Пермский национальный исследовательский политехнический университет

**Лабораторная работа №5**

**Дисциплина: Информатика**

**Вариант № 9**

**"Наследование. Виртуальные функции. Полиморфизм."**

Выполнил: Зайченко Никита Геннадьевич

Проверила: Доцент кафедры ИТАС Полякова О. А.

Пермь 2022

**Содержание отчета**

1) Постановка задачи (общая и конкретного варианта).

2) Описание класса.

3) Определение компонентных функций.

4) Определение глобальных функций.

5) Функция main().

6) Объяснение результатов работы программы.

7) Ответы на контрольные вопросы.

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

1. Определить абстрактный класс.

2. Определить иерархию классов, в основе которой будет находиться абстрактный

класс (см. лабораторную работу №4).

3. Определить класс Вектор, элементами которого будут указатели на объекты

иерархии классов.

4. Перегрузить для класса Вектор операцию вывода объектов с помощью потоков.

5. В основной функции продемонстрировать перегруженные операции и

полиморфизм Вектора.

6. Базовый класс:

ТРОЙКА\_ЧИСЕЛ (TRIAD)

Первое\_число (first) - int

Второе\_число (second) – int

Третье\_число (third) - int

Определить методы изменения полей и сравнения триады.

Создать производный класс TIME с полями часы, минуты и секунды. Определить

полный набор операций сравнения временных промежутков.

**Описание класса.**

class Triad :

public Object

{

public:

Triad(void); //конструктор без параметров

public:

virtual ~Triad(void); //деструктор

void Show(); //функция для просмотра атрибутов класса с помощью указателя

Triad(int, int, int); //констрктор с параметрами

Triad(const Triad&); //конструктор копирования

int Get\_first() { return first; } //селектор

int Get\_second() { return second; } //селектор

int Get\_third() { return third; } //селектор

void Set\_first(int); //модификатор

void Set\_second(int); //модификатор

void Set\_third(int); //модификатор

Triad& operator++(); //перегрузка префиксной операции

Triad& operator=(const Triad&); //перегрузка операции присваивания

bool operator==(const Triad& l); //перегрузка операции сравнения

bool operator!=(const Triad& l); //перегрузка операции сравнения

friend istream& operator>>(istream& in, Triad& c); //операция ввода

friend ostream& operator<<(ostream& out, const Triad& c); //операция вывода

protected: //атрибуты

int first;

int second;

int third;

};

//класс Time наследуется от класса Triad

class Time :

public Triad

{

public:

Time(void);//конструктор без параметров

public:

~Time(void);//деструктор

Time(int, int, int);//конструктор с параметрами

Time(const Time&); //конструктор копирования

Time& operator++(); //перегрузка префиксной операции

Time& operator=(const Time&); //операция присваивания

bool operator>(const Time& l); //перегрузка операции сравнения

bool operator<(const Time& l); //перегрузка операции сравнения

bool operator>=(const Time& l); //перегрузка операции сравнения

bool operator<=(const Time& l); //перегрузка операции сравнения

void Show(); //функция для просмотра атрибутов класса с помощью указателя

void Set\_hours(int G); //модификатор

void Set\_minutes(int G); //модификатор

void Set\_seconds(int G); //модификатор

int Get\_hours() { return hours; } //селектор

int Get\_minutes() { return minutes; } //селектор

int Get\_seconds() { return seconds; } //селектор

friend istream& operator>>(istream& in, Time& l);//операция ввода

friend ostream& operator<<(ostream& out, const Time& l); //операция вывода

protected: //атрибуты

int hours;

int minutes;

int seconds;

};

class Vector

{

public:

Vector(void);//конструктор без параметров

Vector(int);//конструктор копирования

public:

~Vector(void);//деструктор

void Add(Object\*);

//добавление элемента в вектор

friend ostream& operator<<(ostream& out, const Vector&);//операция вывода

private:

Object\*\* beg;//указатель на первый элемент вектора

int size;//размер

int cur;//текущая позиция

};



**Определение компонентных функций.**

class Triad

//конструктор без параметров

Triad::Triad(void) { first = 0; second = 0; third = 0; }

//конструктор с параметрами

Triad::Triad(int M, int C, int P) { first = M; second = C; third = P; }

//конструктор копирования

Triad::Triad(const Triad& Triad) { first = Triad.first; second = Triad.second; third = Triad.third; }

//деструктор

Triad::~Triad(void) {}

//функция для просмотра атрибутов класса с помощью указателя

void Triad::Show(){ cout << "\nTriad: " << first << " : " << second << " : " << third;}

//модификаторы

void Triad::Set\_first(int M) { first = M; }

void Triad::Set\_second(int C){ second = C; }

void Triad::Set\_third(int P){ third = P; }

//перегрузка префиксной операции инкремент

Triad& Triad::operator++()

{

while(third > 60) { second += 1; third -= 60; }

while (second > 60) { first += 1; second -= 60; }

return \*this;

}

//перегрузка операции присваивания

Triad& Triad::operator=(const Triad& c)

{

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

first = c.first; second = c.second; third = c.third;

return \*this;

}

//перегрузка операции сравнения ==

bool Triad::operator==(const Triad& l) { return (first == l.first && second == l.second && third == l.third); }

//перегрузка операции сравнения !=

bool Triad::operator!=(const Triad& l) { return !(\*this == l); }

class Time

//конструктор без параметров

Time::Time(void) :Triad() { hours = 0; minutes = 0; seconds = 0; }

//конструктор с параметрами

Time::Time(int hours, int minutes, int seconds) :Triad(hours, minutes, seconds)

{

this->hours = hours; this->minutes = minutes; this->seconds = seconds;

}

//конструктор копирования

Time::Time(const Time& L) { hours = L.hours; minutes = L.minutes; seconds = L.seconds;}

//дестрктор

Time::~Time(void) {}

//модификаторы

void Time::Set\_hours(int H){ hours = H; }

void Time::Set\_minutes(int M){ minutes = M; }

void Time::Set\_seconds(int S){ seconds = S; }

//функция для просмотра атрибутов класса с помощью указателя

void Time::Show() { cout << "\nTime: " << hours << " : " << minutes << " : " << seconds; }

//оперция присваивания

Time& Time::operator=(const Time& l)

{

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

hours = l.hours; minutes = l.minutes; seconds = l.seconds;

return \*this;

}

//перегрузка префиксной операции инкремент

Time& Time::operator++()

{

while (seconds > 60) { minutes += 1; seconds -= 60; }

while (minutes > 60) { hours += 1;minutes -= 60; }

return \*this;

}

//перегрузка операции сравнения >

bool Time::operator>(const Time& l) {

if (hours > l.hours) { return true; }

else if (hours == l.hours && minutes > l.minutes) { return true; }

else if (hours == l.hours && minutes == l.minutes && seconds > l.seconds) { return true; }

else { return false; }

}

//перегрузка операции сравнения <

bool Time::operator<(const Time& l) {

if (hours < l.hours) { return true; }

else if (hours == l.hours && minutes < l.minutes) { return true; }

else if (hours == l.hours && minutes == l.minutes && seconds < l.seconds) { return true; }

else { return false; }

}

//перегрузка операции сравнения >=

bool Time::operator>=(const Time& l) {

if (hours > l.hours) { return true; }

else if (hours == l.hours && minutes > l.minutes) { return true; }

else if (hours == l.hours && minutes == l.minutes && seconds > l.seconds) { return true; }

else if (hours == l.hours && minutes == l.minutes && seconds == l.seconds) { return true; }

else { return false; }

}

//перегрузка операции сравнения <=

bool Time::operator<=(const Time& l) {

if (hours < l.hours) { return true; }

else if (hours == l.hours && minutes < l.minutes) { return true;}

else if (hours == l.hours && minutes == l.minutes && seconds < l.seconds) { return true; }

else if (hours == l.hours && minutes == l.minutes && seconds == l.seconds) { return true; }

else { return false; }

}

class Vector

//конструктор без параметров

Vector::Vector(void) { beg = 0; size = 0; cur = 0; }

//деструктор

Vector::~Vector(void)

{

if (beg != 0)delete[] beg;

beg = 0;

}

//конструктор с параметрами

Vector::Vector(int n) { beg = new Object \* [n]; cur = 0; size = n; }

//добавление объекта, на который указывает указатель p в вектор

void Vector::Add(Object\* p) { if (cur < size) { beg[cur] = p; cur++; } }

**Определение глобальных функций.**

class Triad

//глобальная функция для ввода

istream& operator>>(istream& in, Triad& c)

{

cout << "\nfirst: "; in >> c.first;

cout << "second: "; in >> c.second;

cout << "third: "; in >> c.third;

return in;

}

//глобальная функция для вывода

ostream& operator<<(ostream& out, const Triad& c)

{

out << "\nTriad: " << c.first << " : " << c.second << " : " << c.third;

return out;

}

class Time

//операция ввода

istream& operator>>(istream& in, Time& l)

{

cout << "\nhours: "; in >> l.hours;

cout << "minutes: "; in >> l.minutes;

cout << "seconds: "; in >> l.seconds;

return in;

}

//операция вывода

ostream& operator<<(ostream& out, const Time& l)

{

out << "\nTime: " << l.hours << " : " << l.minutes << " : " << l.seconds;

return out;

}

class Vector

//операция вывода

ostream& operator<<(ostream& out, const Vector& v)

{

if (v.size == 0) out << "Empty" << endl;

Object\*\* p = v.beg;//указатель на указатель типа Object

for (int i = 0; i < v.cur; i++)

{

(\*p)->Show();//вызов метода Show() (позднее связывание) полифоризм

p++;//передвигаем указатель на следующий объект

}

return out;

}

**Функция main().**

void main()

{

setlocale(LC\_ALL, "Russian");

//работа с классом Triad

cout << "Работа с классом Triad:";

Triad a;

cin >> a; ++a;

cout << a;

Triad b(10, 86, 115); ++b;

cout << b;

if(a==b){ cout << endl << "Временные промежутки a и b равны"; }

else{ cout << endl << "Временные промежутки a и b не равны"; }

if(a != b){ cout << endl << "Временные промежутки a и b не равны" << endl;}

else{ cout << endl << "Временные промежутки a и b равны" << endl; }

//работа с классом Time

cout << endl << "Работа с классом Time:";

Time c;

cin >> c; ++c;

cout << c;

Time d(10, 86, 115); ++d;

cout << d;

if (c > d) { cout << endl << "Временной промежуток c больше d"; }

else { cout << endl << "Временной промежуток c меньше d"; }

if (c < d) { cout << endl << "Временной промежуток c меньше d" << endl; }

else { cout << endl << "Временной промежуток c больше d" << endl; }

if (c >= d) { cout << "Временной промежуток c больше или равен d"; }

else { cout << "Временной промежуток c меньше d"; }

if (c <= d) { cout << endl << "Временной промежуток c меньше или равен d" << endl; }

else { cout << endl << "Временной промежуток c больше d" << endl; }

//работа с Vector

cout << endl << "Работа c Vector:";

Vector v(5);//вектор из 5 элементов

Triad f;//объект класса Triad

cin >> f; ++f;

Time g;// объект класса Time

cin >> g; ++g;

Object\* p = &f;//ставим указатель на объект класса Triad

v.Add(p);//добавляем объект в вектор

p = &g;//ставим указатель на объект класса Time

v.Add(p); //добавляем объект в вектор

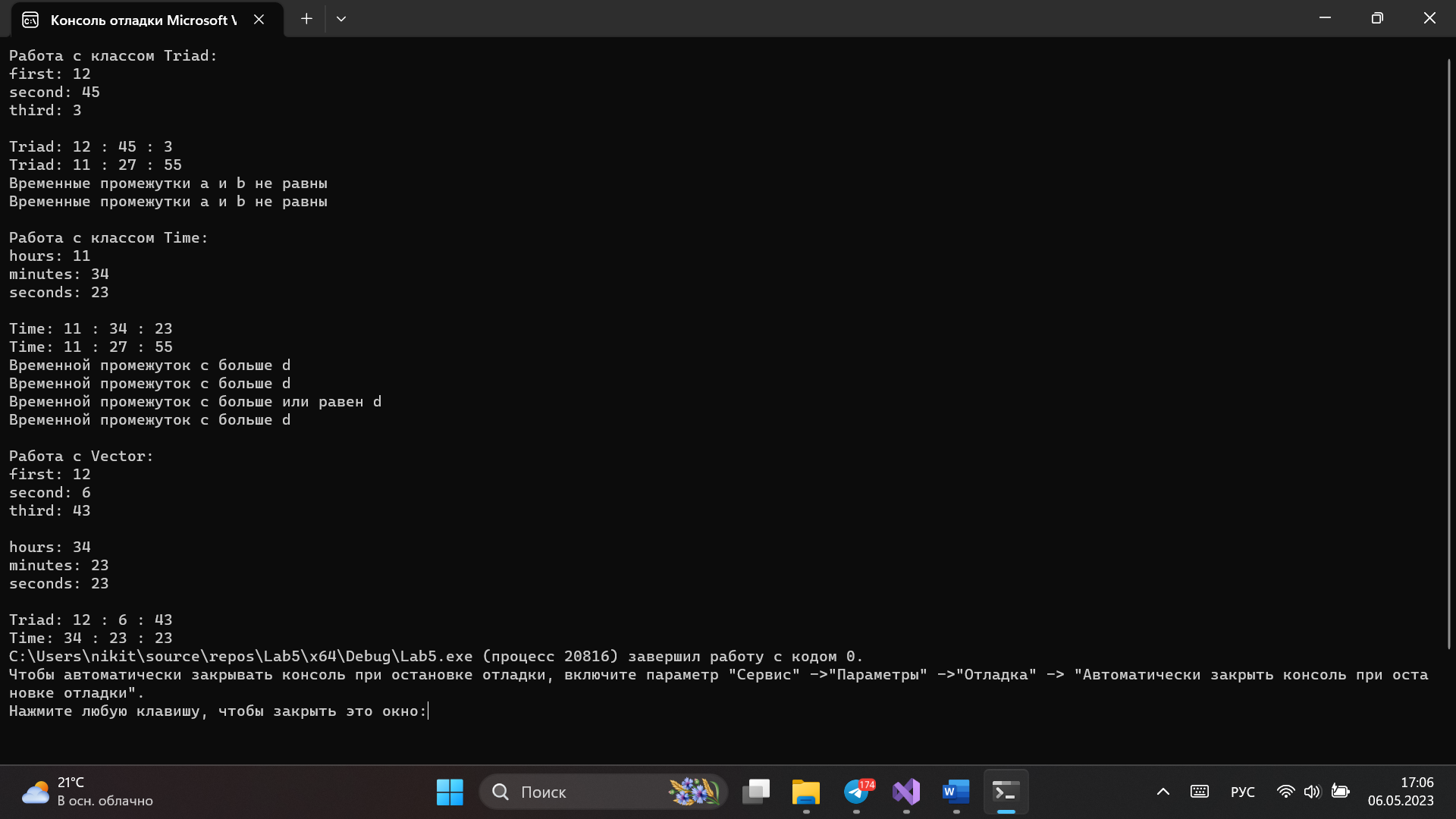
cout << v;//вывод вектора

}

**Объяснение результатов работы программы.**

В данном коде представлена работа с временными промежутками, с помощью класса Tried и класса Time, который наследуется от класса Tried.

А также продемонстрированы перегруженные операции и полиморфизм Вектора и методы сравнения временных промежутков.



**Ответы на контрольные вопросы**

**1. Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?**

Чисто виртуальный метод - это метод в абстрактном классе, для которого не определена реализация, только заголовок функции с ключевым словом "=0". Отличие от виртуального метода заключается в том, что в классе, содержащем чисто виртуальный метод, нельзя создать экземпляр (объект).

**2. Какой класс называется абстрактным?**

Абстрактный класс - это класс, содержащий как минимум один чисто виртуальный метод. Он не может быть использован для создания объекта, только в качестве базового класса для производных классов.

**3. Для чего предназначены абстрактные классы?**

Абстрактные классы предназначены для определения общих свойств и методов, которые будут использоваться во всех производных классах. Они позволяют задать общий интерфейс для классов, не определяя реализацию каждого класса отдельно.

**4. Что такое полиморфные функции?**

Полиморфные функции - это функции, которые могут принимать разные типы данных и соответственно выполнять различные действия. Это достигается благодаря механизму перегрузки функций.

**5. Чем полиморфизм отличается от принципа подстановки?**

Полиморфизм - это способность объектов разных типов обрабатываться одинаково, если они имеют общий интерфейс. Принцип подстановки же заключается в том, что экземпляры производного класса могут использоваться вместо экземпляров базового класса без изменения работы программы.

**6. Привести примеры иерархий с использованием абстрактных классов.**

Примеры иерархий с использованием абстрактных классов: класс "Животное" может быть абстрактным, от него могут быть производные классы "Собака", "Кошка", "Птица" и т.д. Также можно создать абстрактный класс "Транспорт", от него может быть производный класс "Автомобиль", "Самолет", "Корабль" и т.д.

**7. Привести примеры полиморфных функций.**

Примеры полиморфных функций: функция "вывода на экран" может работать с объектами разных типов, функция "расчета площади" может принимать на вход разные геометрические фигуры, например, "Круг", "Прямоугольник", "Треугольник".

**8. В каких случаях используется механизм позднего связывания?** Механизм позднего связывания (или динамическое связывание) используется в случаях, когда вызываемый метод определяется только во время выполнения программы, а не на этапе компиляции. Это происходит, когда метод является виртуальным или когда ссылка на объект имеет тип базового класса.