

Звіт

Лабораторна робота №10

Тема: Markdown.

Розробник: студент Клименко Станіслава Олександрівна, група 120-а.

Перевірів: асистент Челак Віктор Володимирович.

Індивідуальне завдання: Реалізувати функцію, що визначає скільки серед заданої послідовності чисел таких пар, у котрих перше число більше за наступне.

Опис програми 1:

- *Функціональне призначення:* Визначення чи є число досконалим. Якщо сума всіх дільників числа буде дорівнювати самому числу, то число досконале.

- *Опис логічної структури:*

- Функція ``main``. Оголошує два масиви. Перший масив це наш масив з числами, що перевіряють на досконалість, а другий це масив відповідей, що покаже нам досконале число, чи ні. Викликає функцію ``perfect_numbers``.
- Функція ``perfect_numbers``. Оголошуємо значення суми дільників нашого числа. Цикл допомагає нам знайти значення суми дільників, а наступні наші дії заповнюють наш масив відповіді залежно від порівняння числа та суми його дільників.(рис.1)

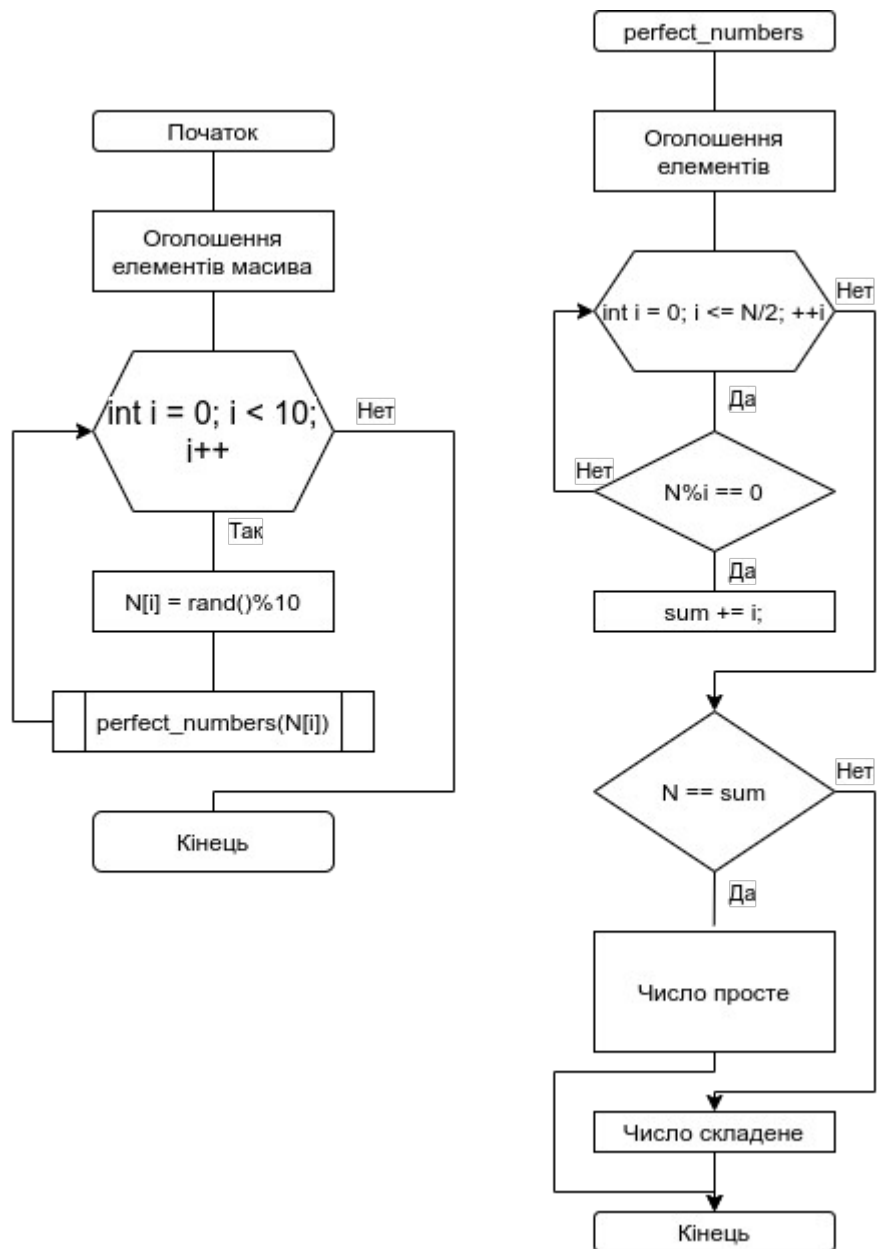


Рисунок 1 — блок-схема програми

Варіанти використання програми 1

- Поставимо точку зупину наприкінці нашої головної функції, та побачимо результат перевірки нашого числа на досконалість.(рис.2)

```

C ex1.c > ...
1  #include <stdio.h>
2  #include <math.h>
3
4  /**
5   * @file ex1.c виконання перевірки числа на досконалість
6   */
7
8  /**
9   * перевірка числа на простоту за допомогою флагів
10  * @param N наше задане число
11  */
12
13  char perfect_numbers (int N);
14
15  /**
16  головна функція {задати функцію, що кожного разу дає нове псевдовипадкове число,
17  оголосити число,
18  виконати перевірку на досконалість}
19  */
20
21  int main (){
22      srand(time(NULL));
23      int N[10];
24      char ans[10];
25      for (int i = 0; i < 10; i++){
26          N[i] = rand()%10;
27          ans[i] = perfect_numbers(N[i]);
28      }
29  }
30
31  char perfect_numbers (int N){
32      int sum = 0;
33      for (int i = 1; i <= N/2; ++i){
34          if (N%i == 0){
35              sum += i;
36          }
37      }
38      if (N == sum){
39          return 'Y';
40      }
41      else {
42          return 'N';
43      }
44  }

```

Рисунок 2 — готова програма з постановкою точки зупину

Опис програми 2

- *Функціональне призначення*: Заповнення масиву простими числами, що не повторюються.

- *Опис логічної структури*:

- Функція ``main``. Оголошує розміри масиву і самі масиви. Викликає функцію ``generation_array``.
- Функція ``generation_array``. Оголошуємо значення числа ,яке ми перевіряємо на простоту, а після заносимо у масив. Перший цикл відповідає за внесення числа у масив, другий за перевірку числа на простість.(рис.3)

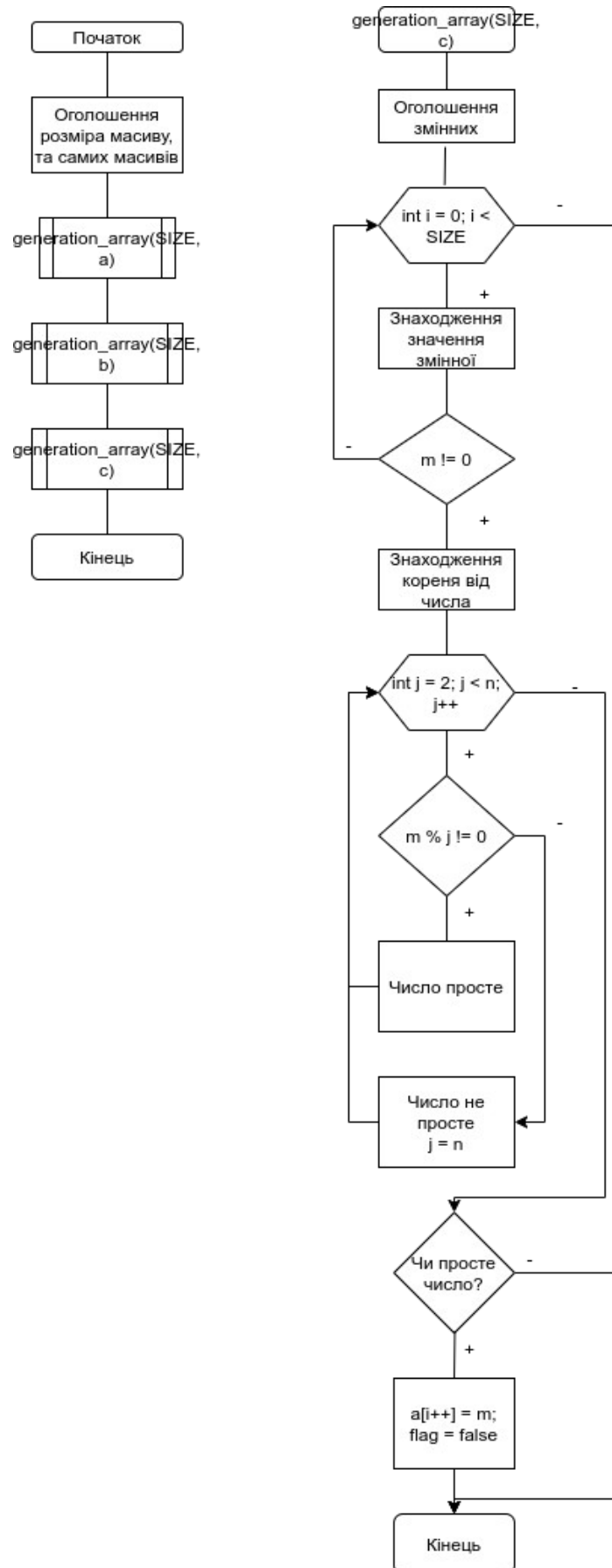


Рисунок 3 — блок-схема програми

Варіанти використання програми 2

- Ми можемо змінювати розмір масиву залежно від нашого бажання, цей масив буде заповнюватися простими числами. Поставимо точку зупину наприкінці нашої головної функції, та побачимо вже заповнений масив.(рис.4)

```
int main() {
    srand(time(NULL));
    int SIZE = 2;
    int a[SIZE];
    int b[SIZE];
    int c[SIZE];

    generation_array(SIZE, a);
    generation_array(SIZE, b);
    generation_array(SIZE, c);
    return 0;
}

void generation_array (int SIZE, int a[]){
    int m;
    bool flag = false;
    for (int i = 0; i < SIZE; i) {
        m = rand() % 100;
        if (m != 0){
            const double eps=0.01;
            double n=0;
            while(n * n <=m){
                n+=eps;
            };
            for (int j = 2; j < n; j++) {
                if (m % j != 0) {
                    flag = true;
                }
                else{
                    flag = false;
                    j = n;
                }
            }
            if (flag)
            {
                a[i++] = m;
                flag = false;
            }
        }
    }
    return ;
}
```

Рисунок 4 — готова програма з постановкою точки зупину

Опис програми 3

- *Функціональне призначення*: Реалізування функції, що буде визначати кількість пар у яких перше число більше за наступне.

- *Опис логічної структури*:

- Функція ``main``. Оголошує змінну, що відповідає на наше функціональне призначення, та показує кількість пар, у котрих друге число менше за попереднє. Викликає функцію ``function``.

- Функція ``function``. Оголошуємо перемінну, що відповідає за наше функціональне призначення. Змінну, завдяки якій ми зможемо порівнювати значення з попереднім значенням. ``result`` - змінюється після перевірки на те, чи задовільняє вона нашій умові.(рис.5)

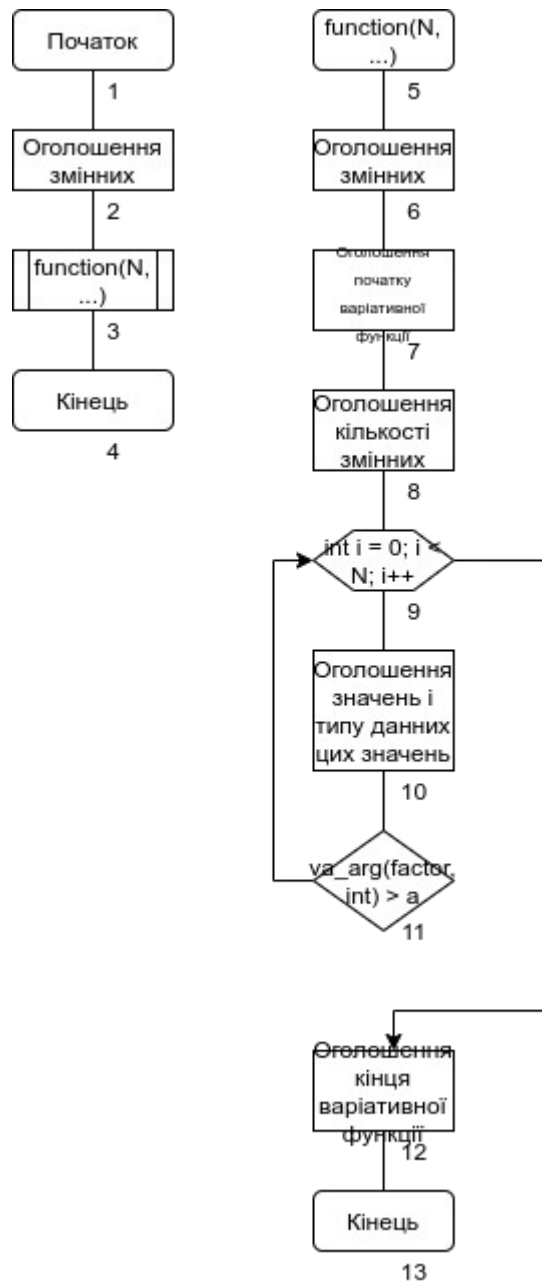


Рисунок 5 — блок-схема програми

Варіанти використання програми 3:

- Ми можемо дізнатися, скільки таких пар, де друге число менше за попереднє. Поставивши точку зупину наприкінці нашої головної функції ми побачимо змінну ,та чому вона дорівнює, що відповідає за те, скільки у нас пр чисел, що відповідають умові завдання.(рис.6)


```

C ex3.c > ...
1  #include <stdarg.h>
2
3  /**
4   * @file ex3.c реалізація функції, що визначає, скільки серед заданої пос
5   *
6   */
7
8  /**
9   * реалізація варіативної функції, що повертає значення кількості пар, що
10  * @param N значення, що відповідає за кількість значень викликаної функц
11  */
12
13  int function(int N, ...);
14
15  /**
16  головна функція { Оголошення змінної та виклик варіативної функції }
17
18  */
19
20  int main(){
21      int result = function(12, 17, 0, -34, 8, 5, 10, -9, 99, 10, 13);
22      return 0;
23  }
24
25  int function(int N, ...){
26
27      int a = 0;
28      int result = 0;
29      va_list factor;
30      va_start (factor, N);
31      for (int i = 1; i < N; i++){
32          va_arg(factor, int);
33          if(va_arg(factor, int) > a){
34              result++;
35          }
36          a = va_arg(factor, int);
37      }
38      va_end(factor);
39      return (result);
40  }
41  }

```

Рисунок 6 — готова програма з постановкою точки зупину

Висновок:

Для виконання лабораторної роботи ми навчилися створювати та реалізовувати Markdoc для функцій.