

program 11: Binary Search Tree

Date	
Page	

- a) construct a binary search tree
 - b) traverse using all three methods
 - c) display the elements

pseudocode: a) struct Node* insertb(struct Node* node,
 int key){
 if (node == NULL) return newNode(key);
 if (key < node->key)
 node->left = insertb(node->left, key);
 else if (key > node->key)
 node->right = insertb(node->right, key);
 return node;

b) void inorder(struct Node * root) {

if (root != NULL) {

```
    if (root != null) {
        inorder(root.left);
        printf("%d", root.key);
        inorder(root.right);
    }
}
```

inorder(moot->right), (ov) :-

$$\cdot (\text{cosec } \theta)^2 \cdot (-\sec \theta) + \sec \theta = -\sec^3 \theta - \sec \theta$$

(stop & score value) + 1 score

```
void preorder(struct Node *root) {
```

if (root == NULL) return;

```
printf("%d", root->data);
```

preorder (root \rightarrow left),

~~preorder (root, right); }~~

```
void postorder( struct Node* root ) {  
    if( root != NULLE ) {  
        postorder( root->left );  
        postorder( root->right );  
        cout << root->data;  
    }  
}
```

~~postorder (root → left), i.e.) recursion~~

~~postorder (root → right);
left (root → data)~~

```
printf("%d", root->data);
```

7 (you don't have to say big
print you,

code:

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

```
struct Node {
```

```
    int data;
```

```
    struct Node *left, *right;
```

```
};
```

```
(union, struct Node);
```

```
struct Node* createNode(int value) {
```

```
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = value;
}
```

```
newNode->left = newNode->right = NULL;
```

```
return newNode;
```

```
}
```

```
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
```

```
}
```

```
void inorder(struct Node* root) {
```

```
    if (root == NULL) return;
```

```
    inorder(root->left); printf("%d", root->data);
```

```
    inorder(root->right);
```

```
}
```

```
;
```

```
;(stob<-root, "left"); printing
```

```
void preorder(struct Node* root) {
```

```

if (root==NULL) return;
printf ("%d", root->data);
preorder (root->left);
preorder (root->right);
}

```

```

void postorder (struct Node *root) {
    if (root==NULL) return;
    postorder (root->left);
    postorder (root->right);
    printf ("%d", root->data);
}

```

```

void displayTree (struct Node *root) {
    int level=0, indentSpace=5;
    if (root==NULL) return;
    for (i=0; i<(level*indentSpace); i++)
        printf (" ");
    printf ("%d \n", root->data);
}

```

```

if (root->left!=NULL) {
    for (i=0; i<(level*indentSpace); i++)
        printf (" ");
    printf ("\n");
    printTree (root->left, level+1, indentSpace);
}

```

```

if (root->right !=NULL) {
    for (i=0; i<(level*indentSpace); i++)
        printf (" ");
    printf ("\n");
}

```

printTree (root->right, level + 1, indentSpace);
3 (left) return "left" library
3 (right) return "right"
(empty root) returning

```
int main() {
    struct Node *root = NULL;
    int choice, value;
    printf("1: Insert into BST");
    printf("2: Inorder traversal");
    printf("3: Preorder traversal");
    printf("4: Postorder traversal");
    printf("5: Display");
    printf("6: Exit");
    while(1) {
        switch(choice) {
            case 1: // insert & print
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2: // print inorder traversal
                printf("Inorder traversal: ");
                inorder(root);
                break;
            case 3: // print pre-order traversal
                printf("Preorder traversal: ");
                preorder(root);
                break;
            case 4: // print post-order traversal
                printf("Postorder traversal: ");
                postorder(root);
                break;
            case 5: // print tree structure
                printTree(root, 1, 0);
                break;
            case 6: // exit
                exit(0);
        }
    }
}
```

case 2: ~~inorder traversal~~ if

```
printf("Inorder traversal: ");
inorder(root);
printf("Inorder traversal: ");
break;
```

case 3:

```
printf("Preorder traversal: ");
preorder(root);
printf("Preorder traversal: ");
break;
```

case 4:

```
printf("postorder traversal: ");
postorder(root);
```

```
printf("\n");
break;
```

```
; if tree is empty (root == NULL) then
```

case 5: Inorder, * left-right-traversal

```
display tree(root);
break;
```

case 6:

```
exit 0;
```

default:

```
printf("invalid choice! Try again (n): ");

```

?

z

```
return 0;
```

g

Output: n -- Binary Search Tree Menu--

1. Insert into BST

2. Inorder

3. Preorder

4. Postorder

5. Exit

Enter choice :

Enter value 10

Enter choice 1

Enter value 30

Enter choice 1

Enter value 5

Enter choice 1

Enter value 100

enter choice : 3

preorder : 10 5 30 100

enter choice : 2

inorder : 5 10 30 100

enter choice : 4

post order : 5 100 30 10

enter choice : 5

BST Elements : 5 10 30 100

enter choice : 6