Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

за 1 семестр

По дисциплине: «Языки программирования»

Тема: «Наследования и виртуальные функции»

Выполнил:

Студент 2 курса

Группы ПО-4 (2)

Синяк Д.А

Проверил:

Хацкевич М. В.

2020

Лабораторная работа №2

Наследование и виртуальные функции

**Цель работы:** *получить практические навыки создания иерархии классов и использования статических компонентов класса.*

**Вариант 1**

Написать программу, в которой создается иерархия классов. Включить полиморфные объекты в связанный список, используя статические компоненты класса. Показать использование виртуальных функций.

Краткие теоретические сведения.

Статические члены класса.

Такие компоненты должны быть определены в классе, как статические (static). Статические данные классов не дублируются при создании объектов, т.е. каждый статический компонент существует в единственном экземпляре. Доступ к статическому компоненту возможен только после его инициализации. Для инициализации используется конструкция

тип имя\_класса : : имя\_данного инициализатор;

Например, int complex : : count = 0;

Это предложение должно быть размещено в глобальной области после определения класса. Только при инициализации статическое данное класса получает память и становится доступным. Обращаться к статическому данному класса можно обычным образом через имя объекта

имя\_объекта.имя\_компонента

Но к статическим компонентам можно обращаться и тогда, когда

объект класса еще не существует. Доступ к статическим компонентам возможен не только через имя объекта, но и через имя класса

имя\_класса : : имя\_компонента

Однако так можно обращаться только к public компонентам.

Для обращения к private статической компоненте извне можно с помощью статических компонентов-функций. Эти функции можно вы-

звать через имя класса.

имя\_класса : : имя\_статической\_функции

Пример.

#include <iostream.h>

class TPoint

{

double x,y;

static int N; // статический компонент − данное : количество точек

public:

TPoint(double x1 = 0.0,double y1 = 0.0){N++; x = x1; y = y1;}

static int& count(){return N;} // статический компонент-функция

};

int TPoint : : N = 0; //инициализация статического компонента данного

void main(void)

{TPoint A(1.0,2.0);

TPoint B(4.0,5.0);

TPoint C(7.0,8.0);

cout<< \nОпределены ”<<TPoint : : count()<<“точки”; }

Указатель this.

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет имя this и неявно определен в каждой функции класса следующим образом:

имя\_класса \*const this = адрес\_объекта

Указатель this является дополнительным скрытым параметром каждой нестатической компонентной функции. При входе в тело принадлежащей классу функции this инициализируется значением адреса того объекта, для которого вызвана функция. В результате этого объект становится доступным внутри этой функции.

В большинстве случаев использование this является неявным. В частности, каждое обращение к нестатической функции-члену класса неявно использует this для доступа к члену соответствующего объекта.

Примером широко распространенного явного использования this являются операции со связанными списками.

Задание:

1. Определить иерархию классов (в соответствии с вариантом).

Перечень классов: студент, преподаватель, персона, завкафедрой;

2. Определить в классе статическую компоненту – указатель на начало связанного списка объектов и статическую функцию для просмотра списка (инициализировать вне определения класса, в глобальной области).

3. Реализовать классы. Определить в классах все необходимые конструкторы и деструктор.

4. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается.

5. Сделать соответствующие методы не виртуальными и посмотреть, что будет.

6. Реализовать вариант, когда объект добавляется в список при создании, т.е. в конструкторе.

1. Иерархия класса в виде графа:



1. Определение пользовательских классов:

class Person

{

protected:

char\* name;

int years\_old;

virtual void show() = 0;

public:

Person();

static Person\* head; //указатель на начало связанного списка объектов

Person\* next;

void add();

static void look\_up\_list(); //статическая функция для просмотра списка

virtual~Person();

};

class ZavKafedroi :public Person

{

protected:

int number;

public:

ZavKafedroi();

ZavKafedroi(char\* name, int years\_old, int salary);

void set(char\* name, int years\_old, int salary);

void show();

~ZavKafedroi();

};

class Prepod :public Person {

protected:

char\* subject ;

public:

Prepod();

Prepod(char\* name, int year\_old, char\* subject);

void set(char\* name, int year\_old, char\* subject);

void show();

~Prepod();

};

class Student :public Prepod

{

protected:

int number\_of\_students;

public:

Student();

Student(char\* name, int year\_old, char\* subject, int number\_of\_students);

void set(char\* name, int year\_old, char\* subject, int number\_of\_students);

void show();

~Student();

};

1. Реализация конструкторов и деструктора:

Person::Person() {

add();

}

Person::~Person() {}

ZavKafedroi::ZavKafedroi() : Person() {}

ZavKafedroi::ZavKafedroi(char\* name, int years\_old, int salary): Person() {

this->name = name;

this->years\_old = years\_old;

this->number = number;

}

ZavKafedroi::~ZavKafedroi() {}

Prepod::Prepod() : Person() {}

Prepod::Prepod(char\* name, int year\_old, char\* subject) : Print\_edition() {

this->title = name;

this->year\_of\_publishing = year\_old;

this->author = subject;

}

Prepod::~Prepod() {}

Student::Student() : Prepod() {}

Student::Student(char\* name, int year\_old, char\* subject, int number\_of\_students) : Prepod(name, year\_old, subject) {

this->title = name;

this->year\_of\_publishing = year\_old;

this->author = subject;

this->number\_of\_pages = number\_of\_students;

}

Student::~Student() {}

4) Реализация методов для добавления объекта в список:

void Person::add() {

Person \*p = this;

p->next = head;

head = p;

}

5) Реализация методов для просмотра списка:

void Student::look\_up\_list() {

Student \*p = head;

while (p) {

cout << "………………………………………………………………………………………………………………………………" << endl;

p->show(); p = p->next;

}

}

6) Листинг демонстрационной программы:

Person.h

#pragma once

#include <iostream>

using namespace std;

class Person {

protected:

char\* name;

int year\_old;

virtual void show() = 0;

public:

Person();

static Person\* head; //указатель на начало связанного списка

Person\* next;

void add();

static void look\_up\_list(); //функция для просмотра списка

};

Person realization.cpp

#include "Person.h"

Person\* Person::head = NULL;

Person::Person() {

add();

}

void Person::show() {

cout << "на месте виртуальной функции вывожу это" << endl;

}

void Person::add() {

Person\* p = this;

p->next = head;

head = p;

}

void Person::look\_up\_list() {

Person\* p = head;

cout << "Просмотр списка" << endl;

while (p) {

cout << "<!-------------------------------------------------------!>" << endl;

p->show();

p = p->next;

}

}

Person::~Person() {}

Prepod.h

#pragma once

#include "Person.h"

class Prepod : public Person {

protected:

char\* subject;

public:

Prepod();

Prepod(char\* name, int year\_old, char\* subject);

void set(char\* name, int year\_old, char\* subject);

void show();

~Prepod();

};

Prepod realization.cpp

#include "Prepod.h"

Prepod::Prepod() : Person() {}

Prepod::Prepod(char\* name, int year\_old, char\* subject) : Person() {

this->name = name;

this->year\_old = year\_old;

this->subject = subject;

}

void Prepod::set(char\* name, int year\_old, char\* subject) {

this->name = name;

this->year\_old = year\_old;

this->subject = subject;

}

void Prepod::show() {

cout << "ФИО Преподавателя " << name << endl;

cout << "Возраст" << year\_old << endl;

cout << "Предмет обучения " << subject << endl;

}

Prepod::~Prepod() {}

Student.h

#pragma once

#include "Prepod.h"

class Student : public Prepod

{

protected:

int number\_of\_student;

public:

Student();

Student(char\* name, int year\_old, char\* subject, int number\_of\_students);

void set(char\* name, int year\_old, char\* subject, int number\_of\_students);

void show();

~Student();

};

Student realization.cpp

#include "Student.h"

Student::Student(): Prepod() {}

Student::Student(char\* name, int year\_old, char\* subject, int number\_of\_students) : Prepod(name,year\_old,subject)

{

this->name= name;

this->year\_old = year\_old;

this->subject = subject;

this->number\_of\_student = number\_of\_students;

}

void Student::set(char\* name, int year\_old, char\* subject, int number\_of\_students)

{

this->name = name;

this->year\_old = year\_old;

this->subject = subject;

this->number\_of\_student = number\_of\_students;

}

void Student::show()

{

cout << "ФИО Преподавателя: " << name << endl;

cout << "Возраст: " << year\_old << endl;

cout << "Предмет обучения: " << subject << endl;

cout << "Количество обучаемых: " << number\_of\_student << endl;

}

Student::~Student() {}

ZavKafedroi.h

#pragma once

#include "Person.h"

class ZavKafedroi :public Person {

protected:

int salary;

public:

ZavKafedroi();

ZavKafedroi(char\* name, int year\_old, int salary);

void set(char\* name, int year\_old, int salary);

void show();

~ZavKafedroi();

};

ZavKafedroi.cpp

#include "ZavKafedroi.h"

ZavKafedroi::ZavKafedroi() : Person() {}

ZavKafedroi::ZavKafedroi(char\* name, int years\_old, int salary) : Person() {

this->name = name;

this->year\_old = years\_old;

this->salary = salary;

}

void ZavKafedroi::set(char\* name, int years\_old, int salary) {

this->name = name;

this->year\_old = years\_old;

this->salary = salary;

}

void ZavKafedroi::show() {

cout << "ФИО Преподавателя: " << name << endl;

cout << "Возраст: " << year\_old << endl;

cout << "Зарплата: " << salary << endl;

}

ZavKafedroi::~ZavKafedroi() {}

Main.cpp

#include"Person.h"

#include"Prepod.h"

#include"Student.h"

#include"ZavKafedroi.h"

int main()

{

setlocale(LC\_ALL, "rus");

system("color F0");

char\* name = new char[100];

cin.getline(name, 100);

ZavKafedroi object1(name, 21, 900);

ZavKafedroi p = object1;

p.add(); //объект сам добавляет себя в список

object1.show();

cout << endl;

name = new char[100];

char\* subject = new char[100];

cin.getline(name, 100);

cin.getline(subject, 100);

Prepod\* object2 = new Prepod(); //включение объекта в список при создании объекта

object2->set(name, 24, subject);

object2->show();

cout << endl;

name = new char[100];

subject = new char[100];

cin.getline(name, 100);

cin.getline(subject, 100);

Student object3 = Student();

object3.set(name, 26, subject, 121);

object3.show();

cout << endl;

Person::look\_up\_list();

delete object2;

delete[] name;

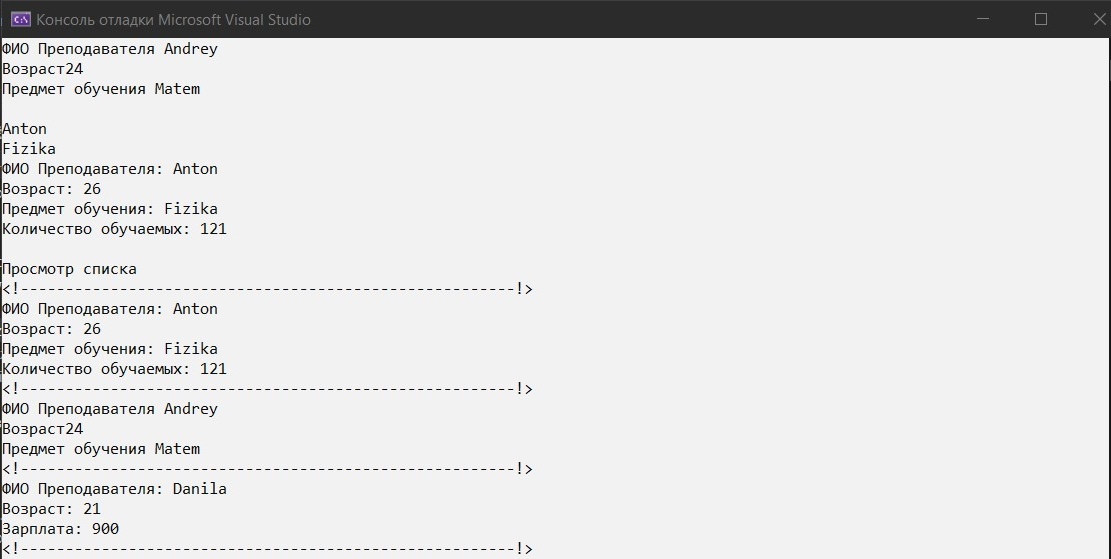
delete[] subject;

system("pause");

return 0;

}

**Результат программы:**

****

**Вывод:** *получил практические навыки создания иерархии классов и использования статических компонентов класса.*