BINARY TREES USING LINKED LIST

AIM

write a menu driven C program to implement binary trees using linked lists and perform operations

ALGORITHM

```
1  Start
2  Enter data of root node, create tree from root
3. Build tree (root, data) to build tree from root
   3.1 If ptr != NULL
       3.1.1 set ptr -> data = data
   3.2 If node has left child
       3.2.1 create left ptr for left child
       3.2.2 set ptr -> left = leftptr
       3.2.3 Enter data of left child ; leftdata
       3.2.4 call build tree (leftptr, leftdata), goto step 3
   3.3 Else set ptr -> left = NULL
   3.4 ~~If~~ node has right child
       3.4.1 create right ptr for child
       3.4.2 set ptr -> right = rightptr
       3.4.3 Enter data of right child; right data
       3.4.4 Call Buildtree (rightptr, right data), goto step 3
   3.5 goto step 4
4  Display preorder (ptr)
```

```
  4.1 if ptr! = NULL
      4.1.1 print ptr -> data
      4.1.2 call preoder (ptr -> left)
      4.1.3 call preoder (ptr -> right)
5 Enter choice for menu
6 case 1: insertion
   6.1 Enter parent node of new node
   6.2 searchnode (ptr, key)
       6.2.1 if ptr = NULL or ptr -> data = key
             6.2.1.1 return ptr
       6.2.2 else if searchnode(ptr -> left, key) = NULL
             6.2.2.1 call searchnode (ptr -> right, key)
  6.3 call searchnode(root, parent), goto step 6.2
  6.4 check if ptr = NULL
     6.4.1 parent node not found, return
  6.5 if ptr -> left = NULL or ptr -> right = NULL
     6.5.1 if insertion as left child
           6.5.1.1 if ptr -> left = NULL
                   6.5.1.1.1 Enter data of new node
                   6.5.1.1.2 set ptr -> left = new node
           6.5.1.2 Else left child is not empty
     6.5.2 if insertion as right child
           6.5.2.1 if ptr -> right = NULL
                   6.5.2.1.1 Enter data of new node
                   6.5.2.1.2 set ptr -> right = new node
           6.5.2.2 else right child is not empty
```

```
    6.6 Else left and right children are not empty
    6.7 goto step 4
7   case 2: Deletion
    7.1 search parent (ptr, data)
        7.1.1 if ptr = NULL, return ptr.
        7.1.2 Else if ptr -> left -> data = data or ptr->right->data
              = data
            7.1.2.1 return ptr
        7.1.3 if searchparent (ptr -> left) = NULL
            7.1.3.1 Call search parent (ptr -> right)
    7.2 if root = NULL
        7.2.1 Tree is empty, return
    7.3 Enter node to be deleted
    7.4 call searchnode (root, node) goto step 6.2
        store address in ptr
    7.5 if ptr = NULL
        7.5.1 Node not found
    7.6 Else if ptr -> left = NULL and ptr -> right = NULL
        7.6.1 call searchparent (root, node), jump to ~~pointers~~
              step 7.1 and store address
    7.7 if ptrparent -> left = ptr.
        7.7.1 set parent -> left = NULL
    7.8 else parent -> right = NULL
    7.9 Free ptr.
```

```
8 case 3 : search
    8.1 Enter node to search
    8.2. call searchnode (root, node), goto step 6.2,
         store address in ptr
    8.3 if ptr ! = NULL
        8.3.1 Node found
    8.4 Else node not found.
    8.5 goto step 4
9 stop
```

CONCLUSION

The program has been executed correctly and output has been varied.