Пермский национальный исследовательский политехнический университет

**Лабораторная работа №4**

**Дисциплина: Информатика**

**Вариант № 9**

**"Простое наследование. Принцип подстановки"**

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) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Создание иерархии классов с использованием простого наследования.

3) Изучение принципа подстановки.4) 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:

Triad(void); //конструктор без параметров

public:

virtual ~Triad(void); //деструктор

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++();

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++();

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 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;

};



**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) {}

//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; }

//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; }

}

**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;

}

**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; }

//принцип подстановки

cout << endl << "Принцип подстановки: ";

f1(c);//передаем объект класса Lorry

a = f2();//создаем в функции объект класса Lorry

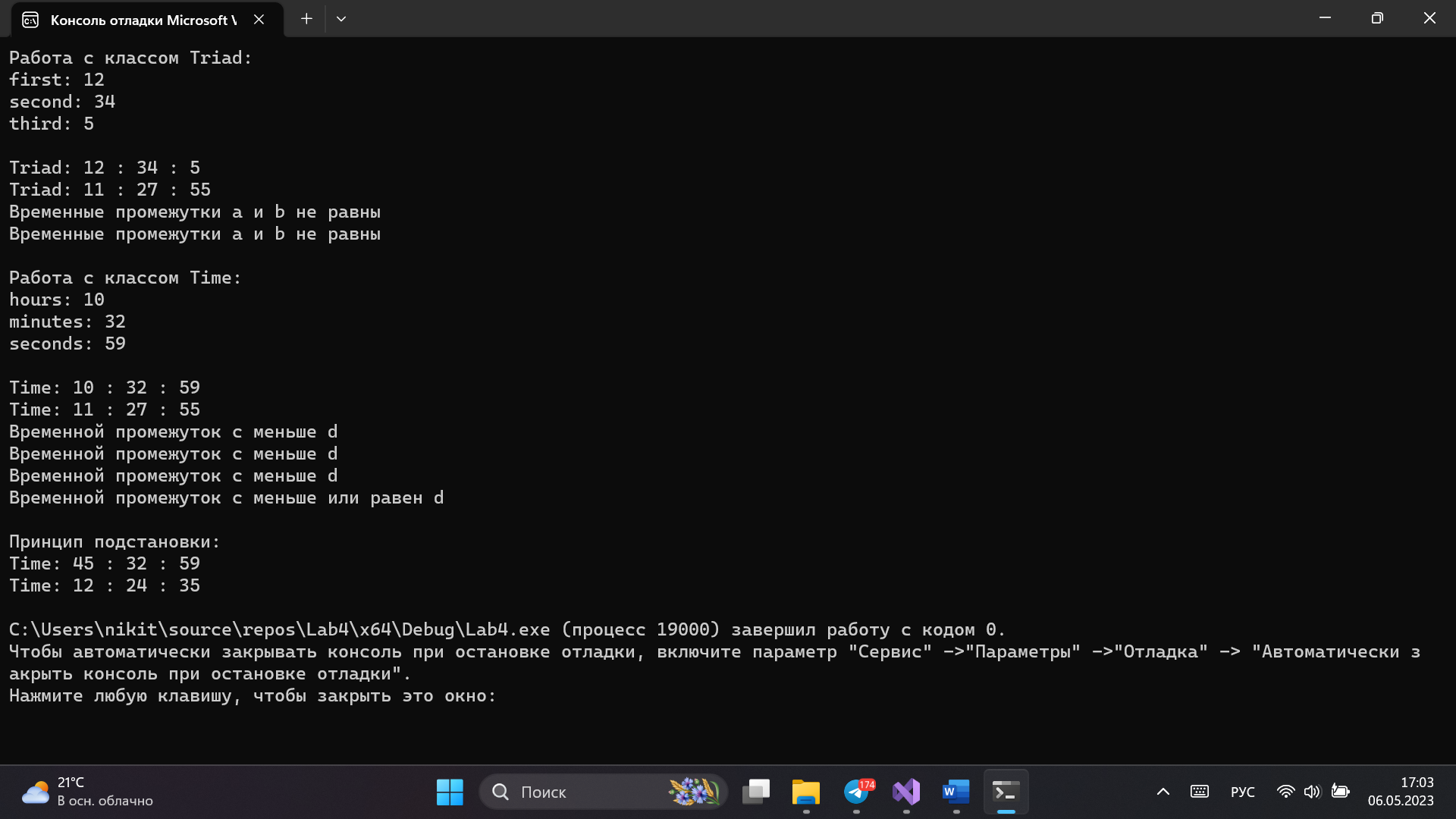
cout << endl;

}

**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 определены методы сравнения временных промежутков.



**Ответы на контрольные вопросы**

**1. Для чего используется механизм наследования?**

Механизм наследования в объектно-ориентированном программировании используется для создания нового класса на основе уже существующего класса, который называется базовым или родительским классом. При этом новый класс, который называется производным или потомком, наследует все свойства и методы базового класса, а также может добавлять свои собственные свойства и методы.

**2. Каким образом наследуются компоненты класса, описанные со спецификатором public?**

Компоненты класса, описанные со спецификатором public, наследуются производным классом без изменений и могут быть использованы в производном классе так же, как в базовом классе.

**3. Каким образом наследуются компоненты класса, описанные со спецификатором private?**

Компоненты класса, описанные со спецификатором private, не наследуются производным классом. Таким образом, производный класс не имеет доступа к private свойствам и методам базового класса.

**4. Каким образом наследуются компоненты класса, описанные со спецификатором protected?**

Компоненты класса, описанные со спецификатором protected, наследуются производным классом, но не могут быть использованы в любом месте в производном классе. Они могут быть использованы только в методах производного класса или в его производных классах.

**5. Каким образом описывается производный класс?**

Производный класс описывается ключевым словом class, за которым идет имя класса. Затем в фигурных скобках описываются свойства и методы производного класса, которые могут быть унаследованы от базового класса или добавлены производным классом. Для того, чтобы указать базовый класс, используется ключевое слово extends, за которым идет имя базового класса.

**6. Наследуются ли конструкторы?**

Конструкторы наследуются, но при этом конструктор производного класса должен вызывать конструктор базового класса для инициализации членов, которые были унаследованы.

**7. Наследуются ли деструкторы?**

Деструкторы также наследуются, но при этом деструктор производного класса должен вызывать деструктор базового класса для корректной очистки памяти.

**8. В каком порядке конструируются объекты производных классов?**

Сначала конструируется базовый класс, затем в порядке наследования конструируются производные классы.

**9. В каком порядке уничтожаются объекты производных классов?**

В порядке, обратном порядку конструирования.

**10. Что представляют собой виртуальные функции и механизм позднего связывания?**

Виртуальные функции — это функции, которые могут быть переопределены производными классами. Механизм позднего связывания позволяет вызывать функцию на основе типа объекта, а не на основе типа ссылки или указателя на объект. Таким образом, механизм позднего связывания позволяет единому интерфейсу, определенному базовым классом, иметь различные реализации в каждом производном классе.

**11. Могут ли быть виртуальными конструкторы? Деструкторы?**

Конструкторы не могут быть виртуальными, так как они создают объект, а виртуальность определяется на основе объекта. Деструкторы могут быть виртуальными, чтобы гарантировать полное удаление объекта, когда он уничтожается через ссылку базового класса.

**12. Наследуется ли спецификатор virtual?**

Спецификатор virtual не наследуется, он должен быть указан в каждом классе заново.

**13. Какое отношение устанавливает между классами открытое наследование?**

Открытое наследование устанавливает отношение "является" между базовым и производным классами. Производный класс является расширенной версией базового класса, который дополняет его функциональность.

**14. Какое отношение устанавливает между классами закрытое наследование?**

Закрытое наследование устанавливает отношение "реализует" между базовым и производным классами. Производный класс реализует интерфейс базового класса и не является его расширенной версией.

**15. В чем заключается принцип подстановки?**

Принцип подстановки заключается в том, что объекты производных классов могут быть использованы везде, где ожидается объект базового класса, при этом не нарушая работу программы. То есть производный класс должен быть совместим с базовым классом, чтобы можно было использовать его вместо базового класса.

**16. Имеется иерархия классов:**

**class Student**

**{**

**int age;**

**public:**

**string name;**

**... };**

**class Employee : public Student**

**{**

**protected:**

**string post;**

**... };**

**class Teacher : public Employee**

**{**

**protected: int stage;**

**... };**

**Teacher x;**

**Какие компонентные данные будет иметь объект х?**

Объект x класса Teacher будет иметь компоненты age, name, post и stage, унаследованные от базовых классов.

**17. Для классов Student, Employee и Teacher написать конструкторы без параметров.**

Student() { age = 0; name = "" }

Employee() { post = ""; }

Teacher() { stage = 0; }

**18. Для классов Student, Employee и Teacher написать конструкторы с параметрами.**

Student(int a, string n) { age = a; name = n; }

Employee(string p) { post = p; }

Teacher(int s) { stage = s; }

**19. Для классов Student, Employee и Teacher написать конструкторы копирования.**

Student(const Student& other) {

age = other.age;

name = other.name;

}

Employee(const Employee& other) : Student(other) {

post = other.post;

}

Teacher(const Teacher& other) : Employee(other) {

stage = other.stage;

}

**20. Для классов Student, Employee и Teacher определить операцию присваивания.**Student& operator=(const Student& other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

Employee& operator=(const Employee& other) {

if (this != &other) {

Student::operator=(other);

post = other.post;

}

return \*this;

}

Teacher& operator=(const Teacher& other) {

if (this != &other) {

Employee::operator=(other);

stage = other.stage;

}

return \*this;

}