Assignment 1

Problem Statement : Create a Binary Tree and perform following operations:

a. Create
b. Display (using all three traversals with recursion)
c. Depth of a tree (non recursion)
d. Create a copy of a tree (non recursion)
e. Display leaf nodes (non recursion)
f. Insert

Code :

```
#include<iostream>
using namespace std;

struct btree
{
  int data;
  btree *left;
  btree *right;
};

class stack
{
  public:
    btree *st[20];
    int data,top;
  public:
    stack()
    {
      top = -1;
    }
    int isEmpty()
    {
      if(top==-1)
        return 1;
      else
        return 0;
    }
    void push(btree *nwnode)
    {
      top++;
      st[top] = nwnode;
    }
    btree *pop()
    {
      btree *nwnode;
      nwnode = st[top];
      top--;
      return(nwnode);
    }
};

class queue
{
  btree *que[20];
  int data,rear,front;
  public:
    queue()
    {
      rear = front = -1;
    }
    int isEmpty()
    {
      if(rear==front)
        return 1;
      else
        return 0;
    }
    int isFull()
    {
      if(rear==20)
        return 1;
      else
        return 0;
    }
    void add(btree *nwnode)
    {
      if(isFull())
        cout<<"\nQueue Overflow";
      else
      {
        rear++;
        que[rear] = nwnode;
      }
    }
    btree *del()
    {
      btree *nwnode;
      if(isEmpty())
      {
        cout<<"\nQueue is Empty";
      }
      else
      {
        front++;
        nwnode = que[front];
        return(nwnode);
      }
    }
};

class tree
{
  private:
    int z=1;
  public:
    btree *root = NULL;
    btree *copy = NULL;
    btree* create(btree *);
    void insert();
    void preorder(btree *);
    void inorder(btree *);
    void postorder(btree *);
    void bfs(btree *);
    void dfs(btree *);
    void display_traversals();
    btree* treecopy(btree *);
    void leaf_nodes(btree *);
};

void tree::insert()
{
  int n;
  cout<<"\nEnter number of nodes : ";
  cin>>n;
  for(int i=0;i<n;i++)
  {
    root = create(root);
  }
}

btree* tree::create(btree *root)
{
  int data;
  char ch;
  btree *temp = new btree;
  if(root==NULL)
  {
    root = new btree;
    cout<<"\nYou are at Level 0";
    cout<<"\nEnter the value of root : ";
    cin>>data;
    root->data=data;
    root->left = root->right = NULL;
  }
  else
  {
    temp = root;
    cout<<"\nYou are at Level "<<z;
    cout<<"\nWhere do you want to insert (l/r) : ";
    cin>>ch;
    if(ch=='l')
    {
      if(temp->left!=NULL)
      {
        z++;
        create(temp->left);
        z--;
      }
      else
      {
        cout<<"\nEnter the value of node : ";
        cin>>data;
        temp->left = new btree;
        temp = temp->left;
        temp->data=data;
        temp->left=temp->right=NULL;
      }
    }
    else if(ch=='r')
    {
      if(temp->right!=NULL)
      {
        z++;
        create(temp->right);
        z--;
      }
      else
      {
        cout<<"\nEnter the value of node : ";
        cin>>data;
        temp->right = new btree;
        temp = temp->right;
        temp->data=data;
        temp->left=temp->right=NULL;
      }
    }
    return root;
  }
}
```

```
void tree::preorder(btree *nwnode)
{
  if(nwnode!=NULL)
  {
    cout<<nwnode->data<<" ";
    preorder(nwnode->left);
    preorder(nwnode->right);
  }
}

void tree::inorder(btree *nwnode)
{
  if(nwnode!=NULL)
  {
    inorder(nwnode->left);
    cout<<nwnode->data<<" ";
    inorder(nwnode->right);
  }
}

void tree::postorder(btree *nwnode)
{
  if(nwnode!=NULL)
  {
    postorder(nwnode->left);
    postorder(nwnode->right);
    cout<<nwnode->data<<" ";
  }
}

void tree::bfs(btree *nwnode)
{
  queue Q;
  while(1)
  {
    cout<<nwnode->data<<" ";
    if(nwnode->left!=NULL)
      Q.add(nwnode->left);
    if(nwnode->right!=NULL)
      Q.add(nwnode->right);
    if(Q.isEmpty())
      break;
    nwnode = Q.del();
  }
}

void tree::dfs(btree *nwnode)
{
  stack S;
  while(1)
  {
    cout<<nwnode->data<<" ";
    if(nwnode->right!=NULL)
      S.push(nwnode->right);
    if(nwnode->left!=NULL)
      S.push(nwnode->left);
    if(S.isEmpty())
      break;
    nwnode = S.pop();
  }
}

void tree::display_traversals()
```

```
{
  cout<<"\n--------------------------------------------------------------------------------";
  cout<<"\nPreorder Traversal : ";
  preorder(root);
  cout<<"\n";
  cout<<"\nInorder Traversal : ";
  inorder(root);
  cout<<"\n";
  cout<<"\nPostorder Traversal : ";
  postorder(root);
  cout<<"\n";
  cout<<"\nBreadth-First Traversal : ";
  bfs(root);
  cout<<"\n";
  cout<<"\nDepth-First Traversal : ";
  dfs(root);
  cout<<"\n";
}

btree* tree::treecopy(btree *nwnode)
{
  stack S,S1;
  btree *tmp = nwnode;
  btree *tmp1 = new btree;
  btree *clone = new btree;
  clone->data = nwnode->data;
  while(1)
  {
    while(tmp!=NULL)
    {
      S.push(tmp);
      S1.push(tmp);
      tmp1 = S1.pop();
      clone = new btree;
      clone->data = tmp1->data;
      S1.push(clone);
      tmp = tmp->left;
      clone = clone->left;
    }
    if(S.isEmpty())
      break;
    tmp = S.pop();
    clone = S1.pop();
    tmp = tmp->right;
    clone = clone->right;
  }
  cout<<"\nCopied Successfully";
  return nwnode;
}

void tree::leaf_nodes(btree *nwnode)
{
  stack S;
  cout<<"\nLeaf nodes : ";
  while(1)
  {
    if(nwnode->left==NULL && nwnode->right==NULL)
      cout<<nwnode->data<<" ";
    if(nwnode->right!=NULL)
      S.push(nwnode->right);
    if(nwnode->left!=NULL)
      S.push(nwnode->left);
    if(S.isEmpty())
      break;
    nwnode = S.pop();
  }
}

int main()
{
  tree obj;
  btree *root1;
  int choice,ch;
  cout<<"\n1. Create a Binary Tree \n2. Exit"<<endl;
  cout<<"\nEnter your choice : ";
  cin>>choice;
  cout<<"\n----------------------------------------------------------------------";
  if(choice==1)
  {
    obj.insert();
    cout<<"\nTree Created Successfully"<<endl;
    while(1)
    {
      cout<<"\n----------------------------------------------------------------------";
      cout<<"\n1. Insert node \n2. Display Traversals \n3. Copy of a Tree \n4. Display Leaf nodes \n5. Exit";
      cout<<"\nEnter your choice : ";
      cin>>ch;
      cout<<"\n----------------------------------------------------------------------";
      if(ch==1)
      {
        obj.insert();
        cout<<"\nNode Inserted Successfully";
      }
      else if(ch==2)
        obj.display_traversals();
      else if(ch==3)
      {
        root1 = obj.treecopy(obj.root);
        cout<<"\nCopy of the tree is : ";
        obj.preorder(root1);
      }
      else if(ch==4)
        obj.leaf_nodes(obj.root);
      else
      {
        cout<<"\nProgram Exited";
        break;
      }
    }
  }
  else
  {
    cout<<"\nProgram exited";
  }
}
```

OUTPUT

```
C:\Users\Safir\Desktop\ADS>g++ 1_binary_tree.cpp
C:\Users\Safir\Desktop\ADS>a
 1. Create a Binary Tree
2. Exit
Enter your choice : 1
 -----------------------------------------------------------------------
Enter number of nodes : 4

You are at Level 0
Enter the value of root : 2

You are at Level 1
Where do you want to insert (l/r) : l

Enter the value of node : 1

You are at Level 1
Where do you want to insert (l/r) : l

You are at Level 2
Where do you want to insert (l/r) : l

Enter the value of node : 5

You are at Level 1
Where do you want to insert (l/r) : r

Enter the value of node : 6

Tree Created Successfully

-----------------------------------------------------------------------
1. Insert node
2. Display Traversals
3. Copy of a Tree
4. Display Leaf nodes
5. Exit
Enter your choice : 2
-----------------------------------------------------------------------
 -----------------------------------------------------------------------
Preorder Traversal : 2 1 5 6
Inorder Traversal : 5 1 2 6
Postorder Traversal : 5 1 6 2
Breadth-First Traversal : 2 1 6 5
Depth-First Traversal : 2 1 5 6
 -----------------------------------------------------------------------
1. Insert node
2. Display Traversals
3. Copy of a Tree
4. Display Leaf nodes
5. Exit
Enter your choice : 3
 -----------------------------------------------------------------------
Copied Successfully
Copy of the tree is : 2 1 5 6
-----------------------------------------------------------------------
1. Insert node
2. Display Traversals
3. Copy of a Tree
4. Display Leaf nodes
5. Exit
Enter your choice : 4
-----------------------------------------------------------------------
Leaf nodes : 5 6
 -----------------------------------------------------------------------
Enter your choice : 5
-----------------------------------------------------------------------
Program Exited
```