ASSIGNMENT NO:1 **DATE: / /2015**

PROGRAM TITLE:Create a Binary Tree. Insert elements into it using Breadth First Search and Depth First Search.

PROGRAM ALGORITHM:

```
insert(root,item)
{
      if(root is NULL)
            set root;
      else
            ask for the parent;
            search for the parent via BFS or DFS as told;
            if(parent found)
                  ask if item to be left or right child;
                  if(that child is empty)
                        insert at that position;
}
bfs(root,n)
{
      insert root in queue;
      while(queue is not empty)
      {
            insert the left child in the queue;
            Insert the right child in the queue;
            if(data to be searched present in the current node)
                  return that node;
            else
                  delete that node from the queue;
      }
      return NULL;
}
dfs(root,n)
{
      if(data(root) is n)
            return root;
      else
      {
            if(left(root)!=NULL)
                  temp1= dfs(left(root),n);
            if(root->right!=NULL)
                  temp2= dfs(right(root),n);
            if(temp1!=NULL)
                  return temp1;
            if(temp2!=NULL)
                  return temp2;
            return NULL;
      }
}
```

PROGRAM CODE:

```
//C Program to Implement Insertion into a Binary Tree using BFS and DFS methods
#include <stdio.h>
#include <stdlib.h>
```

```
/*Node of the tree*/
struct node
{
      int data;
      struct node *left;
      struct node *right;
};

/*Node for the queue required to implement the BFS*/
struct qnode
{
      struct node *data;
      struct qnode *next;
};
int insert(struct node **root,int item);//Function for insertion into the tree
int display(struct node **root,int level);//Function for displaying as a tree
struct node* dfs(struct node *root,int n);//Function for finding the parent via
depth first search
int notempty(struct qnode *front);//Function to check if the queue for BFS is
empty or not
int queuein(struct qnode **rear,struct node *inp);//Function to Insert a node
into the queue
int queuedel(struct qnode **front);//Function to Delete a node from the queue
struct node* bfs(struct node *root,int n);//Function for finding the parent via
breadth first search
int main()
{
      int ch=1,n;
      struct node* root=NULL;
      while (ch!=0)
      {

            /*Prints the Main Menu*/
            printf("\n\tMENU::\n\t1.Insert\n\t2.Display as
Tree\n\t0.Exit\n\tEnter choice::");
            scanf("%d",&ch);
            switch(ch)
            {
            case 1:    printf("\n\tEnter the element::");
                  scanf("%d",&n);
                  insert(&root,n);
                  break;
            case 2: printf("\n");
                  display(&root,0);
                  break;
            case 0: return 0;
                  break;
            default:printf("\n\tInvalid Choice");
            }
      }
      return 0;
}
int insert(struct node **root,int item)
{
      if(*root==NULL)
      {
```

```
            struct node *temp=(struct node *)malloc(sizeof(struct node));
            temp->data=item;
            temp->left=NULL;
            temp->right=NULL;
            *root=temp;
            printf("\n\tRoot Set\n");
      }
      else
      {
            struct node *temp=(struct node *)malloc(sizeof(struct node));
            temp->data=item;
            temp->left=NULL;
            temp->right=NULL;
            int n,ch;
            struct node *par=NULL;
            printf("\n\tDeclare the parent of the node::");
            scanf("%d",&n);
            printf("\n\tSpecify how to you want to search for the
parent:\n\t1.BFS\tor\t2.DFS\n\tEnter Choice::");
            scanf("%d",&ch);
            if(ch==1)
                  par=bfs(*root,n);
            else if(ch==2)
                  par=dfs(*root,n);
            else
            {
                  printf("\n\tInvalid choice.Exiting Insertion.");
                  return 1;
            }
            if(par==NULL)
            {
                  printf("\n\tItem not found in tree.");
                  return 2;
            }
            printf("\n\tEnter 1 if left child, 2 if right child:");
            scanf("%d",&ch);
            if(ch==1)
            {
                  if(par->left==NULL)
                        par->left=temp;
                  else
                        printf("\n\tChild already present.");
            }
            else if(ch==2)
            {
                  if(par->right==NULL)
                        par->right=temp;
                  else
                        printf("\n\tChild already present.");
            }
            else
            {
                  printf("\n\tInvalid choice.Exiting Insertion.");
                  return 3;
            }
      }
      return 0;
```

```
}
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;
}

/*Searches for the parent using recursive DFS*/
struct node* dfs(struct node *root,int n)
{
      if(root->data==n)
            return root;
      else
      {
            struct node*temp1=NULL,*temp2=NULL;
            if(root->left!=NULL)
                  temp1= dfs(root->left,n);
            if(root->right!=NULL)
                  temp2= dfs(root->right,n);
            if(temp1!=NULL)
                  return temp1;
            if(temp2!=NULL)
                  return temp2;
            return NULL;
      }
}
int notempty(struct qnode *front)
{
      if(front==NULL)
            return 0;
      else
            return 1;
}

/*Inserts the node into the queue*/
int queuein(struct qnode **rear,struct node *inp)
{

      /*Checks if the element sent is equal to NULL or not. Inserts only if non-
NULL*/
      if(inp!=NULL)
      {
            struct qnode *temp=(struct qnode *)malloc(sizeof(struct qnode));
            temp->data=inp;
            temp->next=NULL;
            (*rear)->next=temp;
            *rear=temp;
      }
      return 0;
}
```

```
int queuedel(struct qnode **front)
{
      if(notempty(*front))
      {
            struct qnode *p=*front;
            (*front)=(*front)->next;
            free(p);
            p=NULL;
      }
      return 0;
}
struct node* bfs(struct node *root,int n)
{
      struct qnode *front=(struct qnode *)malloc(sizeof(struct qnode));
      front->data=root;
      front->next=NULL;
      struct qnode *rear=front;
      while(notempty(front))
      {

            /*Inserts the left child*/
            queuein(&rear,(front->data)->left);

            /*Inserts the right child*/
            queuein(&rear,(front->data)->right);

            /*Searches if the present node is the reqd parent*/
            if((front->data)->data==n)
                  return (front->data);

            /*Deletes the present node if not present*/
            queuedel(&front);
      }
      return NULL;
}
```

OUTPUT:

```
      MENU::
      1.Insert
      2.Display as Tree
      0.Exit
      Enter choice::1

      Enter the element::25

      Root Set

      MENU::
      1.Insert
      2.Display as Tree
      0.Exit
      Enter choice::1

      Enter the element::26
```

```
Declare the parent of the node::20

Specify how to you want to search for the parent:
1.BFS or   2.DFS
Enter Choice::1

Item not found in tree.
MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::1

Enter the element::24

Declare the parent of the node::25

Specify how to you want to search for the parent:
1.BFS or   2.DFS
Enter Choice::2

Enter 1 if left child, 2 if right child:1

MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::2

25
     24

MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::1

Enter the element::25

Declare the parent of the node::26

Specify how to you want to search for the parent:
1.BFS or   2.DFS
Enter Choice::2

Item not found in tree.
MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::1

Enter the element::65

Declare the parent of the node::25
```

```
Specify how to you want to search for the parent:
1.BFS or   2.DFS
Enter Choice::2

Enter 1 if left child, 2 if right child:2

MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::1

Enter the element::45

Declare the parent of the node::24

Specify how to you want to search for the parent:
1.BFS or   2.DFS
Enter Choice::1

Enter 1 if left child, 2 if right child:2

MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::2

      65
25
            45
      24

MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::1

Enter the element::54

Declare the parent of the node::24

Specify how to you want to search for the parent:
1.BFS or   2.DFS
Enter Choice::2

Enter 1 if left child, 2 if right child:2

Child already present.
MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::2

      65
```

```
25
          45
     24

MENU::
1.Insert
2.Display as Tree
0.Exit
Enter choice::0
```

DISCUSSION:

1. For Depth First Search, Stack principle is used, therefore, we can represent it using a simple recursive function.
2. For Breadth First Search, principle of Queue is used, therefore, we had to design some more functions to maintain and perform operations on the queue.
3. Both the BFS and DFS return the position of the parent if found, otherwise it returns NULL.