**Глава 14. Рекурсивные функции. Виды рекурсий**

**14.1 Понятие рекурсии**

Рекурсия — это метод программирования, при котором объект может определять себя или содержать элементы, ссылающиеся на него. Алгоритм считается рекурсивным, если в нем присутствуют условия выхода из рекурсивной функции, известные как базис рекурсии, а также шаг рекурсии, который в итоге должен привести к завершению этого алгоритма.

Рекурсивные алгоритмы не используются в случае если:

1. Ограниченная память: функция хранит локальные переменные и значения параметров. При глубокой рекурсии объем занимаемой памяти возрастает, и при переполнении стека рекурсии возникает ошибка.
2. Бесконечная рекурсия: не сработало условие выхода, ненамеренная косвенная рекурсия (перегруженные функции с одной функциональностью)
3. Слишком глубокая рекурсия
4. Слишком большие переменные

**14.2 Виды рекурсии**

Существует несколько видов рекурсии, которые различаются по способу их использования и структуре:

1. Простая (прямая) рекурсия. Функция вызывает сама себя напрямую. Этот вид рекурсии наиболее распространен.

2. Сложная (косвенная) рекурсия. В этом случае функция A вызывает функцию B, а функция B, в свою очередь, вызывает функцию A. Это может быть полезно в более сложных структурах, когда функции работают с разными аспектами одной задачи.

3. Хвостовая рекурсия. Рекурсия, в которой последний шаг функции – это вызов самой себя. Этот тип рекурсии может быть оптимизирован компилятором для использования меньшего количества ресурсов, что позволяет избежать переполнения стека.

Для понимания рассмотрены примеры, иллюстрирующие использование рекурсии.

Постановка задачи 1:

Даны два натуральных числа m и n. Реализовать программу, которая будет находить наибольший общий делитель чисел m и n.

Алгоритм решения задачи:

1. Ввод данных: Пользователь вводит значения n и m.
2. Проверка чисел на натуральность: являются ли числа m и n натуральными?
3. Если оба числа равны 0, в консоль выводится сообщение об

ошибке.

1. Если одно из чисел m или n равно 0, выводится число отличное от

нуля.

1. Выполнение алгоритма Евклида, условие выхода из цикла m == n.
2. Выводится любое из чисел m и n.

На рисунке 14.1 представлена блок-схема программы на языке C++, которая демонстрирует процесс нахождения наибольшего общего делителя (НОД) между двумя натуральными числами [2].

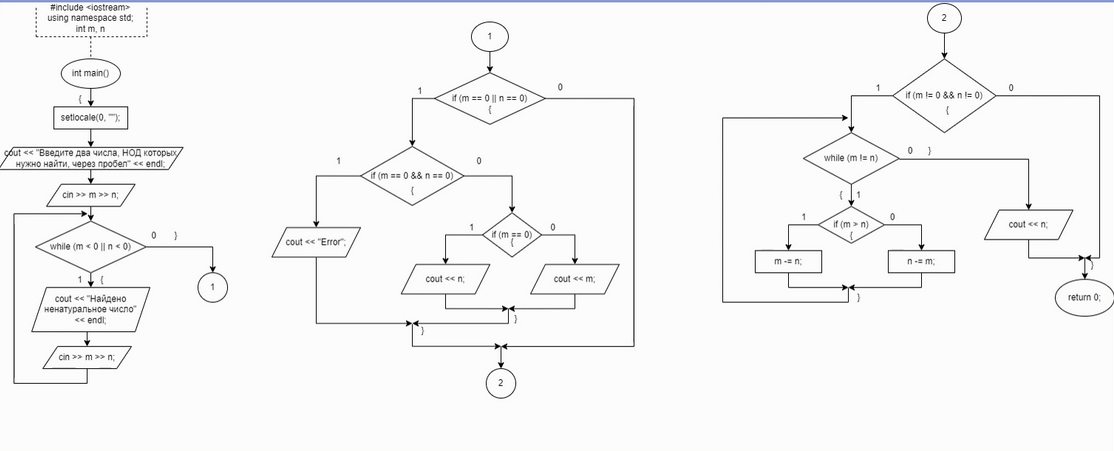


Рисунок 14.1 – Блок-схема решения задачи нахождения наибольшего общего делителя

Постановка задачи 2:

Вычислить приближенно сумму n членов ряда. Аргументы функции – n и x, где n – количество членов ряда, x – переменная. Ряд представлен на рисунке 14.2.

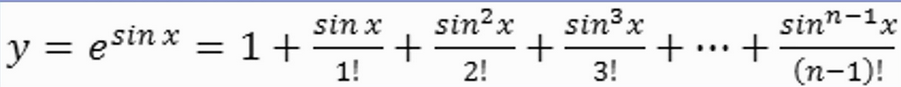


Рисунок 14.2 – Пример ряда для задачи 2

Алгоритм решения

1. Ввод данных: Пользователь вводит значения n и x. Необходимо убедиться, что n является натуральным числом, так как количество членов ряда не может быть отрицательным или равным нулю.

2. Цикл суммирования: Используется цикл, который будет проходить от 1 до n. Внутри цикла вычисляется соответствующий член ряда в зависимости от значения i и переменной x. Этот член добавляется к общей сумме.

3. После завершения цикла программа выводит окончательную сумму n членов ряда.

Решение представлено в виде блок-схемы представлено на рисунке 14.2, решение в виде кода на языке C++ представлено на рисунке 14.3.

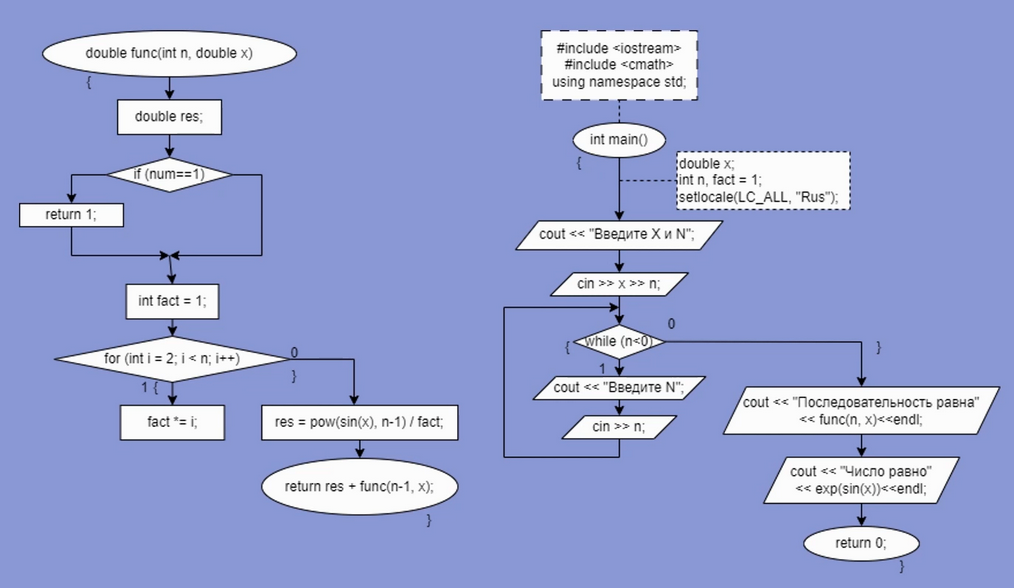


Рисунок 14.3 – Блок-схема решения задачи приближенного вычисления суммы n членов ряда

Существует два подхода к записи рекурсивных функций: префиксная и постфиксная формы.

В префиксной форме записи функции рекурсивный вызов осуществляется перед выполнением основных действий функции. Это означает, что функция сначала вызывает саму себя, а затем выполняет необходимые операции после этого вызова. Такой подход может использоваться, например, для предварительного расчетного анализа перед выполнением других операций.

В постфиксной форме записи функций сначала выполняются необходимые действия, а затем вызывается рекурсивный вызов. Это позволяет обработать результат промежуточных вычислений до того, как функция снова вызовет саму себя.

Рассмотрим пример функции, вычисляющей факториал числа с использованием хвостовой рекурсии.

Постановка задачи 3:

Дано натуральное число N. Реализовать программу, в результате выполнения которой будут выведены числа от N до 0.

Рекурсивная функция func(int num): Если аргумент num больше нуля, функция выводит значение этого аргумента и вызывает саму себя с уменьшенным значением на единицу. Это приводит к тому, что последовательность чисел от N до единицы будет выведена в консоль. Решение в виде кода на языке C++ представлено на рисунке 14.4 [3].

|  |
| --- |
|  |
| Рисунок 14.4 – Код на языке C++ для задачи 3 |

**14.3 Задача нахождения n-го числа Фибоначчи**

Числа Фибоначчи – это ряд, состоящий из целых чисел. Их особенность заключается в том, что каждый элемент представляет собой сумму двух предыдущих чисел (кроме первого и второго числа).

Последовательность Фибоначчи начинается с 0 и 1: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Алгоритм рекурсивной функции:

1. Если n = 0, вернуть 0.
2. Если n = 1, вернуть 1.
3. В противном случае вернуть сумму двух предыдущих чисел   
   F(n-1) + F(n-2).

Реализация нахождения n-ого числа Фибоначчи на языке C++ представлена в виде блок-схемы на рисунке 14.5.

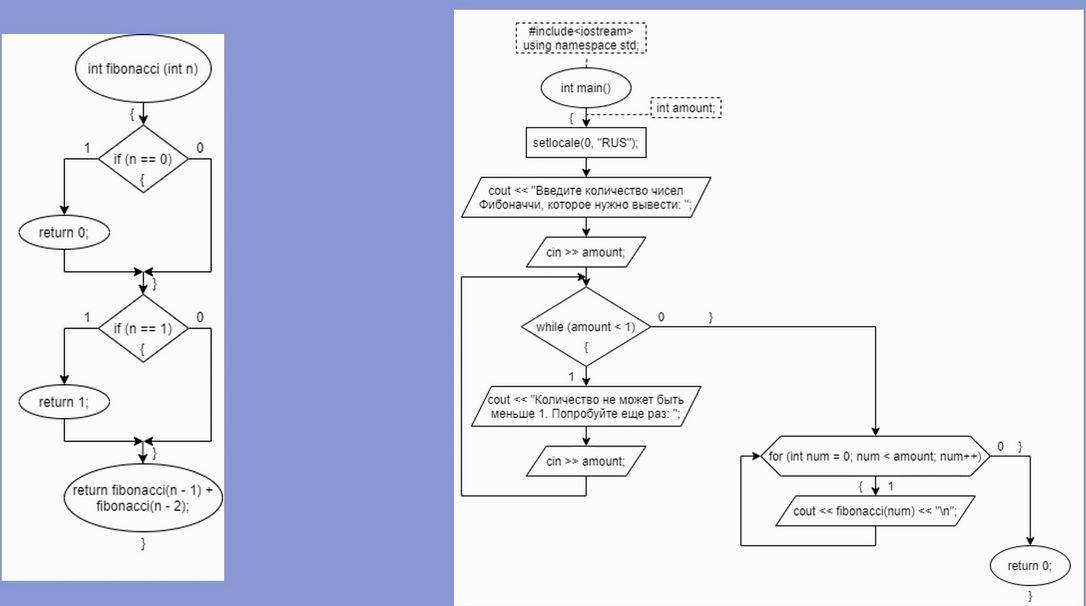


Рисунок 14.5 – Блок-схема нахождения чисел Фибоначчи

**14.4 Сложная рекурсия**

Сложная рекурсия представляет собой процесс, в рамках которого одна функция инициирует вызов другой функции, а затем та, в свою очередь, вновь вызывает первую функцию. Этот взаимный обмен вызовами может продолжаться до тех пор, пока не будет достигнуто условие завершения рекурсии.

Сигнатура функции, в свою очередь, представляет собой часть общего объявления функции, которая помогает компилятору или интерпретатору понять, что данная функция существует и может быть использована в дальнейших вычислениях или манипуляциях. Сигнатура функции содержит несколько ключевых компонентов:

1. Имя функции.
2. Число, тип и порядок следования параметров — данная информация указывает, сколько и каких параметров ожидает функция, а также, в каком порядке они должны быть переданы.
3. Тип возвращаемого значения – указывает, какой тип данных функция будет возвращать после выполнения. Это может быть любой тип, включая примитивные типы (например, целые числа, строки) или сложные структуры данных (например, списки, объекты).