**Purpose:**

To store data in a vector implementation of a heap.

**Requirements:**

The data must maintain the heap property, i.e., any given node must be greater than both it’s children.

**Classes**

PriorityQueue:

Purpose: To store data as a heap.

Member data:

* data: a template vector
* heapSize: maintains the number of elements in the heap

Member methods:

* void push(T value): Adds data to the heap, while maintaining the heap property
* void pop(): removes the top from the heap
* T top(): returns the greatest element in the heap
* int left(int i): return 2i+1, the index of the left node
* int right(int i): return 2i+2, the index of the right node
* void maxHeapify(int i): restores the heap property if the heap has been modified.
  1. If the element at i is a leaf, then return.
  2. Call maxHeapify(left(i))
  3. If there is no right node, then:
     1. If the left node is greater than the current node, swap the two.
     2. Return.
  4. Call maxHeapify(right(i))
  5. If the greater of the two children is also greater than the current node, swap the current node with the greater child.
  6. Return.

**High-Level Architecture:**

The main method creates an instance of the class PriorityQueue and calls the various methods provided by it.

The push and pop functions call the maxHeapify method to maintain the heap property whenever data is added or removed.

**User Interface**

The program uses a text-based interface. The user provides no input, and the output is displayed at the command line.

**Test Cases**

The program is tested by pushing random values and ensuring that they are pushed correctly.

The special corner case is that of the empty priority queue, where an exception is thrown if pop() or top() is called.

The sample case used is:

* Call pop() while the heap is empty
* Call top() while the heap is empty
* Push 4,10,15,7,9,3,2,11 and print the heap every time a new variable is pushed
* Call pop() twice
* Push 50
* Print

Output:

