МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Программирование

Отчет по лабораторной работе №4

Линейные списки

Выполнили студенты группы M3О-211Б-21

Плоцкий Б.А.

Раужев Ю.М.

Проверила Дмитриева Е.А.

Москва 2022 г.

Оглавление

[Задание 3](#_Toc118718321)

[Общая блок-схема программы 4](#_Toc118718322)

[Структурные схемы алгоритмов 5](#_Toc118718323)

[Функция list\_create() 5](#_Toc118718324)

[Функция list\_delete() 6](#_Toc118718325)

[Функция list\_find\_max\_elem () 9](#_Toc118718326)

[Функция list\_insert() 11](#_Toc118718327)

[Функция list\_pop() 13](#_Toc118718328)

[Функция list\_print() 16](#_Toc118718329)

[Функция list\_push() 18](#_Toc118718330)

[Код программы 20](#_Toc118718331)

[Тестирование программы 37](#_Toc118718332)

[Тест 1 37](#_Toc118718333)

[Тест 2 39](#_Toc118718334)

[Вывод 42](#_Toc118718335)

# Задание

Реализовать заданный согласно варианту задания линейный список, состоящий из 20 элементов заданного типа. Интерфейс должен включать **для всех вариантов** следующие операции**:**

- создание списка;

- вывод на экран и/или в файл значений элементов списка с их индексами (номерами);

- удаление списка,

а также **некоторые из дополнительных операций** (согласно варианту задания):

1. Поиск в списке наличия элемента с заданным значением с получением его номера в списке (повторное вхождение одного и того же значения может быть разрешено или запрещено – см. вариант).
2. **Поиск в списке элемента с максимальным значением с получением его номера в списке (повторное вхождение одного и того же значения может быть разрешено или** запрещено – см. вариант).
3. Включение нового элемента в начало списка.
4. Включение нового элемента в конец списка.
5. **Включение нового элемента в позицию списка с заданным в программе номером.**
6. Удаление элемента из начала списка.
7. **Удаление элемента из конца списка.**
8. Удаление элемента из позиции списка с заданным в программе номером.

После выполнения операций включения или удаления вывести содержимое списка. Выполнение операций организовать с помощью меню.

# Общая блок-схема программы



# Структурные схемы алгоритмов

## Функция list\_create()

1. Назначение:

Создание списка.

1. Прототип функции:

template<typename T>

list<T>\* list\_create();

1. Обращение:

list<double>\* lst = list\_create<double>();

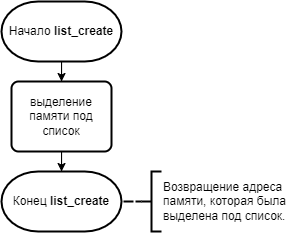


Рисунок 2. Структурная схема алгоритма функции list\_create().

## Функция list\_delete()

1. Назначение:

Удаление списка.

1. Прототип функции:

template<typename T>

void list\_delete(

list<T>\*& \_list // адрес списка

);

1. Обращение

list\_delete(lst);

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| lst | list<T> \*& | список | Входной/Выходной |

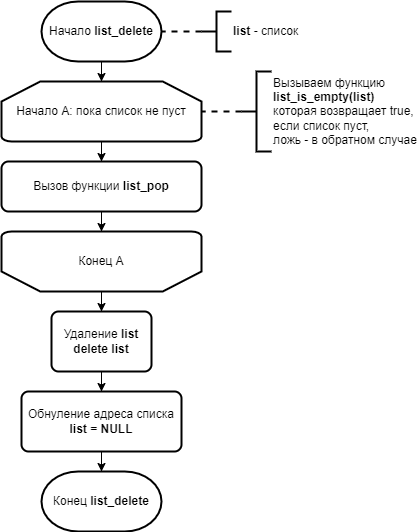


Рисунок 3. Структурная схема алгоритма функции list\_delete ().

## Функция list\_find\_max\_elem ()

1. Назначение:

Нахождение значения и позиции максимального элемента.

1. Прототип функции:

template<typename T>

node<T>\* list\_find\_max\_elem(

list<T>\* \_list, // указатель на список

int& \_pos // позиция элемента в списке

);

1. Обращение:

node<T>\* max\_elem = list\_find\_max\_elem(\_list, pos);

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| lst | list<T> \* | список | Входной/Выходной |
| pos | int | позиция вставки | Входной |

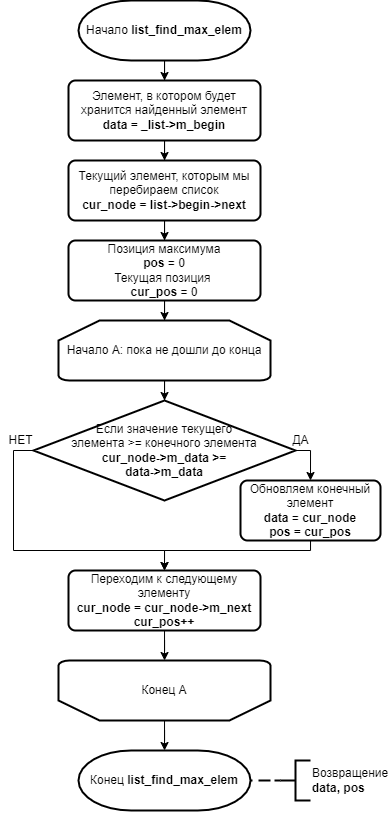


Рисунок 4. Структурная схема алгоритма функции list\_find\_max\_elem().

## Функция list\_insert()

1. Назначение

Вставка элемента в заданную позицию в списке.

1. Прототип функции

template<typename T>

void list\_insert(

list<T>\* \_list, // указатель на список

int \_pos, // позиция вставки

T \_insert\_data // информация для вставки

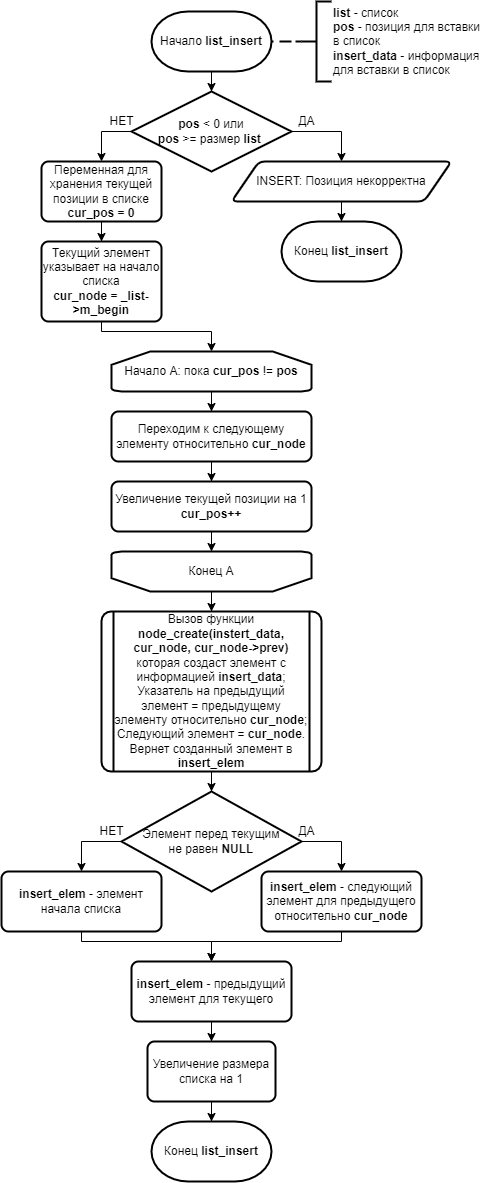
);

1. Обращение

list\_insert(lst, 4, -10.0);

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| lst | list<T> \* | список | Входной/Выходной |
| pos | int | позиция вставки | Входной |
| \_insert\_data | T | информация для вставки | Входной |

Рисунок 5. Структурная схема алгоритма функции improved\_list\_insert()

## Функция list\_pop()

1. Назначение:

Удаление последнего элемента списка.

1. Прототип функции:

template<typename T>

void list\_pop(

list<T>\* \_list // указатель на список

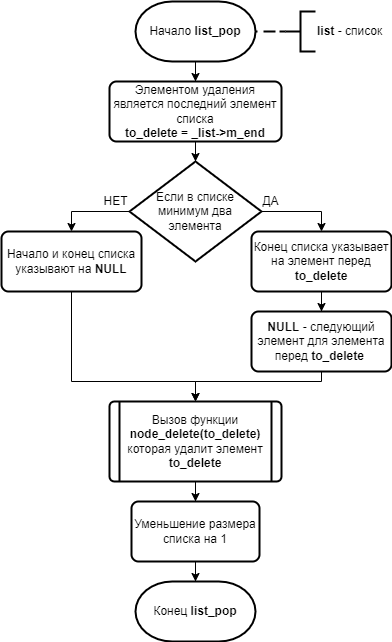
);

1. Обращение:

list\_pop(lst)

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| lst | list<T> \* | список | Входной/Выходной |

Рисунок 6. Структурная схема алгоритма функции list\_pop()

## Функция list\_print()

1. Назначение:

Печать списка.

1. Прототип функции:

void list\_print(

list<T>\* \_list, // печатаемый список

ostream& \_out\_stream = cout // поток для печати

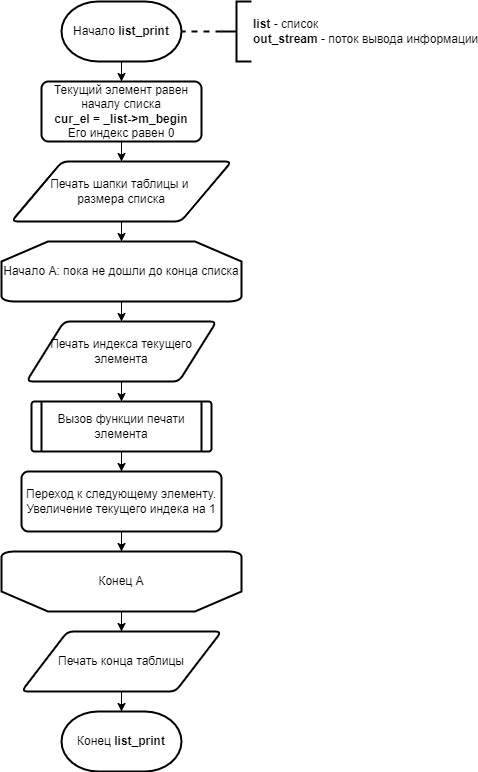
);

1. Обращение:

list\_print(lst);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| lst | list<T> \* | список | Входной/Выходной |

Рисунок 7. Структурная схема алгоритма функции list\_print()

## Функция list\_push()

1. Назначение:

Сортировка последовательности быстрым алгоритмом

1. Прототип функции:

template<typename T>

void list\_push(

list<T>\* \_list, // указатель на список

T data // информация для хранения в элементе

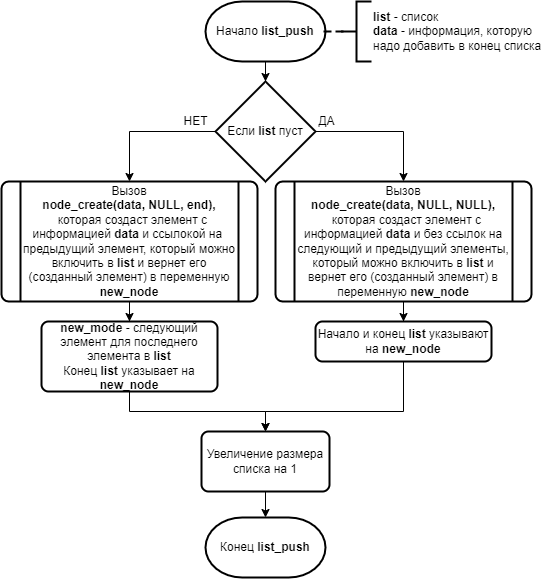
);

1. Обращение:

list\_push(lst, elem);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| lst | list<T> \* | список | Входной/Выходной |
| elem | T | элемент списка | Входной |

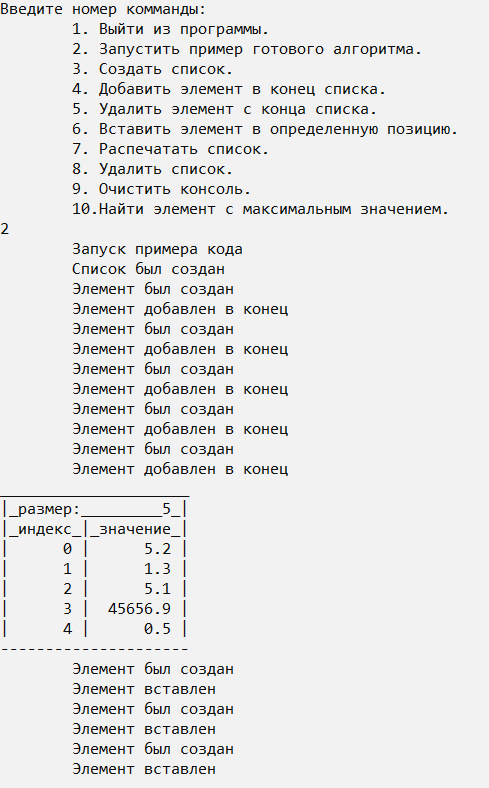
Рисунок 8. Структурная схема алгоритма функции list\_push()

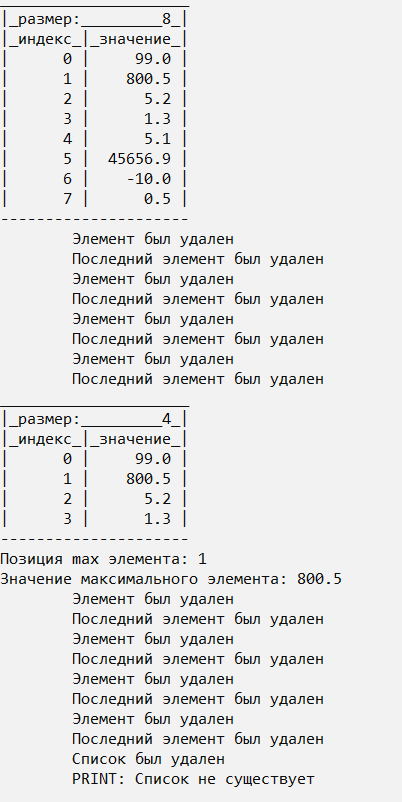
# Код программы

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\  
\*                    кафедра № 304 2 курс 3 семестр информатика         \*  
\*-----------------------------------------------------------------------\*  
\*   Project type : solution                                             \*  
\*   Project name : LW4                                                  \*  
\*   File name    : main.cpp                                             \*  
\*   Language     : c/c++                                                \*  
\*   Programmers  : Плоцкий Б.А. Раужев Ю. М.                            \*  
\*   Created      :  27/10/22                                            \*  
\*   Last revision:  11/11/22                                            \*  
\*   Comment(s)   :                                                      \*  
\*                                                                       \*  
\*   Реализовать заданный согласно варианту задания линейный список,     \*  
\*   состоящий из 20 элементов заданного типа. Интерфейс должен          \*  
\*   включать для всех вариантов следующие операции:                     \*  
\*       - создание списка;                                              \*  
\*       - вывод на экран и/или в файл значений элементов списка         \*  
\*           с их индексами (номерами);                                  \*  
\*       - удаление списка,                                              \*  
\*   а также некоторые из дополнительных операций                        \*  
\*   (согласно варианту задания):                                        \*  
\*       2.Поиск в списке элемента с максимальным значением с            \*  
\*           получением его номера в списке (повторное вхождение         \*  
\*           одного и того же значения может быть разрешено или          \*  
\*           запрещено – см. вариант).                                   \*  
\*       5.Включение нового элемента в позицию списка с заданным         \*  
\*           в программе номером.                                        \*  
\*       7.Удаление элемента из конца списка.                            \*  
\*                                                                       \*  
\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include <iostream>  
#include <fstream>  
#include <iomanip>****using******namespace****std;  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                      К О Н С Т А Н Т Ы                        \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// если нужна печать по каждому действию (0, 1)  
#define NEED\_PRINT\_DEBUG 1  
   
// минимальное и максимальное значение ввода элементов  
#define MAX\_VALUE 1000.0  
#define MIN\_VALUE -1000.0  
   
// минимальное и максимальное значение ввода элементов  
#define MAX\_VALUE\_INSERT 100000  
#define MIN\_VALUE\_INSERT 0  
   
// вывод в консоль сообщения  
#define INFO(str) if(NEED\_PRINT\_DEBUG) cout<<"\t"<<str<<"\n";  
   
// не существует ли список  
#define LIST\_NOT\_EXSISTS(f\_name)\****if****(!list\_exists(\_list)){\  
        cout << "\t" << f\_name << ": Список не существует\n";\****return****;\  
    }  
   
// не существует ли список, возвращаем ret\_obj  
#define LIST\_NOT\_EXSISTS\_RET(f\_name, ret\_obj)\****if****(!list\_exists(\_list)){\  
        cout << "\t" << f\_name << ": Список не существует\n";\****return****ret\_obj;\  
    }  
   
// пуст ли список  
#define LIST\_EMPTY(f\_name)\****if****(list\_is\_empty(\_list)){\  
        cout << "\t" << f\_name << ": Список пуст\n";\****return****;\  
    }  
   
// пуст ли список, возвращаем ret\_obj  
#define LIST\_EMPTY\_RET(f\_name, ret\_obj)\****if****(list\_is\_empty(\_list)){\  
        cout << "\t" << f\_name << ": Список пуст\n";\****return****ret\_obj;\  
    }  
   
// заполнение len элементов элементом symb  
#define OUT\_W(symb, len) fixed << setfill(symb) << setw(len)  
   
// коды для взаимодействия пользователья с программой****enum******class****input\_codes  
{  
    exit = -1,  
    template\_program,  
    list\_create,  
    list\_push,  
    list\_pop,  
    list\_insert,  
    list\_print,  
    list\_delete,  
    clear\_console,  
    find\_max  
};  
   
// строка с коммандами****const******char****\* command\_str =  
"\nВведите номер комманды:\n\  
\t1. Выйти из программы.\n\  
\t2. Запустить пример готового алгоритма.\n\  
\t3. Создать список.\n\  
\t4. Добавить элемент в конец списка.\n\  
\t5. Удалить элемент с конца списка.\n\  
\t6. Вставить элемент в определенную позицию.\n\  
\t7. Распечатать список.\n\  
\t8. Удалить список.\n\  
\t9. Очистить консоль.\n\  
\t10.Найти элемент с максимальным значением.";  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              П Р О Т О Т И П Ы   Ф У Н К Ц И Й                \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*        В С П О М О Г А Т Е Л Ь Н Ы Е   Ф У Н К Ц И И          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// ввод и проверка значений  
template<typename T =****int****>  
T input\_and\_check(  
    T \_min,  
    T \_max,****const******char****\* welcome\_str,****const******char****\* err\_str = "Было введено некорректное значение"  
);  
   
// функция ведения диалога с пользователем  
template<typename T>****void****dialog();  
   
// пример работы с List****void****example\_program();  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              N O D E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// элемент двусвязного списка  
template<typename T>****struct****node  
{  
    T m\_data;       // информация  
    node\* m\_next;   // указатель на следующий элемент  
    node\* m\_prev;   // указатель на предыдущий элемент  
   
    // конструктор  
    node() :m\_next(NULL), m\_prev(NULL), m\_data(T(0)) {};  
};  
   
// создание элемента списка  
template<typename T>  
node<T>\* node\_create(  
    T \_data,        // информация для хранения в элементе  
    node<T>\* \_next,   // указатель на следующий элемент  
    node<T>\* \_prev    // указатель на предыдущий элемент  
);  
   
// удаление элемента списка  
template<typename T>****void****node\_delete(  
    node<T>\*& \_node   // ссылка на элемент  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              L I S T                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// список  
template<typename T>****struct****list  
{  
    node<T>\* m\_begin; // указатель на первый элемент  
    node<T>\* m\_end;       // указатель на элемент после последнего****int****m\_size;         // количество элементов в списке  
   
    // конструктор  
    list() :m\_begin(NULL), m\_end(NULL), m\_size(0) {};  
};  
   
// пуст ли список  
template<typename T>****bool****list\_is\_empty(  
    list<T>\* \_list    // указатель на список  
);  
   
// существует ли список  
template<typename T>****bool****list\_exists(  
    list<T>\* \_list    // указатель на список  
);  
   
// функция инициализации list  
template<typename T>  
list<T>\* list\_create();  
   
// функция добавления элемента в конец списка  
template<typename T>****void****list\_push(  
    list<T>\* \_list,   // указатель на список  
    T data          // информация для хранения в элементе  
);  
   
// функция удаления элемента из конца списка  
template<typename T>****void****list\_pop(  
    list<T>\* \_list    // указатель на список  
);  
   
// удаление списка  
template<typename T>****void****list\_delete(  
    list<T>\*& \_list   // адрес списка  
);  
   
// печать списка  
template<typename T>****void****list\_print(  
    list<T>\* \_list,                   // печатаемый список  
    ostream& \_out\_stream = cout     // поток для печати  
);  
   
// поиск максимального значения  
template<typename T>  
node<T>\* list\_find\_max\_elem(  
    list<T>\* \_list,               // указатель на список****int****& \_pos                   // позиция элемента в списке  
);  
   
// включение элемента в определенную позицию списка  
template<typename T>****void****list\_insert(  
    list<T>\* \_list,   // указатель на список****int****\_pos,       // позиция вставки  
    T \_insert\_data  // информация для вставки  
);  
   
// печать индекса максимального элемента и его значения  
template<typename T>****void****list\_print\_max\_elem(  
    list<T>\* \_list    // указатель на список  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                Г Л А В Н А Я   Ф У Н К Ц И Я                  \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/****int****main()  
{  
    setlocale(LC\_ALL, "ru");  
        
    // запуск диалога с пользователем  
    dialog<****double****>();****return****0;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              Р Е А Л И З А Ц И Я   Ф У Н К Ц И Й              \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*        В С П О М О Г А Т Е Л Ь Н Ы Е   Ф У Н К Ц И И          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// ввод и проверка значений  
template<typename T>  
T input\_and\_check(T \_min, T \_max,****const******char****\* welcome\_str,****const******char****\* err\_str)  
{  
    // размер массива  
    T num;  
   
    // вывод сообщения  
    cout << welcome\_str << "\n";  
    cin >> num;  
   
    // если было введено некорректное значение****if****(num > \_max || num < \_min) {  
        // если была введена не цифра****if****(cin.fail())  
        {  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
        }  
   
        // отчистка консоли  
        //system("cls");  
        cout << err\_str << "\n";  
   
        // рекурсивное обращение  
        num = input\_and\_check(\_min, \_max, welcome\_str, err\_str);  
    }****return****num;  
}  
   
// функция ведения диалога с пользователем  
template<typename T>****void****dialog()  
{  
    /\* коды комманд  
    exit                -1  
    template\_program    0  
    list\_create         1  
    list\_push           2  
    list\_pop            3  
    list\_insert         4  
    list\_print          5  
    list\_delete         6  
    clear\_console       7  
    find\_max            8  
    \*/  
   
    // переменная содержащая коды действий  
    input\_codes in\_code;  
   
    // элемент для вставки  
    T elem;  
   
    //позиция вставки****int****pos\_to\_insert;  
   
    // переменная списка  
    list<T>\* lst = NULL;  
   
    // максимальный элемент  
    node<T>\* max\_elem = NULL;****do*** *{  
        // запрос у пользователя следующих действий  
        in\_code = input\_codes(input\_and\_check(1, 10, command\_str) - 2);  
   
        // запуск соответствующих функций****switch****(in\_code)  
        {****case****input\_codes::exit:  
            INFO("Произведен выход");****break****;****case****input\_codes::template\_program:  
            INFO("Запуск примера кода");  
            example\_program();****break****;****case****input\_codes::list\_create:****if****(!list\_exists(lst))  
                lst = list\_create<T>();****else*** *INFO("DIALOG: Список уже существует");****break****;****case****input\_codes::list\_push:  
            elem = input\_and\_check(MIN\_VALUE, MAX\_VALUE,  
                "Введите элемент для вставки в конец");  
            list\_push(lst, elem);****break****;****case****input\_codes::list\_pop:  
            list\_pop(lst);****break****;****case****input\_codes::list\_insert:  
            elem = input\_and\_check(MIN\_VALUE, MAX\_VALUE,  
                "Введите элемент для вставки");  
   
            pos\_to\_insert = input\_and\_check(MIN\_VALUE\_INSERT, MAX\_VALUE\_INSERT,  
                "Введите позицию для вставки");  
   
            // вставка элемента  
            list\_insert(lst, pos\_to\_insert, elem);****break****;****case****input\_codes::list\_print:  
            // печать списка  
            list\_print(lst);****break****;****case****input\_codes::list\_delete:  
            // удаление списка  
            list\_delete(lst);****break****;****case****input\_codes::clear\_console:  
            system("cls");****break****;****case****input\_codes::find\_max:  
            list\_print\_max\_elem(lst);****break****;****default****:  
            INFO("Неизвестный код")****break****;  
        }  
   
    }****while****(  
        // пока пользователь не захотел выйти из программы  
        // или пока не запустил пример программыЫ  
        in\_code != input\_codes::exit &&  
        in\_code != input\_codes::template\_program  
        );  
}  
   
// пример работы с List****void****example\_program()  
{  
    // создание списка  
    list<****double****>\* lst = list\_create<****double****>();  
   
    // добавление элементов в конец  
    list\_push(lst, 5.2);  
    list\_push(lst, 1.3);  
    list\_push(lst, 5.1456);  
    list\_push(lst, 45656.9);  
    list\_push(lst, 0.456);  
   
    // вывод списка  
    list\_print(lst);  
   
    // вставка элемента в определенную позицию  
    list\_insert(lst, 4, -10.0);  
    list\_insert(lst, 0, 800.5);  
    list\_insert(lst, 0, 99.009);  
   
    // вывод списка  
    list\_print(lst);  
   
    // удаление элементов списка с конца  
    list\_pop(lst);  
    list\_pop(lst);  
    list\_pop(lst);  
    list\_pop(lst);  
   
    // вывод списка  
    list\_print(lst);  
   
    // печать максимального элемента списка  
    list\_print\_max\_elem(lst);  
   
    // удаление списка  
    list\_delete(lst);  
   
    // вывод списка  
    list\_print(lst);  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              N O D E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// создание элемента списка  
template<typename T>  
node<T>\* node\_create(T \_data, node<T>\* \_next, node<T>\* \_prev)  
{  
    // создаем первый элемент в списке  
    node<T>\* new\_node =****new****node<T>;  
   
    // настраиваем только что созданный элемент  
    new\_node->m\_data = \_data;  
    new\_node->m\_next = \_next;  
    new\_node->m\_prev = \_prev;  
   
    INFO("Элемент был создан");****return****new\_node;  
}  
   
// удаление элемента списка  
template<typename T>****void****node\_delete(node<T>\*& \_node)  
{  
    // разрыв связей с другими элементами  
    \_node->m\_data = T(0);  
    \_node->m\_next = NULL;  
    \_node->m\_prev = NULL;  
   
    // обнуление памяти  
    delete \_node;  
   
    // обнуление адреса  
    \_node = NULL;  
   
    INFO("Элемент был удален");  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              L I S T                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// пуст ли список  
template<typename T>****bool****list\_is\_empty(list<T>\* \_list)  
{****return****\_list->m\_begin == NULL;  
}  
   
// существует ли список  
template<typename T>****bool****list\_exists(list<T>\* \_list)  
{****return****\_list != NULL;  
}  
   
// функция инициализации list  
template<typename T>  
list<T>\* list\_create()  
{  
    // выделение памяти под список  
    list<T>\* \_list =****new****list<T>;  
   
    INFO("Список был создан");  
   
    // возвращение адресса списка list****return****\_list;  
}  
   
// функция добавления элемента в конец списка  
template<typename T>****void****list\_push(list<T>\* \_list, T data)  
{  
    // выход, если список не существует  
    LIST\_NOT\_EXSISTS("PUSH");  
   
    // элемент добавления  
    node<T>\* new\_node;  
   
    // если список пуст****if****(list\_is\_empty(\_list))  
    {  
        // создаем новый элемент списка list  
        new\_node = node\_create<T>(data, NULL, NULL);  
   
        // записываем адресс нового элемента в list  
        // в качестве первого элемента  
        \_list->m\_begin = new\_node;  
        \_list->m\_end = new\_node;  
    }  
    // если в списке есть элементы****else*** *{  
        // создаем элемент для вставки в list  
        new\_node = node\_create<T>(  
            data, NULL, \_list->m\_end);  
   
        // настравиваем последний элемент в list  
        \_list->m\_end->m\_next = new\_node;  
   
        // настраиваем сам list  
        \_list->m\_end = new\_node;  
    }  
   
    //увеличиваем количество элементов  
    \_list->m\_size++;  
   
    INFO("Элемент добавлен в конец");  
}  
   
// функция удаления элемента из конца списка  
template<typename T>****void****list\_pop(list<T>\* \_list)  
{  
    // выходим, если список не существует или пуст  
    LIST\_NOT\_EXSISTS("POP");  
    LIST\_EMPTY("POP");  
   
    // элемент удаления  
    node<T>\* to\_delete = \_list->m\_end;  
   
    // если в списке минимум 2 элемента****if****(\_list->m\_begin != to\_delete)  
    {  
        // настройка списка list  
        \_list->m\_end = to\_delete->m\_prev;  
   
        // разрыв связей с элементом удаления  
        to\_delete->m\_prev->m\_next = NULL;  
    }****else*** *{  
        // обнуляем адреса списка, указвающие  
        // на первый и последний элементы списка  
        \_list->m\_begin = \_list->m\_end = NULL;  
    }  
   
    // удаления самого элемента  
    node\_delete(to\_delete);  
   
    // уменьшение размера списка на 1  
    \_list->m\_size--;  
   
    INFO("Последний элемент был удален");  
}  
   
// удаление списка  
template<typename T>****void****list\_delete(list<T>\*& \_list)  
{  
    // выходим, если список не существует  
    LIST\_NOT\_EXSISTS("DELETE");  
   
    // пока список не пуст****while****(!list\_is\_empty(\_list))  
    {  
        // удаляем элементы списка с конца  
        list\_pop(\_list);  
    }  
   
    // очищение памяти  
    delete \_list;  
   
    // обнуление адреса  
    \_list = NULL;  
   
    INFO("Список был удален")  
}  
   
// печать списка  
template<typename T>****void****list\_print(list<T>\* \_list, ostream& \_out\_stream)  
{  
    // выходим, если список не существует или пуст  
    LIST\_NOT\_EXSISTS("PRINT");  
    LIST\_EMPTY("PRINT");  
   
    // создаем элемент для чтения данных из list'а  
    node<T>\* cur\_el = \_list->m\_begin;  
   
    // индекс элемента****int****i = 0;  
   
    // печать шапки таблицы  
    \_out\_stream << OUT\_W('\_', 22) << '\n';  
   
    // дополнительная информация о списке list  
    \_out\_stream << "|\_размер:\_" <<  
        OUT\_W('\_', 9) << \_list->m\_size << "\_|\n";  
    \_out\_stream << "|\_индекс\_|\_значение\_|\n";  
   
    // идем по list'у, пока на наткнемся на конечный элемент****while****(cur\_el != NULL)  
    {  
        // вывод данных элемента  
        \_out\_stream << "| " << OUT\_W(' ', 6) << i  
            << " | " << setprecision(1)  
            << OUT\_W(' ', 8) << cur\_el->m\_data  
            << " |\n";  
   
        // переход к следующему элементу  
        cur\_el = cur\_el->m\_next;  
        i++;  
    }  
    // печать конца таблицы  
    \_out\_stream << OUT\_W('-', 22) << '\n';  
}  
   
// поиск максимального значения  
template<typename T>  
node<T>\* list\_find\_max\_elem(list<T>\* \_list,****int****& \_pos)  
{  
    // выходим, если список не существует или пуст  
    LIST\_NOT\_EXSISTS\_RET("FIND", NULL);  
    LIST\_EMPTY\_RET("FIND", NULL);  
   
    // элемент списка, который мы ищем  
    node<T>\* data = \_list->m\_begin;  
   
    // берем элемент, с помощью которого  
    // пройдемся по всему списку  
    // он равен второму по счету элементу в списке  
    node<T>\* cur\_node = \_list->m\_begin->m\_next;  
   
    // обнуление позиции элемента****int****\_cur\_pos = 0;  
   
    // пока не дошли до конца списка****while****(cur\_node != NULL)  
    {  
        // сравниваем с помощью \_comp информацию в  
        // cur\_node и в следующем элементе****if****(cur\_node->m\_data >= data->m\_data)  
        {  
            data = cur\_node;  
            \_pos = \_cur\_pos;  
        }  
   
        \_cur\_pos++;  
        // переход к следующему элементу  
        cur\_node = cur\_node->m\_next;  
    }  
   
    // возвращение найденного элемента списка  
    // или возвращение NULL****return****data;  
}  
   
// включение элемента в определенную позицию списка  
template<typename T>****void****list\_insert(list<T>\* \_list,****int****\_pos, T \_insert\_data)  
{  
    // выход, если список не существует  
    LIST\_NOT\_EXSISTS("INSERT");  
   
    // позиция должна быть меньше размера списка****if****(\_pos < 0 || \_pos >= \_list->m\_size)  
    {  
        cout << "\tINSERT: Позиция некорректна\n";****return****;  
    }  
   
    // текущая позиция****int****cur\_pos = 0;  
       
    // текущий элемент  
    node<T>\* cur\_node = \_list->m\_begin;  
   
    // пока не дошли до нужной позиции****while****(cur\_pos != \_pos)  
    {  
        // смещаемся вправо по списку  
        cur\_node = cur\_node->m\_next;  
        cur\_pos++;  
    }  
   
    // создание нового элемента списка  
    node<T>\* instert\_elem =  
        node\_create<T>(\_insert\_data, cur\_node, cur\_node->m\_prev);  
   
    // настройка связей у элементов в списке  
    // если перед cur\_node есть элементы****if****(cur\_node->m\_prev != NULL)  
    {  
        cur\_node->m\_prev->m\_next = instert\_elem;  
    }  
    // иначе меняем m\_begin у списка на instert\_elem****else*** *{  
        \_list->m\_begin = instert\_elem;  
    }  
    cur\_node->m\_prev = instert\_elem;  
   
    // увеличение размера списка  
    \_list->m\_size++;  
   
    INFO("Элемент вставлен");  
}  
   
template<typename T>****void****list\_print\_max\_elem(list<T>\* \_list)  
{  
    LIST\_NOT\_EXSISTS("PRINT MAX ELEM");  
    LIST\_EMPTY("PRINT MAX ELEM");  
   
    // позиция в списке****int****pos = 0;  
   
    // поиск максимального значения в списке  
    node<T>\* max\_elem = list\_find\_max\_elem(\_list, pos);  
   
    // печать позиции максимального элемента  
    // и самого элемента  
    cout << "Позиция max элемента: " << pos + 1 << endl;  
    cout << "Значение максимального элемента: "  
        << max\_elem->m\_data << endl;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Of LW4.cpp File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

# Тестирование программы

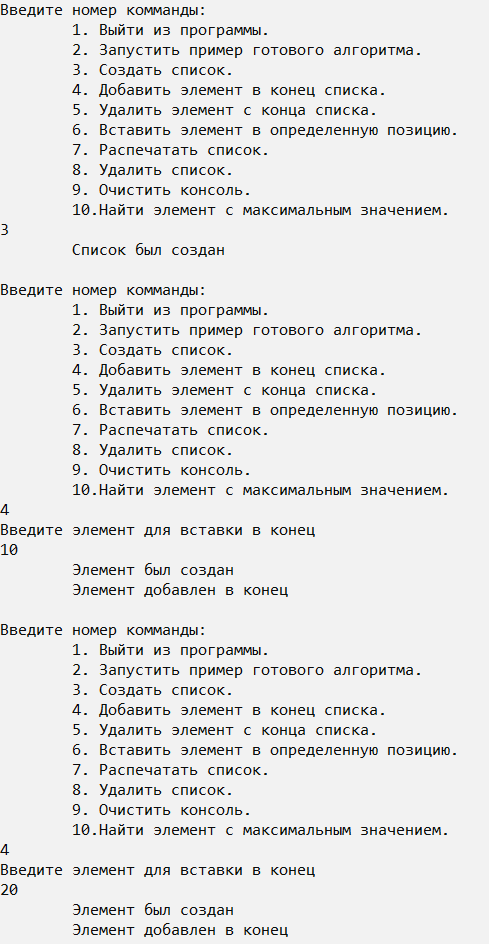
### Тест 1

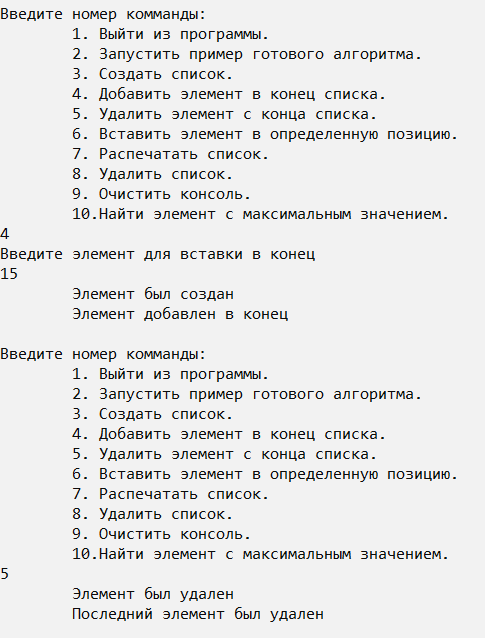
Выполнение заранее созданного алгоритма.

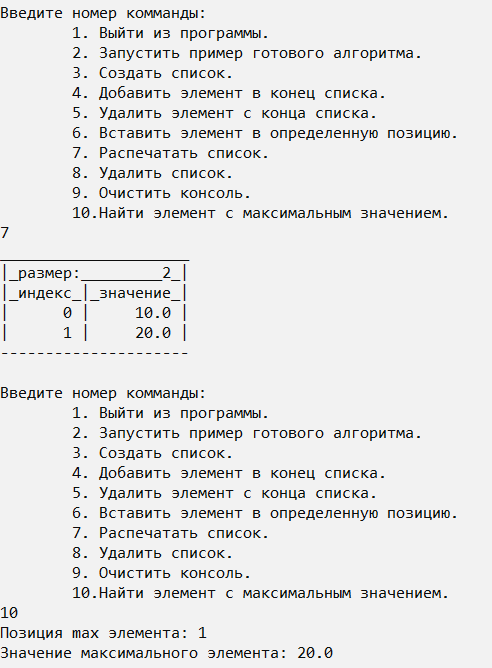


### Тест 2

Создание собственного списка.







# Вывод

Работа программы завершена на основании:

1) Полученные результаты совпали с ожидаемыми;

2) Считаем набор тестов полным.