№ 3 Наследование и полиморфизм

Определить иерархию и/или композицию классов/объектов (в соответствии с вариантом). Реализовать классы.

Написать демонстрационную программу, в которой создаются объекты различных классов. Определение классов, их реализацию, демонстрационную программу поместить в отдельные файлы.

Каждый класс должен иметь отражающее смысл название и информативный состав. Наследование должно применяться только тогда, когда это имеет смысл. При кодировании должны быть использованы соглашения об оформлении кода c++ code convention. Для хранения параметров инициализации желательно использовать файлы (например xml).

В одном из классов сделайте внутренний (nested) класс.

Далее приведен перечень классов:

Вариант 1	Классы - ПО, Текстовый процессор, Word, Вирус, Игрушка, Сапер, Разработчик.
Вариант 2	Классы – Овощи, Лук, Хлеб, Мясо, Еда, Ланч.
Вариант 3	Классы - Транспортное средство, Машина, Двигатель, Разумное существо, Человек, Трансформер;
Вариант 4	Классы – Цветок, Стебель, Роза, Хризантема, Букет.
Вариант 5	Классы – Товар, Техника, Печатающее устройство, Сканер, Компьютер, Планшет.
Вариант 6	Классы - Журнал, Книга, Печатное издание, Учебник, Персона, Автор, Издательство.
Вариант 7	Классы - Тест, Экзамен, Выпускной экзамен, Испытание, Вопрос;
Вариант 8	Классы – Телевизинная программа, Фильм, Новости, Худ. фильм, Мультфильм, Реклама, Режиссер.
Вариант 9	Классы - Продукт, Товар, Цветы, Торт, Часы, Конфеты;
Вариант 10	Классы - Квитанция, Накладная, Документ, Чек, Дата.
Вариант 11	Классы - Автомобиль, Поезд, Транспортное средство, Экспресс, Двигатель, Вагон.
Вариант 12	Классы – Инвентарь, Скамейка, Брусья, Мяч, Маты.
Вариант 13	Классы – Море, Континент, Государство, Остров, Суша;
Вариант 14	Классы - Млекопитающие, Птицы, Животное, Рыба, Лев, Сова, Тигр, Крокодил, Акула, Попугай.
Вариант 15	Классы – Транспортное средство, Корабль, Пароход, Парусник, Корвет, Капитан;
Вариант 16	Классы – Кондитерское изделие, Конфета, Карамель, Шоколадная конфета, Печенюшка.
Вариант 17	Классы – Овощ, Соус, Морковь, Лук, Картошка, Мясо, Яйцо.
Вариант 18	Классы – Камень, Драгоценный камень, Полудрагоценный камень, Рубин, Алмаз, Изумруд.
Вариант 19	Классы – Трнспорт, Грузовой самолет, Пассажирский, Военный.
Вариант 20	Классы – Тариф, Корпоративный, Индивидульный, Стандарт, Бизнес Про и т.д.
Вариант 21	Классы – Кофе, Зерновой, Молотый, Растворимы, В банках, В пакетиках.
Варинт 22	Классы – Персона, Адрес, Счет, Накопительный, Валютный, Расчетный, Общий и т.д.
Варинт 23	Классы – Фигура, Прямоугольник, Элемент управления, Кнопка, Меню, Окно.
Варинт 24	Классы – Фигура, Прямоугольник, Элемент управления,Textbox, Окно, Окно просмотра, Кнопка.
Варинт 25	Классы – Товар, Монитор, ПК, Наушники, Проектор, Рабочая станция, Экран.

Варинт 26	Классы – Мебель, Диван, Кровать, Шкаф, Шкаф-купе, Вешалка.

ПРИМЕР ВЫПОЛНЕНИЯ

main.cpp

```
#include "stdafx.h"
#include "person.h"
#include "prepod.h"
#include "student.h"
#include "zavkafedr.h"
#include "locale"
#include <iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
	setlocale(LC_ALL,"rus");

	Person a;
	a.setname("Александр");
	a.getname();

	cout<<endl;

	Prepod b;
	b.setname("Михаил Юрьевич");
	b.getname();
	b.setzarplata(1800000);
	b.getzarplata();

	cout<<endl;

	Student c;
	c.setname("Иванов Петр");
	c.getname();
	c.setstependia(101900);
	c.getstependia();

	cout<<endl;

	Zavkafedr d;
	d.setname("Гурин Владимир Иванович");
	d.getname();
	d.setzarplata(2300000);
	d.getzarplata();
	d.setkafedra("ИСиТ");
	d.getkafedra();

	cout<<endl;


	return 0;
}
```

//person.cpp

```
#include "StdAfx.h"
#include "person.h"
#include <iostream>
using namespace std;

Person::Person(void)
{
	cout<<"вызывается конструктор персоны"<<endl;
}
void Person::setname(char *name)
{
this->name=name;
}

void Person::getname()
{
	cout<<"имя персоны: "<<this->name<<endl;
}

Person::~Person(void)
{
	cout<<"вызывается деструктор персоны"<<endl;
}
```

//person.h

```
#pragma once

class Person
{

	char *name;

public:

	Person(void);

	void setname(char *name);
	void getname();

	~Person(void);
};
```

//prepod.cpp

```
#include "StdAfx.h"
#include "prepod.h"
#include <iostream>
using namespace std;

Prepod::Prepod(void)
{
	cout<<"вызывается конструктор преподавателя"<<endl;
```

```
}

void Prepod::setzarplata( int zarplata)
{
     this->zarplata=zarplata;
}

void Prepod::getzarplata()
{
     cout<<"зарплата преподавателя: "<<this->zarplata<<endl;
}
Prepod::~Prepod(void)
{
     cout<<"вызывается деструктор преподавателя"<<endl;
}
```

//prepod.h

```
#pragma once
#include "person.h"

class Prepod :
     public Person
{
     int zarplata;

public:
     Prepod(void);
     void setzarplata( int zarplata);
     void getzarplata();

     ~Prepod(void);
};
```

//student.cpp

```
#include "StdAfx.h"
#include "student.h"
#include <iostream>
using namespace std;

Student::Student(void)
{
     cout<<"вызывается конструктор студента"<<endl;
}

void Student::setstependia(int x)
{
this->stependia=x;
}

void Student::getstependia()
{
     cout<<"степендия студента: "<<this->stependia<<endl;
}
```

```
Student::~Student(void)
{
	cout<<"вызывается деструктор студента"<<endl;
}
```

//student.h

```
#pragma once
#include "person.h"

class Student :
	public Person
{
	int stependia;

public:
	Student(void);
	void setstependia (int x);
	void getstependia();
	~Student(void);
};
```

//zavkafedr.cpp

```
#include "StdAfx.h"
#include "zavkafedr.h"
#include <iostream>
using namespace std;

Zavkafedr::Zavkafedr(void)
{
	cout<<"вызывается конструктор завкафедры"<<endl;
}

void Zavkafedr::setkafedra(char *kafedra)
{
	this->kafedra=kafedra;
}

void Zavkafedr::getkafedra()
{
	cout<<"кафедра заведующего: "<<this->kafedra<<endl;
}

Zavkafedr::~Zavkafedr(void)
{
	cout<<"вызывается деструктор завкафедры"<<endl;
}
```

//zavkafedr.h

```
#pragma once
#include "prepod.h"

class Zavkafedr :
```

```
      public Prepod
{
      char *kafedra;

public:
      Zavkafedr(void);

      void setkafedra(char *kafedra);
      void getkafedra();

      ~Zavkafedr(void);
};
```

Теория

Определение производного класса.

Наследование - это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.
Существующие классы называются ***базовыми***, а новые – ***производными***. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций- членов и изменением прав доступа. Таким образом, наследование позволяет повторно использовать уже разработанный код, что повышает производительность программиста и уменьшает вероятность ошибок. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.
Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах. Сообщение, обработку которого не могут выполнить методы производного класса, автоматически передается в базовый класс. Если для обработки сообщения нужны данные, отсутствующие в производном классе, то их пытаются отыскать автоматически и незаметно для программиста в базовом классе.
При наследовании некоторые имена методов и данных базового класса могут быть по-новому определены в производном классе. В этом случае соответствующие компоненты базового класса становятся недоступными из производного класса. Для доступа к ним используется операция указания области видимости '::'.
В иерархии производный объект наследует разрешенные для наследования компоненты всех базовых объектов (*public, protected*).
Допускается множественное наследование – возможность для некоторого класса наследовать компоненты нескольких никак не связанных между

собой базовых классов. В иерархии классов соглашение относительно доступности компонентов класса следующие:

private – Член класса может использоваться только функциями-членами данного класса и функциями-"друзьями" своего класса. В производном классе он недоступен.

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.

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.

Следует иметь в виду, что объявление friend не является атрибутом доступа и не наследуется.

Синтаксис определение производного класса:

class имя_класса : список_базовых_классов
{список_компонентов_класса};

В производном классе унаследованные компоненты получают статус доступа private, если новый класс определен с помощью ключевого слова class, и статус public, если с помощью struct.

Например,

а) class S : X, Y, Z {...};

S – производный класс;

X, Y, Z – базовые классы.

Здесь все унаследованные компоненты классов X, Y, Z в классе S получают статус доступа private.

б) struct S : X, Y, Z {...};

S – производный класс;

X, Y, Z – базовые классы.

Здесь все унаследованные компоненты классов X, Y, Z в классе S получают статус доступа public.

Явно изменить умалчиваемый статус доступа при наследовании можно с помощью атрибутов доступа – private, protected и public, которые указываются непосредственно перед именами базовых классов. Как изменяются при этом атрибуты доступа в производном классе показано в следующей таблице

атрибут, указанный при наследовании	**атрибут в базовом классе**	**атрибут, полученный в производном классе**
Public	public protected private	public protected недоступен

Protected	public protected private	protected protected недоступен
Private	public protected private	private private недоступен

Пример

```
class B
{
protected:
  int t;
public:
  char u;
private:
  int x;
};
struct S : B {}; // наследуемые члены t, uимеют атрибут доступа public
class E : B {}; // t, u имеют атрибут доступа private
class M : protected B {}; // t, u – protected
class D : public B {}; // t – protected, u – public
class P : private B {}; // t, u – private
```

Таким образом, можно только сузить область доступа, но не расширить.
Таким образом, внутри производного класса существует четыре уровня, для которых определяется атрибут доступа:

- ✓ для членов базового класса;
- ✓ для членов производного класса;
- ✓ для процесса наследования;
- ✓ для изменения атрибутов при наследовании.

Рассмотрим как при этом регулируется доступ к членам класса извне класса и внутри класса.

Доступ извне.

Доступными являются лишь элементы с атрибутом public.
Собственные члены класса.
Доступ регулируется только атрибутом доступа, указанным при описании класса.
Наследуемые члены класса.
Доступ определяется атрибутами доступа базового класса, ограничивается атрибутом доступа при наследовании и изменяется явным указанием атрибута доступа в производном классе.

Пример.

```
class Basis
{
public:
  int a;
protected:
  int b, c;
};
class Derived : public Basis

{
public:
  Basis::c;
};
int main (void)
{
  Basis ob;
  Derived od;
  ob.a; // правильно
  ob.b; // ошибка
  od.c; // правильно
  od.b; // ошибка
  return 0;
}
```

Доступ изнутри

Собственные члены класса.
private и protected члены класса могут быть использованы только функциями-членами данного класса.
Наследуемые члены класса.
private-члены класса могут использоваться только собственными функциями-членами базового класса, но не функциями членами производного класса.
protected или public члены класса доступны для всех функций-членов.
Подразделение на public, protected и private относится при этом к описаниям, приведенным в базовом классе, независимо от формы наследования.
Пример.

```
     class Basis
{
public:
  void f1(int i)
  {
    a = i;
    b = i;
  }
```

```
  int b;
private:
  int a;
};

class Derived : private Basis
{
public:
  void f2(int i)
  {
    a = i; // ошибка
    b = i; // правильно
  }
};
```

Конструкторы и деструкторы производных классов

Поскольку конструкторы не наследуются, при создании производного класса наследуемые им данные-члены должны инициализироваться конструктором базового класса. Конструктор базового класса вызывается автоматически и выполняется до конструктора производного класса. Если наследуется несколько базовых классов, то их конструкторы выполняются в той последовательности, в которой перечислены базовые классы в определении производного класса. Конструктор производного класса вызывается по окончании работы конструкторов базовых классов. Параметры конструктора базового класса указываются в определении конструктора производного класса. Таким образом происходит передача аргументов от конструктора производного класса конструктору базового класса.

Пример

```
class Basis
{
public:
  Basis(int x, int y)
  {
    a = x;
    b = y;
  }
private:
  int a, b;
};

class Inherit : public Basis
{
public:
  Inherit(int x, int y, int s)
  : Basis (x, y)  {     sum = s;   }
private:
```

```
  int sum;
};
```

Запомните, что конструктор базового класса вызывается автоматически и мы указываем его в определении конструктора производного класса только для передачи ему аргументов.

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

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

Как мы знаем, объект уничтожается при завершении программы или при выходе из области действия определения объектов и эти действия выполняет деструктор. Статус деструктора по умолчанию public. Деструкторы не наследуется, поэтому даже при отсутствии в производном классе деструктора, он не передается из базового, а формируется компилятором как умалчиваемый. Классы, входящие в иерархию, должны иметь в своем распоряжении виртуальные деструкторы. Деструкторы могут переопределяться, но не перегружаться.

В любом классе могут быть в качестве компонентов определены другие классы. В этих классах могут быть свои деструкторы, которые при уничтожении объекта охватывающего (внешнего) класса выполняются после деструктора охватывающего класса. Деструкторы базовых классов выполняются в порядке, обратном перечислению классов в определении производного класса. Таким образом, порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

Пример.

```
// Определение класса базового класса ТОЧКА и производного класса
ПЯТНО.
class Point // Определение класса ТОЧКА
{
public:
        Point(int x1 = 0, int y1 = 0);
        int &getx(void);
        int &gety(void);
        void show(void);
        void move(int x1 = 0, int y1 = 0);
protected:
        int x, y;
private:
        void hide(void);
};

class Spot : public Point // Определение класса ПЯТНО
{
```

```
public:
  Spot(int, int, int);
  void show(void);
  void hide(void);
  void move(int, int);
  void change(int); // изменить размер
protected:
  int r; // радиус
  int vis; // признак видимости
  int tag; // признак сохранения видимого образа объекта в памяти
  spot *pspot; // указатель на область памяти для видимого образа
};

// Определение функций - членов класса ТОЧКА
Point::Point(int x1, int y1)
{
      x = x1;
      y = y1;
}

int &Point::getx(void)
{
      return x;
}

int &Point::gety(void)
{
      return y;
}

void Point::move(int x1, int y1)
{
      hide();
      x = x1;
      y = y1;
      show();
}


// Определение функций - членов класса ПЯТНО
Spot::Spot(int x1, int y1, int r1): Point(x1, y1)
{
      int size; // размер памяти для хранения изображения
      vis = 0;
      tag = 0;
      r = r1;
      pspot = (spot *) new char[size];
}

Spot::~Spot()
{
```

```
      hide();
      tag = 0;
      delete pspot;
}

void Spot::show(void)
{
      if (!tag)
            {
            // нарисовать и
            // запомнить изображение
            tag = 1;
            }
      else
            {
            // отобразить запомненное изображение
            }
vis = 1;
}

void Spot::hide()
{
      if (!vis) return;
      // стереть
      vis = 0;
}


// Создаются два объекта, показываются,
// затем один перемещается, а другой
// изменяет размеры
int main(void)
{
  Spot A(200, 50, 20);
  Spot B(500, 200, 30);
  A.show();
  B.show();
  A.move(50, 60);
  B.change(3);
  return 0;
}
```

В этом примере в объекте Spot точка создается как безымянный объект класса Point.

Вопросы

1. Что такое производный и базовый классы?
2. В чем заключена основная задача наследования?

3. Пусть базовый класс содержит метод `basefunc()`, а производный класс не имеет метода с таким именем. Может ли объект производного класса иметь доступ к методу `basefunc()`? Если да, то при каких условиях?
4. Напишите первую строку описания класса B, который является public-производным класса A.
5. Допустим, что базовый и производный классы включают в себя методы с одинаковыми именами. Какой из методов будет вызван объектом производного класса, если не использована операция разрешения имени?
6. Напишите объявление конструктора без аргументов для производного класса B, который будет вызывать конструктор без аргументов класса A.
7. Предположим, что существует класс D, производный от базового класса B. Напишите объявление конструктора производного класса, принимающего один аргумент и передающего его в конструктор базового класса.
8. Истинно ли следующее утверждение: класс D может быть производным класса C, который, в свою очередь, является производным класса B, производного от класса A.
9. Напишите первую строку описания класса `Petrov`, который является `public`-производным классов `Homo` и `Worker`.
10. C++ обеспечивает …, которое позволяет производному классу наследовать несколько базовых классов, даже если эти базовые классы неродственные.
11. Истинно ли утверждение о том, что указатель на базовый класс может ссылаться на объекты порожденного класса.
12. Можно ли использовать объект базового класса в производном классе в явном виде?
13. Пусть `class B : A { }`. Куда перейдет `public`-часть класса A?
14. Пусть `class B : public A { }`. Куда перейдет public-часть класса A?
15. Что такое единичное и множественное наследование?
16. Что такое полиморфизм?
17. Определите назначение виртуальных функций.
18. Определите класс с виртуальной функцией.
19. Что будет выведено на экран в результате выполнения следующего кода?

```
class ABase {
public:
    void f(int i) const { cout << 1;}
    void f(char ch) const { cout << 2; }
};
```

```
class BBase {
public:
    void f(double d) const { cout << 3;}
};

class ABBase : public ABase, public BBase {
public:
    using ABase::f;
    using BBase::f;
    void f(char ch) const { cout << 4; }
};

void g(ABBase& ab) {
    ab.f('c');
    ab.f(2.5);
    ab.f(4);
}

int main() {
    ABBase ab;
    g(ab);
}
```

20. Что будет выведено в консоль при попытке выполнить следующую программу:

```
class A {
  int j;
public:
  A(int i) : j(i) { }

  void print()
  {
    cout << sizeof(j) << endl;
  }
};

class B : virtual public A {
public:
  B(int i) : A(i) { }
};

class C : public B {
public:
  C(int i) : B(i) { }
};

int main(int argn, char * argv[])
{
  C c(1);
```

```
    c.print();
    return 0;
}
```

21. В каких из перечисленных строк произойдут ошибки компиляции:

```
class Base {
    public:
        void method1();
    protected:
        void method2();
    private:
        void method3();
};

class Child : public Base {
    protected:
        void method1() { }
        void method2() { }
        void method3() { }
};

int main() {
    Base* base = new Child();
    base->method1();          // 1
    base->method2();          // 2
    base->method3();          // 3
    return 0;
}
```

22. Что выведет следующая программа:

```
class A
{
public:
    A() { f(); }
    virtual void f()
    {
        std::cout << "A::f";
    }
};
class B : public A
{
public:
    void f()
    {
        std::cout << "B::f";
    }
};

int main(int argc, char * argv[])
```

```
{
    A * a = new B();
    delete a;
    return 0;
}
```

23. Чем можно заменить строку // 1 чтобы программа скомпилировалась?

```
class A
{
public:
  void someMethod() { /* ... */ }
};

class B : public A
{
public:
  void someMethod(int someArg) { /* ... */ }
  // 1
};

int main(int argc, char* argv[])
{
  B b;
  b.someMethod();
  return 0;
}
```

24. Что напечатает следующий код при создании экземпляра класса X:

```
class Y {
  public:
    Y() { cout << "Y"; }
};
class Z {
  public:
    Z() { cout << "Z"; }
};
class X : public Z {
  private:
    Y m_objY;
  public:
    X() { cout << "X"; }
};
```

25. В какой последовательности вызовутся деструкторы?

```
class A {
    public:
    A () {}
    ~A() { cout << "~A";}
};
```

```
class B : public A {
    public:
    B () {}
    ~B () { cout << "~B"; }
};

int main () {
    A *b = new B ();
    delete b;
    return 0;
}
```

26. Что выведет следующий код:

```
struct A { A() { cout << "A"; } };
struct B : A { B() { cout << "B"; } };
struct C : A { C() { cout << "C"; } };
struct D : B, C { D() { cout << "D"; } };

int main() {
  D d;
}
```