ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | В. А. Кузнецов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3.1 |
| РЕКУРСИВНЫЙ АЛГОРИТМ |
| по курсу: |
| ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4326 |  |  |  | Г. С. Томчук |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#__RefHeading___Toc801_316780651)

[2 Схема алгоритма решения 4](#__RefHeading___Toc809_316780651)

[3 Полное описание реализованной функции 5](#__RefHeading___Toc807_316780651)

[4 Листинг программы 6](#__RefHeading___Toc805_316780651)

[5 Результаты тестирования программы и функции 7](#__RefHeading___Toc803_316780651)

[6 Вывод по результату тестирования 10](#__RefHeading___Toc801_3167806512)

1. Постановка задачи

Задача: реализовать программную функцию на языке C/С++, выполняющую поставленную задачу. Вариант задания, пример входных и выходных данных представлен в таблице 1.

* Написать код функции, принимающей в качестве аргументов и возвращающей все необходимые параметры, без использования глобальных переменных. Допустимо использование дополнительных функций.
* Протестировать функцию для всех возможных исключительных ситуаций, особое значение придается тестам на возникновение ошибок в ходе работы программы.
* Из наименования функции и принимаемых аргументов должно быть ясно их назначение.
* В ходе тестирования функции при каждом вызове рекурсивной функции необходимо вывести отладочную информацию: порядковый номер вызова рекурсивной функции, значения изменяющегося аргумента и возвращаемого значения, если они присутствуют. Привести глубину рекурсии для каждого тестового примера.

Таблица 1 – Вариант

|  |  |  |  |
| --- | --- | --- | --- |
| N | Текст задания | Вход | Выход |
| 7 | Реализовать рекурсивную функцию вычисления (ax + b)((a - 1) x + b - 1)...((a - k ) x + b - k )...( x + 1) по заданным a,b,x. При условии, что  если a-k< 1,то a-k = 1,  если b-k<1,то b-k = 1. | 2, 3, 1 | 30 |

1. Схема алгоритма решения

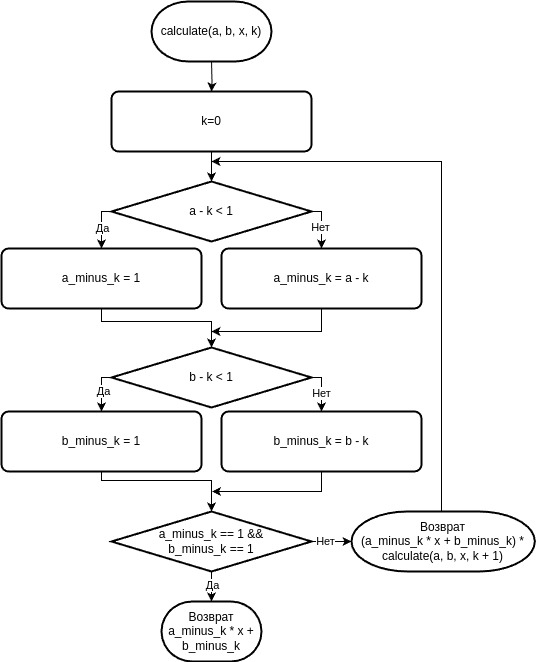


Рисунок 1 – Блок-схема алгоритма

1. Полное описание реализованной функции

Функция calculate\_recursion\_with\_debug рекурсивно вычисляет значение выражения (ax + b)((a - 1) x + b - 1)...((a - k ) x + b - k )...( x + 1) с учетом заданных значений a, b, и x. При этом, если значение выражения (a−k) или (b−k) становится меньше 1, то оно заменяется на 1. Функция также выводит отладочную информацию о каждом вызове. Принимает следующие аргументы:

1. int a — начальное значение параметра a.
2. int b — начальное значение параметра b.
3. int x — начальное значение параметра x.
4. int &call\_count — ссылка на переменную, отслеживающую количество вызовов функции.
5. int k — текущий шаг рекурсии (по умолчанию 0).

Возвращает int64\_t — результат вычисления выражения. Работа функции происходит следующим образом:

1. Увеличиваем call\_count и сохраняем его значение в current\_call\_count.
2. Вычисляем текущие значения (a - k) и (b - k). Если одно из значений меньше 1, заменяем его на 1.
3. Выводим отладочную информацию, включающую текущий номер вызова, значение k, и текущие значения (a - k) и (b - k).
4. Проверяем базовый случай: если (a - k) == 1 и (b - k) == 1, вычисляем результат как 1 \* x + 1, выводим отладочную информацию и возвращаем результат.
5. Если базовый случай не достигнут, вычисляем рекурсивно выражение для следующего шага k + 1 и умножаем его на текущее выражение (a - k) \* x + (b - k).
6. Выводим отладочную информацию о возвращаемом значении и возвращаем результат.
7. Листинг программы

Листинг 1

#include <iostream>  
  
int64\_t calculate\_recursion\_with\_debug(int a, int b, int x, int &call\_count, int k = 0) {  
 // Увеличиваем номер вызова и сохраняем значение  
 int current\_call\_count = ++call\_count;  
  
 // Вычисляем текущие значения (a - k) и (b - k)  
 int a\_minus\_k = (a - k < 1) ? 1 : (a - k);  
 int b\_minus\_k = (b - k < 1) ? 1 : (b - k);  
  
 // Вывод отладочной информации  
 std::cout << "Вызов #" << current\_call\_count << ": k=" << k << ", a-k=" << a\_minus\_k << ", b-k="  
 << b\_minus\_k << std::endl;  
  
 // Базовый случай: прекращаем рекурсию  
 if (a\_minus\_k == 1 && b\_minus\_k == 1) {  
 int64\_t result = a\_minus\_k \* x + b\_minus\_k;  
 std::cout << "Возврат из вызова #" << current\_call\_count << ": " << result << std::endl;  
 return result;  
 }  
  
 // Рекурсивный случай: продолжаем умножение  
 int64\_t result = (a\_minus\_k \* x + b\_minus\_k) \* calculate\_recursion\_with\_debug(a, b, x, call\_count, k + 1);  
 std::cout << "Возврат из вызова #" << current\_call\_count << ": " << result << std::endl;  
  
 return result;  
}  
  
int main() {  
 int a, b, x;  
  
 std::cout << "a: ";  
 std::cin >> a;  
  
 std::cout << "b: ";  
 std::cin >> b;  
  
 std::cout << "x: ";  
 std::cin >> x;  
  
 int call\_count = 0;  
  
 int64\_t result = calculate\_recursion\_with\_debug(a, b, x, call\_count);  
 std::cout << "Результат: " << result << ", глубина: " << call\_count;  
  
 return 0;  
}

1. Результаты тестирования программы и функции

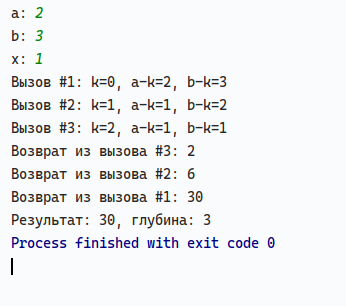


Рисунок 2

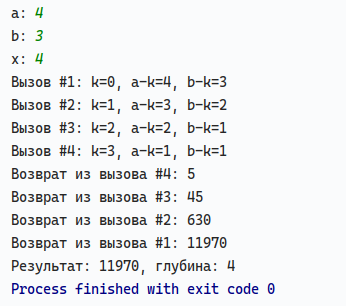


Рисунок 3

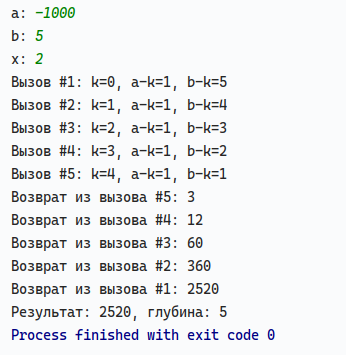


Рисунок 4

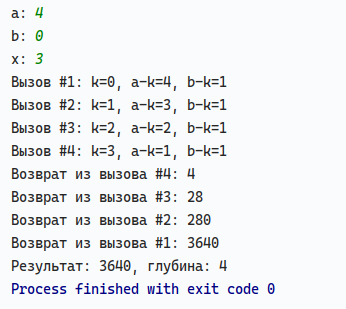


Рисунок 5

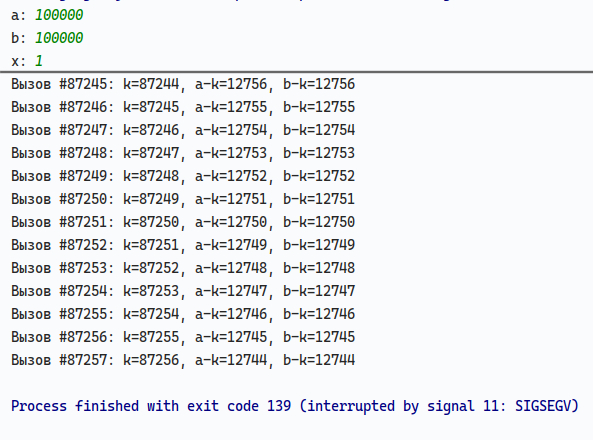


Рисунок 6

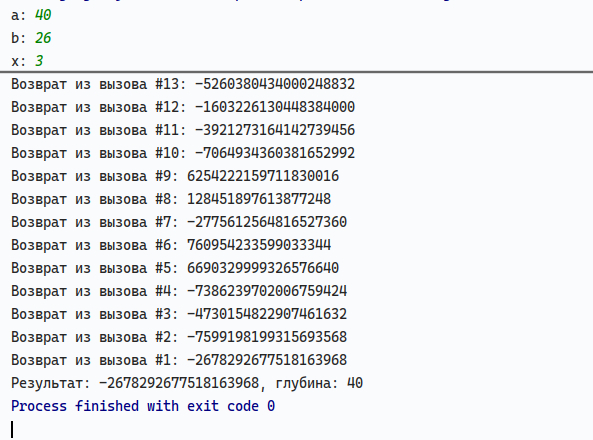


Рисунок 7

1. Вывод по результату тестирования

**Нормальные значения.** Входные данные: a = 2, b = 3, x = 1. Ожидаемый результат: 30. Фактический результат: 30. Глубина рекурсии: 3. Функция корректно вычисляет результат для нормальных значений (рис. 2, 3).

**Отрицательные и нулевые значения.** Входные данные: a = -1000, b = 5, x = 2. Фактический результат: 2520. Глубина рекурсии: 5. При значениях a, b < 2 значение (a-k) или (b-k) сразу становится ≤ 1, соответственно приравниваются 1 и вычисления происходят обычным образом (рис. 4, 5).

**Большие значения.** Функция сталкивается с переполнением стека (рис. 6) или переполнением целочисленных типов (рис. 7) при больших значениях. Чтобы избежать переполнения стека, стоит использовать итеративный метод для решения задачи, или же искусственно ограничить глубину рекурсии. Для работы с большими значениями потребуется использование специальных библиотек для длинной арифметики.

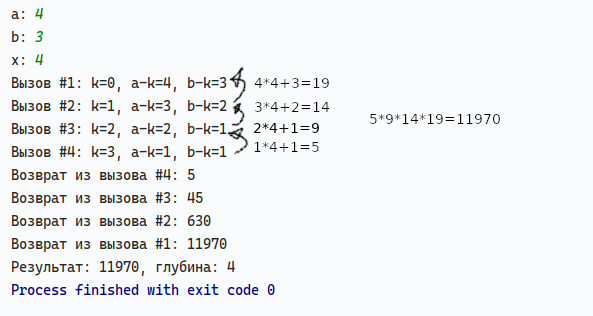


Рисунок 8

Из рис. 8 видно, что функция алгоритмически корректно решает задачу (при условии, что входные значения не слишком велики).