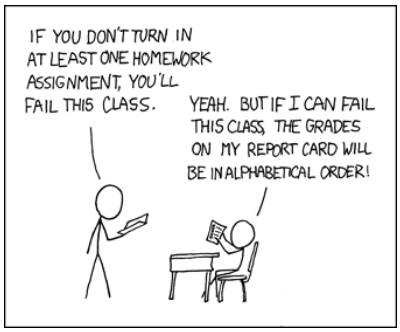
COMP 322 - Assignment 2 - Winter 2016

Due: Mar 18th, 2016 at 11:30PM

Introduction

In this second assignment we continue studying heaps, by allowing the construction of heaps with any kind of priorities (see Figure on the side). We will pay particular attention to two of the most famous algorithms in computer science: ***heapify-up*** and ***heapify-down***. But before we do that, we will set our priorities straight and make sure that we build our code with class. We will design a class `Heap`, that is.



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 many of you for my Office Hours, and, yes, I use the arrival time as the priority.
- 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 `priorities.h`. Your submission should compile using the `make` command on Trottier (3rd floor) machines when run on the **original** `Makefile`, **original** `priorities.h` and your submission `prioritiesStudent.cpp`. ONLY submit the `.cpp` file.

Heaps

You should be familiar by now with the concept of a heap, which we introduced in the last assignment and which you have already experimented with. There is only one part which I want to emphasize a second time. Make sure that the statement below has a big priority in your head as you solve this assignment.

```
The following property makes a heap a heap: a node has priority over all its descendants.
```

The statement does not say 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.

Since we are only going to use the array implementation of a heap (using the `vector` class), we will rephrase the statement as follows:

```
The item at index i has priority over the items at index 2*i+1 and 2*i+2.
```

Although I am sure that you will find videos and tutorials about heapifying all over the internet, I will make an attempt to explain the algorithms here:

Heapify up

Assume that we already have a heap where all the items satisfy the heap priority. How do we insert an item in the heap? Since we want to maintain the heap *balanced*, we will ***first*** add the item at the end of the `vector` storing the items. But now the property is violated, therefore we now have to fix that. To do it, we *heapify-up* the item that was inserted. That is, we compare the item with *the parent*, and ***swap*** if necessary. We continue swapping up until the item inserted is at its rightful place (i.e. it has no priority over its parent).

```
While the new item has priority over its parent, swap item with its parent.
Note that after every swap its index goes from i to (i-1)/2, and its parent changes!
```

Heapify down

Assume that we already have a heap where all the items satisfy the heap priority. How do we remove the item with top priority from the heap? Since we want to maintain the heap *balanced*, we will ***first*** swap the last element in the heap (i.e. the item at the end of the `vector` storing the items) with the top. We *next* remove the last element. But now the property is violated, therefore we now have to fix that. To do it, we *heapify-down* the item that was placed at the top. That is, we compare the item with its children, and ***swap*** the item with the child that has priority over all three items (parent, left child, and right child) if necessary. We continue swapping down until the item falls at its rightful place (i.e. it has priority over its children).

```
While the item does not have priority over both its children, swap item with
the children of higher priority.
Note that after every swap its index goes from i to 2*i+1, if left child has priority over right child,
or to 2*i+2 otherwise.
```

Assignment requirements

Question 1 (0 credits): Familiarize yourself with the declaration code for the class `Heap`, and the few member functions for which we provide in-line code. Note the following:

- To keep together every item and its priority, we will use a `struct` of type `Node`, which is declared under the `private` part of the class.
- Content is saved in a `vector` of `Nodes`, much as we used arrays in the first homework.
- There is a method `swap` that swaps items at two given indeces. Make sure to use it when implementing your solutions.
- `lengthOfContent, heapifyUp, heapifyDown` are all private: users of `Heaps` should not know of their existence.
- There are 4 constructors, of which you will implement some in Question 1, and one in Question 7.
- The in-line member function `top` simply returns the content at the top of the heap.
- The in-line member function `has` simply returns true if a given `string` is existent in the heap.
- The in-line member function `empty` simply returns true if the heap is empty.
- The non-member function `operator<<` is declared as a `friend` because we would like to give it access to the data declared private in class `Heap`.

Implement all its constructors, ***except*** the one taking an `istream` as a parameter. That is, the copy constructor and the constructor that would build a heap where the priority of items is the index in a `vector` (as done in the first homework).

```
Heap::Heap(const vector<string>&);
Heap::Heap(const Heap&);
```

Question 2 (0 credits): Based on the solution that was given to you for the first homework, overwrite the printing operator, `operator<<`. Note that all your printing should be done to the `ostream` given as parameter, and NOT at `cout`. Also, make sure to return ***the same*** `ostream` when you are done with the function. You might find it useful to implement the helper method `lengthOfContent` recursively before writing the `operator<<`.

```
size_t Heap::lengthOfContent(unsigned long) const;
ostream &operator<<(ostream &, const Heap&)
```

The format of the output should be the exact same as the one described in the first assignment. Do not change the format.

Question 3 (15 credits): Write the equivalent of the function `printLinear` from the first homework. That is, go over the elements in `content` and create a string in the format `(name, priority)` and add it to a `vector` of `strings`, which you have to return. The method to implement is

```
vector<string> Heap::printLinear() const;
```

Question 4 (10 credits): Implement the element access operator, which gives the priority of an item using the square bracket notation. That is, the method returns the priority of the item that has the name given as parameter. If the item is not in the heap, return 0.

```
unsigned int Heap::operator[](string) const;
```

Question 5 (30 credits): Write a function that inserts an element in the heap. First, you will have to implement the private method `heapifyUp` (using recursion). Next, use `heapifyUp` and the instructions given on the previous page to insert the given item with the given priority. Don't forget to implement the increment operator `operator+=`, which takes another heap as input and inserts all items from this additional heap into the current heap. Note that if a node with the same data (i.e. name) exists, it should not add it a second time, nor should it change the priority.

```
void Heap::heapifyUp(unsigned long);
void Heap::push(string, unsigned int);
Heap& Heap::operator+=(const Heap&)
```

Question 6 (30 credits): Write a function that removes the top element in the heap. First, you will have to implement the private method `heapifyDown` (using recursion). Next, use `heapifyDown` and the instructions given on the previous page to remove the top item.

```
void Heap::heapifyDown(unsigned long);
string Heap::pop();
```

Question 7 (15 credits): White the last constructor, which creates a heap based on the frequency of words coming from an `istream`. In the last assignment, this will be a text file. Your constructor should do the following:

1. Create a map that keeps a count of the words in the input stream.
2. Read word by word the input stream and update the counts as you do that. Do not worry about punctuation. We will do that in the next homework. A word is any sequence of characters separated by white spaces. The following is a word: `"way!"`.
3. Find the word that occurs the most times in the stream. Save this value into `max`.
4. Insert every word in the heap with priority `max - count`, such that the word that is most frequent has priority value 0, and all words that are less frequent have higher and higher priority values.

The method you have to submit for this question is

```
Heap::Heap(istream &);
```