

MIRROR TREE

In this approach I have used Queue data structure .which work like top to bottom approach
For making mirror tree we have to swap left child with right child of all the nodes

Algorithm:

Step1:

Initialise queue data structure and a temp struct node

Step2:

While !isEmptyQueue()

 Push the head into Queue

 Qpush(head)

 Pop the Queue

 Qpop()

 Swap child of Qpop element

 temp=head->left

 head->left=head->right

 head->right=temp

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

```
int count=0;
```

```
struct Node
```

```
{
```

```
    int data;
```

```
    struct Node* left;
```

```
    struct Node* right;  
};
```

```
struct Node* newNode(int data)  
{  
    struct Node* newnode=(struct  
Node*)malloc(sizeof(struct Node));  
  
    newnode->data=data;  
    newnode->left=NULL;  
    newnode->right=NULL;  
    return newnode;  
}
```

```
struct Node* queue[30];  
int front=-1;  
int rear=-1;  
void Qpush(struct Node* x)  
{  
    if(rear!=-1)  
    {  
        queue[++rear]=x;  
    }  
    else  
    {
```

```

        queue[++rear]=x;
        ++front;
    }
}
void Qpop()
{
    front++;
}
bool isemptyQueue()
{
    if(front==rear+1 || front==-1)
        return true;
    return false;
}
struct Node* Qfront()
{
    return queue[front];
}
struct Node* Qfront2()
{
    return queue[front+1];
}
int Qsize()
{
    return (rear-front) + 1;
}

```

```

struct Node* insert(struct Node* node, int data)
{
    if(node==NULL)
    {
        return(newNode(data));
    }
    else
    {
        if(node->data > data)
        {
            node->left=insert(node->left,data);
        }
        else
        {
            node->right=insert(node->right,data);
        }

        return node;
    }
}

void mirror(struct Node* head)
{
    struct Node* temp;
    Qpush(head);
}

```

```

while(!isEmptyQueue())
{
    head=Qfront();
    Qpop();
    temp=head->left;
    head->left=head->right;
    head->right=temp;
    if(head->left)
    {
        Qpush(head->left);
    }
    if(head->right)
    {
        Qpush(head->right);
    }
}
}

```

```

void print(struct Node* head)
{
    if(head==NULL)
        return;
    print(head->right);
    printf("%d\t",head->data);
    print(head->left);
}

```

```
}  
int main()  
{  
    struct Node* node=NULL,*head=NULL;  
    /* node=insert(node, 15);  
    insert(node, 7);  
    insert(node, 10);  
    insert(node, 12);  
    insert(node, 9);  
    insert(node, 8);  
    insert(node, 6);  
    insert(node, 20);  
    insert(node, 30);  
    */
```

```
// 37 23 108 59 86 64 94 14 105
```

```
node= insert(node, 37);  
insert(node, 23);  
insert(node, 108);  
insert(node, 59);  
insert(node, 86);  
insert(node, 64);  
insert(node, 94);  
insert(node, 14);  
insert(node, 105);
```

```
insert(node, 17);  
insert(node, 111);  
insert(node, 65);  
insert(node, 55);  
insert(node, 31);  
insert(node, 79);  
insert(node, 97);  
insert(node, 78);  
insert(node, 25);  
insert(node, 50);
```

```
// 22 66 46 104 98 81 90 68 40 103
```

```
insert(node, 22);  
insert(node, 66);  
insert(node, 46);  
insert(node, 104);  
insert(node, 98);  
insert(node, 81);  
insert(node, 90);
```

```
insert(node, 68);  
insert(node, 40);  
insert(node, 103);
```

```
// 77 74 18 69 82 41 4 48 83 67
```

```
insert(node, 77);  
insert(node, 74);  
insert(node, 18);  
insert(node, 69);  
insert(node, 82);  
insert(node, 41);  
insert(node, 4);  
insert(node, 48);  
insert(node, 83);  
insert(node, 67);
```

```
// 6 2 95 54 100 99 84 34 88 27
```

```
insert(node, 6);  
insert(node, 2);  
insert(node, 95);  
insert(node, 54);  
insert(node, 100);  
insert(node, 99);  
insert(node, 84);  
insert(node, 34);
```



```
insert(node, 88);  
insert(node, 27);
```

```
// 72 32 62 9 56 109 115 33 15 91
```

```
insert(node,72);  
insert(node,32);  
insert(node,62);  
insert(node,9);  
insert(node,56 );  
insert(node,109 );  
insert(node,115 );  
insert(node,33 );  
insert(node,15 );  
insert(node,91 );
```

```
// 29 85 114 112 20 26 30 93 96 87
```

```
insert(node,29);  
insert(node,85 );  
insert(node,114 );  
insert(node,112 );  
insert(node,20 );  
insert(node,26 );
```

```
insert(node,30 );  
insert(node,93 );  
insert(node,96);  
insert(node,87 );
```

```
// 42 38 60 7 73 35 12 10 57 80
```

```
insert(node, 42);  
insert(node, 38 );  
insert(node, 60 );  
insert(node, 7 );  
insert(node, 73);  
insert(node,35 );  
insert(node,12 );  
insert(node, 10);  
insert(node,57 );  
insert(node,80 );
```

```
// 13 52 44 16 70 8 39 107 106 63
```

```
insert(node,13 );  
insert(node,52 );  
insert(node,44 );  
insert(node,16 );  
insert(node,70 );
```

```
insert(node,8 );  
insert(node,39 );  
insert(node,107 );  
insert(node,106 );  
insert(node, 63);
```

```
// 24 92 45 75 116 5 61 49 101 71
```

```
insert(node,24 );  
insert(node,92 );  
insert(node,45 );  
insert(node,75 );  
insert(node,116 );  
insert(node,5 );  
insert(node,61 );  
insert(node,49 );  
insert(node,101 );  
insert(node, 71);
```

```
// 11 53 43 102 110 1 58 36 28 76
```

```
insert(node,11);  
insert(node,53 );
```

```
insert(node,43 );  
insert(node,102);  
insert(node,110 );  
insert(node,1 );  
insert(node,58);  
insert(node,36);  
insert(node,28);  
insert(node,76);
```

```
// 47 113 21 89 51 19 3
```

```
insert(node,47);  
insert(node,113 );  
insert(node,21 );  
insert(node,89);  
insert(node,51 );  
insert(node,19 );  
insert(node,3);
```

```
printf(" inorder traversal:\n");  
print(node);  
printf("\n\n\ninorder traversal:\n");  
mirror(node);  
print(node);
```

```
    return 0;  
}
```