

Федеральное государственное автономное образовательное учреждение
высшего образования «Пермский национальный исследовательский
политехнический университет»

Лабораторная работа
«Рекурсия»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-36

Акбашева Софья Руслановна

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Сумма функционального ряда

Вариант №25

Цель: научиться решать задачи с использованием рекурсивных функций.

Постановка задачи: приближенно вычислить значение функции двумя способами.

$$S = x + \frac{x^3}{3!} + \dots + \frac{x^{2n+1}}{(2n+1)!}$$

1) Через сумму функционального ряда с помощью рекурсивной функции. Аргументы функции – n и x , где n – количество членов ряда, x – переменная.

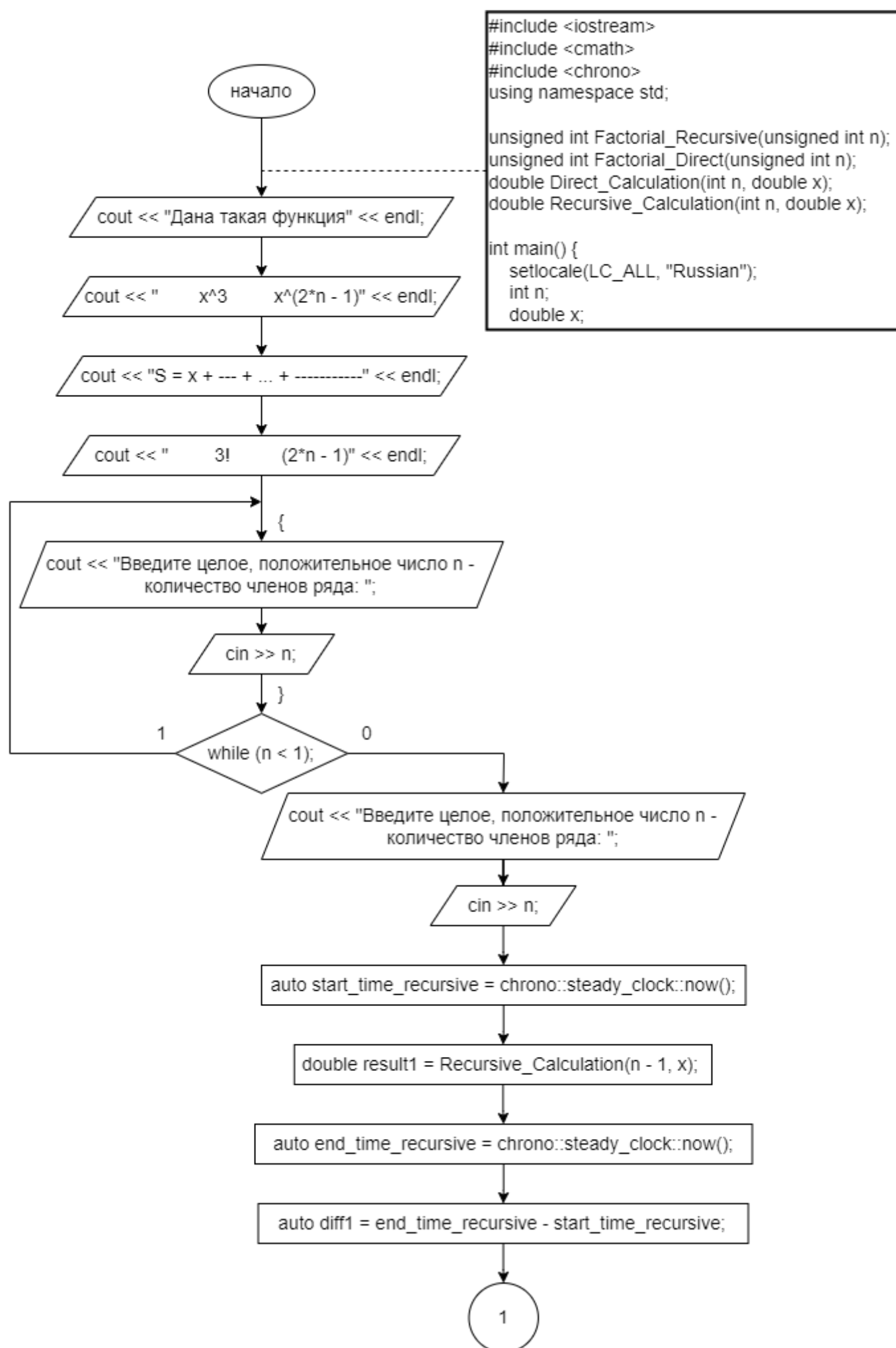
2) Через прямое вычисление значения функции. Аргумент функции – переменная x .

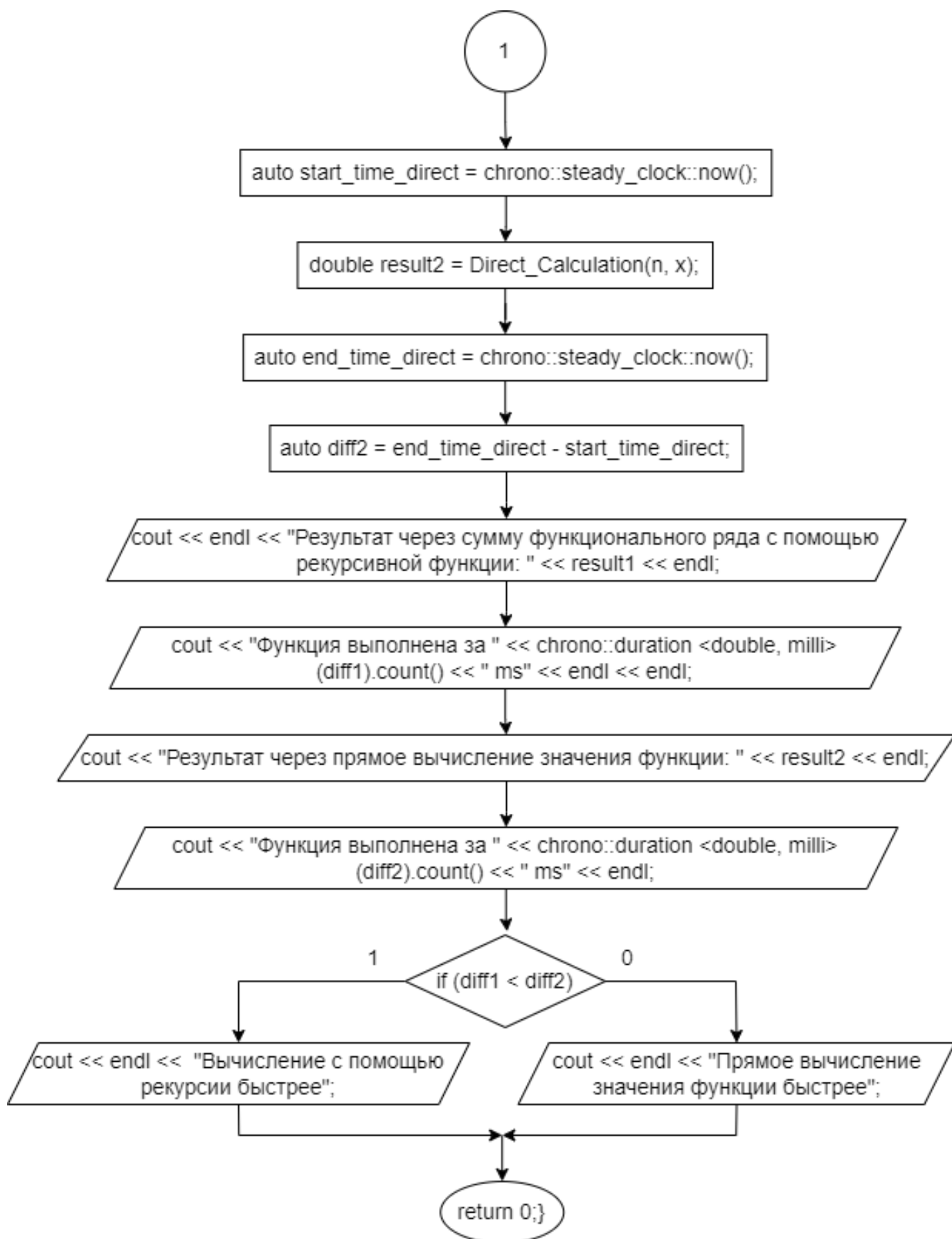
Сравнить полученные результаты.

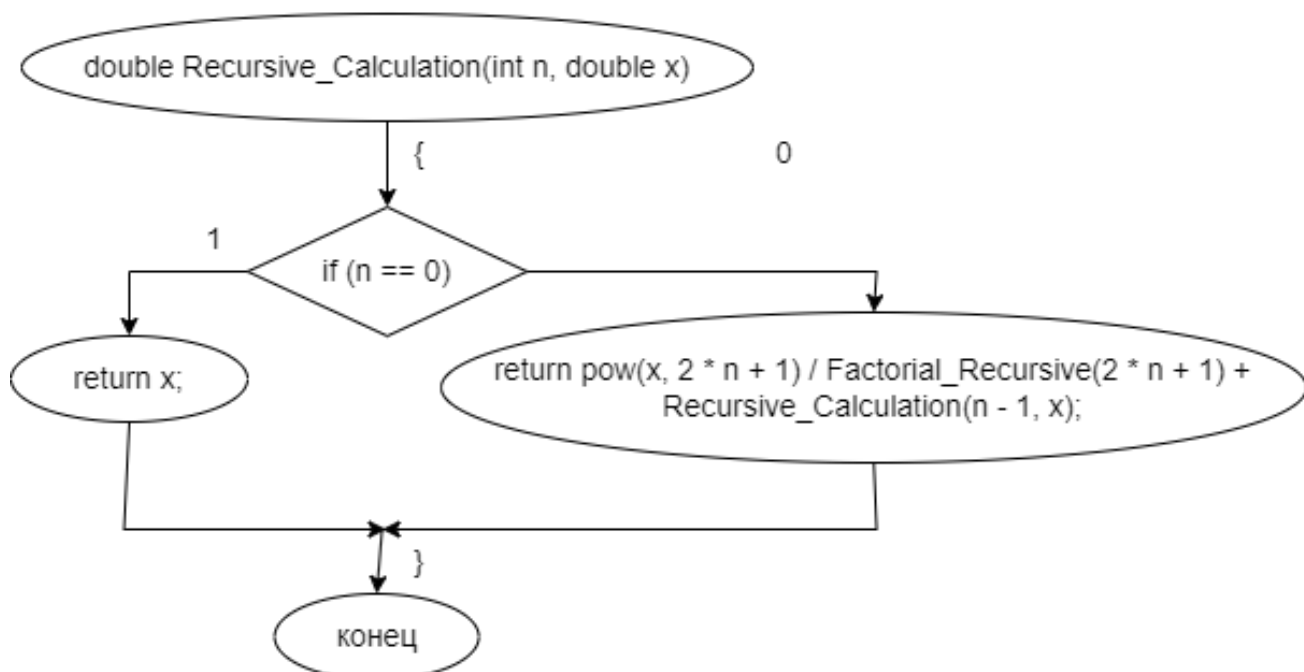
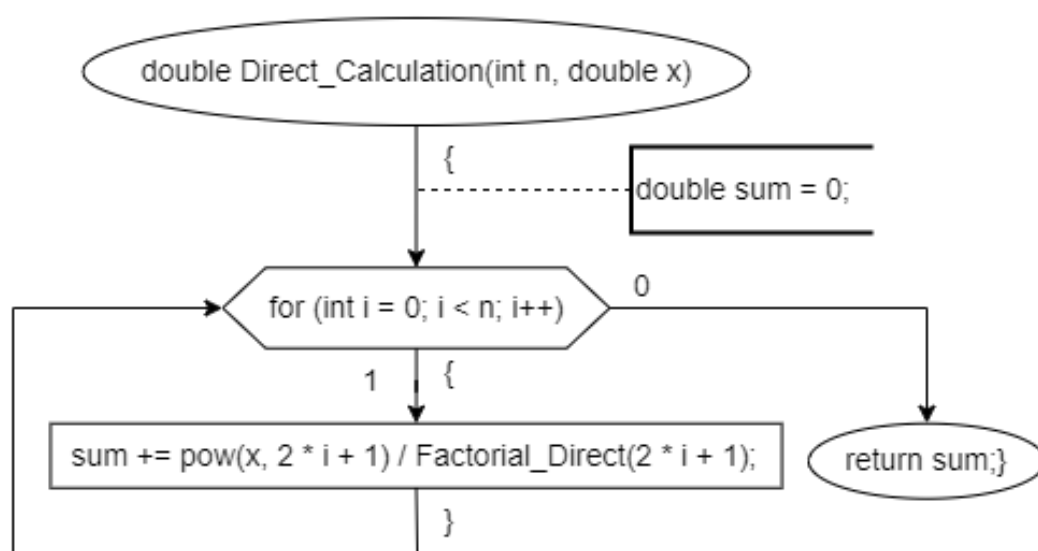
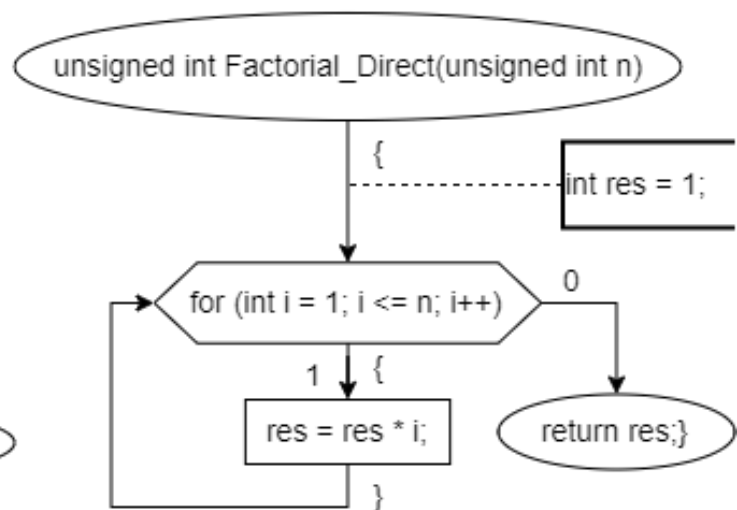
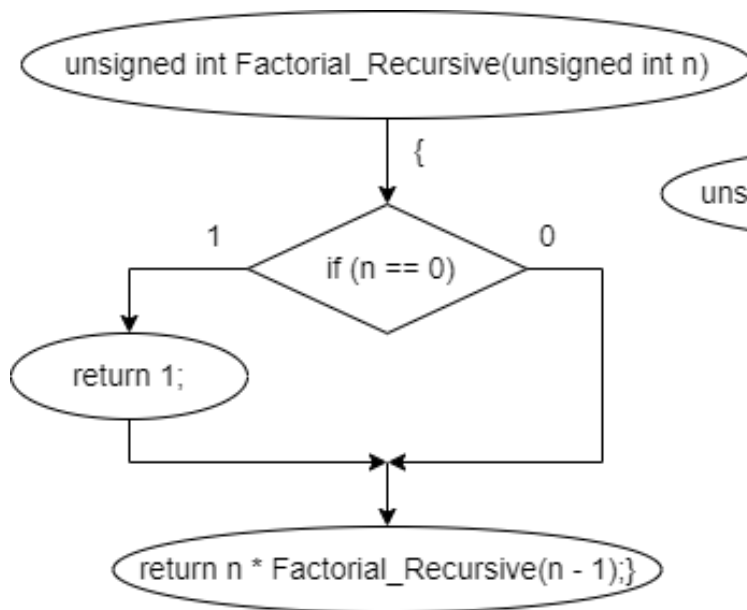
Анализ задачи

1. Пользователь вводит число n – количество членов ряда.
2. Пользователь вводит число x – переменную.
3. Рекурсивная функция возвращает саму себя, прибавляя новый член функционального ряда.
4. Функция рекурсивного вычисления факториала возвращает текущее значение переменной, умноженное на эту функцию от переменной меньшей текущей на 1.
5. Функция прямого вычисления функционального ряда содержит арифметический цикл.
6. Функция прямого вычисления факториала содержит арифметический цикл.
7. Для измерения работы функции необходимо подключение библиотеки `chrono`.

Блок схема







Код программы

```
#include <iostream>
#include <cmath>
#include <chrono>
using namespace std;

unsigned int Factorial_Recursive(unsigned int n);
unsigned int Factorial_Direct(unsigned int n);
double Direct_Calculation(int n, double x);
double Recursive_Calculation(int n, double x);

int main() {
    setlocale(LC_ALL, "Russian");
    int n; //количество членов ряда
    double x; //переменная

    cout << "Дана такая функция" << endl;
    cout << "          x^3          x^(2*n - 1)" << endl;
    cout << "S = x + --- + ... + ----" << endl;
    cout << "          3!          (2*n - 1)" << endl;

    do{
        cout << "Введите целое, положительное число n - количество членов ряда: ";
        cin >> n; //ввожу количество членов ряда
    } while (n < 1);

    cout << "Введите число x - переменную: ";
    cin >> x; //ввожу переменную

    auto start_time_recursive = chrono::steady_clock::now(); // начальное время
    double result1 = Recursive_Calculation(n - 1, x); //вычисляю рекурсией
    auto end_time_recursive = chrono::steady_clock::now(); // конечное время

    auto diff1 = end_time_recursive - start_time_recursive;
                //разница между начальным и конечным временем

    auto start_time_direct = chrono::steady_clock::now(); // начальное время
    double result2 = Direct_Calculation(n, x); //вычисляю прямым способом
    auto end_time_direct = chrono::steady_clock::now(); // конечное время

    auto diff2 = end_time_direct - start_time_direct; //разница между начальным и конечным временем

    cout << endl << "Результат через сумму функционального ряда с помощью рекурсивной функции: " << result1 << endl;
    cout << "Функция выполнена за " << chrono::duration <double, milli>(diff1).count() << " ms" << endl << endl;

    cout << "Результат через прямое вычисление значения функции: " << result2 << endl;
    cout << "Функция выполнена за " << chrono::duration <double, milli>(diff2).count() << " ms" << endl;

    if (diff1 < diff2) cout << endl << "Вычисление с помощью рекурсии быстрее";
    else cout << endl << "Прямое вычисление значения функции быстрее";

    return 0;
}
```

```

unsigned int Factorial_Recursive(unsigned int n) { //вычисление факториала через рекурсию
    if (n == 0)
        return 1;
    return n * Factorial_Recursive(n - 1);
}

unsigned int Factorial_Direct(unsigned int n) { //вычисление факториала через цикл
    int res = 1;
    for (int i = 1; i <= n; i++) {
        res = res * i;
    }
    return res;
}

double Direct_Calculation(int n, double x) { // прямое вычисление значения функции
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += pow(x, 2 * i + 1) / Factorial_Direct(2 * i + 1);
    }
    return sum;
}

double Recursive_Calculation(int n, double x) { // рекурсивное вычисление функции
    if (n == 0) {
        return x; // когда вычисляется первый элемент ряда
    }
    return pow(x, 2 * n + 1) / Factorial_Recursive(2 * n + 1) + Recursive_Calculation(n - 1, x);
    //pow(x, 2 * n + 1) возвожу в необходимую степень
    //Factorial_Recursive(2 * n + 1) вычисляю факториал
    //Recursive_Calculation(n - 1, x) прибавляю предыдущее значение ряда
}

```

Результат работы программы при разных входных данных

1. Пусть n (количество членов) равно 10, а x (переменная) равна 5

Дана такая функция

$$S = x + \frac{x^3}{3!} + \dots + \frac{x^{(2*n - 1)}}{(2*n - 1)!}$$

Введите целое, положительное число n - количество членов ряда: 10
Введите число x - переменную: 5

Результат через сумму функционального ряда с помощью рекурсивной функции: 174242
Функция выполнена за 0.01 ms

Результат через прямое вычисление значения функции: 174242
Функция выполнена за 0.0008 ms

Прямое вычисление значения функции быстрее

2. Пусть n (количество членов) равно 16, а x (переменная) равна 3

```

Дана такая функция
      x^3      x^(2*n - 1)
S = x + --- + ... + -----
      3!      (2*n - 1)
Введите целое, положительное число n - количество членов ряда: 16
Введите число x - переменную: 3

Результат через сумму функционального ряда с помощью рекурсивной функции: 864885
Функция выполнена за 0.0123 ms

Результат через прямое вычисление значения функции: 864885
Функция выполнена за 0.0013 ms

Прямое вычисление значения функции быстрее

```

Вариант с циклом быстрее рекурсии, так как операций столько же, сколько и в рекурсивной функции, но нет дополнительных вычислительных затрат на организацию вложенных вызовов. Поэтому рекурсия в данном случае работает медленнее. Однако обе функции дают верные ответы.

Вывод

В ходе работы я применила знания о работе с рекурсивными функциями. Также, я использовала библиотеку `chrono`, для измерения времени работы функций (рекурсивной и не рекурсивной), для последующего сравнения их время затратности. В результате, мне удалось реализовать поставленную задачу и я выяснила, что не рекурсивная функция работает быстрее (поскольку вызовы методов потребляют больше ресурсов, чем исполнение обычных операторов).

Числа Фибоначчи

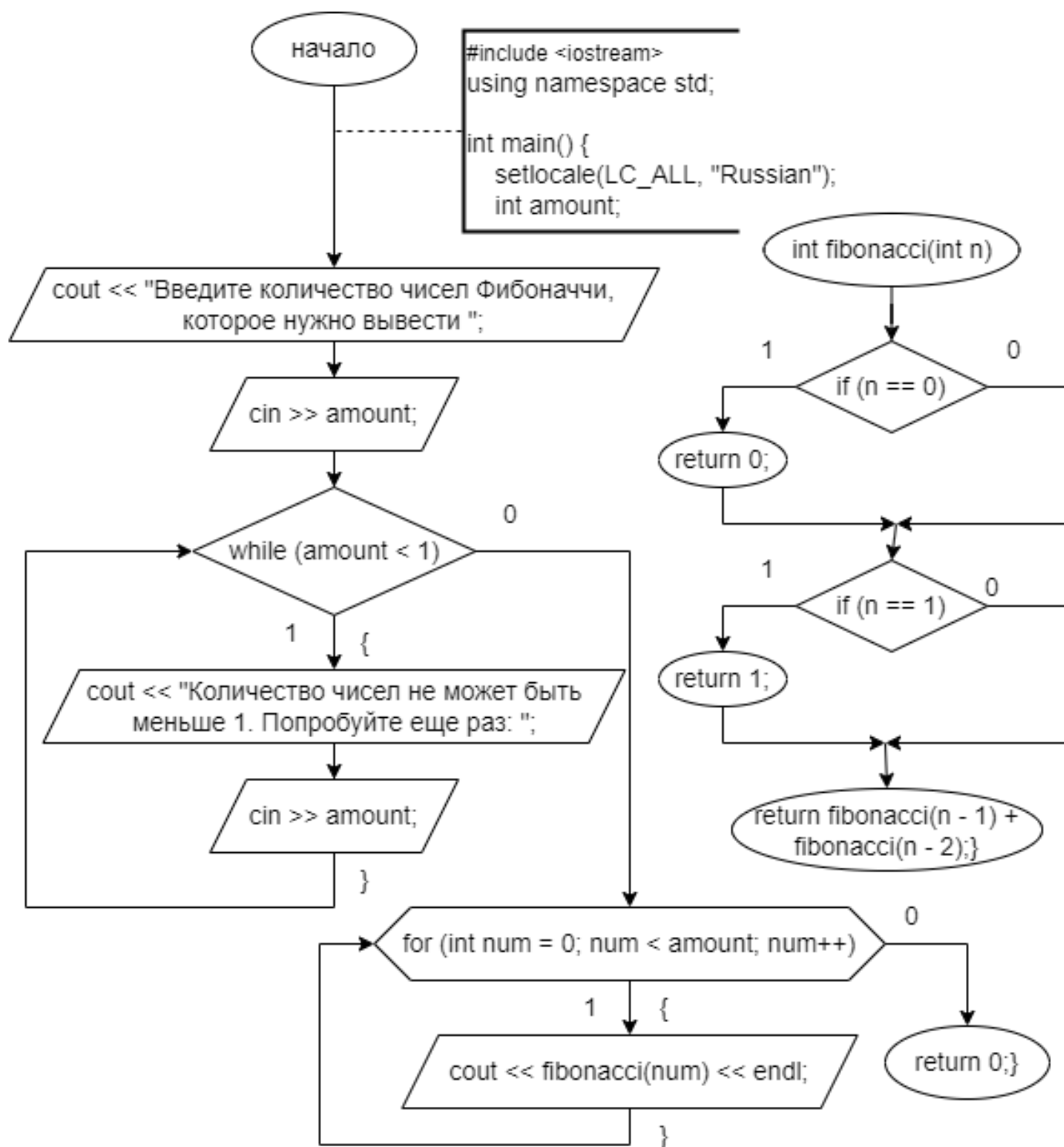
Цель: научиться решать задачи с использованием рекурсивных функций.

Постановка задачи: реализовать рекурсивную функцию для вычисления n-го числа Фибоначчи.

Анализ задачи

1. Каждый элемент ряда представляет собой сумму двух предыдущих чисел (кроме первого и второго числа – первое число равно 0, а второе число равно 1).
2. Функция `fibonacci()` вычисляет n-ое число Фибоначчи. Если в функцию передано значение 0, возвращается 0; если передано 1 – возвращается 1. Иначе возвращается сумма двух предыдущих чисел.
3. Пользователь вводит значение количества чисел Фибоначчи, которые нужно вывести.
4. Если введенное значение меньше 1, пользователь осуществляет ввод до тех пор, пока значение не удовлетворит заданному условию.
5. При помощи цикла `for` осуществляется вывод заданного количества чисел Фибоначчи.

Блок схема



Код программы

```

#include <iostream>
using namespace std;

int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
    // сумма 2х предыдущих чисел ряда Фибоначчи
}

int main() {
    setlocale(LC_ALL, "Russian");
    int amount;
    cout << "Введите количество чисел Фибоначчи, которое нужно вывести ";
    cin >> amount;
    while (amount < 1) {
        cout << "Количество чисел не может быть меньше 1. Попробуйте еще раз: ";
        cin >> amount;
    }
    for (int num = 0; num < amount; num++) { //вывод чисел ряда Фибоначчи
        cout << fibonacci(num) << endl;
    }
    return 0;
}

```

Результат работы

Пусть, необходимо вывести 13 чисел Фибоначчи

```

Введите количество чисел Фибоначчи, которое нужно вывести 13
0
1
1
2
3
5
8
13
21
34
55
89
144

```

Вывод

В ходе работы я применила знания о работе с рекурсивными функциями. Мне удалось реализовать функцию нахождения чисел Фибоначчи, а также вывод этих чисел от первого, до необходимого.

Ханойская башня

Цель: научиться решать задачи с использованием рекурсивных функций.

Постановка задачи: Даны 1 стержень с дисками разного размера и 2 пустых стержня. Нужно переместить диски с одного стержня на другой, перекладывать можно только по одному диску за ход, складывать диски можно только меньший на больший. Реализовать программу, которая определяет перестановки этих дисков с помощью наименьшего количества ходов.

Анализ задачи

1. Нужно решать задачу не с начала, а с конца. Чтобы переложить пирамидку на нужный стержень, нужно переложить на нужный стержень нижний диск, а сделать это можно только тогда, когда $n - 1$ дисков будут на свободном стержне.

2. Перекладываем $n - 1$ дисков на свободный стержень.

3. Перекладываем n -ый диск на нужный стержень.

4. Перекладываем $n - 1$ дисков на нужный стержень.

5. Чтобы переложить $n - 1$ дисков, нужно: перекладываем $n - 2$ дисков на свободный стержень.

6. Перекладываем $n - 1$ диск на нужный стержень.

7. Перекладываем $n - 2$ дисков на нужный стержень.

8. Рекурсивный алгоритм продолжается до тех пор, пока n не достигнет 0.

9. Для визуализации проводимых операций необходимо реализовать функцию вывода башен и функцию изменения выводимых башен. Для этих функций необходим массив, где будут храниться текущие позиции колец.

10. Вывод башен должен происходить послойно, т.е. сначала верхний слой всех башен, затем слой, который ниже верхнего и так далее.

Блок схема

начало

```
#include <iostream>
using namespace std;

void moving_the_ring(int* ptr_tower, int start, int end, int max_size, int n);
void one_layer_of_the_tower(int* ptr_tower, int n, int layer, int num_tower);
void print_tower(int* ptr_tower, int n, int max_size);
int Han(int n, int start, int end, int point, int* ptr_tower, int max_size, int constant_n);

int main() {
    setlocale(LC_ALL, "Russian");
    const int max_size = 100;
    int start, end, point, n, tower_list[max_size * 3];
```

```
cout << " Ханойская башня " << endl << "Даны 3
стержня" << endl;
```

```
{
```

```
cout << "Введите номер СТАРТОВОГО стержня (1, 2 или 3) ";
```

```
cin >> start;
```

```
}
```

1

```
while (start < 1 || start > 3);
```

0

```
{
```

```
cout << "Введите номер КОНЕЧНОГО стержня (1, 2 или 3) ";
```

```
cin >> end;
```

```
}
```

1

```
while (end < 1 || end > 3 || start == end);
```

0

1

```
switch (start + end){
```

```
case 3:
```

```
point = 3;
```

```
break;
```

```
case 4:
```

```
point = 3;
```

```
break;
```

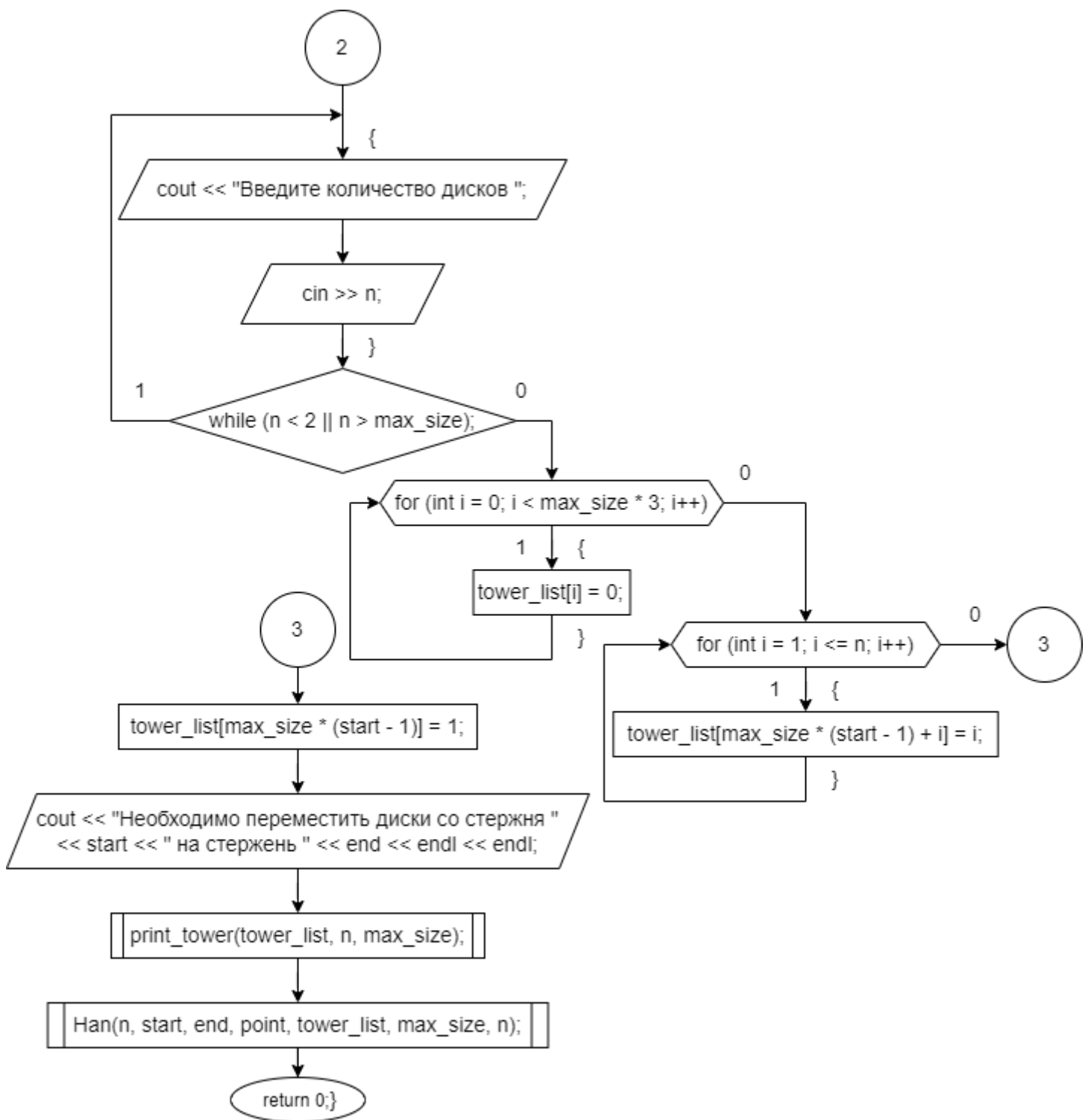
```
default:
```

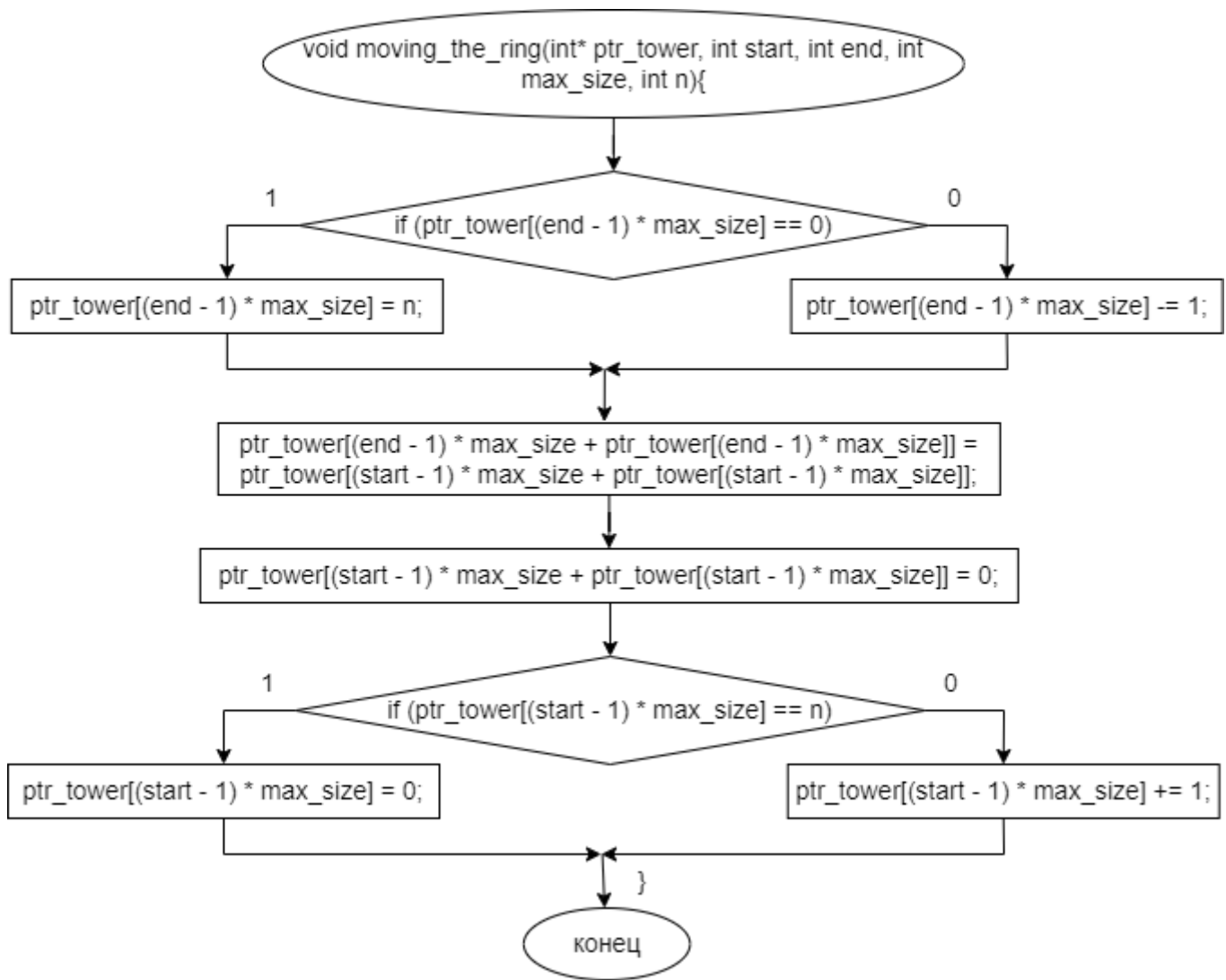
```
point = 1;
```

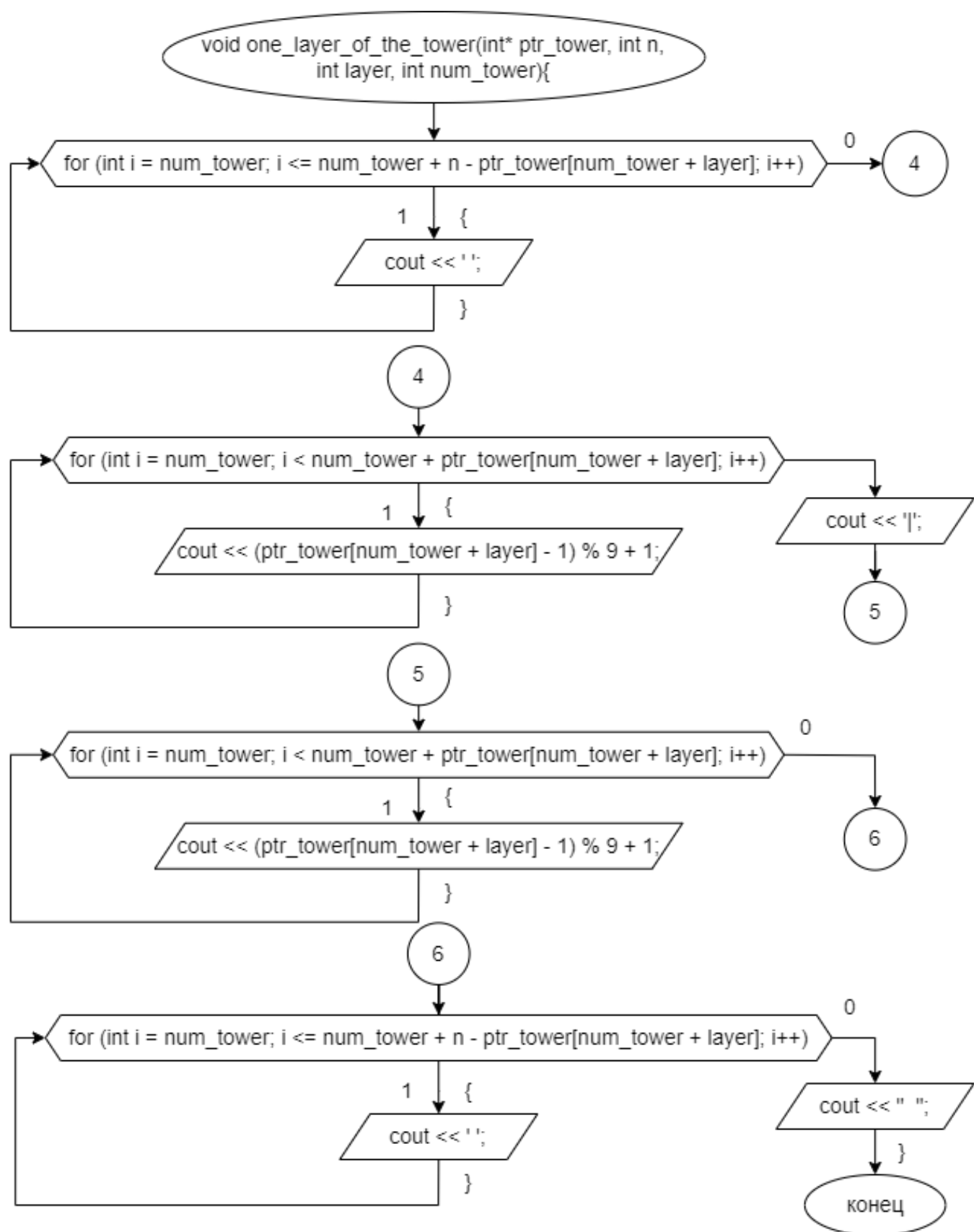
```
break;
```

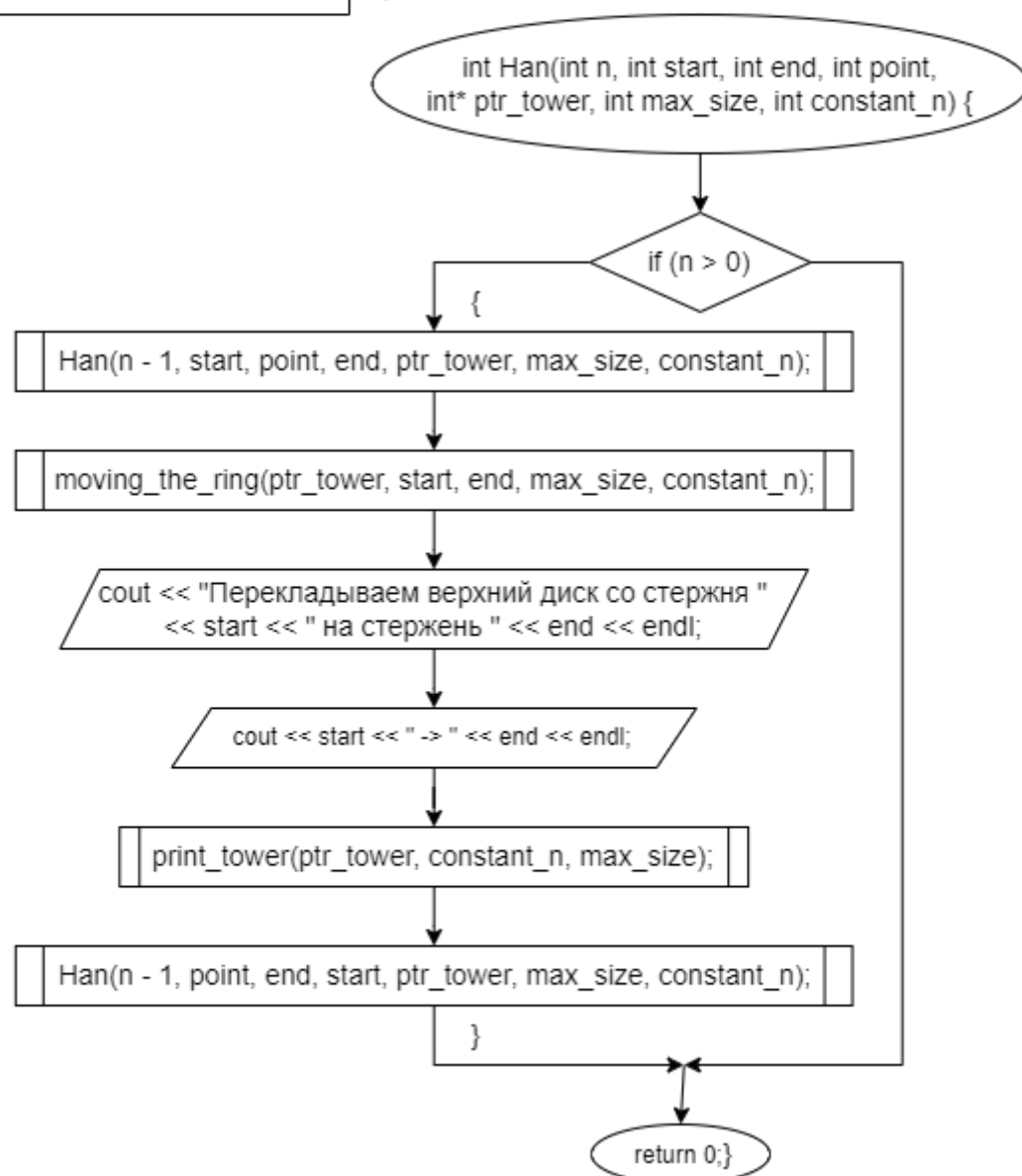
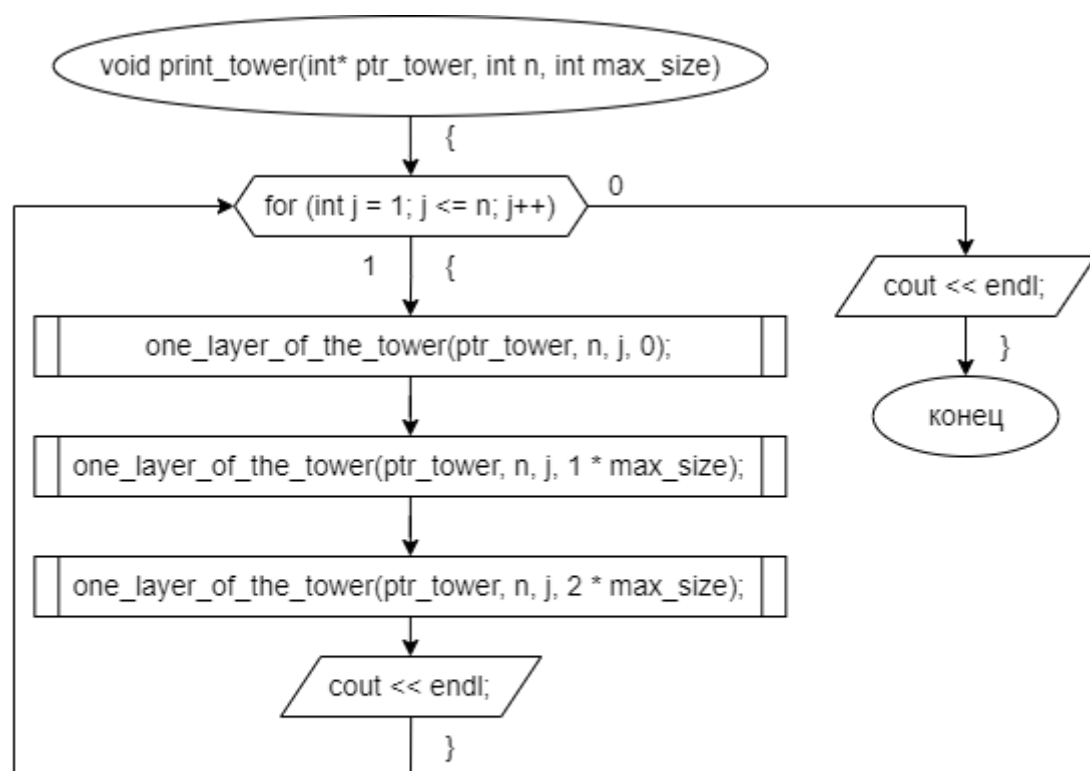
```
}
```

2









Код программы

```
#include <iostream>
using namespace std;

void moving_the_ring(int* ptr_tower, int start, int end, int max_size, int n); //передвигаю кольцо с одной башни на другую
void one_layer_of_the_tower(int* ptr_tower, int n, int layer, int num_tower); //вывожу один слой ВСЕХ башен
void print_tower(int* ptr_tower, int n, int max_size); //вывод всех 3х башен
int Han(int n, int start, int end, int point, int* ptr_tower, int max_size, int constant_n);

int main() {
    setlocale(LC_ALL, "Russian");
    const int max_size = 100;
    int start, end, point, n, tower_list[max_size * 3];

    cout << " Ханойская башня " << endl << "Даны 3 стержня" << endl;
    do {
        cout << "Введите номер СТАРТОВОГО стержня (1, 2 или 3) ";
        cin >> start;
    } while (start < 1 || start > 3);
    do {
        cout << "Введите номер КОНЕЧНОГО стержня (1, 2 или 3) ";
        cin >> end;
    } while (end < 1 || end > 3 || start == end);

    switch (start + end) { //нахожу номер промежуточного стержня
    case 3: point = 3; break;
    case 4: point = 2; break;
    default: point = 1; break;
    }

    do {
        cout << "Введите количество дисков ";
        cin >> n;
    } while (n < 2 || n > max_size);

    for (int i = 0; i < max_size * 3; i++) { //заполняю нулями перечень башен
        tower_list[i] = 0;
    }
    for (int i = 1; i <= n; i++) {
        tower_list[max_size * (start - 1) + i] = i; //заполняю дисками стартовый стержень
    }
    tower_list[max_size * (start - 1)] = 1; //номер верхнего диска на стартовой башне

    cout << "Необходимо переместить диски со стержня " << start << " на стержень " << end << endl << endl;

    print_tower(tower_list, n, max_size);
    Han(n, start, end, point, tower_list, max_size, n);

    return 0;
}
```

```

void moving_the_ring(int* ptr_tower, int start, int end, int max_size, int n) { //передвигаю кольцо с одной башни на другую
    if (ptr_tower[(end - 1) * max_size] == 0) { //добавляю кольцо
        ptr_tower[(end - 1) * max_size] = n; //меняю верхнее кольцо конечной башни если на башне нет колец
    }
    else {
        ptr_tower[(end - 1) * max_size] -= 1; //меняю верхнее кольцо конечной башни если на башне есть кольца
    }
    ptr_tower[(end - 1) * max_size + ptr_tower[(end - 1) * max_size]] = ptr_tower[(start - 1) * max_size + ptr_tower[(start - 1) * max_size]];
    //ставлю новое кольцо

    ptr_tower[(start - 1) * max_size + ptr_tower[(start - 1) * max_size]] = 0; //убираю верхнее кольцо с начальной башни
    if (ptr_tower[(start - 1) * max_size] == n) {
        ptr_tower[(start - 1) * max_size] = 0; //меняю верхнее кольцо начальной башни если на башне нет колец
    }
    else {
        ptr_tower[(start - 1) * max_size] += 1; //меняю верхнее кольцо начальной башни если на башне есть кольца
    }
    //ptr_tower список, где хранятся кольца всех башен
    //start номер башни, с КОТОРОЙ надо ПЕРЕДВИНУТЬ кольцо
    //end номер башни, НА КОТОРУЮ надо ПЕРЕДВИНУТЬ кольцо
    //max_size сколько ячеек отведено на описание одной башни в массиве
    //n сколько всего ВСЕХ колец
}

void one_layer_of_the_tower(int* ptr_tower, int n, int layer, int num_tower) { //вывожу один слой ВСЕХ башен
    for (int i = num_tower; i <= num_tower + n - ptr_tower[num_tower + layer]; i++) { //пробелы до кольца
        cout << ' ';
    }
    for (int i = num_tower; i < num_tower + ptr_tower[num_tower + layer]; i++) { //кольцо
        cout << (ptr_tower[num_tower + layer] - 1) % 9 + 1;
    }
    cout << '|'; //стержень башни
    for (int i = num_tower; i < num_tower + ptr_tower[num_tower + layer]; i++) { //кольцо
        cout << (ptr_tower[num_tower + layer] - 1) % 9 + 1;
    }
    for (int i = num_tower; i <= num_tower + n - ptr_tower[num_tower + layer]; i++) { //пробелы после кольца
        cout << ' ';
    }
    cout << " "; //пробел между башен
    //ptr_tower список, где хранятся кольца всех башен
    //n сколько всего ВСЕХ колец
    //layer слой башни, где 1 - верхний, n - нижний
    //num_tower индекс, с которого начинаются ячейки очередной башни
}

void print_tower(int* ptr_tower, int n, int max_size) { //вывод всех 3х башен
    for (int j = 1; j <= n; j++) {
        one_layer_of_the_tower(ptr_tower, n, j, 0); //слой башни 1
        one_layer_of_the_tower(ptr_tower, n, j, 1 * max_size); //слой башни 2
        one_layer_of_the_tower(ptr_tower, n, j, 2 * max_size); //слой башни 3
        cout << endl;
        //ptr_tower список, где хранятся кольца всех башен
        //n сколько всего ВСЕХ колец
        //max_size сколько ячеек отведено на описание одной башни в массиве
    }
    cout << endl;
}

int Han(int n, int start, int end, int point, int* ptr_tower, int max_size, int constant_n) { //перекладываю кольца рекурсией
    if (n > 0) {
        Han(n - 1, start, point, end, ptr_tower, max_size, constant_n);

        moving_the_ring(ptr_tower, start, end, max_size, constant_n); //передвигаю кольцо с одной башни на другую

        cout << "Перекладываем верхний диск со стержня " << start << " на стержень " << end << endl;
        cout << start << " -> " << end << endl; //с какой башни на какую передвинули кольцо
        print_tower(ptr_tower, constant_n, max_size); //вывожу башни

        Han(n - 1, point, end, start, ptr_tower, max_size, constant_n);
    }
    //n текущее кольцо, с которым проводятся операции
    //start стержень с которого надо переместить кольцо
    //end стержень, на который надо переместить кольцо
    //point промежуточный стержень
    //ptr_tower список, где хранится расположение всех колец всех башен
    //max_size сколько ячеек отведено на описание одной башни в массиве
    //constant_n сколько всего ВСЕХ колец
    return 0;
}

```

Рассмотрю алгоритм работы

Пусть дано 3 диска, первый стержень – начальный, третий стержень – конечный.

Ханойская башня

Даны 3 стержня

Введите номер СТАРТОВОГО стержня (1, 2 или 3) 1

Введите номер КОНЕЧНОГО стержня (1, 2 или 3) 3

Введите количество дисков 3

Необходимо переместить диски со стержня 1 на стержень 3

1 1		
22 22		
333 333		

Параметры функции

Нап(3, 1, 3, 2, [1, 1, 2, 3, ... , 0, 0, 0, 0, ... , 0, 0, 0, 0], 100, 3)

Необходимо переложить со стержня 1 на стержень 3 диск радиуса 3

Параметры функции

Нап(2, 1, 2, 3, [1, 1, 2, 3, ... , 0, 0, 0, 0, ... , 0, 0, 0, 0], 100, 3)

Необходимо переложить со стержня 1 на стержень 2 диск радиуса 2

Параметры функции

Нап(1, 1, 3, 2, [1, 1, 2, 3, ... , 0, 0, 0, 0, ... , 0, 0, 0, 0], 100, 3)

Необходимо переложить со стержня 1 на стержень 3 диск радиуса 1

Параметры функции

Нап(0, 1, 2, 3, [1, 1, 2, 3, ... , 0, 0, 0, 0, ... , 0, 0, 0, 0], 100, 3)

Перекладываем верхний диск со стержня 1 на стержень 3

1 -> 3

22 22		1 1
333 333		

Параметры функции

Нап(0, 2, 3, 1, [2, 0, 2, 3, ... , 0, 0, 0, 0, ... , 3, 0, 0, 1], 100, 3)

Перекладываем верхний диск со стержня 1 на стержень 2

1 -> 2

333 333	22 22	1 1

Параметры функции

Нап(1, 3, 2, 1, [3, 0, 0, 3, ... , 3, 0, 0, 2, ... , 3, 0, 0, 1], 100, 3)

Необходимо переложить со стержня 3 на стержень 2 диск радиуса 1

Параметры функции

Нап(0, 3, 1, 2, [3, 0, 0, 3, ... , 3, 0, 0, 2, ... , 3, 0, 0, 1], 100, 3)

Перекладываем верхний диск со стержня 3 на стержень 2

3 -> 2

333 333	1 1	
	22 22	

Параметры функции

Нап(0, 1, 2, 3, [3, 0, 0, 3, ... , 2, 0, 1, 2, ... , 0, 0, 0, 0], 100, 3)

Перекладываем верхний диск со стержня 1 на стержень 3

1 -> 3

```

|           |           |
|           1|1         333|333
|          22|22

```

Параметры функции

Нап(2, 2, 3, 1, [0, 0, 0, 0, ... , 2, 0, 1, 2, ... , 3, 0, 0, 3], 100, 3)

Необходимо переложить со стержня 2 на стержень 3 диск радиуса 2

Параметры функции

Нап(1, 2, 1, 3, [0, 0, 0, 0, ... , 2, 0, 1, 2, ... , 3, 0, 0, 3], 100, 3)

Необходимо переложить со стержня 2 на стержень 1 диск радиуса 1

Параметры функции

Нап(0, 2, 3, 1, [0, 0, 0, 0, ... , 2, 0, 1, 2, ... , 3, 0, 0, 3], 100, 3)

Перекладываем верхний диск со стержня 2 на стержень 1

2 -> 1

```

|           |           |
1|1         22|22         333|333

```

Параметры функции

Нап(0, 3, 1, 2, [3, 0, 0, 1, ... , 3, 0, 0, 2, ... , 3, 0, 0, 3], 100, 3)

Перекладываем верхний диск со стержня 2 на стержень 3

2 -> 3

```

|           |           |
1|1         |           22|22
|          333|333

```

Параметры функции

Нап(1, 1, 3, 2, [3, 0, 0, 1, ... , 0, 0, 0, 0, ... , 2, 0, 2, 3], 100, 3)

Необходимо переложить со стержня 1 на стержень 3 диск радиуса 1

Параметры функции

Нап(0, 1, 2, 3, [3, 0, 0, 1, ... , 0, 0, 0, 0, ... , 2, 0, 2, 3], 100, 3)

Перекладываем верхний диск со стержня 1 на стержень 3

1 -> 3

```

|           |           1|1
|           |          22|22
|           |          333|333

```

Параметры функции

Нап(0, 2, 3, 1, [0, 0, 0, 0, ... , 0, 0, 0, 0, ... , 1, 1, 2, 3], 100, 3)

Результат работы программы

1) Пусть дано 3 диска, начальный стержень – первый, конечный стержень – второй.

Ханойская башня
Даны 3 стержня
Введите номер СТАРТОВОГО стержня (1, 2 или 3) 1
Введите номер КОНЕЧНОГО стержня (1, 2 или 3) 3
Введите количество дисков 3
Необходимо переместить диски со стержня 1 на стержень 3

1 1		
22 22		
333 333		

Перекладываем верхний диск со стержня 1 на стержень 3
1 -> 3

22 22		1 1
333 333		

Перекладываем верхний диск со стержня 1 на стержень 2
1 -> 2

333 333	22 22	1 1

Перекладываем верхний диск со стержня 3 на стержень 2
3 -> 2

333 333	1 1	
22 22		

Перекладываем верхний диск со стержня 1 на стержень 3
1 -> 3

	1 1	
	22 22	333 333

Перекладываем верхний диск со стержня 2 на стержень 1
2 -> 1

1 1	22 22	333 333

Перекладываем верхний диск со стержня 2 на стержень 3
2 -> 3

1 1		22 22
		333 333

Перекладываем верхний диск со стержня 1 на стержень 3
1 -> 3

		1 1
		22 22
		333 333

2) Пусть дано 4 диска, начальный стержень – второй, конечный стержень – первый.

Ханойская башня

Даны 3 стержня

Введите номер СТАРТОВОГО стержня (1, 2 или 3) 2

Введите номер КОНЕЧНОГО стержня (1, 2 или 3) 1

Введите количество дисков 4

Необходимо переместить диски со стержня 2 на стержень 1

	1 1	
	22 22	
	333 333	
	4444 4444	

Перекладываем верхний диск со стержня 2 на стержень 3

2 -> 3

	22 22		
	333 333		
	4444 4444	1 1	

Перекладываем верхний диск со стержня 2 на стержень 1

2 -> 1

	333 333		
22 22	4444 4444	1 1	

Перекладываем верхний диск со стержня 3 на стержень 1

3 -> 1

1 1	333 333		
22 22	4444 4444		

Перекладываем верхний диск со стержня 2 на стержень 3

2 -> 3

1 1			
22 22	4444 4444	333 333	

Перекладываем верхний диск со стержня 1 на стержень 2

1 -> 2

	1 1		
22 22	4444 4444	333 333	

Перекладываем верхний диск со стержня 1 на стержень 3

1 -> 3

	1 1	22 22	
	4444 4444	333 333	

Перекладываем верхний диск со стержня 2 на стержень 3

2 -> 3

		1 1
		22 22
	4444 4444	333 333

Перекладываем верхний диск со стержня 2 на стержень 1

2 -> 1

		1 1
		22 22
4444 4444		333 333

Перекладываем верхний диск со стержня 3 на стержень 1

3 -> 1

1 1		22 22
4444 4444		333 333

Перекладываем верхний диск со стержня 3 на стержень 2

3 -> 2

1 1		
4444 4444	22 22	333 333

Перекладываем верхний диск со стержня 1 на стержень 2

1 -> 2

	1 1	
4444 4444	22 22	333 333

Перекладываем верхний диск со стержня 3 на стержень 1

3 -> 1

333 333	1 1	
4444 4444	22 22	

Перекладываем верхний диск со стержня 2 на стержень 3

2 -> 3

333 333		
4444 4444	22 22	1 1

Перекладываем верхний диск со стержня 2 на стержень 1

2 -> 1

22 22		
333 333		
4444 4444		1 1

Перекладываем верхний диск со стержня 3 на стержень 1

3 -> 1


1 1		
22 22		
333 333		
4444 4444		



Вывод




В ходе работы я применила знания о работе с рекурсивными функциями. Мне удалось реализовать алгоритм, который переносит кольца с одного стержня на другой. Также я выполнила визуализацию проводимых действий с помощью массива, в котором хранились текущие позиции колец, и соответствующих функций.



GitHub


Ссылка: <https://github.com/SonyAkb/laboratory-work-recursion.git>









 **laboratory-work-recursion** Public



 Pin  Unwatch 1

 main  1 Branch  0 Tags

 Add file  Code








































 SonyAkb Update README.md 03f6f8f · 2 hours ago 43 Commits

 README.md	Update README.md	2 hours ago
 lab work recursion.drawio.png	Обновление lab work recursion.drawio.png	last week
 ЛР Рекурсия.cpp	Убираю else	last week
 ЛР Рекурсия.docx	Отчет	last week
 Ханойская башня - блок схема.drawio.png	Обновление Ханойская башня - блок схема.drawio.png	3 hours ago
 Ханойская башня.cpp	Мелкие исправления	2 hours ago
 Числа Фибоначчи - блок схема.drawio.png	Обновление Числа Фибоначчи - блок схема.drawio.png	yesterday
 Числа Фибоначчи.cpp	Смена кодировки	yesterday

 README 

Лабораторная работа "Рекурсия".

- Сумма функционального ряда - Вариант №25. Цель: научиться решать задачи с использованием рекурсивных функций. Постановка задачи: приблизительно вычислить значение функции двумя способами. Сравнить полученные результаты. Функция: $S = x + (x^3)/3 + \dots + (x^{(2n+1)})/((2n+1)!)$
 - Через сумму функционального ряда с помощью рекурсивной функции. Аргументы функции – n и x, где n – количество членов ряда, x – переменная.
 - Через прямое вычисление значения функции. Аргумент функции - переменная x.
- Числа Фибоначчи Цель: научиться решать задачи с использованием рекурсивных функций. Постановка задачи: реализовать рекурсивную функцию для вычисления n-го числа Фибоначчи.
- Цель: научиться решать задачи с использованием рекурсивных функций. Постановка задачи: Даны 1 стержень с дисками разного размера и 2 пустых стержня. Нужно переместить диски с одного стержня на другой, перекладывать можно только по одному диску за ход, складывать диски можно только меньший на больший. Реализовать программу, которая определяет перестановки этих дисков с помощью наименьшего количества ходов.

Обновление Ханойская башня - блок схема.drawio.png	 SonyAkb committed 3 hours ago	736e687		
Обновление Ханойская башня - блок схема.drawio.png	 SonyAkb committed 20 hours ago	e198d48		
Добавлено Ханойская башня - блок схема.drawio.png	 SonyAkb committed yesterday	ac5cbae		
Визуализация	 SonyAkb committed yesterday	<div>Verified</div> 72e73be		
 Commits on Feb 8, 2024				
Ханойская башня	 SonyAkb committed yesterday	<div>Verified</div> 1a4f4e2		
Смена кодировки	 SonyAkb committed yesterday	<div>Verified</div> f828dd9		
Обновление Числа Фибоначчи - блок схема.drawio.png	 SonyAkb committed yesterday	7a338f3		
Обновление Числа Фибоначчи - блок схема.drawio.png	 SonyAkb committed yesterday	37584cb		
Добавлено Числа Фибоначчи - блок схема.drawio.png	 SonyAkb committed yesterday	76442e8		
Числа Фибоначчи	 SonyAkb committed yesterday	<div>Verified</div> 18e82b4		
 Commits on Feb 1, 2024				
Отчет	 SonyAkb committed last week	<div>Verified</div> a91b3ff		
Update README.md	 SonyAkb committed last week	<div>Verified</div> 2748bb6		
Update README.md	 SonyAkb committed last week	<div>Verified</div> 32a97e8	