Your work for this must be finished and shown to your lab TA by the end of this session or the start of your next session.

Almost every CS lab (including this one) begins with instructions about doing essentially the same thing in some standard environment: Get the supplied files, run an initial compile, and run the program.

First, log in to your ugrad account
If you're already logged in, “cd” (with no arguments) to change to home directory

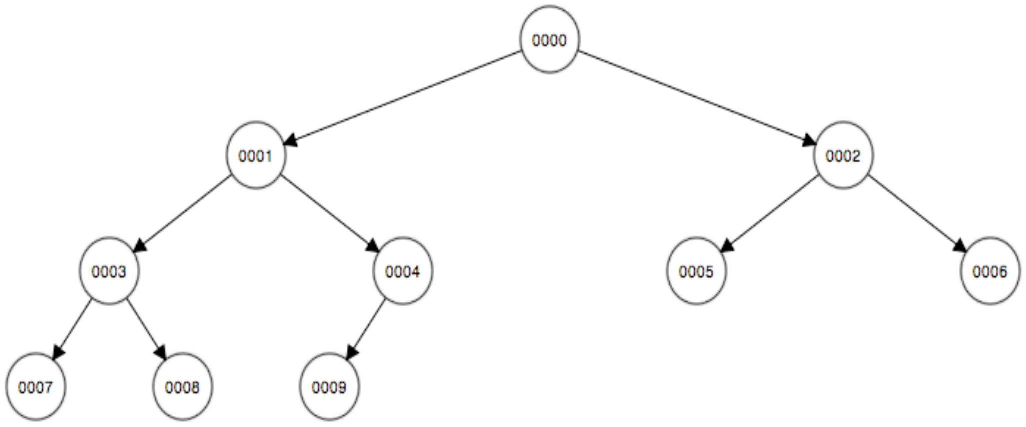
```
cd cs221
mkdir lab5
```

somehow put the contents of lab5_files.zip into cs221/lab5 here

```
cd lab5
make
./minheap
```

The middle step is up to you. You might have it on a flash drive, or you might use Filezilla or Xftp.

This lab uses another kind of binary tree called a minimum heap. (These are NOT binary search trees.) We are implementing them using an array; the “pointers” from parent-to-child (and vice-versa) are subscript values. For this implementation to work, the tree MUST always be nearly-complete: every level is full, except perhaps the bottom level which might have open slots on the right.



There's no need to memorize how to compute the subscripts of a child or parent. The diagram above represents the following array. (Note: each element's value equals its subscript; it is a minimum heap.)

0	1	2	3	4	5	6	7	8	9

Referring to the diagram, a function that returns the index of the left child given the index of the parent should be obvious. (If not, look at parent 3 with child 7, and parent 2 with child 5.)

Adding 1 to this result gives the index of the right child.

The inverse of these functions finds the parent, given the subscript of a child.
Hint: use integer division (drop the remainder).

The “Heap Visualizer” found at https://www.cs.usfca.edu/~galles/visualization/Heap.html can be used to build your own examples of minimum heaps.

Your work for this must be finished and shown to your lab TA by the end of this session or the start of your next session.

If you haven't already, compile and run minheap. You'll see there are some unit tests that failed. Completing the code in Minheap.cc correctly will fix these. (Only Minheap.cc needs changing in this lab.) The Object Oriented code is a little different than the implementation you've seen in lectures, because the physical array and the size of the current heap are data members of the object.

Most of the methods are completed and working. Before you start coding, look at Minheap.h to get an overview of what is in the Class, so you can see what you have to work with.

Q1. Complete the print_tree() method, so that the minheap in `0 2 1 4 3 9 5 7 6 8`
Prints as:

```
    5
  1
    9
0
    3
      8
  2
      6
    4
      7
```

This is just another tree traversal, with a line printed on each "visit". But it's not quite the same as any of the ones in the BST lab – very close to one of them but slightly different. This won't get rid of any of the errors, but you can use it to print the tree if you have trouble any of the rest of the methods.

Q2. Complete the insert() method.

Q3. Complete the delete_min() method.

Q4. Complete the remove_matching() method.

And that's all.