ЛАБОРАТОРНА РОБОТА №4

Перша частина (базове завдання)

ДИНАМІЧНІ МАСИВИ

Мета: Навчитися працювати із одновимірними динамічними масивами.

Вхідні дані: Розмір одновимірного динамічного масиву та значення елементів.

Вихідні дані: Обраховане середнє значення елементів масиву, знайдене максимальне та мінімальне значення в масиві.

Виконання лабораторної роботи передбачає опанування використання функцій по роботі із динамічною пам'яттю з метою створення одновимірних динамічних масивів.

Для того, щоб виділити необхідний об'єм динамічної пам'яті для збереження значень елементів одновимірного динамічного масиву, можна скористатися такими функціями:

`calloc()`
`malloc()`

Прототипи таких функцій мають вигляд:

```
void * calloc(size_t  num_Of_Items, size_t  size);
void * malloc(size_t  size);
```

Функція `calloc()` повинна отримувати два параметри — кількість елементів в динамічному масиві (параметр `num_Of_Items`), а також розмір в байтах типу даних масиву (іншими словами - необхідно задати скільки байтів займає значення одного елементу масиву обраного типу даних в пам'яті комп'ютера). Для того щоб визначити розмір в байтах обраного типу даних потрібно скористатися операцією `sizeof`.

Особливість динамічних масивів полягає в тому, що розмір масиву задається на етапі виконання програми, а не на етапі її написання як це відбувається при роботі із статичними масивами, для яких розмір задається у вигляді константи.

Функція `malloc()` повинна отримувати в якості параметру кількість байтів, яку необхідно виділити в динамічній пам'яті для зберігання значень елементів масиву. Ця кількість байтів може бути розрахована як добуток кількості елементів масиву на кількість байтів, яка виділяється для зберігання значення обраного типу даних.

Наприклад, нехай розмір динамічного масиву задається змінною `Size`. Значення змінної зчитується з клавіатури. Тоді наступний фрагмент коду дозволяє виділити пам'ять для двох динамічних одновимірних масивів, що мають імена `A` та `B`, які зберігають значення типу `unsigned int` та `float`, і для виділення пам'яті цим динамічним масивам використовуються функції відповідно `calloc()` та `malloc()`:

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

int main()
{
    unsigned int * A;
    float * B;
    unsigned int Size;

    printf("Enter size of array: ");
    scanf("%u", &Size);

    A = (unsigned int *) calloc (Size, sizeof(unsigned int) );

    B = (float *) malloc (Size * sizeof(float) );

    return 0;
}
```

Враховуючи, що функції `malloc()` та `calloc()` повертають адресу на перший байт пам'яті виділеної області, тому для вичерпного запису цієї операції в наведеному прикладі також присутня операція приведення типу даних.

При виділенні пам'яті для динамічного масиву `A`, повернене функцією `calloc()` значення вказівника приводиться до типу даних **`unsigned int *`**.

При виділенні пам'яті для динамічного масиву `B`, повернене функцією `malloc()` значення вказівника приводиться до типу даних **`float *`**.

Якщо функції `malloc()` та `calloc()` не змогли виділити пам'ять, вони повертають значення **`NULL`**.

Таким чином, перед тим як продовжувати роботу із динамічним масивом, потрібно перевірити, чи була для нього виділена пам'ять. Якщо пам'ять не була виділена, то в цьому випадку можливо завершити програму за допомогою функції `exit()`.

Для того, щоб використовувати функції `exit()`, `malloc()` та `calloc()` необхідно підключити заголовочний файл `<stdlib.h>`

Після додавання перевірки на предмет виділення пам'яті в код попереднього прикладу, отримаємо:

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

int main()
{
    unsigned int * A;
    float * B;
    unsigned int Size;

    printf("Enter size of array: ");
    scanf("%u", &Size);

    A = (unsigned int *) calloc (Size, sizeof(unsigned int) );

    B = (float *) malloc (Size * sizeof(float) );


    if( A == NULL  ||  B == NULL ){
        printf("Memory has not been allocated");
        exit(0);
    }

    return 0;
}
```

Після того, як відбулося успішне виділення пам'яті для динамічних масивів, робота із ними може виконуватися аналогічно як із статичними масивами.

Наприклад, для того щоб задати значення для елементів масиву A з клавіатури, може бути написана окрема функція, що буде мати ім'я fill_int_array(), і яка може мати наступний прототип:

```
void fill_int_array(unsigned int * uiptr, unsigned int size_of_array);
```

Імена параметрів в прототипах ігноруються компілятором, але для того, щоб дати більше інформації про призначення відповідних параметрів, вони були вказані в прототипі, при цьому імена параметрів були обрані такими, щоб дати користувачу зрозуміти призначення відповідних параметрів.

Інша функція, ім'я якої fill_float_array(), призначена для присвоєння елементам масиву B відповідних значень. Прототип функції має вигляд:

```
void fill_float_array( float * fptr, unsigned int size_of_array );
```

Також необхідно написати дві функції, які будуть виводити на екран значення елементів масивів A та B, які зберігають значення відповідно unsigned int та float. Враховуючи, що дані функції не передбачають зміну значень елементів масивів, а лише виконують виведення їх значень на екран, то можна використати службове слово **const** щоб унеможливити модифікацію значень елементів масивів у відповідних функціях. Прототипи таких функцій будуть наступними:

```
void print_int_array(const unsigned int * uiptr,
                     unsigned int size_of_array );

void print_float_array(const float * fptr,
                       unsigned int size_of_array );
```

Також, необхідно написати функції, які б виконували певну обробку масивів. Наприклад, нехай перша функція знаходить суму елементів в масиві A (елементи масиву мають тип даних unsigned int) , а друга функція знаходить добуток елементів масиву B (елементи масиву мають тип даних float). Оскільки передбачається, що функції не будуть змінювати значення елементів масивів, тому також можна використати службове слово **const,** щоб відповідні функції сприймали масив як константний, тобто такий, який не можна модифікувати. Кожна окрема функція буде отримувати вказівник на перший елемент масиву та розмір масиву, а повертати буде відповідно суму і добуток елементів масиву. Прототипи таких функцій можуть мати наступний вигляд:

```
unsigned int sum (const  unsigned int * uiptr,
                  unsigned int size_of_array );

float product(const float * fptr,
              unsigned int size_of_array );
```

Функція `sum()` буде знаходити суму елементів масиву `A`, а враховуючи, що елементи мають тип даних `unsigned int`, тому і результат, який повертає функція `sum()` буде мати тип даних `unsigned int`.

Аналогічно, функція `product()` буде знаходити добуток елементів масиву `B`, а враховуючи, що елементи масиву мають тип даних `float`, тому і результат, який буде повертатися функцією повинен мати відповідний тип даних. Але враховуючи, що добуток може бути досить значним по величині, то щоб коректно зберегти результат і повернути його в місце виклику функції `product()`, то замість типу даних `float` (тип даних результату, що повертає функція `product()`), можна було б обрати тип даних `double` для забезпечення кращої точності результату обрахунку. Тоді прототип функції `product()` буде мати вигляд:

```
double product(const float * fptr,
               unsigned int size_of_array );
```

Після того, як всі дії над масивами було виконано, необхідно звільнити пам’ять, яка раніше була виділена для масивів `A` та `B`. Це можна зробити наступним чином:

```
free(A);
free(B);
```

Текст програми, в якому виконуються обумовлені дії, може мати наступний вигляд:

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

//  Прототипи
void fill_int_array( unsigned int * uiptr, unsigned int size_of_array );
void fill_float_array( float * fptr, unsigned int size_of_array );

void print_int_array( const unsigned int * uiptr, unsigned int size_of_array );
void print_float_array( const float * fptr, unsigned int size_of_array);
```

```
unsigned int sum ( const  unsigned int * uiptr, unsigned int size_of_array );
double product( const float * fptr, unsigned int size_of_array );


// Опис функції main()

int main()
{
    unsigned int * A;
    float * B;

    unsigned int Size;   // Змінна для збереження розміру динамічного масиву

    unsigned int amount; // змінна для збереження суми елементів масиву A
    double mult;         // змінна для збереження добутку елементів масиву B


    printf("Enter size of array: ");
    scanf("%u", &Size);

    A = (unsigned int *) calloc (Size, sizeof(unsigned int) );

    B = (float *) malloc (Size * sizeof(float) );


    if( A == NULL  ||  B == NULL ){
        printf("Memory has not been allocated");
        exit(0);
    }

    fill_int_array( A, Size );
    fill_float_array( B, Size );

    print_int_array( A, Size );
    print_float_array( B, Size );


    amount = sum( A, Size );
    mult = product( B, Size );

    printf("\n\nSum = %u", amount);
    printf("\nProduct = %.3lf", mult);

    free(A);
    free(B);

    return 0;
}
```

```
//-----------------  Опис Функцій ------------------------
//-------------------------------------------------------

void fill_int_array( unsigned int * A, unsigned int Size )
{
     unsigned int j;
     unsigned int temp;

     printf("\n\nEnter values of UNSIGNED INT elements of array.\n");

     for( j = 0;  j <= Size-1;  j++ ){
         printf("A[%u]= ", j);
         scanf("%u", &temp);
         A[j] = temp;
     }
}


//-------------------------------------------------------

void fill_float_array( float * B, unsigned int Size )
{
     unsigned int j;
     float temp;

     printf("\n\nEnter values of FLOAT elements of array.\n");

     for( j = 0;  j <= Size-1;  j++ ){
         printf("B[%u]= ", j);
         scanf("%f", &temp);
         B[j] = temp;
     }
}


//-------------------------------------------------------------

void print_int_array( const unsigned int * A, unsigned int Size )
{
     unsigned int j;

     printf("\n\nArray of UNSIGNED INT values:\n");

     for( j = 0;  j < Size;  j++ )
        printf("%5d", A[j] );

}

//-------------------------------------------------------------
```

```
void print_float_array( const float * B, unsigned int Size )
{
     unsigned int j;

     printf("\n\nArray of FLOAT values:\n");

     for( j = 0;  j < Size;  j++ )
        printf("%7.2f", B[j] );

}


//---------------------------------------------------------------


unsigned int sum (const  unsigned int * A, unsigned int Size )
{
     unsigned int j, S;


     S = 0;         // змінна, яка використовується для обрахунку значення
                    // суми елементів масиву

     for( j = 0;  j < Size;  j++ )
        S += A[j];

     return S;
}


//---------------------------------------------------------------


double product(const float * B, unsigned int Size )
{
      unsigned int j;
      double M;

      M = 1.0;       //змінна, яка використовується для обрахунку значення
                     // добутку елементів масиву


      for( j = 0;  j < Size;  j++ )
        M *= B[j];

      return M;
}

//---------------------------------------------------------------
```

Завдання на лабораторну роботу

1. Запустити програму, перевірити правильність виконання обрахунків.

2. Написати функцію, яка підраховує середнє значення елементів масиву A, що містить значення типу `unsigned int`. Прототип такої функції може мати вигляд:

```
double mean_value(const unsigned int * A, unsigned int Size );
```

Функція повинна знаходити середнє значення в масиві A, повертати результат в функцію `main()`, де знайдене середнє значення повинно присвоюватися відповідній змінній на ім'я `average_value`, яку треба оголосити в програмі в функції `main()`. Така змінна може бути оголошена наступним чином:

```
double  average_value;
```

Змінна `average_value` отримує значення, яке повертається функцією `mean_value()`, і потім значення змінної `average_value` повинно бути відображене на екрані.

3. Написати функцію, яка знаходить мінімальне значення в масиві `A`. Прототип такої функції може мати вигляд:

```
unsigned int find_min(const unsigned int * A, unsigned int Size );
```

Функція повинна знаходити мінімальне значення в масиві A, повертати результат в функцію `main()`, де він повинен присвоюватися відповідній змінній на ім'я `min_value`, яку треба оголосити в програмі в функції `main()`. Така змінна може бути оголошена наступним чином:

```
unsigned int min_value;
```

Змінна `min_value`, отримує значення, яке повертається функцією `find_min()`. Після цього значення змінної `min_value` повинно бути відображене на екрані.

4. Написати функцію, яка знаходить максимальне значення в масиві `B`. Прототип такої функції може мати вигляд:

```
float find_max(const float * B, unsigned int Size );
```

Функція повинна знаходити максимальне значення в масиві `B`, повертати результат в функцію `main()`, де він повинен присвоюватися відповідній змінній на ім'я `max_value`, яку треба оголосити в програмі в функції `main()`. Така змінна може бути оголошена наступним чином:

```
float max_value;
```

Змінна `max_value`, отримує значення, яке повертається функцією `find_max()`. Після цього значення змінної `max_value`, повинно бути відображене на екрані.

5. Внести зміни у функцію `fill_float_array()`, щоб значення елементів масиву вводились не з клавіатури, а визначались за допомогою наступного виразу:

```
B[j] = 5.0 * j + 5.0;
```

6. Внести зміни у функцію `fill_int_array()`, щоб значення елементів масиву вводились не з клавіатури, а визначались за допомогою наступного виразу:

```
A[j] = Size - (j + 1);
```

де `Size` — розмір масиву.

7. Підготувати звіт.