**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №8

Дисциплина: Объектно-ориентированное программирование

по теме Создание шаблонов классов в С++.

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Буханов Д.Г.

**Белгород 2019**

Цель работы: Получение теоретических знаний о шаблонах классов в С++. Получение практических навыков по созданию классов-шаблонов С++.

Задания к лабораторной работе

1. Изучить теоретические сведения о шаблонах классов в С++.

2. Разработать программу в соответствии с заданным вариантом задания.

3. Оформить отчет.

Задание.

Реализовать шаблон класса в соответствии с указанным вариантом. Предусмотреть необходимые методы для работы со структурой данных, указанной в варианте. Предусмотреть исключительные ситуации, которые могут возникнуть в процессе работы.

Вариант 6

#include <iostream>

using namespace std;

template <class Type>

class ListEl{

Type data;

ListEl\* next\_link;

ListEl\* prev\_link;

public:

ListEl();

ListEl(Type data);

void set\_next\_link(ListEl\* link);

void set\_prev\_link(ListEl\* link);

void set\_data(Type data);

Type get\_data();

ListEl \*get\_next();

ListEl \*get\_prev();

void print();

};

template <class Type>

class DoubleLinkedList{

ListEl<Type>\* list\_start;

ListEl<Type>\* list\_end;

ListEl<Type>\* list\_current;

public:

DoubleLinkedList();

void add\_el(Type element);

Type pop\_el();

Type get\_el();

void ptr\_to\_start();

void ptr\_to\_end();

int is\_end();

void clear();

void next\_el();

void prev\_el();

void print();

};

template <class Type>

class Deck{

DoubleLinkedList<Type> list;

unsigned current\_size;

unsigned max\_size;

public:

Deck(unsigned size);

void push\_back(Type data);

void push\_front(Type data);

Type pop\_back();

Type pop\_front();

int is\_empty();

void clear();

};

//методы ListEl

template <class Type>

ListEl<Type>::ListEl(){

this->next\_link=nullptr;

this->prev\_link=nullptr;

};

template <class Type>

ListEl<Type>::ListEl(Type data):data(data){

this->next\_link=nullptr;

this->prev\_link=nullptr;

};

template <class Type>

void ListEl<Type>::set\_next\_link(ListEl<Type>\* link){

this->next\_link=link;

};

template <class Type>

void ListEl<Type>::set\_prev\_link(ListEl<Type>\* link){

this->prev\_link=link;

};

template <class Type>

void ListEl<Type>::set\_data(Type data){

this->data=data;

};

template <class Type>

Type ListEl<Type>::get\_data(){

return this->data;

};

template <class Type>

ListEl<Type> \*ListEl<Type>::get\_next(){

return this->next\_link;

};

template <class Type>

ListEl<Type> \*ListEl<Type>::get\_prev(){

return this->prev\_link;

};

template <class Type>

void ListEl<Type>::print(){

cout<<"Data: "<<this->data<<endl;

};

//Методы DoubleLinkedList

template <class Type>

DoubleLinkedList<Type>::DoubleLinkedList(){

this->list\_start= new ListEl<Type>();

this->list\_end= new ListEl<Type>();

this->list\_current=this->list\_start;

};

template <class Type>

void DoubleLinkedList<Type>::add\_el(Type element){

ListEl<Type> \*temp\_ptr = new ListEl<Type>(element);

if(this->list\_current->get\_next()==nullptr){

temp\_ptr->set\_next\_link(nullptr);

temp\_ptr->set\_prev\_link(list\_current);

}else{

temp\_ptr->set\_next\_link(this->list\_current->get\_next());

temp\_ptr->set\_prev\_link(list\_current);

this->list\_current->get\_next()->set\_prev\_link(temp\_ptr);

}

this->list\_current->set\_next\_link(temp\_ptr);

this->next\_el();

};

template <class Type>

Type DoubleLinkedList<Type>::pop\_el(){

Type temp=this->list\_current->get\_data();

ListEl<Type> \*pr=this->list\_current->get\_prev();

ListEl<Type> \*nx=this->list\_current->get\_next();

pr->set\_next\_link(nx);

nx->set\_prev\_link(pr);

return temp;

};

template <class Type>

Type DoubleLinkedList<Type>::get\_el(){

return this->list\_current->get\_data();

};

template <class Type>

void DoubleLinkedList<Type>::ptr\_to\_start(){

this->list\_current=this->list\_start;

this->next\_el();

};

template <class Type>

void DoubleLinkedList<Type>::ptr\_to\_end(){

this->list\_current=this->list\_end;

this->prev\_el();

};

template <class Type>

void DoubleLinkedList<Type>::clear(){

this->ptr\_to\_start();

while(this->list\_current!=this->list\_end){

this->pop\_el();

}

};

template <class Type>

int DoubleLinkedList<Type>::is\_end(){

return this->list\_current==this->list\_end;

};

template <class Type>

void DoubleLinkedList<Type>::next\_el(){

this->list\_current=this->list\_current->get\_next();

};

template <class Type>

void DoubleLinkedList<Type>::prev\_el(){

this->list\_current=this->list\_current->get\_prev();

};

template <class Type>

void DoubleLinkedList<Type>::print(){

ListEl<Type> \*temp\_pos=this->list\_current;

this->ptr\_to\_start();

int num=1;

while(this->list\_current!=this->list\_end){

cout<<"Element : "<<num<<endl;

this->list\_current->print();

this->next\_el();

num++;

};

this->list\_current=temp\_pos;

};

//методы Deck

template <class Type>

Deck<Type>::Deck(unsigned size):max\_size(size),size(0),list(DoubleLinkedList<Type>()){};

template <class Type>

void Deck<Type>::push\_back(Type data){

this->list.ptr\_to\_end();

this->list.add\_el(data);

this->current\_size++;

};

template <class Type>

void Deck<Type>::push\_front(Type data){

this->list.ptr\_to\_start();

this->list.add\_el(data);

this->current\_size++;

};

template <class Type>

Type Deck<Type>::pop\_back(){

this->list.ptr\_to\_end();

this->current\_size--;

return this->list.pop\_el();

};

template <class Type>

Type Deck<Type>::pop\_front(){

this->list.ptr\_to\_start();

this->current\_size--;

return this->list.pop\_el();

};

template <class Type>

int Deck<Type>::is\_empty(){

return this->current\_size==0;

};

template <class Type>

void Deck<Type>::clear(){

this->list.clear();

this->current\_size=0;

};