POINTERI LA FUNCȚII

Mecanismul de callback: o funcție primește ca parametru o altă funcție, pe care o utilizează în momentul în care are nevoie.

Funcții dispecer (dispatch function): furnizează o funcție (handler) în raport de valoarea unui parametru de intrare.

Declararea pointerilor la funcții:

```
tip_rezultat (*nume_pointer)(tipurile parametrilor)
```

Exemple:

- `int (*pf)(int, int)` -> un pointer `pf` la o funcție care primește ca parametrii două numere întregi și furnizează un număr întreg
- `void (*pf)(float)` -> un pointer `pf` la o funcție care primește ca parametru un număr real și nu furnizează nicio valoare

```
#include<stdio.h>

int suma(int x, int y)
{
    return x+y;
}

int produs(int x, int y)
{
    return x*y;
}

int main()
{
    int a = 18, b = 5, r, i;

    int (*pf)(int, int);

    for(i = 0; i < 10; i++)
    {
        if(i%2 == 0)
            pf = &produs;
        else
            pf = &suma;

        r = (*pf)(i, i);
```

```
            printf("Rezultat: %d\n", r);
        }

        return 0;
    }
```

Un pointer la o funcție conține o adresă din zona de cod asociată programului, respectiv adresa din zona de cod unde încep instrucțiunile programului (adresa primei instrucțiuni).

Pointerii la funcții NU pot fi modificați (nu au aritmetică) și NU pot fi alocați dinamic!

Un pointer la o funcție (adresa primei instrucțiuni) este echivalent cu numele funcției!

Exemplu: apelarea indirectă a unei funcții (folosind un pointer)

```
//pf = pointer la o functie = o adresa din zona de cod = un numar intreg
int (*pf)(int, int);
int s;

//varianta 1
pf = &suma;
s = (*pf)(7, 9);    //*pf = *(&suma) = suma
printf("s = %d\n", s);

//varianta 2
pf = suma;
s = (*pf)(7, 9);    //*pf = *(&suma) = suma
printf("s = %d\n", s);
//varianta 3
pf = suma;
s = pf(7, 9);    //*pf = *(&suma) = suma
printf("s = %d\n", s);
```

Exemplu: apelarea indirectă a unei funcții (folosind un pointer)

```
#include<stdio.h>

int suma(int x, int y)
{
    return x+y;
}
```

```
int produs(int x, int y)
{
    return x*y;
}

int main()
{
    int a = 18, b = 5, r, i;

    int (*pf)(int, int);

    for(i = 0; i < 10; i++)
    {
        if(i%2 == 0)
            pf = produs;
        else
            pf = suma;

        r = pf(i, i);
        printf("Rezultat: %d\n", r);
    }

    return 0;
}
```

Exemple: utilizarea mecanismul de callback

```
#include<stdio.h>

int suma(int x, int y)
{
    return x+y;
}

int produs(int x, int y)
{
    return x*y;
}

int expresie(int x, int y)
{
    return 2*x + 7*y;
}

//funcție generică = funcție care primește ca parametru o altă funcție
int calcul(int (*pf)(int, int), int x, int y)
{
    return pf(x, y);  //mecanismul de callback
}
```

```
int main()
{
    int a = 18, b = 5, r;

    r = calcul(suma, a, b);
    printf("Rezultat: %d\n", r);

    r = calcul(produs, a, b);
    printf("Rezultat: %d\n", r);

    r = calcul(expresie, a, b);
    printf("Rezultat: %d\n", r);

    return 0;
}
```

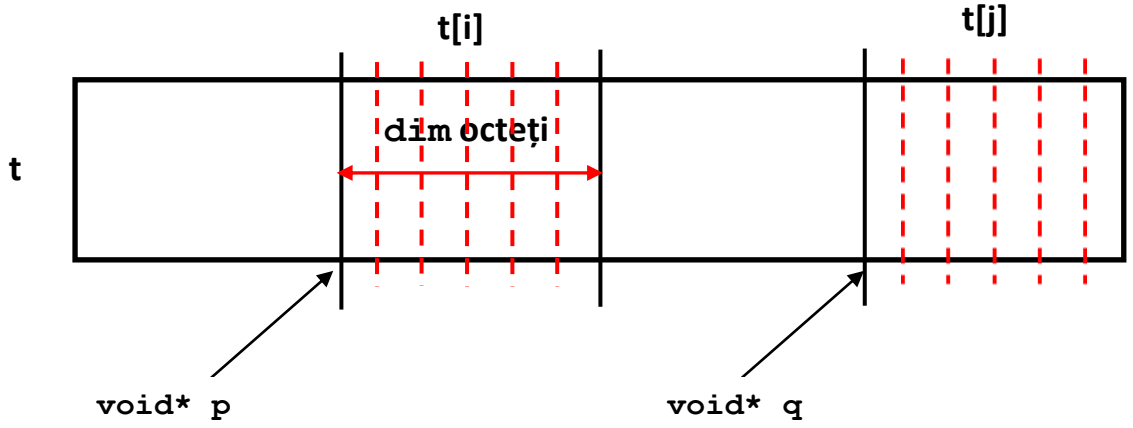
Funcția generică `qsort` (`stdlib.h`) – metoda Quicksort

Antetul funcției `qsort`:

```
void qsort(void* t, int n, int dim,
          int (*cmpElemente)(const void*, const void*))
```

Un tablou generic = adresa primului element (`t`) + numărul de elemente (`n`) + dimensiunea unui element în octeți (`dim`)

Parametrul `cmpElemente` este numele unei funcții comparator (un pointer la funcție) care stabilește când se vor interschimba elementele ale căror adrese sunt furnizate prin cei 2 pointeri generici (de tip `void*`).



O funcție comparator `int cmpElemente(const void* p, const void* q)` trebuie să furnizeze următoarele valori:

- **o valoare negativă dacă vrem ca t[i] să fie poziționat înainte lui t[j] sau, altfel zis, t[i] "<" t[j]**
- **valoarea 0 dacă nu ne interesează pozițiile relative ale lui t[i] și t[j] sau, altfel zis, t[i] "=" t[j]**
- **o valoare pozitivă dacă vrem ca t[i] să fie poziționat după t[j] sau, altfel zis, t[i] ">" t[j]**

Exemple:

```
int cmpCrescator(const void* p, const void* q)
{
    int vp = *(int*)p;
    int vq = *(int*)q;

    if(vp < vq)
        return -1;

    if(vp > vq)
        return 1;

    return 0;   //daca vp = vq
}

int cmpDescrescator(const void* p, const void* q)
{
    int vp = *(int*)p;
    int vq = *(int*)q;

    if(vp < vq)
        return 1;

    if(vp > vq)
        return -1;

    return 0;   //daca vp = vq
}

//valorile pare inaintea celor impare
int cmpParitate(const void* p, const void* q)
{
    int vp = *(int*)p;
    int vq = *(int*)q;
```

```
    if((vp % 2 == 0) && (vq % 2 != 0))
        return -1;

    if((vp % 2 != 0) && (vq % 2 == 0))
        return 1;

    return 0;   //daca vp are aceeasi paritate cu vq
                 //(ambele sunt pare sau ambele sunt impare)
}

//numerele pare înaintea celor impare
//numerele pare sortate crescător
//numerele impare sortate descrescător
int cmpParImpar(const void* p, const void* q)
{
    int vp = *(int*)p;
    int vq = *(int*)q;

    if(vp % 2 == 0 && vq % 2 != 0) return -1;
    if(vp % 2 != 0 && vq % 2 == 0) return 1;

    if(vp % 2 == 0 && vq % 2 == 0)
    {
        if(vp < vq) return -1;
        if(vp > vq) return 1;
        return 0;
    }

    if(vp % 2 != 0 && vq % 2 != 0)
    {
        if(vp > vq) return -1;
        if(vp < vq) return 1;
        return 0;
    }
}

void afisare(int v[], int n)
{
    int i;

    printf("\nTabloul:\n");
    for(i = 0; i < n; i++)
        printf("%d ", v[i]);
    printf("\n");
}

int main()
{
    int t[] = {2,10,-3,-7,1,-8,2,4,7,6,-3,1,-3,1,10,10};
```

```
    int n = sizeof(t) / sizeof(t[0]);

    afisare(t, n);

//    qsort(t, n, sizeof(t[0]), cmpCrescator);
//    afisare(t, n);
//
//    qsort(t, n, sizeof(t[0]), cmpDescrescator);
//    afisare(t, n);

    qsort(t, n, sizeof(t[0]), cmpParitate);
    afisare(t, n);

     return 0;
}
```

FUNCȚII CU NUMĂR VARIABIL DE PARAMETRI (varargs)

Funcțiile (macrouri) și tipurile de date necesare implementării funcțiilor cu număr variabil de parametri sunt definite în biblioteca `stdarg.h`.

Definirea unei funcții cu număr variabil de parametri:

```
tip_rezultat nume_funcție(parametri ficși, ...)
{
     ……….……
}
```

cel puțin un parametru fix — **parametrii variabili**

Listele de parametri variabili au tipul de date `va_list`**, definit în** `stdarg.h`.

Parametrilor variabili ai unei funcții NU li se pot preciza tipurile de date, ci acesta sunt definite formal de către programator (se stabilește o ordine a lor).

De asemenea, NU se poate afla numărul parametrilor variabili, ci programatorul trebuie să-l transmită explicit funcției (de obicei, prin primul parametru fix).

Funcții predefinite pentru manipularea listelor de parametri variabili (definite în `stdarg.h`):

- `va_start(va_list lparam, numele_ultimului_parametru_fix)`: **extrage în lista `lparam` parametrii variabili ai funcției**

- `va_arg(va_list lparam, tip_de_date)`: **furnizează valoarea curentă din lista parametrilor variabili `lparam` considerând-o ca fiind de tipul de date indicat, după care avansează automat la următorul element din lista `lparam`**

- `va_end(va_list lparam)`: **eliberează zona de memorie alocată listei** lparam

- `va_copy(va_list destinație, va_list sursă)`: **copiază în lista `destinație` lista `sursă` (atenție, o listă de parametrii variabili poate fi parcursă o singură dată!!!)**

Exemplu: funcție care calculează suma unui număr variabil de valori întregi, folosind parametrul fix n pentru a preciza numărul parametrilor variabili

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

//n = numarul parametrilor variabili
int suma(int n, ...)
{
    int x, s, i;
    va_list lparam;

    va_start(lparam, n);

    s = 0;
    for(i = 0; i < n; i++)
    {
        x = va_arg(lparam, int);
        s = s + x;
        //s = s + va_arg(lparam, int);
```

```
    }

    va_end(lparam);

    return s;
}

int main()
{
    printf("Suma cu 2 termeni: %d\n", suma(2, 10, 7));
    printf("Suma cu 3 termeni: %d\n", suma(3, 10, 7, -15));
    return 0;
}
```

Exemplu: funcție care calculează suma unui număr variabil de valori întregi, folosind valoarea 0 pentru a marca sfârșitul listei parametrilor variabili

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

//x = primul număr din sumă
int suma(int x, ...)
{
    int t, s, i;
    va_list lparam;

    va_start(lparam, x);

    s = x;
    do
    {
        t = va_arg(lparam, int);
        if(t != 0)
            s = s + t;
    }
    while(t != 0);

    va_end(lparam);

    return s;
}

int main()
{
    printf("Suma cu 2 termeni: %d\n", suma(10, 7, 0));
    printf("Suma cu 3 termeni: %d\n", suma(10, 7, -15, 20, 0));

    return 0;
}
```