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

Студенты группы РИС-22-2б

Худеньких В.Д.

Проверил доц. Кафедры ИТАС

Полякова О.А.

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.
  2. 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.
  2. 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?
  2. 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.

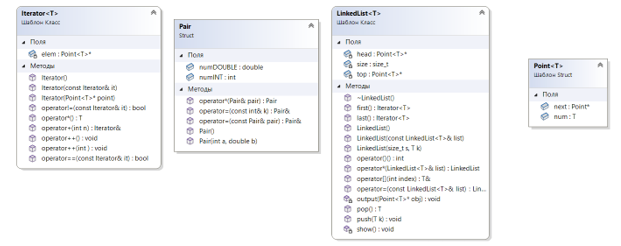
Point <int> a(13,15);

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

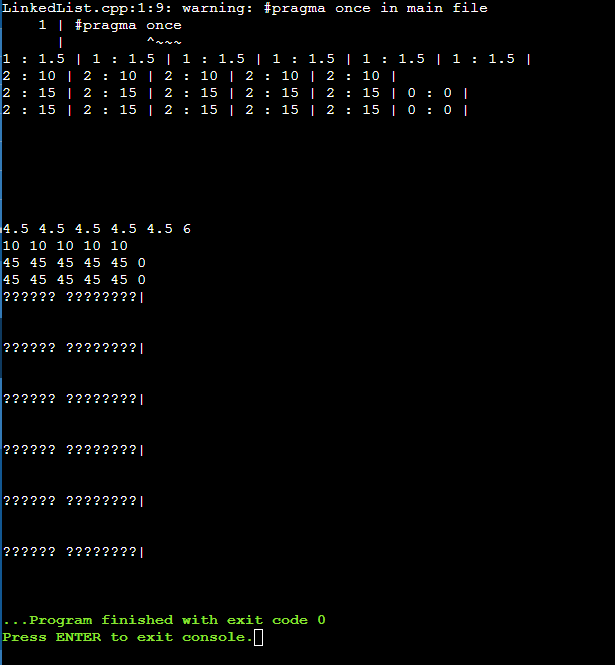
Встретив такие объявления, компилятор генерирует код исходного класса.

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>

using namespace std;

struct **Pair**

{

**Pair**() {}

**Pair**(int a, double b)

{

numINT = a;

numDOUBLE = b;

}

int numINT = 0;

double numDOUBLE = 0;

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;

}

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;

}

Pair operator \* (Pair& pair)

{

Pair new\_pair;

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

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

return new\_pair;

}

};

template<typename T>

struct **Point**

{

T num;

Point\* next = nullptr;

};

template <typename T>

class **Iterator**

{

private:

Point<T>\* elem = nullptr;

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; };

void operator ++ (int) { elem = elem->next; };

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; };

};

template <typename T>

class **LinkedList**

{

private:

size\_t size = 0;

Point<T>\* head = nullptr;

Point<T>\* top = nullptr;

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**();

**LinkedList**(size\_t s, T k);

~**LinkedList**();

**LinkedList**(const LinkedList<T>& list);

T& operator [](int index) const;

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

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

int operator ()() { return size; };

void **push**(T k);

T **pop**();

Iterator<T> **first**() { return Iterator<T>(head); }

Iterator<T> **last**() { return Iterator<T>(top); }

};

template <typename T>

LinkedList<T>::**LinkedList**()

{

head = nullptr;

top = nullptr;

size = 0;

}

template <typename T>

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

{

size = s;

if (size != 0) {

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

obj->num = k;

head = obj;

Point<T>\* temp = head;

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

{

obj = new Point<T>;

temp->next = obj;

obj->num = k;

temp = obj;

}

top = obj;

}

}

template <typename T>

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

{

if (size == 0)

{

head = new Point<T>;

head->num = k;

top = head;

}

else {

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

top->next = obj;

obj->num = k;

top = obj;

}

size++;

}

template <typename T>

T LinkedList<T>::**pop**()

{

T k = head->num;

Point<T>\* temp = head;

head = head->next;

size--;

delete temp;

return k;

}

template <typename T>

LinkedList<T>::~**LinkedList**()

{

cout << endl << "?????? ??????????!" << endl << endl;

while (head != nullptr)

{

pop();

}

}

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 << "?????? ????" << endl;

else output(*head*);

}

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("?????? ??????????????");

cout << endl;

exit(1);

}

template <typename T>

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

{

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

{

this->push(list[i]);

}

}

template <typename T>

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

{

int size\_temp = size;

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

{

this->pop();

}

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

{

this->push(list[i]);

}

return \*this;

}

template <typename T>

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

{

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;

}