11. Alocarea dinamică C/C++. Gestiunea memoriei (Dynamic allocation in C/C++. Memory management)

1. Obiective:

- Înţelegerea noţiunii de alocare dinamică a datelor în memorie
- Scrierea şi rularea de programe în care este folosită alocare dinamică a datelor în memorie

1'. Objectives:

- Understanding how the data is dynamically allocated in memory
- Writing and running programs that make use of the dynamic memory allocation.

2. Rezumat:

Cele mai utilizate ***funcţii*** pentru **alocarea dinamică în limbajul C** sunt :

malloc()
calloc()
free()

- Funcţia *malloc()* are prototipul: *void * malloc(**unsigned** n);*
- Eliberarea zonei alocate cu funcţia *malloc()* se face folosind funcţia standard *free()* cu următorul prototip:
  *void free(void *p);* // p este pointerul obţinut la apelul funcţiei malloc()
- O altă funcţie standard utilizată pentru a aloca dinamic zona de memorie este funcţia *calloc()* cu prototipul:
  *void * calloc(**unsigned** nrelem, **unsigned** dimelem);*
  Eliberarea acestei zone se face folosind tot funcţia *free()*.
- Realocarea memoriei se poate face cu funcţia *realloc()* declarată în *stdlib.h* având următorul prototip: *void *realloc(void *block, size_t size);*

În **limbajul C++ alocarea dinamică a memoriei** mai poate fi efectuată şi cu ***operatorii*** *new* şi *delete*.

- Operatorul *new*, permite alocarea în zona heap a memoriei. Acesta este un operator unar şi are prioritate ca şi ceilalţi operatori unari. Operatorul *new* are ca valoare adresa de început a zonei de memorie alocată în memoria heap sau zero dacă alocarea eşuează;
- Operatorul *delete* este folosit pentru eliberarea zonei de memorie alocată pe heap cu operatorul *new*. Dacă p este un pointer spre un *tip*, *tip* *p; şi p = *new tip*; atunci zona de memorie alocată în heap se eliberează cu: *delete* p;
- Iniţializări zone de memorie:

*void * memset(void *dst, int c, size_t n);* -setează primii *n* octeţi începând cu adresa *dst* la valoarea dată de caracterul conţinut în parametrul *c* şi returnează adresa şirului pe care lucrează;

- Copierea zonelor de memorie:

a)*void * memcpy(void *dest, const void *src, size_t n);*- copiază un bloc de memorie de lungime *n* de la adresa *src* la adresa *dest*., nu se garantează o funcţionare corectă în cazul în care blocul sursa şi cel destinaţie se suprapun, funcţia returnează adresa blocului destinaţie;
b)*void * _memccpy(void *dest, const void *src, int c, size_t n);*- copiază *n* octeţi de la sursa (adresa *src*) la destinaţie (adresa *dest*) însă copierea se poate opri în următoarele situaţii:

- la întâlnirea caracterului *c* (care este copiat la destinaţie);
- s-au copiat deja *n* octeţi la adresa de destinaţie.

- Mutarea zonelor de memorie:

*void * memmove(void *dest, const void *src, size_t n);-* mută un bloc de lungime *n* de la adresa sursă *src* la adresa destinaţie *dest*. Se garantează copierea corectă, chiar dacă cele două zone de memorie se suprapun; funcţia returnează adresa zonei destinaţie;

- Compararea zonelor de memorie:
  *int memcmp(const void *s1, const void *s2, unsigned n);*
  *int _memicmp(const void *s1, const void *s2, unsigned n);*

Funcţia *memcmp()* compară primii n octeţi ai zonelor de memorie s1 şi s2 şi returnează un întreg mai mic decît, egal cu, sau mai mare decît zero, dacă s1 este mai mic decat, coincide, respectiv este mai mare decît s2.
Funcţia *_memicmp()* face acelaşi lucru, dar fără a ţine cont de litere mari respectiv litere mici (ignoring case).

- Căutarea într-o zonă de memorie:

*void * memchr(const void *s, int c, unsigned n);- f*uncţia *memchr()* caută caracterul *c* în primii n octeţi de memorie indicaţi de s; căutarea se opreşte la primul octet care are valoarea c (interpretată ca unsigned char); funcţia returnează un pointer la octetul găsit sau NULL dacă valoarea nu există în zona de memorie.

- Copierea cu inversarea octetilor adiacenți

*void _swab(char *src, char *dest, int n);*
Funcţia swab() copiază *n* octeţi de la adresa sursă *src* la o adresă destinaţie *dest*, cu interschimbarea octeţilor adiacenţi. Este utilizată pentru a copia date pe o altă maşină, la care ordinea octeţilor este diferită. Argumentul *n* trebuie să fie un număr intreg pozitiv şi par. Pentru *n* impar se vor copia doar *n-1* octeţi.

3. Exemple:
Exemplul 1: program pentru alocarea unui tablou unidimensional folosind funcţii de bibliotecă.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <malloc.h>

int main(void){
int i,n, *tab=NULL;
    printf("\nIntroduceti dimensiunea tablolului: ");
    scanf("%d", &n);
    if((tab = (int *)malloc(n * sizeof(int))))
    {
        printf("\nIntroduceti elementele tablolului: \n");
        for(i=0; i<n; i++)
        {
            printf("\t elementul %d: ", i);
            scanf("%d", tab+i);
        }
        for(i=0; i<n; i++)
            printf("\n tab[%d] = %d", i, tab[i]);
            //    printf("\n tab[%d] = %d", i, *(tab +i));
        printf("\n");
    }
    else
        printf("\nAlocare nereusita !");
    if(tab)
        free(tab);
}
```

Exemplul 2: alocarea unui tablou unidimensional folosind operatorii C++.

```
#include <iostream>
using namespace std;

int main( ){
int i, n, *tab;
        cout << "\n Introduceti dimensiunea tabloului: ";
        cin >> n;
        tab = new int[n];
        if(tab != 0)  {
                cout << "\n Introduceti elementele tabloului : ";
                for(i=0;i<n; i++)  {
                        cout << "\n\t Elementul " << i << " : ";
                        cin >> tab[i];
                        //cin >>*(tab+i);
                }
                cout << "\n\n Elementele tabloului sunt : ";
                for(i=0;i<n;i++)
                        cout << tab[i] << ' ';
                        //cout << *(tab+i) << ' ';
        }
        else
                cout << "Alocare nereusita !";
        if(tab)
                delete [ ]tab;
return 0;
}
```

Exemplul 3: alocare tablou dinamic unidimensional exploatat ca unul bidimensional.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

void cit_mat(int*, int, int);
void afis_mat(int*, int, int);

int main(void) {
        int* tab, m, n;
        printf("\n Introduceti m si n, nr. linii si col.: ");
        //cout<<"\n Introduceti m si n, nr. linii si col.: ";
        scanf("%d%d", &m, &n);
        //cin >> m >> n;
        if ((m <= 0) || (n <= 0)) {
                printf("\nNumere invalide!\tMai incearcati!");
                //cout<<"\nNumere invalide!\tMai incearcati!";
                return 1;
        }// end if
        // alocare dinamica si test pointer
        tab = (int*)malloc(m * n * sizeof(int));
        //tab = new int[m * n];
        if (tab == 0) {
                //cout<<"\n Eroare de alocare !";
```

```
            printf("\n Eroare de alocare !");
            return 1;
        }
        printf("\nIntroduceti elementele matricei %dx%d :", m, n);
        //cout<<"\nIntroduceti elementele matricei " << m <<"x"<< n <<": ";
        cit_mat(tab, m, n);

        // aici se pot face si alte prelucrari...

        printf("\nMatricea arata astfel:");
        //cout<<"\nMatricea arata astfel:";
        afis_mat(tab, m, n);
        // dealocare
        if (tab) free(tab);
        //if (tab) delete [] tab;
}// end main

void cit_mat(int* tb, int l, int c) {
        int i, j;
        for (i = 0; i < l; i++) {
            //cout<<"\nLinia "<< i <<": ";
            printf("\nLinia %d: ", i);
            for (j = 0; j < c; j++)
                //cin >> *(tb + i * c + j);
                scanf("%d", tb + i * c + j);
                                                    }
        }

void afis_mat(int* tb, int l, int c) {
        int i, j;
        for (i = 0; i < l; i++) {
            //cout << "\n\t";
            printf("\n\t");
            for (j = 0; j < c; j++)
                //cout << *(tb + i * c + j) << "  ";
                printf("%d ", *(tb + i * c + j));
        }// end for
        //cout<<endl;
        printf("\n");
}
```

Exemplul 4: alocarea unui tablou bidimensional (matrice) gestionat linie cu linie; citire şi afişare elemente.

a) tablou de intregi

```
#include <iostream>
using namespace std;

int main(void)
{
int i, j, m,n, **tab;
        cout << "\n Introduceti numarul de linii: ";
        cin >> m;
        if((tab = new int *[m]))
```

```
	{
		cout << "\n Introduceti numarul de coloane: ";
		cin >> n;
		for(i=0; i<m; i++)
		{
			tab[i] = new int [n];
			if(tab[i] == 0)
			{
					cout << "\n Eroare de alocare !";
					return 1;
			}
		}
		cout << "\n Introduceti elementele tabloului:";
		for(i=0; i<m; i++)
		{
			cout << "\n\t Linia " << i<<":\n";
			for(j=0; j<n; j++)
			{
			cout << "\t Elementul tab[" << i << "]["<<j<<"]: ";
				cin >> tab[i][j];
			}
		}

		cout << "\n\n Elementele tabloului sunt: \n";
		for(i=0; i<m; i++)
		{
			for(j=0; j<n; j++)
				cout <<"\t"<< tab[i][j];
			cout << "\n";
		}
	}
	else
		cout << "Alocare nereusita !";

	if(tab)
	{
		for(i=0; i<m; i++)
			delete []tab[i];
		delete []tab;
	}
}
```

b) tablou de siruri de caractere

```
#include <iostream>
using namespace std;

int main( ) {
	int i, m, n;
	char** tab;
	cout << "\n Enter no. of strings : ";
	cin >> m;
	tab = new char* [m];//array of pointers to rows
	if (tab != 0)
	{
```

```
		cout << "\n Enter dim of strings : ";
		cin >> n;
		for (i = 0; i < m; i++) {
			tab[i] = new char[n];//each row pointer to n elems allocated as characters
			if (tab[i] == 0) {
				cout << "\n Allocation error! " << endl;
				return 1;
			}
		}
		cout << "\n Enter the " << m << " strings :\n";
		for (i = 0; i < m; i++)
			cin >> tab[i];
		cout << "\n\n Strings are: " << endl;
		for (i = 0; i < m; i++)
			cout << tab[i] << "\n";
	}
	else cout << "Wrong allocation !";
	if (tab) {
		for (i = 0; i < m; i++)
			delete[] tab[i];//for each row remove the space of char values
		delete[] tab;
	}//remove the array of pointers to rows – no. of strings
	return 0;
}
```

Exemplul 5: functie ce returneaza adresa unui tablou alocat dinamic

```
#include <stdio.h>
#define DIM 10
int * foo( );

int main( ) {
   int *p = foo( );
for(int i=0;i<DIM;i++)printf("\nArrays values are = %d", *(p+i));
}
int * foo( ) {
   int *arr = new int[DIM];//dynamic arrays or static arrays may be returned as address
for(int i=0;i<DIM; i++) *(arr+i)=i;
   return arr;
}
```

Exemplul 6: programul copiază în memorie conţinutul variabilei *alphabet* până la întâlnirea caracterului'K' .

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

int main(void) {
   const char *alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   char target[27];
   char* result;

   result = (char*)_memccpy(target, alphabet, 'K', strlen(alphabet));
   if (result)      *result = NULL;
   printf(target); printf("\n");
}//end main
```

Exemplul 7: compară locaţiile de memorie unde au fost stocate şirurile "AAAAA", "BBBBB" şi "aaaaaA" şi demonstrează diferenţa între funcţiile *memcmp( )* şi *_memicmp( ).*

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

int main(void) {
        const char* a = "AAAAA";
        const char* b = "BBBBB";
        const char* c = "aaaaaA";

        printf("Rezultatul compararii  %s cu %s folosind  memcmp( ) este:  %d\n", a, b, memcmp(a, b, strlen(a)));
        printf("Rezultatul compararii  %s cu %s folosind _memicmp( ) este: %d\n", a, c, _memicmp(a, c, strlen(a)));
}
```

Exemplul 8: programul exemplifică utilizarea funcţiei *memmove().*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
const int dim = 5;

int main(void) {
   float values[] = { 1.1f, 2.2f, 3.3f, 4.4f, 5.5f, 6.6f };
   float empty[dim];
   int i;
   memmove(empty, values, sizeof(empty));//mutam atat cat e destinatia, byte cu byte
   for (i = 0; i < _countof(empty); i++)
      printf("%3.1f\t", empty[i]);
   printf("\n");
}
```

Exemplul 9: programul exemplifică funcţia *_swab( ).* Analizati modul de functionare.

```
//memory swab
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main( )
{
        const char* source = "Sir de initializare12345";
        char target[64];
        memset(target, NULL, sizeof(target));
        _swab((char*)source, target, strlen(source));
        printf("Sursa: %s \n", source);
        printf("Destinatie: %s\n", target); // iS redi initlazira1e3254
}
```

4. Întrebări:

- Ce înseamană memoria „*heap*”?
- Prin ce diferă funcţia *malloc* de funcţia *calloc*?

- Cum se poate face alocarea dinamică în limbajul C ? Dar în limbajul C++ ?
- Faceţi o comparaţie între funcţiile de alocare dinamică a memoriei în C şi cele din C++.

5. Teme:

1. Să se scrie un program care citeşte *n* numere reale, pe care le stochează într-un tablou alocat dinamic, afişează suma elementelor negative citite, iar la sfârşit eliberează zona de memorie alocată.
2. Fie o aplicaţie de gestiune distribuită care consideră că tratează activitatea din 5 oraşe diferite în fiecare oraş fiind 3 birouri de vânzare pe teritoriul respectiv. Să se creeze în cadrul unei funcţii un tablou de 5 pointeri către date de tip flotant, fiecare pointer din acest tablou referind o zonă în heap alocată dinamic de 3 date flotante ce reprezintă situaţia vânzărilor la sfârşitul unei zile. Se cere prelucrarea datelor din fiecare oraş, respectiv din fiecare birou de vânzare, prelucrare ce va include:
   - funcţie care permite introducerea datelor pentru cele 5 oraşe şi respectiv pentru fiecare oraş pentru cele 3 magazine din oraş;
   - funcţie ce permite determinarea totalului de vânzări pe fiecare oraş în parte, valoare pe care o va returna astfel că în programul principal se va calcula şi afişa media vânzărilor din toate cele 5 oraşe;
   - funcţie care va permite eliberarea spaţiului de memorie alocat dinamic astfel încât dacă aceeaşi firmă are alte 3 magazine în cele 5 oraşe de profil diferit să poată realoca un spaţiu echivalent pentru noile prelucrări pe care le va efectua.
3. Să se scrie o aplicaţie C/C++, care alocă dinamic memorie pentru stocarea elementelor a două matrici de "m" linii şi "n" coloane. Să se scrie o funcţie care calculează suma celor două matrici şi o funcţie pentru afişarea unei matrici. Să se afişeze matricile iniţiale şi matricea obţinută.
4. Să se scrie o aplicaţie C/C++ care alocă dinamic memorie pentru "n" şiruri de caractere, care se vor citi de la tastatură.
5. Declaraţi un pointer global de tip float. În funcţia *main()* citiţi o valoare întreagă N care reprezintă dimensiunea tabloului de numere reale. Alocaţi memoria necesară pentru stocarea tabloului şi citiţi elementele de la tastatură. Determinaţi valoarea medie a celor N elemente şi calculaţi $Mn=(sum(pow((x_i-xmed),n))/N$, unde $n=1,2,3$. Afişaţi rezultatele şi apoi eliberaţi memoria. Folosiţi funcţiile *malloc()* şi *free()*. Generaţi numerele din tablou folosind funcţia de bibliotecă care generează numere aleatoare şi determinaţi pentru acestea media valorilor şi *Mn*. Realizati aceeasi aplicatie folosind operatorii *new* şi *delete*.
6. Folosiţi alocarea dinamică pentru o matrice m x n cu valori întregi (m, n <7). Initializaţi elementele matricii. Dacă matricea este pătratică, folosiţi metoda lui Sarrus pentru a obţine valoarea determinantului. Afişaţi rezultatul şi eliberaţi memoria.
7. Să se scrie o aplicaţie C/C++ care alocă dinamic memorie necesară pentru stocarea a 10.000 de numere întregi. Programul iniţializează numerele cu valori aleatoare între 1 şi 100 (folosiţi funcţiile s*rand()* şi/sau *rand()* în VC++). Scrieţi o funcţie care afişează cele mai frecvente 10 numere şi numărul lor de apariţii în tabloul iniţial.
8. Să se scrie o aplicaţie C/C++ în care se alocă dinamic memorie pentru n numere întregi, numere ce vor fi citite de la tastatură. Să se scrie funcţia care extrage radicalul din fiecare număr şi stochează valorile obţinute într-un alt tablou alocat dinamic. Să se afişeze numerele iniţiale şi valorile din tabloul format. Eliberaţi la sfârşit memoria alocată.
9. Scrieţi un program în care se citesc *n* şiruri de caractere şi se concatenează într-un alt şir, alocat dinamic. Repetaţi operaţia de câte ori doreşte utilizatorul. După fiecare afişare a şirului obţinut prin concatenare eliberaţi memoria alocată dinamic.
10. Să se scrie o aplicaţie C/C++, care alocă dinamic memorie pentru stocarea elementelor unei matrici de dimensiune nxn. Să se scrie o funcţie care calculează

suma numerelor pozitive pare de sub diagonala principală şi o funcţie pentru afişarea matricei. Să se afişeze matricea şi suma cerută. Eliberaţi memoria alocată dinamic.

11. Generați un tablou de pointeri către șiruri constante folosind funcția *strdup( )* sau o altă funcție specifică. Afișați valorile pare din tablou.
12. Scrieți o aplicație C/C ++ care citește de la tastatură un tablou de până la 10 numere întregi valori în intervalul 2, ..., 10. Dimensiunea reală este introdusă de la tastatură, tabloul fiind alocat dinamic. Definiți două funcții care consideră tabloul ca parametru și returnează valorile minime și maxime din tablou. Definiți o altă funcție care returnează valoarea medie a elementelor tabloului introduse.
    Definiți o altă funcție, *Olympic_filter(...),* care care consideră tabloul inițial, returnând adresa unui tablou alocat dinamic fără prima apariție a valorii minime și maxime.
    În *main( )* se afișează, de asemenea, tablou inițial, valoarea minimă și maximă, media elementelor tabloului inițial, elementele tabloului nou obținute de funcția *Olympic_filter(...) si* media obținută dupa aplicarea filtrului olimpic.
    Validați introducerea datelor în conformitate cu cerințele problemei (dimensiunea tabloului, elementele introduse cu reluare). Operațiile cu elementele tablourilor se vor face folosind pointeri.

5. Homework

1. Write a program that reads from the keyboard *n* real numbers, stores them into a dinamically allocated array, displays the sum of the negative elements and frees the allocated memory.
2. Consider an application for managing a company's distributed activity (5 different cities with 3 selling points in each one of them). Define a function that declares an array of float pointers, each pointer being used for referencing a heap zone of memory that stores 3 float values that represent the total sales at the end of day. The program needs to process the data using functions for:
    - reading the data for all the selling points from all the cities;
    - calculating and returning the total amount of sales in every city. The values will be centralized in the main function and the average value will be displayed.
    - eliberating all the used memory
3. Write a program that allocates the necessary amount of memory for storing the elements from two [m x n] integer matrices. Write a function that calculates the sum of the inital matrices and another one for displaying both the original values and the result.
4. Write a program reads *n* characters arrays from the keyboard and stores them into *n* dinamically allocated memory zones.
5. Declare a global float pointer. In function main(), read an integer value N that represents the dimension of an array of float numbers. Allocate the necessary ammount of memory for storing the array and than read it's elements from the keyboard. Determine the average value of those N elements and calculate $Mn=(sum(pow((x_i-xmed),n))/N$, where $n=1,2,3$. Display the results and then free the allocated memory. Use the functions *malloc()* and *free().* In another function, generate the numbers in the array using a library function that generates *random* numbers and determine the same values. Use *new* and *delete* operators.
6. Use the dynamic allocation for a matrix of m x n integer values (m, n <7). Initialize the elements of the matrix. If n is equal to m, use the Sarrus method to obtain the value of the "determinant". Display the result and free the allocated memory.
7. Write a C/C++ application that allocates the necessary amount of memory for storing 10.000 integer numbers. The program automatically initializes the numbers with random values between 1 and 100 (use the library functions *srand()* and/or *rand()* in VC++). Write a function that displays the most 10 frequently numbers and the number of their appearances in the initial array.
8. Write a C/C++ application that allocates memory for n integer numbers that will be read from the keyboard. Write a function that determines the square root of each

number and stores the result into another dynamically allocated array. Display the initial and computed values. Free the allocated memory.

9. Write a program that reads n arrays of characters and concatenates them into another dynamically allocated array. Repeat the operation as many times as the user desires. After each displaying of the result, the allocated memory is freed.
10. Write a C/C++ application that allocates the necessary amount of memory for storing an nxn integer matrix. Write a function that calculates the sum of the positive numbers located below the main diagonal and another function that displays the matrix. Print the matrix and the calculated sum. Free the allocated memory.
11. Generate an array of pointers to constant strings using the *strdup()* method or a specific method. Display the even entries of the array.
12. Write a C/C ++ application that reads from the keyboard an array of up to 10 integers in range 2, ..., 10. The actual size is entered from the keyboard, the array being allocated dynamically. Define two functions that consider the array as a parameter and return the minimum and maximum values in the array. Define another function that returns the average value of the elements of the entered array.
    Define another function, *Olympic_filter(...),* which considers the initial array, returning the address of a dynamically allocated array without the first occurrence of the minimum and maximum value.
    In *main( )* also display the initial array, the minimum and maximum value, the average of the elements of the initial array, the elements of the newly obtained array returned by the function *Olympic_filter(...)* and the average obtained after the olympic filter. Validate the data entry according to the requirements of the problem (array size, elements entered with resumption). The operations with the elements of the arrays will be done using pointers.