ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
ДИМИТРОГРАДСКИЙ ИНСТИТУТ ТЕХНОЛОГИЙ,
УПРАВЛЕНИЯ И ДИЗАЙНА

Лабораторная работа №4
по курсу: “ООП”
“*Иерархия*”
Вариант №6

Выполнил студент гр. ВТ-31: Потеренко А.Г.
Проверил преподаватель: Наскальнюк А.Н.

Димитровград 2006 г.

Содержание.

Задание для работы.

Создать класс четырехугольников, прямоугольников и квадратов. Создать из них иерархию. Определить функции вычисления площади и периметра.

Теория к заданиям.

Механизм наследования классов позволяет строить иерархии, в которых производные классы получают элементы родительских, или базовых, классов и могут дополнять их или изменять их свойства. При большом количестве никак не связанных классов управлять ими становится невозможным. Наследование позволяет справиться с этой проблемой путем упорядочивания и ранжирования классов, то есть объединения общих для нескольких классов свойств в одном классе и использования его в качестве базового.

Классы, находящиеся ближе к началу иерархии, объединяют в себе наиболее общие черты для всех нижележащих классов. По мере продвижения вниз по иерархии классы приобретают все больше конкретных черт. Множественное наследование позволяет одному классу обладать свойствами двух и более родительских классов.

Ключи доступа

Ключ доступа	Спецификатор в базовом классе	Доступ в производном классе
private	private protected public	нет private private
protected	private protected public	нет protected protected
public	private protected public	нет protected public

При описании класса в его заголовке перечисляются все классы, являющиеся для него базовыми. Возможность обращения к элементам этих классов регулируется с помощью ключей доступа private, protected, public:

```
class имя : [private | protected | public] базовый класс

 { тело класса };
```

Если базовых классов несколько, они перечисляются через запятую. Ключ доступа может стоять перед каждым классом, например:

```
class A {…};

class B {…};

class C {…};

class D: A, protected B, public C {…};
```

По умолчанию для классов используется ключ доступа private, а для структур public.

До сих пор мы рассматривали только применяемые к элементам класса спецификаторы доступа private и public. Для любого элемента класса может также использоваться спецификатор protected, который для одиночных классов, не входящих в иерархию, равносилен private. Разница между ними проявляется при наследовании, что можно видеть из приведенной таблицы:

Как видно из таблицы, private элементы базового класса в производном классе недоступны вне зависимости от ключа. Обращение к ним может осуществляться только через методы базового класса.

Элементы protected при наследовании с ключом private становятся в производном классе private, в остальных случаях права доступа к ним не изменяются.

Доступ к элементам public при наследовании становится соответствующим ключу доступа.

Если базовый класс наследуется с ключом private, можно выборочно сделать некоторые его элементы доступными в производном классе, объявив их в секции public с производного класса с помощью операции доступа к области видимости:

```
class Base
  {
   …
   public: void f();
  };
class Derived : private Base
  {
   …
   public:
     Base::void f();
  };
```

Простое наследование

Простым называется наследование, при котором производный класс имеет одного родителя. Для различных методов класса существуют разные правила наследования — например, конструкторы и операция присваивания в производном классе не наследуются, а деструкторы наследуются.

Рассмотрим правила наследования различных методов.

Конструкторы не наследуются, поэтому производный класс должен иметь собственные конструкторы. Порядок вызова конструкторов определяется приведенными ниже правилами.

? Если в конструкторе производного класса явный вызов конструктора базового класса

  отсутствует, автоматически вызывается конструктор базового класса по умолчанию (то есть

  тот, который можно вызвать без параметров).

? Для иерархии, состоящей из нескольких уровней, конструкторы базовых классов вызываются

  начиная с самого верхнего уровня. После этого выполняются конструкторы тех элементов

  класса, которые являются объектами, в порядке их объявления в классе, а затем

  исполняется конструктор класса.

? В случае нескольких базовых классов их конструкторы вызываются в порядке объявления.

ВНИМАНИЕ

Если конструктор базового класса требует указания параметров, он должен быть явным образом вызван в конструкторе производного класса в списке инициализации (это продемонстрировано в трех последних конструкторах).

? Вызов функций базового класса предпочтительнее копирования фрагментов кода из функций

базового класса в функции производного. Кроме сокращения объема кода, этим достигается упрощение модификации программы: изменения требуется вносить только в одну точку программы, что сокращает количество возможных ошибок.

Ниже перечислены правила наследования деструкторов.

? Деструкторы не наследуются, и если программист не описал в производном классе деструктор, он формируется по умолчанию и вызывает деструкторы всех базовых классов.

? В отличие от конструкторов, при написании деструктора производного класса в нем не требуется явно вызывать деструкторы базовых классов, поскольку это будет сделано автоматически.

? Для иерархии классов, состоящей из нескольких уровней, деструкторы вызываются в порядке, строго обратном вызову конструкторов: сначала вызывается деструктор класса, затем — деструкторы элементов класса, а потом деструктор базового класса.

Множественное наследование

Множественное наследование означает, что класс имеет несколько базовых классов. Если в базовых классах есть одноименные элементы, при этом может произойти конфликт идентификаторов, который устраняется с помощью операции доступа к области видимости:

```
class monstr
  {
    public: int get_health();
    …
  };
class hero
  {
    public:  int get_health();
    …
  };
class ostrich: public monstr, public hero{
…
};
int main()
  {
    ostrich A;
    cout « A.monstr::get_health();
    cout « A.hero::get_health();
}
```

Как видно из примера, для вызова метода get_health требуется явно указать класс, в котором он описан. Использование обычной для вызова метода класса конструкции

```
A.get_health();
```

приведет к ошибке, поскольку компилятор не в состоянии разобраться, к методу какого из базовых классов требуется обратиться.

Если у базовых классов есть общий предок, это приведет к тому, что производный от этих базовых класс унаследует два экземпляра полей предка, что чаще всего является нежелательным.

Чтобы избежать такой ситуации, требуется при наследовании общего предка определить его как виртуальный класс:

```
class monstr
{
…
};
class daemon: virtual public monstr
{
…
};
class lady: virtual public monstr
{
…
};
class baby: public daemon , public lady
{
…
};
```

Класс baby содержит только один экземпляр полей класса monstr. Если класс наследуется и как виртуальный, и обычным образом, в производном классе присутствовать отдельные экземпляры для каждого невиртуального вхождения и еще один экземпляр для виртуального.

Множественное наследование применяется для того, чтобы обеспечить производный класс свойствами двух или более базовых. Чаще всего один из этих классов является основным, а другие обеспечивают некоторые дополнительные свойства, поэтому они называются классами подмешивания. По возможности классы подмешивания должны быть виртуальными и создаваться с помощью конструкторов без параметров, что позволяет избежать многих проблем, возникающих при ромбовидном наследовании (когда у базовых классов есть общий предок).

Алгоритм работы программ.

Последовательно создаем 3 экземпляра класса. Класс четырехугольников имеет методы вычисления площади и периметра. Алгоритм вычисления площади четырехугольника был позаимствован из математики. При этом пользователь вводит 4 координаты на плоскости. Методы классов поочередно проверяют координаты вершин и выдают сообщение о том, какой вид четырехугольника ввел пользователь, затем выдает его характеристики. Используя иерархию, каждый класс добавляет свой метод определения вида четырехугольника, но характеристики вычисляются методами класса родителя. Код достаточно прокомментирован.

Листинги.

```
#pragma hdrstop
#pragma argsused
#include <math.h>
#include <conio.h>
#include <iostream.h>
//////////////////////////////////////////////////////////////////////////////
class CLASS_CHET
 {
   protected:
     float x1,y1,x2,y2,x3,y3,x4,y4;
   public:
     CLASS_CHET (float i1, float j1, float i2, float j2, float i3, float j3, float i4, float j4)
       {
         x1=i1;
         y1=j1;
         x2=i2;
         y2=j2;
         x3=i3;
         y3=j3;
         x4=i4;
         y4=j4;
       };
     virtual bool TOTAL_PROVERKA() //Проверка в производных класса на правильность фигуры
```

```
      {
        cout << endl;
        cout << "USER VVEL CHETIREHUGOLNIC!";
        cout << endl;
        return true; //В четырехугольнике проверять нечего
      };
     float PLOSHAD()
       {
          float S1,S2,p1,p2,a1,b1,c1,a2,b2;
          a1=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
          a2=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
          b1=sqrt((x4-x1)*(x4-x1)+(y4-y1)*(y4-y1));
          b2=sqrt((x4-x3)*(x4-x3)+(y4-y3)*(y4-y3));
          c1=sqrt((x4-x2)*(x4-x2)+(y4-y2)*(y4-y2));
          p1=(a1+b1+c1)/2;
          p2=(a2+b2+c1)/2;
          S1=sqrt(p1*(p1-a1)*(p1-b1)*(p1-c1));
          S2=sqrt(p2*(p2-a2)*(p2-b2)*(p2-c1));
          return (S1+S2);
       };
     float PERIMETR()
       {
          float P,a1,b1,a2,b2;
          a1=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
          a2=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
          b1=sqrt((x4-x1)*(x4-x1)+(y4-y1)*(y4-y1));
          b2=sqrt((x4-x3)*(x4-x3)+(y4-y3)*(y4-y3));
          P=a1+b1+a2+b2;
          return P;
       };
 };
////////////////////////////////////////////////////////////////////////////
class CLASS_PR: public CLASS_CHET
  {
    //Прямоугольник на плоскости, е если:
    // 1. Внутренние отрезки равны
    protected:
     bool PROVERKA_1() //Проверка в производных класса на правильность фигуры
        {
         float s1 = sqrt((x3-x1)*(x3-x1)+(y3-y1)*(y3-y1));
         float s2 = sqrt((x4-x2)*(x4-x2)+(y4-y2)*(y4-y2));
         if (s1==s2)
           return true;
         else
           return false;
        };
    public:
      CLASS_PR (float i1, float j1, float i2, float j2, float i3, float j3, float i4, float j4)
        : CLASS_CHET (i1,j1,i2,j2,i3,j3,i4,j4)
       {
         x1=i1;
         y1=j1;
         x2=i2;
         y2=j2;
         x3=i3;
         y3=j3;
         x4=i4;
         y4=j4;
       };
     bool TOTAL_PROVERKA() //Проверка 1 условия на правильность фигуры
      {
        if (PROVERKA_1())
         {
          cout << endl;
          cout << "USER VVEL PRAMOUGOLNIK!";
          cout << endl;
          return true;
         }
        else
          {
            return false;
          };
      };
  };
////////////////////////////////////////////////////////////////////////////
class CLASS_KV: public CLASS_PR
  {
```

```
    //Квадрат на плоскости, е если:
    // 1. Все стороны равны
    // 2. Внутренние отрезки равны
    protected:
      bool PROVERKA_2() //Проверка в производных класса на правильность фигуры
        {
         float s1 = sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
         float s2 = sqrt((x4-x3)*(x4-x3)+(y4-y3)*(y4-y3));
         float s3 = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
         float s4 = sqrt((x1-x4)*(x1-x4)+(y1-y4)*(y1-y4));
         if (s1==s2 && s2==s3 && s3==s4 && s1==s4)
                return true;
         else
           return false;
        };
    public:
     CLASS_KV (float i1, float j1, float i2, float j2, float i3, float j3, float i4, float j4)
        : CLASS_PR (i1,j1,i2,j2,i3,j3,i4,j4)
       {
         x1=i1;
         y1=j1;
         x2=i2;
         y2=j2;
         x3=i3;
         y3=j3;
         x4=i4;
         y4=j4;
       };
     bool TOTAL_PROVERKA() //Проверка 1 условия на правильность фигуры
      {
        if (PROVERKA_1() && PROVERKA_2())
         {
          cout << endl;
          cout << "USER VVEL KVADRAT!";
          cout << endl;
          return true;
         }
        else
          {
            return false;
          };
      };
  };
////////////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
   float x1,y1,x2,y2,x3,y3,x4,y4;
   cout << "BBEDITE KOORDINATI CHETIREHUGOLNIKA:" ;
   cout << endl;
   cout << "x1:";
   cout << endl;
   cin >> x1;
   cout << endl;
   cout << "y1:";
   cout << endl;
   cin >> y1;
   cout << endl;
   cout << "x2:";
   cout << endl;
   cin >> x2;
   cout << endl;
   cout << "y2:";
   cout << endl;
   cin >> y2;
   cout << endl;
   cout << "x3:";
   cout << endl;
   cin >> x3;
   cout << endl;
   cout << "y3:";
   cout << endl;
   cin >> y3;
   cout << endl;
   cout << "x4:";
   cout << endl;
   cin >> x4;
   cout << endl;
```

```
   cout << "y4:";
   cout << endl;
   cin >> y4;
   ////////////////////////////////////////////////////////////////////////////
   CLASS_KV CL1(x1,y1,x2,y2,x3,y3,x4,y4);
   CLASS_PR CL2(x1,y1,x2,y2,x3,y3,x4,y4);
   CLASS_CHET CL3(x1,y1,x2,y2,x3,y3,x4,y4);
   CLASS_CHET * UK;  //Указатель на базовый класс
   UK=&CL1;
   if (UK->TOTAL_PROVERKA())
    {
       ///////////////////////Квадрат/////////////////////////////////////////////
       cout << "PLOSHAD: " << UK->PLOSHAD();
       cout << endl;
       cout << "PERIMETR: " << UK->PERIMETR();
    }
   else
    {
      UK=&CL2;
      if (UK->TOTAL_PROVERKA())
        {
            ////////////////////////////Прямоугольник//////////////////////////////
            cout << "PLOSHAD: " << UK->PLOSHAD();
            cout << endl;
            cout << "PERIMETR: " << UK->PERIMETR();
        }
      else
        {
            ////////////////////////////Четырехугольник//////////////////////////
            UK=&CL3;
            if (UK->TOTAL_PROVERKA()) //Здесь всегда TRUE - только покажем что это за фигура
             {
              cout << "PLOSHAD: " << UK->PLOSHAD();
              cout << endl;
              cout << "PERIMETR: " << UK->PERIMETR();
             }
        };
    }
   getch();
   return 0;
}
```