

Elementele de baza ale limbajului de programare C++



1. Competențe	3
2. Noțiuni introductive	4
3. Structura generala a unui program C++	8
4. Elementele de bază ale unui limbaj de programare	10
5. Vocabularul limbajului C++	11
6. Tipuri simple de date (standard)	17
7. Constante și variabile	20
8. Operatori și expresii	24
9. Operații de citire și scriere	37
10. Instrucțiunile limbajului C++	43
11. Aplicații	73
12. Bibliografie & webografie	74



Competențe generale

- *implementarea algoritmilor într-un limbaj de programare*
- *aplicarea algoritmilor fundamentali în prelucrarea datelor*

Competențe specifice

- *transcrierea algoritmilor din limbaj pseudocod în limbaj de programare*
- *elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specialității*
- *alegerea unui algoritm eficient de rezolvare a unei probleme*



2. Noțiuni introductive

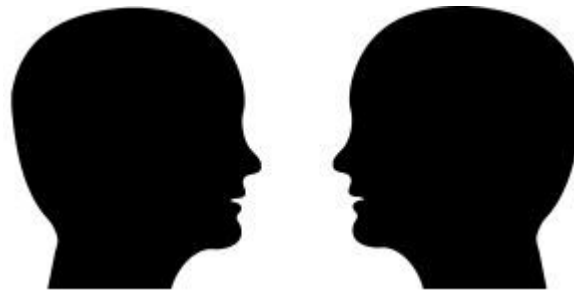
Orice limbaj constituie un mijloc de comunicare între două entități: emițătorul și receptorul.

În general limbajele sunt de două tipuri:

- limbaje naturale;
- limbaje artificiale.

Limbajele naturale s-au constituit de-a lungul timpului, în procesul conlucrării membrilor societății.

Limbajele artificiale au fost și sunt create pentru comunicarea într-un domeniu particular de activitate.



Limbajele de programare fac parte din categoria limbajelor artificiale, fiind utilizate în procesul de comunicare om-calculator.

Un limbaj de programare reprezintă un mijloc de comunicare între programator și calculator.

Un ***limbaj de programare*** este un mijloc de comunicare particular, în care informația ce trebuie comunicată este codificată printr-un program pe baza a trei componente:

- un ***set de acțiuni***, care acționează asupra unui
- ***set de date*** într-o anumită
- ***secvență de acționare***.



Repere istorice în evoluția limbajelor de programare:

- 1955 – FORTRAN (FORmula TRANslation)
- 1960 – ALGOL (ALGOrithmic Language)
- 1960 – COBOL (COmmon Business Oriented Language)
- 1971 – Pascal (Blaise PASCAL)
- 1972 – C
- 1980 – C++
- 1995 – Java

La începutul anilor 70 a apărut limbajul C – creația lui Dennis Ritchie și Brain Kernighan.

Limbajul C++ este creația lui Bjarne Stroustrup și reprezintă o extensie a limbajului C care permite programarea pe obiecte.



Realizarea unui program scris în C++ necesită parcurgerea a patru etape:

- **editare** – scrierea programului sursă, prin crearea unui fișier cu extensia **cpp**;
- **compilare** – se aduce în memoria internă programul sursă, se verifică erori și se convertește acest program în program obiect, având extensia **obj**;
- **link-editare** – se leagă programul obiect cu bibliotecile de sistem și se transformă într-un program executabil având extensia **exe**;
- **execuție** – se lansează în execuție programul obiect: se efectuează citirea datelor, calculele și scrierea rezultatelor, formându-se fișierul.



3. Structura generală a unui program C++

- un program C++ este constituit dintr-o succesiune de module, denumite **funcții**
- una dintre aceste funcții este funcția principală, denumită **main()**
- **main()** este o funcție specială, care trebuie să apară obligatoriu o singură dată în orice program C++
- execuția oricărui program începe cu funcția **main()**
- o funcții este constituită din **antet** și **corp**
- **antetul** funcției conține numele funcției, tipul rezultatului pe care îl calculează funcția și o listă de parametri prin care funcția comunică cu exteriorul ei, încadrată între paranteze rotunde
- **corpul** funcției conține declarații și instrucțiuni care specifică prelucrările realizate de funcția respectivă



Exemple

```
1.    void main()  
      {  
          . . . . .  
      }
```

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

Instrucțiunea **return** este utilizată pentru a încheia execuția unei funcții și a returna valoarea expresiei specificate în instrucțiunea **return** ca valoare a funcției.



4. Elementele de bază ale unui limbaj de programare

Limbajul C++ este caracterizat de:

- ***sintaxă*** – este formată din totalitatea regulilor de scriere corectă a programelor;
- ***semantică*** – reprezintă semnificația construcțiilor corecte din punct de vedere sintactic;
- ***vocabular*** – este format din totalitatea cuvintelor care pot fi folosite într-un program.



5. Vocabularul limbajului C++

Vocabularul limbajului C++ este format din:

- setul de caractere;
- identificatori;
- cuvinte cheie;
- comentarii;
- separatori.



a. Setul de caractere

Setul de caractere utilizat pentru scrierea programelor C++ este setul de caractere al codului ASCII.

Codul ASCII este format din:

- literele mari și mici ale alfabetului latin (A-Z, a-z);
- cifrele sistemului de numerație zecimal (0-9);
- caracterele speciale (blank, +, *, %, =, {, !, #, etc.).



b. Identificatori

Identificatorii (numele) au rolul de a denumi elemente ale programului precum constante, variabile, funcții etc.

Identificatorii:

- reprezintă o secvență de litere, cifre și _ (linia de subliniere) care trebuie să înceapă cu _ sau cu o literă;
- nu pot fi cuvinte cheie (rezervate) ale limbajului.

Exemple suma
 Suma
 suma1
 suma_1
 _suma

Contraexemple suma 1
 1suma
 suma+1
 suma&nr
 suma nr



c. Cuvinte cheie (rezervate)

Cuvintele cheie (keywords) sunt cuvinte care au un înțeles bine definit și nu pot fi folosite în alt context.

Exemple

<code>bool</code>	<code>default</code>	<code>for</code>	<code>struct</code>
<code>break</code>	<code>do</code>	<code>if</code>	<code>switch</code>
<code>case</code>	<code>double</code>	<code>int</code>	<code>unsigned</code>
<code>char</code>	<code>else</code>	<code>long</code>	<code>void</code>
<code>const</code>	<code>float</code>	<code>return</code>	<code>while</code>



d. Comentarii

Pentru ca un program să fie ușor de înțeles se folosesc comentariile. Acestea sunt texte care vor fi ignorate de compilator, dar au rolul de a explica pentru programator anumite secvențe de program.

```
// comentariu
```

sau

```
/*comentariu  
comentariu  
.....*/
```



e. Separatori

Separatorii se folosesc pentru a delimita unitățile sintactice.

Separatori:

- blank
- TAB
- caracterele de control CR+LF generate de tasta Enter
- virgula



6. Tipuri simple de date (standard)

Prin **date** se înțelege, 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;
- operațiile care se pot efectua cu datele respective.

Clasificarea tipurilor de date:

- tipuri de date predefinite - asociate cu un cuvânt cheie, utilizat în declarație;
- tipuri de date definite de utilizator.



Tipuri standard în C++:

- **int** – pentru memorarea numerelor întregi;
- **float** și **double** pentru memorarea numerelor reale;
- **char** – pentru memorarea caracterelor;
- **bool** – pentru memorarea valorilor de adevăr (*true* sau *false*)
- **void** – pentru tip neprecizat;
- tipul *pointer* – pentru memorarea adreselor de memorie.

Tipul **void** este un tip special, pentru care mulțimea valorilor este vidă. Acest tip se utilizează atunci când este necesar să specificăm absența oricărei valori. De exemplu, poate fi utilizat pentru a specifica tipul unei funcții care nu returnează niciun rezultat.



Tipuri simple de date (standard)

Tipuri standard în C++. Domeniul de valori și dimensiunea memoriei ocupate:

	Tip	Valori	Număr octeți
Tip întreg	int	[-2147483648, 2147483647]	4
	unsigned int	[0, 4294967295]	4
	long int	[-2147483648, 2147483647]	4
	unsigned long int	[0, 4294967295]	4
Tip real	float	$[3.4 \cdot 10^{-38}, 3.4 \cdot 10^{+38}] \cup [-3.4 \cdot 10^{+38}, -3.4 \cdot 10^{-38}]$	4
	double	$[1.7 \cdot 10^{-308}, 1.7 \cdot 10^{+308}] \cup [-1.7 \cdot 10^{+308}, -1.7 \cdot 10^{-308}]$	8
	long double	$[3.4 \cdot 10^{-4932}, 1.1 \cdot 10^{+4932}] \cup [-3.4 \cdot 10^{+4932}, -1.1 \cdot 10^{-4932}]$	12
Tip caracter	char	[-128, 127]	1
	unsigned char	[0, 255]	1



7. Constante și variabile

O categorie aparte de date o reprezintă constantele și variabilele.

Constantele

- constanta are un tip și o valoare fixă pe toată durata execuției programului care o conține;
- tipul și valoarea unei constante se definesc prin caracterele care compun constanta respectivă.

Constantele se clasifică astfel:

- numerice: - întregi
- reale
- caracter
- șir de caractere



Declaraarea constantelor

Sintaxa:

```
const [tip_dată] nume=valoare;
```

unde:

- **const** este un cuvânt cheie care înseamnă definirea unei constante simbolice;
- **tip_dată** precizează tipul constante (poate lipsi);
- **nume** este identificatorul constantei;
- **valoare** este valoarea constantei.

Exemple:

```
const int a=0;  
const int x=-5;  
const b=0;  
const float PI=3.14;  
const char a='a';  
const char sir[]="info";
```



Variabile

- nume asociat cu una sau mai multe locații de memorie;
- valoarea păstrată în aceste locații se poate modifica în cursul execuției programului;
- trebuie declarate – se specifică tipul și numele.

Declararea variabilelor

Sintaxa:

```
tip_dată nume;
```

unde:

- ***tip_dată*** precizează tipul datei memorate în variabila de memorie;
- ***nume*** este identificatorul variabilei de memorie.



Exemple

```
int a;  
int x,y;  
char b;  
int a,b=1, c=2;  
float d=1;  
float e=1.234;  
char f='a';  
long x1,x2;  
unsigned int p,q;  
char sir[]="info";
```



Operatori

Operatorii sunt caractere speciale care indică operația care se efectuează în cadrul unui program.

Clasificarea operatorilor:

- operatori aritmetici;
- operatori relaționali;
- operatori de egalitate;
- operatori de incrementare și decrementare;
- operatori logici;
- operatori de atribuire;
- operatorul ‘,’ (virgulă);
- operatorul de conversie explicită.



a. Operatori aritmetici

- - minus (unar) – pentru semn
- + plus (unar) – pentru semn
- + (binar) – adunare
- - (binar) – scădere
- * (binar) – înmulțire
- / (binar) – împărțire întreagă
- % (binar) – restul împărțirii întregi

Exemple

```
int a=3,b=4,p,c,r;
```

```
p=a*b;
```

```
c=a/b+p;
```

```
r=a%b;
```



b. Operatori de comparație (relaționali)

- < mai mic
- > mai mare
- <= mai mic sau egal
- >= mai mare sau egal

Rezultatul obținut în cazul aplicării unuia dintre operatorii relaționali este *true* sau *false*.

Exemple

`2<=5`

`4<3`

`int x=4,y=5,c;`

`c=x>y;`



c. Operatori de egalitate

- == egal
- != diferit

Rezultatul obținut în cazul aplicării unuia dintre operatorii de egalitate este **true** sau **false**.

Exemple

3==3

5==8

3!=6

4!=4

```
int a=8,b=8,x;
```

```
x=a==b;
```



d. Operatori de incrementare și decrementare

- ++ incrementare (adună 1)
- -- decrementare (scade 1)

Exemple

```
int a=8 , b=5 , c=3 , x ;  
a++ ;                //a=9  
x=b-- ;              //x=5, b=4  
x=++c ;              //x=4, c=4
```



e. Operatori logici

- `&&` ȘI logic
- `||` SAU logic
- `!` negație

Rezultatul obținut în cazul aplicării unuia dintre operatorii logici este *true* sau *false*.

Exemple

`a<=b && a<=c`

`a>5 || b<8`

`!(a==b)`

p	q	p && q
0	0	0
0	1	0
1	0	0
1	1	1

p	q	p q
0	0	0
0	1	1
1	0	1
1	1	1

p	!p
0	1
1	0

f. Operatori de atribuire

- = egal
- *=
- /=
- %=
- +=
- -=

Exemple

```
int a=2 ,b=3 ,c=4 ;
```

```
a=b ;
```

```
b+=a ;
```

```
c=b=a ;
```

```
//b=b+a
```



g. Operatorul ‘,’ (virgulă)

Separă mai multe expresii.

Exemple

```
int a=1, b=5;  
float c;  
c=a=b+1, a=c+2, b=b+1;  
    //b+1=6; a=6; c=6  
    //a=6+2=8;  
    //b=5+1=6;
```



h. Operatorul de conversie explicită

Pentru ca un operand să intre în calcul convertit așa cum ne dorim (nu implicit) înaintea operandului se trece tipul său.

Exemple

```
float x=25.79;  
x=(int)x;           //x=25  
x=int(x);          //x=25  
float a=8, b=3, c;  
c=a/b;             //c=2.66667
```



Prioritatea operatorilor

Precedență	Operatori	Simbol	Asociativitate
1	apel funcție / selecție	() [] . ->	SD
2	unari	* & - ! ~ ++ - sizeof	DS
3	multiplicativi	* / %	SD
4	aditivi	+ -	SD
5	deplasări	<< >>	SD
6	relaționali	< > <= >=	SD
7	egalitate / neegalitate	== !=	SD
8	ȘI pe biți	&	SD
9	SAU exclusiv pe biți	^	SD
10	SAU inclusiv pe biți		SD
11	ȘI logic	&&	SD
12	SAU logic		SD
13	condițional	?:	DS
14	atribuire	= op=	DS
15	virgula	,	SD



Expresii

O **expresie** este alcătuită din unul sau mai mulți operanzi legați între ei prin operatori. Operanzii pot fi constante, variabile sau funcții.

Operanzii reprezintă valorile care intră în calcul, iar operatorii desemnează operațiile care se execută în cadrul expresiei.

expresie = operatori + operanzi

Tipul unei expresii reprezintă tipul valorii expresiei.

Expresiile se împart în două categorii:

- expresii aritmetice;
- expresii logice.



a. Expresii aritmetice

- expresiile aritmetice sunt cele care efectuează operații aritmetice având ca rezultat un număr

Exemple

```
int x=7, y=2, r;
r=x/y; //r=3
```

```
float x=7, y=2, r;
r=x/y; //r=3.5
```

```
int a;
a=25/2*4-3+7/2; //a=48
```



b. Expresii logice

- o expresie logică descrie o condiție
- valoarea unei expresii logice reprezintă valoarea de adevăr a expresiei aferente
- o condiție poate fi **fa~~l~~să**/**fa~~l~~se** (valoarea 0) sau **adevărată**/**true** (o valoare diferită de 0)

Exemple

```
int x=7, y=2;
x>=y           //true
x!=y           //true
x<y            //false
```



9. Operații de citire și scriere

În limbajul C++ operațiile de introducere și extragere date se execută prin fluxurile de date.

Un **flux de date** (stream) reprezintă fluxul datelor de la sursă (de exemplu tastatură) la destinație (de exemplu ecranul monitorului).

Prin fluxurile de date echipamentele periferice de intrare-ieșire sunt conectate la programul C++.

Fluxuri de date standard

1. flux de date de intrare (***cin***);
2. flux de date de ieșire (***cout***).

Pentru operațiile de citire și scriere se folosesc instrucțiunile expresie prin care se creează fluxurile de date, cu ajutorul operatorilor **>>** și **<<**.



a. Flux de date de intrare (*cin*)

- conectează tastatura la program
- execută operații de citire
- datele de intrare sunt furnizate programului
- datele sunt păstrate în variabile de memorie
- *cin* reprezintă tastatura
- operatorul de intrare `>>` înseamnă transmiterea unei valori de la tastatură

Sintaxa:

```
cin>>nume_var;
```

sau

```
cin>>nume_var1>>nume_var2 >> ... >>nume_varn;
```



Exemplu:

```
int x=7,y=2,z=4;
```

x y z

```
cin>>x;
```

```
cin>>y;
```

```
cin>>z;
```

// considerăm că se introduc de la tastatură valorile 10, 20 și 30

x y z



2. Flux de date de ieșire (cout)

- conectează monitorul la program
- execută operații de scriere
- datele de ieșire sunt furnizate de program
- datele sunt transmise către monitor
- *cout* reprezintă monitorul
- operatorul de ieșire << înseamnă transmiterea unei valori către monitor

Sintaxa:

```
cout<<nume_var|constantă;
```

sau

```
cout<<nume_var1|constantă1<< nume_var2|constantă2<<  
... <<nume_varn|constantăn;
```



Exemplu:

```
int x=7,y=2,z=4;  
cout<<x;  
cout<<y;  
cout<<z;
```

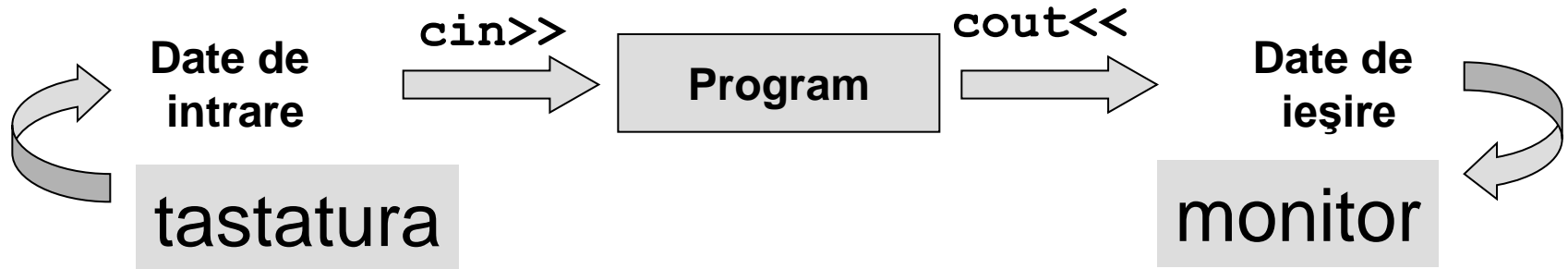
se va afișa: 724

iar pentru

```
cout<<x<<" ";  
cout<<10<<endl;  
cout<<z;
```

se va afișa: 7 10
4





Funcțiile de sistem `cin>>` și `cout<<` sunt definite în fișierele bibliotecii ale limbajului C++. Informațiile despre funcțiile de sistem (nume funcției, numărul și tipul parametrilor, tipul rezultatului funcției) se numesc **prototipul** funcției. Prototipurile funcțiilor de sistem se găsesc în fișierele antet (header).

Pentru a putea folosi în program funcțiile de sistem se scrie în program directiva pentru procesor:

```
#include<iostream>
```

10. Instrucțiunile limbajului C++

Pentru a genera rezultatele dorite, un program trebuie să acționeze asupra datelor într-un mod bine precizat. Descrierea acestor acțiuni se face cu ajutorul instrucțiunilor limbajului de programare.

Comenzile pe care programul le dă calculatorului, atunci când programul este rulat se numesc ***instrucțiuni***.



Instrucțiunile limbajului C++ sunt:

- instrucțiunea expresie;
- instrucțiunea compusă;
- instrucțiunea **if**;
- instrucțiunea **switch**;
- instrucțiunea **break**;
- instrucțiunea **while**;
- instrucțiunea **do while**;
- instrucțiunea **for**.



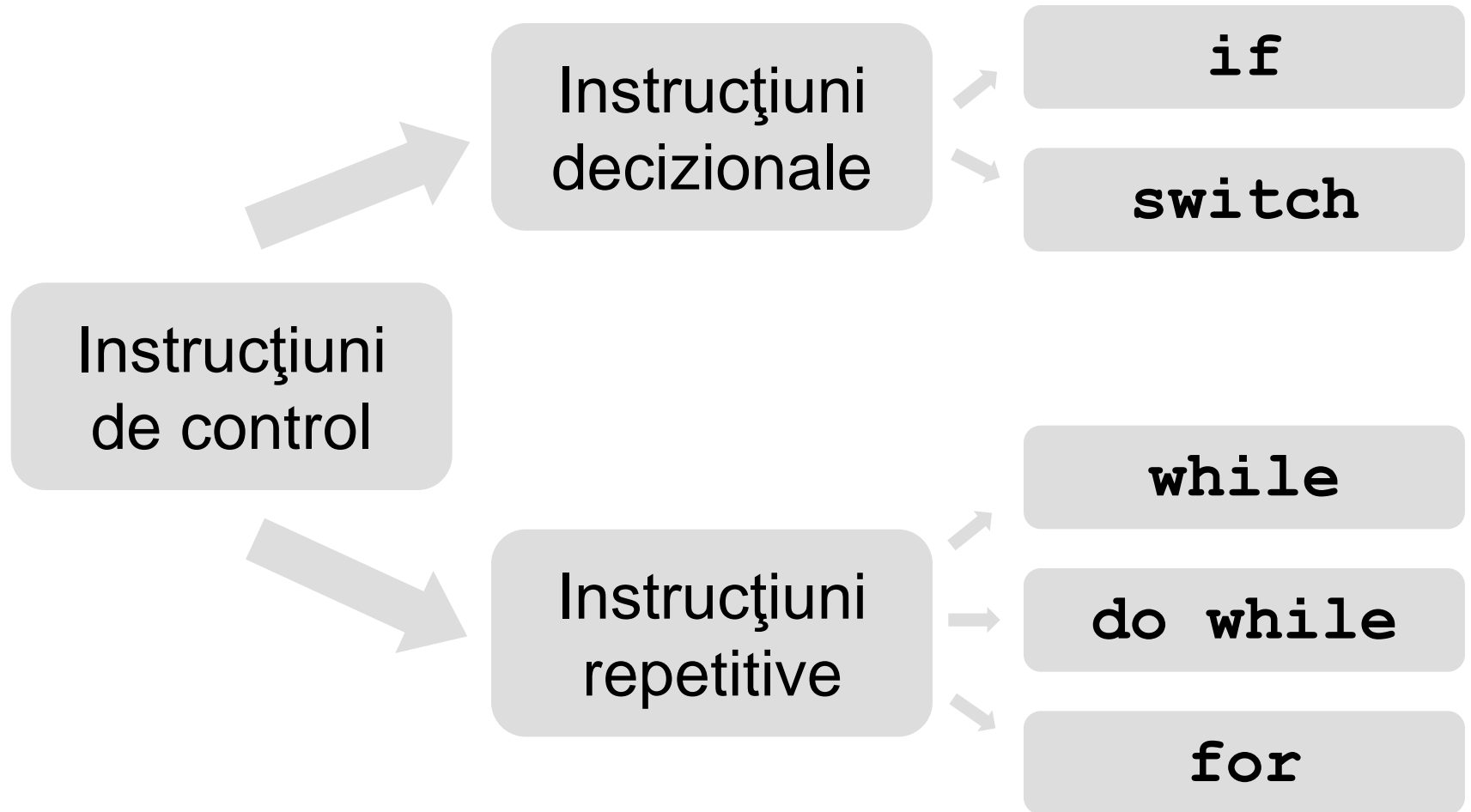
Instrucțiunile limbajului C++ se împart în două categorii:

- instrucțiuni simple;
- instrucțiuni de control (structurate).

Instrucțiunile simple nu conțin alte instrucțiuni (exp. instrucțiunea de atribuire).

Instrucțiunile de control specifică ordinea în care se execută instrucțiunile programului, controlând fluxul de execuție al programului.





a. Instrucțiunea expresie

Instrucțiunea expresie (de atribuire) este destinată atribuirii de valori variabilelor sau returnarea de valori în cazul funcțiilor.

Sintaxa:

expresie;

Efect:

- se evaluează expresia.

Se scrie caracterul “;” după o expresie (de atribuire, apelul unei funcții).



Exemple

```
s=a+5;
```

```
a=b=c=d=10;
```

```
i++;
```

```
p=abs (8) ;
```

```
clrscr () ;
```

```
p=1 ;
```

```
x+=2 ;
```

```
k=2*a-b*b+c ;
```



b. Instrucțiunea compusă

Reprezintă o succesiune de declarații urmate de instrucțiuni, incluse între acolade.

Sintaxa:

```
{  
    declarații;  
    instrucțiuni;  
}
```

Efect:

- se execută în ordine instrucțiunile specificate.



Exemplu

```
{  
    int a,x,p=5;  
    a=0;  
    x=p+2;  
    x++;  
    p+=x;  
    cout<<a<<x<<p;  
}
```



c. Instrucțiunea if

Instrucțiunea decizională (condițională) ***if*** realizează selectarea în vederea execuției a unei singure instrucțiuni din mai multe posibile.

Există două forme ale instrucțiunii decizionale ***if***.



Forma 1

Sintaxa:

```
if (expresie logică)
    instrucțiune1;
else
    instrucțiune2;
```

Efect:

- pasul 1: se evaluează **expresie logică**;
- pasul 2: dacă valoarea produsă de **expresie logică** este diferită de 0 (este adevărată) se execută **instrucțiune₁**, iar dacă valoarea produsă este 0 (este falsă) se execută **instrucțiune₂**.



Forma 2

Sintaxa:

```
if (expresie logică)  
    instrucțiune;
```

Efect:

- pasul 1: se evaluează **expresie logică**;
- pasul 2: dacă valoarea produsă de **expresie logică** este diferită de 0 (este adevărată) se execută **instrucțiune**.



Exemple

```
1.  if (n%2==0)
        cout<<"Numarul este par";
    else
        cout<<"Numarul este impar";

2.  if (x>10)
    {
        x++;
        y=10;
    }
    else
        x--;
    y=20;
```



```
3.  if (x>y)
        if (y>z)
                x=y+z;
        else
                x=y-z;
```

```
4.  if (x>y)
    {
        if (y>z)
            x=y+z;
    }
    else
        x=y-z;
```



d. Instrucțiunea switch

Instrucțiunea decizională ***switch*** realizează selectarea în vederea execuției a unei singure instrucțiuni din mai multe posibile.

Instrucțiunea ***switch*** este o generalizare a instrucțiunii decizionale ***if***, putând fi înlocuită cu instrucțiuni decizionale ***if*** imbricate.



Sintaxa:

```
switch (expresie logică)
{
    case c1: instrucțiune1;
            break;
    case c2: instrucțiune2;
            break;
    .....
    case cn: instrucțiunen;
            break;
    [default: instrucțiunen+1;]
}
```



Efect:

- pasul 1: se evaluează **expresie logică**
- pasul 2: dacă aceasta produce o valoare egală cu cea produsă de c_i , se execută **instrucțiune_i** și se încheie execuția instrucțiunii switch, altfel se execută **instrucțiune_{n+1}**.



Exemplu

```
a=b=5;
cout<<"1: adunare";
cout<<"2: scadere";
cout<<"Introdu optiune:";cin>>op;
switch (op)
{
    case 1: cout<<a+b;
            break;
    case 2: cout<<a-b;
            break;
    default: cout<<"optiune gresita";
}
}
```



e. Instrucțiunea break

Instrucțiunea ***break*** se folosește în instrucțiunea decizională ***switch*** sau în instrucțiunile repetitive.

Sintaxa:

```
break ;
```

Efect:

- determină ieșirea necondiționată din instrucțiunea în care apare (***switch, while, do while*** sau ***for***).



Exemplu

```
int i;  
cin>>i;  
switch(i)  
{  
    case 1:cout<<"am citit 1";  
           break;  
    case 2:cout<<"am citit 2";  
           break;  
    default:cout<<"am citit altceva";  
}
```



f. Instrucțiunea while

Instrucțiunea repetitivă ***while*** specifică faptul că anumite instrucțiuni se execută de mai multe ori.

Instrucțiunea ***while*** este o instrucțiune repetitivă:

- cu test inițial;
- cu număr necunoscut de pași.



Sintaxa:

```
while (expresie logică)  
    instrucțiune;
```

Efect:

- pasul 1: se evaluează **expresie logică**;
- pasul 2: dacă valoarea produsă de aceasta este adevărată (diferită de 0), se execută **instrucțiune**, apoi se trece la pasul 1, altfel (are valoarea 0) se trece la instrucțiunea următoare din program.



Exemple

```
1. int a=1;
   while (a<5)
   {
       a++;
       cout<<a<<endl;
   }
```

```
2. int a=1;
   while (a<5)
       a++;
   cout<<a<<endl;
```



```
3. int a=10;  
   while (a<5)  
       a++;  
   cout<<a<<endl;
```

```
4. int a=1;  
   while (a<5) ;  
       a++;  
   cout<<a<<endl;
```



g. Instrucțiunea do while

Instrucțiunea repetitivă ***do while*** specifică faptul că anumite instrucțiuni se execută de mai multe ori.

Instrucțiunea ***do while*** este o instrucțiune repetitivă:

- cu test final;
- cu număr necunoscut de pași.



Sintaxa:

```
do
    instrucțiune;
while (expresie logică);
```

Efect:

- pasul 1: se execută **instrucțiune**;
- pasul 2: se evaluează **expresie logică**; dacă valoarea produsă de aceasta este 0, execuția se încheie, altfel se trece la pasul 1.



Exemple

```
1. int a=1;
   do
   {
       a++;
       cout<<a<<endl;
   }while (a<5) ;
```

```
2. int a=10;
   do
   {
       a++;
       cout<<a<<endl;
   }while (a<5) ;
```



```
3. int a=10;  
   do  
   {  
   }while (a<5) ;  
   cout<<a;
```

```
4. int a=10;  
   do  
   ;  
   while (a<5) ;  
   cout<<a<<endl;
```



h. Instrucțiunea for

Instrucțiunea repetitivă *for* specifică faptul că anumite instrucțiuni se execută de mai multe ori.

Instrucțiunea *for* este o instrucțiune repetitivă:

- cu număr cunoscut de pași.



Sintaxa:

```
for (expresie1; expresie2; expresie3)  
    instrucțiune;
```

Efect:

- pasul 1: se evaluează **expresie₁**;
- pasul 2: se evaluează **expresie₂**; dacă aceasta produce o valoare diferită de 0, se execută **instrucțiune**, apoi se trece la pasul 3, altfel instrucțiunea **for** se încheie;
- pasul 3: se evaluează **expresie₃** și se revine la pasul 2.



Exemple

```
1.   int i;  
     for (i=1;i<5;i++)  
         cout<<i<<" ";
```

```
2.   int i;  
     for (i=5;i<5;i++)  
         cout<<i<<" ";
```

```
3.   int i;  
     for (i=5;i>1;i--)  
         cout<<i<<" ";
```

```
4.   int i;  
     for (i=5;i>1;i=i-2)  
         cout<<i<<" ";
```



Fișe de lucru

- Operatori și tipuri de date
- Aplicații instrucțiunea de atribuire
- Aplicații instrucțiunea decizională ***if***
- Aplicații instrucțiunea repetitivă ***while***
- Aplicații instrucțiunea repetitivă ***do while***
- Aplicații instrucțiunea repetitivă ***for***



12. Bibliografie și webografie

1. Miloșescu M., *Informatică. Manual pentru clasa a IX-a*, Editura Didactică și Pedagogică, București, 2004
2. Munteanu F., *Programarea calculatoarelor. Manual pentru licee de informatică clasele X-XII*, Editura Didactică și Pedagogică, București, 1994
3. Popescu C., *Culegere de probleme de informatică*, Editura Donaris-Info, Sibiu, 2002
4. Ministerul Educației, Cercetării și Tineretului, Centrul Național pentru Curriculum și Evaluare în Învățământul Preuniversitar, *Proba scrisă la informatică. Examenul de bacalaureat – Variante (1-100)*, București 2008
5. <http://www.cplusplus.com/>
6. <https://www.w3schools.com/cpp/>

