**Practical File**

of

**Data Structure and Algorithms Lab**

**(PCC-CS-303)**

submitted in partial fulfillment of the requirement for the award of degree of

**Bachelor of Technology (B.Tech)**

in

**Computer Engineering**

by

**Anuj Sharma**

**(21001003015)**

Under the guidance of

**Mr. Piyush Gupta**

**Assistant Professor**



**Department of Computer Engineering**

**J. C. BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY, YMCA**

**SECTOR-6 FARIDABAD**

**HARYANA-121006**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Problem** | **List of Programs** | **Date** |
|  | Searching | Given an array of integers *nums* and an integer *target*, write a function to search *target* in *nums*. If *target* exists, then return its index. Otherwise, return *-1*. | 12-10-2022 |
|  | Given an array of integers *nums* which is sorted in ascending order, and an integer *target*, write a function to search target in *nums*. If target exists, then return its index. Otherwise, return *-1*. | 12-10-2022 |
|  | Given a sorted array of *n* elements, possibly with duplicates, find the number of occurrences of the *target* element. | 12-10-2022 |
|  | Given a **0-indexed** integer array *nums*, find a **peak element**, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.  \*A peak element is an element that is strictly greater than its neighbors | 12-10-2022 |
|  | There is an integer array *nums* sorted in ascending order (with distinct values). After the possible **rotation** of the given array, find an integer *target*, return the index of *target* if it is in *nums*, or -1 if it is not in *nums*. | 12-10-2022 |
|  | Given an array *arr* of positive integers sorted in a strictly increasing order, and an integer *k*.  Write a function to return the *kth* positive integer that is missing from this array. | 12-10-2022 |
|  | Stack | Write a program to implement stack using array (Show all the operations like insertion, deletion and display) | 02-11-2022 |
|  | Write a program to convert Infix expression into Postfix expression and also analyze its Complexity. | 02-11-2022 |
|  | Write a program to evaluate the Postfix expression. | 02-11-2022 |
|  | Queue | Write a program to implement Simple Queue using arrays (Show all the operations like insertion, deletion and display) | 16-11-2022 |
|  | Write a program to implement Circular Queue using arrays (Show all the operations like insertion, deletion and display) | 16-11-2022 |
|  | Write a program to implement Priority Queue using both ordered and unordered arrays (Show all the operations like insertion, deletion and display) | 16-11-2022 |
|  | Linked List | Write a Program to insert and delete a node  (i) at the beginning  (ii) at the end  (iii) at any specific position of a singly linked list. | 23-11-2022 |
|  | Write a program to implement linked representation of Stack and Queue. | 23-11-2022 |
|  | Write a Program to insert and delete a node  (i) at the beginning  (ii) at the end  (iii) at any specific position of a Doubly linked list. | 23-11-2022 |
|  | Write a program to search an element in a given singly linked list. | 30-11-2022 |
|  | Write a program to search an element in a given doubly linked list. | 30-11-2022 |
|  | Write a Program to insert and delete a node  (i) at the beginning  (ii) at the end  (iii) at any specific position of a Circular linked list. | 30-11-2022 |
|  | Write a program to search an element in a given Circular linked list. | 07-12-2022 |
|  | Trees | Write a program to implement Binary Search Tree and its operations like insertion, deletion and searching. | 07-12-2022 |
|  | Write a program to traverse Binary Search Tree. | 07-12-2022 |
|  | Write a program to implement AVL tree and its operations like insertion and deletion. | 14-12-2022 |
|  | Sorting | Write a program to implement Selection sort. | 21-12-2022 |
|  | Write a program to implement Bubble sort. | 21-12-2022 |
|  | Write a program to implement Insertion sort. | 21-12-2022 |
|  | Write a program to implement Merge sort. | 28-12-2022 |
|  | Write a program to implement Quick sort. | 28-12-2022 |
|  | Write a program to implement Heap sort. | 28-12-2022 |
|  | Graph | Write a program to implement Breadth First Search traversal technique. | 04-01-2023 |
|  | Write a program to implement Depth First Search traversal technique. | 04-01-2023 |

**Searching**

**Program No. 1:** Given an array of integers *nums* and an integer *target*, write a function to search *target* in *nums*. If *target* exists, then return its index. Otherwise, return *-1*.

**Code:**

#include <bits/stdc++.h>

#define N 9

using namespace std;

int linear\_search(int\* A,int key){

    for(int i=0;i<N;i++){

        if(A[i]==key) return i;

    }

    return 0;

}

int main(){

    int A[N] = {1,4,3,24,5,26,64,11,9};

    int key;

    cout << "Enter the value to search for: ";

    cin >> key;

    int flag = linear\_search(A,key);

    if(flag){

        cout << "element found at index " << flag << endl;

    }else{

        return -1;

    }

}

**Input:**

Enter the value to search for: 11

**Output:**

**element found at index 7**

**Program No. 2:** Given an array of integers *nums* which is sorted in ascending order, and an integer *target*, write a function to search target in *nums*. If target exists, then return its index. Otherwise, return *-1*.

**Code:**

#include <bits/stdc++.h>

#define N 9

using namespace std;

int binary\_search(int\* A,int key){

    int start=0,end=N-1;

    while(start!=end){

        int mid=(start+end)/2;

        if(A[mid]==key) return mid;

        else if(A[mid]>key){

            end=mid-1;

        }else if(A[mid]<key){

            start=mid+1;

        }

    }

    return -1;

}

int main(){

    int A[N] = {1,2,3,4,5,6,7,8,9};

    int key;

    cout << "Enter the value to search for: ";

    cin >> key;

    int flag = binary\_search(A,key);

    if(flag){

        cout << "element found at index " << flag << endl;

    }else if(flag==-1){

        return -1;

    }

}

**Input:**

Enter the value to search for: 3

**Output:**

**element found at index 2**

**Program No. 3:** Given a sorted array of *n* elements, possibly with duplicates, find the number of occurrences of the *target* element.

**Code:**

#include <bits/stdc++.h>

#define N 9

using namespace std;

int binary\_search(int\* A,int key){

    int start=0,end=N-1;

    while(start!=end){

        int mid=(start+end)/2;

        if(A[mid]==key)return mid;

        else if(A[mid]>key){

            end=mid-1;

        }else if(A[mid]<key){

            start=mid+1;

        }

    }

    return -1;

}

int countOccurrences(int \*A, int key)

{

    int ind = binary\_search(A, key);

    if (ind == -1)

        return 0;

    int count = 1;

    int left = ind - 1;

    while (left >= 0 && A[left] == key)

        count++, left--;

    int right = ind + 1;

    while (right < N && A[right] == key)

        count++, right++;

    return count;

}

int main(){

    int A[N] = {1,2,3,3,3,3,7,8,9};

    int key;

    cout << "Enter the value to search for: ";

    cin >> key;

    int flag = countOccurrences(A,key);

    cout << "Number of occurences: " << flag << endl;

}

**Input:**

Enter the value to search for: 3

**Output:**

**Number of occurences: 4**

**Program No. 4:** Given a **0-indexed** integer array *nums*, find a **peak element**, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

\*A peak element is an element that is strictly greater than its neighbors.

**Code:**

#include <bits/stdc++.h>

#define N 9

using namespace std;

int main(){

    int A[N] = {3,26,34,63,68,57,63,1,32};

    int peaky,count=0;

    for(int i=0;i<N;i++){

        switch(i){

            case 0:

                if(A[0]>A[N-1]){

                    if(A[0]>A[1]){

                        peaky=0;

                        count++;

                    }

                }

                break;

            case N-1:

                if(A[0]<A[N-1]){

                    if(A[N-1]>A[N-2]){

                        peaky=N-1;

                        count++;

                    }

                }

                break;

            default:

                if(A[i]>A[i-1]){

                    if(A[i]>A[i+1]){

                        peaky=i;

                        count++;

                    }

                }

                break;

        }

    }

    cout << "There are " << count << " peaks and one of them is at " << peaky << "th index." << endl;

}

**Input:**

{3,26,34,63,68,57,63,1,32}

**Output:**

**There are 3 peaks and one of them is at 8th index.**

**Program No. 5:** There is an integer array *nums* sorted in ascending order (with distinct values). After the possible **rotation** of the given array, find an integer *target*, return the index of *target* if it is in *nums*, or -1 if it is not in *nums*.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int rotateArray(int \*arr,int size){

    int last=arr[size-1];

    for(int i=0;i<size;i++){

        arr[size-i]=arr[size-1-i];

    }

    arr[0]=last;

}

int main(){

    int size,n,index,data;

    bool found=false;

    cout<<"Enter the size of the array:";

    cin>>size;

    int arr[size];

    cout<<"Enter the elements of the array:"<<endl;

    for(int i=0;i<size;i++){

        cin>>arr[i];

    }

    cout<<"Enter the value you want to search in the array:";

    cin>>data;

    int start=0,end=size-1;

    while(start<=end){

        int mid=(start+end)/2;

        if(arr[mid]==data){

            found=true;

            index=mid;

            break;

        }

        else if(arr[mid]>data){

            end=mid-1;

        }

        else{

            start=mid+1;

        }

    }

    cout<<"Enter the times you want to rotate an array: ";

    cin>>n;

    for(int i=0;i<n;i++){

        rotateArray(arr,size);

    }

    index=(index+n)%size;

    if(found==true){

        cout<<data<<" found at "<<index<<endl;

    }

    else{

        cout<<"Element not found."<<endl;

    }

    return 0;

}

**Input:**

Enter the size of the array: 8

Enter the elements of the array: 3 5 8 12 29 30 39 46

Enter the value you want to search in the array: 30

Enter the times you want to rotate an array: 3

**Program No. 6:** Given an array *arr* of positive integers sorted in a strictly increasing order, and an integer *k*. Write a function to return the *kth* positive integer that is missing from this array.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int missingK(int a[], int k, int n){

    int difference = 0, ans = 0, count = k;

    bool flag = 0;

    if (a[0] != 1) {

        difference = a[0] - 1;

        if (difference >= count)

            return count;

        count -= difference;

    }

    for (int i = 0; i < n - 1; i++) {

        difference = 0;

        if ((a[i] + 1) != a[i + 1]) {

            difference += (a[i + 1] - a[i]) - 1;

            if (difference >= count) {

                ans = a[i] + count;

                flag = 1;

                break;

            }

            else

                count -= difference;

        }

    }

    if (flag)

        return ans;

    else

        return -1;

}

int main(){

    int a[] = { 2, 8, 9, 11, 19 };

    int k = 11;

    int n = sizeof(a) / sizeof(a[0]);

    int missing = missingK(a, k, n);

    cout << missing << endl;

    return 0;

}

**Input:**

{2, 8, 9, 11, 19}

**Output:**

**15**

**Stack**

**Program No. 7:** Write a program to implement stack using array (Show all the operations like insertion, deletion and display)

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Stack{

    int\* arr;

    int top;

    int size;

public:

    Stack(){

        size=1001;

        arr=new int[size];

        top=-1;

    }

    void isEmpty(){

        if(top==-1){

            cout<<"Stack is Empty"<<endl;

        }

        else{

            cout<<"Stack is not Empty"<<endl;

        }

    }

    void push(int data){

        if(top==size){

            cout<<"Full Stack Exception"<<endl;

        }

        else{

            top++;

            arr[top]=data;

        }

    }

    void pop(){

        if(top==-1){

            cout<<"Empty Stack Exception"<<endl;

        }

        else{

            int item=arr[top];

            top--;

        }

    }

    void Stack\_top(){

        if(top==-1){

             cout<<"Empty Stack Exception"<<endl;

        }

        else{

            cout<<arr[top]<<endl;

        }

    }

};

int main(){

    Stack S;

    S.push(10);

    S.push(20);

    S.push(30);

    S.Stack\_top();

    S.pop();

    S.Stack\_top();

    S.isEmpty();

    return 0;

}

**Input:**

**Output:**

**30**

**20**

**Stack is not Empty**

**Program No. 8:** Write a program to convert Infix expression into Postfix expression and also analyze its Complexity.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int prec(char c){

    if (c == '^')

        return 3;

    else if (c == '/' || c == '\*')

        return 2;

    else if (c == '+' || c == '-')

        return 1;

    else

        return -1;

}

void infixToPostfix(string s){

    stack<char> st;

    string result;

    for (int i = 0; i < s.length(); i++) {

        char c = s[i];

        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')

            || (c >= '0' && c <= '9'))

            result += c;

        else if (c == '(')

            st.push('(');

        else if (c == ')') {

            while (st.top() != '(') {

                result += st.top();

                st.pop();

            }

            st.pop();

        }

        else {

            while (!st.empty()

                   && prec(s[i]) <= prec(st.top())) {

                result += st.top();

                st.pop();

            }

            st.push(c);

        }

    }

    while (!st.empty()) {

        result += st.top();

        st.pop();

    }

    cout << result << endl;

}

int main(){

    string exp = "a+b\*(c^d-e)^(f+g\*h)-i";

    infixToPostfix(exp);

    return 0;

}

**Input:**

a+b\*(c^d-e)^(f+g\*h)-i

**Output:**

**abcd^e-fgh\*+^\*+i-**

**Program No. 9:** Write a program to evaluate the Postfix expression.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int evaluatePostfixExpression(string expression) {

    stack<int> st;

    for (int i = 0; i < expression.length(); i++){

        char c = expression[i];

        if (c >= '0' && c <= '9'){

            int temp = (int)(c - '0');

            st.push(temp);

        }

        else{

            int op1 = st.top();

            st.pop();

            int op2 = st.top();

            st.pop();

            switch(c){

                case '+':

                    st.push(op2 + op1);

                    break;

                case '-':

                    st.push(op2 - op1);

                    break;

                case '\*':

                    st.push(op2 \* op1);

                    break;

                case '/':

                    st.push(op2 / op1);

                    break;

            }

        }

    }

    return st.top();

}

int main() {

    string expression = "23\*45+\*";

    cout << evaluatePostfixExpression(expression) << endl;

    return 0;

}

**Input:**

23\*45+\*

**Output:**

**54**

**:**

**Queue**

**Program No. 10:** Write a program to implement Simple Queue using arrays (Show all the operations like insertion, deletion and display)

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Queue{

private:

    int\* arr;

    int size;

    int front;

    int rear;

public:

    Queue(){

        size=1001;

        arr=new int[size];

        front=0;

        rear=0;

    }

    bool isEmpty(){

        if(front==rear){

            return true;

        }

        else{

            return false;

        }

    }

    void enqueue(int data){

        if(rear==size){

            cout<<"Queue is full"<<endl;

        }

        else{

            arr[rear]=data;

            rear++;

        }

    }

    int dequeue(){

        if(front==rear){

            cout<<"Queue is Empty"<<endl;

            return -1;

        }

        else{

            int item=arr[front];

            arr[front]=-1;

            front++;

            if(front==rear){

                front=0;

                rear=0;

            }

            return item;

        }

    }

    void display(){

        if(isEmpty()==true){

            cout<<"Empty Queue"<<endl;

            return;

        }

        else{

            for(int i=front;i<rear;i++){

                cout<<arr[i]<<" ";

            }

            cout<<endl;

        }

    }

};

int main(){

    Queue Q;

    Q.enqueue(10);

    Q.display();

    Q.enqueue(20);

    Q.display();

    Q.dequeue();

    Q.display();

    return 0;

}

**Input:**

**Output:**

**10**

**10 20**

**20**

**Program No. 11:** Write a program to implement Circular Queue using arrays (Show all the operations like insertion, deletion and display)

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Circular\_Queue{

private:

    int\* arr;

    int size;

    int front;

    int rear;

public:

    Circular\_Queue(){

        size=5;

        arr=new int[size];

        front=-1;

        rear=-1;

    }

    bool isEmpty(){

        if(front==-1){

           return true;

        }

        else{

            return false;

        }

    }

    bool isFull(){

        if((rear==size-1 && front==0) || (rear==front-1)){

            return true;

        }

        else{

            return false;

        }

    }

    void enqueue(int data){

        if(isFull()==true){

            cout<<"Full Queue Exception"<<endl;

            return;

        }

        else{

            if(front==-1 && rear==-1){

                front=0;

            }

            rear=(rear+1)%size;

            arr[rear]=data;

        }

    }

    int dequeue(){

        if(isEmpty()==true){

            cout<<"Empty Queue Exception"<<endl;

        }

        else{

            int item=arr[front];

            if(front==rear){

                front=rear=-1;

            }

            else{

                front=(front+1)%size;

            }

        return item;

        }

    }

    void display(){

        if(isEmpty()==true){

            cout<<"Empty Queue"<<endl;

            return;

        }

        else{

            if(rear>front){

                for(int i=front;i<=rear;i++){

                cout<<arr[i]<<" ";

                }

                cout<<endl;

            }

            else{

                for(int i=rear;i<size;i++){

                cout<<arr[i]<<" ";

                }

                cout<<endl;

            }

        }

    }

};

int main(){

    Circular\_Queue Q;

    Q.enqueue(10);

    Q.enqueue(20);

    Q.enqueue(30);

    Q.enqueue(40);

    Q.enqueue(50);

    Q.display();

    Q.dequeue();

    Q.display();

    Q.enqueue(60);

    Q.display();

    return 0;

}

**Input:**

**Output:**

**10 20 30 40 50**

**20 30 40 50**

**60 20 30 40 50**

**Program No. 12:** Write a program to implement Priority Queue using both ordered and unordered arrays (Show all the operations like insertion, deletion and display)

**UNORDERED QUEUE**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Unordered\_Priority\_Queue{

    int\* arr;

    int size;

    int front;

    int rear;

public:

    Unordered\_Priority\_Queue(){

        size=1001;

        arr=new int[size];

        front=0;

        rear=0;

    }

    bool isEmpty(){

        if(front==rear){

            return true;

        }

        else{

            return false;

        }

    }

    bool isFull(){

        if(rear==size){

            return true;

        }

        else{

            return false;

        }

    }

    void enqueue(int data){

        if(isFull()==true){

            cout<<"Full Queue Exception"<<endl;

        }

        else{

            arr[rear]=data;

            rear++;

        }

    }

    int findMin(){

        int min=front;

        for(int i=front;i<rear;i++){

            if(arr[i]<arr[min]){

                min=i;

            }

        }

        return min;

    }

    void deleteMin(){

        int min=findMin();

        int item=arr[min];

        arr[min]=arr[rear-1];

        rear--;

    }

    void display(){

        if(isEmpty()==true){

            cout<<"Empty Queue"<<endl;

            return;

        }

        else{

            for(int i=front;i<rear;i++){

                cout<<arr[i]<<" ";

            }

            cout<<endl;

        }

    }

};

int main(){

    Unordered\_Priority\_Queue Q;

    Q.enqueue(8);

    Q.enqueue(3);

    Q.enqueue(9);

    Q.enqueue(0);

    Q.display();

    Q.deleteMin();

    Q.display();

    Q.deleteMin();

    Q.display();

    return 0;

}

**Input:**

**Output:**

**8 3 9 0**

**8 3 9**

**8 9**

**ORDERED QUEUE**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Ordered\_Priority\_Queue{

    int\* arr;

    int size;

    int front;

    int rear;

public:

    Ordered\_Priority\_Queue(){

        size=1001;

        arr=new int[size];

        front=0;

        rear=0;

    }

    bool isEmpty(){

        if(front==rear){

            return true;

        }

        else{

            return false;

        }

    }

    bool isFull(){

        if(rear==size){

            return true;

        }

        else{

            return false;

        }

    }

    void enqueue(int data){

        if(isFull()==true){

            cout<<"Full Queue Exception"<<endl;

        }

        else{

            int i=rear-1;

            while(i>=0 && data<arr[i]){

                arr[i+1]=arr[i];

                i--;

            }

            arr[i+1]=data;

            rear++;

        }

    }

    int deleteMin(){

        if(isEmpty()==true){

            cout<<"Empty Queue Exception"<<endl;

        }

        else{

            int item=arr[front];

            front++;

            return item;

        }

    }

    void display(){

        if(isEmpty()==true){

            cout<<"Empty Queue Exception"<<endl;

            return;

        }

        else{

            for(int i=front;i<rear;i++){

                cout<<arr[i]<<" ";

            }

            cout<<endl;

        }

    }

};

int main(){

    Ordered\_Priority\_Queue Q;

    Q.enqueue(2);

    Q.enqueue(4);

    Q.display();

    Q.enqueue(1);

    Q.display();

    Q.deleteMin();

    Q.display();

    return 0;

}

**Input:**

**Output:**

**2 4**

**1 2 4**

**2 4**

**Linked List**

**Program No. 13:** Write a Program to insert and delete a node

(i) at the beginning

(ii) at the end

(iii) at any specific position of a singly linked list.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

};

struct Node\* createNode(int data){

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

    n->data=data;

    n->next=NULL;

    return n;

}

void displayLinkedList(struct Node\* head){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while((ptr)!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->next;

    }

}

struct Node\*insertAtFirst(struct Node\* head,int data){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr->data=data;

    ptr->next=head;

    return ptr;

}

struct Node\*insertAtLast(struct Node\* head,int data){

    struct Node\* ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr2=head;

    while(ptr2->next!=NULL){

        ptr2=ptr2->next;

    }

     ptr2->next=ptr1;

    ptr1->next=NULL;

    ptr1->data=data;

    return head;

}

struct Node\* insertIndex(struct Node\* head,int data,int pos){

    struct Node\*ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\*ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr2=head;

    for(int i=1;i<pos-1;i++){

        ptr2=ptr2->next;

    }

    ptr1->data=data;

    ptr1->next=ptr2->next;

    ptr2->next=ptr1;

    return head;

}

struct Node\* deleteAtFirst(struct Node\* head){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr=head;

    head=head->next;

    free(ptr);

    return head;

}

struct Node\* deleteAtLast(struct Node\* head){

    struct Node\* ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr1=head;

    ptr2=head->next;

    while((ptr2->next)!=NULL){

        ptr1=ptr1->next;

        ptr2=ptr2->next;

    }

    ptr1->next=NULL;

    free(ptr2);

    return head;

}

struct Node\* deleteAtIndex(struct Node\* head,int pos){

    struct Node\* ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr1=head;

    ptr2=head->next;

    for(int i=1;i<pos-1;i++){

        ptr2=ptr2->next;

        ptr1=ptr1->next;

    }

    ptr1->next=ptr2->next;

    free(ptr2);

    return head;

}

int main(){

    struct Node\* head=createNode(7);

    struct Node\* second=createNode(11);

    struct Node\* third=createNode(14);

    struct Node\* fourth=createNode(20);

    head->next=second;

    second->next=third;

    third->next=fourth;

    fourth->next=NULL;

    // Before inserting

    displayLinkedList(head);

    // Inserting at first node.

    head=insertAtFirst(head,5);

    cout<<"Linked List after inserting 5 at first:"<<endl;

    displayLinkedList(head);

    // Inserting at last node.

    head=insertAtLast(head,23);

    cout<<"Linked List after inserting 23 at last:"<<endl;

    displayLinkedList(head);

    // Inserting at any index

    head=insertIndex(head,0,3);

    cout<<"Linked List after inserting 0 at 3rd position:"<<endl;

    displayLinkedList(head);

    // Deletion at first

    head=deleteAtFirst(head);

    cout<<"Linked List after deleting from first index"<<endl;

    displayLinkedList(head);

    // Deletion at last

    head=deleteAtLast(head);

    cout<<"Linked List after deleting from last index"<<endl;

    displayLinkedList(head);

    // Deletion at last

    head=deleteAtIndex(head,3);

    cout<<"Linked List after deleting from 3rd index"<<endl;

    displayLinkedList(head);

    return 0;

}

**Input:**

**Output:**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Linked List after inserting 5 at first:**

**Element: 5**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Linked List after inserting 23 at last:**

**Element: 5**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Element: 23**

**Linked List after inserting 0 at 3rd position:**

**Element: 5**

**Element: 7**

**Element: 0**

**Element: 11**

**Element: 14**

**Element: 20**

**Element: 23**

**Linked List after deleting from first index**

**Element: 7**

**Element: 0**

**Element: 11**

**Element: 14**

**Element: 20**

**Element: 23**

**Linked List after deleting from last index**

**Element: 7**

**Element: 0**

**Element: 11**

**Element: 14**

**Element: 20**

**Linked List after deleting from 3rd index**

**Element: 7**

**Element: 0**

**Element: 14**

**Element: 20**

**Program No. 14:** Write a program to implement linked representation of Stack and Queue.

**STACK USING LINKED LIST**

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

};

struct Node\* createNode(int data){

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

    n->data=data;

    n->next=NULL;

    return n;

}

void displayLinkedList(struct Node\* head){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while((ptr)!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->next;

    }

}

struct Node\*insertAtLast(struct Node\* head,int data){

    struct Node\* ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr2=head;

    while(ptr2->next!=NULL){

        ptr2=ptr2->next;

    }

     ptr2->next=ptr1;

    ptr1->next=NULL;

    ptr1->data=data;

    return head;

}

struct Node\* deleteAtLast(struct Node\* head){

    struct Node\* ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr1=head;

    ptr2=head->next;

    while((ptr2->next)!=NULL){

        ptr1=ptr1->next;

        ptr2=ptr2->next;

    }

    ptr1->next=NULL;

    free(ptr2);

    return head;

}

int main(){

    struct Node\* head=createNode(7);

    struct Node\* second=createNode(11);

    struct Node\* third=createNode(14);

    struct Node\* fourth=createNode(20);

    head->next=second;

    second->next=third;

    third->next=fourth;

    fourth->next=NULL;

    // Before inserting

    displayLinkedList(head);

    // Inserting at last node.

    head=insertAtLast(head,23);

    cout<<"Linked List after inserting 23 at last:"<<endl;

    displayLinkedList(head);

    // Deletion at last

    head=deleteAtLast(head);

    cout<<"Linked List after deleting from last index"<<endl;

    displayLinkedList(head);

    return 0;

}

**Input:**

**Output:**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Linked List after inserting 23 at last:**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Element: 23**

**Linked List after deleting from last index**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**QUEUE USING LINKED LIST**

**Code:**

#include<bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

};

struct Node\* createNode(int data){

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

    n->data=data;

    n->next=NULL;

    return n;

}

void displayLinkedList(struct Node\* head){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while((ptr)!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->next;

    }

}

struct Node\*insertAtLast(struct Node\* head,int data){

    struct Node\* ptr1=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr2=(struct Node\*)malloc(sizeof(struct Node));

    ptr2=head;

    while(ptr2->next!=NULL){

        ptr2=ptr2->next;

    }

     ptr2->next=ptr1;

    ptr1->next=NULL;

    ptr1->data=data;

    return head;

}

struct Node\* deleteAtFirst(struct Node\* head){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr=head;

    head=head->next;

    free(ptr);

    return head;

}

int main(){

    struct Node\* head=createNode(7);

    struct Node\* second=createNode(11);

    struct Node\* third=createNode(14);

    struct Node\* fourth=createNode(20);

    head->next=second;

    second->next=third;

    third->next=fourth;

    fourth->next=NULL;

    // Before inserting

    displayLinkedList(head);

    // Inserting at last node.

    head=insertAtLast(head,23);

    cout<<"Linked List after inserting 23 at last:"<<endl;

    displayLinkedList(head);

    // Deletion at first

    head=deleteAtFirst(head);

    cout<<"Linked List after deleting from first index"<<endl;

    displayLinkedList(head);

    return 0;

}

**Input:**

**Output:**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Linked List after inserting 23 at last:**

**Element: 7**

**Element: 11**

**Element: 14**

**Element: 20**

**Element: 23**

**Linked List after deleting from first index**

**Element: 11**

**Element: 14**

**Element: 20**

**Element: 23**

**Program No. 15:** Write a Program to insert and delete a node

(i) at the beginning

(ii) at the end

(iii) at any specific position of a Doubly linked list.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

    struct Node\* prev;

};

struct Node\* createNode(int data){

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

    n->data=data;

    n->next=NULL;

    n->prev=NULL;

    return n;

}

void display\_Linked\_List\_Forward(struct Node\* head){

    struct Node\* ptr=(struct  Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while(ptr!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->next;

    }

}

void display\_Linked\_List\_Reverse(struct Node\* head){

    struct Node\* ptr=(struct  Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while(ptr->next!=NULL){

        ptr=ptr->next;

    }

    while(ptr!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->prev;

    }

}

struct Node\* insertAtFirst(struct Node\* head,int data){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr->prev=NULL;

    ptr->data=data;

    ptr->next=head;

    head->prev=ptr;

    head=ptr;

    return head;

}

struct Node\* insertAtLast(struct Node\* head,int data){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr1=head;

    while(ptr1->next!=NULL){

        ptr1=ptr1->next;

    }

    ptr1->next=ptr;

    ptr->prev=ptr1;

    ptr->data=data;

    ptr->next=NULL;

    return head;

}

struct Node\* insertAtIndex(struct Node\* head,int data,int pos){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    struct Node\* ptr1=head->next;

    struct Node\* ptr2=head;

    for(int i=1;i<pos-1;i++){

        ptr1=ptr1->next;

        ptr2=ptr2->next;

    }

    ptr2->next=ptr;

    ptr->prev=ptr2;

    ptr->data=data;

    ptr->next=ptr1;

    ptr1->prev=ptr;

    return head;

}

struct Node\* deleteAtFirst(struct Node\* head){

    struct Node\* ptr=head;

    head=ptr->next;

    head->prev=NULL;

    free(ptr);

    return head;

}

struct Node\* deleteAtLast(struct Node\* head){

    struct Node\* ptr1=head;

    struct Node\* ptr2=head->next;

    while((ptr2->next)!=NULL){

        ptr1=ptr1->next;

        ptr2=ptr2->next;

    }

    ptr1->next=NULL;

    free(ptr2);

    return head;

}

struct Node\* deleteAtIndex(struct Node\* head,int pos){

    struct Node\* ptr1=head;

    struct Node\* ptr2=head->next;

    for(int i=1;i<pos-1;i++){

        ptr1=ptr1->next;

        ptr2=ptr2->next;

    }

    ptr1->next=ptr2->next;

    struct Node\* ptr3=ptr2;

    ptr2=ptr2->next;

    ptr2->prev=ptr1;

    free(ptr3);

    return head;

}

int main(){

    struct Node\* head=createNode(10);

    struct Node\* first=createNode(20);

    struct Node\* second=createNode(30);

    struct Node\* third=createNode(40);

    head->prev=NULL;

    head->next=first;

    first->prev=head;

    first->next=second;

    second->prev=first;

    second->next=third;

    third->prev=second;

    third->next=NULL;

    // Displaying

    cout<<"Displaying Linked List in Forward:"<<endl;

    display\_Linked\_List\_Forward(head);

    cout<<"Displaying Linked List in Reverse:"<<endl;

    display\_Linked\_List\_Reverse(head);

    // Insertion at first

    head=insertAtFirst(head,0);

    cout<<"Linked List after inserting 0 at first"<<endl;

    display\_Linked\_List\_Forward(head);

    // Insertion at last

    head=insertAtLast(head,50);

    cout<<"Linked List after inserting 50 at last"<<endl;

    display\_Linked\_List\_Forward(head);

    // Insertion at any index

    head=insertAtIndex(head,25,3);

    cout<<"Linked List after inserting 25 at 3rd index"<<endl;

    display\_Linked\_List\_Forward(head);

    // Deletion at first

    head=deleteAtFirst(head);

    cout<<"Linked List after deletion at first"<<endl;

    display\_Linked\_List\_Forward(head);

    // Deletion at last

    head=deleteAtLast(head);

    cout<<"Linked List after deletion at last"<<endl;

    display\_Linked\_List\_Forward(head);

    // Deletion at any index

    head=deleteAtIndex(head,3);

    cout<<"Linked List after deletion at 3rd index"<<endl;

    display\_Linked\_List\_Forward(head);

    return 0;

}

**Input:**

**Output:**

**Displaying Linked List in Forward:**

**Element: 10**

**Element: 20**

**Element: 30**

**Element: 40**

**Displaying Linked List in Reverse:**

**Element: 40**

**Element: 30**

**Element: 20**

**Element: 10**

**Linked List after inserting 0 at first**

**Element: 0**

**Element: 10**

**Element: 20**

**Element: 30**

**Element: 40**

**Linked List after inserting 50 at last**

**Element: 0**

**Element: 10**

**Element: 20**

**Element: 30**

**Element: 40**

**Element: 50**

**Linked List after inserting 25 at 3rd index**

**Element: 0**

**Element: 10**

**Element: 25**

**Element: 20**

**Element: 30**

**Element: 40**

**Element: 50**

**Linked List after deletion at first**

**Element: 10**

**Element: 25**

**Element: 20**

**Element: 30**

**Element: 40**

**Element: 50**

**Linked List after deletion at last**

**Element: 10**

**Element: 25**

**Element: 20**

**Element: 30**

**Element: 40**

**Linked List after deletion at 3rd index**

**Element: 10**

**Element: 25**

**Element: 30**

**Element: 40**

**Program No. 16:** Write a program to search an element in a given singly linked list.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

};

struct Node\* createNode(int data){

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

    n->data=data;

    n->next=NULL;

    return n;

}

void displayLinkedList(struct Node\* head){

    struct Node\* ptr=(struct Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while((ptr)!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->next;

    }

}

int search(struct Node\* head,int value){

    struct Node\* ptr=head;

    if(head!=NULL){

        while(ptr!=NULL){

            if(ptr->data==value){

                return ptr->data;

            }

            ptr=ptr->next;

        }

    }

    return -1;

}

int main(){

    struct Node\* head=createNode(7);

    struct Node\* second=createNode(11);

    struct Node\* third=createNode(1);

    struct Node\* fourth=createNode(20);

    head->next=second;

    second->next=third;

    third->next=fourth;

    fourth->next=NULL;

    // Displaying Linked List

    displayLinkedList(head);

    int value=search(head,10);

    if(value==-1){

        cout<<"Element not found."<<endl;

    }

    else{

        cout<<"Element "<<value<<" Found."<<endl;

    }

    return 0;

}

**Input:**

**Output:**

**Element: 7**

**Element: 11**

**Element: 1**

**Element: 20**

**Element not found.**

**Program No. 17:** Write a program to search an element in a given doubly linked list.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

    struct Node\* prev;

};

struct Node\* createNode(int data){

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

    n->data=data;

    n->next=NULL;

    n->prev=NULL;

    return n;

}

void display\_Linked\_List\_Forward(struct Node\* head){

    struct Node\* ptr=(struct  Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while(ptr!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->next;

    }

}

void display\_Linked\_List\_Reverse(struct Node\* head){

    struct Node\* ptr=(struct  Node\*)malloc(sizeof(struct Node));

    ptr=head;

    while(ptr->next!=NULL){

        ptr=ptr->next;

    }

    while(ptr!=NULL){

        cout<<"Element: "<<ptr->data<<endl;

        ptr=ptr->prev;

    }

}

int search(struct Node\* head,int value){

    struct Node\* ptr=head;

    if(head!=NULL){

        while(ptr!=NULL){

            if(ptr->data==value){

                return ptr->data;

            }

            ptr=ptr->next;

        }

    }

    return -1;

}

int main(){

    struct Node\* head=createNode(10);

    struct Node\* first=createNode(20);

    struct Node\* second=createNode(30);

    struct Node\* third=createNode(40);

    head->prev=NULL;

    head->next=first;

    first->prev=head;

    first->next=second;

    second->prev=first;

    second->next=third;

    third->prev=second;

    third->next=NULL;

    // Displaying

    cout<<"Displaying Linked List in Forward:"<<endl;

    display\_Linked\_List\_Forward(head);

    cout<<"Displaying Linked List in Reverse:"<<endl;

    display\_Linked\_List\_Reverse(head);

    int value=search(head,30);

    if(value==-1){

        cout<<"Element not found."<<endl;

    }

    else{

        cout<<"Element "<<value<<" Found."<<endl;

    }

    return 0;

}

**Input:**

**Output:**

**Displaying Linked List in Forward:**

**Element: 10**

**Element: 20**

**Element: 30**

**Element: 40**

**Displaying Linked List in Reverse:**

**Element: 40**

**Element: 30**

**Element: 20**

**Element: 10**

**Element 30 Found.**

**Program No. 18:** Write a Program to insert and delete a node

(i) at the beginning

(ii) at the end

(iii) at any specific position of a Circular linked list.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* addToEmpty(struct Node\* last, int data){

    if (last != NULL)

        return last;

    struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

    temp->data = data;

    last = temp;

    last->next = last;

    return last;

}

struct Node\* addBegin(struct Node\* last, int data){

    if (last == NULL)

        return addToEmpty(last, data);

    struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

    temp->data = data;

    temp->next = last->next;

    last->next = temp;

    return last;

}

struct Node\* addEnd(struct Node\* last, int data){

    if (last == NULL)

        return addToEmpty(last, data);

    struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

    temp->data = data;

    temp->next = last->next;

    last->next = temp;

    last = temp;

    return last;

}

struct Node\* addAfter(struct Node\* last, int data, int item){

    if (last == NULL)

        return NULL;

    struct Node \*temp, \*p;

    p = last->next;

    do {

        if (p->data == item) {

            temp

                = (struct Node\*)malloc(sizeof(struct Node));

            temp->data = data;

            temp->next = p->next;

            p->next = temp;

            if (p == last)

                last = temp;

            return last;

        }

        p = p->next;

    } while (p != last->next);

    cout << item << " not present in the list." << endl;

    return last;

}

void traverse(struct Node\* last){

    struct Node\* p;

    if (last == NULL) {

        cout << "List is empty." << endl;

        return;

    }

    p = last->next;

    do {

        cout << p->data << " ";

        p = p->next;

    } while (p != last->next);

}

int main()

{

    struct Node\* last = NULL;

    last = addToEmpty(last, 6);

    last = addBegin(last, 4);

    last = addBegin(last, 2);

    last = addEnd(last, 8);

    last = addEnd(last, 12);

    last = addAfter(last, 10, 8);

    traverse(last);

    return 0;

}

**Input:**

**Output:**

**2 4 6 8 10 12**

**Program No. 19:** Write a program to search an element in a given Circular linked list.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct node{

    int data;

    struct node \*next;

};

struct node \*head = NULL;

struct node \*tail = NULL;

void add(int data){

    struct node \*newNode = (struct node\*)malloc(sizeof(struct node));

    newNode->data = data;

    if(head == NULL){

        head = newNode;

        tail = newNode;

        newNode->next = head;

    }

    else{

        tail->next = newNode;

        tail = newNode;

        tail->next = head;

    }

}

void search(int element) {

    struct node \*current = head;

    int i = 1;

    bool flag = false;

    if(head == NULL) {

        cout<<"List is empty"<<endl;;

    }

    else {

        do{

            if(current->data ==  element) {

                flag = true;

                break;

            }

            current = current->next;

            i++;

        }while(current != head);

        if(flag)

            cout<<"Element "<<element<<" is present in the list at the position :"<<i<<endl;

        else

            cout<<"Element "<<element<<" is not present in the list"<<endl;

    }

}

int main(){

    add(1);

    add(2);

    add(3);

    add(4);

    search(2);

    search(7);

    return 0;

}

**Input:**

**Output:**

**Element 2 is present in the list at the position :2**

**Element 7 is not present in the list**

**Trees**

**Program No. 20:** Write a program to implement Binary Search Tree and its operations like insertion, deletion and searching.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class BST {

    int data;

    BST \*left, \*right;

public:

    BST();

    BST(int);

    BST\* Insert(BST\*, int);

    void Inorder(BST\*);

};

BST ::BST()

    : data(0)

    , left(NULL)

    , right(NULL)

{

}

BST ::BST(int value){

    data = value;

    left = right = NULL;

}

BST\* BST ::Insert(BST\* root, int value){

    if (!root) {

        return new BST(value);

    }

    if (value > root->data) {

        root->right = Insert(root->right, value);

    }

    else if (value < root->data){

        root->left = Insert(root->left, value);

    }

    return root;

}

void BST ::Inorder(BST\* root){

    if (!root) {

        return;

    }

    Inorder(root->left);

    cout << root->data << endl;

    Inorder(root->right);

}

int main(){

    BST b, \*root = NULL;

    root = b.Insert(root, 50);

    b.Insert(root, 30);

    b.Insert(root, 20);

    b.Insert(root, 40);

    b.Insert(root, 70);

    b.Insert(root, 60);

    b.Insert(root, 80);

    b.Inorder(root);

    return 0;

}

**Input:**

**Output:**

**20**

**30**

**40**

**50**

**60**

**70**

**80**

**Program No. 21:** Write a program to traverse Binary Search Tree.

**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node{

    int data;

    Node\* left;

    Node\* right;

};

struct Node\* createNode(int data){

    struct Node\* newNode=(struct Node\*) malloc(sizeof(struct Node));

    newNode->data=data;

    newNode->left=NULL;

    newNode->right=NULL;

    return newNode;

}

struct Node\* inorder(struct Node\* root){

    if(root!=NULL){

        inorder(root->left);

        cout<<root->data<<" ";

        inorder(root->right);

    }

}

struct Node\* preorder(struct Node\* root){

    if(root!=NULL){

        cout<<root->data<<" ";

        preorder(root->left);

        preorder(root->right);

    }

}

struct Node\* postorder(struct Node\* root){

    if(root!=NULL){

        postorder(root->left);

        postorder(root->right);

        cout<<root->data<<" ";

    }

}

/\*

           Let the BST be:

                5

               / \

              3   6

             / \

            1   4

\*/

int main(){

    Node\* root=createNode(5);

    Node\* first=createNode(3);

    Node\* second=createNode(6);

    Node\* third=createNode(1);

    Node\* fourth=createNode(4);

    // Linking

    root->left=first;

    root->right=second;

    first->left=third;

    first->right=fourth;

    // inorder traversal

    cout<<"Inorder traversal:";

    inorder(root);

    cout<<endl;

    // preorder traversal

    cout<<"preorder traversal:";

    preorder(root);

    cout<<endl;

    // postorder traversal

    cout<<"postorder traversal:";

    postorder(root);

    cout<<endl;

    return 0;

}

**Input:**

**Output:**

**Inorder traversal:1 3 4 5 6**

**preorder traversal:5 3 1 4 6**

**postorder traversal:1 4 3 6 5**

**Program No. 22:** Write a program to implement AVL tree and its operations like insertion and deletion.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Node{

    public:

    int key;

    Node \*left;

    Node \*right;

    int height;

};

int height(Node \*N){

    if (N == NULL)

        return 0;

    return N->height;

}

int max(int a, int b){

    return (a > b)? a : b;

}

Node\* newNode(int key){

    Node\* node = new Node();

    node->key = key;

    node->left = NULL;

    node->right = NULL;

    node->height = 1;

    return(node);

}

Node \*rightRotate(Node \*y){

    Node \*x = y->left;

    Node \*T2 = x->right;

    x->right = y;

    y->left = T2;

    y->height = max(height(y->left),

                    height(y->right)) + 1;

    x->height = max(height(x->left),

                    height(x->right)) + 1;

    return x;

}

Node \*leftRotate(Node \*x){

    Node \*y = x->right;

    Node \*T2 = y->left;

    y->left = x;

    x->right = T2;

    x->height = max(height(x->left),

                    height(x->right)) + 1;

    y->height = max(height(y->left),

                    height(y->right)) + 1;

    return y;

}

int getBalance(Node \*N){

    if (N == NULL)

        return 0;

    return height(N->left) - height(N->right);

}

Node\* insert(Node\* node, int key)

{

    if (node == NULL)

        return(newNode(key));

    if (key < node->key)

        node->left = insert(node->left, key);

    else if (key > node->key)

        node->right = insert(node->right, key);

    else

        return node;

    node->height = 1 + max(height(node->left),height(node->right));

    int balance = getBalance(node);

    // Left Left Case

    if (balance > 1 && key < node->left->key)

        return rightRotate(node);

    // Right Right Case

    if (balance < -1 && key > node->right->key)

        return leftRotate(node);

    // Left Right Case

    if (balance > 1 && key > node->left->key){

        node->left = leftRotate(node->left);

        return rightRotate(node);

    }

    // Right Left Case

    if (balance < -1 && key < node->right->key){

        node->right = rightRotate(node->right);

        return leftRotate(node);

    }

    return node;

}

void preOrder(Node \*root){

    if(root != NULL){

        cout << root->key << " ";

        preOrder(root->left);

        preOrder(root->right);

    }

}

int main(){

    Node \*root = NULL;

    root = insert(root, 10);

    root = insert(root, 20);

    root = insert(root, 30);

    root = insert(root, 40);

    root = insert(root, 50);

    root = insert(root, 25);

    cout << "Preorder traversal of the constructed AVL tree is:"<<endl;

    preOrder(root);

     /\*

    The constructed AVL Tree would be

            30

            / \

           20 40

          / \   \

         10 25   50

    \*/

    return 0;

}

**Input:**

**Output:**

**Preorder traversal of the constructed AVL tree is:**

**30 20 10 25 40 50**

**Sorting**

**Program No. 23:** Write a program to implement Selection sort.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int findminindex(vector<int> &A,int start){

    int min\_index = start;

    ++start;

    while(start < A.size()){

        if(A[start] < A[min\_index]){

            min\_index = start;

        }

        ++start;

    }

    return min\_index;

}

void selection\_sort(vector<int> &A){

    for(int i = 0; i < A.size(); i++){

        int min\_index = findminindex(A,i);

        if(i != min\_index){

            swap(A[min\_index],A[i]);

        }

    }

}

int main(){

    vector<int> A = {8,7,6,5,4,3,2,1};

    selection\_sort(A);

    for(int i = 0; i < A.size(); i++){

        cout << A[i] << " ";

    }

    cout << endl;

    return 0;

}

**Input:**

{8,7,6,5,4,3,2,1}

**Output:**

**1 2 3 4 5 6 7 8**

**Program No. 24:** Write a program to implement Bubble sort.

**Code:**

#include <bits/stdc++.h>

using namespace std;

void bubble\_sort(int \*arr, int arrsize){

    for(int i = 0; i < arrsize-1; i++){

        int flag = 0;

        for(int j = 0; j < arrsize-i-1; j++){

            if(arr[j] > arr[j+1]){

                swap(arr[j],arr[j+1]);

                flag = 1;

            }

        }

        if(flag == 0){

            break;

        }

    }

}

int main(){

    int A[] = {5,4,3,2,1};

    bubble\_sort(A,5);

    for(int i = 0; i < 5; i++){

        cout << A[i] << " ";

    }

    cout << endl;

}

**Input:**

{5,4,3,2,1}

**Output:**

**1 2 3 4 5**

**Program No. 25:** Write a program to implement Insertion sort.

**Code:**

#include <bits/stdc++.h>

using namespace std;

//member functions declaration

void insertionSort(int arr[], int length);

void printArray(int array[], int size);

// main function

int main()

{

    int array[6] = {5, 1, 6, 2, 4, 3};

    // calling insertion sort function to sort the array

    insertionSort(array, 6);

    return 0;

}

void insertionSort(int arr[], int length)

{

    int i, j, key;

    for (i = 1; i < length; i++)

    {

        key = arr[i];

        j = i-1;

        while (j >= 0 && arr[j] >key)

        {

            arr[j+1] = arr[j];

            j--;

        }

        arr[j +1] = key;

    }

    cout << "Sorted Array: ";

    // print the sorted array

    printArray(arr, length);

}

// function to print the given array

void printArray(int array[], int size)

{

    int j;

    for (j = 0; j < size; j++)

    {

        cout <<" "<< array[j];

    }

    cout << endl;

}

**Input:**

{5, 1, 6, 2, 4, 3}

**Output:**

**1 2 3 4 5 6**

**Program No. 26:** Write a program to implement Merge sort.

**Code:**

#include <bits/stdc++.h>

using namespace std;

void merge(int \*arr, int start, int mid, int end){

    int temp\_arr[end - start + 1];

    int i = start;

    int j = mid + 1;

    int k = 0;

    while(i <= mid && j <= end){

        if(arr[i] <= arr[j]){

            temp\_arr[k] = arr[i];

            i++;

            k++;

        }else{

            temp\_arr[k] = arr[j];

            j++;

            k++;

        }

    }

    while (i <= mid){

        temp\_arr[k] = arr[i];

        i++;

        k++;

    }

    while(j <= end){

        temp\_arr[k] = arr[j];

        j++;

        k++;

    }

    for(i = start; i <= end; i++){

        arr[i] = temp\_arr[i - start];

    }

}

void merge\_sorting(int \*arr, int start, int end){

    if(start < end){

        int mid = (start + end)/2;

        merge\_sorting(arr, start, mid);

        merge\_sorting(arr, mid+1, end);

        merge(arr, start, mid, end);

    }

}

int main(){

    int arr[8] = {8,7,6,5,4,3,2,1};

    merge\_sorting(arr, 0, 7);

    for(int i = 0; i < 8; i++){

        cout << arr[i] << " ";

    }

    cout << endl;

}

**Input:**

{8,7,6,5,4,3,2,1}

**Output:**

**1 2 3 4 5 6 7 8**

**Program No. 27:** Write a program to implement Quick sort.

**Code:**

#include <bits/stdc++.h>

using namespace std;

void quick\_sort(int \*arr, int start, int end){

    int pindex = end;

    for(int i = 0; i < end; i++){

        if(arr[pindex] <= arr[i]){

            swap(arr[i],arr[pindex]);

        }

    }

    if(start >= end){

        return;

    }else{

        quick\_sort(arr,start,pindex-1);

        quick\_sort(arr,pindex+1,end);

    }

}

int main(){

    int A[8] = {24,345,35,452,3,52,534,222};

    quick\_sort(A,0,7);

    for(int l = 0; l < 8; l++){

        cout << A[l] << " ";

    }

    cout << endl;

    return 0;

}

**Input:**

{24,345,35,452,3,52,534,222}

**Output:**

**3 24 35 52 222 345 452 534**

**Program No. 28:** Write a program to implement Heap sort.

**Code:**

#include <bits/stdc++.h>

using namespace std;

void heapify(int arr[], int N, int i){

    int largest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    if (l < N && arr[l] > arr[largest])

        largest = l;

    if (r < N && arr[r] > arr[largest])

        largest = r;

    if (largest != i) {

        swap(arr[i], arr[largest]);

        heapify(arr, N, largest);

    }

}

void heapSort(int arr[], int N){

    for (int i = N / 2 - 1; i >= 0; i--){

        heapify(arr, N, i);

    }

    for (int i = N - 1; i > 0; i--) {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0);

    }

}

void printArray(int arr[], int N){

    for (int i = 0; i < N; ++i){

        cout << arr[i] << " ";

    }

    cout << endl;

}

int main(){

    int arr[] = { 12, 11, 13, 5, 6, 7 };

    int N = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, N);

    cout << "Sorted array is:"<<endl;

    printArray(arr, N);

    return 0;

}

**Input:**

{12,11,13,5,6,7}

**Output:**

**Sorted array is:**

**5 6 7 11 12 13**

**Program No. 29:** Write a program to implement Breadth First Search traversal technique.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Graph {

    int V;

    vector<list<int> > adj;

public:

    Graph(int V);

    void addEdge(int v, int w);

    void BFS(int s);

};

Graph::Graph(int V){

    this->V = V;

    adj.resize(V);

}

void Graph::addEdge(int v, int w){

    adj[v].push\_back(w);

}

void Graph::BFS(int s){

    vector<bool> visited;

    visited.resize(V, false);

    list<int> queue;

    visited[s] = true;

    queue.push\_back(s);

    while (!queue.empty()) {

        s = queue.front();

        cout << s << " ";

        queue.pop\_front();

        for (auto adjecent : adj[s]) {

            if (!visited[adjecent]) {

                visited[adjecent] = true;

                queue.push\_back(adjecent);

            }

        }

    }

}

int main(){

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal (starting from vertex 2):"<<endl;

    g.BFS(2);

    return 0;

}

**Input:**

**Output:**

**Following is Breadth First Traversal (starting from vertex 2):**

**2 0 3 1**

**Program No. 30:** Write a program to implement Depth First Search traversal technique.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Graph {

public:

    map<int, bool> visited;

    map<int, list<int> > adj;

    void addEdge(int v, int w);

    void DFS(int v);

};

void Graph::addEdge(int v, int w){

    adj[v].push\_back(w);

}

void Graph::DFS(int v){

    visited[v] = true;

    cout << v << " ";

    list<int>::iterator i;

    for (i = adj[v].begin(); i != adj[v].end(); ++i)

        if (!visited[\*i])

            DFS(\*i);

}

int main(){

    Graph g;

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal (starting from vertex 2):"<<endl;

    g.DFS(2);

    return 0;

}

**Input:**

**Output:**

**Following is Depth First Traversal (starting from vertex 2):**

**2 0 1 3**