This exercise has total 6 points. Each point amounts to one percent of the total module mark. You are not allowed to use any external library in your code. We might ask you to explain your code.

All assessment deadlines in this module are strict. Late submissions will get a mark of 0. All submissions must work on the virtual machine as specified.

[Before starting this exercise, consider having a look at the linkedlist_basic.c source code file which is present inside Modules/Week2 on Canvas]

Your task is to implement a binary search tree for integers (int). In this exercise, we assume that there are **no duplicate nodes**. In bst.h, there is already a header file that declares the interface of that tree. You must implement the following functions for the tree operations in bst.c file. test_bst.c is a simple testbench that tests your code. You may edit this file to include more test cases.

Implement the following functions in the bst.c file.

1.

Node* insertNode(Node *root, int value);

This function is used to insert a new node in the tree as a leaf node. The member 'data' of the new node is assigned the input 'value'. During the insertion, a traversal starts from the root of the tree. The traversal follows the right branch if the value of the new node is greater than the value of the currently visited node. Otherwise the traversal follows the left branch. For this question we will restrict to the case that all data values are unique (that is to say there will be no duplicates) for simplicity.

2.

Node* deleteNode(Node *root, int value);

This function is used to delete an existing node in the tree. For this question we will restrict to the case that all data values in the tree are unique.

3.

void printSubtree(Node *N);

This function is for Inorder printing of the node values. It prints

i) the values in the left subtree

ii) then the value of the node and

iii) finally, the values in the right subtree.

Thus, the function prints the node values in sorted ascending format.

4.

```
int countNodes(Node *N);
```

This function returns the number of nodes in the subtree rooted at node N.

5.

```
Node* freeSubtree(Node *N);
```

This function deallocates the entire subtree rooted at N. Here you will need to free each node exactly once, so that there are neither double frees nor memory leaks. Note that the tree could be empty, so freeSubtree() should work, but do nothing, as there are no nodes to free. It returns the same pointer which it received.

Code submission

Use the Makefile to compile your code. There is a script in test.sh, that runs test_bst and compares the output with a reference output.

Use Valgrind to check memory leaks and errors.

Upload the bst.c file **ONLY**, any other form of submission will not be accepted.

Marking scheme: You get less points when your solution leaks memory, handles errors incorrectly or crashes for some inputs. Your code must compile on the virtual machine we had recommended.

- 1 point is given if insertNode() inserts correctly.
- 3 points are given if deleteNode() deletes correctly without causing memory leaks.
- 0.5 points are given if printSubtree() prints the subtree correctly (i.e in ascending order)
- 0.5 points are given if countNodes() returns the number of nodes correctly
- 1 point is given if freeSubtree() deallocates the subtree without causing memory leaks

We might ask you to explain your code.

Remarks

- Only return an error when malloc or a similar function fails. Do not set a hard limit for the memory in your code.
- Do not upload object files or binaries, just the C code.

- PrintSubtree should print one node value per line, for example:
  2
  3
  4
  5

  Not 2 3 4 5
- Any code which does not compile on the virtual machine using the Makefile provided will be awarded 0 marks and not be reviewed.
- You must not change the compiler options in the Makefile. These options imply that any warnings will be treated as errors, resulting in code that doesn't compile and thus will not be marked.
- For marking we will use additional, more advanced, test scripts which check whether your program satisfies the specification. If the provided test script fails, all the more advanced test scripts are likely to fail as well.
- There must be no memory leaks in your tree implementation.

Useful Links

Using make and writing Makefiles

https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html