CSE 101
Introduction to Data Structures and Algorithms
Programming Assignment 7

In this project you will re-create the your Dictionary ADT from pa6, now based on a Red-Black tree. You will also write a new top level client called Order.c that will print keys from the dictionary in the orders given by the tree traversals: pre-order, in-order and post-order. This new client is essentially Lex.c again, but now printing only the keys, no values.

Red black trees are covered in Chapter 13 of the text, and will be discussed in lecture. All relevant algorithms for RBTs (and BSTs) are posted on the webpage under Examples/Pseudo-code. Aside from having a RBT as its underlying data structure, your Dictionary ADT will have only slight changes to its interface.

The new prototype for function `printDictionary()` is

```
void printDictionary(FILE* out, Dictionary D, const char* ord);
```

If the parameter `ord` is one of the strings "`pre`", "`in`" or "`post`", then `printDictionary()` will print out the keys (but not the values) in the corresponding tree traversal order. If `ord` is some other string, the function will print nothing. Note that this function implicitly refers to the underlying RBT data structure, and therefore should not be considered an ADT operation. Instead, think of it as a diagnostic function. Since your client module Order.c will call `printDictionary()` on each order, it is really just a test module for the RBT implementation of the Dictionary. Its purpose in the project is to verify that you really have built a RBT, and are not just re-using the BST from pa6.

You will notice one other slight change to the header file Dictionary.h, which is posted on the webpage in Examples/pa7. The `VAL_TYPE` macro has been changed from `int` to `int*`, and other macros related to values are changed accordingly. This is to facilitate another client called WordFrequency.c, also posted on the webpage. This program parses a text file, and for each word in the file, calculate the number of times it occurs, i.e. its frequency. There are two large text tiles included in Examples/pa7 for testing purposes. The first is Big.txt, and contains several large books, one of which is *War and Peace* by Leo Tolstoy. The second is Shakespeare.txt, containing the complete works of William Shakespeare. The reason for the `int*` value type is so that this client can increment the number of occurrences of each word. This explains why `printDictionary()` no longer prints values. Since your Order.c client will not use these values, you may simply load `NULL` as the value associated with each key in that program.

A number of input-output file pairs for Order.c are posted at Examples/pa7. To receive full credit, your output file format must match these examples exactly. There is also a random input file generator called RandomInput7.py. Note that this program produces files with (possibly) repeated lines, so the Dictionary you create in Order.c must allow non-unique keys.

What to turn in
Submit the following 6 files to your pa7 directory on GitLab @ UCSC.

README	Written by you, a catalog of submitted files and any notes to the grader
Makefile	Provided, alter as you see fit
Dictionary.h	Provided, **do not alter**
Dictionary.c	Written by you, based on an RBT
DictionaryTest.c	Written by you, your test client of the Dictionary ADT

Order.c	Written by you, a modification of Lex.c, printing 3 orders, keys only

Makefile should be capable of making the executables WordFrequency and Order, and should contain a clean utility that removes all binary files. To get full credit, your project must implement all required files and functions, compile without errors or warnings, produce correct output on our ADT and client tests, and produce no memory leaks under valgrind.

As usual, do not turn in any executable files, binaries, or anything not listed above. Start early, ask plenty of questions, and submit your project by the due date.