priority_queue class

```
#ifndef PRIORITY_QUEUE_H
#define PRIORITY_QUEUE_H

#include <vector>
#include <algorithm>     // need swap
using namespace std;

namespace KW {

  /** Priority queue based on a heap stored in a vector */
  template <typename Item_Type, typename Compare = std::less<Item_Type> >
  class priority_queue {
  private:

    /** The vector to hold the data */
    vector<Item_Type> the_data;

    /** The comparator function object */
    Compare comp;

    // define tree relationships for convenience
    int parent(int i)    { return (i-1)/2;}
    int left(int i) { return 2*i + 1;}
    int right(int i) { return 2*i + 2;}

  public:

    /** Construct an empty priority queue */
    priority_queue() { }

    /** Insert an item into the priority queue */
    void push(const Item_Type& item) {
      the_data.push_back(item);
      int walk = size()-1;   // newest element
      while (walk >= 0 && comp(the_data[parent(walk)], the_data[walk])) {
        // parent is too small;  trade places
        swap(the_data[parent(walk)], the_data[walk]);
        walk = parent(walk);
      }
    }

    /** Remove the smallest item */
    void pop() {
      // move last item to root
      the_data[0] = the_data[size() - 1];
      the_data.pop_back();
      int walk = 0;
      bool possibleViolation = true;
      while (possibleViolation) {
        possibleViolation = false;
        if (left(walk) < size()) {   // we have a left child
          int maxChild = left(walk);
          if (right(walk) < size() &&
              comp(the_data[left(walk)], the_data[right(walk)]))
            maxChild = right(walk);       // right child is greater
          if (comp(the_data[walk], the_data[maxChild])) {
            // parent is smaller than a child
            swap(the_data[walk], the_data[maxChild]);
            walk = maxChild;
            possibleViolation = true;
          }
        }
      }
    }

    /** Return true if the priority queue is empty */
    bool empty() const {
      return the_data.empty();
    }

    /** Return the number of items in the priority queue */
    int size() const {
      return the_data.size();
    }

    /** Return a reference to the smallest item */
    const Item_Type& top() const {
      return the_data.front();
    }
  };   // end of priority_queue class
} // end of KW namespace

#endif
```