Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра **«Информационные технологии и автоматизированные системы»**

направление подготовки: 09.03.04 - «Программная инженерия»

Лабораторная работа №7

По теме **«Шаблоны классов»**

Вариант №10

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

Постановка задачи:

* 1. Определить шаблон класса-контейнера (см. лабораторную работу №6).
  2. Реализовать конструкторы, деструктор, операции вода-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.

Задание:

Класс-контейнер МНОЖЕСТВО с элементами типа int.

Реализовать операции:

[] доступа по индексу;

() определение размера вектора;

- разность множеств;

Пользовательский класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой.

Анализ задачи:

1. Реализовать пользовательский класс Money в файлах money.h и money.cpp.
2. С помощью template <class T> в контейнер могут заполнятся разные значения.
3. Все остальные функции реализованы как в лаб. работе 6.

Решение

Код

Файл Lab.cpp

#include "vector.h"

#include "money.h"

#include <Windows.h>

#include <iostream>

using namespace std;

int main(){

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Money M;

cout << "Введите денежную сумму" << endl;

cin >> M;

cout << M << endl << endl;

Vector<Money> a(4, M);

cout << "Введите денежные суммы" << endl;

cin >> a;

cout << "Вектор a: " << a << endl;

Vector<Money> b(6, M);

cout << "Вектор b: " << b << endl;

b = a;

cout << "b = a : " << b << endl << endl;

cout << "Доступ по индексу a[1] = " << a[1] << endl;

cout << "Длина вектора a = " << a() << endl << endl;

Vector<Money> c(4, M);

cout << "Введите значения" << endl;

cin >> c;

b = c - a;

cout << "Разность векторов c - a: " << b << endl;

return 0;

}

Файл vector.h

#pragma once

#include <iostream>

using namespace std;

template <class T> //T - параметр шаблона

class Vector{

private:

int size;

T\* data;

public:

Vector(int s, T k);

//конструктор копирования

Vector(const Vector<T>&a);

//деструктор

~Vector();

//оператор присваивания

Vector& operator=(const Vector<T>& a);

//операция доступа по индексу

T& operator[](int index);

//операция разности множеств

template <class U>

friend Vector<U> operator-(const Vector<U>& a, const Vector<U>& b);

//операция, возвращающая длину вектора

int operator()();

//перегруженные операции ввода-вывода

// <> - указывают на то, что функция является шаблоном

friend ostream& operator<< <>(ostream& out, const Vector<T>& a);

friend istream& operator>> <>(istream& in, Vector<T>& a);

};

//конструктор с параметрами

template <class T>

Vector<T>::Vector(int s,T k){

size = s;

data = new T[size];

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

data[i] = k;

}

//конструктор копирования

template <class T>

Vector<T>::Vector(const Vector& a){

size = a.size;

data = new T[size];

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

data[i] = a.data[i];

}

//деструктор

template <class T>

Vector<T>::~Vector() { delete[] data; data = 0; }

//операция присваивания

template <class T>

Vector<T>& Vector<T>::operator=(const Vector<T>& a){

if (this == &a)

return \*this;

size = a.size;

if (data != 0)

delete[]data;

data = new T[size];

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

data[i] = a.data[i];

return \*this;

}

//операция доступа по индексу

template <class T>

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

if (index < size)

return data[index];

else cout << "\nError! Index>size";

}

template <class U>

//операция для разности множеств

Vector<U> operator- (const Vector<U>& a, const Vector<U>& b){

if (a.size != b.size)

cout << "a.size != b.size" << endl;

Vector<U> temp(a.size, U());

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

{

bool flag = true;

for (int j = 0; j < b.size; ++j)

if (a.data[i] == b.data[j]) {

flag = false;

break;

}

if (flag)

temp.data[i] = a.data[i];

}

return temp;

}

//операция для получения длины вектора

template <class T>

int Vector<T>::operator ()() { return size; }

//операции для ввода-вывода

template <class T>

ostream& operator<< (ostream& out, const Vector<T>& a){

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

out << a.data[i] << " ";

return out;

}

template <class T>

istream& operator>> (istream& in, Vector<T>& a){

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

in >> a.data[i];

return in;

}

Файл money.h

#pragma once

#include <iostream>

using namespace std;

class Money{

private:

long rub;

int kop;

public:

Money(void);

Money(long, int);

Money(const Money&);

virtual ~Money(void) {};

Money& operator=(const Money&);

//операция сравнения

bool operator==(const Money& m) const {

return (rub == m.rub) && (kop == m.kop);

}

//перегруженные операции ввода-вывода

friend ostream& operator<< (ostream& out, const Money&);

friend istream& operator>> (istream& in, Money&);

};

Файл money.cpp

#include "money.h"

#include <iostream>

using namespace std;

Money::Money(void){

rub = 0;

kop = 0;

}

Money::Money(long R, int K){

rub = R;

kop = K;

}

Money::Money(const Money& m){

rub = m.rub;

kop = m.kop;

}

Money& Money::operator =(const Money& m){

if (this == &m)

return \*this;

rub = m.rub;

kop = m.kop;

return\*this;

}

ostream& operator<<(ostream& out, const Money& m){

out << m.rub << "," << m.kop;

return out;

}

istream& operator>>(istream& in, Money& m){

cout << "\nрубли: ";

in >> m.rub;

do {

cout << "копейки: ";

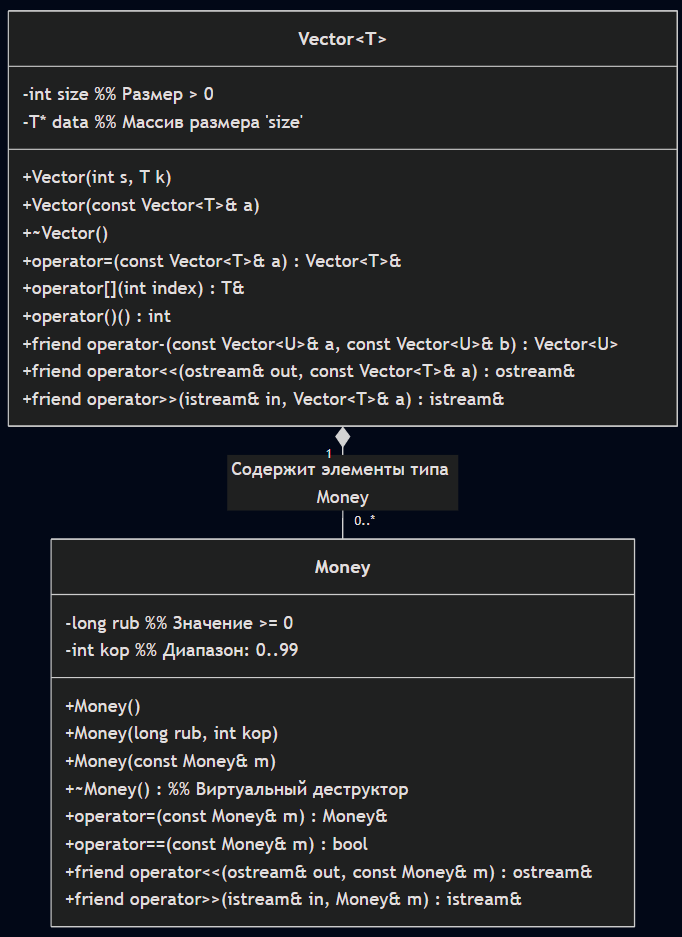
in >> m.kop;

} while (m.kop < 0 || m.kop > 99);

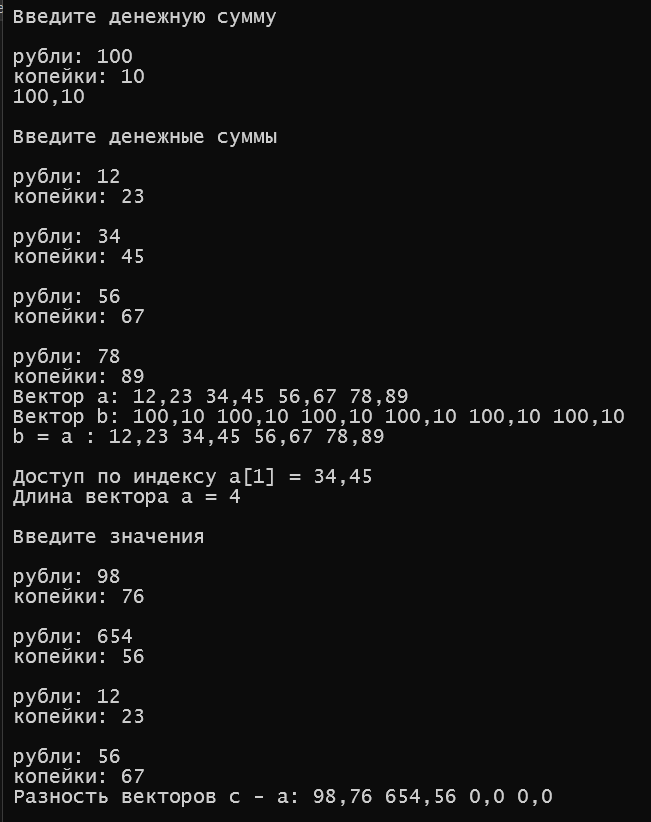
return in;

}

UML-диаграмма



Результат работы программы:



Контрольные вопросы:

1. В чем смысл использования шаблонов?

Шаблоны позволяют создавать обобщённые функции и классы, которые могут работать с разными типами данных без дублирования кода. Они обеспечивают:

1. Повторное использование кода (один шаблон - много типов).
2. Типобезопасность (контроль типов на этапе компиляции).
3. Гибкость (можно специализировать шаблоны для конкретных типов).
4. Каковы синтаксис/семантика шаблонов функций?

template <typename T> //template – ключевое слово для объявления шаблона

T func(T a, T b) { //typename T – параметр шаблона (может быть типом

return a + b; //или значением)

}

Компилятор генерирует конкретные версии функции при вызове с разными типами.

1. Каковы синтаксис/семантика шаблонов классов?

template <class T>

class MyClass {

T value;

public:

MyClass(T v) : value(v) {}

T getValue() { return value; }

};

Шаблон класса позволяет создавать классы для разных типов T.

При инстанцировании компилятор генерирует конкретный класс.

1. Что такое параметры шаблона функции?

Параметры шаблона – это аргументы, переданные в template<...>.

Они могут быть:

1. Типами (typename T, class T).
2. Значениями (int N, bool flag).
3. Шаблонами (шаблонные шаблонные параметры).
4. Перечислите основные свойства параметров шаблона функции.

Могут быть типами или не-типами (значениями).

Могут иметь значения по умолчанию (template <typename T = int>).

Выводятся из аргументов функции (если возможно).

1. Как записывать параметр шаблона?

Для типов: template <typename T> или template <class T>.

Для значений: template <int N>.

Для шаблонных параметров: template <template <typename> class Container>.

1. Можно ли перегружать параметризованные функции?

Да, шаблонные функции можно перегружать другой шаблонной функцией или обычной функцией (которая будет приоритетнее при совпадении).

1. Перечислите основные свойства параметризованных классов.

Могут иметь несколько параметров (template <typename T, int N>).

Поддерживают частичную и полную специализацию.

Могут содержать статические члены (каждый инстанс имеет свои копии).

1. Все ли компонентные функции параметризованного класса являются параметризованными?

Да, все методы класса неявно зависят от параметров шаблона. Однако можно явно специализировать отдельные методы.

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

Нет, не являются, но по умолчанию эти функции будут друзьями для всех классов, которые описаны по этому шаблону.

1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Нет, шаблоны методов класса не могут быть виртуальными.

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

Через полный синтаксис с указанием параметров:

template <class T>

class MyClass {

T value;

public:

T getValue();

};

template <typename T>

T MyClass<T>::getValue() { return value; } // Определение вне класса

1. Что такое инстанцирование шаблона?

Это процесс создания конкретного экземпляра шаблона (функции или класса) для заданных аргументов.

Явное инстанцирование: template class MyClass<int>;

Неявное инстанцирование: MyClass<double> obj; (компилятор генерирует автоматически).

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

На этапе компиляции. Компилятор подставляет переданные типы/значения в шаблон и создаёт конкретный код для них.