Пермский национальный исследовательский политехнический университет

**Лабораторная работа №7**

**Дисциплина: Информатика**

**Вариант № 9**

**"Шаблоны классов"**

Выполнил: Зайченко Никита Геннадьевич

Проверила: Доцент кафедры ИТАС Полякова О. А.

Пермь 2022

**Содержание отчета**

1) Постановка задачи (общая и конкретного варианта).

2) Описание параметризированного класса-контейнера.

3) Определение компонентных функций.

4) Описание пользовательского класса и его компонентных функций

5) Функция main().

6) Объяснение результатов работы программы.

7) Ответы на контрольные вопросы.

**Постановка задачи (общая и конкретного варианта)**

1) Создание консольного приложения, состоящего из нескольких файлов в системе

программирования Visual Studio.

2) Реализация шаблона класса-контейнера.

3) Класс- контейнер МНОЖЕСТВО с элементами типа int.

Реализовать операции:

[] – доступа по индексу;

!= - проверка на неравенство;

< число – принадлежность числа множеству;

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.

**Описание параметризированного класса-контейнера.**

template <class T> //T - параметр шаблона

class Set

{

public:

Set(int s = 0); //конструктор с параметрами

Set(const Set<T>& a); //конструктор копирования

~Set(); //деструктор

Set& operator=(const Set<T>& a); //оператор присваивания

T& operator[](int index); //операция доступа по индексу

int operator()(); //операция, возвращающая длину множества

bool operator<(const T& a) const; //принадлежность числа множеству;

//перегруженные операции ввода-вывода <> - указывают на то, что функция является шаблоном

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

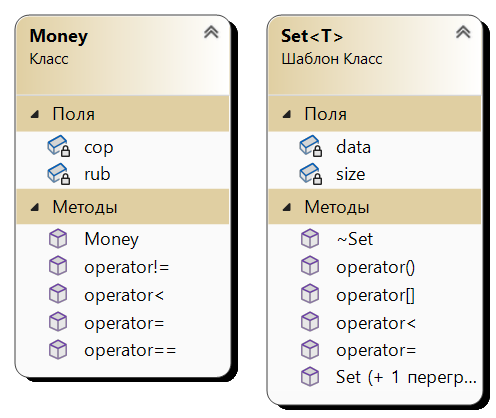
friend istream& operator>> <>(istream& in, Set<T>& a);

private:

int size; //размер множества

T\* data; //укзатель на динамический массив значений множества

};



**Определение компонентных функций.**

//конструктор с параметрами

template<class T>

Set<T>::Set(int n) {

size = n;

data = new T[size];

}

//конструктор копирования

template <class T>

Set<T>::Set(const Set& a)

{

size = a.size;

data = new T[size];

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

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

}

//деструктор

template <class T>

Set<T>::~Set()

{

delete[]data;

data = 0;

}

//операция присваивания

template<class T>

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

if (this != &a) { // проверка на самоприсваивание

delete[] data;

size = a.size;

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

{

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

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

}

//операция для получения длины множества

template <class T>

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

//принадлежность числа множеству;

template<class T>

bool Set<T>::operator<(const T& a) const {

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

if (data[i] == a) {

return true;

}

}

return false;

}

//операции для ввода-вывода

template <class T>

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

{

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

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

return out;

}

template <class T>

istream& operator>> (istream& in, Set<T>& a)

{

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

in >> a.data[i];

return in;

}

**Описание пользовательского класса и его компонентных функций**

class Money {

long rub;

int cop;

public:

Money(long r = 0, int c = 0) { rub = r; cop = c; }

bool operator==(const Money& m);

Money& operator=(const Money& m);

bool operator<(const Money& a) const;

bool operator!=(const Money& a) const;

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

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

};

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

{

if (rub == m.rub && cop == m.cop) { return true; }

else { return false; }

}

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

{

if (&m == this) return \*this;

rub = m.rub;

cop = m.cop;

return \*this;

}

bool Money::operator<(const Money& a) const {

if (rub < a.rub)

return true;

else if (rub == a.rub && cop < a.cop)

return true;

else

return false;

}

bool Money::operator!=(const Money& a) const {

if (rub != a.rub || cop != a.cop)

return true;

else

return false;

}

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

out << a.rub << "," << a.cop;

return out;

}

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

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

cout << "Введите копейки = "; in >> a.cop;

cout << endl;

return in;

}

**Функция main().**

void main()

{

setlocale(LC\_ALL, "Russian");

Set<Money> A(5); // создаем множество размером 5

cout << "Введите " << A() << " элементов класса Money:\n";

cin >> A;

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

Set<Money> B(3); // создаем множество размером 3

cout << "Введите " << B() << " элементов класса Money:\n";

cin >> B;

cout << "B: " << endl << B << endl;

Money x(100, 50);

if (A<x) // < число – принадлежность числа множеству

cout << x << " Принадлежит A" << endl;

else

cout << x << " Не принадлежит A" << endl;

if (x != B[0]) //[] – доступа по индексу и != - проверка на неравенство

cout << x << " Не равен первому элементу B" << endl;

else

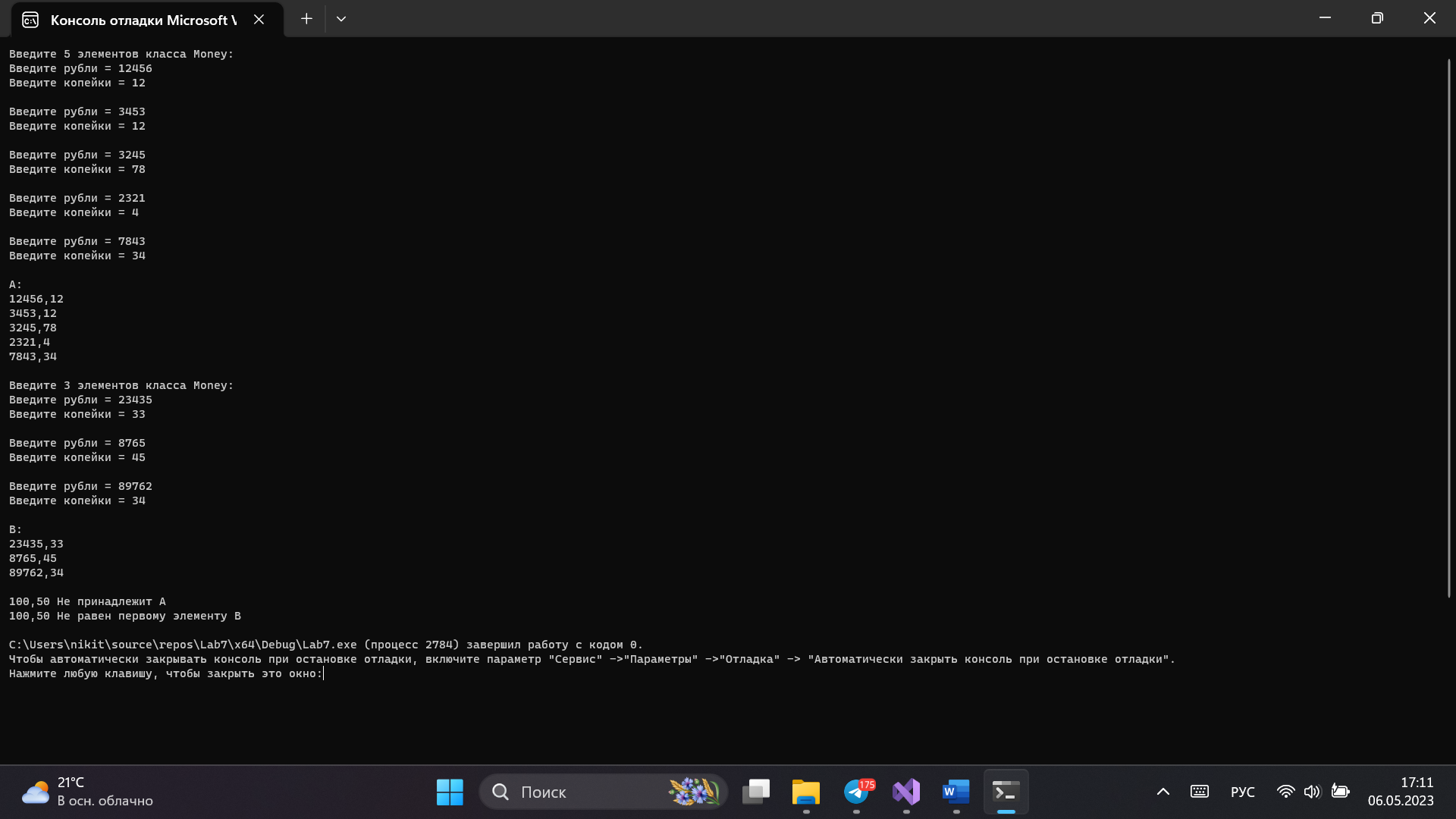
cout << x << " Равен первому элементу B" << endl;

return;

}

**Объяснение результатов работы программы.**

Программа демонстрирует работу с шаблонным классом Set, и пользовательским классом Money, создается 2 множества и сравниваются операциями < и !=, а так же проверяет принадлежность множеству.



**Ответы на контрольные вопросы**

**1. В чем смысл использования шаблонов?**

Шаблоны используются для создания универсального кода, который может работать с разными типами данных без повторения кода.

**2. Каковы синтаксис/семантика шаблонов функций?**

Синтаксис шаблонов функций выглядит так: template <typename T> void function\_name(T parameter). Семантика заключается в том, что функция использует параметр T для работы с разными типами данных.

**3. Каковы синтаксис/семантика шаблонов классов?**

Синтаксис шаблонов классов выглядит так: template <typename T> class class\_name { /\*...\*/ };. Семантика заключается в том, что класс использует параметр T для работы с разными типами данных внутри его компонентных функций и переменных.

**4. Что такое параметры шаблона функции?**

Параметры шаблона функции - это типы данных, которые передаются в шаблон функции как параметры в угловых скобках.

**5. Перечислите основные свойства параметров шаблона функции.**

Основные свойства параметров шаблона функции - это возможность работать с разными типами данных и упрощение кода, за счет универсальности функции.

**6. Как записывать параметр шаблона?**

Параметр шаблона записывается следующим образом: template <typename T>, где T - это имя параметра.

**7. Можно ли перегружать параметризованные функции?**

Да, параметризованные функции могут быть перегружены в зависимости от типов данных, переданных в шаблон.

**8. Перечислите основные свойства параметризованных классов.**

Основные свойства параметризованных классов - это возможность работы с разными типами данных и упрощение кода, за счет универсальности класса.

**9. Все ли компонентные функции параметризованного класса являются**

**параметризованными?**

Нет, не все компонентные функции параметризованного класса являются параметризованными. Компонентные функции могут использовать параметры шаблона класса.

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

Да, дружественные функции, описанные в параметризованном классе, также могут быть параметризованными.

**11. Могут ли шаблоны классов содержать виртуальные компонентные функции?**

Да, шаблоны классов могут содержать виртуальные компонентные функции.

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

Компонентные функции параметризованных классов определяются так же, как и обычные функции, но с использованием темплейта при определении функции за пределами класса.

**13. Что такое инстанцирование шаблона?**

Инстанцирование шаблона - это процесс создания экземпляра функции или класса с конкретными типами данных, переданными в качестве параметров шаблона.

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

Генерирование определения класса по шаблону происходит в момент компиляции и зависит от конкретных типов данных, переданных в качестве параметров шаблона при инстанцировании класса.