```
//  ============================================================
// FILE:  mem_demo.h
//
 class mem_demo {
    private:
       int n   ;    // Number of data items in the array, 'A'.
       int * A ;
    public:
       mem_demo() ;                                      // Constructor.
       ~mem_demo() ;                                     // Destructor.
       mem_demo( const mem_demo & B ) ;                  // Copy constructor.
       mem_demo & operator=( const mem_demo & X ) ;  // Overloaded '='
       void setup( ) ;
       int show( )   ;
} ;
```

```
//  ============================================================
// FILE:  mem_demo.cc
//
#include <iostream>
#include <cstdio>
using namespace std ;

#include "mem_demo.h"


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
mem_demo::mem_demo()    // Constructor.
{
   cout << "constructor for class mem_demo." << endl ;
   n = 0 ;
   A = NULL ;
}


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
mem_demo::mem_demo( const mem_demo & X )    // Copy Constructor.
{
   cout << "copy constructor for class mem_demo." << endl ;
   n = X.n ;
   A = new (nothrow) int [ n ] ;
   printf( "    allocated address: %lx\n", (unsigned long) A ) ;
   for ( int i = 0 ; i < n ; i++ ) {
     A[i] = X.A[i] ;
   }
   cout << "returning from copy constructor" << endl ;
}


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
mem_demo::~mem_demo()           //  Destructor
{
   cout << "destructor for class mem_demo." << endl ;
   printf( "    deleting address: %lx\n", (unsigned long) A ) ;
   delete [] A ;
   n = 0       ;
   A = NULL    ;
   cout << "returning from destructor" << endl ;
}
```

```
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
mem_demo & mem_demo::operator=( const mem_demo & X )  // Overloaded '='.
{
   cout << "overloaded '=' for class mem_demo." << endl ;
   // Guard against self-copy.
   if ( this != &X ) {
       n = X.n                              ;
       A = new (nothrow) int [ n ]          ;
       printf( "    allocated address: %lx\n", (unsigned long) A ) ;
       for ( int i = 0 ; i < n ; i++ ) {
         A[i] = X.A[i]  ;
       }
   }
   cout << "returning from overloaded '='" << endl ;
   return * this ;
}


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void mem_demo::setup( )
{
    const int N = 8 ;

    cout << "entering setup()" << endl ;
    n = N ;
    A = new (nothrow) int [ n ] ;
    printf( "    allocated address: %lx\n", (unsigned long) A ) ;
    for ( int j = 0 ; j < n ; j++ ) A[j] = ( j + 1 ) * ( j + 1 ) ;
    cout << "returning from setup()" << endl ;
}


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
int mem_demo::show( )
{
    cout << "entering show()" << endl ;
    printf( "    showing contents of address: %lx\n", (unsigned long) A ) ;
    for ( int j = 0 ; j < n ; j++ ) {
      cout << A[j] ;
      if ( (j+1) < n ) cout << " " ;
    }
    cout << endl ;
    cout << "returning from show()" << endl ;
}
```

```
//  ============================================================
// FILE:  main.cc
//

#include <iostream>
using namespace std ;

#include "mem_demo.h"


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void g( mem_demo y )
{
   mem_demo z ;
   cout << "entering g()" << endl ;
   z = y ;
   z.show()  ;
   cout << "returning from g()" << endl ;
}


// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
int main()
{

   mem_demo d ;

   cout << "entering main()" << endl ;
   d.setup( ) ;

   cout << "main() is calling g()" << endl ;
   g( d ) ;
   cout << "main() gets return from g()" << endl ;
   cout << "returning from main()" << endl ;
}
```

```
--------------------------- Sample Session ---------------------------

gottlieb% make
g++ -c main.cc
g++ -c mem_demo.cc
g++ -o main main.o mem_demo.o


gottlieb% main
constructor for class mem_demo.
entering main()
entering setup()
    allocated address: 1f60030
returning from setup()
main() is calling g()
copy constructor for class mem_demo.
    allocated address: 1f60060
returning from copy constructor
constructor for class mem_demo.
entering g()
overloaded '=' for class mem_demo.
    allocated address: 1f60090
returning from overloaded '='
entering show()
    showing contents of address: 1f60090
1 4 9 16 25 36 49 64
returning from show()
returning from g()
destructor for class mem_demo.
    deleting address: 1f60090
returning from destructor
destructor for class mem_demo.
    deleting address: 1f60060
returning from destructor
main() gets return from g()
returning from main()
destructor for class mem_demo.
    deleting address: 1f60030
returning from destructor
```