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 №5

Деки

Выполнили студенты группы M3О-211Б-21

Плоцкий Б.А.

Раужев Ю.М.

Проверила Дмитриева Е.А.

Москва 2022 г.

Оглавление

[Задание 3](#_Toc119917439)

[Общая блок-схема программы 4](#_Toc119917440)

[Структурные схемы алгоритмов 5](#_Toc119917441)

[Функция deq\_create() 5](#_Toc119917442)

[Функция deq\_delete() 6](#_Toc119917443)

[Функция deq\_get\_front() 8](#_Toc119917444)

[Функция deq\_pop\_back() 9](#_Toc119917445)

[Функция deq\_print() 11](#_Toc119917446)

[Функция deq\_push\_back() 13](#_Toc119917447)

[Код программы 15](#_Toc119917448)

[Тестирование программы 28](#_Toc119917449)

[Тест 1 28](#_Toc119917450)

[Вывод 29](#_Toc119917451)

# Задание

Программно реализовать дек в виде **двунаправленного списка** согласно варианту задания. Выполнение операций организовать с помощью меню:

* создание дека;
* вывод на экран или в файл значений элементов дека с их индексами (номерами);
* очистка дека,
* дополнительные операции(согласно варианту задания):

1. выборка элемента из начала дека;
2. выборка элемента из конца дека;
3. добавление элемента в начало дека;
4. добавление элемента в конец дека;

После выполнения операций добавления или выборки вывести новое содержимое дека.

**Варианты заданий**

|  |  |  |
| --- | --- | --- |
| №  вар. | Тип элементов | Дополнительные  операции |
| 1 | вещественный | 1, 4 |
| 2 | символьный | 2, 3 |
| 3 | целочисленный | 1, 3 |
| 4 | символьный | 2, 4 |
| 5 | целочисленный | 1, 4 |
| 6 | вещественный | 2, 3 |
| 7 | символьный | 1, 2 |
| 8 | целочисленный | 2, 4 |

Отчет должен содержать:

* задание на лабораторную работу, соответствующее варианту;
* структурные схемы алгоритмов программы и одной функции (по собственному выбору);
* текст программы и всех функций;
* результаты работы программы;
* выводы по работе.

# Общая блок-схема программы



# Структурные схемы алгоритмов

## Функция deq\_create()

1. Назначение:

Создание дека.

1. Прототип функции:

template<typename T>

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

1. Обращение:

deq<double> \* dq = deq\_create<double>();

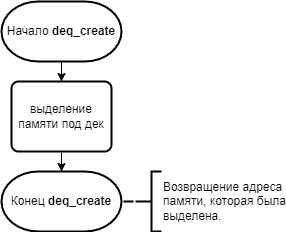


Рисунок 2. Структурная схема алгоритма функции deq\_create().

## Функция deq\_delete()

1. Назначение:

Удаление дека.

1. Прототип функции:

template<typename T>

void deq\_delete(

deq<T>\*& \_deq // адрес списка

);

1. Обращение

deq\_delete(dq);

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| dq | deq<T> \*& | дек | Входной/Выходной |

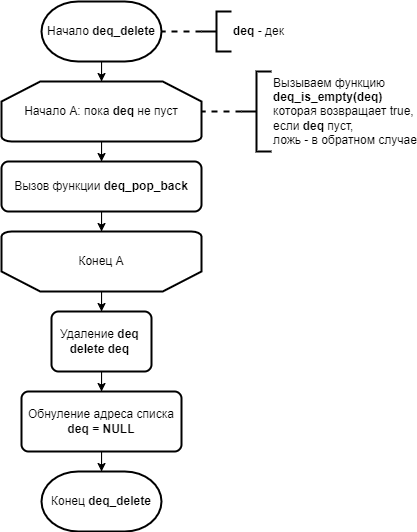


Рисунок 3. Структурная схема алгоритма функции deq\_delete ().

## Функция deq\_get\_front()

1. Назначение:

выборка элемента из левого конца (начала).

1. Прототип функции:

template<typename T>

node<T>\* deq\_get\_front(

deq<T>\* \_deq // указатель на дек

);

1. Обращение:

node<double>\* front = deq\_get\_front(dq);

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| dq | deq<T> \* | список | Входной/Выходной |



Рисунок 4. Структурная схема алгоритма функции deq\_get\_front().

## Функция deq\_pop\_back()

1. Назначение:

Удаление последнего элемента дека.

1. Прототип функции:

template<typename T>

void deq\_pop\_back(

deq<T>\* \_deq // указатель на дек

);

1. Обращение:

deq\_pop\_back(dq)

1. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| dq | deq<T> \* | дек | Входной/Выходной |

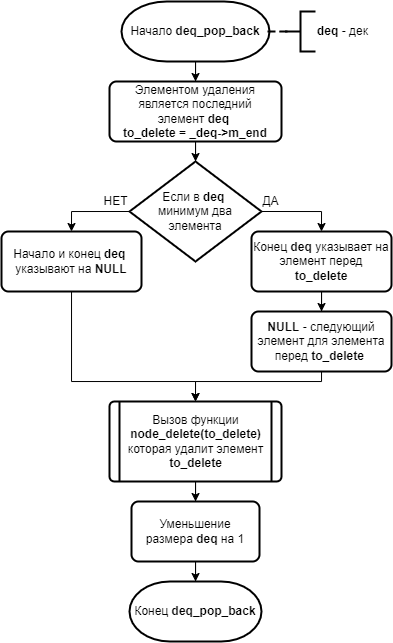


Рисунок 6. Структурная схема алгоритма функции deq\_pop\_back()

## Функция deq\_print()

1. Назначение:

Печать дека.

1. Прототип функции:

void deq\_print(

deq<T>\* \_deq, // печатаемый дек

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

);

1. Обращение:

deq\_print(dq);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| dq | deq<T> \* | дек | Входной/Выходной |

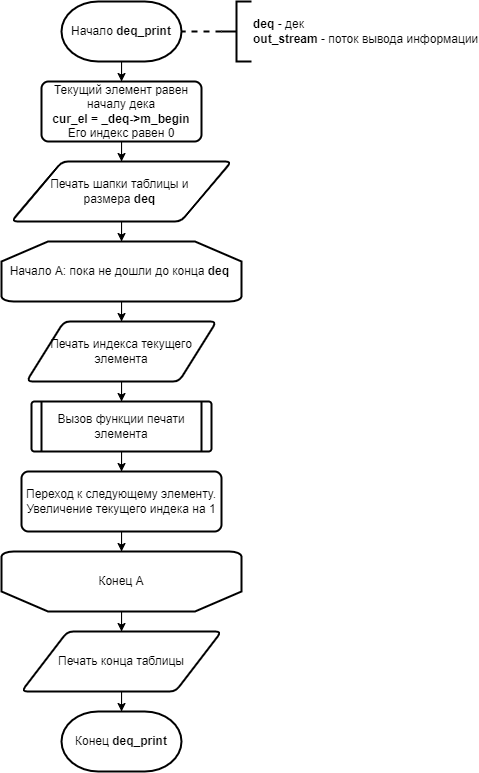


Рисунок 7. Структурная схема алгоритма функции deq\_print()

## Функция deq\_push\_back()

1. Назначение:

Функция добавления элемента в конец дека.

1. Прототип функции:

template<typename T>

void deq\_push\_back(

deq<T>\* \_deq, // указатель на дек

T data // информация для хранения в элементе

);

1. Обращение:

deq\_push(dq, elem);

1. Описание параметров

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

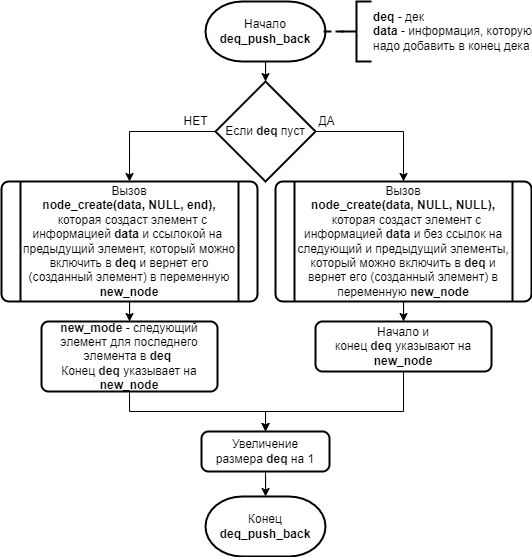


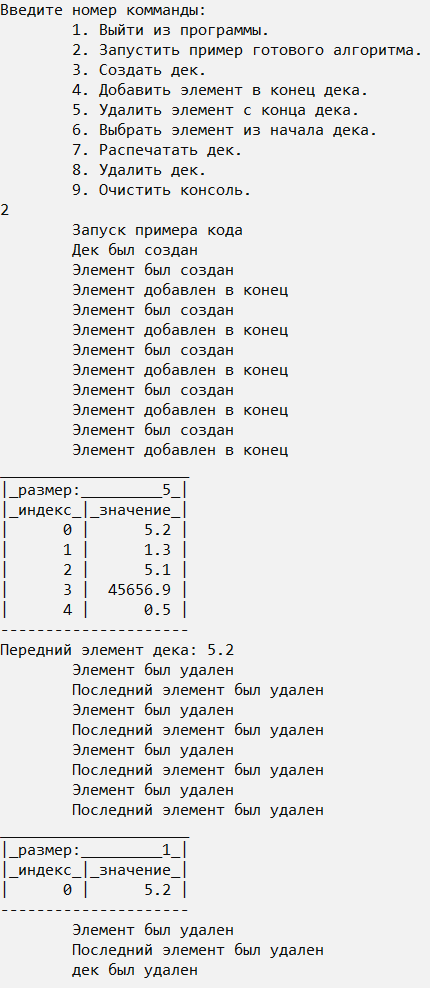
Рисунок 8. Структурная схема алгоритма функции deq\_push\_back()

# Код программы

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\  
\*                    кафедра № 304 2 курс 3 семестр информатика         \*  
\*-----------------------------------------------------------------------\*  
\*   Project type : solution                                             \*  
\*   Project name : LW5                                                  \*  
\*   File name    : main.cpp                                             \*  
\*   Language     : c/c++                                                \*  
\*   Programmers  : Плоцкий Б.А. Раужев Ю. М.                            \*  
\*   Created      : 17/11/22                                             \*  
\*   Last revision: 21/11/22                                             \*  
\*   Comment(s)   :                                                      \*  
\*                                                                       \*  
\*   Программно реализовать дек в виде двунаправленного дека согласно    \*  
\*   варианту задания. Выполнение операций организовать с помощью меню:  \*  
\*   •    создание дека;                                                 \*  
\*   •    вывод на экран или в файл значений элементов дека с их         \*  
\*       индексами (номерами);                                           \*  
\*   •    очистка дека,                                                  \*  
\*   •   дополнительные операции(согласно варианту задания):             \*  
\*       1.  выборка элемента из начала дека;                            \*  
\*       4.  добавление элемента в конец дека;                           \*  
\*                                                                       \*  
\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#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 дек  
#define DEQ\_NOT\_EXSISTS(f\_name)\****if****(!deq\_exists(\_deq)){\  
        cout << "\t" << f\_name << ": дек не существует\n";\****return****;\  
    }  
   
// не существует ли дек, возвращаем ret\_obj  
#define DEQ\_NOT\_EXSISTS\_RET(f\_name, ret\_obj)\****if****(!deq\_exists(\_deq)){\  
        cout << "\t" << f\_name << ": дек не существует\n";\****return****ret\_obj;\  
    }  
   
// пуст ли дек  
#define DEQ\_EMPTY(f\_name)\****if****(deq\_is\_empty(\_deq)){\  
        cout << "\t" << f\_name << ": дек пуст\n";\****return****;\  
    }  
   
// пуст ли дек, возвращаем ret\_obj  
#define DEQ\_EMPTY\_RET(f\_name, ret\_obj)\****if****(deq\_is\_empty(\_deq)){\  
        cout << "\t" << f\_name << ": дек 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,  
    deq\_create,  
    deq\_push\_back,  
    deq\_pop\_back,  
    deq\_get\_front,  
    deq\_print,  
    deq\_delete,  
    clear\_console  
};  
   
// строка с коммандами****const******char****\* command\_str =  
"\nВведите номер комманды:\n\  
\t1. Выйти из программы.\n\  
\t2. Запустить пример готового алгоритма.\n\  
\t3. Создать дек.\n\  
\t4. Добавить элемент в конец дека.\n\  
\t5. Удалить элемент с конца дека.\n\  
\t6. Выбрать элемент из начала дека.\n\  
\t7. Распечатать дек.\n\  
\t8. Удалить дек.\n\  
\t9. Очистить консоль.";  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              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 =****double****>****void****dialog();  
   
// пример работы с деком****void****example\_program();  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              N O D E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// элемент двусвязного дека  
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 дека  
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 дека  
template<typename T>****void****node\_delete(  
    node<T>\*& \_node   // ссылка на элемент  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              L I S T                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// дек  
template<typename T>****struct****deq  
{  
    node<T>\* m\_begin; // указатель на первый элемент  
    node<T>\* m\_end;       // указатель на элемент после последнего****int****m\_size;         // количество элементов в деке  
   
    // конструктор  
    deq() :m\_begin(NULL), m\_end(NULL), m\_size(0) {};  
};  
   
// пуст ли дек  
template<typename T>****bool****deq\_is\_empty(  
    deq<T>\* \_deq  // указатель на дек  
);  
   
// существует ли дек  
template<typename T>****bool****deq\_exists(  
    deq<T>\* \_deq  // указатель на дек  
);  
   
// функция инициализации дека  
template<typename T>  
deq<T>\* deq\_create();  
   
// функция добавления элемента в конец дека  
template<typename T>****void****deq\_push\_back(  
    deq<T>\* \_deq, // указатель на дек  
    T data          // информация для хранения в элементе  
);  
   
// функция удаления элемента из конца дек  
template<typename T>****void****deq\_pop\_back(  
    deq<T>\* \_deq  // указатель на дек  
);  
   
// выборка элемента из левого конца (начала)  
template<typename T>  
node<T>\* deq\_get\_front(  
    deq<T>\* \_deq  // указатель на дек  
);  
   
// удаление дека  
template<typename T>****void****deq\_delete(  
    deq<T>\*& \_deq // адресс дек  
);  
   
// печать дека  
template<typename T>****void****deq\_print(  
    deq<T>\* \_deq,             // печатаемый дек  
    ostream& \_out\_stream = cout // поток для печати  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                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();****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()  
{  
    // переменная содержащая коды действий  
    input\_codes in\_code;  
   
    // элемент для вставки  
    T elem;  
   
    //позиция вставки****int****pos\_to\_insert;  
   
    // переменная дека  
    deq<T>\* dq = NULL;  
   
    // временный элемент  
    node<T>\* temp = NULL;****do*** *{  
        // запрос у пользователя следующих действий  
        in\_code = input\_codes(input\_and\_check(1, 9, command\_str));  
   
        // 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::deq\_create:****if****(!deq\_exists(dq))  
                dq = deq\_create<T>();****else*** *INFO("DIALOG: Дек уже существует");****break****;****case****input\_codes::deq\_push\_back:  
            elem = input\_and\_check(MIN\_VALUE, MAX\_VALUE,  
                "Введите элемент для вставки в конец");  
            deq\_push\_back(dq, elem);  
            deq\_print(dq);****break****;****case****input\_codes::deq\_pop\_back:  
            deq\_pop\_back(dq);****break****;****case****input\_codes::deq\_get\_front:  
            temp = deq\_get\_front(dq);****if****(temp)  
            {  
                cout << "Передний элемент дека: " << temp->m\_data << endl;  
                deq\_print(dq);  
            }****break****;****case****input\_codes::deq\_print:  
            // печать дека  
            deq\_print(dq);****break****;****case****input\_codes::deq\_delete:  
            // удаление дека  
            deq\_delete(dq);****break****;****case****input\_codes::clear\_console:  
            system("cls");****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 деком****void****example\_program()  
{  
    // создание дека  
    deq<****double****>\* dq = deq\_create<****double****>();  
   
    // добавление элементов в конец  
    deq\_push\_back(dq, 5.2);  
    deq\_push\_back(dq, 1.3);  
    deq\_push\_back(dq, 5.1456);  
    deq\_push\_back(dq, 45656.9);  
    deq\_push\_back(dq, 0.456);  
   
    // вывод дека  
    deq\_print(dq);  
   
    // элемент в начале  
    node<****double****>\* front = deq\_get\_front(dq);  
    cout << "Передний элемент дека: " << front->m\_data << endl;  
   
    // удаление элементов дека с конца  
    deq\_pop\_back(dq);  
    deq\_pop\_back(dq);  
    deq\_pop\_back(dq);  
    deq\_pop\_back(dq);  
   
    // вывод дека  
    deq\_print(dq);  
   
    // удаление дека  
    deq\_delete(dq);  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              N O D E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// создание элемента дека  
template<typename T>  
node<T>\* node\_create(T \_data, node<T>\* \_next, node<T>\* \_prev)  
{  
    // создаем первый элемент в деке  
    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 дека  
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 дек  
template<typename T>****bool****deq\_is\_empty(deq<T>\* \_deq)  
{****return****\_deq->m\_begin == NULL;  
}  
   
// существует ли дек  
template<typename T>****bool****deq\_exists(deq<T>\* \_deq)  
{****return****\_deq != NULL;  
}  
   
// функция инициализации  
template<typename T>  
deq<T>\* deq\_create()  
{  
    // выделение памяти под дек  
    deq<T>\* \_deq =****new****deq<T>;  
   
    INFO("Дек был создан");  
   
    // возвращение адресса дека****return****\_deq;  
}  
   
// функция добавления элемента в конец дека  
template<typename T>****void****deq\_push\_back(deq<T>\* \_deq, T data)  
{  
    // выход, если дек не существует  
    DEQ\_NOT\_EXSISTS("PUSH");  
   
    // элемент добавления  
    node<T>\* new\_node;  
   
    // если дек пуст****if****(deq\_is\_empty(\_deq))  
    {  
        // создаем новый элемент дека  
        new\_node = node\_create<T>(data, NULL, NULL);  
   
        // записываем адресс нового элемента в дек  
        // в качестве первого элемента  
        \_deq->m\_begin = new\_node;  
        \_deq->m\_end = new\_node;  
    }  
    // если в деке есть элементы****else*** *{  
        // создаем элемент для вставки в дек  
        new\_node = node\_create<T>(  
            data, NULL, \_deq->m\_end);  
   
        // настравиваем последний элемент в деке  
        \_deq->m\_end->m\_next = new\_node;  
   
        // настраиваем сам дек  
        \_deq->m\_end = new\_node;  
    }  
   
    //увеличиваем количество элементов  
    \_deq->m\_size++;  
   
    INFO("Элемент добавлен в конец");  
}  
   
// функция удаления элемента из конца дека  
template<typename T>****void****deq\_pop\_back(deq<T>\* \_deq)  
{  
    // выходим, если дек не существует или пуст  
    DEQ\_NOT\_EXSISTS("POP");  
    DEQ\_EMPTY("POP");  
   
    // элемент удаления  
    node<T>\* to\_delete = \_deq->m\_end;  
   
    // если в деке минимум 2 элемента****if****(\_deq->m\_begin != to\_delete)  
    {  
        // настройка дека  
        \_deq->m\_end = to\_delete->m\_prev;  
   
        // разрыв связей с элементом удаления  
        to\_delete->m\_prev->m\_next = NULL;  
    }****else*** *{  
        // обнуляем адреса дека, указвающие  
        // на первый и последний элементы дека  
        \_deq->m\_begin = \_deq->m\_end = NULL;  
    }  
   
    // удаления самого элемента  
    node\_delete(to\_delete);  
   
    // уменьшение размера дека на 1  
    \_deq->m\_size--;  
   
    INFO("Последний элемент был удален");  
}  
   
template<typename T>  
node<T>\* deq\_get\_front(deq<T>\* \_deq)  
{  
    // выходим, если дек не существует  
    DEQ\_NOT\_EXSISTS\_RET("GET\_FRONT", NULL);****return****\_deq->m\_begin;  
}  
   
// удаление дека  
template<typename T>****void****deq\_delete(deq<T>\*& \_deq)  
{  
    // выходим, если дек не существует  
    DEQ\_NOT\_EXSISTS("DELETE");  
   
    // пока дек не пуст****while****(!deq\_is\_empty(\_deq))  
    {  
        // удаляем элементы дека с конца  
        deq\_pop\_back(\_deq);  
    }  
   
    // очищение памяти  
    delete \_deq;  
   
    // обнуление адреса  
    \_deq = NULL;  
   
    INFO("дек был удален")  
}  
   
// печать дека  
template<typename T>****void****deq\_print(deq<T>\* \_deq, ostream& \_out\_stream)  
{  
    // выходим, если дек не существует или пуст  
    DEQ\_NOT\_EXSISTS("PRINT");  
    DEQ\_EMPTY("PRINT");  
   
    // создаем элемент для чтения данных  
    node<T>\* cur\_el = \_deq->m\_begin;  
   
    // индекс элемента****int****i = 0;  
   
    // печать шапки таблицы  
    \_out\_stream << OUT\_W('\_', 22) << '\n';  
   
    // дополнительная информация о деке  
    \_out\_stream << "|\_размер:\_" <<  
        OUT\_W('\_', 9) << \_deq->m\_size << "\_|\n";  
    \_out\_stream << "|\_индекс\_|\_значение\_|\n";  
   
    // идем по деку, 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';  
}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Of LW5.cpp File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

# Тестирование программы

### Тест 1

Выполнение заранее созданного алгоритма.

# Вывод

Работа программы завершена на основании:

1) Полученные результаты совпали с ожидаемыми;

2) Считаем набор тестов полным.