Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский

политехнический университет»

Лабораторная работа №7

«Шаблоны классов»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Шаблоны классов

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Реализация шаблона класса-контейнера.

**Постановка задачи**

1. Определить шаблон класса-контейнера (см. лабораторную работу

№6).

1. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
2. Перегрузить операции, указанные в варианте.
3. 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).

1. Перегрузить для пользовательского класса операции ввода-

вывода.

1. Перегрузить операции необходимые для выполнения операций контейнерного класса.
2. Инстанцировать шаблон для пользовательского класса.
3. Написатьтестирующуюпрограмму,иллюстрирующую

выполнение операций для контейнера, содержащего элементы пользовательского класса.

**Задание**

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.

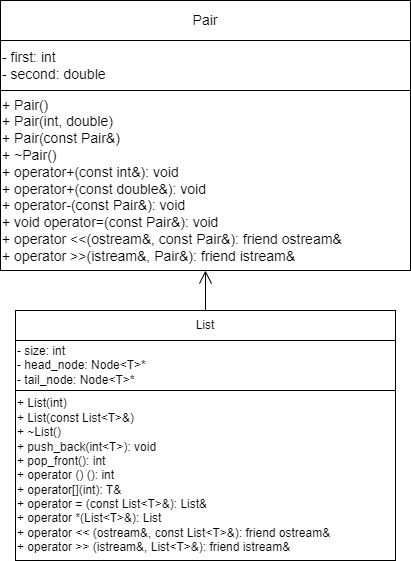
**Анализ задачи**

1. Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.
2. В классе Pair необходимо прописать операторы перегрузки для необходимых операций.
3. Для реализации класса List необходимо создать структуру Node.

При этом структура должна быть шаблонной, что бы она могла хранить разные типы данных.

1. Для класса List добавить определение методов параметризированного класса.
2. Класс также включает в себя конструктор для создания списка с заданным размером и данными, копирования списка и деструкторы для очистки памяти.
3. В основной функции необходимо продемонстрировать возможности шаблонного класса.

**UML диаграмма**

****

**Код**

Файл ООП7.cpp

#include <iostream>

using namespace std;

#include "List.h"

#include "Pair.h"

int main()

{

system("chcp 1251>null");

List<int> list1(5);

List<float> list2(5);

List<double> list3(5);

List<Pair> list4(3);

List<int> list5(5);

cout << "list1 int\n";

cin >> list1;

cout << list1;

cout << "\nВывод элемента под номером 1: ";

cout << list1[0] << "\n";

cout << "\nlist2 float\n";

cin >> list2;

cout << list2;

cout << "\nВывод элемента под номером 3: ";

cout << list2[2] << "\n";

cout << "\nlist3 double\n";

cin >> list3;

cout << list3 << "\n";

cout << "\nlist4 Pair\n";

cin >> list4;

cout << list4;

cout << "\nВывод элемента под номером 2: \n";

cout << list4[1] << "\n";

list4[1] + 100;

list4[1] + 99.3;

cout << list4 << "\n";

cout << "\nlist5 = list1\n";

list5 = list1;

cout << list5 << "\n";

cout << "\nlist1 \* list5\n" << list1 \* list5;

return 0;

}

Файл List.h

#pragma once

#include <iostream>

template <typename T>

struct Node

{

T data;

Node<T>\* ptr\_to\_next\_node = nullptr;

};

template <typename T>

class List

{

private:

int size;

Node<T>\* head\_node;

Node<T>\* tail\_node;

public:

List(int);

List(const List<T>&);

~List();

void push\_back(T);

void pop\_front();

int operator () ();

T& operator[](int);

List<T>& operator = (const List<T>&);

List<T> operator \*(List<T>&);

friend std::ostream& operator << (std::ostream& stream, const List<T>& list)

{

std::cout << "Вывод элементов: \n";

Node<T>\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

stream << current\_node->data << ' ';

current\_node = current\_node->ptr\_to\_next\_node;

}

return stream;

}

friend std::istream& operator >> (std::istream& stream, List<T>& list)

{

std::cout << "Введите элементы: ";

Node<T>\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

stream >> current\_node->data;

current\_node = current\_node->ptr\_to\_next\_node;

}

return stream;

}

};

template <typename T>

List<T>::List(int size)

{

if (size > 0)

{

Node<T>\* node = new Node<T>;

this->head\_node = node;

this->tail\_node = node;

for (int i = 1; i < size; i++)

{

Node<T>\* New\_Node = new Node<T>;

tail\_node->ptr\_to\_next\_node = New\_Node;

tail\_node = New\_Node;

}

this->size = size;

}

else

{

this->head\_node = nullptr;

this->tail\_node = nullptr;

}

}

template <typename T>

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

{

this->size = 0;

Node<T>\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

push\_back(current\_node->data);

current\_node = current\_node->ptr\_to\_next\_node;

}

}

template <typename T>

List<T>::~List()

{

while (head\_node != nullptr)

{

this->pop\_front();

}

}

template <typename T>

void List<T>::push\_back(T data)

{

Node<T>\* New\_Node = new Node<T>;

New\_Node->data = data;

if (this->head\_node == nullptr)

{

this->head\_node = New\_Node;

this->tail\_node = New\_Node;

}

else

{

this->tail\_node->ptr\_to\_next\_node = New\_Node;

this->tail\_node = New\_Node;

}

this->size++;

}

template <typename T>

void List<T>::pop\_front()

{

if (this->tail\_node != nullptr)

{

Node<T>\* current\_node = this->head\_node;

head\_node = current\_node->ptr\_to\_next\_node;

this->size--;

}

}

template <typename T>

int List<T>::operator () ()

{

return this->size;

}

template <typename T>

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

{

Node<T>\* current\_node = this->head\_node;

for (int i = 0; i != index; i++)

{

current\_node = current\_node->ptr\_to\_next\_node;

}

return current\_node->data;

}

template <typename T>

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

{

if (this != &list)

{

while (head\_node != nullptr)

{

this->pop\_front();

}

size = 0;

Node<T>\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

push\_back(current\_node->data);

current\_node = current\_node->ptr\_to\_next\_node;

}

}

return \*this;

}

template <typename T>

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

{

int temp\_size;

if (this->size > list.size)

{

temp\_size = list.size;

}

else

{

temp\_size = this->size;

}

List<T> temp\_list(temp\_size);

for (int i = 0; i < temp\_size; i++)

{

temp\_list[i] = (\*this)[i] \* list[i];

}

return temp\_list;

}

Файл Pair.h

#pragma once

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int, double);

Pair(const Pair&);

~Pair() {}

void operator - (const Pair&);

void operator + (const int&);

void operator + (const double&);

void operator = (const Pair&);

friend ostream& operator << (ostream&, const Pair&);

friend istream& operator >> (istream&, Pair&);

};

Pair::Pair()

{

this->first = 0;

this->second = 0.0;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

void Pair::operator-(const Pair& other)

{

this->first -= other.first;

this->second -= other.second;

}

void Pair::operator+(const int& x)

{

this->first += x;

}

void Pair::operator+(const double& x)

{

this->second += x;

}

void Pair::operator=(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

ostream& operator<<(ostream& stream, const Pair& pair)

{

stream << "(" << pair.first << ":" << pair.second << ")";

return stream;

}

istream& operator>>(istream& stream, Pair& pair)

{

int first; double second;

stream >> first >> second;

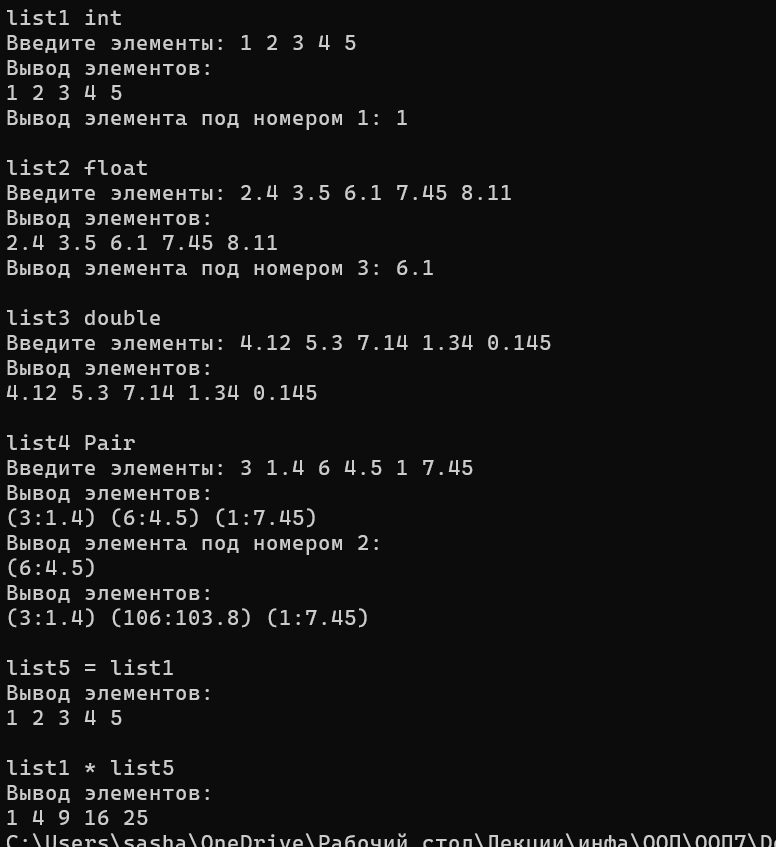
pair.first = first;

pair.second = second;

return stream;

}

**Результаты работы**



**GitHub**

https://github.com/Korovay4ik/Laboratory-works

**Контрольные вопросы**

1. **В чем смысл использования шаблонов?**

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные.

1. **Каковы синтаксис/семантика шаблонов функций?**

template<параметры\_шаблона>

заголовок функции

{тело функции}

Пример:

template<typename type>

type abs(type x)

{

if (x<0) return -x;

else return x;

}

1. **Каковы синтаксис/семантика шаблонов классов?**

template<параметры\_шаблона>

class имя\_класса

{ ... };

пример:

template<class T>

class Point

{

T x, y;

public:

Point(T x = 0, T y = 0):x(x), y(y){}

void Show();

{;

1. **Что такое параметры шаблона функции?**

Параметры шаблона функции — это типы или значения, которые определяют конкретную версию функции. Они указываются внутри угловых скобок после ключевого слова template перед именем функции.

1. **Перечислите основные свойства параметров шаблона**

**функции.**

* шаблоны не могут быть виртуальными
* шаблоны могут содержать статические элементы, дружественные функции и классы
* шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

1. **Как записывать параметр шаблона?**

Параметр шаблона записывается внутри угловых скобок после ключевого слова template, например <int>, <double>, <T>, где T - это псевдоним типа, который будет определен позже.

1. **Можно ли перегружать параметризованные функции?**

Да, можно перегружать параметризованные функции. Перегрузка

осуществляется так же, как и для обычных функций, с учетом параметров

шаблона.

1. **Перечислите основные свойства параметризованных классов.**

* Параметры шаблона могут быть как значениями, так и типами.

Например, можно создать шаблон класса с параметром, который задает размер массива.

* Параметры шаблона могут быть любыми типами данных, включая примитивные типы, пользовательские типы и другие шаблоны классов.

-Шаблонные классы могут быть унаследованы от других шаблонных классов и могут иметь шаблонные функции-члены.

-Параметры шаблона могут использоваться внутри класса для объявления переменных, функций и типов.

-Каждый конкретный экземпляр параметризованного класса является отдельным типом, который определяется параметрами шаблона.

1. **Все ли компонентные функции параметризованного**

**класса** **являются параметризованными?**

Да, все компонентные функции (методы) параметризированного класса

являются параметризованными, то есть они используют те же параметры

шаблона, что и сам класс.

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

Да

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

Да, шаблоны классов могут содержать виртуальные компонентные

функции. Виртуальность работает так же, как и для обычных классов, с учетом

параметров шаблона.

**12.** **Как** **определяются** **компонентные** **функции**

**параметризованных классов вне определения шаблона класса?**

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса с помощью использования template<class T> перед каждым определением функции, где T - это параметр шаблона.

1. **Что такое инстанцирование шаблона?**

Инстанцирование шаблона — это процесс генерации конкретной версии шаблона для определенных типов. Оно происходит автоматически, когда компилятор встречает использование шаблона с конкретными типами.

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

Генерирование определения класса по шаблону происходит во время

компиляции, когда компилятор встречает использование шаблона класса с

конкретными типами.