Laborator 8

Alocarea dinamica a memoriei

Alocarea dinamică se referă la procesul prin care un program își rezervă și eliberează resurse de memorie la nevoie, pe durata execuției. Aceasta este o practică comună în programare, în special atunci când nu cunoaștem dinainte câtă memorie va fi necesară sau când vrem să gestionăm eficient resursele disponibile.

Exista 3 moduri de alocare a memoriei:

Static: memoria este alocata la compilare in segmentul de date si nu se mai poate modifica in cursul executiei. Variabilele globale, definite in afara functiilor, sunt implicit statice, dar pot fi declarate static (static) si variabile locale, definite in cadrul functiilor.

Automatic: memoria este alocata automat la executie, cand se apeleaza o functie, in zona stiva alocata unui program si este eliberata automat la terminarea functiei. Variabilele locale ale functiilor sunt implicit din clasa auto.

Dinamic: Alocarea dinamică are loc într-o zonă a memoriei cunoscută sub numele de heap (sau stivă). Este o regiune de memorie mai flexibilă decât stiva, care este folosită pentru stocarea temporară a variabilelor locale și a informațiilor de control în timpul execuției. În limbajul de programare C, funcțiile malloc, calloc, realloc și free sunt folosite pentru a aloca și elibera memorie din heap.

Biblioteca ***stdlib.h*** furnizează funcții pentru manipularea memoriei și alte operații de bază. Iată câteva funcții importante și directii definite în această bibliotecă**.**

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

Functii uzuale de alocare dinamica a memoriei in C(si C++)

malloc()

realloc()

calloc () free()

malloc(): Această funcție (memory allocation) este utilizată pentru a aloca un bloc de memorie de o dimensiune specificată. Sintaxa este:

```
void *malloc(size_t size);
```

Funcția malloc primește un parametru de tip size_t care reprezintă dimensiunea în bytes a blocului de memorie pe care doriți să îl alocați. Funcția întoarce un pointer de tip void * către începutul blocului de memorie alocat.

Exemplu de secventa in C++

```
int* array = (int*)malloc(5 * sizeof(int));
```

Această instrucțiune alocă spațiu pentru un tablou de 5 elemente de tip int în heap.

In C nu este necesara specificarea tipului pentru a atribui lui p valoarea returnata de malloc(), deoarece un pointer de tip *void este automat convertit la tipul pointerului din stinga atribuirii: **p=malloc(n*sizeof(tip);**

In C++ este obligatorie specificarea explicita de tip atunci cand se atribuie un pointer de tip *void altui tip de pointer :

tip *p=(tip*)malloc(n*sizeof(tip);

Zona de memorie alocata nu este initializata *! θ* Pentru evitarea erorilor se testeaza daca exista memorie disponibila.

Functia free():este functia opusa functiei malloc() si are ca efect eliberarea memoriei alocate dinamic anterior.

Alocarea si de-alocarea unei zone de memorie in C++ pentru 50 de nr. intregi

```
int *p; p= (int *)
malloc(50*sizeof(int));
 … free(*p);
```

Functia realloc(): Această funcție (reallocate) este utilizată pentru a modifica dimensiunea unui bloc de memorie deja alocat. Sintaxa este:

```
void* realloc(void* ptr, size_t new_size);
```

Daca realocarea nu este posibila din lipsa de memorie libera atunci realloc() returneaza 'NULL'. Re-alocarea dinamica a unei zone de memorie in C++

```
int *p;
//se aloca dinamic memorie pentru 50 nr intregi
p= (int *)malloc(50*sizeof(int)); …
//se realoca memoria dinamic pentru un sir de 100 nr intregi //utilizand acelasi pointer
p= (int *)realloc(p, 100 * sizeof(int));
```

Funcția primește un pointer către blocul de memorie pe care doriți să îl redimensionați (ptr) și dimensiunea în bytes la care doriți să redimensionați blocul (size).

Functia calloc():(contiguous allocation) este similară cu malloc, dar inițializează memoria alocată cu zero. Sintaxa este identică cu cea a malloc:

```
void* calloc(size_t num_elements, size_t element_size);
```

Problema 1 :

Se aloca un vector cu N valori, Se cere un numar X de la tastatura si se va afisa valorile din vector ce se afla in intervalul [X-5, X+5].

```c
#include <stdio.h>

int main()
{
    float *a, x;
    int n, i;
    /* Citirea numarului de elemente */
    printf("Cate numere? ");
    scanf("%d", &n);

    /* Alocarea zonei de memorie pentru vector */
    a = (float *) malloc(n * sizeof(float));

    if (!a) {
        printf("Nu pot aloca memorie.\n");
    }

    /* Citirea vectorului */
    for (i = 0; i < n; i++) {
        printf("a[%d]: ", i);
        scanf("%f", &a[i]);
        }

    /* Citim numarului X */
    printf("x: ");
    scanf("%f", &x);

    /* Afisarea valorilor din interval */
    printf("Numerele din intervalul (%.2f, %.2f) sunt: ", x - 5, x + 5);
    for (i = 0; i < n; i++)
        if (fabs(x - a[i]) < 5)
            printf("%.2f ", a[i]);
    printf("\n");

    /*Eliberarea memoriei */
    free(a);
    return 0;

}
```

Problema 2 :

Folosind alocarea dinamica, sa se calculeze produsul a doua matrice.

```
Matrice alocare_matrice(int *aloc)  //functie care returneaza o structura de tip matrice
{
     Matrice A;
     cout<<"Introduceti matricea A"<<endl;
     cout<<"Numar de linii: ";
     cin>>A.n;
     cout<<"Nr de coloane: ";
     cin>>A.m;

     A.elem=(int**)calloc(A.n,sizeof(int*)); //alocam nr de linii

     if(!A.elem)
     {
       cout<<"eroare la alocarea zonei pointerilor de linie";
       aloc=0;
     }
     else
     {
         for(i=0;i<A.n;i++) //alocam m elemente pentru fiecare linie
         {
           A.elem[i]=(int*)calloc(A.m,sizeof(int));
           if(!A.elem[i])
           {
              cout<<"eroare la alocarea zonei pointerilor de coloana";
              aloc=0;
           }
         }
     }


     if(aloc)
     {
         for(i=0;i<A.n;i++)
         {
             cout<<"Dati linia "<<i<<" cu elem despartite prin spatiu: "<<endl;
             for(j=0;j<A.m;j++)
               cin>>A.elem[i][j];
         }
       return A;
     }
     else
       cout<< "Alocare Esuata"<<endl;

}
```

```cpp
int main() {
    Matrice A, B, C;
    int aloc1 = 1, aloc2 = 1, aloc3 = 1;
    // Alocare memorie pentru matricele A si B
    A = alocare_matrice(&aloc1);
    B = alocare_matrice(&aloc2);
    // Verificare alocare reusita si compatibilitate pentru inmultirea matricilor
    if (aloc1 == 1 && aloc2 == 1 && (A.m - B.n) == 0) {
        // Alocare memorie pentru matricea rezultat C
        C.n = A.n;
        C.m = B.m;
        C.elem = (int**)calloc(C.n, sizeof(int*));
        if (!C.elem) {
            cout << "Eroare la alocarea zonei pointerilor de linie" << endl;
            aloc3 = 0;
        } else {
            for (int i = 0; i < C.n; i++) {
                C.elem[i] = (int*)calloc(C.m, sizeof(int));
                if (!C.elem[i]) {
                    cout << "Eroare la alocarea zonei pointerilor de coloana" << endl;
                    aloc3 = 0;
                }
            }
        }
        // Calculul produsului matricial
        if (aloc3) {
            for (int i = 0; i < C.n; i++) {
                for (int j = 0; j < C.m; j++) {
                    C.elem[i][j] = 0;
                    for (int k = 0; k < A.m; k++) {
                        C.elem[i][j] += A.elem[i][k] * B.elem[k][j];
                    }
                }
            }

            // Afisarea matricei rezultat
            for (int i = 0; i < C.n; i++) {
                cout << endl;
                for (int j = 0; j < C.m; j++) {
                    cout << C.elem[i][j] << " ";
                }
            }
            // Eliberare memorie
            for (int i = 0; i < A.n; i++) {
                free(A.elem[i]);
            }
            free(A.elem);

            for (int i = 0; i < B.n; i++) {
                free(B.elem[i]);
            }
            free(B.elem);

            for (int i = 0; i < C.n; i++) {
                free(C.elem[i]);
            }
            free(C.elem);
        }
    } else {
        cout << "Alocare nereusita sau numarul de coloane nu coincide cu numarul de linii" << endl;
    }

    return 0;
}
```

Problema 3. Static vs non-static

```c
#include "stdio.h"

// Functie1 cu variabila statica
int functie1()
{
    static int x = 0; // Variabila statica, retine valoarea intre apelurile functiei
    x += 10;
    return x;
}

// Functie2 cu variabila non-statica
int functie2()
{
    int x = 0; // Variabila non-statica, este reinitializata la fiecare apel
    x += 10;
    return x;
}

// Functia principala
int main()
{
    int n = 10; // Numarul de apeluri pentru fiecare functie

    // Afisare rezultate pentru functia cu variabila statica
    printf("Functie cu variabila static: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", functie1());
    }

    // Afisare rezultate pentru functia cu variabila non-statica
    printf("\nFunctie cu variabila non-static: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", functie2());
    }

    return 0;
}
```