(x,20)-6*pow(x,16)+8*pow(x,9)-7*pow(x,5)+1);
}
```

;	#include <iomanip>
	double my_double = 42.256;
	cout << fixed << setprecision(4);
	cout << my_double << endl;

```
double NR=125.356;
  cout.setf(ios::showpos);

   cout.precision(2); // two digits after decimal point
  cout.width(5);     // in a field of 10 characters
  cout << NR << endl << NR << "\n";
cout.setf(ios::scientific);
  cout.precision(2); // two digits after decimal point
  cout.width(10);     // in a field of 10 characters
  cout << NR << "\n ";
  cout.width(10);     // set width to 10
  cout << NR << "\n";
```

Să se scrie un program care citeşte valoarea variabilei X şi a coeficienţilor polinomului:
$q(x) = c4*x^4 + c3*x^3 + c2*x^2+c1*x +c0$,
calculează valoarea polinomului q(x) şi afişează rezultatul. Calculele se fac în virgulă flotantă dublă precizie.
Calculul valorii unui polinomului q(x) se poate face folosind funcţia de bibliotecă poly (definită în math.h) cu formatul următor:
double poly(double x, int n, double c]).
Această funcţie returnează valoarea polinomului de grad n, pentru valoarea lui x, coeficenţii polinomului fiind c[0], c[1], c[2], ... , c[n]; c[0] este termenul liber.

```
#include <stdio.h>
#include<math.h>
main()
{
        double x, c[5];
                printf("valoarea lui x= ");
        scanf("%lf",&x);
printf("coeficentii polinomului \n");
printf("c0 = ");
scanf("%lf", &c[0]);
printf("c1 = ");
scanf("%lf", &c[1]);
printf("c2 = ");
scanf("%lf", &c[2]);
printf("c3 = ");
scanf("%lf", &c[3]);
printf("c4 = ");
scanf("%lf", &c[4]);
printf("x = %g \t q(x)=%g \n", x, poly(x,4,c));
}
```

1.1.1. Regula conversiilor implicite

Această regulă se aplică la evaluarea expresiilor. Ea acţionează atunci când un operator binar se aplică la doi operanzi de tipuri diferite.
În cazul în care un operator se aplică la doi operanzi de tipuri diferite, operandul de tip "inferior" se converteşte spre tipul "superior" al celuilalt operand şi rezultatul este de tip "superior". Ordinea operanzilor este următoarea: long double, double, float, unsigned long, long, unsigned int.
Regula conversiilor implicite are în vedere şi nişte conversii generale independente de

operatori. Ea se aplică în mai mulţi paşi.
Înainte de toate se convertesc operanzii de tip char în tipul int.
Dacă operatorul curent se aplică la operanzi de acelaşi tip, atunci se execută operatorul respectiv, iar tipul rezultatului coincide cu tipul comun al operanzilor. Dacă rezultatul aplicării operatorului reprezintă o valoare în afara limitelor tipului respectiv, atunci rezultatul este eronat (are loc o "depăşire').

1.2. Operatori de relaţie

Operatorii de relaţie sunt:
< - mai mic;
<= - mai mic sau egal;
> - mai mare;
>= - mai mare sau egal.
Ei au aceeaşi prioritate care este mai mică decât a operatorilor aditivi.
O expresie de forma: E1 operator_de _relaţie E2, poate avea valoarea 1 dacă relaţia este adevărată şi 0 în caz contrar.

Aplicaţie
Să se scrie un program care citeşte două numere, primul este de tip int, iar al doilea este de tip double. Programul afişează valoarea 2 dacă numărul flotant este mai mare decât cel întreg şi 1 în caz contrar.

```
#include<stdio.h>
main()
{
        int i;
        double f;
        scanf("%d  %lf", &i, &f);
        printf("i=%d \t f=%f \t %d \n", i, f, (f > i) + 1);
}
```

1.2.1. Operatori egalitate

Operatori de egalitate sunt doi:
= = - egal;
!= - diferit.
Ei au aceeaşi prioritate care este imediat mai mică decât cea a operatorilor de relaţie .
O expresie de forma: E1 operator_de _relaţie E2, poate avea valoarea 1 dacă relaţia este adevărată şi 0 în caz contrar.

1.3. Operatori logici

Aceştia sunt:
! - negaţie logică (operator unar);
&& - şi logic;
|| - sau logic.
Negaţia logică are aceeaşi proprietate ca şi ceilalţi operatori unari ai limbajului C. De altfel, toţi operatorii unari au aceeaşi prioritate care este imediat mai mare decât cea a operatorilor multiplicativi.

Expresia: ***! a*** are valoarea zero (fals) dacă ***a*** are o valoare diferită de zero şi valoarea unu dacă ***a*** are valoarea zero.

Operatorul && are prioritatea mai mică decât cei de egalitate. O expresie de forma: ***E1&&E2*** are valoarea 1, dacă expresiile E1 şi E2 sunt ambele diferite de zero.

Operatorul || are prioritatea imediat mai mică decât operatorul &&. Expresia E1|| E2 are valoarea zero, dacă ambele expresii E1 şi E2 la care se aplică operatorul || au valoarea zero.

Aplicaţie

Să se scrie un program care citeşte un număr şi afişează unu dacă numărul respectiv aparţine intervalului [-1000, 1000] şi zero în caz contrar.

Dacă notăm cu a numărul citit, atunci el trebuie să satisfacă simultan relaţiile:

a>=-1000

şi

a<=1000.

Acest fapt se poate exprima cu ajutorul expresiei:

(a>=-1000)&&(a<=1000).

```
#include<stdio.h>
main()
{
        int i;
        double f;
        scanf("%d  %lf", &i, &f);
        printf("i=%d \t f=%f \t %d \n", i, f, (f > i) + 1);
}
```

1.3.1. Operatori logici pe biţi

Operatorii logici pe biţi sunt:

~ - complement faţă de unu (operator unar);
<< - deplasare la stânga;
>> - deplasarea dreapta;
& -şi logic pe biţi;
^ - sau exclusiv logic pe biţi;
| - sau logic pe biţi.

Aceşti operatori se aplică la operanzi de tip întreg. Ei se execută bit cu bit şi operanzii se extind la 16 biţi dacă este necesar.

Complementul faţă de unu are aceeaşi proprietate ca ceilalţi operatori unari. el schimbă fiecare bit al operandului cu zero şi fiecare bit zero al acestuia cu 1.

Operatorul << realizează o deplasare la stânga a valorii primului său operand cu un număr de poziţii binare egal cu valoarea celui de-al doilea operand al său. Această operaţie este echivalentă cu o inmulţire cu puteri ale lui 2.

În mod analog, operatorul >> realizează o deplasare la dreapta a valorii primului său operand cu un număr de poziţii binare egal cu valoarea celui de-al doilea operand al său. Această operaţie este echivalentă cu o împărţire cu puteri ale lui 2.

1.4. Operatorii de atribuire

Operatorul de atribuire, în forma cea mai simplă, se notează prin caracterul "=". El se utilizează în expresii de forma:

v=(expresie),
unde v este o variabilă simplă, referenţiază un element de tablou sau de structură.
Operatorul de atribuire are prioritate mai mică decât toţi operatorii pe care i-am întâlnit până în prezent. Acest operator are ca efect atribuirea valorii expresiei aflată în dreapta semnalului de atribuire variabilei v.
La atribuire se face şi o conversie dacă este necesar. Astfel, valoarea expresiei din dreapta semnalului de atribuire se va converti spre tipul variabilei din stânga lui înainte de a se face atribuirea, dacă cele două tipuri sunt diferite.
Pentru operatorii de atribuire, în afara semnului = se mai poate folosi şi succesiunea de caractere:
op=
unde op este un operator binar aritmetic sau logic pe biţi. Deci op poate fi unul din operatorii:
/, % ,* ,-, +, <<, >>, &, ^, | .
această construcţie se foloseşte pentru a face prescurtări.
Expresia v op = expresie
este echivalentă cu expresia de atribuire:
v=vop(expresie).

1.5. Operatori de incrementare şi decrementare

Aceşti operatori sunt unari şi au aceeaşi prioritate ca şi ceilalţi operatori unari ai limbajului C.
Operatorul de incrementare se notează ++, iar cel de decrementare se notează cu - -.
Operatorul de incrementare măreşte valoare operandului său cu 1, iar cel de decrementare micşorează valoarea operandului său cu 1.
Aceşti operatori pot fi: prefixaţi (++operand sau - -operand) sau postfixaţi (operand++ sau operand - -).
În situaţia în care un operator de incrementare sau decrementare este prefixat, se foloseşte valoarea operandului la care s-a aplicat operatorul respectiv.
În cazul în care un operator de incrementare sau decrementare este postfixat, se foloseşte valoarea operandului dinaintea aplicării operatorului respectiv.
Exemplu:
Presupunem că valoarea x are valoarea 3. Dacă considerăm expresia expresia de atribuire y =++x atunci lui y i se atribuie valoarea 4 (la atribuire se foloseşte valoarea incrementată).
Dacă utilizăm expresia de atribuire y=x++, atunci lui y i se atribuie valoarea 3 (valoarea dinaintea incrementării).

1.5. Operatorul de forţare a tipului sau de conversie explicită (expresie cast)

Putem să specificăm conversia valorii unui operand spre un tip dat folosind o construcţie de forma:
(tip) operand.
Într-o astfel de construcţie (tip) este un operator unar (operator de forţare a tipului sau de conversie explicită) iar valoarea operandului se converteşte spre tipul indicat în paranteze.
Această construcţie o vom numi *expresie cast*.
Exemplu:
Presupunem că funcţia f are un parametru de tip double. Fie declaraţia:
int n;
Atunci, pentru a apela f cu parametrul n, este necesar ca, n să se convertească spre double. Acest lucru se poate realiza printr-o atribuire:
double x;

. . .
f(x=n);
Un alt mod mai simplu de conversie a parametrului întreg spre tipul double este utilizarea unei expresii cast:
f((double)n).
Operatorul (tip) fiind unar, are aceeaşi prioritate ca şi ceilalţi operatori ai limbajului C.

1.6. Operatorul dimensiune

Dimensiunea în octeţi a unei date sau a unui tip se poate determina folosind operatorul *sizeof*. El poate fi folosit sub forma:
sizeof data sau sizeof(data)
sau
sizeof(tip)
unde data poate fi:

- un nume de variabilă simplă;
- un nume de tablou;
- un nume de structură;
- referirea la un element de tablou (variabilă cu indici);
- referirea la elementul unei structuri.

Tip poate fi:

- un cuvânt sau cuvintele cheie ale unui tip predefinit;
- o construcţie care defineşte un tip.

Operatorul dimensiune (sizeof) fiind unar, are aceeaşi prioritate ca şi ceilalţi operatori ai limbajului C.
Exemplu:
int x;
sizeof x – are valoarea 2, deoarece pentru variabila x se alocă 2 octeţi.

long double y[7];
sizeof y[3]; - are valoarea 10;
sizeof y; - are valoarea 70.

1.7. Operatorul adresă

Operatorul adresă este unar şi se notează prin caracterul &. El se aplică pentru a determina adresa de început a zonei de memorie alocată unei date. În forma cea mai simplă, acest operator se utilizează în construcţii de forma:
&nume
unde *nume* este numele unei variabile simple sau a unei structuri.
Menţionăm că în cazul în care nume este numele unui tablou, atunci acesta are ca valoare chiar adresa de început a zonei de memorie alocată tabloului respectiv. Deci în acest caz nu se mai utilizează operatorul &.

1.8. Operatorii paranteză

Parantezele rotunde se utilizează fie pentru a include o expresie, fie la apelul funcţiilor.
Acestea au prioritate maximă.

Operanzii obţinuţi prin includerea unei expresii între paranteze impun anumite limite asupra operatorilor. De exemplu, nu se pot aplica operanzii de incrementare , decrementare sau adresă asupra unor expresii între paranteze.
Exemple de cazuri eronate:

- (i+1)++ ;
- (x+y);
- &(x+y).

Parantezele pătrate includ expresii care reprezintă indici. Ele se numesc operatori de indexare.

1.9. Operatori condiţionali

Operatorii condiţionali permit construirea unor expresii a căror valoare să depindă de valoarea unei condiţii.
Prin condiţie înţelegem o expresie care poate avea două valori: adevărat sau fals. În limbajul C, o condiţie se reprezintă printr-o expresie oarecare.
Ea are valoarea adevărat dacă este diferită de zero şi fals în caz contrar.
Numim expresie condiţională, o expresie a cărei valoare şi tip este dependentă de valoarea unei condiţii.
O expresie condiţională are formatul:
E1?E2:E3
unde: E1, E2, E3 sunt expresii.
Această expresie condiţională se evaluează astfel:
Se determină valoarea expresiei E1.
Dacă E1 are o valoare diferită de zero (are valoarea adevărat), atunci valoarea şi tipul expresiei condiţionale coincide cu valoarea şi tipul expresiei E2. Altfel (adică E1 are valoarea zero) valoarea şi tipul expresiei condiţionale coincide cu valoarea şi tipul expresiei E1.
Operatorii condiţionali **?** şi **:** se folosesc totdeauna împreună şi în ordinea indicată în formatul expresiei condiţionale.
Ei au o prioritate mai mică decât prioritatea operatorului sau logic (||) şi imediat mai mare decât prioritatea operatorilor de atribuire.
De exemplu putem scrie: v=E1?E2:E3

Aplicaţie
Să se scrie un program care citeşte două numere şi afişează maximul dintre ele.

```
#include<stdio.h>
main()
{
    double a,b;
        scanf("%lf %lf", &a, &b);
        printf("a= %g\t b=%g\tmax(a,b)=%g\n",a,b,a>b?a:b);
}
```

Să se scrie un program care citeşte valoarea variabilei x şi afişează valoarea funcţiei f(x) definită ca mai jos:

$$f(x)=\begin{cases} 3x^2+7x-10 \ pentru \ x<0; \\ 2 \ pentru \ x=0; \\ 4x^2-8 \ pentru \ x>0. \end{cases}$$

```
#include<stdio.h>
main()
{
double x;
scanf("%lf", &x);
printf("x= %g \t f(x) =%g\n ", x, x<0?3*x*x+7*x-10:(x>0?4*x*x-8:2));
}
```

Aplicaţii

1. Să se realizeze un program care să permite introducerea a două numere întregi de la tastatură. Programul va afişa rezultatul corect al înmulţirii celor două numere.

2. Să se realizeze un program care să permite introducerea a două numere întregi de la tastatură. Programul va afişa rezultatul corect al împărţirii celor două numere.

3. Să se realizeze un program care să permită introducerea a două numere de la tastatură. Programul va afişa rezultatul 1 dacă cele două numere au acelaşi semn şi rezultatul 0 dacă cele două numere a semne diferite.

Numar prim:

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
    cout << "Hello world!" << endl;
    int tab[20],n,i=0,var,ok,d,nr;
    printf("\n numarul numerelor=");
    scanf("%d",&n);
        nr=0;
    do{
    printf("\n introduceti un numar");
    scanf("%d",&var);
        ok=0; d=2;
            while((d<=var/2)&&!ok)
                {if((var%d)==0)
                    {ok=1;d++;}
                else d++;
                 }
            if(!ok)
                {nr++;printf("\n numar prim %d",var);}
    tab[i]=var;
    i++;
    }
    while(i<n);
    {
    cout << endl;
    printf("numere prime=%d",nr);
    }
    return 0;
}
```