**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: Обьектно-ориентированное программирование

по теме «Создание шаблонов классов в С++»

Выполнил: ст. группы ВТ-22  
Ковалев Павел ВТ-22

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.

#include <QCoreApplication>

#include <QTextCodec>

//cout << QString::fromUtf8("").toLocal8Bit().data();

#include <iostream>

using namespace std;

template<typename Type>

struct **Node** { // структура узел

Type x; // информационное поле узла

Node \*next,\*prev; // указатели на структуру

};

template<typename Type>

class **DoubleList** { // класс двунаправленный список

public:

Node<Type> \*head,\*tail; // указатели на начало и конец списка

public:

**DoubleList**():head(NULL),tail(NULL) {} // инициализируем их пустым значением

~**DoubleList**(); // деструктор

void **AddElement**(Type y); // добавление элемента

void **ShowFromBegin**(); // вывод списка с начала

void **ShowFromEnd**(); // вывод списка с конца

void **Head**(); // вывод головы

void **Tail**(); // вывод хвоста

void **AddHead**(Type y); // добавить в начало списка

void **deletNumber**(int n); // удаление элемента по номеру

int **Size**(); // размер списка

int **findNumber**(int n); // вернуть элемент расположенный на n месте сверху

};

template<typename Type>

DoubleList<Type>::~**DoubleList**() {

while(head!=NULL) { // пока список не пуст

tail=head->next; // последний элемент равен следующему за 1 элементом

delete head; // удаляем 1

head=tail; // первым становится последний

}

}

template<typename Type>

void DoubleList<Type>::**AddElement**(Type y) {

Node<Type> \*temp=new Node<Type>; // выделяем память для нового элемента

temp->next=NULL; // указываем что за этим элементом нет других

temp->x=y; // присваиваем ему значение переданное аргументом

if(head!=NULL) { // пока список не пуст

temp->prev=tail; // следующий за нашим элементом элемент будет хвостом

tail->next=temp; // элемент следующий за хвостом наш

tail=temp; // наш элемент равен хвосту

} else { // если же список пуст

temp->prev=NULL; // тогда перед нашим элементом пусто

head=tail=temp; // 1 элемент равен последнему и равен нашему элементу

}

}

template<typename Type>

void DoubleList<Type>::**ShowFromBegin**() {

Node<Type> \*temp=head; // temp равен началу списка

while(temp!=NULL) { // пока список не пуст

cout<<temp->x<<" "; // выводим знание поля

temp=temp->next; // перемешаем на следующий элемент

}

}

template<typename Type>

void DoubleList<Type>::**ShowFromEnd**() {

Node<Type> \*temp=tail; // temp равен концу списка

while(temp!=NULL) { // пока элементы есть

cout<<temp->x<<" "; // выводим значение

temp=temp->prev; // перемещаем элемент на предыдущий

}

}

template<typename Type>

void DoubleList<Type>::**Head**() {

if(head!=NULL) {

cout<<head->x<<endl;

}

}

template<typename Type>

void DoubleList<Type>::**Tail**() {

if(tail!=NULL) {

cout<<tail->x<<endl;

}

}

template<typename Type>

void DoubleList<Type>::**AddHead**(Type y) {

Node<Type> \*temp=new Node<Type>;

temp->x=y;

if(temp!=NULL) {

temp->next=head;

head=temp;

} else {

head=tail=temp;

}

}

template<typename Type>

void DoubleList<Type>::**deletNumber**(int n) {

Node<Type> \*temp=head;

if(temp!=NULL && n<=Size()) {

for(int i=0; i!=n; i++) {

temp=temp->next;

}

if(temp==head) {

head=temp->next;

} else {

temp->prev->next=temp->next;

}

delete temp;

}

}

template<typename Type>

int DoubleList<Type>::**Size**() {

Node<Type> \*temp=head;

int c=0;

while(temp!=NULL) {

temp=temp->next;

c++;

}

return c;

}

template<typename Type>

int DoubleList<Type>::**findNumber**(int n) {

Node<Type> \*temp=head;

while(temp!=NULL && n<=Size()) {

for(int i=0; i!=n; i++) {

temp=temp->next;

}

return temp->x;

}

}

int **main**(int argc, char \*argv[])

{

QCoreApplication a(*argc*, argv);

#ifdef Q\_OS\_WIN32

QTextCodec::setCodecForLocale(QTextCodec::codecForName("IBM 866"));

#endif

#ifdef Q\_OS\_LINUX

QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));

#endif

DoubleList<int> list;

for(int i=0; i!=5; i++){

list.AddElement(i);

}

list.ShowFromBegin();

cout<<endl;

list.ShowFromEnd();

cout<<endl;

list.Head();

list.Tail();

list.AddElement(10);

cout<<endl;

list.ShowFromBegin();

list.AddHead(7);

cout<<endl;

list.ShowFromBegin();

cout<<endl;

list.deletNumber(4);

list.ShowFromBegin();

return a.exec();

}