**Linear Search**

#include<iostream>

using namespace std;

class Linear{

    public:

    int arr[10];

     int i = 0;

    Linear(){

        getData();

    }

    void getData(){

        i = 0;

        cout << "Enter Values In array : ";

        while(i != 10){

            cin >> arr[i];

            i++;

        }

    }

    int getNumber(){

        int n;

        cout << "Enter Values You want to Search : ";

        cin >> n;

        return n;

    }

    int linearsearch(int n) {

        i = 0;

        while(i != 10){

            if(arr[i] == n) {

                return i;

            }

        i++;

        }

        return -1;

    }

};

int main(){

    Linear l;

    char ch;

    while(1){

        int search = l.getNumber();

        int result = l.linearsearch(search);

        if(result < 0) cout << "Value Not Present in Array List." << endl;

        else cout << "Search Value Located At " << result << " index of the array" << endl;

        cout << "Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : ";

        cin >> ch;

        if(ch != 'c')

            exit(0);

    }

}

**Output :**

PS E:\MCA\MCA SEM 3\DS\40%> .\linear.exe

Enter Values In array : 21

12

34

8

5

40

39

20

11

1

Enter Values You want to Search : 12

Search Value Located At 1 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 34

Search Value Located At 2 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 21

Search Value Located At 0 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 1

Search Value Located At 9 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 5

Search Value Located At 4 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : z

**Binary Search**

#include<iostream>

using namespace std;

class Binary{

    public:

    int arr[10];

     int i = 0;

    Binary(){

        getData();

    }

    void getData(){

        i = 0;

        cout << "Enter Values In array : ";

        while(i != 10){

            cin >> arr[i];

            i++;

        }

    }

    int getNumber(){

        int n;

        cout << "Enter Values You want to Search : ";

        cin >> n;

        return n;

    }

    int binarySearch(int n) {

        i = 0;

        int j = 9;

        while(j >= i){

            int k = (i+j)/2;

            if(arr[k] < n){

                i = k + 1;

            }

            else if (arr[k] > n) {

                j = k - 1;

            }

            else{

                return k;

            }

        }

        return -1;

    }

};

int main(){

    Binary l;

    char ch;

    while(1){

        int search = l.getNumber();

        int result = l.binarySearch(search);

        if(result < 0) cout << "Value Not Present in Array List." << endl;

        else cout << "Search Value Located At " << result << " index of the array" << endl;

        cout << "Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : ";

        cin >> ch;

        if(ch != 'c')

            exit(0);

    }

}

**Output :**

PS E:\MCA\MCA SEM 3\DS\40%> .\Binary.exe

Enter Values In array : 10

12

14

18

19

22

47

49

62

81

Enter Values You want to Search : 12

Search Value Located At 1 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 31

Value Not Present in Array List.

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 62

Search Value Located At 8 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : m

**Expression Tree**

#include<iostream>

#include<cstring>

using namespace std;

class Stack {

    class Value {

        public:

        string data;

        Value \*Next;

        Value(char n) {

            data = n;

            Next = NULL;

        }

    };

    Value \*top;

    int size, count;

    public:

    Stack(int size)

    {

        if (size < 1)

            size = 5;

        this->size = size;

        count = 0;

        top = NULL;

    }

    bool isFull(){

        return (count >= size);

    }

    bool isEmpty(){

        return !top;

    }

    void push(char n){

        if(isFull()){

            cout << "Overflowed" << endl;

            return;

        }

        Value \*val = new Value(n);

        val->Next = top;

        top = val;

        count++;

    }

    void edit(char n){

        if(isEmpty()) {

            cout << "UnderFlowed" << endl;

        }

        top->Next->data += n;

        string poped = pop();

        top->data += poped;

    }

    string pop() {

        string lastdelete = top->data;

        top = top->Next;

        count--;

        return lastdelete;

    }

    void display() {

        if(isEmpty())

            cout << "Empty" << endl;

        else

            cout << top->data << endl;

    }

    bool isOperator(char c)

    {

        if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^') return true;

        return false;

    }

};

int main(){

    char Post\_Expresion[] = "ab+cd/+efg\*-\*";

    int i = 0,length = strlen(Post\_Expresion);

    Stack s(length);

    while(Post\_Expresion[i] != '\0') {

        if(s.isOperator(Post\_Expresion[i])){

            s.edit(Post\_Expresion[i]);

        }

        else {

            s.push(Post\_Expresion[i]);

        }

        i++;

    }

    s.display();

    return 0;

}

**Output :**

PS E:\MCA\MCA SEM 3\DS\40%> .\Expression.exe

a+b+c/d\*e-f\*g

**Binary Search Tree**

#include<iostream>

using namespace std;

struct tree{

    int data;

    tree \*left,\*right;

};

// method declaration

void menu(struct tree \*);

struct tree \* construct(struct tree \*);

void preorder(struct tree \*);

void postorder(struct tree \*);

void inorder(struct tree \*);

void minmax(struct tree \*,int []);

void searching(struct tree \*);

bool advancesearch(struct tree \*,int);

struct tree \* insert(struct tree \*);

struct tree \* deleteNode(struct tree \*,int);

struct tree \* minfromRight(struct tree \* );

int main(){

    struct tree \*head = NULL;

    menu(head);

    return 0;

}

void menu(struct tree \*head) {

    int listen = 0;

    while(1){

        cout << endl << "1. Create Tree" << endl << "2. PreOrder Traversal" << endl << "3. PostOrder Traversal" << endl << "4. InOrder Traversal" << endl << "5. Insertion" << endl << "6. Searching" << endl << "7. Find Minimum & Maximum" << endl << "8. Deletion" << endl << "9. Exit"  << endl << "Enter Your Choice : ";

        cin >> listen;

        switch (listen)

        {

        case 1:

            head = construct(head);

            break;

        case 2:

            preorder(head);

            break;

        case 3:

            postorder(head);

            break;

        case 4:

            inorder(head);

            break;

        case 5:

            head = insert(head);

            break;

        case 6:

            searching(head);

            break;

        case 7:

        {

            int m[2] = {head->data};

            minmax(head,m);

            cout << "Minimum From Tree is : " << m[0] << endl;

            cout << "Maximum From Tree is : " << m[1] << endl;

        }

        break;

        case 8:

            {

                int value;

                cout << "Enter The Number You want to delete : ";

                cin >> value;

                head = deleteNode(head,value);

            }

            break;

        case 9:

            exit(0);

        default:

            cout << "Select Valid Options." << endl;

            menu(head);

        }

    }

}

void preorder(struct tree \*head) {

    struct tree \*temp;

    temp = head;

    if(temp == NULL)

        return;

    cout << temp->data << " ";

    preorder(temp->left);

    preorder(temp->right);

}

void postorder(struct tree \*head) {

    struct tree \*temp;

    temp = head;

    if(temp == NULL)

        return;

    postorder(temp->left);

    postorder(temp->right);

    cout << temp->data << " ";

}

void inorder(struct tree \*head) {

    struct tree \*temp;

    temp = head;

    if(temp == NULL)

        return;

    inorder(temp->left);

    cout << temp->data << " ";

    inorder(temp->right);

}

void minmax(struct tree \*head,int m[]) {

    struct tree \*temp;

    temp = head;

    if(temp == NULL)

        return;

    preorder(temp->left);

    if(temp->data < m[0])

        m[0] = temp->data;

    if(temp->data > m[1])

        m[1] = temp->data;

    preorder(temp->right);

}

void searching(struct tree \*head) {

    bool found = false;

    int getNum;

    cout << "Enter the Number You want to find from tree : ";

    cin >> getNum;

    if(head == NULL) {

        cout << "The Tree is Empty." << endl;

        return;

    }

    found = advancesearch(head,getNum);

    if(found)

        cout << "The Number You searching is present in the tree" << endl;

    else

        cout << "The Number You searching is not present in the tree" << endl;

}

bool advancesearch(struct tree \*head,int getNum) {

    struct tree \*temp;

    temp = head;

    bool found = false;

    if(temp == NULL)

        return false;

    advancesearch(temp->left,getNum);

    if(temp->data == getNum) {

        return true;

    }

    advancesearch(temp->right,getNum);

    return false;

}

struct tree \* construct(struct tree \*head) {

    int i = 0;

    cout << "Total Data You Want : ";

    cin >> i;

    while(i > 0) {

        head = insert(head);

        i--;

    }

    return head;

}

struct tree \* insert(struct tree \*head) {

    bool target = false;

    int input;

    struct tree \*n,\*temp;

    temp = head;

    cout << "Enter Value : ";

    cin >> input;

    n = (struct tree \*)malloc(sizeof(struct tree));

    if(head == NULL){

        head = n;

    }

    else {

        while(!target) {

            if(temp->data > input && temp->left != NULL){

                temp = temp->left;

            }

            else if(temp->data < input && temp->right != NULL ) {

                temp = temp->right;

            }

            if((temp->data < input && temp->right == NULL) || (temp->data > input && temp->left == NULL)) {

                target = true;

            }

        }

        if(temp->data < input) {

            temp->right = n;

        }

        else {

            temp->left = n;

        }

    }

    n->data = input;

    n->left = n->right = NULL;

    return head;

}

struct tree \* deleteNode(struct tree\* head, int deletethis) {

    struct tree \*temp;

    temp = head;

    if (temp == NULL)

        return temp;

    if (deletethis < temp->data)

        temp->left = deleteNode(temp->left, deletethis);

    else if (deletethis > temp->data)

        temp->right = deleteNode(temp->right, deletethis);

    else

    {

        if (temp->left == NULL)

        {

            struct tree \*temp2 = temp->right;

            free(temp);

            return temp2;

        }

        else if (temp->right == NULL)

        {

            struct tree \*temp2 = temp->left;

            free(temp);

            return temp2;

        }

        struct tree\* temp2 = minfromRight(temp->right);

        temp->data = temp2->data;

        temp->right = deleteNode(temp->right, temp2->data);

    }

    return temp;

}

struct tree \* minfromRight(struct tree\* temp) {

    struct tree\* current = temp;

    while (current && current->left != NULL)

        current = current->left;

    return current;

}

**Output :**

PS E:\MCA\MCA SEM 3\DS\40%> .\BST.exe

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 1

Total Data You Want : 5

Enter Value : 4

Enter Value : 2

Enter Value : 3

Enter Value : 7

Enter Value : 5

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 2

4 2 3 7 5

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 3

3 2 5 7 4

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 4

2 3 4 5 7

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 5

Enter Value : 6

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 4

2 3 4 5 6 7

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 6

Enter the Number You want to find from tree : 5

The Number You searching is present in the tree

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 7

Minimum From Tree is : 2

Maximum From Tree is : 7

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 8

Enter The Number You want to delete : 5

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 4

2 3 4 6 7

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 2

4 2 3 7 6

1. Create Tree

2. PreOrder Traversal

3. PostOrder Traversal

4. InOrder Traversal

5. Insertion

6. Searching

7. Find Minimum & Maximum

8. Deletion

9. Exit

Enter Your Choice : 9