

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

**Отчет по выполнению практического задания номер семь**

**Тема: Рекурсивные функции**

**Дисциплина: Структуры и алгоритмы обработки данных**

Выполнил студент Михайлюк Д. С.

Группы ИНБО-07-21

Москва 2022

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ 3](#_Toc100793787)

[2 Определение списка операций над списком, которые выявлены в процессе исследования задач дополнительного задания 4](#_Toc100793788)

[2.1 Проверка списка на пустоту 4](#_Toc100793789)

[2.2 Метод добавления узла в начало 5](#_Toc100793790)

[2.3 Метод добавления узла в конец 7](#_Toc100793791)

[2.4 Метод вывода списка на экран 9](#_Toc100793792)

[2.5 Метод поиска узлов по значению 11](#_Toc100793793)

[2.6 Метод удаления первого узла 12](#_Toc100793794)

[2.7 Метод удаления последнего узла 14](#_Toc100793795)

[2.8 Метод удаления узла по его значению 16](#_Toc100793796)

[2.9 Метод перемещения первых n-узлов в конец списка 19](#_Toc100793797)

[2.10 Метод удаления предпоследнего узла 21](#_Toc100793798)

[2.11 Метод обмена местами 2х узлов 23](#_Toc100793799)

[2.12 Метод обмена местами узлов с наибольшим и наименьшим значениями 25](#_Toc100793800)

[3 КОД ВСЕЙ ПРОГРАММЫ 27](#_Toc100793801)

[4 ВЫВОДЫ 32](#_Toc100793802)

[5 ОТВЕТЫ НА ВОПРОСЫ 33](#_Toc100793803)

# ПОСТАНОВКА ЗАДАЧИ

Разработать и протестировать рекурсивные функции в соответствии с задачами варианта

Персональный вариант - №2:

Реализовать задачи:

1. Найти n-ое число Фибоначчи.

2. В однонаправленном списке из n элементов найти элемент с заданным значением и вернуть на него указатель.

# Задание – 1

Найти n-ое число Фибоначчи.

## Описание алгоритма – рекуррентная зависимость

## Коды используемых функций

fibo(int a)

{

if (a == 1 or a == 2)

{

return 1;

}

a = fibo(a - 1) + fibo(a - 2);

return a;

}

Код функции нахождения N-ого числа фибоначи

## Ответы на задания по задаче 1: список требований к задаче 1

### приведите итерационный алгоритм решения задачи



Рисунок 1 – Итерационная схема работы рекурсивного алгоритма задания 1

### Определите теоретическую сложность алгоритма

У рекурсивного алгоритма поиска n-ого числа Фибоначчи нет явной сложности,

Неявная же сложность имеет квадратичный вид - **O(n2 )**

### Определите глубину рекурсии, изменяя исходные данные

Таблица 1 – Глубины рекурсий для различных значений n

|  |  |
| --- | --- |
| n | Глубина рекурсии |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 4 |
| 10 | 9 |

В общем случае глубина рекурсии для произвольного n:

если n != 1, в противном случае :

### определите сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии

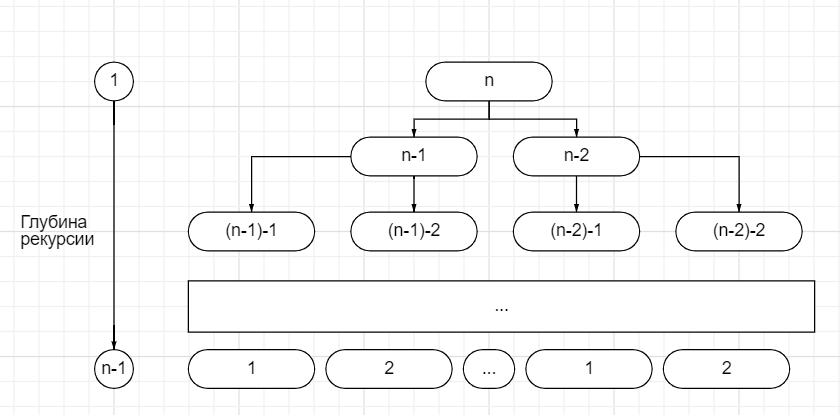
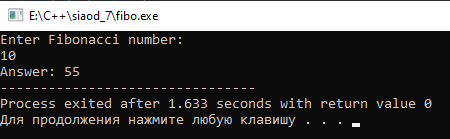


Рисунок 2 – Дерево рекурсии алгоритма поиска n-ого числа Фибоначчи

### Результаты тестирования



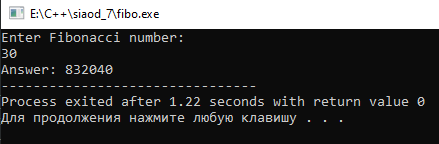


Рисунок 3 – Результаты тестирования программы

## Код программы

#include <iostream>

using namespace std;

int fibo(int a)

{

if (a == 1 or a == 2)

{

return 1;

}

a = fibo(a - 1) + fibo(a - 2);

return a;

}

int main()

{

cout << "Enter Fibonacci number:\n";

int a;

cin >> a;

cout << "Answer: " << fibo(a);

}

Полный код программы поиска чисел Фибоначчи

# Задание – 2

В однонаправленном списке из n элементов найти элемент с заданным значением и вернуть на него указатель

## Описание алгоритма – рекуррентная зависимость

## Коды используемых функций

struct Node

{

int val;

Node\* next;

Node(int \_val) : val(\_val), next(nullptr) {}

};

Структура узла в списке

struct list

{

Node\* first;

Node\* last;

list() : first(nullptr), last(nullptr) {}

}

Структура списка с конструктором

bool is\_empty()

{

return first == nullptr;

}

Метод проверки узла на Пустоту

void print()

{

if (is\_empty()) return;

Node\* p = first;

while (p) {

cout << p->val << " ";

p = p->next;

}

cout << endl;

}

Метод вывода узла на экран

void push\_back(int \_val)

{

Node\* p = new Node(\_val);

if (is\_empty())

{

first = p;

last = p;

return;

}

last->next = p;

last = p;

}

Метод добавления нового узла в конец списка

Node\* getfirst()

{

return first;

}

Геттер указателя на первый узел спеиска

Node\* find(int \_val, Node\* curr)

{

if (curr->val != \_val)

{

curr = curr->next;

find(\_val, curr);

}

else

{

return curr;

}

}

Рекурсивный метод поиска узла по значению

## Ответы на задания по задаче 2: список требований к задаче 2

### Определите глубину рекурсии



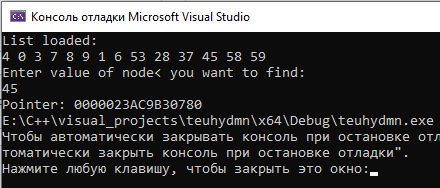
Рисунок 4 - Дерево рекурсии алгоритма поиска узла по значению

Из схемы очевидно, что глубина рекурсии в худшем случае ровна количеству узлов в списке – n, обычно меньше.

### Определите теоретическую сложность алгоритма

Алгоритм линейно перебирает все узлы списка, пока не найдет узел с нужным значением информационной части узла, следовательно сложность алгоритма так же будет линейной:

### Результаты тестов



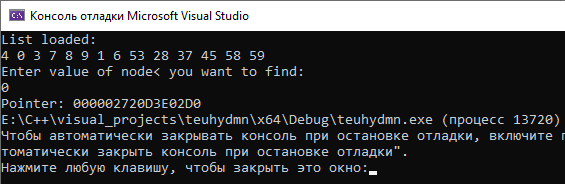


Рисунок 5 – Результаты работы программы

## Код программы

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node

{

int val;

Node\* next;

Node(int \_val) : val(\_val), next(nullptr) {}

};

struct list

{

Node\* first;

Node\* last;

list() : first(nullptr), last(nullptr) {}

bool is\_empty()

{

return first == nullptr;

}

void print()

{

if (is\_empty()) return;

Node\* p = first;

while (p) {

cout << p->val << " ";

p = p->next;

}

cout << endl;

}

void push\_back(int \_val)

{

Node\* p = new Node(\_val);

if (is\_empty())

{

first = p;

last = p;

return;

}

last->next = p;

last = p;

}

Node\* getfirst()

{

return first;

}

Node\* find(int \_val, Node\* curr)

{

if (curr->val != \_val)

{

curr = curr->next;

find(\_val, curr);

}

else

{

return curr;

}

}

};

int main()

{

list l;

ifstream fe("List.txt");

string st;

float a;

while (getline(fe, st))

{

a = stof(st);

l.push\_back(a);

}

cout << "List loaded: \n";

l.print();

cout << "Enter value of node< you want to find: \n";

cin >> a;

cout << "Pointer: " << l.find(a, l.getfirst());

}