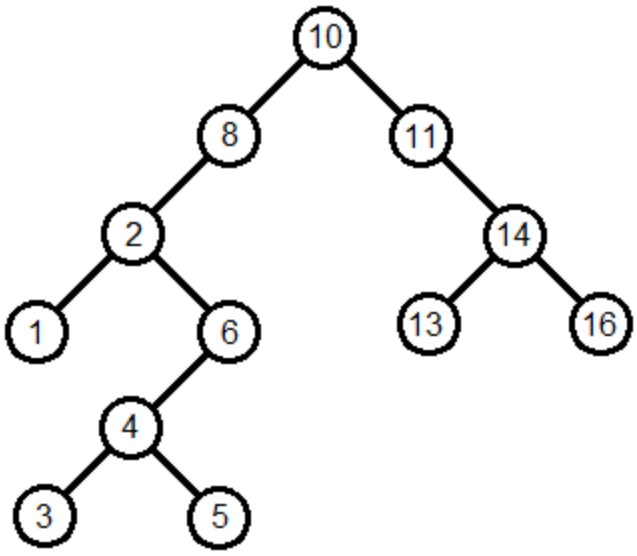
CMPS 12B
Data Structures
Assignment 3



Binary Search Tree

The purpose of this assignment is to create the functions needed to Binary Search Tree (BST) operations. The assignment will be written in C and you will turn in only the functions required. You will write your own "main" program to test and validate your BST functions.

In this assignment, write all error messages to **stderr** instead of **stdout**.

You will create files **BSTClient.c** (for your own use), **BST.c** and **BST.h**.

BSTClient.c - contains the main() and tests. It is the driver for your BST function testing
BST.c - contain the BST manipulation routines
BST.h - contains the BST node definition and BSTfunction prototypes [supplied]

You **BST.h** file should contain this *exact* structure definition and function prototypes:

```
// Exported reference type
// Tree node to store strings, space must be allocated for the strings
typedef struct BSTObj {
    char               *term;           /* string data in each block */
    struct BSTObj      *leftChild;      /* ptr to left child */
    struct BSTObj      *rightChild;     /* ptr to right child */
} BSTObj;
```

```
// add a new node to the BST with the new_data values, space must be allocated in the
BST node
void insert(char *new_data,  BSTObj **pBST);

// print to OUT the inorder traveral
void inorderTraverse(FILE *out, BSTObj *T);

// print to OUT the preorder traveral
void preorderTraverse(FILE *out, BSTObj *T);

// print to OUT the postorder traveral
void postorderTraverse(FILE *out, BSTObj *T);

// print the only the leaves of the tree in order to OUT
void inorderLeaves(FILE *out, BSTObj *T);

// return TRUE if the item_to_find matches a string stored in the tree
int  find(char *term_to_find, BSTObj *T);

// compute the height of the tree
int treeHeight(BSTObj *T, int height);

// create and return a complete mempry independent copy of the tree
BSTObj * copyTree(BSTObj *T);

// remove all the nodes from the tree, doallocate space and reset Tree pointer
void makeEmpty(BSTObj **pT);
```

Your **BST.h** file should also contain the function prototypes exactly as specified above. You may not change any of the definitions. If you do, then your code will not compile in the grading test suite and you will receive **no** credit for the assignment. You need to add comments that describe the contents of **BST.h**. Download file from canvas.

In **list.c** you will implement the following list manipulation routines defined in the **BST.h** file. Note that there is a significant amount of testing to validate the functions.

Each node in the BST will contain a string. The ordering of nodes is determined by their lexicographic string comparison, that is their dictionary order. You can easily compute this using the strcmp() function in the C standard library. You will need to include string.h.

Each function definition must include a comment block as a header that describes any requirements for using the function, any side effects, return values, any error conditions that can result and how you handle these error conditions. Any preconditions assumed by the function should be documented as well. In addition to the function header, you should include a file header comment block which describes the contents of the file. It is acceptable to use either "/* … */" or "//…" comments for your headers but you must be consistent or you will lose points.

What to turn in

You should turn in (using canvas) a .zip file that contains your **Makefile**, **BST.c** & **BST.h**. Do not turn in any file that contains "main()". You will need to create your own main.c file that contains the main() function to test your assignment. You are responsible for creating your own test cases to exercise the code and verify correct operation yourself; this is part of the assignment. You are not allowed to share test cases

with students: You can *talk* about testing strategies and cases but you must not *share code* that exercises the list functions.

Your project will be evaluated with the weighting of 70% for code correctness and 30% for programming style, documentation and clarity. Your file headers must include your name, date, and class information as well. Non-compiling programs will earn 0 points as they are impossible to evaluate. For partial credit, a program must compile and successfully link with our grading environment main.c. Points will be given if some functions work and you follow the guidelines for programming style.

Some advice on getting started

Create your **Makefile** first. In Lab4 you were given a generic Makefile style for any ADT. Edit **BST.h** to add comments. Create a **BST.c** file with only the insert() function. Comment out any functions in **BST.h** and **BST.c** that you are not implementing yet. Compile a simple test program that only implements insert(). Once you have this working them implement one of the traversal routines to check your output. The concept here is to keep your code compiling and functioning while you continue to add functionality and test each improvement.