ASSIGNMENT NO:2 **DATE: / /2015**

PROGRAM TITLE:Create a Binary Tree given its Inorder and Preorder sequences.

PROGRAM ALGORITHM:

```
pi=1;      //it is a global variable that keeps track of the length of the
preorder array that has been evaluated.

convert(pre[],in[],root,li,ri)
{
      //pre[] is the preorder array, in[] is the inorder array, root is the tree
(or subtree) root, li and ri signifies the left and right indexes of the
inorder subtree we are currenly working on.

      create temp node;
      if(li is equal to ri)      //i.e we have reached the leaf node with no
children
      {
            data(temp)=pre[pi];
            increase pi by 1;
            left(temp)=NULL;
            right(temp)=NULL;
            set root as temp;     //root is currently a location, changing this
also links the temp node to its parent
      }
      else
      {
            data=pre[pi];
            for(i=li to ri)
            {
                  if(in[i]==data)
                        break;     //i.e  we have identified the left subtree of
the current root
            }
            data(temp)=pre[pi];
            left(temp)=NULL;
            right(temp)=NULL;
            set root as temp;
            increase pi by 1;
            if(there are elements in the left subtree)
                  convert(pre,in,left(root),li,i-1);
            if(there are elements in the right subtree)
                  convert(pre,in,right(root),i+1,ri);
      }
}
```

PROGRAM CODE:

```
//C Program to Create a Binary Tree from Infix and Prefix Sequences
#include <stdio.h>
#include <stdlib.h>
struct node
{
      int data;
```

```
      struct node *left;
      struct node *right;
};
int pi=0;

/*The recursive function that forms a tree from the two sequences*/
int convert(int *pre,int *in,struct node**root,int li,int ri)
{
      struct node *temp=(struct node *)malloc(sizeof(struct node));
      if(li==ri)
      {
            temp->data=pre[pi];
            pi=pi+1;
            temp->left=NULL;
            temp->right=NULL;
            *root=temp;
      }
      else
      {
            int data,i;
            data=pre[pi];
            for(i=li;i<=ri;i++)
            {
                  if(in[i]==data)
                        break;
            }
            temp->data=pre[pi];;
            temp->left=NULL;
            temp->right=NULL;
            *root=temp;
            pi++;
            if(li<=i-1)
                  convert(pre,in,&(*root)->left,li,i-1);
            if(ri>=i+1)
                  convert(pre,in,&(*root)->right,i+1,ri);
      }
      return 1;
}
int display(struct node **root,int level)
{
      int i=0;
      if(*root==NULL)
            return 0;
      display(&(*root)->right,level+1);
      for(i=0;i<=level;i++)
            printf("\t");
      printf("%d\n",(*root)->data);
      display(&(*root)->left,level+1);
      return 1;
}
int main()
{
      int n,i=0;
      printf("\n\tEnter number of nodes:");
      scanf("%d",&n);
      int *in=(int*)malloc(n*sizeof(int));
      int *pre=(int*)malloc(n*sizeof(int));
```

```
	printf("\n\tEnter Inorder Sequence::\n\t");
	while(i<n)
	{
		scanf("%d",&in[i++]);
		printf("\t");
	}
	i=0;
	printf("\n\tEnter Preorder Sequence::\n\t");
	while(i<n)
	{
		scanf("%d",&pre[i++]);
		printf("\t");
	}
	struct node* root=NULL;
	convert(pre,in,&root,0,n-1);
	printf("\n\tThe Tree is::\n");
	display(&root,0);
	return 0;
}
```

OUTPUT:

```
	Enter number of nodes:9

	Enter Inorder Sequence::
	4 7 2 8 5 1 6 9 3

	Enter Preorder Sequence::
	1 2 4 7 5 8 3 6 9

	The Tree is::
		3
				9
			6
	1
			5
				8
		2
				7
			4
```

DISCUSSION:

1. Inorder sequence follows Left-Root-Right traversal.
2. Preorder sequence follows Root-Left-Right traversal.
3. Thus, we always get the root of the tree first by the Preorder Traversal. Using this, we can identify the left and right subtrees of the root from the Inorder sequence. Doing this recursively gives us a unique tree from an Inorder and Preorder sequence of a Binary tree.
4. The code is so designed that it stops exactly after a complete traversal of the preorder sequence.
5. The code believes that the user will enter no invalid inputs.