**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.2

Вариант №11

Выполнил студент группы РИС-20-2б

Савельев Виталий Антонович

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с одним из механизмов объектно-ориентированного программирование – с наследованием классов.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, демонстрирующую работу механизма наследования классов и метода подстановки детей и родителей.

**Исходные данные для варианта №11:**

* Класс-родитель – тройка чисел;
* Класс-ребёнок – время (состоит из трёх чисел);

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

Объявление класса-родителя «Тройка чисел» в заголовочном файле:

#pragma once

#include <iostream>

using namespace std;

class Triad

{

protected:

int \_first;

int \_second;

int \_third;

public:

int First() { return \_first; }

int Second() { return \_second; }

int Third() { return \_third; }

virtual void SetTriad(int first, int second, int third);

virtual void SetFirst(int first);

virtual void SetSecond(int second);

virtual void SetThird(int third);

virtual void IncFirst();

virtual void IncSecond();

virtual void IncThird();

Triad& operator=(Triad& triad);

friend istream& operator>>(istream& input, Triad& triad);

friend ostream& operator<<(ostream& output, Triad& triad);

Triad();

Triad(int first, int second, int third);

Triad(Triad& copied);

};

Производный от него класс «Время»:

#pragma once

#include "Triad.h"

#include <iostream>

using namespace std;

class Time : public Triad

{

public:

int Hour() { return \_first; }

int Minute() { return \_second; }

int Second() { return \_third; }

void SetFirst(int first);

void SetSecond(int second);

void SetThird(int third);

void SetTriad(int first, int second, int third);

void IncFirst();

void IncSecond();

void IncThird();

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

friend ostream& operator<<(ostream& output, Time& time);

Time();

Time(int first, int second, int third);

Time(Time& copied);

};

Класс-ребёнок наследует от «родителя» все его методы и конструктор с private-полями; Такие решение позволяет создавать множество похожих объектов без необходимости переписывать много кода;

Различия в реализации модификаторов полей классов:

Тройка чисел:

void Triad::SetTriad(int first, int second, int third)

{

\_first = first;

\_second = second;

\_third = third;

}

void Triad::SetFirst(int first)

{

\_first = first;

}

void Triad::SetSecond(int second)

{

\_second = second;

}

void Triad::SetThird(int third)

{

\_third = third;

}

Время (закладываются дополнительные проверки на временные промежутки):

void Time::SetFirst(int first)

{

\_first = first % 24;

}

void Time::SetSecond(int second)

{

if (second >= 0)

{

if (second >= 60)

{

SetFirst(\_first + second / 60);

\_second = second % 60;

}

else

\_second = second;

}

else

cout << "Error: invalid value = " << second << "!\n";

}

void Time::SetThird(int third)

{

if (third >= 0)

{

if (third >= 60)

{

SetSecond(\_second + third / 60);

\_third = third % 60;

}

else

\_third = third;

}

else

cout << "Error: invalid value = " << third << "!\n";

}

void Time::SetTriad(int first, int second, int third)

{

SetFirst(first);

SetSecond(second);

SetThird(third);

}

Код класса-родителя, передаваемый по наследству «ребёнку»:

void Triad::IncFirst()

{

\_first++;

}

void Triad::IncSecond()

{

\_second++;

}

void Triad::IncThird()

{

\_third++;

}

Triad& Triad::operator=(Triad& triad)

{

\_first = triad.\_first;

\_second = triad.\_second;

\_third = triad.\_third;

return \*this;

}

std::istream& operator>>(std::istream& input, Triad& triad)

{

int

first,

second,

third;

std::cout << "Enter the first number: ";

input >> first;

std::cout << "Enter the second number: ";

input >> second;

std::cout << "Enter the third number: ";

input >> third;

triad.SetTriad(first, second, third);

return input;

}

std::ostream& operator<<(std::ostream& output, Triad& triad)

{

output << "First = " << triad.First() << std::endl;

output << "Second = " << triad.Second() << std::endl;

output << "Third = " << triad.Third() << std::endl << std::endl;

return output;

}

Triad::Triad()

{

\_first = 0;

\_second = 0;

\_third = 0;

}

Triad::Triad(int first, int second, int third)

{

SetTriad(first, second, third);

}

Triad::Triad(Triad& copied)

{

\_first = copied.\_first;

\_second = copied.\_second;

\_third = copied.\_third;

}

Основная программа, что демонстрирует работу механизма наследования:

#include <iostream>

#include <windows.h>

#include <string>

#include "Header files/Triad.h"

#include "Header files/Time.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Triad withNoAgrumens;

Triad withArguments(43, -21, 67);

Triad copied(withArguments);

cout << withNoAgrumens << withArguments << copied << endl;

copied.IncFirst();

copied.IncSecond();

copied.IncThird();

cout << copied << endl;

Time nullTime;

Time selectedTime(23, 59, 59);

Time copiedTime(selectedTime);

cout << nullTime << selectedTime << copiedTime << endl;

selectedTime.IncThird();

selectedTime.SetSecond(65);

cout << selectedTime << endl;

Triad triad = copiedTime;

cout << endl << triad;

return 0;

}

**Полный исходный код программы на языке программирования C++:**

Triad.h

#pragma once

#include <iostream>

using namespace std;

class Triad

{

protected:

int \_first;

int \_second;

int \_third;

public:

int First() { return \_first; }

int Second() { return \_second; }

int Third() { return \_third; }

virtual void SetTriad(int first, int second, int third);

virtual void SetFirst(int first);

virtual void SetSecond(int second);

virtual void SetThird(int third);

virtual void IncFirst();

virtual void IncSecond();

virtual void IncThird();

Triad& operator=(Triad& triad);

friend istream& operator>>(istream& input, Triad& triad);

friend ostream& operator<<(ostream& output, Triad& triad);

Triad();

Triad(int first, int second, int third);

Triad(Triad& copied);

};

Triad.cpp

#include <iostream>

#include <string>

#include "../Header files/Triad.h"

void Triad::SetTriad(int first, int second, int third)

{

\_first = first;

\_second = second;

\_third = third;

}

void Triad::SetFirst(int first)

{

\_first = first;

}

void Triad::SetSecond(int second)

{

\_second = second;

}

void Triad::SetThird(int third)

{

\_third = third;

}

void Triad::IncFirst()

{

\_first++;

}

void Triad::IncSecond()

{

\_second++;

}

void Triad::IncThird()

{

\_third++;

}

Triad& Triad::operator=(Triad& triad)

{

\_first = triad.\_first;

\_second = triad.\_second;

\_third = triad.\_third;

return \*this;

}

std::istream& operator>>(std::istream& input, Triad& triad)

{

int

first,

second,

third;

std::cout << "Enter the first number: ";

input >> first;

std::cout << "Enter the second number: ";

input >> second;

std::cout << "Enter the third number: ";

input >> third;

triad.SetTriad(first, second, third);

return input;

}

std::ostream& operator<<(std::ostream& output, Triad& triad)

{

output << "First = " << triad.First() << std::endl;

output << "Second = " << triad.Second() << std::endl;

output << "Third = " << triad.Third() << std::endl << std::endl;

return output;

}

Triad::Triad()

{

\_first = 0;

\_second = 0;

\_third = 0;

}

Triad::Triad(int first, int second, int third)

{

SetTriad(first, second, third);

}

Triad::Triad(Triad& copied)

{

\_first = copied.\_first;

\_second = copied.\_second;

\_third = copied.\_third;

}

Time.h

#pragma once

#include "Triad.h"

#include <iostream>

using namespace std;

class Time : public Triad

{

public:

int Hour() { return \_first; }

int Minute() { return \_second; }

int Second() { return \_third; }

void SetFirst(int first);

void SetSecond(int second);

void SetThird(int third);

void SetTriad(int first, int second, int third);

void IncFirst();

void IncSecond();

void IncThird();

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

friend ostream& operator<<(ostream& output, Time& time);

Time();

Time(int first, int second, int third);

Time(Time& copied);

};

Time.cpp

#include "../Header files/Time.h"

#include <iostream>

void Time::SetFirst(int first)

{

\_first = first % 24;

}

void Time::SetSecond(int second)

{

if (second >= 0)

{

if (second >= 60)

{

SetFirst(\_first + second / 60);

\_second = second % 60;

}

else

\_second = second;

}

else

cout << "Error: invalid value = " << second << "!\n";

}

void Time::SetThird(int third)

{

if (third >= 0)

{

if (third >= 60)

{

SetSecond(\_second + third / 60);

\_third = third % 60;

}

else

\_third = third;

}

else

cout << "Error: invalid value = " << third << "!\n";

}

void Time::SetTriad(int first, int second, int third)

{

SetFirst(first);

SetSecond(second);

SetThird(third);

}

void Time::IncFirst()

{

SetFirst(\_first + 1);

}

void Time::IncSecond()

{

SetSecond(\_second + 1);

}

void Time::IncThird()

{

SetThird(\_third + 1);

}

Time::Time()

{

\_first = 0;

\_second = 0;

\_third = 0;

}

Time::Time(int first, int second, int third)

{

SetTriad(first, second, third);

}

Time::Time(Time& copied)

{

\_first = copied.\_first;

\_second = copied.\_second;

\_third = copied.\_third;

}

istream& operator>>(istream& input, Time& time)

{

int

first,

second,

third;

std::cout << "Enter hours: ";

input >> first;

std::cout << "Enter minutes: ";

input >> second;

std::cout << "Enter seconds: ";

input >> third;

time.SetTriad(first, second, third);

return input;

}

ostream& operator<<(ostream& output, Time& time)

{

output << "Time = " << time.Hour() << ':' << time.Minute() << ':' << time.Second() << ' ';

return output;

}

main.cpp

#include <iostream>

#include <windows.h>

#include <string>

#include "Header files/Triad.h"

#include "Header files/Time.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Triad withNoAgrumens;

Triad withArguments(43, -21, 67);

Triad copied(withArguments);

cout << withNoAgrumens << withArguments << copied << endl;

copied.IncFirst();

copied.IncSecond();

copied.IncThird();

cout << copied << endl;

Time nullTime;

Time selectedTime(23, 59, 59);

Time copiedTime(selectedTime);

cout << nullTime << selectedTime << copiedTime << endl;

selectedTime.IncThird();

selectedTime.SetSecond(65);

cout << selectedTime << endl;

Triad triad = copiedTime;

cout << endl << triad;

return 0;

}

**Ответы на вопросы:**

**Для чего используется механизм наследования?**

Для получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

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

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

**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.

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

Член класса может использоваться функциями – членами данного и производных классов и функциями – “друзьями” своего и производных классов.

**Каким образом описывается производный класс?**

При описании через двоеточие указывается спецификатор доступа для назначения его наследуемым объектам, а также название класса-родителя.

Пример:

class имяКласса : publicсписокБазовыхКлассов

{списокКомпонентовКласса};

**Наследуются ли конструкторы?**

Нет.

**Наследуются ли деструкторы?**

Нет.

**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.

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

Уничтожаются объекты в следующем порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.Таким образом, порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

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

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными.Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual. Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов. Выбор того, какую виртуальную функцию вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Для выполнения позднего связывания используются указатели на функции. Указатель на функцию — это тип указателя, который указывает на функцию вместо переменной. Функция, на которую указывает указатель, может быть вызвана через указатель и оператор вызова функции.

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

Конструкторы не могут быть виртуальными.

Деструкторы могут.

**Наследуется ли спецификатор virtual?**

Да. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию, причем спецификатор virtual может не использоваться.

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

Открытое наследование устанавливает между классами отношение «является»: класс-наследник является частью класса-родителя. Данное положение называется принципом подстановки.

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

Закрытое наследование – это наследование реализации, в этом случае принцип подстановки не соблюдается.

**В чем заключается принцип подстановки?**

Принцип звучит так: везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса.

**Имеется иерархия классов:**

**class Student**

**{**

**public:**

**int age;**

**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?**

string name;

string post;

int stage;

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

Student()

{

age=0;

name=””;

}

Employee()

{

name=””;

post=””;

}

Teacher()

{

name=””;

post=””;

stage=0;

}

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

Student(int a, string n)

{

age=a;

name=n;

}

Employee(string n, string p)

{

name=n;

post=p;

}

Teacher(string n, string p, int s)

{

name=n;

post=p;

stage=s;

}

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

Student(Student& s)

{

age=s.age;

name=s.name;

}

Employee(Employee& e)

{

name=e.name;

post=e.post;

}

Teacher(Teacher& t)

{

name=t.name;

post=t.post;

stage=t.stage;

}

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

Student&Student::operator =(Student& s)

{

this->age=s.age;

this->name=s.name;

}

Employee&Employee::operator =((Employee& e)

{

this->name=e.name;

this->post=e.post;

}

Teacher&Teacher::operator =((Teacher& t)

{

this->name=t.name;

this->post=t.post;

this->stage=t.stage;

}

Скриншот выполненной программы:

