Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

(ПНИПУ)

**Факультет:** электротехнический

**Направление:** Промышленная мехатроника и робототехника

ОТЧЁТ

По лабораторной работе №7

«Классы»

Семестр 1

Выполнил:

студент группы ПРТ-21-1б

Поздняков Е.С.

Пермь 2022

**Задание №1**

**Постановка задачи**

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**

1. Объявляем класс а с параметром размера множества. Пользователь вводит значения элементов множества.
2. Объявляем класс a1 из 5 элементов равных 7. Объявляем класс c и инициализируем его значением разности двух классов.
3. При помощи класса Iterator осуществляется вывод значений итератора i. При помощи перегруженных операторов инкремента и декремента осуществляется изменение значения i.

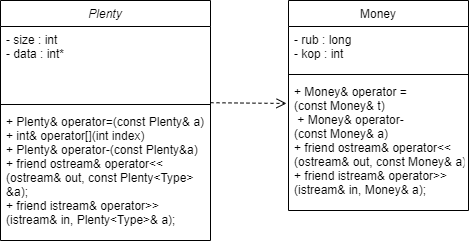
**Используемые переменные**

Plenty a – первое множество;

Plenty a1 – второе множество;

int с – значение элемента множества;

**UML-диаграмма**



**Код C++**

#include <iostream>

using namespace std;

template <typename T>

class Plenty

{

private:

int size; // размер множества

T\* data; // указатель на массив значений множества

public:

Plenty(int s, T k = 0)

{

size = s;

data = new T[size];

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

data[i] = k;

}

Plenty(const Plenty<T>& a)

{

size = a.size;

data = new T[size];

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

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

}

~Plenty()

{

delete [] data;

}

Plenty& operator=(const Plenty<T>& a)

{

data = new T[size];

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

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

return \*this;

}

int& operator[](int index)

{

return data[index];

}

Plenty& operator-(const Plenty<T>&a)

{

if (size != a.size)

{

cout << "Error, sizes don't match" << endl;

return \*this;

}

else

{

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

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

return \*this;

}

}

template <typename Type>

friend ostream& operator<<(ostream& out, const Plenty<Type>&a);

template <typename Type>

friend istream& operator>>(istream& in, Plenty<Type>& a);

};

template <typename Type>

ostream& operator<<(ostream& out, const Plenty<Type>&a)

{

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

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

return out;

}

template <typename Type>

istream& operator>>(istream& in, Plenty<Type>& a)

{

cout << "Input size: ";

cin >> a.size;

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

{

cout << "Input elem: ";

in >> a.data[i];

}

return in;

}

class Money

{

private:

long rub;

int kop;

public:

Money() : rub(0), kop(0) {};

Money(long r, int k) : rub(r), kop(k) {};

Money (const Money& t) : rub(t.rub), kop(t.kop) {};

~Money(){};

Money& operator = (const Money& t)

{

rub = t.rub;

kop = t.kop;

return \*this;

}

Money& operator-(const Money& a)

{

long t3 = abs(a.rub \* 100 + a.kop - (rub \* 100 + kop));

rub = t3 / 100;

kop = t3 % 100;

return \*this;

}

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

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

};

ostream& operator<<(ostream& out, const Money& a)

{

out << a.rub << "," << a.kop << endl;

return out;

}

istream& operator>>(istream& in, Money& a)

{

cout << "rub = "; in >> a.rub;

cout << "kop = "; in >> a.kop;

return in;

}

int main()

{

Plenty<double> a(10);

cout << "Plenty a" << endl;

cin >> a;

Plenty<double> a1(5, 7);

cout << "Plenty a1 : " << a1 << endl;

Plenty<double> c = a1-a;

cout << "a1 - a = " << c << endl;

Money t1; cout << "t1..." << endl; cin >> t1;

Money t2; cout << "t2..." << endl; cin >> t2;

Money t3 = t2 - t1;

cout << "Difference between t2 and t1 = " << t2;

Money t;

cin >> t;

cout << t;

Plenty<Money>A(5, t);

cin >> A;

cout << A << endl;

Plenty<Money>B(5, t);

cout << B << endl;

Plenty<Money>C(5, t);

C = B - A;

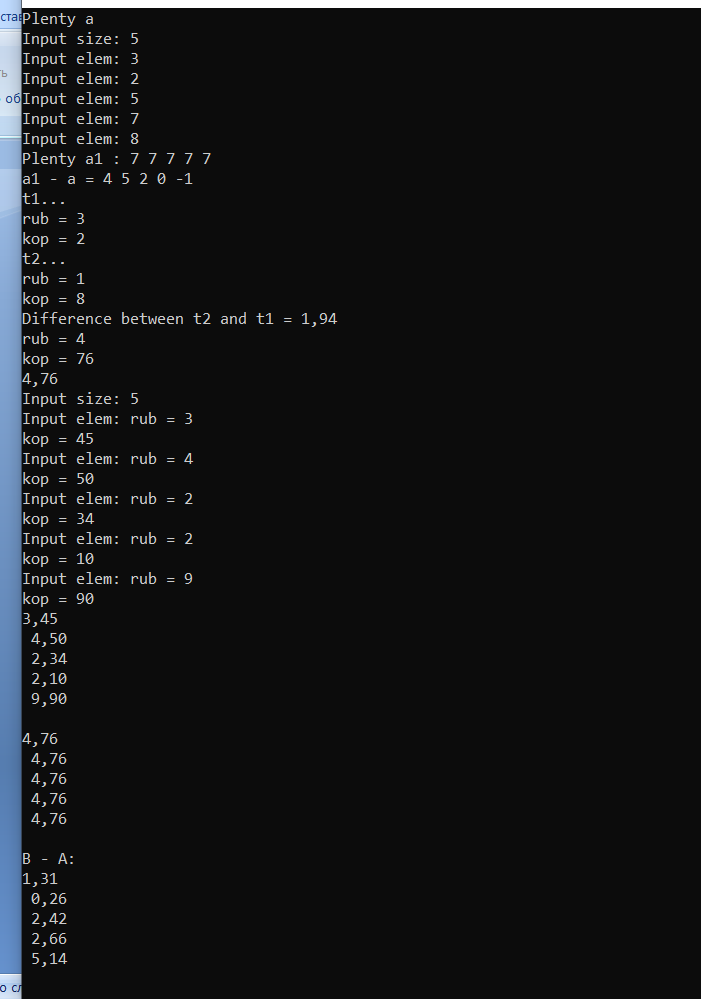
cout << "B - A:" << endl;

cout << C;

return 0;

}

**Скриншот результата**

****

**Анализ результатов**

Пользователь вводит значения элементов первого множества. Из элементов второго множества вычитаются элементы первого множества и выводятся на экран.

**Контрольные вопросы**

1. В чем смысл использования шаблонов?

С помощью шаблона функций можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра.

2. Каковы синтаксис/семантика шаблонов функций?

template <параметры\_шаблона>

заголовок\_функции

{тело функции}

3. Каковы синтаксис/семантика шаблонов классов?

Template <параметры\_шаблона>

Class имя\_класса

{};

4. Что такое параметры шаблона функции?

Параметр шаблона – его формальный аргумент.

5. Перечислите основные свойства параметров шаблона функции.

* 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 каждому 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 непараметризованных формальных параметров. Может быть непараметризованно 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 одинаковы.

6. Как записывать параметр шаблона?

Template <typename T>

7. Можно ли перегружать параметризированные функции?

Да, меняя тип данных параметра, либо меняя параметры местами, в том случае, если они разного типа.

8. Перечислите основные свойства параметризированных классов.

* Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью *template.*
* Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.
* Если *friend*-функция содержит в своем описании параметр типа параметризованного класса, то для каждого созданного по данному шаблону класса имеется собственная *friend*-функция.
* В рамках параметризованного класса нельзя определить *friend*-шаблоны (дружественные параметризованные классы).
* С одной стороны, шаблоны могут быть производными (наследоваться) как от шаблонов, так и от обычных классов, с другой стороны, они могут использоваться в качестве базовых для других шаблонов или классов.
* Шаблоны функций, которые являются членами классов, нельзя описывать как*virtual.*
* Локальные классы не могут содержать шаблоны в качестве своих элементов.

9. Все ли компонентные функции параметризированного класса являются параметризированными?

Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.

10. Являются ли дружественные функции, описанные в параметризированном классе, параметризированными?

Yes

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

12. Как определяются компонентные функции параметризированных классов вне определения шаблона?

При помощи расширения видимости

13. Что такое инстанцирование шаблона?

Инстанцирование шаблона – процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона

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

На этапе компиляции, когда компилятор узнает, с каким типом мы работаем.