

Introducere în programare 2016 - 2017

3

Bogdan Pătruț

bogdan@info.uaic.ro

după Corina Forăscu

<http://profs.info.uaic.ro/~bogdan/ip/>

Curs 3: conținut

- Operatori
- Conversii
- Intrare - ieșire
- Instrucțiuni
- Funcții

Utilizare **typedef** (1)

- Mecanism prin care se asociază un tip unui identificador:

```
typedef char litera_mare;  
typedef short varsta;  
typedef int vector[20];  
typedef char string[30];  
typedef float matrice[10][10];  
typedef struct { double re, im; } complex;
```

Utilizare **typedef** (2)

- Identificatorul respectiv se poate utiliza pentru a declara variabile:

```
litera_mare u, v='a';  
varsta v1, v2;  
vector x; string s;  
matrice a;  
complex z;
```

- sau funcții:

```
complex suma(complex z1, complex z2) {  
    complex z;  
    z.re=z1.re+z2.re; z.im=z1.im+z2.im;  
    return z;  
}
```

Operatorul condițional ?:

exp1 ? exp2 : exp3

- Se evaluează *exp1*
- Dacă *exp1* are valoare true (nenulă), atunci valoarea expresiei este valoarea lui *exp2*; *exp3* nu se evaluează
- Dacă *exp1* are valoare false (nulă), atunci valoarea expresiei este valoarea lui *exp3*; *exp2* nu se evaluează
- Operatorul ?: este drept asociativ

Operatorul condițional ?: Example

```
x >= 0 ? x : y
```

```
x > y ? x : y
```

```
x > y ? x > z ? x : z : y > z ? y : z
```

```
joc=(raspuns=='1')?JocSimplu();JocDublu();
```

```
#include <iostream>
using namespace std;
```

```
void main(){
    int a=1, b=2, c=3;
    int x, y, z;
    x = a?b:c?a:b;
    y = (a?b:c)?a:b; /* asociere stanga */
    z = a?b:(c?a:b); /* asociere dreapta */
    cout<< "x=" << x << "\ny=" << y << "\nz=" << z;
}
/* x=2 y=1 z=2 */
```

Operatorul virgulă ,

expresia_virgula ::= expresie, expresie

- Se evaluează prima expresie apoi cea de-a doua.
- Valoarea și tipul întregii expresii este valoarea și tipul operandului drept.
- Operatorul virgulă are cea mai mică precedență.

```
a = 1, b = 2;  
i = 1, j = 2, ++k + 1;  
k != 1, ++x * 2.0 + 1;  
for(suma = 0, i = 1; i <= n; suma += i, ++i);
```

Operatorul **sizeof()**

- Operator unar ce permite găsirea numărului de octeți pe care se reprezintă un obiect (tip, expresie)

```
sizeof(int),          sizeof(double);  
sizeof(b*b-4*a*c),  sizeof(i);
```

sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)

sizeof(signed) = sizeof(unsigned) = sizeof(int)

sizeof(float) <= sizeof(double) <= sizeof(long double)

Operatorul `sizeof()`

```
#include <iostream>
using namespace std;

void main(){
    int x = 1; double y = 9; long z = 0;
    cout << "Operatorul sizeof()\n\n\n";
    cout << "sizeof(char) = " << sizeof(char) << endl;
    cout << "sizeof(int) = " << sizeof(int) << endl;
    cout << "sizeof(short) = " << sizeof(short) << endl;
    cout << "sizeof(long) = " << sizeof(long) << endl;
    cout << "sizeof(float) = " << sizeof(float) << endl;
    cout << "sizeof(double) = " << sizeof(double) << endl;
    cout << "sizeof(long double) = " << sizeof(long double) << endl;
    cout << "sizeof(x + y + z) = " << sizeof(x+y+z) << endl;
}
```

Operatorul sizeof()

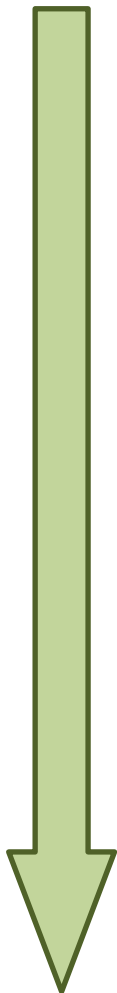
Rezultatul executiei Visual C++:

Operatorul sizeof()

```
Operatorul sizeof()
sizeof(char) = 1
sizeof(int) = 4
sizeof(short) = 2
sizeof(long) = 4
sizeof(float) = 4
sizeof(double) = 8
sizeof(long double) = 8
sizeof(x + y + z) = 8
Press any key to continue . . .
```

Operatori

Nr.	Clasa de operatori	Operatori	Asociativitate
1.	Primari	() [] . -> ::	de la stânga la dreapta
2.	Unari	! ~ ++ -- sizeof(tip) -(unar) *(deferentiere) &(referentiere)	de la stânga la dreapta
3.	Multiplicativi	* / %	de la stânga la dreapta
4.	Aditivi	+ -	de la stânga la dreapta
5.	Deplasare pe bit	<< >>	de la stânga la dreapta
6.	Relaționali	< <= > >=	de la stânga la dreapta
7.	De egalitate	== !=	de la stânga la dreapta
8.		& (SI logic pe bit)	de la stânga la dreapta
9.		^ (XOR pe bit)	de la stânga la dreapta
10.		(SAU logic pe bit)	de la stânga la dreapta
11.		&&	de la stânga la dreapta
12.			de la stânga la dreapta
13.	Condițional	?:	de la dreapta la stânga
14.	De atribuire	= += -= *= %= &= ^= = <<= >>=	de la dreapta la stânga
15.	Virgula	,	de la stânga la dreapta



Reguli pentru conversia implicită

- În absența unui **unsigned** , obiectele se convertesc la tipul cel mai “înalt” în ordinea (descrescătoare):
long double, double, float, long int, int
- Regulile pentru operanzii **unsigned** depind de implementare.
- Conversia la **unsigned** se face doar în caz de necesitate (de ex. valoarea din **unsigned** nu “încapă” în celălalt operand).

Reguli pentru conversia implicită

- Regula “*integer promotion*” : operațiile se fac cel puțin în **int**, deci **char** și **short** sunt “promovați” la **int**.
- La o asignare ($v = \text{exp}$) tipul membrului drept se convertește la tipul membrului stâng (care este și tipul rezultatului).
 - ATENȚIE! Se pot produce:
 - Pierderea preciziei (**double** ->**float** ->**long int**)
 - Pierderea unor biți semnificativi (**long** ->**int**)
 - Nedeterminări

Exemplu

```
#include <iostream>
using namespace std;

int main(void){
    char c1 = -126, c2;          /* c1 = 10000010    */
    unsigned char c3, c4 = 255; /* c4 = 11111111    */
    short s1, s2 = -32767; /* s2=10000000 00000001 */
    short s3 = -1, s4;        /* s3 = 11111111 11111111 */
    s1 = c1;
    cout << "c1=" << (int)c1 << " s1=" << s1 << endl;
    c2 = s2;
    cout << "c2=" << (int)c2 << " s2=" << s2 << endl;
    c3 = s3;
    cout << "c3=" << (int)c3 << " s3=" << s3 << endl;
    s4 = c4;
    cout << "c4=" << (int)c4 << " s4=" << s4 << endl;
    return 0;
}
```

Exemplu (rezultatul execuției)

c1 = 10000010 s1 = c1;
c1 = -126, s1 = -126

s2=10000000 00000001 c2 = s2;
c2 = 1, s2 = -32767

s3 = 11111111 11111111 c3 = s3;
c3 = 255, s3 = -1

c4 = 11111111 s4 = c4;
c4 = 255, s4 = 255

```
char c1 = -126, c2;  
unsigned char c3, c4 =  
255;  
short s1, s2 = -32767;  
short s3 = -1, s4;
```

Forțarea tipului - **cast**

- Conversia explicită la tipul *numetip*:

(*numetip*) *expresie*

- Exemple:

```
(long)('A' + 1.0)
(int)(b*b-4*a*c)
(double)(x+y)/z
(float)x*y/z
x / (float)2
```


Exemplu cast

```
#include <iostream>
using namespace std;

int main(void){
    int i, j; double x, y, z, t;
    i = 5/2; x = 5/2;
    y = (double)(5/2); j = (double)5/2;
    z = (double)5/2; t = 5./2;
    cout << i << ", " << x << ", ";
    cout << y << ", " << j << ", ";
    cout << z << ", " << t << ", " << endl;
    return 0;
}

/* 2, 2, 2, 2, 2.5, 2.5 */
```

Fișiere în bibliotecă relative la tipuri

- **<limits.h>** - pentru tipurile întregi
 - Întregul min/max: `INT_MIN`, `INT_MAX`
 - Numărul de biți pe caracter `CHAR_BIT`
 - Etc.
- **<float.h>** - pentru tipurile flotante:
 - Exponentul maxim
 - Precizia zecimală, etc.
- **<stdlib.h>** - conține funcții de conversie:
 - Șir de caractere în **int**: **`atoi(const char*)`**
 - Șir de caractere în **float**: **`atof(const char*)`**

Intrări / ieșiri

```
cin >> var;    /* citește var de la cin */
```

- Se pot prelua tipurile aritmetice, șiruri de caractere

```
cout << expr;   /* scrie expr la cout */
```

- Se pot transfera tipurile aritmetice, șiruri de caractere, pointeri de orice tip în afară de char

- Sunt posibile operații multiple, de tipul:

```
cin >> var1 >> var2 ... >> varN;
```

```
cout << var1 << var2 ... << varN;
```

Intrări / ieșiri

`cin >> variabile`

`cout << expresii`

- citirea unui int

```
std::cout << "introduceți valoarea lui a: ";  
std::cin >> a;
```

- afișarea unui int

```
int a = 10;  
std::cout << a;
```

Intrări / ieșiri: exemplu - caractere

```
/* Exemple de constante caracter */  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    char a, b, c, d;  
    a = 'A'; b = 65; c = '\101'; d = '\x41';  
    cout << a << b << c << d << endl;  
    cout << a << (int)a;  
    cout << oct << (int)a << hex << (int)a;  
    return 0;  
}
```

A A A A

A 65 101 41

Intrări / ieșiri: exemplu – coduri ASCII

```
#include <iostream>
using namespace std;

int main ()
{
    short c;
    for(c = 0; c <= 127; c++){ // for(c='a'; c<='z'; c++)
        cout << "cod ASCII: " << (int)c;
        cout << ", character: ";
        cout << (char)c << endl;
    }
    return 0;
}
```

Instrucțiuni

- Expresii: `; expresie;`
- Compuse (bloc): `{declarații instrucțiuni}`
- Condiționale: `if if-else switch-case`
- Iterative: `for while do-while`
- Întreruperea secvenței:
`continue; break; return expr;`
- Salt necondiționat: `goto`

Instrucțiunea expresie

instr_expresie ::= {*expresie*}_{opt} ;

- Example:

```
a = b;  
a + b + c;  
;  
cout << a;  
sizeof(int);
```


Instrucțiunea compusă (bloc)

instr_compusa ::=

{ {lista_declaratii | lista_instructiuni}₀₊ }

- Grupează instrucțiuni într-o unitate executabilă.
- O instrucțiune compusă este ea însăși o instrucțiune: oriunde poate să apară o instrucțiune, este corect să apară și o instrucțiune compusă.

Instrucțiunea compusă - Example

```
{  
    int a=3, b=10, c=7;  
    a += b += c;  
    cout << a << ", " << b << ", " << c; // ?, ?, ?  
}
```

```
if (x > y){  
    int temp;  
    temp = x; x = y; y = temp;  
    cout << x << y;  
}
```

```
{  
    int a, b, c;  
    {  
        b = 2; c = 3; a = b += c;  
    }  
    cout << "a= " << a << endl;  
} // ?
```

Instrucțiunile condiționale

if și if-else

instr_if ::= if (<boolean_expression>)
 {<yes_statement>;}

instr_if-else ::= if (<boolean_expression>)
 {<yes_statement>;}
 else
 {<no_statement>;}

boolean_expression este construită cu:

- Expresii aritmetice
- Comparatori: ==, !=, <, <=, >, >=
- Conectori logici: &&, ||, !

Exemple **if** și **if-else**

```
if(b == a)
    aria = a*a;
```

```
if(a%2) if(b%2) p = 1; else p = 2;
```

```
if(x < y)
    min = x;
else
    min = y;
```

```
if(a%2){
    if(b%2)
        p = 1;
}
else p = 2;
```

Exemple `if` și `if-else`

```
int i, j, k, l, max;  
if(i>j)  
    if(k>l)  
        if(i>k) max = i;  
        else max = k;  
    else  
        if(i>l) max = i;  
        else max = l;  
else  
    if(k>l)  
        if(j>k) max = j;  
        else max = k;  
    else  
        if(j>l) max = j;  
        else max = l;
```

“Dangling else Problem”

```
int a=1, b=2; // b=3
if (a == 1)
    if (b == 2) // b=2
        cout << "*****\n";
else
    cout << "ooooo\n";
```

- Nu lăsați forma codului să vă ducă în eroare!
- Atenție la diferența dintre operatorii de egalitate și cel de asignare
- Regula este: **else** este atașat celui mai apropiat **if**.

If-else multiplu

```
if ( conditie-1 ) {  
    instructiuni-1;  
}  
else if ( conditie-2 ) {  
    instructiuni-2;  
    ...  
}  
  
else if ( conditie-n ) {  
    instructiuni-n;  
}  
else {  
    instructiuni-pt-restul-posibilitatilor;  
}
```

If-else. Exemplu

```
int main(void){
    float x, y, rezultat;
    char op;
    cout << "Expresia:(numar operator numar - FARA SPATII)\n";
    cin >> x >> op >> y;
    if(op == '+')
        rezultat = x+y;
    else if(op == '-')
        rezultat = x-y;
    else if(op == '*')
        rezultat = x*y;
    else if(op == '/')
        rezultat = x/y;
    else{
        cout << "Eroare in scrierea expresiei!";
        return 1;
    }
    cout << "Rezultatul este: " << rezultat << "\n";
    return 0;
}
```


Instrucțiunea **switch**

```
switch (expression)
{
    case constant1:                !! case n:
        group of statements 1;    !! case (1..3):
        break;
    case constant2:
        group of statements 2;
        break;
    .
    .
    .
    default:
        default group of statements
}
```

Instrucțiunea **switch**

```
switch (x) {  
    case 1:  
        cout << "x is 1";  
        break;  
    case 2:  
        cout << "x is 2";  
        break;  
    default:  
        cout << "value of x unknown";  
}
```

```
if (x == 1)  
{  
    cout << "x is 1";  
}  
else if (x == 2) {  
    cout << "x is 2";  
}  
else {  
    cout << "value of x unknown";  
}
```

Instrucțiunea **switch**

- Valoarea expresiei `expression`, care este de tip `int`, se compară cu constantele `constanti`.
- În caz de egalitate se execută instrucțiunea corespunzătoare și **toate cele ce urmează**. Există posibilitatea de ieșire cu instrucțiunea **break**.
- Dacă valoarea determinată diferă de oricare din constantele specificate, se execută instrucțiunea specificată la **default**, care apare o singură dată, nu neaparat la sfârșit. Dacă **default** lipsește se iese din **switch**.
- Valorile constantelor trebuie să fie diferite; ordinea lor nu are importanță.
- Acoladele ce grupează mulțimea **case**-urilor sunt obligatorii. După fiecare **case** pot apărea mai multe instrucțiuni fără a fi grupate în acolade.

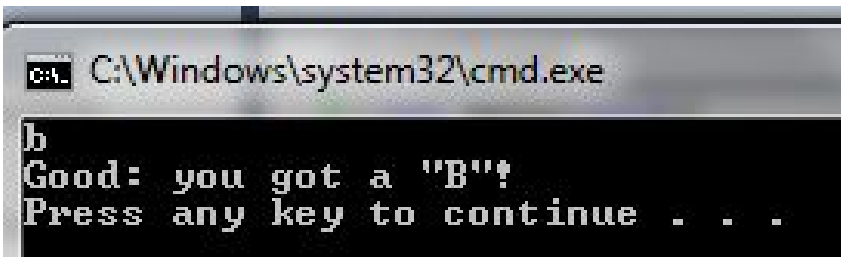
Instrucțiunea **switch**. Example

```
int i;  
cin >> i;  
switch(i){  
case 1: cout << " 1";  
case 2: cout << " 2";  
case 3: cout << " 3"; // break;// ???  
case 4: cout << " 4";  
default: cout << " blabla! ";  
  
//3
```

```
3  
3 4 blabla! Press any key to continue . . .
```

Instrucțiunea **switch**. Exemple

```
char eval;  
cin >> eval;  
switch (eval) {  
    case 'A':  
    case 'a':  
        cout << "Excellent: you got an \"A\"!\n";  
        break;  
    case 'B':  
    case 'b':  
        cout << "Good: you got a \"B\"!\n";  
        break;  
}
```



```
cmd.exe C:\Windows\system32\cmd.exe  
b  
Good: you got a "B"?  
Press any key to continue . . .
```

Instrucțiunea **while**

instrucțiunea_while ::=

while(*expresie*) *instrucțiune*

while (*expresie*){

instrucțiune

}

instrucțiunea_urmatoare

- Se evaluează *expresie*: dacă valoarea sa este nenulă se execută *instrucțiune* și controlul este transferat înapoi, la începutul instrucțiunii *while*; dacă valoarea este nulă se execută *instrucțiunea_urmatoare*.
- Așadar *instrucțiune* se execută de zero sau mai multe ori.

Instrucțiunea **while**. Exemplu

```
int n, i=1, factorial=1;  
cin>>n;  
while (i++ < n)  
    factorial *= i;  
cout<<factorial;
```

Instrucțiunea **do..while**

```
instrucțiunea_do..while ::=  
    do instrucțiune while(conditie);  
  
    do{  
        instrucțiune  
    } while (conditie);  
    instrucțiunea_urmatoare
```

- Se execută *instrucțiune*.
- Se evaluează *conditie*: dacă valoarea sa este nenulă controlul este transferat înapoi, la începutul instrucțiunii **do..while**; dacă valoarea este nulă se execută *instrucțiunea_urmatoare*.
- Așadar *instrucțiune* se execută o dată sau de mai multe ori.

Instrucțiunea **do..while**. Exemplu

```
unsigned long n;  
do {  
    cout << "Enter number (0 to end): ";  
    cin >> n;  
    cout << "You entered: " << n << "\n";  
} while (n != 0);  
return 0;
```

cmd C:\Windows\system32\cmd.exe

Enter number (0 to end): 25

You entered: 25

Enter number (0 to end): 36

You entered: 36

Enter number (0 to end): 0

You entered: 0

Press any key to continue . . .

Exemplu - calculator

```
int main(void){
    float x, y, rezultat;
    char op, c;
    int ERROR;
    cout << "Calculator pentru expresii de forma \n numar operator numar\n";
    cout << "Folositi operatorii + - * / \n";

    do{
        ERROR = 0;
        cout << "Expresia: ";
        cin >> x >> op >> y;
        switch(op){
            case '+': rezultat = x+y; break;
            case '-': rezultat = x-y; break;
            case '*': rezultat = x*y; break;
            case '/': if(y != 0) rezultat = x/y;
                     else {cout << "Impartire prin zero!\n"; ERROR = 1;}
                     break;
            default : {cout << "Operator necunoscut!\n"; ERROR = 1;}
        }
        if(!ERROR)
            cout << x << " " << op << " " << y << " = " << rezultat;
        cin.sync();
        do{cout << "\n Continuati (d/n)?"; c = getchar();
        } while (c != 'd' && c != 'n');
    } while (c != 'n');
    cout << "See you later!\n";
    return 0;
}
```

Instrucțiunea **for**

instrucțiunea_for ::=

for (*expr-initial*; *expr-cond*; *expr-in/decrementare*)
instrucțiune;

for (*expr1*; *expr2*; *expr3*){
 instrucțiuni;
}
instrucțiunea_urmatoare

- Una, doua sau toate trei dintre expresii pot lipsi, dar cei doi separatori (;) sunt obligatorii.

Instrucțiunea for

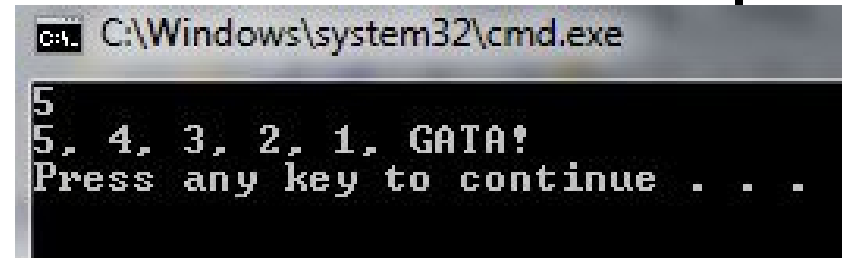
- Dacă *instructiune* nu conține **continue** și *expr-cond* este prezentă, atunci **for** este echivalent cu:
expr-init;
 while (*expr-cond*) {
 instructiune;
 expr-in/decrementare;
 }
 instructiunea_urmatoare
- Dacă există **continue** atunci aceasta transferă controlul la *expr-in/decrementare*.

Instrucțiunea for

- Se evaluează *expr-init* - în general aceasta se utilizează pentru **inițializarea iterației**.
- Se evaluează *expr-cond* - în general aceasta este o expresie logică ce se utilizează pentru **controlul iterației**. Dacă valoarea sa este nenulă(true), se execută corpul buclei do (instrucțiune), se evaluează *expr-in/decrementare* și controlul este trecut la începutul buclei do, fără a se mai evalua *expr-init*.
- În general *expr-in/decrementare* face trecerea la **iterația următoare**: modifică o variabilă ce intră în componența lui *expr-cond*.
- Procesul continuă până când valoarea *expr-cond* este nulă (false). Controlul este transferat următoarei instrucțiuni (cea de după for).

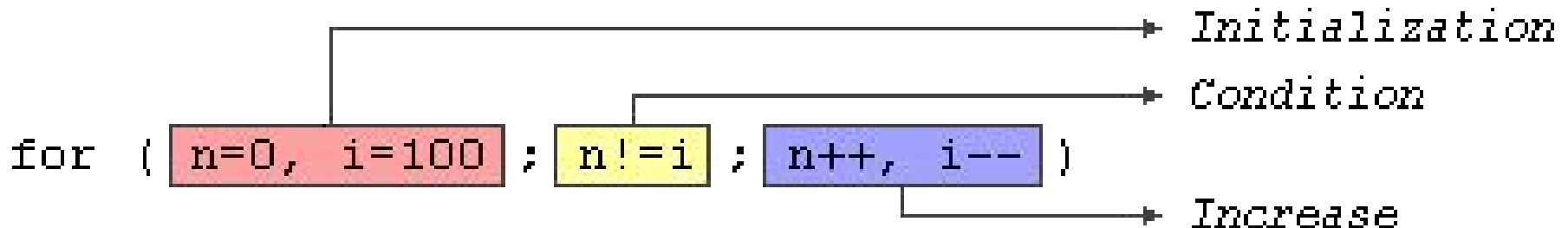
Instrucțiunea **for**. Example

```
int n;  
cin>>n;  
    for (; n>0; n--) {  
        cout << n << ", ";  
    }  
    cout << "GATA!\n";
```



C:\Windows\system32\cmd.exe

```
5  
5, 4, 3, 2, 1, GATA!  
Press any key to continue . . .
```



Instrucțiunea **for**. Exemple

```
i = 1;  
suma = 0;  
for(;i <= N;++i) suma += i;
```

```
i = 1;  
suma = 0;  
for(;i <= N;) suma += i++;
```

```
i = 1;  
suma = 0;  
for(;;) suma += i++; // Bucla infinita
```

Instrucțiuni de întrerupere a secvenței

- **break;**
 - se referă la bucla sau instrucțiunea `switch` cea mai apropiată.
 - produce ieșirea din buclă sau din `switch` și trece controlul la instrucțiunea următoare
- **continue;**
 - se referă la bucla (`for`, `while`, `do...while`) cea mai apropiată.
 - întrerupe execuția iterației curente și trece controlul la iterația următoare.
- **goto;** Edsger Dijkstra (1964) GOTO considered harmful
 - Permite saltul la o anumită secțiune din program, identificată prin punctul de începere.
- **return *expr*;** *sau* **return;**
 - în funcții, întrerupe execuția și transferă controlul apelantului, eventual cu transmiterea valorii expresiei *expr*.

Exemple: instrucțiuni de întrerupere a secvenței

```
int n;  
for (n=10; n>0; n--)  
{  
    cout << n << ", ";  
    if (n==3)  
    {  
        cout << "countdown aborted!";  
        break;  
    }  
}
```

```
for (int n=10; n>0; n--) {  
    if (n==5) continue;  
    cout << n << ", ";  
}  
cout << "FIRE!\n";
```

```
int n=10;  
loop:  
    cout << n << ", ";  
    n--;  
    if (n>0) goto loop;  
    cout << "FIRE!\n";  
    return 0;
```

Exemplu – for..continue

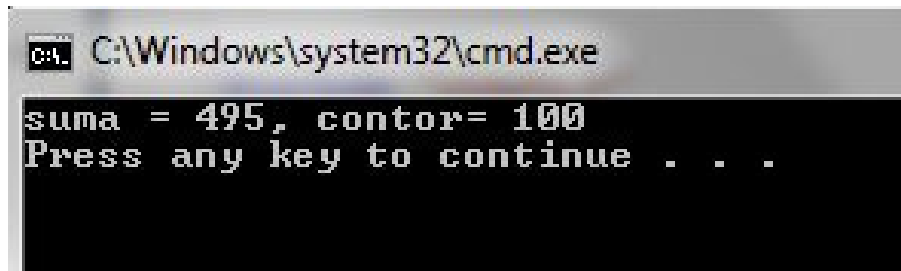
```
int i, suma=0;
    for(i = 1; i<=N; i++){
if(i%3 != 0) continue;
suma+=i;
    }
    cout << "suma = " << suma; /* suma multiplilor de 3 până la N */
```

Instrucțiuni de iterație - recomandare

- Pentru expresia de control a iterației se recomandă operatorii relaționali în locul celor de (ne)egalitate:

```
int main(){
int contor = 0;
double suma = 0.0, x;
for(x = 0.0; x != 9.9; x += 0.1){
    suma += x;
    ++contor;
}
cout << "suma = " << suma << ", contor= " << contor << "\n";
return 0;
}
```

- Folositi **$x < 9.9$** în loc de **$x \neq 9.9$**



A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\system32\cmd.exe'. The command prompt displays the output of the program: 'suma = 495, contor= 100' followed by 'Press any key to continue . . .'. The text is in a monospaced font on a black background.

Funcții

```
int suma(int n)
{
    int s = 0;
    int i;
    for(i=1; i<=n; ++i)
        s += i;
    return s;
}
```

Funcții

- Definiția unei funcții:

tip_returnat *nume_funcție*(*tip1* *var1*,...){

lista_de_declarații

lista_de_instrucțiuni

Antetul funcției

Corpul funcției

Parametrii funcției
(parametri formali
sau generici)

Funcții

- Apelul unei funcții:

nume_funcție(*expr1*,...)



Argumente

- Argumentele sunt expresii ce substituie parametrii la un apel: parametrii funcției sunt inițializați cu valorile argumentelor.

Funcții. Exemplu

```
void swap(int x, int y){  
    int temp = x; x = y; y = temp;  
    cout << "x=" << x<< ",y=" << y<< "\n";  
}
```

```
int main(void){  
    int a = 2, b = 3;  
    swap(a, b);    // x = 3, y = 2  
    cout << "a=" << a<< ",b=" << b << "\n";  
    // a = 2, b = 3
```

Funcții. Exemplu

```
void swap(int x, int y){  
    int temp = x; x = y; y = temp;  
    cout << "x=" << x<< ",y=" << y<< "\n";  
}
```

```
int main(void){  
    int a = 2, b = 3;  
    swap(a, b);    // x = 3, y = 2  
    cout << "a=" << a<< ",b=" << b << "\n";  
    // a = 2, b = 3
```


Transmiterea parametrilor (1)

```
#include <iostream>
```

```
int x=10,t;
```

variabile globale, vizibile și în funcția **egale**

```
char egale(int x, int y)
```

```
{ int a=1; t=x+y+a;
```

```
return x==y?'D':'N'; }
```

variabilă
locală

parametri formali ai
funcției *egale*, vizibili doar
în funcția *egale*

```
void main() {
```

```
int y=2,z=3; t=1;
```

```
cout<<t<<x<<y<<z;
```

variabile globale invizibile în funcția *egale*

```
if (egale(y,z))== 'D' cout<<"Da"; else cout<<"NU";
```

```
cout<<t;
```

```
} // 1,10,2,3,Nu (deoarece y->x, adica 2->x, z->y, adica z->y și  
2!=3, apoi t=6
```

Transmiterea parametrilor (2)

```
#include <iostream>
```

```
void F(int x, int y)
```

parametri formali **transmiși prin valoare**

```
{
```

```
    x=2; y=3; cout<<x<<y;
```

```
}
```

```
void main()
```

```
{
```

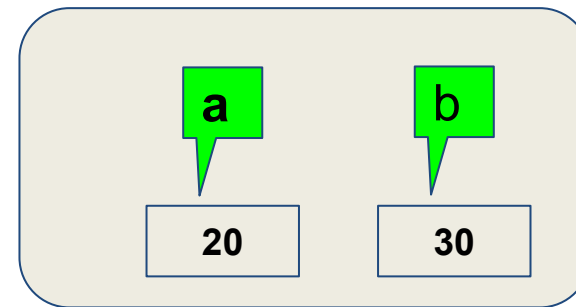
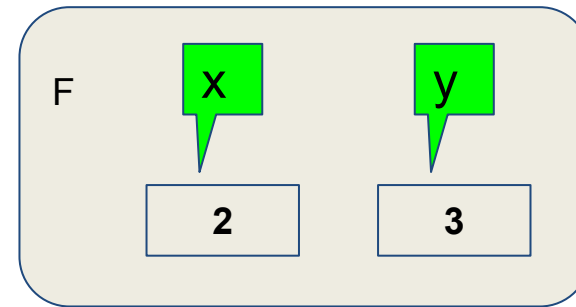
```
    int a=20;
```

```
    int b=30;
```

```
    F(a,b);
```

```
    cout<<a<<b;    // 2, 3, 20, 30
```

```
}
```



Transmiterea parametrilor (3)

```
#include <iostream>
```

```
void F(int& x, int& y)
```

parametri formali **transmiși prin referință**

```
{
```

```
    x=2; y=3; cout<<x<<y;
```

```
}
```

```
void main()
```

```
{
```

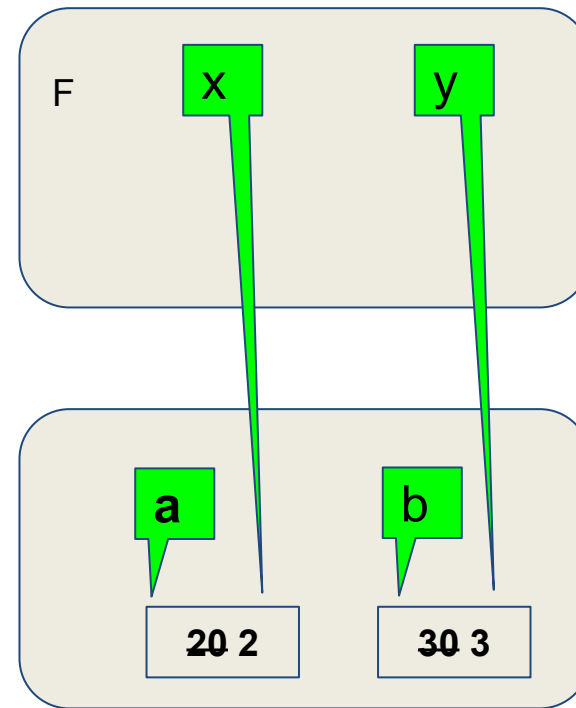
```
    int a=20;
```

```
    int b=30;
```

```
    F(a,b);
```

```
    cout<<a<<b;    // 2, 3, 2, 3
```

```
}
```



Transmiterea parametrilor (4)

```
#include <iostream>
```

parametrii formali **transmiși prin valoare**
sunt folosiți pentru *input*

```
void suma(int x, int y, int& suma)
```

```
{
```

```
    x=2; y=3; suma=x+y;
```

```
}
```

parametrii formali **transmiși prin referință**
sunt folosiți pentru *outpur*

```
void main()
```

```
{
```

```
    int a=20; int b=30; int c;
```

```
    suma(2,3,c); cout<<c;
```

```
    suma(2,b,c); cout<<c;
```

```
    suma(2+a,b*3,c); cout<<c;
```

```
}
```

modalități de apelare a funcției:
suma(*expresie*, *expresie*, *variabila*)
nu putem apela **suma(a,b,5)** sau
suma(a,b,5+c)

Transmiterea parametrilor (5)

```
#include <iostream>
```

parametrii formali **transmiși prin valoare**
sunt folosiți pentru *input*

```
int suma(int x, int y)
```

```
{ int suma;
```

```
  x=2; y=3; suma=x+y;
```

```
  return suma;
```

parametrii formali **transmiși prin referință**
sunt folosiți pentru *outpur*

```
}
```

```
void main() {
```

```
  int a=20; int b=30; int c;
```

```
  cout<<suma(2,3);
```

```
  cout<<suma(2+b,a);
```

modalități de apelare a funcției **suma** în
care valoarea returnată prin tipul funcției

```
  c=suma(2+a,b-1); cout<<c;
```

```
}
```

Exemplu - numere complexe

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
typedef struct {
    double re, im;
} complex;
void scrie(complex z, char nume)
{
    printf("Numarul complex %c este: (%7.3lf,%7.3lf)\n",
        nume,z.re,z.im);
}
void citeste(complex *pz, char nume)
{
    double a, b;
    printf("Dati %c.re: ",nume); scanf("%lf",&a);
    printf("Dati %c.im: ",nume); scanf("%lf",&b);
    /* doua modalitati de a referi campurile variabilei dinamice
    indicate de pz */
    (*pz).re=a; pz->im=b;
}
void citestel(complex & z, char nume)
{
    double a,b;
    printf("Dati %c.re: ",nume); scanf("%lf",&a);
    printf("Dati %c.im: ",nume); scanf("%lf",&b);
    z.re=a; z.im=b;
}
double modul(complex z)
{
    return sqrt(z.re*z.re+z.im*z.im);
}
complex suma(complex z1, complex z2)
{

```