CENG313 Fall 2019

PROGRAMMING ASSIGNMENT 2

Due date:**27.10.2019 23:55**

Write a C program that implements a “binary search tree and stack” data structures.

Your program begins by generating tasks. Task types and task ids are listed below.

task_id	task_type
1	insert,
2	delete,
3	minimum,
4	search.

Firstly, your program should create an initial binary search tree by adding some values. The number of value is up to you.

Then, your program should generate 10 random tasks. According to task, your program processes them one by one. If the task causes a change in the tree, current binary tree must be displayed according to in order traversal. The detailed task definitions are given below.

task id	task_definition
1	insert a value to binary search tree.(The value is up to you.)
2	delete a value from binary search tree.
3	find the minimum element in the binary search tree.
4	search a value in tree with no duplicates and show how many steps the value is found (the value is up to you).

All tasks that cause changes in the tree must be kept in the stack. For example; if the task is delete; the task_id and deleted value must be kept in the stack.

After the task generation completed, the changes made to the tree will be undone as specified by the user. If the user enter 3, program pops 3 modifications from stack and reverse them respectively. After each reverse task, the program must display the current tree according to in order traversal.

Your program must have 2 header files for stack and binary search tree. Your program should have the following data structures and C functions:

```
struct node{
    int data;
    struct node *left;
    struct node *right;
};

typedef struct node Node;
Node* createNode(int value);
void inorder(Node *root);
Node* insert(Node* root, int data);
int search(Node* root, int data);
Node * minValueNode(Node* node);
Node* deleteNode(Node* root, int data); Note: you don’t have to return any value for delete function.
```

```
struct stack {
    int top;
    int capacity;
    int* task;
    int* values;
};
typedef struct stack Stack;
Stack* createStack(int capacity);
int isFull(Stack* stack);
int isEmpty(Stack* stack);
void push(Stack* stack, int t, int item);Note: t refers task, item refers the value.
int pop(Stack* stack);
```

Your program should display the task generation and completion information. In the final phase, it should print the contents of the tree.

Example execution:

Initial tree
20 ->30 ->40 ->50 -> (note: in this example root is assumed as 50.)

task->search:40
searched 40 is found in 3 steps

task->search:43
searched 43 is not found

task->insert:72
the value:72 is inserted
20 ->30 ->40 ->50 ->72 ->

task->delete:30
the value:30 is deleted
20 ->40 ->50 ->72 ->

task->insert:43
the value:43 is inserted
20 ->40 ->43 ->50 ->72 ->

task->min
minval 20 is found

task→insert:12
the value:12 is inserted
12->20->40 ->43 ->50 ->72 ->

task->min
minval 12 is found

task->search:42
searched 42 is not found

task->search:50
searched 50 is found in 1 step

how many steps back:5
inserted value 12 is deleted.
20->40 ->43 ->50 ->72 ->
inserted value 43 is deleted.
20->40->50 ->72
deleted value 30 is inserted.
20 ->30 ->40->50 ->72->
inserted value 72 is deleted.
20 ->30 ->40->50 ->
There is no task to be undone.