一、[25 分，每小题 5 分] 指明下列各程序中存在的不符合 C++语言规范之处（每题一处），说明你的理由或依据。解答方式可参考下面例子。

例子程序：

```
#include <iostream.h>
class   A   {
protected:
      void f( ) { cout << "A::f( )" << endl;}
};
void main( )   {
      A a;        a.f( );
}
```

对左边例子程序可以解答如下：

成员函数 f 被说明成 protected，这与 main 函数中对它的使用：a.f( )语句不符。理由是：一个类中 protected 访问权限下的成员除了可用于实现该类、实现该类的派生类或后裔类、实现友员外，不能被使用。

（1）

```
#include <iostream.h>
class   B {
public:
      virtual void f( ) const
      { cout << "B::f( )" << endl; }
      B(int int1,int int2) : j(int2), i(int1)
      {   }
private:
      int i;
      int j;
};
class   D:   public B
{
public:
      D(int anInt) : i(anInt) {   }
      void f( ) const
      {   cout << "D::f( )" << endl; }
private:
      int   i;
};
void main( )   {
      D d;
      d.f( );
}
```

(2)

```
class   A   {
public:
      A* f( )
      {
            pA = new A;
            return pA;
      }
      ~A( )
          {   if (pA)   delete pA;   }
      static A* g( )
          { return this;    }
private:
      int i;
      static A* pA;
};

A* A::pA = 0;
void main( )
{
      A a;
      A* p = a.f( );
      A* q = a.g( );
}
```

（3）

```
#include <iostream.h>
class B;
class A   {
public:
      A(int anInt=0):i(anInt)   { }
      A(const B& aB);
      int getI( ) const      {   return i; }
private:
      int i;
};
class B   {
public:
      B(int int1=0, int int2=0): i(int1),j(int2)
      {     }
      int getI( ) const      {   return i;     }
      int getJ( ) const      {     return j;   }
      operator A( )   {
            int k = (i<j)?i:j;
            return A(k);
      }
private:
      int i;
      int j;
};
A::A(const B& aB) {
      if (aB.getI( )> aB.getJ( ))
            i = aB.getI( );
      else
            i = aB.getJ( );
}
void main( ) {
      B b(10,20);
      A a(b);
      cout << a.getI( ) << endl;
}
```

（4）

```
#include <iostream.h>
class A   {
public:
      A(int anInt=0): a(anInt) { }
      void f( )
      {   cout << "A::f( )" << endl; }
      void g( )  {
            if (i>0)    a.f( );
            else    cout << "A::g( )" << endl;
      }
private:
      int i;
      A a;
};
void main( )
{
      A a1(100);
      a1.g( );
      A a2(-10);
      a2.g( );
}
```

（5）

```
#include <iostream.h>
class Complex   {
public:
      Complex(float r=0.0,float i=0.0)
            : rPart(r), iPart(i)   {      }
private:
      float rPart;      float iPart;
};
void main( )   {
      Complex c1;   Complex c2(3.14);
      Complex c3(c2);    c1 = c2;
      cout << c1 << c2 <<c3 << endl;
}
```

二、[20 分，每小题 10 分] 分别写出下列两个程序的运行结果.

（1）

```
#include <iostream.h>
class A   {
public:
      A(int anInt = 0 ):i(anInt)
         { cout << "A::A( )" << endl; }
      A(const A& anA)
         { cout << “A::A(const A&)” << endl;
           i = anA.i;    }
      int getI( ) const    {    return i;    }
      ~A( )    { cout << "A::~A( )" << endl; }
private:    int i; };
class B    {
public:
      B( ) { cout << "B::B( )" << endl; }
      B(const A& anA): a(anA)
      { cout << "B::B(constA&) " << endl;    }
      virtual void f( )
          {    cout << "B::f( )" << endl;
               cout << a.getI( ) << endl; }
      virtual ~B( ) {cout <<"B::~B( )"<<endl;}
private:    A    a;
};
class D : public B {
public:
      D( ) { cout << "D::D( )" << endl; }
      D(const A& anA): a(anA)
         { cout << "D::D(constA&)" << endl; }
      void f( )
         { B::f( );      cout << "D::f( )" << endl;
           cout << a.getI( ) << endl;    }
    ~D( ) { cout << "D::~D( )" << endl;}
private:    A    a;
};
void main( )    {
      A a(10);     B* pB = new D(a);
      pB->f( );          delete pB;    }
```

（2）

```
#include <iostream.h>
class B    {
public:
      B( ) { cout << "B::B( )" << endl; num++;}
      virtual void f( ) = 0;
      virtual void g( ) = 0;
      void k( )
      { f( ); cout << "B::k( )" << endl; g( ); }
      virtual ~B( )
      { cout << "B::~B( )" << endl;num--;}
      static int getNum( ) { return num; }
private:    static int num;
};
class D1: public B {
public:
      D1( ) { cout << "D1::D1( )" << endl; }
      void f( )   { cout << "D1::f( )" << endl;  }
      void g( )  { cout << "D1::g( )" << endl; }
      ~D1( ) { cout << "D1::~D1( )" << endl;}
};
class D2: public B {
public:
      D2( ) { cout << "D2::D2( )" << endl; }
      void f( )   { cout << "D2::f( )" << endl;  }
      void g( )  { cout << "D2::g( )" << endl; }
      ~D2( ) { cout << "D2::~D2( )" << endl; }
};
int B::num = 0;
void main( )    {
      B* buf[3];              buf[0] = new D1;
      buf[1] = new D2;    buf[2] = new D1;
      cout << B::getNum( ) << endl;
      for (int i = 0; i<3; i++)    buf[i]->k( );
      for (i = 0; i<3; i++)          delete buf[i];
      cout << B::getNum( ) << endl;
}
```

三、[20 分，每题 4 分] 判断下列的论述是否正确（回答：正确或错误），对于你认为错误的论述，说明你判断的理由（可以举反例）。

1. 假设类 D 以 public 继承方式继承了类 B，f 是子类 D 中定义的一个非静态成员函数，在包括 B 在内的任何一个 D 的祖先类中没有关于 f 的定义。那么，无论 f 如何实现，对于在 main 函数中用子类 D 说明产生的对象 d 来说，它在执行 f 时，永远不会直接或间接地访问到（它所拥有的）在类 B 中以 private 形式说明的数据成员。
2. 类 A 有一个非静态的成员函数 f，其函数原型是：void A::f( ) const，则该函数被调用时，一定是通过类 A 或类 A 的某后裔类的一个用 const 修饰符说明的常量对象调用的。
3. 类 A 有一个非静态的成员函数 f，若其函数原型是：void A::f(const int& anInt)，那么，该函数被调用时，形式参数 anInt 是实际参数的引用，不需要为它分配空间，并且在执行过程中，不会改变实际参数的数值。若把 f 的函数原型改变为：void A::f(int anInt)，则形式参数 anInt 是实际参数的一个拷贝，该函数被调用时，需要为 anInt 分配空间，并用实际参数为它初始化。如果不考虑是否需要为参数 anInt 分配空间，并且，在 f 的函数体内没有对参数 anInt 进行任何数值改变的语句，则用这两种方式定义的函数 f 的作用是一样的。
4. 若类 A 的所有构造函数、析构函数、赋值运算用 protected 访问权限进行修饰，并且，类 A 中没有任何用 static 修饰的成员函数，那么，类 A 不能产生任何对象，进而可以得出结论：定义并实现类 A 对编写一个需要产生对象的程序来说，没有任何用处，定义并实现类 A 的代码是冗余代码。
5. 对一个类来说，赋值运算是一个很基本，也很重要的操作，因此，定义一个类时，即使程序员不为它定义并实现赋值运算，编译器也会为该类自动产生一个。一个类的赋值运算可看作该类的一个成员函数。若用某个类 A 和它的两个非常量对象 a1、a2 举例来说，类 A 中用 public 访问权限修饰的赋值运算的用途主要体现在两种场合，场合一是：若 a1 和 a2 均有定义，且因类 A 有赋值运算，则可以写出代码 a1 = a2；场合二是：若 a2 有定义，且因类 A 有赋值运算，则可写出代码 A a1 = a2.

四、[10 分] “四支老鼠抬花轿，一支老鼠放鞭炮，前面两只当鼓手，咚隆咚隆真热闹，老猫来贺喜，一只一只全吃掉。”。这首儿歌描述了老鼠婚嫁的场面。完成下面两个问题：

1. [7 分] 就这段描述，找出所有可能对象，用 C++定义相应的类，用文字说明各成员数据和函数的作用，不需给出成员函数的实现。
2. [3 分] 在 main 函数中，用你定义的类产生对象，并用这些对象尽可能地模拟出儿歌中描述的场面。

五、[15 分] 已知某程序中类 MyString 是自定义的一个用来描述字符串的类，它在 main 函数中使用情况如下：

```
void main( )  {
    MyString s1("Hello ");    MyString s2("World!");
    cout << s1.getString() << s2.getString( ) << endl;
    MyString s3(s1);       cout << s3.getString( ) << endl;
    s3 = s1 + s2;              cout << s3.getString( ) << endl;   }
```

程序执行时，输出结果如下：

```
Hello World!
Hello
Hello World!
```

根据以下已知条件，完成对 MyString 类的定义和实现。

（1） 可以使用 iostream.h 中定义的类及相关内容。

（2） 可以使用 string.h 中定义的如下函数：

int strlen(const char*); 用于计算标准表示的字符串长度；

char* strcpy(char *strDestination, const char *strSource);用于将 strSource 串拷贝给 strDestination。返回的是 strDestination 串,一般不用。

char* strcat(char *strDestination, const char *strSource);用于将 strScource 串接于 strDestination 串后。返回的是 strDestination 串，一般不用。

本题目要求：根据上述已知条件，不借用其它标准函数库或类库，遵循上面程序输出结果，完成MyString类的定义和实现。

六、[10 分]在如下定义和实现的两个类 Male 和 Female 中，各自存在着指向对方对象的指针成员。且已知在运行时刻，对于任意 Male 类对象 m,若 m.pWife 的值不空，则有 m.pWife->getHusband( )==&m；同时，对任何 Felmale 类对象 f，若 f.pHusband 的值不空，有 f.pHusband->getWife( ) == &f，即：若 m 内的指针指向 f，则 f 内的指针指向 m，反之亦然。这种情况称为双向依赖。现请你在不改变这两个类已有的各成员函数的原型及功能的条件下，重新实现这两个类，使得两个类之间不存在双向依赖。允许增加类，允许增加成员函数，也允许修改成员数据。

```
class Female;
class Male{
public:
        Female* getWife( ) const
            { return pWife; }
        // 其它方法及实现，略。
private:
        Female*    pWife;     };
```

```
class Female {
public:
        Male* getHusband( ) const
            { return pHusband; }
            //其它方法及实现，略。
private:
            Male*      pHusband;
};
```