9. Pointeri şi operaţii cu ei. Transferul prin adresă a parametrilor către funcţii. Referinţe. (Pointers. Operations using pointers. Parameters' transfer by address. References)

1. Obiective:
- Înţelegerea noţiunilor de *pointer* şi *referinţă* şi a modalităţii de definire şi utilizare a lor;
- Familiarizarea cu operaţiile cu pointeri;
- Scrierea şi rularea de programe în care sunt folosiţi pointeri, referinţe şi utilizarea lor în transferul argumentelor către funcţii;

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 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*: conţin adresa unor variabile sau constante din memorie;
- *pointeri de functii*: conţin adresa codului executabil din funcţii;
- *pointeri generici sau pointeri void*: conţin adresa unui obiect oarecare.
- *pointeri smart*: permit eliberarea automata a memoriei prin garbage collection, introduşi recent in 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 conţine adresa unei zone de memorie care conţine un obiect de tipul *tip*.

Pointerii de date pot fi asociaţi mai multor variabile la momente diferite de timp (reutilizaţi în program) şi deci pot conţine adrese diferite.

2.1. Operatori specifici pentru pointeri

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

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

Operatorul de indirectare, (dereferenţiere) *: asociat unui pointer, permite accesul la conţinutul locaţiei de memorie a cărei adresă este conţinută de pointer:

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

Indirectarea este un mecanism puternic de acces la memorie.

2.2. Operaţii cu pointeri

Cu pointerii se pot efectua operaţii de: *atribuire, comparare, adunare, scădere, incrementare, decrementare*. Se ţine cont de tipul pointerilor şi de faptul că adresele conţinute 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 relaţionali, dar numai în cazul în care pointerii pointează pe obiecte de acelaşi tip.

Operaţii de adunare/scădere şi incrementare/decrementare
La un pointer nu pot fi adunate sau scăzute decât cantităţi întregi. Adunarea pointerilor nu este permisă.

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

De exemplu:
Fie declaraţia: tip *id_ptr;

Operaţiile: id_ptr + n şi
id_ptr - n
înseamnă adunarea/scăderea la conţinutul variabilei id_ptr a valorii: *n * sizeof(tip).*

Analog se efectuează şi operaţiile de incrementare/decrementare, doar că n = 1.
Operaţia de incrementare/decrementare se poate aplica:
- asupra pointerului însuşi (selecăm obiectul următor);
- asupra obiectului pe care îl pointează.

Scăderea a doi pointeri
este permisă doar scăderea a doi pointeri la obiecte de acelaşi tip, rezultatul fiind o valoare care reprezintă diferenţa de adrese divizată la dimensiunea tipului de bază. (e recomandat ca aceste adrese sa fie specifice unui tablou). Practic, prin diferenţă obţinem numărul de elemente dintre cele două adrese.
Datorită rolului tipului pointerului la adunare şi scădere, operanzii nu pot fi pointeri *void* sau pointeri spre funcţii.

2.3. Referinţe
Referinţele, ca şi pointerii, conţin adrese de memorie.
- Ele se declară cu ajutorul operatorului de adresare "&"
- se iniţializează 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 referinţe se face simplu, fără a mai folosi operatorul de indirectare ca în cazul pointerilor:

```
int i;                                  int i;
int *p;//declarare                      int &r = i;//definire (init)
...                                             …
p = &i;//definire prin asignare
*p = 100; // i = 100                    r = 1000; // i = 1000
```

Referinţa conţine tot timpul adresa aceleiaşi variabile, fiind de fapt o redenumire (alias) a variabilei.
Nici o referinţă nu poate avea valoarea NULL

Uzual, referinţele se folosesc la transmiterea parametrilor prin referinţă, caz în care indirectarea este ascunsă şi nu este necesară dereferenţierea în funcţie (utilizarea operatorului *).

2.4. Apelul prin adresă utilizând parametri de tip pointeri şi referinţă
În cadrul limbajului C/C++ transferul parametrilor către funcţii este implicit efectuat prin valoare şi constă în încărcarea valorilor parametrilor efectivi în zona de memorie a parametrilor formali. Dacă parametrul efectiv este o variabilă, orice operaţie efectuată în funcţie asupra parametrului formal nu afectează variabila.

Spunem că se lucrează cu copii ale parametrilor actuali, adică variabile locale pentru funcţie. Acest lucru este o protecţie utilă în cele mai multe cazuri.
Dacă vrem ca o funcţie să *modifice o variabilă parametru*, atunci trebuie să transmitem funcţiei adresa variabilei, deci ca *argumente* se folosesc *adrese*, iar ca *parametri formali* se folosesc *pointeri* sau *referinţe* unde se vor copia aceste adrese.

3. Exemple

Exemplul 1: declarare pointeri, utilizarea operatorilor de adresare şi 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: %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:  %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", 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,
n,        *((unsigned char *)pn+3),
          *((unsigned char *)pn+2),
          *((unsigned char *)pn+1),
          *((unsigned char *)pn));
printf("\n Bitii 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 şi utilizare referinţe.

```
#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: operaţii 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 şi se iniţializează un pointer ?
- Ce operator se foloseşte pentru a obţine adresa unui obiect ?
- Ce operator se foloseşte pentru a afla valoarea memorată într-o locaţie de memorie dată prin adresă ?
- Ce este o referinţă ?
- Cum se declară o referinţă şi cum se iniţializează ?
- Care sunt asemănările între pointeri şi referinţe ?
- Care sunt diferenţele între pointeri şi referinţe ?
- Ce operaţii se pot face asupra pointerilor ?
- Ce se înţelege prin transferul parametrilor prin adresă ?
- Cum se face transferul parametrilor prin adresă ?

5. Probleme

1. Să se scrie un program C/C++ care citeşte elementele a doua tablouri unidimensionale de numere întregi şi afişează produsul scalar al acestora. Se va folosi o functie care preia elementele de la tastatură şi o altă funcţie, care calculează produsul scalar. Ambele vor utiliza pointeri. Citirea numărului de elemente ale tabloului şi afişarea rezultatului se va face în funcţia main().

2. Să se scrie o aplicaţie 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 funcţie care elimină din tabloul unidimensional creat numerele impare. Funcţia va utiliza pointeri. Afişati în main() tabloul iniţial şi cel obţinut 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 funcţie 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 şi se vor afişa în *main()* valorile din cele două tablouri.

4. Să se scrie un program care citeşte de la tastatură un şir de caractere, apoi elimină din şir caracterele care se repetă şi afişează în final şirul obţinut, folosind pointeri.

5. Să se scrie un program care citeşte de la tastatură două şiruri de caractere şi afişează numărul de caractere prin care ele diferă (adică numărul de caractere care există în primul şi nu există în al doilea + numărul de caractere care există în al doilea şi nu există în primul). Folosiţi pointeri pentru accesul la elementele tablourilor.

6. Să se scrie o aplicaţie C/C++ care citeşte de la tastatură un şir de caractere. Să se scrie o funcţie care afişează caracterele ce compun şirul şi numărul de apariţii 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.