DEPARTMENT OF
SOFTWARE TECHNOLOGY

CCDSALG

Project 3 – Binary Search Trees

Major Details

Groupings:	At most 3 members in a group
Deadline:	April 12, 2023 (W) 10:30 AM
Percentage:	15%
Submission guidelines:	Submit the `zip` file to AnimoSpace
Filename format:	`CCDSALG-Project3-<Section>-Group<#>.zip`

Deliverables

`Zip` file containing:

- Program source codes – `c` files
- Documentation – `PDF` file

Specifications

For this project, you are tasked to implement binary search trees. Read, understand, and comply with project specifications described below.

Tools

Open the file **`bst.h`**. The `struct` below represents a node in your linked list:

```
1  struct Node{
2      int data;
3      struct Node *pLeft;
4      struct Node *pRight;
5  };
6  typedef struct Node sNode;
```

Figure 1. Code for `struct Node`.

The members of `struct Node` are as follows:

Variable	Description
`data`	The integer value stored in a `Node` in the linked list.
`pLeft`	Pointer to the left `Node` of the current node.
`pRight`	Pointer to the right `Node` of the current node.

Table 1. Members of `struct Node`.

Use `struct Node` in constructing the linked list implementation of your binary search tree.

Tasks

Task 1 – Design and implement a stack data structure

You are tasked to implement `bst` operations using linked list.

Open the file `bst.h`. The `struct bst` implements a `bst` using a linked list. You are not allowed to modify any part of the given code.

```
1  typedef struct {
2        sNode *pRoot;
3  } bst;
```

Figure 2. Code for the linked list implementation of a `bst` as defined in `bst.h`.

The members of `struct bst` are as follows:

Variable	Description
`pRoot`	A pointer pointing to the root node in the binary search tree.

Table 2. Members of the linked list implementation of a `bst` as defined in `bst.h`.

Complete the body of the functions listed below.

Function Prototype	Description
`bst* create();`	Returns an empty `bst`.
`void insert(bst *t, int x);`	Inserts a new node in the `bst`.
`sNode* search(bst *t, int x);`	Returns an `sNode` pointer to the node containing the value `x`. Returns `NULL` if the value `x` is not found in the `bst`.
`void inorder(sNode *pPointer);`	Prints the inorder traversal of the `bst` rooted by the node pointed to by `pPointer`.
`void preorder(sNode *pPointer);`	Prints the preorder traversal of the `bst` rooted by the node pointed to by `pPointer`.
`void postorder(sNode *pPointer);`	Prints the postorder traversal of the `bst` rooted by the node pointed to by `pPointer`.
`sNode* maximum(sNode *pPointer);`	Returns an `sNode` pointer to the node containing the largest value in the `bst` rooted by the node pointed to by `pPointer`.

`sNode* minimum(sNode *pPointer);`	Returns an `sNode` pointer to the node containing the smallest value in the `bst` rooted by the node pointed to by `pPointer`.
`sNode* parent(bst *t, int x);`	Returns an `sNode` pointer to the parent of the node containing the value `x`.
`sNode* successor(bst *t, int x);`	Returns an `sNode` pointer to the successor of the node containing the value `x`.
`sNode* predecessor(bst *t, int x);`	Returns an `sNode` pointer to the predecessor of the node containing the value `x`.

Table 3. Function prototypes of the linked list implementation of `bst` operations as defined in `bst.h`.

Task 2 – Documentation

Open the file `Documentation.docx`. Edit this file for the documentation of your work for this project.

- Examine the functions that you have implemented for Task 1.
- In the documentation file, provide the big-oh for each implemented function.
- In the documentation file, you also need to provide an implementation summary which will explain how you implemented the functions. Your implementation summary should also explain the worst-case time complexity of the function.

Required Program Interaction

Based on the previous sections of the project specifications, you do not need to create and to submit a `c` file which contains the `main()` function to test your header file. Thus, your submission should only include the header file and the documentation file. Submission of a `c` file containing a `main()` function will merit deductions.

You may create a `c` file containing a `main()` function to test your implementation, but you do not need to submit it.

Working With Groupmates

For this project, you are encouraged to work in groups of at most 3 members. 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.

Deliverables

Submit a `zip` file containing the source code files and a `PDF` file of the documentation via AnimoSpace. Do not include any executable file in your `zip` file submission.

Academic Honesty Policy

Honesty policy applies. 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.

The student handbook states that (Sec. 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."

The student handbook also states that (Sec. 10.3):

A student caught cheating, as defined in Sec. 5.3.1.1., shall be penalized with a grade of 0.0 in the requirement or in the course, at the discretion of the faculty member, without prejudice to an administrative sanction. In cases of alleged cheating, the faculty member should report the incident to the Student Discipline Formation Office (SDFO).

RUBRIC FOR GRADING

Criteria	Ratings			Points
Task 1 – create()	**COMPLETE** **1 pt** The function is implemented properly and completely.		**NO MARKS** **0 pt** The function is not implemented at all; or the function is implemented incorrectly.	1 pt
Task 1 – insert()	**COMPLETE** **5 pts** The function is implemented properly and completely.	**INCOMPLETE** **2 pt** The function is implemented incorrectly.	**NO MARKS** **0 pt** The function is not implemented at all.	5 pts
Task 1 – search()	**COMPLETE** **8 pts** The function is implemented properly and completely.	**INCOMPLETE** **4 pt** The function is implemented incorrectly.	**NO MARKS** **0 pt** The function is not implemented at all.	8 pts
Task 1 – inorder()	**COMPLETE** **7 pts** The function is implemented properly and completely.	**INCOMPLETE** **3 pt** The function is implemented incorrectly.	**NO MARKS** **0 pt** The function is not implemented at all.	7 pts
Task 1 – preorder()	**COMPLETE** **7 pts** The function is implemented properly and completely.	**INCOMPLETE** **3 pt** The function is implemented incorrectly.	**NO MARKS** **0 pt** The function is not implemented at all.	7 pts

Task 1 – postorder()	**COMPLETE** **7 pts** The function implemented properly and completely.	**INCOMPLETE** **3 pt** The function is implemented incorrectly.	**NO MARKS** **0 pt** The function is not implemented at all.	7 pts
Task 1 – maximum()	**COMPLETE** **3 pts** The function is implemented properly and completely.		**NO MARKS** **0 pt** The function is not implemented at all; or the function is implemented incorrectly.	3 pts
Task 1 – minimum()	**COMPLETE** **3 pts** The function is implemented properly and completely.		**NO MARKS** **0 pt** The function is not implemented at all; or the function is implemented incorrectly.	3 pts
Task 1 – parent()	**COMPLETE** **3 pts** The function is implemented properly and completely.		**NO MARKS** **0 pt** The function is not implemented at all; or the function is implemented incorrectly.	3 pts
Task 1 – successor()	**COMPLETE** **3 pts** The function is implemented properly and completely.		**NO MARKS** **0 pt** The function is not implemented at all; or the function is implemented incorrectly.	3 pts
Task 1 – predecessor()	**COMPLETE** **3 pts** The function is implemented properly and completely.		**NO MARKS** **0 pt** The function is not implemented at all; or the function is implemented incorrectly.	3 pts

Task 2 - Documentation	**COMPLETE** **15 pts** The documentation contains the correct big-oh and comprehensive implementation details for **all functions** implemented for Task 1.	**INCOMPLETE** **7 pts** The documentation contains the big-oh and implementation details for **some functions** implemented for Task 1. The documentation contains errors in the big-oh of some functions implemented for Task 1. Some parts of the documentation are based on incorrect implementation for Task 1.	**NO MARKS** **0 pt** The documentation does not contain any big-oh nor implementation details for any function for Task 1.	15 pts
			Total points:	65