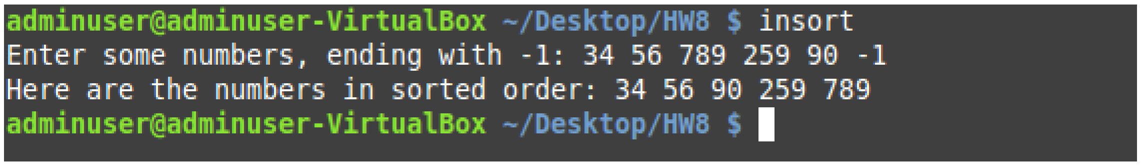
CSCI 2271 Computer Systems

Assignment : Structs

Due Friday, April 26

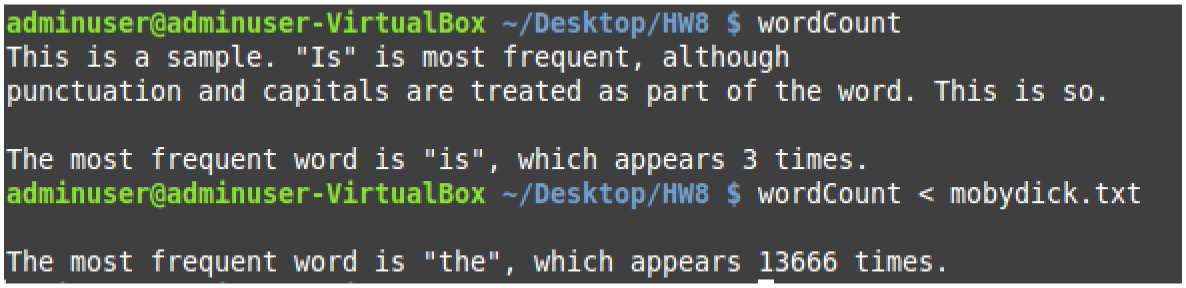
1. (This is a warmup exercise.) Write a program, named **intsort.c**, which reads in a sequence of **positive integers**, saving each one in a sorted chain of nodes. The input value -1 indicates the end of input, at which point the program traverses the chain and prints the integers in sorted order. The following screenshot shows the program in action:

   ```
   adminuser@adminuser-VirtualBox ~/Desktop/HW8 $ insort
   Enter some numbers, ending with -1: 34 56 789 259 90 -1
   Here are the numbers in sorted order: 34 56 90 259 789
   adminuser@adminuser-VirtualBox ~/Desktop/HW8 $
   ```

   To make grading easier, your program should use the struct definitions and functions defined in the file **intsort.h**. In particular, your program should implement (and use) the functions createNode, insertNode, and printAll, as defined in intsort.h.

2. Write a program, called wordcount.c, that reads one word at a time from the standard input. It keeps track of the words read and the number of occurrences of each word. When it encounters the end of input, it prints the most frequently occurring word and its count. The following screenshot shows the program in action:

   ```
   adminuser@adminuser-VirtualBox ~/Desktop/HW8 $ wordCount
   This is a sample. "Is" is most frequent, although
   punctuation and capitals are treated as part of the word. This is so.

   The most frequent word is "is", which appears 3 times.
   adminuser@adminuser-VirtualBox ~/Desktop/HW8 $ wordCount < mobydick.txt

   The most frequent word is "the", which appears 13666 times.
   ```

   The first time I called wordcount, I entered two lines of text as input, followed by a **<CTRL-D>.** (When entering text from the keyboard, typing **<CTRL-D>** at the beginning of a line signals end-of-input.) The line beginning "The most frequent word..." is the **output**. The second time I called the program, I used **redirection** (<) to have the file mobydick.txt be treated as the standard input.

Your program should consider a "word" to be a sequence of non-blank characters, as retrieved by the "%s" format in scanf. No need to remove punctuation or turn uppercase letters into lower-case ones. For example, "is", "Is", and "is." are all different words.

Your program should store the word information in a **binary search tree**. The file bst.h gives the definitions of the BSTnode structure, and defines some functions you should implement. In particular, a node contains a pointer to an item, plus a pointer to the left and right child nodes. The item is a **(void *)** pointer, which means that the node is able to store anything. You should write a file bst.c to implement these functions.

The program wordcount.c will use a BST that has one node in the tree for each distinct word. The item for the node should be a pointer to a WordCount structure, defined as follows:

```
struct wordcount {
     char word[80];
     int count;
};
typedef struct wordcount WordCount;
```

You are free to implement the code as you like. I strongly encourage that you decompose it into small, testable functions.

When you are finished, submit your files insort.c, bst.c, and wordcount.c to Canvas.