МИНИСТЕРСТВО НАУКИ ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«**МИРЭА - Российский технологический университет**»

РТУ МИРЭА

Отчет по выполнению практического задания №7
Тема:
«Рекурсивные алгоритмы и их реализация»
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Подоплелов А.С.

Группа: ИНБО-21-23

Москва - 2024

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Цель работы: получить знания и практические навыки по разработке и реализации рекурсивных алгоритмов.

1.1 УСЛОВИЕ ЗАДАНИЯ

Разработать и протестировать рекурсивные функции в соответствии с задачами варианта.

Мой индивидуальный вариант 7:

1. Найти максимальный элемент в массиве из n элементов.

2. Создание очереди на однонаправленном списке.

1.2 ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ

1) Требования к выполнению первой задачи варианта:

- приведите итерационный алгоритм решения задачи;

- реализуйте алгоритм в виде функции и отладьте его;

- определите теоретическую сложность алгоритма;

- опишите рекуррентную зависимость в решении задачи;

- реализуйте и отладьте рекурсивную функцию решения задачи;

- определите глубину рекурсии, изменяя исходные данные;

- определите сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии;

- приведите для одного из значений схему рекурсивных вызовов;

- разработайте программу, демонстрирующую выполнение обеих функций и покажите результаты тестирования.

2) Требования к выполнению второй задачи варианта:

- рекурсивную функцию для обработки структуры списка согласно варианту. Информационная часть узла – простого типа – целого;

- для создания списка может быть разработана простая или рекурсивная функция по желанию (в тех вариантах, где не требуется рекурсивное создание списка);

- определите глубину рекурсии;

- определите теоретическую сложность алгоритма;

- разработайте программу, демонстрирующую работу функций и покажите результаты тестов.

3) Составить отчет по выполненному заданию.

РЕШЕНИЕ ЗАДАНИЯ 1

2.1 ИТЕРАЦИОННЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

Цель: разработать алгоритм поиска максимального элемента в массиве из n элементов с помощью языка C++.

Данный алгоритм будет состоять из:

- одного цикла «for», с помощью которого будет происходить проход по массиву;

- инициализации переменной «max», которой изначально будет присвоено значение элемента заданного массива с индексом «0».

- результатом работы будет возврат целочисленной переменной «max».

Представим блок-схема алгоритма «Поиск максимального элемента в массиве из n элементов» на рисунке 1.

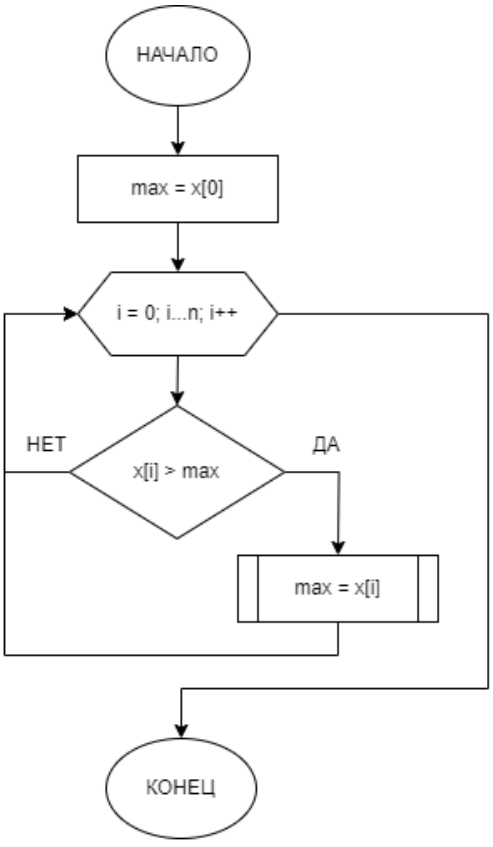
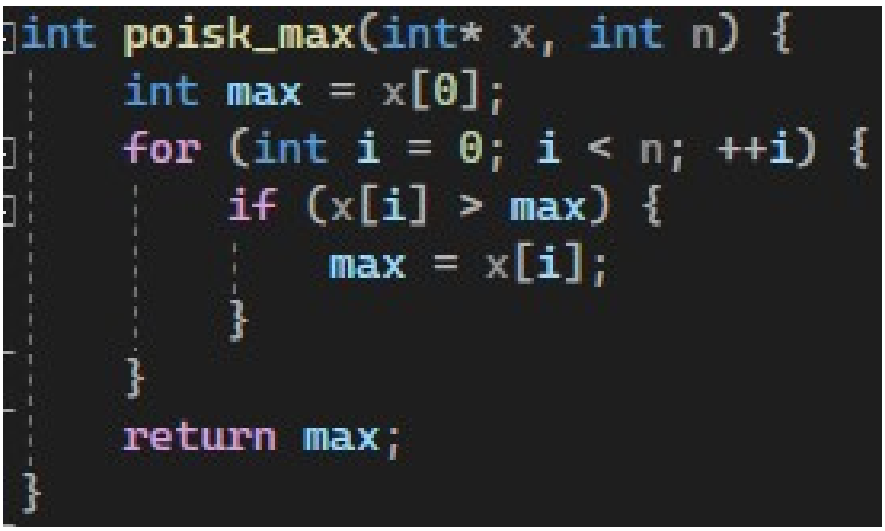


Рисунок 1 — Блок-схема алгоритма «Поиск максимального элемента в массиве из n элементов»

Реализуем данный алгоритм с помощью языка C++ (См. Рисунок 2).



```
int poisk_max(int* x, int n) {
    int max = x[0];
    for (int i = 0; i < n; ++i) {
        if (x[i] > max) {
            max = x[i];
        }
    }
    return max;
}
```

Рисунок 2 — Реализация алгоритма «Поиск максимального элемента в массиве из n элементов» с помощью языка C++

2.2 ТЕОРЕТИЧЕСКАЯ СЛОЖНОСТЬ АЛГОРИТМА

Единственный способ точно найти самое большое число в случайном массиве будет перебор каждого числа в поисках максимума. Поэтому сложность такого алгоритма – O(n).

Таблица 1 — Теоретическая сложность алгоритма.

n	**T(n), мс**	**$T_п(n) = C_ф + M_ф$**
100	0	3500
1000	0	35000
10_000	0	350000
100_000	1	3500000
1_000_000	4	35000000

2.3 РЕАЛИЗАЦИЯ И ОТЛАДКА РЕКУРСИВНОЙ ФУНКЦИИ

Перейдем к рекурсивной части реализации данного алгоритма.

Рекурсивная зависимость будет выражается в том, что пока длина массива, который мы передаем в качестве параметра рекурсивной функции, не будет равна 1, то рекурсивная функция не прекратит свою работу.

Реализуем рекурсивную функцию алгоритма «Поиск максимального элемента в массиве из n элементов» с помощью языка C++ (См. Рисунок 3).

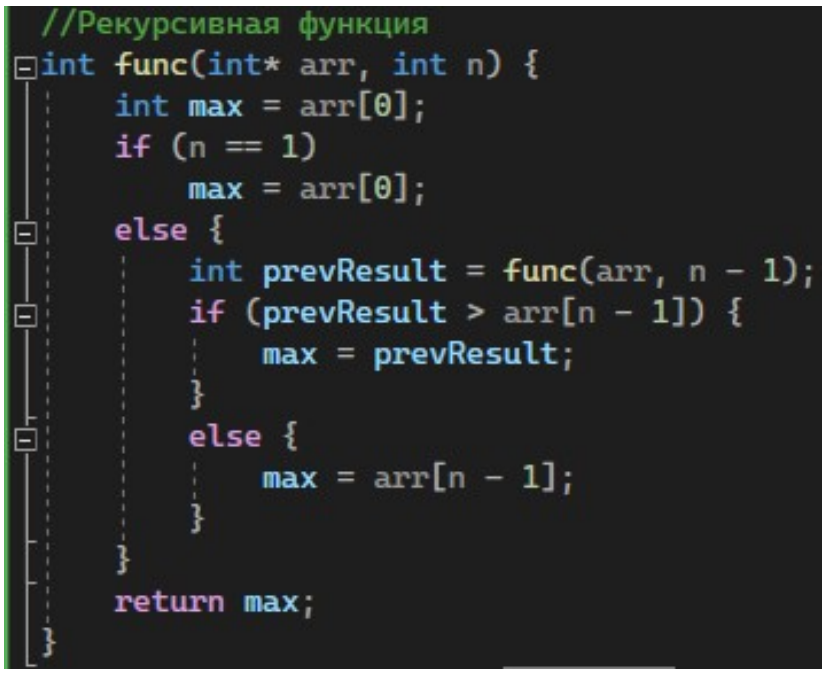


Рисунок 2 — Реализация рекурсивной функции алгоритма «Поиск максимального элемента в массиве из n элементов» с помощью языка C++

2.4 ГЛУБИНА РЕКУРСИИ И СЛОЖНОСТЬ РЕКУРСИВНОГО АЛГОРИТМА

Глубина рекурсии - это количество раз, которое функция вызывает саму себя в процессе выполнения.

Определим глубину рекурсии нашего алгоритма с помощью добавления счётчика в нашу рекурсивную функцию в то место, где происходит вход в функцию с измененным параметром (См. Таблица 2).

Таблица 2 — Глубина рекурсии при различных входных данных

n	**Глубина рекурсии**
100	110
1000	1070
10_000	13700
100_000	Невозможно измерить
1_000_000	Невозможно измерить

Сложность рекурсивного алгоритма будет вычисляться по формуле $O(n*\log_n n)$.

Приведем схему рекурсивных вызовов, при n = 10 (См. Рисунок 3).

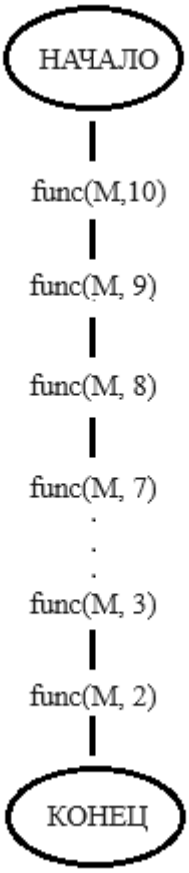


Рисунок 3— Реализация схемы рекурсивных вызовов, при n = 10

2.5 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Реализуем основную подпрограмму, где будет осуществляться выполнение двух функций для поиска максимального элемента массива и проведем тесты, при n = 10, n = 100, n = 1000.

```
int main() {
    srand(time(NULL));
    cout << "Start!" << "\n";
    int n;
    cin >> n;
    int* M = new int[n];
    for (int i = 0; i < n; i++) {
        M[i] = rand();
    }
    print(M, n);
    cout << "MAX1 = " << poisk_max(M, n) << "\n";
    cout << "MAX2 = " << func(M, n) << "\n";
}
```

Рисунок 4 – Реализация основной функции main

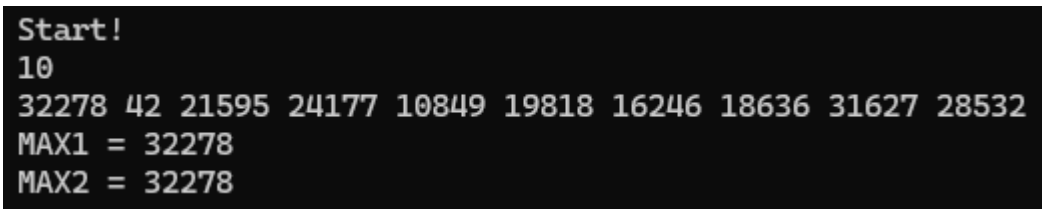


Рисунок 5 – Результат работы программы, при n = 10

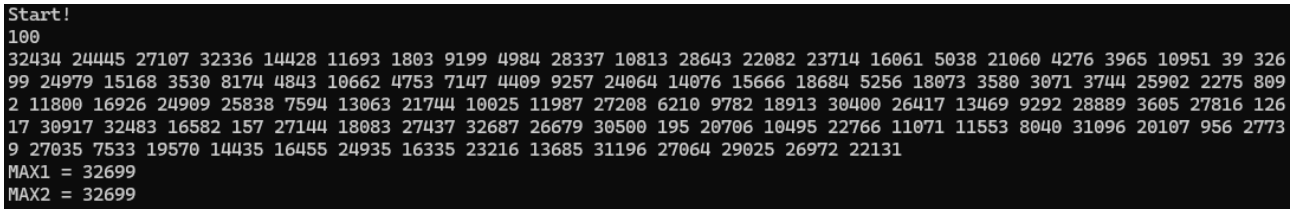


Рисунок 6 – Результат работы программы, при n = 100

Рисунок 7 – Результат работы программы, при n = 1000

В результате тестирование никаких ошибок не обнаружено, программа работает успешно.

2.6 ВЫВОД ПО ПЕРВОМУ ЗАДАНИЮ

В процессе выполнению индивидуального задания было изучено много новой информации о рекурсивных функциях.

РЕШЕНИЕ ЗАДАНИЯ 2

Цель: разработать алгоритм, который будет создавать очередь на однонаправленном списке.

Информационная часть узла — простого типа — целого.

2.1 РЕАЛИЗАЦИЯ ОСНОВНЫХ ФУНКЦИЙ

Реализуем узел и инициализируем три переменных: указатель на голову очереди, указатель на хвост очереди и количество элементов с помощью языка C++ (См. Рисунок 8).

```
struct main_{
    int data;
    struct main_* next;
};
main_* head, * tail;
int cnt = 0; //Количество элементов
```

Рисунок 8 – Реализация схемы узла и необходимых переменных с помощью языка C++

Далее реализуем все необходимые функции для реализации очереди однонаправленного списка (См. Рисунок 8-11).

```
//Создание очереди
void Create() {
    head = (main_*)malloc(sizeof(main_));
    head->next = NULL;
    tail = head;
    cnt = 0;
}
//Проверка на пустой список
void is_empty() {
    if (cnt == 0) {
        cout << "Список пуст!" << endl;
        system("pause");
        menu();
    }
}
/*Функция, которая добавляет элемент в конец*/
void Add_last(main_* temp) {
    tail->data = temp->data;
    tail->next = (main_*)malloc(sizeof(main_));
    tail = tail->next;
    tail->next = NULL;  //Обнуление указателя на следующий элемент
    cnt++;
}
```

Рисунок 9 – Реализация функций Create(), is_empty() и Add_last()

```
/*Считывание элемента из "головы" и перенос в "хвост"*/
void Head_to_tail() {
    main_* buff = head;
    tail->data = buff->data;
    tail->next = (main_*)malloc(sizeof(main_));
    tail = tail->next;
    tail->next = NULL;
    buff = head->next;
    free(head);
    head = buff;
}
/*Вывод списка*/
void Tablitsa() {
    for (int i = 0; i < cnt; i++) {
        printf("%d) %d\n", i + 1, head->data);
        Head_to_tail();
    }
}
/*Вывод информации для главного меню*/
void show_menu() {
    system("cls");
    cout << "1 - Добавить элемент" << endl;
    cout << "2 - Просмотр одного элемента" << endl;
    cout << "3 - Просмотр всех элементов" << endl;
    cout << "4 - Выход" << endl;
}
/*Загрузка данных из очереди в массив*/
void Loading(struct main_* array) {
    for (int i = 0; i < cnt; i++) {
        array[i].data = head->data;
        Head_to_tail();
    }
}
```

Рисунок 10 – Реализация функций Head_to_tail(), Tablitsa(), show_menu() и Loading()

```
void input() {
    system("cls");
    main_ queue;
    int num = 0;
    cout << "Введите число: ";
    cin >> num;
    queue.data = num;
    Add_last(&queue);
}
/*Информация об элементе*/
void inf() {
    int Num = 0;
    system("cls");
    cout << "Введите номер элемента." << "\n";
    cin >> Num;
    Num = Num - 1;
    main_* array = new main_[cnt];
    Loading(array);
    cout << "Число-> " << array[Num].data << endl;
}
```

Рисунок 11 – Реализация функций input() и inf()

После реализации необходимых функций определим теоретическую сложность алгоритма, которая будет равной количеству действий, выполненных пользователем в ходе работы программы.

2.2 ОСНОВНАЯ ПОДПРОГРАММА И ТЕСТИРОВАНИЕ ВСЕЙ ПРОГРАММЫ

Реализуем основную подпрограмму, где будут выполняться остальные подпрограммы (См. Рисунок 12).

```
void menu() {
    char ch;
    show_menu();
    while (1) {
        ch = _getch();
        if (ch == 49) {
            system("cls");
            input();
            system("pause");
            menu();
        }
        if (ch == 50) {
            system("cls");
            //is_empty();
            inf();
            system("pause");
            menu();
        }
        if (ch == 51) {
            system("cls");
            //is_empty();
            Tablitsa();
            system("pause");
            menu();
        }
        if (ch == 52) {
            exit(0);
        }
    }
}
```

Рисунок 12 – Реализация основной подпрограммы

Проведем пару тестов данной программы (См. Рисунок 13-14).

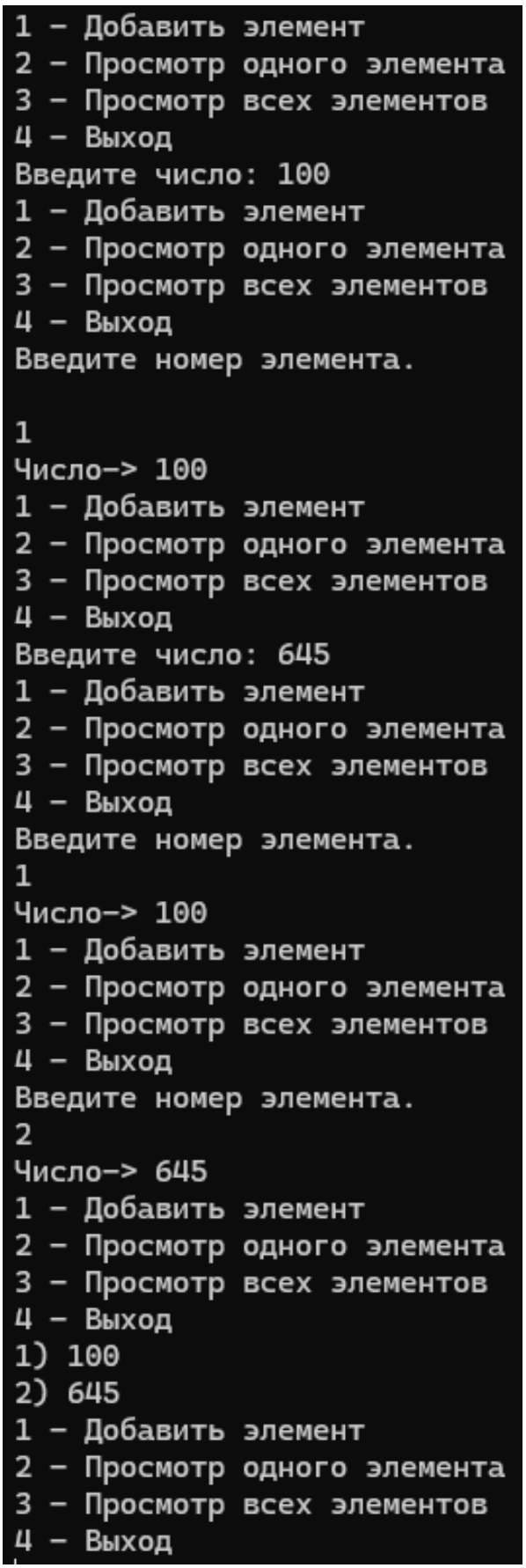


Рисунок 13 – Результаты теста №1

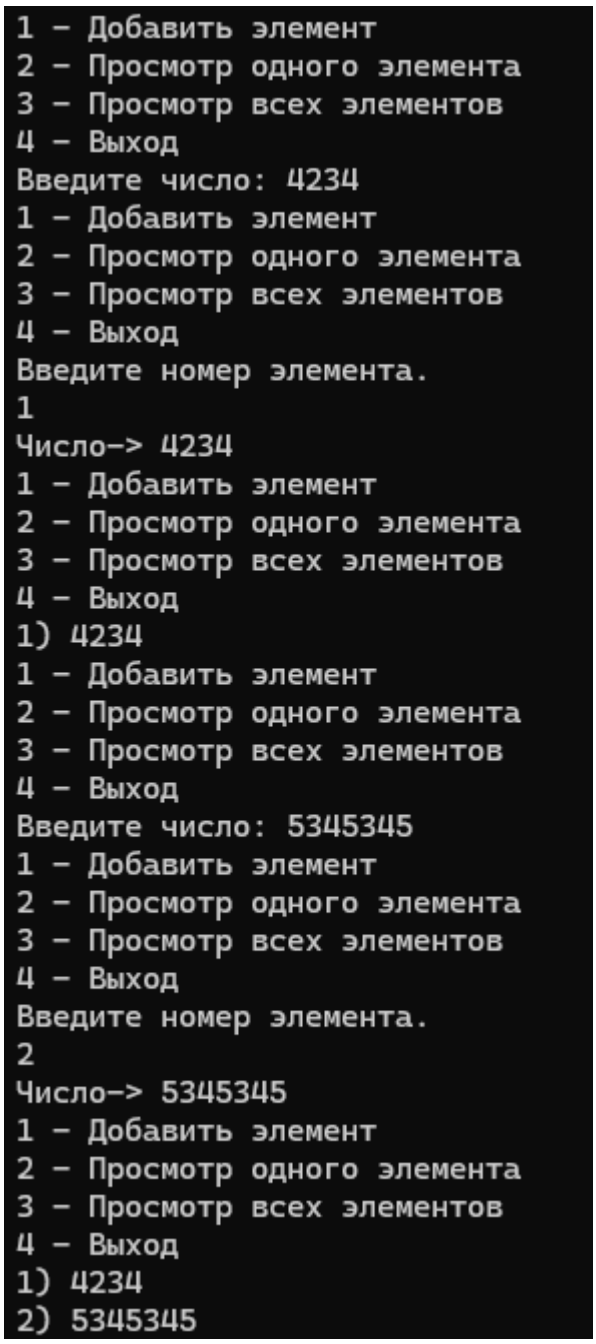


Рисунок 14 – Результаты теста №2

Результаты тестирования показали, что программа функционирует успешно.

ВЫВОД ПО ВТОРОМУ ЗАДАНИЮ

Был разработан алгоритм, который будет создавать очередь на однонаправленном списке.

ОБЩИЙ ВЫВОД

Улучшены навыки реализации рекурсивных алгоритмов и однонаправленных списков.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Структуры данных и проектирование программ : Пер. с англ. / Р. Круз. — М.: БИНОМ. Лаборатория знаний, 2017. — 766 с.

2. Полный справочник по C++ : Пер. с англ. / Г. Шилдт. — М.: ООО "И.Д.Вильямс", 2016. — 796 с.: ил. — Предм. указ.: с. 787-796