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 m_storedValue;
};

#endif
```

IntCell.C (part 1)

```
#include "IntCell.h"
using namespace std;

// Construct the IntCell with initialValue
IntCell::IntCell( int initialValue ) :
        m_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
const IntCell & IntCell::operator=( const IntCell & rhs )
{
        if (this != &rhs)
           Write( rhs.Read( ) );

        return *this;

}

// Return the stored value (accessor)
 int IntCell::Read(  ) const
 {
        return m_storedValue;
 }

// Store x (mutator)
void IntCell::Write( int x )
{
        m_storedValue = x;
}
```

TestIntCell.C

```
#include <iostream>
#include "IntCell.h"
 using namespace std;

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 m_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 )
                        :m_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)
{
      if (this != &rhs) Write( rhs.Read(  ) );
      return *this;
}
```

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 m_storedValue;
}

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

TestMemCell.C

```
#include <iostream>
#include <string>
#include "MemCell.h"
using namespace std;

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 ( 0 );
}
```