Pointeri si referinte

In C++ exista doua modalitati de lucra cu adrese de memomorie: pointeri si referinte.

Pointeri

Pointerii sunt variabile care contin adresa unei alte zone de memorie. Ei sunt utilizati pentru a date care sunt cunoscute prin adresa zonei de momorie unde sunt alocate.

Sintaxa utilizata pentru declararea lor este:

```
tip *variabila_pointer;
```

Exemplu:

```
// declaratie variabile
int i = 17, j = 3;
// declaratie pointer
int *p;
```

Continutul memoriei in urma acestor declaratii va fi:

Variabila	p		i	j
Continut	?		17	3
Adresa	2145	2146	2147	2148

Se observa ca pointerul la acest moment nu este initializat. Referirea prin intermediul pointerului neinitializat va genera o eroare la rularea programului.

In lucrul cu pointeri se folosesc doi operatori unari:

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

Exemplu:

<table>
<tr><td><pre>// p ia adresa lui i
p = &i;</pre></td><td>

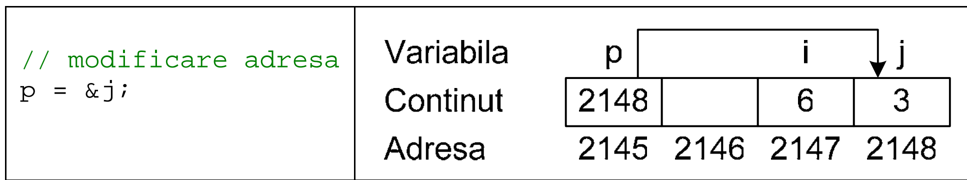
Variabila	p		i	j
Continut	2147		17	3
Adresa	2145	2146	2147	2148

</td></tr>
<tr><td><pre>// modificarea
// continutul zonei
// de memorie
// pointate de p
(*p) = 6;</pre></td><td>

Variabila	p		i	j
Continut	2147		6	3
Adresa	2145	2146	2147	2148

</td></tr>
</table>

Un pointer poate fi refolosit, in sensul ca poate contine adrese diferite la diferite momente de timp:



Operatiile permise asupra pointerilor sunt urmatoarele:

- extragerea obiectului referit de catre pointer folosind operatorul * sau operatorul [] (prezentat in sectiunea **Masive**)
- extragerea adresei unui pointer folosind operatorul **&** (se va obtine un pointer la pointer)
- atribuirea intre doi pointeri care refera acelasi tip de data
- incrementarea/decrementarea (va muta pointerul inainte/inapoi cu un numar de bytes egal cu dimensiunea tipului referit)
- adunarea/scaderea cu o valoare intreaga (va muta pointerul inainte/inapoi cu un numar de bytes egal cu dimensiunea tipului referit inmultita cu valoarea intreaga)
- diferenta a doi pointeri de acelasi tip (se obtine numarul de elemente de tipul respectiv ce incap intre cei doi pointeri)
- compararea a doi pointeri
- conversia pointerilor (se realizeaza ca si pentru celelalte tipui folosind operatorul de cast)

Referinte

Referintele, ca si pointerii, sunt variabile care contin adresa unei zone de memorie. Semantic, ele reprezinta aliasuri ale unor variabile existente.

Referintele sunt legate de variabile la declaratie si nu pot fi modificate pentru a referi alte zone de memorie. Sintaxa folosita pentru declararea unei referinte este:

```
Tip & referinta = valoare;
```

Exemplu:

Sintaxa utilizata pentru manipularea pointerului este aceeasi cea a variabilei ce care este legata (indirectarea este realizata automat de catre compilator). Toate modificarile aplicate referintei se vor reflecta asupra variabilei referite.

Exemple:

Cod	Memorie

`// modificarea variabilei`
`// prin referinta`
`r = 7;`
	Variabila: p, i, j
	Continut: 2148, , 6, 3
	Adresa: 2145, 2146, 2147, 2148

`// atribuirea are ca efect`
`// copierea continutului`
`// din i in j si nu`
`// modificarea adresei referintei`
`r = i;`
	Variabila: r, i, j
	Continut: 2148, , 6, 6
	Adresa: 2145, 2146, 2147, 2148

Spre deosebire de pointeri, referintele nu au operatii speciale. Toti operatorii aplicati asupra referintelor sunt de fapt aplicati asupra variabilei referite. Chiar si extragerea adresei unei referinte va returna adresavariabilei referite. Pentru exemplul prezentat, expresia **&r** va returna valoarea 2148 (operatorul de extragere de adresa se aplica de fapt asupra variabilei **j**).

Proprietatile cele mai importante ale referintelor sunt:

- referintele trebuie sa fie initializate la declaratie (spre deosebire de pointeri care pot fi initializati in orice moment);
- dupa initializare, referinta nu poate fi modificata pentru a referi o alta zona de memorie (pointerii pot fi modificati pentru a referi alta zona)
- intr-un program C++ valid nu exista referinte nule

Referintele utilizate in principal pentru transmiterea parametrilor in functii.

Trimiterea parametrilor in functii

Trimiterea parametrilor in functii se poate face prin doua mecanisme:

- **prin valoare**: valorile parametrilor sunt copiate pe stiva; modificarile efectuate de functie asupra parametrilor nu se vor reflecta in apelant
- **prin adresa**: se copiaza pe stiva adresele de memorie unde se afla datele corespunzatoare parametrilor; modificarile efectuate de functie vor fi vizibile si in apelant

Transferul prin adresa se poate realiza prin intermediul pointerilor sau referintelor. Recomandarea generala este sa sa foloseasca referintele datorita sintaxei mai simple si a faptului ca permit evitarea unor probleme specifice pointerilor (pointeri nuli, pointeri catre zone dezalocate, …).

Exemple de transmitere parametri:

Functie	**Apel**

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

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

<table>
<tr><td><pre>   i++;
}</pre></td><td><pre>
Rezultat: 11</pre></td></tr>
<tr><td>Prin pointeri:<pre>void IncPointer(int *pi)
{
      (*pi)++;
}</pre></td><td><pre>int i = 10;
IncPointer(&i);
cout << i;

Rezultat: 11</pre></td></tr>
</table>

Pointeri si referinte constante

Modificatorul **const** poate fi folosit pentru a declara pointeri constanti sau pointeri la zone constante.

Pointerii constanti sunt pointeri care nu-si pot modifica adresa referita. Sintaxa folosita este:

```
tip * const pointer_ct = adresa;
```

In cazul pointerilor constanti, initializarea 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 momorie constante sunt pointeri prin intermediul carora nu se poate modifica continutul zonei referite.

Sintaxa de declarare 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 doua forme pot fi folosite simultan pentru a declara pointeri constanti la zone de memorie constanta:

```
tip const * const pointer_ct_zona_ct;
```

In acest caz nu poate fi modificata nici adresa referita, nici constinutul acesteia.

Referintele sunt implicit echivalente cu un pointer constant (adresa referita nu poate fi modificata). In cazul lor, modificatorul **const** poate fi utilizat pentru a crea referinte prin intermediul carora nu se pot efectua modificari asupra continutului. Sintaxa utilizata este:

```
tip const& referinta_ct;
```

Comportamentul este echivalent cu al pointerilor constanti la zone de memorie constanta (referinta va putea fi utilizata numai pentru citirea valorii referite).

Masive

Masivele sunt structuri de date omogene cu un numar finit si conscut de elemente ce ocupa un spatiu continuu de memorie.

La declararea masivelor se precizeaza numele masivului, tipul elementelor, numarul de dimensiuni si numarul de elemente pentru fiecare dimensiune.

Sintaxa de declarare este:

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

unde `dim1…dim_n` reprezinta numarul de elemente din fiecare dimensiune.

Lista de initializare este optionala. In cazul in care aceasta este prezenta, numarul de elementre pentru prima dimensiune poate lipsi (este dedus automat din lista de initializare).

Exemple:

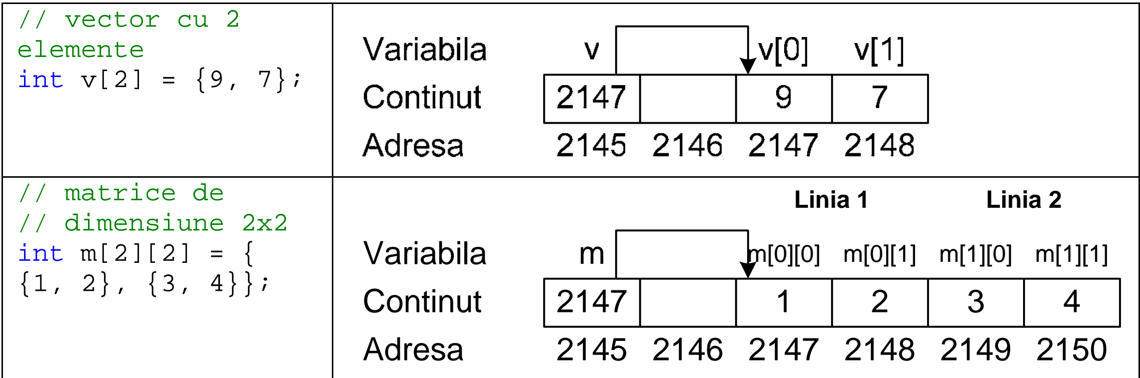
```
// vector de 10 elemente, fara initializare
int m1[10];

// 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 intr-un spatiu continuu de memorie; masivele multidimensionale sunt memorate in linie dupa linie. Numele masivului este de fapt

un pointer constant (nu poate fi modificat pentru a referi o alta zona de memorie) care refera primul element din vector. La accesarea unui element, adresa acestuia este calculata pe baza adresei de inceput a masivului si a numarului de elemente pentru fiecare dimensiune.

Exemplu:



Cod	Reprezentare in memorie
`// vector cu 2 elemente` `int v[2] = {9, 7};`	Variabila: v, v[0], v[1] Continut: 2147, , 9, 7 Adresa: 2145, 2146, 2147, 2148

`// matrice de`
`// dimensiune 2x2`
`int m[2][2] = { {1, 2}, {3, 4}};`
	Linia 1, Linia 2
	Variabila: m, m[0][0], m[0][1], m[1][0], m[1][1]
	Continut: 2147, , 1, 2, 3, 4
	Adresa: 2145, 2146, 2147, 2148, 2149, 2150

Datorita tratarii similare, masivele se pot folosi ca pointeri constanti si pointerii se pot folosi utilizand aceeasi sintaxa ca pentru masive.

Operatorul **[n]**, care se poate folosi atat pentru masive cat si pentru vectori are ca efect extragerea continutului de la adresa aflata la **n** elemente distanta de adresa indicata de ponter/masiv. El este echivalent cu expresia ***(v + n)**.

Exemple:

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

// declaratie pointer
int *p;

// initializare pointer cu adresa masivului
p = v;

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

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

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

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

Alocarea dinamica a memoriei

Alocarea dinamica a memoriei in C++ se face utilizand operatorii **new** si **delete**.

Sintaxa utilizata este:

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

Cateva precizari referitoare la operatorii **new** si **delete**:

- la alocarea unui element, partea de initializare poate lipsi
- in cazul in care alocarea nu se poate realiza, operatorul intoarce valoarea NULL
- operatorul **delete** aplicat pe un pointer nul nu produce nici un 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 massive unidimensionale. Pentru alocarea masivelor cu mai multe dimensiuni (care sa poata fi accesat cu sintaxa obisnuita se vor utilize 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 obisnuita: mat[linie][coloana].

Dezalocarea se va face urmand succesiunea pasilor in ordine inversa:
```
	// dezalocare vectori pentru fiecare linie
	for (int i = 0; i < m; i++)
		delete [] mat[i];

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

Pointeri la functii

Pointerii la functii variabile care contin adresele de inceput ale unei functii. Ei sunt utlizati pentru implementarea algoritmilor cu un grad mare de generalitate.

Declararea pointerilor la functii se face dupa modelul oferit de prototipul functiilor ce vor fi referite prin intermediul pointerului:

```
tip_returnat (*nume_pointer)(tip_parametri);
```

unde

- **`tip_returnat`** – tipul de data returnat de functie
- **`nume_pointer`** – numele pointerului declarat
- **`tip_parametri`** – lista de parametri din prototipul functiei

Incarcarea pointerului se face prin atribuire cu numele functiei:

```
nume_pointer = nume_functie;
```

Apelul functiei prin intermediul pointerului se face utilizand sintaxa:

```
(*nume_pointer)(lista_valori_param);
```

Exemplu de utilizare:

```
#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 variabile
        int a, b;
        cout << "a=";
        cin >> a;
        cout << "b=";
        cin >> b;

        // declarare 'pf' ca pointer la functie
        // care 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;
}
```

Aplicatie

Se considera aplicatia din capitolul anterior si se cer urmatoarele:

- sa se scrie functia pentru calcularea mediei unui student
- sa se citeasca de la tastatura numarul de studenti dintr-o grupa, sa se aloce dinamic spatiul necesar pentru memorarea acestora si sa se citeasca de la tastatura datele corespunzatoare
- sa se sorteze studentii alfabetic si dupa medie si sa se afiseze cele doua liste

Rezolvare

Pentru functia de calcul a mediei unui student avem doua variante:

a) folosind transferul prin pointeri:

```
void CalculMedie(Student* pStudent)
{
        // initializam media
        (*pStudent).Media = 0;

        // calculam suma notelor
        for (int i = 0; i < (*pStudent).NumarNote; i++)
                (*pStudent).Media += (*pStudent).Note[i];

        // calculam media
        (*pStudent).Media = (*pStudent).Media / (*pStudent).NumarNote;
}
```

b) folosind transferul prin referinte

```
void CalculMedie(Student& student)
{
	// initializam media
	student.Media = 0;

	// calculam suma notelor
	for (int i = 0; i < student.NumarNote; i++)
		student.Media += student.Note[i];

	// calculam media
	student.Media = student.Media / student.NumarNote;
}
```

In continuare vom folosi a doua varianta datorita simplicitatii si lizibilitatii codului.

Alocarea memoriei se va face folosind operatorul **new** in varianta a doua (pentru masive):

```
	int n; // numarul de studenti
	cout << "Numarul de studenti:";
	cin >> n;

	// declarare si alocare masiv
	Student *grupa = new Student[n];
```

Citirea datelor si calculul mediilor se va face folosind functiile create anterior:

```
	// citire date si calcul medie
	for (int i = 0; i < n; i++)
	{
		// elementele masivului alocat
		// for fi accesate folosind
		// operatorul [] asupra pointerului

		// citire date student
		grupa[i] = CitireStudent();

		// calcul medie folosind varianta
		// de transfer prin referinta
		CalculMedie(grupa[i]);
	}
```

Pentru sortarea listei in functie de doua criterii vom construi doua functii pentru compararea a doi studenti in functie de nume/medie:

```
bool ComparaNume(Student const & s1, Student const & s2)
{
	return strcmp(s1.Nume, s2.Nume) > 0;
}

bool ComparaMedie(Student const & s1, Student const & s2)
{
	return s1.Media > s2.Media;
}
```

Functia de sortare va primi un pointer la functia de comparare a doi studenti:

```
void SortareLista(Student * studenti, int n, bool (*pf)(Student const & s1,
Student const & s2))
```

```
{
	// sortare prin metoda bulelor

	bool modificat;
	do
	{
		modificat = false;

		// parcurgem grupa si interschimbam
		// elementele daca este cazul
		for(int i = 0; i < n-1; i++)
			if ((*pf)(studenti[i],studenti[i+1]) == true)
			{
				Student s = studenti[i];
				studenti[i] = studenti[i+1];
				studenti[i+1] = s;

				modificat = true;
			}

		// continuam pana cand nu mai
		// exista interschimburi de efectuat
	} while (modificat);
}
```

Apelul functiei de sortare si afisarea listelor:

```
	// sortare lista alfabetic
	SortareLista(grupa, n, ComparaNume);

	// afisare lista
	cout << "Lista ordonata alfabetic" << endl;
	for (int i = 0; i < n; i++)
		AfisareStudent(grupa[i]);

	// sortare lista dupa medii
	SortareLista(grupa, n, ComparaMedie);

	// afisare lista
	cout << "Lista ordonata in functie de medie" << endl;
	for (int i = 0; i < n; i++)
		AfisareStudent(grupa[i]);
```

Codul sursa complet:

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

// numarul maxim de note pentru un student
const int DIM_MAX_NOTE = 20;

// dimensiunea maxima a numelui
const int DIM_MAX_NUME = 20;

// declaratie structura
struct Student
{
	// numele studentului
	char Nume[DIM_MAX_NUME];

	// notele obtinute
	int NumarNote;
	int Note[DIM_MAX_NOTE];

	// media notelor
```

```
	float Media;
};

Student CitireStudent()
{
	// declarare variabila de tip structura
	Student stud;

	// citire nume
	// se foloseste functia getline pentru
	// a se putea introduce un nume care
	// contine spatii
	cout << "Nume:";
	cin >> ws; // elimina eventualele spatii
	cin.getline(stud.Nume, DIM_MAX_NUME);

	// citire numar note
	cout << "Numar note:";
	cin >> stud.NumarNote;

	// citire note
	for (int i = 0; i < stud.NumarNote; i++)
	{
		cout << "Nota " << i+1 << ": ";
		cin >> stud.Note[i];
	}

	// initializare medie
	stud.Media = 0;

	// intoarcere rezultat
	return stud;
}


void AfisareStudent(Student student)
{
	// afisare nume aliniat la stanga
	cout << setw(DIM_MAX_NUME) << setiosflags(ios::left) << student.Nume;

	// afisare medie (daca a fost calculata)
	if (student.Media > 0)
		// se folosesc manipulatorii fixed, showpoint
		// si 'setprecision' pentru a forta afisarea
		// mediei cu 2 zecimale
		cout << fixed << showpoint
			<< setprecision(2) << student.Media << " ";

	// afisare note
	cout << "Note:";
	for (int i = 0; i < student.NumarNote; i++)
		cout << " " << student.Note[i];
	cout << endl;
}

void CalculMedie(Student& student)
{
	// initializam media
	student.Media = 0;

	// calculam suma notelor
	for (int i = 0; i < student.NumarNote; i++)
		student.Media += student.Note[i];

	// calculam media
	student.Media = student.Media / student.NumarNote;
}
```

```
bool ComparaNume(Student const & s1, Student const & s2)
{
	return strcmp(s1.Nume, s2.Nume) > 0;
}

bool ComparaMedie(Student const & s1, Student const & s2)
{
	return s1.Media > s2.Media;
}

void SortareLista(Student * studenti, int n, bool (*pf)(Student const & s1,
Student const & s2))
{
	// sortare prin metoda bulelor

	bool modificat;
	do
	{
		modificat = false;

		// parcurgem grupa si interschimbam
		// elementele daca este cazul
		for(int i = 0; i < n-1; i++)
			if ((*pf)(studenti[i],studenti[i+1]) == true)
			{
				Student s = studenti[i];
				studenti[i] = studenti[i+1];
				studenti[i+1] = s;

				modificat = true;
			}

		// continuam pana cand nu mai
		// exista interschimburi de efectuat
	} while (modificat);
}

void main()
{
	int n; // numarul de studenti
	cout << "Numarul de studenti:";
	cin >> n;

	// declarare si alocare masiv
	Student *grupa = new Student[n];

	// citire date si calcul medie
	for (int i = 0; i < n; i++)
	{
		// elementele masivului alocat
		// for fi accesate folosind
		// operatorul [] asupra pointerului

		// citire date student
		grupa[i] = CitireStudent();

		// calcul medie folosind varianta
		// de transfer prin referinta
		CalculMedie(grupa[i]);
	}

	// sortare lista alfabetic
	SortareLista(grupa, n, ComparaNume);

	// afisare lista
	cout << "Lista ordonata alfabetic" << endl;
	for (int i = 0; i < n; i++)
```

```
                AfisareStudent(grupa[i]);

        // sortare lista dupa medii
        SortareLista(grupa, n, ComparaMedie);

        // afisare lista
        cout << "Lista ordonata in functie de medie" << endl;
        for (int i = 0; i < n; i++)
                AfisareStudent(grupa[i]);
}
```