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.

În C nu este necesară specificarea tipului pentru a atribui lui p valoarea returnată de malloc(), deoarece un pointer de tip *void este automat convertit la tipul pointerului din stânga atribuirii:

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

În C++ este obligatorie specificarea explicită a tipului atunci când se atribuie un pointer de **tip *void altui tip de pointer:**

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

Zona de memorie alocată nu este inițializată! Pentru evitarea erorilor, se testează dacă există memorie disponibilă.

Funcția free(): Este funcția opusă funcției malloc() și are ca efect eliberarea memoriei alocate dinamic anterior.

Alocarea și dealocarea unei zone de memorie în C++ pentru 50 de numere întregi:ie disponibila.

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

// ... cod care utilizează vectorul ...

free(p);
```

Funcția 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);
```

Dacă realocarea nu este posibilă din lipsă de memorie liberă, atunci realloc() returnează NULL.

Re-alocarea dinamică a unei zone de memorie în C++:

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).

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

```
void* calloc(size_t num, size_t size);
```

1. Problema 1: Se alocă un vector cu N valori. Se cere un număr X de la tastatură și se vor afișa valorile din vector ce se află în intervalul [X-5, X+5].

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *vector;      // pointer pentru vectorul alocat dinamic
    int N, X;
    // Citirea dimensiunii vectorului
    printf("Introduceți numărul de elemente (N): ");
    scanf("%d", &N);
    // Alocare dinamică pentru N elemente de tip int
    vector = (int*)malloc(N * sizeof(int));
    // Verificare dacă alocarea a fost realizată cu succes
    if (vector == NULL) {
        printf("Eroare la alocarea memoriei!\n");
        return 1;
    }
    // Citirea valorilor în vector
    printf("Introduceți cele %d valori:\n", N);
    for (int i = 0; i < N; i++) {
        printf("vector[%d] = ", i);
        scanf("%d", &vector[i]);
    }
    // Citirea numărului X
    printf("Introduceți numărul X: ");
    scanf("%d", &X);
    // Afișarea valorilor din vector care sunt în intervalul [X - 5, X + 5]
    printf("Valorile din intervalul [%d, %d] sunt:\n", X - 5, X + 5);
    for (int i = 0; i < N; i++) {
        if (vector[i] >= X - 5 && vector[i] <= X + 5) {
            printf("%d ", vector[i]);
        }
    }
    // Eliberarea memoriei alocate
    free(vector);

    return 0;
}
```

2. Scrie un program în limbajul C care citește de la tastatură un număr întreg N, alocă dinamic un vector de N numere întregi și citește elementele acestuia. Programul trebuie să creeze un al doilea vector, tot alocat dinamic, care să conțină doar valorile pare din primul vector. La final, se vor afișa aceste valori pare și se va elibera memoria alocată.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *v, *v_pare;
    int N, i, contor_pare = 0;
    printf("Introduceți numărul de elemente: ");
    scanf("%d", &N);
    // Alocăm dinamic vectorul principal
    v = (int*)malloc(N * sizeof(int));
    if (v == NULL) {
        printf("Eroare la alocarea memoriei!\n");
        return 1;
    }
    printf("Introduceți cele %d valori:\n", N);
    for (i = 0; i < N; i++) {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
    }
    // Alocăm vectorul pentru valori pare (dimensiunea maximă posibilă e N)
    v_pare = (int*)malloc(N * sizeof(int));
    if (v_pare == NULL) {
        printf("Eroare la alocarea memoriei pentru vectorul de valori pare!\n");
        free(v);
        return 1;}
    // Selectăm valorile pare
    for (i = 0; i < N; i++) {
        if (v[i] % 2 == 0) {
            v_pare[contor_pare] = v[i];
            contor_pare++;
        }}
    printf("Valorile pare sunt: ");
    for (i = 0; i < contor_pare; i++) {
        printf("%d ", v_pare[i]);}
    printf("\n");
    // Eliberăm memoria
    free(v);
    free(v_pare);
    return 0;}
```

3. Scrie un program în limbajul C care, folosind alocarea dinamică a memoriei, citește două matrice de dimensiuni compatibile de la tastatură și calculează produsul lor. Programul trebuie să aloce dinamic toate cele trei matrice: A, B și rezultatul C. La final, afișează rezultatul și eliberează memoria alocată.

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

int main() {
    int **A, **B, **C;
    int i, j, k;
    int liniiA, coloaneA, liniiB, coloaneB;

    // Citim dimensiunile matricelor
    printf("Introduceți dimensiunile matricei A (linii coloane): ");
    scanf("%d %d", &liniiA, &coloaneA);

    printf("Introduceți dimensiunile matricei B (linii coloane): ");
    scanf("%d %d", &liniiB, &coloaneB);

    // Verificăm dacă se pot înmulți
    if (coloaneA != liniiB) {
        printf("Matricele nu pot fi înmulțite!\n");
        return 1;
    }

    // Alocare matrice A
    A = (int**)malloc(liniiA * sizeof(int*));
    for (i = 0; i < liniiA; i++) {
        A[i] = (int*)malloc(coloaneA * sizeof(int));
    }

    // Alocare matrice B
    B = (int**)malloc(liniiB * sizeof(int*));
    for (i = 0; i < liniiB; i++) {
        B[i] = (int*)malloc(coloaneB * sizeof(int));
    }

    // Alocare matrice rezultat C
    C = (int**)malloc(liniiA * sizeof(int*));
    for (i = 0; i < liniiA; i++) {
        C[i] = (int*)malloc(coloaneB * sizeof(int));
    }

    // Citire matrice A
    printf("Introduceți elementele matricei A:\n");
    for (i = 0; i < liniiA; i++) {
        for (j = 0; j < coloaneA; j++) {
            printf("A[%d][%d] = ", i, j);
            scanf("%d", &A[i][j]);
```

```
        }
    }

    // Citire matrice B
    printf("Introduceți elementele matricei B:\n");
    for (i = 0; i < liniiB; i++) {
        for (j = 0; j < coloaneB; j++) {
            printf("B[%d][%d] = ", i, j);
            scanf("%d", &B[i][j]);
        }
    }

    // Calculul produsului matricelor: C = A * B
    for (i = 0; i < liniiA; i++) {
        for (j = 0; j < coloaneB; j++) {
            C[i][j] = 0;
            for (k = 0; k < coloaneA; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    // Afișare matrice C
    printf("Rezultatul înmulțirii A * B este:\n");
    for (i = 0; i < liniiA; i++) {
        for (j = 0; j < coloaneB; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }

    // Eliberare memorie
    for (i = 0; i < liniiA; i++) free(A[i]);
    free(A);

    for (i = 0; i < liniiB; i++) free(B[i]);
    free(B);

    for (i = 0; i < liniiA; i++) free(C[i]);
    free(C);

    return 0;
}
```

4. Scrie un program care citește N numere întregi de la tastatură, stocate într-un vector alocat dinamic. Programul calculează media aritmetică a elementelor, apoi creează un nou vector (tot dinamic) care conține doar valorile mai mari sau egale cu media. Afișează acest nou vector.
5. Creează un program care alocă dinamic un vector de N numere întregi, citește valorile, apoi creează un al doilea vector (tot alocat dinamic) care conține aceleași elemente în ordine inversă. Afișează ambii vectori.
6. Scrie un program care citește două șiruri de caractere de la tastatură (alocate dinamic) și creează un al treilea șir, tot alocat dinamic, care conține concatenarea lor. Afișează rezultatul și eliberează toată memoria.
7. Scrie un program care, folosind alocare dinamică, generează o **matrice identitate** de dimensiune N x N, unde N este citit de la tastatură. Afișează matricea.