Pointers Idioms in C++

- Initializing pointers
  - use new (or new [])
    \[
    \text{int * } x = \text{new int;}
    \]
  - use another pointer
    \[
    \text{int * } y = x;
    \]
  - use 0 (i.e., NULL)
    \[
    \text{int * } z = 0;
    \]
  - use address-of operator
    \[
    \text{int * } \text{zz} = \&i;
    \]

- Typical purposes for pointers
  - sharing objects (like Java)
  - collection of subclass instances
C++ idioms

- **What happens with the statement** `myDay = d;` ?
  - assignment is memberwise unless operator `=` overloaded
  - copy constructor used in passing parameters by value
- **If you need one of: destructor, assignment operator, copy constructor, you need all of them**
  - heuristic only: managing resources other than memory
  - preventing objects from being copied
  - what about non-copyable state, e.g., stream
- **In assignment operator, watch for self-assignment**
- **Study implementation of string/vector**
copy constructor

- **Used for “first-time” creation**
  
  ```
  Date d(1,1,2000);
  Date copy(d);
  ```

- **Used for pass-by-value**
  
  ```
  DoStuff(Date d);
  //...
  Date first(1,1,2000);
  DoStuff(first);
  ```

  ```
  // possible vector copy constructor
  template <class Item>
  vector::vector (const vector<Item>& other)
  : myLength(length),
    myList(new Item[other.myLength])
  {
    // copy elements
    for (int k = 0; k < myLength; k++)
    {
      myList[k] = other.myList[k];
    }
  }
  ```
Assignment operator

- **We want to have deep copy when assigning as well as when we copy**
  
  ```
  Object x(23, 4);
  Object y;
  y = x;           // assignment operator
  Object z = x;    // copy constructor!!!
  z = y = x;       // how does this work?
  ```

- **Assignment operator returns *this**
  - Won’t be const reference return, will be reference

- **Assignment operator checks for not assigning to self**
  - Can assign to self via aliasing, e.g., pass-by-reference

- **Assign to every data member (deep copy as needed)**

- **See tvector for details**
Destructor

- **What is purpose of destructor?**
  - Free resources
  - What’s a resource: memory, files, network connections

- **When is the destructor called?**
  - Automatically when a stack object goes out of scope
  - When you call delete on a heap object

- **What’s the problem with this “automatic destruction”?**
  - It’s not automatic, it’s fraught with problems getting it right

- **If you need copy constructor: also need assignment operator, and destructor**
  - Only needed if class contains pointer instance variables