Instrucţiuni

1. Instrucţiunea "if - else"

În această lucrare de laborator este prezentată instrucţiunea "**if - else**" şi modul de utilizare a acesteia în construirea unor aplicaţii.

Instrucţiunea "if - else" permite programarea unei structuri de decizie în care o condiţie (rezultatul evaluării unei expresii) determină una din două posibilităţi:

- executarea sau nu a unei secvenţe de instrucţiuni;
- executarea unei secvenţe din două alternative.

Sintaxa:

Varianta 1:

```
if (expresie)
{
instrucţiune_a;
}
```

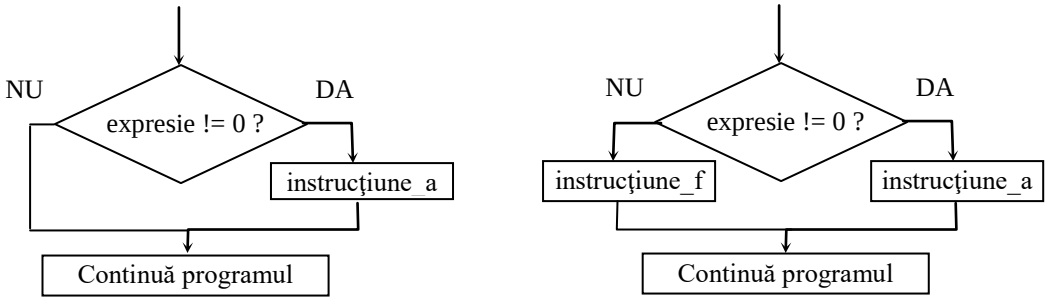
Varianta 2:

```
if (expresie)
{
instrucţiune_a;
}
else
{
instrucţiune_f;
}
```

unde:

expresie : are o valoarea de tip scalar şi reprezintă condiţia de decizie;

instrucţiune_a, instrucţiune_f: sunt instrucţiuni oarecare, executate dacă expresia este adevărată, respectiv falsă.



Schemele logice ale instrucţiunii **if**

Dacă avem de făcut o selecţie multiplă, aceasta se poate programa cu mai multe instrucţiuni **if-else** în cascadă:

```
if (expr)  {
    if (expr1)  instr1;
    }
else  instr2;
```

```
if (expr)
    if (expr1)  instr_a;
     else  instr_f;
```

```
if (expr_1) instr_a1;
else  if (expr_2)  instr_a2;
        else  if(expr_3) instr_a3;
                else instr_f3;
...
```

Exemple pentru instrucțiunea **if**

```
if (7 > 4) {
cout << "Yes";
}
```

Operatori Relaționali

Operator	**Descriere**	**exemple**		
			Valoare booleana	Valoare numerica
>	Mai mare ca	7>4	True	1
>=	Mai mare sau egal cu	7>=4	True	1
<	Mai mic ca	7<4	False	0
<=	Mai mic sau egal cu	7<=4	False	0
==	Egal cu (identic cu)	7==4	False	0
!=	Neegal cu (diferit fata de)	7!=4	True	1

Exemplu 2:

```
int a = 98;
int b = 76;
if (a  > b) {
  cout << "greatest is " << a << endl;
}
if (b > a) {
  cout << "greatest is " << b << endl;
}
```

Exemplu 3:

```
#include <iostream>
using namespace std;
int main()
{
    int mark = 100;
    cout<<"Introduceti un numar intreg de la 0 la 100 : ";
    cin>> mark;
    if (mark >= 50) {
        cout << "Ai trecut examenul." << endl;
        if (mark == 100) {
            cout <<"Perfect!" << endl;
        }
    }
    else {
        cout << "Ai picat examenul." << endl;
    }
    return 0;
}
```

Exemplu 4:

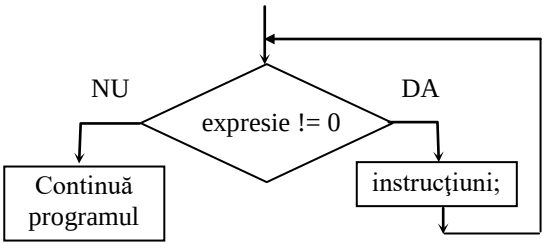
```
#include <iostream>
using namespace std;
int main()
{
    int age = 100;
    cout<<"Introduceti varsta de la 0 la 100 : ";
    cin>> age;

    if (age >= 0 && age <= 1)
            cout << "Bebelus";
  else  if(age > 1 && age < 14)
        cout << "Copil";
  else  if (age >=14 && age <= 18)
        cout << "Adolescent";
  else    if (age >=18 && age<= 65)
          cout << "Adult";
   else   if (age > 65)
        cout << "Batran";
   else
        cout << "Ceva este gresit";
   return 0;
}
```

2. Instrucţiunea "while"

Permite programarea ciclurilor cu test iniţial, conform schemei logice următoare:

Sintaxa:

```
while (expresie)
{
instrucţiune;
}
```



Schema logică a instrucţiunii **while**

Instrucţiunea (sau instrucţiunile) se execută de zero sau mai multe ori, atâta timp cât condiţia (expresia) testată este adevărată (nenulă).

De obicei, valoarea expresiei care constituie condiţia de ciclare este modificată în cursul iteraţiilor de secvenţa de instrucţiuni repetată, pentru a asigura ieşirea din ciclu – de exemplu cu expresia **i++**.

```
int num = 1;
while (num < 6) {
  cout << "Number: " << num << endl;
  num = num + 1;
}
```

```
int num = 1;
while (num < 6) {
  cout << "Number: " << num << endl;
  num = num + 3;
}
```

Calculul sumei primelor 5 numere intregi:

```
int num = 1;
int number;
int total = 0;

while (num <= 5) {
  cin >> number;
  total += number;
  num++;
}
cout << total << endl;
```

3. Instrucţiunea "for"

Instrucţiunea "for" oferă cea mai compactă metodă de programare a ciclurilor cu test iniţial, motiv pentru care este cel mai des utilizată.

Sintaxa:

```
for (<expresie_1>;<expresie_2>;<expresie_1>)
{
<instrucţiuni>;
}
```

Rolurile celor trei expresii sunt:

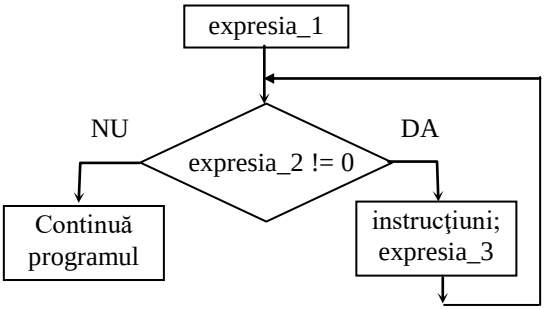
Expresia_1: se evaluează o singură dată, înaintea primei iteraţii, pentru a efectua iniţializările necesare (de obicei contorul ciclului şi/sau alţi parametrii ai iteraţiilor) – de ex: **i = 0**;

Expresia_2: este evaluată şi testată înaintea fiecărei iteraţii şi reprezintă condiţia de ieşire din ciclu, de exemplu: **i<=5**;

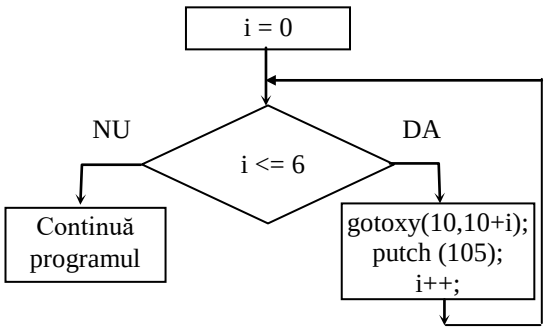
Expresia_3: se evaluează la sfârşitul fiecărei iteraţii, pentru a actualizarea parametrilor ciclului, de exemplu **i++**.

Această sintaxă reprezintă de fapt forma compactizată a unui ciclu **while**. Efectul este similar secvenţei:

```
expresie_1;
while (expresie_2)
        {
        instrucţiune;
        expresie_3;
        }
```



Schema logică a instrucţiunii **for**



Schema logică a instrucţiunii **for** pentru construirea unei drepte verticale din 7 caractere "105"

Afișarea numerelor de la 0 la 9:

```
for (int a = 0; a < 10; a++) {
  cout << a << endl;
}
```

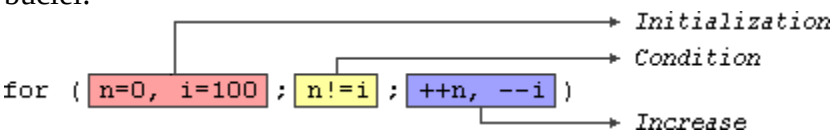
Afișarea numerelor din 10 în 10

```
for (int a = 0; a < 50; a+=10) {
  cout << a << endl;
}
```

Instrucțiunea for cu declarații complexe:

```
for ( n=0, i=100 ; n!=i ; ++n, --i )
{
  // cod oarecare
}
```

Această buclă se va executa de 50 de ori în cazul în care nici n sau i sunt modificate în cadrul buclei:



n începe cu o valoare de 0 și i cu 100, condiția este n! = i (adică, n care nu este egală cu i). Deoarece n este mărită cu una și am scăzut cu câte una pe fiecare iterație, condiția buclă va deveni falsă după a 50-a iterație, când atât n cât și i sunt egale cu 50.

Bucla for bazata pe o plaja de valori

for (declaration : range) statement;

Acest tip de loop iterează peste toate elementele din raza de acoperire, unde declarația declară o variabilă care poate lua valoarea unui element din acest interval. Range-urile sunt secvențe de elemente, inclusiv matrice, containere și orice alt tip care susține funcțiile încep și termină; Cele mai multe dintre aceste tipuri nu au fost încă introduse în acest moment.

```
// range-based for loop
#include <iostream>
#include <string>
using namespace std;

int main ()
{
 string str {"Hello!"};
 for (char c : str)
 {
  cout << "[" << c << "]";
 }
 cout << '\n';
}
```

```
[H][e][l][l][o][!]
```

Această buclă este automată și nu necesită declararea explicită a oricărei variabile contor.

Buclele bazate pe range-uri utilizează, de obicei, și deducerea de tip pentru tipul de elemente cu auto. În mod obișnuit, bucla bazată pe range-uri de mai sus poate fi, de asemenea, scrisă ca:

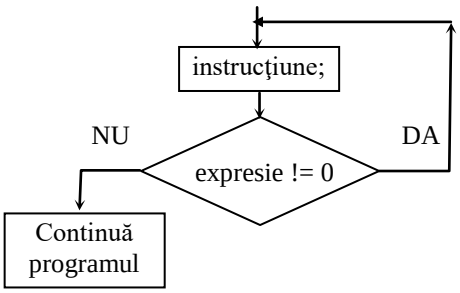
```
for (auto c : str)
  cout << "[" << c << "]";
```

4. Instrucţiunea "do - while"

Permite programarea ciclurilor cu test final, conform schemei logice:

Sintaxa:

do instrucţiune;
while (expresie);



Schema logică a instrucţiunii **do - while**

Instrucţiunea se execută cel puţin odată şi se repetă atâta timp cât expresia este adevărată.

```
int a = 0;
do {
  cout << a << endl;
  a++;
} while(a < 5);
```

5. Instrucţiunea "switch"

Instrucțiunea **switch** testează o variabilă (care poate fi de tip **char**, **short**, **byte**, **int** sau **string**) dintr-o listă de valori, care se numesc **case**, pentru a determina când/dacă variabila este egală cu oricare din valori.

Sintaxa:

```
switch (variabilă) {
        case valoare1:
                Instructiuni;
        break;
        case valoare2:
                Instructiuni;
        break;
        …
        case valoaren:
                Instructiuni;
        break;
        default:
                Instructiuni;
```

```
int i=2;
cout<<"i = "; cin>>i;
switch(i)
{
case 1: cout<<(" cazul 1" ); break;
case 2: cout<< ("cazul 2");break;
case 3: cout<< ("cazul 3");break;
case 4: cout<< ("czzul 4");break;
default: cout<< ("etc.");
}
```

```
int age = 42;
switch (age) {
  case 16:
    cout << "Too young";
    break;
  case 42:
    cout << "Adult";
    break;
  case 70:
    cout << "Senior";
    break;
}
```

1. Aplicaţii:

Instrucţiunea FOR

Exemplul 1: Suma a N numere introduse de utilizator:

```
int i, n;
float a, suma=0;
cout << "N = "; cin >> n;

for(i=1;i<=n;i++)
   {
   cout<< "numarul " << i  << " = " ;
  cin >> a;
   suma+=a;

   }
cout << "suma  =  " <<  suma;
```

Observaţii: Variabila **i** este utilizată pe post de "contor" al instrucţiunii **for**, numărând la a câta iteraţie s-a ajuns. Execuţia instrucţiunii **for** se încheie atunci când numărul de iteraţii devine egal cu **n**. Iniţializarea lui **i** cu 1 se realizează o singură dată, la început; **i<=n** este condiţia de continuare a execuţiei; **i++** se efectuează după fiecare execuţie a ciclului (postincrementare).

Instrucţiunea WHILE

Exemplul 2 Suma a N numere introduse de utilizator

```
void main()
{
float a;
int i,n;
float suma=0;
i=1;
printf("Numarul de elemente N="); scanf("%d", &n);

while(i<=n)
  {
```

```
  printf("Elementul %d este:", i); scanf("%f", &a);
 suma=suma+a; /* se mai poate scrie suma+=a */
 i++;
 }
printf("Suma numerelor este %f=\n", suma);
getch();
}
```

Instrucţiunea DO-WHILE

Dacă rescriem exemplul 1 utilizând **do..while**, obţinem:

```
..... .
do
        {
        scanf("%f", &a);
        suma+=a;
        i++;
        } while(i<=n);
```

***Exemplul 3*:** Scrieţi un program care calculează maximul a 3 numere.

```
void main (void)
{
int a,b,c;
int max;
printf(”a =”);  scanf("%d", &a);
printf(”b =”);  scanf("%d", &b);
printf(”c =”);  scanf("%d", &c);
if (a<=b)
        if (c<=b) max=b;
        else max=c;
else    if (c<=a) max=a;
        else max=c;
printf(”Maximul este= %d,\n”,max);
getch();
}
```

Obs. A se observa şi modul cum a fost scris programul, astfel încât să se vadă clar cărei instrucţiuni **if** îi aparţine fiecare **else**.

Exemplul 4: Se dau trei numere întregi a,b,c. Să se testeze dacă numerele date pot fi lungimile laturilor unui triunghi.
Obs: Nu am mai verificat daca a, b , c sunt pozitive.

```
void main (void)
{
int a,b,c;
printf(”a =”);  scanf("%d", &a);
printf(”b =”);  scanf("%d", &b);
printf(”c =”);  scanf("%d", &c);

if ((a<b+c) && (b<a+c) && (c<a+b))
   Printf(”Numerele pot fi lungimile laturilor unui triunghi !”);
else
   Printf(”Numerele nu pot fi lungimile laturilor unui triunghi !”);
}
```

Obs: Am folosit o singura instrucţiune if, cele trei condiţii sunt conectate prin operatorul ŞI (&&) logic. După cum se ştie, dacă avem propoziţiile p,q,r, atunci propoziţia p&&q&&r va fi adevărată dacă toate propoziţiile componente sunt adevărate si falsă dacă una dintre propoziţiile componente este falsă. Este indicată aceasta scriere pentru că sporeşte lizibilitatea programului.

Exemplul 5: Rezolvarea ecuaţiei de gradul I.

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
    cout << "Hello world!" << endl;
    int a,b;
float x;
printf("a ="); scanf("%d", &a);
printf("b ="); scanf("%d", &b);
if (a!=0)
 {
 x=(float)-b/a;
 printf("x= %f\n",x);
 }
else printf(" Ecuatia nu are solutii\n");
    return 0;
}
```

Varianta cu cout si cin:

```
cout << "a = " ; cin >> a;
cout << "b = " ; cin >> b;
if (a!=0)
 {
 x=(float)-b/a;
 cout << "x = " << x ;
 }
else printf(" Ecuatia nu are solutii\n");
```

Exerciţii propuse:

1. Calculaţi minimul a trei numere întregi
2. Se citeşte o valoare reală x. Să se determine modulul lui x.
3. Să se ordoneze crescător trei numere întregi introduse de la tastatura.
4. Sa se determine aria si perimetrul unui triunghi daca se cunosc lungimile laturilor, verificând în prealabil dacă triunghiul dat există. $S = \sqrt{p(p-a)(p-b)(p-c)}$; unde $p = \frac{a+b+c}{2}$ semiperimetrul
5. Să se determine tipul unui triunghi (oarecare, isoscel , echilateral, dreptunghic) dacă se cunosc lungimile laturilor.
6. Să se scrie un program C++ care să rezolve ecuaţia de gradul al II-lea