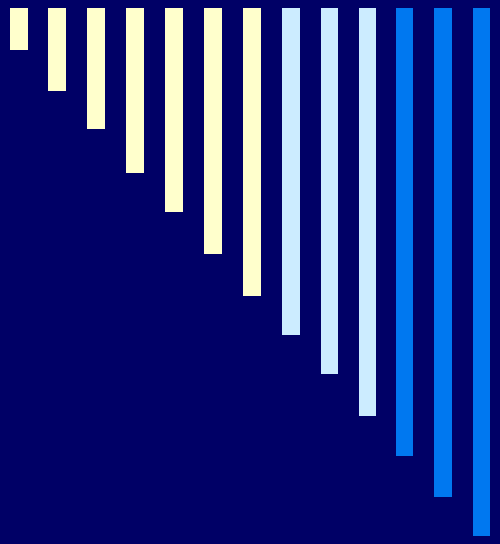
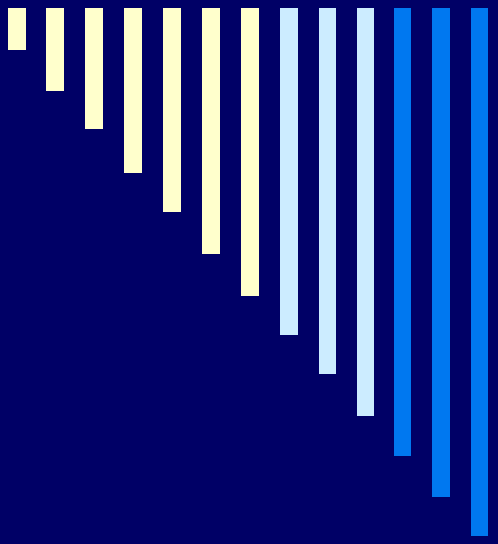


Subprograme

Modul de transmitere a parametrilor

Profesor: Sanda Popescu

Cum se rezolvă problemele dintr-o firmă într-o lume tehnologică?



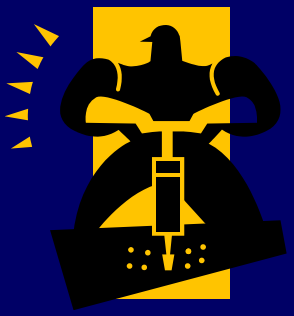
Cum poate rezolva directorul (managerul) toate problemele?

Singur...

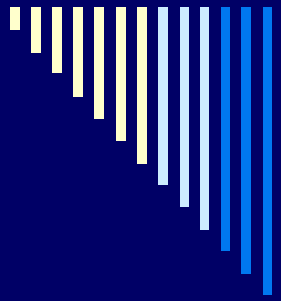
...sau poate trasa diferite sarcini subalternilor.

Aceştia le pot rezolva singuri...

... sau pot apela la subalternii lor...



...pentru a rezolva probleme mai mici



Noțiuni teoretice referitoare la subprograme în C++

Teminologie folosită

- **Definiție:** Un subprogram este o secvență de instrucțiuni care rezolvă o anumită sarcină în afara funcției main() și lansată în execuție de câte ori este nevoie
- **Modul apelant:** Modulul/subprogramul care apelează la alte subprograme pentru rezolvarea unei probleme
- **Modul apelat:** Modulul/subprogramul care este apelat de un alt modul, pentru a-i rezolva acestuia o problemă

Terminologie folosită

- **Variabile locale:** variabile declarate în corpul unui subprogram; sunt păstrate temporar pe stiva STACK de memorie, doar atât cât se execută subprogramul, adică sunt vizibile DOAR în interiorul subprogramului care le-a definit; au o valoare nedefinită, reziduală, pentru aceasta TREBUIE inițializate;
- **Variabile globale:** sunt definite în afara oricărui subprogram; sunt vizibile în toate subprogramele definite DUPĂ declararea lor, adică pot fi folosite și modificate de orice subprogram; ele sunt păstrate permanent în zona/segmentul de date din memorie; la declarare ele sunt inițializate cu 0;
- **Observație:** Dacă sunt definite două variabile, una locală și una globală, CU ACELAȘI NUME, în subprogramul în care s-a definit variabila globală, compilatorul va folosi variabila locală.

Elementele unui subprogram

- **Prototipul subprogramului:** linia de program care declară subprogramul, și care conține următoarele informații: tipul rezultatului, numele subprogramului și tipul parametrilor folosiți pentru comunicare (nu și numele parametrilor); prototipul se termină cu caracterul **;**(punct-virgulă)
- **Antetul subprogramului:** linia de recunoaștere/definire a subprogramului, în care se specifică tipul subprogramului, numele acestuia și lista parametrilor formali, separați prin caracterul **,**(virgulă); antetul nu se termină cu caracterul **;** (punct-virgulă)
- **Apelul subprogramului:** linia de program care lansează în execuție subprogramul

Identificarea elementelor unui subprogram

- Prototipul subprogramului
- Antetul subprogramului
- Apelul subprogramului
- Definiţia subprogramului
- Modulul apelant
- Modulul apelat

```
# include <iostream.h>
void functie();
void main()
{ functie();
}
void functie()
{ cout<<"exemplu de functie";
}
```

Prototipul şi antetul unei functii

O functie poate fi declarată prin ***prototip***:

void citeste(int , float); // se încheie cu ;

O funcţie trebuie să aibă un ***antet*** prin care se precizează interfaţa dintre program şi subprogram:

void citeste (int &n, float &m)

Parametrii de comunicare

Stabilesc legăturile dintre modulul apelant și modulul apelat, realizează legătura dintre module; Tipul parametrilor poate fi: întreg, real, caracter, pointer, tablou (vectori, matrice, string-uri). Aceștia sunt:

- **Parametrii de intrare:** datele ce ”se dau” subprogramului, ce vor fi prelucrate de subprogram
- **Parametrii de ieșire:** rezultatele obținute de subprogram după ce le prelucrează și pe care le comunică modulului apelant, după ce subprogramul își termină execuția; reprezintă valorile returnate de către subprogram
- **Parametrii de intrare/ieșire:** datele ale căror valori pot fi modificate atât de subprogram cât și de modulul apelant; reprezintă valorile returnate de către subprogram

Clasificarea parametrilor după locul unde apar aceștia

- **Parametrii formali:** apar în antet, precedaţi de tipul lor, separaţi prin caracterul ,(virgulă)
- **Parametrii efectivi/actuali:** apar la apel, doar cu numele, separaţi prin caracterul ,(virgulă)

Observaţii:

1. Parametrii formali şi cei actuali trebuie să corespundă ca număr, tip şi poziţie
2. Numele parametrilor actuali pot fi diferite de numele parametrilor formali

Cum circulă datele?

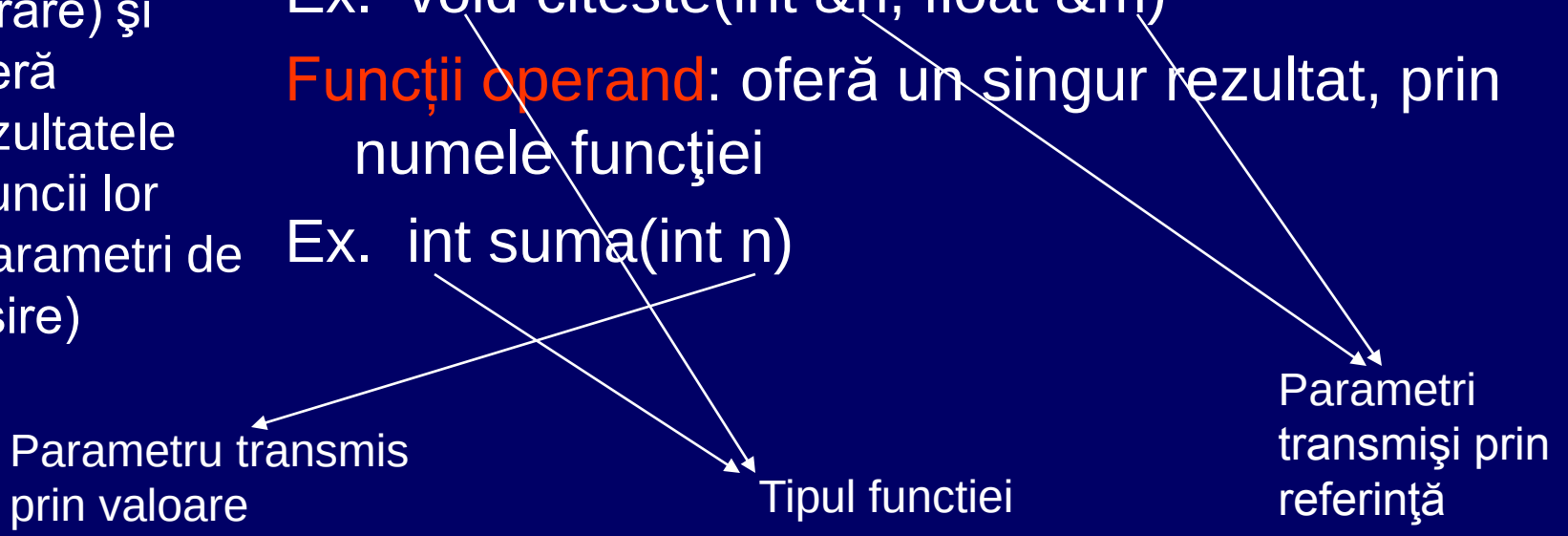
Subalternii primesc sarcini (parametri de intrare) şi oferă rezultatele muncii lor (parametri de ieşire)

Funcţiile procedurale: oferă unul sau mai multe rezultate prin intermediul parametrilor, sau nu oferă nici un rezultat

Ex. void citeste(int &n, float &m)

Funcţii operand: oferă un singur rezultat, prin numele funcţiei

Ex. int suma(int n)



Funcție operand:

- subprogram care returnează un rezultat chiar prin numele lui, dar eventual poate returna și alte rezultate prin intermediul parametrilor de ieșire; apelul unei funcții operand se realizează în expresii (atribuire, instrucțiuni de control (if, while, for)) sau ca parametrii ai unei alte funcții (de exemplu: if (prim(invers(x))), sau cout<<prim(x));
- Tipul unei funcţii operand poate fi: numeric(intreg sau real), caracter, pointer, înregistrare(struct), NU poate fi tablou (vectori, matrici, string-uri).
- O funcție operand trebuie să conțină cel puțin o instrucțiune return, prin care returnează o valoare modulului apelant.
- La prima întâlnire a unei instrucțiuni return, execuția funcției se întrerupe, restul instrucțiunilor nu se mai execută.

Funcție procedurală:

- subprogram care furnizează modulului apelant unul, mai multe sau nici un rezultat, rezultatele sunt furnizate prin intermediul parametrilor; apelul unei funcții procedurale se realizează printr-o instrucțiune procedurală cu sintaxa: **nume_subprogram(listă de parametri actuali), de** exemplu sch(a,b);
- Tipul unei funcţii procedurale este void (nici un tip)

Utilizarea stivei (zona STACK de memorie)

În zona stack se păstrează temporar informațiile despre modulul apelat;

Etapele executate la apel sunt:

1. se întrerupe execuția modulului apelant
2. se salvează pe stack:
    i. adresa de revenire a instrucțiunii imediat următoare apelului
    ii. valorile parametrilor formali
    iii. valorile variabilelor locale

- când execuția subprogramului s-a încheiat:
    i. se eliberează zona stack de toate variabilele locale și parametrii
    ii. se revine în modulul apelant la adresa de revenire

Modul de transfer al parametrilor

- Prin valoare: parametrului i se atribuie o valoare, o expresie sau conţinutul unei variabile; se folosește, de obicei, numai pentru parametrii de intrare; dacă se utilizează, totuși, acest mod pentru transmiterea rezultatelor, se pot folosi variabile de tip pointer (Varianta 2)
- Prin referinţă: subprogramul primeşte o adresă de memorie la care ste stocată variabila primită ca parametru; se foloseşte pentru parametrii de ieşire, sau intrare/ieşire; în lista parametrilor formali, sunt precedaţi de operatorul adresă de memorie (caracterul **&)**;

Transfer prin valoare

Varianta 1 (transfer prin valoare a parametrilor de intrare)

```
# include <iostream.h>
int a,b;
void sch (int x, int y)
{ int z;
   z=x; x=y; y=z;
}
int main()
{ cin>>a>>b;            //a=10, b=20
   sch(a,b);
   cout<<a<<` `<<b;   //a=10, b=20
   return 0;
}
```

Varianta 2 (transfer prin valoare a parametrilor de ieșire, folosind ponteri)

```
# include <iostream.h>
int a,b;
void sch (int *x, int *y)
{ int z;
   z=*x; *x=*y; *y=z;
}
int main()
{ cin>>a>>b;            //a=10, b=20
   sch(&a,&b);          //atenție la
                        //parametrii
   cout<<a<<` `<<b;     //a=20, b=10
   return 0;
}
```

Observații:

Parametrii efectivi/actuali corespunzători parametrilor formali de intrare transmişi prin valoare POT fi: variabile, constante, expresii, apel de funcţie operand, de exemplu:

- **cout<<prim(x);**
- **cout<<prim(15);**
- **cout<<prim(x+y);**
- **cout<<prim(invers(x));**

unde funcţia prim(x) verifică dacă x este număr prim, iar funcţia invers(x) returnează inversul parametrului x.

Parametrii formali corespunzători parametrilor valoare pot fi iniţializaţi în antetul subprogramului, ca în exemplul de mai jos:

```
# include <iostream.h>
int a,b;
int test (int a=10, int b=20)
{ return a+b;
}
int main()
{  cout<<test(30,40)<<endl;      //se afiseaza 70
    cout<<test(30)<<endl;        //se afiseaza 50
    cout<<test()<<endl;          //se afiseaza 30
   return 0;
}
```

Transfer prin referinţă

Varianta 3 (transfer prin referinţă a parametrilor de ieşire)

Parametrii efectivi corespunzători parametrilor formali transmişi prin referinţă TREBUIE să fie doar variabile, adică nu pot fi constante, expresii, apeluri ale altor funcţii;

```
# include <iostream.h>
int a,b;
void sch (int &x, int &y)
{ int z;
   z=x; x=y; y=z;
}
int main()
{ cin>>a>>b;  //a=10, b=20
   sch(a,b);       /* nu se poate apela sch(20, 30),
     parametrii actuali TREBUIE să fie variabile */
   cout<<a<<` `<<b;  //a=20, b=10
   return 0;
}
```

Transmiterea parametrilor prin valoare

- Declararea functiei

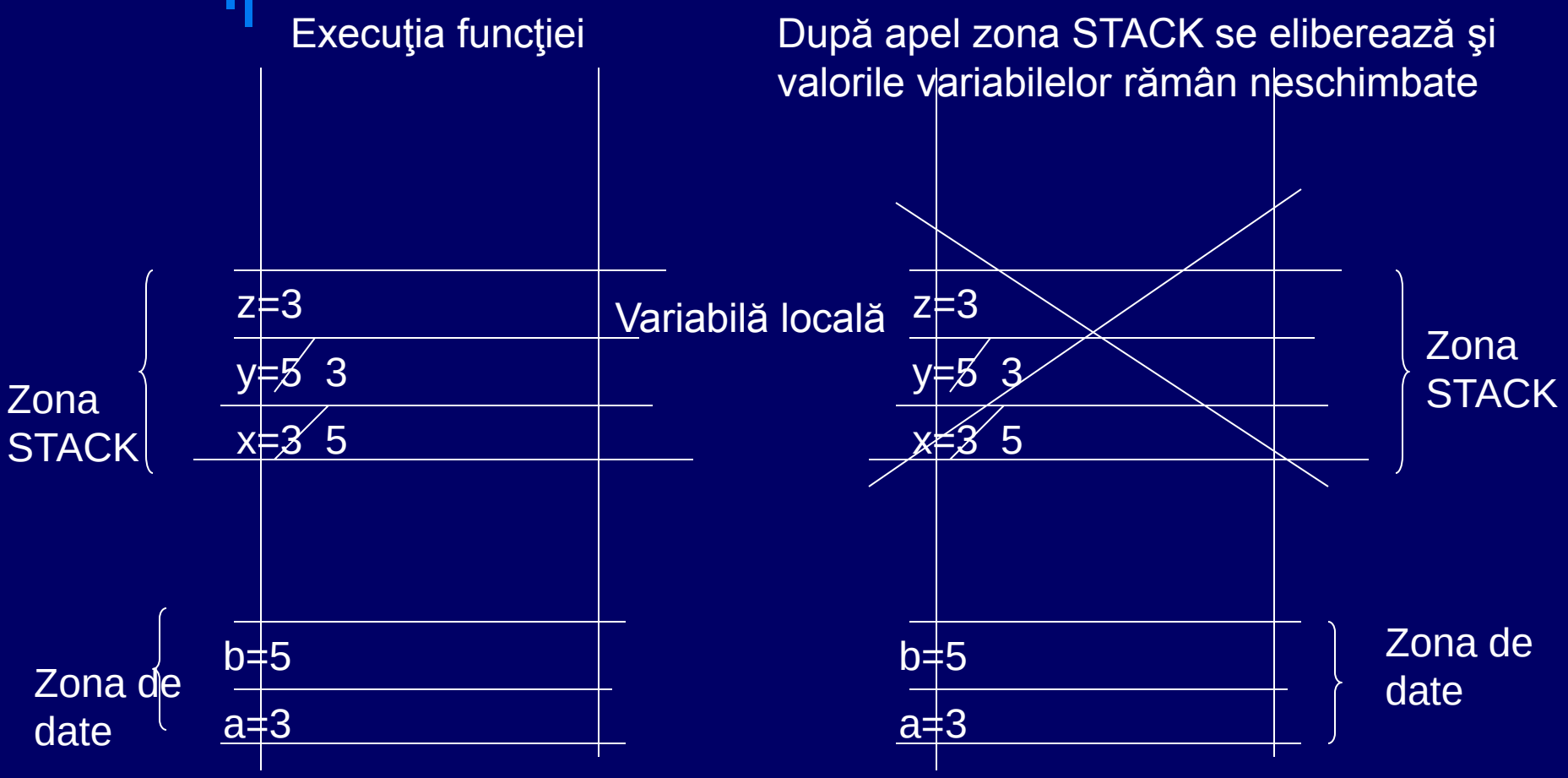
```
void sch(int x, int y)
{ int z;
  z=x;
  x=y;
  y=z;
}
```

- Apelul funcţiei

```
void main()
{ cin>>a>>b;
  cout<<a<<‘ ‘<<b;
  sch(a,b);
 cout<<a<<‘ ‘<<b;
}
```

Observăm că nu intevine nicio schimbare a valorilor celor doua variabile a şi b, în urma apelului funcţiei ***sch,*** deoarece parametrii au fost transmişi prin valoare

Reprezentarea pe stivă (STACK)



Transmiterea parametrilor prin valoare folosind variabile de tip pointer

- Declararea functiei

```
void sch(int *x, int *y)
{ int  z;
   z=*x;
  *x=*y;
  *y=z;
}
```

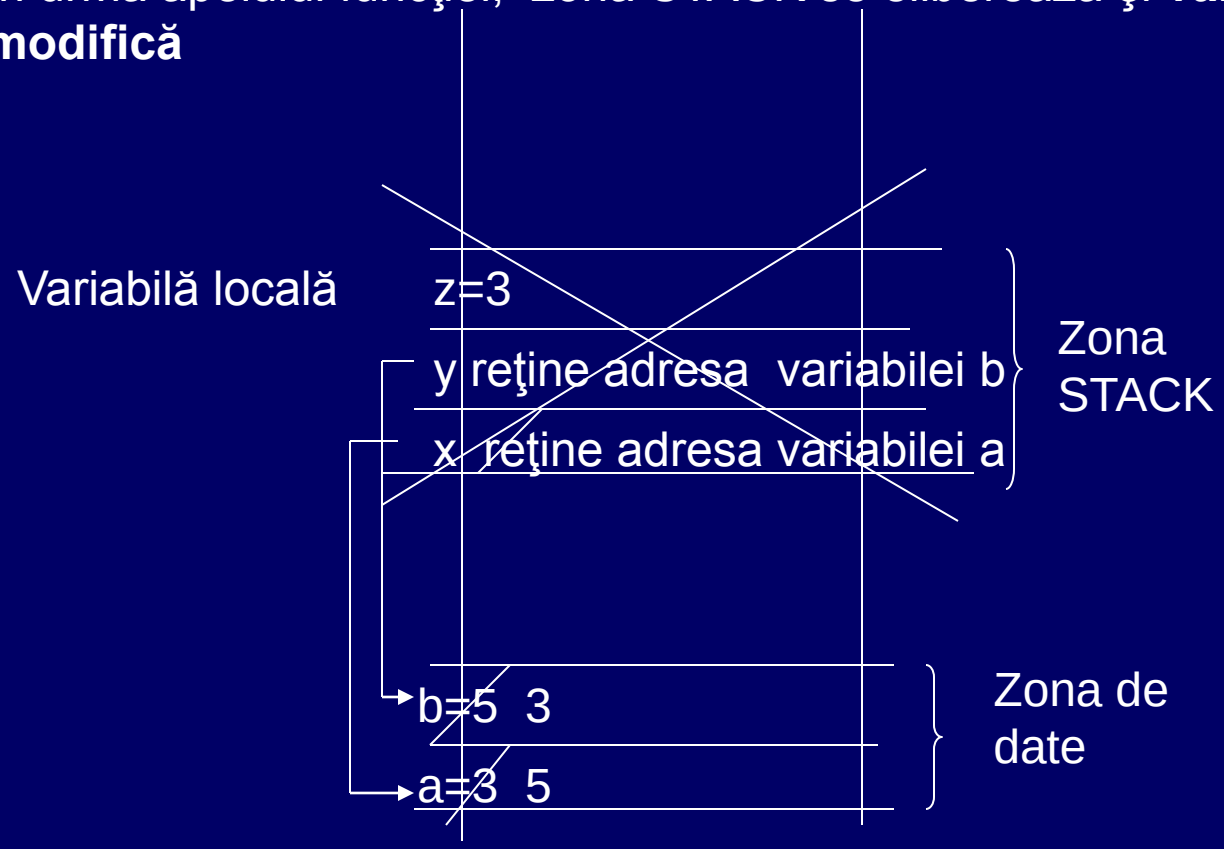
- Apelul functiei

```
void main()
{ cin>>a>>b;
  cout<<a<<‘ ‘<<b;
  sch(&a, &b);
 cout<<a<<‘ ‘<<b;
}
```

Observăm că intevine schimbarea valorilor celor două variabile a şi b, în urma apelului functiei ***sch,*** deoarece pe stivă s-a transmis adresa variabilelor .

Reprezentarea pe stivă (STACK)

În urma apelului funcţiei, zona STACK se eliberează şi **valorile variabilelor a şi b se modifică**



Transmiterea parametrilor prin referinţă

- Declararea functiei

```
void sch(int &x, int &y)
{ int z;
  z=x;
  x=y;
  y=z;
}
```

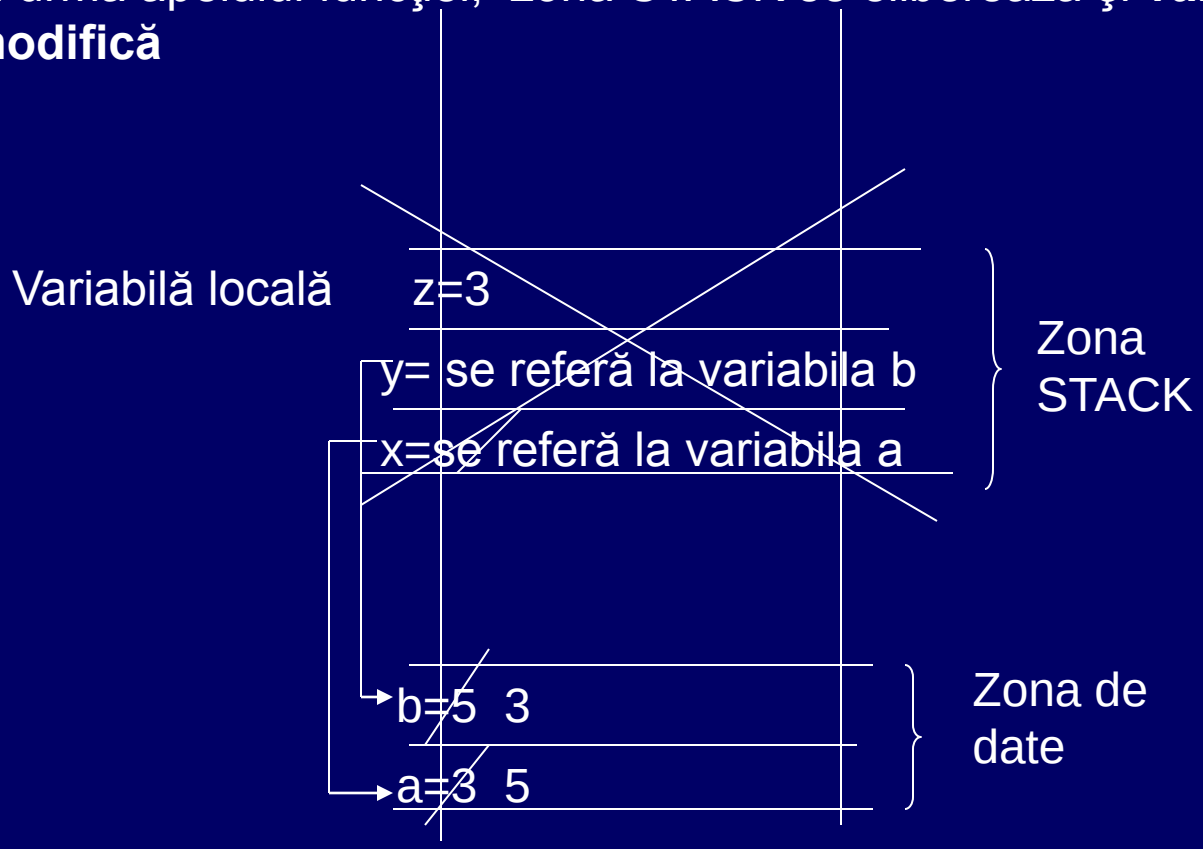
- Apelul functiei

```
void main()
{ cin>>a>>b;
  cout<<a<<‘ ‘<<b;
  sch(a,b);
 cout<<a<<‘ ‘<<b;
}
```

Observăm că intevine schimbarea valorilor celor doua variabile a şi b, în urma apelului functiei ***sch,*** x şi y fiind variabile de tip referinţă

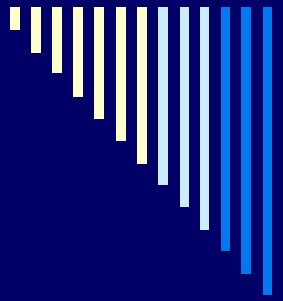
Reprezentarea pe stivă (STACK)

În urma apelului funcţiei, zona STACK se eliberează şi **valorile variabilelor a şi b se modifică**



Tablourile de memorie ca parametri de funcţii

- Dacă parametrii de ieşire sau intrare/ieşire sunt tablouri sau şiruri de caractere, NU trebuie folosit transferul prin referinţă, deoarece numele tablourilor (vectori, matrici, stringuri...) sunt pointeri (adrese ale primului element al tabloului). Prin urmare se foloseşte transfer prin valoare folosind pointeri, pe stiva stack se transferă adresa tabloului şi orice modificare se face la acea adresă.



Vectori ca parametrii

- Dacă un parametru formal este un tablou unidimensional (vector), la declararea acestuia nu trebuie precizată lungimea fizică a acestuia, parantezele drepte ce apar după numele vectorului arată compilatorului că este vorba despre un vector

de exemplu **void F(int n, int v[ ])**.

Matrici ca parametri

- Dacă se transmite ca paramentru un tablou bidimensional, adică o matrice, la declararea acestuia în lista parametrilor formali, nu trebuie precizat numărul de rânduri, în schimb numărul de coloane este obigatoriu să apară.

de exemplu **void G(int n, int a[ ][20])**.

Bibliografie

- Mariana Miloşescu, Manual de informatică pentru clasa a X-a, Editura Didactică şi Pedagogică
- Emanuela Cerchez, Marinel Şerban, Programarea în limbajul C/C++ pentru liceu, Editura Polirom
- Clara Ionescu, Informatică pentru grupele de performanţă, Editura Dacia Educaţional