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»

Тема: «Простое наследование. Принцип подстановки.»

Выполнил работу

Студент группы РИС-22-1Б

Шуракова А.А.

Проверил

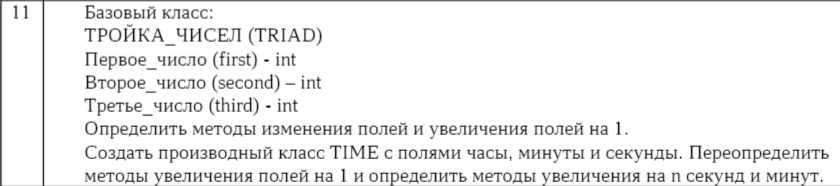
Доцент кафедры ИТАС

Полякова О.А.

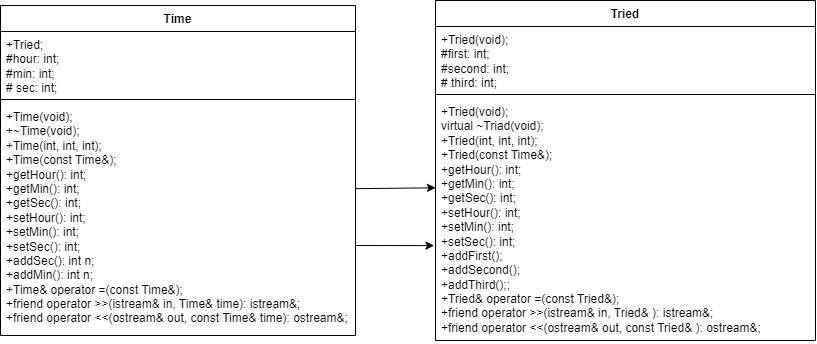
Г. Пермь-2024

**Постановка задачи:**

* Определить пользовательский класс.
* Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
* Определить в классе деструктор.
* Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
* Перегрузить операцию присваивания.
* Перегрузить операции ввода и вывода объектов с помощью потоков.
* Определить производный класс.
* Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.
* Реализовать функции, получающие и возвращающие объект базового класса.
* Продемонстрировать принцип подстановки.



**Uml диаграмма:**

****

**Код программы:**

Triad.h

#pragma once

#include <iostream>

using namespace std;

class Triad {

//конструктор без параметров

public:

Triad(void);

public:

//деструктор

virtual ~Triad(void);

//конструктор с параметрами

Triad(int, int, int);

//конструктор копирования

Triad(const Triad&);

//модификаторы

int getFirst() { return first; }

int getSecond() { return second; }

int getThird() { return third; }

//селекторы

void setFirst(int);

void setSecond(int);

void setThird(int);

void addFirst();

void addSecond();

void addThird();

//перегрузка операции присваивания

Triad& operator = (const Triad&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, Triad& T);

friend ostream& operator<<(ostream& out, const Triad& T);

//атрибуты

protected:

int first;

int second;

int third;

};

Triad.cpp

#include <iostream>

#include <fstream>

#include "Triad.h"

using namespace std;

//конструктор без параметров

Triad::Triad(void) {

first = 0;

second = 0;

third = 0;

}

//деструктор

Triad::~Triad(void) {}

//конструктор с параметрами

Triad::Triad(int F, int S, int T) {

first = F;

second = S;

third = T;

}

//конструктор копирования

Triad::Triad(const Triad& triad) {

first = triad.first;

second = triad.second;

third = triad.third;

}

//модификаторы

void Triad::setFirst(int F) {

first = F;

}

void Triad::setSecond(int S) {

second = S;

}

void Triad::setThird(int T) {

third = T;

}

void Triad::addFirst() {

first++;

}

void Triad::addSecond() {

second++;

}

void Triad::addThird() {

third++;

}

//перегрузка операции присваивания

Triad& Triad::operator=(const Triad& T) {

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

first = T.first;

second = T.second;

third = T.third;

return \*this;

}

//глобальная функция для вывода

istream& operator>>(istream& in, Triad& T) {

cout << "Enter first:";

in >> T.first;

cout << "Enter second:";

in >> T.second;

cout << "Enter third:";

in >> T.third;

return in;

}

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

{

out << "\nfirst: " << T.first;

out << "\nsecond: " << T.second;

out << "\nthird: " << T.third;

out << "\n";

return out;

}

Time.h

#pragma once

#include <iostream>

#include "Triad.h"

using namespace std;

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

//модификаторы

int getHour() { return hour; }

int getMin() { return min; }

int getSec() { return sec; }

//селекторы

void setHour(int);

void setMin(int);

void setSec(int);

//увеличение на 1

//void addFirst();

//void addSecond();

//void addThird();

//увеличение на мин и сек

void addSec(int n);

void addMin(int n);

//перегрузка операции присваивания

Time& operator =(const Time&);

//глобальные операторы-функции ввода-вывода

friend istream& operator >>(istream& in, Time& time);

friend ostream& operator <<(ostream& out, const Time& time);

//атрибуты

protected:

int hour, min, sec;

};

Time.cpp

#include <iostream>

#include "Time.h"

using namespace std;

Time::Time(void) :Triad() {

hour = 0;

min = 0;

sec = 0;

}

Time::~Time(void) {}

Time::Time(int H, int M, int S) {

hour = H;

min = M;

sec = S;

}

Time::Time(const Time& time) {

hour = time.hour;

sec = time.sec;

min = time.min;

}

/\*

void Time::addFirst() {

first++;

if (first > 23) {

first = 0;

}

}

void Time::addSecond() {

second++;

if (second > 59) {

second = 0;

addFirst();

}

}

void Time::addThird() {

third++;

if (third > 59) {

third = 0;

addSecond();

}

}

\*/

void Time::addMin(int n) {

min+=n;

while (min > 59) {

min -= 60;

hour++;

}

}

void Time::addSec(int n) {

sec+=n;

while (sec > 59) {

sec -= 60;

min++;

}

}

void Time::setHour(int H) {

hour = H;

}

void Time::setMin(int M) {

min = M;

}

void Time::setSec(int S) {

sec = S;

}

Time& Time::operator =(const Time& time) {

if (&time == this) {

return \*this;

}

first = time.first;

second = time.second;

third = time.third;

hour = time.hour;

min = time.min;

sec = time.sec;

return \*this;

}

istream& operator >>(istream& in, Time& time) {

/\*

cout << "Enter First:";

in >> time.first;

cout << "Enter Second:";

in >> time.second;

cout << "Enter Third:";

in >> time.third;

\*/

cout << "Enter Hour:";

in >> time.hour;

cout << "Enter Minutes:";

in >> time.min;

cout << "Enter Seconds:";

in >> time.sec;

return in;

}

ostream& operator <<(ostream& out, const Time& time) {

/\*

out << "\nFirst: " << time.first;

out << "\nSecond: " << time.second;

out << "\nThird: " << time.third;

\*/

out << "\nHour: " << time.hour;

out << "\nMinutes: " << time.min;

out << "\nSeconds: " << time.sec;

return out;

}

Lana4\_main.cpp

#include <iostream>

#include "Time.h"

#include "Triad.h"

using namespace std;

int main() {

system("chcp 1251");

Time time;

cin >> time;

cout << time << endl;

int n;

cout << "На сколько секунд увеличить ";

cin >> n;

time.addSec(n);

cout << "После увеличения на " << n << " секунд : ";

cout << time<<endl;

cout << "На сколько минут увеличить ";

cin >> n;

time.addMin(n);

cout << "После увеличения на " << n << " минут : ";

cout << time<<endl;

cout << endl;

cout << "переопределение методов увеличения на 1" << endl;

Triad f;

cin >> f;

f.addFirst();

f.addSecond();

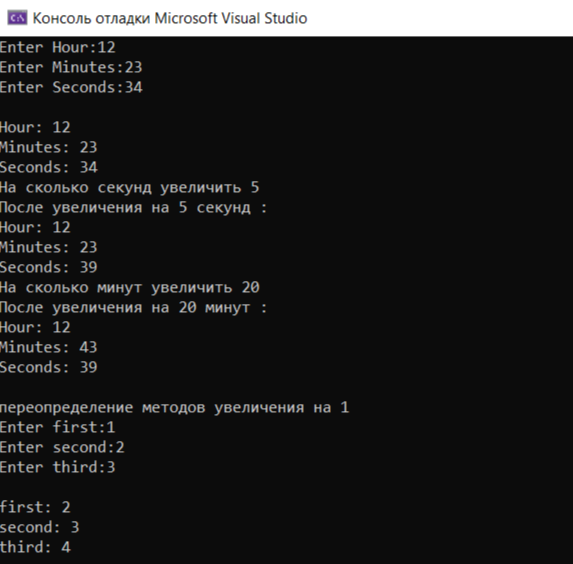
f.addThird();

cout << f;

return 0;

}

**Результат работы программы:**

****

**Ответы на контрольные вопросы:**

1. Для чего используется механизм наследования?

Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах.

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

Член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к public-членам возможен доступ извне через имя объекта.

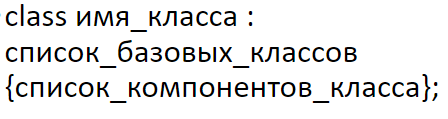
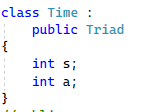
3. Каким образом наследуются компоненты класса, описанные со спецификатором private?

Член класса может использоваться только функциями- членами данного класса и функциями-«друзьями» своего класса. В производном классе он недоступен;

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

То же, что и private, но дополнительно член класса с данным атрибутом доступа может использоваться функциями–членами и функциями-«друзьями» классов, производных от данного;

5. Каким образом описывается производный класс?

  
6. Наследуются ли конструкторы?

Не наследуются.

7. Наследуются ли деструкторы?

не наследуются, но они вызываются, когда дочерний класс инициализирует свой объект.

8. В каком порядке конструируются объекты производных классов?

Объекты класса конструируются снизу вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом объект производного класса содержит в качестве подобъекта объект базового класса.

9. В каком порядке уничтожаются объекты производных классов?

Деструкторы вызываются в обратном порядке от конструкторов.

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

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе необходимо, чтобы функции-компоненты работали иначе, чем базовом.

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы нет, деструкторы да.

12. Наследуется ли спецификатор virtual?

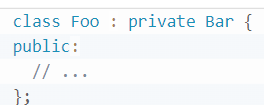
Да

13. Какое отношение устанавливает между классами открытое наследование?

Класс потомок имеет доступ ко всем возможностям класса предка.

14. Какое отношение устанавливает между классами закрытое наследование?

Закрытое наследование означает реализацию посредством - т.е. от базового класса необходимо взять какую-то функциональность, базовый класс и потомок не имеют како-либо концептуальной связи.Закрытое наследование не носит характера отношения подтипов.Закрытое (также как и защищенное) наследование не создает иерархии типов.



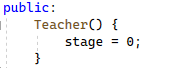
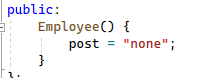
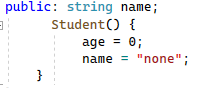
15. В чем заключается принцип подстановки?

Везде, где можно использован объект базового класса, можно использовать вместо него и объект производного.

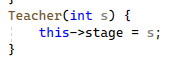
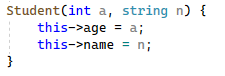
16. Имеется иерархия классов:  
class Student  
{  
int age;  
public:  
};  
string name;  
class Employee : public Student  
{  
protected:  
};  
string post;  
class Teacher public Employee  
{  
protected: int stage;  
};  
Teacher x;  
Какие компонентные данные будет иметь объект х?

Name и Stage, private age не наследуется, post наследуется в private секции.

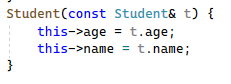
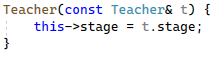
17. Для классов Student, Employee и Teacher написать конструкторы без параметров.



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



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

20. Для классов Student, Employee и Teacher определить операцию присваивания.

