# Лабораторна робота №7. Функції

**Автор:** Примак Герман Валерійович
**Група:** 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.

**2.**Функції повинні задовольняти основну їх причетність - уникати дублювання коду.

Тому, для демонстрації роботи, ваша програма (функція main()) повинна мати можливість викликати розроблену функцію з різними вхідними даними.

**3.**Слід звернути увагу: параметри одного з викликів функції повинні бути згенеровані за допомогою генератора псевдовипадкових чисел random().

**4.**Слід звернути увагу (#2): продемонструвати встановлення вхідних даних через аргументи додатка (параметри командної строки).

Обробити випадок, коли дані не передались - у цьому випадку вони матимуть значення за умовчуванням, обраними розробником.

# Опис програми

### *Функціональне призначення*

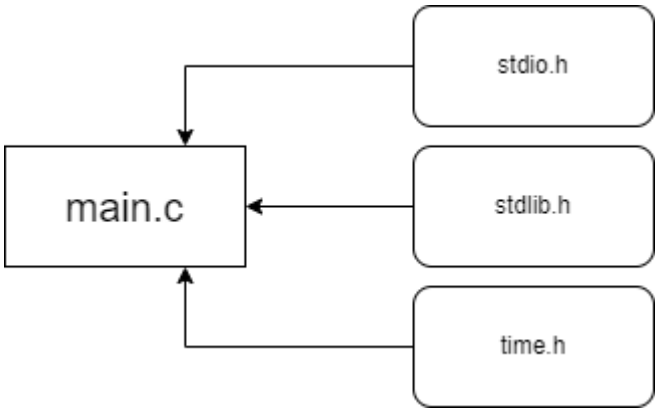
Ця програма виконує дві операції.

- Множить матрицю саму на себе
    1. При запуску програми ви повинні ввести розмір матриці це перший аргумент командного рядка, та значення які зберігаються у матриці.

**Увага!** Після того як ви ввели розмір матриці вам потрібно ввести значення які зберігаються у ній, але якщо ви введете но достатню кількість елементів то матриця не буде перемножена сама на себе.

- Генерує випадкове число та перевіряє просте воно, чи ні.

### *Опис логічної структури*




*(Рис. 1) Графічна структура програми*

***Файл "main.c"***

Головний файл

Це файл, який містить точку входу main,  функції squareMatrix та primenumber .

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

Головна функція.
Містить у собі виклик другорядних функцій squareMatrix та primenumber, які містять код програми для розрахунку множення матриці саму на себе та перевірку на просте чи не просте число.

**Аргументи**

int argc, argv *c[] Аргументі які зберігають значення введені через командний рядок

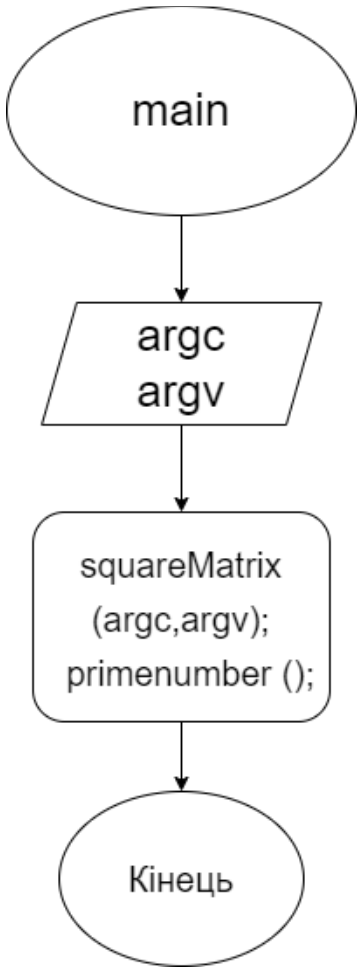
argc - зберігаєє кількість введених значень

argv - зберігае самі значення

**Послідовність дій**

- Присвоїти значення аргументам argc і argv, значення цих аргументів ми передаємо у функцію squareMatrix .
- Виклик функції squareMatrix, у параметрах цієї функції при виклику вказуємо аргументи argc і argv.
- Викликаємо функцію primenumber, у параметрах цієї функції при виклику нічого не ваказуємо.

```
int main (int argc,char *argv[]) {
  squareMatrix (argc,argv);
  primenumber ();
}
```




*(Рис. 2) Схема алгоритму функції main*

## void squareMatrix (int argc,char *argv[])

Ця функція множить матрицю саму на себе.

**Аргументи**

argv - зберігає кількість введених значень у командний рядок, та використовується у перевірці.

args - зберігає значення введені у командний рядок які потім використовуються у множенні матриці.

**Послідовність дій**

- Створення змінних a[10][10], MAT[10][10], b, c, i, j, f, z = 2, t, *N.
    1. *a[10][10] - квадратна матриця, що містить межу 10 рядків і стовпців, але користувач може задати будь-яку квадратну матрицю в цьому діапазоні.*
    2. *MAT[10][10] - квадратна матриця, що містить розрахунок матриці a*a.*
    3. *b і c - змінні що містять у собі кількість рядків і стовпців, користувач задає тільки одне значення і оскільки матриця квадратна b=c=t.*
    4. *t - зберігає значення за умовчанням.*
    5. *і та j - кількість стовпців і рядків матариці, які порівнюються між заданими b та c, та якщо виконується умова вони збільшуються.*
    6. *t - використовується у перетворення значень рядка у число.*
    7. **N - використовується у перетворення значень рядка у число.*
- Перевіряємо чи були введені якісь значення у командний рядок.
- Якщо перевірка була пройдена то перетворюємо перше значення командного рядка у число та присвоюємо змінній t. Після чого присвоюємо змінну t змінним c та b, і запускаємо два цикли у яких записуємо ці значення у матрицю *a*.

```
if (argc > 1)
    {
    t = strtol (argv[1], &N, 10);


    b = t;
```

```
    c = t;
  for (i = 0; i <= b - 1; i++)
  {
     for (j = 0; j <= c - 1; j++)
     {
       a[i][j] = strtol (argv[z], &N, 10);
       z++;
      }
  }

   }
```

- Якщо перевірка не була пройдена то присвоюємо змінну z змінним b та c.

```
else{
     b = z;
     c = z;
     for (i = 0; i <= b - 1; i++)  {
        for (j = 0; j <= c - 1; j++){
           a[i][j] = strtol (argv[z], &N, 10);
           z++;
          }
       }
   }
```

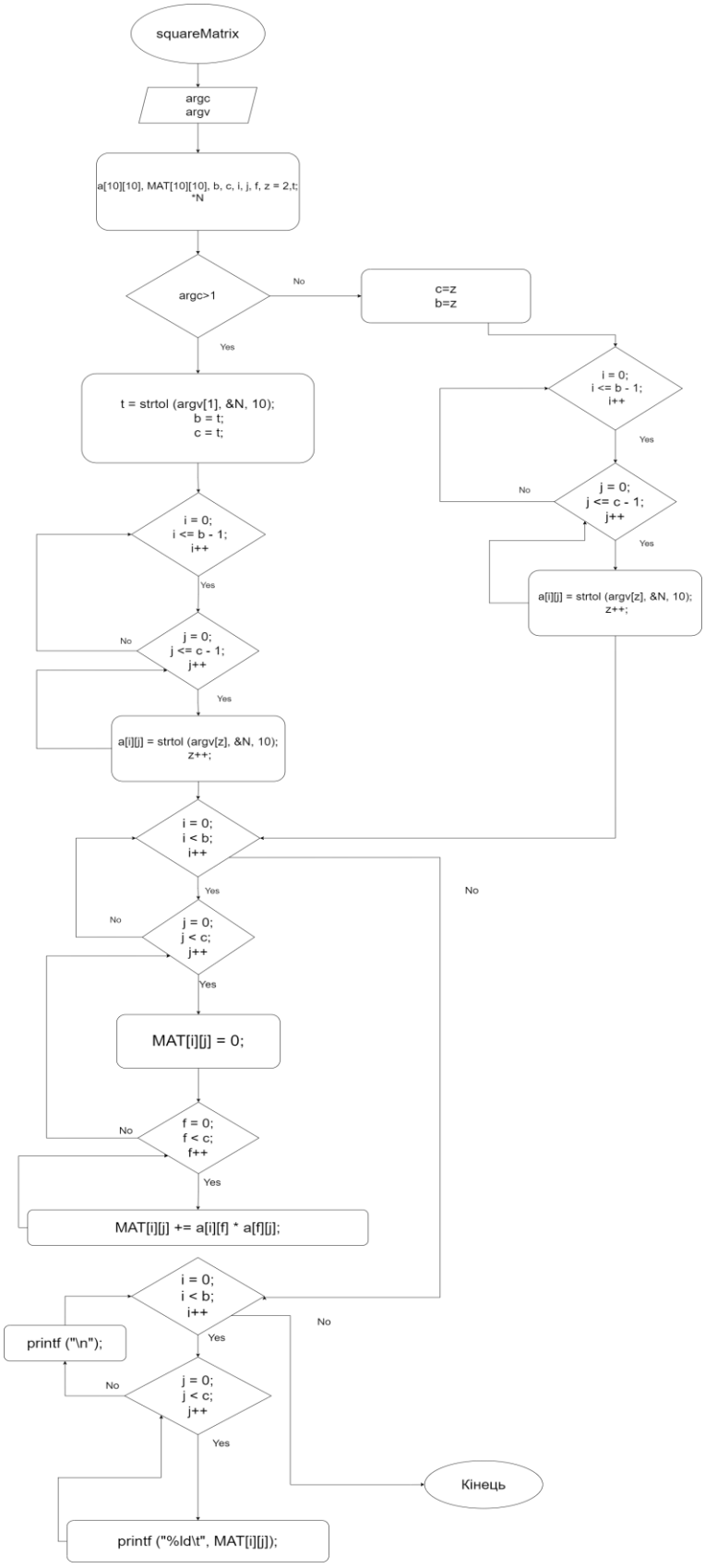
- Тепер для множення матриці самої на себе запускаємо 3 цикли, перші два цикла перебирають значення матриці за адресою, останній цикл множить матрицю саму на себе та присвоює результат множення матриці *MAT* .

```
for (i = 0; i < b; i++)
```

```
{
            for (j = 0; j < c; j++)
        {
            MAT[i][j] = 0;
            for (f = 0; f < c; f++)
            {
              MAT[i][j] += a[i][f] * a[f][j];
             }
        }
}
```

- Після множення матриці саму на себе виводимо результат. Для цього потрібно запустити 2 цикли які перебирають значення функції *MAT*, та потім використовуючи функцію *printf* виводимо значення на екран.

```
for (i = 0; i < b; i++)
  {
            for (j = 0; j < c; j++)
        {
          printf ("%ld\t", MAT[i][j]);
}
            printf ("\n");
        }
```




*(Рис. 3) Схема алгоритму функції squareMatrix*

## void primenumber ()

Ця функція генерую випадкове число та перевіряє просте воно чи ні

**Послідовність дій**

- Створення змінних a, res = 1, i.
    1. *a - зберігає згенероване число.*
    2. *res – використовується при виведені числа на екран*
    3. *i – використовується у перевірці чи число просто чи ні.*
- Потім генеруємо випадкове число використовуючи функцію srand та rand() і присвоюємо це значення змінній *a.*

```
srand ((unsigned int) time (NULL));
 a = (rand () % 49) + 2;
```

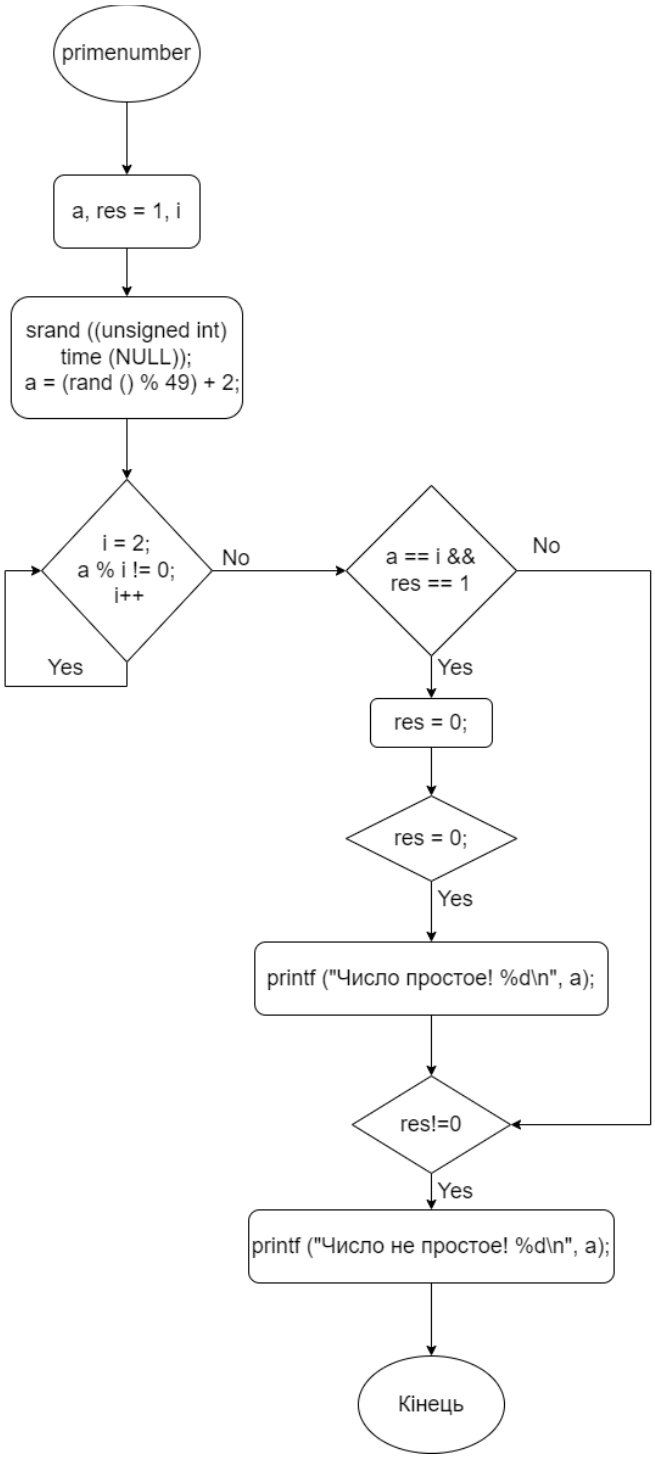
- Запускаємо цикл у якому перевіряємо просте число чи ні.

```
for (i = 2; a % i != 0; i++)
```

- Потім Перевіряємо чи *a == i*, та *res == 1*, якщо умова виконується то задаєм значення *res == 0*, та пишимо що число просте, якщо умова не виконується, то *res == 1* та число не просте.

```
if (a == i && res == 1){
   res = 0;
   if (res == 0) {
      printf ("Число простое! %d\n", a);
   }
}
```

```
if (res != 0){
    {
 printf ("Число не простое! %d\n", a);
    }
```



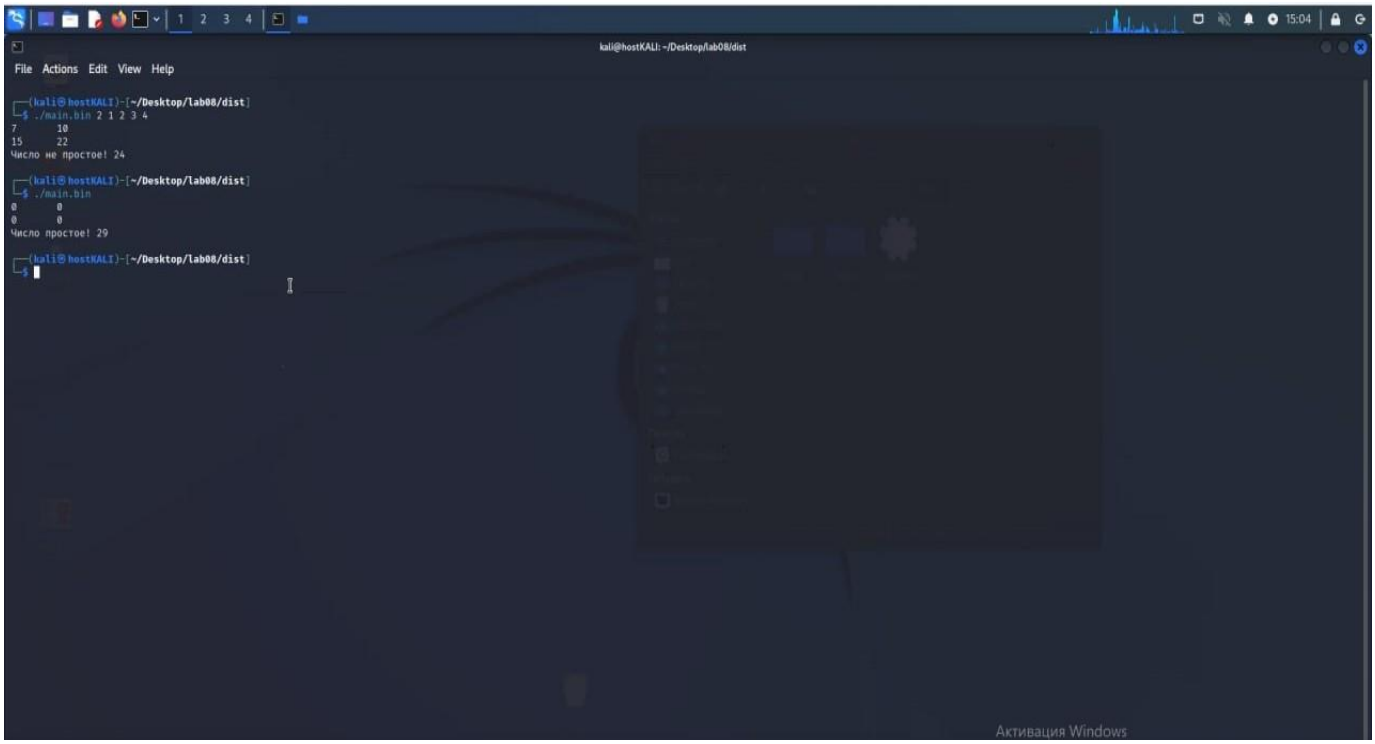

*(Рис. 4) Схема алгоритму функції lab06*

***Структура проекту лабораторної роботи:***

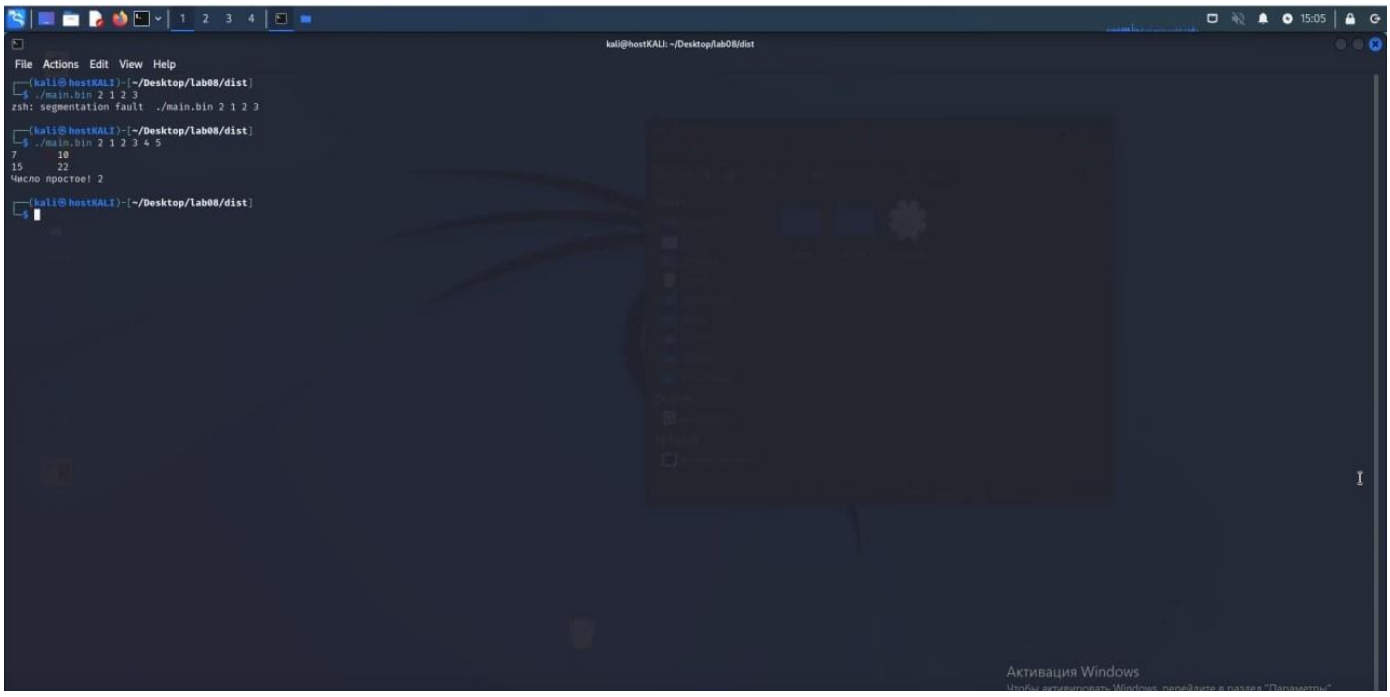
```
├── lab07
├── Makefile
├── README.md
└── src
    └── main.c
```

***Варіанти використання***

- Користуватися цією програму не складно. Для того щоб помножити матрицю саму на себе вам потрібно при запуску двійкового файлу вказати аргументі командного рядка. Перший елемент який ви вказуєте це розмір матриці, всі елементи які будуть вказані  це значення які зберігаються у цій функції. Після цього вам виведе на екран результати множення матриці та випадково згенероване число та підпис просте воно чи ні*(Рис.5)*. Слід зауважити що матриця має бути повністю заповнена, наприклад якщо матриця буде 2 порядку то значення має бути 4, якщо їх буде менше матриця не буде помножена саму на себе, а якщо введених значень буде більше то програма не буде звертати уваги на зайві  *(Рис.6)*




*(Рис. 5)Як правильно користуватися програмою!*




*(Рис. 6)Як не правильно користуватися програмою!*

**Висновки:** У цій роботі було перетворено лабораторні проекти №5 та №6 для використання функцій. Було набуто навичок роботи з функціями, їх декларація, реалізація та виклик. Під час тестування програми були отримані результати функції squareMatrix - це множення матриці саму на себе, і робота функції primenumber - це отримання випадково згенерованого числа та перевірка на те просте воно чи ні .