Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

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

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: Шаблоны классов

Выполнил работу

Студент группы РИС-22-1Б

Лихачев Д.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Постановка задачи**





**Диаграмма классов**



**Диаграмма классов**

**Array.h**

#pragma once

#include <iostream>

#include <fstream>

#include <ctime>

using namespace std;

template<class T>

class Array {

private:

int size;

T\* data;

public:

Array();

Array(int size);

Array(const Array<T>& other);

~Array();

Array<T>& operator=(const Array<T>& other);

int& operator[](int index);

bool operator!=(const Array<T>& a) const;

bool operator<(int num) const;

friend ostream& operator<<(ostream& fout, const Array<T>& other);

friend istream& operator>>(istream& fin, Array<T>& other);

virtual void show()

{

cout << "Вывод множества: ";

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

{

cout << " " << data[i];

}

cout << endl;

}

};

template<class T>

Array<T>::Array() : size(0), data(nullptr) {}

template<class T>

Array<T>::Array(int size) : size(size)

{

if (size <= 0)

{

throw runtime\_error("Error: Invalid size");

}

data = new T[size];

srand(time(nullptr));

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

{

data[i] = rand() % 100;

}

}

template<class T>

Array<T>::Array(const Array<T>& other) : size(other.size)

{

data = new T[size];

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

{

data[i] = other.data[i];

}

}

template<class T>

Array<T>::~Array()

{

delete[] data;

}

template<class T>

Array<T>& Array<T>::operator=(const Array<T>& other)

{

if (this != &other)

{

delete[] data;

size = other.size;

data = new T[size];

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

{

data[i] = other.data[i];

}

}

return \*this;

}

template<class T>

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

{

if (index < 0 || index >= size)

{

throw out\_of\_range("Error: Index out of range");

}

return data[index];

}

template<class T>

bool Array<T>::operator!=(const Array<T>& a) const

{

if (size != a.size)

{

return true;

}

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

{

if (data[i] != a.data[i])

{

return true;

}

}

return false;

}

template<class T>

bool Array<T>::operator<(int num) const

{

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

{

if (data[i] >= num)

{

return false;

}

}

return true;

}

template<class T>

ostream& operator<<(ostream& fout, const Array<T>& other)

{

fout << "Множество: ";

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

{

fout << other.data[i] << " ";

}

fout << endl;

return fout;

}

template<class T>

istream& operator>>(istream& fin, Array<T>& other)

{

cout << "Введите размер множества: ";

int size;

fin >> size;

Array<T> tmp(size);

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

{

cout << "Введите " << i + 1 << "-ый элемент: ";

fin >> tmp.data[i];

}

other = tmp;

return fin;

}

Money.h

#pragma once

#include <iostream>

using namespace std;

class Money

{

public:

Money(void);

Money(long, int);

Money(const Money&);

Money& operator=(const Money&);

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

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

virtual ~Money(void) { rubles = 0; kop = 0; };

void operator!=(const Money& t);

private:

long rubles;

int kop;

};

Money.cpp

#include "Money.h"

Money::Money(void)

{

rubles = 0;

kop = 0;

}

Money::Money(long r, int k)

{

rubles = r;

kop = k;

}

Money::Money(const Money& t)

{

rubles = t.rubles;

kop = t.kop;

}

Money& Money::operator=(const Money& t)

{

rubles = t.rubles;

kop = t.kop;

return \*this;

}

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

{

out << t.rubles << " , " << t.kop;

return out;

}

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

{

cout << "Enter rubles: "; in >> t.rubles;

cout << "Enter kop: "; in >> t.kop;

return in;

}

void Money::operator!=(const Money& t)

{

if (rubles == t.rubles && kop == t.kop) cout << "ravny." << endl;

else cout << "ne ravny." << endl;

}

Source.cpp

#include <iostream>

#include "Array.h"

#include "Money.h"

using namespace std;

int main()

{

Array<int> A(5);

cout << "A:" << A << endl;

setlocale(LC\_ALL, "rus");

srand(time(0));

int k, size = 5, n = 0;

cout << "Введите k - индекс множества: "; cin >> k;

Array<int> a(size);

// вывод множества а

a.show();

cout << endl << "Число с выбранным индексом: " << a[k] << endl;

cout << "Введите число, которое хотите проверить на принадлежность множетству: "; cin >> n;

a < n;

cout << "Проверка операции равенства множеств." << endl;

Array<int> b(size);

b.show();

Array<int> c(size);

c.show();

b != c;

cout << "Тестирование класса Money:" << endl;

Money t1;

cin >> t1;

cout << "Первый экземпляр класса Money:" << endl;

cout << t1; cout << endl;

Money t2;

cin >> t2;

cout << "Второй экзепляр класса Money: " << endl;

cout << t2; cout << endl;

cout << "Проверка их на неравенство: " << endl;

t1 != t2;

return 0;

}

**Результаты работы программы**



**Ответы на контрольные вопросы**

1. В чем смысл использования шаблонов?

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные.

2. Каковы синтаксис/семантика шаблонов функций?

template<параметры\_шаблона>

заголовок функции

{тело функции}

пример:

template<class type>

type abs(type 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;

public:

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

void Show();

{;

4. Что такое параметры шаблона функции?

Параметр шаблона - то, что указывается в <>.

Параметрами шаблонов могут быть:

* параметры-типы;
* параметры обычных типов;
* параметры-шаблоны.
* 

5. Перечислите основные свойства параметров шаблона функции.

- шаблоны не могут быть виртуальными

- шаблоны могут содержать статические элементы, дружественные функции и классы

- шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

6. Как записывать параметр шаблона?

в угловых скобочках < >

7. Можно ли перегружать параметризованные функции?

Да, меняя тип данных параметра, либо меняя параметры местами, в том случае, если они разного типа, а также можно выполнять перегрузку функции добавляя или исключая параметры.

8. Перечислите основные свойства параметризованных классов.

- Параметры шаблона могут быть любыми типами данных, включая примитивные типы, пользовательские типы и другие шаблоны классов.

- Параметры шаблона могут быть как значениями, так и типами. Например, можно создать шаблон класса с параметром, который задает размер массива.

-Параметры шаблона могут использоваться внутри класса для объявления переменных, функций и типов.

-Каждый конкретный экземпляр параметризованного класса является отдельным типом, который определяется параметрами шаблона.

-Шаблонные классы могут быть унаследованы от других шаблонных классов и могут иметь шаблонные функции-члены.

9. Bce ли компонентные функции параметритизированного класса являются параметризованными?

Да, все компонентные функции (методы) параметризированного класса являются параметризованными, то есть они используют те же параметры шаблона, что и сам класс.

10. Являются ли дружественные функции, описанные в  
классе, параметризованными?  
Да, дружественные функции, описанные внутри класса, являются параметризованными, если класс является параметризованным. Дружественные функции имеют доступ к закрытым членам класса, но они также должны быть параметризованы, чтобы иметь доступ к параметрам шаблона класса.

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Да, шаблоны классов могут содержать виртуальные компонентные функции. Виртуальные функции позволяют реализовать полиморфизм и позволяют вызывать функцию на объекте, указатель на который имеет тип базового класса, но указывает на объект, созданный из класса-наследника.

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

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса с помощью использования template<class T> перед каждым определением функции, где T - это параметр шаблона.



13. Что такое инстанцирование шаблона?

Инстанцирование шаблона - это процесс создания конкретной реализации шаблона класса или функции на основе заданных параметров шаблона. При инстанцировании компилятор создает отдельный экземпляр шаблона для конкретных аргументов шаблона. То есть компилятор подставляет значения параметров шаблона в шаблонный код и создает реализацию класса или функции, которая может быть использована в программе.

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

На этапе компиляции.