Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра штучного інтелекту та програмного забезпечення

ЗВІТ
Практична робота №1
дисципліна: «Алгоритми та структур и даних»

Виконала: студентка групи КС-22
Узенкова Дар’я
Перевірив: викладач
Олешко Олег

Харків
2024

Завдання 1. Реалізувати алгоритми Евкліда (найбільшого загального дільника двох цілих чисел - GCD) та Ератосфена (знаходження всіх простих чисел до деякого цілого числа n).

Для кожного з алгоритмів:
1. Розробити блок-схему алгоритму
2. Привести текстовий опис (по кроках) алгоритму.
3. Виконати програмну реалізацію алгоритму у вигляді функції мовою С, навести приклади роботи алгоритму.

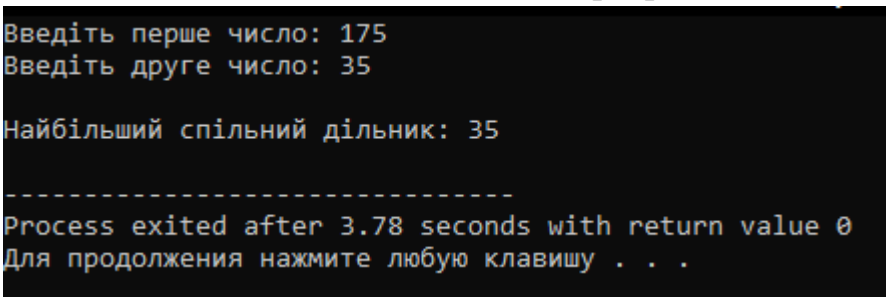
1) Алгоритм Евкліда

Результати виконання завдання 1 наведено:
1. У блок-схемі 1.(див. стор. 13).
2. У лістингу 1 – вихідний код програми.
3. У малюнках 1, 2, 3– результати виконання програми.

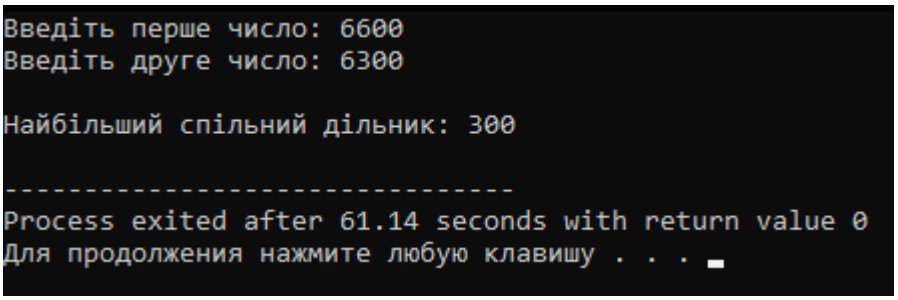
```
#include <stdio.h>

// Функція для обчислення GCD за алгоритмом Евкліда
int findGCD(int a, int b) {
    if (b == 0) {
        return a;
    }
    else {
        return findGCD(b, a % b);
    }
}

int main() {

     system("chcp 1251");
     system("cls");

    int num1, num2;

    printf("Введіть перше число: ");
    scanf("%d", &num1);
```

```
    printf("Введіть друге число: ");
    scanf("%d", &num2);

    printf("\nНайбільший спільний дільник: %d\n", findGCD(num1, num2));

    return 0;
}
```

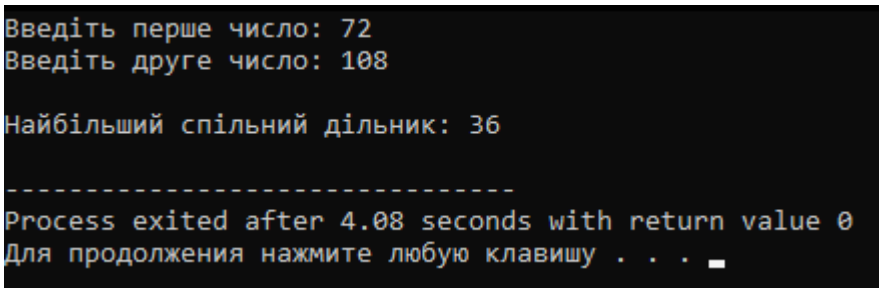
Лістинг 1 – вихідний код програми



Малюнок 1 – результат виконання програми



Малюнок 2 – результат виконання програми



Малюнок 3 – результат виконання програми

Текстовий опис алгоритму Евкліда:

1. Користувач вводить два цілі числа, a і b.
2. Якщо b дорівнює 0, то GCD є a. В цьому випадку алгоритм закінчує свою роботу.
3. Якщо b не дорівнює 0, викликається функція рекурсивно, де:
   - Значення a стає новим b.

- Значення b стає залишком від ділення a на b (тобто a % b).

4. Крок 2 і 3 повторюються, поки b не стане рівним 0.
5. Як тільки b стає 0, результатом є значення a, яке і буде найбільшим спільним дільником двох початкових чисел.

2) Алгоритм Ератосфена

Результати виконання завдання 2 наведено:

1. У блок-схемі 2.(див. стор. 14)
2. У лістингу 1 – вихідний код програми.
3. У малюнках 4, 5, 6 – результати виконання програми.

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

// Функція для знаходження простих чисел за алгоритмом Ератосфена
void SieveOfEratosthene(int n) {
    bool primes[n + 1];
    // Ініціалізація масиву
    int i, j;
    for (i = 0; i <= n; i++) {
        primes[i] = true;
    }
    // Викидання кратних чисел
    for (j = 2; j <= sqrt(n); j++) {
        if (primes[j]) {
            for (i = j * j; i <= n; i += j) {
                primes[i] = false;
            }
        }
    }
    // Виведення простих чисел
    printf("Прості числа до %d: ", n);
    for (j = 2; j <= n; j++) {
        if (primes[j]) {
            printf("%d ", j);
        }
    }
```

```
    printf("\n");
}

int main() {

    system("chcp 1251");
    system("cls");

    int n;
    printf("Алгоритм Ератосфена (знаходження всіх простих чисел до
заданого числа n)\n\n");
    printf("Введіть число n: ");
    scanf("%d", &n);
    printf("\n");

    SieveOfEratosthene(n);

    return 0;
}
```

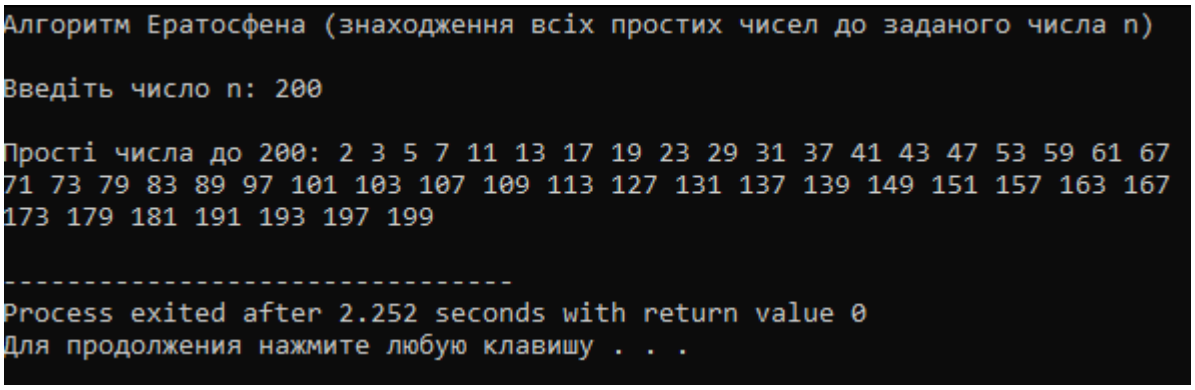
Лістинг 2 – вихідний код програми

```
Алгоритм Ератосфена (знаходження всіх простих чисел до заданого числа n)

Введіть число n: 169

Прості числа до 169: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167

--------------------------------
Process exited after 2.2 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Малюнок 4 – результат виконання програми

```
Алгоритм Ератосфена (знаходження всіх простих чисел до заданого числа n)

Введіть число n: 57

Прості числа до 57: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53

--------------------------------
Process exited after 1.628 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Малюнок 5 – результат виконання програми



Малюнок 6 – результат виконання програми

Текстовий опис алгоритму Ератосфена:

1. Користувач вводить число n.
2. Ініціалізується булевий масив primes[] довжиною n+1, де всі елементи спочатку рівні true.
3. Перший цикл проходить числа від 2 до sqrt(n).
4. Якщо число j є простим (тобто primes[j] == true), всі його кратні (починаючи з j^2) позначаються як складені (тобто primes[i] = false для всіх кратних i).
5. Після завершення циклів програма виводить всі числа від 2 до n, що залишилися з позначкою true у масиві primes[].

Додаткове завдання. Знайдіть, опишіть та реалізуйте інші варіанти "решета", окрім вищезгаданого решета Ератосфена.

1. **Алгоритм Сундарама**

Алгоритм Сундарама починається з ініціалізації масиву для чисел від 1 до (n-1)/2, де n – задане число. Спочатку всі числа вважаються «не викресленими». Потім викреслюються всі числа, які можуть бути представлені у вигляді i + j + 2 * i * j, де $1 \le i \le j$. Після цього залишаються лише ті числа, які не підходять під цю формулу. Кожне невикреслене число 2 * i + 1 є простим, і ми їх виводимо як результат. Додатково враховується просте число 2, яке є єдиним парним простим числом і його потрібно включити до результату.

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

```
// Функція для знаходження простих чисел за алгоритмом Сундарама
void SieveOfSundaram(int n) {
    int nNew = (n – 1) / 2;
    bool marked[nNew + 1];

    // Ініціалізація масиву
    int i, j;
    for (i = 0; i <= nNew; i++) {
        marked[i] = false;
    }
    // Викреслювання чисел за формулою i + j + 2 * i * j
    for (i = 1; i <= nNew; i++) {
        for (j = i; (i + j + 2 * i * j) <= nNew; j++) {
            marked[i + j + 2 * i * j] = true;
        }
    }
    // Виведення простих чисел
    printf(«Прості числа до %d: «, n);
    if (n > 2) {
        printf(«2 «);
    }
    for (i = 1; i <= nNew; i++) {
        if (!marked[i]) {
            printf(«%d «, 2 * i + 1);
        }
    }
    printf(«\n»);
}

int main() {

     system(«chcp 1251»);
     system(«cls»);

    int n;
    printf(«Введіть число n для знаходження простих чисел до n: «);
    scanf(«%d», &n);

    if (n < 2) {
        printf(«Число повинно бути більше за 1.\n»);
```

```
    } else {
        SieveOfSundaram(n);
    }
    return 0;
}
```

Лістинг 3 – вихідний код програми

```
Введіть число n для знаходження простих чисел до n: 211

Прості числа до 211: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211

--------------------------------
Process exited after 3.362 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Малюнок 7 – результат виконання програми

```
Введіть число n для знаходження простих чисел до n: 154

Прості числа до 154: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151

--------------------------------
Process exited after 4.36 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Малюнок 8 – результат виконання програми

```
Введіть число n для знаходження простих чисел до n: 52

Прості числа до 52: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

--------------------------------
Process exited after 7.161 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Малюнок 9 – результат виконання програми

Покроковий опис алгоритму:

1. Алгоритм працює тільки з непарними числами, тому для числа n створюється нова межа nNew = (n – 1) / 2. Це обмежує кількість чисел, з якими ми будемо працювати (фактично до половини чисел, оскільки ми ігноруємо парні числа).
2. Створюється булевий масив розміром nNew + 1, в якому кожен елемент ініціалізується як false. Це означає, що спочатку всі числа вважаються «не викресленими» (потенційно простими).

3. Для кожної пари чисел i і j (де i <= j), які задовольняють умову формули i + j + 2 * i * j <= nNew, викреслюються числа. Це робиться шляхом позначення числа i + j + 2 * i * j в масиві як true. Тобто, такі числа вже не можуть бути простими і виключаються з подальшого розгляду.
4. Число 2 є єдиним парним простим числом, тому воно завжди виводиться окремо, якщо n > 2.
5. Після завершення викреслювання, всі індекси i, для яких значення в масиві залишилися false, відповідають непарним простим числам у вигляді 2 * i + 1. Виводяться всі ці числа як прості.
6. На завершення алгоритму, всі знайдені прості числа (2 і всі непарні прості числа) виводяться на екран.

2) Решето Аткіна

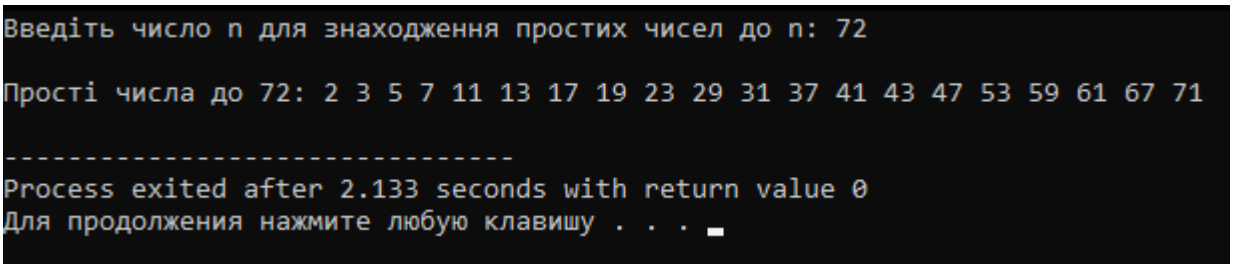
Алгоритм Аткіна є вдосконаленням решета Ератосфена. Він має складність O(n/log log n), як і решето Ератосфена, але використовує інший математичний підхід для знаходження простих чисел.

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

// Функція для знаходження простих чисел за алгоритмом Аткіна
void SieveOfAtkin(int n) {
    if (n < 2) {
        printf("Прості числа відсутні\n");
        return;
    }
    bool sieve[n + 1];
    int i, x, y, j;
    // Ініціалізуємо масив false
    for (i = 0; i <= n; i++) {
        sieve[i] = false;
    }
    // Перевірка умов для квадратів і модулів
    for (x = 1; x * x <= n; x++) {
        for (y = 1; y * y <= n; y++) {
            int num = (4 * x * x + y * y);
            if (num <= n && (num % 12 == 1 || num % 12 == 5)) {
```

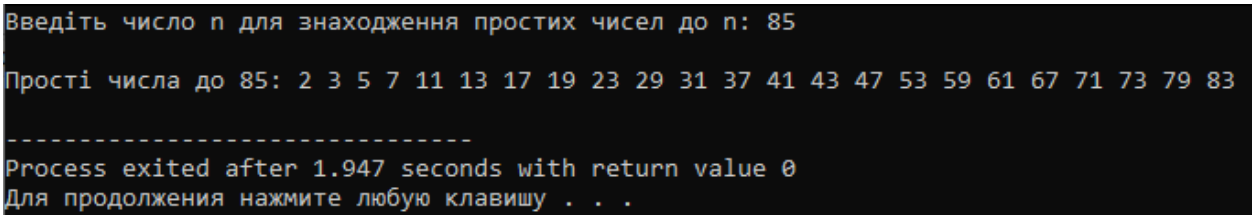
```
                sieve[num] = !sieve[num];
            }

            num = (3 * x * x + y * y);
            if (num <= n && num % 12 == 7) {
                sieve[num] = !sieve[num];
            }

            num = (3 * x * x - y * y);
            if (x > y && num <= n && num % 12 == 11) {
                sieve[num] = !sieve[num];
            }
        }
    }

    // Позначаємо всі кратні квадратів простих чисел
    for (i = 5; i * i <= n; i++) {
        if (sieve[i]) {
            for (j = i * i; j <= n; j += i * i) {
                sieve[j] = false;
            }
        }
    }

    // Виведення простих чисел
    printf("Прості числа до %d: ", n);
    if (n > 2) {
        printf("2 ");
    }
    if (n > 3) {
        printf("3 ");
    }

    for (i = 5; i <= n; i++) {
        if (sieve[i]) {
            printf("%d ", i);
        }
    }
    printf("\n");
}
```

```
int main() {
    system("chcp 1251");
    system("cls");

    int n;
    printf("Введіть число n для знаходження простих чисел до n: ");
    scanf("%d", &n);
    printf("\n");

    SieveOfAtkin(n);

    return 0;
}
```

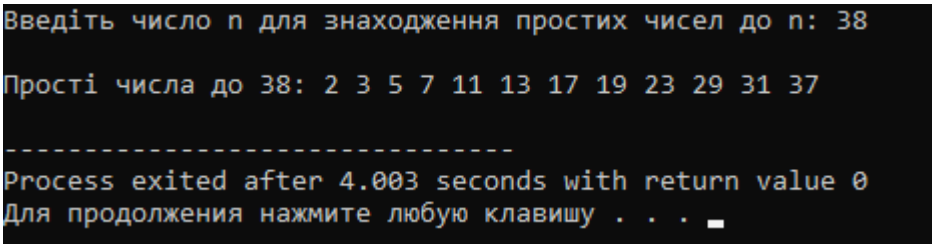
Лістинг 4 – вихідний код програми



Малюнок 10 – результат виконання програми



Малюнок 11 – результат виконання програми



Малюнок 12 – результат виконання програми

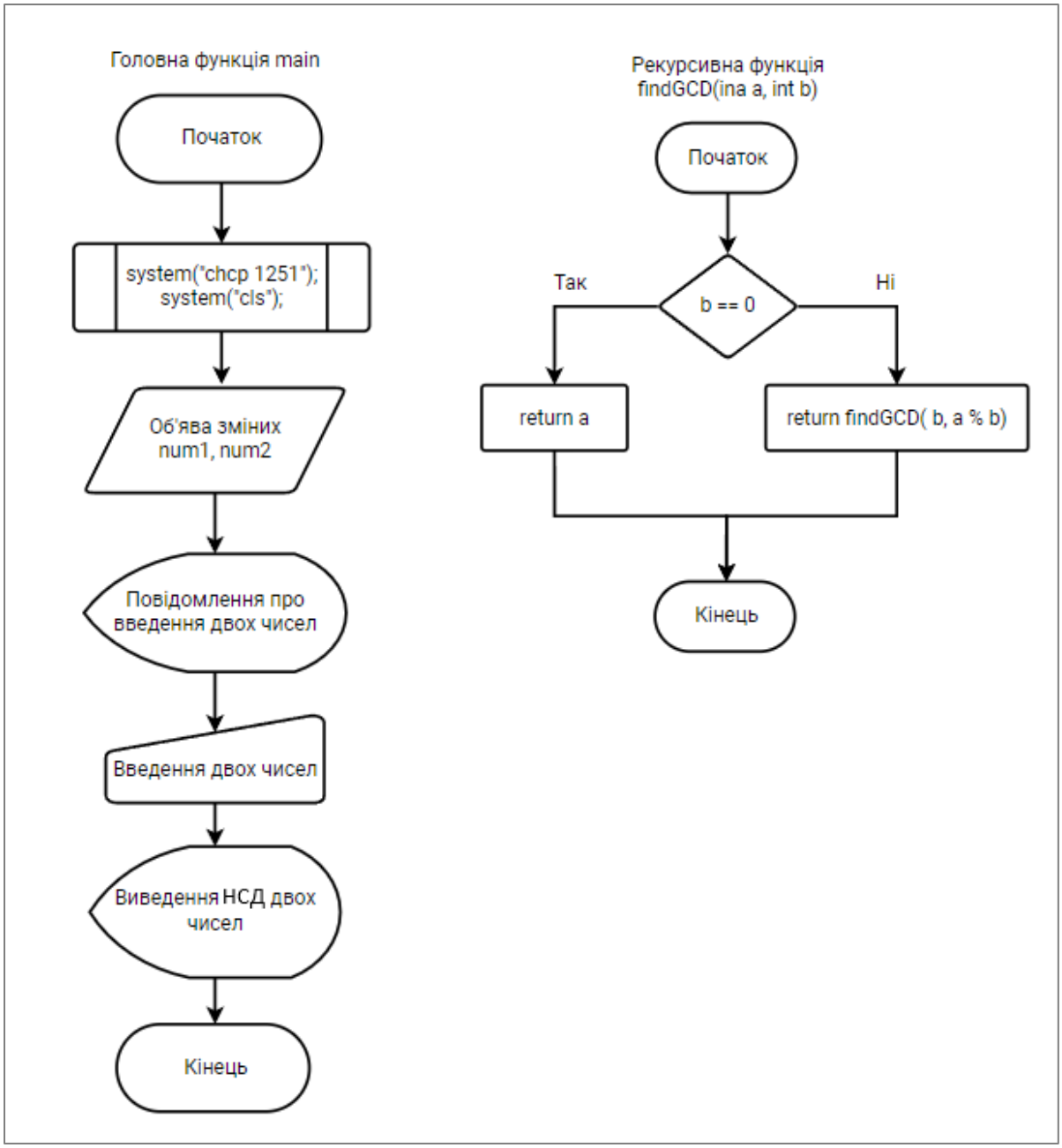
Покроковий опис решета Аткіна:

1) Створюється булевий масив, де всі елементи спочатку встановлені в false.
2) Для кожної пари цілих чисел xxx і yyy перевіряються три формули:

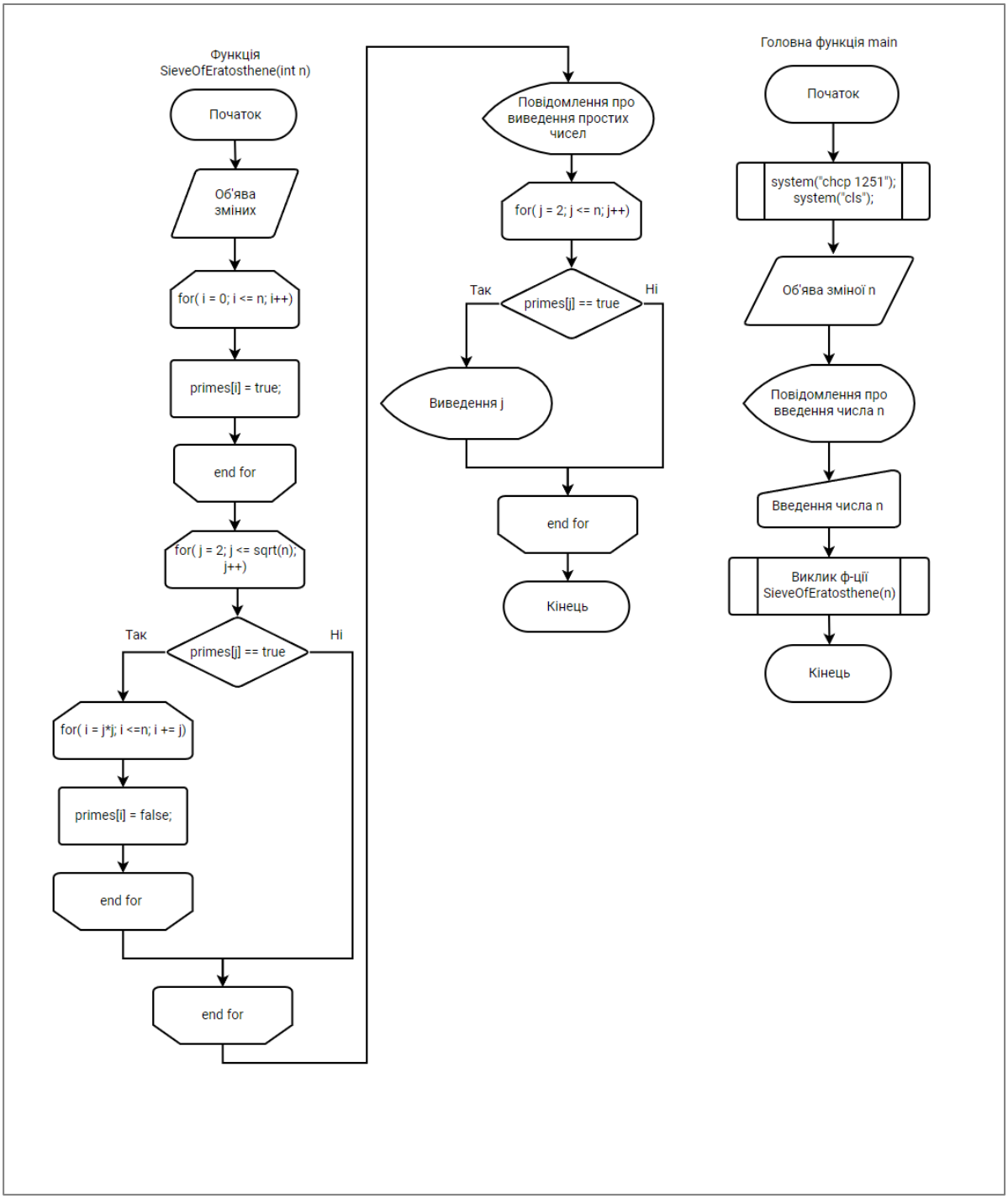
- 4x^2+y^2, якщо результат 1 або 5 при діленні на 12, помітити як потенційно просте.
- 3x^2+y^2, якщо результат 7 при діленні на 12, помітити як потенційно просте.
- 3x^2−y^2, якщо x>y і результат 11 при діленні на 12, помітити як потенційно просте.

3) Викидаються всі числа, які є кратними квадратам простих чисел.
4) Виводяться всі числа, що залишилися поміченими як прості, разом з числом 2 і 3.

Блок-схеми



Блок-схема 1 – алгоритм Евкліда



Блок-схема 2 – алгоритм Ератосфена