ТЕСТОВЕ

ПО

ОБЕКТНО–ОРИЕНТИРАНО ПРОГРАМИРАНЕ

Магдалина Тодорова

март-юни 2009

Тест 1

Тип псевдоним

Задача. Какъв е резултатът от изпълнението на програмата?

```
#include <iostream.h>
void func(int &a, int &b, int c)
{ int x;
  x = a;
  a = b;
  b = x;
  c = 5;
  cout << "func: a = " << a << endl;
  cout << "func: b = " << b << endl;
  cout << "func: c = " << c << endl;
  return;
}
int main()
{ int p = 1;
  int q = 2;
  int r = 3;
  cout << "main: p = " << p << endl;
  cout << "main: q = " << q << endl;
  cout << "main: r = " << r << endl;
  func(p, q, r);
  func(q, r, p);
  func(r, p, q);
  cout << "main: p = " << p << endl;
  cout << "main: q = " << q << endl;
  cout << "main: r = " << r << endl;
  return 0;
}
```

Защо ще бъдат неправилни следните обръщения към func() във функцията main(): func(3, q, r), func(4, q, 2), func(p, 3, 2)? Защо ще бъде правилно обръщението func(p, q, 5) във функцията main()?

Тест 2

Суми със събираеми – рационални числа (на базата на класа rat).

Тест 3 (за домашно)

Задача. Отстранете грешката в програмата. Намерете резултата от изпълнението на поправената програма.

```
#include <iostream.h>
class A
{ public:
      A(int a = 3);
      void f();
      void g();
      int h() const;
      void k() const;
  private:
      int n;
};
A::A(int a)
{ cout << "A(int)\n";
  n = a-5;
}
void A::f()
{ cout << "f()\n";
  n++;
}
void A::g()
{ cout << "g()\n";
  f();
  n = 2*h() + 5;
  f();
}
int A::h() const
{ cout << "h()\n";
  n--;
 return n;
}
void A::k() const
{ cout << "k()\n";
  cout << n << endl;
}
int main()
{ A a;
  A b(5);
  A c = b;
  A d = A(10);
```

```
  a.f(); b.g(); c.f(); d.g();
  d.k();
  A e(a.h()+b.h()+c.h());
  e.k();
  return 0;
}
```

Тест 4

Дефиниране на класове

Задача. Отбележете и обяснете грешките в програмата. Поправете ги, така че да се получи работеща програма. Намерете резултата от изпълнението й.

```
#include <iostream.h>
class A
{ public:
     A(int, int = 0);
     void print();
     int f_Aa();
     int f_Ab();
  private:
     int a, b;
};
A::A(int x, int y)
{ a = x;
  b = y;
}
void A::print()
{  cout << a << " " << b << endl;
}
int A::f_Aa() const
{  return a;
}
int A::f_Ab() const
{  return b;
}
class B
{ public:
      B(double = 0, A);
      void print() const;
      double f_Bb();
      A f_Ba() const;
   private:
      A a;
      double b;
};
B::B(double d, A e)
{ a = e;
  b = d;
}
```

```
void B::print() const
{  a.print();
   cout << b << endl;
}
double B::f_Bb() const
{ cout << a.f_Aa() << " " << a.f_Ab() << endl;
   return b;
}
A B::f_Ba()
{ return a;
}
int main()
{ A a(1), b;
   a.print();
   cout << b.a << " " << b.b << endl;
   B b1(2.5, a);
   b1.print();
   B b2(3.5);
   b2.print();
   return 0;
}
```

Тест 5

Канонично представяне на клас

Задача. Намерете, обяснете и поправете грешките в програмата. Какъв е резултатът от изпълнението на поправената програмата?

```
#include <iostream.h>
#include <string.h>
class first
{ public:
     first(char* ="first", int = 0);
     ~first();
     first(const first&);
     first& operator=(const first&);
    void print() const;
 private:
    char* str;
    int x;
};
first::first(char* s, int y)
{ cout  << "first(" << s << "," << y << ")\n";
  str = new char[strlen(s)+1];
  strcpy(str, s);
  x = y;
}
first::~first()
{ cout << "~first()\n";
 delete str;
}
first::first(const first& s)
{ cout << "first(const s)\n";
 delete str;
 str = new char[strlen(s.str)+1];
 strcpy(str, s.str);
 x = s.x;
}
first& first::operator=(const first& s)
{ cout << "first::operator=()\n";
 if(this != &s)
 { str = new char[strlen(s.str)+1];
   strcpy(str, s.str);
   x = s.x;
 }
```

```
  return *this;
}
void first::print() const
{  cout << str << " " << x << endl;
}
void main()
{ first a("***"), b("+++", 12), c(b);
  a.print(); b.print();
  c.print();
  c = a;
  c.print();
  first* p = new first("---", -5);
  p.print();
  delete p;
}
```

Тест 6

Канонично представяне на клас - 2

Задача. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
#include <string.h>
class first
{ public:
      first(double, int = 0);
      ~first();
      first(const first&);
      first& operator=(const first&);
      void print() const;
   private:
      double* d;
       int x;
};
first::first(double i, int j)
{ cout  << "first(" << i << "," << j << ")\n";
  d = new double(i);
  x = j;
}
first::~first()
{ cout << "~first()\n";
 delete d;
}
first::first(const first& r)
{ cout << "first(const first& r)\n";
  d = new double(*(r.d));
  x = r.x;
}
first& first::operator=(const first& r)
{ cout << "first::operator=()\n";
  if(this != &r)
  { delete d;
    d = new double(*(r.d));
    x = r.x;
   }
   return *this;
}
void first::print() const
{  cout << (*d) << " " << x << endl;
```

```
}
class second
{ public:
      second(double, const first&);
      second(const second&);
      second& operator=(const second&);
      void print() const;
  private:
      double x;
       first f;
};
second::second(double d, const first& e) : f(e)
{ cout << "second::second(d, e)\n";
  x = d;
}
second::second(const second& r) : f(r.f)
{ cout << "second::second(const second& r)\n";
  x = r.x;
}
second& second::operator=(const second& r)
{  cout << "second::operator=()\n";
   if(this != &r)
  { x = r.x;
    f = r.f;
   }
   return *this;
}
void second::print() const
{ cout << x << endl;
  f.print();
}

void main()
{ first a1(2.5, 6), a2(1);
  second b(5, a1), c(10, a2), d(c);
  b.print();
  c.print();
  d.print();
  d = b;
  d.print();
}
```

Тест 7

Даден е стек от стекове от цифри. Да се напише програма, която проверява дали съществува стек, в който с последователно четене на цифри може да се прочете дадено цяло число.

Тест 8

Единично наследяване

Задача 1. За йерархията base->der1->der2->der3:

```
class base                                  class der1 :  public base
{ private : int b1;                         { private: int d1;
  protected: int b2;                          protected: int d2;
  public: int b3();                           public: int d3();
 } b;                                        } x;
class der2 :  der1                          class der3 : protected der2
{ private: int d4;                          { private: int d7;
  protected: int d5;                          protected: int d8;
  public: int d6();                           public: int d9();
 } y;                                        } z;
```

определете възможностите за достъп на обектите: b, x, y и z до компонентите на класовете.

Задача 2. Изберете подходящи спецификатори за достъп и атрибут за област в йерархията base->der:

```
class base                                  class der: …… base
{……: int b1;                               {……: int d1;
  ……: int b2;                                ……: int d2;
  ……: int b3();                              ……: int d3();
};                                           };
```

така че фрагментите:

```
int der::d3()
{ cout << b2 << "\n";
  cout << d1 << " " << d2 << "\n";
  return b2+d2;
};
…
der d1, d2;
d1.d2 = 15;
d1.b2 = 25;
d1.d3();
```

да не предизвикват синтактични грешки, а всяка линия на фрагмента:

```
d2.b1;
d2.d1 = 22;
d2.b3();
```

да предизвиква синтактични грешки.

Приятелски функции

Задача. Разгледайте програмата:

```
#include <iostream.h>
class base
{ private:
    int b1;
  protected:
    int b2;
  public:
    friend void base_friend(base&, int, int);
    void base_read(int a, int b)
    {  b1 = a;
       b2 = b;
    }
    void base_print() const
    {  cout << "b1: " << b1 << endl
            << "b2: " << b2 << endl;
     }
 };
 class der : base
{ private:
     int d1;
  protected:
     int d2;
  public:
     friend void der_friend(der&, base, int, int);
     void readder(int a, int b, int x, int y)
     {  base_read(a, b);
        d1 = x;
        d2 = y;
      }
      void der_print() const
      {  cout << "base_print():" << endl;
         base_print();
         cout << "d1: " << d1 << endl
              << "d2: " << d2 << endl;
       }
  };
  void base_friend(base& f, int a, int b)
  {  f.b1 = a + b;
     f.b2 = a - b;
```

```
}
void der_friend(der& d, base f, int a, int b)
{  cout << "friend function der_friend(): " << endl;
   base_friend(f, a, b);
   d.d1 = f.b1 + a;
   d.d2 = f.b2 - a;
   d.b1 = a + b;
   d.b2 = f.b2 + f.b1 - 2*a;
   cout << "d.base_print(): " << endl;
   d.base_print();
   cout << "d.der_print(): " << endl;
   d.der_print();
}
void main()
{   base b;
    b.base_read(4, -2);
    b.base_print();
    base_friend(b, 2, 1);
    b.base_print();
   der d;
   d.readder(5, -1, 2, 4);
   d.der_print();
   der_friend(d, b, 7, 3);
   cout << "b.base_print()\n";
   b.base_print();
   cout << "d.der_print()\n";
   d.base_print();
}
```

1. Намерете и коментирайте грешките в нея.

2. Поправете грешките като промените единствено спецификаторите за достъп и атрибута за област на класа base.

3. Какъв е резултатът от изпълнението на поправената програма?

Предефиниране на компоненти
(за самостоятелна работа)

Задача. Какъв е резултатът от изпълнението на програмата?

```
#include <iostream.h>
class A
{ public:
     void init(int x, int y)
     {  bx = x;
        by = y;
     }
     void print() const
     {  cout << " A::bx= " << bx << endl
             << " A::by= " << by << endl;
     }
  protected:
     int bx;
  protected:
     int by;
};
class B : public A
{ public:
     void init(int x, int y, int z, int t)
     { A::init(x, y);
       bx = z;
       by = t;
     }
     void print() const
     { A::print();
       cout << " B::bx = " << bx << endl
            << " B::by = " << by << endl;
      }
   protected:
      int bx;
   private:
      int by;
};
class C :  public B
{ public:
     void init(int x, int y, int z, int t, int p, int q)
     { bx = p;
       by = q;
       A::init(x, y);
```

```
        B::init(x, y, z, t);
      }
      void print() const
      { A::print();
        B::print();
        cout << " C::bx = " << bx << endl
             << " C::by = " << by << endl;
      }
  protected:
      int bx;
  private:
      int by;
};
void main()
{ A a;
  B b;
  C c;
  a.init(-3, 2); b.init(4, 1, 9, 7);
  c.init(3, 4, 2, 1, 9, 7);
  a.print(); b.print(); c.print();
  b.A::init(-4, -1);
  b.A::print();
  b.print();
  c.B::init(3, -2, 4, -5);
  c.A::init(8, 1);
  c.print();
}
```

Тест 11

Единично наследяване. Конструктори и деструктори

Задача. Подчертайте, коментирайте и поправете грешките в програмата:

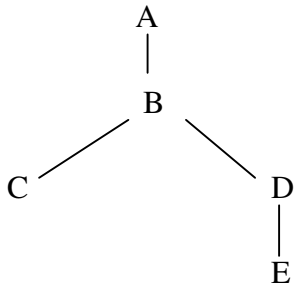
```
#include <iostream.h>
#include <string.h>
class first
{ public:
      first(char* x = "first")
      {  f = new char[strlen(x) + 1];
         strcpy(f, x);
      }
      ~first()
      { delete f;
      }
    protected:
        char* f;
};
class second1 :  public first
{public:
     second1(char* x = "second1") : first("fififi")
      {  s = new char[strlen(x) + 1];
         strcpy(s, x);
      }
      ~second1()
      {  ~first();
         delete s;
      }
      void Print()
      {  cout << "second1:: " << s << "  first:: " << f << endl;
      }
      private:
        char* s;
};
class second2 : public first
{ public:
       second2(char* x = "second2") : first();
       ~second2()
       {  delete s;
       }
       void Print()
       {  cout << "second2:: " << s << "  first:: " << f << endl;
       }
  private:
```

```
    char* s;
};
second2::second2(char* x = "second2") : first()
 {  s = new char[strlen(x) + 1];
    strcpy(s, x);
  }
class second3 : public first
{public:
     second3(char* x = "second3") : first(), first("ERROR"), second1()
      {  s = new char[strlen(x) + 1];
         strcpy(s, x);
      }
      ~second3()
      {  delete s;
         delete f;
      }
      void Print()
      {  cout << "second3:: " << s << "  first:: " << f << endl;
      }
 private:
    char* s;
};
void main()
{  second1 s11("s11");
   second2 s21("s21"), s22;
   second3 s31("s31");
   cout << "s11: "; s11.Print();
   cout << "s21: "; s21.Print();
   cout << "s22: "; s22.Print();
   cout << "s31: "; s31.Print();
}
```

Какъв е резултатът от изпълнението на поправената програма?

Единично наследяване. Конструктори
(за самостоятелна работа)

Задача. Подчертайте и коментирайте грешките в реализацията на следната йерархия:




```
#include <iostream.h>
class A
{ private:
     int y;
  public:
     void print()
     {  cout << "y: " << y << endl;
     }
};
class B : public A
{ private:
     int y;
  public:
     B(int x) : B(x = 0);
     void print()
     {  cout << "y: " << y << endl;
        cout << "A::print():" << endl;
        print();
      }
};
B::B(int x) : A(x = 5)
{  cout << "B(int)\n";
   y = x + 4;
}
class C :  public B
{ private:
     int y;
  public:
     void print()
```

```
      {  cout << "C::y: " << y << endl;
         cout << "B::print():\n";
         B::print();
       }
};
class D :  protected B
{ private:
     int y;
  public:
     D()
    { y = 0;
     }
     D(int x) : B(x = 7);
     void print()
     {  cout << "D::y: " << y << endl;
        cout << "B::print():\n";
        print();
      }
};
D::D(int x) : A(x)
{  cout << "D(int)\n";
   y = x;
}
class E : D
{ private:
      int y;
  public:
      void print()
      {  cout << "E::y: " << y << endl;
         cout << "C::print():\n";
         C::print();
       }
};
void main()
{  B b(3); b.print();
   C c; c.print();
   D d(5); d.print();
   E e; e.print();
}
```

Конструктор за присвояване, операторна функция =

Задача. Какъв е резултатът от изпълнението на програмата?

```
#include <iostream.h>
class first
{ private:
      int y;
  public:
      first()
      { y = 6;
       }
       void print()
       {  cout << "y: " << y << endl;
       }
};
class second : public first
{ private:
     int y;
  public:
      second(int x = 5);
      void print()
      {  cout << "y: " << y << endl;
         cout << "first::print():" << endl;
          first::print();
       }
};
second::second(int x)
{  cout << "second(int)\n";
   y = x + 4;
}
class third :  public second
{ private:
     int y;
  public:
      void print()
      {  cout << "third::y: " << y << endl;
         cout << "second::print():\n";
         second::print();
       }
};
class fourth :  protected second
{ private:
```

```
    int y;
  public:
    fourth()
    { y = 0;
    }
    fourth(int x) ;
    void print()
    {  cout << "fourth::y: " << y << endl;
       cout << "second::print():\n";
       second::print();
     }
};
fourth::fourth(int x) : second(x)
{  cout << "fourth(int)\n";
   y = x;
}
class fifth : fourth
{ private:
     int y;
  public:
     void print()
     {  cout << "fifth::y: " << y << endl;
        cout << "third::print():\n";
        fourth::print();
      }
};
void main()
{  second b(3); b.print();
   third c; c.print();
   fourth d(5); d.print();
   fifth e; e.print();
}
```

Множествено наследяване

Задача. Намерете резултата от изпълнението на програмата.

```
#include <iostream.h>
class base
{ public:
      base(int a = 5)
      { n = a;
        x = -1.0;
        cout << "base: " << n << ", " << x << endl;
      }
      base(const base& p)
      { n = p.n;
        x = p.x;
        cout << "base.n: " << n << endl
             << "base.x: " << x << endl;
      }
      base& operator=(const base& p)
      { if(this!=&p)
        { n = p.n + 5;
          x = p.x + 2.5;
          cout << "base.n: " << n << endl
               << "base.x: " << x << endl;
        }
        return *this;
       }
    private:
       int n;
      double x;
};
class second
{ public:
      second(double b = 2)
      { n = 9;
        y = b;
        cout << "second: " << n << ", " << y << endl;
       }
    private:
       int n;
      double y;
```

```
};
class tirth
{ public:
      tirth(double b = 1)
      { n = 3;
        x = b;
        cout << "tirth: " << n << ", " << x << endl;
       }
       tirth(const tirth& p)
      { n = p.n + 3;
        x = p.x + 1.5;
        cout << "tirth.n: " << n << endl
             << "tirth.x: " << x << endl;
       }
    private:
       int n;
      double x;
};
class fourth : base, protected tirth, public second
{ public:
      fourth(int x = 6, int y = 2, int z = 1): second(y), tirth(z), base(x)
     { n = z;
       m = x - y;
       cout << "fourth: " << n << ", " << m << endl;
      }
      fourth& operator=(const fourth& p)
     { if(this != &p)
       { base::operator =(p);
         n = p.n;
         m = p.m;
        }
        return *this;
     }
  private:
     int n, m;
};
int main()
{ fourth x, y(1, -1, 2), z;
  fourth t = x;
  z = y;
  return 0;
}
```

Преобразуване на типове

Задача. Йерархия от класове е дефинирана по следния начин:

```
#include <iostream.h>
class base
{public:
     base(int x = 0)
     {  b = x;
     }
     int get_b() const
     {  return b;
     }
     void f() const
     {  cout << "b: " << b << endl;
     }
 private:
    int b;
 };
class der1 : public base
{public:
     der1(int x = 0) : base(x)
     {  d = 1;
     }
     int get_d() const
     {  return d;
     }
     void f_der1() const
     {  cout << "class der1: d: " << d
              << " b: " << get_b() << endl;
     }
 private:
    int d;
 };
class der2 : public base
{public:
     der2(int x = 0) : base(x)
     {  d = 2;
     }
     int get_d() const
     {  return d;
     }
```

```
    void f_der2() const
    {  cout << "class der2: d: " << d
               << " b: " << get_b() << endl;
    }
private:
   int d;
};
```

Кои от следните фрагменти са коректни и кои не са? Обяснете грешките. Всеки фрагмент да се разглежда като самостоятелен.

А)

```
der1 d1;  der2 d2;
base x = d2;
d1 = x;
der1 &d3 = d1;
base &y = d3;
d3 = y;
der2 *d4 = &d2;
(*d4).f_der2();
base *z = d4;
(*z).f();
d4 = z;
```

Б)

```
base x;
der2 y = (der2)x;
base *pb = new base;
der2* pd = pb;
pb -> f();
der1 *pc = pb;
(*pc).f_der1();
cout << pc->get_b() << endl;
```

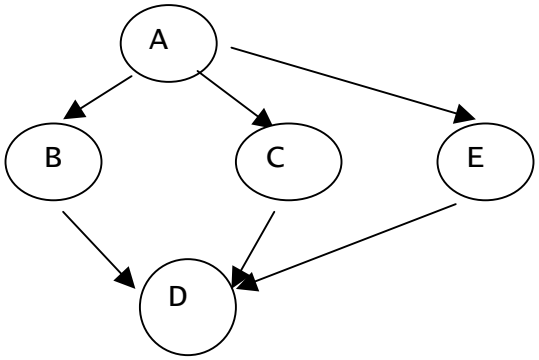
В)

```
void (der1::*pd)() = der1::f_der1;
void (base::*pb)() = pd;
void (der2::*pdd)() const;
void (base::*pbb)() const;
pbb = base::get_b;
pdd = pbb;
```

Тест 16

Виртуални класове

Задача. Да се изгради йерархията:



така, че A да е виртуален клас за класовете B и C и да не е виртуален за класа E. Класовете да съдържат голямата четворка като всеки клас да определя за член-данна символен низ, реализиран като динамичен масив.
Какво е разпределението на паметта за обект от клас D при направената реализация?

Виртуални функции. Полиморфизъм. Статично и динамично свързване

Задача. Разгледайте програмата:

```
#include <iostream.h>
class Base
{ public:
     virtual void f()
     {  cout << "f() \n";
     }
     void fgh()
     {  cout << "fgh()\n";
        f();
        g();
        h();
      }
  private:
    virtual void g()
    {  cout << "g()\n";
    }
  protected:
    virtual void h()
   {  cout << "h()\n";
   }
};
class Der : public Base
{ private:
      virtual void f()
      {  cout << "Der-class\n";
      }
  protected:
    virtual void g()
    {  cout << "Der-g()\n";
    }
  public:
     virtual void h()
    {  cout << "Der-h()\n";
    }
};
void main()
{  Base b; Der d;
```

```
    Base *p = &b;
    Base *q = &d;
    b.f();
    p->f();
    q->f();
    p->g();
    q->g();
    p->h();
    (*q).h();
    p->Base::f();
    Der *r  = new Der;
    r->f();
    r->g();
    p->fgh();
    q->fgh();
    r->fgh();
    delete r;
}
```

а) Намерете и обяснете грешките в процедурата main на горната програма.

б) Кои връзки в нея се разрешават статично и кои динамично?

в) Какъв е резултатът от изпълнението на програмата след отстраняване (задраскване) на линиите, съдържащи неправилни обръщения към виртуални член-функции?

Тест 18

Задача. Да се изгради полиморфен едносвързан списък от стекове и едносвързани списъци от цели числа. От него да се изтрие пъвият контейнер, който съдържа дадено цяло число.

Виртуални функции. Полиморфизъм. Статично и динамично свързване (за самостоятелна работа)

Задача. Разгледайте програмата:

```
#include <iostream.h>
class Base
{ public:
      virtual void virt1()
      {  cout << "Base::virt1() \n";
       }
       Base()
       {  cout << "Base()\n";
          virt1();
          virt2();
          virt3();
        }
   private:
        virtual void virt2()
        {  cout << "Base::virt2()\n";
        }
  protected:
      virtual void virt3()
     {  cout << "Base::virt3()\n";
     }
};
class Der1 : public Base
{    void virt1()
     {  cout << "Der1::virt1()\n";
     }
  protected:
      void virt2()
      {  cout << "Der1::virt2()\n";
       }
  public:
      void virt3()
       {  cout << "Der1::virt3()\n";
        }
 };
class Der2 : public Der1
{ protected:
     void virt1()
```

```
    {  cout << "Der2::virt1()\n";
    }
  public:
    void virt2()
    {  cout << "Der2::virt2()\n";
    }
  private:
     void virt3()
    {  cout << "Der2-virt3()\n";
     }
};
void main()
{  Base b;
   Der1 d1; Der2 d2;
   Base *p = &d1;
   Der1 *q = &d2;
   b.virt1();
   b.Base();
   p->virt1();
   p->virt2();
   p->virt3();
   q->Base();
   q->virt1();
   q->virt2();
   q->virt3();
   p = &d2;
   p->virt1();
   p->virt2();
   p->virt3();
   Der1 *r = new Der2;
   r->virt1();
   r->virt2();
   r->virt3();
   delete r;
}
```

а) Намерете и обяснете грешките в процедурата main на горната програма.
б) Кои връзки в нея се разрешават статично и кои динамично?
в) Какъв е резултатът от изпълнението на програмата след отстраняване (задраскване) на линиите, съдържащи неправилни обръщения към виртуални член-функции?

Виртуални класове. Виртуални деструктори

Задача. По време на изпълнение на програмата:

```
#include <iostream.h>
#include <string.h>
class A
{ public:
      A(char* = "");
      ~A();
      A(const A&);
      A& operator=(const A &);
      virtual void print() const;
  private:
     char* x;
 };
A::A(char* s)
{  x = new char[strlen(s)+1];
   strcpy(x, s);
}
A::~A()
{  cout << "~A()\n";
   delete x;
}
A::A(const A& p)
{  x = new char[strlen(p.x)+1];
   strcpy(x, p.x);
}
A& A::operator=(const A& p)
{  if (this != &p)
   {  delete x;
      x = new char[strlen(p.x)+1];
      strcpy(x, p.x);
    }
    return *this;
}
void A::print() const
{  cout << "A:: x " << x << endl;
}
class B : virtual public A
{public:
      B(char* = "", char* = "");
      ~B();
      B(const B&);
      B& operator=(const B&);
      void print() const;
 private:
     char* x;
```

```
};
B::B(char* a, char* b) : A(a)
{  x = new char[strlen(b)+1];
   strcpy(x, b);
}
B::~B()
{  cout << "~B()\n";
   delete x;
}
B::B(const B& p) : A(p)
{  x = new char[strlen(p.x)+1];
   strcpy(x, p.x);
}
B& B::operator=(const B& p)
{  if (this != &p)
   {  A::operator=(p);
      delete x;
      x = new char[strlen(p.x)+1];
      strcpy(x, p.x);
    }
   return *this;
}
void B::print() const
{  A::print();
   cout << "B:: x " << x << endl;
}
class C : virtual public B
{public:
      C(char* = "", char* = "", char* = " ");
      ~C();
      C(const C&);
      C& operator=(const C&);
      void print() const;
 private:
     char* x;
};
C::C(char* a, char* b, char* c): B(a, b)
{  x = new char[strlen(c)+1];
   strcpy(x, c);
}
C::~C()
{  cout << "~C()\n";
   delete x;
}
C::C(const C& p) : B(p)
{  x = new char[strlen(p.x)+1];
   strcpy(x, p.x);
}
C& C::operator=(const C& p)
{  if (this != &p)
   {  B::operator=(p);
```

```
        delete x;
        x = new char[strlen(p.x)+1];
        strcpy(x, p.x);
      }
    return *this;
}
void C::print() const
{  B::print();
   cout << "C:: x " << x << endl;
}
void main()
{  A *ptr1 = new B("O", "K");
   ptr1->print();
   delete ptr1;
   ptr1 = new C("M", "A", "M");
   ptr1->print();
   delete ptr1;
}
```

възниква грешка. Поправете я. Какъв е резултатът от изпълнението на поправената програма?