

19/02/24



WAP to construct, Traverse (inorder, postorder, preorder) and display

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

```
node *create ()  
{
```

```
    node *temp;  
    printf ("Enter the data: ");  
    temp = (node *) malloc (sizeof (node));  
    scanf ("%d", &temp->data);  
    temp->left = temp->right = NULL;  
    return temp;  
}
```

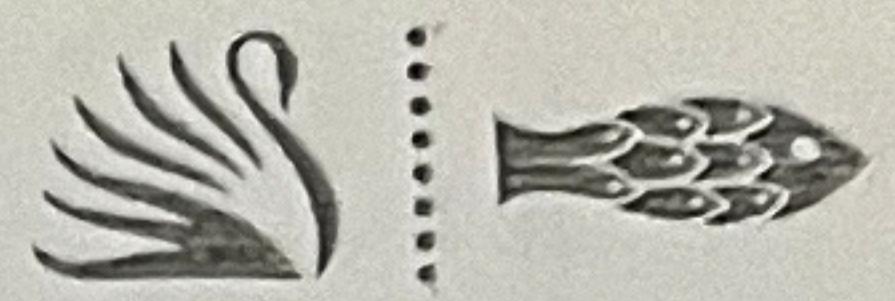
```
void insert (node *root, node *temp)  
{
```

```
    if (temp->data < root->data)  
{
```

```
        if (root->left != NULL)  
            insert (root->left, temp);  
        else
```

```
            root->left = temp;  
    }
```

~~```
 if (temp->data > root->data)
{
```~~~~```
        if (root->right != NULL)  
            insert (root->right, temp);
```~~



and

else

```
    root → right = temp  
}
```

}

void inorder (node * root)

{

```
    if (root != NULL)  
    {
```

inorder (root → left)

printf ("%d", root → data);

inorder (root → right);

}

}

void postorder (node * root)

{

```
    if (root != NULL)  
    {
```

postorder (root → left);

printf ("%d", root → data);

postorder (root → right);

}

}

void preorder (node * root)

{

```
    if (root != NULL)  
    {
```

printf ("%d", root → data);

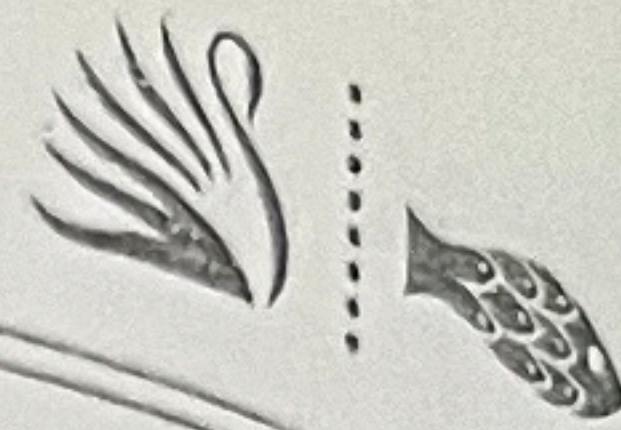
preorder (root → left);

preorder (root → right);

}

}

void main () {



1.

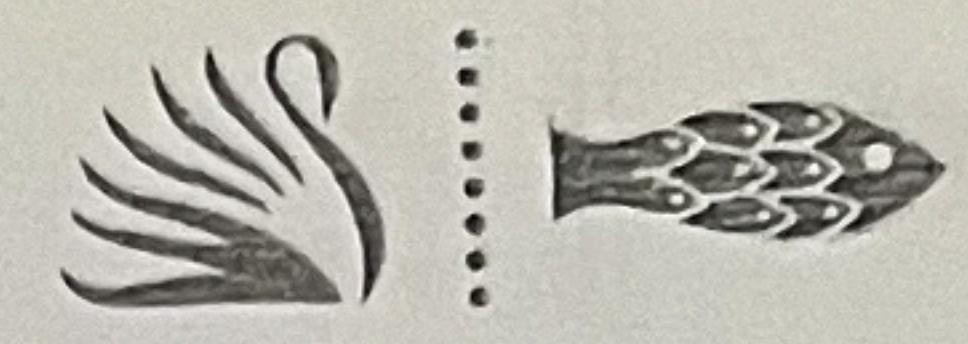
```
root = NULL;  
root = insert (root, 10);  
root = insert (root, 20);  
root = insert (root, 30);  
root = insert (root, 40);  
root = insert (root, 50);  
root = insert (root, 60);  
root = insert (root, 70);  
printf ("Insertion Successful \n");  
inorder (root);  
printf ("\n");  
preorder (root);  
printf ("\n");  
postorder (root);  
printf ("\n");  
}
```

OUTPUT:

Insertion successful

10 20 30 40 50 60 70
10 50 20 60 70 30 40
50 70 60 40 30 20 10

Soham
20.09.24



LeetCode :

1. Delete the middle node of linked list .

```
struct ListNode* deleteMiddle (struct ListNode* head)
```

```
{
```

```
    struct ListNode* ptr = head;
```

```
    struct ListNode* preptr = NULL;
```

```
    int count = 0;
```

```
    int n = 0;
```

```
    if (head == NULL) {
```

```
        return head;
```

```
}
```

```
    else if (head->next == NULL)
```

```
{
```

```
        free(head);
```

```
        return NULL;
```

```
}
```

```
    else {
```

```
        while (ptr != NULL)
```

```
{
```

```
            count++;
```

```
            ptr = ptr->next;
```

```
}
```

```
    ptr = head;
```

```
    while (ptr != count / 2)
```

```
{
```

```
        preptr = ptr;
```

```
        ptr = ptr->next;
```

```
        n++;
```

```
}
```

```
    preptr->next = ptr->next;
```

```
    free(ptr)
```

Leetcode : odd Even Lh

struct ListNode* oddEvenList (struct Node * head)

if (head == NULL || head->next == NULL)

{

return head;

}

struct ListNode * oddNode = head

struct ListNode * headEven = oddNode->next;

struct ListNode * headEven = evenNode

while (evenNode != NULL && evenNode->next != NULL)

{

oddNode->next = evenNode->next;

oddNode = oddNode->next;

evenNode->next = oddNode->next;

evenNode = evenNode->next;

}

oddNode->next = headEven;

return head;

{

26/02

8.