Algorithms and Data Structures 2

Laboratory Sheet 7

Using the following declarations and function templates, create a Binary Search Tree of students.

```
struct student {
        char studentNumber [20];
        char name[20];
};

struct BSTNode {
        struct student *element;
        struct BSTNode *left;
        struct BSTNode *right;

};

struct BSTNode *root = NULL;
```

The main method will be as follows, and will call and execute all of the following functions (remember their signatures should be listed at the top of the program:-

```
int main()
{
        char number[20];

        addStudents ();
        displayInOrder(root);

        printf("Enter the student (via their number) you wish to find in the tree ");
        scanf("%s", number);
        if (search(number))
                printf("%s is stored in the BST\n", searchValue);
        else
                printf("%s is not stored in the BST\n", searchValue);

    removeMin();
    displayInOrder(root);

     removeMax();
    displayInOrder(root);


}//end main
```

```
void displayInOrder(struct BSTNode *current)
{
        //See code in  Trees lecture notes to traverse tree using inorder traversal
        //Values should be displayed in sorted order
}//end displayInOrder


 void addStudents ()  //adds 5 students to the tree, assume numbers are not duplicated
{
        struct student *anElement;
        struct BSTNode *aNode, *current;
        char number [20];
        char name[20];


 for (int i = 1; i <= 5; i++)
{
```

a. Create a new student element using malloc and store its address in *anElement*
b. Ask user to input a student number and name and store in *anElement*
c. Create a new node using malloc and store its address in *aNode*
d. Assign *anElement* to the *aNode's* data part
e. Assign null to the *aNode's* left and right pointers
f. If tree is empty then
    Let *aNode* be the root node
  Else //*need to search the tree for the correct position for new node*
    a. Assign *current* to *root*
    b. While *current* isn't null
        i. If *aNode's* student number < *current's* student number
            1. If *current* has no left child
                a. Assign *current's* left pointer to *aNode*
                b. Assign *current* to NULL
            2. Else
                a. Assign *current* to its left child
        ii. Else //*aNode's data > current's data*
            1. If *current* has no right child
                a. Assign *current's* right pointer to *aNode*
                b. Assign *current* to NULL
            2. Else
                a. Assign *current* to its right child
    c. End while
  }//*end for*
}//*end addStudents*

```
bool search(char *value)
	{
		struct BSTNode *aNode, *current;
		bool found = false;

		if (!isEmpty()) //Make sure you write an isEmpty() function also
		{
			current = root;
			while value hasn't been found and current isn't null
				if current's value equals value
					found = true
				else
					if current's value > value
						Assign current its left child
					Else
						Assign current to its right child
			End While
		}//end if
		return found;
	}//end search
```

removeMin() - Remove the smallest item in the tree. This will be the leftmost node so you will need to set up a loop to keep moving left). Set the left pointer of its parent to NULL and delete the leftmost node (by calling *free*).

removeMax() - Remove the largest item in the tree (this will be the rightmost item). Set the right pointer of its parent to NULL and delete the rightmost node (by calling *free*).