程序设计实习

郭炜　微博 http://weibo.com/guoweiofpku
http://blog.sina.com.cn/u/3266490431
刘家瑛 微博 http://weibo.com/pkuliujiaying

更多多态程序实例

几何形体处理程序

几何形体处理程序：输入若干个几何形体的参数，
要求按面积排序输出。输出时要指明形状。

Input:

第一行是几何形体数目n（不超过100).下面有n行，每行以一个字母c开头.

若 c 是 ‘R’，则代表一个矩形，本行后面跟着两个整数，分别是矩形的宽和高;

若 c 是 ‘C’，则代表一个圆，本行后面跟着一个整数代表其半径

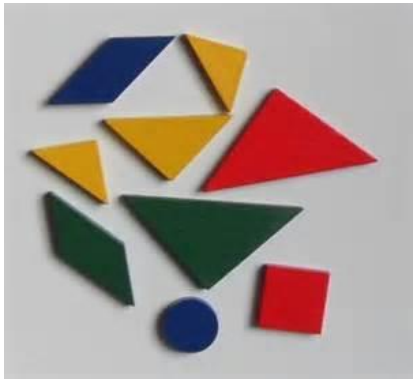
若 c 是 ‘T’，则代表一个三角形，本行后面跟着三个整数，代表三条边的长度

几何形体处理程序

Output:

按面积从小到大依次输出每个几何形体的种类及面积。每行一个几何形体，输出格式为：

形体名称：面积



几何形体处理程序

Sample Input:

3
R 3 5
C 9
T 3 4 5

Sample Output

Triangle:6
Rectangle:15
Circle:254.34

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
class CShape
{
    public:
        virtual double Area() = 0; //纯虚函数
        virtual void PrintInfo() = 0;
};
class CRectangle:public CShape
{
    public:
        int w,h;
        virtual double Area();
        virtual void PrintInfo();
};
```

```
class CCircle:public CShape  {
    public:
        int r;
        virtual double Area();
        virtual void PrintInfo();
};
class CTriangle:public CShape  {
    public:
        int a,b,c;
        virtual double Area();
        virtual void PrintInfo();
};
```

```
double CRectangle::Area()  {
        return w * h;
}
void CRectangle::PrintInfo()  {
   cout << "Rectangle:" << Area() << endl;
}
double CCircle::Area()  {
   return 3.14 * r * r ;
}
void CCircle::PrintInfo()  {
   cout << "Circle:" << Area() << endl;
}
```

```
double CTriangle::Area()  {
   double p = ( a + b + c) / 2.0;
   return sqrt(p * ( p - a)*(p- b)*(p - c));
}
void CTriangle::PrintInfo()  {
   cout << "Triangle:" << Area() << endl;
}
```

```
CShape * pShapes[100];
int MyCompare(const void * s1, const void * s2);

int main()
{
    int i; int n;
    CRectangle * pr; CCircle * pc; CTriangle * pt;
    cin >> n;
    for( i = 0;i < n;i ++ ) {
        char c;
        cin >> c;
        switch(c) {
            case 'R':
                pr = new CRectangle();
                cin >> pr->w >> pr->h;
                pShapes[i] = pr;
                break;
```

```
            case 'C':
                pc  = new CCircle();
                cin >> pc->r;
                pShapes[i] = pc;
                break;
            case 'T':
                pt = new CTriangle();
                cin >> pt->a >> pt->b >> pt->c;
                pShapes[i] = pt;
                break;
        }
    }
    qsort(pShapes,n,sizeof( CShape*),MyCompare);
    for( i = 0;i <n;i ++)
        pShapes[i]->PrintInfo();
    return 0;
}
```

```
int MyCompare(const void * s1, const void * s2)
{
    double a1,a2;
    CShape * * p1 ;  // s1,s2 是 void * ，不可写 “* s1”来取得s1指向的内容
    CShape * * p2;
    p1 = ( CShape * * ) s1; //s1,s2指向pShapes数组中的元素，数组元素的类型是CShape *
    p2 = ( CShape * * ) s2; // 故 p1,p2都是指向指针的指针，类型为 CShape **
    a1 = (*p1)->Area();  // * p1 的类型是 Cshape * ,是基类指针，故此句为多态
    a2 = (*p2)->Area();
    if( a1 < a2 )
        return -1;
    else if ( a2 < a1 )
        return 1;
    else
        return 0;
}
```

```
            case 'C':
                pc  = new CCircle();
                cin >> pc->r;
                pShapes[i] = pc;
                break;
            case 'T':
                pt = new CTriangle();
                cin >> pt->a >> pt->b >> pt->c;
                pShapes[i] = pt;
                break;
        }
    }
    qsort(pShapes,n,sizeof( CShape*),MyCompare);
    for( i = 0;i <n;i ++)
        pShapes[i]->PrintInfo();
    return 0;
}
```

如果添加新的几何形体，比如五边形，则只需要从CShape派生出CPentagon，以及在main中的switch语句中增加一个case，其余部分不变有木有！

用基类指针数组存放指向各种派生类对象的指针，然后遍历该数组，就能对各个派生类对象做各种操作，是很常用的做法

多态的又一例子

```
class Base {
public:
    void fun1()  { fun2(); }
    virtual void fun2()  {    cout << "Base::fun2()" << endl;    }
};
class Derived:public Base {
public:
        virtual void fun2() { cout << "Derived:fun2()" << endl; }
};
int main() {
                Derived d;
                Base * pBase = & d;
                pBase->fun1();
                return 0;
}
```

输出： Derived:fun2()

多态的又一例子

```
class Base {
public:
    void fun1() { this->fun2(); } //this是基类指针，fun2是虚函数，所以是多态
    virtual void fun2()  {   cout << "Base::fun2()" << endl;   }
};
class Derived:public Base {
public:
        virtual void fun2() { cout << "Derived:fun2()" << endl; }
};
int main() {
            Derived d;
            Base * pBase = & d;
            pBase->fun1();
            return 0;
}
```

输出： Derived:fun2()

在非构造函数，非析构函数的成员函数中调用虚函数，是多态!!!

构造函数和析构函数中调用虚函数

在构造函数和析构函数中调用虚函数，不是多态。编译时即可确定，调用的函数是自己的类或基类中定义的函数，不会等到运行时才决定调用自己的还是派生类的函数。

```
class myclass    {
public:
    virtual void hello(){cout<<"hello from myclass"<<endl; };
    virtual void bye(){cout<<"bye from myclass"<<endl;}
};
class son:public myclass{   public:
    void hello(){ cout<<"hello from son"<<endl;};
    son(){ hello(); };
    ~son(){ bye(); };
};
class grandson:public son{   public:
    void hello(){cout<<"hello from grandson"<<endl;};
    void bye() { cout << "bye from grandson"<<endl;}
    grandson(){cout<<"constructing grandson"<<endl;};
    ~grandson(){cout<<"destructing grandson"<<endl;};
};
```

```
int main(){
  grandson gson;
  son *pson;
  pson=&gson;
  pson->hello(); //多态
  return 0;
}
```

派生类中和基类中虚函数同名同参数表的函数，不加virtual也自动成为虚函数

结果：

```
hello from son
constructing grandson
hello from grandson
destructing grandson
bye from myclass
```