7. POINTERI. TABLOURI. ŞIRURI DE CARACTERE

7.1. Pointeri

După cum ştim, memoria calculatorului este împărţită în octeţi fiecare avînd asociat un cod (număr) numit **adresă**. Atunci cînd declarăm o variabilă, acesteia i se rezervă un spaţiu în memorie (un număr de octeţi care depinde de tipul variabilei), unde vor fi stocate valorile variabilei respective. Adresa variabilei este adresa primului octet al zonei de memorie rezervate şi se poate obţine cu ajutorul operatorului adresă **&**. Adresele de memorie pot fi memorate şi utilizate folosind un nou tip de date, numit **tipul pointer**. Prin definiţie o constantă pointer este o adresă de memorie (de exemplu $146E_{16}$), iar o variabilă pointer este o variabilă în care se pot memora adrese de memorie.

Prin utilizarea pointerilor, limbajul C oferă utilizatorilor săi multiple facilităţi, dintre care cele mai importante sunt:

- **accesarea directă a diverselor zone de memorie** prin modificarea adresei memorate în pointer. În felul acesta pot fi create şi exploatate **listele inlănţuite**;
- **crearea de noi variabile în timpul execuţiei programului** în cadrul aşa numitului mecanism de alocare dinamică a memoriei. Limbajul C permite programatorului să solicite la un moment dat alocarea, în zona de memorie rămasă liberă, a unui număr de octeţi şi furnizează adresa octetului de început al acesteia;
- **întoarcerea dintr-o funcţie a mai multor valori** către funcţia apelantă. Am văzut că în mod uzual, o funcţie nu poate întoarce apelantului decît o singură valoare prin intermediul instrucţiunii **return**;
- **accesarea directă** a unor elemente dintr-o structură de date, cum ar fi: şirurile,matricele sau structurile complexe de date.

7.1.1. Declararea şi utilizarea pointerilor

Ca orice tip de variabilă, înainte de a fi utilizată, o variabilă de tipul pointer trebuie declarată. Declararea variabilelor de tipul pointer se face în cadrul unor instrucţiuni avînd urrnătoarea formă generală:

```
tip *nume_pointer_1, ... , *nume_pointer_n;
```

asemănătoare celei folosite pentru declararea variabilelor obişnuite. Astfel, *tip* poate fi oricare dintre tipurile fundamentale de date (char, int, float, double), iar *nume* - un şir de caractere, cu condiţia că primul caracter să fie o literă. Singura deosebire constă în faptul că în faţa numelui se plasează caracterul *, care semnalează compilatorului că s-a declarat o variabilă pointer şi nu o variabilă obişnuită. Deci, un set de instrucţiuni de forma:

```
int     *p_int, *iptr;
float   *p_real,*rptr;
char    *p_car, *carptr;
```

declară variabilele *p_int* şi *iptr* ca pointeri la întreg, variabilele *p_real* şi *rptr* ca pointeri la real simplu, respectiv variabilele *p_car* şi *carptr* ca pointeri la caracter.

Pentru a explica modul de lucru cu pointeri, considerăm programul din exemplul 7.1, în cadrul căruia sunt folosite variabilele *pt_ a* şi *ptr_b* de tipul pointer la un real, respectiv la un întreg şi variabilele *a* şi *b* de tipul real, respectiv întreg.

Exemplul 7.1.

```
/* Programul ex_7_1 */
#include<stdio.h>
void main(void)
 {
float a=5.2, *ptr_a;
int *ptr_b, b=10;
clrscr( );   printf("\nvalorile initiale a=%6.3f b %4d",a,b);
ptr_a=&a; ptr_b=&b;*ptr_a=a+10; *ptr_b=b+10;
printf ("\nValorile finale a=%6.3f b=%d",*ptr_a,*ptr_b);     getch( );
 }
```

Scopul acestui program este de a realiza cu ajutorul pointerilor opetaţiile *a=a+10* şi *b=b+10*. În acest sens instrucţiunea *float a = 5.2, *ptr_a;* declară variabila *a* de tipul real şi o iniţializează cu valoarea 5.2, iar variabila *ptr_a* de tipul pointer la un real. În mod similar, instrucţiunea *int *ptr_b,b=10;* declară variabila *ptr_b* de tipul pointer la întreg, iar variabila *b* de tipul întreg şi o iniţializează cu valoarea 10. Aşadar în cadrul aceleeaşi instrucţiuni putem declara atît variabile simple cît şi variabile pointer, în orice ordine, separate prin virgulă.

După ce, prin apelul funcţiei **printf**, sunt afişate valorile iniţiale ale variabilelor *a* şi *b*, variabilei *ptr_a* îi este atribuită valoarea adresei variabilei *a*, iar variabilei *ptr_b* - valoarea adresei variabilei *b*. În continuare, programul modifică valorile variabilelor *a* şi *b* însumînd la fiecare constanta 10, astfel că noua valoare a lui *a* devine 15.2, iar a lui *b* 20. Pentru aceasta se folosesc adresele variabilelor *a* şi *b* memorate în *ptr_a* şi *ptr_b*, precum şi caracterul "*" în faţa numelui variabilelor pointer. Această utilizare a caracterului "*" poartă numele de **indirectare**, deoarece acţiunea lui în cuvinte se traduce prin: "mergi la adresa conţinută în variabila pointer şi efectuează operaţia".

Astfel, în cadrul instructiunii **ptr_a=a+10;* se accesează zona de memorie a variabilei *a* (prin intermediul adresei conţinute în variabila pointer *ptr_a*) în care se depune valoarea a+10. Această instrucţiune este identică, ca efect, cu instrucţiunea *a=a+10;*. Din acest exemplu deducem că operatorul de indirectare ***** este complementar operatorului adresă, adică instrucţiunea **&a=a+10;* este echivalentă cu instrucţiunea *a=a+10;*.

Ultima instrucţiune a programului ex_7_1 foloseşte în cadrul funcţiei **printf** acelaşi mecanism de indirectare pentru afişarea valorilor finale ale variabilelor. Astfel, prin intermediul operatorului de indirectare *,valorile care vor înlocui descriptorii de format se obţin de la adresele memorate în pointerii *ptr_a* şi respectiv *ptr_b*, adică de la adresele variabilelor *a* şi *b*. Dacă nu am folosi operatorul *, adică instrucţiunea ar fi scrisă sub forma *printf("\n Valorile finale sunt a= %f b= %d",ptr_a,ptr_b);* descriptorii de format ar fi inlocuiţi cu valorile conţinute în variabilele *ptr_a* şi *ptr_b*, deci cu valorile adreselor variabilelor *a* şi *b*, rezultatul obţinut fiind imprevizibil.

Pentru afişarea pe ecran a conţinutului variabilelor pointer, adică a unor adrese de memorie, se foloseşte în funcţia **printf** descriptorul de format "%p". Astfel, putem înlocui ultima instrucţiune din programul anterior cu instrucţiunile:

printf("\n Valoarea conţinută la adresa %p este %f",ptr_a,*ptr_a);
printf("\n Valoarea conţinută la adresa %p este %f",ptr_b,*ptr_b);

pentru afişarea adreselor variabilelor *a* şi *b* (conţinute în *ptr_a* şi *ptr_b*) şi a valorilor acestora (pointate de *ptr_a* şi *ptr_b*).

7.1.2. Utilizarea pointerilor ca parametri ai functiilor

Mecanismul de indirectare şi utilizarea pointerilor ca parametri ai funcţiilor constituie una din modalităţile prin care se pot transmite mai multe valori de la o funcţie la alta. Pentru exemplificare consideram programul din exemplul 7.2.

Exemplul 7.2.

```
/* Programul ex_7_2 */
#include <stdio.h>
void permut{int, int);
void main(void)
 {
int a,b;clrscr( );
printf ("\n Introduceti doua valori intregi ");   scanf ("%d %d",&a,&b);
permut(a,b);   printf("\n Valorile variabilelor in functia apelanta a=%5d b=%5d",a,b);     getch( );
 }
void permut(int x, int y)
 {   /* Schimba valorile intre ele */
int temp;
printf("\n Valorile variabilelor transnise functiei a=%5d b=%5d",x,y);
temp = x;  x = y;  y = temp;
printf("\n Noile valori ale variabilelor a=%5d b=%5d",x,y);
 }
```

În cadrul acestui program sunt citite de la tastatură 2 valori întregi şi sunt memorate în variabilele *a* şi *b*, locale funcţiei **main**. Acestea sunt apoi transmise funcţiei **permut** care are rolul de a schimba între ele cele 2 valori. Pentru a realiza acest lucru, funcţia **permut** foloseşte variabila locală *temp*.

Se remarcă faptul că în funcţia **permut** nu am folosit instrucţiunea **return**, deoarece aceasta nu întoarce o valoare (are tipul void). În astfel de situaţii, revenirea în funcţia apelantă se face în mod automat după executarea tuturor instrucţiunilor ce alcătuiesc corpul funcţiei. Dacă totuşi, într-o funcţie de tipul void, este necesară revenirea în funcţia apelantă dintr-un punct care nu constituie sfîrşitul funcţiei, se utilizează instrucţiunea **return** fără parametru, sub forma: *return;*.

Revenind la programul prezentat, în scopul urmăririi execuţiei acestuia, s-a utilizat funcţia **printf** pentru afişarea valorilor transmise funcţiei **permut**, a valorilor obţinute la sfîrşitul acesteia şi respectiv, a valorilor după revenirea în funcţia principală. Dacă lansăm în execuţie programul şi introducem valorile 5 şi 10, pe ecran vom avea mesajele:

```
Valorile variabilelor transmise functiei:     a = 5   b = 10
Noile valori ale variabilelor:                a = 10  b = 5
Valorile variabilelor in functia apelanta:    a = 5   b = 10
```

din care deducem că, deşi în funcţia **permut** cele 2 valori au fost schimbate între ele, în funcţia apelantă **main** această schimbare nu s-a produs. Explicaţia acestui rezultat constă în faptul că, la apelul funcţiei, valorile variabilelor *a* şi *b* cu care se face apelul sunt copiate în zona de memorie alocată funcţiei. Funcţia operează asupra valoriior copiate, lăsînd neschimbate valorile iniţiale. Acest mecanism este cunoscut sub numele de **transmiterea parametrilor prin valoare**.

Pentru a realiza programul, astfel încît permutarea valorilor să aibă loc şi în funcţia apelantă se utilizează pointeri ca parametri ai funcţiei. Rescriem programul ex_7_2, astfel încît să realizeze acest lucru.

Exemplul 7.3.

```
/* Programul ex_7_3 */
#include <stdio.h>
void permut(int *, int *);
void main(void)
 {
int a,b;clrscr( );
printf ("\n Introduceti doua valori intregi ");   scanf("%d %d",&a,&b);
permut (&a,&b); printf ("\n Valorile variabilelor in functia apelanta a=%5d b=%5d",a,b);    getch( );
 }
void permut(int *x, int *y)
 {   /* Schimba valorile intre ele */
int temp;
printf("\n Valorile variabilelor transmise functiei a %5d b=%5d",*x,*y);
temp =*x; *x = *y;    *y = temp;     printf("\n Noile valori ale variahilelor a=%5d b=%5d",*x,*y);
 }
```

Deosebirea esenţială faţă de programul anterior constă în faptul că, de data aceasta, funcţia **permut** este definită ca avînd drept parametri 2 variabile de tipul pointer la întreg. La declararea funcţiei, pentru a preciza faptul că parametri formali sunt pointeri, după tipul acestora se inserează caracterul "*", precedat de un spaţiu, ca în instrucţiunea *void permut(int *,int *);*.

Mecanismul de transmitere este tot prin valoare,dar de data aceasta valorile transmise şi preluate de funcţie în pointerii *x* şi *y* sunt valorile adreselor variabilelor *a* şi *b*, obţinute cu operatorul **&**. Utilizînd mecanismul de indirectare, funcţia **permut**, prin intermediul celor doi pointeri, va accesa zona de memorie în care se află valorile variabilelor *a* şi *b*,astfel încît schimbarea se va ef-ectua de data aceasta în funcţia apelantă. Acest mecanism este cunoscut sub numele de **transmiterea parametrilor prin adresă**.

7.1.3. Alocarea dinamica a memoriei

Utilizarea pointerilor permite ca în timpul execuţiei unui program să folosim o parte din memoria rămasă liberă pentru memorarea temporară a unor valori, după care, zona poate fi eliberată şi folosită în alte scopuri. Pentru a descrie acest mecanism, considerăm programul din exemplul 7.4 în care se calculează valoarea polinomului de gradul II, $P(x) = a_0x^2+a_1x+a_2$, pentru o anumită valoare a nedeterminatei *x* citită de la tastatură.

Exemplul 7.4.

```
/* Programul ex_7_4 */
#include <stdio.b>
#include <alloc.h>
void main(void)
 {
float *a,val,x;
int i;    clrscr( );
if ((a=(float *) malloc (3*sizeof (float)))==NULL)
 {
    printf ("\Memorie insuficienta");    exit(1);
 }
printf ("\nIntroduceti valoarile coeficientilor \n");
for (i=0; i<=2; i++)
 {
    printf ("a%1d=", i);  scanf("%f",&val);          *(a+i) = val;
 }
printf ("Introduceti valoarea lui x");     scanf ("%f",&x);
val=*a*x*x+*(a+1)*x+*(a+2); free(a);
printf("Valoarea polinomului P(%f)=%f",x,val);  getch( );
 }
```

Noutatea acestui program constă în modul în care sunt memorate şi utilizate valorile coeficienţilor polinomiali a_0,a_1 şi a_2. Programul evită declararea a 3 variabile de tipul float în care să se memoreze coeficienţii polinomului. În locul acestora se foloseşte variabila *a*, de tipul pointer la real, şi mecanismul de accesare indirectă. Spaţiut de memorie necesar stocării celor trei valori este alocat prin apelul funcţiei **malloc**, definită în fişierul antet **alloc.h**. Această funcţie,a cărei formă generală de apel este:

malloc (expresie de tip intreg);

solicită sistemului de operare alocarea pentru programul care o apelează a unei zone de memorie avînd un număr de octeţi egal cu valoarea expresiei primite ca parametru. Dacă solicitarea este satisfăcută, funcţia returnează adresa primului octet al zonei de memorie alocată. În cazul în care, datorită lipsei unui spaţiu de memorie liber, solicitarea nu este satisfăcută, funcţia returnează valoarea NULL.

În cazul programului prezentat, instrucţiunea: *if((a=(float *)malloc(3*sizeof(float)))==NULL)* solicită alocarea unui spaţiu de 12 O, necesar memorării valorilor celor 3 coeficienţi polinomiali, şi testează dacă solicitarea a fost satisfacută. În acest sens, valoarea returnată de funcţia **malloc** este atribuită pointerului *a* şi apoi comparată cu constanta NULL. Dacă solicitarea nu este satisfăcută (valoarea pointerului *a* este 0, adică constanta NULL), programul afişează mesajul: "Memorie insuficientă" şi execuţia se termină prin instrucţiunea **exit**, care solicită revenirea în sistemul de operare sau în mediul de programare.

Revenind la apelul funcţiei **malloc**, se remarcă următoarele:

- utilizarea funcţiei **sizeof** în cadrul expresiei ce determină numărul de octeţi solicitaţi. Această funcţie are următoarea formă generală de apel: *sizeof (tip);* şi returnează un întreg reprezentînd lungimea exprimată în octeţi a tipului de date pe care o primeşte ca parametru.

- utilizarea expresiei *(float *)* în faţa numelui funcţiei. Aşa cum s-a precizat, funcţia **malloc** returnează o adresă pe care o putem atribui unui pointer de orice tip (char, int, float, double). În programul anterior expresia *(float *)* specifică faptul că adresa returnată de **malloc** va fi atribuită variabilei *a* de tipul pointer la real şi constituie un exemplu de utilizare a unui nou tip de operator, numit în limbajul C **operatorul cast**. Acest operator, format dintr-unul din tipurile de date sau de pointeri cuprins între paranteze şi plasat în faţa unei variabile, al unui apel de funcţie, respectiv al unei expresii, are rolul de a converti valorile acestora la tipul precizat. Pentru a înţelege mai bine modul de utilizare să ne gîndim că în cadrul unui program de calcul suma a 2 variabile *alfa* şi *beta*, de tipul real, dorim să o atribuim unei variabile de tipul întreg *gama*. Acest lucru este reaiizabil folosind instrucţiunea de atribuire *gama=(int)(alfa+beta);* în care operatorul *(int)* va converti mai întîi valoarea expresiei reale *alfa+beta* la tipul întreg şi apoi valoarea rezultată este atribuită variabilei întregi *gama.*

Utilizarea spaţiului de memorie alocat în urma apelului funcţiei **malloc** se realizează prin mecanismul de indirectare şi operaţiile cu pointeri. Astfel,în programul ex_7_4, în cadrul buclei for sunt citite şi memorate succesiv valorile celor 3 coeficienfi polinomiali. Mai întîi, valoarea corespunzătoare unui coeficient este citită cu funcţia **scanf** în variabila *val* şi apoi conţinutul acesteia este depus la adresa a+i, prin execuţia instrucţiunii **(a+i)=val;* La o primă analiză, am fi tentaţi să credem că a doua valoare, obţinută pentru i=1, se memorează la adresa conţinută în *a* plus 1, adică în al doilea octet al primei valori, care în acest fel se distruge. Acest lucru nu este adevărat, deoarece compilatorul, ştiind că tipul pointerului *a* este float, va interpreta expresia *a+i* ca fiind *a+i+sizeof(float)* şi deci cele 3 valori citite vor fi memorate corect. Valorile coeficienţilor astfel memorate sunt utilizate în aceeaşi manieră în cadrul instrucţiunii *val=*a*x*x+*(a+1)*x+*(a+2);* pentru a evalua valoarea polinomului.

Din acest exemplu înţelegem importanţa specificării tipului de pointer în instrucţiunile de declarare a acestora. O instrucţiune de forma **(ptr+i)* este interpretată ca **(ptr+i*sizeof(tip))* în care *tip* este tipul cu care a fost declarat pointerul *ptr*.

În finalul programului prezentat este utilizată instrucţiunea *free(a);* care are rolul de a elibera zona de memorie alocată. Dacă nu am utiliza această funcţie şi am executa programul în cadrul unei bucle, după un anumit timp se va obţine mesajul: "Memorie insuficienta" datorită consumării întregului spaţiu de memorie liberă prin apelul repetat al funcţiei **malloc**.

Mecanismul de alocare şi eliberare a memoriei poartă numele de **alocare dinamică** şi constituie una din facilităţile puternice oferite de limbajul C.

Pentru a pune în evidenţă acest lucru, generalizăm programul ex_7_4, astfel încît acesta să calculeze valoarea unui polinom al cărui grad este specificat de utilizator. În acest caz, numărul coeficienţilor variază de la o rulare la alta şi utilizarea variabilelor este practic imposibilă. În exemplul 7.5 este prezentat programul pentru calculul valorii unui polinom de gradul *n*, conform relatiei: $P(x) = a_0x^n+a_1x^{n-1}+...+a_n$ cu ajutorul următoarelor 2 funcţii:

funcţia **val_coef**, de tipul pointer double, primeşte ca parametru un întreg reprezentînd gradul polinomului. În cadrul acesteia se alocă spaţiul de memorie necesar şi sunt citite valorile coeficienţilor polinomiali. De această dată, alocarea memoriei se face prin apelul funcţiei **calloc**. Aceasta este similară cu funcţia **malloc**,singura deosebire constînd în faptul că are 2 parametri. Primul parametru specifică numărul de blocuri de memorie ce vor fi alocate, iar al doilea, lungimea în octeţi a unui bloc. Astfel, instrucţiunea *coef=(double*)calloc(grad,sizeof(double));* este identică cu instrucţiunea *coef=(double*)malloc(grad,sizeof(double));* în sensul că ambele alocă acelaşi spaţiu de memorie şi returnează adresa primului octet.

funcţia **val_poli**, de tipul double, primeşte ca parametri pointerul de tipul double returnat de funcţia **val_coef**, gradul polinomului şi valoarea lui *x*.

În scopul determinării valorii polinomului, mai întîi variabila *val* este iniţializată cu zero şi apoi, în cadrul unei bucle, este adăugată valoarea fiecărui termen adică, a_ix^{n-i}, prin instrucţiunea *val+=*(coef+i)*pow(x,(double)grad -i);*. Pentru evaluarea termenilor este folosită funcţia **pow** din biblioteca matematică. Această funcţie are următoarea formă generală de apel *pow(x,y);* şi returnează valoarea x^y. Cei doi parametri sunt de tipul double şi deaceea pentru a evalua pe x^{n-i} apelul funcţiei **pow** s-a facut utilizînd pentru al doilea parametru operatorul cast *(double)*.

Exemplul 7.5.

```
/* programul ex_7_5 */
#include <stdio.h>
#include <alloc.h>
#include <math.h>
double *val_coef(int);
douhle val_poli(douhle *,int ,double);
rnain( )
 {
double *a,x,val;
int n,i; clrscr( );
printf ("\nIntroduceti gradul polinomului >"); scanf("%d",&n);
a = val_coef(n);
printf("Introduceti valoarea lui x > "); scanf("%lf",&x);
val=val_poli(a,n,x); printf("valoarea polinomului P(%lf)=%lf",x,val);
free(a);     getch( );
 }
double *val_coef(int grad)
 {
int i;
douhle *coef,val;
if((coef=(double *) malloc(grad+1, sizeof(double)))==NULL)
 {
    printf("\nMemorie insuficienta");   exit( );
 }
printf ("Introduceti valorile coef icientilor\n");
for (i=0; i<=grad; i++)
 {
    printf ("a[%d] =", i);      scanf("%1f",aval);    *(coef+i) = val;
 }  return(coef );
 }
double val_poli(double *coef, int grad, double x)
 {
int i;
double val;
va1=0;
for (i=0; i<=grad; i++)
    val+=*(coef+i)*pow(x,(double)grad-i);
return (val);
 }
```

Din exemplele prezentate constatăm că, deşi utilizarea pointerilor pare mai dificilă, ea este indispensabilă în elaborarea unor programe de calcul de mari dimensiuni, datorită avantajelor pe care le oferă în gestionarea şi accesarea memoriei. De fapt, dificultatea utilizării pointerilor constă în confuziile ce sunt adesea generate de utilizarea caracterului "*", atît ca operator de înmulţire aritmetică, cît şi ca operator de indirectare. Semnificaţia lui este generată de context. Un alt aspect, care poate crea dificultăţi mai ales programatorilor fără prea multă experienţă, rezultă din faptul că prin utilizarea pointerilor se poate accesa orice zonă din memoria calculatorului, fără nici un fel de restricţie. Este deci posibil ca, printr-o utilizare eronată a pointerilor, să se distrugă zone din memorie în care se află date utile sau chiar secvenţe de program, consecinţele fiind imprevizibile. Situaţiile de acest gen sunt generate de utilizarea instrucţiunilor de atribuire indirectă, fie fără iniţializarea prealabilă a pointerului, fie prin utilizarea unor indici care generează depăşirea spaţiului de memorie alocat. De exemplu, dacă în programul ex_7_4 ar lipsi instrucţiunea *ptr_a=&a*, atunci

efectul instructiunii **ptr_a=a+10* ar fi imprevizibil. De asemenea, dacă în programul ex_7_4 s-a scris din greşeală instrucţiunea de deschidere a buclei for sub forma *for(i=0; i<=3; i++);* atunci se vor citi patru valori. Deoarece s-a alocat spaţiu doar pentru trei valori, cea de-a patra se va suprapune în meniorie peste o zonă care în acel moment poate fi utilizată de alte programe ale sistemului şi deci se produce blocajul.

În concluzie, utilizaţi pointerii pentru a exploata avantajele pe care aceştia le oferă, iar în caz de nereuşită verificaţi cu atenţie modul în care i-aţi folosit, gîndindu-vă la cele prezentate mai sus.

7.2. Tablouri. Tablouri unidimensionale (vectori)

Tablourile, similare matricelor din algebră, constituie colecţii de date de acelaşi tip, care pot ti accesate prin intermediul aceluiaşi nume de variabilă, folosind indici.

Pentru a înţelege modul în care se declară şi se utilizează tablourile în limbajul C, considerăm programul din exemplul 7.6.

Exemplul 7.6.

```
/* Programul ex_7_6 */
#iaclude <stdio.h>
void main (void)
 {
float temp[7];
float media=0;
int i;    clrscr( );
printf ("\n Introduceti temperaturile zilnice\n\n");
for (i=0; i<7; i++)
 {
    printf("Temperatura din ziua %1d ", i+1); scanf("%f ",&temp[i] );
    media+=temp[i];
 }
media/=7; printf("\nTemperatura medie %f",media); getch( );
 }
```

În cadrul acestui program ne propunem să citim de la tastatură 7 valori reale, care să reprezinte temperaturile înregistrate în cursul unei săptămîni, să le memorăm şi să le calculăm media.

Instrucţiunea *float temp[7];* declară variabila *temp* ca fiind un tablou unidimensional, de tipul float, şi rezervă spaţiul de memorie necesar. În general, declararea unui tablou unidimensional se face printr-o instrucţiune de forma:

```
tip nume_tablou[dimensiune];
```

compusă din:

tip care reprezintă tipul valorilor ce vor fi memorate în tablou. Acesta poate fi oricare dintre tipurile fundamentale de date (char, int, long, float, double) cu sau fără semn;

nume-tablou care reprezintă numele tabloului şi este format, ca orice nume de variabilă, dintr-un şir de caractere;

o pereche de paranteze drepte [ ] care semnalează compilatorului că variabila ce o precede este un tablou. În interiorul parantezelor se află dimensiunea tabloului, care are rolul de a preciza cîte variabile de acelaşi tip, numite **elemente**, va conţine acesta. Dimensiunea este o constantă întreagă, care se precizează fie direct, fie prin intermediul directivei **#define**.

Într-un tablou, fiecărui element îi este asociat un indice, prin intermediul căruia poate fi accesat. Numerotarea indicilor începe de la 0, iar accesarea se face folosind numele tabloului urmat de o pereche de paranteze drepte, între care se află indicele. Astfel, pentru accesarea elementului al doilea din tabloul *temp*, vom folosi sintaxa *temp[1]* şi nu *temp[2]*.

Fiecare element al tabloului constituie o variabilă şi deci are o valoare şi o adresă. Valoarea se obţine prin accesarea directă a acestuia folosind indicele asociat, iar adresa prin intermediul operatorului **&**. Astfel, *temp[1]* reprezintă valoarea celui de-al doilea element al tabloului *temp*, iar *&temp[1]* - adresa acestuia. În programul ex_7_5 am folosit adresele elementelor tabloului furnizate de expresia *&temp[i]* pentru memorarea valorilor citite cu funcţia **scanf** şi, respectiv, valorile acestora furnizate de expresia *temp[i]* pentru calculul mediei.

Întrucît microprocesorul lucrează cu adrese şi nu cu indici, în momentul în care declarăm o variabilă tablou, compilatorul va considera numele acesteia ca pointer, căruia îi atribuie valoarea adresei primului octet al zonei de memorie în care se memorează valorile elementelor, adică adresa primului element. Alocarea zonei de memorie se face de către compilator, dimensiunea acesteia fiind evaluată conform relaţiei *dimensiune*sizeof(tip)*, în care *dimensiune* este dimensiunea tabloului, iar *tip*, tipul acesteia. În concluzie, declararea şi utilizarea tablourilor este transformată de compilator într-un mecanism de alocare a memoriei, iar accesarea acesteia se face în mod indirect prin intermediul pointerilor. Avînd în vedere cele prezentate anterior, putem concluziona că expresia *nume_tablou[i]* este echivalentă cu **(nume_tablou+i)*, iar expresia *&nume-tablou[i]* este echivalentă cu *nume_tablou+i*sizeof(tip)*. Aceste reguli particularizate pentru tabloul *temp* folosit în programul ex_7_5 ne indică faptul că expresia *temp[i]* este echivalentă cu **(temp+i)*, iar expresia *&temp[i]* este echivalentă cu *temp+i*sizeof(float)*.

Deoarece elementele unui tablou constituie variabile obişnuite este posibila iniţializarea acestora în momentul declarării tabloului. Pentru a exemplifica acest lucru, considerăm programul din exemplul 7.7.

Exemplul 7.7.

```
/* Programul ex_7_7 */
#include <stdio.h>
#include <conio.h>
#define    N_VALORI   11
int valori[ ] = {10000,5000,1000,500,200,100,50,20,10,5,1);
void main(void)
 {
int suma,i;clrscr ();
printf ("\nIntroduceti o suma de bani ");    scanf ("%d",&suma);
```

```
for (i=0; i<N_VALORI; i++)
 {
    printf ("\nvaloarea bancnotei %4d Numarul de bancnote %4d",valori[i], suma/valori[i]);
    suma% = valori[i];
 }  getch( );
 }
```

Programul solicită utilizatorului să introducă o sumă de bani şi determină numărul de bancnote de 10000, 5000, 1000, 500, 200, 100, 50, 20, 10, 5, 1 necesare formării acestei sume. Pentru aceasta, este utilizat tabloul *valori* de tipul întreg. În corpul buclei for, pentru fiecare valoare a lui *i* se afişează valoarea corespunzătoare a bancnotei conținută în elementul *valori[i]* din tabloul *valori*, şi numărul acestora, obținut prin împărțirea sumei la valoarea afişată anterior. Suma este apoi modificată, capătînd valoarea rămasă (obținută cu ajutorul operatorului %). Cheia acestui program o constituie instrucțiunea de declarare a vectorului *valori*, în cadrul căruia se face şi inițializarea elementelor sale. Pentru aceasta, valorile pe care elementele tabloului le vor primi sunt specificate în interiorul unei perechi de acolade, precedată de semnul egal. Prima valoare este atribuită primului element, a doua celui de-al doilea ş.a.m.d. Astfel, valori[0] va primi valoarea 10000, valori[1] va primi valoarea 5000 ş.a.m.d., pînă la valori[10] care va primi valoarea 1. Se constată că în acest caz nu am specifcat dimensiunea tabloului. Aceasta este stabilită de cornpilator ca fiind egală cu numărul valorilor, separate prin virgulă, existente în interiorul acoladelor.

Un alt aspect pe care îl remarcăm este că declararea tabloului s-a făcut în afara funcției **main**, astfel că variabila *valori* este o variabilă globală.

Variabilele declarate în interiorul unei funcții sunt variabile locale (automatic). Ele sunt create în momentul apelării funcției şi sunt distruse în momentul în care funcția revine în programul apelant. Din acest motiv nu putem iniția variabilele locale unei funcții, deoarece încercăm să atribuim valori unor variabile care nu au fost create (nu li s-a rezervat spațiu în memorie). Cîteodată este nevoie de a iniția o variabilă simplă sau tablou în cadrul unei functii. Pentru a exemplifica acest aspect, rescriem programul ex_7_6, astfel încît să utilizăm funcția *nr_bancnote* pentru a determina numărul de bancnote de diverse valori necesar formării sumei.

Exemplul 7.8.

```
/* Programul ex_7_8 */
#include <stdio.h>
#define    N_VALORI   11
void nr bacnote(int);
main ()
 {
int suma;  clrscr( );
printf ("\nIntroduceti o suma de bani ");    scanf("%d",&suma);
nr_bacnote(suma);    getch( );
 }
void nr_bacnote (int x)
 {
int i;
static int valori[ ] = {10000,5000,1000,500,200,100,50,20,10,5,1);
for (i=0; i<N_VALORI; i++)
 {
    printf ("\nvaloarea bacnotei %4d Numarul de bacnote %4d",valori[i], x/valori[i]);
    x%=valori[i];
 }
 }
```

La declararea tabloului *valori* în funcția **nr_bancnote** s-a folosit prefixul **static**. Acesta este un cuvînt cheie al limbajului C, utilizat, aşa cum sugerează şi numele, pentru declararea variabilelor statice. Acestea sunt variabile declarate în interiorul funcțiilor, sunt create la lansarea în execuție a programului şi nu sunt distruse în momentul în care se părăseşte corpul funcției. Cu alte cuvinte, ele rămîn în memorie pe toată durata execuției programului, sunt similare variabilelor globale sau externe şi pot fi inițializate. Tipurile de variabile **extern**, **static** şi **automatic** constituie părți ale unui concept general, cunoscut în limbajul C sub numele **clase de memorare**. Consideram programul din exemplul 7.9, în care sunt utilizate tablourile ca parametri de funcții.

Exemplul 7.9.

```
/* Programul ex_7_9 */
#include <stdio.h>
#include <stdlib.h>
#define    DIM   10
void ordon(int[ ],int);
main( )
 {
int sir[DIM],nr_term,i;   clrscr( );
printf("\nIntroduceti numarul de termeni (<=10) ");   scanf("%d",&nr_term);
printf ("\n SIRUL INITIAL");   /* Genereaza termenii sirului */
f or (i=0; i<nr_term; i++)
 {
    sir[i]=rand( );      printf("\nsir[%2d] =%d",i,sir[i]);
 }
ordon(sir,nr_term);
printf("\n\n SIRUL ORDONAT"); /* Afiseaza sirul ordonat */
for(i=0; i<nr_term; i++)
```

```
    printf("\nsir[%2d]=%d",i,sir[i]);
getch ();
}
void ordon(int a[ ],int n)
{
int min,ind_min,i,j;
f or (i=0; i<n-l; i++)
{
    min=a[i];  ind_min=i;
    f or (j=i+1; j<n; j++)
    {
        if (min>a[j] )
        {
            min=a[j];  ind_min=j;
        }
    }
    if(ind_min!=i)
    {
        a[ind_min] = a[i];          a[i] = min;
    }
}
}
```

În cadrul programului este folosită funcția **ordon**, pentru ordonarea crescătoare a valorilor termenilor unui şir de numere. Aceasta are doi parametri de tipul întreg. Primul este un tablou ce conține valorile care vor fi ordonate, iar al doilea - un întreg ce reprezintă numărul de valori conținute în şir.

Într-o instrucțiune de declarare a unei funcții, pentru a specifica că un parametru este tablou, se utilizează o construcție sintactică formată din cuvîntul cheie ce precizează tipul tabloului, urmat de un spațiu şi o pereche de paranteze drepte în interiorul cărora nu se precizează dimensiunea. Astfel, instrucțiunea *void ordon(int[ ],int);* declară funcția **ordon** de tipul void (nu întoarce nici o valoare) ca avînd primul parametru un tablou de întregi, iar al doilea parametru un întreg obişnuit. La definirea unei funcții pentru a specifica faptul că un parametru formal este un tablou, se utilizează o pereche de paranteze drepte între care nu se precizează dimensiunea tabloului.

Avînd în vedere că numele tabloului este de fapt un pointer, la apelul funcției, parametrul formal de tipul tablou va prelua adresa tabloului ce constituie parametrul actual. Deci, prin utilizarea parametrilor de tipul tablou, unei funcții nu i se transmit valorile elementelor tabloului, ci doar adresa de memorie la care acestea sunt memorate. Cunoscînd adresa tabloului, funcția va comunica direct cu zona de memorie a funcției care o apelează şi din acest motiv nu este necesară dimensionarea tabloului ce constituie parametrul formal. Dimensionarea tabloului se face în funcția apelantă, iar funcției care îl utilizează ca parametru formal i se transmite adresa primului element prin intermediul numelui. Astfel, prin instrucțiunea *ordon(sir,nr_elemente)* funcția **ordon** preia din funcția **main** adresa tabloului *sir* şi numărul de elemente.

Algoritmul de ordonare cuprinde 2 bucle. Prima buclă, avînd ca variabila de control pe *i*, stabileşte poziția curentă în şir. În a doua buclă, avînd ca variabilă de control pe *j*, se determină elementul minim din şirul valorilor *a[i],a[i+1],...,a[n]*, precum şi poziția acestuia. Aceste informații sunt memorate în variabilele locale *min* şi *ind_min*. La sfîrşitul buclei interioare, înainte de a se trece la poziția următoare, se modifică şirul schimbînd între ele valorile *a[i]* şi *a[ind_min]*, doar dacă *ind_min* este diferit de *i*. Ordonarea şirului este terminată după efectuarea a n-1 paşi în bucla exterioară. Acest algoritm poate fi utilizat şi pentru ordonarea descrescătoare a şirului, prin simpla schimbare a expresiei logice a primei instrucțiuni if, care devine: *min < a[j]*.

Pentru generarea valorilor elementelor şirului în funcția **main**, s-a utilizat funcția **rand**. Aceasta este o funcție de bibliotecă definită în fişierul antet **stdlib.h**, destinată generării de numere aleatoare.

7.3. Tablouri multidimensionale

Limbajul C permite şi utilizarea tablourilor cu mai multe dimensiuni. Cele mai folosite sunt tablourile cu 2 dimensiuni, similare matricelor din algebră. Pentru a exemplifica modul în care sunt utilizate tablourile multidimensionale vom considera programul din exemplul 7.10.

Exemplul 7.10.

```
/* Programul ex_7_10 */
#include <stdio.h>
int a[3] [2] = {{1,2}, {-1,1}, {5,7}};
int b[2] [2] = {{-1,0},{0,-1}};
int c[3] [2];
void prod_mat(int,int,int);
main ()
{
int i,j;  clrscr( );
printf ("\nMatricea A");
f or (i=0; i<3; i++)
{
    printf ("\n");
    for (j=0; j<2; j++)
        printf ("%2d", a[i][j]);
}
```

```
printf (“\nMatricea B”);
for (i=0; i<2; i++)
 {
    printf(“\n”);
    f or (j=0; j<2; j++)
        printf (“ %2d”,b[i][j] );
 }
prod_mat(3,2,2);
printf(“\nMatricea produs A*B”);
for(i=0;i<3;i++)
 {
    printf(“\n”);
    for (j=0; j <2; j++)
        printf (“ %2d”,c[i][j ]);
 }  getch( );
 }
void prod_mat(int m,int n,int 1)
 {
int i,j,k;
for (i=0; i<m; i++)
    for (j=0; j<l;j++)
     {
        c[i][j] = 0;
        for (k=0; k<n; k++)
            c[i][j] += a[i][k] * b[k][j];
     }
 }
```

În cadrul programului este definită şi utilizată funcția **prod_mat** pentru înmul.irea matricei *A*, avînd 3 linii şi 2 coloane, cu rnatricea *B*, avînd 2 linii şi 2 coloane. Rezultatul este depus în matricea *C(3×2)*. În prima parte a programului sunt declarate tablourile bidimensionale de tipul întreg *a*, *b*, *c*. În general, pentru a declara o variabilă de tipul tablou cu 2 dimensiuni (matrice), se foloseşte o instrucțiune de forma:

```
tip nume_tablou[nr_linii][nr_coloane];
```

Aceasta este similară cu instrucțiunea folosită pentru declararea tablourilor unidimensionale, singura deosebire constînd în faptul că se utilizează două perechi de paranteze drepte: una pentru specificarea numărului liniilor, iar a doua - numărului coloanelor matricei. Astfel, instrucțiunea *int c[3][2];* declară variabila *c* matrice (tablou) de tipul întreg, avînd 3 linii şi 2 coloane. Tablourile bidimensionale pot fi initializate în momentul declarării. Pentru aceasta se foloseşte o pereche de acolade în interiorul cărora sunt plasate alte perechi de acolade separate prin virgulă, ce conțin elementele liniilor matricei. Astfel, primele instrucțiuni din programul ex_7_10 inițializează matricile *a* şi *b*, cu valorile dorite.

Accesarea elementelor din matrice se face cu ajutorul unei perechi de indici în care primul este indicele liniei, iar al doilea este indicele coloanei. Numerotarea acestora, ca şi în cazul tablourilor unidimensionale, se face începînd cu 0. Astfel, elementul a_{21} va fi accesat prin *a[1][0]* şi nu prin *a[2][1]*. Avînd în vedere acest mod de utilizare a indicilor precum şi regula de calcul al elementelor matricei produs, în cadrul funcției **prod_mat** sunt utilizate trei bucle **for**. Prima buclă stabileşte indicele *i* al liniei primei matrice, a doua indicele *j* al coloanei celei de-a doua matrice, iar în ultima buclă se calculează elementul c_{ij} al matricei produs, prin însumarea produselor dintre elementele omoloage de pe linia *i* şi coloana *j*.

La declararea unei variabile de tipul matrice, compilatorul va aloca un spațiu de memorie, avînd dimensiunea exprimată în octeți dată de relația *nr_linii*nr_coloane*sizeof(tip),* şi tratează numele tabloului ca un pointer căruia îi atribuie adresa primului octet din memoria alocata. Acest pointer este utilizat de limbajul C pentru accesarea elementelor tabloului deoarece, aşa cum am precizat, microprocesorul nu lucrează cu indici, ci cu adrese de memorie.

Pentru a explica modul în care sunt folosiți pointerii şi mecanismul de indirectare pentru accesarea elementelor unui tablou bidimensional (matrice), considerăm un program de calcul în cadrul căruia se realizează înmulțirea unei matrice cu un scalar. În exemplul 7.11 este prezentată o variantă a acestui program în care înmulțirea elementelor matricei *mat(4×4)* cu scalarul *alfa* a cărui valoare este 10, se realizează în cadrul a 2 bucle for, fiecare element al matricei fiind accesat prin intermediul indicilor.

Exemplul 7.11.

```
/* Programul ex_7_11 */
#include <stdio.h>
main( )
 {
int mat[4][4]={{1,3,5,7},{2,4,6,8},{-1,-2,-3,-4},{-5,-6,-7,-8}};
int alfa=l0,i,j;
clrscr( );   printf (“\n MATRICEA INITIALA ”);
f or (i=0; i<4; i++)
 {
    printf (“\n”);
```

```
    f or (j=0; j<4; j++)    printf(" %3d",mat[i][j]);
 }
for(i=0; i<4; i++)
    for (j=0; j<4; j++)
        mat[i][j] *= alfa;
printf ("\n MATRICEA MODIFICATA");
for(i=0; i<4; i++)
 {
    printf("\n");
    f or (j=0; j<4; j++)
        printf(" %3d",mat[i][j]);
 }  getch( );
 }
```

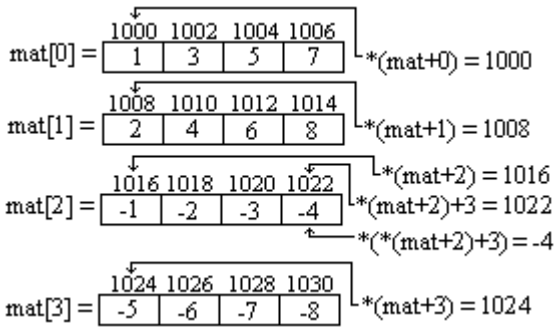

Figura 2.

Pentru accesarea elementelor prin intermediul pointerilor, compilatorul C consideră fiecare linie a matricei ca un element al unui tablou unidimensional la care se poate face referire printr-un singur indice. Astfel, *mat[0]* desemnează prima linie a matricei, *mat[1]* a doua linie ş.a.m.d. Adresa unei linii *i* este adresa primului element al acesteia şi se obţine prin intermediul unei expresii de indirectare de forma **(mat+i)*. Adresa elementului *mat[i][j]* este furnizată de expresia **(mat+i)+j*, iar valoarea acestuia se obţine prin indirectare folosind expresia **(*(mat+i)+j)*.

În figura 2 este prezentat schematic modul de accesare a elementului *mat[2][3]* al matricei din programul ex_7_11, considerînd că adresa primului octet al zonei de memorie în care aceasta este memorată are valoarea 1000.

Mecanismul de accesare a elementelor unui tablou bidimensional cu ajutorul pointerului asociat numelui acestuia poartă numele **dublă indirectare**. Pentru exemplificare, în exemplul 7.12 este prezentată o altă variantă a programului de înmulţire a matricei *mat* cu scalarul *alfa* în cadrul căreia este folosit acest mecanism.

Exemplul 7.12.

```
/* Programul ex_7_12 */
#include <stdio.h>
main( )
 {
int mat[4][4] = {{1,3,5,7}, {2,4,6,8}, {-1,-2,-3,-4},{-5,-6,-7,-8}};
int alfa=10, i, j;
clrscr( );   printf ("\n MATRICEA INITIALA ");
for (i=0; i<4; i++)
 {
    printf ("\n");
    f or (j=0; j<4; j++)
        printf (" %3d",*(*(mat+i)+j));
 }
for (i=0; i<4; i++)
    for (j=0; j<4; j++)
        *(*(mat+i)+j) *= alfa;
printf ("\n MATRICEA MODIFICATA");
for(i=0; i<4; i++)
 {
    printf("\n")
    for (j=0; j<4; j++)
        printf(" %3d",*(*(mat+i)+j));
 }  getch( );
 }
```

7.4. Şiruri de caractere

Şirurile de caractere sunt tipuri de date folosite pentru manevrarea textelor (nume, cuvinte, propoziţii sau fraze). Limbajul C tratează şirurile de caractere ca tablouri unidimensionale de tipul caracter şi oferă utilizatorului un set complet de funcţii pentru manevrarea acestora. Pentru a exemplifica modul în care şirurile de caractere sunt citite de la tastatură, stocate în memoria internă şi afişate pe ecran, considerăm programul din exemplul 7.13.

Exemplul 7.13.

```
/* Pzogramul ex_7_13 */
#include <stdio.h>
#include <string.h>
void main(void)
 {
char nume[50];    clrscr( );
puts("Numele >");    gets(nume);
printf ("Salut %s, Numele tau are %3d caractere",nume,strlen(nume)); getch( );
 }
```

La începutul programului este declarat tabloul *nume* de tipul caracter, în care se va memora şirul de caractere ce reprezintă numele utilizatorului.

Pentru scrierea, respectiv citirea şirurilor de caractere, în limbajul C se utilizează funcţiile **puts** şi **gets**. Acestea sunt definite în fişierul antet **stdio.h** şi au ca unic parametru un şir de caractere, care poate fi o constantă sau o variabilă. **O constantă de tipul şir de**

caractere este o succesiune de caractere delimitată de o pereche de ghilimele, în timp ce o **variabilă de tipul şir de caractere** este un nume de tablou unidimensional de tipul caracter. Astfel, în programul ex_7_13 "Numele >" este o constantă şir de caractere, iar *nume* este o variabilă şir de caractere.

În memoria internă, fiecare caracter al unei constante sau al unei variabile de tipul şir de caractere ocupă un octet. Pentru a marca terminarea şirului, compilatorul adaugă la sfîrşitul acestuia secvența escape "\0".

Pentru memorarea numelui se utilizează doar o parte din cei 50 de octeți ai tabloului *nume*. Deoarece nu există restricții de accesare a memoriei, atunci cînd declarăm o variabilă şir de caractere (un tablou de tip caracter), aceasta trebuie dimensionată la valoarea maximă a lungimii şirurilor pe care anticipăm că i le vom atribui, plus un octet pentru secvența escape "\0" adăugată automat. Astfel, dacă în exemplul anterior am declara variabila *nume[10]*, atunci ultimele caractere ale numelui mai lung decît 10 s-ar suprapune peste o zonă de memorie utilizată şi rezultatul execuției programului ar fi imprevizibil.

Ultima instrucțiune a programului ex_7_13 este un apel la funcția **printf** care va tipări un mesaj conținînd un salut, numele utilizatorului şi lungimea şirului de caractere ce-l alcătuiesc. Pentru a determina lungimea şirului, se foloseşte funcția **strlen** definită în fişierul antet **string.h**. Aceasta primeşte ca parametru un şir de caractere şi returnează lungimea acestuia în octeți.

Într-un şir de caractere, care este un tablou, fiecare caracter poate fi accesat independent fie prin intermediul indicilor, fie prin intermediul pointerului asociat şi al mecanismului de indirectare.Pentru a ilustra acest fapt considerăm programul din ex.7.14.

Exemplul 7.14.

```
/* Pzogramul ex_7_14 */
#include <stdio.h>
#include <string.h>
void main(void)
 {
char fraza[81],litera,*p_sir;
int poz;     clrscr( );
printf("Introduceti o fraza >");  gets(fraza);
printf ("Introduceti litera gresita >");    litera = getche( );
p_sir = strchr(fraza,litera);   poz = p_sir-fraza;
strcpy(&fraza[poz],&fraza[poz+1]);
printf ("\nFraza cautata : ");      puts(fraza);    getch( );
 }
```

La începutul programului se solicită introducerea unei fraze şi se consideră că din greşeală, în corpul frazei s-a tastat o literă în plus care trebuie eliminată. Pentru aceasta se solicită tastarea literei greşite, care este citită cu ajutorul funcției **getche** şi memorată în variabila *litera*. Pentru a depista poziția acesteia în şirul de caractere se foloseşte funcția **strchr**. Această funcție primeşte ca argumente şirul în care se caută şi caracterul urmărit, returnînd un pointer ce reprezintă adresa primei apariții a caracterului în şir. Poziția acestuia se determină scăzînd din pointerul returnat valoarea adresei la început a şirului furnizată de numele acestuia. Astfel prin instrucțiunea *poz=p_sir-fraza;* se scad 2 pointeri şi se obține o valoare întreagă ce reprezintă indicele caracterului căutat.

Pentru eliminarea caracterului este utilizată funcția **strcpy**, care are 2 argumente de tipul pointer la şir de caractere şi realizează copierea caracter cu caracter a şirului reprezentat de al doilea pointer numit **sursă**, în şirul reprezentat de primul numit **destinație**. Astfel, în exemplul anterior, pentru eliminarea caracterului *i* din poziția *poz=1005-1000=5* este necesară transatarea la stînga a tuturor caracterelor şirului *fraza* începînd din poziția 6. Pentru aceasta în apelul funcției **strcpy** s-a folosit ca pointer de destinație adresa elementului cu indicele 5, obținută cu expresia *&fraza[poz]*, iar ca pointer sursă s-a folosit adresa elementului cu indicele 6, obținută cu expresia *&fraza[poz+1]*. În finalul programului, prin apelul funcției **puts**, este afişată fraza corectată.

În numeroase situații este necesară stocarea şi memorarea mai multor şiruri de caractere cum ar fi o listă de nume. Pentru aceasta, se folosesc tablourile de tipul şir de caractere, care de fapt constituie matrice (tablouri bidimensionale) de tipul caracter.

Pentru exemplificare considerăm programul din exemplul 7.15, în cadrul căruia se utilizează un tablou de pointeri la caracter şi un tablou de tip şir de caractere, în vederea memorării unei liste de nume şi afişarea acestora în ordine alfabetică.

Exemplul 7.15.

```
/* Pzogramul ex_7_15 */
#include <stdio.h>
#include <string.h>
#define     MAX  30
void main(void)
 {
char nume[MAX][50], *p_nume[MAX], *min_nume;
int nr_nume=0, i, j, ind_min;     clrscr( );
while (nr-nume<MAX)
 {
    printf("Numele %2d: ",nr_nume+1);    gets(nume[nr_nume]);
    if (strlen(nume[nr_nume] == 0)break;
    p_nume[nr_nume++] = nume[nr_nume];
 }
for (i=0; i<nr_nume-1; i++)
 {  /* ordonare alfabetică */
    ind_min = i;   min_nume = p_nume[i];
    for (j=i+1; j<nr_nume; j++
        if (strcmp(p_nume[j],min_nume)<0)
         {
            ind_min = j;   min_nume = p_nume[j];
         }
```

```
        if (ind_min != i)
         {
            p_nume[ind_min] = p_nume[i];      p_nume[i] = min_nume;
         }
    }  /* afisare lista ordonata */
    puts(" LISTA ORDONATA ALFABETIC");
    for (i=0; i<nr_nume; i++)
        printf ("\nNumele %2d: %s",i+1,p_nume[i];   getch( );
    }
```

În cadrul programului s-a considerat că lungimea maximă a listei de nume este 30. Această valoare poate fi modificată prin simpla schimbare a valorii atribuite constantei MAX din directiva **#define**.

Numele ce alcătuiesc lista inițială sunt citite în cadrul unei bucle **while** care se termină fie în momentul în care valoarea variabilei *nr_nume* a atins valoarea maximă, fie în momentul în care lungimea şirului de caractere introdus de la tastatură este nulă. Aceasta se întîmplă cînd la solicitarea introducerii numelui se apasă tasta <Enter> fără a tasta în prealabil o literă. Solicitarea terminării forțate a introducerii de nume în listă este detectată în program prin intermediul instrucțiunii **if** care are în expresia logică apelul funcției **strlen**.

Deşi *nume* este un tablou bidimensional (matrice), în cadrul programului acesta este utilizat folosind un singur indice. Prin utilizarea unui singur indice asociat cu numele unei matrice este posibilă liniile acesteia sunt tratate ca elemente ale unui tablou unidimensional. Deci, *nume[i]* este pointerul la şirul de caractere ce va fi memorat pe linia *i* a matricei *nume*. Cu alte cuvinte, instrucțiunea *gets(nume[nr_nume]);* va prelua de la tastatură numele cu numărul de ordine *nr_nume+1* şi îl va depune pe linia *nr_nume* din matrice, iar instrucțiunea *p_nume[nr_nume++]=nume[nr_nume];* atribuie elementului *p_nume[nr_nume]* din tabloul de pointeri *p_nume* valoarea adresei liniei *nr_nume*. După atribuire, valoarea contorului este incrementată, adică instrucțiunea anterioară este echivalenta cu instrucțiunile:

p_nume[nr_nume] = nume[nr nume); nr_nume++;

Presupunem că se lansează în execuție programul şi se introduce următoarea listă de 5 nume:

Nume1: Diana
Nume2: Sergiu
Nume3: Andrei
Nume4: Oxana
Nume5: Monica

şi cu adresa tabloului *nume* este 2000. În figura 3.a este prezentat conținutul tabloului de pointeri *p_nume* şi conținutul fiecărei linii a matricei *nume* la terminarea buclei **while**.

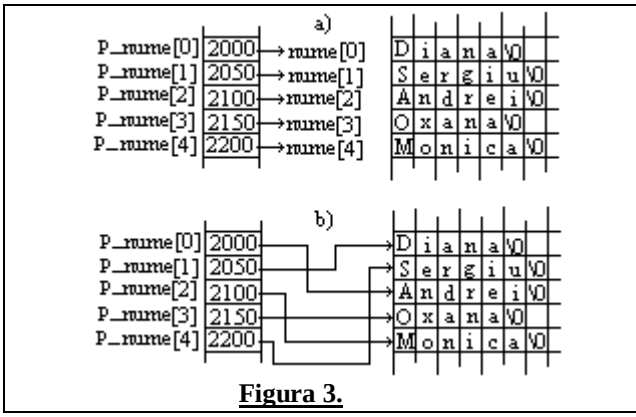

Figura 3.

Algoritmul de ordonare este identic cu cel descris la ordonarea unui şir de numere, deosebirea constînd în folosirea funcției **strcmp** în expresia logică a instrucțiunii de decizie care determină elementul minim. Această funcție a cărei formă generală de apel este: *strcmp(sir2,sir1);* compară şirul din dreapta (*sir1*) cu cel din stînga (*sir2*) şi furnizează: o valoare negativă, dacă sir1<sir2; 0, dacă sir1= sir2 (şiruri identice); o valoare pozitivă, dacă sir1>sir2.

Pentru ordonare este utilizat tabloul de pointeri *p_nume* ce conține adresele numelor şi nu matricea *nume*. Astfel, pentru fiecare valoare a lui *i* din bucla exterioara se determină, în cadrul buclei interioare, elementul minim din şirurile de caractere pointate de catre *p_nume[i], p_nume[i+ 1], ... ,p_nume[nr_nume]* precum şi poziția acestuia. Aceste informații sunt memorate în variabilele *min_nume* şi *ind_min*. La sfîrşitul buclei interioare, dacă este cazul, se modifică conținutul şirului de pointeri schimbînd între ele valorile *p_nume[i]* şi *min_nume*.

După terminarea procesului de ordonare este modificat doar tabloul pointerilor la liniile matricei astfel ca acestea sunt accesate în ordine alfabetică, aşa cum se poate constata şi din figura 3.b.