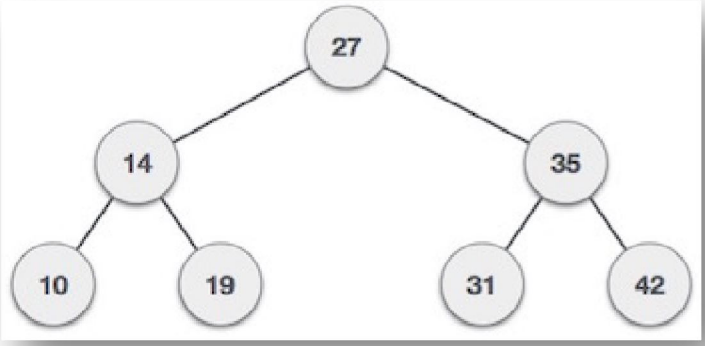
Course Name: A Level (2nd Sem) **Subject:** Data Structure using C++
Topic: Coding for Binary tree traversal methods **Date:** 14-05-2020

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot random access a node in a tree. There are three ways which we use to traverse a tree –

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

We shall now look at the implementation of tree traversal in C programming language here using the following binary tree –




```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *leftChild;
   struct node *rightChild;
};
struct node *root = NULL;
void insert(int data)
{
   struct node *tempNode = (struct node*) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;
   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;
   if(root == NULL)
  {
      root = tempNode;
   }
   else
  {
      current = root;
      parent = NULL;
   while(1)
  {
```

```
        parent = current;
        if(data < parent->data)
       {
          current = current->leftChild;
          if(current == NULL)
          {
            parent->leftChild = tempNode;
            return;
          }
        }
        else
       {
          current = current->rightChild;
          if(current == NULL)
         {
            parent->rightChild = tempNode;
            return;
          }
        }
      }
   }
}

struct node* search(int data)
{
   struct node *current = root;
   printf("Visiting elements: ");
   while(current->data != data)
    {
      if(current != NULL)
         printf("%d ",current->data);
      if(current->data > data)
      {
         current = current->leftChild;
      }
      else
     {
         current = current->rightChild;
      }
      if(current == NULL)
      {
         return NULL;
      }
   }
   return current;
}

void pre_order_traversal(struct node* root)
{
   if(root != NULL)
   {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}
```

```
void inorder_traversal(struct node* root)
{
  if(root != NULL)
  {
    inorder_traversal(root->leftChild);
    printf("%d ",root->data);
    inorder_traversal(root->rightChild);
  }
}

void post_order_traversal(struct node* root)
{
  if(root != NULL)
  {
    post_order_traversal(root->leftChild);
    post_order_traversal(root->rightChild);
    printf("%d ", root->data);
  }
}

void main()
{
  int i;
  int array[7] = { 27, 14, 35, 10, 19, 31, 42 };
  for(i = 0; i < 7; i++)
  insert(array[i]);
  i=31;
  struct node * temp = search(i);
  if(temp != NULL)
  {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }
  else
 {
    printf("[ x ] Element not found (%d).\n", i);
  }
  i=15;
  temp = search(i);
  if(temp != NULL)
  {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }
  else
 {
    printf("[ x ] Element not found (%d).\n", i);
  }
  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);
```

```
   return 0;
}
```

If we compile and run the above program, it will produce the following result −

Output

```
Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27
```