Project 1: Binary SearchTree
KECE208 Data Structures and Algorithms
Fall, 2020

Instructor: **Hwangnam Kim**
TA: **Sangmin Lee, Hyeontae Joo**

Goals and Overview

In project2, you are assigned to use your knowledge with binary search tree (BST) in order to fill in the functions that are provided at project files. Students are required to fill in empty functions in order to get outputs and get grades.

Given Files

BST.c **BST.h** **project2.c**

You are responsible for completing implementation of 'BST.c' in order to make the program run properly. You are not allowed to modify 'BST.h' and 'project2.c'.

- **BST.h**: This is the header file of BST.c which contains all of the functions and values that are going to be used in BST.c. **You should not add any more functions or variables in order to complete the project**.

- **BST.c**: This contains the implementations, definitions of functions, and variables, and you are required to fill some of the functions that contain comments "Your code starts from here". By filling and completing implementation of those functions, you will get proper output for this project.

- **project2.c**: This file is provided to students in order to debug whether their codes works correctly or not. The usage of this code will be explained later during project2 guide lecture. Time and location will be announced shortly.

- **Makefile:** This file compiles the c files. If students type "make", this will compile all the code. When students type "make clean", this will erase executable code and force the user to recompile later.

- **README.txt**: This file is not provided; you have to make it on your own. This file should contain detailed descriptions of each implemented function and student's name and ID.

Data structure which are defined in order to make students to deal with BSTs in proper manner

```
typedef struct bstNode* treeNode;

struct bstNode{
       char data;
       treeNode parentNode;
       treeNode leftNode;
       treeNode rightNode;
};
```

This is written in BST.h file and you are not allowed to modify any of these.

Description for each function listed on BST.c

1. You need to follow exactly what descriptions say and implement each function according to the description.
2. The root node pointer is defined in BST.h as *treeNode rootNode*. It can be accessed in all functions, all file in this project.
3. Each node of BST has character data. The order of character conforms to ASCII order.

(http://www.asciitable.com/)

4. Each function also has some error cases that it should deal with. You should print the error message when node is duplicated or access to node which is not in the tree.
5. All the input that you typed in is limited to **small alphabet (a to z)** except main menus. You don't need to care about this in error case.
6. You are not allowed to create any function other than what we have provided. Students will lose points if any additional functions are included within the project.
7. You are not allowed to include any other header files.

treeNode createNode(*char inputData*) **(provided)**

This function creates and returns a new node with input data. It should return "NULL" when it is failed with allocation of memory to a new node or a pointer of the new node created in this function.

1. int insertNode(char inputData)

This function inserts a new node to BST. If there is no root node, it should put the node as a root node -- *rootNode*. This function creates a node with the *inputData*, and puts the node at the proper location in BST. It should print error message in case of duplication of data or when creating a node is failed. Also, it should print message at successful insertion.

- <Node data> is inserted successfully (ex: c is inserted successfully)

2. int isNodeExist(char inputData)

This function checks the tree with input data. If there is a node which data is *inputData*, it returns "1". If node is empty or node is not in the tree, it should print message and returns "0". And the return value should be used to enable the other functions; *findDepth* and *findNode.*

- Tree is empty
- <Node id> is not in the Tree (ex: c is not in the tree)

3. treeNode findPosition(char inputData, treeNode node)

This function finds and returns the node that is the nearest node with the given data or the exactly matched node. The method to find the node is implemented in the recursive way, and it compares the input data and the data of input node in each step. The results of this function should be used to organize most of the other functions in this project.

4. int findDepth(char inputData)

This function finds a node that has *inputData*, and then calculates depth of the node. It should return "0" when there is no matched node that has *inputData*, otherwise this function should return "the depth of the node". The result of this function should be included in the function, *findHeight* to calculate the heights of trees.

5. treeNode findMin(treeNode node)

This function finds a node that has the minimum value as data in the subtree. Generally, the parameter of this function might be the root node of the tree. If not, it will return the minimum value of subtree which has the input node as its root node. It should return a pointer of the node that has minimum value in the subtree.

6. int findNode(char inputData)

This function finds a node that has *inputData*. If the node exists in the tree, it should print the data of parent node, left child node, right child node, and the depth of the node. If the tree is empty or the node is not in the tree, it should print message like the result of *isNodeExist* function. You may need to use *findPosition* and *findDepth* function.

- <Node id> is in the Tree
  Parent node is <Parent Node id>
  Left child node is <Left Child Node id>
  Right child node is <Right Child Node id>
  Depth of <Node id> is <Depth>

- c is in the Tree
  Parent node is b
  Left child node is NULL
  Right child node is d
  Depth of c is 2

7. int findHeight(void)

This function finds the height of tree. It should return "-1" when there is no tree, otherwise this function should return "height of tree" and print out message. The height of a tree with only one node (the root) is zero. This function has no parameter but use the related functions in this project.

- The height of tree is <height of tree> (ex: The height of tree is 3

Comments

Every single function in BST.c should have a comment formed as follows

```
/*
   @Function name:
   @Parameters:
   @Descriptions:
   @Error cases:
*/
```

1. Also, you should comment within the function in order to make TAs and yourself what you have been tried to do with codes written. If there is some code that is not understandable and there is no comment on that, points will be deducted.

2. Comments should be all in ENGLISH. Sometimes compile error or running error occurs because the written language is not English. If a student didn't write comments in English and some error happened because of that, all the responsibility belongs to the student so functionality points will be deducted.

Project Requirements & Scoring Criteria (100 points in total)

1. This is an individual project. Making a copy of the content of your friend's HW, books, google, naver, etc., will get an F for this course. You won't get any partial points for redoing the work. Even if you have different code or variable names, using the same function(s) that is not originated from C library which is initially installed with your Linux (or cygwin, etc.) will be regarded as copying. The person who provides the source will also be considered as cheating and will be penalized. Also, asking your friends to do your work will definitely be considered as cheating.

2. Compiling in Linux (cygwin) environment is strongly recommended. The initially given code may not be compiled in other compilers. If you use a compiler other than gcc compiler, you must be able to show the compiling process directly to the TA.

3. BST operations: **70 points** in the following functions should be implemented:
   A. *int insertNode*(*char inputData*) (10 points)
   B. *int isNodeExist*(*char inputData*) (10 points)

C. *treeNode findPosition*(*char inputData*, *treeNode node*) (10 points)
D. *int findDepth*(*char inputData*) (10 points)
E. *treeNode findMin*(*treeNode node*) (10 points)
F. *int findNode*(*char inputData*) (10 points)
G. *int findHeight*(*void*) (10 points)

4. Comments (**15 points** in total)

5. Submission: **15 points** in total
   A. **Due date is 11:59 PM, December 7, 2020 (Monday)**
      i. Submit as early as possible. No excuse for returned email (send failure).
      ii. No late submission is allowed.
   B. Include a README file in your zip file which explains well your program. Korean can be allowed to draw up the document: 5 points
   C. Submit your project files to **mortern800@korea.ac.kr** : 2 Points
      i. The title of mail should be : [DSA PROJ2]your student ID_your name : 3 Points
         (ex : [DSA PROJ2]2019170123_홍길동)
   D. All files should be compressed in one .zip file (other types not allowed): 2 points
      i. The file name should be: [DSA PROJ2]your student ID_your name.zip: 3 points
         (ex : [DSA PROJ2]2019170123_홍길동.zip)
      ii. The file should include:
         1. BST.c
         2. README file