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**

Тема: «Простое наследование. Принцип подстановки»

Выполнил работу

Студент группы РИС-24-1Б

Конькова С. С.

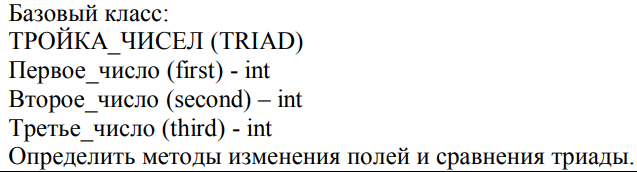
Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь 2025

Постановка задачи.





1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Определить производный класс.

8. Написать программу, в которой продемонстрировать создание объектов и работу

всех перегруженных операций.

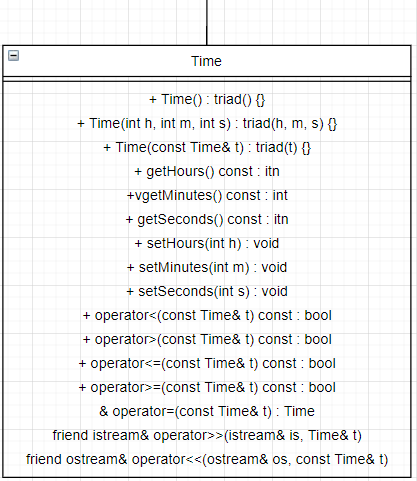
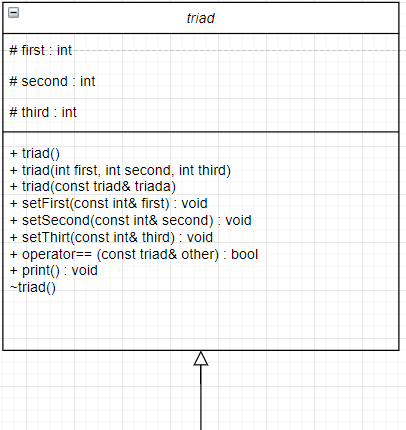
9. Реализовать функции, получающие и возвращающие объект базового класса.

Продемонстрировать принцип подстановки.

**Анализ**

1. Пользователь вводит две тройки чисел
2. Создаются объекты tr1 и tr2 класса triad
3. Триады выводятся на экран
4. Выполняется сравнение триад на равенство
5. Пользователь вводит два временных значения (часы, минуты, секунды)
6. Создаются объекты t1 и t2 класса Time
7. Временные значения выводятся в формате ЧЧ:ММ:СС
8. Выполняется сравнение временных интервалов

UML диаграмма.



Код программы.

Sourse.cpp

#include"Header.h"

int main()

{

system("chcp 1251 >> null");

int x, y, z;

cout << "Введите три целых числа: ";

cin >> x >> y >> z;

triad tr1(x, y, z);

cout << "\nПервая триада: ";

tr1.print();

cout << "\n\nВведите три целых числа: ";

cin >> x >> y >> z;

triad tr2(x, y, z);

cout << "\nВторая триада: ";

tr2.print();

if (tr1 == tr2)

cout << "\nТриады равны\n";

else

cout << "\nТриады не равны\n";

Time t1;

cin >> t1;

cout << "\nВведённое время: "<<t1;

Time t2;

cout << "\n\nВведите новое время\n";

cin >> t2;

cout << "\nВведённое время: " << t2;

if (t1 > t2)

cout << "\n\nПервое время больше\n" << t1;

else if (t1 < t2)

cout << "\n\nВторое время больше\n"<<t2;

else

cout << "\nВременные промежутки равны" << t1 << " = " << t2;

return 0;

}

Header.h

#pragma once

#include<iostream>

using namespace std;

class triad {

protected:

int first, second, third;

public:

triad()

{

first = 0;

second = 0;

third = 0;

}

triad(int f, int s, int t)

{

first = f;

second = s;

third = t;

}

triad(const triad& t)

{

first = t.first;

second = t.second;

third = t.third;

}

int getFirst() const;

int getSecond() const;

int getThird() const;

void setFirst(int f);

void setSecond(int s);

void setThird(int t);

void print();

bool operator==(const triad& t) const;

bool operator!=(const triad& t) const;

~triad() {}

};

class Time : public triad

{public:

Time() : triad() {}

Time(int h, int m, int s) : triad(h, m, s) {}

Time(const Time& t) : triad(t) {}

int getHours() const;

int getMinutes() const;

int getSeconds() const;

void setHours(int h);

void setMinutes(int m);

void setSeconds(int s);

bool operator<(const Time& t) const;

bool operator>(const Time& t) const;

bool operator<=(const Time& t) const;

bool operator>=(const Time& t) const;

Time& operator=(const Time& t);

friend istream& operator>>(istream& is, Time& t);

friend ostream& operator<<(ostream& os, const Time& t);

};

Header.cpp

#include"Header.h"

int triad:: getFirst() const

{

return first;

}

int triad:: getSecond() const

{

return second;

}

int triad:: getThird() const

{

return third;

}

void triad:: setFirst(int f)

{

first = f;

}

void triad:: setSecond(int s)

{

second = s;

}

void triad:: setThird(int t)

{

third = t;

}

void triad:: print()

{

cout << "(" << first << ", " << second << ", " << third << ")";

}

bool triad:: operator==(const triad& t) const

{

return (first == t.first && second == t.second && third == t.third);

}

bool triad:: operator!=(const triad& t) const

{

return !(operator==(t));

}

int Time:: getHours() const

{

return first;

}

int Time:: getMinutes() const

{

return second;

}

int Time:: getSeconds() const

{

return third;

}

void Time:: setHours(int h)

{

first = h;

}

void Time:: setMinutes(int m)

{

second = m;

}

void Time:: setSeconds(int s)

{

third = s;

}

bool Time:: operator<(const Time& t) const

{

if (first < t.first) return true;

else if (first == t.first && second < t.second) return true;

else if (first == t.first && second == t.second && third < t.third) return true;

else return false;

}

bool Time:: operator>(const Time& t) const

{

return !(operator<(t) || operator==(t));

}

bool Time:: operator<=(const Time& t) const

{

return operator<(t) || operator==(t);

}

bool Time:: operator>=(const Time& t) const

{

return !operator<(t);

}

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

{

if (this != &t)

{

first = t.first;

second = t.second;

third = t.third;

}

return \*this;

}

istream& operator>>(istream& is, Time& t)

{

cout << endl << "Введите часы: ";

is >> t.first;

cout << "Введите минуты: ";

is >> t.second;

cout << "Введите секунды: ";

is >> t.third;

return is;

}

ostream& operator<<(ostream& os, const Time& t)

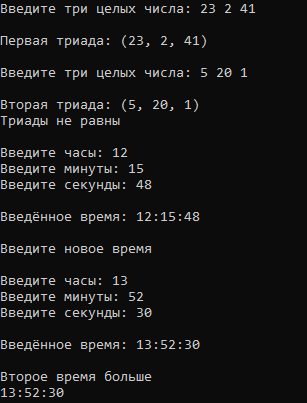
{

os << t.first << ":" << t.second << ":" << t.third;

return os;

}

Вывод.



Контрольные вопросы

1. Для чего используется механизм наследования?

- Для создания новых классов, на основе уже существующих.

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

Компоненты класса, описанные со спецификатором public, становятся доступными в базовом классе. Если тип наследования public, то становятся доступными для обращения из экземпляра производного класса. Если тип наследования protected, то доступны только внутри базового класса.

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

Компоненты, описанные со спецификатором private, не доступны в производном классе.

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

-Компоненты, описанные со спецификатором protected, при типе наследования public/protected остаются protected – доступны в производном классе и в классах-наследниках производного. При типе наследования private, становятся private – Доступны только внутри производного класса.

5. Каким образом описывается производный класс?

Class <имя> : <тип наследования> <имя базового>,…

6. Наследуются ли конструкторы?

-Конструкторы не наследуются, однако могут быть вызваны для передачи параметров.

7. Наследуются ли деструкторы?

- Деструкторы не наследуются.

8. В каком порядке конструируются объекты производных классов?

-Сначала Компоненты базового класса, потом базовый класс, потом компоненты производного класса, затем производный. компоненты конструируются в том порядке, в котором они записаны.

9. В каком порядке уничтожаются объекты производных классов?

В обратном порядке создания. Сначала производный класс, затем компоненты производного (в обратном порядке их записи), затем базовый класс, затем компоненты базового класса(в обратном порядке их записи).

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

Виртуальные функции – это функции которые могут быть переопределены в производном. Используя виртуальную функцию, мы можем вызывать функции производного класса, используя указатель базового класса.

Когда в классе объявляется виртуальный метод, то создается специальная таблица для этого класса. Эта таблица представляет собой массив указателей на виртуальные функции. Во время компиляции нет информации о том, какой метод будет вызван. Вызов метода выполняется во время выполнения программы.

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы виртуальными быть не могут, в отличие от деструкторов.

12. Наследуется ли спецификатор virtual?

Виртуальность наследуется.

13. Какое отношение устанавливает между классами открытое наследование?

- Открытое наследование является наследованием типа. Т. е. производный класс является частным случаем базового класса (если не происходит переопределение методов класса – для реализации принципа подстановки). Или производный класс может замещать методы базового класса (в том случае, если компонентные функции базового класса описаны со спецификатором virtual – для реализации полиморфизма).

14. Какое отношение устанавливает между классами закрытое наследование?

Производный имеет доступ ко всему функционалу, за исключением private компонентов, класса-предка, но ни наследники, ни экземпляры класса не имеют доступ к этому функционалу. Это нужно для того, чтобы средствами базового класса реализовать новый класс.

15. В чем заключается принцип подстановки?

- Если класс D является частным случаем класса B, то везде, где был использован класс B, можно использовать класс D. Принцип подстановки возможен, если не были переопределены методы 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?

- Через объект x можно обратиться только к полю name. Но компонентные функции и друзья класса Teacher имеют доступ к protected полям string post и int stage;