Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №7

Тема: «Шаблоны классов»

Выполнили

Студент группы РИС-22-21б

Анистратов А. Д.

Проверил доц. Кафедры ИТАС

Полякова Ольга Андреевна

Пермь 2023

# Постановка задачи

1. Определить шаблон класса-контейнера (см. лабораторную работу №6).
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
5. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.
6. Реализовать пользовательский класс (см. лабораторную работу №3).
7. Перегрузить для пользовательского класса операции ввода-вывода.
8. Перегрузить операции необходимые для выполнения операций контейнерного класса.
9. Инстанцировать шаблон для пользовательского класса.
10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.

ВАРИАНТ 15:

Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции:

[] – доступа по индексу;

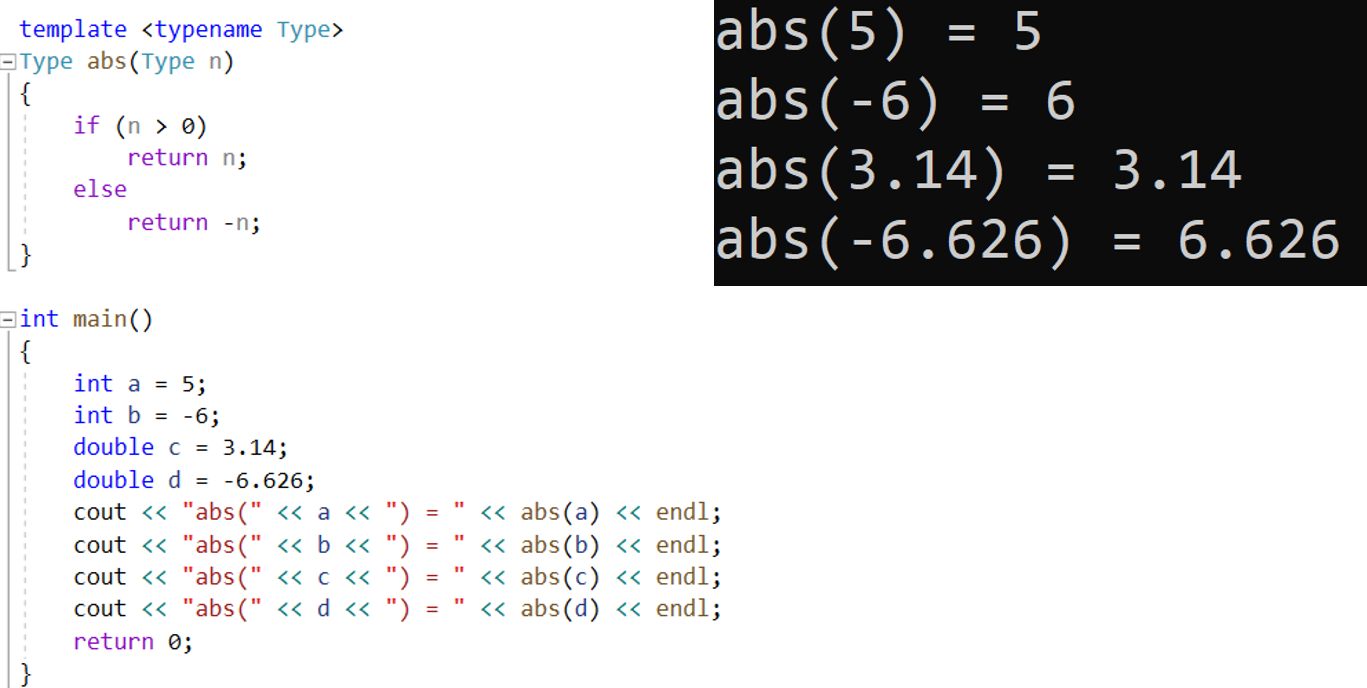
int() – определение размера списка;

\* вектор – умножение элементов списков a[i]\*b[i];

Пользовательский класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

**Контрольные вопросы**

* 1. В чем смысл использования шаблонов?

Для создания функций / классов с разным типом данных.

* 1. Каковы синтаксис/семантика шаблонов функций?

Для функции:

Template<параметры\_шаблона>

Тип или параметр шаблона заголовок\_функции(…)

{тело функции}

* 1. Каковы синтаксис/семантика шаблонов классов?

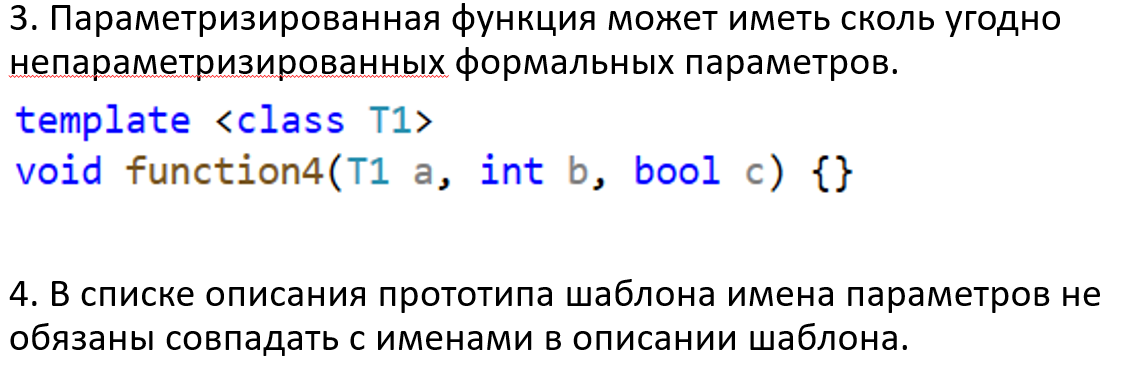
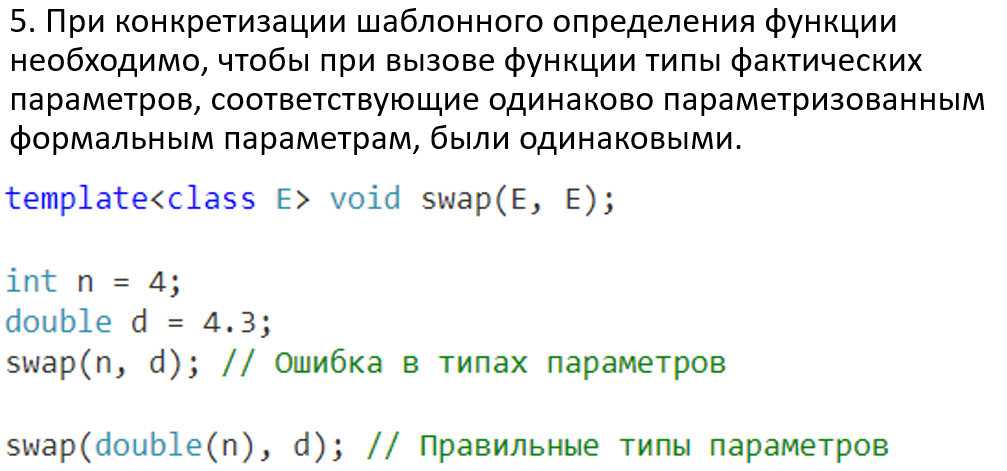
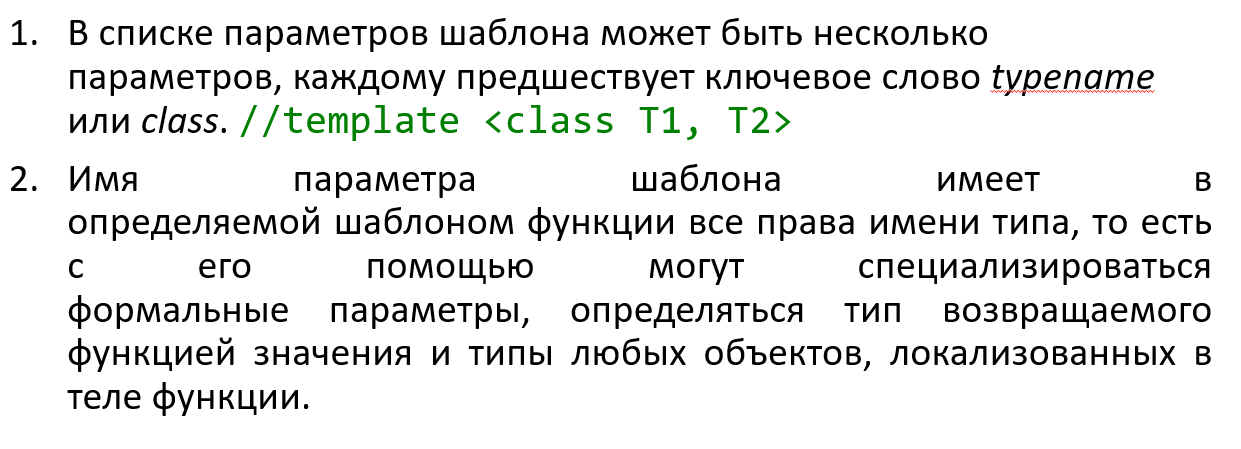
Template<параметры\_шаблона>

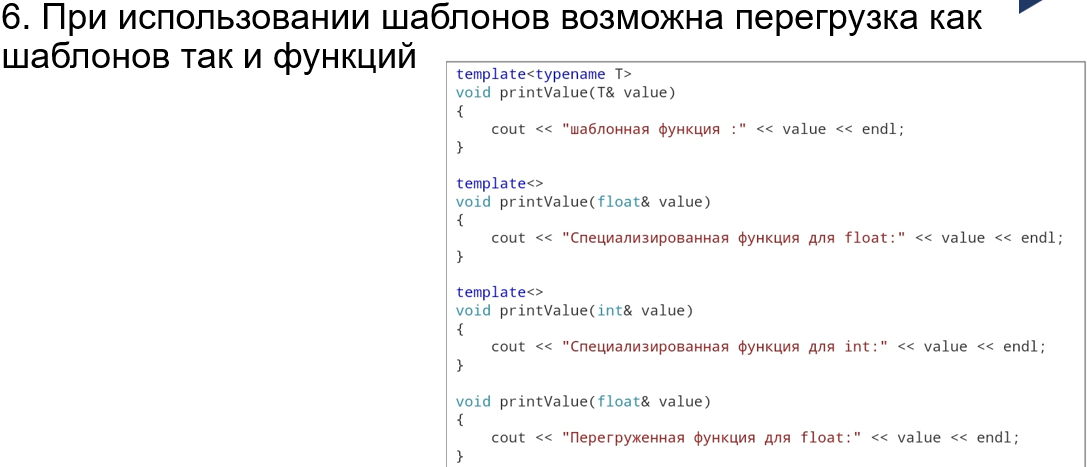
class имя\_класса

{тело класса}

* 1. Что такое параметры шаблона функции?

Это те типы данных, которые будут передаваться в функцию.

* 1. Перечислите основные свойства параметров шаблона функции.



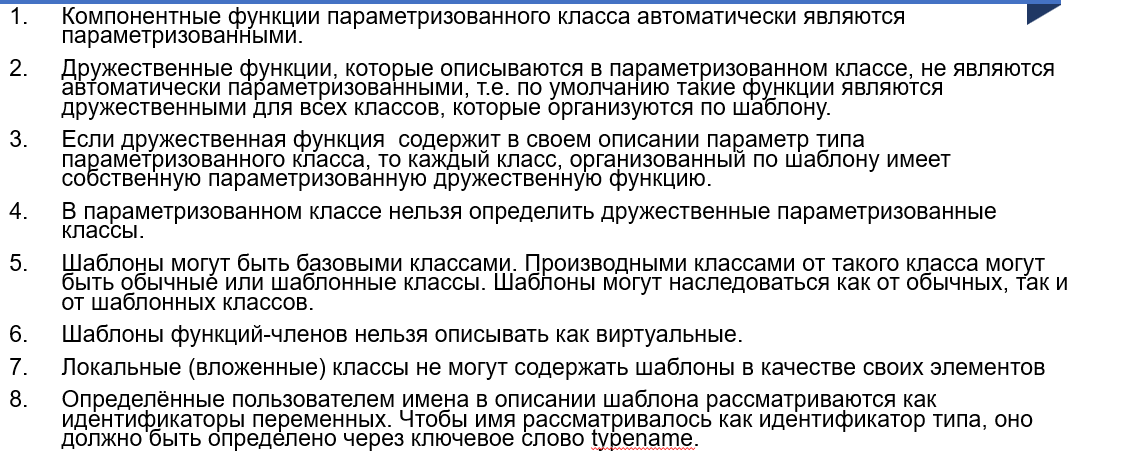
* 1. Как записывать параметр шаблона?

Template <typename / class T>

* 1. Можно ли перегружать параметризованные функции?

Да, можно. Смотреть пункт 5.6.

* 1. Перечислите основные свойства параметризованных классов.



* 1. Все ли компонентные функции параметризованного класса являются параметризованными?

Да. Автоматически. 8.1

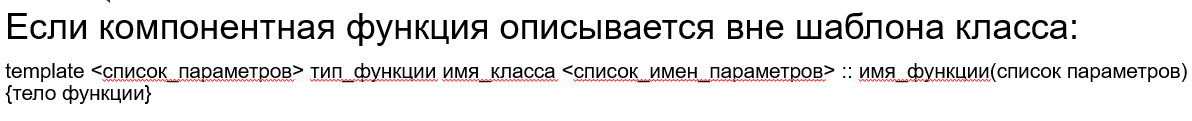
* 1. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

Нет. 8.2

* 1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Нет. 8.6

* 1. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?



* 1. Что такое инстанцирование шаблона?

Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона называется **инстанцированием** шаблона.

* 1. На каком этапе происходит генерирование определения класса по шаблону?

При включении шаблона класса в программу никакие классы на самом деле не генерируются до тех пор, пока не будет создан экземпляр шаблонного класса, в котором вместо параметра шаблона указывается конкретный тип. Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.

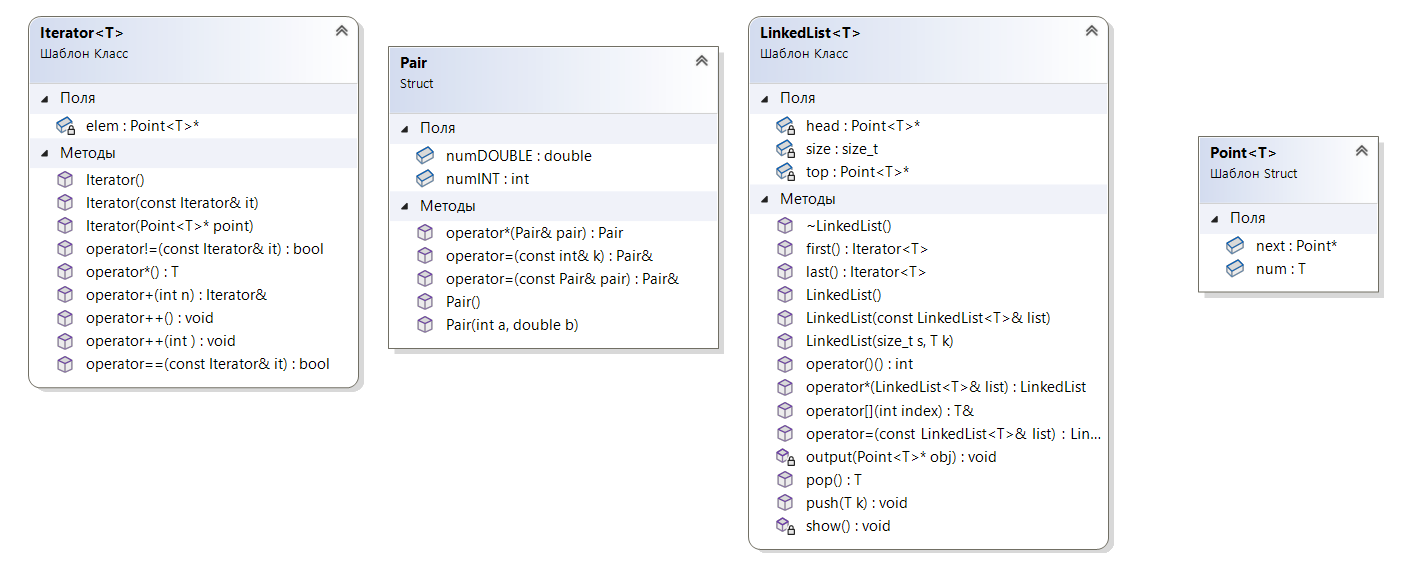
Point <int> a(13,15);

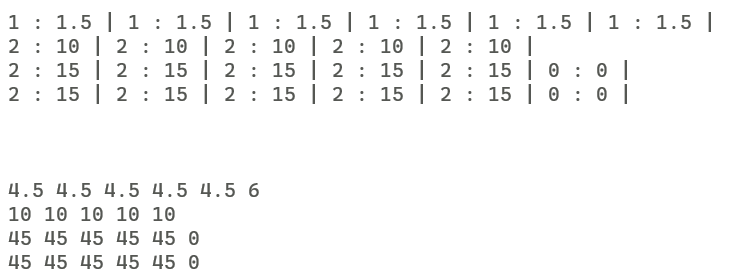
Point <float>\*pa=new Point<float>(10.1,0.55);

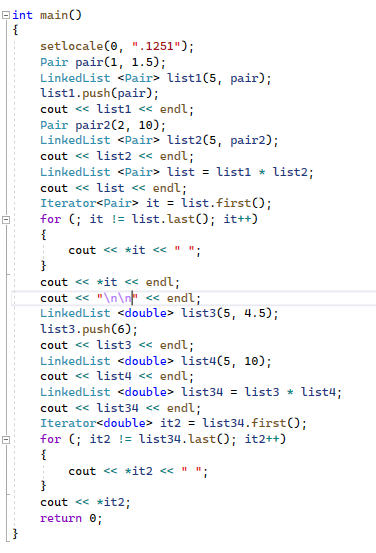
Встретив такие объявления, компилятор генерирует код исходного класса.

В проекте, состоящем из нескольких файлов, определение шаблона класса обычно выносится в отдельный файл. Но для того, чтобы инстанцировался конкретный экземпляр шаблона класса необходимо, чтобы определение шаблона находилось в одной единице трансляции с этим экземпляром. Поэтому все определение шаблонного класса размещается в заголовочном файле

**UML таблица**

****

**Скриншоты работы программы**

****

**Код программы**

**Main.cpp**

#include "LinkedList.h"

int main()

{

setlocale(0, ".1251");

Pair pair(1, 1.5);

LinkedList <Pair> list1(5, pair);

list1.push(pair);

cout << list1 << endl;

Pair pair2(2, 10);

LinkedList <Pair> list2(5, pair2);

cout << list2 << endl;

LinkedList <Pair> list = list1 \* list2;

cout << list << endl;

Iterator<Pair> it = list.first();

for (; it != list.last(); it++)

{

cout << \*it << " ";

}

cout << \*it << endl;

cout << "\n\n\n\n\n" << endl;

LinkedList <double> list3(5, 4.5);

list3.push(6);

cout << list3 << endl;

LinkedList <double> list4(5, 10);

cout << list4 << endl;

LinkedList <double> list34 = list3 \* list4;

cout << list34 << endl;

Iterator<double> it2 = list34.first();

for (; it2 != list34.last(); it2++)

{

cout << \*it2 << " ";

}

cout << \*it2;

return 0;

}

**LinkedList.h**

#pragma once

#include <iostream>

#include <algorithm> // для min/max

using namespace std;

//-------------------------//

// num в связи Point,

// если задан <Pair>

//-------------------------//

struct Pair

{

Pair() {}

Pair(int a, double b)

{

numINT = a;

numDOUBLE = b;

}

int numINT = 0;

double numDOUBLE = 0;

// для присвоения нового Pair или одного числа для пары.

Pair& operator =(const Pair& pair)

{

this->numINT = pair.numINT;

this->numDOUBLE = pair.numDOUBLE;

return \*this;

};

Pair& operator =(const int& k)

{

this->numINT = k;

this->numDOUBLE = k;

return \*this;

}

// вывод

friend ostream& operator<<(ostream& out, const Pair& pair)

{

out << pair.numINT << " : " << pair.numDOUBLE << " |";

return out;

}

friend istream& operator>>(istream& in, Pair& pair)

{

in >> pair.numINT >> pair.numDOUBLE;

return in;

}

// умножение двух пар чисел

Pair operator \* (Pair& pair)

{

Pair new\_pair;

new\_pair.numINT = this->numINT \* pair.numINT;

new\_pair.numDOUBLE = this->numDOUBLE \* pair.numDOUBLE;

return new\_pair;

}

};

//-------------------------//

// связь списка

//-------------------------//

template<typename T>

struct Point

{

T num; // int, float, double, Pair

Point\* next = nullptr; // указатель на следующий элемент списка

};

//-------------------------//

// итератор списка

//-------------------------//

template <typename T>

class Iterator

{

private:

Point<T>\* elem = nullptr; // указатель на связь

public:

Iterator() { elem = nullptr; }

Iterator(Point<T>\* point) { elem = point; }

Iterator(const Iterator& it) { elem = it.elem; };

bool operator == (const Iterator& it) { return elem == it.elem; }

bool operator != (const Iterator& it) { return elem != it.elem; }

void operator ++ () { elem = elem->next; }; // ++i - смещение вправо на 1

void operator ++ (int) { elem = elem->next; }; // i++ - смещение вправо на 1

//---------// i+n - смещение вправо на n

Iterator& operator + (int n)

{

for (int i = 0; i < n && elem->next != nullptr; i++)

elem = elem->next;

return \*this;

}

T operator \* () const { return elem->num; }; // разыменование итератора (вывод)

};

//-------------------------//

// связанный список

//-------------------------//

template <typename T>

class LinkedList

{

private:

size\_t size = 0; // размер списка

Point<T>\* head = nullptr; // указатель на первый элемент

Point<T>\* top = nullptr; // указатель на ласт элемент

//-------------------------------вывод----------------------------------------//

void output(Point<T>\* obj) const;

void show() const;

//---------------------------------------------------------------------------//

public:

friend ostream& operator<<(ostream& out, const LinkedList<T>& list)

{

list.show();

return out;

}

friend istream& operator>>(istream& in, LinkedList<T>& list)

{

T k;

in >> k;

list.push(k);

return in;

}

LinkedList(); // конструктор

LinkedList(size\_t s, T k); // констуктор: s - размер, k - значение по умолчанию

~LinkedList(); // деструктор

LinkedList(const LinkedList<T>& list); // конструктор копирования

T& operator [](int index) const; // доступ по индексу

LinkedList& operator =(const LinkedList<T>& list); // оператор присваивания

LinkedList operator \* (LinkedList<T>& list); // умножение списка на список

int operator ()() { return size; }; // возвращение длины списка

void push(T k); // добавление в конец списка

T pop(); // удаление первого элемента

Iterator<T> first() { return Iterator<T>(head); } // возвращает указатель на первый элемент

Iterator<T> last() { return Iterator<T>(top); } // вовзращает указаетль на последний элемент

};

// конструктор

template <typename T>

LinkedList<T>::LinkedList()

{

head = nullptr;

top = nullptr;

size = 0;

}

// конструктор с параметрами

template <typename T>

LinkedList<T>::LinkedList(size\_t s, T k)

{

size = s;

if (size != 0) {

//---------создаем голову и передаем ей значение k---------//

Point<T>\* obj = new Point<T>;

obj->num = k;

head = obj;

//--------------------------------------------------------//

//-------создаем следующие size - 1 элементы--------------//

Point<T>\* temp = head;

for (int i = size - 1; i != 0; i--)

{

obj = new Point<T>;

temp->next = obj;

obj->num = k;

temp = obj;

}

// хвостик + итераторы

top = obj;

}

}

// добавление в конец

template <typename T>

void LinkedList<T>::push(T k)

{

//------------------------------------------//

// Если размер 0, то задаем голову = хвосту

//------------------------------------------//

if (size == 0)

{

head = new Point<T>;

head->num = k;

top = head;

}

//------------------------------------------//

// Иначе через хвост добавляем новый элемент

//------------------------------------------//

else {

Point<T>\* obj = new Point<T>;

top->next = obj;

obj->num = k;

top = obj;

}

// увеличиваем размер, итератор на конец

size++;

}

// удаление начального элемента

template <typename T>

T LinkedList<T>::pop()

{

// создаем промежуточную связь, удаляем голову, перемещаем голову

T k = head->num;

Point<T>\* temp = head;

head = head->next;

size--;

delete temp;

return k;

}

// деструктор

template <typename T>

LinkedList<T>::~LinkedList()

{

cout << endl << "Вызван деструктор!" << endl << endl;

while (head != nullptr)

{

pop();

}

}

//----------------------------вывод----------------------------------//

template <typename T>

void LinkedList<T>::output(Point<T>\* obj) const

{

cout << obj->num << " ";

if (obj == top) return;

output(obj->next);

}

template <typename T>

void LinkedList<T>::show() const

{

if (size == 0) cout << "Список пуст" << endl;

else output(head);

}

//-------------------------------------------------------------------//

// индекс

template <typename T>

T& LinkedList<T>::operator[](int index) const

{

if (index < size && index >= 0)

{

Point<T>\* temp = head;

int count = 0;

while (temp != nullptr)

{

if (count == index)

return temp->num;

temp = temp->next;

++count;

}

}

cout << endl << endl;

perror("Ошибка индексирования");

cout << endl;

exit(1);

}

// конструктор копирования

template <typename T>

LinkedList<T>::LinkedList(const LinkedList<T>& list)

{

for (int i = 0; i < list.size; i++)

{

this->push(list[i]);

}

}

// присваивание

template <typename T>

LinkedList<T>& LinkedList<T>::operator =(const LinkedList<T>& list)

{

// создаем size\_temp, т.к. pop уменьшает size, нужно застанить

int size\_temp = size;

// удаляем все элементы

for (int i = 0; i < size\_temp; i++)

{

this->pop();

}

// копируем через push()

for (int i = 0; i < list.size; i++)

{

this->push(list[i]);

}

return \*this;

}

// умножение

template <typename T>

LinkedList<T> LinkedList<T>::operator \* (LinkedList<T>& list)

{

// создаем новый список по макс рзамеру, обнуляем через второй параметр

T nulik;

nulik = 0;

LinkedList<T> new\_list;

for (int i = 0; i < min((int)size, list()); i++)

{

new\_list.push((\*this)[i] \* list[i]);

}

for (int i = 0; i < max((int)size, list()) - min((int)size, list()); i++)

{

new\_list.push(nulik);

}

return new\_list;

}

LinkedList.cpp

Ничего, т.к. шаблон и класс должны быть в одном файле.