# Implementing A Priority Queue, Reading

Here's our specification for how our priority queue should work:

```
PriorityQueue<int> pq;
pq.push(100); // requires that less-than be defined
cout << pq.top( ); // returns a copy of the largest value
cout << pq.size( ); // initially zero
if (pq.empty( )) // there's no values -- size is zero
pq.pop( ); // void setter loses highest value
pq.clear( ); // setter that sets siz to zero
```

Here are some notes:

1. The constructor should set the capacity to 2 by default, and allow an int parameter for the programmer to choose the initial capacity upon declaration.
2. The "peek" implementation, PriorityQueue::top, should be a *getter* and return a **V**, *not a* **V&** because allowing its value to be changed would upset the order in the queue. It *could* return a dummy data member, but there's really no reason not to return any unused value in the values array. So either one...

Writing of the function templates is left as an exercise for the student (who has two pop implementations from which to choose).

## Private Data Members

For the arrayed implementation, there's no struct Node -- just a **V* values;** array. There's a "siz" and a "cap", and no need for a dummy.

There's no good reason to initialize the data array to **typename V**'s default value. Elements of the array are not even accessible until a value's been assigned.

## The C++ STL priority_queue Template

Here's how the **STL's priority queue (http://en.cppreference.com/w/cpp/container/priority_queue)** works -- note that it's included in the "queue" library:

```
#include <queue>
using namespace std;
...
int main()
{
  priority_queue<int> pq;
  pq.push(100); // requires that less-than be defined
  cout << pq.top( ); // returns a copy of the largest value
  cout << pq.size( ); // initially zero
  if (pq.empty( )) // there's no values -- size is zero
  pq.pop( ); // void setter loses highest value
}
```