ECE 361 Fall 2024
Homework #5

THE ASSIGNMENT IS DUE BY 11:59 PM ON THU, 05-DEC. **NO SUBMISSIONS WILL BE ACCEPTED AFTER 8:00 AM ON FRI, 06-DEC**. THIS IS AFTER OUR LAST CLASS MEETING OF THE TERM SO I WILL POST A VIDEO DISCUSSING MY SOLUTION AFTER THE DUE DATE.

THE ASSIGNMENT IS WORTH 100 POINTS. IT IS EASIEST TO GRADE, AND I BELIEVE IT IS EASIEST FOR YOU TO SUBMIT, SOURCE CODE FILES AND TRANSCRIPTS AS TEXT FILES INSTEAD OF DOING A CUT/PASTE INTO A `.doc` OR `.docx` FILE. CLEARLY NAME THE FILES SO THAT WE KNOW WHICH QUESTION THE ANSWER REFERS TO. SUBMIT THE PACKAGE AS A SINGLE .ZIP, .7Z,.TGZ, OR .RAR FILE (EX: `ece361f23_rkravitz_hw5.zip`) TO YOUR CANVAS HOMEWORK #5 DROPBOX. IF YOU COMPLETE THE OPTIONAL GITHUB PORTION OF THE ASSIGNMENT, MAKE YOUR FINAL PUSH AT THE SAME TIME. GITHUB HAS AN OPTION TO GENERATE A .ZIP FILE SO YOU CAN "KILL TWO BIRDS WITH ONE STONE.

IMPORTANT: PLEASE NOTE THAT WE HAVE BEEN LENIENT THIS TERM, ACCEPTING SOME ASSIGNMENTS THAT WERE EMAILED TO US SHORTLY AFTER THE LATE SUBMISSION DEADLINE AND/OR ALLOWING SOME RESUBMISSIONS IF WE DISCOVERED THAT YOU SUBMITTED THE WRONG FILES. THAT WILL NOT BE THE CASE ON THIS ASSIGNMENT! WE WILL GRADE YOUR LATEST SUBMISSION TO YOUR CANVAS DROPGOX PRIOR TO THE LATE SUBMISSION DEADLINE – THE CONSEQUENCES OF MISSING THE DEADLINE OR NOT DOUBLE CHECKING THAT YOU SUBMITTED THE FILES YOU MEANT TO SUBMIT PRIOR TO THE DEADLINE ARE ON YOU. WHILE YOU WILL NEED TO BE CREATIVE ON THIS PROJECT, WE DO NOT WANT YOU TO BE "INNOVATIVE." PLEASE USE THE IOM361_R2 API AND BUILD YOUR CODE ON THE EXAMPLES WE PROVIDED IN THE STARTER FOLDER. WE WANT TO GIVE YOU TIMELY FEEDBACK ON THIS ASSIGNMENT AND THAT'S HARD TO DO IF WE HAVE 40+ VARIANTS TO WORK THROUGH. THANKS!

CONSOLE TRANSCRIPTS SHOULD INCLUDE YOUR NAME, USING THE `whoami` COMMAND AND THE STATEMENTS WITHIN YOUR CODE THAT SHOWS THE WORKING DIRECTORY. TRANSCRIPTS SHOULD BE A .TXT FILE SAVED FROM THE BASH COMMAND LINE (OR EQUIVALENT). PLEASE INCLUDE A NUMBER OF TEST CASES (INCLUDING EDGE CASES) IN EACH TRANSCRIPT TO DEMONSTRATE THE FULL FUNCTIONALITY OF YOUR PROGRAM. THE TRANSCRIPT IS YOUR CHANCE TO SHOW OFF YOUR HARD WORK.

Introduction:

You are going to make one more use of the simulated AHT20 Temperature/Humidity sensor in `iom361_R2` for this assignment. There is only one programming assignment in this homework and the application is conceptually straightforward – you will generate and read one month's (28, 30, or 31 days, your choice) random temperature and humidity "readings" from `iom361_r2`. You can choose the starting day and month. Your application will "take" one reading per your calendar period and add a timestamp to the readings using the date/time-related functions in the C time library (`time.h`). The time-stamped readings are stored in a **B**inary **S**earch **T**ree.

Your application should query the user for a specific date, search the BST, and display the temperature and humidity for that date if the date is in the tree. A reference to a specific date that is not in the table should display a suitable message. Data entry should continue until the user enters a blank line. Once the user enters a blank line your application should do an Inorder traversal of the tree to print out a table of date and temperature/humidity pairs. An Inorder traversal should result in a sorted list.

Although we have provided examples of BST code in the starters folder, there is no specific implementation for this BST. As I mentioned in class, you can think of a Binary Search tree as an architecture and data structure, but the actual code is often written for the specific application.

In addition to producing a working application you will also need to create a `makefile` for the assignment. You may choose to use git and GitHub on this assignment but it is not required, although those that successfully use git and GitHub on the assignment will receive extra credit.

EPOCH Time (https://en.wikipedia.org/wiki/Epoch_(computing)):

(edited by Roy) In computing, an **epoch** is a fixed date and time used as a reference from which a computer measures system time. Most computer systems determine time as a number representing the seconds removed from a particular arbitrary date and time. For instance, Unix and POSIX measure time as the number of seconds that have passed since Thursday 1 January 1970 00:00:00 UTC, a point in time known as the *Unix epoch*. Windows NT systems, up to and including Windows 11 and Windows Server 2022, measure time as the number of 100-nanosecond intervals that have passed since 1 January 1601 00:00:00 UTC, making that point in time the epoch for those systems.

Computers do not generally store arbitrarily large numbers. Instead, each number stored by a computer is allotted a fixed amount of space. Therefore, when the number of time units that have elapsed since a system's epoch exceeds the largest number that can fit in the space allotted to the time representation, the time representation overflows, and problems can occur. Most famously, older systems that counted time as the number of years elapsed since the epoch of 01-January-1900 and which only allotted enough space to store the numbers 0 through 99, experienced the Year 2000 problem (Y2K). Even systems that allocate more storage to the time representation are not immune from this kind of error. Many Unix-like operating systems which keep time as seconds elapsed from the epoch date of 1 January 1970, and allot timekeeping enough storage to store numbers as large as 2 147 483 647 will experience an overflow problem on 19 January 2038. This is known as the *Year 2038 problem*.

Programming Assignment: The Binary Search Tree

1. (20 pts) Study Karumanchi's binary tree and binary search tree code. Refactor (rewrite) the code into an Abstract Data Type with an API following the same sort of process you followed when creating other ADT's (like your Stack ADT) in this course; that is, create an `.h` file, a `.c` file. Use Doxygen-style comments to document each function in your API. Your data could be defined in a struct like this one:

```
#include <time.h>
#include <stdint.h>

struct temp_humid_data {
      time_t timestamp,
      uint32_t temp,
      uint32_t humid
}
```

   At a minimum, your ADT should have API functions to create a new **B**inary **S**earch **T**ree, insert a node into the BST, search for a specific timestamp in the nodes of the BST and, if found, retrieve and return the data. You should also provide a function that performs an `Inorder` traversal of the tree, displaying the values to produce a timestamp-ordered list. Doing this will make it easy for your application to display the table of dates, temperatures, and humidities. You may add additional functionality such as deleting the entire tree as you see fit, but they are not required. ***Include the source code and header file(s) in your submission.***

2. (15 pts) Write and successfully execute a test program for your BST ADT. Your test suite does not need to be exhaustive, but it should test the functionality, and to some extent, the ADT's handling of potential error cases like unintended access to a NULL or invalid pointer. Compile the program using the bash command line and gcc and write your output to `stdout`. ***Include a the source code for your test program and a command line transcript showing the results of your testing.***

3. (15 pts) Design, write, and test a function called `populateBST()` that uses the C `time` library functions (ex: `time(), mktime(), localtime(), strftime(),` …) and either the `iom361_r2 _iom361_setSensor1()` or `the _iom361_setSensor1_rndm()` function and the `iom361_readReg()` function to populate the data in your BST.

   Note: It is not optimal to add nodes to the BST in timestamp order (ex: 10/1, 10/2, …). In fact, the performance of a BST degrades if nodes are added in order – random is better. One way to do ensure some form of randomness is to generate all the data records for all the days of your selected month into an array and declare a second array that randomizes the order you index the array when you add the data record to the BST. Randomizing the order of accessing

an array is often called “shuffling.” Research the best algorithm for shuffling an array. I started with this link: https://stackoverflow.com/questions/6127503/shuffle-array-in-c . *Hint: I used this function in my BST ADT test program to populate the BST so I could test/debug it before including it in my application.* **There is no deliverable for this step – the source code for this function will be included in your application.**

4. (30 pts) Design, debug, and implement an application that populates the BST with {timestamp, temp/humid} pair and provides an interface for the user to request the data for a specific date. The application should continue to process searches until the user enters an empty line displaying the temperature and humidity when the node is found or a descriptive message if it is not. After the user has finished searching by entering a blank line, the application should traverse the BST and display an ordered table of {timestamp, temp/humid} readings. ***Include the source code and header file(s) for your application in your submission.***

5. (5 pts) Run your application at the command line. Try enough test cases to show that your application can successfully build, search, and traverse the BST to get an ordered list of timestamped temperature and humidity readings. ***Include a command line transcript showing the results of your testing.***

6. (10 points) Create and use a makefile to generate your object and executable files. Your makefile should include a `make clean` target to delete the .exe and .o files. ***Include the source code for your makefile in your submission.***

7. (5 points) Demonstrate that you can use your `makefile` to create object files and an executable and that you can successfully run your application from the command line**. Include a transcript of these operations showing the build process exercising the Targets in your makefile in your submission.**

8. (Optional, up to 3 pts of extra credit) Use git and GitHub for version control***. Push your final submission to GitHub but also upload a .zip of your repository to your Canvas Homework #5 dropbox.*** Include a README in your submission that has the link to your repository. You may have to make the repository public during the grading period, but generally, all of your classroom assignments on GitHub should be in private repositories.

Deliverables:

We thought it could be useful to include a checklist of your deliverables for your assignment. Remember: There are no do-overs for this assignment. We will grade your most recent submission before the Late deadline.

☐ **STEP 1** – Design and implement your Binary Search Tree ADT and API: Include the source code and header file(s) of your BST ADT in your submission.

☐ **STEP 2** – Design, implement, and execute a test program for your Binary Search Tree ADT and API: Include the source code for your test program and a command line transcript showing the results of your testing in your submission.

☐ **STEP 3** – Design and implement the `populateBST()` function that can be used to populate the BST with {timestamp, temperature/humidity} data. There is no deliverable for this step – the code for this function will be included in your application**.**

☐ **STEP 4** – Design, implement, and debug your application: Include the source code and header file(s) for your application in your submission**.**

☐ **STEP 5** – Run your application at the command line: Include a command line transcript showing the results of your testing.

☐ **STEP 6** – Create and use a makefile to generate your object and executable files: Include the source code for your makefile in your submission.

☐ **STEP 7** – Demonstrate your makefile: Include the source code for your makefile in your submission. Include a transcript of these operations showing the build process exercising the Targets in your makefile in your submission.

☐ **(OPTIONAL) STEP 8** – Use git and GitHub for version control: Push your final submission to GitHub but also upload a .zip of your repository to your Canvas Homework #5 dropbox. Include a README in your submission with a link to the repository.