**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе № 3 на тему:   
**«Понятие рекурсии. Рекурсивные функции»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Серафимович Г. П.

«02» октября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ Серафимович Г. П./

Принял: к.ф.-м.н., доцент, И.А. Моисеев

«02» октября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Основная часть**

**Цель работы:**

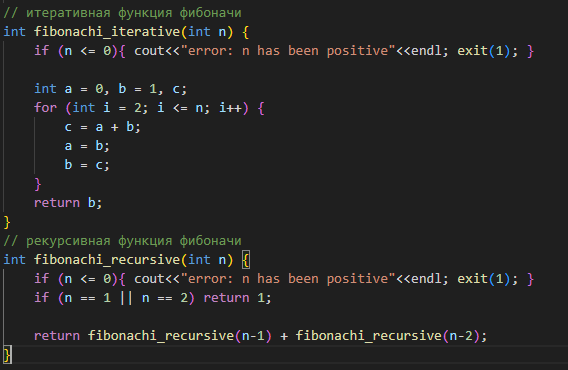
Изучить понятие рекурсии, рекурсивные функции в программировании, приемы построения рекурсивной функции при решении задач, научиться применять рекурсивные методы в решении задач на языке С++.

**Результаты выполнения работы:**

В данной работе были реализованы следующие методы:

1. Итеративные и рекурсивные методы фибоначи
2. Итеративные и рекурсивные методы сочетаний
3. Итеративные и рекурсивные методы максимального делителя
4. Вспомогательные и главные методы высчитывающая сумму кубов цифр
5. Главный метод суммы кубов

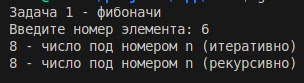
Были реализованы методы фибоначи(итеративный и рекурсивный, Рисунок 1.1)



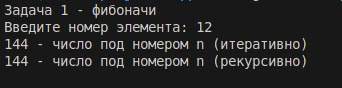
**Рисунок 1.1 Метода фибоначи итеративно и рекурсивно**

В итеративном методе для начала проверяется чтобы порядковый номер цепочки фибоначи был положительным. Затем идет цикл который идет ровно до порядкового номера n, в нем с каждой итерацией определяется новое значение числа, так, чтобы оно представляло из себя сумму двух предыдущих значений. В рекурсивном методе аналогично, разве что благодаря рекурсии он строит некое дерево значений, причем таких чтобы каждое число представляло из себя сумму двух предыдущих.

Вывод методов фибоначи представлен на рисунке 1.2

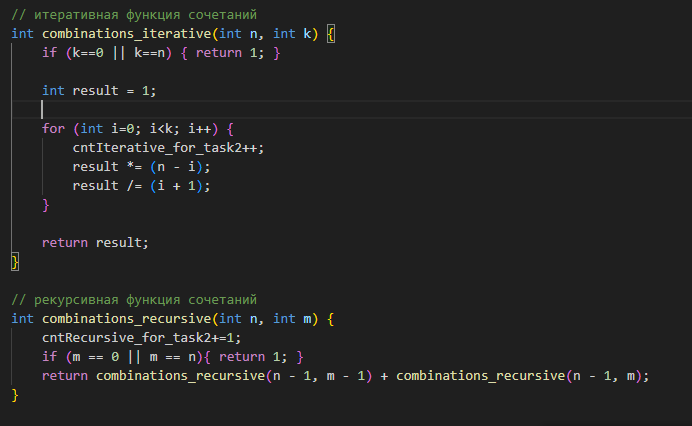


**Рисунок 1.2. Вывод метода фибоначи итеративно и рекурсивно**



**Рисунок 1.3. Вывод метода фибоначи итеративно и рекурсивно**

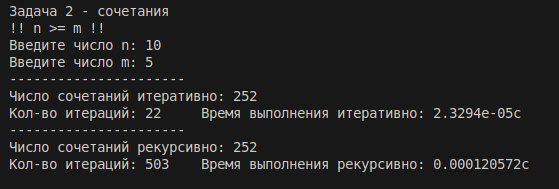
Были реализованы методы сочетаний(итеративный и рекурсивный, Рисунок 2.1)



**Рисунок 2.1 Метод сочетаний итеративно и рекурсивно**

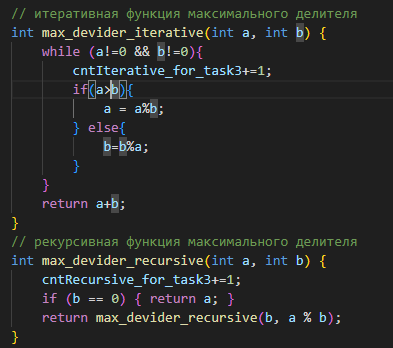
В итеративной функции для вычисления числа сочетаний. Если k равно 0 или n, функция сразу возвращает 1, так как в этих случаях количество сочетаний равно 1. В цикле, который выполняется k раз, происходит умножение на оставшиеся элементы и деление на количество выбранных элементов. Это позволяет постепенно вычислить значение сочетания без использования рекурсии. В конце функция возвращает итоговое значение сочетания.В рекурсивном методе для начала увеличивается счетчик отвечающий за кол-во итераций, затем рекурсия возвращает сама себя для того чтобы удовлетворять формуле сочетаний.

Вывод методов сочетаний представлен на рисунке 2.2



**Рисунок 2.2 Вывод метода сочетаний итеративно и рекурсивно**

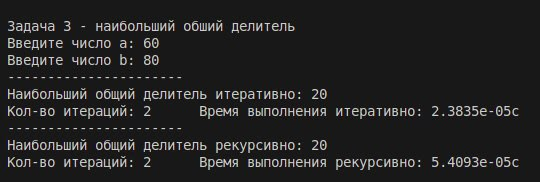
Были реализованы методы максимального делителя(итеративный и рекурсивный, рисунок 3.1)



**Рисунок 3.1 Метод максимального делителя итеративно и рекурсивно**

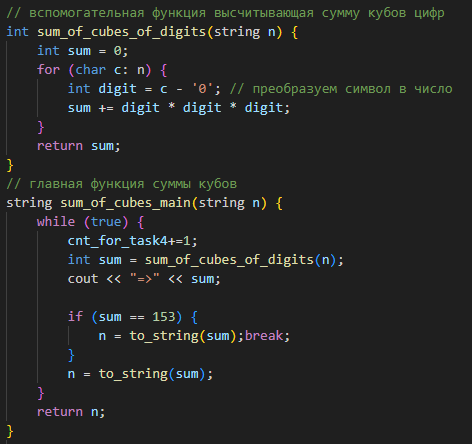
В итеративной функции чуть чуть сложноватый цикл while, который благодаря обновлению переменных на остаток деления их же на другую переменную возвращает их максимальный общий делитель. В рекурсивной функции реализован точно такой же алгоритм только для рекурсии, важное отличие-рекурсивная функция вызывает саму себя и каждый раз меняет переменный a и b местами. В обоих функциях ведется счет кол-ва итераций.

Вывод методов максимального делителя представлен на рисунке 3.2



**Рисунок 3.2 Вывод метода максимального делителя итеративно и рекурсивно**

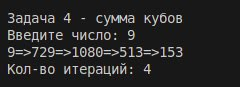
Были реализованы методы суммы кубов(вспомогательный и главный, Рисунок 4.1)



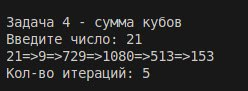
**Рисунок 4.1 Вспомогательный и главный метод суммы кубов**

В вспомогательном методе считается сумма цифр в числе благодаря преобразованиям из char в int. В главной функции запускается бесконечный цикл, в котором также считается кол-во итераций и выводится сумма кубов числа до тех пор, пока она не будет равна 153.

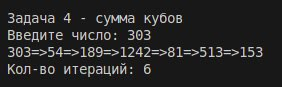
Вывод методов суммы кубов представлен на рисунке 4.2



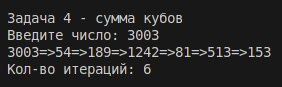
**Рисунок 4.2 Вывод вспомогательного и главного метода суммы кубов**



**Рисунок 4.3 Вывод вспомогательного и главного метода суммы кубов**

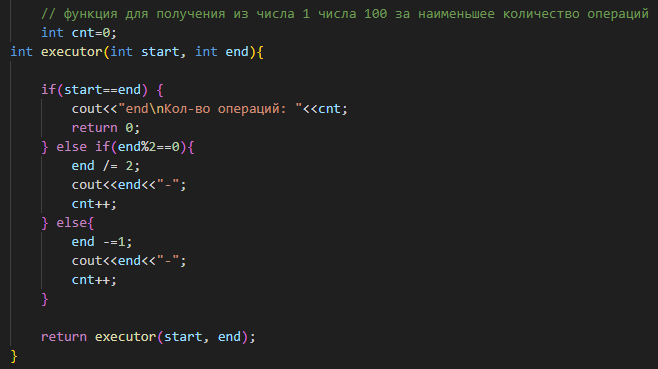


**Рисунок 4.4 Вывод вспомогательного и главного метода суммы кубов**



**Рисунок 4.5 Вывод вспомогательного и главного метода суммы кубов**

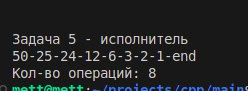
Был реализован метод исполнителя(Рисунок 5.1)



**Рисунок 5.1 Метод для получения из числа 1 числа 100 за наименьшее количество операций**

Ведется счетчик, а также простая логика функция, которая гласит делить число на 2 если оно четное, в противном случае отнимать 1, такой метод как бы обратен задаче, но благодаря нему можно узнать самый короткий маршрут.

Вывод метода исполнителя представлен на рисунке 5.2



**Рисунок 5.2 Вывод метода для получения из числа 1 числа 100 за наименьшее количество операций**

**Выводы:**

Я многое понял из реализации данной практической работы, разработанная программа успешно решает поставленные задачи, демонстрируя различные подходы к вычислениям. Эффективность рекурсивных и итеративных методов зависит от задачи, соответственно время выполнения тоже. **Приложение**

Листинг программы:

#include <iostream>

#include <chrono>

using namespace std;

// счетчики для счета кол-ва итераций

int cntIterative\_for\_task2, cntRecursive\_for\_task2, cntIterative\_for\_task3, cntRecursive\_for\_task3, cnt\_for\_task4;

// итеративная функция фибоначи

int fibonachi\_iterative(int n) {

if (n <= 0){ cout<<"error: n has been positive"<<endl; exit(1); }

int a = 0, b = 1, c;

for (int i = 2; i <= n; i++) {

c = a + b;

a = b;

b = c;

}

return b;

}

// рекурсивная функция фибоначи

int fibonachi\_recursive(int n) {

if (n <= 0){ cout<<"error: n has been positive"<<endl; exit(1); }

if (n == 1 || n == 2) return 1;

return fibonachi\_recursive(n-1) + fibonachi\_recursive(n-2);

}

// итеративная функция сочетаний

int combinations\_iterative(int n, int k) {

if (k==0 || k==n) { return 1; }

int result = 1;

for (int i=0; i<k; i++) {

cntIterative\_for\_task2++;

result \*= (n - i);

result /= (i + 1);

}

return result;

}

// рекурсивная функция сочетаний

int combinations\_recursive(int n, int m) {

cntRecursive\_for\_task2+=1;

if (m == 0 || m == n){ return 1; }

return combinations\_recursive(n - 1, m - 1) + combinations\_recursive(n - 1, m);

}

// итеративная функция максимального делителя

int max\_devider\_iterative(int a, int b) {

while (a!=0 && b!=0){

cntIterative\_for\_task3+=1;

if(a>b){

a = a%b;

} else{

b=b%a;

}

}

return a+b;

}

// рекурсивная функция максимального делителя

int max\_devider\_recursive(int a, int b) {

cntRecursive\_for\_task3+=1;

if (b == 0) { return a; }

return max\_devider\_recursive(b, a % b);

}

// вспомогательная функция высчитывающая сумму кубов цифр

int sum\_of\_cubes\_of\_digits(string n) {

int sum = 0;

for (char c: n) {

int digit = c - '0'; // преобразуем символ в число

sum += digit \* digit \* digit;

}

return sum;

}

// главная функция суммы кубов

string sum\_of\_cubes\_main(string n) {

while (true) {

cnt\_for\_task4+=1;

int sum = sum\_of\_cubes\_of\_digits(n);

cout << "=>" << sum;

if (sum == 153) {

n = to\_string(sum);break;

}

n = to\_string(sum);

}

return n;

}

// функция для получения из числа 1 числа 100 за наименьшее количество операций

int cnt=0;

int executor(int start, int end){

if(start==end) {

cout<<"end\nКол-во операций: "<<cnt;

return 0;

} else if(end%2==0){

end /= 2;

cout<<end<<"-";

cnt++;

} else{

end -=1;

cout<<end<<"-";

cnt++;

}

return executor(start, end);

}

int main() {

// объявление переменных

int n, m;

string n\_str, command1 = "+1", command2 = "\*2";

// задача 1

cout << "Задача 1 - фибоначи\nВведите номер элемента: ";

cin >> n;

cout << fibonachi\_iterative(n) << " - число под номером n (итеративно)" <<endl;

cout << fibonachi\_recursive(n) << " - число под номером n (рекурсивно)" <<endl<<endl<< endl << endl;

// задача 2

cout << "Задача 2 - сочетания\n!! n >= m !!\nВведите число n: ";

cin >> n;

cout << "Введите число m: ";

cin >> m;

// для подсчета времени(это старт рекурсивного и итеративного таймера)

auto timeIterative\_start\_task2 = chrono::high\_resolution\_clock::now();

auto timeRecursive\_start\_task2 = chrono::high\_resolution\_clock::now();if (n < m) { cout << "error: n должно быть меньше m." << endl; return 1; }

cout << "----------------------\nЧисло сочетаний итеративно: " << combinations\_iterative(n, m);

cout << "\nКол-во итераций: " << cntIterative\_for\_task2;

// для подсчета времени(это конец итеративного таймера)

auto timeIterative\_end\_task2 = chrono::high\_resolution\_clock::now();

// для подсчета времени(это старт вычисление продолжительности итеративного таймера)

chrono::duration<double> durationIterative = timeIterative\_end\_task2 - timeIterative\_start\_task2;

cout << "\tВремя выполнения итеративно: " << durationIterative.count() << "c" << endl;

cout << "----------------------\nЧисло сочетаний рекурсивно: " << combinations\_recursive(n, m);

cout << "\nКол-во итераций: " << cntRecursive\_for\_task2;

// для подсчета времени(это конец рекурсивного таймера)

auto timeRecursive\_end\_task2 = chrono::high\_resolution\_clock::now();

// для подсчета времени(это старт вычисление продолжительности рекурсивного таймера)

chrono::duration<double> durationRecursive\_task2 = timeRecursive\_end\_task2 - timeRecursive\_start\_task2;

cout << "\tВремя выполнения рекурсивно: " << durationRecursive\_task2.count() << "c" << endl << endl << endl << endl;

// задача 3

cout << "Задача 3 - наибольший обший делитель\nВведите число a: ";

cin >> n;

cout << "Введите число b: ";

cin >> m;

// для подсчета времени(это старт рекурсивного и итеративного таймера)

auto timeIterative\_start\_task3 = chrono::high\_resolution\_clock::now();

auto timeRecursive\_start\_task3 = chrono::high\_resolution\_clock::now();

cout<<"----------------------\nНаибольший общий делитель итеративно: " << max\_devider\_iterative(n,m);

cout << "\nКол-во итераций: " << cntIterative\_for\_task3;

// для подсчета времени(это конец итеративного таймера)

auto timeIterative\_end\_task3 = chrono::high\_resolution\_clock::now();

// для подсчета времени(это старт вычисление продолжительности итеративного таймера)

chrono::duration<double> durationIterative\_task3 = timeIterative\_end\_task3 - timeIterative\_start\_task3;

cout << "\tВремя выполнения итеративно: " << durationIterative\_task3.count() << "c" << endl;

cout<<"----------------------\nНаибольший общий делитель рекурсивно: " << max\_devider\_recursive(n,m);

cout << "\nКол-во итераций: " << cntIterative\_for\_task3;

// для подсчета времени(это конец рекурсивного таймера)

auto timeRecursive\_end\_task3 = chrono::high\_resolution\_clock::now();

// для подсчета времени(это старт вычисление продолжительности рекурсивного таймера)

chrono::duration<double> durationRecursive\_task3 = timeRecursive\_end\_task3 - timeRecursive\_start\_task3;

cout << "\tВремя выполнения рекурсивно: " << durationRecursive\_task3.count() << "c" << endl << endl<< endl << endl;

// analiz

// задача 4

cout << "Задача 4 - сумма кубов\nВведите число: ";

cin >> n\_str;

cout<<n\_str;

sum\_of\_cubes\_main(n\_str);

cout << "\nКол-во итераций: " << cnt\_for\_task4 <<endl<< endl << endl;

// зависимость чем больше цифр в числе тем больше кол-во итераций

// задача 5

cout<< "Задача 5 - исполнитель\n";

// старт и конец исполнителя

int start = 1, end = 100;

executor(start, end);

cout<<endl;

return 0;

}