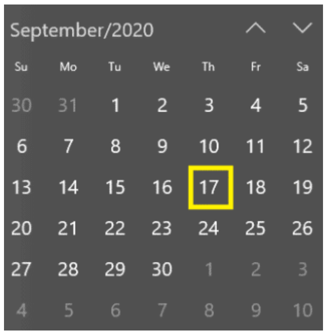
Project 3 – Word List (Binary Search Tree Application)



September/2020						
Su	Mo	Tu	We	Th	Fr	Sa
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

Groupings : At most 3 members in a group
Deadline : September **17**, 2020 (Thursday) 11:59 PM
// FYI: Final Exam Week is from Sept 28 (Mon) to Oct. 3(Sat)
// Grade consultation day is on Oct. 5 (Mon)
Percentage : 10% (part of the 45% for Research Work/Machine Project)
Programming Language: C / Java / Python

Let's apply what you learned in Binary Search Trees. For this project, you will design and develop a Word List program that will create an alphabetical list of words with corresponding frequency count from a given text file. Read, understand and comply with the project specs described below.

I. INPUT

The input is a named text file. It is made up of a collection of symbols such as alphabets, digits, punctuation marks, and invisible characters (such as space, tab, newline) and other characters found in a typical keyboard such as #, @, {, |, etc. The text file contains mostly words.

II. OUTPUT

Based on the contents of the input text file, the Word List program will produce as output a text file named WORDS.TXT. It contains a list of English words found in the input file, arranged in alphabetical order with their corresponding frequency count. Only words with a length of at least 3 should be written in the output file.

The example below shows the contents of a sample INPUT.TXT file and the contents of the corresponding output WORDS.TXT file.

INPUT.TXT
Hello, how are you today? I'm fine, thank you! How about you? Today is the present. Tomorrow is a gift.

WORDS.TXT	
about	1
are	1
fine	1
gift	1
hello	1
how	2
present	1
thank	1
the	1
today	2
tomorrow	1
you	3

III. REQUIRED PROGRAM INTERACTION

There should be minimal program interaction as shown in the sample runs below. The program will just ask the user to input the name of the input text file. If the text file exists, its contents will be processed, WORDS.TXT file will be output, and the program terminates. If the text file does not exist, the program outputs "<FILENAME.TXT> not found." error message and then terminates.

Example Run #1 - INPUT.TXT file exists, output file WORDS.TXT will be produced.

```
Input filename: INPUT.TXT
```

Example Run #2 – XYZ.TXT file does not exist; there will be no output text file.

```
Input filename: XYZ.TXT
XYZ.TXT not found.
```

IV. TASKS

Task 1. Design and implement your Binary Search Tree (BST) data structure.

- Decide how many BST you'll need. You must use at least one BST for storing the words found in the input text file. The option is to have one BST for each group of words that start with the same letter.
- At the minimum you must function definitions for the following BST operations:
  - `Create()` – produces an empty BST
  - `Search()` – determines if a search key exists in the BST
  - `Insert()` – adds a new node in the BST
  - `Inorder()` – performs inorder traversal of the BST
  - `Destroy()` – This function should be called as a clean-up operation before the program actually terminates. If dynamic memory allocation was used, then this function will also free up the memory space. The BST will become empty after calling `Destroy()`.
- Practice modular programming. That is, compartmentalize the data structures and operations by storing the implementation codes in separate source code files.
- For example, the codes for the BST data structure is stored in `bst.h` and `bst.c`.

Task 2. Implement the algorithms for parsing, processing the contents of the input file, looking up and storing data in the BST, and writing the word list in the output file.

- Again, practice good programming. Compartmentalize your solution/functions in separate source code file(s).
- NOTE: you should not CALL any of the usual sorting algorithm for the project! The BST should handle the alphabetical ordering of words.

Task 3. Integrate the modules.

- Create the main module which will include the other modules, and call the appropriate functions to achieve the task.

Task 4. Test your solution.

- Design your test cases, and perform exhaustive testing.

Task 5. Document your solution.

- Give a brief description of how you implemented the algorithms. Disclose what is not working.

V. DELIVERABLE

Submit via Canvas a ZIP file named GROUPNAME.ZIP which contains:

- Source files for the BST, and other modules.
- Input text file you used in testing your solution.
- Corresponding output file WORDS.TXT.
- Documentation named GROUPNAME.PDF (see attached Word template).

Do NOT include any EXEcutable file in your submission.

VI. WORKING WITH GROUPMATES

You are to accomplish this project in collaboration with your fellow students. Form a group of at least 2 to at most 32 members. The group may be composed of students from different sections. Make sure that each member of the group has approximately the same amount of contribution for the project. Problems with groupmates must be discussed internally within the group, and if needed, with the lecturer.

VII. NON-SUBMISSION POLICY

Non-submission of the project by the due date will result to your deferral for the course. Based on the existing university policy, you will be given the next trimester to complete a missing requirement.

VIII. HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honest policy applies. You are encouraged to read and gather information you need to implement the project. **However, please take note that you are NOT allowed to borrow and/or copy-and-paste -- in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by other people that are not online).** **You should develop your own codes from scratch by yourselves, i.e., in cooperation with your groupmates.** According to the handbook (5.2.4.2), "faculty members have the right to demand the presentation of a student's ID, to give a grade of 0.0, and to deny admission to class of any student caught cheating under Sec. 5.3.1.1 to Sec. 5.3.1.1.6. The student should immediately be informed of his/her grade and barred from further attending his/her classes."

IX. RUBRIC FOR GRADING

REQUIREMENT	RATINGS		
1. BST	**COMPLETE [50 pts]** Correctly implemented the BST data structure and the specified operations.	**INCOMPLETE [5 to 30 pts]** Implementation is not entirely correct.	**NO MARKS [0 pt]** Not implemented.
2. Input File Parsing	**COMPLETE [20 pts]** Correctly parsed the contents of the input file.	**INCOMPLETE [5 to 12 pts]** Implementation is not entirely correct.	**NO MARKS [0 pt]** Not implemented.
3. Output File	**COMPLETE [15 pts]** Output file contents are correct and complete.	**INCOMPLETE [5 to 10 pts]** Output file contents are not entirely correct or not complete.	**NO MARKS [0 pt]** Not implemented.
5. Documentation	**COMPLETE [10 pts]** Required details were covered.	**INCOMPLETE [1 to 6]** Some required details were not discussed.	**NO MARKS [0 pt]** No discussion or documentation.
6. Compliance with Instructions	**COMPLIANT [5 pts]** All instructions were complied with.	**NON-COMPLIANT [0 to 4]** Deduction of 1 point for every instruction not properly complied with.	**NO MARKS [0 pt]** More than 4 instructions not complied with.
			Maximum Total Points: 100

Question? Please post it in the Canvas Discussion thread. Thank you for your cooperation.

サルバドール・フロランテ