

Лабораторна робота №10. Документація проекту

1. Вимоги

1.1 Розробник

Макаренко Владислав Олександрович;

студент групи КІТ-120а;

16 - грудня - 2020

1.2 Загальне завдання

Розробити повноцінний звіт для лабораторної роботи “Функції”

1.3 Індивідуальне завдання

За допомогою функцій отримати корінь заданого числа.

За допомогою функцій перемножити матрицю саму на себе.

За допомогою варіативної функції визначити, скільки серед заданої послідовності чисел таких пар, у котрих перше число менше наступного.

2. Опис програми №1

2.1 Функціональне призначення

Програма добуває квадратний корінь з числа за допомогою функції `Square_root()`. Результат зберігається у змінній `result`. Демонстрація результату передбачає покрокове виконання програми.

2.2 Опис логічної структури програми

Для визначення квадратного кореня з числа викликаємо функцію `Square_root()`, яка приймає параметр: число `number` з якого буде добуватись квадратний корінь. Функція збільшує значення параметру `result` на `0.0001` поки значення `result * result` не буде дорівнювати заданому `number`.

Головна функція `main()`. Приймає задане нами число. Задає випадкове число від 1 до 50. Двічі викликає функцію `Square_root`. Схема алгоритму функції подана на рис. 1.

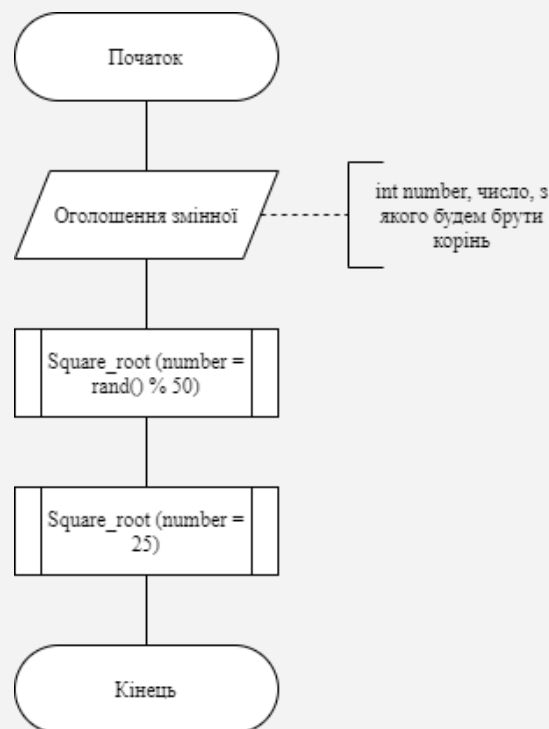


Рисунок 1 — Схема алгоритму функції `main`

Функція, що добуває корінь з числа `Square_root`. Добуває квадратний корінь з числа. Параметри: `number` – число, з якого потрібно добути корінь; `result` – добутий корінь з заданого числа. Функція повертає `result`. Схema алгоритму функції подана на рис. 2.

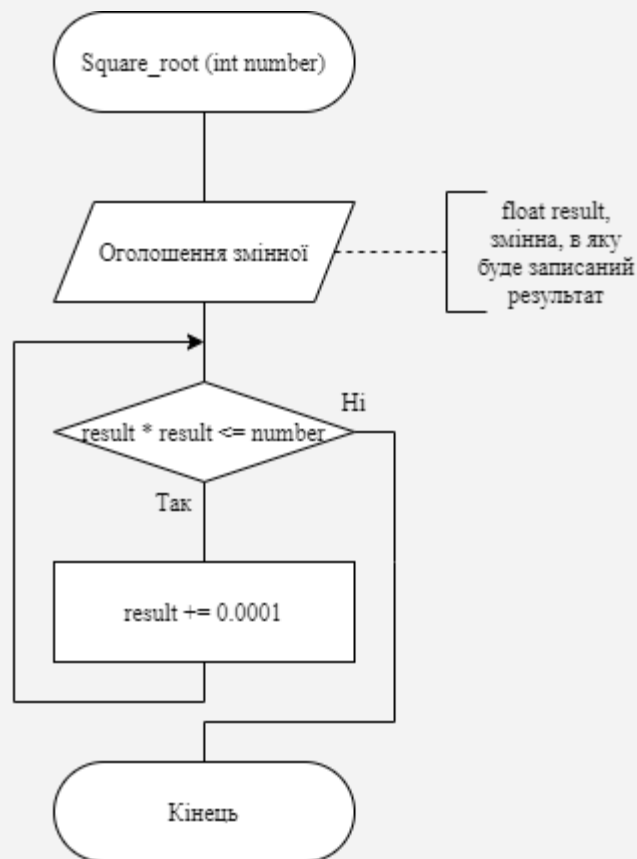


Рисунок 2 — Схема алгоритму функції `Square_rot`

2.3 Важливі фрагменти програми

Підключення заголовочного `stdlib.h` та `time.h` для генерації випадкових чисел.

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

Виклик функції для випадкового числа

```
result1 = Square_root (number = rand() % 50);
```

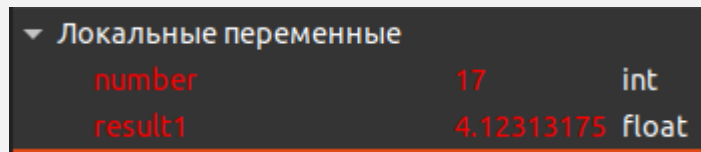
Добування кореню за допомогою цикла

```
while (result * result <= number) {  
    result += 0.0001;  
}
```

2.4 Варіанти використання

Для демонстрації результатів використовується покрокове виконання програми та інші засоби налагодження відладчика `netiver`. Нижче наводиться послідовність дій запуску програми у режимі відладження.

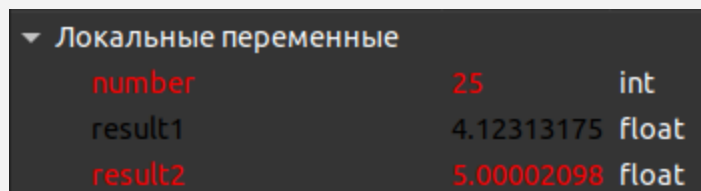
Крок 1 (рис. 3). Виконуємо функцію для випадково згенерованого числа



Локальные переменные		
number	17	int
result1	4.12313175	float

Рисунок 3 — Результат виконання функції для першого кроку

Крок 2 (див. рис. 4). Виконуємо функцію для заданого нами числа



Локальные переменные		
number	25	int
result1	4.12313175	float
result2	5.00002098	float

Рисунок 4 — Результат виконання функції для другого кроку

3. Опис програми №2

3.1 Функціональне призначення

Програма перемножає матрицю саму на себе за допомогою функції `matrix_multiplication()`. Результат зберігається у `result_matrix`. Демонстрація результату передбачає покрокове виконання програми.

3.2 Опис логічної структури програми

Для множення матриці викликаємо функцію `matrix_multiplication()`, яка приймає масив `matrix` який будемо множитись. Функція за допомогою циклів перемножає матрицю саму на себе, за правилом множення матриць.

Головна функція `main()`. Заповнює масив випадковими числами від 1 до 9. Викликає функцію `matrix_multiplication()`. *Схема алгоритму функції* подана на рис. 5.

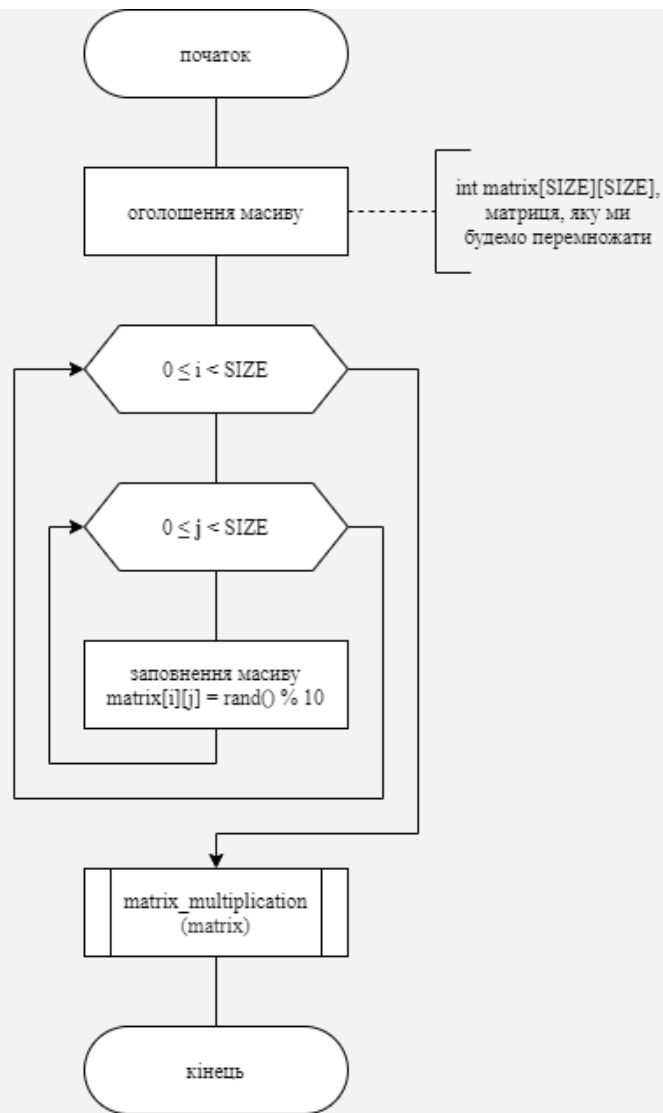


Рисунок 5 — Схема алгоритму функції main

Функція `matrix_multiplication()`. Перемножає матрицю саму на себе. Параметри: `result_matrix`— масив, в який буде записаний результат множення матриць. *Схема алгоритму функції* подана на рис. 6.

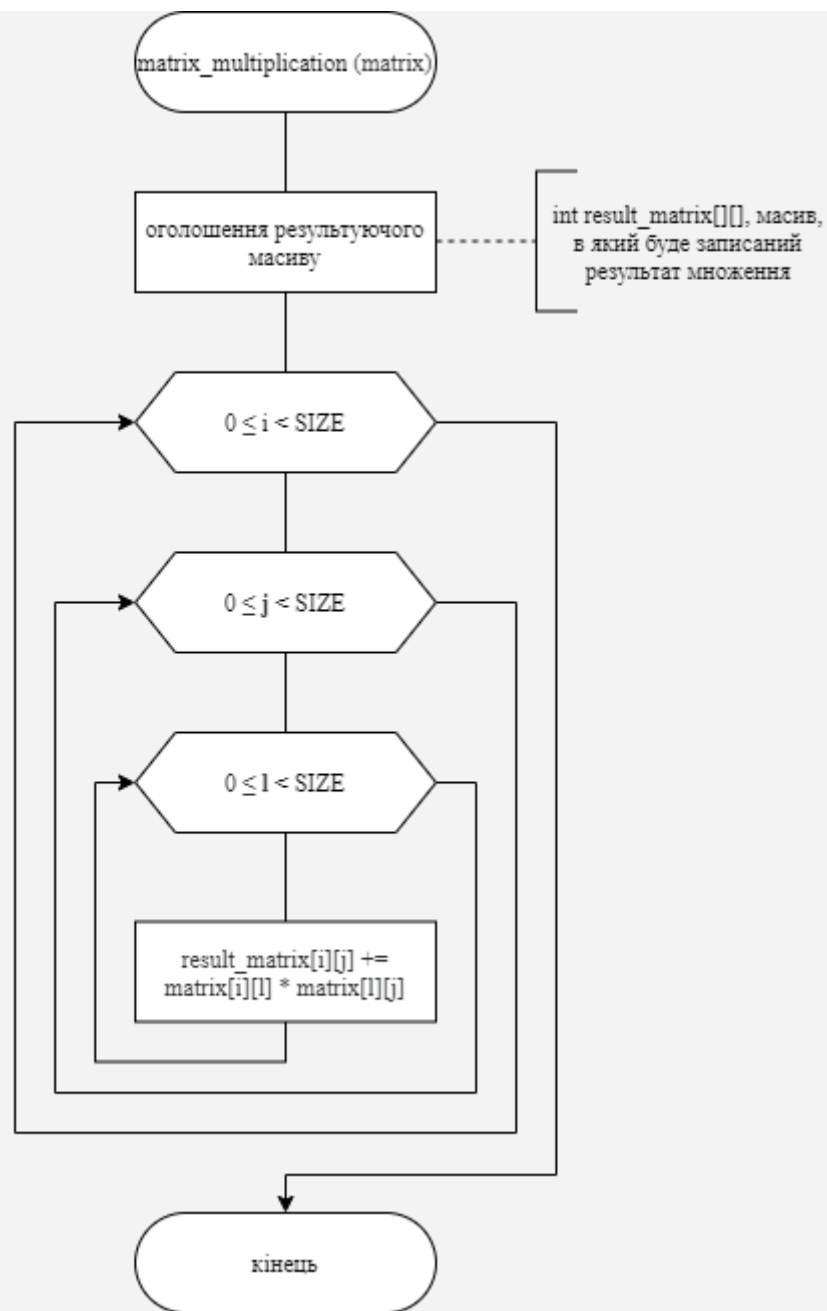


Рисунок 6 — Схема алгоритму функції `matrix_multiplication()`

3.3 Важливі фрагменти програми

Підключення заголовочного `stdlib.h` та `time.h` для генерації випадкових чисел.

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

Виклик функції для множення матриці

```
matrix_multiplication(matrix)
```

Процес множення матриці саму на себе

```
for (int i = 0; i < SIZE; i++){
    for (int j = 0; j < SIZE; j++){
        for (int l = 0; l < SIZE; l++) {
            result_matrix[i][j] += matrix[i][l] * matrix[l][j];
        }
    }
}
```

3.4 Варіанти використання

Для демонстрації результатів використовується покрокове виконання програми та інші засоби налагодження відладчика `netiver`. Нижче наводиться послідовність дій запуску програми у режимі відладження.

Крок 1 (рис. 7). Заповнюємо масив випадковими числами

Переменная	Значение	Тип
Локальные переменные		
matrix	[3]	int [3][3]
0	[3]	int [3]
0	6	int
1	5	int
2	7	int
1	[3]	int [3]
0	4	int
1	5	int
2	3	int
2	[3]	int [3]
0	9	int
1	8	int
2	5	int

Рисунок 7 — Результат виконання функції для першого кроку

Крок 2 (див. рис. 8). Виконуємо функцію для заданого масиву

▼ Локальные переменные		
▼ result_matrix	[3]	int [3][3]
▼ 0	[3]	int [3]
0	119	int
1	111	int
2	92	int
▼ 1	[3]	int [3]
0	71	int
1	69	int
2	58	int
▼ 2	[3]	int [3]
0	131	int
1	125	int
2	112	int
▼ Параметры функции		

Рисунок 8 — Результат виконання функції для другого кроку

4. Опис програми №3

4.1 Функціональне призначення

Програма, що визначає, скільки серед заданої послідовності чисел таких пар,

у котрих перше число менше наступного за допомогою варіативної функції `function()`. Результат зберігається у змінній `result`. Демонстрація результату передбачає покрокове виконання програми.

4.2 Опис логічної структури програми

Для визначення кількості пар чисел викликаємо функцію `function()`, яка приймає параметр: число `number` — кількість чисел в послідовності, та саму послідовність. Функція перевіряє кожен елемент ряду із усіма наступними, якщо

елемент менший за один із наступних елементів то змінна `result` збільшується на 1.

Головна функція `main()`. Викликає `function()`. *Схема алгоритму функції* подана на рис. 9.

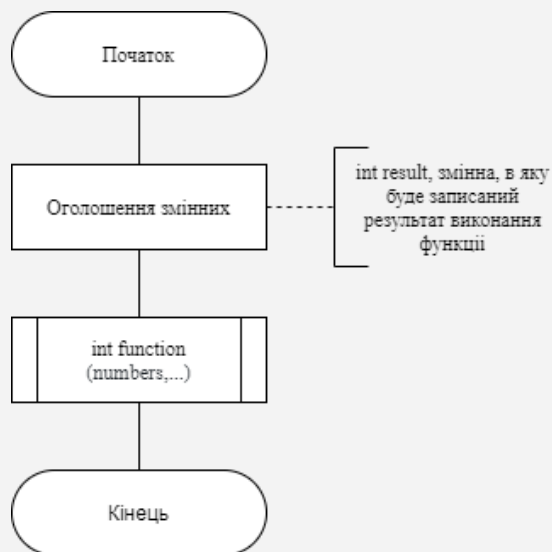


Рисунок 9 — Схема алгоритму функції `main`

Функція `function()`. Функція перевіряє кожен елемент послідовності та знаходить кількість пар, де перше число менше наступного. *Схема алгоритму функції* подана на рис. 10.

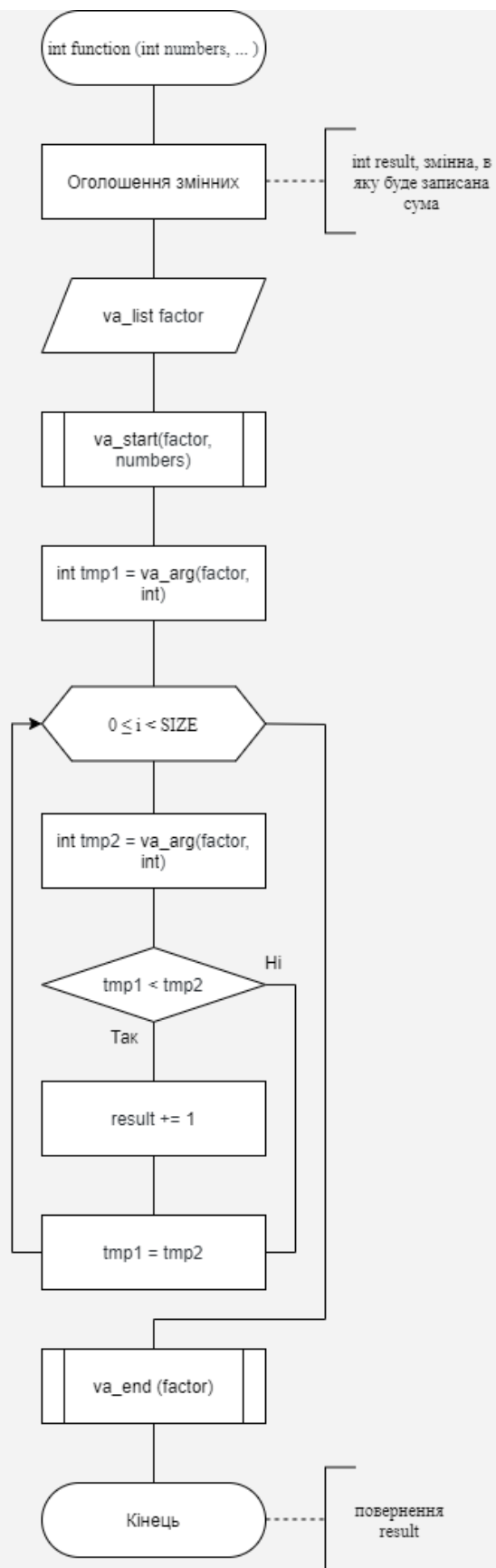


Рисунок 10 — Схема алгоритму функції `function()`

4.3 Важливі фрагменти програми

Підключення заголовочного `<stdarg.h>` для роботи з варіативними функціями.

```
#include <stdarg.h>
```

Виклик варіативної функції

```
int result = function(SIZE, 3, 9, 3, 9, 1, 1, 4)
```

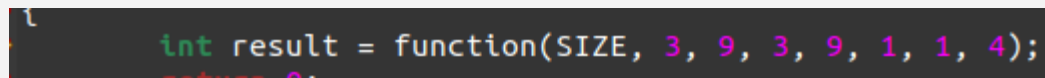
Процес знаходження кількості пар чисел

```
int tmp1 = va_arg(factor, int);
for (int i = 0; i < SIZE; i++){
    int tmp2 = va_arg(factor, int);
    if (tmp1 < tmp2) {
        result += 1;
    }
    tmp1 = tmp2;
}
```

4.4 Варіанти використання

Для демонстрації результатів використовується покрокове виконання програми та інші засоби налагодження відладника `netiver`. Нижче наводиться послідовність дій запуску програми у режимі відладження.

Крок 1 (рис. 11). Вхідні данні функції



```
int result = function(SIZE, 3, 9, 3, 9, 1, 1, 4);
```

Рисунок 11 — Результат виконання функції для першого кроку

Крок 2 (рис. 12). Виконуємо функцію для заданої послідовності

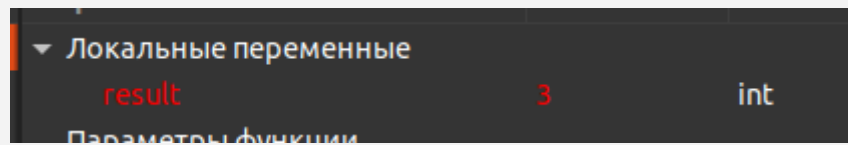


Рисунок 8 — Результат виконання функції для другого кроку

Структура проекту

```
├── lab10
│   ├── Doxyfile
│   ├── Makefile
│   └── doc
│       ├── lab10.md
│       ├── lab10.docx
│       ├── lab10.pdf
│       └── assets
└── src
    ├── main1.c
    ├── main2.c
    └── main3.c
```

Висновки

Ми навчилися документувати проект за допомогою Markdown та в doc форматі, згідно ДСТУ.