Laborator II

ALOCAREA DINAMICĂ A MEMORIEI

1. Scopul lucrării

Lucrarea de laborator propune experimentarea alocării dinamice a memoriei și utilizarea pointerilor în acest context. Studiați materialul pe care-l aveți la dispoziție pentru a pregăti lucrarea de laborator. Nu vă lăsați înșelați, nu lăsați pregătirea lucrării în laborator.

2. Alocarea dinamică a memoriei

În continuare se va aloca dinamic memorie pentru o matrice de *int*. Dimensiunile matricii sunt preluate de la tastatură. În *figura 1* este prezentată soluția de memorare pentru matrici bidimensionale utilizată în continuare. Exista și alte soluții de memorare. (Încercați să imaginați cel puțin încă una!) Avantajele majore ale acestei soluții sunt: posibilitatea utilizării imediate a indexării utilizând operatorul de indexare; spre exemplu A[i][j] unde A este pointer la pointer la obiect de tipul *tip* funcționează, și sunt necesare mai multe zone de memorie continuă cu dimensiunea maximă egala cu dimensiunea unei linii.

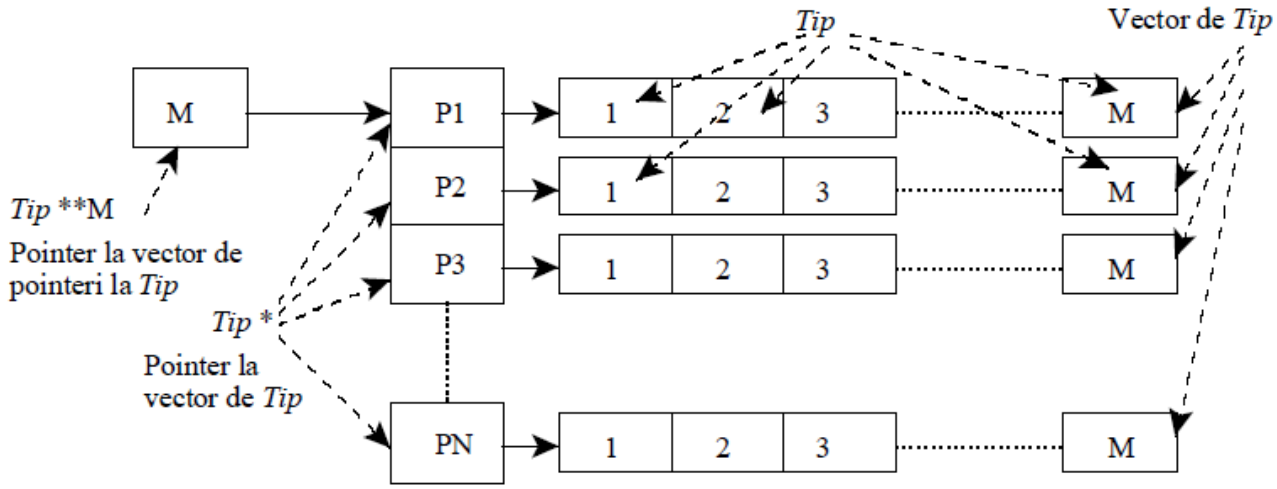

Figura 1. O soluție de memorare pentru matrici bidimensionale

3. Program 1

Acest program este scris folosind facilitățile limbajului C deoarece aceste facilități sunt disponibile în C++ și se constituie în "unelte" puternice de programare. Programul ilustrează paradigma de programare funcțională.

Construiți un proiect din fișierele matrixc1.c, memalloc.c, memerror.c, printmat.c. din directorul *matrici1*. Conținutul acestor fișiere este prezentat în continuare.

```
/*********************************************
matrixc1.h
*********************************************/
```

```
#include <malloc.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>

#ifndef _MATRICIC1_INCLUDED_
#define _MATRICIC1_INCLUDED_
```

```
enum Boolean
{
    false = 0,
    true = 1
};

enum Error
{
    error = 0,
    success = 1
};

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True, altfel returneaza False.*/
int NULLMemTest(void *);

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True si afiseaza un mesaj, altfel returneaza False.*/
int AllocError(void *);

/*Aloca dinamic memorie pentu o matrice cu NrLin linii si NrCol coloane.
Returneaza error sau success.*/
int AllocMatrixInt2(int ***, int , int);

/*Elibereaza memoria alocata dinamic pentru o matrice.
Memoria poate fi alocata folosind malloc sau calloc !!!
Returneaza error sau success.*/
int FreeMatrixInt2(int ***, int);

/*Tipareste la consola matricea de intregi specificata.*/
void PrintMatrixInt2(int **, int , int);

#endif
```

```
/*********************************************
matrix1.c
*********************************************/
```

```
#include "matrixc1.h"

void main(void)
{
    int **Matrix = NULL;
    int NrLin, NrCol;
    int contlin, contcol;

    puts("Introduceti numarul de linii: ");
    scanf("%d",&NrLin);

    puts("Introduceti numarul de coloane: ");
    scanf("%d",&NrCol);


    if(!AllocMatrixInt2(&Matrix, NrLin, NrCol))
        return;

    PrintMatrixInt2(Matrix, NrLin, NrCol); /*Matricea nu a fost initializata ...*/

    for(contlin = 0; contlin < NrLin; contlin++)
        for(contcol = 0; contcol < NrCol; contcol++)
                Matrix[contlin][contcol] = contlin+ contcol;

    PrintMatrixInt2(Matrix, NrLin, NrCol);
```

```
    FreeMatrixInt2(&Matrix, NrLin);

    /*Apel aproape incorect. Era rau tare daca functia FreeMatrixInt2 nu stia sa
rezolva situatia ...*/
    FreeMatrixInt2(&Matrix, NrLin);

    /*free(Matrix);*/ /*Apel total eroant.*/

    getch();
}

/*********************************************
memalloc.c
*********************************************/

#include "matrixc1.h"

/*Aloca dinamic memorie pentu o matrice cu NrLin linii si NrCol coloane.
Returneaza error sau success.*/
int AllocMatrixInt2(int ***Matrix, int NrLin, int NrCol)
{
    int contorlin, contorlinsupl;
    int **MatrixTemp;

    /*Se aloca memorie pentru pointeri la linii ...*/
    *Matrix = (int **)malloc(NrLin * sizeof(int *));
    if(AllocError(*Matrix))
        return error;

    MatrixTemp = *Matrix;
    for(contorlin = 0; contorlin < NrLin; (contorlin++, MatrixTemp++))
    {
        /*Se aloca memorie pentru o linie ...*/
        *MatrixTemp = (int *)malloc(NrCol * sizeof(int));
        if(AllocError(*MatrixTemp))
        {
            /*In caz de eroare la alocare se sterge memoria deja alocata ...*/
            MatrixTemp = *Matrix;
              for(contorlinsupl=0; contorlinsupl<contorlin; (contorlinsupl++, MatrixTemp++))
              free(*MatrixTemp);
            free(*Matrix);
            *Matrix = NULL;
            return error;
        }
    }
    return success;
}

/*Elibereaza memoria alocata dinamic pentru o matrice. Memoria poate fi alocata
folosind malloc sau calloc !!! Returneaza error sau success.*/
int FreeMatrixInt2(int ***Matrix, int NrLin)
{
    int contlin;
    int **MatrixTemp = *Matrix;

    /*Se testeaza daca memoria pentru matrice a fost alocata ...*/
    if(NULLMemTest(*Matrix))
        return error;
    /*Se sterge memoria pe linii ...*/
    for(contlin = 0; contlin < NrLin; (contlin++, MatrixTemp++))
    {
        /*Se testeaza daca a fost alocata memorie pentru linia curenta ...*/
        if(NULLMemTest(*MatrixTemp))
                return error;
```

```
        free(*MatrixTemp); /*Se sterge memoria pentru linia curenta ...*/
    }
    free(*Matrix);
    *Matrix = NULL;
    return success;
}
```

```
/*********************************************
memerror.c
*********************************************/
```

```
#include "matrixc1.h"

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True, altfel returneaza False.*/
int NULLMemTest(void *Ptr)
{
    if(!Ptr)
        return true;
    return false;
}

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True si afiseaza un mesaj, altfel returneaza False.*/
int AllocError(void * Ptr)
{
    if(NULLMemTest(Ptr))
    {
        puts("Eroare la alocarea memoriei ...\n");
        return true;
    }
    return false;
}
```

```
/*********************************************
printmat.c
*********************************************/
```

```
#include "matrixc1.h"

/*Tipareste la consola matricea de intregi specificata.*/
void PrintMatrixInt2(int **Matrix, int NrLin, int NrCol)
{
    char Test;
    int contlin, contcol;

    puts("Sa afisez matricea ? (y/n)");
    do
    {
        Test = toupper(getch());
    }
    while(Test != 'Y' && Test != 'N');

    if(Test == 'Y')
    {
        puts("Matricea este :");
        for(contlin = 0; contlin < NrLin; contlin++)
        {
            for(contcol = 0; contcol < NrCol; contcol++)
                    printf("%3d",Matrix[contlin][contcol]);
            printf("\n");
        }
    }
}
```

Rulați programul pas cu pas și observați comportarea acestuia. Modificați sursele pentru a accesa defectuos memoria: depășirea domeniului alocat, eliberarea multiplă a aceleiași zone de memorie.

4. Program 2

În continuare este prezentat același program ilustrând paradigma de programare prin abstractizarea datelor.

Construiți un proiect folosind fișierele: matrix2.c, memalloc.c, memerror.c, printmat.c, pe care le găsiți în directorul *matrici2*.

```
/*********************************************
matrixc2.h
*********************************************/
```

```c
#include <malloc.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>

#ifndef _MATRICIC1_INCLUDED_
#define _MATRICIC1_INCLUDED_

enum Boolean
{
    false = 0,
    true = 1
};

enum Error
{
    error = 0,
    success = 1
};

typedef struct MatrixInt2
{
    int **Matrix;
    int NrLin;
    int NrCol;
};

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True, altfel returneaza False.*/
int NULLMemTest(void *);

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True si afiseaza un mesaj, altfel returneaza False.*/
int AllocError(void *);

/*Aloca dinamic memorie pentu o matrice cu NrLin linii si NrCol coloane.
Returneaza error sau success.*/
int AllocMatrixInt2(struct MatrixInt2 *);

/*Elibereaza memoria alocata dinamic pentru o matrice.
Memoria poate fi alocata folosind malloc sau calloc !!!
Returneaza error sau success.*/
int FreeMatrixInt2(struct MatrixInt2 *);

/*Tipareste la consola matricea de intregi specificata.*/
void PrintMatrixInt2(struct MatrixInt2 *);

#endif
```

```
/*********************************************
matrix2.c
*********************************************/
```

```c
#include "matrixc2.h"

void main(void)
{
    struct MatrixInt2 Matrix;
    int contlin, contcol;

    puts("Introduceti numarul de linii: ");
    scanf("%d",&Matrix.NrLin);

    puts("Introduceti numarul de coloane: ");
    scanf("%d",&Matrix.NrCol);


    if(!AllocMatrixInt2(&Matrix))
        return;

    PrintMatrixInt2(&Matrix); /*Matricea nu a fost initializata ...*/

    for(contlin = 0; contlin < Matrix.NrLin; contlin++)
        for(contcol = 0; contcol < Matrix.NrCol; contcol++)
                Matrix.Matrix[contlin][contcol] = contlin+ contcol;

    PrintMatrixInt2(&Matrix);

    FreeMatrixInt2(&Matrix);

    /*Apel aproape incorect. Era rau tare daca functia FreeMatrixInt2 nu stia sa
      rezolva situatia ...*/
    FreeMatrixInt2(&Matrix);

    /*free(Matrix.Matrix);*/ /*Apel total eroant.*/

    getch();
}
```

```
/*********************************************
memalloc.c
*********************************************/
```

```c
#include "matrixc2.h"

/*Aloca dinamic memorie pentu o matrice cu NrLin linii si NrCol coloane.
Returneaza error sau success.*/
int AllocMatrixInt2(struct MatrixInt2 *PtrMatrix)
{
    int contorlin;
    int **MatrixTemp;

    /*Se aloca memorie pentru pointeri la linii ...*/
    PtrMatrix->Matrix = (int **)malloc(PtrMatrix->NrLin * sizeof(int *));
    if(AllocError(PtrMatrix->Matrix))
        return error;

    MatrixTemp = PtrMatrix->Matrix;
    for(contorlin = 0; contorlin < PtrMatrix->NrLin; (contorlin++, MatrixTemp++))
    {
        /*Se aloca memorie pentru o linie ...*/
        *MatrixTemp = (int *)malloc(PtrMatrix->NrCol * sizeof(int));
        if(AllocError(*MatrixTemp))
```

```
        {
            /*In caz de eroare la alocare se sterge memoria deja alocata ...*/
            PtrMatrix->NrLin = contorlin;
            FreeMatrixInt2(PtrMatrix);
            free(PtrMatrix->Matrix);
            PtrMatrix->Matrix = NULL;
            PtrMatrix->NrLin = 0;
            PtrMatrix->NrCol = 0;
            return error;
        }
    }
    return success;
}

/*Elibereaza memoria alocata dinamic pentru o matrice. Memoria poate fi alocata
folosind malloc sau calloc !!! Returneaza error sau success.*/
int FreeMatrixInt2(struct MatrixInt2 *PtrMatrix)
{
    int contlin;
    int **MatrixTemp = PtrMatrix->Matrix;

    /*Se testeaza daca memoria pentru matrice a fost alocata ...*/
    if(NULLMemTest(PtrMatrix->Matrix))
        return error;
    /*Se sterge memoria pe linii ...*/
    for(contlin = 0; contlin < PtrMatrix->NrLin; (contlin++, MatrixTemp++))
    {
        /*Se testeaza daca a fost alocata memorie pentru linia curenta ...*/
        if(NULLMemTest(*MatrixTemp))
                return error;
        free(*MatrixTemp); /*Se sterge memoria pentru linia curenta ...*/
    }
    free(PtrMatrix->Matrix);
    PtrMatrix->Matrix = NULL;
    PtrMatrix->NrLin = 0;
    PtrMatrix->NrCol = 0;
    return success;
}
```

```
/*********************************************
memerror.c
*********************************************/
```

```
#include "matrixc2.h"

/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True, altfel returneaza False.*/
int NULLMemTest(void *Ptr)
{
    if(!Ptr)
        return true;
    return false;
}
/*Testeaza daca valoarea pointerului este NULL.
Daca da returneaza True si afiseaza un mesaj, altfel returneaza False.*/
int AllocError(void * Ptr)
{
    if(NULLMemTest(Ptr))
    {
        puts("Eroare la alocarea memoriei ...\n");
        return true;
    }
    return false;
}
```

```
/*********************************************
printmat.c
*********************************************/
```

```
#include "matrixc2.h"

/*Tipareste la consola matricea de intregi specificata.*/
void PrintMatrixInt2(struct MatrixInt2 *PtrMatrix)
{
    char Test;
    int contlin, contcol;

    puts("Sa afisez matricea ? (y/n)");
    do
    {
        Test = toupper(getch());
    }
    while(Test != 'Y' && Test != 'N');

    if(Test == 'Y')
    {
        puts("Matricea este :");
        for(contlin = 0; contlin < PtrMatrix->NrLin; contlin++)
        {
            for(contcol = 0; contcol < PtrMatrix->NrCol; contcol++)
                    printf("%3d",PtrMatrix->Matrix[contlin][contcol]);
            printf("\n");
        }
    }
}
```

5. Temă

Folosind paradigma de programare prin abstractizarea datelor, adăugați la ultimul proiect structura de date și funcționalitățile necesare pentru vectori (tablou unidimensional):

- alocarea și eliberarea memoriei Heap,
- afișarea valorilor vectorului,
- funcție care realizează produsul matrice * vector.

Modificați funcția *main* pentru a ilustra aceste facilități.

Folosiți intensiv, pentru inspirație, codul pe care-l aveți la dispoziție.