Data Structures and Fundamentals of Programming

Problem #1

In C++ implement a generic **singly-linked-list** class, called `Stack<item>`, which uses dynamic memory allocation. `item` is the type of data stored in the **stack**. This should implement the stack ADT. The stack should look something like the following:

TOS -> X_1 ->X_2 -> … -> X_n

where X_1 is the node on the top of the stack and X_n is at the bottom of the stack, TOS is the Top of Stack pointer.

Along with the class definition(s), you must implement the following methods for `Stack`:

- `Stack()` - Default constructor
- `~Stack()` - Destructor
- `push(item)` –creates a new item and push to the stack
- `item pop()` – removes a node from the stack.
- `item getmin()` – return the minimum node inside the stack. Note that `MIN_ITEM` and `MAX_ITEM` are the possible minimum and maximum value of all the `items`, respectively. You can directly use an existing function `min(item1,item2)` to compute the minimal one of the two items `item1` and `item2`.

Note: Your implementation can **NOT** use STL or any other libraries (standard or otherwise).

Problem #2

A "binary search tree" (BST) is a type of binary tree where the nodes are arranged in order: for each node, all elements in its left subtree are less-or-equal to the node (<=), and all the elements in its right subtree are greater than he node (>).

Given a binary search tree. Each node is defined by a struct as

```
struct TreeNode
{
      Treenode *left;
      Treenode *right;
      ElementType data;
}
```

Work on the recursive programs for the following two questions:

1. Filling in the empty line to complete a function which finds a given element `item`

```
bool find(struct TreeNode* node, ElementType &item)
{
  if (node==NULL)
    return false;
  else
      {
      if (item = node->data)
            return true;
      else if (item < node->data)
            return ________________;
      else
            return ________________;
      }
```

2. Write a function `size()` to compute the total number of nodes in a BST tree.

```
int size(struct TreeNode* node)
{

}
```

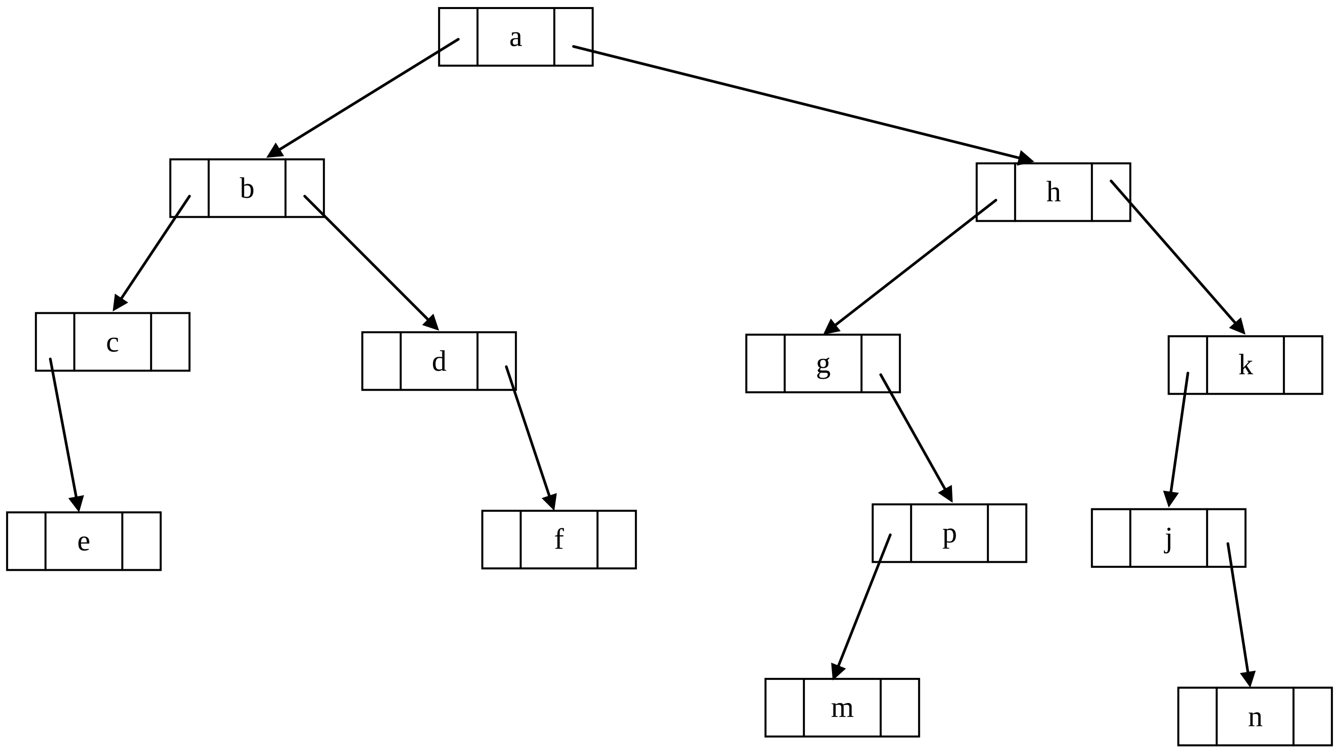
Problem #3

A) Convert the following infix expressions into postfix and prefix.

`a * b - c * d * e * (d – f) - g`

`a * (b + c) * (d - e) – d * f`

B) Give the Preorder, Postorder, and Inorder traversals of the tree below:


a
b
h
c
d
g
k
e
f
p
j
m
n