Assignment #04

Name : Anam Batool

Section: Sp22-BCS-B

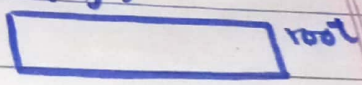
Roll No : Sp22-BCS-112

Submitted to: Mam Yasmeen Jana

"Preorder Traversal"

∴ from main function

```
struct *node *root = newnode(1);
```



∴ go to newnode() function

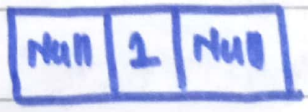
```
node *temp = newnode();
```

∴ move to struct node {

```
int data;
struct Node *left , *right;
}
```

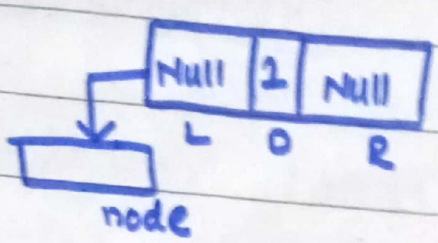
∴ go back to newnode

```
new node( int data)
{
    temp->data = data;
    temp->left = temp->right = null;
    return temp;
}
```



∴ go back to main function

```
root->left = newnode(2);
```



∴ go back to newnode(int data)

{ ~~temp->data = data;~~

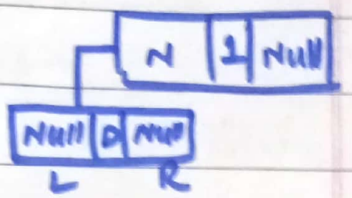
node *temp=new node;

∴ call struct node

{ int data;

struct node *left, *right;

}



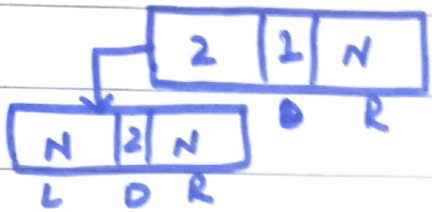
∴ back to newnode() function

{ temp->data = data;

temp->left = temp->right = Null;

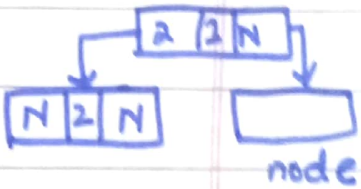
return temp;

}



∴ again back to main function()

temp->right = newnode(3);



∴ call newnode function

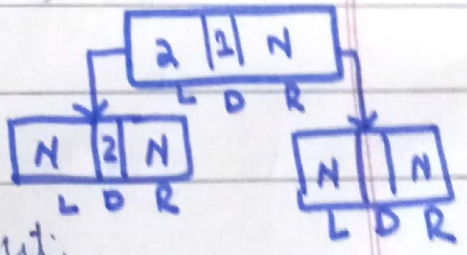
node *temp = newnode;

struct node {

int data;

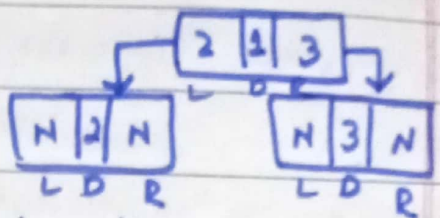
struct node *left, *right;

}



∴ back to newnode function

```
{
    temp → data = data;
    temp → left = temp → right = Null;
    return temp;
}
```

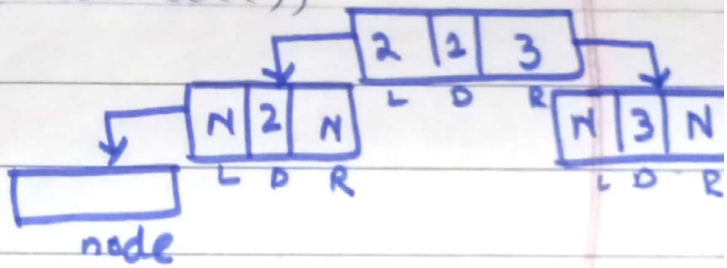


∴ again move back to main function

root → left → left = newnode(4);

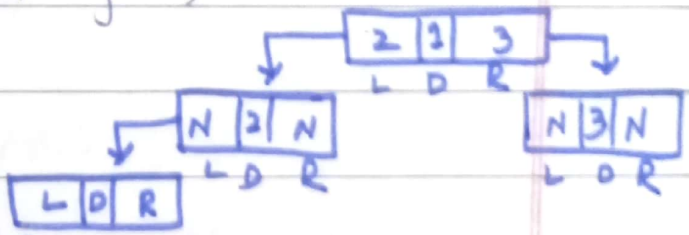
call newnode function

```
{
    Node * temp = newnode();
}
```



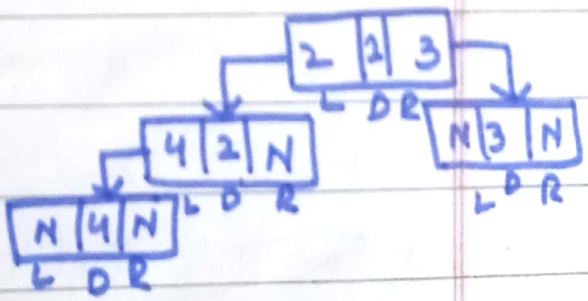
∴ call struct node()

```
{
    int data;
    struct node * left, * right;
}
```



∴ back to newnode function

```
{
    temp → data = data;
    temp → left = temp → right = null;
    return temp;
}
```



➡ from main function

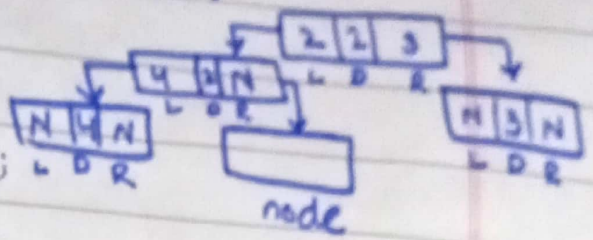
root → left → right = newnode(5);

➡ call newnode function

{

Node *temp = newnode();

}



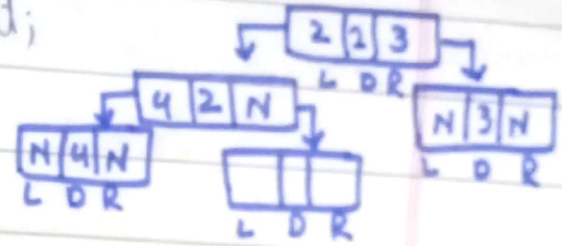
➡ go to struct node

{

int data;

struct node *left, *right;

}



➡ back to newnode function

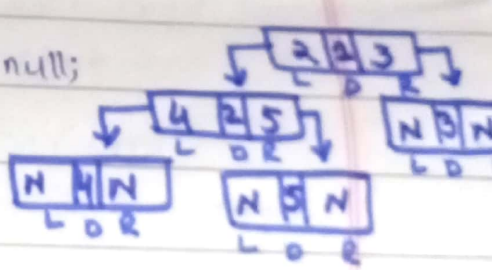
{

temp → data = data;

temp → left = temp → right = null;

return temp;

}



- again main function

{

root → right → left = newnode(6);

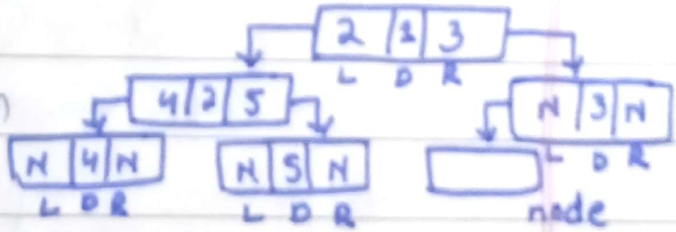
}

- call newnode function

{

Node *temp = newnode();

}



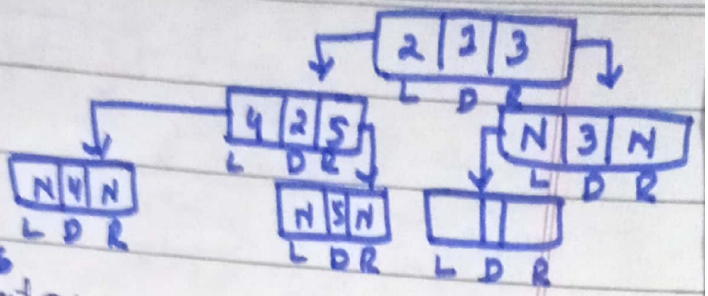
- go to struct node() function

{

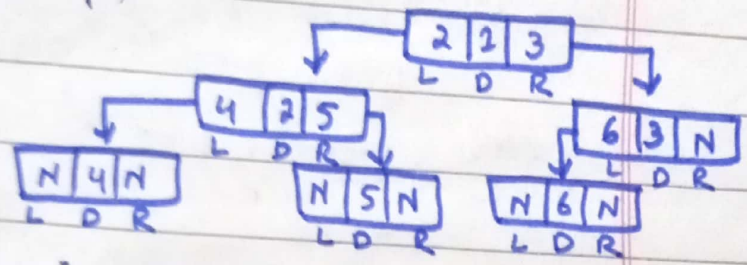
int data;

struct node *left, *right;

Date:




```
{
    temp->data = data;
    temp->left = temp->right = null;
    return temp;
}
```

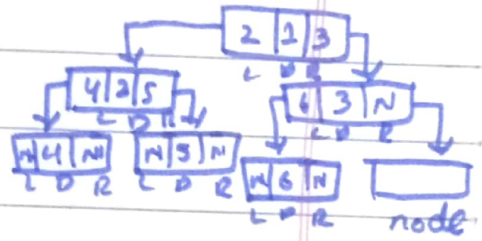


again main function

```
{
    root->right->right = newnode(7);
}
```

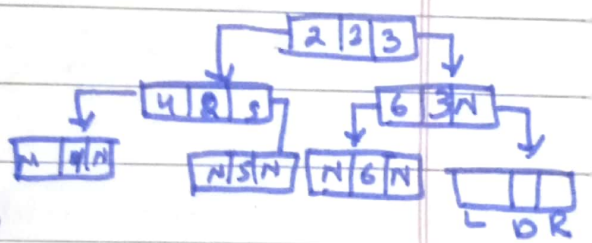
call newnode function

```
{
    Node *temp = newnode();
}
```



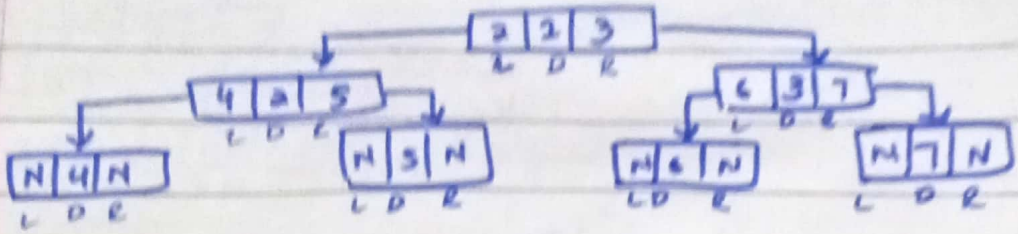
go to struct node function

```
{
    int data;
    struct node *left;
    struct node *right;
}
```



again move to newnode function

```
{
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```



Now go back to main function

```
int main ()
{
    print preorder(root;
}
```

goto preorder function

```
if (root == null) → false
{
    return root;
}
else
    cout<< node→data;  1
    print preorder (root→left);  2
    print preorder( root→right);
```

go to if conditon

```
if (root == null) → false
{
    return ;
}
    cout<< node→data;  1 2
    print preorder( root→left) 4
```

move to if condition

```
if(root == null) → false
{
    return ;
}
```

cout<<node->data 1 2 4

print preorder(root->right) 5

cout<<node->data 1 2 4 5

Now go to root 1 and then move to right

print preorder(root->right) 3

cout<<node->data 1 2 4 5 3

go to left side of node 3;

print prorder(root->left) 6

cout<<node->data 1 2 4 5 3 6

move to right side of root 3

print preorder(root->right) 7

cout<<node->data 7

output ⟶ 1 2 4 5 3 6 7

Postorder traversal

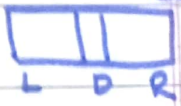
from main function

struct node *root = new node(1);

go to newnode function

Node *temp = new Node;

go to struct node function



{ struct node *left, *right;

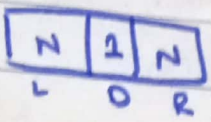
int data;

}

Day: Date:

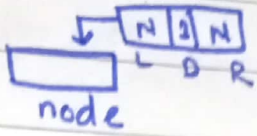
Go back to newnode (int data)

```
temp → data = data;
temp → left = temp → right = NULL;
return temp;
}
```



∴ again from main function

```
*root → left = newnode (2);
```

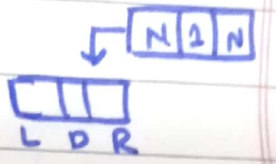


call newnode function

```
Node *temp = new node;
```

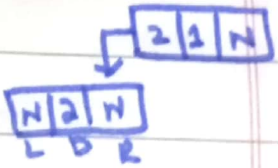
call struct node

```
{
    int data;
    struct Node *left;
    struct Node *right;
}
```



again from newnode function

```
{
    temp → data = data;
    temp → left = temp → right = NULL;
    return temp;
}
```

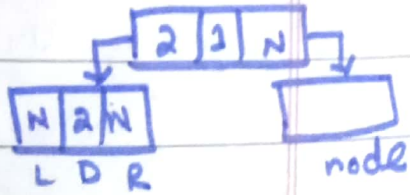


again main function

```
{
    root → right = newnode (3);
}
```

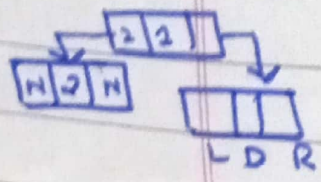
call newnode function

```
{
    Node *temp = new node();
}
```



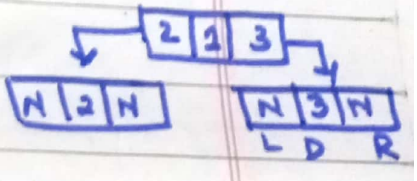
move to struct node function

```
{
    int data;
    struct node * left;
    struct node * right;
}
```



again back to newnode function

```
{
    temp -> data = data;
    temp -> left = temp -> right = null;
    return temp;
}
```

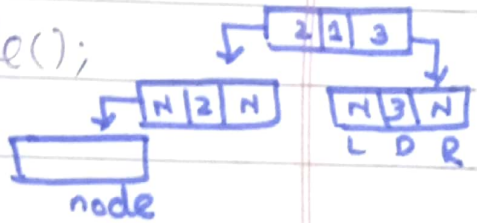


→ again main function

```
{
    root -> left -> left = newnode(4);
}
```

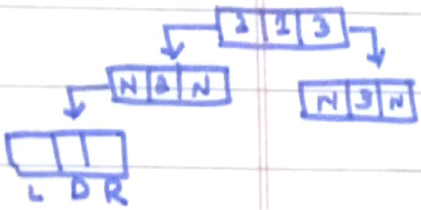
now call newnode function

```
{
    Node * temp = newnode();
}
```



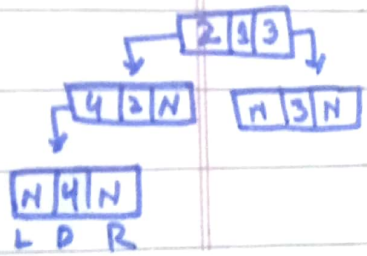
move to struct node ()

```
{
    int data;
    struct node * left;
    struct node * right;
}
```



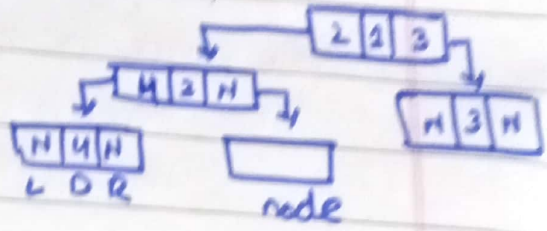
move back to newnode function

```
{
    temp -> data = data;
    temp -> left = temp -> right = null;
    return temp;
}
```

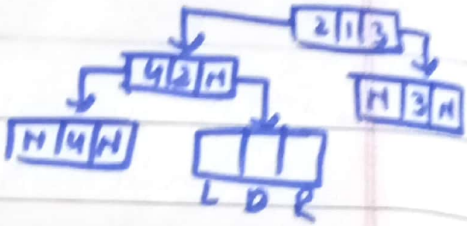


go to main function
{
root->left->right = newnode(5)
}

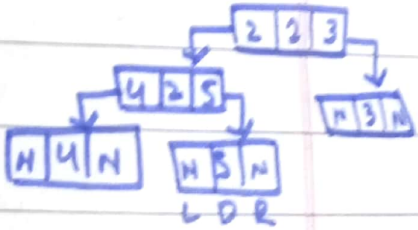
move to newnode function
{
struct node * temp = newnode();
}



call struct node function
{
int data;
struct node *left;
struct node *right;
}

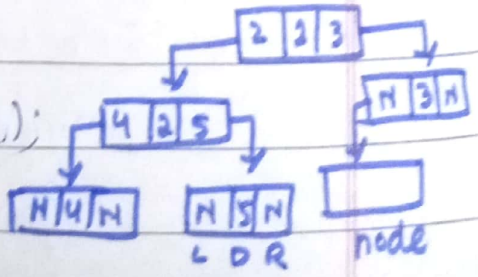


again newnode function
{
temp->data = data;
temp->left = temp->right = null;
return temp;
}



again main function
{
root->right->left = newnode(6)
}

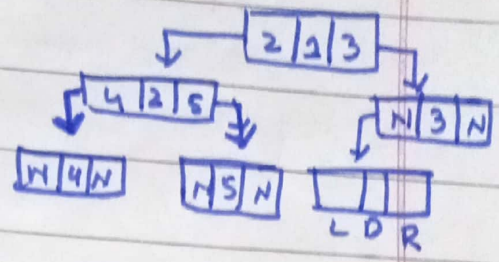
go to newnode function
{
Node * temp = newnode();
}



Date:

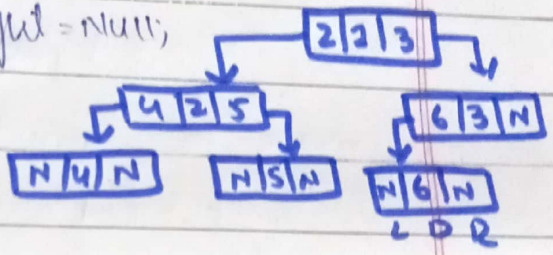
now move to struct node function

```
{
    int data;
    struct node *left;
    struct node *right;
}
```



move back to newnode(int data) function

```
{
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```

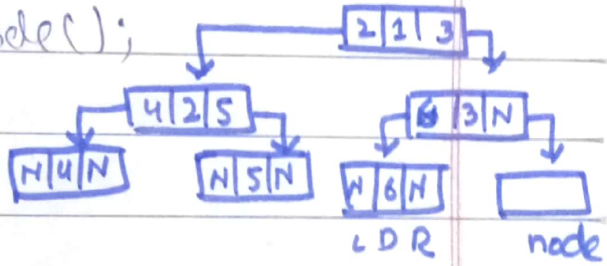


move to main function

```
{
    root->right->right = newnode(7)
}
```

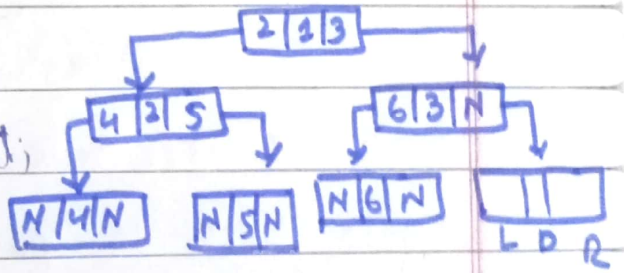
go to newnode function

```
{
    node *temp = newnode();
}
```



call struct node function

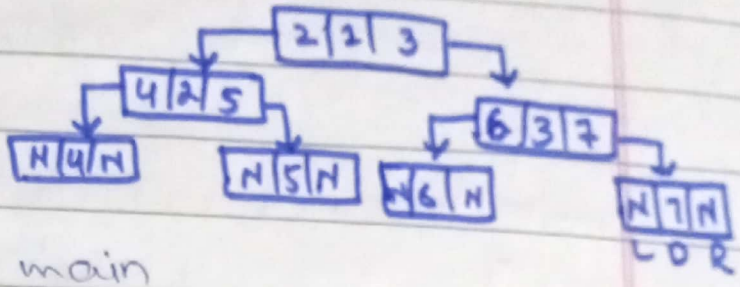
```
{
    int data;
    struct node *left;
    struct node *right;
}
```



move to newnode(int data)

```
{
    temp->data = data;
```

temp->left : temp->right = null;
return temp;
}



go back to int main

int main ()
{
print postorder (root;

move to print postorder function

if (root == null) → false
{
return root;
}

Print postorder(root → left);

print postorder(node → right);

cout << node → data <<

go back to main conditions

if (node == null) → false
{
return node;
}

print postorder (node → left)

cout << node → data << 4

again back to postorder function

if (node == null) → false

return node;
}

print postorder(node→left)
print postorder(node→right)
cout<< node→data<< 4 5

again move to node 2; print the data of current node

cout<< node→data<< 4 5 2

now call postorder function

print postorder(node→right) 3

go to postorder function

print postorder(node→left) 6

cout<< node→data<< 4 5 2 6

As there is no child of node 6 so move back to node 3

print postorder(node→right) 7

cout<< node→data<< 4 5 2 6 7 3 1

output → 4526731

(Inorder Traversal)

From main() function

{

struct node* root = newnode();

}

go to newnode function

{

struct node* newnode(int value);

Struct node* temp = new struct node;

}

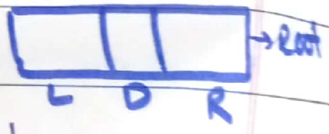
go to struct part

struct node{

int data;

struct node* left, *right;

}



now again go back to newnode function

{

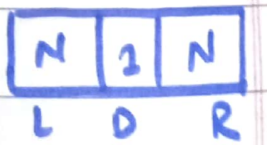
Node → data = value;

Node → left = null;

Node → right = NULL;

return node;

}

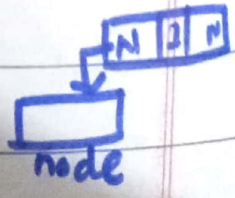


again main function

{

*root → left = newnode;

}



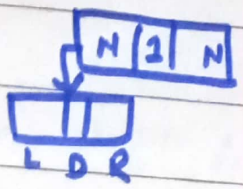
move to newnode function

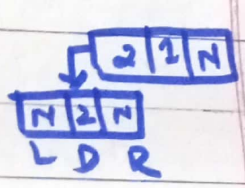
Date:

struct node * newnode(int value) 2
struct node * temp = new struct node;
}

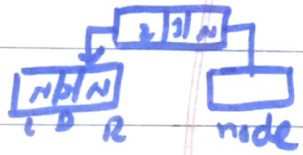
move to struct part
struct node{
int data;
Node *left, *right;
}

again from newnode function
{
Node→data = value; 2
Node→left = Null;
Node→right = Null;
}





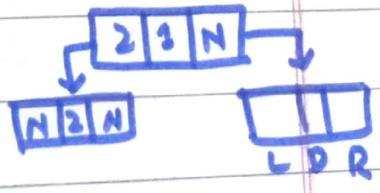
from main function
{
root→right = newnode();
}



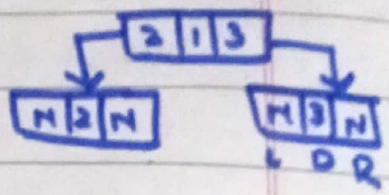
from newnode function
{
struct node * newnode(int value); 3
struct node *temp = new struct node;
}

As per struct part
struct node
{
int data;
Node * left, *right;
}

again move to new node function



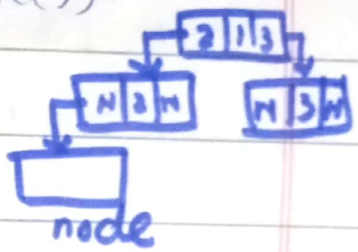
```
{   Node->data = value;
    Node->left = Null;
    Node->right = Null;
    return node;
}
```



∴ again main function

```
{
    root->left->left = newnode();
}
```

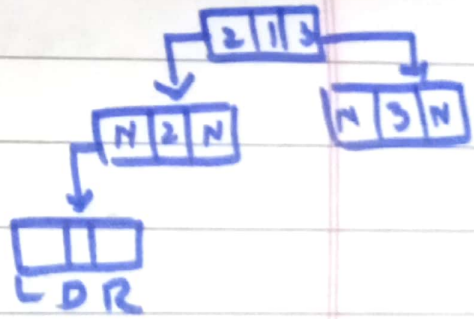
move to newnode function




```
{
    struct node *newnode(int value)
    struct node *temp = newnode;
}
```

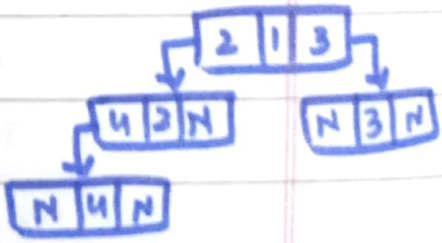
from struct part

```
struct node
{
    int data;
    node *left, *right;
}
```



again newnode function

```
{   Node->data = value;
    Node->left = Null;
    Node->right = Null;
    return node;
}
```



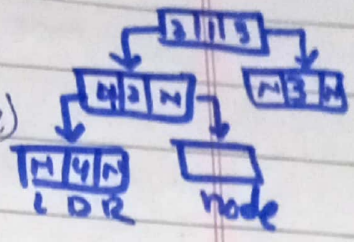
∴ again from main function

{

root → left → right = newnode();
}

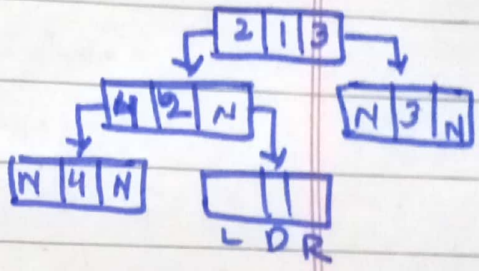
move to newnode function

{ struct node *newnode(int value)
struct node *temp = newnode;
}



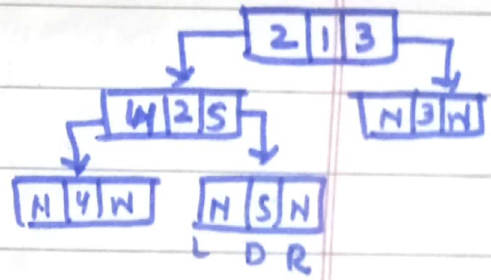
again at struct part

struct node
{ int data;
node *left, *right;
}



now from newnode function

{ Node → data = value;
Node → left = NULL;
node → right = NULL;
return node;
}

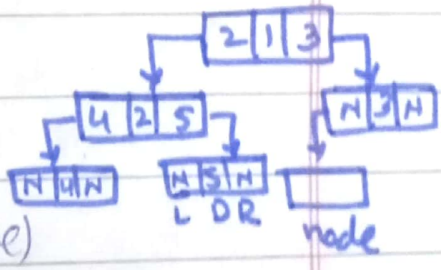


→ again main function

{ root → right → left = newnode(6);
}

go to newnode function

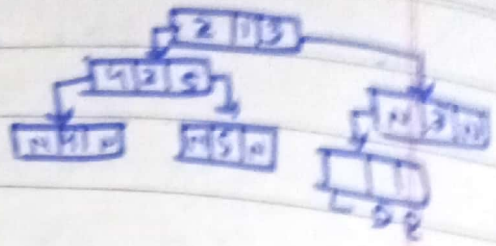
{ struct node *newnode(int value)
struct node *temp = newnode;
}



now move to struct part

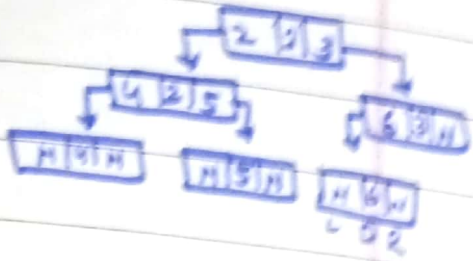
{

```
struct node
{
    int data;
    node *left, *right;
};
```



again move to newnode function

```
{
    Node->data = value;
    Node->left = Null;
    Node->right = Null;
    return Node;
}
```

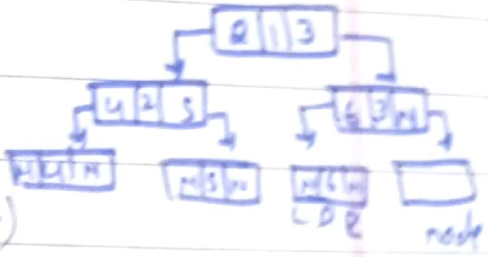


again main function

```
{
    root->right->right = newnode(7);
}
```

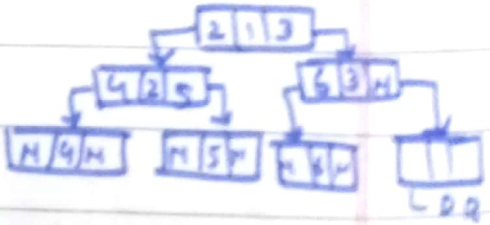
move to newnode function

```
{
    struct node *newnode(int value)
    struct node *temp = newnode;
}
```



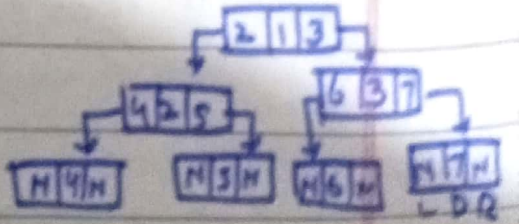
move to struct part

```
{
    int data;
    Node *left, *right;
}
```



again back to newnode function

```
{
    Node->data = value;
    Node->left = Null;
    Node->right = Null;
```



Traversal inorder (root)

go to traversal inorder function

if (root == null) false

{ return root;

}

traversal inorder (root → left)

go to left of root 2

traversal inorder (root → left)

cout << node → data << 4

go back to root 2

cout << node → data << 4 2

now move to the right side of root 2

traversal inorder (root → right)

cout << node → data << 4 2 5

Back to root 1

cout << node → data << 4 2 5 1

move to right side of root 1

traversal inorder (root → right)

left side of root 3

traversal inorder (root → left)

Cout << node → data << 4 2 5 1 6

Now the root 3

cout << node → data << 4 2 5 1 6 3

Date:

traversal inorder (root → right)

Cout<< node → data << 4 2 5 1 6 3 7

output → 4 2 5 1 6 3 7