MINISTERUL EDUCAŢIEI NAŢIONALE

Mariana Miloşescu

Informatică

Profilul real

Specializarea:
matematică-informatică, intensiv informatică

Manual pentru clasa a X - a

EDITURA DIDACTICĂ ŞI PEDAGOGICĂ S.A.

Cuprins

1. Implementarea structurilor de date

1.1. Datele prelucrate de algoritmi

Ştiți deja că orice rezolvare de problemă începe prin definirea datelor, continuă cu prelucrarea lor (de exemplu, atribuirea de valori sau efectuarea unor calcule numerice) şi se termină fie cu afişarea valorii lor, fie cu stocarea lor pe un mediu de memorare, în vederea unei prelucrări ulterioare. Mai ştiți că:

Datele sunt şiruri de biți care sunt prelucrate de calculator.

Data este o resursă la dispoziția programatorului. Orice limbaj de programare permite folosirea mai multor **tipuri de date**. Indiferent de tipul de date ales, reprezentarea sa în memoria calculatorului (fie internă, fie externă) se face printr-un şir de biți. Pentru a realiza această reprezentare, sunt implementați **algoritmi de codificare** ce asigură corespondența dintre tipul de dată şi şirul de biți, atât la scrierea datelor, cât şi la citirea lor. Tipul de dată ales de către programator influențează calitatea programului, deoarece el determină dimensiunea zonei de memorie alocate, algoritmul de codificare şi operatorii admişi pentru prelucrare.

Datele pot fi clasificate folosind mai multe criterii:

criteriul naturii (tipului) datelor

Date de tip întreg
În limbajul C++ sunt implementate mai multe subtipuri *întregi* de date:
- → Tipuri de date întregi pozitive: **unsigned char** (1 octet), **unsigned int** (2 octeți), **unsigned long** (4 octeți);
- → Tipuri de date întregi cu semn: **char** (1 octet), **int** (2 octeți), **long** (4 octeți);

Date de tip real
În limbajul C++ sunt implementate mai multe subtipuri *reale* de date: **float** (4 octeți), **double** (8 octeți), **long double** (10 octeți).

Date de tip logic (boolean).
În limbajul C++ nu este implementat tipul *logic*; pentru evaluarea unei operații logice se folosesc valorile numerice **1** sau **diferit de 0** (**true**) şi **0** (**false**).

Date de tip caracter
În limbajul C++ este implementat tipul **char**.

Tipul datelor determină modul în care sunt reprezentate datele în memoria internă şi operatorii permişi pentru prelucrarea acestor date. Pentru fiecare tip de dată trebuie implementați algoritmi pentru încărcarea valorii datei în zona de memorie, algoritmi pentru adresarea zonei de memorie alocate, algoritmi pentru extragerea valorii datei din zona de memorie alocată etc. De exemplu, tipurile *unsigned char* şi *unsigned int* se folosesc amândouă pentru reprezentarea numerelor întregi pozitive. Numai că, tipul *unsigned char* se memorează într-un octet şi permite memorarea unor valori pozitive cuprinse între 0 şi

255 (255=2^8-1), iar tipul *unsigned int* se memorează în doi octeți (cea mai frecventă implementare a limbajului C++ pe microcalculatoare) și permite memorarea unor valori pozitive cuprinse între 0 și 65535 (65535=2^{16}-1). Tipul *int* se reprezintă pe 2 octeți, prin mărime și semn (folosindu-se primul bit pentru semn, iar următorii 15 biți pentru reprezentarea în binar a numărului). Acest tip poate fi folosit pentru reprezentarea numerelor întregi cu semn care au o valoare cuprinsă în domeniul -32768 .. .32767 (32767=2^{15}-1). Tipul *float* se reprezintă pe 4 octeți în virgulă mobilă, prin reprezentarea exponentului și a mantisei (folosindu-se primul bit pentru semnul numărului, următorul bit pentru semnul exponentului, 7 biți pentru exponent și 23 de biți pentru mantisă). Acest tip poate fi folosit pentru reprezentarea numerelor reale cu 7 cifre semnificative, în domeniul -10^{38} ... 10^{38}.

Se poate testa lungimea în octeți a ficărui tip de dată folosind operatorul sizeof(). Acesta furnizează numărul de octeți folosiți pentru memorarea unei date, precizată printr-o expresie sau prin tipul datei.

Sintaxa operatorului este: **sizeof(<tip>)**, unde *<tip>* este tipul datei care se testează, sau **sizeof(<expresie>)**, unde *<expresie>* este o expresie care se evaluează.

Exemple:

→ operatorul **sizeof(int)** va furniza rezultatul 2, deoarece reprezentarea acestui tip de dată se face pe 2 octeți.

→ considerând declarațiile **int a; float b;** – operatorul **sizeof(a+b);** va furniza rezultatul 4, deoarece reprezentarea rezultatului obținut în urma evaluării expresiei necesită 4 octeți.

Pe lângă aceste tipuri de date, în limbajul C++ mai sunt implementate:

→ **tipul pointer** și
→ **tipul referință**.

criteriul compunerii

Date simple sau **date elementare**.
Sunt date independente unele de altele, din punct de vedere al reprezentării lor în memorie: localizarea unei date pe suportul de memorare nu se face în funcție de locația unei alte date pe suport.

Date compuse sau **structuri de date**.
Sunt colecții de date între care există anumite relații (**relații structurale**). Fiecare componentă a structurii are o anumită poziție în cadrul structurii. Pentru fiecare tip de structură de date, în limbajul de programare trebuie să fie definiți algoritmi de localizare a componentelor în cadrul structurii de date. Între componentele structurii există și legături de conținut, adică întregul ansamblu de date din colecție poate caracteriza un obiect, o persoană, un fenomen, un proces.

În limbajul C++ ați studiat structurile de date:

→ **tabloul de memorie** și
→ **fișierul**.

Tabloul de memorie este o structură de date omogenă, internă și temporară, iar fișierul este structură de date omogenă, externă și permanentă. Pentru rezolvarea unor probleme nu sunt suficiente numai aceste structuri de date. Limbajul C++ mai permite și folosirea următoarelor structuri fizice de date:

→ **șirul de caractere** și
→ **înregistrarea**.

criteriul variabilității

Constante
Sunt date a căror valoare nu se modifică în timpul execuției programului. Într-un program pot fi folosite **constante simbolice**. Acestea sunt constante care au fost puse în corespondență cu un identificator, iar în program se va folosi identificatorul. În limbajul C++, definirea constantelor simbolice se face cu declarația **const**.

Variabile
Sunt date a căror valoare se modifică în timpul execuției programului, când pot avea o valoare inițială, mai multe valori intermediare și o valoare finală. Identificarea lor în cadrul programului se face printr-un nume care li se atribuie la începutul programului, atunci când li se stabilește și tipul.

Se recomandă **folosirea constantelor simbolice** în următoarele cazuri:
- → valoarea datei rămâne constantă pe o perioadă de timp, după care se poate modifica (de exemplu, într-un program pentru calcularea salariilor, baza de impozitare și deducerea personală se modifică după anumite perioade de timp și, folosind constante simbolice, se poate face mult mai ușor modificarea în programul sursă),
- → data este o constantă matematică (de exemplu, constanta **pi**) sau
- → o dată constantă este folosită în mai multe locuri în program.

Analiza datelor se poate face în trei moduri:
- → **Conceptual** (la nivelul algoritmului pentru rezolvarea problemei). De exemplu, o dată *n* este un număr întreg pozitiv, cu valori cuprinse între 0 și 40000.
- → **Logic** (la nivelul implementării în limbajul de programare). De exemplu, pentru data *n* se alege tipul *unsigned int* care este o dată numerică de tip întreg cu valori pozitive, ce poate lua valori de la 0 la 65535 și care permite aplicarea operatorilor aritmetici și relaționali. La nivelul limbajului de programare este necesar să fie implementați diferiți algoritmi care să permită folosirea acestui tip de dată: algoritmi de încărcare a valorii datei în zona de memorie, algoritmi de adresare a zonei de memorie alocate, algoritmi de extragere a valorii din zona de memorie etc.
- → **Fizic** (la nivelul reprezentării ei în memoria internă corespunzător modului de implementare în limbajul de programare). De exemplu, data de tip *unsigned int* este reprezentată în doi octeți de memorie, prin codificarea în binar a valorii numerice.

Memoria internă a calculatorului este organizată sub forma unor locații de dimensiunea unui octet, numerotate consecutiv, pornind de la 0. Aceste numere, exprimate în hexazecimal, se numesc **adrese de memorie**. Locațiile de memorie pot fi manipulate individual sau în grupuri contigue.

O dată manipulată de program prin intermediul unei variabile de memorie este caracterizată de mai multe atribute:
- → **Numele**. Este identificatorul folosit pentru referirea datei în cadrul programului.
- → **Adresa**. Este adresa de memorie internă la care se alocă spațiu datei respective pentru a stoca valoarea datei.
- → **Valoarea**. Este conținutul, la un moment dat, al zonei de memorie rezervate datei.
- → **Tipul**. Determină: *domeniul de definiție intern* al datei (mulțimea în care poate lua valori data), *operatorii* care pot fi aplicați pe acea dată și modul în care data este *reprezentată în memoria internă* (metoda de codificare în binar a valorii datei).
- → **Lungimea**. Este dimensiunea zonei de memorie alocate datei și se măsoară în octeți. Ea depinde de modul de reprezentare internă a tipului de dată.

Data primește aceste atribute în momentul în care este declarată în program.

Exemplu: Prin următoarea declarare a unei date în program:

```
unsigned int a=100;
```

data va primi următoarele atribute:

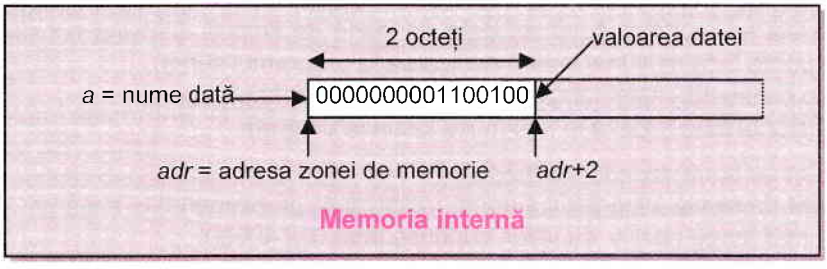
→ **Numele** – *a*.

→ **Adresa** – *adr* (un număr binar care reprezintă adresa unui octet de memorie). Atribuirea adresei este făcută de sistem, în funcție de locațiile cu dimensiunea de 2 octeți libere din zona de lucru alocată programului.

→ **Valoarea** – *100*.

→ **Tipul** – *unsigned int*. Determină: **domeniul de definiție intern** al datei (intervalul [0, 65535]), **operatorii** care pot fi aplicați pe acea dată (operatorii permiși de tipul numeric – aritmetici, relaționali, logici, logici pe biți etc.) și modul în care data este **reprezentată în memoria internă** (reprezentarea prin conversia numărului din zecimal în binar).

→ **Lungimea** – *2 octeți* (datei i se alocă un grup contiguu de 2 octeți).



În urma declarării datelor într-un program, sistemul își construiește o tabelă prin care stabilește corespondența dintre *numele* datei, *adresa* la care se memorează data și *lungimea* zonei de memorie alocată. Pentru exemplul precedent, se va memora în tabelă: numele *a*, adresa *adr* și lungimea *2*. Atunci când se va cere printr-o instrucțiune din program să se afișeze valoarea acestei date, sistemul va căuta în tabelă identificatorul *a*. Dacă îl găsește, va citi conținutul zonei de memorie care începe de la adresa *adr* și are lungimea de *2* octeți, îl va converti din modul de reprezentare internă în modul de reprezentare externă și va afișa valoarea obținută în urma conversiei, la dispozitivul desemnat pentru afișarea ei. Dacă nu găsește acest identificator, sistemul va afișa un mesaj de eroare prin care va preciza că nu există această dată.

Pentru a putea manipula o dată, sistemul trebuie să identifice zona de memorie în care este stocată. Identificarea acestei zone se poate face atât prin numele datei (*a*), cât și prin adresa la care este memorată (*adr*).

1. Declarați două constante: una de tip numeric întreg și una de tip numeric real.
2. Declarați două variabile: una de tip numeric și una de tip caracter.
3. Enumerați, pentru fiecare tip de dată, subtipurile implementate. Arătați modul în care pot fi declarate și precizați spațiul de memorie alocat fiecărui subtip și domeniul de valori.
4. Analizați data de tip **char** (reprezentare în memoria internă, încadrare în criteriile prezentate, operatori permiși etc.).
5. Structurii de date *a*, declarată cu **double** a[10];, i se alocă un spațiu de memorare continuu, începând de la adresa 200. La ce adresă se va găsi data *a[4]*?
6. Analizați următoarele tipuri de date din punct de vedere al modului în care se încadrează în criteriile prezentate.

```
const int MAXIM=1000, MZ[3][3]={0};
const float PI=3.14;
typedef unsigned char byte;
typedef enum {False,True} boolean;
int a, b[10]; float c, d[2][3]; char x; byte y; boolean Z;
```

a) Câte date au fost definite? Enumerați-le!
b) Câte variabile de memorie au fost definite? Enumerați-le!
c) Câte date structurate au fost definite? Enumerați-le!

1.2. Tipul de dată pointer

Ați învățat să manipulați o dată într-un program, folosind numele pe care l-ați atribuit datei atunci când ați definit-o. Ea poate fi însă manipulată şi cu ajutorul adresei de memorie la care este stocată valoarea sa. Pentru a putea manipula datele cu ajutorul adreselor de memorie a fost implementat un tip de dată numit **pointer** sau **adresă**.

Pointerul este o variabilă de memorie în care se memorează o adresă de memorie.

Observație:

Un pointer se asociază întotdeauna unui tip de dată (de exemplu: int, char, float etc.) numit **tip de bază**. Se spune că un pointer **indică** o zonă de memorie care conține o dată care are tipul de bază. Este necesar să se asocieze pointerului un tip de dată, deoarece el memorează numai o adresă şi, pentru a manipula conținutul unei zone de memorie, compilatorul trebuie să cunoască dimensiunea zonei de memorie care începe de la acea adresă şi semnificația conținutului, adică ce algoritm de decodificare trebuie să folosească pentru a transforma secvența de biți (din acea zonă de memorie) într-o dată.

1.2.1. Declararea variabilelor de tip pointer

Aşadar, pentru gestionarea adreselor de memorie se foloseşte tipul de dată numit **pointer**. Declararea unei variabile de tip pointer se poate face prin instrucțiunea declarativă:

Exemplul 1:

```
tip_dată  *nume_variabilă;
```

tipul datei stocate la adresa memorată în pointer (**tipul de bază**)

numele variabilei de tip pointer în care se memorează adresa unei variabile de memorie care are tipul de bază

operatorul de **referință**

```
char *p;
int *q;
float *r;
```

S-au definit trei variabile de tip adresă (pointeri): *p* către tipul **char**, *q* către tipul **int** şi *r* către tipul **float**. Numele acestor variabile de memorie sunt *p*, *q* şi *r*, iar tipul de bază al fiecărei variabile de tip pointer este **char**, **int** respectiv **float**. Aşadar, *p* este o variabilă de tip adresă a unei locații care conține un caracter, adică o dată de tip **char** care ocupă 1 octet, *q* este o variabilă de tip adresă a unei locații care conține o dată numerică întreagă de tip **int** care ocupă 2 octeți, iar *r* este o variabilă de tip adresă a unei locații care conține o dată numerică reală de tip **float** care ocupă 4 octeți.

Un pointer este şi el tot o variabilă de memorie şi este caracterizat de aceleaşi atribute ca orice variabilă de memorie. De exemplu, variabila *p* de tip pointer are următoarele atribute:

- → **Numele** – *p*.
- → **Adresa** – *adr* (un număr binar care reprezintă adresa unui octet de memorie). Atribuirea este făcută de sistem, în funcţie de locaţiile libere din zona de lucru alocată programului.
- → **Valoarea** – un număr binar care reprezintă o adresă de memorie.
- → **Tipul** – *char** determină: **domeniul de definiţie** al datei (un număr binar ce reprezintă o adresă de memorie a unei date de tip **char** care ocupă un octet de memorie), **operatorii** care pot fi aplicaţi pe acea dată (operatorii permişi, de tipul pointer, cât şi unitatea folosită în calcul de operatorii aritmetici) şi modul în care data este **reprezentată în memoria internă** (reprezentarea în binar a unei adrese de memorie).
- → **Lungimea** – Dimensiunea zonei de memorie alocate unei variabile de tip pointer depinde de mai mulţi factori: tipul de maşină (calculator) pe care rulează programul, modelul de memorie în care este compilat programul, modul de implementare al limbajului C++ (compilatorul) etc. Pentru a afla dimensiunea zonei de memorie alocate unui pointer se poate folosi operatorul **sizeof()**. În exemplele următoare vom considera lungimea egală cu 2 octeţi (situaţia cel mai des întâlnită în implementările pe microcalculatoare).

Exemplul 2:

```
int *p,*q,*r,i,j;
cout<<sizeof(p);
```

S-au definit trei variabile de tip adresă (pointeri) către tipul **int** (*p*, *q* şi *r*) şi două variabile de memorie de tip **int** (*i* şi *j*). Instrucţiunea **cout** afişează rezultatul expresiei **sizeof**(*p*), adică dimensiunea zonei de memorie ocupate de variabila de memorie de tip pointer *p*.

Există următoarele **categorii speciale de pointeri**:

a. **Pointeri fără tip**. Aceşti pointeri se asociază tipului de dată **void**. Ei sunt pointeri universali, putând fi asociaţi oricărui tip de dată. Sunt utilizaţi pentru a putea referi, la momente diferite, tipuri de bază diferite. În exemplul următor, pointerul *p* este un pointer fără tip:

```
void *p;
```

b. **Pointerul zero**. Acest pointer are valoarea 0 şi el indică zona de memorie de la adresa 0. Pentru limbajul C++ adresa 0 înseamnă o adresă care nu este validă pentru date, adică o adresă inexistentă. Altfel spus, pointerul zero nu indică nimic. Acest tip de pointer este folosit de obicei ca terminator în diverse structuri de date. În locul constantei numerice literale 0, se poate folosi constanta simbolică predefinită **NULL**, care are valoarea „adresă nulă" (adică, adresă inexistentă sau adresă nealocată). Constanta NULL este definită în fişierele antet **<stdio.h>** şi **<stdlib.h>**. Această constantă este utilă în unele cazuri pentru iniţializarea valorii unui pointer.

1. Definiţi o variabilă de tip pointer către tipul **float** şi o variabilă de tip pointer către tipul **double**. Scrieţi un program în care, folosind operatorul **sizeof**(), aflaţi dimensiunea zonei de memorie ocupate de fiecare dintre aceste variabile de memorie de tip pointer. Ce constataţi? Explicaţi rezultatul afişat.
2. Definiţi o variabilă de tip pointer către tipul **int** pe care o iniţializaţi cu pointerul zero şi o variabilă de tip pointer către tipul **char**. Scrieţi un program în care afişaţi conţinutul acestor variabile de memorie. Ce constataţi? Explicaţi rezultatul afişat.

1.2.2. Constante de tip adresă

O constantă simbolică de tip adresă este un pointer al cărui conținut nu poate fi modificat. Ea poate fi definită prin instrucțiunea declarativă:

```
tip_dată  *const nume=adresă;
```

tipul datei stocate la adresa memorată în constantă (**tipul de bază**)

numele constantei de tip pointer în care se memorează o adresă ce are o valoare constantă

valoarea constantă a adresei de memorie

Exemplu:

```
int * const zero=0;
```

Instrucțiunea definește constanta *zero* de tip adresă către un întreg. Constanta are valoarea 0 (adresă inexistentă).

1.2.3. Tablourile de memorie și adresele

La declararea unui tablou de memorie, acestuia i se alocă o zonă de memorie de dimensiune fixă, începând de la o anumită adresă de memorie. Adresa de la care se alocă zona de memorie este o constantă (nu poate fi modificată) și ea este asociată cu numele tabloului de memorie.

Numele tabloului de memorie este folosit de compilatorul C++ ca **o constantă simbolică de tip adresă**, ce reprezintă adresa de la care începe zona de memorie internă alocată tabloului. Din această cauză, nu se poate folosi operatorul de atribuire între două variabile de tip tablou de memorie, ca în exemplul următor:

```
int a[10],b[10];
a=b;          /* Eroare! Unei constante de tip adresă (a) nu i se
poate modifica valoarea */
```

În schimb, se poate atribui unui pointer o variabilă de tip tablou de memorie:

```
int a[10], *p;
p=a;          /* Corect! Unei variabile de memorie (p) i se poate
atribui valoarea unei constante (a) */
```

Tabloul de memorie este o structură de date omogenă, adică o colecție de elemente de același tip. Fiecare element ocupă același număr de octeți (în exemplul de mai sus, fiecare element ocupă 2 octeți). Acest mod de memorare permite determinarea adresei fiecărui element pornind de la adresa simbolică a tabloului, care va reprezenta și adresa primului element. Identificarea unui element de tablou de memorie se face folosind pentru fiecare element un grup de indici (numărul de indici fiind egal cu dimensiunea tabloului), adresa elementului calculându-se față de un element de referință care este adresa tabloului.

Exemplul 1 – tabloul de tip vector:

```
int a[5];
```

Zona de memorie alocată vectorului are dimensiunea $n \times sizeof(tip\_bază)$. În exemplu, vectorului *a* i se va aloca o zonă de memorie de 5×sizeof(int)=5×2=10 octeți.

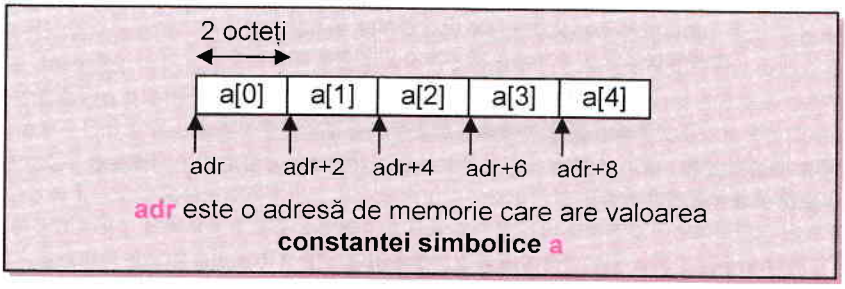
Deoarece adresa vectorului *a* este și adresa primului element, în limbajul C++ numerotarea indicilor se face pornind de la *0*, indicele reprezentând deplasarea elementului față de

adresa vectorului. Identificarea unui element al vectorului se face folosind un indice (*i*). Astfel, primul element are indicele 0, al doilea element are indicele 1 ş.a.m.d. Dacă adresa vectorului *a* este *adr*, ea este şi adresa elementului *a[0]*, iar adresa elementului *a[i]* va fi:

$$adr + i \times sizeof(tip\_bază).$$

Identificarea unui element al vectorului se face folosind un singur indice (*i*). Pentru exemplul de mai sus, adresa elementului *a[2]* va fi:

$$adr + 2 \times sizeof(int) = adr + 2 \times 2 = adr + 4.$$



Exemplul 2 – tabloul de tip matrice:

```
int a[4][2];
```

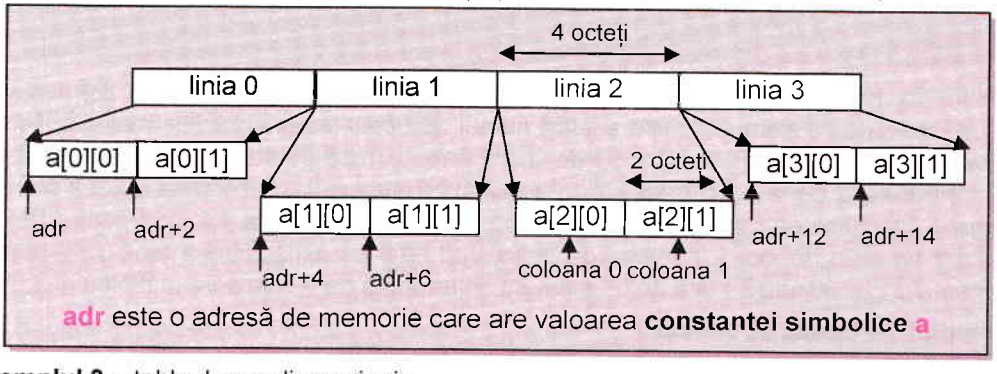
Zona de memorie alocată matricei are dimensiunea $m \times n \times sizeof(tip\_bază)$. În exemplu, matricei *a* i se va aloca o zonă de memorie de $4 \times 2 \times sizeof(int) = 4 \times 2 \times 2 = 16$ octeți.

Alocarea memoriei unui tablou de tip matrice se face în mod contiguu, memorarea elementelor făcându-se linie cu linie, astfel încât matricea apare ca un vector de linii. Dacă matricea are *m* linii şi *n* coloane, ea va avea $m \times n$ elemente (în exemplu, un vector cu $4 \times 2 = 8$ elemente) şi va fi memorată ca un vector cu *m* elemente de tip vector cu *n* elemente care au tipul de bază al matricei (în exemplu, un vector cu *4* elemente de tip vector cu *2* de elemente de tip **int**). Identificarea unui element al matricei se face folosind doi indici (*i* şi *j*). Adresa tabloului este şi adresa primului element *a[0][0]*. Dacă adresa tabloului este *adr*, adresa elementului *a[i][j]* va fi:

$$adr + i \times n \times sizeof(tip\_baza) + j \times sizeof(tip\_baza) = adr + (i \times n + j) \times sizeof(tip\_baza).$$

Pentru exemplul de mai sus, adresa elementului *a[3] [1]* va fi:

$$adr + 3 \times 2 \times sizeof(int) + 1 \times sizeof(int) = adr + 3 \times 2 \times 2 + 1 \times 2 = adr + 14.$$



Exemplul 3 – tabloul cu *n* dimensiuni:

```
int a[50][50][50]...[50];
```

Alocarea memoriei unui tablou cu *n* dimensiuni se face în mod contiguu, memorarea elementelor făcându-se linie cu linie, astfel încât el apare ca un vector de tablouri cu *n-1* di-

mensiuni. Dacă numerele de elemente după cele *n* dimensiuni sunt *d1*, *d2*, ..., *dn*, tabloul de memorie va avea *d1*×*d2*×...×*dn* elemente (în exemplu, un vector cu 50 de elemente de tip tablou cu *50*×*50*×*50*×... ×*50* de elemente – produsul având *n-1* termeni). Zona de memorie alocată tabloului cu *n* dimensiuni va avea lungimea de *d1*×*d2*×...×*dn*×*sizeof(tip_baza)*. De exemplu, tabloului *a* cu *4* dimensiuni (`int a[50][40][30][20];`) i se va aloca o zonă de memorie de 50×40×30×20×sizeof(int) = 50×40×30×20×2 = 2400000 octeți.

Identificarea unui element al tabloului se face folosind *n* indici (*i1,i2, ..., in*). Adresa tabloului este şi adresa primului element *a[0][0] ... [0]*. Dacă adresa tabloului este *adr*, adresa elementului *a[i1] [i2] ... [in]* va fi:

$$adr + (i1 \times d2 \times d3 \times ... \times dn + i2 \times d3 \times d4 \times ... \times dn + i3 \times d4 \times d5 \times ... \times dn + ... + in\text{-}1 \times dn + in) \times sizeof(tip\_baza).$$

Pentru exemplul de mai sus, adresa elementului *a[30] [20] [10] [5]* va fi:

$$adr + (30 \times 40 \times 30 \times 20 + 20 \times 30 \times 20 + 10 \times 20 + 5) \times sizeof(int) =$$
$$= adr + (720000+12000+200+5) \times 2 = adr + 732205 \times 2 = adr + 1464410.$$

1.2.4. Operatori pentru variabile de tip pointer

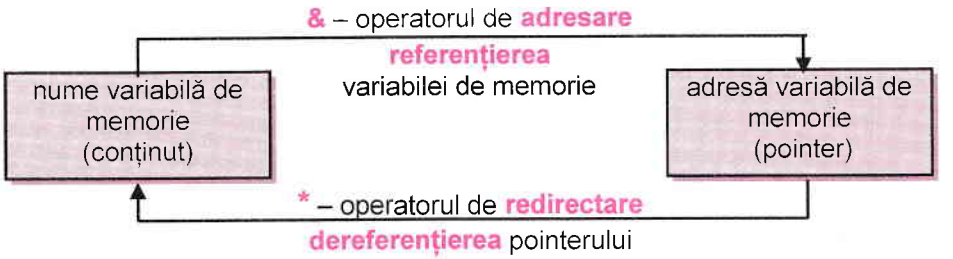
Pe variabilele de tip pointeri se pot aplica următoarele tipuri de operatori:

- → **operatori specifici**,
- → **operatorul de atribuire**,
- → **operatori aritmetici**,
- → **operatori relaționali**.

1.2.4.1. Operatori specifici

Operatorii specifici variabilelor de tip pointer sunt:

- → **Operatorul &** – operatorul de **adresare**. Se aplică pe o variabilă de memorie sau un element de tablou de memorie şi furnizează adresa variabilei de memorie, respectiv a elementului de tablou. Nu se poate aplica pe o expresie, o constantă sau o variabilă de tip registru (va fi definită ulterior). Rezultatul obținut prin aplicarea acestui operator este de tip pointer. Operația se numeşte **referențierea** unei variabile de memorie.
- → **Operatorul *** – operatorul de **redirectare**. Se aplică pe o variabilă de tip pointer şi furnizează valoarea variabilei de memorie care se găseşte la adresa memorată în pointer. Rezultatul obținut prin aplicarea acestui operator este de tipul datei asociate pointerului. Operația se numeşte **dereferențierea** unui pointer.



Aşadar:

- → Pointerii reprezintă adrese ale unei zone de memorie (conținutul unei variabile de memorie de tip pointer *p* este o adresă de memorie).
- → Accesul la o zonă de memorie se poate face fie folosind identificatorul asociat zonei de memorie (numele variabilei de memorie), fie folosind adresa zonei de memorie

obținută prin operatorul de redirectare * (conținutul adresei de memorie indicate de un pointer *p* se referă cu ***p**).

→ Dimensiunea şi semnificația conținutului unei zone de memorie indicate de un pointer depind de tipul pointerului.

Localizarea unei date memorate într-o variabilă de memorie se poate face prin:

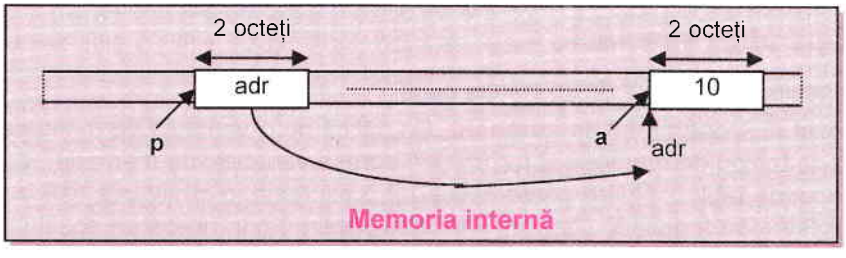
→ **adresare directă** – folosind numele variabilei de memorie;

→ **adresare indirectă** – folosind o variabilă de tip pointer în care este memorată adresa zonei de memorie în care este stocată data.

Exemplul 1:

```
int a=10, *p=&a;
cout<<*p<<endl;
cout<<a<<endl;
cout<<p<<endl;
cout<<&a<<endl;
```

S-a definit o variabilă de memorie de tip **int** (*a*) şi o variabilă de tip pointer către tipul **int** (**p*) căreia i s-a atribuit ca valoare adresa variabilei *a* (prin operatorul de adresare **&** aplicat pe variabila de memorie *a*). În memoria internă se vor aloca două zone de memorie: una de 2 octeți pentru variabila de memorie identificată prin numele *a*, în care se păstrează o valoare numerică întreagă, şi una de 2 octeți pentru variabila pointer identificată prin numele *p*, în care se păstrează o adresă de memorie (în exemplu, adresa variabilei de memorie *a*). Conținutul acestor zone de memorie va fi:



Atenție Prin instrucțiunea **cout<<***p**;** se afişează conținutul variabilei de memorie *a*. Referirea la această variabilă de memorie se face nu prin numele variabilei, ci prin operatorul de redirectare * aplicat pe variabila pointer *p* în care este memorată adresa variabilei *a*. Această instrucțiune are acelaşi efect ca şi instrucțiunea **cout<<a;**. Prin instrucțiunea **cout<<p;** se afişează o constantă hexazecimală care reprezintă o adresă de memorie (adresa *adr* la care este memorată variabila *a*). Această instrucțiune are acelaşi efect ca şi instrucțiunea **cout<<&a;**.

Temă Scrieți un program în care declarați o variabilă de memorie de tip **char** şi un pointer către tipul **char** şi apoi citiți de la tastatură şi afişați valoarea variabilei de tip **char** folosind cele două metode de adresare: adresarea directă şi apoi adresarea indirectă.

Exemplul 2:

```
int a=200; *(&a)=100; cout<<a;                    //afişează 100
/*operatorul & furnizează operatorului * adresa variabilei a, iar
variabilei de la această adresă de memorie (adică variabilei a) i
se atribuie valoarea 100 */
```