11. Funcţii (2).

11.1. Mecanisme de transfer ale parametrilor.

În limbajele de programare există două mecanisme principale de transfer ale parametrilor: ***transferul prin valoare*** şi ***transferul prin referinţă***.

În C parametrii se transferă numai prin valoare- aceasta înseamnă că parametrii actuali sunt copiaţi în zona de memorie rezervată pentru parametrii formali. Modificarea parametrilor formali se va reflecta asupra copiilor parametrilor actuali şi nu afectează parametrii actuali, adică ***parametrii actuali nu pot fi modificaţi !***

Astfel funcţia:

```
void schimb(int a, int b)
{ int c=a;
      a=b;
      b=c;
}
```

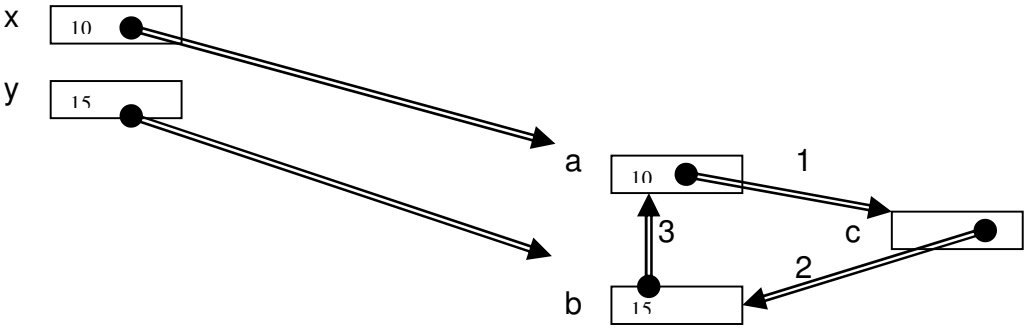
nu produce interschimbul parametrilor actuali **x** şi **y** din funcţia **main():**

```
int main()
{ int x=10,y=15;
  printf("%d\t%d\n", x, y);
  schimb(x, y);
  printf("%d\t%d\n", x, y);
}
```

Se afişează:

```
10      15
10      15
```

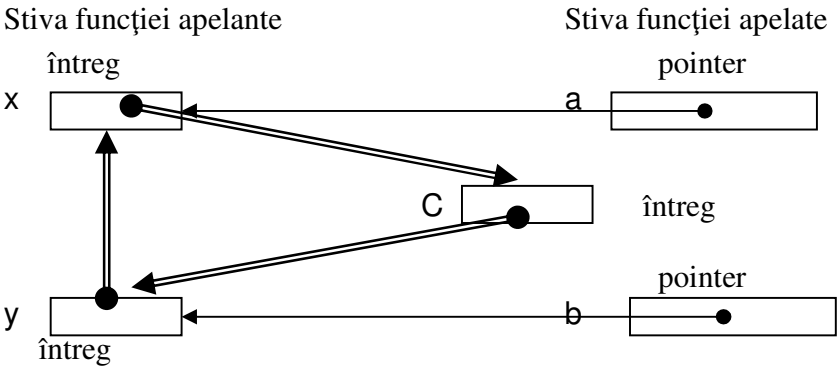
Stiva funcţiei apelante Stiva funcţiei apelate



În cazul transferului prin referinţă, pentru modificarea parametrilor actuali, funcţiei i se transmit nu valorile parametrilor actuali, ci adresele lor.
Forma corectă a funcţiei **schimb()** obţinută prin ***simularea transferului prin referinţă cu pointeri*** este:

```
void schimb(int *a, int *b)
{ int c;
   c = *a;
  *a = *b;
```

```
  *b = c;
}
int main()
{ int x=10,y=15;
  printf("%d\t%d\n", x, y);
  schimb(&x, &y);
  printf("%d\t%d\n", x, y);
}
```



In consecinţă, pentru a transmite un rezultat prin lista de parametri (adică pentru a modifica un parametru) va trebui să declarăm parametrul formal ca pointer.

Tablourile sunt transmise întotdeauna prin referinţă, adică un parametru formal tablou reprezintă o adresă (un pointer) şi anume adresa de început a tabloului.

11.2. Funcţii care întorc pointeri.

Funcţia **`strcpy()`** întoarce adresa şirului destinaţie:

```
char *strcpy(char *d, char *s)
{ char *p=d;
  while (*p++=*s++)
         ;
  return d;
}
```

Aceasta ne permite ca simultan cu copierea să calculăm lungimea şirului copiat:

```
n=strlen((strcpy)d,s));
```

Dacă funcţia întoarce adresa unei variabile locale, atunci aceasta trebuie să fie în mod obligatoriu în clasa **`static`.**
De asemeni nu trebuiesc întoarse adrese ale unor parametri, deoarece aceştia sunt transmişi prin stivă.

Să considerăm ca exemplu, două funcţii: una care evaluează un polinom într-un punct, cealaltă care calculează derivata unui polinom. A doua funcţie întoarce un pointer la tabloul coeficienţilor polinomului derivat. Acest tablou, declarat local funcţiei de derivare nu poate reprezenta rezultatul întors de funcţie, decât dacă este alocat static, nu în stivă, ci în zona de date. În funcţia `main()` evaluăm polinomul derivat într-un punct, compunând apelurile celor două funcţii, în lista parametrilor funcţiei de evaluare apare apelul funcţiei de derivare.

```
#include <stdio.h>
#include <stdlib.h>
double peval(int, double*,double); //evaluare polinom
double *pderiv(int, double*, int*); //derivare polinom
int main()
{
    int i, na, nb;
    double a[]={2.,5.,-3.,4.}; // polinomul 2x^3+5x^2-3x+4
    na = sizeof(a)/sizeof(a[0])-1; //gradul polinomului
    printf("%4.0lf\n", peval(nb, pderiv(na,a,&nb), 1.));
    getchar();
    return 0;
}
double peval(int n, double *a,double x){
    int i;
    double p=a[0];
    for(i=1; i<=n; i++)
       p = x*p + a[i];
    return p;
}
double *pderiv(int n, double *a, int *nd){
    static double b[10];
    int i;
    for(i=0; i<n; i++)
       b[i] = (n-i)*a[i];
    *nd = n-1;
    return b;
}
```

11.3. Pointeri la funcţii.

Numele unei funcţii reprezintă adresa de memorie la care începe funcţia. Numele funcţiei este, de fapt, un pointer la funcţie.
Se poate stabili o corespondenţă între variabile şi funcţii prin intermediul pointerilor la funcţii. Ca şi variabilele, aceşti pointeri:

- pot primi ca valori funcţii;
- pot fi transmişi ca parametrii altor funcţii
- pot fi intorşi ca rezultate de către funcţii

La declararea unui pointer către o funcţie trebuiesc precizate toate informaţiile despre funcţie, adică:

- tipul funcţiei
- numărul de parametri
- tipul parametrilor

care ne vor permite să apelăm indirect funcţia prin intermediul pointerului.

Declararea unui pointer la o funcţie se face prin:

```
tip (*pf)(listă_parametri_formali);
```

Dacă nu s-ar folosi paranteze, ar fi vorba de o funcţie care întoarce un pointer.
Apelul unei funcţii prin intermediul unui pointer are forma:

```
(*pf)(listă_parametri_actuali);
```

Este permis şi apelul fără indirectare:

```
pf(listă_parametri_actuali);
```

Este posibil să creem un tablou de pointeri la funcţii; apelarea funcţiilor, în acest caz, se face prin referirea la componentele tabloului.
De exemplu, iniţializarea unui tablou cu pointeri cu funcţiile matematice uzuale se face prin:

```
double (*tabfun[])(double) = {sin, cos, tan, exp, log};
```

Pentru a calcula rădăcina de ordinul **5** din **e** este suficientă atribuirea:

```
y = (*tabfun[3])(0.2);
```

Numele unei funcţii fiind un pointer către funcţie, poate fi folosit ca parametru în apeluri de funcţii.
În acest mod putem transmite în lista de parametri a unei funcţii – numele altei funcţii.
De exemplu, o funcţie care calculează integrala definită:

$$I = \int_a^b f(x)\,dx$$

va avea prototipul:

```
double integrala(double, double, double(*)(double));
```

Exemplul : Definiţi o funcţie pentru calculul unei integrale definite prin metoda trapezelor,cu un număr fixat **n** ***de puncte de diviziune:***

$$\int_a^b f(x)dx = h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+ih)\right] \quad cu \quad h = \frac{b-a}{n}$$

Folosiţi apoi această funcţie pentru calculul unei integrale definite cu o precizie dată **ε**. ***Această precizie este atinsă în momentul în care diferenţa între două integrale, calculate cu*** **n**, ***respectiv*** **2n** ***puncte de diviziune este inferioară lui*** **ε**

```
#include <math.h>
double sinxp(double x){ return sin(x*x); }
double trapez(double,double,int,double(*)());
double a=0.0, b=1.0, eps=1E-6;
int N=10;
int main()
{ int n=N;
  double In,I2n,vabs;
  In=trapez(a,b,n,sinxp);
  do { n*=2;
       I2n=trapez(a,b,n,sinxp);
       if((vabs=In-I2n)<0) vabs=-vabs;
       In=I2n;
  } while(vabs > eps);
  printf("%6.2lf\n", I2n);
}
double trapez(double a,double b,int n,double(*f)())
{ double h,s;
  int i;
```

```
  h=(b-a)/n;
  for(s=0.0,i=1;i<n;i++)
    s+=(*f)(a+i*h);
  s+=((*f)(a)+(*f)(b))/2.0;
  s*=h;
  return s;
}
```

11.4. Declaratii complexe (şarade).

O declaratie complexă este o combinaţie de pointeri, tablouri si functii. In acest scop se folosesc ***atributele***:

() – functie
[] – tablou
***** - pointer

care pot genera urmatoarele *combinatii*:

*** ()** – funcţie ce returnează un pointer
(*)() – pointer la o funcţie
*** []** - tablou de pointeri
(*) [] – pointer la tablou
[][] – tablou bidimensional

Există şi ***combinaţii incorecte***, provenite din faptul că în C nu este permisă declararea:

- unui tablou de funcţii
- unei funcţii ce returnează un tablou.

Acestea sunt:

() [] – funcţie ce returnează un tablou
[] () – tablou de funcţii
() () – funcţie ce returnează o funcţie

Pentru a interpreta o declaraţie complexă vom inlocui *atributele* prin următoarele *şabloane text*:

Atribut	Şablon text
()	funcţia returnează
[n]	tablou de **n**
*****	pointer la

Descifrarea unei declaraţii complexe se face aplicând ***regula dreapta – stânga***, care presupune următorii paşi:

- se incepe cu identificatorul
- se caută în dreapta identificatorului un atribut
- dacă nu există, se caută în partea stangă
- se substituie atributul cu şablonul text corespunzător
- se continuă substituţia dreapta-stânga
- se opreşte procesul la întâlnirea tipului datei.

De exemplu:

```
int (* a[10]) ( );
```

tablou de 10
pointeri la
funcţii ce returnează **int**

double (*(*pf)())[3][4];
pointer la
o funcţie ce returnează un pointer
la un tablou cu 3 linii si 4 coloane de **double**

In loc de a construi declaraţii complexe, se preferă, pentru creşterea clarităţii, să definim progresiv noi tipuri folosind **typedef**. Reamintim că declaraţia **typedef** asociază un nume unei definiri de tip:

```
typedef  <definire de tip> identificator;
```

Exemple:

```
typedef char *SIR;        /*tipul SIR=sir de caractere*/
typedef float VECTOR[10];/*tipul VECTOR=tablou de 10float*/
typedef float MATRICE[10][10];/*tipul MATRICE= tablou de 10x10
float*/
typedef double (*PFADRD)(double);/*tipul PFADRD=pointer la
          functie de argument double si rezultat double */
```

Vom putea folosi aceste tipuri noi în definirea de variabile:

```
SIR s1, s2;
VECTOR b, x;
MATRICE a;
PFADRD pf1;
```

11.5. Probleme propuse.