Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Дисциплина: Основы алгоритмизации и программирования

Тема: Шаблоны классов.

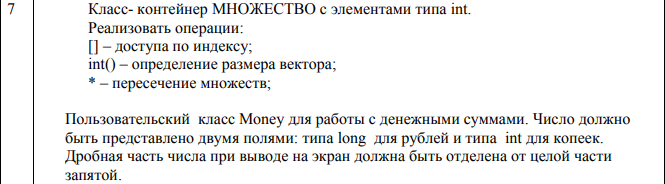
|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Бреднев М.П. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

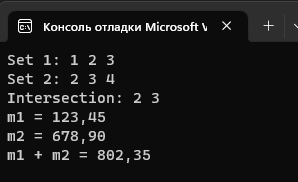
**Анализ предметной области**

**Постановка задачи**

1. Определить шаблон класса контейнера.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Реализовать пользовательский класс.
5. Перегрузить для пользовательского класса операции ввода вывода.
6. Перегрузить операции необходимые для выполнения операций контейнерного класса.
7. Инстанцировать шаблон для пользовательского класса.
8. Реализовать итератор. Реализовать с его помощью операции последовательного доступа.
9. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.



**Тестирование программы**

****

*Рис. 1 – Тестирование программы.*

# Контрольные вопросы

1. В чем смысл использования шаблонов?

С помощью шаблона функций можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра. Шаблоны классов предоставляют аналогичную возможность, позволяя создавать параметризированные классы. Параметризированный класс создает семейство родственных классов, которые можно применять к любому типу данных, передаваемому в качестве параметра. Если использовать в качестве параметризированного класса контейнер, то такой контейнер можно будет применять к любым типам данных, не переписывая код.

Шаблон служит для автоматического формирования конкретных описаний

функций по тем вызовам, которые компилятор обнаруживает в программе.

2. Каковы синтаксис/семантика шаблонов функций?

template <параметры\_шаблона>

заголовок\_функции (параметры\_функции)

{тело функции}

*Пример*

template <typename T>

T abs(T x) {

if (x>0) return x;

else return -x;

}

3. Каковы синтаксис/семантика шаблонов классов?

template <параметры шаблона>

class имя\_класса

{…};

Пример

template <class T>

class Point {

T x,y;//координаты точки

public:

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

void Show ();

};

template<class T>

void Point::Show() { cout<<”(“<<x<<” , ”<<y<<”)”;}

Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.

Point <int> a(13,15);

Point <float>\*pa=new Point<float>(10.1,0.55);

4. Что такое параметры шаблона функции?

Можно считать, что параметры шаблона являются его формальными параметрами, а типы тех параметров, которые используются в конкретных обращениях к функции, служат фактическими параметрами шаблона. Именно по ним выполняется параметрическая настройка и с учетом этих типов генерируется конкретный текст определения функции.

template <typename T>

T abs(T x) {

if (x>0) return x;

else return -x;

}

Если в программе вызов функции осуществляется как abs(-1.5), то компилятор формирует определение функции double abs(double x)

5. Перечислите основные свойства параметров шаблона функции.

* Имена параметров шаблона должны быть уникальными во всем определении шаблона.
* Список параметров шаблона не может быть пустым, для того, чтобы компилятор мог инстанцировать шаблон.
* В списке параметров шаблона может быть несколько параметров, и каждому из них должно предшествовать ключевое слово class.  
  template<class type1, class type2>  
  Соответственно, неверен заголовок:  
  template<class type1, type2, type3>
* Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, то есть ошибочен такой заголовок:  
  template<class t, class t, class t>
* Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.
* Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции. Таким образом, будет ошибочным такой шаблон:

template<class A, class B, class C>  
B func(A n, C m) { B value; };

* Определенная с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть непараметризованно и возвращаемое функцией значение. Например, в следующей программе шаблон определяет семейство функций, каждая из которых подсчитывает количество нулевых элементов одномерного массива параметризованного типа:

template<class D> long count0(int, D \*); //Прототип шаблона

int main(void) {

int A[] = { 1, 0, 6, 0, 4, 10 };

int n = sizeof(A) / sizeof A[0];

cout << "\ncount0(n,A) = " << count0(n, A);

float X[] = { 10.0, 0.0, 3.3, 0.0, 2.1 };

n = sizeof(X) / sizeof X[];

cout << "\ncount0(n,X) = " << count0(n, X);

return 0; }

* В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.
* При конкретизации шаблонного определения функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковыми. Для определенного ниже шаблона функций с данным прототипом недопустимо использовать такое обращение к функции:

template<class E> void swap(E, E);  
int n = 4;   
double d = 4.3;   
swap(n, d); // Ошибка в типах параметров  
swap(double(n), d); // Правильные типы параметров

Для правильного обращения к такой функции требуется явное приведение типа одного из параметров.

* При использовании шаблонов функций возможна перегрузка как шаблонов, так и функций. Могут быть шаблоны с одинаковыми именами, но разными параметрами. Или с помощью шаблона может создаваться функция с таким же именем, что и явно определенная функция. В обоих случаях "распознавание" конкретного вызова выполняется по сигнатуре, т.е. по типам, порядку и количеству фактических параметров.

6. Как записывать параметр шаблона?

* Каждому из разных типов параметров должно предшествовать ключевое слово class. *Пример:* template<class type1, class type2>
* Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции.
* Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, то есть ошибочен такой заголовок:  
  template<class t, class t, class t>

7. Можно ли перегружать параметризованные функции?

Да. Шаблон функции может перегружать функции, отличные от шаблона, с тем же именем. В этом сценарии компилятор сначала пытается разрешить вызов функции с помощью вычета аргументов шаблона для создания экземпляра шаблона функции с уникальной специализацией.

*Пример*

template <typename T>

void print(T value) {cout << “Шаблон” << value;}

template <>

void print(int value) {cout << “Специализация для int” << value;}

void print(int value) {cout << “Перегрузка для int” << value;}

8. Перечислите основные свойства параметризованных классов.

* В определении класса, входящего шаблон, имя класса является не именем отдельного класса, а параметризованным именем семейства 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.
* 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.
* Локальные классы не могут содержать шаблоны в качестве своих элементов.

9. Все ли компонентные функции параметризованного класса являются

параметризованными?

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

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

В рамках параметризованного класса нельзя определить friend-шаблоны (дружественные параметризованные классы).

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Шаблоны функций, которые являются членами классов, нельзя описывать как virtual.

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

Реализация компонентной функции шаблона класса, которая находится вне определения шаблона класса, должна включать дополнительно следующие два элемента:

* Определение должно начинаться с ключевого слова template, за которым следует такой же список\_параметров\_типов в угловых скобках, какой указан в определении шаблона класса.
* За именем\_класса, предшествующим операции области видимости (::), должен следовать список\_имен\_параметров шаблона.

template<список\_типов>тип\_возвр\_значения имя\_класса<список\_имен\_ параметров> : : имя\_функции(список\_параметров){ . . . }

13. Что такое инстанцирование шаблона?

Инстанцирование шаблона – это генерация кода функции или класса по шаблону для конкретных параметров.

Неявное инстанцирование:

template <typename T>

T sqrt(T x) {return x\*x;}

Если в программе вызов функции осуществляется как sqrt(-1), то компилятор формирует определение функции int sqrt(int x)

Явное инстанцирование:

template <typename T>

T sqrt(T x) {return x\*x;}

template double sqrt(double);

template int sqrt(int);

Через явное инстанцирование будут доступны только те типы, которые были явно инстанцированы ключевым словом template.

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

Компилятор производит генерацию кода по шаблону, когда происходит явное или неявное инстанцирование. Это может существенно увеличить время компиляции и компоновки, особенно в тех случаях, когда шаблон инстанцируется с одинаковыми параметрами во многих модулях. С помощью extern теперь можно указать компилятору не инстанцировать шаблон в данной единице трансляции (файле)

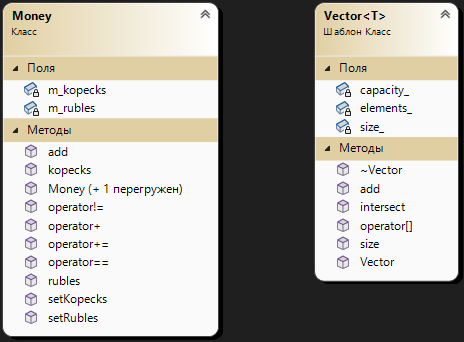
extern template vector<int>;

Естественно, в одном из модулей должно быть сделано явное или неявное инстанцирование, чтобы при компоновке программы все ссылки были разрешены.

template vector<int>;

**Приложения**

Приложение UML-диаграмма



Приложение Б – код программы

Main.cpp:

#include <iostream>

#include "Vector.h"

#include "Money.h"

using namespace std;

int main() {

// Создание и заполнение множеств

Vector<int> s1;

s1.add(1);

s1.add(2);

s1.add(3);

Vector<int> s2;

s2.add(2);

s2.add(3);

s2.add(4);

// Вывод содержимого множеств

cout << "Set 1: ";

for (int i = 0; i < s1.size(); i++) {

cout << s1[i] << " ";

}

cout << endl;

cout << "Set 2: ";

for (int i = 0; i < s2.size(); i++) {

cout << s2[i] << " ";

}

cout << endl;

// Пересечение множеств

Vector<int> intersection = s1.intersect(s2);

cout << "Intersection: ";

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

cout << intersection[i] << " ";

}

cout << endl;

// Работа с денежными суммами

Money m1(123, 45);

Money m2(678, 90);

cout << "m1 = " << m1 << endl;

cout << "m2 = " << m2 << endl;

Money sum = m1 + m2;

cout << "m1 + m2 = " << sum << endl;

return 0;

}

Money.h:

#ifndef MONEY\_H

#define MONEY\_H

#include <iostream>

class Money {

private:

long m\_rubles;

int m\_kopecks;

public:

Money() : m\_rubles(0), m\_kopecks(0) {}

Money(long rubles, int kopecks) : m\_rubles(rubles), m\_kopecks(kopecks) {}

long rubles() const {

return m\_rubles;

}

int kopecks() const {

return m\_kopecks;

}

void setRubles(long rubles) {

m\_rubles = rubles;

}

void setKopecks(int kopecks) {

m\_kopecks = kopecks;

}

void add(const Money& other) {

m\_rubles += other.m\_rubles;

m\_kopecks += other.m\_kopecks;

if (m\_kopecks >= 100) {

m\_rubles += m\_kopecks / 100;

m\_kopecks %= 100;

}

}

Money operator+(const Money& other) const {

Money result(\*this);

result.add(other);

return result;

}

Money& operator+=(const Money& other) {

add(other);

return \*this;

}

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

return m\_rubles == other.m\_rubles && m\_kopecks == other.m\_kopecks;

}

bool operator!=(const Money& other) const {

return !(\*this == other);

}

friend std::ostream& operator<<(std::ostream& os, const Money& money) {

os << money.m\_rubles << ',' << (money.m\_kopecks < 10 ? "0" : "") << money.m\_kopecks;

return os;

}

};

#endif // MONEY\_H

Vector.h:

#pragma once

#ifndef VECTOR\_H

#define VECTOR\_H

#include <iostream>

template<typename T>

class Vector

{

public:

Vector() : size\_(0), capacity\_(10), elements\_(new T[capacity\_]) {}

~Vector() { delete[] elements\_; }

void add(const T& element);

int size() const { return size\_; }

T operator[](int index) const { return elements\_[index]; }

Vector<T> intersect(const Vector<T>& other) const;

private:

int size\_;

int capacity\_;

T\* elements\_;

};

template<typename T>

void Vector<T>::add(const T& element) {

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

if (elements\_[i] == element) {

return;

}

}

if (size\_ >= capacity\_) {

capacity\_ \*= 2;

T\* newElements = new T[capacity\_];

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

newElements[i] = elements\_[i];

}

delete[] elements\_;

elements\_ = newElements;

}

elements\_[size\_] = element;

size\_++;

}

template<typename T>

Vector<T> Vector<T>::intersect(const Vector<T>& other) const {

Vector<T> result;

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

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

if (elements\_[i] == other[j]) {

result.add(elements\_[i]);

break;

}

}

}

return result;

}

#endif // VECTOR\_H