Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Лабораторная работа №7

" Шаблоны классов "

Вариант: 12

Выполнил студент ИВТ-24-2б:
Шишкин Максим Григорьевич

_____
(дата, подпись)

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

_____
(дата, подпись)

Пермь 2025

Содержание

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

[] – доступа по индексу;

() – определение размера вектора;

+ число – добавляет константу ко всем элементам вектора;
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.

2 Код на C++

Pair.h Pair.cpp List.h List.inl main.cpp

7лабапеределка

```cpp
#include <iostream>
#include <locale>
#include "Pair.h"
#include "List.h"

int main() {
    setlocale(LC_ALL, "ru");

    List<int> intList;
    cin >> intList;
    cout << "intList:\n" << intList;
    cout << "Размер: " << intList() << endl;
    cout << "intList + 5:\n" << (intList + 5) << endl;

    List<Pair> pairList;
    cin >> pairList;
    cout << "pairList:\n" << pairList;
    cout << "Размер: " << pairList() << endl;
    cout << "pairList + 2.5:\n" << (pairList + 2.5) << endl;

    return 0;
}
```

Pair.h Pair.cpp List.h List.inl main.cpp

7лабапеределка

```cpp
#pragma once
#include <iostream>
using namespace std;

class Pair {
    int first;
    double second;

public:
    Pair(int f = 0, double s = 0.0);

    friend ostream& operator<<(ostream& os, const Pair& p);
    friend istream& operator>>(istream& is, Pair& p);

    Pair operator+(double val) const;
};
```

Pair.h Pair.cpp List.h List.inl main.cpp

7лабапеределка

```cpp
#include "Pair.h"

Pair::Pair(int f, double s) : first(f), second(s) {}

ostream& operator<<(ostream& os, const Pair& p) {
    os << p.first << ":" << p.second;
    return os;
}

istream& operator>>(istream& is, Pair& p) {
    is >> p.first >> p.second;
    return is;
}

Pair Pair::operator+(double val) const {
    return Pair(first, second + val);
}
```

Pair.h Pair.cpp List.h List.inl main.cpp

7лабапеределка (Глобальная о

```h
#pragma once
#include <iostream>
using namespace std;

const int MAX_SIZE = 100;

template <typename T>
class List {
    int keys[MAX_SIZE];
    T values[MAX_SIZE];
    int size;

public:
    List();
    List(const List& other);
    ~List() = default;

    List& operator=(const List& other);
    T& operator[](int index);
    int operator()() const;
    List operator+(double val) const;

    template <typename U>
    friend ostream& operator<<(ostream& os, const List<U>& list);

    template <typename U>
    friend istream& operator>>(istream& is, List<U>& list);
};

#include "List.inl"
```

Pair.h Pair.cpp List.h List.inl main.cpp

7лабапеределка

```inl
template <typename T>
List<T>::List() : size(0) {}

template <typename T>
List<T>::List(const List& other) {
    size = other.size;
    for (int i = 0; i < size; ++i) {
        keys[i] = other.keys[i];
        values[i] = other.values[i];
    }
}

template <typename T>
List<T>& List<T>::operator=(const List& other) {
    if (this != &other) {
        size = other.size;
        for (int i = 0; i < size; ++i) {
            keys[i] = other.keys[i];
            values[i] = other.values[i];
        }
    }
    return *this;
}

template <typename T>
T& List<T>::operator[](int index) {
    for (int i = 0; i < size; ++i) {
        if (keys[i] == index) {
            return values[i];
        }
    }
    keys[size] = index;
    values[size] = T();
    return values[size++];
}

template <typename T>
int List<T>::operator()() const {
    return size;
}

template <typename T>
List<T> List<T>::operator+(double val) const {
    List<T> result;
    for (int i = 0; i < size; ++i) {
        result.keys[i] = keys[i];
        result.values[i] = values[i] + val;
    }
    result.size = size;
    return result;
```

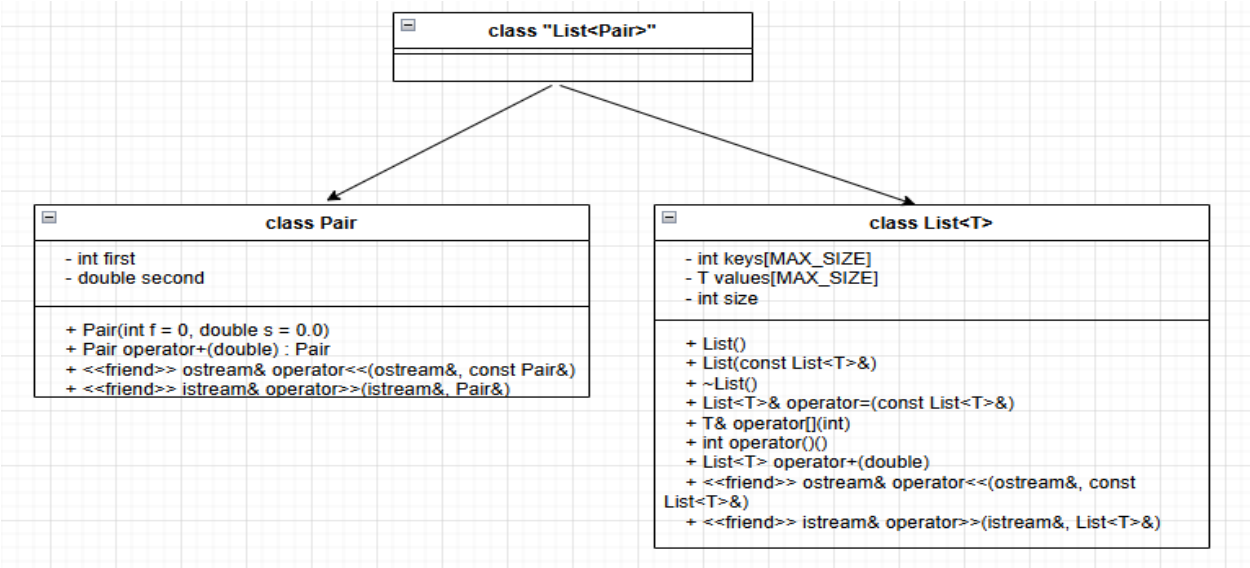
```
51      }
52
53      template <typename T>
54      ostream& operator<<(ostream& os, const List<T>& list) {
55          for (int i = 0; i < list.size; ++i) {
56              os << "[" << list.keys[i] << "] = " << list.values[i] << endl;
57          }
58          return os;
59      }
60
61      template <typename T>
62      istream& operator>>(istream& is, List<T>& list) {
63          cout << "Введите количество элементов: ";
64          is >> list.size;
65          for (int i = 0; i < list.size; ++i) {
66              cout << "Ключ: ";
67              is >> list.keys[i];
68              cout << "Значение: ";
69              is >> list.values[i];
70          }
71          return is;
72      }
```

3 Результаты работы

```
Введите количество элементов: 1
Ключ: 1
Значение: 1
intList:
[1] = 1
Размер: 1
intList + 5:
[1] = 6

Введите количество элементов: 1
Ключ: 1
Значение: 1
1
pairList:
[1] = 1:1
Размер: 1
pairList + 2.5:
[1] = 1:3.5
```

4 UML-диаграмма классов



5 Ответы на контрольные вопросы

1. Повторное использование кода для разных типов данных без дублирования.
2. Синтаксис: template T func(T a); Семантика — работает с любым типом.
3. Синтаксис: template class MyClass {}; Семантика — класс работает с любыми типами.
4. Типовые параметры, передаваемые при вызове шаблона (например, int, double и т.д.).
5. Могут быть типами, значениями или шаблонами. Определяются при инстанцировании.
6. С помощью ключевого слова `template` и параметров: `template<typename T>`.
7. Да, можно перегружать как обычные, так и шаблонные функции.
8. Универсальность, отложенная компиляция, возможность специализации.
9. Нет, функции могут быть непараметризованными.
10. Да, если они объявлены внутри шаблона, они тоже параметризованы.
11. Нет, шаблонные классы не поддерживают виртуальные функции.
12. Через `template<>` и уточнение шаблона: `template<typename T> ReturnType Class<T>::func()`.
13. Процесс создания конкретной версии шаблона с определенным типом.
14. На этапе компиляции, когда шаблон используется с конкретным типом.

6 Ссылка на github

ссылка на github - https://github.com/MAKSPOWERO/mas1