Pointeri și referințe

În C++ există două modalități de lucra cu adrese de memorie: **pointeri** și **referințe**.

1. Pointeri

Pointerii sunt variabile care conțin adresa unei alte zone de memorie. Ei sunt utilizați pentru a utiliza date care sunt cunoscute prin adresa zonei de momorie unde sunt alocate. Sintaxa utilizată pentru declararea lor este:

tip *variabila_pointer;

Exemplu:

	Conținutul memoriei în urma acestor declarații va fi:

// declaratie de variabile
int i = 17, j = 3;
// declaratie pointer
int *p;
	Variabila p i j
	Conținut ? 17 3
	Adresă 2145 2146 2147 2148

Se observă că pointerul, la acest moment, nu este inițializat. Referirea prin intermediul pointerului neinițializat va genera o eroare la rularea programului. În lucrul cu pointeri se folosesc doi operatori unari:

- **&** : extragerea adresei unei variabile
- * : referirea conținutului zonei de memorie indicate de pointer (indirectare)

Exemplu:

// p ia adresa lui i
 	
p = &i;
	Conținutul memoriei va fi:
	
	Variabila p i j
	Conținut 2147 17 3
	Adresă 2145 2146 2147 2148

// modificarea continutul zonei
// de memorie pointate de p
 	
(*p) = 6;
	Conținutul memoriei va fi:
	
	Variabila p i j
	Conținut 2147 6 3
	Adresă 2145 2146 2147 2148

Un pointer poate fi refolosit, în sensul că poate conține adrese diferite la diferite momente de timp:

	Conținutul memoriei în urma acestor declarații va fi:

// modificare adresa
 	
p = &j;
	Variabila p i j
	Conținut 2148 6 3
	Adresă 2145 2146 2147 2148

Operațiile permise asupra pointerilor sunt următoarele:

- extragerea obiectului referit de către pointer folosind operatorul * sau operatorul [ ] ;
- extragerea adresei unui pointer folosind operatorul **&** (se va obține un pointer la pointer) ;
- atribuirea între doi pointeri care referă același tip de dată ;

- incrementarea/decrementarea (va muta pointerul înainte/înapoi cu un număr de bytes egal cu dimensiunea tipului referit) ;
- adunarea/scăderea cu o valoare întreagă (va muta pointerul înainte/înapoi cu un număr de bytes egal cu dimensiunea tipului referit înmultită cu valoarea întreagă) ;
- diferența a doi pointeri de același tip (se obține numărul de elemente de tipul respectiv ce încap între cei doi pointeri) ;
- compararea a doi pointeri ;
- conversia pointerilor (se realizează ca și pentru celelalte tipuri folosind operatorul de cast) .

2. Referințe

Referințele, ca și pointerii, sunt variabile care conțin adresa unei zone de memorie. Semantic, ele reprezintă aliasuri ale unor variabile existente. Referințele sunt legate de variabile la declarație și nu pot fi modificate pentru a referi alte zone de memorie. Sintaxa folosită pentru declararea unei referințe este următoarea:

tip & referinta = valoare;

Exemplu:

// declaratii de variabile
int i = 6, j = 3;
// declaratie referinta
int & r = j;
	Conținutul memoriei în urma acestor declarații va fi:
	
	Variabila r i j
	Conținut 2148 6 3
	Adresă 2145 2146 2147 2148

Sintaxa utilizată pentru manipularea pointerului este aceeași ca cea a variabilei de care este legată (indirectarea este realizată automat de către compilator). Toate modificările aplicate referinței se vor reflecta asupra variabilei referite.

Exemplu:

// modificarea variabilei
// prin referinta
 	
r = 7;
	Conținutul memoriei va fi:
	
	Variabila r i j
	Conținut 2148 6 7
	Adresă 2145 2146 2147 2148

// atribuirea are ca efect
// copierea continutului
// din i in j si nu
// modificarea adresei referintei
r = i;
	Conținutul memoriei va fi:
	
	Variabila r i j
	Conținut 2148 6 6
	Adresă 2145 2146 2147 2148

Spre deosebire de pointeri, referințele nu au operații speciale. Toți operatorii aplicați asupra referințelor sunt de fapt aplicați asupra variabilei referite. Chiar și extragerea adresei unei referințe va returna adresa variabilei referite.

Pentru exemplul prezentat, expresia **&r** va returna valoarea 2148 (operatorul de extragere de adresă se aplică de fapt asupra variabilei **j**).

Proprietățile cele mai importante ale referințelor sunt:

- referințele trebuie să fie inițializate la declarație (spre deosebire de pointeri, care pot fi inițializați în orice moment) ;
- după inițializare, referința nu poate fi modificată pentru a referi o altă zonă de memorie (pointerii pot fi modificați pentru a referi altă zonă) ;
- într-un program C++ valid nu există referințe nule .

Trimiterea parametrilor în funcții

Trimiterea parametrilor în funcții se poate face prin două mecanisme:

- **prin valoare**: valorile parametrilor sunt copiate pe stivă; modificările efectuate de funcție asupra parametrilor nu se vor reflecta în apelant.
- **prin adresă**: se copiază pe stivă adresele de memorie unde se află datele corespunzătoare parametrilor; modificările efectuate de funcție vor fi vizibile și în apelant.

Transferul prin adresă se poate realiza prin intermediul pointerilor sau referințelor. Recomandarea generală este să se folosească referințele datorită sintaxei mai simple și a faptului că permit evitarea unor probleme specifice pointerilor (pointeri nuli, pointeri către zone dezalocate, …).

Exemple de transmitere parametri:

Funcție	Apel

Prin valoare:
`void Inc(int i)`
`{`
`  i++;`
`}`
	`int i = 10;`
	`Inc(i);`
	`cout << i;`
	
	Rezultat: 10

Prin referinta:
`void IncReferinta(int &i)`
`{`
`  i++;`
`}`
	`int i = 10;`
	`IncReferinta(i);`
	`cout << i;`
	
	Rezultat: 11

Prin pointeri:
`void IncPointer(int *pi)`
`{`
`  (*pi)++;`
`}`
	`int i = 10;`
	`IncPointer(&i);`
	`cout << i;`
	
	Rezultat: 11

Pointeri și referințe constante

Cuvântul **const** poate fi folosit pentru a declara pointeri constanți sau pointeri la zone constante. Pointerii constanți sunt pointeri care nu-și pot modifica adresa referită. Sintaxa folosita este:

tip * const pointer_ct = adresa;

În cazul pointerilor constanți, inițializarea la declarare este obligatorie.

Exemplu:

```
// declarare si initializare pointer constant
int * const pConstant = &i;
```

```
// modificarea continutului este permisa
(*pConstant) = 5;

// modificarea adresei nu este permisa
pConstant = &j; // => eroare de compilare
```

Pointerii la zone de memorie constante sunt pointeri prin intermediul cărora nu se poate modifica conținutul zonei referite. Sintaxa de declarare a acestora este:

tip const * pointer_zona_ct;

Exemplu:

```
// declarare pointer la zona constanta
int const * pZonaCt;

// modificarea adresei referite este permisa
pZonaCt = &i;
pZonaCt = &j;

// modificarea continutului nu este permisa
(*pZonaCt) = 7;      // => eroare de compilare
```

Cele două forme pot fi folosite simultan pentru a declara pointeri constanți la o zonă de memorie constantă:

tip const * const pointer_ct_zona_ct;

În acest caz nu poate fi modificată nici adresa referită, nici conținutul acesteia.

Referințele sunt implicit echivalente cu un pointer constant (adresa referită nu poate fi modificată). În cazul lor, modificatorul **const** poate fi utilizat pentru a crea referințe prin intermediul cărora nu se pot efectua modificări asupra conținutului. Sintaxa utilizată este:

tip const& referinta_ct;

Comportamentul este echivalent cu al pointerilor constanți la zone de memorie constantă (referința va putea fi utilizată numai pentru citirea valorii referite).

3. Masive

Masivele sunt structuri de date omogene, cu un număr finit și conoscut de elemente ce ocupă un spațiu continuu de memorie. La declararea masivelor se precizează numele masivului, tipul elementelor, numărul de dimensiuni și numărul de elemente pentru fiecare dimensiune. Sintaxa de declarare este:

tip nume[dim1][dim2]…[dim_n] = {lista_initializare};

unde **dim1,…,dim_n** reprezintă numărul de elemente din fiecare dimensiune. Lista de inițializare este opțională. În cazul în care aceasta este prezentă, numărul de elemente pentru prima dimensiune poate lipsi (este dedus automat din lista de inițializare).

Exemple:

```
// vector de 10 elemente, fara initializare
int m1[10];

// vector de 3 elemente complet initializat
int m2[ ] = {1, 2, 3};

// vector de 5 elemente partial initializat
double m3[5] = {14.2, 15.1, 16.522};

// matrice de 2x2 elemente, fara initializare
int m4[2][2];

// matrice de 2x3 elemente cu initializare
int m5[ ][3] =
        {
                { 1, 2, 3},    // linia 1
                { 4, 5, 6},    // linia 2
        };
```

Referirea elementelor masivului se face prin utilizarea operatorului [ ]. Numerotarea elementelor pentru fiecare dimensiune se face de la **0** pana la **dim-1**. Intern, masivele sunt memorate într-un spațiu continuu de memorie; masivele multidimensionale sunt memorate linie după linie. Numele masivului este de fapt un pointer constant (nu poate fi modificat pentru a referi o altă zonă de memorie) care referă primul element din vector. La accesarea unui element, adresa acestuia este calculată pe baza adresei de început a masivului și a numărului de elemente pentru fiecare dimensiune.

Exemplu:

```
// vector cu 2 elemente
int v[2] = {9, 7};
```

Conținutul memoriei va fi:

Variabila	v		v[0]	v[1]
Conținut	2147		9	7
Adresă	2145	2146	2147	2148

```
// matrice de dimensiune 2x2
int m[2][2] = { {1, 2}, {3, 4} };
```

Conținutul memoriei va fi:

			Linia 1		Linia 2	
Variabila	m		m[0][0]	m[0][1]	m[1][0]	m[1][1]
Conținut	2147		1	2	3	4
Adresă	2145	2146	2147	2148	2149	2150

Datorită tratării similare, masivele se pot folosi ca pointeri constanți și pointerii se pot folosi utilizând aceeași sintaxă ca pentru masive. Operatorul **[n]**, care se poate folosi atât pentru masive cât și pentru vectori are ca efect extragerea conținutului de la adresa aflată la **n** elemente distanță de adresa indicată de pointer/masiv. El este echivalent cu expresia ***(v + n)**.

Exemple:

```
// 1. declaratie vector cu 4 elemente
int v[ ] = {1, 2, 3, 4}, i;
```

```
// 2. declaratie pointer
int *p;
```

```
// 3. initializare pointer cu adresa masivului
p = v;
```

```
// 4. extragerea celui de-al treilea element folosind masivul
i = v[2];
```

```
// 5. extragerea celui de-al treilea element
// folosind masivul ca pointer
i = *(v+2);
```

```
// 6. folosirea pointerului pentru a extrage
// cel de-al treilea element din vector
i = *(p+2);
```

```
// 7. folosirea operatorului [ ] pe pointer
// pentru a extrage elementul
i = p[2];
```

4. Alocarea dinamică a memoriei

Alocarea dinamică a memoriei în C++ se face utilizand operatorii **new** și **delete**. Sintaxa utilizată este:

Alocare element:	**pointer = new tip(initializare);**
Alocare masiv:	**pointer = new tip[nr_elemente];**
Dezalocare element:	**delete pointer;**
Dezalocare masiv:	**delete [ ] pointer;**

Câteva precizări referitoare la operatorii **new** și **delete**:

- la alocarea unui element, partea de inițializare poate lipsi ;
- în cazul în care alocarea nu se poate realiza, operatorul întoarce valoarea NULL ;
- operatorul **delete** aplicat pe un pointer nul nu produce niciun rezultat .

Exemple de utilizare:

```
int *pi, *pj, *pv;

// alocare intreg fara initializare
pi = new int;

// alocare intreg cu initializare
// este echivalent cu:
// pj = new int; (*pj) = 7;
pj = new int (7);

// alocare masiv cu 3 elemente
pv = new int [3];

// dezalocare memorie
delete pi;
delete pj;

// dezalocare masiv
delete [ ] pv;
```

Operatorul **new** poate aloca numai masive unidimensionale. Pentru alocarea masivelor cu mai multe dimensiuni se vor utiliza vectorii de pointeri.

Exemplu de alocare pentru matrice:

```
// dimensiunile matricei
int m = 3, n = 4;

// declararea matricei ca pointer
int **mat;

// alocarea vectorului de pointeri
mat = new int * [m];

// alocarea vectorilor pentru fiecare linie
for (int i = 0; i < m; i++)
      mat[i] = new int [n];
```

Elementele matricei astfel alocate pot fi accesate folosind sintaxa obișnuită: mat[linie][coloana].

Dezalocarea se va face urmând succesiunea pașilor în ordine inversă:

```
// dezalocare vectori pentru fiecare linie
for (int i = 0; i < m; i++)
      delete [ ] mat[i];

// dezalocare vector de pointeri
delete [ ] mat;
```

5. Pointeri la funcții

Numele unei funcții reprezintă adresa de memorie la care începe funcția. Numele funcției este, de fapt, un pointer la funcție. Se poate stabili o corespondență între variabile şi funcții prin intermediul pointerilor la funcții. Ca şi variabilele, aceşti pointeri:

- pot primi ca valori funcții ;
- pot fi transmişi ca parametrii altor funcții ;
- pot fi întorşi ca rezultate de către funcții .

La declararea unui pointer către o funcție trebuie precizate toate informațiile despre funcție, adică:

- tipul funcției ,
- numărul de parametri ,
- tipul parametrilor .

Aceasta ne va permite să apelăm funcția prin intermediul pointerului.

Declararea unui pointer la o funcție se face prin sintaxa:

tip_date_returnat (*nume_pointer)(listă_parametri_formali);

Dacă nu s-ar folosi paranteze care să încadreze identificatorul **nume_pointer**, ar fi vorba despre o funcție care întoarce ca rezultat un pointer.

Apelul unei funcții prin intermediul unui pointer are forma:

(*nume_pointer)(listă_parametri_actuali);

Este permis şi apelul fără indirectare:

nume_pointer (listă_parametri_actuali);

Este posibilă construirea unui tablou de pointeri la funcții; apelarea funcțiilor în acest caz se va face prin referirea la componentele tabloului. De exemplu, inițializarea unui tablou cu pointeri cu funcțiile matematice uzuale se face prin sintaxa:

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

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

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

Numele unei funcții, fiind un pointer către funcție, poate fi folosit ca parametru în apeluri de funcții. În acest, mod putem transmite în lista de parametri a unei funcții numele altei funcții.

Exemplul 1.

```
#include <iostream>
using namespace std;

int adunare(int a, int b)
{
	return a + b;
}

int scadere(int a, int b)
{
	return a - b;
}

// functie care primeste ca parametru un pointer la functie
int aplica_operatie(int a, int b, int (*pFunctie)(int, int) )
{
	// apel functie prin intermediul pointerului primit ca parametru
	return (*pFunctie)(a,b);
}

void main()
{
	// declarare si citire de variabile
	int a, b;
	cout << "a="; cin >> a;
	cout << ""b="; cin >> b;

	// declarare 'pf' ca pointer la functie ce primeste doi intregi ca parametri si intoarce un intreg
	int (*pf)(int, int);

	// incarcare pointer la functie
	pf = adunare;

	// apel de functie prin pointer
	cout << "Suma : " << (*pf)(a,b) << endl;

	// trimitera pointerului ca parametru
	cout << "Suma : " << aplica_operatie(a,b, pf) << endl;

	// folosirea directa a numelui functiei pentru trimiterea parametrului de tip pointer la functie
	cout << "Diferenta : " << aplica_operatie(a,b,scadere) << endl;
}
```

Exemplul 2. O funcție care calculează integrala definită:

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

poate avea prototipul:

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

Problemă. Definiți o funcție pentru aproximarea unei integrale definite prin metoda trapezelor, cu un

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

număr fixat n de puncte de diviziune:

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

```
#include<iostream>
#include<math.h>
using namespace std;
double (*tabfun[ ])(double) = {sin, cos, tan, exp};
double trapez(double,double,int,double(*)(double));
double a=0.0, b=1.0, eps=1E-6;
int N=10;

void main(void)
{
        int poz, n=N;
        double In, I2n, vabs;
        cout << "Pentru ce functie se calculeaza integrala ?"<<endl;
        cout << "( sin = 1 ; cos = 2 ; tg = 3 ; exp = 4 ) : ";
        cin >> poz;
        In = trapez(a,b,n,(*tabfun[poz-1]));
        do {
                n* = 2;
                I2n = trapez(a,b,n,(*tabfun[poz-1]));
                if((vabs = In-I2n) < 0)
                        vabs = -vabs;
                In = I2n;
        } while(vabs > eps);
        cout << "Valoarea integralei este : " << I2n << endl;
}

double trapez(double a,double b,int n,double(*f)(double))
{
        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;
}
```