COMP 322 - Assignment 1 - Winter 2016

Due: Feb 5th, 2016 at 11:30PM

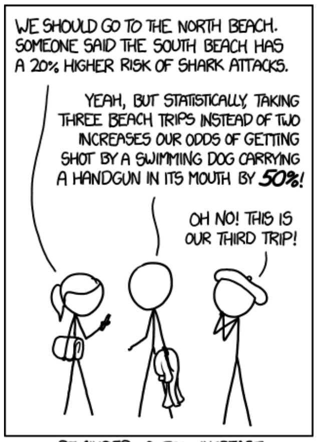
Introduction

The Sorites Paradox is a term used to refer to a number of flawed dialectical arguments that are based on the vagueness of spoken language. For example, based on the following two assumptions,

- One grain of salt is not a heap.
- Adding a single grain of salt to something that is not a heap will not result in a heap.

one could conclude that "No matter how much sand you add you will never have a heap." See picture on the right for a boffo example of this paradox and a good comeback to anybody using such arguments. Watch out for that dog! Save the sharks!

In this first assignment we will start a nice exploration into the beautiful world of heaps. Not sand heaps, as in the argument above; not snow heaps, as in your backyard; we will look at *the heap data structure*, which is used to implement the abstract data type know as a *priority queue*. If you have not seen the latter already in a previous course, a priority queue is exactly what it sounds like: Romanian health care! But before we get to that, let's have some fun with other type of arguments. Function arguments, that is.


WE SHOULD GO TO THE NORTH BEACH. SOMEONE SAID THE SOUTH BEACH HAS A 20% HIGHER RISK OF SHARK ATTACKS.
YEAH, BUT STATISTICALLY, TAKING THREE BEACH TRIPS INSTEAD OF TWO INCREASES OUR ODDS OF GETTING SHOT BY A SWIMMING DOG CARRYING A HANDGUN IN ITS MOUTH BY 50%!
OH NO! THIS IS OUR THIRD TRIP!

REMINDER: A 50% INCREASE IN A TINY RISK IS *STILL TINY.*

Before you start working on the assignment

- Feel free to submit any solution for the 0-credit question. Do not worry about style or commenting, unless you want the instructor or the TAs to help you with it.
- Collaboration is strongly encouraged for all questions, but make sure your graded submission reflects individual work. The Discussion Board, me and the TAs are there for you to use as well. I expect a heap of students for my Office Hours!
- It might happen that some topics in this homework won't be covered in class. This is normal as we only have one hour of lecture per week. You should not have trouble finding out answers in C++ online tutorials, and as always, don't forget to e-mail us or discuss with other students.
- We provide a `Makefile` and a header file `heaps.h`. Your submission should compile using the `make` command on Trottier (3rd floor) machines when run on the **original** `Makefile`, **original** `heaps.h` and your submission `heapsStudent.cpp`. ONLY submit the `.cpp` file.

Heaps

Before we go into the mathematical properties of heaps, we will fist look at the structure of a heap (see Figure 1). This data structure has its elements organized in a complete binary tree:

- each data item is a node in this tree.
- each node has at most 2 children.
- the tree is fully balanced, i.e. the heap has elements at level k only if the higher level $k-1$ has the most elements possible, 2^{k-1}.

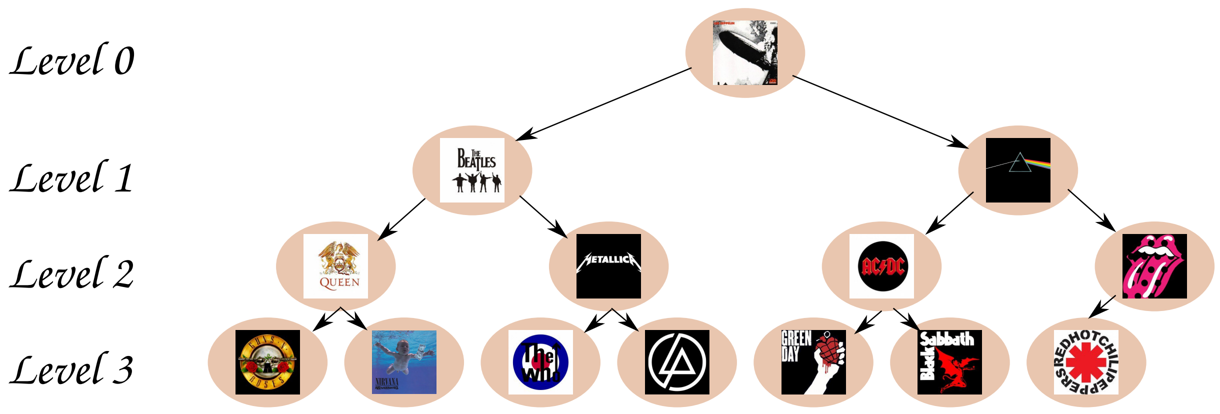


Figure 1: A heap built based on one's musical preference

What makes a heap a heap is the following additional property:

- a node has priority over all its descendants.

Note that the statement above **does not** mean that if a node N_1 has priority over a node N_2, then N_2 is a descendant of N_1. Nope. It only means that if a node N_2 is a descendant of N_1, then N_1 has priority over N_2.

Assignment requirements

Question 1 (0 credits): Assume that we have an array of n `string`s and that we use the position of the elements as priority.

```
string s[n];
```

Prove that the following is a heap:

- The root of the heap is the element at item 0.
- The children of node at position i are the items at position $2 * i + 1$ (as the left child) and $2 * i + 2$ (as the right child). If $2 * i + 2 = n$, then the item at position i has only one child. If $2 * i + 2 > n$, then the item at position i has no children.

Question 2 (0 credits): We will use a `struct` named `Heap` to store the contents of a heap based on the tree structure. That is

```
struct  Heap {
        string name;
        Heap *left;
        Heap *right;
};
```

Write a function that takes as input an array of strings and returns the `Heap` `struct` built using the strategy in Question 1. If a node does not have a left or a right child, the pointer that corresponds to the missing child should be equal to null pointer (i.e. `left = nullptr;`). The declaration of the function is:

```
Heap *heapFromArray(string  *input, int length);
```

Note: you will find it easy to solve this question if you create the left and right children of each node as soon as you create a `Heap` `struct` and then save the address of these newly created `struct`s in an array which is ordered the same as the content.

Question 3 (20 credits): Write a function that takes as input a `Heap` `struct` and returns the number of elements in that heap. The declaration of the function is:

```
int numElements(Heap h);
```

Question 4 (20 credits): Write a function that takes as input a `Heap` `struct` and returns sum of the lengths of all the strings in that heap. The declaration of the function is:

```
int lengthOfContent(Heap h);
```

Question 5 (30 credits): Write a function that takes as input a `Heap` `struct` and returns the content of the heap as an array, based on the following property:

- the item at position 0 is the head of the heap.
- the item at position i in the returned array has the items at positions $2 * i + 1$ and $2 * i + 2$, as the left child and as the right child, respectively.

The declaration of the function is:

```
string *printLinear(Heap h);
```

Note: you might find it easier to first write a method that returns an array with pointers to all the heaps in the order in which you have to print the content. That is, write a method `Heap **returnAllHeaps(Heap h);`. We will not grade this helper method. Do it only if you think it helps you.

Question 6 (30 credits): Write a function that takes as input a `Heap` `struct` and return a single string that represents a visual representation of the heap that follows the following characteristics:

- prints items at level k on the $(k + 1)^{th}$ line.
- all content in the left child heap is horizontally to the left of the content of the node.
- all content in the right child heap is horizontally to the right of the content of the node.

You might find it easy to use the `returnAllHeaps` in Question 5.

Note that to do this you have to print just enough white spaces before you print the content of a node. To know how many white spaces you need to print to the left or to the right of the content, you can use the functions in Questions 1 to 4. Note also that sometimes you will also need to leave an additional amount of white space that is equal to the length of *the parent of the closest ancestor that is a left child* (for the example in Figure 1, after "Metallica" you have to print enough white space to cover "Led Zeppelin"). Luckly, the heap is complete, so the index of left children are always odd, and to get the index of the parent of i, you simply do $(i - 1)/2$.

The declaration of the function is:

```
string printPretty(Heap h);
```

The output corresponding to Figure 1 should be:

```
                                                               Led Zeppelin
                         The Beatles                                                                 Pink Floyd
             Queen                         Metallica                                ACDC                           Rolling Stones
Guns N' Roses     Nirvana           The Who         Linkin Park            Green Day    Black Sabbath          RHCP
```