CMSC 341

C++ and OOP

Intcell.H

```
#ifndef _IntCell_H_
#define _IntCell_H_

// A class for simulating an integer memory cell.

class IntCell
 {
   public:
      explicit IntCell( int initialValue = 0 );
      Intcell(const Intcell & ic)
      ~Intcell();
      const Intcell & operator =(const Intcell & rhs);

      int read( ) const;
      void write( int x );

   private:
      int _storedValue;
};

#endif
```

IntCell.C (part 1)

```
#include "IntCell.h"

// Construct the IntCell with initialValue
IntCell::IntCell( int initialValue ) :
storedValue(initialValue)
{
      // no code
}

//copy constructor
Intcell::Intcell(const Intcell & ic)
{
      write (ic.read());
}


// destructor
Intcell::~Intcell()
{
      // no code
}
```

IntCell.C (part 2)

```
//assignment operator
Intcell::operator=(const IntCell & rhs)
{
        if (this != &rhs)
            write(rhs.read());

        return *this;

}

// Return the stored value (accessor)
 nt IntCell::read( ) const
 {
        return _storedValue;
 }



// Store x (mutator)
void IntCell::write( int x )
{
        _storedValue = x;
}
```

TestIntCell.C

```
#include <iostream.h>
#include "IntCell.h"

int main( )
{
        IntCell m;   // Or, IntCell m( 0 ); but not IntCell m( );
        Intcell n;

        n = m;
        m.write( 5 );
        cout << "Cell m contents: " << m.read( ) << endl;
        cout << "Cell n contents: " << n.read( ) << endl;

        return 0;
}
```

MemCell.H

```
#ifndef _MEMCELL_H
#define _MEMCELL_H
// A class for simulating a memory cell.
template <class Object>
class MemCell
{
   public:
      explicit MemCell(const Object &initialValue =Object( ) );
      MemCell(const MemCell & mc);

      const MemCell & operator=(const MemCell & rhs);
      ~MemCell();

      const Object & read( ) const;
      void write( const Object & x );

   private:
      Object _storedValue;
};

//Because separate compilation doesn't always work
#include "MemCell.C"
#endif
```

MemCell.C(part 1)

```
#include "MemCell.h“

// Construct the MemCell with initialValue
template <class Object>
MemCell<Object>::MemCell( const Object & initialValue )
                        : _storedValue( initialValue )
{
      // no code
}

//copy constructor
template <class Object>
MemCell<Object>::MemCell(const MemCell<Object> & mc)
{
      write(mc.read( ));
}

//assignment operator
template <class Object>
const MemCell<Object> &
      MemCell<Object>::operator=(const MemCell<Object> & rhs)
{
      write(rhs.read( ));
}
```

MemCell.C (part 2)

```
// destructor
template <class Object>
MemCell<Object>::~MemCell( )
{
      // no code
}

// Return the stored value.
template <class Object>
const Object & MemCell<Object>::read( ) const
{
      return _storedValue;
}

// Store x.
template <class Object>
void MemCell<Object>::write( const Object & x )
{
      storedValue = x;
}
```

TestMemCell.C

```
#include <iostream.h>
#include "MemCell.h"
#include "string.h"
#include <stdlib.h>

int main( )
 {
      MemCell<int>    m1;
      MemCell<string> m2( "hello" );

      m1.write( 37 );
      string str = m2.read();
      str += " world";
      m2.write(str);

      cout << m1.read( ) << endl << m2.read( ) << endl;

      return (EXIT_SUCCESS);
}
```