**Project2**

For solving the second project, you need first to decompose the problem into smaller tasks and next, identify how each task is solved. For doing so, you need to get the appropriate sequence of steps (that is, specific operation on specific data structure) to follow with your input data to build the correct tree, and next to use that tree to create the infix expression and the output sequence.

*General information:*
There are 5 types of operations performed on any kind of data structure:
- insert in the structure/ with the particular case of build the structure (which is not necessary done via insertions)
- delete from the structure
- update in the structure
- search in the structure
- traverse the structure.

Now, for the traversal, there are several strategies, depending on the structure.
Forward/backward if it comes to arrays and lists (from head to tail/tail to head respectively).
Preorder, inorder and postorder for the trees (if the tree is Left Node Right, the traversals are:
preorder = Node Left Right
inorder = Left Node Right
postorder = Left Right Node.

*The tasks of problem are:*
1. **Task1**: *building* the binary tree from the input array containing the postfix expression (so, the *Construct Tree* mentioned in the requirements is a build tree operation); *HINT*: to have a good image of the problem, draw (pencil and paper) the tree from the expression in the example; if not enough, draw several other trees; once you have the view of an actual tree, you should properly identify how to build it (*hint* in the hint: what's the content of the root? Where from can you get that information? What's the left sub-tree? The right one?);
2. **Task2:** *build* the infix expression (so, it would be another array created out of a tree; that tree needs traversed; which order, this is the aspect you should identify (shouldn't be difficult));
3. **Task3:** from the expression built in the previous step, *generate* the 3 address instructions sequence. HINT: to do so, the approach in the first project should help; it's just that instead of evaluating each subexpression, you should write in a file the appropriate sequence.