```
#include<iostream>
using namespace std;
class Node
{
public:
int data;
Node *next;
Node(int d)
{ data=d;
next=NULL;
}
};
class Linkedlist
{
public:
Node *hptr=NULL;
void create(int d)
{
if(hptr==NULL)
{
Node *temp=new Node(d);
hptr=temp;
}
else
{
Node *temp=hptr;
while(temp->next!=NULL)
{
temp=temp->next;
}
Node *nptr=new Node(d);
temp->next=nptr;
}
}
void display()
{
Node *t=hptr;
while(t!=NULL)
{
cout<<t->data<<"->";
t=t->next;
}
}
void deletenode(int x)
{
if(hptr==NULL)
{
cout<<"List is empty\n";
return;
}
Node *temp=hptr,*prev;
while(temp!=NULL)
{
if(temp->data==x)
{
if(temp==hptr)
{
}
else
{
}
hptr=temp->next;
prev->next=temp->next;
cout<<"Node deleted\n";
delete temp;
return;
}
prev=temp;
temp=temp->next;
}
cout<<"Element not found\n";
}
void addatbeg(int x)
{
Node *temp=hptr;
Node *nn=new Node(x);
if(nn==NULL)
{
cout<<"Unable to create node\n";
return;
}
nn->next=temp;
hptr=nn;
}
void addatpos(int x,int p)
{
Node *temp=hptr;
Node *nn=new Node(x);
for(int i=1;i<p;i++)
{
temp=temp->next;
if(temp==NULL)
{
create(x);
return;
}
}
nn->next=temp->next;
temp->next=nn;
}
void length()
{
Node *t=hptr;
int len=0;
while(t!=NULL)
{
//cout<<t->data<<"->";
t=t->next;
len++;
}
cout<<"length of the list:"<<len;
}
};
int main()
{
Linkedlist l;
int x,ch,ele,pos;
while(1)
{
cout<<"\n1.Create\n";
```

```
cout<<"2.Display\n";
cout<<"3.Insertion_at_begin\n";
cout<<"4.Insert_at_position\n";
cout<<"5.Delete\n";
cout<<"6.length\n";
cout<<"7.exit\n";
cout<<"Enter choice\n";
cin>>ch;

switch(ch)
{
case 1:cout<<"Enter a value\n";
cin>>x;
l.create(x);
break;
case 2: l.display();
break;
case 3:cout<<"Enter a value\n";
cin>>x;
l.addatbeg(x);
break;
case 4: cout<<"Enter a value\n";
cin>>x;
cout<<"enter the position\n";
cin>>pos;
l.addatpos(x,pos);
break;
case 5: cout<<"Enter a element to delete\n";
cin>>ele;
l.deletenode(ele);
break;
case 6:l.length();
break;
case 7:exit(0);
}
}
return 0;
}
```

```
#include<iostream>
using namespace std;
class Node
{
public:
int data;
Node *next;
Node *prev;
Node(int d)
{
data=d;
next=NULL;
prev=NULL;
}
void display()
{
cout<<data<<endl;
}
};
class DLL
{
public:
Node *hptr=NULL;
Node *tptr=NULL;
void create(int d)
{
Node *nptr = new Node(d);
if(hptr == NULL)
{
hptr = tptr = nptr;
}
else
{
tptr->next = nptr;
nptr->prev = tptr;
tptr = nptr;
}
}
// add node at front / Beginning function

void addatbeg(int d)
{

Node *nptr = new Node(d); // new allocate new node oftype Node and return
address
if(hptr == NULL)
{
hptr = tptr = nptr;
}
else
{
hptr->prev = nptr; // previous first node point to new node
nptr->next = hptr; // new node next point to previous first node whose
address is
hptr = nptr; // head point to new node
}
}

void delete_node(int d)
{
```

```
if(hptr->data == d)
{
Node *temp = hptr;
temp->next->prev = NULL;
hptr = temp->next;
}
else
{
if(tptr->data == d)
{
Node *temp = tptr;
temp->prev->next = NULL;
tptr = temp->prev;
}
else
{
Node* temp = hptr;
while(temp!=NULL)
{
if(temp->next!=NULL && temp->next->data == d)
{
temp->next = temp->next->next;
temp->next->prev = temp;
break;
}
temp = temp->next;
}
if(temp == NULL)
cout<<"Element Not Found!"<<endl;
}// else ends
}// else ends

}
void forward_display()
{
Node *temp = hptr;
while(temp!=NULL)
{
temp->display();
temp = temp->next;
}
}
void reverse_display()
{
Node *temp = tptr;
while(temp!=NULL)
{
temp->display();
temp = temp->prev;
}
}
void addAfter(int ndata, int afteritem)
{
if (hptr == NULL)
cout<<"List is empty";
else
{
Node *temp, *p;
p = hptr;
do
```

```
{
if(p ->data == afteritem)
{
Node *temp = new Node(ndata);
temp -> next = p -> next;
temp ->prev = p;
p->next->prev =temp;
p -> next = temp;
/*if(p->next == NULL)
last = temp;*/
return;
}
p = p -> next;
}while(p!=NULL);
cout <<"\n"<< afteritem << " not present in the list." << endl;
return;
}

}
};
int main()
{
DLL dll;
int ch, x , ele, after;
while(1)
{
cout<<"\n1.Create\n";
cout<<"2.Display\n";
cout<<"3.Insertion_at_begin\n";
cout<<"4.Insert value after specified value\n";
cout<<"5.Delete\n";
cout<<"6.Print in reverse\n";
cout<<"7.exit\n";
cout<<"Enter choice\n";
cin>>ch;
switch(ch)
{
case 1:cout<<"Enter a value\n";
cin>>x;
dll.create(x);;
break;
case 2: cout<<"Elements of List "<<endl;
dll.forward_display();
break;
case 3:cout<<"Enter a value\n";
cin>>x;
dll.addatbeg(x);
break;
case 4: cout<<"Enter a value\n";
cin>>x;
cout<<"enter the value after which to add\n";
cin>>after;
dll.addAfter(x,after);
break;
case 5: cout<<"Enter a element to delete\n";
cin>>ele;
dll.delete_node(ele);
break;
case 6: cout<<"Reverse Display"<<endl;
dll.reverse_display();
```

```
break;
case 7:exit(0);
break;
}
}
return 0;
}
```

```
/*
* C++ Programto Implement Circular Linked List
*/
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;
/*
* Node Declaration
*/
class node
{
public:
int info;
node *next;
}*last;
/*
* Class Declaration
*/
class circular_llist
{
public:
void create_node(int value);
void add_begin(int value);
void add_after(int value, int position);
void delete_element(int value);
void display_list();
void count();
circular_llist()
{
last = NULL;
}
};
/*
* Main :contains menu
*/
int main()
{
int choice, element, position;
circular_llist cl;

while (1)
{
cout<<endl<<"---------------------------"<<endl;
cout<<endl<<"Circular singly linked list"<<endl;
cout<<endl<<"---------------------------"<<endl;
cout<<"1.Create Node"<<endl;
cout<<"2.Add at beginning"<<endl;
cout<<"3.Add after"<<endl;
cout<<"4.Delete"<<endl;
cout<<"5.Display"<<endl;
cout<<"6.Count"<<endl;
cout<<"7.Quit"<<endl;
cout<<"Enter your choice : ";
cin>>choice;
switch(choice)
{
case 1:
cout<<"Enter the element: ";
cin>>element;
cl.create_node(element);
cout<<endl;
break;
case 2:
cout<<"Enter the element: ";
cin>>element;
cl.add_begin(element);
cout<<endl;
break;
case 3:
cout<<"Enter the element: ";
cin>>element;
cout<<"Insert element after position: ";
cin>>position;
cl.add_after(element, position);
cout<<endl;
break;
case 4:
if(last == NULL)
{
cout<<"List is empty, nothing to delete"<<endl;
break;
}
cout<<"Enter the element for deletion: ";
cin>>element;
cl.delete_element(element);
cout<<endl;
break;
case 5:
cl.display_list();

break;
case 6:
cl.count();
break;
case 7:
exit(1);
break;
default:
cout<<"Wrong choice"<<endl;
}
}
return 0;
}
/*
* Create Circular Link List
*/
void circular_llist::create_node(int value)
{
node *temp;
temp = new(node);
temp->info = value;
if(last == NULL)
{
last = temp;
temp->next = last;
}
else
{
temp->next = last->next;
```

```
last->next = temp;
last = temp;
}
}
/*
* Insertion of element at beginning
*/
void circular_llist::add_begin(int value)
{
if(last == NULL)
{
cout<<"First Create the list."<<endl;
return;
}
node *temp;
temp = new(node);
temp->info = value;
temp->next = last->next;

last->next = temp;
}
/*
* Insertion of element at a particular place
*/
void circular_llist::add_after(int value, int pos)
{
if(last == NULL)
{
cout<<"First Create the list."<<endl;
return;
}
node *temp, *s;
s = last->next;
for (int i = 0;i < pos-1;i++)
{
s = s->next;
if(s == last->next)
{
cout<<"There are less than ";
cout<<pos<<" in the list"<<endl;
return;
}
}
temp = new(node);
temp->next = s->next;
temp->info = value;
s->next = temp;
/*Element inserted at the end*/
if (s == last)
{
last=temp;
}
}
/*
* Deletion of element fromthe list
*/
void circular_llist::delete_element(int value)
{
node *temp, *s;
s = last->next;
```

```
/* If List has only one element*/
if(last->next == last && last->info == value)
{
temp = last;
last = NULL;
free(temp);
return;

}
if(s->info == value) /*First Element Deletion*/
{
temp = s;
last->next = s->next;
free(temp);
return;
}
while (s->next != last)
{
/*Deletion of Element in between*/
if(s->next->info == value)
{
temp = s->next;
s->next = temp->next;
free(temp);
cout<<"Element "<<value;
cout<<" deleted from the list"<<endl;
return;
}
s = s->next;
}
/*Deletion of last element*/
if(s->next->info == value)
{
temp = s->next;
s->next = last->next;
free(temp);
last = s;
return;
}
cout<<"Element "<<value<<" not found in the list"<<endl;
}
/*
* DisplayCircular Link List
*/
void circular_llist::display_list()
{
node *s;
if(last == NULL)
{
cout<<"List is empty, nothing to display"<<endl;
return;
}
s = last->next;
cout<<"Circular Link List: "<<endl;
while (s != last)

{
cout<<s->info<<"->";
s = s->next;
}
```

```
cout<<s->info<<endl;
}
/*
* count Circular Link List
*/
void circular_llist::count()
{
node *temp = last;
int c=0;
if (last == NULL)
{
cout << "List is empty." << endl;
return;
}
// Pointing to first Nodeofthe list.
temp = last -> next;
// Traversing the list.
do
{
c++;
temp = temp-> next;
}
while(temp != last->next);
cout<<"Total no of nodes in list:"<<c;
}
```

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 3
class Stack {
int top;
public:
int a[MAX]; // Maximum size of Stack
Stack()
{ top = -1;
}
void push(int x);
void pop();
int peek();
bool isEmpty();
void display()
{
cout<<"The elements of stack are \n";
for(int i=top;i>=0;i--)
{
cout<<a[i]<<endl;
}
}
};
void Stack::push(int x)
{
if (top >= (MAX - 1)) {
cout << "Stack Overflow";
}

else {
a[++top] = x;
cout << x << " pushed into stack\n";
}
}
void Stack::pop()
{
if (top < 0) {
cout << "Stack Underflow";
}
else {
int x = a[top--];
cout<<"deleted element: "<<x;
}
}
int Stack::peek()
{
if (top < 0) {
cout << "Stack is Empty";
return 0;
}
else {
int x = a[top];
cout<<x<<":The top element of stack";
return 0;
}
}
bool Stack::isEmpty()
{
return (top < 0);
}
```

```
// Driver programto test above functions
int main()
{
Stack s;
int ch,ele;
while(1)
{
cout<<"\n1.push 2.pop 3.display 4. peek 5.exit\nEnter ur choice ";
cin>>ch;
switch(ch)

{
case 1: cout<<"enter the element ";
cin>>ele;
s.push(ele);
break;
case 2: s.pop();
break;
case 3: s.display();
break;
case 4: s.peek();
break;
case 5: exit(0);
}
}
return 0;
}
```

```
// Stack using linked list
#include <iostream>
using namespace std;
//Structure of the Node
class Node
{
public:
int data;
Node *link;
};
class Stack1
{
public:
// top pointer to keep track of the top of the stack
Node *top = NULL;
//Function to check if stack is empty or not
bool isempty()
{
if(top == NULL)
return true;
else
return false;
}

//Function to insert an element in stack
void push (int value)
{
Node *ptr = new Node();
ptr->data = value;
ptr->link = top;
top = ptr;
}
//Function to delete an element from the stack
void pop ( )
{
if( isempty() )
cout<<"Stack is Empty";
else
{
Node *ptr = top;
top = top -> link;
delete(ptr);
// Function to show the element at the top of the stack
}
}

void showTop()
{
if ( isempty() )
cout<<"Stack is Empty";
else
cout<<"Element at top is : "<< top->data;
}
// Function to Display the stack
void displayStack()
{
if ( isempty() )
cout<<"Stack is Empty";
else
{
```

```
Node *temp=top;
while(temp!=NULL)
{
cout<<temp->data<<" ";
temp=temp->link;
}
cout<<"\n";
}
}
};
// Main function

int main()
{
Stack1 s;
int choice, flag=1, value;
//Menu Driven Program using Switch
while( flag == 1)
{
cout<<"\n1.Push 2.Pop 3.showTop 4.displayStack 5.exit\n";
cin>>choice;
switch (choice)
{
case 1: cout<<"Enter Value:\n";
cin>>value;
s.push(value);
break;
case 2: s.pop();
break;
case 3: s.showTop();
break;
case 4: s.displayStack();
break;
case 5: flag = 0;
break;
}
}
return 0;
}
```

```
#include<iostream>
#include<stack>
#include<string>
using namespace std;
class ITFConversion
{
public:
int prec(char c)
{
if(c == '+' || c == '-'){
return 1;
}
if(c == '*' || c == '/'){
return 2;
}
}
bool isOperator(char c)
{
if(c == '+'|| c =='-'|| c =='*' || c =='/')
{
return true;
}
else
{
return false;
}
}
string infixToPostfix(string s)
{
stack<char> st;
string res;
for(int i=0;i<s.length();i++)
{
if(!isOperator(s[i])){
res = res+s[i];
continue;

}
if(st.empty()){
st.push(s[i]);
}
else{
while(!st.empty() && prec(st.top())>=prec(s[i])){
res = res + st.top();
st.pop();
}
st.push(s[i]);
}
}
while(!st.empty())
{
res = res + st.top();
st.pop();
}
return res;
}
};
int main()
{
ITFConversion itf;
string exp ;
cout<<"Enter expression"<<endl;
cin>>exp;
cout<<"Postfix form: "<<itf.infixToPostfix(exp)<<endl;
return 0;
}
```

```
#include<iostream>
#include<string.h>
#include<bits/stdc++.h>
using namespace std;
stack<int> s;
int main()
{
//taking user input
string exp;
cout<<"Enter postfix expression: ";
cin>>exp;
//traversing postfix expression from left to right
for(int i=0;i<exp.length();i++)
{
//ifsymbol is a digit push it in stack
if (isdigit(exp[i]))
s.push(exp[i] - '0');
//ifsymbol is an operator then pop top 2 elements from stack, perform
specific operation
//and push the result back into stack
else
{
int op2=s.top();
s.pop();
int op1=s.top();
s.pop();
if(exp[i]=='+')
s.push(op1+op2);
else if(exp[i]=='-')

s.push(op1-op2);
else if(exp[i]=='*')
s.push(op1*op2);
else if(exp[i]=='/')
s.push(op1/op2);
}
}
cout<<"After evalution we get: "<<s.top();
return 0;
}
```

```
#include <iostream>
#include<stack>
using namespace std;
class BracesBalance
{
public:
bool areBalancedBraces(string str)
{
int i;
char c;
int n = str.length();
stack<char> st;
for (i = 0; i < n; i++)
{
c = str.at(i);
if (st.empty())
{
st.push(c);
}
else
if (st.top() == '(' && c == ')' || st.top() == '{' && c == '}'|| st.top()
== '[' && c == ']')
{
st.pop();
}
else


st.push(c);
}
if (st.empty())
{
return true;
}
else

{
return false;
}
}
};
int main()
{
BracesBalance bb;
string expr;
cout<<"Enter expression"<<endl;
cin>>expr;
if (bb.areBalancedBraces(expr))
cout << "Braces are balanced";
else
cout << "Braces are imbalanced";
return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
// Function that returns true
// if string is a palindrome
bool isPalindrome(string s)
{
int length = s.size();
// Creating a Stack
stack<char> st;
// Finding the mid
int i, mid = length / 2;
for (i = 0; i < mid; i++) {
st.push(s[i]);
}
// Checking ifthe length of the string
// is odd, if odd then neglect the
// middle character
if (length % 2 != 0) {
i++;
}
char ele;
// While not the end ofthe string
while (s[i] != '\0')
{
ele = st.top();
st.pop();
// Ifthe characters differ then the
// given string is not a palindrome
if (ele != s[i])
return false;
i++;
}
return true;
}
// Driver code
int main()
{
string s;

cout << "Enter the string"<<endl;
cin>>s;
if (isPalindrome(s))
{
}
else
{
}
cout << "String is palindrome";
cout << "String is not palindrome";
return 0;
}
```

```
#include <iostream>
#define MAX_SIZE 3
using namespace std;
class Queue
{
private:
int myqueue[MAX_SIZE], front, rear;
public:
Queue(){
front = -1;
rear = -1;
}
bool isFull(){
if(front == 0 && rear == MAX_SIZE - 1){
return true;
}
return false;
}
bool isEmpty(){
if(front == -1) return true;
else return false;
}
void enQueue(int value){
if(isFull()){
cout << endl<< "Queue is full!!";
} else {
if(front == -1) front = 0;
rear++;
myqueue[rear] = value;
cout << value << " ";
}

}
int deQueue(){
int value;
if(isEmpty())
{
cout << "Queue is empty!!" << endl; return(-1);
}
else
{
value = myqueue[front]; if(front >= rear){
//only one element in queue
front = -1;
rear = -1;
}
else{
front++;
}


front++;
cout << endl << "Deleted => " << value << " from myqueue";
return(value);
}
}
/* Function to display elements of Queue */
void displayQueue()
{
int i;
if(isEmpty()) {
cout << endl << "Queue is Empty!!" << endl;
}
else {
cout << endl << "Front = " << front;
cout << endl << "Queue elements : ";
for(i=front; i<=rear; i++)
cout << myqueue[i] << "\t";
cout << endl << "Rear = " << rear << endl;
}
}
};
int main()
{
Queue myq;
int ch,ele;
while(1){
cout<<"\n1.Enqueue(insertion) 2.Dequeue(deletion) 3.display 4. Exit
\nEnter ur choice ";
cin>>ch;
switch(ch){
case 1: cout<<"enter the element ";

cin>>ele;
myq.enQueue(ele);
break;
case 2: myq.deQueue();
break;
case 3: myq.displayQueue();
break;
case 4: exit(0);

}
}
return 0;
}
```

```
#include <iostream>
using namespace std;
class QNode
{
public:
int data;
QNode* next;
QNode(int d)
{
data = d;
next = NULL;
}
};
class Queue
{
public:
QNode *front, *rear;
Queue()
{
front = rear = NULL;
}
void enQueue(int x)
{
// Create a new LL node
QNode* temp = new QNode(x);
// If queue is empty, then
// new node is front and rear both
if (rear == NULL) {
front = rear = temp;
return;
}
// Add the new node at
// the end of queue and change rear
rear->next = temp;
rear = temp;
}
// Function to remove
// a key fromgiven queue q
void deQueue()
{
54
// If queue is empty, return NULL.
if (front == NULL)
return;
// Store previous front and
// move front one node ahead
QNode* temp = front;
front = front->next;
// If front becomes NULL, then
// change rear also as NULL
if (front == NULL)
rear = NULL;
delete (temp);
}
void Display() {
QNode* temp = front;
if ((front == NULL) && (rear == NULL))
{
cout<<"Queue is empty"<<endl;
return;
}
cout<<"Queue elements are: ";
while (temp != NULL) {
cout<<temp->data<<" ";
temp = temp->next;
}
cout<<endl;
}
};
// Driven Program
int main()
{
Queue myq;
int ch,ele;
while(1){
cout<<"\n1.Enqueue(insertion) 2.Dequeue(deletion) 3.display 4. Exit
\nEnter ur choice ";

cin>>ch;
switch(ch)
{
case 1: cout<<"enter the element ";
cin>>ele;
myq.enQueue(ele);
break;
case 2: myq.deQueue();
break;
case 3: myq.Display();
break;
55
case 4: exit(0);
}
}
return 0;
}
```

```
#include <iostream>
using namespace std;
int cqueue[5];
int front = -1, rear = -1, n=5;
void insertCQ(int val) {
if ((front == 0 && rear == n-1) || (front == rear+1))
{
cout<<"Queue Overflow \n";
return;
}
if (front == -1) {
front = 0;
rear = 0;
} else {
if (rear == n - 1)
rear = 0;
else
rear = rear + 1;
}
cqueue[rear] = val ;
}
void deleteCQ() {
if(front == -1) {
cout<<"Queue Underflow\n";
return ;
}
cout<<"Element deleted fromqueue is : "<<cqueue[front]<<endl;
if (front == rear) {
front = -1;
rear = -1;
} else {
if (front == n - 1)
front = 0;
else
front = front + 1;
}
}
void displayCQ() {
int f = front, r = rear;
if (front == -1) {
cout<<"Queue is empty"<<endl;
return;
}
cout<<"Queue elements are :\n";

if (f <= r) {
while (f <= r){
cout<<cqueue[f]<<" ";
f++;
}
} else {
while (f <= n - 1) {
cout<<cqueue[f]<<" ";
f++;
}
f = 0;
while (f <= r) {
cout<<cqueue[f]<<" ";
f++;
}
}
cout<<endl;
}
int main() {
int ch, val;
cout<<"1)Insert\n";
cout<<"2)Delete\n";
cout<<"3)Display\n";
cout<<"4)Exit\n";
do {
cout<<"Enter choice : "<<endl;
cin>>ch;
switch(ch) {
case 1:
cout<<"Input for insertion: "<<endl;
cin>>val;
insertCQ(val);
break;
case 2:
deleteCQ();
break;
case 3:
displayCQ();
break;
case 4:
cout<<"Exit\n";
break;
default: cout<<"Incorrect!\n";
}
} while(ch != 4);
return 0;

}
```

```
#include<iostream>
#define SIZE 100
using namespace std;
class node
{
public:
node()
{
next = NULL;
}
int data;
node *next;
}*front=NULL,*rear=NULL,*n,*temp,*temp1;
class cqueue
{
public:
void insertion();
void deletion();
void display();
};
int main()
{
cqueue cqobj;
int ch;
do
{
cout<<"\n\n\tMain Menu";
cout<<"\n##########################";
cout<<"\n1. Insert\n2. Delete\n3. Display\n4. Exit\n\nEnter Your Choice:
";
cin>>ch;
switch(ch)
{
case 1:
cqobj.insertion();
cqobj.display();
break;
case 2:
cqobj.deletion();
break;
case 3:
cqobj.display();

break;
case 4:
break;
default:
cout<<"\n\nWrong Choice!!! Try Again.";
}
}while(ch!=4);
return 0;
}
void cqueue::insertion()
{
n=new node[sizeof(node)];
cout<<"\nEnter the Element: ";
cin>>n->data;
if(front==NULL)
{
front=n;
}
else
{
rear->next=n;
}
rear=n;
rear->next=front;
}
void cqueue::deletion()
{
int x;
temp=front;
if(front==NULL)
{
cout<<"\nCircular Queue Empty!!!";
}
else
{
if(front==rear)
{
x=front->data;
delete(temp);
front=NULL;
rear=NULL;
}
else
{
x=temp->data;
front=front->next;
rear->next=front;
delete(temp);

}
cout<<"\nElement "<<x<<" is Deleted";
display();
}
}
void cqueue::display()
{
temp=front;
temp1=NULL;
if(front==NULL)
{
cout<<"\n\nCircular Queue Empty!!!";
}
else
{
cout<<"\n\nCircular Queue Elements are:\n\n";
while(temp!=temp1)
{
cout<<temp->data<<" ";
temp=temp->next;
temp1=front;
}
}
}
```

```
#include <bits/stdc++.h>
using namespace std;
class Queue {
public:
stack<int> s1, s2;
// Enqueue an item to the queue
void enQueue(int x)
{
// Push item into the first stack
s1.push(x);
}
// Dequeue an item from the queue
void deQueue()
{
// if both stacks are empty
if (s1.empty() && s2.empty()) {
cout << "Q is empty";
exit(0);
}
// ifs2 is empty, move
// elements from s1
if (s2.empty()) {
while (!s1.empty()) {
s2.push(s1.top());
s1.pop();
}
}
// return the top item from s2
int x = s2.top();
s2.pop();
cout<< x<<endl;
}
};
// Driver code
int main()
{
Queue myq;
int ch,ele,x;
while(1){
cout<<"\n1.Enqueue(insertion) 2.Dequeue(deletion) 3. Exit \nEnter ur
choice ";
cin>>ch;
switch(ch){
case 1: cout<<"enter the element ";
cin>>ele;

myq.enQueue(ele);
break;
case 2: myq.deQueue();
break;
/*case 3: myq.displayQueue();
break;*/
case 3: exit(0);
}
}
return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
// Function to check if given queue element
// can be sorted into another queue using a
// stack.
bool checkSorted(int n, queue<int>& q)
{
stack<int> st;
int expected = 1;
int fnt;
// while given Queue is not empty.
while (!q.empty()) {
fnt = q.front();
q.pop();
// if front element isthe expected element
if (fnt == expected)
expected++;
else {
// ifstack is empty, push the element
if (st.empty()) {
st.push(fnt);
}
// iftop element is less than element which
// need to be pushed, then return false.
else if (!st.empty() && st.top() < fnt) {
return false;
}
// else push into the stack.
else
st.push(fnt);
}
// while expected element are coming from
// stack, pop them out.
while (!st.empty() && st.top() == expected) {
st.pop();
expected++;
}
}
// ifthe final expected element value is equal
// to initial Queue size and the stack is empty.
68
if (expected - 1 == n && st.empty())
return true;
return false;
}
// Driven Program
int main()
{
queue<int> q;
char ans;
int val;
do
{
cout<<"Enter value to be pushed:"<<endl;
cin>>val;
q.push(val);
cout<<"Do you want to add another element : press[y for yes and n for
no]:";
cin>>ans;
} while(ans=='y');
int n = q.size();
(checkSorted(n, q) ? (cout << "Yes") :(cout << "No"));
return 0;
}
```

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b)
{ //swap the content of a and b
int temp;
temp = a;
a = b;
b = temp;
}
void display(int *array, int size)
{
for(int i = 0; i<size; i++)
cout << array[i] << " ";
cout << endl;
}
void merge(int *array, int l, int m, int r)
{
int i, j, k, nl, nr;
//size of left and right sub-arrays
nl = m-l+1; nr = r-m;
int larr[nl], rarr[nr];
//fill left and right sub-arrays
for(i = 0; i<nl; i++)
larr[i] = array[l+i];
for(j = 0; j<nr; j++)
rarr[j] = array[m+1+j];

i = 0; j = 0; k = l;
//merge temp arrays to real array
while(i < nl && j<nr)
{
if(larr[i] <= rarr[j])
{
array[k] = larr[i];
i++;
}
else{
array[k] = rarr[j];
j++;
}

k++;
}
while(i<nl)
{ //extra element in left array
array[k] = larr[i];
i++; k++;
}
while(j<nr)
{ //extra element in right array
array[k] = rarr[j];
j++; k++;
}
}
void mergeSort(int *array, int l, int r)
{
int m;
if(l< r)
{
int m= l+(r-l)/2;
// Sort first and second arrays
mergeSort(array, l, m);
mergeSort(array, m+1, r);
merge(array, l, m, r);
}
}
int main()
{

int n;
cout << "Enter the number of elements: ";
cin >> n;
int arr[n]; //create an array with given number of elements
cout << "Enter elements:" << endl;

for(int i = 0; i<n; i++) {

cin >> arr[i];
}
cout << "Array before Sorting: ";
display(arr, n);
mergeSort(arr, 0, n-1); //(n-1) for last index
cout << "Arrayafter Sorting: ";
display(arr, n);
}
```

```
#include <iostream>
#include<conio.h>
#include<stdlib.h>
#define MAX_SIZE 5
using namespace std;
void quick_sort(int, int);
int arr[MAX_SIZE];
int main()
{
int i;
cout << "\nEnter " << MAX_SIZE << " Elements for Sorting : " << endl;
for (i = 0; i < MAX_SIZE; i++)
cin >> arr[i];
cout << "\nElements before sorting :";
for (i = 0; i < MAX_SIZE; i++) {
cout << "\t" << arr[i];
}
quick_sort(0, MAX_SIZE - 1);
cout << "\n\n ";
cout << "\nElements After sorting :";
for (i = 0; i < MAX_SIZE; i++)
{
cout << "\t" << arr[i];
}
getch();
}
void quick_sort(int f, int l)
{
int i, j, t, p = 0;
if (f < l) {
p = f;

i = f;
j = l;
while (i < j)
{
while (arr[i] <= arr[p] && i < l)
i++;
while (arr[j] > arr[p])
j--;
if (i < j) // internalswap
{
t = arr[i];
arr[i] = arr[j];
arr[j] = t;
}
}
t = arr[p]; // swap with pivot element to place in sorted form
arr[p] = arr[j];
arr[j] = t;
quick_sort(f, j - 1);
quick_sort(j + 1, l);
}
}
```

```
#include <iostream>
using namespace std;
// Get maximum value from array.
int getMax(int arr[], int n)
{
int max = arr[0];
for (int i = 1; i < n; i++)
if (arr[i] > max)
max = arr[i];
return max;
}
// Count sort of arr[].
void countSort(int arr[], int n, int exp)
{
/*Count[i] array will be counting the number of array values having that
'i'
digit at their (exp)th place.*/
int output[n], i, count[10] = {0};
/* Count the number of times each digit occurred at (exp)th place in
every input.*/
for (i = 0; i < n; i++)
count[(arr[i] / exp) % 10]++;
// Calculating their cumulative count.
for (i = 1; i < 10; i++)
count[i] += count[i-1];
// Inserting values according to the digit '(arr[i] / exp) % 10' fetched
into
count[(arr[i] / exp) % 10];
for (i = n - 1; i >= 0; i--)
{
output[count[(arr[i] / exp) % 10] - 1] = arr[i];
count[(arr[i] / exp) % 10]--;
}
// Assigning the result to the arr pointer of main().
for (i = 0; i < n; i++)

arr[i] = output[i];
}
// Sort arr[] of size n using Radix Sort.
void radixsort(int arr[], int n)
{
int exp, m;
m = getMax(arr, n);
// Calling countSort() for digit at (exp)th place in every input.
for (exp = 1; m/exp > 0; exp *= 10)
countSort(arr, n, exp);
}
int main()
{
int n, i;
cout<<"\nEnter the number of data element to be sorted: ";
cin>>n;
int arr[n];
for(i = 0; i < n; i++)
{
cout<<"Enter element "<<i+1<<": ";
cin>>arr[i];
}
radixsort(arr, n);
// Printing the sorted data.
cout<<"\nSorted Data ";
for (i = 0; i < n; i++)
cout<<"->"<<arr[i];
return 0;
}
```

```
#include <iostream>
using namespace std;
void linearSearch(int a[], int n)
{
int temp = -1;
for (int i= 0; i < 5; i++)
{
if(a[i] == n)
{
cout << "Element found at position: " << i+ 1 << endl;
temp = 0;
break;
}
}
if (temp == -1) {
cout << "No Element Found" << endl;
}
}
int main()
{
int arr[5];
cout << "Please enter 5 elements of the Array" << endl;
for (int i = 0; i < 5; i++)
{
cin >> arr[i];
}
cout << "Please enter an element to search" << endl;
int num;
cin >> num;
linearSearch(arr, num);
return 0;
}
```

```
#include <iostream>
using namespace std;
int binarySearch(int arr[], int left, int right, int x)
{
while (left <= right)
{
int mid = left + (right - left) / 2;
if(arr[mid] == x)
{
return mid;
}
else if (arr[mid] < x)
{
left = mid + 1;
}
else
{
right = mid - 1;
}
}
return -1;
}
int main()
{
int myarr[10];
int num;
int output;
cout << "Please enter 5 elements ASCENDING order" << endl;
for (int i = 0; i < 5; i++)
{
cin >> myarr[i];
}
cout << "Please enter an element to search" << endl;
cin >> num;
output = binarySearch(myarr, 0, 9, num);
if (output == -1)
{
cout << "No Match Found" << endl;
}
else {

cout << "Match found at position: " << output + 1<< endl;
}
return 0;
}
```

```
/*
* C++ Programto Implement Hash Tables with Linear Probing
*/
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
const int TABLE_SIZE = 5;
/*
* HashNode Class Declaration
*/
class HashNode
{
public:
int key;
int value;
HashNode(int key, int value)
{
this->key = key;
this->value = value;
}
};
/*
* DeletedNode Class Declaration
*/
class DeletedNode:public HashNode
{

private:
static DeletedNode *entry;
DeletedNode():HashNode(-1, -1)
{}
public:
static DeletedNode *getNode()
{
if (entry == NULL)
entry = new DeletedNode();
return entry;
}
};
DeletedNode *DeletedNode::entry = NULL;
/*
* HashMap Class Declaration
*/
class HashMap
{
private:
HashNode **htable;
public:
HashMap()
{
htable = new HashNode* [TABLE_SIZE];
for (int i = 0; i < TABLE_SIZE; i++)
{
htable[i] = NULL;
}
}

~HashMap()
{
```

```
for (int i = 0; i< TABLE_SIZE; i++)
{
if (htable[i] != NULL && htable[i] != DeletedNode::getNode())
delete htable[i];
}
delete[] htable;
}
/*
* Hash Function
*/
int HashFunc(int key)
{
return key% TABLE_SIZE;
}
/*
* Insert Element at a key
*/
void Insert(int key, int value)
{
int hash_val = HashFunc(key);
int init = -1;
int deletedindex = -1;
while (hash_val != init && (htable[hash_val]
== DeletedNode::getNode() || htable[hash_val]
!= NULL && htable[hash_val]->key != key))
{
if(init == -1)
init = hash_val;

if (htable[hash_val] == DeletedNode::getNode())
deletedindex = hash_val;
hash_val = HashFunc(hash_val + 1);
}
if (htable[hash_val] == NULL || hash_val == init)
{
if(deletedindex != -1)
htable[deletedindex] = new HashNode(key, value);
else
htable[hash_val] = new HashNode(key, value);
}
if(init != hash_val)
{
if(htable[hash_val] != DeletedNode::getNode())
{
if(htable[hash_val] != NULL)
{
if (htable[hash_val]->key == key)
htable[hash_val]->value = value;
}
}
else
htable[hash_val] = new HashNode(key, value);
}
}
/*
* Search Element at a key
*/
int Search(int key)
{
```

```
int hash_val = HashFunc(key);
int init = -1;
while (hash_val != init && (htable[hash_val]
== DeletedNode::getNode() || htable[hash_val]
!= NULL && htable[hash_val]->key != key))
{
if(init == -1)
init = hash_val;
hash_val = HashFunc(hash_val + 1);
}
if (htable[hash_val] == NULL || hash_val == init)
return -1;
else
return htable[hash_val]->value;
}
/*
* Remove Element at a key
*/
void Remove(int key)
{
int hash_val = HashFunc(key);
int init = -1;
while (hash_val != init && (htable[hash_val]
== DeletedNode::getNode() || htable[hash_val]
!= NULL && htable[hash_val]->key != key))
{
if(init == -1)
init = hash_val;
hash_val = HashFunc(hash_val + 1);
}

if(hash_val != init && htable[hash_val] != NULL)
{
delete htable[hash_val];
htable[hash_val] = DeletedNode::getNode();
}
}
};
/*
* Main Contains Menu
*/
int main()
{
HashMap hash;
int key, value;
int choice;
while(1)
{
cout<<"\n --------------------- "<<endl;
cout<<"Operations on Hash Table"<<endl;
cout<<"\n --------------------- "<<endl;
cout<<"1.Insert element into the table"<<endl;
cout<<"2.Search element from the key"<<endl;
cout<<"3.Delete element at a key"<<endl;
cout<<"4.Exit"<<endl;
cout<<"Enter your choice: ";
cin>>choice;
switch(choice)
{
case 1:
```

```
cout<<"Enter element to be inserted: ";
cin>>value;
cout<<"Enter keyat which element to be inserted: ";
cin>>key;
hash.Insert(key, value);
break;
case 2:
cout<<"Enter key of the element to be searched: ";
cin>>key;
if(hash.Search(key) == -1)
{
cout<<"No element found at key "<<key<<endl;
continue;
}
else
{
cout<<"Element at key "<<key<<" : ";
cout<<hash.Search(key)<<endl;
}
break;
case 3:
cout<<"Enter key of the element to be deleted: ";
cin>>key;
hash.Remove(key);
break;
case 4:
exit(1);
default:
cout<<"\nEnter correct option\n";
}

}
return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
class Bt
{
public:
char tree[10]={'\0'};
int root(char key)
{
tree[0] = key;
return 0;
}
int set_left(char key, int parent)
{
if (tree[parent] == '\0')
cout << "Can't set child at "<< (parent * 2) + 1 <<
" , no parent found at "<< parent <<endl;
else
tree[(parent * 2) + 1] = key;
return 0;
}

int set_right(char key, int parent)
{
if (tree[parent] == '\0')
cout << "Can't set child at "<< (parent * 2) + 2 << " , no parent found
at "<< parent <<endl;
else
tree[(parent * 2) + 2] = key;
return 0;
}

int print_tree()
{
int size=sizeof(tree)/sizeof(tree[0]);
int i;
cout << "Index : ";
for (i = 0; i<size-1;i++)
{
cout << i <<" ";
}
cout<<endl<<"Value : ";
for (i = 0; i<size-1;i++)
{
if(tree[i]== '\0')
cout<<"- ";
else
cout << tree[i]<<" ";
}

return 0;
}
};
// Driver Code
int main()
{
Bt bt;
bt.root('A');
bt.set_left('B',0);
bt.set_right('C',0);
bt.set_left('D',1);
bt.set_right('E',2);
bt.set_right('F',3);
bt.print_tree();
return 0;
}
```

```
#include <iostream>
using namespace std;
// Data structure to store a binary tree node
class Node
{
public:
int key;
Node *left, *right;
};
Node *root=NULL,*temp;
// Recursive function to delete a given binary tree
void insertElements()
{
Node *nc,*pNode;
int v;
cout<<"\n enter the value ";
cin>>v;
temp=new(Node);
temp->key=v;
temp->left=NULL;
temp->right=NULL;
if(root==NULL)
{
root=temp;
}
else
{
nc=root;
while(nc!=NULL)
{
pNode=nc;
if(v<nc->key)
nc=nc->left;
else
nc=nc->right;
}
if(v<pNode->key)
{
pNode->left=temp;
}
else
pNode->right=temp;
}
}
void deleteBinaryTree(Node* &root)
{
// Base case: empty tree
if (root == NULL) {
return;
}

// delete left and right subtree first (Postorder)
deleteBinaryTree(root->left);
deleteBinaryTree(root->right);
// delete the current node after deleting its left and right subtree
delete root;
// set root as null before returning
root = NULL;
}
void display(Node *temp)
```

```
{
if(temp==NULL)
{
}
else
{
return;
cout<<" "<<temp->key;
display(temp->left);
display(temp->right);
}
}
int main()
{
int ch;
while(1)
{
cout<<"\n 1.insert Elements \n 2.display \n 3.delete \n 4.exit";
cout<<"\n enter ur choice ";
cin>>ch;
switch(ch)
{
case 1:insertElements();
break;
case 2:display(root);
break;
case 3:deleteBinaryTree(root);
if (root == NULL)
{
cout << "Tree Successfully Deleted";
}
break;
case 4:exit(1);
break;
default:cout<<"invalid operation";
}
}
return 0;
}
```

```
#include<iostream>
#include<stdlib.h>
using namespace std;
class st
{
public:
int data;
st *left;st*right;
};

st *root=NULL,*temp;
void insertElements();
void preorder(st *);
void inorder(st *);
void postorder(st *);
int main()
{
int ch;
while(1)
{
cout<<"\n 1.insert Elements \n 2.preorder \n 3.inorder \n 4.postorder
\n5.exit";
cout<<"\n enter ur choice ";
cin>>ch;
switch(ch)
{
case 1:insertElements();break;
case 2:preorder(root);break;
case 3:inorder(root);break;
case 4:postorder(root);break;
case 5:exit(1);break;
default:cout<<"invalid operation";
}
}
}
void insertElements()
{
st *nc,*pNode;
int v;
cout<<"\n enter the value ";
cin>>v;
temp=new(st);
temp->data=v;
temp->left=NULL;
temp->right=NULL;
if(root==NULL)
{

root=temp;
}
else
{
nc=root;
while(nc!=NULL)
{
pNode=nc;
if(v<nc->data)
nc=nc->left;
else
nc=nc->right;
}
if(v<pNode->data)
{
pNode->left=temp;
}
else
pNode->right=temp;
}
}
void preorder(st *temp)
{
if(temp!=NULL)
{
cout<<" "<<temp->data;
preorder(temp->left);
preorder(temp->right);
}
}
void inorder(st *temp)
{
if(temp!=NULL)
{
inorder(temp->left);
cout<<" "<<temp->data;
inorder(temp->right);
}
}
void postorder(st *temp)
{
if(temp!=NULL)
{
postorder(temp->left);
postorder(temp->right);
cout<<" "<<temp->data;
}
}
```

```
// Recursive CPP program for level order traversal of BinaryTree
#include <bits/stdc++.h>
using namespace std;
class node
{
public:
int data;
node* left, *right;
};
void printGivenLevel(node* root, int level);
int height(node* node);
node* newNode(int data);
void printLevelOrder(node* root)
{
int h = height(root);
int i;
for (i = 1; i <= h; i++)
printGivenLevel(root, i);
}
void printGivenLevel(node* root, int level)
{
if (root == NULL)
return;
if (level == 1)
cout << root->data << " ";
else if (level > 1)
{
printGivenLevel(root->left, level-1);
printGivenLevel(root->right, level-1);
}
}
int height(node* node)
{
if (node == NULL)
return 0;
else

{
int lheight = height(node->left);
int rheight = height(node->right);
if (lheight > rheight)
return(lheight + 1);
else
return(rheight + 1);
}
}
node* newNode(int data)
{
node* Node = new node();
Node->data = data;
Node->left = NULL;
Node->right = NULL;
return(Node);
}
int main()
{
node *root = newNode(1);
root->left = newNode(2);
root->right = newNode(3);
root->left->left = newNode(4);
root->left->right = newNode(5);
cout << "Level Order traversal of binary tree is";
printLevelOrder(root);
return 0;
}
```

```
//---------To find Second minimum node in a Binarytree.
#include<iostream>
using namespace std;
class Node{
public:
int data;
Node *left,*right;
Node(int d){
data = d;
left = NULL;
right = NULL;
}
void display(){
cout<<data<<" ";
}
};
class BinaryTree{
public:
Node *rptr = NULL;
BinaryTree(int d){
Node *nptr = new Node(d);
rptr = nptr;
}
Node* add_Child(Node *parent,int d,int lor){
Node *cnode = new Node(d);
if(lor == 0){
parent->left = cnode;
}
else{
parent->right = cnode;
}
return cnode;
}
void inorder(Node *node){
if(node == NULL){
return;
}
else{
inorder(node->left);
node->display();
inorder(node->right);
}
}
int FindSecondMin(Node *node)
{

/* if root not equal to NULL then assign root->data to min else -1*/
int min=(node && node->data != 0) ? node->data : -1;
int secondmin =-1; // initialize secondmin to -1
TraverseNodes(node, min, secondmin); // traverse tree to find second
minimum
//value node
return secondmin;
}
void TraverseNodes(Node* node, int min, int& secondmin)
{
if (!node || node->data == 0) // return when reach leaf node or data
equal to zero
{
return;
}
if (node->data > min)
{
if(secondmin == -1 || node->data < secondmin)
{
secondmin = node->data;
}
}
TraverseNodes(node->left, min, secondmin);
TraverseNodes(node->right, min, secondmin);
}
};
int main(){
BinaryTree bt(10);
Node *l = bt.add_Child(bt.rptr,20,0);
Node *r = bt.add_Child(bt.rptr,50,1);
Node *ll = bt.add_Child(l,70,0);
Node *lr = bt.add_Child(l,15,1);
Node *rl = bt.add_Child(r,40,0);
Node *rr = bt.add_Child(r,90,1);
cout << "Inoder traversal\n";
bt.inorder(bt.rptr);
cout<<"\nSecond minimum node value = " <<
bt.FindSecondMin(bt.rptr);
}
```

```
# include <iostream>
# include <cstdlib>
using namespace std;
/*
* Node Declaration
*/
class node
{
public:
int info;
node *left;
node *right;
}*root;
/*
* Class Declaration
*/
class BST
{
public:
void find(int, node **, node **);
void insert(node *, node *);
void del(int);
void case_a(node *,node *);
void case_b(node *,node *);
void case_c(node *,node *);
void preorder(node *);
void inorder(node *);
void postorder(node *);
void display(node *, int);
void search(node* , int );
BST()
{
root = NULL;
}
};
/*
* Main Contains Menu
*/
int main()

{
int choice, num,key;
BST bst;
node *temp;
while (1)
{
cout<<" ---------------- "<<endl;
cout<<"Operations on BST"<<endl;
cout<<" ---------------- "<<endl;
cout<<"1.Insert Element "<<endl;
cout<<"2.Delete Element "<<endl;
cout<<"3.Inorder Traversal"<<endl;
cout<<"4.Preorder Traversal"<<endl;
cout<<"5.Postorder Traversal"<<endl;
cout<<"6.Display"<<endl;
cout<<"7.Search"<<endl;
cout<<"8.Quit"<<endl;
cout<<"Enter your choice : ";
cin>>choice;
switch(choice)
{
case 1:
temp = new node;
cout<<"Enter the number to be inserted : ";
cin>>temp->info;
bst.insert(root, temp);
break;
case 2:
if (root == NULL)
{
cout<<"Tree is empty, nothing to delete"<<endl;
continue;
}
cout<<"Enter the number to be deleted : ";
cin>>num;
bst.del(num);
break;
case 3:
cout<<"Inorder Traversal of BST:"<<endl;
bst.inorder(root);
cout<<endl;
break;
case 4:
cout<<"Preorder Traversal of BST:"<<endl;
bst.preorder(root);
cout<<endl;
break;
case 5:
cout<<"Postorder Traversal of BST:"<<endl;
bst.postorder(root);
cout<<endl;

break;
case 6:
cout<<"Display BST:"<<endl;
bst.display(root,1);
cout<<endl;
break;
case 7:
if (root == NULL)
{
cout<<"Tree is empty, nothing to search"<<endl;
continue;
}
cout<<"Enter the number to be search : ";
cin>>key;
bst.search(root,key);
break;
case 8:
exit(1);
default:
cout<<"Wrong choice"<<endl;
}
}
}
/*
* Find Element in the Tree
*/
void BST::find(int item, node **par, node **loc)
{
```

```
node *ptr, *ptrsave;
if (root == NULL)
{
*loc = NULL;
*par = NULL;
return;
}
if (item== root->info)
{
*loc = root;
*par = NULL;
return;
}
if (item < root->info)
ptr = root->left;
else
ptr = root->right;
ptrsave = root;
while (ptr != NULL)
{
if (item == ptr->info)

{
*loc = ptr;
*par = ptrsave;
return;
}
ptrsave = ptr;
if(item < ptr->info)
ptr = ptr->left;
else
ptr = ptr->right;
}
*loc = NULL;
*par = ptrsave;
}
/*
* Inserting Element into the Tree
*/
void BST::insert(node *tree, node *newnode)
{
if (root == NULL)
{
root = new node;
root->info = newnode->info;
root->left = NULL;
root->right = NULL;
cout<<"Root Node is Added"<<endl;
return;
}
if (tree->info == newnode->info)
{
cout<<"Element already in the tree"<<endl;
return;
}
if(tree->info > newnode->info)
{
if(tree->left != NULL)
{
insert(tree->left, newnode);
}
else
{
tree->left = newnode;
(tree->left)->left = NULL;
(tree->left)->right = NULL;
cout<<"Node Added To Left"<<endl;
return;
}
}
else
{

if(tree->right != NULL)
{
insert(tree->right, newnode);
}
else
{
tree->right = newnode;
(tree->right)->left = NULL;
(tree->right)->right = NULL;
cout<<"Node Added To Right"<<endl;
return;
}
}
}
/*
* Delete Element fromthe tree
*/
void BST::del(int item)
{
node *parent, *location;
if (root == NULL)
{
cout<<"Tree empty"<<endl;
return;
}
find(item, &parent, &location);
if (location == NULL)
{
cout<<"Item not present in tree"<<endl;
return;
}
if (location->left == NULL && location->right == NULL)
case_a(parent, location);
if (location->left != NULL && location->right == NULL)
case_b(parent, location);
if (location->left == NULL && location->right != NULL)
case_b(parent, location);
if (location->left != NULL && location->right != NULL)
case_c(parent, location);
free(location);
}
/*
* Case A
*/
void BST::case_a(node *par, node *loc )
{
if(par == NULL)
```

```
{

root = NULL;
}
else
{
if (loc == par->left)
par->left = NULL;
else
par->right = NULL;
}
}
/*
* Case B
*/
void BST::case_b(node *par, node *loc)
{
node *child;
if(loc->left != NULL)
child = loc->left;
else
child = loc->right;
if(par == NULL)
{
root = child;
}
else
{
if (loc == par->left)
par->left = child;
else
par->right = child;
}
}
/*
* Case C
*/
void BST::case_c(node *par, node *loc)
{
node *ptr, *ptrsave, *suc, *parsuc;
ptrsave = loc;
ptr = loc->right;
while (ptr->left != NULL)
{
ptrsave = ptr;
ptr = ptr->left;
}
suc = ptr;
parsuc = ptrsave;
if(suc->left == NULL && suc->right == NULL)

case_a(parsuc, suc);
else
case_b(parsuc, suc);
if(par == NULL)
{
root = suc;
}
else
{
```

```
if (loc == par->left)
par->left = suc;
else
par->right = suc;
}
suc->left = loc->left;
suc->right = loc->right;
}
void BST::search(node *root, int data) //searching
{
int depth = 0;
node *temp = new node;
temp = root;
while(temp != NULL)
{
depth++;
if(temp->info == data)
{
cout<<"\nData found at depth: "<<depth<<endl;
return;
}
else if(temp->info > data)
temp = temp->left;
else
temp = temp->right;
}
cout<<"\n Data not found"<<endl;
return;
}
/*
* Pre Order Traversal
*/
void BST::preorder(node *ptr)
{
if (root == NULL)
{
cout<<"Tree is empty"<<endl;
return;
}
if (ptr != NULL)
{

cout<<ptr->info<<" ";
preorder(ptr->left);
preorder(ptr->right);
}
}
/*
* In Order Traversal
*/
void BST::inorder(node *ptr)
{
if (root == NULL)
{
cout<<"Tree is empty"<<endl;
return;
}
if (ptr != NULL)
{
inorder(ptr->left);
```

```
cout<<ptr->info<<" ";
inorder(ptr->right);
}
}
/*
* Postorder Traversal
*/
void BST::postorder(node *ptr)
{
if (root == NULL)
{
cout<<"Tree is empty"<<endl;
return;
}
if (ptr != NULL)
{
postorder(ptr->left);
postorder(ptr->right);
cout<<ptr->info<<" ";
}
}
/*
* DisplayTree Structure
*/
void BST::display(node *ptr, int level)
{
int i;
if (ptr != NULL)
{
display(ptr->right, level+1);

cout<<endl;
if (ptr == root)
cout<<"Root->: ";
else
{
for (i = 0;i < level;i++)
cout<<" ";
}
cout<<ptr->info;
display(ptr->left, level+1);
}
}
```

```
//The graph using adjacency matrix i.e. 2D array
#include<iostream>
using namespace std;
int vertArr[20][20]; //the adjacency matrix initially 0
int count = 0;
void displayMatrix(int v) {
int i, j;
for(i = 0; i < v; i++) {
for(j = 0; j < v; j++) {
cout << vertArr[i][j] << " ";
}
cout << endl;
}
}
void add_edge(int u, int v) { //function to add edge into the matrix
vertArr[u][v] = 1;
vertArr[v][u] = 1;
}
int main() {
int v = 6; //there are 6 vertices in the graph
add_edge(0, 4);
add_edge(0, 3);
add_edge(1, 2);
add_edge(1, 4);
add_edge(1, 5);
add_edge(2, 3);
add_edge(2, 5);

add_edge(5, 3);
add_edge(5, 4);
displayMatrix(v);
return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
// A utility function to add an edge in an
// undirected graph.
void addEdge(vector<int> adj[], int u, int v)
{
adj[u].push_back(v);
adj[v].push_back(u);
}
// A utility function to print the adjacency list
// representation of graph
void printGraph(vector<int> adj[], int V)
{
for (int v = 0; v < V; ++v) {
cout <<"\n Adjacency list of "<< v<<" " << "vertex ";
for (auto x : adj[v])
cout << "- " << x;
printf("\n");
}
}
// Driver code
int main()
{
int V = 5;
vector<int> adj[V];
addEdge(adj, 0, 1);
addEdge(adj, 0, 4);
addEdge(adj, 1, 2);
addEdge(adj, 1, 3);
addEdge(adj, 1, 4);
addEdge(adj, 2, 3);
addEdge(adj, 3, 4);
printGraph(adj, V);
return 0;
}
```

```
// C++ program to print DFS traversal for a given given graph
#include <bits/stdc++.h>
using namespace std;
class Graph {
// A function used by DFS
void DFSUtil(int v);
public:
map<int, bool> visited;
map<int, list<int> > adj;
// function to add an edge to graph
void addEdge(int v, int w);
// prints DFS traversal of the complete graph
void DFS();
};

void Graph::addEdge(int v, int w)
{
adj[v].push_back(w); // Add w to v’s list.
}
void Graph::DFSUtil(int v)
{
// Mark the current node as visited and print it
visited[v] = true;
cout << v << " ";
// Recur for all the vertices adjacent to this vertex
list<int>::iterator i;
for (i = adj[v].begin(); i!= adj[v].end(); ++i)
if (!visited[*i])
DFSUtil(*i);
}
// The function to do DFS traversal. It uses recursive
// DFSUtil()
void Graph::DFS()
{
// Call the recursive helper function to print DFS
// traversal starting from all vertices one by one
for (auto i : adj)
if (visited[i.first] == false)
DFSUtil(i.first);
}
// Driver Code
3333333333333333333333
int main()
{
// Create a graph given in the above diagram
Graph g;
g.addEdge(0, 1);
g.addEdge(0, 9);
g.addEdge(1, 2);
g.addEdge(2, 0);
g.addEdge(2, 3);
g.addEdge(9, 3);
cout << "Following is Depth First Traversal \n";
g.DFS();
return 0;
}
```

```
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include<iostream>
#include <list>
using namespace std;
// This class represents a directed graph using
// adjacency list representation
class Graph
{
int V; // No. of vertices
// Pointer to an array containing adjacency lists
list<int> *adj;
public:
Graph(int V); // Constructor
// function to add an edge to graph
void addEdge(int v, int w);
// prints BFS traversal from a given source s
void BFS(int s);
};
Graph::Graph(int V)
{
this->V = V;
adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
adj[v].push_back(w); // Add w to v’s list.
}
void Graph::BFS(int s)
{
// Mark all the vertices as not visited
bool *visited = new bool[V];
for(int i = 0; i < V; i++)
visited[i] = false;
// Create a queue for BFS
list<int> queue;

// Mark the current node as visited and enqueue it
visited[s] = true;
queue.push_back(s);
// 'i' will be used to get all adjacent
// vertices of a vertex
list<int>::iterator i;
while(!queue.empty())
{
// Dequeue a vertex from queue and print it
s = queue.front();
cout << s << " ";
queue.pop_front();
// Get all adjacent vertices ofthe dequeued
// vertex s. If a adjacent has not been visited,
// then mark it visited and enqueue it
for (i = adj[s].begin(); i != adj[s].end(); ++i)
{
if (!visited[*i])
{
visited[*i] = true;
queue.push_back(*i);
}
}
}
}
// Driver program to test methods of graph class
int main()
{
// Create a graph given in the above diagram
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) \n";
g.BFS(2);
return 0;
}
```