DS ALL PRACTICAL CODES

Practical 1a

```
//Write a program to store the element inn 1-D array and perform operations like searching sorting and reversing the array


//C++ Program - Reverse Array


#include<iostream>
#include<conio.h>
using namespace std;


int main()
{


    int arr[50], size, i, j, temp;
    cout<<"Enter Array Size: ";
    cin>>size;
    cout<<"Enter Array elements: ";
    for(i=0;i<size;i++)
    {
        cin>>arr[i];
    }
    j=i-1;
    i=0;
    while(i<j)
    {
            temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
            i++;
            j--;
    }
    cout<<"Now the Reverse of the Array is: \n";
    for(i=0; i<size;i++)
    {
      cout<<arr[i]<<" ";
    }
    getch();
}
```

Practical 1b

```
//Practical 1-b
//c++ program - Linear search

#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
        int arr[10], i, num, n, c=0, pos;
        cout<<"Enter the array size: ";
        cin>>n;
        cout<<"Enter Array ELements: ";
        for(i=0;i<n;i++)
        {
                cin>>arr[i];
        }
        cout<<"Enter the number to be search: ";
        cin>>num;
        for(i=0;i<n;i++)
        {
                if(arr[i]==num)
                {
                        c=1;
                        pos=i+1;
                        break;
                }
        }
        if(c==0)
        {
                cout<<"Number not found...!!";
        }
        else
        {
                cout<<num<<" found at position "<<pos;
        }
getch();
}
```

Practical 1c

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

using namespace std;

int main()
{
  int i, a[10], temp, j;
  cout << "Enter any 10 numbers in an array: \n";

  // You should loop from 0 to 9 to input 10 elements into the array.
  for (i = 0; i < 10; i++)
  {
    cin >> a[i];
  }

  cout << "\n Data before sorting:  ";
  for (j = 0; j < 10; j++)
  {
    cout << a[j] << " "; // Add a space to separate the numbers.
  }

  // You should loop only up to 9 in both loops to avoid going out of bounds.
```

```
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9 - i; j++) // Reduce the inner loop by 'i' iterations since the largest elements are already sorted.
        {
            if (a[j] > a[j + 1])
            {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

    cout << "\n Data after sorting: ";
    for (j = 0; j < 10; j++)
    {
        cout << a[j] << " "; // Add a space to separate the numbers.
    }

    getch();
    return 0; // Add a return statement to indicate successful program completion.
}
```

Practical 2

```
//Practical 2

//Read two arrays from the user and merge them and display the element
sorted order


#include<iostream>

#include<conio.h>

using namespace std;


int main()

{

	int arr1[50], arr2[250], size1, size2, size, i, j,k, merge[100];

	cout<<"Enter Array 1 size";

	cin>>size1;

	cout<<"Enter Array 1 Elements: ";

	for(i=0;i<size1;i++)

	{

		cin>>arr1[i];

	}

	cout<<"Enter Array 2 Size";

	cin>>size2;

	cout<<"Enter Array 2 Elements: ";

	for(i=0;i<size2;i++)

	{

		cin>>arr2[i];

	}
```

```
    for(i=0;i<size1;i++)
    {
        merge[i]=arr1[i];
    }
    size=size1+size2;
    for(i=0, k=size1; k<size && i<size2; i++, k++)
    {
        merge[k]=arr2[i];
    }
    cout<<"Now the new array after merging is: \n";
    for(i=0;i<size;i++)
    {
        cout<<merge[i]<<" ";
    }
    getch();
}
```

Practical 3a (addition)

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

int main()
{
    int mat1[3][3], mat2[3][3], i, j, mat3[3][3];
    cout<<"Enter matrix 1 elements :";
```

```
for(i=0; i<3; i++)
{
        for(j=0; j<3; j++)
        {
           cin>>mat1[i][j];
        }
}
cout<<"Enter matrix 2 elements :";
for(i=0; i<3; i++)
{
        for(j=0;j<3;j++)
        {
                cin>>mat2[i][j];

        }
}
cout<<"Adding the two matrix to form the third matrix......\n";
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {

                mat3[i][j]=mat1[i][j] + mat2[i][j];
        }
}
cout<<"The two matrix addede successfully....!!";
```

```
	cout<<"The new matrix will be....\n";
	for(i=0;i<3;i++)
	{
		for(j=0;j<3;j++)
		{
			cout<<mat3[i][j]<<" ";
		}
		cout<<"\n";
	}
	getch();

}
```

Practical 3b(subtraction)

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

int main()
{
	int mat1[3][3], mat2[3][3], i, j, mat3[3][3];
	cout<<"Enter matrix 1 elements :";
	for(i=0; i<3; i++)
	{
		for(j=0; j<3; j++)
		{
```

```
           cin>>mat1[i][j];
        }
}
cout<<"Enter matrix 2 elements :";
for(i=0; i<3; i++)
{
        for(j=0;j<3;j++)
        {
                cin>>mat2[i][j];

        }
}
cout<<"Subtracting the two matrix to form the third matrix......\n";
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {

                mat3[i][j]=mat1[i][j] - mat2[i][j];
        }
}
cout<<"\nThe two matrix subtracted successfully....!!";
cout<<"\nThe new matrix will be....\n";
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
```

```
                {
                        cout<<mat3[i][j]<<" ";
                }
                cout<<"\n";
        }
        getch();

}
```

Practical 4

//Write a program to create a single linked list and display the node elements in reverse order

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

struct node
{
        int info;
        node *next;
}
*start, *newptr, *save, *ptr;

node *create_new_node(int);
void insert_at_beg(node *);
void display(node *);
```

```
int main()
{
        start = NULL;
        int inf;
        char ch='y';
        while(ch=='y'||ch=='Y')
        {
                cout<<"Enter Information for the new node: ";
                cin>>inf;
                cout<<"\n Creating new node!!Press any key to continue.";
                getch();
                newptr = create_new_node(inf);
                if(newptr != NULL)
                {
                        cout<<"\n\n New node created successfully...!!\n";
                        cout<<"Press any key to continue.";
                        getch();
                }
                else
                {
                        cout<<"\n Sorry cannot create new node!!!Aborting!!!";
                        cout<<"Press any key to exit";
                        getch();
                        exit(1);
                }
                cout<<"\n\n Now inserting this node at the beginning of the list...\n";
                cout<<"\n Press any key to continue..\n";
                getch();
                insert_at_beg(newptr);
                cout<<"\n Node successfully inserted at the beginning of the list. \n";
                cout<<"Now the list is: \n";
                display(start);
                cout<<"\n Want to enter more nodes?(y/n)...";
                cin>>ch;
        }
        getch();
}
node *create_new_node(int n)
{
        ptr = new node;
        ptr->info = n;
        ptr->next = NULL;
        return ptr;
}

void insert_at_beg(node *np)
{
        if(start==NULL)
        {
                start = np;
        }
        else
        {
                save = start;
                start = np;
                np->next = save;
        }
}
void display(node *np)
{
        while(np != NULL)
        {
                cout<<np->info<<" ->";
                np = np->next;
        }
        cout<<"!!\n";
}
```

Practical 5

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

// Node class to represent elements in the linked list
class Node {
public:
   int data;
```

```
    Node* next;

    Node(int val) {
        data = val;
        next = NULL;
    }
};

// Linked List class
class LinkedList {
public:
    Node* head;

    LinkedList() {
        head = NULL;
    }

    // Function to insert a new element at the end of the linked list
    void insert(int val) {
        Node* newNode = new Node(val);
        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    // Function to search for an element in the linked list
    bool search(int val) {
        Node* temp = head;
        while (temp != NULL) {
            if (temp->data == val) {
                return true; // Element found
            }
            temp = temp->next;
        }
        return false; // Element not found
    }

    // Function to display the linked list
    void display() {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
```

```
};

int main() {
    LinkedList myList;

    // Insert elements into the linked list
    int numElements;
    cout << "Enter the number of elements to insert: ";
    cin >> numElements;

    for (int i = 0; i < numElements; i++) {
        int element;
        cout << "Enter element " << i + 1 << ": ";
        cin >> element;
        myList.insert(element);
    }

    cout << "Linked List: ";
    myList.display();

    int searchValue;
    cout << "Enter the value to search for: ";
    cin >> searchValue;

    if (myList.search(searchValue)) {
        cout << "Element " << searchValue << " found in the linked list." << endl;
    } else {
        cout << "Element " << searchValue << " not found in the linked list." <<
endl;
    }

    return 0;
}
```

Practical 6

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

using namespace std;
int c = 0;

struct node {
    node* next, * prev;
    int data;
} * head = NULL, * tail = NULL, * p = NULL, * r = NULL, * np = NULL;

void create(int x) {
    np = new node;
    np->data = x;
    np->next = NULL;
    np->prev = NULL;
```

```
if (c == 0) {
    tail = np;
    head = np;
    p = head;
    p->next = NULL;
    p->prev = NULL;
    c++;
}
else {
    p = head;
    r = p;
    if (np->data < p->data) {
        np->next = p;
        p->prev = np;
        np->prev = NULL;
        head = np;
        p = head;
        do {
            p = p->next;
        } while (p->next != NULL);
        tail = p;
    }
    else if (np->data > p->data) {
        while (p != NULL && np->data > p->data) {
            r = p;
            p = p->next;
```

```
                if (p == NULL) {
                    r->next = np;
                    np->prev = r;
                    np->next = NULL;
                    tail = np;
                    break;
                }
                else if (np->data < p->data) {
                    r->next = np;
                    np->prev = r;
                    np->next = p;
                    p->prev = np;
                    if (p->next != NULL) {
                        do {
                            p = p->next;
                        } while (p->next != NULL);
                        tail = p;
                        break;
                    }
                }
            }
        }
    }
}

void traverse_tail() {
    node* t = tail;
    while (t != NULL) {
        cout << t->data << "\t";
        t = t->prev;
    }
    cout << endl;
}

void traverse_head() {
    node* t = head;
    while (t != NULL) {
        cout << t->data << "\t";
        t = t->next;
    }
    cout << endl;
}

int main() {
    int i = 0, n, x, ch;
    cout << "Enter the no. of nodes \n";
    cin >> n;
    while (i < n) {
        cout << "Enter the data for node " << i + 1 << ": ";
        cin >> x;
        create(x);
        i++;
    }
    cout << "\nTraversing Doubly Linked List Head first \n";
    traverse_head();
    cout << "\nTraversing doubly Linked List tail first \n";
    traverse_tail();
    getch();
}
```

Practical 7

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

class stack
{
        int stk[5];
        int top;
        public:
                stack()
                {
                        top=-1;
                }
        void push(int x)
        {
                if(top>4)
```

```
    {
        cout<<"Stack Overflow";
        return;
    }
    stk[++top]=x;
    cout<<"inserted"<<x;
}
void pop()
{
    if(top <0)
    {
        cout<<"Stack under flow";
        return;
    }
    cout<<"\n deleted \t"<<stk[top--];
}
void display()
{
    if(top<0)
    {
        cout<<"Stack Empty.";
        return;
    }
    for(int i=top; i>=0; i--)
    {
        cout<<stk[i]<<" ";
        }
    }
};
    main()
    {
        int ch;
        stack st;
        while(1)
        {
            cout<<"\n 1.push 2.pop 3.display 4.exit \n Enter your
choice: ";
            cin>>ch;
            switch(ch)
            {
                case 1:cout<<"Enter the Element: ";
                cin>>ch;
                st.push(ch);
                break;
                case 2: st.pop();
                break;
                case 3: st.display();
                break;
                case 4: exit(0);
            }
        }
        return(0);
    }
```

Practical 8

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

int getPrecedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}

string infixToPostfix(const string& infix) {
    stack<char> operators;
    string postfix = "";

    for (int i = 0; i < infix.length(); ++i) {
        char ch = infix[i];
        if (isalnum(ch)) {
            postfix += ch;
        } else if (ch == '(') {
            operators.push(ch);
        } else if (ch == ')') {
            while (!operators.empty() && operators.top() != '(') {
                postfix += operators.top();
                operators.pop();
            }
            if (!operators.empty() && operators.top() == '(') {
                operators.pop();
            }
        } else {
            while (!operators.empty() && getPrecedence(ch) <=
getPrecedence(operators.top())) {
                postfix += operators.top();
                operators.pop();
            }
            operators.push(ch);
        }
    }

    while (!operators.empty()) {
        postfix += operators.top();
        operators.pop();
    }

    return postfix;
}

string postfixToInfix(const string& postfix) {
    stack<string> operands;

    for (int i = 0; i < postfix.length(); ++i) {
        char ch = postfix[i];
        if (isalnum(ch)) {
            string operand(1, ch);
            operands.push(operand);
        } else {
            string operand2 = operands.top();
            operands.pop();
            string operand1 = operands.top();
            operands.pop();
            string result = "(" + operand1 + ch + operand2 + ")";
            operands.push(result);
        }
    }

    return operands.top();
}

int main() {
    string infixExpression = "A*(B+C)/D";
    string postfixExpression = infixToPostfix(infixExpression);
    string infixExpressionFromPostfix = postfixToInfix(postfixExpression);

    cout << "Infix to Postfix Conversion:" << endl;
    cout << "Infix Expression: " << infixExpression << endl;
    cout << "Postfix Expression: " << postfixExpression << endl;

    cout << "\nPostfix to Infix Conversion:" << endl;
    cout << "Postfix Expression: " << postfixExpression << endl;
    cout << "Infix Expression: " << infixExpressionFromPostfix << endl;

    return 0;
}
```

Practical 9

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

class queue
{
    int queue1[5];
    int rear, front;

public:
    queue()
    {
        rear = -1;
        front = -1;
    }
```

```
void insert(int x)
{
   if(rear > 4)
   {
      cout << "Queue overflow!";
      front = rear = -1;
      return;
   }
   queue1[++rear] = x;
   cout << "Inserted " << x << endl;
}

void delet()
{
   if(front == rear)
   {
      cout << "Queue underflow!";
      return;
   }
   cout << "Deleted " << queue1[++front] << endl;
}

void display()
{
   if(rear == front)
```

```
    {
      cout << "Queue Empty" << endl;
      return;
    }
    for(int i = front + 1; i <= rear; i++)
      cout << queue1[i] << " ";
    cout << endl;
  }
};

int main()
{
  int ch;
  queue qu;

  while(1)
  {
    cout << "\n1. Insert 2. Delete 3. Display 4. Exit" << "\n Enter your choice: ";
    cin >> ch;

    switch(ch)
    {
      case 1:
        cout << "Enter the Element: ";
        cin >> ch;
        qu.insert(ch);
```

```
             break;
          case 2:
             qu.delet();
             break;
          case 3:
             qu.display();
             break;
          case 4:
             exit(0);
       }
   }

   return 0;
}
```

Practical 10

```
//program to implement circular queue

#include<iostream>
using namespace std;
class cqueue
{
        private:
                int *arr;
                int front, rear;
                int MAX;
        public:
            cqueue(int maxsize = 10);
            void addq(int item);
            int delq();
            void display();
};
cqueue :: cqueue(int maxsize)
{
    MAX = maxsize;
    arr =  new int [MAX];
    front = rear = -1;
    for(int i=0; i<MAX; i++)
    {
        arr[i]=0;
    }
}
void cqueue :: addq(int item)
{
    if((rear +1)% MAX == front)
    {
        cout<<"\n Queue is full";
        return;
    }
    rear = (rear + 1)%MAX;
    arr[rear] = item;
    if(front == -1)
```

```
        {
                front = 0;
        }
}
int cqueue :: delq()
{
        int data;
                if(front == -1)
                {
                        cout<<"\n Queue is Empty";
                        return NULL;
                }
                data = arr[front];
                arr[front]=0;
                if(front == rear)
                {
                        front = -1;
                        rear = -1;
                }
                else
                {
                        front = (front + 1)% MAX;
                        return data;
                }
}
void cqueue :: display()
```

```
{
        cout<<endl;
        for(int i=0; i<MAX; i++)
        cout<<arr[i]<<" ";
        cout<<endl;
}
int main()
{
        cqueue a(10);
        a.addq(14);
        a.addq(22);
        a.addq(13);
        a.addq(-6);
        a.addq(25);
        cout<<"\n Elements in the circular queue: ";
        a.display();
        int i = a.delq();
        cout<<"\n Item Deleted: "<<i;
        cout<<"\n Elements in thr Circular Queue after Deletion: ";
        a.display();
        a.addq(21);
        a.addq(17);
        a.addq(18);
        a.addq(9);
        a.addq(20);
        cout<<"Elements in the circular queue after addition: ";
        a.display();
        a.addq(32);
        cout<<"Elements in the circular queue after addition: ";
        a.display();
}
```

Practical 11

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

using namespace std;

class node {
public:
  int data;
  class node* next;
  class node* prev;
};

class dequeue : public node { // Corrected class name to "dequeue"
  node* head, * tail;
  int top1, top2;

public:
  dequeue() {
    top1 = 0;
    top2 = 0;
    head = NULL;
    tail = NULL;
}

void push(int x) {
    node* temp;
    int ch;
    if (top1 + top2 >= 5) {
        cout << "dequeue overflow!";
        return;
    }
    if (top1 + top2 == 0) {
        head = new node;
        head->data = x;
        head->next = NULL;
        head->prev = NULL;
        tail = head;
        top1++;
    }
    else {
        cout << "Add elements 1.FIRST 2.LAST \n Enter your choice: ";
        cin >> ch;
        if (ch == 1) {
            top1++;
            temp = new node;
            temp->data = x;
            temp->next = head;
            temp->prev = NULL;
            head->prev = temp;
            head = temp;
        }
        else {
            top2++;
            temp = new node;
            temp->data = x;
            temp->next = NULL;
            temp->prev = tail;
            tail->next = temp;
            tail = temp;
        }
    }
}

void pop() {
    int ch;
    cout << "Delete 1.FIRST node 2.LAST node. \n Enter your choice: ";
    cin >> ch;
    if (top1 + top2 <= 0) {
        cout << "\n Dequeue underflow";
        return;
    }
```

```
        if (ch == 1) {
            head = head->next;
            head->prev = NULL;
            top1--;
        }
        else {
            top2--;
            tail = tail->prev; // Changed '-' to '='
            tail->next = NULL;
        }
    }

    void display() {
        int ch;
        node* temp;
        cout << "Display from 1.Starting 2.Ending. \n Enter your choice: "; // Added ':' at the end
        cin >> ch;
        if (top1 + top2 <= 0) {
            cout << "under flow";
            return;
        }
        if (ch == 1) {
            temp = head;
            while (temp != NULL) {
                cout << temp->data << " ";
                temp = temp->next;
            }
        }
        else {
            temp = tail;
            while (temp != NULL) {
                cout << temp->data << " ";
                temp = temp->prev;
            }
        }
    }
};

int main() {
    dequeue d1;
    int ch;
    while (1) {
        cout << "\n 1.INSERT 2.DELETE 3.DISPLAY 4.EXIT \n Enter your choice: ";
        cin >> ch;
        switch (ch) {
        case 1: cout << "Enter Element: ";
            cin >> ch;
            d1.push(ch);
            break;
        case 2: d1.pop();
            break;
        case 3: d1.display();
```

```
           break;
        case 4: exit(1);
        }
    }
}
```

Practical 12

```
#include<iostream>
using namespace std;
int main()
{
            int a [50],n,i,j,temp;
            cout<<"Enter size of the array:";
            cin>>n;
            cout<<"Enter the array elements:";
            for(i=0;i<n;++i)
            cin>>a[i];
            for(i=1;i<n;++i)
            {
                        for(j=0;j<(n-i);++j)
                        if(a[j]>a[j+1])
                        {
                                    temp=a[j];
                                    a[j]=a[j+1];
                                    a[j+1]=temp;
                        }
```

```
	}
	cout<<"Array after bubble sort:";
	for(i=0;i<n;++i)
	cout<<" "<<a[i];
	return 0;
}
```

Practical 13

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
	int size,arr[50],i,j,temp;
	cout<<"Enter Array Size:";
	cin>>size;
	cout<<"Enter Array Elements:";
	for(i=0;i<size;i++)
	{
		cin>>arr[i];
	}
	cout<<"Sorting Array using selection sort...\n";
	for(i=0;i<size;i++)
	{
		for(j=i+1;j<size;j++)
		{
```

```
                    if(arr[i]>arr[j])
                    {
                              temp=arr[i];
                              arr[i]=arr[j];
                              arr[j]=temp;
                    }
          }
     }
     cout<<"Now the array after sorting is:\n";
     for(i=0;i<size;i++)
     {
          cout<<arr[i]<<" ";
     }
     getch();
}
```

Practical 14

```
#include<iostream>
using namespace std;
int main()
{
int size,arr[50],i,j,temp;
cout<<"Enter array size: ";
cin>>size;
cout<<"Enter array elements: ";
for(i=0;i<size;i++)
{
```

```
cin>>arr[i];
}
cout<<"Sorting array using insertion sort!\n";
for(i=0;i<size;i++)
{
temp=arr[i];
j=i-1;
while((temp<arr[j])&&(j>=0))
{
arr[j+1]=arr[j];
j=j-1;
}
arr[j+1]=temp;
}
cout<<"Now the array after sorting is: \n";
for(i=0;i<size;i++)
{
cout<<arr[i]<<" ";
}
return 0;
}
```

Practical 15

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

```
int main() {
    int n, i, arr[50], search, first, last, middle;

    cout << "Enter total number of Elements: ";
    cin >> n;

    cout << "Enter " << n << " numbers in sorted order: ";
    for (i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << "Enter a number to find: ";
    cin >> search;

    first = 0;
    last = n - 1;
    middle = (first + last) / 2;

    while (first <= last) {
        if (arr[middle] == search) {
            cout << search << " found at location " << middle + 1 << "\n";
            break;
        } else if (arr[middle] < search) {
            first = middle + 1;
        } else {
            last = middle - 1;
        }
        middle = (first + last) / 2;
    }

    if (first > last) {
        cout << "Not found! " << search << " is not present in the list." << endl;
    }

    return 0;
}
```

Practical 16, 17 and 20

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

class Node {
    int key;
    Node* left;
    Node* right;

public:
    Node() {
        key = -1;
        left = NULL;
        right = NULL;
    };
    void setKey(int aKey) {
        key = aKey;
    };
    void setLeft(Node* aLeft) {
        left = aLeft;
    };
    void setRight(Node* aRight) {
        right = aRight;
    };
    int Key() {
        return key;
    };
    Node* Left() {
        return left;
    };
    Node* Right() {
        return right;
    };
};

// Tree class
class Tree {
    Node* root;

public:
    Tree();
    ~Tree();
    Node* Root() {
        return root;
    };
    void addNode(int key);
    void inOrder(Node* n);
    void preOrder(Node* n);
    void postOrder(Node* n);

private:
    void addNode(int key, Node* leaf);
    void freeNode(Node* leaf);
};

// Constructor
Tree::Tree() {
    root = NULL;
}

// Destructor
Tree::~Tree() {
    freeNode(root);
}

// Free the node
void Tree::freeNode(Node* leaf) {
    if (leaf != NULL) {
        freeNode(leaf->Left());
        freeNode(leaf->Right());
        delete leaf;
    }
}

// Add a node
void Tree::addNode(int key) {
    if (root == NULL) {
        cout << "Add root node... " << key << endl;
        Node* n = new Node();
        n->setKey(key);
        root = n;
    } else {
        cout << "Add other node... " << key << endl;
        addNode(key, root);
    }
}

// Add a node (private)
void Tree::addNode(int key, Node* leaf) {
    if (key <= leaf->Key()) {
        if (leaf->Left() != NULL)
            addNode(key, leaf->Left());
        else {
            Node* n = new Node();
            n->setKey(key);
            leaf->setLeft(n);
        }
    } else {
        if (leaf->Right() != NULL)
            addNode(key, leaf->Right());
        else {
            Node* n = new Node();
            n->setKey(key);
            leaf->setRight(n);
        }
    }
}

// Print the tree in-order
// Traverse the left sub-tree, root, right sub-tree
void Tree::inOrder(Node* n) {
    if (n) {
        inOrder(n->Left());
        cout << n->Key() << " "; // Add a space here
        inOrder(n->Right());
    }
}

// Print the tree in-order
// Traverse the left sub-tree, root, right sub-tree
```

```
void Tree::preOrder(Node* n) {
    if (n) {
        cout << n->Key() << " "; // Add a space here
        preOrder(n->Left());
        preOrder(n->Right());
    }
}

// Print the tree post-order
// Traverse the left sub-tree, root, right sub-tree, root
void Tree::postOrder(Node* n) {
    if (n) {
        postOrder(n->Left());
        postOrder(n->Right());
        cout << n->Key() << " "; // Add a space here
    }
}

// Test main program
int main() {
    Tree* tree = new Tree();
    tree->addNode(30);
    tree->addNode(10);
    tree->addNode(20);
    tree->addNode(40);
    tree->addNode(50);
    cout << "In order traversal" << endl;
    tree->inOrder(tree->Root());
    cout << endl;
    cout << "Pre order traversal" << endl;
    tree->preOrder(tree->Root());
    cout << endl;
    cout << "Post order traversal" << endl;
    tree->postOrder(tree->Root());
    cout << endl;
    delete tree;
    return 0;
}
```

Practical 18

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

const int tableSize = 10;

class HashTable {
private:
    int table[tableSize];

public:
    HashTable() {
        for (int i = 0; i < tableSize; i++) {
            table[i] = -1; // Initialize all slots to -1 (indicating empty)
        }
    }

    // Hash function: simple modulo operation
    int hash(int key) {
        return key % tableSize;
    }

    // Insert a key into the hash table using linear probing
    void insert(int key) {
        int index = hash(key);

        // If the slot is empty, insert the key
        if (table[index] == -1) {
            table[index] = key;
        } else {
            // Linear probing: keep looking for the next available slot
            int newIndex = (index + 1) % tableSize;
            while (table[newIndex] != -1) {
                newIndex = (newIndex + 1) % tableSize;
            }
            table[newIndex] = key;
        }
    }

    // Search for a key in the hash table
```

```
bool search(int key) {
    int index = hash(key);

    // If the key is found at the initial index, return true
    if (table[index] == key) {
        return true;
    } else {
        // Linear probing: keep looking for the key
        int newIndex = (index + 1) % tableSize;
        while (table[newIndex] != -1) {
            if (table[newIndex] == key) {
                    return true;
            }
            newIndex = (newIndex + 1) % tableSize;
        }
        return false; // Key not found
    }
}

// Display the hash table
void display() {
    cout << "Hash Table:" << endl;
    for (int i = 0; i < tableSize; i++) {
        cout << "[" << i << "] -> ";
        if (table[i] != -1) {
            cout << table[i];
            } else {
                cout << "Empty";
            }
            cout << endl;
        }
    }
};

int main() {
    HashTable ht;

    // Insert some keys into the hash table
    ht.insert(12);
    ht.insert(22);
    ht.insert(42);
    ht.insert(7);
    ht.insert(32);
    ht.insert(17);

    // Display the hash table
    ht.display();

    // Search for a key
    int keyToSearch = 42;
    if (ht.search(keyToSearch)) {
        cout << "Key " << keyToSearch << " found in the hash table." << endl;
    } else {
        cout << "Key " << keyToSearch << " not found in the hash table." << endl;
    }

    return 0;
}
```

Practical 19

```
#include<stdio.h>
#define size 7
int arr[size];
void init()
{
 int i;
 for(i = 0; i < size; i++)
 arr[i] = -1;
}
void insert(int value)
{
 int key = value % size;

 if(arr[key] == -1)
 {
 arr[key] = value;
 printf("%d inserted at arr[%d]\n", value,key);
 }
 else
 {
 printf("Collision : arr[%d] has element %d already!\n",key,arr[key]);
 printf("Unable to insert %d\n",value);
 }
}
void print()
{
 int i;
 for(i = 0; i < size; i++)
 printf("arr[%d] = %d\n",i,arr[i]);
}
int main()
{
 init();
 insert(10); //key = 10 % 7 ==> 3
 insert(4); //key = 4 % 7 ==> 4
 insert(2); //key = 2 % 7 ==> 2
 insert(3); //key = 3 % 7 ==> 3 (collision)
 printf("Hash table\n");
 print();
 printf("\n");
 return 0;
}
```