Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе 18.7**

Тема: Классы и объекты. Шаблоны классов.

Выполнил работу

студент группы РИС-20-1б

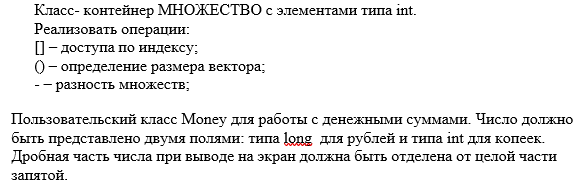
Томилов В.А.

Проверила

Доцент кафедры ИТАС

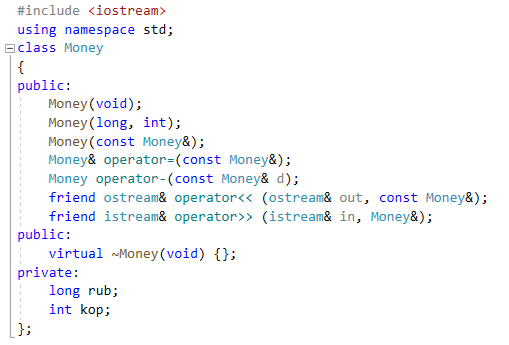
к.т.н ПоляковаО.А.

Пермь, 2021

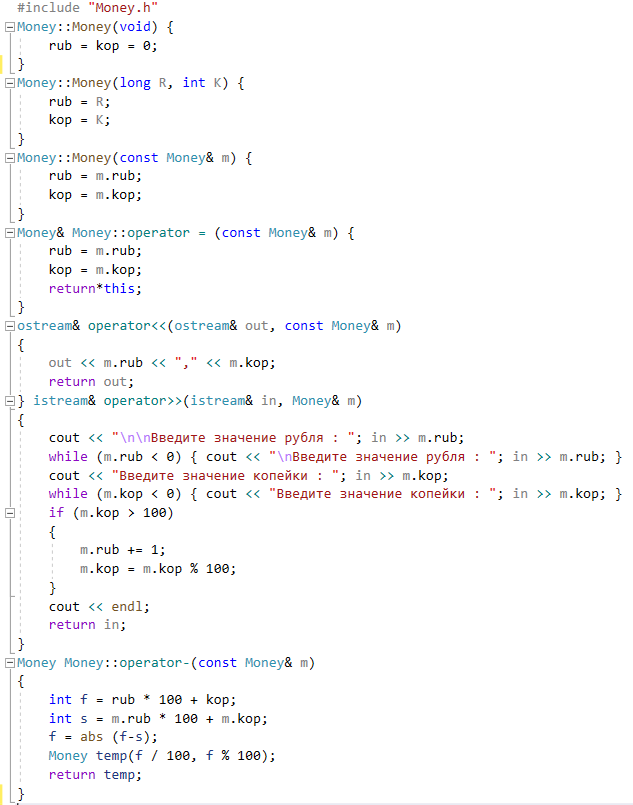
Постановка задачи

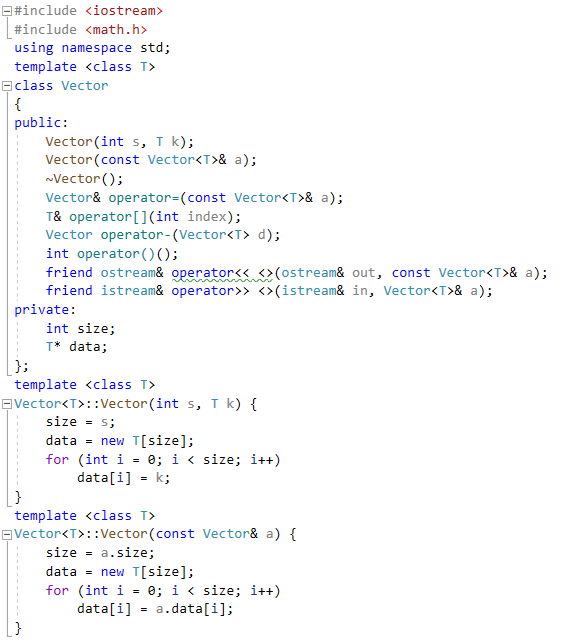
Анализ задачи

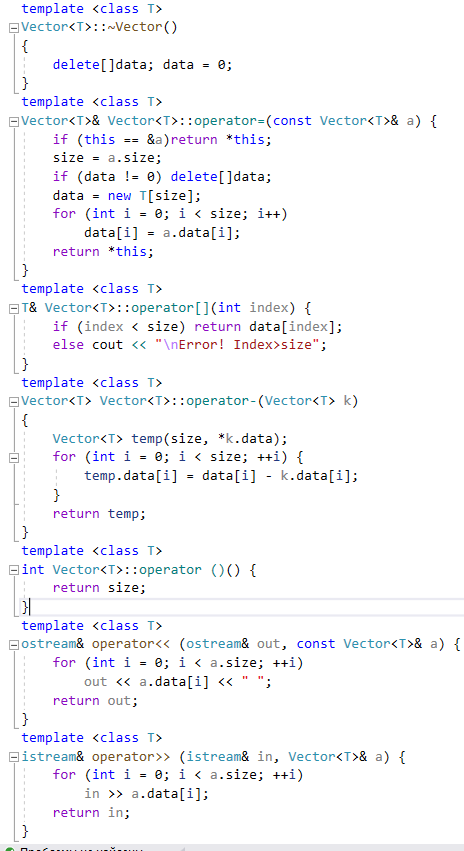
* 1. Для решения задачи необходимо
  2. Добавить в проект файл Money.h, содержащий описание класса.

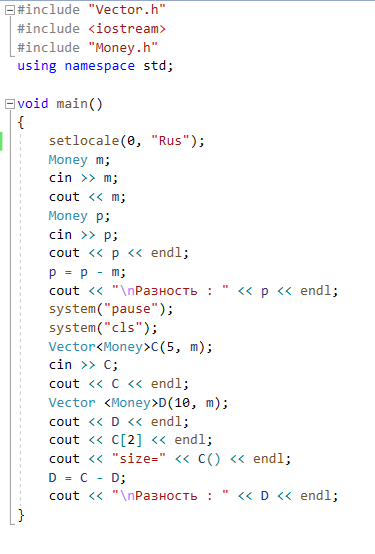


* 1. Добавим в проект файл Money.cpp, содержащий описание методов класса Money.



* 1. Добавить в проект файл Vector.h, содержащий описание класса Vector и описание его методов.

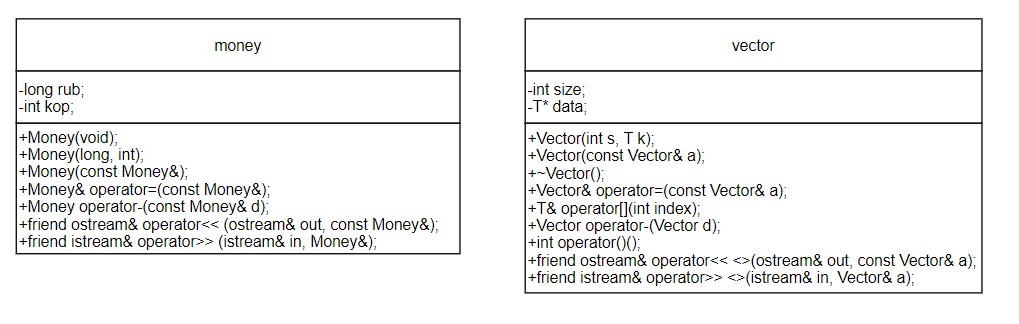


* 1. Добавить в проект файл Lab7\_main.cpp, содержащий основную программу.
  2. В ходе работы были использованы следующие типы данных:

2.1 (int) rub, kop, size – количество рублей и копеек, размер контейнера

2.2 (T\*) data – значение элемента контейнера

UML – диаграмма



Код программы

Main.cpp

#include "Vector.h"

#include <iostream>

#include "Money.h"

using namespace std;

void main()

{

setlocale(0, "Rus");

Money m;

cin >> m;

cout << m;

Money p;

cin >> p;

cout << p << endl;

p = p - m;

cout << "\nРазность : " << p << endl;

system("pause");

system("cls");

Vector<Money>C(5, m);

cin >> C;

cout << C << endl;

Vector <Money>D(10, m);

cout << D << endl;

cout << C[2] << endl;

cout << "size=" << C() << endl;

D = C - D;

cout << "\nРазность : " << D << endl;

}

Money.h

#pragma once

#include <iostream>

using namespace std;

class Money

{

public:

Money(void);

Money(long, int);

Money(const Money&);

Money& operator=(const Money&);

Money operator-(const Money& d);

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

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

public:

virtual ~Money(void) {};

private:

long rub;

int kop;

};

Money.cpp

#include "Money.h"

Money::Money(void) {

rub = 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) {

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 << "\n\nВведите значение рубля : "; in >> m.rub;

while (m.rub < 0) { cout << "\nВведите значение рубля : "; in >> m.rub; }

cout << "Введите значение копейки : "; in >> m.kop;

while (m.kop < 0) { cout << "Введите значение копейки : "; in >> m.kop; }

if (m.kop > 100)

{

m.rub += 1;

m.kop = m.kop % 100;

}

cout << endl;

return in;

}

Money Money::operator-(const Money& m)

{

int f = rub \* 100 + kop;

int s = m.rub \* 100 + m.kop;

f = abs (f-s);

Money temp(f / 100, f % 100);

return temp;

}

Vector.h

#pragma once

#include <iostream>

#include <math.h>

using namespace std;

template <class T>

class Vector

{

public:

Vector(int s, T k);

Vector(const Vector<T>& a);

~Vector();

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

T& operator[](int index);

Vector operator-(Vector<T> d);

int operator()();

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

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

private:

int size;

T\* data;

};

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;

}

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];

}

template <class T>

Vector<T>::~Vector()

{

delete[]data; data = 0;

}

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;

}

template <class T>

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

if (index < size) return data[index];

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

}

template <class T>

Vector<T> Vector<T>::operator-(Vector<T> k)

{

Vector<T> temp(size, \*k.data);

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

temp.data[i] = data[i] - k.data[i];

}

return temp;

}

template <class T>

int Vector<T>::operator ()() {

return size;

}

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;

}

Ответы на вопросы

1. С помощью шаблона функций можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра.
2. template <параметры\_шаблона>

заголовок\_функции

{тело функции}

1. template <параметры шаблона>

class имя\_класса

{…};

1. Типы данных
2. Свойства:

* Имена параметров шаблона должны быть уникальными во всем определении шаблона.
* Список параметров шаблона не может быть пустым.
* В списке параметров шаблона может быть несколько параметров, и каж-дому из них должно предшествовать ключевое слово class.
* Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.
* Определенная с помощью шаблона функция может иметь любое количество непараметризированных формальных параметров. Может быть непараметризированно и возвращаемое функцией значение.
* В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.
* При конкретизации параметризованной функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были оди-наковы.

1. –
2. Да
3. –
4. Да
5. Нет
6. Нет
7. –
8. Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона называется инстанцированием шаблона.
9. Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.