CSD 311

ASSIGNMENT 3

Documentation - PROLOG

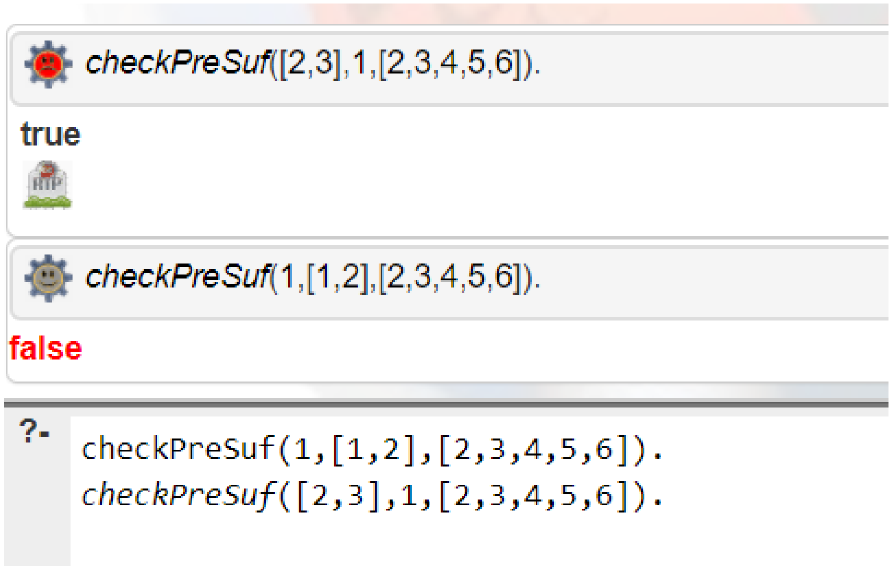
1. **Prefix and Suffix Checker in PROLOG -**

   1.1. CODE -

```
appendList([], FinalList, FinalList).
appendList([Start|EndList], FinalList, [Start|EndFinal]) :- appendList(EndList, FinalList, EndFinal).

prefixChecker(PrefixList, ParentList) :- appendList(PrefixList, _, ParentList).
suffixChecker(SuffixList, ParentList) :- appendList(_, SuffixList, ParentList).


checkPreSuf(PrefixList, 1, ParentList ) :- prefixChecker(PrefixList, ParentList). %% This format is used for Checking prefix only
checkPreSuf(1, SuffixList, ParentList) :- suffixChecker(SuffixList, ParentList). %% This format is used for checking suffix only
checkPreSuf(PrefixList, SuffixList, ParentList) :- prefixChecker(PrefixList, ParentList) , suffixChecker(SuffixList, ParentList).
```

   1.2. Test Case -

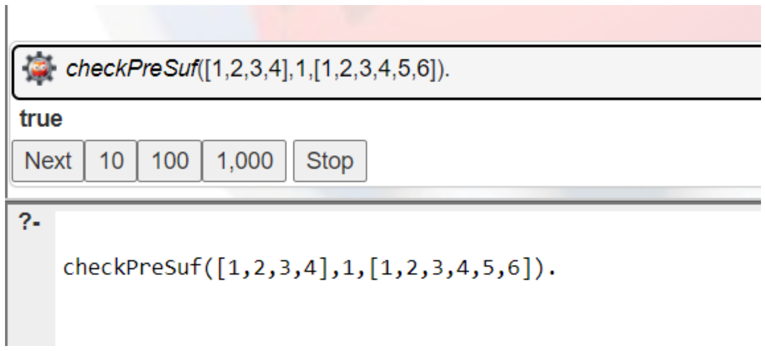
   1.2.1. Test Case provided on the submission link -



```
checkPreSuf([2,3],1,[2,3,4,5,6]).
true

checkPreSuf(1,[1,2],[2,3,4,5,6]).
false

?- checkPreSuf(1,[1,2],[2,3,4,5,6]).
   checkPreSuf([2,3],1,[2,3,4,5,6]).
```

   1.2.2. Test Case given by us -

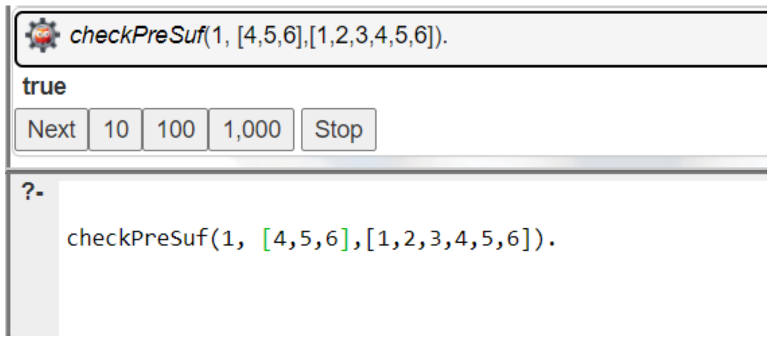
   For checking prefix -



```
checkPreSuf([1,2,3,4],1,[1,2,3,4,5,6]).
true
```

```
?- checkPreSuf([1,2,3,4],1,[1,2,3,4,5,6]).
```

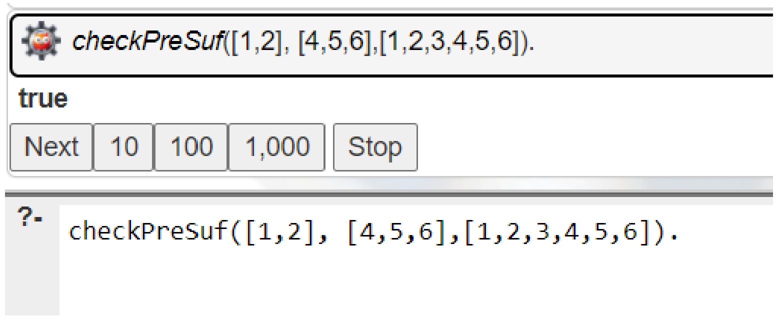
For checking Suffix -



```
checkPreSuf(1, [4,5,6],[1,2,3,4,5,6]).
true
```

```
?- checkPreSuf(1, [4,5,6],[1,2,3,4,5,6]).
```

For checking both -



```
checkPreSuf([1,2], [4,5,6],[1,2,3,4,5,6]).
true
```

```
?- checkPreSuf([1,2], [4,5,6],[1,2,3,4,5,6]).
```

2. Binary Search Tree -

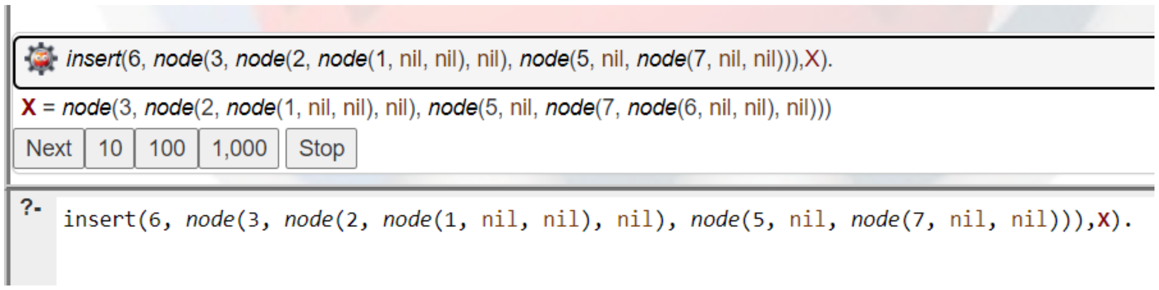
2.1. Code -

```
%% this is used to individually insert an element into a Binary Search Tree
%%  Base Case
insert(Root, nil, node(Root,nil,nil)).
insert(Ele, node(Key, L, R), node(Key1, L1, R1)) :- Ele < Key ->  insert( Ele, L, U), (L1, Key1, R1) = (U, Key, R).
insert(Ele, node(Key, L, R), node(Key1, L1, R1)) :- Ele > Key ->  insert(Ele, R, U), (L1, Key1, R1) = (L, Key, U) ; (L1, Key1, R1) = (L, Key, R).

createTree([Head|Tail], T0, T) :-insert(Head, T0, T1), createTree(Tail, T1, T).
createTree([], T, T). %% Base Case

%% used to take a list and create a Binary Search Tree
%% Construct uses createtree and createTree uses insert to insert an element into the tree.
construct(X, T) :- createTree(X,nil,T).


swapNodes(nil,nil).
swapNodes(node(Head, L1, R1), node(Head, R, L)) :- swapNodes(L1,L), swapNodes(R1,R).

%% Mirror uses construct and swap nodes function to create mirror of the given Binary Search Tree
mirror(X, Tree) :- construct(X,T1), swapNodes(T1, Tree).
```

2.2. Test Case -

2.2.1. Test Case -

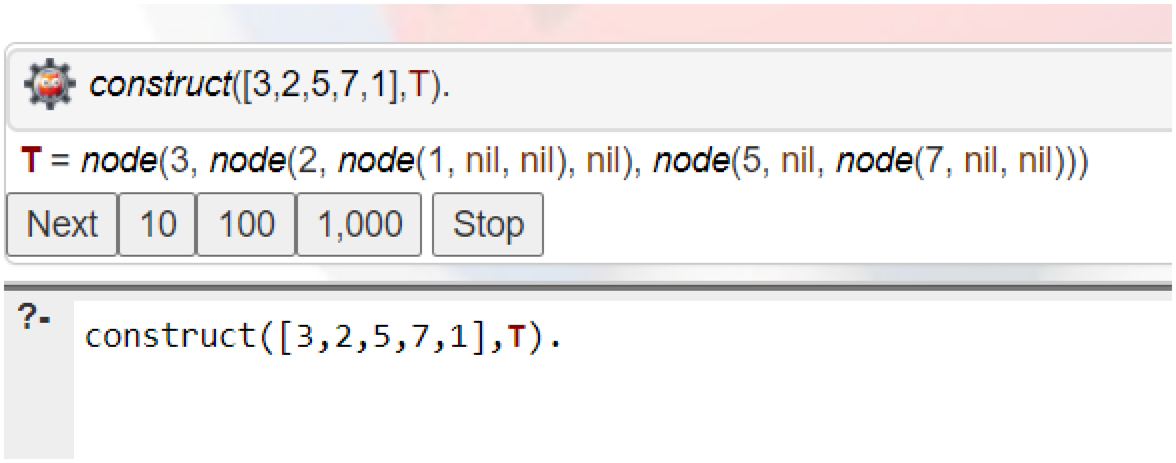
Insert -



```
insert(6, node(3, node(2, node(1, nil, nil), nil), node(5, nil, node(7, nil, nil))),X).
X = node(3, node(2, node(1, nil, nil), nil), node(5, nil, node(7, node(6, nil, nil), nil)))
```

```
?- insert(6, node(3, node(2, node(1, nil, nil), nil), node(5, nil, node(7, nil, nil))),X).
```

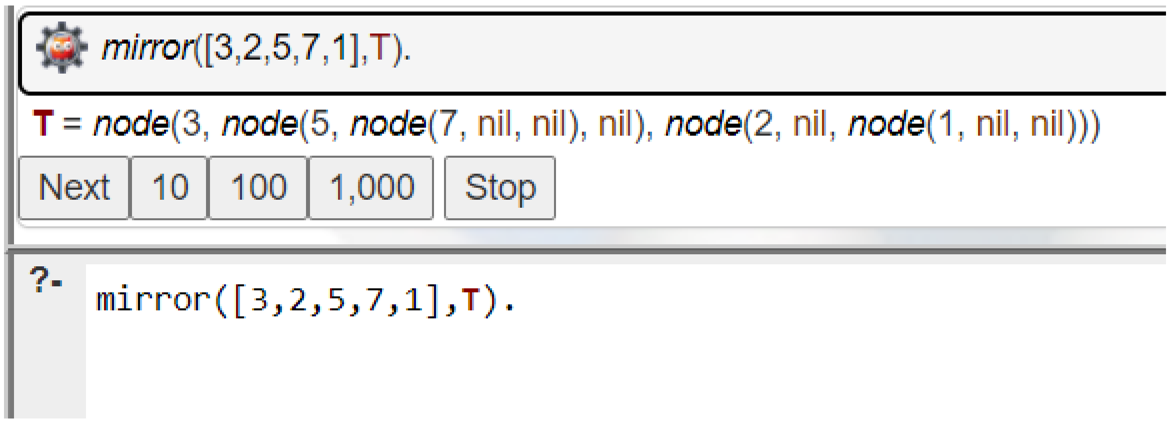
Construct -



```
construct([3,2,5,7,1],T).
T = node(3, node(2, node(1, nil, nil), nil), node(5, nil, node(7, nil, nil)))
```

```
?- construct([3,2,5,7,1],T).
```

Mirror -



```
mirror([3,2,5,7,1],T).
T = node(3, node(5, node(7, nil, nil), nil), node(2, nil, node(1, nil, nil)))
```

```
?- mirror([3,2,5,7,1],T).
```

3. Contributors -

Aman Yadav	Aryaman Singh Rana