

**Name: Kashif Ali**  
**Roll No: 20P-0648**  
**Section: 3D**  
**Lab 8 tasks**

# Task 1

## code

output

```
C task1.cpp x
C task1.cpp > check(Node *)
1 // Task 1 to check whether it is bst or not
2 #include<iostream>
3 using namespace std;
4
5 class Node{
6 public:
7     Node *left;
8     Node *right;
9     int data;
10    Node(int val){
11        data=val;
12        this->left=NULL;
13        this->right=NULL;
14    }
15
16 };
17 Node *Insert(Node *root,int data){
18     if(root==NULL){
19         return new Node(data);
20     }
21     else if(root->data>data){
22         root->left=Insert(root->left,data);
23     }
24     else{
25         root->right=Insert(root->right,data);
26     }
27     return root;
28 }
29
30
31
32 int check(Node *root){
33
34     if(root->left!=NULL && root->left->data<root->data){
35         return 0;
36     }
37
38     if(root->right!=NULL && root->right->data>root->data){
39         return 0;
40     }
41     if(!check(root->left) || !check(root->right)){
42         return 0;
43     }
44     return 1;
45 }
46
47 int main(){
48     Node *root;
49     root=NULL;
50     root=Insert(root,21);
51     Insert(root,12);
52     Insert(root,13);
53
54     cout<<endl;
55     if(check(root)==NULL){
56         cout<<" Marks= 10 ";
57     }
58     else{
59         cout<<" Marks=0 ";
60     }
```

```
kashiii@kashiii:~/Downloads/dslab8$ ./task1.exe
```

```
Marks= 10
```

```
kashiii@kashiii:~/Downloads/dslab8$
```

```
1  #include<iostream>
2  using namespace std;
3  class node{
4      public:
5          node *left;
6          node *right;
7          string data;
8          node(string val){
9              this->left=NULL;
10             this->right=NULL;
11             val=data;
12         }
13 };
14 class BinaryTree{
15     private:
16         node *root;
17     public:
18         void addnode(int key,string name){
19             node *tmp;
20             tmp=new node(name);
21             if(root==NULL){
22                 return new node(name);
23             }
24             else if(root->data>name){
25                 root->left=addnode(key, name);
26             }
27         }
28     }
29 };
30
31
```

Task 2 A code

```

C Task #2.cpp > C Task #2.cpp > C Task #2.cpp
C Task #2.cpp > main()
1  #include<iostream>
2  using namespace std;
3
4  class node{
5  public:
6      node *left;
7      node *right;
8      int data;
9      node(int val){
10         data=val;
11         this->left=NULL;
12         this->right=NULL;
13     }
14 };
15
16
17
18
19 node *Insert(node *root,int data){
20     if(root==NULL){
21         return new node(data);
22     }
23     else if(root->data>data){
24         root->left=Insert(root->left,data);
25     }
26     else{
27         root->right=Insert(root->right,data);
28     }
29     return root;
30 }
31
32 int main(){
33
34     node *root;
35     root=NULL;
36     root=Insert(root,15);
37     Insert(root,10);
38     Insert(root,8);
39     Insert(root,20);
40     Insert(root,25);
41     Insert(root,12);
42     Insert(root,16);
43
44     // preorder(root);
45
46 }
47

```

## Task 2 A

```

task2b.cpp > preorder(node *)
1  #include<iostream>
2  using namespace std;
3
4  class node{
5  public:
6      node *left;
7      node *right;
8      int data;
9      node(int val){
10         data=val;
11         this->left=NULL;
12         this->right=NULL;
13     }
14 };
15
16
17
18
19 node *Insert(node *root,int data){
20     if(root==NULL){
21         return new node(data);
22     }
23     else if(root->data>data){
24         root->left=Insert(root->left,data);
25     }
26     else{
27         root->right=Insert(root->right,data);
28     }
29     return root;
30 }
31 void preorder(node *root){
32     if(root==NULL){
33         return;
34     }
35     else{
36         cout<<root->data<<" ";
37         preorder(root->left);
38         preorder(root->right);
39     }
40 }
41
42 int main(){
43
44     node *root;
45     root=NULL;
46     root=Insert(root,15);
47     Insert(root,10);
48     Insert(root,8);
49     Insert(root,20);
50     Insert(root,25);
51     Insert(root,12);
52     Insert(root,16);
53
54     preorder(root);

```

Task 2 b code

output

```

kashiii@kashiii:~/Downloads/dslab8$ ./task2b.exe
15 10 8 12 20 16 25 kashiii@kashiii:~/Downloads/dslab8$

```

```

C task3a.cpp x
C task3a.cpp > main()
1  #include<iostream>
2  using namespace std;
3
4  class node{
5  public:
6      node *left;
7      node *right;
8      int data;
9      node(int val){
10         data=val;
11         this->left=NULL;
12         this->right=NULL;
13     }
14 };
15
16 node *Insert(node *root,int data){
17     if(root==NULL){
18         return new node(data);
19     }
20     else if((root->data>data){
21         root->left=Insert(root->left,data);
22     }
23     } else{
24         root->right=Insert(root->right,data);
25     }
26     return root;
27 }
28
29 void Inorder(node *root){
30     if(root==NULL){
31         return;
32     }
33     else{
34         Inorder(root->left);
35         cout<<root->data<<" ";
36         Inorder(root->right);
37     }
38 }
39
40 void pre_order(node *root){
41     if(root==NULL){
42         return;
43     }
44     else{
45         cout<<root->data<<" ";
46         pre_order(root->left);
47         pre_order(root->right);
48     }
49 }
50
51
52 int main(){
53     node *root;
54     root=NULL;
55     root=Insert(root,21);
56     Insert(root,12);
57     Insert(root,13);
58     cout<<"Inorder "<<endl;
59     Inorder(root);
60     cout<<endl;
61     //check(root);
62
63     cout<<endl;
64     cout<<"pre order "<<endl;
65     pre_order(root);
66 }
67
68

```

## Task 3 A code

output

```

kashiii@kashiii:~/Downloads/dslab8$ ./task3a.exe
Inorder
12 13 21
pre order
kashiii@kashiii:~/Downloads/dslab8$

```

```

C task3b.cpp > iterativePreorder(node*)
1 // C++ program to implement iterative preorder traversal
2 #include <bits/stdc++.h>
3
4 using namespace std;
5 struct node {
6     int data;
7     struct node* left;
8     struct node* right;
9 };
10 struct node* newNode(int data)
11 {
12     struct node* node = new struct node;
13     node->data = data;
14     node->left = NULL;
15     node->right = NULL;
16     return (node);
17 }
18
19 void iterativePreorder(struct node* root)
20 {
21     if (root == NULL)
22         return;
23
24     stack<node*> nodeStack;
25     nodeStack.push(root);
26
27     while (nodeStack.empty() == false) {
28         struct node* node = nodeStack.top();
29         printf("%d ", node->data);
30         nodeStack.pop();
31
32         if (node->right)
33             nodeStack.push(node->right);
34         if (node->left)
35             nodeStack.push(node->left);
36     }
37 }
38
39 int main()
40 {
41     struct node* root = newNode(10);
42     root->left = newNode(8);
43     root->right = newNode(2);
44     root->left->left = newNode(3);
45     root->left->right = newNode(5);
46     root->right->left = newNode(2);
47     iterativePreorder(root);
48     return 0;
49 }
50
51
52
53
54
55

```

## Task 3 B code

## output

```

kashiii@kashiii:~/Downloads/dslab8$ g++ task3b.cpp -o task3
task3a.cpp task3a.exe task3b.cpp
kashiii@kashiii:~/Downloads/dslab8$ g++ task3b.cpp -o task3b.exe
kashiii@kashiii:~/Downloads/dslab8$ ./task3b.exe
10 8 3 5 2 2 kashiii@kashiii:~/Downloads/dslab8$

```

C task4.cpp &gt; removeOutsideRange(node \*, int, int)

```
1  #include<iostream>
2  using namespace std;
3
4  class node
5  { public:
6      int key;
7      node *left;
8      node *right;
9  };
10
11  node* removeOutsideRange(node *root, int min, int max)
12  {
13      if (root == NULL)
14          return NULL;
15      root->left = removeOutsideRange(root->left, min, max);
16      root->right = removeOutsideRange(root->right, min, max);
17
18
19      if (root->key < min)
20      {
21          node *rChild = root->right;
22
23          return rChild;
24      }
25      if (root->key > max)
26      {
27          node *lChild = root->left;
28          return lChild;
29      }
30      return root;
31  }
32  node* newNode(int num)
33  {
34      node* temp = new node;
35      temp->key = num;
36      temp->left = temp->right = NULL;
37      return temp;
38  }
```

## Task 4 code part a



```
39
40 node* insert(node* root, int key)
41 {
42     if (root == NULL)
43         return newNode(key);
44     if (root->key > key)
45         root->left = insert(root->left, key);
46     else
47         root->right = insert(root->right, key);
48     return root;
49 }
50
51 void inorderTraversal(node* root)
52 {
53     if (root)
54     {
55         inorderTraversal( root->left );
56         cout << root->key << " ";
57         inorderTraversal( root->right );
58     }
59 }
60 int main()
61 {
62     node* root = NULL;
63     root = insert(root, 15);
64     root = insert(root, 10);
65     root = insert(root, 20);
66     root = insert(root, 8);
67     root = insert(root, 12);
68     root = insert(root, 18);
69     root = insert(root, 25);
70     root = insert(root, 18);
71
72
73     cout << "Inorder traversal of the given tree is: ";
74     inorderTraversal(root);
75
76     root = removeOutsideRange(root, 8, 10);
77
78     cout << "\nInorder traversal after removing nodes tree is: ";
79     inorderTraversal(root);
80
81     return 0;
82 }
83
84
```

## Task 4 code part b

## Task 4 output

```
kashiii@kashiii:~/Downloads/dslab8$ g++ task4.cpp -o task4.exe
kashiii@kashiii:~/Downloads/dslab8$ ./task4.exe
Inorder traversal of the given tree is: 8 10 12 15 18 18 20 25
Inorder traversal after removing nodes tree is: 8 10 kashiii@kashiii:~/Downloads/dslab8$
```

🐞 Debug ⚙️ ▶ ⚠️ 🗑️ 🔗 Live Share

# Thank you

...