POINTERI LA FUNCŢII

1. Declararea unui pointer la funcţii

Pointerii folosiţi până acum au fost pointeri la diferite tipuri de date, dar este posibil să avem şi pointeri la funcţii. Pointerii la funcţii sunt folosiţi din aceleaşi motive ca şi pointerii la date: atunci când se doreşte un alt nivel de indirectare, când dorim ca aceeaşi secvenţă de cod să apeleze funcţii diferite depinzând de condiţiile concrete ale programului.

Ca şi în cazul pointerilor la date, pentru utilizarea pointerilor la funcţii trebuie să declarăm o variabilă care să conţină un pointer la funcţie. Un pointer la o funcţie se declară astfel:

```
tip (*pf)(tip1 p1, tip2 p2, ..., tipn pn);
```

unde

tip este tipul funcţiei (tipul valorii returnate de funcţie)

tip_1 p_1, tip_2 p_2, ..., tip_n p_n este lista parametrilor funcţiei care va fi accesată prin intermediul pointerului. Numele parametrilor, adică **p_1, p_2, ..., p_n** pot lipsi.

Exemplu, dacă scriem

```
int (*pfi)(float a, int b);
```

se declară **pfi** ca fiind un pointer la o funcţie care va returna un întreg. Ca şi în alte declaraţii ***** indică faptul că avem un pointer, iar **( )** arată că avem de a face cu o funcţie. Parantezele din **(*pfi)** sunt necesare deoarece şi în declaraţii există o anumită precedenţă a operatorilor (o anumită ordine de evaluare – interpretare) ca şi în expresii şi când ordinea implicită nu este cea dorită, trebuie să o schimbăm folosind parantezele de explicitare. În declaraţii, **( )** - operatori de funcţie şi **[ ]** - operatorii de indexare sunt mai prioritari decât ***** indicând pointerii. Fără parantezele menţionate, declaraţia de mai sus arată astfel:

```
int *pfi(float a, int b);
```

şi declară o funcţie **pfi** care va returna un pointer la întreg. Cu parantezele explicite, **int (*pfi)()** ne spune că **pfi** este mai întâi un pointer, că acest pointer indică o funcţie şi mai apoi că funcţia respectivă returnează un întreg.

Pointerii la funcţii se pot defini şi ca noi tipuri de date prin utilizarea declaraţiei de tip **typedef**. De exemplu, putem scrie

```
typedef int (*FPTR)(float a, int b);
```

şi identificatorul **FPTR** este un sinonim pentru tipul de dată **pointer la o funcţie care returnează un întreg**, astfel încât declaraţia

```
FPTR pfi;
```

este echivalentă cu

```
int (*pfi)(float a, int b);
```

O dată declarat, unui pointer la funcţie i se poate atribui valoarea adresei de început a funcţiei dorite. Dacă avem prototipurile următoarelor funcţii

```
int f1(float a, int b);
int f2(float a, int b);
int f3(float a, int b);
```

atunci putem scrie:

```
pfi = &f1;
```

sau

```
if (conditie)
   pfi = &f2;
else
   pfi = &f3;
```

Bineînţeles, nu vom fi restrânşi la aceste două forme, putem asigna pointeri la funcţii în orice condiţii dorim. Al doilea exemplu poate fi scris mai compact:

```
pfi = conditie ? &f2 : &f3;
```

În aceste exemple am folosit operatorul **&**, aşa cum am făcut până acum pentru a genera un pointer. Totuşi când generăm pointeri la funcţii, operatorul **&** este opţional, deoarece atunci când menţionăm numele unei funcţii fără să o apelăm menţionăm de fapt adresa funcţiei respective, numele unei funcţii fiind de fapt un pointer la funcţia dată. Astfel se poate scrie:

```
pfi = f1;
```

sau

```
if (conditie)
    pfi = f2;
else
    pfi = f3;
```

sau

```
pfi = conditie ? f2 : f3;
```

Faptul că un pointer la o funcţie este generat automat când o funcţie apare într-o expresie, dar nu este apelată este asemănător, şi de fapt este de legat de faptul că un

pointer la primul element al unui vector este generat automat atunci când un vector apare într-o expresie.

Având o variabilă pointer la o funcţie care conţine adresa unei funcţii, putem apela (folosi) funcţia respectivă astfel:

1. scriem numele variabilei pointer la funcţie

```
pfi
```

2. se foloseşte operatorul * în faţă pentru "a accesa conţinutul acelui pointer"

```
*pfi
```

Această expresie reprezintă tocmai funcţia pe care dorim să o folosim.

3. adăugăm lista de parametri în paranteze împreună cu un set de paranteze pentru a avea precedenţa dorită a operaţiilor:

```
(*pfi)(arg1, arg2)
```

Formăm astfel apelul la funcţie.

Parantezele din expresia **(*pfi)** au aceeaşi explicaţie ca la declararea pointerului la o funcţie. Dacă scriem

```
*pfi(arg1, arg2)
```

interpretarea este: apelează funcţia **pfi** (care va returna un pointer), transferă-i argumentele arg1 şi arg2 şi ia conţinutul de la adresa indicată de variabila pointer la întoarcere. Totuşi ceea ce dorim să facem este următorul lucru: ia conţinutul lui **pfi** (care este un pointer la o funcţie), apelează funcţia spre care pointează, transmiţându-i argumentele **arg1** şi **arg2**. Din nou, parantezele explicite schimbă precedenţa implicită, astfel încât se aplică mai întâi operatorul * şi apoi se apelează funcţia.

Expresia

```
pfi(arg1, arg2)
```

este echivalentă cu

```
(*pfi)(arg1, arg2).
```

Atunci când apelăm o funcţie pointată de un pointer la funcţie, operatorul * este opţional. Este recomandabilă folosirea lui pentru a scoate în evidenţa faptul că folosim un pointer la funcţie şi nu o funcţie propriu-zisă.

Pentru fiecare funcţie folosită trebuie să avem un prototip pentru a permite compilatorului să genereze corect codul pentru apelul funcţiilor şi să verifice dacă funcţia este apelată cu numărul şi tipul adecvat pentru argumente.

În general nu se va şti decât în momentul rulării programului care este funcţia pointată de **pfi**, astfel încât compilatorul nu poate verifica dacă apelul s-a făcut corect. Pe de altă parte, atunci când am declarat un pointer la funcţie a trebuit să declarăm tipul

valorii returnate de funcţia respectivă. De asemenea, putem declara prototipul argumentelor folosite de funcţie, adică putem scrie:

```
int (*pfi)(float a, int b);
```

Acum ştim că **pfi** este un pointer la o funcţie care acceptă două argumente şi care returnează un întreg. Având toate acestea specificate, compilatorul va putea să verifice corectitudinea anumitor apeluri pentru funcţia respectivă. De exemplu, dacă scriem:

```
(*pfi)(1, 2, 3)
```

compilatorul va semnaliza eroare, pentru că el ştie că funcţia nu poate accepta decât două argumente. De asemenea, compilatorul va verifica dacă funcţiile spre care pointează variabila pointer la funcţie au lista de argumente şi valoarea de retur în concordanţă cu declaraţia pentru pointer.

Deci pentru o variabilă pointer la funcţie trebuie să declară atât tipul valorii returnate, cât şi tipul argumentelor din lista de argumente.

Alte exemple de construire a pointerilor la funcţii:

```
double (*a)(int);
float (*b)(char *);
int * (*f1)(double, int);
double *(*f2)(char *, int , double *);
```

2. Pointeri la funcţii ca argumente în alte funcţii

În continuare să presupunem că o funcţie **f** are ca parametru o funcţie **g**. Dacă se scrie:

```
f(g);
```

înseamnă că lui **f** i se transmite un pointer la funcţia **g**.

Dacă prototipul funcţiei **g** este

```
tip_g g(lista_g);
```

atunci prototipul funcţiei **f** în care **g** este parametru va fi

```
tip_f f(tip_g (*g)(lista_g));
```

sau

```
tip_f f(tip_g (*)(lista_g));
```

iar definiţia funcţiei **f** va fi

```
tip_f f(tip_g (*p)(lista_g))  { ….}
```

unde **(*p)** poate fi şi **(*g)**.

Exemple:

```
int g(double);
double f(int (*)(double));
```

2.1. Exemple

2.1.1. Calculul ariei domeniului mărginit de graficul unei funcţii

Valoarea ariei domeniului mărginit de graficul unei funcţii este valoarea integralei acelei funcţii, valoare calculată luând ca limite capetele intervalului de reprezentare.

Fie funcţia, considerată fără bucle:

$$f:[a,b]\to R \tag{1}$$

Pentru calculul integralei se va folosi metoda trapezelor. Intervalul [a,b] va fi împărţit în n diviziuni, astfel că mărimea unei diviziuni va fi:

$$dx=\frac{b-a}{n} \tag{2}$$

Punctele x_i se vor calcula cu expresia:

$$x_i=a+i\cdot dx=x_{i-1}+dx \quad \text{cu} \quad i=1,\ldots,n-1 \tag{3}$$

iar aria corespunzătoare unei diviziuni (aria parţială) este:

$$A_i=\frac{f(x_i)+f(x_{i+1})}{2}\cdot(x_{i+1}-x_i) \tag{4}$$

Integrala se obţine ca sumă a ariilor parţiale:

$$I_n=\sum_{i=0}^{n-1}A_i=\frac{b-a}{n}\cdot\left[\frac{f(a)+f(b)}{2}+\sum_{i=1}^{n-1}f(x_i)\right] \tag{5}$$

Această modalitate de calcul a ariei este acelaşi, indiferent de expresia funcţiei f. Dacă vom lua în considerare de fiecare dată forma funcţiei f va trebui să scriem o funcţie care calculează integrala pentru fiecare funcţie pe care trebuie să o utilizăm. Pointerii la funcţii ne permit să transmitem ca parametru funcţia pentru care vrem să calculăm integrala astfel încât vom lua în considerare următorul prototip:

double integralaTrapez(double a, double b, int n, double (*f)(double));

în care:

a, b reprezintă capetele intervalului de integrare

n numărul de diviziuni ale intervalului

f pointer la o funcţie care primeşte un parametru de tip **double** şi returnează o valoare de tip **double**. Această funcţie calculează valorile funcţiei pentru care vrem să calculăm integrala într-un punct specificat ca parametru. Funcţia poate fi o funcţie proprie sau o funcţie din biblioteca matematică (de exemplu sin, sqrt, etc).

Funcţia **integralaTrapez** implementează formula (5) şi are forma:

```
double integralaTrapez(double a, double b, int n,
                       double (*f)(double))
```

```
{
    double val;       // Valoarea integralei
    double dx = (b-a)/n;
    double x;
    val = ((*f)(a) + (*f)(b))/2;
    for(x=a+dx; x<b; x = x+dx)
    {
        val = val + (*f)(x);
    }
    val = val * dx;
    return val;
}
```

Ce semnifică următoarele declaraţii?

```
int f1(int a, double (*f)(float *b));
int f2(int a, double f(char c), int (*g)(int d, int *e));
float *f3(double *a, int * (*f)(double));
double (*fc)(int a,  float ff(void));
int *(*fd)(float *a, double *f(int *d));
```

3. Tablouri de pointeri la funcţii

Ca orice tip de date pointerii la funcţii se pot grupa în tablouri.

Exemplu:avem un tablou de pointeri la funcţii care primesc ca parametru un double şi returnează un double:

```
double (*fm[10])(double);
```

Pentru citirea acestor declaraţii folosim aşa-numite regulă de citire dreapta-stânga: se porneşte de la numele variabilei **fm**, în dreapta avem **[10]** (deci este un;tablou, aici de 10 elemente); mergem în stânga: avem ***** (tablou de 10 elemente pointeri); din nou în dreapta **(** (tablou de 10 elemente pointeri la funcţii); din nou stânga **double** (tablou de 10 pointeri la funcţii care returnează un double); la dreapta **double** – lista de argumente a funcţiilor. în concluzie avem un tablou de 10 elemente, fiecare din ele este un pointer la o funcţie care returnează un double şi primeşte ca parametru o dată de tip double.

Exemplu: să se scrie un program care tabeloează funcţiile trigonometrice sin, asin, cos, acos, tan, atan între 0 şi 1 rad cu pasul de 0.05.

```
#include <stdio.h>
#include <math.h>

int main(void)
{
```

```
    double (*fm[])(double x) =
                {sin,  asin, cos, acos, tan, atan};
    int i;
    double x;
    double dx = 0.05;

    int nf = sizeof(fm)/sizeof(fm[0]);

    puts("Tabelul");
    for(x=0; x<=1; x+= dx)
    {
        printf("x = %5.3lf f= ", x);
        for(i=0; i<nf; i++)
            printf("%5.3lf ", (*fm[i])(x));
        printf("\n");
    }
    return 0;
}
```

4. Pointeri la funcţii ca membri în structuri

Pointerii la funcţii pot apare şi ca membri în structuri de date.

Această construcţie poate fi utilizată pentru construirea unor meniuri, de exemplu.

Fie tipul de dată FM sinonim pentru un pointer la funcţie care primeşte un double şi returnează un double:

```
typedef double (*FM)(double x);
```

O structură care are în componenţă acest pointer se declară astfel:

```
struct S1 {
    ..............  /* declaraţii pentru alţi membri */
    FM f;    /* echivalent cu double (*f) (double x); */
    ..............  /* declaraţii pentru alţi membri */
};
```

Putem construi astfel o structură care să conţină numele unei funcţii şi un pointer la funcţia respectivă. Exemplu:

```
struct functii {
    char *nume;
    double (*f) (double x);
};
```

În programul principal putem iniţializa un tablou de astfel de structuri pentru a-l folosi la construirea unui meniu. Exemplu:

```
struct functii tab_f[] = {
        {"sinus", sinus},
        {"cosinus", cos},
        {"tangenta", tan}
};
```

Definiţia funcţiei pentru construirea meniului este:

```
void meniu(struct functii tab[], int nf, char *msg)
{
    int i;
    puts(msg);
    for(i=0; i<nf; i++)
        printf("\t%d - %s\n",i+1, tab[i].nume);
    printf("\t0 - exit\n");
    printf("\t >> ");
}
```

În această funcţie tab[i].nume este numele prelucrării (funcţiei) dorite, iar apelul acestei funcţii este `(*tab_f[i].fm)(x)` (tab_f este tabloul de structuri definit mai sus).

TEMA

Problema nr. 1

Folosind tablouri de pointeri la funcţii, să se scrie un program care tabelează funcţiile de bibliotecă sinus, cosinus şi tangentă pentru valori cuprinse între 0 şi π cu un pas egal cu $\pi/20$ (în **C** există constanta **M_PI** a cărei valoare este egală cu π).

Problema nr. 2

Să se calculeze următoarele integrale:

$$\int_{-1}^{1} \sin(x^2 + 3x)dx$$

$$\int_{0}^{2} \left(x^2 + 4x + e^x\right)dx$$

Indicaţie:

Se va folosi pentru calculul integralei funcţia **integralaTrapez** prezentată la curs.

Problema nr. 3

Să se declare o structură **norme** care conţine ca membri un pointer la caracter şi un pointer la o funcţie care returnează un double şi primeşte ca parametri un pointer la double şi un întreg (după modelul structurii functii dată mai sus).

Să se scrie un program care calculează, pentru un tablou unidimensional (vector) următoarele norme:

$\|x\|_\infty = \max|x_i|$ - norma infinit

$\|x\|_1 = |x_1| + |x_2| + ... + |x_n|$ - norma 1

$\|x\|_2 = \sqrt{|x_1|^2 + |x_2|^2 + ... + |x_n|^2}$ - norma 2

Programul defineşte şi iniţializează (după modelul dat mai sus) un tablou de structuri **norme** cu numele şi funcţiile care calculează cele trei norme, afişează un meniu care foloseşte acest tablou de structuri, iar prelucrarea dorită se va face prin intermediul pointerilor la funcţii.

Se poate folosi, ca exemplu, funcţia **meniu** dată mai sus.

Problema nr. 4

Să se definească o funcţie **generică** de ordonare a unui şir de date (metoda folosită este metoda bulelor). Funcţia primeşte ca parametri adresa zonei de memorie unde se găseşte şirul de date (un pointer **generic**), numărul de date care se ordonează, dimensiunea unui element din şir, precum şi un pointer la o funcţie care realizează compararea a două elemente din şir.

Să se definească o funcţie **generică** de interschimbare a două elemente dintr-un şir de date (funcţie ce poate fi folosită pe orice tip de date).

Folosind aceste funcţii să se scrie un program care face ordonarea unor date citite de la tastatură şi afişează şirul ordonat pe monitor. Aceste date pot fi: un şir de numere reale în dublă precizie sau un text.

Atenţie! Trebuie scrisă o singură funcţie de ordonare (sortare) care va fi folosită atât pentru sortarea textului, cât şi a şirului de date numerice şi o singură funcţie de interschimbare.

Problema nr. 5

Se reia **Problema nr. 4** folosind funcţia de bibliotecă **qsort.**