

## Experiment No. 1

// To implement various searching and sorting algorithms on database of students implemented using array of structure.

```
#include <iostream>
#include <string.h>
using namespace std;

struct student
{
    char name[25];
    int roll_no;
    float sgpa;
} s[20];

void input(int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << "Enter details for student no : " << i + 1 << "\n";
        cout << "Name : ";
        cin >> s[i].name;
        cout << "Roll No. : ";
        cin >> s[i].roll_no;
        cout << "SGPA : ";
        cin >> s[i].sgpa;
        cout << endl;
    }
}

void display(int n)
{
    cout << "\nSR.no \tName \tRoll No. \tSGPA \n";
    cout << ".....\n";
    for (int i = 0; i < n; i++)
    {
        cout << "\n"
            << i + 1;
        cout << "\t" << s[i].name;
        cout << "\t"
            << "\t" << s[i].roll_no;
        cout << "\t" << s[i].sgpa;
    }
    cout << endl;
}

void insertionsort(int n){
    struct student present ;
    int i, j;

    for(i=1; i<n; i++){
        present = s[i];
        j=i-1 ;
```

```

while (j>=0 && strcmp(s[j].name, present.name) > 0 ){

    s[j+1] = s[j] ;
    j-- ;
}
s[j+1] = present ;
}

}

void linearsearch(int n)
{
    float key;
    cout << "Enter SGPA you want to search ";
    cin >> key;
    bool flag = 0;
    for (int i = 0; i < n; i++)
    {
        if (key == s[i].sgpa)
        {
            flag = 1;
            cout << "Name of student : " << s[i].name << endl;
            cout << "Roll No. of student : " << s[i].roll_no << endl;
            cout << "SGPA of student : " << s[i].sgpa << endl;
            cout << "\n";
        }
    }
    if (flag == 0)
    {
        cout << " Not Found " << endl;
    }
}

void binarysearch(int n)
{
    int low = 0;
    n -= 1;
    char Name[15];
    cout << "Enter Name you want to search ";
    cin >> Name;
    int flag = 0;
    while (low <= n)
    {
        int mid = (low + n) / 2;
        int comp = strcmp(Name, s[mid].name);
        if (comp == 0)
        {
            flag = 1;
            cout << "Name of student : " << s[mid].name << endl;
            cout << "Roll No. of student : " << s[mid].roll_no << endl;
            cout << "SGPA of student : " << s[mid].sgpa << endl;
            return;
        }
        else if (comp > 0)
        {

```

```

        low = mid + 1;
    }
    else
    {
        n = mid - 1;
    }
}
if (flag == 0)
{
    cout << "Not Found " << endl;
}
}

```

```

void Bubble_sort(int n)
{
    float temp1;
    char temp2[10];
    int counter = 1, i, j, temp3;
    for (i = 1; i < n; i++)
    {
        for (j = 0; j < n - counter; j++)
        {
            if (s[j].roll_no > s[j + 1].roll_no)
            {
                temp1 = s[j].sgpa;
                strcpy(temp2, s[j].name);
                temp3 = s[j].roll_no;

                s[j].sgpa = s[j + 1].sgpa;
                strcpy(s[j].name, s[j + 1].name);
                s[j].roll_no = s[j + 1].roll_no;

                s[j + 1].sgpa = temp1;
                strcpy(s[j + 1].name, temp2);
                s[j + 1].roll_no = temp3;
            }
        }
        counter++;
    }

    cout << " Sorting Completed " << endl;
}

```

```

void swap(struct student s[], int i, int j)
{
    float temp1;
    char temp2[20];
    int temp3;

    temp3 = s[j].roll_no;
    temp1 = s[j].sgpa;
    strcpy(temp2, s[j].name);

```

```

s[j].roll_no = s[i].roll_no;
s[j].sgpa = s[i].sgpa;
strcpy(s[j].name, s[i].name);

s[i].roll_no = temp3;
s[i].sgpa = temp1;
strcpy(s[i].name, temp2);
}

int partition(struct student s[], int low, int high)
{
    int pivot = low;
    int i = low + 1;
    int j = high;

    while (s[pivot].sgpa > s[i].sgpa || s[pivot].sgpa < s[j].sgpa)
    {
        if (s[pivot].sgpa > s[i].sgpa)
        {
            i++;
        }
        else if (s[pivot].sgpa < s[j].sgpa)
        {
            j--;
        }
    }

    if (j < i)
    {
        swap(s, j, pivot);
        return j;
    }
    else
    {
        swap(s, i, j);
        swap(s, i, pivot);
        return i;
    }
}

void quick_Sort(struct student s[], int low, int high)
{
    if (low < high)
    {
        int pos = partition(s, low, high);
        quick_Sort(s, low, pos - 1);
        quick_Sort(s, pos + 1, high);
    }
}

int main()
{

```

```

int n, ch;
cout << "Enter the no. of students";
cin >> n;

do
{
    cout << "\n1. Accept Record \n2. display Record \n3. linear search SGPA" ;
    cout<<"\n4. Binary search Name \n5. sort Roll No. \n6. quick sort\n7. insertion sort\n8. Exit\n";
    cout << "Enter Your choice ";
    cin >> ch;

    switch (ch)
    {
        case 1:
            input(n);
            break;
        case 2:
            display(n);
            break;
        case 3:
            linearsearch(n);
            break;
        case 4:
            binarysearch(n);
            break;
        case 5:
            Bubble_sort(n);
            break;
        case 6:
            quick_Sort(s, 0, n-1);
            display(10) ;
            break;
        case 7:
            insertionsort(n);
            display(n) ;
            break;
        case 8:
            cout << " You are out ";
            break;
        default:
            cout << "Invalid choice ";
            break;
    }

} while (ch != 8);

return 0;

```

Output:

```

C:\Windows\System32\cmd.exe - a
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign1dsa.cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a
Enter the no. of students 15

1. Accept Record
2. display Record
3. linear search SGPA
4. Binary search Name
5. sort Roll No.
6. quick sort
7. insertion sort
8. Exit
Enter Your choice 6

SR.no    Name    Roll No.    SGPA
.....
1         0         0
2         0         0
3         0         0
4         0         0
5         0         0
6         0         0
7         0         0
8         0         0
9         0         0
10        0         0

1. Accept Record
2. display Record
3. linear search SGPA
4. Binary search Name
5. sort Roll No.
6. quick sort
7. insertion sort
8. Exit
Enter Your choice

```

## Experiment No. 2

// Implementation of Stack ADT and expression conversion

```
#include<iostream>
#include "stack.h"
#include<math.h>

using namespace std ;

int precedence(char operand){

    if(operand=='^' || operand=='$'){
        return 4;
    }

    else if (operand=='*' || operand=='/'){
        return 3;
    }

    else if (operand=='+' || operand=='-'){
        return 2;
    }

    else if(operand=='('){
        return 1;
    }

    else {
        return -1;
    }
}

void convert_inftopost(string str)
{
    stack st(str.length());
    string result;
    for (int i = 0; i < str.length(); i++)
    {
        char operand = str[i];
        if (operand == ' ')
        {
            continue;
        }

        else if (isalnum(operand))
        {
            result += operand;
        }

        else if(operand=='(')
        {
            st.push(operand) ;
        }
    }
}
```

```

else if (operand=='')
{
    operand = st.pop();
    while (operand!='(')
    {
        result += operand;
        operand = st.pop();
    }
}
else{
    while((!st.isEmpty()) && (preference(st.gettop())>=preference(operand))){
        result += st.pop();
    }

    st.push(operand);
}

}

while (!st.isEmpty())
{
    result += st.pop() ;
}

cout<<result<<endl ;

}

void convert_infthopre(string str)
{
    stack st(str.length());
    string result;

    for (int i =(str.length()-1); i >=0; i--)
    {
        char operand = str[i];
        if (operand == ' ')
        {
            continue;
        }

        else if (isalnum(operand))
        {
            result += operand;
        }

        else if(operand=='('){
            operand = st.pop();
            while (operand!='')
            {
                result += operand;
                operand = st.pop();
            }
        }
    }
}

```



```

else if (operand=='') {
    st.push(operand);
}

else {
    while ((!st.isEmpty()) && (preference(st.gettop()) > preference(operand))) {
        result += st.pop();
    }

    st.push(operand);
}

}

while (!st.isEmpty())
{
    result += st.pop();
}

for(int i=(result.length()-1); i>=0; i--){
    cout<<result[i];
}
cout<<endl;
}

int evaluate_prefix(string str){
    stack st(str.length());

    for(int i=(str.length()-1); i>=0; i--){

        if(isalnum(str[i])){
            st.push(str[i]-'0');
        }
        else{

            int operator1 = st.gettop();
            st.pop();
            int operator2 = st.gettop();
            st.pop();

            if (str[i]=='+' ){
                st.push(operator1+operator2);
            }
            else if (str[i]=='-'){
                st.push(operator1-operator2);
            }
            else if (str[i]=='*'){
                st.push(operator1*operator2);
            }
            else if (str[i]=='/'){
                st.push(operator1/operator2);
            }
        }
    }
}

```

```

        else if (str[i]=='^' || str[i]=='$'){
            int result = pow(operator1,operator2);
            st.push(result);
        }
        else if (str[i]=='%'){
            st.push(operator1%operator2);
        }
    }
}

return st.gettop() ;
}

```

```

int evaluate_postfix(string str){
    stack st(str.length()) ;
    for(int i=0; i<str.length(); i++){

        if(isalnum(str[i])){
            st.push(str[i]-'0');
        }
        else{

            int operator2 = st.gettop();
            st.pop() ;
            int operator1 = st.gettop();
            st.pop() ;

            if (str[i]=='+'){
                st.push(operator1+operator2);
            }
            else if (str[i]=='-'){
                st.push(operator1-operator2);
            }
            else if (str[i]=='*'){
                st.push(operator1*operator2);
            }
            else if (str[i]=='/'){
                st.push(operator1/operator2);
            }
            else if (str[i]=='^' || str[i]=='$'){
                int result = pow(operator1,operator2);
                st.push(result);
            }
            else if (str[i]=='%'){
                st.push(operator1%operator2);
            }
        }
    }

    return st.gettop() ;
}

```

```

int main(){

```

```

string str ;
int ch ;

do
{
    cout<<"1.Convert infix to prefix\n2.Convert infix to postfix\n3.Evaluate Prefix
expression\n4.Evaluate Postfix expression\n5.Exit\nEnter the choice to perform operation - ";
    cin>>ch;

    switch(ch)
    {
        case 1:
            cout<<"Enter infix Expression : \n" ;
            cin>>str ;
            cout<<"Converted Prefix Expression- \n";
            convert_inftopre(str) ;
            break;

        case 2:
            cout<<"Enter infix Expression : \n" ;
            cin>>str ;
            cout<<"Converted Postfix Expression- \n";
            convert_inftopost(str) ;
            break;

        case 3:
            cout<<"Enter Prefix Expression : \n";
            cin>>str;
            cout<<"Result : \n"<<evaluate_prefix(str)<<endl;
            break;

        case 4:
            cout<<"Enter Postfix Expression : \n";
            cin>>str;
            cout<<"Result : \n"<<evaluate_postfix(str)<<endl;
            break;
        case 5:
            cout<<"You are out\n";
            break;
        default:
            cout<<"Invalid choice\n";
            break;

    }

} while (ch!=5);

return 0;
}
Output:

```

```
C:\Windows\System32\cmd.exe - a
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign2dsa(1).cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a
1.Convert infix to prefix
2.Convert infix to postfix
3.Evaluate Prefix expression
4.Evaluate Postfix expression
5.Exit
Enter the choice to perform operation - 2
Enter infix Expression: a*b+c/d
Converted Postfix Expression: ab*cd/+
1.Convert infix to prefix
2.Convert infix to postfix
3.Evaluate Prefix expression
4.Evaluate Postfix expression
5.Exit
Enter the choice to perform operation -
```

### Experiment No. 3

#### // Implementation of Circular Queue using Array

```
#include<iostream>
using namespace std;

class circular_queue{
private:
    int front, rear, size ;
    int *arr ;

public :
    circular_queue(int s){
        size = s ;
        front = -1 ;
        rear = -1 ;
        arr = new int[s] ;
    }

    bool Full(){
        if ((rear+1)%size == front)
        {
            return true ;
        }

        return false ;
    }

    bool Empty(){
        if(front==-1 && rear == -1){
            return true;
        }

        return false;
    }

    void enqueue(int element){
        if(Full()){
            cout<<"The Queue is full. Cannot Enter the element."<<endl ;
            return ;
        }

        if(front==-1 && rear == -1){
            front++ ;
            rear++ ;
        }
        else{
            rear = (rear+1)%size ;
        }

        arr[rear] = element ;
        cout<<"Element "<<element<<" is inserted in the queue."<<endl ;
    }
}
```

```

void dequeue(){
    if(Empty()){
        cout<<"The Queue is empty."<<endl ;
        return ;
    }

    if(front==rear){
        front = -1 ;
        rear = -1 ;
        cout<<"Deleted Successfully."<<endl ;
    }
    else{
        front = (front+1)%size ;
        cout<<"Deleted Successfully."<<endl ;
    }

}

void Display(){
    if(Empty()){
        cout<<"The Queue is Empty."<<endl ;
    }
    else{
        cout<<"Elements in queue : " ;
        int i=front ;
        do{
            cout<<arr[i]<<" " ;
            i = (i+1)%size ;

        }while(i != (rear+1)%size);
        cout<<endl ;
    }
}

};

int main(){
    int ch, size, element ;

    cout<<"Enter the size of circular queue : " ;
    cin>>size ;
    circular_queue que(size) ;
    do
    {
        cout<<"\n1. Display the queue \n2. Insert an element in queue \n3. Delete an element from queue
\n4. Exit \nEnter your choice : " ;
        cin>>ch ;
        switch(ch)
        {
            case 1:
                que.Display();
                break;
            case 2:
                cout<<"Enter the element you want to insert in the queue - \n";

```

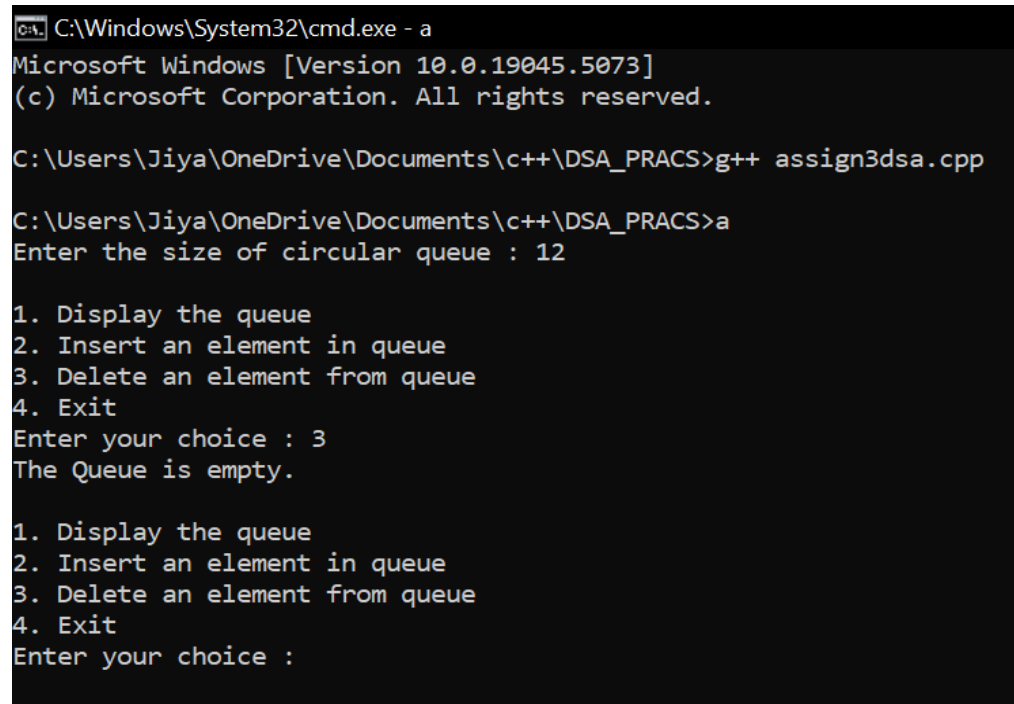
```

        cin>>element;
        que.enqueue(element);
        break;
    case 3:
        que.dequeue();
        break;
    case 4:
        cout<<"Exit\n";
        break;
    default:
        cout<<"Invalid Choice\n";
        break;
}

} while (ch!=4);

return 0;
}
Output:

```



```

C:\Windows\System32\cmd.exe - a
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign3dsa.cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a
Enter the size of circular queue : 12

1. Display the queue
2. Insert an element in queue
3. Delete an element from queue
4. Exit
Enter your choice : 3
The Queue is empty.

1. Display the queue
2. Insert an element in queue
3. Delete an element from queue
4. Exit
Enter your choice :

```

#### Experiment No. 4

// To construct an expression tree and perform recursive and non-recursive traversals.

```
#include<iostream>
#include <stack>
using namespace std ;

struct node{
    char data ;
    node* left, *right;

    node(char value){
        data = value ;
        left = right = NULL ;
    }
};

class tree{
public :
    struct node * root ;

    tree(){
        root = NULL ;
    }
    node* expresion_tree_by_post(){

        stack<node*> st ;
        char Exp[21] ;
        cout<<"Enter postfix Expression to create tree : " ;
        cin>>Exp ;
        for(int i=0; Exp[i]!='\0';i++){
            if(isalnum(Exp[i])){
                node* temp ;
                temp = new node(Exp[i]);
                st.push(temp);
            }
            else{                // operator
                root = new node(Exp[i]);
                root->right = st.top();
                st.pop();
                root->left = st.top();
                st.pop();
                st.push(root) ;
            }
        }
        root = st.top();
        st.pop() ;
        return root ;
    }

    node* expresion_tree_by_prefix(){

        stack<node*> st ;
```



```

string Exp ;
cout<<"Enter prefix Expression to create tree : " ;
cin>>Exp ;
for(int i=(Exp.length()-1); i>=0;i--){
    if(isalnum(Exp[i])){
        node* temp ;
        temp = new node(Exp[i]);
        st.push(temp);
    }
    else{          // operator
        root = new node(Exp[i]);
        root->left = st.top();
        st.pop();
        root->right = st.top();
        st.pop();
        st.push(root) ;
    }
}
root = st.top();
st.pop() ;
return root ;
}

```

```

void preOrder_nonrec(node* root){
    stack<node *> st ;
    st.push(root);

```

```

    while(!st.empty()) {
        root = st.top();
        st.pop() ;
        cout<<root->data<<" " ;
        if(root->left && root->right){
            st.push(root->right) ;
            st.push(root->left) ;
        }
        else if(root->left){
            st.push(root->left) ;
        }
        else if(root->right){
            st.push(root->right) ;
        }
    }
}

```

```

}
void inOrder_nonrec(node * root){
    stack<node *> st ;
    node *current = root ;
    while(current != NULL || (!st.empty())){
        while (current != NULL ){
            st.push(current) ;
            current = current->left ;
        }

```

```

        node * popped ;
        popped = st.top() ;
        st.pop();

```

```

        cout<<poped->data<<" ";
        current = poped->right ;
    }
}
void postOrder_nonrec(node* root){
    stack<node *> st ;
    st.push(NULL) ;

    while (!st.empty())
    {
        while(root != NULL){
            if(root->right != NULL){
                st.push(root->right) ;
            }
            st.push(root) ;
            root = root->left ;
        }

        root = st.top();
        st.pop() ;
        if(root->right != NULL && root->right == st.top()){
            st.pop() ;
            st.push(root) ;
            root = root->right ;
        }
        else
        {
            if(st.top() == NULL){
                st.pop() ;
            }
            cout<<root->data<<" ";
            root =NULL ;
        }

    }

}

void preorder(node* root) {
    if(root!=NULL){
        cout<<root->data<<" ";
        preorder(root->left);
        preorder(root->right) ;
    }
}

void inorder(node* root){
    if(root!=NULL){
        inorder(root->left) ;
        cout<<root->data<<" ";
        inorder(root->right) ;
    }
}

void postorder(node* root) {

```

```

        if(root!=NULL){
            postorder(root->left) ;
            postorder(root->right) ;
            cout<<root->data<<" " ;
        }
    }

};

int main(){

    tree t ;
    int ch;
    node* root ;
    do
    {
        cout << "\n1. create tree using Postfix Expression \n2. create tree using Prefix Expression ";
        cout<<"\n3. Non Recursive Preorder \n4. Non Recursive inorder \n5. Non Recursive Postorder ";
        cout<<"\n6. Recursive Preorder \n7. Recursive inorder \n8. Recursive Postorder \n9. Exit";
        cout << "\nEnter your choise : ";
        cin >> ch;
        switch (ch)
        {
            case 1 :
                root = t.expresion_tree_by_post() ;
                cout<<"Press 3 to 8 to see traversal of Expression tree "<<endl ;
                break;
            case 2 : root = t.expresion_tree_by_prefix();
                cout<<"Press 3 to 8 to see traversal of Expression tree "<<endl ;
                break;
            case 3 :
                {
                    cout <<"Non recursive Preorder : " ;
                    t.preOrder_nonrec(root) ;
                    cout<<endl ;
                }
                break;
            case 4 :
                {
                    cout <<"Non recursive inorder : " ;
                    t.inOrder_nonrec(root);
                    cout<<endl ;
                }
                break;
            case 5 :
                {
                    cout <<"Non Recursive Postorder : " ;
                    t.postOrder_nonrec(root);
                    cout<<endl ;
                }
                break;
            case 6 :
                {
                    cout <<"Recursive Preorder : " ;
                    t.preorder(root);

```

```

        cout<<endl ;
    }
    break;
case 7 :
{
    cout <<"Recursive inorder : " ;
    t.inorder(root);
    cout<<endl ;
}
    break;
case 8 :
{
    cout <<"Recursive Postorder : " ;
    t.postorder(root);
    cout<<endl ;
}
    break;
case 9 :
    cout<<"You are out.";
    break;
}
} while (ch != 9);

return 0;
}

```

Output:

```
C:\Windows\System32\cmd.exe - a
9. Exit
Enter your choise : 5
Non Recursive Postorder :
C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign4dsa.cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a

1. create tree using Postfix Expression
2. create tree using Perfix Expression
3. Non Recursive Preorder
4. Non Recursive inorder
5. Non Recursive Postorder
6. Recursive Preorder
7. Recursive inorder
8. Recursive Postorder
9. Exit
Enter your choise : 1
Enter postfix Expression to create tree : 23+
Press 3 to 8 to see traversal of Expression tree

1. create tree using Postfix Expression
2. create tree using Perfix Expression
3. Non Recursive Preorder
4. Non Recursive inorder
5. Non Recursive Postorder
6. Recursive Preorder
7. Recursive inorder
8. Recursive Postorder
9. Exit
Enter your choise :
```

## Experiment No. 5

// Implementation of binary search tree DETAILED

```
#include<iostream>
```

```
using namespace std;
```

```
struct bstnode
```

```
{
    int data;
    struct bstnode *leftchild, *rightchild;
};
```

```
struct bstnode *newNode(char val)
```

```
{
    bstnode* temp = new bstnode();
    temp->data = val;
    temp->leftchild = temp->rightchild = NULL;
    return temp;
}
```

```
void inorder(struct bstnode *root)
```

```
{
    if (root != NULL)
    {
        inorder(root->leftchild);
        cout<<root->data<<" ";
        inorder(root->rightchild);
    }
}
```

```
struct bstnode* insert(struct bstnode* node, int key)
```

```
{
    if (node == NULL) return newNode(key);

    if (key < node->data)
        node->leftchild = insert(node->leftchild, key);
    else
        node->rightchild = insert(node->rightchild, key);
    return node;
}
```

```
struct bstnode* search(struct bstnode* root, int key)
```

```
{
    if (root == NULL || root->data == key)
        return root;

    if (root->data < key)
        return search(root->rightchild, key);
    else
        return search(root->leftchild, key);
}
```

```
struct bstnode * minValueNode(struct bstnode* node)
```

```
{
```

```

    struct bstnode* current = node;
    while (current && current->leftchild != NULL)
        current = current->leftchild;
    return current;
}

struct bstnode* deleteNode(struct bstnode* root, int key)
{
    if (root == NULL)
        return root;
    if (key < root->data)
        root->leftchild = deleteNode(root->leftchild, key);
    else if (key > root->data)
        root->rightchild = deleteNode(root->rightchild, key);
    else
    {
        if (root->leftchild == NULL)
        {
            struct bstnode *temp = root->rightchild;
            free(root);
            return temp;
        }

        else if (root->rightchild == NULL)
        {
            struct bstnode *temp = root->leftchild;
            free(root);
            return temp;
        }
        else
        {
            struct bstnode* temp = minValueNode(root->rightchild);
            root->data = temp->data;
            root->rightchild = deleteNode(root->rightchild, temp->data);
        }
    }
    return root;
}

void mirror(struct bstnode* node)
{
    if (node == NULL)
        return;
    else
    {
        struct bstnode* temp;
        mirror(node->leftchild);
        mirror(node->rightchild);

        temp = node->leftchild;
        node->leftchild = node->rightchild;
        node->rightchild = temp;
    }
}

```

```

struct bstnode* copy(struct bstnode *root)
{
    bstnode *root2;
    if(root==NULL)
        return NULL;
    root2=new bstnode;
    root2->leftchild=copy(root->leftchild);
    root2->rightchild=copy(root->rightchild);
    root2->data=root->data;

    return root2;
}

int Maxdepth(struct bstnode *root)
{
    if(root==NULL)
    {
        return 0;
    }
    else
    {
        int depth1=Maxdepth(root->leftchild);
        int depth2=Maxdepth(root->rightchild);
        if(depth1>depth2)
            return (depth1+1);
        else
            return (depth2+1);
    }
}

int main()
{
    struct bstnode *root = NULL;
    struct bstnode *root2=NULL;
    struct bstnode *root1=NULL;
    struct bstnode *root3=NULL;
    int ch,n,d,depth;
    do
    {
        cout<<"\n1.Insert the elements in the BST.\n2.Delete elements from the BST.\n3.Search an
element from the BST.\n4.Traverse through the BST.\n5.Depth of BST.\n6.Display copy of
BST.\n7.Display the Mirror image of BST.\n8.EXIT.\nEnter your choice - "<<endl;
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"\nEnter total number of elements in the BST - ";
                cin>>n;
                cout<<"\nEnter the values to create BST: ";
                for(int i=0; i<n; i++)
                {
                    cin>>d;
                    root = insert(root, d);
                }

```



```

        break;
    case 2:
        cout<<"\nEnter the element to be deleted: ";
        cin>>d;
        root3=deleteNode(root, d);
        break;
    case 3:
        cout<<"\nEnter the element to be searched: ";
        cin>>d;
        root1=search(root, d);
        if(root1!=NULL)
            cout<<"\nElement Found in BST!!";
        else
            cout<<"\nElement not Found in BST!!";
        break;
    case 4:
        cout<<"\nTraversal of BST: ";
        inorder(root);
        break;
    case 5:
        depth=Maxdepth(root);
        cout<<"The depth of BST is: "<<depth;
        break;
    case 6:
        root2=copy(root);
        cout<<"The copy of BST is: ";
        inorder(root2);
        break;
    case 7:
        mirror(root);
        cout <<"\nInorder traversal of the mirror BST is: ";
        inorder(root);
        mirror(root);
        break;
    case 8:
        cout<<"YOU ARE OUT";
        break;
    default:
        cout<<"\nInvalid choice";
    }
} while(ch!=8);

return 0;
}
Output:

```

```
C:\Windows\System32\cmd.exe - a
1.Insert the elements in the BST.
2.Delete elements from the BST.
3.Search an element from the BST.
4.Traverse through the BST.
5.Depth of BST.
6.Display copy of BST.
7.Display the Mirror image of BST.
8.EXIT.
Enter your choice -
1

Enter total number of elements in the BST - 6

Enter the values to create BST: 2
4
5
6
7
8

1.Insert the elements in the BST.
2.Delete elements from the BST.
3.Search an element from the BST.
4.Traverse through the BST.
5.Depth of BST.
6.Display copy of BST.
7.Display the Mirror image of BST.
8.EXIT.
Enter your choice -
```

## Experiment No. 6

// Implementation In-order Threaded Binary Tree (TBT)

```
#include <iostream>
```

```
using namespace std;
```

```
struct Node
```

```
{
    struct Node *left,*right;
    int data;
```

```
    bool leftthread;
```

```
    bool rightthread;
```

```
};
```

```
struct Node *Insert(struct Node*root, int key)
```

```
{
    struct Node *node = root;
    struct Node *par = NULL;
    while(node != NULL)
    {
        if(key == (node->data))
        {
            cout<<"Duplicate key"<<endl;
            return root;
        }
```

```
        par = node;
```

```
        if(key < (node->data))
        {
            if(node->leftthread == false)
            {
                node = node->left;
            }
            else
                break;
        }
```

```
        else
        {
            if(node->rightthread == false)
            {
                node = node->right;
            }
            else
                break;
        }
```

```
    }
```

```
    struct Node *temp = new Node;
```

```
    temp->data = key;
```

```
    temp->leftthread = true;
```

```
    temp->rightthread = true;
```

```

if(par == NULL)
{
    root = temp;
    temp->left = NULL;
    temp->right = NULL;
}
else if (key < (par -> data))
{
    temp -> left = par -> left;
    temp -> right = par;
    par -> leftthread = false;
    par -> left = temp;
}
else
{
    temp -> left = par;
    temp -> right = par -> right;
    par -> rightthread = false;
    par -> right = temp;
}

return root;
}

struct Node *InorderSuccessor(struct Node *node)
{
    if(node->rightthread == true)
    {
        return (node->right);
    }
    else
    {
        node = node->right;
        while(node->leftthread == false)
        {
            node = node->left;
        }
    }

    return node;
}

void InorderTBT(struct Node *root)
{
    if(root == NULL)
    {
        cout<<"Empty Tree"<<endl;
    }
    else
    {
        struct Node *node = root;

        while(node->leftthread == false)
        {

```

```

        node = node->left;
    }
    while(node != NULL)
    {
        cout<<" "<<node->data;
        node = InorderSuccessor(node);
    }
}
}

```

```

void PreorderTBT(struct Node *root )
{
    struct Node *node;
    if(root == NULL)
    {
        cout<<"Empty Tree "<<endl;
        return;
    }

    node = root;
    while(node != NULL)
    {
        cout<<" "<<node->data;

        if(node->leftthread == false)
        {
            node=node->left;
        }
        else if(node->rightthread == false)
        {
            node = node->right;
        }

        else
        {
            while(node != NULL && node->rightthread == true)

                node = node->right;
            if(node != NULL)
                node = node->right;
        }
    }
}

int main()
{
    struct Node *root = NULL;
    int choice;

    while(1)
    {

        cout<<"1.Insert "<<endl;
        cout<<"2.InOrder Traversal "<<endl;
        cout<<"3.Preorder Traversal"<<endl;
    }
}

```

```

cout<<"Enter your choice : "<<endl;
cin>>choice;

switch(choice)
{
    case 1:
    {
        cout<<"Enter the element you want to Insert : "<<endl;
        cin>>choice;
        root = Insert(root, choice);

    }
    break;

    case 2:
    {
        cout<<"Inorder Traversal Of TBT : ";
        InorderTBT(root);
        cout<<endl;
    }
    break;

    case 3:
    {
        cout<<"Preorder Traversal Of TBT : ";
        PreorderTBT(root);
        cout<<endl;
    }
    break;

    case 4:
    {
        exit(0);
    }
    break;

    default:
    {
        cout << "Invalid Choice." << endl;
    }
    break;

}

}
return 0;

}

```

Output:

```
C:\Windows\System32\cmd.exe - a
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign6dsa.cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a
1.Insert
2.InOrder Traversal
3.Preorder Traversal
Enter your choice :
1
Enter the element you want to Insert :
3
1.Insert
2.InOrder Traversal
3.Preorder Traversal
Enter your choice :
```

## Experiment No. 7

// Implementation of Minimum Spanning tree using Prims and Kruskals Algorithm

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
#define nl "\n"
```

```
#define inf INT_MAX
```

```
const int N = 1e4;
```

```
int cost[N][N]; // Adjacency matrix having weights
```

```
void Prims(int v){ // Graph's vertex should start from 0
```

```
    int dist[v];
```

```
    int from[v];
```

```
    int visited[v];
```

```
    fill(dist, dist+v, inf);
```

```
    fill(visited, visited+v, 0);
```

```
    fill(from, from+v, 0);
```

```
    vector< pair<pair<int, int>, int> >MSTedges; // For output
```

```
    dist[0] = 0;
```

```
    int cur = 0;
```

```
    for(int iteration = 1; iteration < v; iteration++){
```

```
        visited[cur] = 1;
```

```
        for(int i = 0; i < v; i++){
```

```
            if (!visited[i]){
```

```
                if (cost[cur][i] < dist[i]){
```

```
                    dist[i] = cost[cur][i];
```

```
                    from[i] = cur;
```

```
                }
```

```
            }
```

```
        }
```

```
        int mn = inf;
```

```
        for(int i = 0; i < v; i++){
```

```
            if (!visited[i]){
```

```
                if (dist[i] < mn){
```

```
                    mn = dist[i];
```

```
                    cur = i;
```

```
                }
```

```
            }
```

```
        }
```

```
        MSTedges.push_back({ {from[cur], cur}, mn});
```

```
    }
```

```
    // For displaying the result
```

```
    int sum = 0;
```



```

cout<<"\n***** PRIM'S ALGORITHM *****\n";
for(auto it : MSTedges){
    cout<<"Edge : "<<it.first.first<<" - "<<it.first.second<<" Length : "<<it.second<<nl;
    sum+=it.second;
}
cout<<"Minimum Cost of Edges : "<<sum<<nl;
}

// Union-Find Algorithm is used for detection of cycle in Kruskal's Algorithm
void Union(int x, int y, int parent[], int v){ // Naive Implementation
    int tempx = parent[x];
    int tempy = parent[y];

    for(int i = 0; i < v; i++){
        if (parent[i] == tempx or parent[i] == tempy){
            parent[i] = tempx;
        }
    }
}

int find(int x, int parent[]){
    return parent[x];
}

void Kruskal(int v){
    int parent[v];
    for(int i = 0; i < v; i++) parent[i] = i;

    vector< pair<int, pair<int, int>> >edges;
    vector< pair<int, pair<int, int>> >MSTedges;

    for(int i = 0; i < v; i++){
        for(int j = 0; j < v; j++){
            if (cost[i][j] != inf){
                edges.push_back({cost[i][j], {i, j}});
            }
        }
    }

    sort(edges.begin(), edges.end());

    for(auto it : edges){
        if (find(it.second.first, parent) != find(it.second.second, parent)){
            Union(it.second.first, it.second.second, parent, v);

            MSTedges.push_back({it.first, {it.second.first, it.second.second}});
        }
    }

    // For displaying the output
    int sum = 0;

    cout<<"\n***** KRUSKAL'S ALGORITHM *****\n";
    for(auto it : MSTedges){

```

```

        cout<<"Edge : "<<it.second.first<<" - "<<it.second.second<<" Length : "<<it.first<<nl;
        sum+=it.first;
    }
    cout<<"Minimum Cost of Edges : "<<sum<<nl;
}

```

```

int main(){
    int v = 0, op = 0, op1 = 1, op2 = 0, src = 0, dest = 0, weight = 0;

    do{
        cout<<"\n***** Minimum Spanning Tree *****"<<nl<<nl;
        cout<<"Choose The Desired Option:\n";
        cout<<"1. Add a New Graph\n";
        cout<<"2. MST using Prim's Algorithm\n";
        cout<<"3. MST using Kruskal's Algorithm\n";
        cout<<"*. Press any other numeric key to exit\n";

        cin>>op;

        switch(op){
            case 1:
                cout<<"Enter the number of vertices in the graph : ";
                cin>>v;

                for(int i = 0; i <= v; i++){
                    for(int j = 0; j <= v; j++){
                        cost[i][j]=inf;
                    }
                }

                while(op1!=0){
                    cout<<"1. Add Edges"<<nl;
                    cout<<"2. Delete Edge"<<nl;
                    cout<<"0. Go Back To The Main Menu"<<nl;
                    cin>>op1;

                    switch(op1){
                        case 1:
                            cout<<"Enter the number of edges you want to add : ";
                            cin>>op2;
                            for(int i = 0; i < op2; i++){
                                cout<<"Enter the source vertex : ";
                                cin>>src;
                                cout<<"Enter the destination vertex : ";
                                cin>>dest;
                                cout<<"Enter the length of the edge : ";
                                cin>>weight;

                                cost[src][dest] = weight;
                                cost[dest][src] = weight;
                            }
                            break;
                        case 2:
                            cout<<"Enter the source vertex : ";

```

```

        cin>>src;
        cout<<"Enter the destination vertex : ";
        cin>>dest;

        cost[src][dest] = inf;
        cost[dest][src] = inf;
        break;
    default:
        op1 = 0;
        break;
    }
}
op1 = 1;
break;
case 2:
    Prims(v);
    break;
case 3:
    Kruskal(v);
    break;

default:
    op = 0;
    break;
}
} while(op != 0);
}

```

Output:

```

C:\Windows\System32\cmd.exe - a

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a

***** Minimum Spanning Tree *****

Choose The Desired Option:
1. Add a New Graph
2. MST using Prim's Algorithm
3. MST using Kruskal's Algorithm
*. Press any other numeric key to exit
1
Enter the number of vertices in the graph : 7
1. Add Edges
2. Delete Edge
0. Go Back To The Main Menu
1
Enter the number of edges you want to add : 3
Enter the source vertex : 5
Enter the destination vertex : 6
Enter the length of the edge : 2
Enter the source vertex : 7
Enter the destination vertex : 2
Enter the length of the edge : 1
Enter the source vertex : 4
Enter the destination vertex : 9
Enter the length of the edge : 2
1. Add Edges
2. Delete Edge
0. Go Back To The Main Menu

```

## Experiment No. 8

// Implementation of Dijkstra's Algorithm

```
#include<iostream>
using namespace std;
int N;
int graph[100][100];
int dist[100];
bool visited[100];
int parent[100];
void createGraph()
{
    int i,j,max,u,w,v;

    cout<<"\nEnter the number of Vertices : ";
    cin>>N;

    for(i=0;i<=N;i++)
        for(j=0;j<=N;j++)
            graph[i][j]=0;
    max=N*(N+1);
    for(i=0;i<max;i++)
    {
        cout<<"\n";
        cout<<"Enter Vertices and Distance : ";
        cin>>u>>v>>w;

        if(u== -1)
        {
            break;
        }
        else
        {
            graph[u][v]=w;
            graph[v][u]=w;
        }
    }
}
int minDistance()
{
    int min = 10000, minDist;
    for (int v = 0; v < N; v++)
        if (visited[v] == false && dist[v] <= min)
        {
            min = dist[v];
            minDist = v;
        }
    return minDist;
}
void printPath(int j)
{
    if (parent[j]==-1)
        return;
    printPath(parent[j]);
    cout<<j<<" ";
}
```

```

}
void dijkstra()
{
    int src;
    cout<<"Enter the Source : ";
    cin>>src;
    for (int i = 0; i < N; i++)
    {
        parent[0] = -1;
        dist[i] = 10000;
        visited[i] = false;
    }
    dist[src] = 0;
    for (int count = 0; count < N-1; count++)
    {
        int u = minDistance();
        visited[u] = true;
        for (int v = 0; v < N; v++)
            if (!visited[v] && graph[u][v] &&
                dist[u] + graph[u][v] < dist[v])
            {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }
    string land [5] = {"Railway Station","Bus Stand"," Temple ", "Hospital","Water Tank"};

    cout<<"===== "<<endl;
    cout<<"Src->Dest\t\t\t Distance\t\tPath"<<endl;
    for (int i = 1; i < N; i++)
    {
        cout<<land[src]<<"->"<<land[i]<<"\t\t"<<dist[i]<<" km"<<"\t\t"<<src<<" ";
        printPath(i);
        cout<<endl;
    }
}

int main()
{
    cout<<"===== "<<endl;
    cout<<"*****Dijkstra's Algorithm*****"<<endl;

    cout<<"===== "<<endl;
    cout<<endl;
    createGraph();
    dijkstra();
    return 0;
}

```

Output:

```

C:\Windows\System32\cmd.exe
=====
Enter the number of Vertices : 5
Enter Vertices and Distance : 0 1 10
Enter Vertices and Distance : 0 2 20
Enter Vertices and Distance : 0 3 15
Enter Vertices and Distance : 1 2 5
Enter Vertices and Distance : 2 3 10
Enter Vertices and Distance : 1 4 8
Enter Vertices and Distance : 2 4 6
Enter Vertices and Distance : 3 4 9
Enter Vertices and Distance : -1 -1 -1
Enter the Source : 0
=====
Src->Dest          Distance          Path
Railway Station->Bus Stand      10 km          0 1
Railway Station-> Temple      15 km          0 1 2
Railway Station->Hospital      15 km          0 3
Railway Station->Water Tank    18 km          0 1 4
C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>

```

## Experiment No. 9

// Implementation of Heap sort

#include <iostream>

using namespace std;

// A function to heapify the array.

void MaxHeapify(int a[], int i, int n)

```
{
    int j, temp;
    temp = a[i];
    j = 2*i;

    while (j <= n)
    {
        if (j < n && a[j+1] > a[j])
            j = j+1;
        // Break if parent value is already greater than child value.
        if (temp > a[j])
            break;
        // Switching value with the parent node if temp < a[j].
        else if (temp <= a[j])
        {
            a[j/2] = a[j];
            j = 2*j;
        }
    }
    a[j/2] = temp;
    return;
}
```

void HeapSort(int a[], int n)

```
{
    int i, temp;
    for (i = n; i >= 2; i--)
    {
        // Storing maximum value at the end.
        temp = a[i];
        a[i] = a[1];
        a[1] = temp;
        // Building max heap of remaining element.
        MaxHeapify(a, 1, i - 1);
    }
}
```

void Build\_MaxHeap(int a[], int n)

```
{
    int i;
    for(i = n/2; i >= 1; i--)
        MaxHeapify(a, i, n);
}
```

int main()

```
{
    int n, i;
    cout<<"\nEnter the number of data element to be sorted: ";
    cin>>n;
    n++;
}
```

```

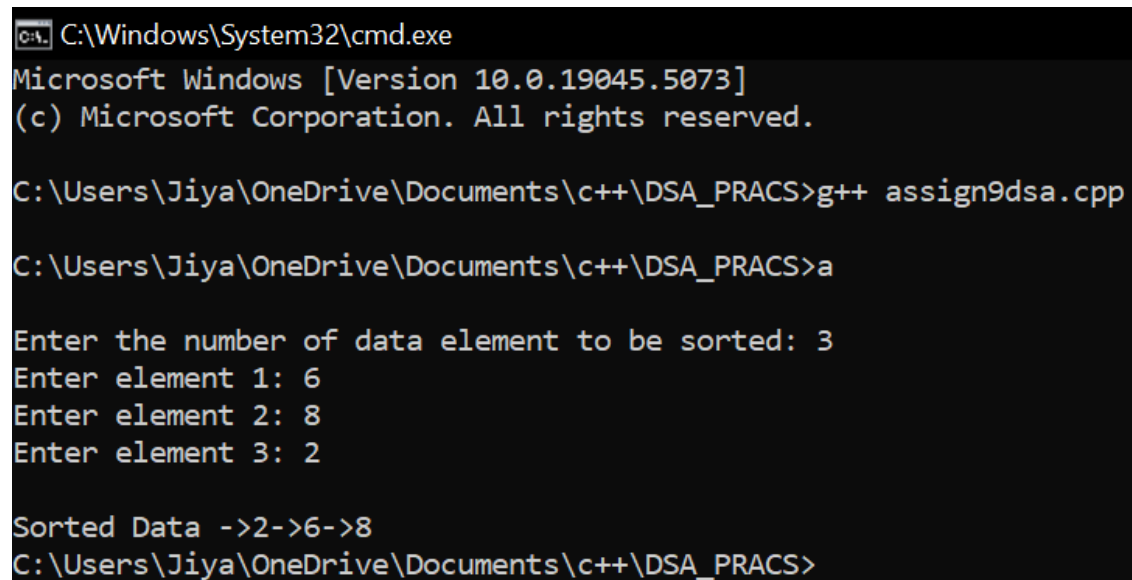
int arr[n];
for(i = 1; i < n; i++)
{
    cout<<"Enter element "<<i<<": ";
    cin>>arr[i];
}
// Building max heap.
Build_MaxHeap(arr, n-1);
HeapSort(arr, n-1);

// Printing the sorted data.
cout<<"\nSorted Data ";

for (i = 1; i < n; i++)
    cout<<"->"<<arr[i];

return 0;
}
Output:

```



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign9dsa.cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a

Enter the number of data element to be sorted: 3
Enter element 1: 6
Enter element 2: 8
Enter element 3: 2

Sorted Data ->2->6->8
C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>

```



## Experiment No. 10

// To implement program for Sequential Access File

```
#include<iostream>
#include<fstream>
#include<cstring>
#include<iomanip>
using namespace std;
const int MAX=20;
class Student
{
    int rollno;
    char name[20],city[20];
    char div;
    int year;
public:
    Student()
    {
        strcpy(name,"");
        strcpy(city,"");
        rollno=year=div=0;
    }
    Student(int rollno,char name[MAX],int year,char div,char city[MAX])
    {
        strcpy(this->name,name);
        strcpy(this->city,city);
        this->rollno=rollno;
        this->year=year;
        this->div=div;
    }
    int getRollNo()
    {
        return rollno;
    }
    void displayRecord()
    {
        cout<<endl<<setw(5)<<rollno<<setw(20)<<name<<setw(5)<<year<<setw(5)<<div<<setw(10)<<city
        ;
        }
    };
//=====File Operations =====
class FileOperations
{
    fstream file;
public:
    FileOperations(char* filename)
    {
        file.open(filename,ios::in|ios::out|ios::ate|ios::binary);
    }
    void insertRecord(int rollno, char name[MAX],int year, char div,char city[MAX])
    {
        Student s1(rollno,name,year,div,city);
        file.seekp(0,ios::end);
    }
};
```

```

file.write((char *)&s1,sizeof(Student));
file.clear();
}
void displayAll()
{
    Student s1;
    file.seekg(0,ios::beg);
    while(file.read((char *)&s1, sizeof(Student)))
    {
        s1.displayRecord();
    }
    file.clear();
}
void displayRecord(int rollNo)
{
    Student s1;
    file.seekg(0,ios::beg);
    bool flag=false;
    while(file.read((char *)&s1,sizeof(Student)))
    {
        if(s1.getRollNo()==rollNo)
        {
            s1.displayRecord();
            flag=true;
            break;
        }
    }
    if(flag==false)
    {
        cout<<"\nRecord of "<<rollNo<<"is not present.";
    }
    file.clear();
}
void deleteRecord(int rollno)
{
    ofstream outFile("new.dat",ios::binary);
    file.seekg(0,ios::beg);
    bool flag=false;
    Student s1;

    while(file.read((char *)&s1, sizeof(Student)))
    {
        if(s1.getRollNo()==rollno)
        {
            flag=true;
            continue;
        }
        outFile.write((char *)&s1, sizeof(Student));
    }
    if(!flag)
    {
        cout<<"\nRecord of "<<rollno<<" is not present.";
    }
    file.close();
    outFile.close();
}

```

```

remove("student.dat");
rename("new.dat", "student.dat");
file.open("student.dat",ios::in|ios::out|ios::ate|ios::binary);
}
~FileOperations()
{
    file.close();
    cout<<"\nFile Closed.";
}
};

int main() {
    ofstream newFile("student.dat",ios::app|ios::binary);
    newFile.close();
    FileOperations file((char*)"student.dat");
    int rollNo,year,choice=0;
    char div;
    char name[MAX],address[MAX];
    while(choice!=5)
    {
        //clrscr();
        cout<<"\n**Student Database**\n";
        cout<<"1) Add New Record\n";
        cout<<"2) Display All Records\n";
        cout<<"3) Display by RollNo\n";
        cout<<"4) Deleting a Record\n";
        cout<<"5) Exit\n";
        cout<<"Choose your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1 : //New Record
                cout<<endl<<"Enter RollNo and name : \n";
                cin>>rollNo>>name;
                cout<<"Enter Year and Division : \n";
                cin>>year>>div;
                cout<<"Enter address : \n";
                cin>>address;
                file.insertRecord(rollNo,name,year,div,address);
                cout<<"\nRecord Inserted.";
                break;
            case 2 :

                cout<<endl<<setw(5)<<"ROLL"<<setw(20)<<"NAME"<<setw(5)<<"YEAR"<<setw(5)<<"DIV"<<setw(10)<<"CITY";
                file.displayAll();
                break;
            case 3 :
                cout<<"Enter Roll Number";
                cin>>rollNo;
                file.displayRecord(rollNo);

                break;
            case 4:
                cout<<"Enter rollNo";
                cin>>rollNo;

```

```

        file.deleteRecord(rollNo);
        break;
    case 5 :break;
}

}

return 0;
}

```

Output:

```

C:\Windows\System32\cmd.exe - a
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>g++ assign10dsa.cpp

C:\Users\Jiya\OneDrive\Documents\c++\DSA_PRACS>a

**Student Database**
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
69
Jiya
Enter Year and Division :
2024
B
Enter address :
Pune

Record Inserted.
**Student Database**
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
73
Samiksha
Enter Year and Division :
2024
B
Enter address :
Pune

```