8. STRUCTURI ŞI UNIUNI

Pînă acum pentru memorarea şi utilizarea datelor am folosit variabilele simple şi tablourile. Aceste 2 mecanisme de manevrare a datelor în cadrul unui program de calcul sunt limitate de faptul că permit tratarea numai a datelor de acelaşi tip. Există însă situaţii în care este necesar ca date de tipuri diferite să fie tratate împreună. De exemplu, considerăm că este necesar ca în cadrul unui program de calcul să utilizăm datele referitoare la un angajat dintr-o societate comercială. În acest caz, trebuie să memorăm numele angajatului (un şir de caractere), numărul legitimaţiei de serviciu (un întreg), salariul (un real) şi alte date care să poată fi considerate ca o entitate. Apare deci necesitatea unui mecanism care să permită definirea şi utilizarea unor tipuri complexe de date, adică să permita memorarea la un moment dat a mai multor date de diverse tipuri ce pot fi tratate fie individual, fie ca o entitate. Acest deziderat, care a constituit punctul de plecare pentru dez-voltarea tehnicilor de programare orientate pe obiect, se poate realiza în limbajul C cu ajutorul structurilor.

În alte situaţii, apare necesitatea ca o aceeaşi zonă de memorie să fie accesată la momente diferite de timp, în contexte diferite. De exemplu, considerăm că la începutul unui program dorim ca o anumită zonă de memorie să fie accesata prin intermediul unei variabile de tip întreg, urmînd ca ulterior aceeasi zonă de memorie să fie accesată prin intermediul unei variabile de tip real. Acest mecanism de utilizare a memoriei este posibil în limbajul C prin utilizarea uniunilor.

8.1. Structuri

O **structură** constă dintr-un număr oarecare de date, care pot fi de acelaşi tip sau de tipuri diferite, grupate împreună. Pentru exemplificare considerăm programul ex_8_1, în cadrul căruia folosim o structură. Datele care alcătuiesc structura (de tipuri diferite) sunt destinate memorării informaţiilor referitoare la un candidat la un concurs de admitere şi constau din:

- un şir de caractere pentru memorarea numelui candidatului;
- un întreg pentru memorarea numărului legitimaţiei de concurs;
- trei reali pentru memorarea notelor obţinute la cele 2 probe de concurs şi a mediei finale.

Exemplul 8.1.

```
/* Programul ex 8 1 */
#include <string.h>
void main (void)
 {
struct candidat
 {
    char nume[80];    int nr_leg;
    float notal;     float nota2;    float media;
 }
struct candidat x;
strcpy(x.nume,"Popescu I Victor Andrei");
x.nr_leg = 132;    x.nota1= 6.75;     x.nota2= 8.25;
x.media = (x.notal+x.nota2)/2;
 }
```

Acest exemplu evidenţiază 3 aspecte fundamentale în utilizarea structurilor, şi anume: definirea structurii, declararea variabilelor de tip structură şi accesarea elementelor structurii.

8.1.1. Definirea structurilor

În cazul tipurilor fundamentale de date (caracter, întreg, real), în urma declarării variabilelor, compilatorul C rezervă în mod implicit spaţiul de memorie necesar (1,2,4 sau 8 O). Deoarece o structură poate conţine un număr oarecare de date, pentru rezervarea spaţiului de memorie, compilatorul trebuie informat despre tipul datelor care alcătuiesc structura înainte de utilizarea acesteia. Setul de instrucţiuni:

```
struct candidat
 {
char nume[80];    int nr_leg;
float notal;     float nota2;    float media;
 }
```

din programul ex_8_1 constituie un exemplu de definire a unei structuri. Scopul acesteia îl constituie definirea unui nou tip de date numit **structură candidat**. Forma generală a instrucţiunii de definire a unei structuri este:

```
struct nume_structură
 {
elementele structurii
 }
```

şi constă din:

- cuvîntul cheie **struct** urmat de numele structurii. În exemplul prezentat, numele structurii este *candidat*.
- o pereche de acolade între care sunt declarate elementele structurii. Declararea elementelor structurii se realizează cu ajutorul instrucţiunilor de declarare a variabilelor simple sau de tipul tablou.
- caracterul ";" care marchează sfîrşitul instrucţiunii de definire a structurii.

Numele structuri nu este un nume de variabilă, ci un nume de tip de date care se mai numeşte şi **eticheta structurii**.

Aşadar, putem defini o structură ca fiind un nou tip de date al cărui format este stabilit de programator prin intermediul unei instrucţiuni de definire.În cadrul unui program de calcul se poate utiliza un număr oarecare de instrucţiuni de definire a structuri-lor. Acestea vor fi plasate la începutul funcţiei în cadrul căreia se utilizează noile tipuri de date, înaintea declaraţiilor de variabile. În cazul

în care fişierul sursă conţine mai multe funcţii în cadrul cărora dorim să utilizăm noile tipuri de date, instrucţiunile de definire a structurilor se vor plasa la începutul fişierului în afara oricărei funcţii. În acest caz, se recomandă plasarea instrucţiunilor de definire a structurilor într-un fişier antet şi includerea acestuia în fişierul sursă cu ajutorul directivei **#include**.

8.1.2. Declararea variabilelor de tip structură

Noile tipuri de date create prin mecanismul de definire a structurilor pot fi utilizate în cadrul unui program de calcul prin intermediul variabilelor. Ca orice tip de variabilă şi variabilele de tipul structură trebuie declarate înainte de a fi utilizate. În programul ex_8_1 instrucţiunea *struct candidat x;* declară variabila *x* de tipul structură candidat şi are ca efect alocarea spaţiului de memorie necesar stocării membrilor săi. Forma generală a unei instrucţiuni de declarare a variabilelor de tipul structură este:

```
struct nume_structura lista_de_variabile;
```

şi constă din:

- cuvîntul cheie **struct** urmat de numele structurii (eticheta acesteia);
- lista variabilelor formată din nume de variabile separate prin virgula.

Limbajul C permite comprimarea într-o singură instrucţiune a instrucţiunilor de definire a structurii şi de declarare a variabilelor de tip structură. Acest lucru se realizează folosind lista de variabile înaintea caracterului ";" care marchează sfîrşitul instrucţiunii de definire a structurii. Astfel, în programul ex_8_1 instrucţiunea *struct candidat x;* poate fi eliminată rescriind instrucţiunea de definire a structurii sub forma:

```
struct candidat
 {
char nume[80];    int nr_leg;
float notal;      float nota2;    float media;
 }x;
```

Forma generală a unei instrucţiuni compuse prin intermediul căreia se defineşte o structură şi se declară un număr oarecare de variabile de tipul structurii definite este:

```
struct nume structura
 {
instrucţiuni de declarare a membrilor structurii
 } lista variabilelor de tip structură;
```

8.1.3. Accesarea elementelor structurii

Pentru accesarea elementelor individual se utilizează operatorul punct "**.**" care se mai numeşte şi **operator membru**. Acesta are rolul de a efectua conexiunea între un nume de variabilă de tipul structură şi un element membru al acesteia. Astfel, instrucţiunea *x.nota1=6.75;* din programul ex_8_1 atribuie valoarea 6.75 elementului *nota1* al variabilei *x*.

Mecanismul de accesare a elementelor unei variabile de tipul structură poate fi utilizat atît în cadrul expresiilor (de atribuire, aritmetice, logice), cît şi în cadrul parametrilor funcţiilor de citire/scriere. Pentru exemplificare, considerăm programul ex_8_2 în cadrul căruia se citesc de la tastatură informaţiile referitoare la candidaţii la concursul de admitere, se calculează media şi se afişează rezultatul sub forma unui mesaj de tipul *admis* sau *respins*.

Exemplul 8.2.

```
/* Programul ex_8_2 */
 #include <stdio.h>
#include <conio.h>
#include <string.h>
void main(void)
 {
struct candidat
 {
    char nume[80];    int nr_leg;
    float notal,nota2,media;
 } x;
while(1)
 {  clrscr( );
    printf (" Numele candidatului: ");   fflush(stdin);  gets(x.nume);
    if (strlen(x.nume)==0)    exit(0);
    printf("Numarul legitimatiei de concurs: ");    scanf("%d",&x.nr_leg);
    printf("Notele obtinute: ");        scanf("%f %f",&x.notal,&x.nota2);
    x.media = (x.notal+x.nota2)/2;
    if(x.media>=s)    printf ("%s admis",x.nume);
    else              printf("%s respins",x.nume);
    getch( );
 }
 }
```

După definirea structurii candidat şi declararea variabilei *x* de tipul structură candidat datele sunt citite de la tastatură folosind funcţiile **gets** şi **scanf** în cadrul unei bucle infinite. Instrucţiunile sunt similare cu cele folosite pentru citirea şi scrierea variabilelor simple, deosebirea constînd în utilizarea operatorului punct pentru accesarea elementelor variabilei *x*. Astfel, prin expresia *x.nume* este accesat primul element al structurii care, de fapt, constituie pointerul la şirul de caractere în care se va memora numele candidatului. Avînd în vedere acest aspect, înţelegem că prin instrucţiunea *gets(x.nume);* este preluat de la tastatură numele candidatului şi depus în structură.

Pentru a citi valori cu ajutorul funcției **scanf**, acesteia trebuie să-i transmitem în lista parametrilor adresele variabilelor în care se vor memora valorile. Şi în cazul structurilor, adresa unui element se obține folosind tot operatorul de adresă **&** plasat de data aceasta în fața expresiei ce desemnează membrul structurii. Rezultatele obținute în cadrul programului de calcul ex_8_2 sunt afişate pe ecran folosind funcția **printf** şi modul de accesare a membrilor structurii.

Terminarea programului se realizează prin apăsarea tastei <Enter>, la solicitarea numelui candidatului, care semnifică introducerea unui şir de caractere cu lungimea 0. Acest fapt este testat prin intermediul instrucțiunii de decizie **if**, care foloseşte în expresia sa logică funcția **strlen**, având ca parametru elementul nume al variabilei *x* de tipul structură.

Din exemplul prezentat putem concluziona că fiecare element al unei structuri este interpretat de compilatorul C ca fiind o variabilă obişnuită.Conținutul acestuia(valoarea memorată)este obținut printr-o expresie de forma:*nume_structură.nume_element* iar adresa, printr-o expresie de forma: *&nume_structură.nume_element*.

În cadrul programului ex_8_2, înaintea instrucțiunii de citire a numelui candidatului s-a utilizat instrucțiunea *fflush(stdin);*. Pentru a explica rolul acesteia, precizăm faptul că introducerea datelor de la tastatură se realizează prin intermediul unei zone de memorie, numita **zonă tampon** sau **buffer**, în care sunt memorate codurile tastelor apăsate. Preluarea datelor de către program se face prin intermediul funcției **scanf**, în conformitate cu descriptorii de format. Aceasta nu va prelua caracterul "*newline*" care încheie o secvență de date, astfel că la urmatoarea citire a unui şir de caractere acesta este interpretat ca terminator, realizîndu-se o citire falsă. Este, deci, necesară eliberarea bufferului asociat dispozitivului de intrare (tastatura) prin apelul funcției *fflush(stdin)*, definită în fişierul antet **stdio.h**.

8.1.4. Structuri înlănțuite. Inițializarea structurilor

Două dintre facilitățile importante oferite de limbajul C în lucrul cu structuri constau în utilizarea unei structuri ca element al altei structuri (structuri îmlănțuite) şi inţtializarea structurilor. Pentru exemplificare, se consideră programul din exemplul 8.3.

Exemplul 8.3.

```
/* Programul ex_8_3 */
#include <stdio.h>
struct membru
 {
char nume [80];
int nr_ore_zi;
float sal ora;
 };
struct echipa
 {
struct membru sef;    struct membru munc1;    struct membru munc2;
int nr_zile;
 };
void main (void)
 {
struct echipa st= {{"Cabac Eugen",8,425.50}, {"Migali Elina",6,375.25), {"Rusnac Oxana",7,315.30},4};
float suma;     clrscr( );
printf ("SUMELE CUVENITE MEMBRILOR ECHIPEI");
suma=st.sef.nr_ore_zi*st.sef.sal_ora*st.nr_zile;              printf ("\n 1. %s %12.2f lei.",st.sef.nume,suma);
suma=st.muncl.nr_ore_zi*st.munc1.sal_ora*st.nr_zile;   printf("\n 2. %s %12.2f lei.",st.muncl.nume,suma);
suma=st.munc2.nr_ore_zi*st.munc2.sal_ora*st.nr_zile;printf("\n 3. %s %12.2f lei.",st.munc2.nume,suma);
getch ();
 }
```

Programul este destinat calculării sumelor cuvenite membrilor unei formații de lucru constituită din 3 persoane (un şef de echipă şi doi muncitori), care efectuează o lucrare într-un anumit număr de zile. Pentru aceasta, în prima parte a programului sunt definite 2 structuri. Prima structură *membru* este destinată memorării informațiilor referitoare la un membru al echipei. Ea va conține numele persoanei (un şir de caractere), numărul de ore lucrate zilnic (un întreg) şi salariul pentru o oră lucrată (un real). A doua structură **echipa** conține trei variabile de tipul structură *membru* destinate memorării informațiilor referitoare la fiecare membru al echipei şi variabila de tipul întreg nr_zile în care se va memora numărul de zile în care s-a efectual lucrarea. Aceste 2 structuri au fost definite în afara funcției **main**, dar ele pot fi definite şi în interiorul acesteia.

Prima instrucțiune a funcției principale defineşte variabila *st* de tipul structură *echipa* şi ji atribuie un set de valori inițiale (o inițializează). După cum se poate constata, inițializarea unei structuri este similară cu inițializarea unui tablou. Este strict necesară păstrarea corespondenței între ordinea în care au fost declarați membrii structurii şi ordinea în care sunt plasate valorile acestora în interiorul acoladelor. În cazul variabilelor de tipul structuri înlănțuite se folosesc mai multe perechi de acolade pentru precizarea valorilor inițiale. Astfel, în exemplul prezentat, interiorul perechii de acolade ce delimitează valorile elementelor variabilei *st* conține încă trei perechi de acolade care delimiteazu valorile atribuite variabilelor *sef, munc1* şi *munc2* de tipul structură *membru*.

Ca urmare a instrucțiunii de inițializare, deoarece variabila *sef* este primul membru al variabilei *st*, membrii ei vor primi variabile conținute în prima pereche de acolade interioare. În conformitate cu acest mecanism, membrii variabilelor *munc1* şi *munc2* vor primi valorile conținute în interiorul următoarelor 2 perechi de acolade, iar variabila *nr_zile* va primi valoarea 4.

Pentru accesarea elementelor unei structuri se foloseşte operatorul punct, care are rolul de a conecta o variabila de tipul structură cu elementele sale. Astfel, expresia *st.sef* accesează primul element al variabilei *st* de tipul structură *echipa*. Acest element fiind la rîndul său o variabilă de tipul structură *membru*, pentru accesarea elementelor sale se foloseşte tot operatorul punct. Deci expresia *st.sef.nr_ore_zi*, folosită pentru evaluarea sumei cuvenite unui membru al echipei, accesează elementul *nr_ore_zi* al variabilei *sef*, de tipul structură *membru*.

În concluzie, pentru accesarea elementelor unei variabile *struct1* (de tipul structură) care este la rîndul ei membru al altei variabile *struct* (de tipul structură) se foloseşte o expresie de forma: *struct.struct1.membru_struct1,* în care *membru_struct1* este un

element al structurii *struct1*, care poate fi la rîndul său o variabilă simplă sau o altă structură. În cazul în care variabila *membru_struct1* este un tablou,pentru accesarea, elementelor acestuia se folosesc indicii. De exemplu,expresia: *st.munc.nume[2]* accesează al treilea caracter al şirului de caractere, în care se memorează numele primului muncitor din echipă.

În limbajul C, procesul de înlănţuire a structurilor este practic nelimitat dar, din motive de simplificare a expresiilor, se folosesc de regulă maximum 2 nivele de înlănţuire.

8.1.5. Redenumirea tipurilor de date

Prin intermediul cuvîntului cheie **typedef**, programatorul are posibilitatea să redenumească un tip de date existent. Forma generală de utilizare este:

```
typedef tip_date_existent tip_nou;
```

şi are ca efect atribuirea semnificaţiei tipului de date *tip_date_existent* identificatorului *tip_nou*. Astfel, printr-o declaraţie de forma: *typedef int intreg;* cuvîntul *intreg* devine cuvînt cheie avînd aceeaşi semnificaţie ca şi **int** şi deci poate fi utilizat în declararea variabilelor de tipul întreg.

Utilizarea acestei tehnici de redenumire a tipurilor de date conduce în numeroase situaţii la clarificarea programelor prin folosirea ca nume de tipuri de date a unor expresii mai scurte şi mult mai semnificative. Pentru exemplificare, considerăm următoarele instrucţiuni cu ajutorul cărora tipul de date **unsigned char** este redenumit **byte**, iar un tablou de caractere cu lungimea 256 O este redenumit **string**.

```
typedef unsigned char byte;
typedef char string[257];
```

Acum putem declara variabilele de tipul **unsigned char** şi pe cele de tipul tablou de caractere scriind: *byte var1,var2;* şi respectiv *string sir1,sir2;* in loc de: *unsiged char var1,var2;* şi *char sir1[257],sir2[257];*.

Redenumirea tipurilor de date este eficientă cînd se lucrează cu structuri. În astfel de situaţii, se recomandă folosirea fişierelor antet în care se defineşte şi apoi se redenumeşte structura folosind o declaraţie de tipul **typedef**. De exemplu, dacă intr-un program de calcul este necesară folosirea numerelor complexe, se va crea fişierul antet **complex.h** al cărui conţinut este:

```
struct complex
 {
double re,in;
 };
typedef struct complex complex;
```

Fişierul cuprinde instrucţiunea de definire a structurii care conţine 2 variabile de tipul real destinate memorării părţii reale şi respectiv a părţii imaginare a numărului complex şi instrucţiunea de redenumire a structurii. Cele 2 instrucţiuni pot fi comprimate într-o singură instrucţiune rescriind fişierul astfel:

```
typedef struct complex {double re,in;} complex;
```

Prin includerea fişierului antet **complex.h** într-un fişier sursă C, folosind directiva **#include**, declararea variabilelor *z1* şi *z2* de tipul complex se va face prin intermediul instrucţiunii: *complex z1,z2;*.

Această tehnică de programare se va utiliza şi în exemplele din programele următoare, în cadrul cărora se foloseşte fişierul antet **candidat.h** al cărui conţinut este:

```
typedef struct candidat
 {
char nume [80];   int nr_leg;
float nota1,nota2,media;
 } candidat;
```

Prin includerea acestui fişier în fişierele sursă C, declararea variabilelor de tipul structură *candidat* se va face scriind: *candidat lista_variabile;*.

8.1.6. Tablouri de structuri

Noile tipuri de date create prin mecanismul de definire şi redenumire a structurilor pot fi utilizate jn cadrul programelor C jn aceeaşi manieră în care sunt utilizate tipurile fundamentale de date. Vom prezenta modul de declarare şi de utilizare a tablourilor de structuri, operaţia de atribuire cu variabile de tipul structură şi utilizarea structurilor ca parametri ai funcţiilor. Pentru exemplificare, considerăm următorul program:

Exemplul 8.4.

```
/* Programul ex 8 4 */
#include <stdio.h>
#include "candidat.h"
#define     MAX_CAND200
candidat citire(int);
void main(void)
 {
candidat tab[MAX_CAND];     /* tablou de structuri */
candidat cand_x;
int nr_cand,i; clrscr( );
printf("Introduceti numarul de candidati >");  scanf("%d",&nr_cand);
for(i=0; i<nr_cand; i++)
 {
    cand_x =citire (i);     tab[i] = cand_x;
    if(tab[i].media >= 5)       printf ("%s admis", tab[i].nume);
    else                        printf ("%s respins",tab[i].nume);
```

```
    getch( );
 }
 }
candidat citire (int i)
 {  /* citire date candidat */
candidat x;    clrscr( );
printf (" DATELE CANDIDATULUI %d\n",i+1);    fflush(stdin);
printf("Numele candidatului ");          gets(x.nume);
printf ("Legitimatia de concurs: "); scanf("%d",&x.nr_leg);
printf("Nota la proba 1: ");             scanf("%f",&x.notal);
printf("Nota la proba 2: ");             scanf("%f",&x.nota2);
x.media = (x.notal+x.nota2)/2;       return(x);
 }
```

Programul ex_8_4 constituie o variantă îmbunătățită a programului ex_8_2, în sensul că informațiile referitoare la candidați sunt memorate într-un tablou de structuri. El permite citirea şi memorarea datelor referitoare la un număr de maximum 200 candidați. Capacitatea de memorare poate fi modificată prin simpla înlocuire a numărului 200 din directiva de definire a constantei *MAX_CAND*. Prin includerea fişierului antet "candidat.h", variabilele de tipul structură *candidat* sunt declarate folosind identificatorul *candidat*.

După cum se poate constata, declararea tablourilor de structuri se realizează la fel ca şi declararea tablourilor de variabile simple, adică prin instrucțiuni de forma: *struct nume_structura nume_tablou[n];* în cazul tablourilor unidimensionale, respectiv de forma: *struct nume_structură nume_tablou[n][m];* în cazul tablourilor bidimensionale (matrice). În cazul în care structura a fost redenumită în locul perechii *struct nume_structură* se va utiliza noul cuvînt atribuit ca identificator al structurii.

Pentru citirea datelor, programul ex_8_4 foloseşte funcția **citire**. Aceasta are ca parametru un întreg reprezentînd numărul de ordine al candidatului în lista candidaților şi returnează variabila *x* de tipul structură *candidat*. Acesta este motivul pentru care funcția a fost declarată de tipul *candidat*. În general, tipurile de date construite cu ajutorul structurilor pot fi utilizate atît pentru definirea tipului de funcții, cît şi pentru declararea parametrilor acestora.

0 altă facilitate oferită de limbajul C o constituie atribuirea unei variabile de tipul structură altei variabile de acelaşi tip. În urma unei operații de atribuire cu structuri, valorile membrilor variabilei structură aflate în partea dreaptă a semnului egal sunt atribuite membrilor variabilei structură aflate în stînga acestuia. Astfel, instrucțiunea: *cand_x=citire(i);* atribuie valorile variabilei structură candidat *x*, returnată de funcția de citire, variabilei *cand_x* de acelaşi tip, iar instrucțiunea: *tab[i]=cand_x;* atribuie valorile variabilei *cand_x* elementului *i* din tabloul de structuri *tab*. Desigur, cele 2 instrucțiuni pot fi înlocuite printr-o singură instrucțiune de atribuire de forma: *tab[i]=citire(i);*. Această variantă este mult mai avantajoasă deoarece elimină variabila auxiliara *cand_x* şi o operație de atribuire, contribuind la reducerea memoriei necesare programului şi la creşterea vitezei de execuție. Aceste aspecte privind tehnica de programare sunt extrem de utile în cadrul programelor de dimensiuni mari.

În exemplul prezentat, pentru a decide dacă un candidat a obținut media minimă de promovare se utilizează expresia *tab[i].media* în cadrul instrucțiunii de decizie. Aceasta accesează membrul *media* al elementului *i* din tabloul de structuri *tab*.

În general, pentru accesarea unui membru al unui element dintr-un tablou de structuri se utilizează o expresie de forma: *nume_tablou[i].nume_membru* în cazul tablourilor unidimensionale, respectiv de forma: *nume_tablou[i][j].nume_membru* în cazul tablourilor bidimensionale. În aceste expresii, *i* şi *j* reprezintă indici la elementele tabloului de structuri.

8.1.7. Pointeri la structuri. Liste înlănțuite

Aşa cum un pointer este utilizat pentru a memora adresa unei variabile simple, el poate fi utilizat pentru memorarea adresei unei variabile de tipul structură. Pentru exemplificare, rescriem programul ex_8_2, utilizînd mecanismuf de alocare a memoriei, pentru crearea unei variabile dinamice de tipul structură *candidat*, şi un pointer, pentru accesarea elementelor acesteia.

Exemplul 8.5.

```
/* Programul ex_8_5 +/
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include "candidat.h"
void main(void)
 {
candidat *px;
float notal, nota2;
px=(candidat *)malloc(sizeof(candidat));
while (1)
 {  clrscr( );
    printf ("Numele candidatului: ");    fflush(stdin);  gets(px->nume);
    if(strlen(px->nume)==0)exit( );
    printf ("Numarul legitimatiei de concurs: ");   scanf ("%d",&px->nr_leg);
    printf ("Nota la proba 1: "); scanf ("%f",&notal); px->notal = nota1;
    printf("Nota la proba 2: ");  scanf("%f",&nota2); px->nota2= nota2;
    px->media = (px->nota1+px->nota2)/2;
    if(px->media >= 5)   printf("%s admis",px->nume);
    else                 printf("%s respins",px->nume);getch( );
 }
```

Deosebirea esențiala față de programul ex_8_2 constă în folosirea variabilei *px* ca pointer la o structură de tipul candidat, declarată cu instrucțiunea: *candidat *px;*. Declararea unui pointer la o structură este similară cu declararea unui pointer la o variabilă

simplă, adică se realizează prin plasarea în fața numelui variabilei a caracterului "*". Forma generală a unei instrucțiuni de declarare a unui pointer la o structură este:

struct nume_structură *nume_pointer;

În cazul în care structura a fost redenumită, în locul perechii *struct nume_structură* se va utiliza cuvîntul atribuit ca identificator al structurii prin declarația **typedef**. Deoarece **candidat** a devenit un cuvînt cheie, semnificînd un tip de date, el este utilizat ca parametru în apelul funcției **sizeof** pentru a stabili numărul de octeți necesari unei variabile de tipul structură candidat. Alocarea efectivă a memoriei este realizată cu instrucțiunea: *px=(candidat *)malloc(sizeof(candidat))*, în care funcția **malloc** primește ca parametru valoarea returnată de funcția **sizeof**. Pentru a specifica faptul că adresa returnată de funcția **malloc** este o adresă la o structură de tipul candidat, în fața acesteia s-a folosit declaratia *candidat** (operatorul cast). În cazul în care cererea de alocare a memoriei este satisfăcută, pointerul *px* va conține adresa de memorie rezervată. S-a creat astfel o variabilă dinamică de tipul structură candidat.

Din cele prezentate se constată ca mecanismul de alocare dinamică pentru variabilele de tipul structură este identic cu cel utilizat pentru variabilele simple.

Pentru o variabilă dinamică de tipul structură se pune problema accesării membrilor acesteia. Întrucît pointerul asociat conține adresa variabilei, o modalitate de accesare a membrilor o constituie utilizarea operatorului punct şi a mecanismului de indirectare. Astfel, pentru a accesa elementul *nume* al structurii candidat a cărei adresă este memorată în *px*, se poate utiliza expresia *(*px).nume*. Utilizarea parantezelor este necesară deoarece operatorul punct are o precedență mai mare decît operatorul de indirectare. Deşi sunt corecte, astfel de expresii nu sunt utilizate deoarece limbajul C oferă **operatorul săgeata** (->), format din semnul minus urmat de semnul mai mare strict, pentru a accesa membrii unei structuri prin intermediul unui pointer. Astfel, expresia *px->nume* echivalentă cu expresia *(*px).nume* accesează tabloul de caractere destinat memorării numelui candidatului. Prin utilizarea operatorului -> împreună cu un pointer la o structură, se accesează conținutul unei variabile membru al structurii. Pentru a obține adresa acestuia, se utilizează operatorul de adresă **&**. Acesta este motivul pentru care, în lista variabilelor funcției **scanf**, s-au folosit expresiile *&px->nr_leg, &px->nota1* şi *&px->nota2*.

În concluzie, o expresie de forma: *nume_pointer->nume_membru* furnizează conținutul unei variabile membru al unei structuri, iar o expresie de forma: *&nume_pointer->nume_membru* furnizează adresa acestuia.

Folosind operatorul ->, în cadrul buclei while din programul ex_8_5 sunt citite informațiile referitoare la un candidat, se calculează media şi se afişează rezultatul folosind aceleaşi instrucțiuni ca şi în programul ex_8_2. La terminarea programului, zona de memorie alocată este eliberată prin apelul funcției **free**.

0 importantă aplicație a pointerilor la structuri o constituie crearea listelor înlănțuite. **O listă simplu înlănțuită** constă din structuri conectate între ele prin intermediul pointerilor. Fiecare structură conține printre elementele sale un pointer care accesează structura următoare din listă. Adresa primei structuri din listă este memorată separat într-un pointer, iar pointerul din ultima structură are valoarea 0 (NULL). Acest mecanism oferă o mare flexibilitate, în sensul că se pot insera elemente noi în listă sau pot fi eliminate elemente din listă. Pentru exemplificare considerăm programul din exemplul 8.6.

Exemplul 8.6.

```
/* Programul ex_8_6 */
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <string.h>
struct candidat
 {
char nume[80];    int nr_leg;
float notal, nota2, media;
struct candidat *p_urm;       // pointer de inlantuire
 };
typedef struct candidat candidat;
void main (void)
 {
candidat *p_start,*pa,*pc,*px;
int ind,contor;
float notal, nota2;
p_start = NULL;
do {
    clrscr( );   px= (candidat *) malloc (sizeof(candidat));
    if (px==NULL)
     {
        printf ("EROARE! Memorie insuficienta");    getch( );   exit( );
     }
    printf (" DATELE CANDIDATULUI");   fflush(stdin);
    printf("\n Numele: ");                  gets(px->nume);
    printf(" Numarul legitimatiei: ");    scanf ("%d",&px->nr_leg);
    ptintf (" Notele obtinute: ");      scanf("%f %f",&nota1,&nota2);
    px->notal = nota1;    px->nota2 = nota2;   px->media = (px->nota1+px->nota2)/2;    px->p_urm = NULL;
    ind = 1;    /* inserarea noii structuri in lista */
    if (p_start==NULL)  p_start = px;
    else {
        pa = NULL;   pc = p_start;
        while(pc != NULL)
```

```
            {
               if(strcmp(px->nume,pc->nume)<0)
                {
                   ind = 0;    px->p_urm = pc;
                   if (pa==NULL)
                    {
                       p_start = px;  break;
                    }
                   else
                    {
                       pa->p_urm = px; break;
                    }
                }
               else
                {
                   pa = pc;    pc = pc->p_urm;
                }
            }
           if (ind)pa->p_urm = px;
        }
       printf("\n Continuati rularea ? (d/n) ");
    }
    while (getch( ) != 'n');
    clrscr( );   pc = p_start;        /*  afişarea listei */
    printf(" Lista candidatilor in ordine alfabetica ");
    if (pc==NULL)
    {
       printf(" Lista este goala ");  getch( );   exit(0);
    }
    else
    {
       while(pc != NULL)
        {
           ++contor;
           gotoxy(10,contor+2);     printf("%2d %s",contor,pc_nume);
           gotoxy(30,contor+2);     printf("%4.2f %4.2f %4.2f",pc_nota1,pc_nota2,pc_media);
           pc = pc->p_urm;
        }
    }  getch( );
    }
```

Acest program constituie o variantă îmbunătățită a programului ex_8_4, în sensul că, pentru memorarea informațiilor referitoare la candidații la concursul de admitere, în locul tabloului de structuri *tab* este utilizată o listă de structuri simplu înlănțuite. Se elimină astfel inconvenientul determinat de necesitatea cunoaşterii anticipate a numărului de candidați, necesar atît pentru dimensionarea tabloului *tab* cît şi pentru execuția buclei **for**. În vederea realizării listei înlănțuite,este utilizat pointerul *p_start* pentru memorarea adresei de început a acesteia, iar structura candidat a fost modificată prin adăugarea la elementele sale a pointerului *p_urm*, de tipul structură candidat, necesar înlănțuirii.

Programul permite introducerea într-o ordine aleatoare a informațiilor referitoare la candidați şi realizează lista acestora în ordine alfabetică. În acest sens, este utilizată o buclă **do while**. Mai întîi, prin intermediul unui set de instrucțiuni, similar celui din programul ex_8_5, sunt preluate de la tastatură informațiile referitoare la un candidat oarecare şi memorate în variabila *px* creată dinamic. Apoi, printr-un proces de căutare se determină poziția din listă unde trebuie inserată noua structură. Se disting următoarele 2 cazuri:

- lista este goală, fapt indicat de valoarea NULL a pointerului *p_start*. În acest caz se atribuie lui *p_start* valoarea *px*.
- lista conține un număr oarecare de structuri care sunt înlănțuite în ordinea alfabetică a numelor candidaților. În acest caz, pentru a determina poziția de inserare a noii structuri, se parcurge lista începînd cu primul element. Pentru aceasta, se utilizează pointerii de tipul structură candidat *pa* şi *pc*. Pointerul *pc* constituie poziția curentă în procesul de căutare,iar *pa* poziția anterioară poziției curente. Inițial, lui *pc* i se atribuie valoarea *p_start*, iar lui *pa* valoarea NULL. Determinarea poziției de inserare se realizează în cadrul buclei **while** comparînd numele candidatului din structura *px* cu cel al candidatului din structura *pc*. În acest sens, se utilizează instrucțiunea de decizie **if else** avînd ca expresie logică apelul funcției **strcmp**. Dacă valoarea returnată de **strcmp** este negativă, atunci structura *px* trebuie inserată între structurile *pa* şi *pc*. În caz contrar, se avansează în listă atribuind variabilei *pa* valoarea *pc,* iar variabilei *pc* valoarea adresei următoarei structuri din listă, furnizată de expresia: *pc-> p_urm;*.

Referitor la poziția de inserare a lui *px* în listă, se disting următoarele trei cazuri:

- poziția de inserare este chiar la începutul listei.Acest caz corespunde situației în care *pa=NULL*,iar *pc=p_start*. Se realizează înlănțuirea lui *px* cu *pc* şi se modifică valoarea lui *p_start* care devine *px*.
- poziția de inserare este în interiorul listei, situație în care atît *pa*, cît şi *pc* sunt diferite de NULL. În acest caz se înlănțuie *px* cu *pc* prin intermediul instrucțiunii *px->p_urm=pc*, iar *pa* cu *px*, prin instrucțiunea: *pa->p_urm=px*.
- poziția de inserare este la sfîrşitul listei, situație în care *pc* are valoarea NULL. În acest caz se realizează înlănțuirea lui *pa* cu *px*. În cadrul programului, pentru a detecta acest caz, se utilizează variabila întreaga *ind*, care la începutul procesului de căutare este inițializată cu 1. Dacă structura *px* a fost inserată înainte ca variabila *pc* să capete valoarea NULL, atunci se modifică valoarea lui *ind*

care devine 0. La ieşirea din bucla **while**, prin testarea lui *ind* în cadrul instrucțiunii *if(ind)* se decide dacă *px* va fi inserat sau nu la sfîrşitul listei.

Pot fi utilizate şi alte criterii de găsire a poziției în listă. Astfel, dacă în programul anterior în locul numelor cand-lor se utilizează media obținută de aceştia ca test de depistare a poziției de inserare,se va obține lista în ordinea descrescătoare a mediilor.

0 altă aplicație importantă a listelor simplu înlănțuite o constituie utilizarea acestora la memorarea matricelor de mari dimensiuni în care ponderea elementelor nenule este scăzută, numite **matrice rare** sau **sparte**.În astfel de situații,din considerente de economie de memorie şi creştere a vitezei de execuție a programelor, este avantajoasă numai memorarea elementelor nenule. În acest sens, pentru fiecare linie a matricei se construieşte cîte o listă înlănțuită formată din structuri ce conțin un întreg în care se va memora indicele coloanei elementului nenul, un întreg sau un real în care se va memora valoarea acestuia şi un pointer de înlănțuire. Pointerii care marchează începutul fiecărei linii vor fi memorați într-un tablou de pointeri la structuri. Programul de calcul este prezentat în exemplul 8.7.

Exemplul 8.7.

```
/* Prograeul ex_8_7 */
#include <stdio.h>
#include <alloc.h>
#define    L_MAX   100
void main(void)
 {
struct el_aij
 {
    int col;float val;
    struct el_aij *p_urm;
 }
typedef struct el_aij element;
element *p_start[L_MAX],*pc,*pa,*px;
int n_lin,i,i_col,ind;
float f_val;    clrscr( );
printf (“Introduceti numarul de linii (maximum %d) ”, L_MAX);    scanf (“%d”,&n_1in);
for(i=0; i<n_lin; i++)
 {
    p_start = NULL;  clrscr( );
    printf (“Introduceti elementele de pe linia %d\n”,i);
    while(1)
     {
        px = (element *)malloc(sizeof(element));
        if (px==NULL)
         {
            printf (“EROARE! Memorie insuficienta”);    getch( );    exit( );
         }
        printf (“ Introduceti indicele coloanei si valoarea > ”);    scanf (“%d %f”,&i_col,&f_val);
        if (i_col < 0)  break;
        px->col = i_col;  px->val = f_val;  px->p_urm = NULL;
        ind = 1;    /* inserarea in lista */
        if (p_start[i]==NULL)    p_start[i] = px;
        else {
            pa = NULL;   pc = p_start[i];
            while(pc != NULL)
             {
                if(px->col < pc->col)
                 {
                    ind = 0;    px->p_urm = pc;
                    if (pa==NULL)
                     {
                        p_start[i] = px;    break;
                     }
                    else
                     {
                        pa->p_urm = px; break;
                     }
                 }
                else
                 {
                    pa = pc;    pc = pc->p_urm;
                 }
             }
            if (ind)pa->p_urm = px;
        }
    }
```

```
}       /*  afişarea listei */
for (i=0; i<n_lin; i++)
{  clrscr( );
   printf(" Elementele nenule de pe linia %d\n col val",i);
   pc = pc->p_start[i];
   while (pc != NULL)
    {
   printf ("\n%6d %10.2f",pc->col,pc->val);
   pc=pc->p urm;
    }  getch( );
}
}
```

Mai întîi, programul solicită utilizatorului numărul de linii ale matricei ce urmează a fi memorată. Numărul maxim de linii este specificat în cadrul directivei **#define**, utilizînd constanta L_MAX. După definirea structurii, necesară memorării, şi redenumirea acesteia, se declară tabloul de pointeri *p_start*, avînd dimensiunea L_MAX, şi pointerii *pa, pc* şi respectiv *px*, necesari memorării datelor introduse şi inserării în listele înlănţuite.

Algoritmul de inserare este identic cu cel descris anterior, cu menţiunea că pentru a detecta poziţia în listă a unui element a_{ij} se folosesc indicii de coloană.

Prin parcurgerea buclei **for** se construiesc în mod secvenţial listele aferente liniilor matricei. Elementele nenule ale unei linii pot fi introduse într-o ordine aleatoare, iar pentru a marca faptul că nu mai sunt de introdus elemente, se tastează o valoare negativă la solicitarea indicelui de coloana.

Programul poate fi modificat astfel încît să permită introducerea într-o ordine aleatoare a tuturor elementelor nenule ale matricei. În afara listelor simplu înlănţuite se pot crea şi alte tipuri de liste înlănţuite, cum ar fi listele circulare, listele dublu înlănţuite ş.a.

8.2. Uniuni

Uniunile constituie o modalitate prin care un număr oarecare de variabile de acelaşi tip sau de tipuri diferite pot accesa la momente diferite de timp o aceeaşi zonă de memorie.

8.2.1. Uniuni simple

Pentru a prezenta modul de definire şi utilizare a uniunilor, considerăm următorul program de calcul:

Exemplul 8.8.

```
/* Programul ex_8_8 */
#include <stdio.h>
void main (void)
 {
union numar
 {
    char ch;    int i_val;   float f_val;
 };
union numar x;    clrscr( );
printf ("Dimensiunea in octeti a variabilei x este %1d",sizeof(x));
x.ch='A';           printf("\nContinutul variabilei x este caracterul %c",x.ch);
x.i_va1=725; printf("\nContinutul variabilei x este intregul %d",x.i_val);
x.f va1=-5.25;     printff("\nContinutul variahilei x este realul %f",x.f_val);
getch( );
 }
```

După cum se poate constata, definirea unei uniuni, declararea variabilelor de tipul uniune şi accesarea valorilor acestora se realizează într-o manieră similară celei folosite în cazul structurilor. Deoarece conceptual sunt total diferite, asemănarea uni-unilor cu structurile se rezumă doar la cele menţionate mai sus. Acest fapt este ilustrat şi de rezultatele obţinute prin rularea prog-ramului ex_8_8. Astfel,deşi uniunea are ca elemente un caracter, un întreg şi un real, dimensiunea ei este de doar 4 O. Acest spaţiu de memorie este suficient pentru a memora fiecare element individual, dar nu este suficient pentru a le memora pe toate simultan.

În cadrul programului accesăm mai întîi elementul *ch* al variabilei *x* de tipul uniune *numar*, căruia îi atribuim valoarea A. Apoi este accesat elementul *i_val* căruia îi atribuim valoarea 725, iar în final elementul f_val căruia ii atribuim valoarea -5.25. Rezultatele afişate prin apelul succesiv al funcţiei **printf** demonstrează faptul că, la un moment dat, cei 4 O de memorie rezervaţi variabilei *x* pot fi ocupaţi doar de unul din elementele uniunii. Spunem că elementele uniunii partajează acelaşi spaţiu de memorie.

Conţinutul zonei de memorie rezervate uniunii poate fi modificat printr-o operaţie de atribuire, folosind oricare dintre membri, dar odată atribuirea făcută, preluarea valorii nu este posibilă decît prin intermediul membrilor de acelaşi tip cu cel utilizat în operatia de atribuire. Astfel, dacă în programul anterior, după ce am atribuit lui *x.i_val* valoarea 725, vom încerca să accesam zona de memorie prin intermediul lui *x.f_val*, apare un nonsens deoarece programul va încerca să interpreteze o valoare întreagă ca un număr reprezentat în virgulă flotantă (mantisă si exponent).

8.2.2. Uniuni de structuri. Accesul funcţiilor ROM BIOS

Aşa cum o structură poate fi un element membru al unei alte structuri, ea poate fi şi un element membru al unei uniuni.

Pentru exemplificare, considerăm următorul program în cadrul căruia, prin utilizarea unei structuri şi a unei uniuni, este creat un mecanism de accesare individuală a celor 2 O aferenţi unei variabile de tipul întreg.

Exemplul 8.9.

```
/* Programul ex_8_9 */
```

```
void main(void)
 {
struct octet
 {
    char octetl;    char octet2;
 }
union intreg
 {
    struct octet val;   int i_val;
 } x;
clrscr( );   printf (“Dimensiunea in octeti a variabilei x este %1d”,sizeof(x));
x.val.octet1=5;    x.val.octet2=1;
printf(“\nValoarea continuta in primul octet: %d”,x.val.octetl);
printf(“\nValoarea continuta in al doilea octet: %d”,x.val.octet2);
printf("\nValoarea accesata de i_val %d”,x.i_val);     getch( );
 }
```

Mai întîi, este definită structura *octet*, care are ca elemente variabilele de tipul caracter *octet1* şi *octet2*. Apoi, această structură este utilizată pentru a declara variabila *val* ca membru al uniunii *intreg* alături de variabila de tipul întreg *i_val*.Deoarece fiecare membru al uniunii ocupă cîte 2 O, dimensiunea zonei de memorie alocată variabilei *x* de tipul **union intreg** va fi de 2 O. Acest fapt este confirmat de rezultatul afişat de primul apel al funcţiei **printf**.

Prin operatiile de atribuire: *x.val.octet1=5; x.val.octet2=1;* în primul octet se va depune valoarea 1, iar în al doilea octet valoarea 5. Deoarece prin intermediul elementelor unei structuri sunt accesate zone de memorie diferite, valorile 1 şi 5 se găsesc simultan în memorie şi pot fi interpretate ca elemente componente ale inei valori întregi care, după cum ştim, ocupa 2 O. Primul dintre aceştia se numeşte **octetul mai puţin semnificativ** (**low**), iar al doilea se numeşte **octetul cel mai semnificativ** (**high**). Avînd în vedere reprezentarea binară a valorilor 1 şi 5 avem: 0000000100000101, concluzionăm că valoarea accesata de x.i_val va fi $1 \cdot 2^8 + 1 \cdot 2^2 + 1 \cdot 2^0 = 261$, fapt confirmat de ultimul apel al funcţiei **printf**, care are în lista de variabile pe x.i_val.

Din exemplul prezentat constatăm că, pentru a accesa elementele membre ale unei structuri care este la rîndul ei un element membru al unei uniuni, se foloseşte de 2 ori operatorul punct. Procesul de înlănţuire este general, în sensul că aşa cum o structură poate fi element membru al altei structuri sau al unei uniuni, o uniune poate fi la rîndul ei element membru al unei structuri sau al altei uniuni.

Un exemplu de uniune de structuri îl constituie uniunea **REGS**, definită în fişierul antet <**dos.h**>, utilizată pentru transmiterea parametrilor funcţiei de bibliotecă **int86**, prin intermediul careia putem accesa funcţiile **ROM-BIOS** ale sisternului de calcul. Rutinele ROM-BIOS sunt scrise în limbajul de asamblare şi sunt realizate pentru a fi apelate de către programe scrise în acest limbaj. Compilatoarele C oferă posibilitatea accesării rutinelor ROM BIOS folosind sistemul de întreruperi prin intermediul funcfiei int86. Forma de apel a acestei funcţii este:

int86 (nr_intrerupere,*in_regs,&out_regs)

în care: *nr_intrerupere* este o variabilă de tipul întreg în care se memorează numărul întreruperii; *in_regs* şi *out_regs* sunt 2 variabile de tipul uniune **REGS**, prin intermediul cărora se transmit informaţiile şi se recepţionează răspunsuri.

Pentru a înţelege modul de utilizare al funcţiei **int86** sunt necesare cîteva precizări privind transmiterea parametrilor şi apelul funcţiilor ROM BIOS prin intermediul uniunilor de tipul **REGS** şi al numărului întreruperii.

Atunci cînd apelăm o funcţie în C, îi putem transmite acesteia valori prin intermediul argumentelor plasate în interiorul perechii de paranteze care urmează după numele funcţiei. Aceste valori sunt plasate într-o zona de memorie numită **stivă** (**stack**) de unde funcţia le poate prelua şi efectua operaţii cu ele.

În cazul apelului unei rutine ROM BIOS, procesul de transmitere a valorilor este complet diferit. În loc ca valorile transmi-se să fie plasate în stivă, ele sunt plasate în registrele de lucru ale microprocesorului. Acestea sunt dispozitive hard asemănătoare locaţiilor de memorie, dar care au mai multe posibilităţi, fiind utilizate de microprocesor pentru efectuarea de operaţii aritmetico-logice şi a multor altor operaţii. Microprocesoarele din familia 80x86 (8086,80826,80836 şi 80486) dispun, pe lîngă alte registre, de patru registre de lucru notate *ax, bx, cx* şi *dx*. Fiecare registru constă din 2 O al căror conţinut poate fi accesat fie simultan, asemanător unei variabile de tipul int, fie individual.

În cazul accesării individuale, pentru octeţii cei mai semnificativi se folosesc notaţiile *ah, bh, ch* şi *dh*, iar pentru octeţii mai puţin semnificativi notaţiile *al, bl, cl* şi *dl*. Mecanismul de accesare dualã a registrelor se realizează, la fel ca în exemplul 8.9, prin intermediul unei variabile de tipul uniune REGS. Aceasta are ca elemente 2 structuri. Prima structură *x* conţine elementele de tipul întreg *ax, bx, cx* şi *dx*, iar a doua *h* conţine elementele de tipul caracter *al, bl, cl, dl, ah, bh, ch* şi *dh*.

Deci, dacă definim variabila *regl* de tipul uniune REGS prin instrucţiunea: *REGS regl;*, pentru accesarea registrelor *ax, bx, cx, dx* se folosesc expresiile: *regl.x.ax, regl.x.bx, regl.x.cx* şi *regl.x.dx*, iar pentru accesarea registrelor *ah,bh,ch,dh, al*,bl,*cl* şi *dl* expresii de forma: *regl.h.ah ... regl.h.dl*.

Revenind la apelul funcţiei **int86** se precizează faptul că numărul întreruperii este utilizat pentru a accesa un grup de funcţii ROM-BIOS. Selectarea unei anumite rutine din grupul de funcţii definit de întrerupere se realizează prin plasarea în registrele de lucru a unor valori specifice. Datele specifice fiecărei funcţii ROM-BIOS sunt prezentate în documentaţia tehnică a sistemului de calcul sub forma:

Scopul: Numărul întreruperii în hexazecimal:

Conţinutul registrelor la apelul funcţiei:

Conţinutul registrelor după revenirea din funcţie:

Pentru exemplificare, prezentăm în continuare trei aplicaţii care utilizeazu serviciile ROM-BIOS.

Prima aplicaţie o constituie apelul funcţiei ROM-BIOS pentru detectarea dimensiunii spaţiului de memorie al unui calculator. Datele specifice funcţiei sunt:

Scopul: Determinarea dimensiunii memoriei de bază Numărul întreruperii: 0x12

Conţinutul registrelor la apel: -

Continutul registrelor după apel: ax conține dimensiunea memoriei în kO.

Programul de calcul este următorul:

Exemplul 8.10.

```
/* Programul ex_8_10 */
#include <dos.h>
#include <stdio.h>
void main(void)
 {
union REGS regl;
unsigned int dim_mem;
int86(0x12,&regl,&regl);     dim_mem = regl.x.ax;
clrscr( );   printf("Calculatorul are %d kOi memorie de baza, (fara extensie)",dim_mem);     getch( );
 }
```

A doua aplicație o constituie apelul funcției ROM-BIOS în vederea modificării dimensiunii cursorului de pe ecran. Înălțimea maximă a cursorului este înălțimea unui caracter şi se obține prin suprapunerea unui număr de linii orizontale, având fiecare lungimea egaiă cu lățimea unui caracter. Aceste dimensiuni depind de tipul adaptorului video cu care este echipat calcula-torul. În cazul adaptorului video EGA, cele 14 linii ce definesc inălțimea caracterului sunt numerotate de sus în jos, de la 0 la 13.

Dimensiunea cursorului poate fi modificată afişînd pe ecran numai o parte din cele 14 linii. Pentru aceasta se utilizează funcția ROM-BIOS cu următoarele date specifice:

Scopul: Setarea dimensiunii cursorului Numărul întreruperii: 0x10

Conținutul registrelor la apel: ah = 01 ch = linia de start (0,13) cl = linia terminală (0,13)

Conținutul registrelor după apel: -

Se menționează faptul că intreruperea 0x10 grupează setul de funcții ROM-BIOS care sunt destinate serviciilor video. Pentru a selecta funcția de setare a dimensiunii cursorului, în registrul *ah* se depune valoarea 1. De asemenea, este necesar ca indicele liniei de start conținut în registrul *ch* să fie mai mic sau cel mult egal cu indicele ultimei linii afişate conținut în registrul *cl*. Programul de calcul este următorul:

Exemplul 8.11.

```
/* Programul ex_8_11 */
#include <dos.h>
#include <stdio.h>
void main(void)
 {
union REGS regl;
int start, stop;
while(i)
 {
    clrscr( );   printf ("Introduceti numarul liniei de start");   scanf("%d",&start);
    if (start<0 )    exit( );
    printf ("Introduceti numarul liniei de stop");   scanf("%d",&stop);
    regl.h.ah = 0x01; regl.h.cl = (char)stop;     regl.h.ch = (char)start;        int86(0x10,&regl,&regl);
 }
 }
```

A treia aplicație o constituie apelul funcției ROM-BIOS prin care cursorul este făcut invizibil (cursor off). Datele specifice acestei funcții sunt:

Scop: cursor off Numărul întrerupere: 0x10

Conținutul registrelor la apel: ah = 01 ch = 0x20

Conținutul registrelor duph apel: -

iar programul de calcul este următorul:

Exemplul 8.12.

```
/* Programul ex_8_12 */
#include <dos.h>
void main(void)
 {
union REGS regl;     clrscr( );
regl.h.ah = l;  regl.h.ch = 0x20;      int86(0xl0,&regl,&regl);
printf("Cursorul este invizibil. Apasati o tasta");  getch( );
regl.h.ah = l;  regl.h.ch = 12;     regl.h.c1= 13;int86(0x10,&regl,&regl);
printf ("\nCursorul este din nou vizibil. Apasati o tasta");     getch( );
 }
```

Pentru reafişarea cursorului pe ecran (cursor on) se foloseşte apelul funcției ROM-BIOS descris în exemplul 8.11. Această tehnică de trecere a cursorului din vizibil în invizibil şi invers este frecvent utilizată în cadrul programelor de calcul care lucrează cu meniuri.

8.3. Cîmpuri pe biți

Limbajul C permite definirea şi prelucrarea datelor pe biți. Utilizarea lor poate conduce la economisirea de memorie. Într-adevar, adesea avem nevoie de date care au numai 2 valori, zero sau unu. O astfel de dată poate fi păstrată pe un singur bit. De aceea, pentru astfel de date, nu se justifică să alocăm un octet sau chiar doi. În general, nu este util ca date de valori mici să fie păstrate pe

octeți sau pe 16 biți, mai ales atunci cînd aceste date sînt în număr mare. În acest scop, limbajul C oferă posibilitatea de a declara date care să se aloce pe biți.

Un şir de biți adiacenți formează un **cîmp**. Un cîmp trebuie să se poată păstra într-un cuvînt calculator. Mai multe cîmpuri pot fi păstrate într-un acelaşi cuvînt calculator. Cîmpurile se grupează formînd o structură. O astfel de structură se declară ca o structură obişnuită care are ca componente cîmpuri:

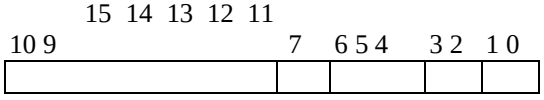
struct nume {
cimp_1; cimp 2; . . . cimp n;
} numel,nume2,...,numem;
Un cîmp se declară astfel:
tip nume_cimp: lungime_in_biți
sau tip: lungime_in_biți

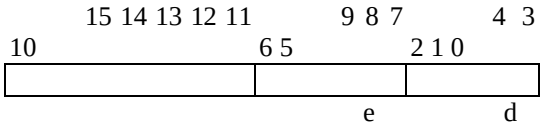
De obicei, *tip* este cuvîntul cheie *unsigned*, ceea ce înseamnă ca şirul de biți din cîmpul respectiv se interpretează ca fiind un întreg fără semn. Alte posibilități pentru *tip* sînt: *int*; *unsigned char*; *char*. Cîmpurile se alocă de la biții de ordin inferior ai cuvîntului spre cei de ordin superior.

Cîmpurile cu semn se utilizează pentru a păstra întregi de valori mici prin complement față de doi. De aceea, în acest caz, bitul cel mai semnificativ al cîmpului este bit semn. Dacă un cîmp nu se poate aloca în cuvîntul curent, el se va aloca în cuvîntul următor. Un cîmp fără nume nu se poate referi. El defineşte o zonă neutilizată dintr-un cuvînt. Lungimea în biți poate fi egală cu zero. În acest caz, data următoare se alocă în cuvîntul următor.

Cîmpurile se pot referi folosind aceleaşi convenții ca şi în cazul structurilor obişnuite. Exemplu:

struct {
unsigned a:2; int b:2; unsigned :3; unsigned c:2; unsigned :0; int d:5; unsigned e:5;
} x,y;
Pentru x se alocă 2 cuvinte, astfel:





x.a=1 - atribuie cîmpului *a* al datei *x* valoarea 1, deci bitul 0 devine 1, iar bitul 1 ia valoarea 0;

x.b=-1 - atribuie cîmpului *b*, al datei *x*, valoarea -1. Aceasta înseamnă că ambii biți ai lui *b* se fac egali cu 1 (11 este reprezentarea lui -1 pe 2 biți prin complement față de 2).

Nu se pot defini tablouri de cimpuri. De asemenea, operatorul adresă (& unar) nu se poate aplica la un cîmp.

Datele pe biți conduc la programe care, de obicei, nu sînt portabile sau au o portabilitate redusă. De aceea, se recomandă utilizarea lor cu precauție. De asemenea, datele pe biți necesită instrucțiuni suplimentare (deplasări, setări şi/sau mascări de biți etc.) față de cazul cînd sînt păstrate în mod obişnuit (ca date de tip *int* sau *char*). De aceea, utilizarea lor se justifică numai atunci cind alocarea pe biți conduce la o economie substanțială de memorie față de alocarea pe octeți sau pe cuvinte de 16 biți.

Observație: Prelucrarea datelor pe biți se poate realiza şi fără a defini cîmpuri de biți, utilizînd operatorii logici pe biți. Utilizarea lor poate conduce însă la un efort de programare suplimentar care poate fi destul de mare. De asemenea, utilizarea operatorilor respectivi poate să nu fie făcută optim sau să conducă la folosirea unor expresii eronate. Aceasta nu înseamnă că trebuie să renunțăm la utilizarea operatorilor logici pe biți. Există situații cînd utilizarea lor permite scrierea unor programe mai performante decît dacă se utilizează, în aceleaşi scopuri, cîmpuri de biți.

8.4. Tipul enumerare

Tipul enumerare permite programatorului să folosească nume sugestive pentru valori numerice. De exemplu, în locul numărului unei luni calendaristice este mai sugestiv să folosim denumirea lunii respective sau eventual o prescurtare: *ian* - pentru ianuarie în locul cifrei 1; *feb* - pentru februarie în locul cifrei 2; ş.a.m.d. Un alt exemplu se referă la posibilitatea de a utiliza cuvintele FALS şi ADEVARAT pentru valorile 0 respectiv 1. În felul acesta se obține o mai mare claritate în programele sursă, deoarece valorile numerice sînt înlocuite prin sensurile atribuite lor într-un anumit context.

În acest scop se utilizează **tipul enumerare**. Un astfel de tip se declară printr-un format asemănător cu cel utilizat în cadrul structurilor. Un prim format general este:

enum nume { nume0,nurnel,nume2,...,numek } dl,d2,...,dn;

unde: *nume* - este numele tipului de enumerare introdus prin această declarație; *nume0,numel,...,numek* - sînt nume care se vor utiliza în continuare în locul valorilor numerice şi anume *numei* are valoarea *i*; *d1,d2,...,dn* - sînt date care se declară de tipul *nume*. Aceste date sînt similare cu datele de tip *int*.

Ca şi în cazul structurilor, în declarația de mai sus nu sînt obligatorii toate elementele. Astfel, poate lipsi *nume*, dar atunci va fi prezent cel puțin *dl*. De asemenea, poate lipsi în totalitate lista *dl,d2,...,dn*, dar atunci va fi prezent *nume*. În acest caz, se vor defini ulterior date de tip *nume* folosind un format de forma: *enum nume dl,d2,...,dn;*

Exemple:

1. *enum {ileg,ian,feb,mar,apr,mai,iun,iul,aug,sep,oct,nov,dec) luna;* Prin această declarație, numărul lunii poate fi înlocuit prin denumirea prescurtata a lunii respective. De exemplu, o atribuire de forma: *luna = 3;* se poate înlocui cu una mai sugestivă: *luna=mar;* deoarece, conform declarației de mai sus, *mar* are valoarea 3. La fel o expresie de forma:*luna==7;* este identică cu expresia: *luna==iul;* Dacă s-ar fi utilizat declarația de tip enumerare: *enum dl(ileg,ian,feb,mar,apr,mai,iun,iul,aug,sep,oct,nov,dec);* atunci putem declara ulterior data *luna* de tip *dl* astfel: *enum dl luna;* Data *luna* declarată în acest fel este o dată identică cu data *luna* declarată la început.

2. Fie tipul enumerare *Boolean* declarat astfel: *enum Boolean(false,true);* Declarăm data *bisect* de tip *Boolean*: *enum Boolean bisect;* Atribuirea: *bisect=an%4==0&&an%100||an%400==0;* atribuie variabilei *bisect* valoarea 1 sau 0, după cum anul definit de variabila *an* este bisect sau nu (se presupune ca anul aparține intervalului [1600,4900]). În continuare se pot folosi expresii de forma: *bisect==false* sau *bisect==true.*

3. Fie f o funcție care returnează numai 2 valori: 0 sau 1. O astfel de funcție se consideră, de obicei, că returnează o valoare booleană. Atunci, antetul ei poate fi definit astfel:

enum Boolean f(...)

În continuare se pot folosi expresii de forma: *f(...)==false* sau *f(...)==true*.

La declararea tipurilor enumerare se poate folosi cuvîntul cheie *typedef*, ca şi în cazul tipurilor utilizator definite prin *struct*.

Tipul enumerare poate fi declarat impunînd valori altele decît cele care rezultă implicit. În acest caz, *numei* se va înlocui cu: *numei=eci* unde: *eci* - este o expresie constantă. Numele *numei* va avea ca valoare, valoarea expresiei *eci*. Dacă numelui următor nu i se atribuie o valoare, atunci acesta va avea ca valoare, valoarea lui *numei* mărit cu 1. Folosind această observație, modificăm tipul *dl* astfel: *typedef enum (ian=1,feb,mar,apr,mai,iun,iul,aug,sep,oct,nov,dec) DL;* Menționăm ca datele de acest tip nu sînt supuse la controale din partea compilatorului C şi de aceea se pot scrie expresii care să nu corespundă scopului pentru care s-au definit datele respective. De exemplu, fie: *DL dl,d2,d3;* Expresiile de mai jos nu sînt interpretate ca eronate de compilator:

d3 = d1+ d2;
d3 = dl *d2;
d3 = d1/d2;
etc.

Aceasta, deoarece datele de tip enumerare sînt tratate ca simple date de tip *int*.