#### Программирование сложных объектно-ориентированных систем

# Лабораторная **работа** № 3

**Тема**: Управление доступом к компонентам класса. Одиночная спадкоемнiсть. Виртуальный функции. Виртуальные деструкторы.

**Цель**: Изучить взаимосвязь между базовыми и производными классами. Научиться пользоваться модификаторами доступа к компонентам классов.

**теоретические сведения**

**1. Модификаторы доступа**

Компоненты класса получают атрибуты доступа или по умолчанию (в зависимости от ключа класса и местонахождение объявления), или при использовании какого-либо из спецификаторов доступа: **public**, **private** или **protected**. Значение этих атрибутов следующие:

**public** Компонент может быть использован любой функцией.

**private** Компонент может быть использован только функциями-компонентами и "друзьями" класса, в котором он объявлен.

**protected** То же самое, что для private, но кроме того, компонент может быть использован функциями-компонентами и друзьями классов, производных от объявленного класса, но только в объектах производного типа. (Производные классы рассматриваются в следующем разделе).

Компоненты класса по умолчанию имеют атрибут **private**, поэтому для переопределения данного объявления спецификаторы доступа public или protected должны задаваться явно.

Компоненты **struct** по умолчанию имеют атрибут **public**, но вы можете переопределить это умолчание с помощью спецификаторов доступа **private** или **protected**.

Компоненты **union** по умолчанию имеют атрибут **public**. Переопределить его нельзя. Все три спецификаторы доступа задавать для компонентов объединения недопустимо.

Модификатор доступа по умолчанию или заданный для переопределения атрибута доступа остается действительным для всех последующих объявлений компонентов, пока не встретится другой модификатор доступа. К примеру:

     class X {

        int i; // X :: i по умолчанию private

        char ch; // также

     public:

        int j; // Следующие два компонента - public

        int k;

     protected:

        int l; // X :: l - protected

     };

     struct Y {

        int i; // Y :: i по умолчанию public

      private:

        int j; // Y :: j - private

      public:

        int k; // Y :: k - public

     };

     union Z {

        int i; // Public по умолчанию, других вариантов нет

        double d;

     };

Спецификаторы доступа могут быть перечислены и сгруппированы в любой удобной последовательности. Можно сэкономить место при наборе программы, объявив все компоненты private сразу, и т. Д.

**2 Доступ к базовым и производных классов**

Сначала дадим определение базового и производного классов. Класс, который порождает новые классы, называется базовым. Класс, который наследует из базового класса данные-члены и функции-члены называется производным классом.

При объявлении производного класса D вы перечисляете базовые классы В1, В2 ... в базовому\_списку разделяемой запятой:

ключ\_класу D: базовий\_список

{<список\_компонентив>}

Поскольку сам базовый класс может быть производным классом, то вопрос о атрибут доступа решается рекурсивно: вы отслеживаете его до тех пор, пока не доберетесь до "самого" базового класса, породивший все остальные.

D наследует все компоненты базовых классов. (Переопределены компоненты базовых классов наследуются, и при необходимости доступ к ним возможен с помощью переопределений контекста). D может использовать только компоненты базовых классов с атрибутами public и protected. О том, что будут представлять атрибуты доступа унаследованных компонентов с точки зрения D? D может понадобиться использовать public-компонент базового класса, но при этом сделать его для внешних функций private. Решение здесь заключается в том, чтобы использовать спецификаторы доступа в базовому\_списку.

При объявлении D можно задать спецификатор доступа public или private перед классами в базовому\_списку:

  class D: public B1, private B2 ... {

    ...

  }

Задать в базовому\_списке protected нельзя. Объединения не могут содержать базовых классов и не могут сами быть использованы в виде базовых классов.

Эти модификаторы не изменяют атрибутов доступа базовых членов с точки зрения базового класса, хотя и могут изменить атрибуты доступа базовых компонентов с точки зрения производных классов.

По умолчанию является private, если D является объявление класса class, и public, если D является объявление структуры struct.

Производный класс наследует атрибуты доступа базового класса следующим образом:

базовый класс public: компоненты public базового класса становятся членами public производного класса. Компоненты protected базового класса становятся компонентами protected производного класса. Компоненты private базового класса остаются для базового класса private.

базовый класс private: и public, и protected компоненты базового класса становятся private компонентами производного класса. Компоненты private базового класса остаются для базового класса private.

В обоих случаях отметим, что компоненты private базового класса были и остаются недоступными для функций-компонентов производного класса, пока в описании доступа базового класса не будут явно заданные объявления friend. К примеру,

class X: A {

// По умолчанию для класса - private A

  ....

 };

/ \* Класс X является производным от класса А \* /

class Y: B, public C {// переопределяет умолчания для С.

  ....

 };

/ \* Класс Y является производным (множественная преемственность) от B и C. По умолчанию - private B \* /

struct S: D {// по умолчанию для struct - public D

  .... / \* Struct S - производная от D \* /

 }

struct T: private D, E {// переопределяет умолчания для D.

// E по умолчанию public

  ....

 }

/ \* Struct T является производной (множественная преемственность) от D и T. По умолчанию - public E \* /

Действие спецификаторов доступа в базовом списке можно скорректировать с помощью квалифицированного имени в объявлениях public или protected для производного класса. К примеру,

class B {

int a; // По умолчанию private

public:

        int b, c;

        int Bfunc (void);

};

class X: private B {// теперь в Х a, b, c и Bfunc private

     int d; // По умолчанию private. Примечание: f в Х недостижима

  public:

     B:: c; // C была private; теперь она public

     int e;

     int Xfunc (void);

 };

int Efunc (X & x); // Внешняя По поводу В и Х

Функция Efunc может использовать только имена с атрибутом public, например c, e и Xfunc.

Функция Xfunc в X является производной от private B, поэтому она имеет доступ к:

  - "Скорректированной-к-типа-public" c

  - "Private-о-X" компонентам В: b и Bfunc

  - Собственным private и public компонентам: d, e и Xfunc.

Однако, Xfunc не имеет доступа к "private-относительно-B" компонента a.

**3 Порядок вызова конструкторов**

В случае, когда класс использует один или более базовых классов, конструкторы базовых классов запускаются до того, как будут вызваны конструкторы производного класса. Конструкторы базового класса вызываются в последовательности их объявления. В следующем примере:

      class Y {....};

      class X: public Y {....};

      X one;

вызов конструкторов происходит в следующей последовательности:

Y (); // Конструктор базового класса

X (); // Конструктор производного класса

Конструкторы элементов массива запускаются в порядке возрастания индексов массива.

Конструктор может использовать список инициализаторов, что находится к телу функции:

class X

             {

    int a, b;

  public:

    X (int i, int j): a (i), b (j) {}

 };

В этом случае инициализация X x (1, 2) присваивает значение 1 x :: a и значение 2 x :: b. Список инициализаторов обеспечивает механизм передачи значений конструкторам базового класса.

Примечание: Для обеспечения их вызова из походного класса, конструкторы базового класса должны объявляться с атрибутами public или protected.

class base1

{

int x;

public:

base1 (int i) {x = i; }

};

class base2

{

int x;

public:

base2 (int i): x (i) {}

};

class top: public base1, public base2

{

int a, b;

public:

top (int i, int j): base1 (i \* 5), base2 (j + i), a (i) {b = j; }

};

В случае такой иерархии класса объявления top one (1,2) приведет к инициализации base1 значением 5, а base2 значением 3. Способы инициализации могут комбинироваться друг с вторым (чередоваться).

Как было описано выше, базовые классы инициализируются в последовательности описания. После этого происходит инициализация элементов, также в последовательности их объявления, независимо от их взаимного расположения в списке инициализации.

class X

 {

    int a, b;

  public:

    X (int i, j): a (i), b (a + j) {}

 };

В пределах класса объявления X x (1,1) приведет к присвоению числа 1 x :: a и числа 2 x :: b.

Конструкторы базовых классов вызываются перед конструированием любых элементов производных классов. Значение производного класса не могут изменяться и после этого влиять на создание базового класса.

class base

 {

    int x;

  public:

    base (int i): x (i) {}

 };

class derived: base

 {

    int a;

  public:

    derived (int i): a (i \* 10), base (a) {} // Обратите внимание! base будет передано неинициализированных a

 };

В данном примере вызов производного d (1) не приведет к присвоению элемента базового класса х значение 10. Значение, переданное конструктору базового класса, будет неопределенным.

Если вы хотите иметь список инициализаторов в не встроенной конструкторе, не оставляйте этот список в определении класса. Вместо этого расположите его в точку определения функции:

derived :: derived (int i): a (i) {

  ....

 }

**4 Виртуальные функции**

Виртуальные функции могут быть только функциями-членами. В случае виртуальных функций вы не можете просто изменить тип функции. Итак, недопустимо переопределения виртуальной функции таким образом, чтобы она отличалась только типом возврата. Если две функции с одним i тем самым именем имеют разные аргументы, С ++ рассматривает их как разные функции, i механизм виртуальных функций игнорирует.

     Если базовый класс В содержит виртуальный функцию vf (), а класс D, является производной от класса B, содержит функцию vf () того же типа, то если функция vf () вызывается для объекта В или D, выполняется вызов D: : vf (), даже если доступ определен через указатель или посылку на B.

К примеру :

     struct B {

        virtual void vf1 ();

        virtual void vf2 ();

        virtual void vf3 ();

        void f ();

     }

     class D: public B {

        virtual void vf (); // Спецификатор virtual допустим, но лишний

        void vf2 (int) // НЕ virtual, поскольку здесь используется другой список аргументов

        char f (); // Так нельзя: изменяет только тип возврата void f ();

     };

     void main () {

        D d; // Объявление объекта D

        B \* bp = & d; // Стандартное преобразование из D \* на B \*

        bp-> vf1 (); // Вызов D :: vf1

        bp-> vf2 (); // Вызов B :: vf2, потому vf2 с D имеет отличные аргументы

        bp-> f (); // Вызов B :: f (Не виртуальной)

     }

Переопределяя функция vf1 в классе D автоматически становится виртуальной. Специфiкатор virtual может использоваться в определении переопределяя функции в походному классе, но на самом деле он является в данном случае лишним.

Интерпретация вызова виртуальной функции зависит от типа того объекта, для которого она вызывается. В случае вызова невiртуальних функций интерпретация зависит только от типа указателя или посылки, определяющие объект, для которого она вызывается.

Виртуальные функции должны быть членами некоторого класса, но они не могут быть статическими членами. Виртуальная функция может быть дружественной (friend) для другого класса.

Виртуальный функции в базовом классе, как i все функции-элементы базового класса, должны определяться или объявляться как функции без побочных эффекта ( "чистые" функции).

**5 Виртуальные деструкторы**

Деструктор может быть объявлен как виртуальный (virtual). Это позволяет указателю объекта базового класса вызвать необходимый деструктор в случае, когда указатель фактически ссылается на объект производного класса. Деструктор класса, производного от класса с виртуальным деструктором, сам является виртуальным.

class color {

  public:

     virtual ~ color (); // Виртуальный деструктор для color

 };

class red: public color {

  public:

    ~ Red (); // Деструктор для red также является виртуальным

 };

class brightred: public red {

  public:

    ~ Brightred (); // Деструктор для brightred также виртуальный

 };

Ранее перечисленные в следующих объявлениях класса:

color \* palette [3];

palette [0] = new red;

palette [1] = new brightred;

palette [2] = new color;

даст следующие результаты:

delete palette [0]; // Деструктор для red вызывается после деструктора для color

delete palette [1]; // Деструктор для brightred вызывается после ~ red и ~ color

delete palette [2]; // Запуск деструктора для color

Однако, если ни один из деструкторов не был объявлен виртуальным, delete palette [0] delete palette [1] и delete palette [2] вызывают только деструктор для класса color. Это приведет к неправильному уничтожения первых двух элементов, фактически имели тип red и brightred.

Контрольные вопросы к Л.Р. 3:

1. Какой класс называют базовым?

2. Какой класс называют производным?

3. Какие члены наследует производный класс из базового класса, а какие нет?

4. Должен конструктор производного класса вызвать конструктор базового класса, если да, то каким образом?

5. Что означает такая запись: public <базовый класс »?

6. Что значит такая запись: private <базовый класс »?

7. Что значит такая запись: protected <базовый класс »?

8. С помощью каких действий можно скорректировать действие спецификаторов доступа в базовом списке?

9. С какими атрибутами должны объявляться конструкторы базового класса чтобы обеспечить возможность их вызова из походного класса?

10. Какие функции называют виртуальными?

11. Что называют полиморфизмом?

Задание:

**1**. Создать базовый класс Сar (машина), характеризующийся торговой маркой (строка), числом цилиндров, мощностью. Визачиты методы переопределения изменения мощности. Создать производных класс Lorry (грузовик), который характеризуется также вантажопидйомом кузова. Определить функцию переназначення марки и изменения грузоподъемника.

**2**. Дано:

class base {

public: virtual void iam ()

{

cout << base \ n ";

};

Создайте два производных класса от класса base и для каждого из них определите функцию iam () так, щoб она выводила имя класса. Создайте объекты этих классов и вызовите для них функцию iam (). Используя указатель base \* на базовый класс, вызовите функции iam () из производных классов.

**3**. Разработайте класс, который способен запоминать список объектов незаданные типа класса (для любого класса).

**4**. На основе предыдущей задачи напишите программу, которая способна запоминать и отображать содержание каталога диска в виде списка строчных объектов.

Задания для самостоятельной работы

1. Создать базовый класс рабочий, затем расширить его, добавив менеджера. Функции-члены: вывод информации о рабочих и менеджеров.

2. Создать базовый класс с функциями для обработки целых или десятичных чисел. Округления для int - до десятков (34 = 30). Для double - до десятых (10,367 = 10,4). Отбросить остаток. Параметры функций: делитель, количество знаков после запятой.

3. Написать программу-диалог, которая генерирует код создания класса с унаследованными классами.

4. Базовый класс линий (координаты начала, координаты конца) определяет длину линии (стороны). Класс-наследник - прямоугольник с помощью конструктора класса линий и данных-членов (вершины начал и концов сторон) определяет площадь. Класс-наследник - пирамида определяет объем пирамиды. Данное-член - высота.

5.Створиты класс Pair (пара чисел); определить методы изменения полей и сравнения пар: пара р1 больше пары р2, если (first.p1> first.p2) или (first.p1 = first.p2) и (second.p1> second.p2). определить класс-потомок Fraction с полями: целая часть числа и дробная часть числа. Определить полный набор методов сравнения.

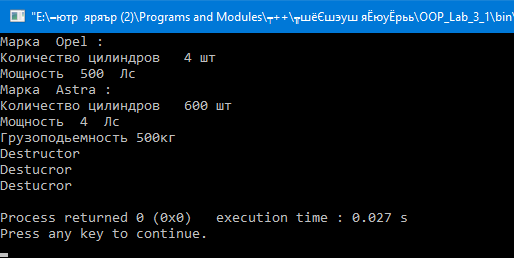
6. Создать класс Liquid (жидкость), который имеет поля названия и плотности. Определить методы переопределения и изменение плотности. Создать производный класс Alcohol (спирт), который имеет прочность. Определить методы переопределения и изменение прочности.

7. Создать класс Pair (пара чисел); определить методы изменение полей и вычисления умножения чисел. Определить производный класс Rectangle (прямоугольник) с полями-сторонами. Определить методы вычисления периметра и площади прямоугольника.

8. Создать класс Man (человек), с данными: имя, возраст, пол и вес. Определить методы переопределения имени, возраста и веса. Создать производный класс Student, который имеет дополнительное данное: год обучения. Определить методы переопределения и увеличение года обучения.

Задание:

1. Создать базовый класс Сar (машина), характеризующийся торговой маркой (строка), числом цилиндров, мощностью. Определить методы переопределения изменения мощности. Создать производных класс Lorry (грузовик), который характеризуется также грузоподьемностью кузова. Определить функцию переназначення марки и изменения грузоподъемника.



#ifndef CAR\_H

#define CAR\_H

#include <string.h>

class Car

{

public:

char \*name;

int len;

int power;

int count\_cil;

Car();

Car(const char\* a, int b, int c);

~Car();

void changed(const int);

void show();

};

#endif // CAR\_H

#include "Car.h"

#include <iostream>

Car::Car()

{

name = new char[4];

strcpy(name,"Car");

power =4;

count\_cil =500;

}

Car::Car(const char\* a, int b, int c)

{

len= strlen(a);

name = new char[len+1];

strcpy(name,a);

count\_cil= b;

power = c;

}

Car::~Car()

{

std::cout << "Destucror \n";

delete [] name;

}

void Car::changed(const int str)

{

power = str;

}

void Car::show()

{

std::cout << "Марка " << name << " :\n" ;

std::cout << "Количество цилиндров " << count\_cil << " шт\n";

std::cout << "Мощность " << power << " Л\с\n" ;

}

#ifndef LORRY\_H

#define LORRY\_H

#include <Car.h>

class Lorry : public Car

{

private:

int Gruzop;

public:

Lorry(const char\* a, int b, int c, int d);

~Lorry();

void changed\_gruz(const char \*s, int b);

void show();

};

#endif // LORRY\_H

#include "Lorry.h"

#include <iostream>

#include <cstring>

Lorry::Lorry(const char\* a, int b, int c, int d)

{

power = b;

count\_cil= c;

Gruzop = d;

len=strlen(a);

name = new char[len+1];

strcpy(name, a);

}

Lorry::~Lorry()

{

std::cout << "Destructor\n";

}

void Lorry:: changed\_gruz(const char \*s, int b)

{

Gruzop = b;

len=strlen(s);

name = new char[len+1];

strcpy(name, s);

}

void Lorry::show()

{

std::cout << "Марка " << name << " :\n" ;

std::cout << "Количество цилиндров " << count\_cil << " шт\n";

std::cout << "Мощность " << power << " Л\с\n" ;

std::cout << "Грузоподьемность " << Gruzop << "кг\n";

}

#include <iostream>

#include <Car.h>

#include <Lorry.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Car object("Opel", 4, 500);

object.show();

Lorry object\_2("Astra", 4,600,500);

object\_2.show();

return 0;

}

**2**. Дано:

class base {

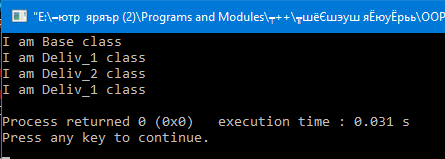
public: virtual void iam ()

{

cout << base \ n ";

};

Создайте два производных класса от класса base и для каждого из них определите функцию iam () так, щoб она выводила имя класса. Создайте объекты этих классов и вызовите для них функцию iam (). Используя указатель base \* на базовый класс, вызовите функции iam () из производных классов.



//Листинг Base.h

#ifndef BASE\_H

#define BASE\_H

#include <iostream>

class Base

{

public:

Base();

~Base();

virtual void iam(){std::cout<< "I am Base class \n";};

};

class Deliv\_1:public Base

{

public:

Deliv\_1();

~Deliv\_1();

void iam(){std::cout<< "I am Deliv\_1 class \n";};

};

class Deliv\_2:public Base

{

public:

Deliv\_2();

~Deliv\_2();

void iam(){std::cout<< "I am Deliv\_2 class \n";};

};

#endif // BASE\_H

//Листинг main.cpp

#include <iostream>

#include <Base.h>

using namespace std;

int main()

{

Base object;

object.iam();

Deliv\_1 object\_1;

object\_1.iam();

Deliv\_2 object\_2;

object\_2.iam();

Base \*base;

base = new Deliv\_1;

base->iam();

return 0;

}

3/ Разработайте класс, который способен запоминать список объектов незаданного типа класса

(для любого класса).

class Catalog

{

private: char\* data = "";

public:

Catalog();

~Catalog();

void concat(char\* mas);

char\* get();

};

Catalog::Catalog()

{

data = new char[250];

}

Catalog::~Catalog()

{

delete[] data;

}

void Catalog::concat(char\* mas)

{

strcat(data, mas);

}

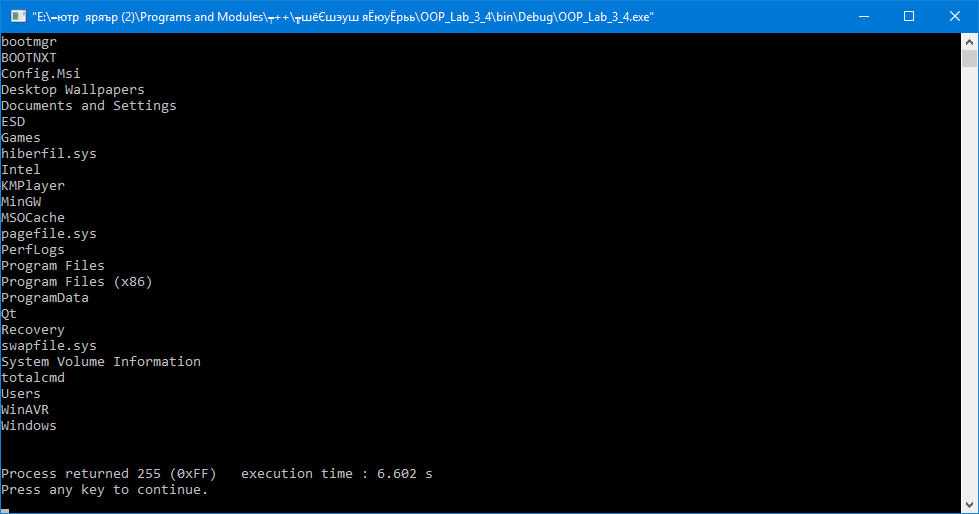
char\* Catalog:: get()

{

return data;

}

4/ На основе предыдущей задачи напишите программу, которая способна запоминать и отображать содержание каталога диска в виде списка строчных объектов.



//листинг OOP\_Lab\_3\_4.cpp

#include <iostream>

#include <windows.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus"); // локализация

cout << "Каталог C:\\" << endl;

Catalog object; // экземпляр класса

WIN32\_FIND\_DATA FindFileData; // структура информация о файле

HANDLE handle; // дескриптор потока подпрограммы

handle=FindFirstFile("c:\\\*", &FindFileData);

if (handle!=INVALID\_HANDLE\_VALUE)

{ do {

object.concat(FindFileData.cFileName);

object.concat("\n");

}

while (FindNextFile(handle,&FindFileData)!=0);

FindClose(handle);

cout << object.get() << "\n";

} else {

cout << "Error....\n";

}

return 0;

}

Контрольные вопросы к Л.Р. 3:

1. Какой класс называют базовым?

*Класс, от которого новый класс наследуется, называется: предком (англ. parent), базовым классом (англ. base class)*

1. Какой класс называют производным?

*Класс, опредёленный через наследование от другого класса, называется: производным классом, классом потомком*

1. Какие члены наследует производный класс из базового класса, а какие нет?

*Производный класс наследует из базового класса данные-члены и функции-члены, но не конструкторы и деструкторы.*

1. Должен конструктор производного класса вызвать конструктор базового класса, если да, то каким образом?

*В начале вызывается конструктор базового класса а потом производного.*

1. Что означает такая запись: public <базовый класс »?

*Производный класс наследует члены базового класса со спецификатором доступа public.*

*Все члены базового класса не изменяют уровня доступа*

1. Что значит такая запись: private <базовый класс »?

*То что все члены базового класса будут закрытыми в производном*

1. Что значит такая запись: protected <базовый класс »?

*То что открытые члены базового класса будут защищенными в производном*

1. С помощью каких действий можно скорректировать действие спецификаторов доступа в базовом списке?

*при помощи уточненного-имени в объявлениях public или protected для производного класса*

1. Какие функции называют виртуальными?

*Виртуальная функция - это функция, вызов которой зависит от типа объекта*

1. Что называют полиморфизмом?

***Полиморфизм*** *- это условия, в которых вид имеет две различные морфологические формы. Это означает, что отдельные сообщения могут вызывать различные действия на этапе выполнения программы. Конкретная форма реализации сообщения определяется и связывается с объектом во время выполнения программы.*