SANGOLA COLLEGE, SANGOLA

Class-B.Sc(ECS)-II, SEM-IV 2024-25

Practical Assignments

Sub- Data Structure using C++-II

Assignment No- 3

1) Write a program to implement binary search tree with tree traversal methods.

```
 #include<iostream.h>
#include<conio.h>

class node
{
	node *left, *right;
	int info;
	public:
		node* create();
		void insert(int);
		void preorder(node*);
		void inorder(node*);
		void postorder(node*);
}*root;

node* node :: create()
{
	node *p = new node;
	return(p);
}

void node :: insert(int x)
{
	node *temp, *p = create();
	p->left = NULL;
	p->info = x;
	p->right= NULL;
```

```
if(root == NULL)
{
        root = p;
}
else
{
        temp = root;
        while(temp != NULL)
        {
                if(p->info < temp->info)
                {
                        if(temp->left == NULL)
                        {
                                temp->left = p;
                                break;
                        }
                        else
                        {
                                temp = temp->left;
                        }
                }
                else if(p->info > temp->info)
                {
                        if(temp->right == NULL)
                        {
                                temp->right = p;
                                break;
                        }
                        else
                        {
                                temp = temp->right;
                        }
                }
                else
                {
                        cout<<"\nDo Not repeat node...\n";
                        break;
                }
                }
        }
}

void node :: preorder(node *p)
{
        if(p != NULL)
        {
                cout<<p->info<<"\t";
                preorder(p->left);
                preorder(p->right);
        }
}

void node :: inorder(node *p)
{
        if(p != NULL)
        {
                inorder(p->left);
                cout<<p->info<<"\t";
                inorder(p->right);
        }
}

void node :: postorder(node *p)
{
        if(p != NULL)
        {
                postorder(p->left);
                postorder(p->right);
                cout<<p->info<<"\t";
        }
}

void main()
{
        int ch = 0, x;
        node obj;
        clrscr();
```

```
	do
	{
		cout<<"\nEnter your choice : ";
		cout<<"\n1. Insert\n2. Pre-order\n3. In-order\n4. Post-
order\n5. Exit.\n";
		cin>>ch;

		switch(ch)
		{
			case 1:
				cout<<"\nEnter node : ";
				cin>>x;
				obj.insert(x);
				break;

			case 2:
				cout<<"\nNodes in tree by pre-order method : ";
				obj.preorder(root);
				break;

			case 3:
				cout<<"\nNodes in tree by in-order method : ";
				obj.inorder(root);
				break;

			case 4:
				cout<<"\nNodes in tree by post-order method : ";
				obj.postorder(root);
				break;

			case 5:
				cout<<"\nProgram stoped...\n";
				break;
		}
	}while(ch != 5);
}
```

2) Write a program that check entered node is leaf or having one child or having two children.

```
#include<iostream.h>
#include<conio.h>

class node
{
        node *left, *right;
        int info;
        public:
                node* create();
                void insert(int);
                void check(int);
}*root;

node* node :: create()
{
        node *p = new node;
        return(p);
}

void node :: insert(int x)
{
        node *temp, *p = create();
        p->left = NULL;
        p->info = x;
        p->right= NULL;

        if(root == NULL)
        {
                root = p;
        }
        else
        {
                temp = root;
                while(temp != NULL)
                {
                        if(p->info < temp->info)
                        {
                                if(temp->left == NULL)
                                {
                                        temp->left = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->left;
                                }
                        }
                        else if(p->info > temp->info)
                        {
                                if(temp->right == NULL)
                                {
                                        temp->right = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->right;
                                }
                        }
                        else
                        {
                                cout<<"\nDo Not repeat node...\n";
                                break;
                        }
                }
        }
}


void node :: check(int x)
{

        int f = 0;
        node *temp = root;
        while(temp != NULL)
        {
                if(temp->info == x)
                {
                        f = 1;
                        break;
                }
                else if(temp->info > x)
                {
                        temp = temp->left;
                }
                else if(temp->info < x)
                {
                        temp = temp->right;
                }
        }

        if(f == 1)
        {
                if(temp->left == NULL && temp->right == NULL)
                {
                        cout<<"\nEntered node is leaf...\n";
                }
                else if(temp->left != NULL && temp->right != NULL)
                {
                        cout<<"\nEntered node has two child...\n";
                }
                else
                {
                        cout<<"\nEntered node has one child...\n";
                }
        }
        else
        {
                cout<<"\nNode is NOT found...\n";
        }
}
```

```
void main()
{
	int ch, x;
	node obj;
	clrscr();

	do
	{
		cout<<"\nEnter your choice : ";
		cout<<"\n1. Insert\n2. Number of child\n3. Exit.\n";
		cin>>ch;

		switch(ch)
		{
			case 1:
				cout<<"\nEnter node : ";
				cin>>x;
				obj.insert(x);
				break;

			case 2:
				cout<<"\nEnter node to check child : ";
				cin>>x;
				obj.check(x);
				break;

			case 3:
				cout<<"\nProgram stoped...\n";
				break;

			default:
				cout<<"\nWrong choice...\n";
				break;
		}
	}while(ch != 3);
}
```

3) Write a program to implement binary search tree with following operations.1) Insert( ) 2) Count_leaf( ) 3) Count_total( ) 4) Search( )

```
#include<iostream.h>
#include<conio.h>

class node
{
        node *left, *right;
        int info;
        public:
                node* create();
                void insert(int);
                void search(int);
                int count_leaf(node*);
                int count_total(node*);
}*root;

node* node :: create()
{
        node *p = new node;
        return(p);
}

void node :: insert(int x)
{
        node *temp, *p = create();
        p->left = NULL;
        p->info = x;
        p->right= NULL;

        if(root == NULL)
        {
                root = p;
        }
        else
        {

                temp = root;
                while(temp != NULL)
                {
                        if(p->info < temp->info)
                        {
                                if(temp->left == NULL)
                                {
                                        temp->left = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->left;
                                }
                        }
                        else if(p->info > temp->info)
                        {
                                if(temp->right == NULL)
                                {
                                        temp->right = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->right;
                                }
                        }
                        else
                        {
                                cout<<"\nDo Not repeat node...\n";
                                break;
                        }
                }
        }
}

int node :: count_leaf(node *p)
{
        if(p == NULL)
        {
                return 0;
        }
        else if(p->left == NULL && p->right == NULL)
        {
                return 1;
        }
        else
        {
                return(count_leaf(p->left) + count_leaf(p->right));
        }
}

int node :: count_total(node *p)
{
        if(p == NULL)
        {
                return 0;
        }
        else if(p->left == NULL && p->right == NULL)
        {
                return 1;
        }
        else
        {
                return(count_total(p->left) + count_total(p->right) + 1);
        }
}

void node :: search(int x)
{
        int f = 0;
        node *temp = root;

        while(temp != NULL)
        {
                if(temp->info == x)
                {
                        f = 1;
                        break;
```

```
		}
		else if(temp->info > x)
		{
			temp = temp->left;
		}
		else if(temp->info < x)
		{
			temp = temp->right;
		}
	}
	if(f == 1)
	{
		cout<<"\nNode is found...\n";
	}
	else
	{
		cout<<"\nNode is NOT found...\n";
	}
}

void main()
{
	int ch = 0, x;
	node obj;
	clrscr();

	do
	{
		cout<<"\nEnter your choice : ";
		cout<<"\n1. Insert\n2. Count-leaf\n3. Count-total\n4.
Search\n5. Exit.\n";
		cin>>ch;

		switch(ch)
		{
			case 1:
				cout<<"\nEnter node : ";
				cin>>x;
				obj.insert(x);
```

```
                    break;

                case 2:
                    x = obj.count_leaf(root);
                    cout<<"\nLeaf nodes in tree : "<<x<<"\n";
                    break;

                case 3:
                    x = obj.count_total(root);
                    cout<<"\nTotal nodes in tree : "<<x<<"\n";
                    break;

                case 4:
                    cout<<"\nEnter searching node : ";
                    cin>>x;
                    obj.search(x);
                    break;

                case 5:
                    cout<<"\nProgram stoped...\n";
                    break;

                default:
                    cout<<"\nWrong choice...\n";
                    break;
            }
        }while(ch != 5);
}
```

4) Write a program to implement binary search tree with following operations. 1) Insert( ) 2) find_max( ) 3) find_min( ) 4) display_even( ) 5) display_odd( )

```
#include<iostream.h>
#include<conio.h>

class node
{
	node *left, *right;
	int info;
	public:
		node* create();
		void insert(int);
		void find_max();
		void find_min();
		void display_even(node*);
		void display_odd(node*);
}*root;

node* node :: create()
{
	node *p = new node;
	return(p);
}

void node :: insert(int x)
{
	node *temp, *p = create();
	p->left = NULL;
	p->info = x;
	p->right= NULL;

	if(root == NULL)
	{
		root = p;
	}

	else
```

```
        {
                temp = root;
                while(temp != NULL)
                {
                        if(p->info < temp->info)
                        {
                                if(temp->left == NULL)
                                {
                                        temp->left = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->left;
                                }
                        }
                        else if(p->info > temp->info)
                        {
                                if(temp->right == NULL)
                                {
                                        temp->right = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->right;
                                }
                        }
                        else
                        {
                                cout<<"\nDo Not repeat node...\n";
                                break;
                        }
                }
        }
}
```

```
void node :: find_max()
{
        node *temp = root;

        while(temp->right != NULL)
        {
                temp = temp->right;
        }

        cout<<"\nMax node is : "<<temp->info<<"\n";
}

void node :: find_min()
{
        node *temp = root;

        while(temp->left != NULL)
        {
                temp = temp->left;
        }

        cout<<"\nMin node is : "<<temp->info<<"\n";
}

void node :: display_even(node *p)
{
        if(p != NULL)
        {
                if(p->info % 2 != 1)
                {
                        cout<<p->info<<"\t";
                }
                display_even(p->left);
                display_even(p->right);
        }
}
```

```
void node :: display_odd(node *p)
{
        if(p != NULL)
        {
                if(p->info % 2 == 1)
                {
                        cout<<p->info<<"\t";
                }
                display_odd(p->left);
                display_odd(p->right);
        }
}

void main()
{
        int ch = 0, x;
        node obj;

        clrscr();

        do
        {
                cout<<"\nEnter your choice : ";
                cout<<"\n1. Insert\n2. Find max\n3. Find min\n4. Display
even\n5. Display odd\n6. Exit.\n";
                cin>>ch;

                switch(ch)
                {
                        case 1:
                                cout<<"\nEnter node : ";
                                cin>>x;
                                obj.insert(x);
                                break;

                        case 2:
                                obj.find_max();
                                break;
```

```
                    case 3:
                            obj.find_min();
                            break;

                    case 4:
                            cout<<"\nEven nodes : ";
                            obj.display_even(root);
                            break;

                    case 5:
                            cout<<"\nOdd nodes : ";
                            obj.display_odd(root);
                            break;

                    case 6:
                            cout<<"\nProgram stoped...\n";
                            break;

                    default:
                            cout<<"\nWrong choice...\n";
                            break;
            }
        }while(ch != 6);
}
```

5) Write a menu dr iven program that deletes node from binary search tree. Hint: Menu will look like1: Insert 2) Inorder 3) Delete 4) Exit.

```
#include<iostream.h>
#include<conio.h>

class node
{
        node *left, *right;
        int info;
        public:
                node* create();
                void insert(int);
                void inorder(node*);
                void del_leaf(node*, node*);
                void del_one(node*, node*);
                void del_two(node*);
                void del(int);
}*root;

node* node :: create()
{
        node *p = new node;
        return(p);
}

void node :: insert(int x)
{
        node *temp, *p = create();

        p->left = NULL;
        p->info = x;
        p->right= NULL;

        if(root == NULL)
        {
                root = p;
        }
        else
        {
                temp = root;
                while(temp != NULL)
                {
                        if(p->info < temp->info)
                        {
                                if(temp->left == NULL)
                                {
                                        temp->left = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->left;
                                }
                        }
                        else if(p->info > temp->info)
                        {
                                if(temp->right == NULL)
                                {
                                        temp->right = p;
                                        break;
                                }
                                else
                                {
                                        temp = temp->right;
                                }
                        }
                        else
                        {
                                cout<<"\nDo Not repeat node...\n";
                                break;
                        }
                }
        }
}
```

```
void node :: inorder(node *p)
{
        if(p != NULL)
        {
                inorder(p->left);
                cout<<p->info<<"\t";
                inorder(p->right);
        }
}

void node :: del_leaf(node *p, node *c)
{
        if(c == p->right)
        {
                p->right = NULL;
        }
        else if(c == p->left)
        {
                p->left = NULL;
        }

        delete(c);
        cout<<"\nNode is deleted...\n";
}

void node :: del_one(node *p, node *c)
{
        if(c == p->left)
        {
                if(c->left != NULL)
                {
                        p->left = c->left;
                }
                else
                {
                        p->left = c->right;
                }
        }
        else if(c == p->right)
        {
                if(c->right != NULL)
                {
                        p->right = c->right;
                }
                else
                {
                        p->right = c->left;
                }
        }

        delete(c);
        cout<<"\nNode is deleted...\n";
}

void node :: del_two(node *c)
{
        node *p = c, *lft = c->left;

        while(lft->right != NULL)
        {
                p = lft;
                lft = lft->right;
        }

        c->info = lft->info;

        if(lft->left == NULL && lft->right == NULL)
        {
                del_leaf(p, lft);
        }
        else
        {
                del_one(p, lft);
        }
}
```

```
void node :: del(int x)
{
        int f = 0;
        node *p , *c;
        p = c = root;

        while(c != NULL)
        {
                if(x == c->info)
                {
                        f = 1;
                        break;
                }
                else if(x < c->info)
                {
                        p = c;
                        c = c->left;
                }
                else if(x > c->info)
                {
                        p = c;
                        c = c->right;
                }
        }

        if(f == 1)
        {
                if(c->left == NULL && c->right == NULL)
                {
                        del_leaf(p, c);
                }
                else if(c->left != NULL && c->right != NULL)
                {
                        del_two(c);
                }
                else
                {
                        del_one(p, c);
                }
        }
        else
        {
                cout<<"\nNode is NOT found...\n";
        }
}

void main()
{
        int ch, x;
        node obj;
        clrscr();

        do
        {
                cout<<"\nEnter your choice : ";
                cout<<"\n1. Insert\n2. In-order\n3. Delete\n4. Exit.\n";
                cin>>ch;

                switch(ch)
                {
                        case 1:
                                cout<<"\nEnter node : ";
                                cin>>x;
                                obj.insert(x);
                                break;

                        case 2:
                                cout<<"\nNodes in tree by in-order method : ";
                                obj.inorder(root);
                                break;

                        case 3:
                                cout<<"\nEnter node to delete : ";
                                cin>>x;
                                obj.del(x);
                                break;

                        case 4:
```

```
                                        cout<<"\nProgram stoped...\n";
                                        break;

                                default:
                                        cout<<"\nWrong choice...\n";
                                        break;
                        }
                }while(ch != 4);
        }
```
