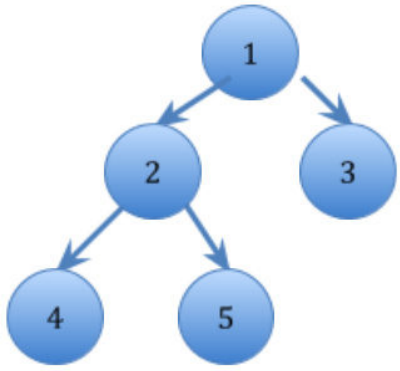
BCSL-033 solved assignment july 2017 – january 2018 session

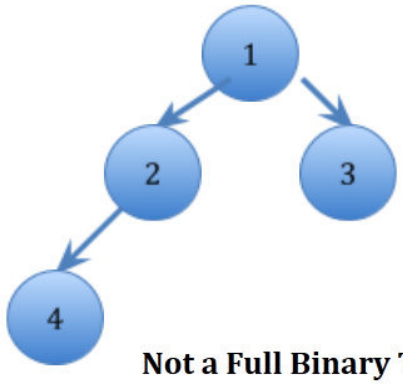
1. Write an algorithm and program that accepts a Binary Tree as Input and Checks if the input Binary tree is Complete Binary Tree or a Full Binary Tree and prints appropriate message.

Ans 1.

A full binary tree is defined as a binary tree in which all nodes have either zero or two child nodes. Conversely, there is no node in a full binary tree, which has one child node. For Example:



Full Binary Tree



Not a Full Binary Tree

Algorithm:-

1) If a binary tree node is NULL then it is a full binary tree.

2) If a binary tree node does have empty left and right sub-trees, then it is a full binary tree by definition

3) If a binary tree node has left and right sub-trees, then it is a part of a full binary tree by definition. In this case recursively check if the left and right sub-trees are also binary trees themselves.

4) In all other combinations of right and left sub-trees, the binary tree is not a full binary tree.

Program:-

```
// C program to check whether a given Binary Tree is full or not

#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

/* Tree node structure */
struct Node
{
 int key;
 struct Node *left, *right;
};

/* Helper function that allocates a new node with the
given key and NULL left and right pointer. */
struct Node *newNode(char k)
{
 struct Node *node = (struct Node*)malloc(sizeof(struct Node));
 node->key = k;
 node->right = node->left = NULL;
 return node;
}

/* This function tests if a binary tree is a full binary tree. */
bool isFullTree (struct Node* root)
{
 // If empty tree
 if (root == NULL)
  return true;

 // If leaf node
 if (root->left == NULL && root->right == NULL)
  return true;

 // If both left and right are not NULL, and left & right subtrees
 // are full
 if ((root->left) && (root->right))
  return (isFullTree(root->left) && isFullTree(root->right));

 // We reach here when none of the above if conditions work
 return false;
}

// Driver Program
```

```
int main()
{
 struct Node* root = NULL;
 root = newNode(10);
 root->left = newNode(20);
 root->right = newNode(30);

 root->left->right = newNode(40);
 root->left->left = newNode(50);
 root->right->left = newNode(60);
 root->right->right = newNode(70);

 root->left->left->left = newNode(80);
 root->left->left->right = newNode(90);
 root->left->right->left = newNode(80);
 root->left->right->right = newNode(90);
 root->right->left->left = newNode(80);
 root->right->left->right = newNode(90);
 root->right->right->left = newNode(80);
 root->right->right->right = newNode(90);

 if (isFullTree(root))
  printf("The Binary Tree is full\n");
 else
  printf("The Binary Tree is not full\n");

 return(0);

}
```

2. Write an algorithm and program for implementation of multiple stacks in an Array.

Ans 2.

```
#include <stdio.h>
#define max 10
int top1, top2, stk_arr[max];
void push();
void pop();
void display();
void main()
{
int ch;
top1=-1,top2=max;
do
{
printf("\n 1:push\n 2:pop\n 3:display\n 4:exit\n choice:");
scanf("%d", &ch);
```

```
switch (ch)
{
case 1:push();
break;
case 2:pop();
break;
case 3:display();
break;
case 4:printf("exiting from program\n");
break;
default:printf("wrong choice\n");
break;
}
}while(ch!=4);
}
void push()
{
int x,ch;
if(top1==top2-1)
{
printf("stack overflow \n");
return;
}
printf("enter a no \n");
scanf("%d",&x);
printf("\n press 1 to push in stack1 or press 2 for stack2:");
scanf("%d",&ch);
if(ch==1)
stk_arr[++top1]=x;
else
stk_arr[--top2]=x;
printf("\n %d element is successfully pushed \n",x);
return;
}
void pop()
{
int y,ch;
printf("\n press 1 to pop from stack1 or press 2 for stack2″);
scanf("%d",&ch);
if(ch==1)
{
if(top1==-1)
{
printf("stack underflow\n");
return;
}
y=stk_arr[top1];
stk_arr[top1--]=0;
}
else
{
if(top2==max)
{
```

```
printf(“stack underflow\n”);
return;
}
y=stk_arr[top2];
stk_arr[top2++]=0;
}
printf(“\n%d element is successfully poped from stack \n”, y);
return;
}
void display()
{
int i;
if (top1 == -1)
{
printf(“stack 1 is empty \n”);
}
else
{
printf(“elements of Stack 1 are : \n”);
for (i = 0; i <= top1; i++)
{
printf(“%d\n”,stk_arr[i]);
}
}
if (top2 == max)
{
printf(“stack 2 is empty \n”);
}
else
{
printf(“elements of Stack 2 are : \n”);
for (i = max; i >= top2; i–)
{
printf(“%d\n”,stk_arr[i]);
}
}
return ;
}
```