POINTER LA POINTER.
POINTERI ŞI STRUCTURI

1. Pointer la pointer

În lucrarea de laborator anterioară am luat în consideraţie tipul de dată tablou de pointeri. Având în vedere legătura care există între pointeri şi tablouri, tabloul poate fi privit ca un pointer şi, aşa, cum un tablou de întregi poate fi privit ca o zonă de memorie alocată pentru un pointer la întreg, tot aşa un tablou de pointeri poate fi privit ca un pointer la pointer.

Exemplu (**T1** este un tip oarecare de entitate) în loc de:

```
T1 tab[100];
```

putem folosi în program

```
T1* tab;
```

cu alocarea de memorie corespunzătoare:

```
tab=(T1*)xmalloc(100*sizeof(T1));
```

Dacă, aşa cum am considerat şi în lucrarea anterioară, **T1** este definit prin **typedef** ca un pointer la **T**, adică prin definiţia de tip:

```
typedef T* T1;
```

atunci vom avea, de fapt, declaraţia:

```
T** tab;
```

Adică, un pointer la pointer la T.

În acest caz, trebuie să facem alocare dinamică atât pentru tabloul de pointeri (pe care l-am înlocuit cu un pointer), cât şi pentru fiecare element al tabloului (care este un pointer).

După ce am terminat de lucrat cu un anumit pointer, trebuie să eliberăm zona de memorie alocată pentru fiecare pointer (ca element al unui tablou) şi pentru tabloul de pointeri în ansamblul său.

Un pointer la pointer poate fi echivalat cu un tablou bidimensional (o matrice). Atunci când ne referim la matrice alocate dinamic, ne referim de fapt la tipul de dată pointer la pointer la **T** (**T** este tipul elementelor matricei).

2. Pointeri şi structuri

O structură poate conţine pointeri şi putem defini pointeri la structuri pentru rezolvarea diverselor probleme. Întotdeauna trebuie să facem alocare dinamică pentru pointerul definit (indiferent dacă este membru într-o structură sau un pointer la o structură).

La ieşirea dintr-o funcţie, modificările făcute valorilor membrilor unei structuri nu se păstrează. Din acest motiv, pentru structurile care au membri ale căror valori se modifică în interiorul unor funcţii trebuie să folosim transmiterea informaţiilor cu ajutorul pointerilor la structuri.

De exemplu, pentru structura declarată astfel:

```
struct DATA {
    int zi;
```

```
        char *nume_luna;
        int an;
};
```

şi variabila de tip structură **data** definită:

```
struct DATA d;
```

dacă se apelează funcţia **f**, al cărei parametru este adresa de început a unei zone de memorie rezervată pentru structura **d** de tip **DATA** astfel:

```
f(&d)
```

Funcţia **f** va fi declarată astfel:

```
void f(struct DATA *pd)
```

unde **pd** este un pointer la o structură de tip **DATA**. La apelul funcţiei, pointerul **pd** se va iniţializa cu adresa variabilei de tip structură **DATA** (&d).

Pentru a accesa un membru al ei, utilizăm operatorul de selecţie indirectă "->" (care are prioritate maximă, ca şi operatorul **.**(punct)).

Exemplu

```
pd->an
```

Acest acces este echivalent cu (*pd).an, dar se preferă utilizarea operatorului ->

Deci, pentru a putea modifica valorile membrilor unei structuri în corpul unei funcţii

-se declară funcţia având parametru pointer spre acea structură

```
funcţie( struct nume_structura *nume_parametru_formal)
```

-apelul se face astfel : `functie(&nume_parametru_efectiv)`

-iar în corpul funcţiei se folosesc construcţii de tipul:

```
nume_parametru_formal->nume_membru
```

Se recomandă, pentru lizibilitatea programelor, ca noile nume date de utilizator să fie scrise cu majuscule.

Exemplu

```
typedef struct   _COMPLEX{
    double real, imaginar;
} COMPLEX;
```

Se poate calcula modulul unui număr complex, utilizând următoarea funcţie:

```
double modul (COMPLEX *a)
{
    double val;
    val = sqrt(a->real * a->real + a->imaginar *a->imaginar);
    return val;
}
```

Funcţia modul a utilizat funcţia **sqrt** din biblioteca math.h şi în cazul în care avem secvenţa de cod:

```
COMPLEX c;
double m;
.....................
c={1.2, 3.4};
```

Secvenţa de apel trebuie să fie:

```
m = modul(&c);
```

Se pot defini tablouri de structuri.

Exemplu

Se consideră structura (declarată în header):

```
struct PERSONAL{
    char *nume;
    char initialaTatalui;
    char *prenume;
    struct DATA dataNasterii, dataAngajarii;
};
```

Prin

```
struct PERSONAL tab_pers[20];
```

se definește un tablou de structuri, cu 20 de elemente şi fiecare element, accesat cu tab_pers[i], i = 0, ..., 19, este o structură de tip PERSONAL.

Echivalent, în program se poate folosi un pointer la structura PERSONAL

```
struct PERSONAL *tab_pers;
```

pentru care se face alocare dinamică pentru numărul de elemente necesar. După alocarea dinamică se foloseşte pointerul ca şi cum ar fi un tablou de structuri, tablou a cărui dimensiune coincide cu numărul de elemente stabilit de utilizator.

Dacă se face definiţia:

```
struct PERSONAL *tab_pers[20];
```

atunci **tab_pers** este un tablou de pointeri la structuri şi **tab_pers[i]** este un pointer la o structura de tip PERSONAL. Această construcţie NU este echivalentă cu

```
struct PERSONAL tab_pers[20];
```

TEMA 1

Problema nr. 1.1.

Să se scrie un program care citește de la tastatură elementele de tip real a două matrice alocate dinamic, afișează matricele citite și apoi calculează matricea sumă și o afișează.

Problema nr. 1.2.

Să se definească o structură DATA care corespunde unei date calendaristice și care are patru membri de tip întreg (definind ziua din lună, luna, anul și ziua din an). De exemplu pentru 5 mai 2014, ziua din lună este 5, luna este 4, anul este 2014, iar ziua din an este 125 (dacă anul nu este bisect) sau 126 (dacă anul este bisect)

Să se scrie un program care realizează două tipuri de conversie a datei calendaristice (din care utilizatorul poate alege una singură):

1) citindu-se de la tastatură ziua din lună, luna și anul se calculează ziua din an. Această conversie se face într-o funcție care are prototipul:

```
void conversie1(struct DATA *d);
```

2) citindu-se de la tastatură ziua din an și anul, se calculează ziua din lună și luna. Această conversie se face într-o funcție care are prototipul:

```
void conversie2(struct DATA *d);
```

Indicaţie:

Se va folosi, un tablou pentru numărul de zile din cele 12 luni ale anului. Se va ţine cont şi dacă anul este bisect (se pot declara 2 tablouri, pentru an bisect /nebisect). Funcţiile vor folosi ca argument un pointer la o structură de tip DATA. Apelarea se va face utilizând conversie1(&d) sau conversie2(&d);

Problema nr. 1.3.

Se dă o matrice reală, de dimensiune **n** x **m**, care se citeşte din fişierul **in.txt** şi pentru care se face alocare dinamică. Să se ordoneze crescător liniile matricei luând în considerare elementul maxim al liniei.

Date de test:

Matricea care trebuie ordonată:

```
2    5    6
1    1    0
3    1   -3
```

Matricea ordonată:

```
1    1    0
3    1   -3
2    5    6
```

Programul se va rezolva prin intermediul unui proiect şi se vor scrie funcţii pentru: alocare de memorie cu verificarea alocării (funcţia **xmalloc** din curs), alocare dinamică pentru o matrice (privită ca pointer la pointer), dealocarea memoriei folosită pentru stocarea elementelor matricei, citirea unei matrice, afişarea unei matrice, ordonarea unei matrice în funcție de elementul cu valoarea cea mai mare de pe fiecare linie, determinarea elementului cu valoarea maximă dintr-un vector, interschimbarea a două linii din matrice. Toate funcţiile vor folosi operaţii de lucru cu pointeri.

Funcţia de interschimbare linii are prototipul:

```
void swap(TIP **l1, TIP **l2);
```

unde TIP este tipul elementelor matricei (stabilit de utilizator).

Apelul funcţiei **swap** se face prin instrucţiunea:

```
swap(&a[i], &a[i+1])
```

dacă matricea considerată este notată cu **a**.

TEMA 2

Problema nr. 2.1.

Să se definească tipul de dată MATRICE (asociat unei matrice cu elemente reale) ca o structură care cuprinde:

- numele matricei memorat prin intermediul unui pointer (numele este format din mai multe caractere);
- numărul de linii şi numărul de coloane care sunt numere întregi fără semn;
- elementele matricei (de tip real) stocate într-o zonă de memorie alocată dinamic (printr-un pointer la pointer)

Să se scrie un program care pentru o matrice

1. citeşte de la tastatură numele matricei (unul sau mai multe caractere);
2. stochează toate informaţiile referitoare la matrice într-o structură de tip MATRICE, citind de la tastatură numărul de linii şi de coloane şi elementele matricei;
3. afişează pe linii matricea citită (fiecare element cu câte 2 zecimale);
4. afişează un meniu care dă posibilitatea utilizatorului să aleagă una din următoarele prelucrări:

a) ordonarea matricei astfel încât elementele de pe diagonala principală să fie în secvenţă crescătoare

b) ordonarea matricei astfel încât elementele de pe diagonala secundară să fie în secvenţă descrescătoare

c) ordonarea matricei astfel încât elementele de pe linia mediană verticală să fie în secvenţă crescătoare (în cazul în care matricea are un număr impar de coloane).

Programul poate face o singură prelucrare în funcţie de opţiunea utilizatorului şi va trebui scris astfel încât să permită prelucrarea mai multor seturi de date.

Observaţii:

1) Punctajul maxim se acordă pentru rezolvarea **CORECTĂ** a fiecărei subprobleme.

Barem de notare

1.Încărcarea informaţiilor într-o structură de tip MATRICE (funcţia primeşte ca parametru numele matricei – un pointer şi returnează o structură de tip MATRICE)	1,0
2. Alocarea corectă de spaţiu de memorie pentru pointeri (nume şi matrice)	1,0
3. Citirea elementelor matricei (într-o funcţie care primeşte ca parametri două numere naturale şi returnează un pointer la pointer la real)	1,0
4. Afişarea pe monitor a matricei citite (funcţia are un parametru – un pointer la o structură de tip MATRICE şi nu returnează nimic)	0,5
5. Scrierea meniului de prelucrare (funcţia nu are nici un parametru şi returnează un număr care reprezintă numărul opţiunii de prelucrare)	0,5
6. posibilitatea de reluare a programului (prelucrarea mai multor seturi de date)	0,5
7. Folosire proiect (corect)	0,5
8. Fişier header (corect şi complet)	0,4
8a. Definirea corectă a structurii de tip MATRICE	0,1
9. Funcţia main (complet – inclusiv eliberarea corectă a zonelor de memorie folosite)	1,0
10. Interschimbarea valorilor a doi vectori de reali (funcţia primeşte ca parametri doi pointeri la pointer la real şi nu returnează nimic) – funcţie folosită în funcţiile de ordonare	0,5
11. Ordonarea (prin metoda bulelor) după diagonala principală (funcţia primeşte ca parametru un pointer la o structură de tip MATRICE şi nu returnează nimic)	1,0
12. Ordonarea (prin metoda bulelor) după diagonala secundară (funcţia primeşte ca parametru un pointer la o structură de tip MATRICE şi nu returnează nimic)	1,0
13. Ordonarea (prin metoda bulelor) după linia mediană verticală (funcţia primeşte ca parametru un pointer la o structură de tip MATRICE şi nu returnează nimic)	1,0
TOTAL	**10** p

Problema nr. 2.2.

Se consideră o structură PERSONAL cu următorul prototip:

```
struct   _PERSONAL{
     char *nume;
     char *prenume;
     int virsta;
};
```

Pentru prelucrarea datelor unei companii care poate avea cel mult 20 de angajați se folosește:

a) un vector de structuri alocat static (definit ca mai jos):

```
struct   _PERSONAL tab_pers[20];
```

sau

b) un vector de structuri alocat dinamic

```
struct  _PERSONAL *tab_pers = 0;
tab_pers = (struct  _PERSONAL)
                  xmalloc(n * sizeof(struct  _PERSONAL));
```

unde **n** este numărul angajaților companiei respective.

Să se scrie un program care realizează următoarele:
- citeşte datele pentru **n** persoane; **n** este citit de la tastatură;
- afişează într-un tabel datele pentru aceste **n** persoane;
- afişează toate persoanele care au vârsta sub 30 ani;
- afişează toate persoanele ordonate alfabetic după nume.

Problema nr. 2.3.

Să se determine frecvenţa de apariţie a unor cuvinte cheie dintr-un text (în particular dintr-un program **C**). Textul se memorează pe linii într-un tablou de pointeri la caracter (după exemplul problemei nr. 1 din laboratorul precedent). Textul va conţine cel puţin 3 cuvinte cheie. Se va folosi un tablou de structuri al căror cu prototip este:

```
struct CHEI {
     char *cuv;    /* cuvântul */
     int contor;  /* numărul de apariţii ale cuvântului*/
};
```

Tabloul de structuri va fi inţializat astfel:

```
struct CHEI tab_chei[]= {
     {"break", 0},
     {"case", 0},
     /*alte cuvinte cheie */
     {"while", 0}
     };
```