Laborator 1 - Pointeri

1.1 Probleme rezolvate

P1.1 Să se realizeze un program ce permite citirea de la tastatură a două numere întregi şi calculează suma, diferența şi media aritmetică a lor. Operațiile se vor realiza indirect, prin intermediul unor variabile pointer. Să se afişeze pe ecran valorile, adresele de memorie și valorile indicate de pointeri cât şi valorile şi adresele de memorie pentru rezultate.

Rezolvare:

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

int main()
{
 /* Declarare variabile */
 int x, y, suma, diferenta;
 float medie;

 /* Declarare pointeri */
 int *a, *b;

 /* Initializare pointeri */
 a = &x;
 b = &y;

 /* Citire valori */
 printf("Introduceti primul numar: ");
 scanf("%d", a);
 printf("Introduceti al doilea numar: ");
 scanf("%d", b);

 /* Afisare valoare, adresa si valoare indicata de pointer
 printf("\nContinutul primului pointer este %d\n", a);
 printf("Adresa de memorie a primului pointer este %d\n", &a);
 printf("Valoarea indicata de primul pointer este %d\n", *a);

 printf("\nContinutul celui de-al doilea pointer este %d\n", b);
 printf("Adresa de memorie a celui de-al doilea pointer
        este %d\n", &b);
 printf("Valoarea indicata de al doilea pointer este %d\n\n", *b);
```

```
 /* Calcul valori */
 suma = *a+*b;
 diferenta = *a-*b;
 medie = (float)(*a+*b)/2;

 /* Afisare valori rezultate */
 printf("Suma celor 2 numere este %d\n", suma);
 printf("Diferenta celor 2 numere este %d\n", diferenta);
 printf("Media aritmetica a celor 2 numere este %.2f\n\n", medie);

 /* Afisare adrese de memorie rezultate */
 printf("Adresa de memorie a sumei este %d\n", &suma);
 printf("Adresa de memorie a diferentei este %d\n", &diferenta);
 printf("Adresa de memorie a mediei aritmetice este %d\n", &medie);

 system("PAUSE");
 return 0;
}
```

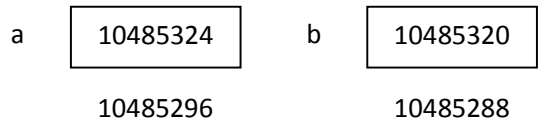
Programul va afişa pe ecran, după compilare şi execuţie următoarele:
(atenție că valorile adreselor de memorie depind de momentul execuției programului și este normal să nu fie identice cu cele din exemplu)

```
Introduceti primul numar: 4
Introduceti al doilea numar: 5

Continutul primului pointer este 10485324
Adresa de memorie a primului pointer este 10485296
Valoarea indicata de primul pointer este 4

Continutul celui de-al doilea pointer este 10485320
Adresa de memorie a celui de-al doilea pointer este 10485288
Valoarea indicata de al doilea pointer este 5

Suma celor 2 numere este 9
Diferenta celor 2 numere este -1
Media aritmetica a celor 2 numere este 4.50

Adresa de memorie a sumei este 10485316
Adresa de memorie a diferentei este 10485312
Adresa de memorie a mediei aritmetice este 10485308
```

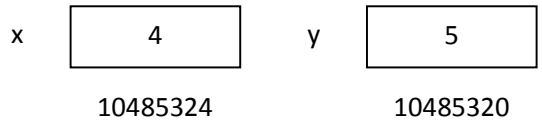
Discuţie:

- Acest program pune în evidenţă declararea şi iniţializarea pointerilor;

- Declararea pointerilor se face prin utilizarea tipului de date pentru valori întregi: `int *a, *b;` Rezultatul este alocarea a două locații de memorie ce vor stoca adrese de memorie pentru valori de tip `int`;
- Pointerii `a` şi `b` se inițializează cu adresele de memorie ale variabilelor `x` şi `y`.



- Funcţia `scanf()` atribuie valorile citite de la tastatură la adresele de memorie ale variabilelor `x` şi `y` specificate prin pointerii `a` și `b`:



- Operațiile se realizează folosind valorile de la adresele de memorie indicate de pointeri prin operatorul de indirectare *: `*a` $\rightarrow$ `4`, `*b` $\rightarrow$ `5`, `*a+*b=4+5=9`. Același raționament se aplică pentru diferență și medie.

P1.2 Să se realizeze un program ce permite citirea de la tastatură a unui vector cu 10 elemente numere întregi. Să se afişeze pe ecran adresele de memorie ale elementelor vectorului şi apoi să se determine suma elementelor acestuia. Operațiile se vor realiza prin intermediul pointerilor.

Rezolvare:

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

int main()
{
 /* Declarare variabile */
 int V[10], dim, i, suma;

 /* Citire dimensiune vector si verificare */
 do
 {
  printf("Introduceti numarul de elemente ale vectorului: ");
  scanf("%d", &dim);
  if ((dim<1) || (dim>10))
   printf("Numar invalid. Acesta trebuie sa fie intre 1 si 10.\n");
 } while ((dim<1) || (dim>10));

 /* Citire vector, element cu element */
 printf("\nCitire valori vector:\n");
```

```
  for (i=0; i<dim; i++)
  {
   printf("V[%d] = ", i+1);
   scanf("%d", V+i);  // stocare la adresele V+i <-> &V[i]
  }

  /* Afisare adrese de memorie ale elementelor vectorului */
  for (i=0; i<dim; i++)
   printf("Adresa de memorie a elementului de indice %d este: %d\n",
            i, V+i);

  /* Calcul suma */
  suma=0;
  for (i=0; i<dim; i++)
   suma += *(V+i);

  printf("Suma elementelor vectorului este %d\n", suma);

  system("PAUSE");
  return 0;
}
```

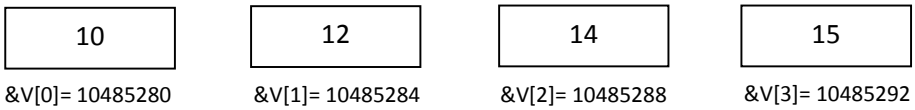
Programul va afişa pe ecran, după compilare şi execuţie:
(atenție că valorile adreselor de memorie depind de momentul execuției programului și este normal să nu fie identice cu cele din exemplu)

```
Introduceti numarul de elemente ale vectorului: 4

Citire valori vector:
V[1] = 10
V[2] = 12
V[3] = 14
V[4] = 15
Adresa de memorie a elementului de indice 0 este: 10485280
Adresa de memorie a elementului de indice 1 este: 10485284
Adresa de memorie a elementului de indice 2 este: 10485288
Adresa de memorie a elementului de indice 3 este: 10485292
Suma elementelor vectorului este 51
```

Discuţie:

- Programul pune în evidenţă utilizarea pointerilor pentru citirea, afişarea şi efectuarea de operaţii cu vectori de valori;
- Citirea elementelor vectorului se face cu ajutorul funcţiei `scanf()`, iar valorile introduse de la tastatură se vor salva la adresele de memorie aferente fiecărui element al vectorului, care sunt exprimate prin pointeri. De reamintit este faptul că prin declararea unui vector, de exemplu `int V[10];` variabila vector `V` va stoca întotdeauna adresa de memorie a primului element din vector, fiind astfel un pointer. Astfel, `V+i` reprezintă de fapt o operație cu adrese,

și anume adăugarea la adresa lui `V` a `i` unități de memorie de tip `int`, în acest caz 32 de biți. Rezultă astfel adresa de memorie a primului element, decalată cu `i` unităţi, adică adresa de memorie a elementului de indice `i` (echivalent cu `&V[i]`). Secvenţa `scanf("%d", V+i)` este echivalentă cu secvenţa `scanf("%d", &V[i])` studiată până acum:



- Suma elementelor vectorului se calculează prin accesarea valorilor de la adresele de memorie indicate de pointeri cu ajutorul operatorului de indirectare *, şi anume `*(V+i)`, secvenţă echivalentă cu `V[i]`. Suma se iniţializează cu 0, după care este incrementată cu fiecare din aceste valori de la fiecare dintre locațiile vectorului.

P1.3 Să se realizeze un program ce permite citirea de la tastatură a unei matrice de numere întregi, de dimensiuni 10 x 20. Citirea matricei se va realiza cu ajutorul unei funcții. Să se afişeze pe ecran minimul şi maximul valorilor de pe fiecare linie a matricei. Calculul minimului şi maximului valorilor unei linii a unei matrice se va realiza cu o funcție.

Rezolvare:

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

void citire_matrice(int M[10][20], int dim1, int dim2)
{
 int i, j;

 for (i=0; i<dim1; i++)
  for (j=0; j<dim2; j++)
  {
   printf("M[%d][%d] = ", i+1, j+1);
   scanf("%d", &M[i][j]);
  }
}

void afisare_matrice(int M[10][20], int dim1, int dim2)
{
 int i, j;

 for (i=0; i<dim1; i++)
 {
  for (j=0; j<dim2; j++)
   printf("%5d", M[i][j]);
```

```
 printf("\n");
 }
}

void calcul_minim_maxim(int M[10][20], int dim2, int linie,
                        int *minim, int *maxim)
{
 int j;

 //initializare minim si maxim cu prima valoare de pe linie
 *minim = M[linie][0];
 *maxim = M[linie][0];

 for (j=1; j<dim2; j++)
 {
  //actualizare minim
  if (M[linie][j] < *minim)
   *minim = M[linie][j];
  //actualizare maxim
  if (M[linie][j] > *maxim)
   *maxim = M[linie][j];
 }
}

int main()
{
 /* Declarare variabile */
 int M[10][20];
 int dim1, dim2, i;
 int minim, maxim;

 /* Citire dimensiuni matrice si verificare */
 do
 {
  printf("Introduceti numarul de linii si de coloane: ");
  scanf("%d", &dim1);
  scanf("%d", &dim2);
  if ((dim1<1) || (dim1>10) || (dim2<1) || (dim2>20))
   printf("Numere invalide. Matricea trebuie sa contina maxim 10
   linii si 20 de coloane.\n");
 } while ((dim1<1) || (dim1>10) || (dim2<1) || (dim2>20));

 /* Citire elemente matrice */
 printf("\nCitire valori matrice:\n");

 citire_matrice(M, dim1, dim2);//dim1 si dim2 sunt transmise prin
 valoare iar M prin adresa
```

```
 /* Afisare elemente matrice */
 printf("\nMatricea M este:\n");

 afisare_matrice(M, dim1, dim2);
 //dim1 si dim2 sunt transmise prin valoare iar M prin adresa


 /* Calcul si afisare minim si maxim pentru fiecare linie */
 for (i=0; i<dim1; i++)
 {
  calcul_minim_maxim(M, dim2, i, &minim, &maxim);
  // Valorile minim si maxim se transmit prin adresa

  printf("Linia %d: minim = %d, maxim = %d\n", i+1, minim, maxim);
 }

 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie:

```
Introduceti numarul de linii si de coloane: 2 2

Citire valori matrice:
M[1][1] = 1
M[1][2] = 2
M[2][1] = 3
M[2][2] = 4

Matricea M este:
    1    2
    3    4
Linia 1: minim = 1, maxim = 2
Linia 2: minim = 3, maxim = 4
```

Discuţie:

- Acest program pune în evidenţă utilizarea pointerilor în lucrul cu funcţii. Folosind pointeri, funcţia `calcul_minim_maxim` permite modificarea a două variabile, şi anume `minim` şi `maxim` ce sunt transmise ca parametri de intrare. Modificarea acestora este posibilă deoarece variabilele sunt transmise prin *adresă*, spre deosebire de cazurile studiate până acum, când variabilele de intrare erau transmise prin *valoare*;
- Funcția de calcul a minimului și maximului este definită cu următorul prototip: `void calcul_minim_maxim(int M[10][20], int dim2, int linie, int *minim, int *maxim);` astfel funcția nu returnează nimic (`void`) și primește la intrare matricea `M`, numărul de coloane transmis prin valoare (`dim2`), indicele liniei pentru care se

calculează valoarea minimă şi respectiv maximă transmis prin valoare (`linie`), cât şi valorile pentru minim şi maxim transmise prin pointeri la int (`int *minim` şi `int *maxim`). În cadrul funcţiei sunt parcurse elementele liniei specificate, în număr de `dim2`, şi calculate valorile pentru minim şi maxim. Acestea sunt stocate folosind operatorul de indirectare *, la locaţiile indicate de pointerii `minim` şi `maxim`, şi anume `*minim` şi `*maxim` (conţinutul locaţiei de memorie indicată de pointeri).

- Funcţia este apelată în programul principal prin `calcul_minim_maxim(M, dim2, i, &minim, &maxim);` furnizând pentru variabilele minim şi maxim adresele de memorie ale acestora (ceea ce este echivalent cu valoarea unui pointer ce ar indica la acestea);
- Având în vedere că parametrii sunt transmişi prin *adresă*, modificarea valorilor de la adresele de memorie indicate de pointeri în interiorul funcţiei implică modificarea acestora şi în programul principal. În acest mod se poate lucra în continuare cu valorile calculate în interiorul funcţiei (de exemplu, afişarea valorilor pentru fiecare linie în programul principal).

P1.4 Să se realizeze un program ce permite citirea de la tastatură şi afişarea a două matrice de numere întregi, de dimensiuni introduse de la tastatură. Programul va calcula şi afişa matricea diferenţă a celor două matrice, calculată element cu element. Alocarea memoriei pentru matrice se va realiza dinamic. Pentru soluționarea problemei se vor folosi pointeri.

Rezolvare:

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

int *citire_matrice(int *dim1, int *dim2)
{
 int i, j;
 int *M;

 if ((*dim1 == 0) && (*dim2 == 0))
 {
  //citire dimensiuni dorite, dim1 si dim2 sunt deja adrese
  printf("Introduceti numarul de linii: ");
  scanf("%d", dim1);
  printf("Introduceti numarul de coloane: ");
  scanf("%d", dim2);
 }

 //alocare memorie (*dim1) x (*dim2) x 32 biti (int)
 M = (int *)malloc((*dim1)*(*dim2)*sizeof(int));

 if (!M)
 {
  printf("\nMemoria nu a putut fi alocata!");
  exit(1);
 }
```

```
  printf("\nCitire valori matrice:\n");

  for (i=0; i<*dim1; i++)
    for (j=0; j<*dim2; j++)
    {
      printf("M[%d][%d] = ",i+1,j+1);
      //stocare la adresele M+*dim2*i+j <-> &M[i][j]
      scanf("%d",M+*dim2*i+j);
    }

  // returnare adresa matrice
  return M;
}

void afisare_matrice(int *M, int dim1, int dim2)
{
  int i,j;

  for (i=0; i<dim1; i++)
  {
    for (j=0; j<dim2; j++)
      //continutul de la adresa M+dim2*i+j <-> M[i][j]
      printf("%8d",*(M+dim2*i+j));

    printf("\n");
  }
}

int *calcul_diferenta(int *M1, int *M2, int dim1, int dim2)
{
  int i, j;
  int *M;

  //alocare memorie dim1 x dim2 x 32 biti (int)
  M = (int *)malloc(dim1*dim2*sizeof(int));
  if (!M)
  {
    printf("\nMemoria nu a putut fi alocata!");
    exit(1);
  }

  for (i=0; i<dim1; i++)
    for (j=0; j<dim2; j++)
      *(M+dim2*i+j) = *(M1+dim2*i+j)-*(M2+dim2*i+j);

  // returnare adresa matrice
  return M;
}


void eliberare_matrice(int *M)
```

```
{
 if (M!=NULL)
  //verificare daca M a fost alocat
  free(M);
}

int main()
{
 /* Declarare variabile */
 int *M1, *M2, *M3; //pointeri la int
 int dim1=0, dim2=0, i, j;

 /* Citire matrice */
 M1 = citire_matrice(&dim1, &dim2);
 M2 = citire_matrice(&dim1, &dim2);
 //dim1 si dim2 sunt transmise prin adresa

 /* Afisare matrice */
 //dim1 si dim2 sunt transmise prin valoare
 printf("\nMatricea M1 este:\n");
 afisare_matrice(M1, dim1, dim2);
 printf("\nMatricea M2 este:\n");
 afisare_matrice(M2, dim1, dim2);

 /* Calcul matrice diferenta */
 M3 = calcul_diferenta(M1, M2, dim1, dim2);

 /* Afisare matrice diferenta */
 printf("\nMatricea diferenta este:\n");
 afisare_matrice(M3, dim1, dim2);

 /* Eliberare memorie alocata */
 eliberare_matrice(M1);
 eliberare_matrice(M2);
 eliberare_matrice(M3);

 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie:

```
Introduceti numarul de linii: 2
Introduceti numarul de coloane: 2

Citire valori matrice:
M[1][1] = 1
M[1][2] = 2
M[2][1] = 3
M[2][2] = 4
```

```
Citire valori matrice:
M[1][1] = 2
M[1][2] = 2
M[2][1] = 2
M[2][2] = 2

Matricea M1 este:
       1       2
       3       4

Matricea M2 este:
       2       2
       2       2

Matricea diferenta este:
      -1       0
       1       2
```

Discuţie:

- Programul pune în evidenţă alocarea dinamică a memoriei pentru variabile, în particular matrice. Alocarea dinamică este utilă deoarece permite folosirea mai eficientă a memoriei. De exemplu, alocăm memorie atunci când avem nevoie de o anumită variabilă, alocăm spaţiu atât cât avem nevoie şi apoi eliberăm memoria când nu mai avem nevoie de acea variabilă;
- Dacă în cazurile studiate până acum spaţiul de memorie era alocat *static*, permanent, indiferent de faptul că foloseam toate locațiile de memorie sau nu - de exemplu `int A[10][10]` are ca efect alocarea pe parcursul rulării programului a unui spațiu de memorie fix de 10x10x32 biţi (`int`) - în cazul alocării dinamice, spaţiul de memorie este specificat de programator;
- Pentru alocarea dinamică este necesară cunoașterea volumului de memorie dorit la momentul alocării. Dacă acest lucru este evident pentru variabile simple, de exemplu știm că o variabilă de tip `int` are nevoie de 32 de biți, o variabilă de tip `char` de 8 biți și așa mai departe, acest lucru nu este evident pentru tipuri de date structurate. Pentru determinarea necesarului de memorie se poate folosi astfel funcția `sizeof(x)` ce returnează numărul de octeți ai variabilei `x` (în locul unei variabile se poate specifica și un tip de date);
- Alocarea dinamică a memoriei se face cu ajutorul funcției `malloc` ce acceptă ca parametru de intrare numărul de octeţi necesari (de exemplu `dim1*dim2*sizeof(int)` - dimensiunea 1 a matricei x dimensiunea 2 x dimensiunea unui `int`) și returnează un pointer la locația de memorie care a fost alocată. În cazul în care memoria nu a putut fi alocată, se returnează un pointer vid (`NULL`);
- Citirea unei matrice se realizează cu funcția `int *citire_matrice(int *dim1, int *dim2)`. Funcția primește drept parametri cele două dimensiuni transmise prin adresă pentru a putea fi citite în cadrul funcției. Este alocată memorie pentru o matrice folosind funcția

`malloc`. Returnarea valorilor matricei se realizează de această dată prin returnarea unui pointer la locația de memorie a matricei și nu prin furnizarea acesteia drept parametru de intrare pentru funcție;

- Afișarea unei matrice se realizează cu funcția `void afisare_matrice(int *M, int dim1, int dim2)`, aceasta nu returnează nici o valoare, iar matricea este transmisă prin adresa de memorie a primului element (`M`). Dimensiunile matricei sunt furnizate prin valoare;
- Valorile matricei sunt parcurse într-un mod mai eficient prin intermendiul adreselor de memorie. Astfel, pentru citire, se generează adresele de memorie la care vor fi stocate valorile matricei prin operații cu adrese: `M+*dim2*i+j`, adică locația de memorie este adresa primului element (`M`) la care se adaugă `i` linii + `j` elemente (ceea ce este echivalent `&M[i][j]`). Pentru afișare se folosește aceeași convenție, doar că de această dată ne interesează conținutul locațiilor de memorie și nu adresele. Se folosește operatorul de indirectare *, astfel *(`M+dim2*i+j`) este echivalent cu `M[i][j]`;
- Diferența dintre matrice se realizează cu funcția `int *calcul_diferenta(int *M1, int *M2, int dim1, int dim2)`. Aceasta primește drept parametri cele două matrice prin intermediul pointerilor (adreselor la primele elemente) cât și dimensiunile acestora transmise prin valoare. Noua matrice calculată este alocată dinamic și apoi returnată ca pointer la adresa primului element;
- Funcția `void eliberare_matrice(int *M)` realizează eliberarea memoriei pentru o matrice transmisă prin adresa primului element. Pentru acest lucru se apelează funcția `free`, care primeste la intrare o adresă de memorie și eliberează spațiul alocat anterior cu `malloc` la această adresă.

1.2 Probleme propuse

1. Să se realizeze un program ce permite citirea de la tastatură a două şiruri de caractere şi afişează pe ecran şirul concatenat. Operaţiile se vor realiza folosind pointeri.

2. Să se realizeze un program ce permite interschimbarea a două linii ale unei matrice pătratice de elemente reale. Se vor defini şi utiliza funcţii diferite pentru citirea, afişarea şi pentru interschimbarea liniilor unei matrice. Operaţiile se vor realiza folosind pointeri.