14.4 Copy Constructors

- Special constructor used when a newly created object is initialized using another object of the same class.
- The default copy constructor (provided by the C++ compiler) copies values of member variables to corresponding member variables.
- Default copy constructor works fine in most cases.

```cpp
Time t1;
Time t2 (t1);
Time t3 = t1;
```

IntCell declaration

- Problem: what if the object contains a pointer?

```cpp
class IntCell
{
  private:
    int *ptrToValue;  // ptr to int
  public:
    IntCell (int initialValue);
    ~IntCell();
    int read () const;
    void write (int x);
};
```

IntCell Implementation

```cpp
#include “IntCell.h”

IntCell::IntCell (int initialValue) {
  ptrToValue = new int;
  *ptrToValue = initialValue;
}

IntCell::~IntCell() {
  delete ptrToValue;
}

int IntCell::read () const {
  return *ptrToValue;
}

void IntCell::write (int x) {
  *ptrToValue = x;
}
```
Problem with member to member copying

- What we get from member to member copying in objects containing dynamic memory (ptrs):

```c
IntCell object1(5);
IntCell object2 = object1; // calls copy constructor
//object2.ptrToValue=object1.ptrToValue
object2.write(13);
cout << object1.read() << endl;
cout << object2.read() << endl;
```

What is output? 

<table>
<thead>
<tr>
<th>5</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

Problem with member to member copying

- Why are they both changed to 13?
- Member-wise copying does a shallow copy. It copies the pointer’s address instead of allocating new memory and copying the value.
- As a result, both objects point to the same location in memory

Programmer-Defined Copy Constructor

- Prototype and definition of copy constructor:

```c
IntCell(IntCell &obj);  
IntCell::IntCell(IntCell &obj) {
    ptrToValue = new int;
    *ptrToValue = obj.read();
}
```

- Copy constructor takes a reference parameter to an object of the class (see text for the reason).
- If the class does dynamic memory allocation, it needs to define its own copy constructor (i.e. all linked list implementations)

Programmer-Defined Copy Constructor

Each object now points to separate dynamic memory:

```c
IntCell object1(5);
IntCell object2 = object1; //now calls MY copy constr
object2.write(13);
cout << object1.read() << endl;
cout << object2.read() << endl;
```

Output:

<table>
<thead>
<tr>
<th>5</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>