19. EXPRESII DE DEFINIRE ŞI REFERIRE ALE STRUCTURILOR DE DATE

19.1 Construirea expresiilor

Structurile statice de date uzuale sunt: vector, matrice, articol, fişier. Structurile dinamice uzuale sunt: listă, stivă, coadă, arbore şi graf.

Limbajele de programare conţin tipuri fundamentale de date şi tipuri derivate care permit implementarea tuturor structurilor de date.

Este important ca reprezentarea structurilor de date să se efectueze cu ajutorul modelului analitic pentru o descriere riguroasă.

Se definesc funcţii de descriere a structurilor de date:

- *cont()* – extragere conţinut structură de date;
- *succ()* – stabilire succesor;
- *pred()* – stabilire predecesor;
- *adr()* – specificare adresă zonă de memorie unde se află o anumită structură;
- *lg()* – specificare lungime zonă de memorie ocupată de structura de date aleasă;
- *tip()* – specificare elemente structură de date sub forma naturii acestora;
- *ref()* – referire variabilă de tip pointer.

Obiectivul acestui capitol este de a evidenţia modul în care se construiesc şi se utilizează expresiile de definire şi referire a tuturor structurilor de date.

Se consideră mulţimea operanzilor formată din identificator şi constante.

Se consideră mulţimea operatorilor utilizaţi în limbajul C++ pentru descrierea şi referirea structurilor de date:

- parantezele: **[ ], ( )**
- operatorul punct: **.**
- operatorul de indirectare: *****
- operatorul de referire a articolelor declarate ca pointeri: **->**

Expresiile sunt definite cu ajutorul unui metalimbaj, astfel [Rosc98]:

<expresie> ::= <termen> |
<expresie> ::= <termen><operator binar><termen> |
<expresie> ::= <operator unar><termen> |
<expresie> ::= <termen><operator unar>

Definirea expresiilor, cu menţionarea explicită a operatorilor, se realizează, conform [Rosc98], prin intermediul diagramelor de sintaxă:

a) definirea masivelor:

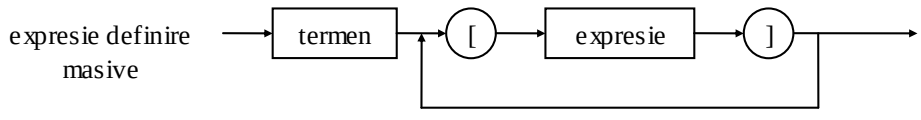

Figura 19.1 *Diagrama de sintaxă pentru definirea masivelor*

b) definirea datelor de tip articol:

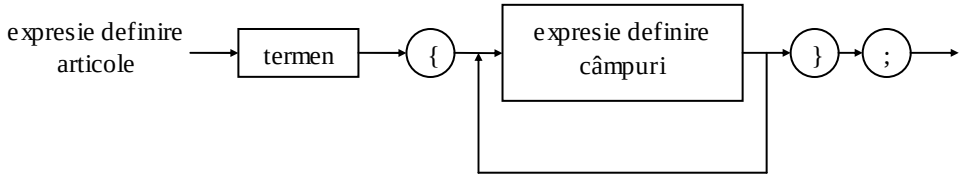

Figura 19.2 *Diagrama de sintaxă pentru definirea articolului*

Metalimbajele au rol universal în construirea expresiilor de definire şi referire a structurilor de date. Diagramele oferă o imagine grafică cu privire la procesul de construirea a acestor expresii.

19.2 Expresii de definire

Pentru masive unidimensionale, expresia de definire în limbajul C++ este:

tip nume_v[expresie];

unde:

- *tip* – natura elementelor reţinute în vector. Este un tip fundamental al limbajului, *char, int, float, double* etc., sau un tip construit de utilizator, de exemplu articol, obiect etc;
- *nume_v* – identificatorului masivului unidimensional în cadrul programului;
- *expresie* – specifică dimensiunea maximă a vectorului, numărul de elemente al acestuia.

Pentru masivele bidimensionale, expresia de definire este:

tip nume_mat[expresie$_1$][expresie$_2$];

unde:

- *tip* – natura elementelor reţinute în matrice. Este un tip fundamental al limbajului, *char, int, float, double* etc., sau un tip construit de utilizator, de exemplu articol, obiect etc;
- *nume_mat* – identificatorului masivului bidimensional în cadrul programului;
- *expresie$_1$, expresie$_2$* – specifică dimensiunea maximă a matricei, adică numărul de linii, respectiv numărul de coloane al acesteia.

Pentru masive n-dimensionale, expresia de definire este:

tip nume_m[expresie$_1$][expresie$_2$] … [expresie$_n$];

unde:

- *tip* – natura elementelor reţinute în masiv. Este un tip fundamental al limbajului, *char, int, float, double* etc., sau un unul construit de utilizator, de tip articol, obiect etc;
- *nume_m* – identificatorului masivului n-dimensional în cadrul programului;
- *expresie*$_1$, *expresie*$_2$,…, *expresie*$_n$ – specifică dimensiunea maximă a masivului, adică numărul componente al acestuia pe fiecare dimensiune.

La definirea masivelor, se remarcă faptul că expresiile care specifică numărul maxim de elemente ale masivului pe fiecare dimensiune trebuie să fie constant. Acesta este un neajuns atunci când se urmăreşte realizarea unei aplicaţii flexibile şi eficiente din punctul de vedere al gestionării memoriei.

Soluţia la această problemă o constituie alocarea dinamică a masivelor, folosindu-se fie funcţiile *malloc*, respectiv *free*, fie operatorii *new*, respectiv *delete*.

Pentru definirea tipului articol se construieşte următoarea expresie:

```
struct nume_art {
        tip1 câmp1;
        tip2 câmp2;
        …
        tipn câmpn;
};
```

unde:

- *struct* – cuvânt cheie care permite definirea articolelor în limbajul C++;
- *nume_art* – identificatorul articolului în cadrul programului;
- *tip*$_1$, *tip*$_2$, …, *tip*$_n$ – natura câmpurilor din cadrul articolului. Tipurile utilizate sunt cele specifice limbajului, *char, int, float, double* etc., sau definite de utilizator. De menţionat că nu se permite definirea în cadrul articolului a unui câmp de tipul articolului, dar se pot defini pointeri către acesta;
- *câmp*$_1$, *câmp*$_2$, …, *câmp*$_n$ – identificatorii câmpurilor articolului prin intermediul cărora se referă componentele tipului articol.

Definirea structurilor autoreferite se realizează după următorul şablon:

```
struct nume_auto {
        tip1 câmp1;
        tip2 câmp2;
        …
        tipn câmpn;
        nume_auto *pcâmp;
};
```

Definirea structurii de tip fişier se realizează cu ajutorul structurii standard *FILE* astfel:

```
FILE *f;
```

unde:

- *FILE* – structura standard care permite definirea identificatorilor de fişier în limbajul C++;
- *f* – identificatorul de fişier folosit în cadrul programului, care se declară ca pointer către structura standard *FILE*.

Lista simplu înlănţuită este o structură de date secvenţială, fiecare element al listei, denumit nod, conţinând informaţia propriu-zisă şi adresa de legătură cu un alt nod. Definirea unui nod al listei se realizează astfel:

```
struct nods {
        tip1 câmp1;
        tip2 câmp2;
        …
        tipn câmpn;
        nods *urm;
};
```

unde:

- *nods* – identificatorul structurii unui element în cadrul listei;
- $tip_1, tip_2, \ldots, tip_n$ – natura elementelor care formează informaţia utilă a nodului;
- $câmp_1, câmp_2, \ldots, câmp_n$ – identificatorii câmpurilor informaţiei utile;
- *urm* – reţine adresa următorului nod în listă.

Lista dublu înlănţuită este o structură de date secvenţială, fiecare element al listei, denumit nod, conţinând informaţia propriu-zisă şi adresele de legătură cu nodul anterior, respectiv nodul următor în listă. Definirea unui nod al listei se realizează astfel:

```
struct nodd {
        tip1 câmp1;
        tip2 câmp2;
        …
        tipn câmpn;
        nodd *pred, *urm;
};
```

unde semnificaţia câmpurilor este aceeaşi ca la lista simplu înlănţuită, cu menţiunea că *pred* reţine adresa nodului predecesor în listă.

Stiva este, de asemenea, o structură liniară, de tip listă simplu înlănţuită, cu deosebirea că inserările şi extragerile de elemente ale liste se face întotdeauna la un acelaşi capăt al listei, respectând disciplina *LIFO* – Last Input First Output.

Definirea structurii de tip stivă se realizează la fel cu cea a unei liste simplu înlănţuite.

Pointerul la stivă prin care se manipulează structura în ansamblu se numeşte vârful stivei. Definirea acestuia are loc astfel:

```
nods *stiva;
```

Conceptul de stivă este implementat folosind şi masivele unidimensionale. Dezavantajul acestei abordări este dat de numărul maxim de elemente care sunt stocate, numărul de elemente din listă fiind acelaşi pe durata de execuţie a programului.

În funcţie de dimensiunea tipurilor elementelor care sunt stocate în structură, există două abordări:

1. dacă dimensiunea este mare, se foloseşte vectorul pentru stocarea pointerilor spre elementele alocate dinamic:

```
struct stiva {
    tip *vec[dim_max];
     int sp;
};
```

unde:

- *stiva* – structură de tip articol;
- *vec* – vectorul de pointeri spre elementele stivei;
- *sp* – vârful stivei.

Această implementare este preferabil să fie evitată, ea combinând dezavantajul inflexibilităţii vectorilor cu inconvenientul ocupării unui spaţiu apreciabil de memorie cu informaţia de legătură. Acesta este motivul pentru care se stiva se implementează cu ajutorul listelor simplu înlănţuite.

2. dacă tipul elementelor este unul standard, vectorul este utilizat pentru stocarea elementelor propriu-zise:

```
struct stiva {
    tip vec[dim_max];
    int sp;
};
```

Această implementare se justifică pentru anumite probleme, ea eliminând necesitatea rezervării unui spaţiu suplimentar de memorie pentru informaţia de legătură.

Coada este implementată având la bază tot structura de listă. În cadrul acestei structuri disciplina de lucru este *FIFO* – First Input First Output. Inserarea nodurilor are loc la sfârşitul listei, iar extragerea acestora la începutul listei.

Definirea acestei structuri are loc astfel:

```
struct coada{
   nods *prim, *ultim;
};
```

unde:

- *coada* – structura de tip coadă;
- *nods* – identificator structură element listă simplu înlănţuită;
- *prim, ultim* – primul, respectiv ultimul nod care permit implementarea operaţiilor specifice structurii de coadă.

Arborele reprezintă o mulţime nevidă şi finită de elemente, numite noduri, cu proprietăţile:

- există doar un singur nod, numit rădăcina arborelui;

- celelalte noduri formează submulţimi disjuncte ale arborelui, care la rândul lor respectă cele două proprietăţi. Aceste submulţimi poartă numele de subarbori ai rădăcinii.

Arborii binari sunt formaţi dintr-o mulţime finită de elemente care este fie vidă, fie formată dintr-un singur element numit rădăcină, iar celelalte elemente sunt distribuite în maxim două submulţimi disjuncte, care reprezintă tot arbori binari.

Descrierea structurii de tip arbore binar se realizează în limbajul C++ astfel:

```
struct arb_bin{
        tip inf;
        arb_bin *st, *dr;
};
```

unde:

- *arb_bin* – identificatorul unui element de tip nod al arborelui în cadrul programului;
- *inf* – informaţia utilă prezentă în nod;
- *st, dr* – elemente ce păstrează adresele de legătură cu subarborele stâng, respectiv drept al arborelui curent.

Matricea rară este un tip particular de masiv bidimensional cu următoarele caracteristici:

- număr foarte mare de linii şi de coloane;
- număr de elemente nenule foarte mic.

În aplicaţiile curente, matricele rare au grade de umplere cuprinse între 0,15% şi 3%.

Memorarea şi lucrul cu matrice rare se efectuează prin intermediul a trei vectori în care se memorează linia, coloana şi valoarea nenulă a tuturor elementelor care nu sunt nule.

Din categoria structurilor de date agregate fac parte şi vectorii de pointeri spre funcţii. Definirea unui vector de pointeri spre funcţii se face conform următorului şablon:

*tip (*pf[10])();*

unde:

- *tip* – tipul returnat de funcţie;
- *pf* – vectorul de pointeri spre funcţii, dimensiunea acestuia fiind de 10 elemente.

Introducerea în vector a pointerilor spre diverse funcţii care returnează *tip* se realizează astfel:

```
pf[i]=functie;
```

unde:

- *pf* – vectorul de pointeri spre funcţii care returnează *tip*, iar lista parametrilor este vidă;
- *functie* – numele dat unei funcţii care întoarce *tip* şi are listă vidă de parameteri; identificatorul este pointer spre funcţia respectivă.

Trecerea la limbajul de programare C++ a permis implementarea de noi structuri de date utilizator pe baza conceptelor definite şi caracterizate în teoria programării orientate obiect.

Astfel, utilizatorul defineşte o clasă de obiecte după modelul:

```
class ClasaUser {
        private:
        //definire atribute şi metode private (încapsulare)
        public:
        //definire metode de tip constructor şi destructor
        //definire atribute şi metode publice
        protected:
        //definire atribute şi metode de tip protected (moştenire)
};
```

De exemplu, definirea clasei de obiecte pentru implementarea structurii de date de tip arbore binar este:

```
class ArbBinClass{
    tip inf;
    arb_bin *st, *dr;
public:
    ArbBinClass();
    ~ArbBinClass();
    //alte metode
};
```

Spre deosebire de structura de date de tip articol, o clasă de obiecte include atât date, denumite atribute, cât şi operaţii care se efectuează asupra acestora.

Deci, alături de stare, o clasă de obiecte include şi comportamentul variabilelor de tip obiect. Comportamentul este evidenţiat prin metodele construite în clasa de obiecte. Acestea operează asupra atributelor, determinând schimbarea stării variabilei obiect prin modificarea valorilor atributelor.

19.3 Expresii de referire

Referirea structurilor de date are loc:

- la nivel de ansamblu (întreaga structură);
- la nivel de element al structurii.

Referirea elementelor masivelor unidimensionale se realizează astfel:

- cu ajutorul operatorului de indexare: *nume[exp]*;
- cu ajutorul operatorului de indirectare: **(nume+exp)*.

Identificatorul de masiv unidimensional *nume* conţine adresa primului element din vector.

Reprezentarea sub formă de zone de memorie a unui vector este ilustrată în figura următoare:

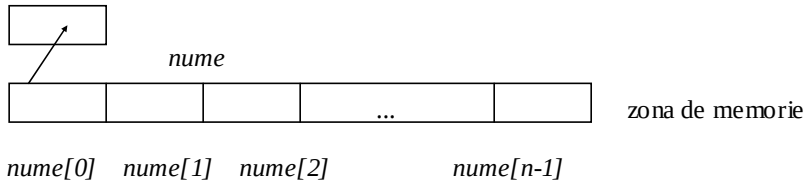

Figura 19.3 *Reprezentarea în memorie a unui vector*

La referirea elementelor unui vector, depăşirea dimensiunii acestuia conduce la accesarea zonelor de memorie învecinate, rezultatul fiind imprevizibil.

Referirea elementelor masivelor bidimensionale:

- cu ajutorul operatorului de indexare: *nume[exp$_1$][exp$_2$]*;
- cu ajutorul operatorului de indirectare: **(*(nume+exp$_1$)+exp$_2$)*.

Identificatorul unui masiv bidimensional conţine adresa vectorului de pointeri către linii. Adresa fiecărei linii a masivului se obţine pornind de la identificatorul de masiv astfel: *nume[exp$_1$]* sau **(nume+exp$_1$)*.

Reprezentarea unui masiv bidimensional sub formă de zone de memorie este:

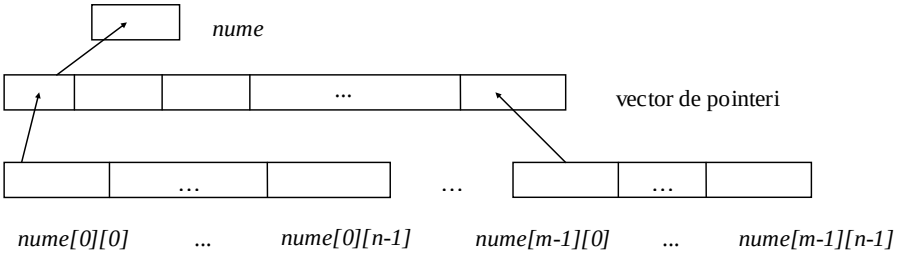

Figura 19.4 *Reprezentarea în memorie a unui masiv bidimensional*

Referirea elementelor masivelor tridimensionale:

- cu ajutorul operatorului de indexare: *nume[exp$_1$][exp$_2$][exp$_3$]*;
- cu ajutorul operatorului de indirectare: **(*(*(nume+exp$_1$)+exp$_2$)+exp$_3$)*.

Identificatorul masivului tridimensional conţine adresa unui vector de pointeri către matrice, vector de dimensiune *exp$_1$*. Vectorul de pointeri către masive bidimensionale este referit astfel:

nume[exp$_1$] sau **(nume+exp$_1$)*

Referirea structurilor de tip articol se realizează prin identificatorii de variabile declarate de acest tip. Astfel, articolele sunt utilizate, la nivel de ansamblu, în următoarele operaţii:

- extragere de adresă a unei variabile de tip structură;
- atribuire;
- transmiterea ca argument în funcţie sau returnarea unei structuri din funcţie.

Nu sunt acceptate comparări de forma: *nume$_1$!=nume$_2$*.

Referirea câmpurilor din structurile de tip articol se realizează prin intermediul operatorului punct, astfel:

nume.câmp;

Reprezentarea şi referirea câmpurilor unei structuri de tip articol:

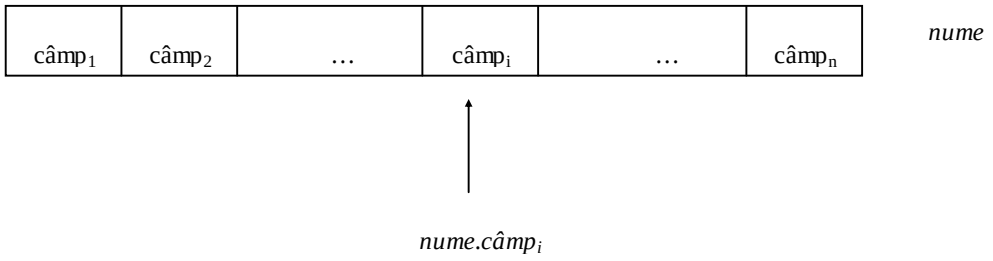

Figura 19.5 *Reprezentarea în memorie şi referirea unui articol*

Dacă articolul conţine câmpuri de tip articol, atunci câmpurile acestora se referă în mod similar, astfel:

nume.câmp1.câmp2. … .câmpn;

Referirea structurilor de date autoreferite se realizează prin intermediul operatorului *„->”*. Astfel, informaţia utilă din nod, precum şi adresa de legătură cu nodul următor, respectiv succesor sunt referite astfel:

```
nodd *listad;
…
listad->inf;
listad->urm;
listad->pred;
…
```

De asemenea, referirea adresei de legătură a unui nod oarecare a unui nod din lista dublu înlănţuită are loc astfel:

```
nodd *listad
…
listad->urm->urm->…->urm;
listad->pred->pred->…->pred;
…
```

Pentru structurile de tip stivă şi coadă, referirea elementelor are loc în mod asemănător celui de la listele simplu înlănţuite.

Referirea elementelor structurii de tip stivă:

```
nods *stiva;
…
stiva->inf;
stiva->urm;
…..
stiva ->urm->urm->…->urm;
…
```

Referirea nodurilor structurii de tip coadă:

```
coada q;
…
q.prim->inf;
q.ultim->inf;
q.prim->urm;
q.ultim->ultim;
…
```

Adresele şi informaţia utilă din nodurile unui arbore binar se referă în mod analog cu cele ale unei liste dublu înlănţuite:

```
arb_bin *arbore;
…
arbore->inf;
arbore->st;
arbore->dr;
…
```

Referirea atributelor şi metodelor definite şi construite într-o clasă de obiecte se efectuează în mod similar cu referirea câmpurilor din structurile de date de tip articol.

19.4 Expresii ale structurilor agregate

Structurile prezentate în paragrafele anterioare sunt agregate sub următoarele forme:

- vector de structuri;
- structură de vectori;
- vectori de pointeri spre structuri;
- vectori de pointeri spre funcţii;
- masive bi- şi n-dimensionale de structuri;
- structură de masive bi- şi n-dimensionale;
- masive bi- şi n-dimensionale de pointeri spre structuri.

Definirea unui vector de structuri se realizează astfel:

```
nume_art vector[expresie];
```

unde:

- *nume_art* – tipul elementelor stocate în vector (articol);
- *vector* – identificatorul vectorului de structuri în cadrul programului;
- *expresie* – dimensiunea maximă a vectorului.

Referirea elementelor acestui vector are loc astfel:

```
vector[i].câmp;
```

unde:

- *vector* – identificator vector de structuri;

- *i* – elementul de rang i care se referă;
- *câmp* – un câmp din cadrul elementului de rang i.

Definirea unei structuri de vectori:

```
struct struct_vect {
        tip1 v1[exp1];
        …
        tipn vn[expn];
};
```

unde:
- *struct_vect* – identificatorul structurii de vectori;
- $tip_1, \ldots, tip_n$ – natura elementelor vectorilor;
- $v_1, \ldots, v_n$ – identificatorii vectorilor;
- $exp_1, \ldots, exp_n$ – dimensiuni maxime ale vectorilor.

Exemplu:

```
struct str{
        int v[100];
};
```

Referirea elementelor unei structuri de vectori:

```
struct_vect s, *ps;
…
s.v1[i];
ps->vn[j];
…
```

unde:
- *s* – variabile de tip articol;
- *ps* – variabilă de tip pointer spre articol;
- *i,j* – rangul elementelor care au fost referite.

Definirea vectorilor de pointeri spre structuri se realizează conform următorului şablon:

```
nume_art *v[exp];
```

unde:
- *nume_art* – denumire structură de tip articol;
- *v* – identificator variabilă de tip masiv unidimensional cu elemente pointeri spre structuri de tip *nume_art*;
- *exp* – dimensiune maximă a vectorului.

Referirea unui câmp a vectorului de pointeri:

```
v[i]->câmp;
```

unde:
- *v* –identificatorul vectorului de pointeri în cadrul programului;
- *i* – rangul termenului a cărui conţinut este referit;

- *câmp* – un câmp al articolului desemnat de elementul i.

Într-o expresie de referire a unei structuri agregate, precizarea câmpului are loc de la dreapta la stânga, astfel:

- *a.b.c* – referă câmpul *c* din structura *b*, structura *b* fiind un câmp al structurii *a*;
- *a.b.c[i]* – referă elementul *i* al câmpului *c* de tip vector, inclus în structura *b*, unde *b* este un câmp al structurii *a*;
- *a.b[i].c* – referă câmpul *c* al elementului *i* din vectorul *b*, *b* fiind un câmp structura *a*;
- *a[i].b.c* – referă câmpul *c* al structurii *b*, care este la rândul ei câmp în elementul de rang *i* al structurii *a*;
- *a->b->c* – referă câmpul *c* din structura *b*, aceasta fiind un câmp de tip pointer al structurii *a*, care este, de asemenea, o dată de tip pointer;
- *a.(*b)* – referă conţinutul zonei de memorie a cărei adresă se află în câmpul *b*, dată de tip pointer, al structurii statice *a*;
- *(*a).b* – referă câmpul *b* al structurii aflate în zona de memorie a cărei adresă este reţinută în *a*, care este o structură alocată dinamic;
- **(a+i).b* – referă câmpul *b* al unei structuri de tip articol care se află stocat la adresa conţinută în vectorul *a*, *i* indicând rangul structurii în vector;
- *a.*(b+i).c* – referă câmpul *c* al articolului aflat pe poziţia *i* în vectorul *b*, *b* fiind câmp al structurii de tip articol *a*;
- *a->b.c* – referă câmpul *c* al structurii statice de tip articol *b*, care este un câmp al structurii dinamice de tip articol *a*;
- *a.b->c* – referă câmpul *c* al structurii dinamice *b*, *b* fiind câmp al structurii statice *a*, de tip articol.

Se consideră următoarea secvenţă scrisă în limbajul de programare C++:

```
#include <stdio.h>
#include <iostream.h>

class matrix{
public:
      int n,m,**a;
      matrix();
      matrix(int k, int l);
      int** aloc_mat(int, int);
};

matrix::matrix()
{
     cout<<"The line number is:";
     cin>>n;
     cout<<"The column number is:";
     cin>>m;
     a=aloc_mat(n,m);
     cout<<"The matrix is:";
     for(int i=0; i<n; i++)
           for(int j=0; j<m; j++){
               cout<<"\n a["<<i<<"]["<<j<<"]=";
               cin>>a[i][j];
          }
}
```

```
matrix::matrix(int k, int l)
{
     n=k;
     m=l;
     a=aloc_mat(n,m);
     for(int i=0; i<n; i++)
          for (int j=0; j<m; j++) a[i][j]=0;
}

int** matrix::aloc_mat(int, int){
      int **pmat;
      pmat=new int*[n];
      for (int i=0; i<n; i++)
      pmat[i]=new int[m];
      return pmat;
}

void main(){
      matrix A(2,3);
      matrix *B=new matrix(3,2);
      matrix *C=new matrix[3];


}
```

În tabelul 19.1 sunt exemplificate expresii de referire a obiectelor şi atributelor, conţinutul accesat şi descrierea acestuia.

Tabelul nr. 19.1 *Exemplificarea expresiilor de referire a obiectelor şi atributelor clasei matrix*

Expresie de referire	Conţinut	Descriere
A.n	2	Numărul de linii al matricei A
A.m	3	Numărul de coloane al matricei A
A.a	0x00441ac0	Adresa început a zonei de memorie alocată dinamic pentru stocarea elementelor matricei A
A.a[0]	0x00441a80	Adresa de început a zonei de memorie alocată dinamic pentru linia 1 a matricei A
A.a[0][0]	0	Valoarea elementului de pe linia 1, coloana 1 din matricea A
&A	0x0013ff68	Adresa de memorie statică la care este stocat obiectul A
&A.n	0x0013ff68	Adresa de memorie statică la care este stocat atributul n
&A.m	0x0013ff6c	Adresa de memorie statică la care este stocat atributul m
&A.a	0x0013ff70	Adresa de memorie statică la care este stocat atributul a
B->n (*B).n	3	Numărul de linii al matricei A
B->m (*B).m	2	Numărul de coloane al matricei A

B->a (*B).a	0x004419c0	Adresa început a zonei de memorie alocată dinamic pentru stocarea elementelor matricei B
B->a[0] (*B).a[0]	0x00441980	Adresa de început a zonei de memorie alocată dinamic pentru linia 1 a matricei B
B->a[0][0] (*B).a[0][0]	0	Valoarea elementului de pe linia 1, coloana 1 din matricea B
&B	0x0013ff64	Adresa de memorie statică la care este stocat obiectul A
&B->n &(*B).n	0x00441a00	Adresa de memorie dinamică la care este stocat atributul n
&B->m &(*B).m	0x00441a04	Adresa de memorie dinamică la care este stocat atributul m
&B->a &(*B).a	0x00441a08	Adresa de memorie dinamică la care este stocat atributul a
C[0].n	2	Numărul de linii al primei matrice din vectorul alocat dinamic C
C[1].m	2	Numărul de coloane al celei de-a doua matrice din vectorul alocat dinamic C
*(C+i)	-	Referirea obiectului matrice cu poziţia i+1 în vectorul alocat dinamic C
C[2].a	0x00441210	Adresa început a zonei de memorie alocată dinamic pentru stocarea elementelor matricei cu poziţia 3 în vectorul de obiecte alocat dinamic C
C[1].a[0]	0x00441310	Adresa de început a zonei de memorie alocată dinamic pentru linia 1 a matricei cu poziţia 2 în vectorul de obiecte alocat dinamic C
C[0].a[0][0]	13	Valoarea elementului de pe linia 1, coloana 1 din matricea cu poziţia 1 în vectorul de obiecte alocat dinamic C

Apelul unei funcţii şi obţinerea rezultatului returnat de aceasta prin intermediul vectorului de pointeri spre funcţii se realizează după cum urmează:

```
(*pf[i])();
```

unde:
- *pf* – vectorul de pointeri spre funcţii;
- *i* – rangul în vectorul de pointeri a funcţiei care se apelează.

Construirea expresiilor în limbajele de programare, conform specificaţiilor prevăzute de fiecare limbaj, prezintă o importanţă deosebită în dezvoltarea corectă din punct de vedere sintactic şi mai ales semantic a programelor sursă. Expresiile trebuie să conducă la atingerea obiectivelor stabilite în etapa de proiectare a aplicaţiei.

Construirea expresiilor este o condiţie necesară, dar nu şi suficientă pentru a obţine rezultate corecte din punct de vedere semantic. Se impune, deci, şi o referire corectă a variabilelor şi expresiilor definite, respectiv construite într-un program sursă pentru ca rezultatele obţinute să fie în concordanţă cu cerinţele utilizatorilor.