Lecture 4
Pointers

- Declaring
  - type * name
  - type is the data type pointed to by the pointer
  - int * i, j;  // first is a pointer to an int, second is just an int
  - int * i, * j;  // both are pointers to ints
Pointers

- *p++
- *++p
- ++*p
- (*p)++
Stack vs Heap

- Stack
  - Static Memory Management
- Heap
  - Dynamic Memory Management
Stack

- **Pros**
  - Fast access
  - No need to deallocate
  - Memory efficient (usually consecutive)

- **Cons**
  - Limit on the size
  - Fixed size variables only
  - Variables have to be local
Heap

- **Pros**
  - Unlimited memory (well within limits)
  - Can be globally accessed

- **Cons**
  - Slower access
  - Potentially fragmented memory
  - Must be managed (you need to delete)

- Putting things on the heap
Templates

- Same theme as overloading
  - when we have the same function with different parameter types
- define a placeholder for the variable data type
  - most commonly T or some other short capitalized letter
Container Classes

- vector
- list
- stack
- queue
- set
- map
- many more!
Container Classes (cont)

- need to include them
  - `#include <vector>`
- (optional) remove the `std::` prefix with "using"
  - `using std::vector;`
Container Classes (cont)

- Initialization
  - `vector<int> v1; // empty vector`
  - `vector<int> v2(5); // vector with 5 spaces allocated (all zero)`
  - `vector<int> v3(v2); // copy v2`
  - `vector<int> v4(v2.begin(), v2.end()); // new vector from v2.begin() to v2.end()`
Create an iterator for the container class
- `std::vector<int>::iterator myIter; // forward iterator`
- `std::vector<int>::reverse_iterator rMyIter; // reverse iterator`

Iterate
- `for (myIter = myvector.begin() ; myIter != myvector.end(); + +myIter) {}`
- `for (rMyIter = myvector.rbegin() ; rMyIter != myvector.rend(); ++rMyIter) {}`

Getting the value
- Dereference the iterator to get/change the value
- `*myIter = 5;`