

Лабораторна робота №7

Тема: Функції

Виконав:

Стоєнко Сергій Максимович

Група виконавця: КН-922Б

Завдання:

1. Переробити програми, що були розроблені під час виконання лабораторних робіт з тем "Масиви" та "Цикли" таким чином, щоб використовувалися функції для обчислення результату.
 - а. Параметри одного з викликів функції повинні бути згенеровані за допомогою генератора псевдовипадкових чисел *random()*.
2. Продемонструвати встановлення вхідних даних через аргументи додатка (параметри командної строки). Обробити випадок, коли дані не передались - у цьому випадку вони матимуть значення за умовчужанням, обраними розробником.

Основна частина:

- Опис роботи основної функції:

1. Записуємо окремо від функції `main` ще 2 функції: `find_nsd` яка приймає значення 2 чисел та повертає значення НСД та `multiply_matrix`, яка приймає матрицю, яку будемо множити та матрицю, в яку будемо вписувати значення помноженої матриці. Функція нічого не повертає, лише проводить вписує значення помноженої матриці. Більш детальний розбір цих 2 функцій був у попередніх звітах.
 - а. Для згенерованих за допомогою генератора псевдовипадкових чисел *random()* параметрів функції я вибрав функцію `find_nsd` та згенерував рандомно 2 числа в яких можна знайти спільний дільник.

- Перелік вхідних даних:

-first_number – перше число, має бути цілим(int)

-second_number – друге число, має бути цілим(int)

- mas – матриця для множення, масив цілих чисел(int)

- array_size - кількість стовпців та кількість рядків матриці, позитивне дійсне число(int)

- Дослідження результатів роботи програми

-NSD – Найбільший спільний дільник, так як всі оперуючі числа є цілими, так і результуюча змінна є цілою. Тому, її тип – int.

- multiply[] - матриця добутку, так як всі оперуючі числа матриці, що множеться є цілими, так і результативна матриця є цілою. Тому, її тип – int.

- для підтвердження коректності роботи програми find_nsd з рандомними числами, зупинено відладчик на строчці „return 0“

```
(lldb) file dist/main.bin
```

```
Current executable set to '/home/serhii/tnp/programing-Stoienko/lab07/dist/main.bin' (x86_64).
```

```
(lldb) b 24
```

```
Breakpoint 1: where = main.bin`main + 161 at main.c:24:2, address = 0x00000000000001221
```

```
(lldb) run
```

```
(lldb) print first_number
```

```
(int) $4 = 44
```

```
(lldb) print second_number
```

```
(int) $5 = 63
```

```
(lldb) print nsd
```

```
(int) $6 = 1
```

- НСД 44 та 63 дійсно дорівнює 1

```
(lldb) print first_number
```

```
(int) $19 = 54
```

```
(lldb) print second_number
```

- (int) \$20 = 22
- (lldb) print nsd
- (int) \$21 = 2
- НСД 54 та 22 дійсно дорівнює 2
 - (lldb) print first_number
 - (int) \$27 = 33
 - (lldb) print second_number
 - (int) \$28 = 66
 - (lldb) print nsd
 - (int) \$29 = 33
- НСД 33 та 66 дійсно дорівнює 33

Бачимо, що функція найбільшого спільного дільника працює коректно.

2. Через командну строку ми можемо встановити вхідні дані за допомогою функції `main(int argc, char *argv[])`, де завдяки `argc` ми дізнаємось чи записано щось у командну строку, а `argv` буде символьним масивом, де в першому символі буде записано розмір матриці, а всі інші числа будуть самою матрицею, але ці дані будуть у символьному форматі. Тому нам потрібна функція `strtol()`, яка перетворює строку у число(якщо там записане число).
- Перевіримо роботу функції за допомогою функції `printf()`:
- `serhii@serhii-VirtualBox:~/tnp/programing-Stoienko/lab07/dist$./main.bin 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16`
- 90 100 110 120
- 202 228 254 280
- 314 356 398 440
- 426 484 542 600
- `serhii@serhii-VirtualBox:~/tnp/programing-Stoienko/lab07/dist$./main.bin 3 1 2 3 4 5 6 7 8 9`
- 30 36 42
- 66 81 96
- 102 126 150

- serhii@serhii-VirtualBox:~/tnp/programing-Stoienko/lab07/dist\$
./main.bin 3 1 2 3 4 5 6 7 8 9 10 11 12
- 30 36 42
- 66 81 96
- 102 126 150
- serhii@serhii-VirtualBox:~/tnp/programing-Stoienko/lab07/dist\$
./main.bin 4 1 2 3 4 5 6 7 8 9 10 11 12
- Segmentation fault (core dumped)
- serhii@serhii-VirtualBox:~/tnp/programing-Stoienko/lab07/dist\$./main.bin
- 0 0 0
- 0 0 0
- 0 0 0

Як бачимо, результати співпадають з реальними матрицями, тому функція множення матриці працює коректно.

Структура проекту лабораторної роботи:

lab07

|— doc

| |— lab07.docx

|— README.md

|— src

|— main.c

Висновки: при виконанні лабораторної роботи були набуті навички створення функцій на мові C, зокрема: навчились генерувати рандомні числа, перетворювати строку у числа, оптимізувати код, перетворивши його у функцію, встановлювати вхідні дані через аргумент додатка.

Код проекту:

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

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

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

```
int find_nsd(int first_number, int second_number);
```

```
void multiply_matrix(int argc, char *argv[], int *multiply);
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    srand((unsigned int)time(0));
```

```
    int multiply[15][15];
```

```
    multiply_matrix(argc, argv, (int *)multiply);
```

```
    int first_number = random() % 100;
```

```
    int second_number = random() % 100;
```

```
    int nsd = find_nsd(first_number, second_number);
```

```
    // show on screen
```

```
    printf(" first number = %d\n second number = %d\n nsd = %d\n",
```

```
        first_number, second_number, nsd);
```

```
    return 0;
```

```
}
```

```
int find_nsd(int first_number, int second_number)
```

```
{
```

```
    // create first and second number
```

```
    int larger_number;
```

```
    int nsd = 1;
```

```

//we need to check which of the numbers are larger
//and assign new value to variable larger_number
//for knowing, what number is larger for cycle for
if (second_number >= first_number) {
    larger_number = second_number;
} else {
    larger_number = first_number;
}

for (int i = 1; i <= larger_number; i++) {
    // if numbers are both evenly divisible by i, then i
    // is common factor, the last i, that gonna be common
    // factor of the numbers, is largest
    if (first_number % i == 0 && second_number % i == 0) {
        nsd = i;
    }
}

return nsd;
}

void multiply_matrix(int argc, char *argv[], int *multiply)
{
    long int mas[15][15];
    long int find_size;
    long int array_size;
    long int symbol_number = 2;
    char *nul;

    if (argc > 1)
        /** Перевіряємо чи були введені данні у командний рядок. */

```

```

{
    find_size = strtol(argv[1], &nul, argc);

    array_size = find_size;
} else {
    array_size = 3;
}

for (int row = 0; row < array_size; row++)

{
    for (int coll = 0; coll < array_size; coll++) {
        mas[row][coll] = strtol(argv[symbol_number], &nul, 10);

        symbol_number++;
    }
}

```

// Так як треба множити матрицю на саму себе, то це повинна бути
 // квадратична матриця, бо інакше число рядків та стовпців не будуть
 // співпадати, а тоді не можна множити(за правилами множення).

// рядок

```

for (int row = 0; row < array_size; row++) {
    // стовпчик
    for (int coll = 0; coll < array_size; coll++) {
        // даємо значення елементу матриці
        multiply[array_size * row + coll] = 0;

        // скільки разів множимо рядок матриці на стовпчик
        for (int times = 0; times < array_size; times++) {
            // Помножимо рядок матриці на стовпчик матриці та отримуємо

```

```

        // елемент на рядку та стовпчику матриці добутка
        multiply[array_size * row + coll] +=
            mas[row][times] * mas[times][coll];
    }
}

for (int row = 0; row < array_size; row++) {
    int new_line = 0;
    // стовпчик
    for (int coll = 0; coll < array_size; coll++) {
        printf("%d\t", multiply[array_size * row + coll]);
        new_line++;
        if (new_line == array_size) {
            new_line = 0;
            printf("\n");
        }
    }
}
}

```