МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Отчет по выполнению практического задания №4
**Тема:** «Рекурсивные алгоритмы и их реализации»
Дисциплина Структуры и алгоритмы обработки данных

Выполнил студент Деревянкин Н.А.

группа ИНБО-11-20

**Москва 2021**

## Оглавление

## 1. Вариант 7

## 2. Цель работы

Получить знания и практические навыки по разработке и реализации рекурсивных процессов

## 3. Постановка задачи

Сортировка Простого обмена (Пузырьковая сортировка).

Задание 1. Найти максимальный элемент в массиве из n элементов

Задание 2. Создание очереди на однонаправленном списке.

## 4. Ответы на вопросы

**1) Рекурсивная функция -** это функция, которая вызывает саму себя. Это в случае прямой рекурсии. Существует и косвенная рекурсия - когда две или более функций вызывают друг друга. Когда функция вызывает себя, в стеке создаётся копия значений её параметров, после чего управление передаётся первому исполняемому оператору функции.

**2)** Существует такое понятие как **шаг рекурсии или рекурсивный вызов**. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор пока не получим базовое решение.

**3)** Количество вложенных вызовов функции или процедуры называется **глубиной рекурсии**. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

**4) Условие завершения рекурсии** — это условие, которое, при его выполнении, остановит вызов рекурсивной функции самой себя.

## 5) Виды рекурсии:

**Линейная** - рекурсия, при которой рекурсивные вызовы на любом рекурсивном срезе, инициируют не более одного последующего рекурсивного вызова, называется линейной. Это наиболее простой и часто встречающийся тип рекурсии.

**Каскадная** - При каскадной рекурсии, рекурсивные обращения, как правило, приводят к необходимости многократно решать одни и те же подзадачи. И поэтому, по возможности, от неё или следует избавляться или предпринимать шаги, освобождающие от необходимости производить повторные решения подзадач.

**6) Прямая рекурсия –** рекурсия, при которой программа вызывает саму себя

**Косвенная рекурсия –** рекурсия, при которой программа вызывает другую программу, а та вызывает первую программу

**7) Организация стека рекурсивных вызовов –** хвостовая рекусрия, при которой рекурсивный вызов является последеней операцией возвратом из функции.

## 5. Отчет по заданию 1

### 1. Условие задачи

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.

*Найти максимальный элемент в массиве из n элементов, используя рекурсивную функцию*

### 2. Код функций

```
int search(int* array, int array_size)
{
    static int maxValue = 0;
    static int currIndex = 0;
    static int Index = 0;

    if (array_size) {
        if (*array > maxValue) {
            maxValue = *array;
            Index = currIndex;
        }
        currIndex++;
        return search(++array, --array_size);
    }
    return Index;
}
```

Рисунок 1 – рекурсивная функция поиска максимального элемента

## 3. Требования к задаче 1

```
#include <iostream>
#include <clocale>
using namespace std;

void FindMax(int max, int* arr, int n)
{
    for (int i = 0; i < n; i++)
    {
        if (max < arr[i])
        {
            max = arr[i];
        }
    }
}

int main()
{
    setlocale(LC_ALL, "RU");
    int n;
    cout << "Введите количесвто элементов: ";
    cin >> n;
    int max = -2000;
    int* arr;
    arr = new int[n];
    int num;
    srand(time(NULL));
    cout << "Получившийся массив: ";
    for (int i = 0; i < n; i++)
    {
        num = rand() % 1000 + 1;
        cout << num << " ";
    }
    FindMax(max, arr, n);
    cout << "\nМаксимальное число: " << max;
}
```

Рисунок 2 – иткрационный алгоритм

```
void FindMax(int max, int* arr, int n)
{
    for (int i = 0; i < n; i++)
    {
        if (max < arr[i])
        {
            max = arr[i];
        }
    }
}
```

Рисунок 3 – функция поиска максимального элемента

*Теоретическая сложность алгоритма = 2n+1*

```
int search(int* array, int array_size)
{
    static int maxValue = 0;
    static int currIndex = 0;
    static int Index = 0;

    if (array_size) {
        if (*array > maxValue) {
            maxValue = *array;
            Index = currIndex;
        }
        currIndex++;
        return search(++array, --array_size);
    }
    return Index;
}
```

Рисунок 4 – рекурсвиная функция поиска макс. элемента

Глубина рекурсии прямо пропорциональна количесвту символов в строке. Поэтому при 4 символах – глубина 2, при 8 – глубина 4, и тд.

## 4. Код программы и результат тестирования

```cpp
#include <iostream>
#include <time.h>
#include <clocale>

using namespace std;

int search(int* array, int array_size)
{
    static int maxValue = 0;
    static int currIndex = 0;
    static int Index = 0;

    if (array_size) {
        if (*array > maxValue) {
            maxValue = *array;
            Index = currIndex;
        }
        currIndex++;
        return search(++array, --array_size);
    }
    return Index;
}

void main()
{
    setlocale(LC_ALL, "RU");
    cout << "Размер массива: ";
    int size;
    cin >> size;
    int* myArray = new int[size];
    srand(time(NULL));

    for (int i = 0; i < size; i++)
    {
        myArray[i] = rand() % 100+1;
        cout << myArray[i] << " ";
    }
    int foundIndex = search(myArray, size);
    cout << "\nИндекс элемента: " << foundIndex << endl;
    cout << "Максимальный элемент: " << myArray[foundIndex] << endl;
}
```

Рисунок 5 – код программы

```
Размер массива: 10
25 88 80 77 67 4 74 10 82 11
Индекс элемента: 1
Максимальный элемент: 88
```

Рисунок 6 – результаты тестирования

## 6. Отчет по заданию 2

### 1. Условие задачи

Требования к выполнению первой задачи варианта:

• рекурсивную функцию для обработки списковой структуры 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.

*Создание очереди на однонаправленном списке.*

### 2. Код функций

```
struct Queue
{
    int Data;
    struct Queue* next;
};
```

Рисунок 7 – объявление гланой структуры

```
void Creation() // Создание очереди
{
    head = (Queue*)malloc(sizeof(Queue));
    head->next = NULL;
    tail = head;
    kol = 0;
}
```

Рисунок 8 – создание очереди

```
void check_kol()
{
    if (kol == 0)
    {
        cout << "Список пуст!" << endl;
        system("pause");
        menu();
    }
}
```

Рисунок 9 – проверка пустоты цикла

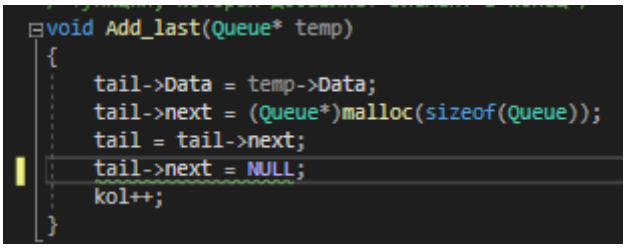



Рисунок 10 – добавление элемента в конец

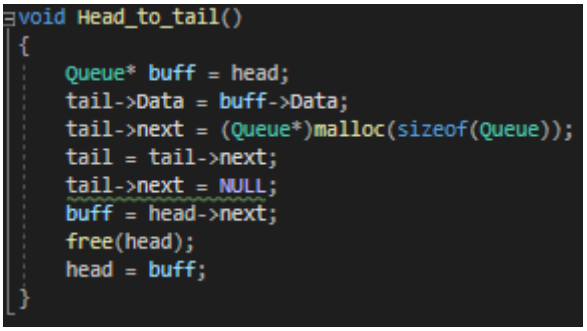



Рисунок 11 – перенос элемента из головы в хвост

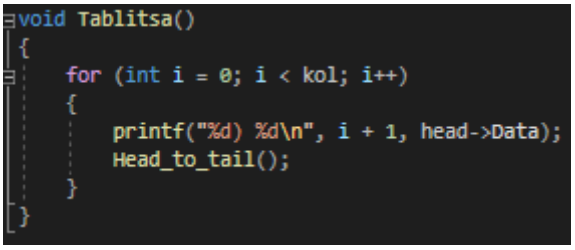



Рисунок 12 – вывод списка

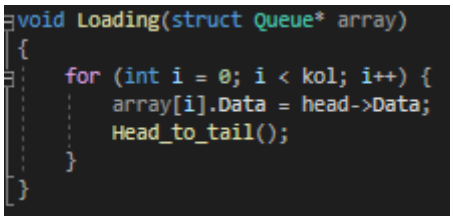



Рисунок 13 – добавление элементов из очереди в массив

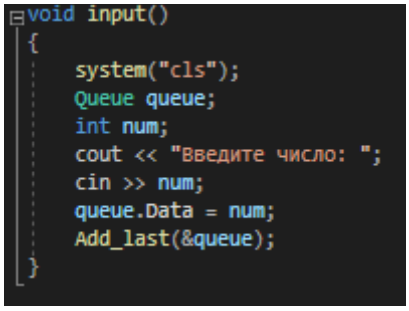
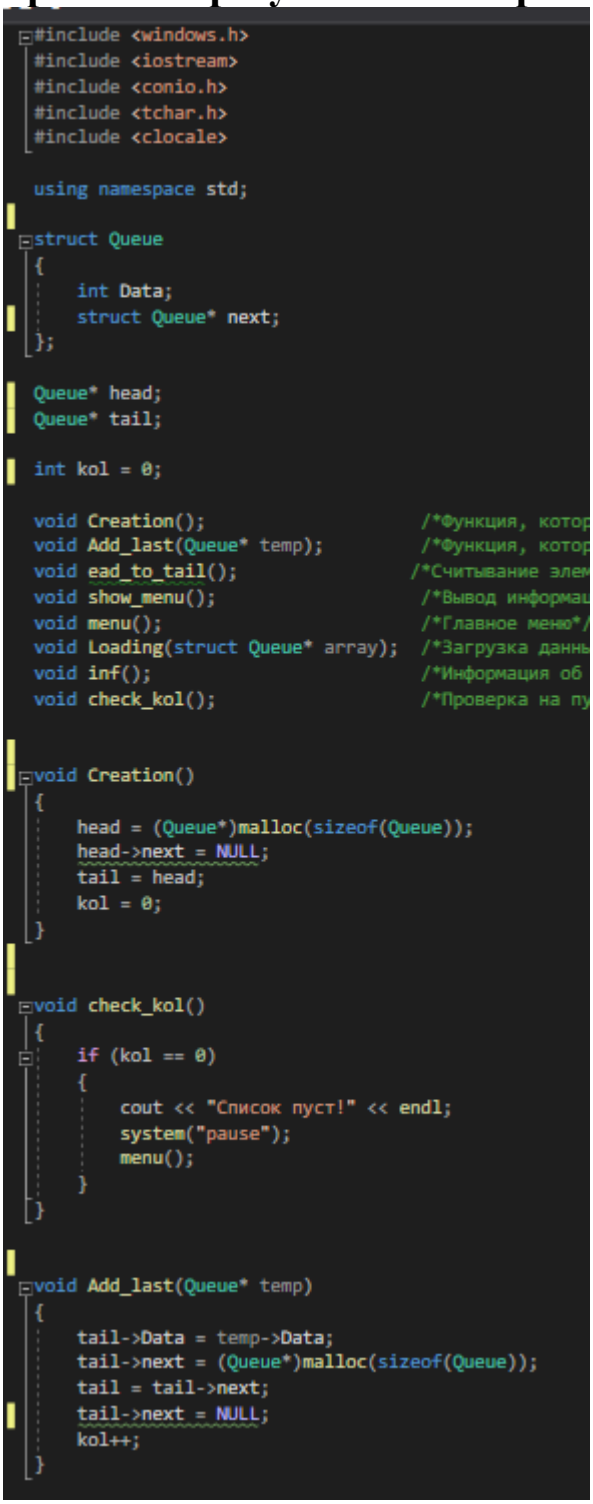



Рисунок 14 – ввод элементов с клавиатуры

Глубина рекурсии прямо пропорциональна количесвту символов в строке. Поэтому при 4 символах – глубина 2, при 8 – глубина 4, и тд.

## 3. Код программы и результат тестирования



```
#include <windows.h>
#include <iostream>
#include <conio.h>
#include <tchar.h>
#include <clocale>

using namespace std;

struct Queue
{
    int Data;
    struct Queue* next;
};

Queue* head;
Queue* tail;

int kol = 0;

void Creation();                        /*Функция, котор
void Add_last(Queue* temp);             /*Функция, котор
void ead_to_tail();                    /*Считывание элем
void show_menu();                       /*Вывод информац
void menu();                            /*Главное меню*/
void Loading(struct Queue* array);      /*Загрузка данны
void inf();                             /*Информация об
void check_kol();                       /*Проверка на пу


void Creation()
{
    head = (Queue*)malloc(sizeof(Queue));
    head->next = NULL;
    tail = head;
    kol = 0;
}


void check_kol()
{
    if (kol == 0)
    {
        cout << "Список пуст!" << endl;
        system("pause");
        menu();
    }
}


void Add_last(Queue* temp)
{
    tail->Data = temp->Data;
    tail->next = (Queue*)malloc(sizeof(Queue));
    tail = tail->next;
    tail->next = NULL;
    kol++;
}
```

Рисунок 15 – 1 часть программы

```
void Head_to_tail()
{
    Queue* buff = head;
    tail->Data = buff->Data;
    tail->next = (Queue*)malloc(sizeof(Queue));
    tail = tail->next;
    tail->next = NULL;
    buff = head->next;
    free(head);
    head = buff;
}

void Tablitsa()
{
    for (int i = 0; i < kol; i++)
    {
        printf("%d) %d\n", i + 1, head->Data);
        Head_to_tail();
    }
}

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;
}

void Loading(struct Queue* array)
{
    for (int i = 0; i < kol; i++) {
        array[i].Data = head->Data;
        Head_to_tail();
    }
}

void input()
{
    system("cls");
    Queue queue;
    int num;
    cout << "Введите число: ";
    cin >> num;
    queue.Data = num;
    Add_last(&queue);
}

void inf()
{
    int Num;
    system("cls");
    cout << "Введите номер элемента: ";
    cin >> Num;
    Num = Num - 1;
    Queue* array = new Queue[kol];
    Loading(array);
    cout << "Число: " << array[Num].Data << endl;
}
```

Рисунок 16 – 2 часть программы

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

int main()
{
    setlocale(LC_ALL, "rus");
    Creation();
    menu();
    return 0;
}
```

Рисунок 17 – 3 часть программы

```
1 - Добавить элемент
2 - Просмотр одного элемента
3 - Просмотр всех элементов
4 - Выход
```

Рисунок 18 – результаты тестирования 1

```
1) 10
2) 213
3) 546
4) 1
5) 91
Для продолжения нажмите любую клавишу . . .
```

Рисунок 19 – результат тестирования 2

## 6. **Вывод к практической работе**

Выполнив практическую работу, я получил знания и практические навыки работы с рекурсией в различных задачах, также разобрался с принципом работы очереди.

## 7. Информационные источники

1) Скворцова Л.А. Структуры и алгоритмы обработки данных, лекции 2 семестр 2021 год, РТУ МИРЭА