Lucrarea de laborator 0

Pointeri

Programul şi datele sale sunt păstrate în memoria calculatorului (RAM – Random Acces Memory – memorie cu acces aleator). Memoria este împărţită în octeţi (opt biţi). Doi octeţi formează un cuvânt, patru octeţi formează un cuvânt lung, iar 16 octeţi formează un paragraf (pe IBM PC). Fiecare octet are în memorie o adresă unică, octeţi consecutivi având adrese consecutive.

Un pointer este o variabilă care păstrează adresa unei date, în loc de a memora data însăşi.

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

- accesarea directă a diverselor zone de memorie prin modificarea adresei memorate în pointer; în acest fel pot fi create şi exploatate liste înlănţuite.
- crearea de noi variabile în timpul execuţiei programului, în cadrul mecanismului de alocare dinamică a memoriei;
- întoarcerea dintr-o funcţie a mai multor valori către funcţia apelantă; în mod uzual, o funcţia întoarce o singură valoare prin intermediul instrucţiunii return;
- accesarea directă unor elemente dintr-o structură de date cum ar fi şiruri sau tablouri.

1. Declararea şi utilizarea pointerilor

Ca orice tip de variabilă, înainte de a fi utilizată, variabila de tip pointer trebuie declarată. Declararea se face în cadrul unei instrucţiuni având forma generală:

```
Tip *nume_pointer 1, .., *nume_pointer n;
```

unde: *tip* poate fi oricare dintre tipurile fundamentale de date (int, float, double, char),

nume_pointer i un şir de caractere cu condiţia ca primul caracter să fie o literă (aceeaşi condiţie ca la numele de variabile).

Singura deosebire faţă de o instrucţiune de declarare de variabile este prezenţa caracterului *, care semnalează compilatorului că a fost declarată o variabilă pointer şi nu o variabilă obişnuită.

Pentru a explica modul de lucru cu pointeri, considerăm exemplele următoare:

Exemplul 1.

```
main()
{
        int varint, *ptrint;
        ptrint = &varint;
        varint = 200;
        printf("adresa varint: %p\n", &varint);
        printf("valoare varint: %d\n", varint);
        printf("valoare ptrint: %p\n", ptrint);
        printf("valoare adresată de ptrint: %d\n",*ptrint);
```

```
}
```

În exemplul 1. sunt declarate două variabile, *varint* şi *ptrint*. Prima este o variabilă de tip întreg, care păstrează o valoare de tip int. A doua este o variabilă de tip pointer la o variabilă întreagă, ea memorând adresa unei variabile de tip int. **Operatorul *** este numit în limbajul C **operator de indirectare.**

Adresa lui *varint* este atribuită lui *ptrint*, iar valoarea întreagă 200 este atribuită lui varint.

Rezultatul execuției programului este:

```
adresa varint: FFD4
valoare varint: 200
valoare ptrint: FFD4
valoare adresată de ptrint: 200
```

Primele două linii dau adresa şi conținutul lui *varint*. A treia linie reprezintă conținutul lui *ptrint* (adresa lui *varint*), respectiv adresa de memorie în care programul a creat *varint*. Această valoare poate diferi de la execuție la execuție. Ultima linie afişează valoarea memorată la adresa respectivă.

Instrucțiunea

```
varint = 200;
```

este echivalentă cu

```
*ptrint = 200;
```

întrucât *varint* şi **ptrint* se referă la aceeaşi locație de memorie.

Exemplul 2.

```
#include <stdio.h>
void main(void)
{
        float a = 5.2, *ptr_a;
        int *ptr_b, b = 10;
        clrscr();
        printf("\n Valorile inițiale  a=%6.3f  b=%d", a,b);
        ptr_a = &a;
        ptr_b = &b;
        *ptr_a = a + 10;
        *ptr_b = b + 10;
        printf("\n Valorile finale  a=%6.3f  b=%d", *ptr_a, *ptr_b);
        getch();
}
```

Programul din acest exemplu realizează, cu ajutorul pointerilor, operațiile $a = a + 10$ şi respectiv $b = b + 10$. Instrucțiunea `float a = 5.2, *ptr_a;` declară variabila *a* de tip real şi o inițializează cu valoarea 5.2 , iar variabila *ptr_a* de tipul pointer la un real. Instrucțiunea

int *ptr_b, b = 10; declară *ptr_b* de tipul pointer la întreg, iar variabila *b* de tipul întreg, inițializând-o cu valoarea 10.

Din aceste două exemple de instrucțiuni de declarare a variabilelor observăm că putem declara, în cadrul aceleaşi instrucțiuni, atât variabile simple, cât şi variabile pointer, în orice ordine, separate prin virgulă.

După ce, prin apelul funcției printf, sunt afişate valorile inițiale ale variabilelor *a* şi *b*, variabilei *ptr_a* îi este atribuită valoarea adresei variabilei *a,* iar variabilei *ptr_b* îi este atribuită valoarea adresei variabilei *b*, folosind instrucțiunile de atribuire ptr_a = &a; şi ptr_b = &b;

În continuare, programul modifică valorile variabilelor *a* şi *b*, rezultând noile valori 15.2 şi respectiv 20. Pentru aceste operații se utilizează adresele variabilelor *a* şi *b*, memorate în *ptr_a* şi respectiv *ptr_b*, precum şi caracterul de indirectare * în fața numelui variabilelor pointer.

În cadrul instrucțiunii **ptr_a = a + 10;* se accesează zona de memorie a variabilei *a* (prin intermediul adresei conținute în variabila pointer *ptr_a*) în care se depune valoarea *a* + 10. Această instrucțiune este identică, ca efect, cu instrucțiunea *a* = *a* + 10; Din această instrucțiune putem deduce că operatorul de indirectare * este complementar operatorului adresă, adică instrucțiunea *&a = a + 10; este echivalentă cu instrucțiunea a = a + 10;.

Penultima instrucțiune din exemplul 2. foloseşte în cadrul funcției 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 obțin de la adresele memorate de pointerii *ptr_a* şi respectiv *ptr_b*, adică de la adresele variabilelor *a* şi respectiv *b*. Dacă nu era utilizat operatorul *, adică dacă instrucțiunea ar fi fost scrisă sub forma printf("\n Valorile finale a=%6.3f b=%d", ptr_a,ptr_b);, descriptorii de format ar fi fost înlocuiți ca valorile conținute în *ptr_a* şi respectiv *ptr_b*, deci cu valorile adreselor variabilelor *a* şi *b*.

2. Alocarea dinamică a memoriei

Utilizarea pointerilor permite ca în timpul execuției 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 ne vom folosi de exemplele următoare.

Exemplul 3. Se modifică programul din exemplul 1. astfel:

```
#include <alloc.h>
main()
{
        int  *ptrint;
        ptrint = (int*)malloc(sizeof(int));
        *ptrint = 200;
        printf("valoare ptrint: %p\n", ptrint);
        printf("valoare adresată de ptrint: %d\n",*ptrint);
}
```

Rezultatul execuţiei va fi de genul:

valoare ptrint: 053E

valoare adresată de ptrint: 200

Spaţiul de memorie necesar stocării valorii variabilei pointer *ptrint* este alocat prin apelul funcţiei malloc, definită în fişierul alloc.h. Această funcţie, a cărei formă generală de apel este

malloc(expresie de tip întreg);

solicită sistemului de operare alocarea pentru programul care o apelează a unei zone de memorie având 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 de spaţiu de memorie liber, solicitarea nu este satisfăcută, funcţia returnează valoarea NULL.

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 reprezentând lungimea exprimată în octeţi a tipului de date pe care îl primeşte ca parametru. De exemplu, sizeof(int) returnează valoarea 2 (numărul de octeţi necesar pentru a memora o dată de tip întreg), iar sizeof(float) retrunează valoarea 4, din aceleaşi considerente.
- utilizarea expresiei (int*) în faţa numelui funcţiei. Funcţia malloc returnează o adresă pe care o putem atribui unui pointer de orice tip. Expresia (int*) specifică faptul că adresa returnată de malloc va fi atribuită unei variabile de tipul pointer la întreg şi constituie un exemplu de utilizare a operatorului cast.

În exemplul 3. lui *ptrint* îi este atribuită valoarea returnată de funcţia malloc declarată în alloc.h. Se observă că valoarea lui *ptrint* diferă faţă de cea din exemplul 1., dar valoarea adresată este aceeaşi.

Expresia sizeof(int) returnează numărul de octeţi necesari unei variabile de tip întreg, şi anume 2.

Funcţia malloc(valoare) grupează un număr de *valoare* octeţi consecutivi din memoria disponibilă, returnând adresa lor de început (adresa primului octet).

Expresia (int*) indică faptul că adresa de început va foi un pointer de tip int, reprezentând o conversie de tip (cast). Expresia nu este necesară în limbajul C, ea fiind scrisă pentru portabilitatea programului.

Adresa returnată este memorată în *ptrint*. A fost astfel creată dinamic o variabilă întreagă, care poate fi referită prin **ptrint*.

Dacă nu s-ar utiliza instrucţiunea descrisă detaliat mai sus, *ptrint* ar conţine o valoare reprezentând adresa utilizată, dar nu există siguranţa că acea zonă de memorie este liberă. Concluzia evidentă este următoarea: înaintea utilizării unui pointer, acestuia trebuie întotdeauna să i se atribuie o adresă.

Exemplul 4.
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.

```
#include <stdio.h>
#include <alloc.h>
void main (void)
{
        float *a, val, x;
        int i;
        clrscr();
        if ((a=(float*)malloc(3*sizeof(float))) = = NULL)
                {
                        printf("Memorie insuficientă");
                        exit(1);
                }
        printf("|nIntroduceți valorile coeficienților \n");
        for (i = 0; i <=2; i++)
                {
                        printf("a%d=",i);
                        scanf("%f",&val);
                        *(a+i) = val ;
                }
        printf("Introduceți valoarea lui x");
        scanf("%f", &x);
        val = *a*x*x+*(a+1)*x+(a+2);
        printf("Valoarea polinomului P(%f) =%f",x,val);
        free(a);
        getch();
}
```

Noutatea acestui program constă în modul în care sunt memorate şi utilizate valorile coeficienților polinomiali a_0, a_1 şi a_2. Programul evită declararea a trei variabile de tipul float, în care să memoreze coeficienții polinomului. În locul acestora se foloseşte variabila *a*, de tipul pointer la real, şi mecanismul de accesare indirectă.

În program, instrucțiunea

```
if ((a=(float*)malloc(3*sizeof(float))) = = NULL)
```

solicită alocarea unui spațiu de 12 octeți, necesar memorării valorilor celor trei coeficienți polinomiali şi ztestează dacă solicitarea a fost satisfăcută. În acest sens, valoarea returnată de funcția malloc este atribuită pointerului a şi apoi comparată cu constanta NULL. Dacă solicitarea nu a fost satisfăcută, programul afişează mesajul "Memorie insuficientă" şi execuția se încheie prin instrucțiunea exit, care solicită revenirea în mediul de programare.

În cadrul buclei for sunt citite şi memorate succesiv valorile celor trei coeficienţi polinomiali. Mai întâi, 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 ;.

A doua valoare, obţinută pentru *i* =1, nu se va memora la adresa *a* +1, adică în al doilea octet al primei valori, aşa cum am fi tentaţi să credem la o privire fugară. Compilatorul, ştiind că tipul pointerului *a* este float, va interpreta expresia *a* + *i* ca fiind *a* + *i*·sizeof(float) şi deci cele trei valori vor fi memorate corect, fiecărei valori fiindu-i rezervaţi 4 octeţi.

Valorile coeficienţilor astfel memorate sunt utilizate în aceeaşi manieră în cadrul instrucţiunii val = *a*x*x+*(a+1)*x+(a+2); petru a evalua valoarea polinomului.

Din acest exemplu este pusă în evidenţă 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 exemplului 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 insuficientă" 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.

3. Pointeri şi structuri

O structură reprezintă o colecţie de date de tipuri diferite. Considerăm că se doreşte păstrarea unor informaţii despre o persoană anumită.

Se defineşte o structură *pers*. Fiecare variabilă declarată în structură (*nume*, *prenume*, *sex*, *vârstă*, *adresa*, *telefon*) reprezintă un membru al structurii. Noul tip de dată poate fi utilizat pentru declararea unor variabile structură de tip *pers*.

Exemplul 5.

```
#include<stdio.h>
#include <conio.h>
typedef struct
        {
                char nume[15];
                char prenume[15];
                char sex;
                int varsta;
                char adesa[50];
                long telefon;
        } pers;
main()
{
```

```
        pers coleg ;
        strcpy(coleg.nume, " Ionescu");
        strcpy(coleg.prenume, "Dan");
        coleg.sex = 'M';
        coleg.varsta = 29;
        strcpy(coleg.adresa, "Iuliu Maniu 1-3, Bucuresti" );
        coleg.telefon = 1234567;
}
```

Fiecare membru al variabilei structură este referit precedându-l de numele structurii şi caracterul '.'.

Posibilitatea de a declara pointeri la structuri este esențială pentru crearea unor structuri de date dinamice, cum ar fi listele înlănțuite sau arborii.

Să considerăm exemplul anterior, modificat astfel:

Exemplul 6.

```
#include<stdio.h>
#include <conio.h>
typedef struct
        {
                char nume[15];
                char prenume[15];
                char sex;
                int varsta;
                char adesa[50];
                long telefon;
        } pers;
main()
{
        pers *coleg ;
        coleg = (pers*)malloc(sizeof(pers)) ;
        strcpy(coleg->nume, " Ionescu");
        strcpy(coleg->prenume, "Dan");
        coleg->sex = 'M';
        coleg->varsta = 29;
        strcpy(coleg->adresa, "Iuliu Maniu 1-3, Bucuresti" );
        coleg->telefon = 1234567;
}
```

În acest caz, *coleg* este declarat ca un pointer la o structură de tip *pers*, fiindu-i alocat un spațiu cu funcția *malloc*. Pentru referirea unui membru, se foloseşte construcția **ptrstruct->membru**, prescurtare a notației **(*ptrstuct).membru**, având semnificația: "membrul structurii pointate de" .