

## 9. Pointeri și operații cu ei. Transferul prin adresă a parametrilor către funcții. Referințe. (Pointers. Operations using pointers. Parameters' transfer by address. References)

### 1. Obiective:

- Înțelegerea noțiunilor de *pointer* și *referință* și a modalității de definire și utilizare a lor;
- Familiarizarea cu operațiile cu pointeri;
- Scrierea și rularea de programe în care sunt folosiți pointeri, referințe și utilizarea lor în transferul argumentelor către funcții;

### 1'. Objectives:

- Understanding what is a *pointer* and a *reference*, how they can be defined and used;
- Getting used with the pointer operations;
- Writing and running programs that use pointers and references and that transfer them as function arguments;

### 2. Breviar teoretic

Un pointer(indicator) este o variabilă care are ca și valoare o adresă, deci un pointer reprezintă adresa de început a unei zone de memorie asociate unui obiect(variabilă, tablou sau o structură de date mai complexă).

Tipuri de pointeri :

- *pointeri de date*: conțin adresa unor variabile sau constante din memorie;
- *pointeri de funcții*: conțin adresa codului executabil din funcții;
- *pointeri generici sau pointeri void*: conțin adresa unui obiect oarecare.
- *pointeri smart*: permit eliberarea automata a memoriei prin garbage collection, introduși recent în C++1y/2z

Pointerii de date se declară astfel: *tip \*nume\_pointer;*

ceea-ce înseamnă că *nume\_pointer* este numele unei variabile-pointer, care va conține adresa unei zone de memorie care conține un obiect de tipul *tip*.

Pointerii de date pot fi asociați mai multor variabile la momente diferite de timp (reutilizați în program) și deci pot conține adrese diferite.

#### 2.1. Operatori specifici pentru pointeri

Operatorul de adresare (referențiere), *&*, asociat variabilei sau mai general obiectului *nume*, obține adresa acelei variabile sau obiect :

Exemplu:

```
char c, *pc;// spunem că pc este un pointer la variabila c
pc=&c;
```

Operatorul de indirectare, (dereferențiere) *\**: asociat unui pointer, permite accesul la conținutul locației de memorie a cărei adresă este conținută de pointer:

Exemplu:

```
int k=1, j=5, *p;
p=&k;           // asignează lui p adresa lui k
*p=2;          // valoarea variabilei k se schimbă din 1 în 2
```

Indirectarea este un mecanism puternic de acces la memorie.

#### 2.2. Operații cu pointeri

Cu pointerii se pot efectua operații de: *atribuire, comparare, adunare, scădere, incrementare, decrementare*. Se ține cont de tipul pointerilor și de faptul că adresele conținute de pointeri au valori numerice întregi fără semn.

**Atribuirea:** e permisă atribuirea valorii unui pointer la un alt pointer, dacă tipurile lor sunt identice sau dacă unul din pointeri este de tipul *void*, iar celălalt de tip oarecare.

**Compararea a doi pointeri:** se face cu operatorii relaționali, dar numai în cazul în care pointerii pointează pe obiecte de același tip.

### Operații de adunare/scădere și incrementare/decrementare

La un pointer nu pot fi adunate sau scăzute decât cantități întregi. Adunarea pointerilor nu este permisă.

Operațiile se efectuează relativ la tipul pointerului (int, float, char, etc.).

De exemplu:

Fie declarația: `tip *id_ptr;`

Operațiile: `id_ptr + n` și  
`id_ptr - n`

înseamnă adunarea/scăderea la conținutul variabilei `id_ptr` a valorii:  $n * \text{sizeof}(\text{tip})$ .

Analog se efectuează și operațiile de incrementare/decrementare, doar că  $n = 1$ .

Operația de incrementare/decrementare se poate aplica:

- asupra pointerului însuși (selecăm obiectul următor);
- asupra obiectului pe care îl pointează.

### Scăderea a doi pointeri

este permisă doar scăderea a doi pointeri la obiecte de același tip, rezultatul fiind o valoare care reprezintă diferența de adrese divizată la dimensiunea tipului de bază. (e recomandat ca aceste adrese să fie specifice unui tablou). Practic, prin diferență obținem numărul de elemente dintre cele două adrese.

Datorită rolului tipului pointerului la adunare și scădere, operanzii nu pot fi pointeri *void* sau pointeri spre funcții.

## 2.3. Referințe

Referințele, ca și pointerii, conțin adrese de memorie.

- Ele se declară cu ajutorul operatorului de adresare "&"
- se inițializează obligatoriu la declarare, cu o adresă (a unei variabile sau a unei constante) :  
`int i;`  
`int &r = i;`

Accesul la o variabilă prin intermediul unei referințe se face simplu, fără a mai folosi operatorul de indirectare ca în cazul pointerilor:

|  |   |
|--|---|
| <code>int i;</code>                              | <code>int i;</code>                           |
| <code>int *p;//declarare</code>                  | <code>int &amp;r = i;//definire (init)</code> |
| <code>...</code>                                 | <code>...</code>                              |
| <code>p = &amp;i;//definire prin asignare</code> |   |
| <code>*p = 100; // i = 100</code>                | <code>r = 1000; // i = 1000</code>            |

Referința conține tot timpul adresa aceleiași variabile, fiind de fapt o redenumire (alias) a variabilei.

Nici o referință nu poate avea valoarea NULL

Uzual, referințele se folosesc la transmiterea parametrilor prin referință, caz în care indirectarea este ascunsă și nu este necesară dereferențierea în funcție (utilizarea operatorului \*).

## 2.4. Apelul prin adresă utilizând parametri de tip pointeri și referință

În cadrul limbajului C/C++ transferul parametrilor către funcții este implicit efectuat prin valoare și constă în încărcarea valorilor parametrilor efectivi în zona de memorie a parametrilor formali. Dacă parametrul efectiv este o variabilă, orice operație efectuată în funcție asupra parametrului formal nu afectează variabila. Spunem că se lucrează cu copii ale parametrilor actuali, adică variabile locale pentru funcție. Acest lucru este o protecție utilă în cele mai multe cazuri.

Dacă vrem ca o funcție să *modifice o variabilă parametru*, atunci trebuie să transmitem funcției adresa variabilei, deci ca *argumente* se folosesc *adrese*, iar ca *parametri formali* se folosesc *pointeri* sau *referințe* unde se vor copia aceste adrese.

### 3. Exemple

Exemplul 1: declarare pointeri, utilizarea operatorilor de adresare și indirectare.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int i = 10, j=50;
        int *iptr;      // declarare pointer

        iptr = &i;      // definire prin asignare pointer
        printf("Adresa din iptr in Hexa: %x \n", iptr);
        printf("Adresa din iptr cu modif. p: %p \n", iptr);
        printf("Valoarea de la adresa data de pointer *iptr: %d\n", *iptr);

        *iptr = 25;      // se modifica valoarea din memorie
        printf("Noua valoare din memorie este: %d\n", i);

        iptr = &j;      // reasignare pointer
        printf("\nAdresa din iptr in Hexa: %x \n", iptr);
        printf("Valoarea de la adresa data de pointer *iptr: %d\n", *iptr);

        *iptr = 25;      // se modifica valoarea din memorie
        printf("Noua valoare din memorie este: %d\n", j);
        return 0;
}
```

Exemplul 2: pointeri si operatorul cast

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void baza2 (unsigned int);

int main(void){
    int n, *pn;
    float r, *pr;

    printf("\nIntroduceti un numar intreg: ");
    scanf("%d", &n);
    pn = &n;
    printf("\nReprezentarea in memorie a lui %d este in hexa %x si pe bytes: 0x%02x %02x %02x %02x\n", n,
        *((unsigned char *)pn+3),
        *((unsigned char *)pn+2),
        *((unsigned char *)pn+1),
        *((unsigned char *)pn));
    printf("\nBitii baza 2 : ");
    baza2(n);

    printf("\nIntroduceti un numar real: ");
    scanf("%f", &r);
}
```

```

pr = &r;
printf("\nReprezentarea in memorie a lui %f este in hexa %a si pe bytes: 0x%02x %02x %02x %02x\n", r,
r,
*((unsigned char *)pr+3),
*((unsigned char *)pr+2),
*((unsigned char *)pr+1),
*((unsigned char *)pr));
return 0;
} //main

void baza2 (unsigned int a){
    if(a!=0)baza2(a>>1);
    printf("%d ", a%2);
} //baza2

```

**Exemplul 3:** declarare și utilizare referințe.

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int i = 10, j=50;
    int &iref1=i;      // declarare referinta
    printf("Valoarea de la adresa data de referinta iref1: %d\n", iref1);

    iref1 = 25;        // se modifica valoarea din memorie
    printf("Noua valoare din memorie este: %d\n", i);

    int &iref2 = j;      // alta referinta
    printf("\nValoarea de la adresa data de referinta iref2: %d\n", iref2);

    iref2 = 25;        // se modifica valoarea din memorie
    printf("Noua valoare din memorie este: %d\n", j);
    return 0;
}

```

**Exemplul 4:** operații cu pointeri.

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

int main(void){
    int *pt_int;
    float *pt_float;
    int i = 10, j = 20;
    float x = 1.2345f, y = 32.14f;
    void *general; //pointer generic

    pt_int = &i;
    *pt_int += j;
    cout << "Valoarea lui i devine: " << *pt_int << "\n";

    general = pt_int;
    *(int *)general = 0;
    cout << "Ultima valoare pentru i este: " << i << "\n";
}

```

```

    pt_float = &x;
    y += 5.0f * (*pt_float);
    cout << "Valoarea lui y este: " << y << "\n";

    general = pt_float;
    *(float *)general = 1.1f;
    cout << "Ultima valoare pentru x este: " << x << "\n";
    return 0;
}

```

Exemplul 5: transmiterea parametrilor prin adresă.

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

void Swap1 (int x, int y);
void Swap2 (int *x, int *y);
void Swap3 (int &x, int &y);

int main (void){
    int i = 10, j = 20;

    Swap1(i, j);
    cout << "i = " << i << ", " << "j = " << j << '\n';

    Swap2(&i, &j);
    cout << "i = " << i << ", " << "j = " << j << '\n';

    Swap3(i, j);
    cout << "i = " << i << ", " << "j = " << j << '\n';
    return 0 ;
}

void Swap1 (int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}

void Swap2 (int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

void Swap3 (int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}

```

#### 4. Intrebări

- Ce este un pointer ?
- Câte tipuri de pointeri există în limbajul C/C++ ?
- Cum se declară și se inițializează un pointer ?
- Ce operator se folosește pentru a obține adresa unui obiect ?
- Ce operator se folosește pentru a afla valoarea memorată într-o locație de memorie dată prin adresă ?
- Ce este o referință ?
- Cum se declară o referință și cum se inițializează ?
- Care sunt asemănările între pointeri și referințe ?
- Care sunt diferențele între pointeri și referințe ?
- Ce operații se pot face asupra pointerilor ?
- Ce se înțelege prin transferul parametrilor prin adresă ?
- Cum se face transferul parametrilor prin adresă ?

#### 5. Probleme

1. Să se scrie un program C/C++ care citește elementele a doua tablouri unidimensionale de numere întregi și afișează produsul scalar al acestora. Se va folosi o funcție care preia elementele de la tastatură și o altă funcție, care calculează produsul scalar. Ambele vor utiliza pointeri. Citirea numărului de elemente ale tabloului și afișarea rezultatului se va face în funcția `main()`.
2. Să se scrie o aplicație C/C++ în care se generează aleator 20 de numere întregi cu valori mai mici decât 50 (Folositi `srand()`, `rand()` si operatorul `%`). Să se scrie o funcție care elimină din tabloul unidimensional creat numerele impare. Funcția va utiliza pointeri. Afișați în `main()` tabloul inițial și cel obținut după eliminarea elementelor impare.
3. Să se scrie un program C/C++ în care se citesc de la tastatură numere reale, ce vor fi stocate într-un tablou unidimensional. Să se scrie o funcție care copiază într-un alt tablou toate valorile din primul tablou, care sunt mai mari decât valoarea medie a numerelor preluate. Se vor folosi pointeri și se vor afișa în `main()` valorile din cele două tablouri.
4. Să se scrie un program care citește de la tastatură un șir de caractere, apoi elimină din șir caracterele care se repetă și afișează în final șirul obținut, folosind pointeri.
5. Să se scrie un program care citește de la tastatură două șiruri de caractere și afișează numărul de caractere prin care ele diferă (adică numărul de caractere care există în primul și nu există în al doilea + numărul de caractere care există în al doilea și nu există în primul). Folosiți pointeri pentru accesul la elementele tablourilor.
6. Să se scrie o aplicație C/C++ care citește de la tastatură un șir de caractere. Să se scrie o funcție care afișează caracterele ce compun șirul și numărul de apariții ale fiecăruia, folosind pointeri.

#### 5'. Homework

1. Write a C/C++ application that reads from the keyboard two one dimensional arrays of integers and displays the scalar product of the two arrays. Use a function that reads the elements from the keyboard and another function that calculates the scalar product, using pointers in both functions. Reading and displaying the arrays' elements should be done in the main function.
2. Write a C/C++ application that generates 20 random numbers, each smaller than 50 (use `srand()`, `rand()` and `%` operator). Write a function that eliminates the odd elements from the one dimensional array (using pointers). Display both the initial and the final array in the main function.

3. Write a C/C++ program that fills-up a float-type, one-dimensional array with values read from the keyboard. Write a function that copies into another array the values greater than the average of all elements from the array, by using pointers. Both arrays should be displayed in the *main()* function.
4. Write a C/C++ application that reads from the keyboard an array of characters and displays the string obtained by eliminating the characters that appear more than once in the original array using pointers.
5. Write a C/C++ application that reads from the keyboard two arrays of characters and displays the total number of individual characters (the number of characters that are in the first array and do not exist in the second one + the number of characters that are in the second array and do not exist in the first one). Use pointers for accessing the arrays of elements.
6. Write a C/C++ program that reads from the keyboard an array of characters. Write a function that displays the characters that are in the array and the number of times they appear. Use pointers.