```
/* Classes:

Keep interfaces complete.
Keep interfaces minimal.
Provide constructors.
Support copying (or prohibit it)
Use types to provide good argument checking.
Identify nonmodifying member functions.
Free all resources in the destructor

*/

// Copying constructors
class vector
{
 public:
    vector(int s)
      :sz(s),
      elem(new double[s])
      {}
   ~vector()
   { delete[] elem; }

   void set(int i, double val)
   { elem[i] = val; }

   double get(int i)
   { return elem[i]; }

   int size() const
   { return sz; }


 private:
    int sz;
    double* elem;
};

void f(int n)
{
   vector v(3);
   v.set(2,2.2);
   vector v2 = v;
}

void main()
{
    f(3);
```

```
49 }
50
51 // including a copy constructor
52 vector(const vector& rhs):
53     sz(rhs.size()),
54     elem(new double[arg.size()]
55 {
56     for (int i=0; i<sz; ++i)
57         elem[i]=rhs.get(i);
58 }
59
60 // including a copy assignment
61 vector& operator=(const vector& rhs)
62 {
63     delete []elem;
64     sz = rhs.size();
65     elem = new double[sz];
66     for (int i=0; i<sz; ++i)
67         elem[i]=rhs.get(i);
68     return *this;
69 }
70
71
72 /*
73 what happens if you do v(10); v=v;
74 */
```