1. Write programs to implement the following using an array:
a) Stack ADT

```
// Stack ADT
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val) {
  if(top>=n-1)
  cout<<"Stack Overflow"<<endl;
  else {
    top++;
    stack[top]=val;
  }
}
void pop() {
  if(top<=-1)
  cout<<"Stack Underflow"<<endl;
  else {
    cout<<"The popped element is "<< stack[top] <<endl;
    top--;
  }
}
void display() {
  if(top>=0) {
    cout<<"Stack elements are:";
    for(int i=top; i>=0; i--)
    cout<<stack[i]<<" ";
    cout<<endl;
  } else
  cout<<"Stack is empty";
}
int main() {
  int ch, val;
  cout<<"1) Push in stack"<<endl;
  cout<<"2) Pop from stack"<<endl;
  cout<<"3) Display stack"<<endl;
  cout<<"4) Exit"<<endl;
```

```
  do {
    cout<<"Enter choice: "<<endl;
    cin>>ch;
    switch(ch) {
      case 1: {
        cout<<"Enter value to be pushed:"<<endl;
        cin>>val;
        push(val);
        break;
      }
      case 2: {
        pop();
        break;
      }
      case 3: {
        display();
        break;
      }
      case 4: {
        cout<<"Exit"<<endl;
        break;
      }
      default: {
        cout<<"Invalid Choice"<<endl;
      }
    }
  }while(ch!=4);
  return 0;
}
```

Output
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit
Enter choice: 1
Enter value to be pushed: 2
Enter choice: 1

Enter value to be pushed: 6
Enter choice: 1
Enter value to be pushed: 8
Enter choice: 1
Enter value to be pushed: 7
Enter choice: 2
The popped element is 7
Enter choice: 3
Stack elements are:8 6 2
Enter choice: 5
Invalid Choice
Enter choice: 4
Exit

1. Write programs to implement the following using an array:
b) Queue ADT.

```
//Queue ADT
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
  int val;
  if (rear == n - 1)
  cout<<"Queue Overflow"<<endl;
  else {
    if (front == - 1)
    front = 0;
    cout<<"Insert the element in queue : "<<endl;
    cin>>val;
    rear++;
    queue[rear] = val;
  }
}
void Delete() {
  if (front == - 1 || front > rear) {
    cout<<"Queue Underflow ";
    return ;
  } else {
    cout<<"Element deleted from queue is : "<< queue[front] <<endl;
    front++;;
  }
}
void Display() {
  if (front == - 1)
  cout<<"Queue is empty"<<endl;
  else {
    cout<<"Queue elements are : ";
    for (int i = front; i <= rear; i++)
    cout<<queue[i]<<" ";
      cout<<endl;
  }   }
```

```
int main() {
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter your choice : "<<endl;
    cin>>ch;
    switch (ch) {
      case 1: Insert();
      break;
      case 2: Delete();
      break;
      case 3: Display();
      break;
      case 4: cout<<"Exit"<<endl;
      break;
      default: cout<<"Invalid choice"<<endl;
    }
  } while(ch!=4);
  return 0;
}
```

OUTPUT:
The output of the above program is as follows
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice : 1
Insert the element in queue : 4
Enter your choice : 1
Insert the element in queue : 3
Enter your choice : 1
Insert the element in queue : 5
Enter your choice : 2
Element deleted from queue is : 4
Enter your choice : 3
Queue elements are : 3 5
Enter your choice : 7
Invalid choice
Enter your choice : 4
Exit

2. Write a program to convert the given infix expression to postfix expression using stack.

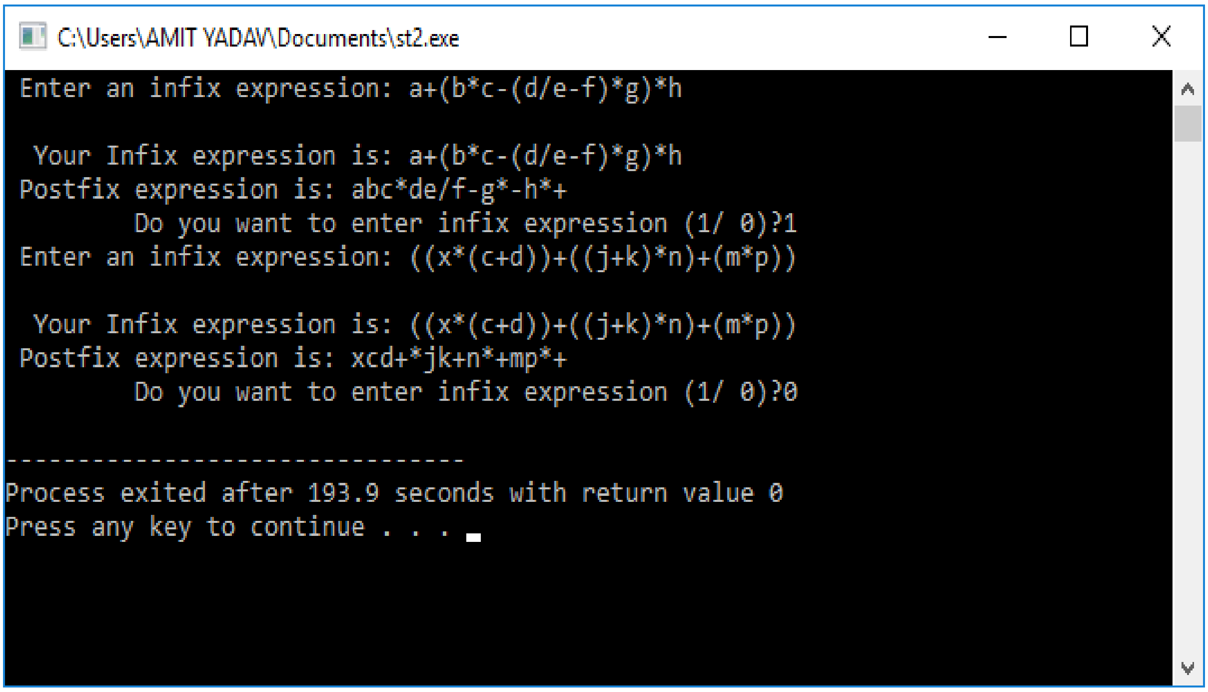
```
// infix expression to postfix expression using stack.
#include<iostream>
#include<stack>
using namespace std;
// defines the Boolean function for operator, operand, equalOrhigher
precedence and the string conversion function.
bool IsOperator(char);
bool IsOperand(char);
bool eqlOrhigher(char, char);
string convert(string);

int main()
{
string infix_expression, postfix_expression;
int ch;
do
{
cout << " Enter an infix expression: ";
cin >> infix_expression;
 postfix_expression = convert(infix_expression);
 cout << "\n Your Infix expression is: " << infix_expression;
cout << "\n Postfix expression is: " << postfix_expression;
cout << "\n \t Do you want to enter infix expression (1/ 0)?";
cin >> ch;
//cin.ignore();
} while(ch == 1);
return 0;
}

// define the IsOperator() function to validate whether any symbol is
operator.
/* If the symbol is operator, it returns true, otherwise false. */
bool IsOperator(char c)
{
if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^' )
```

```
return true;
return false;
}

// IsOperand() function is used to validate whether the character is
operand.
bool IsOperand(char c)
{
if( c >= 'A' && c <= 'Z')  /* Define the character in between A to Z. If not, it
returns False.*/
return true;
if (c >= 'a' && c <= 'z')  // Define the character in between a to z. If not, it
returns False. */
return true;
if(c >= '0' && c <= '9')   // Define the character in between 0 to 9. If not, it
returns False. */
return true;
return false;
}
// here, precedence() function is used to define the precedence to the
operator.
int precedence(char op)
{
if(op == '+' || op == '-')                   /* it defines the lowest precedence */
return 1;
if (op == '*' || op == '/')
return 2;
if(op == '^')                                /* exponent operator has the highest
precedence *
return 3;
return 0;
}
/* The eqlOrhigher() function is used to check the higher or equal
precedence of the two operators in infix expression. */
bool eqlOrhigher (char op1, char op2)
{
int p1 = precedence(op1);
```

```
int p2 = precedence(op2);
if (p1 == p2)
{
if (op1 == '^' )
return false;
return true;
}
return (p1>p2 ? true : false);
}

/* string convert() function is used to convert the infix expression to the
postfix expression of the Stack */
string convert(string infix)
{
stack <char> S;
string postfix ="";
char ch;

S.push( '(' );
infix += ')';

for(int i = 0; i<infix.length(); i++)
{
ch = infix[i];

if(ch == ' ')
continue;
else if(ch == '(')
S.push(ch);
else if(IsOperand(ch))
postfix += ch;
else if(IsOperator(ch))
{
while(!S.empty() && eqlOrhigher(S.top(), ch))
{
postfix += S.top();
S.pop();
```

```
}
S.push(ch);
}
else if(ch == ')')
{
while(!S.empty() && S.top() != '(')
{
postfix += S.top();
S.pop();
}
S.pop();
}
}
return postfix;
}
```

OUTPUT:



3. Write a program to evaluate a postfix expression using stack.

```
// a program to evaluate a postfix expression using stack
#include<iostream>
#include<cmath>
#include<stack>
using namespace std;
float scanNum(char ch) {
  int value;
  value = ch;
  return float(value-'0');   //return float from character
}
int isOperator(char ch) {
  if(ch == '+'|| ch == '-'|| ch == '*'|| ch == '/' || ch == '^')
    return 1;   //character is an operator
  return -1;  //not an operator
}
int isOperand(char ch) {
  if(ch >= '0' && ch <= '9')
    return 1;  //character is an operand
  return -1;  //not an operand
}
float operation(int a, int b, char op) {
  //Perform operation
  if(op == '+')
    return b+a;
  else if(op == '-')
    return b-a;
  else if(op == '*')
    return b*a;
  else if(op == '/')
    return b/a;
  else if(op == '^')
    return pow(b,a); //find b^a
  else
    return INT_MIN; //return negative infinity
}
float postfixEval(string postfix) {
```

```
  int a, b;
  stack<float> stk;
  string::iterator it;
  for(it=postfix.begin(); it!=postfix.end(); it++) {
    //read elements and perform postfix evaluation
    if(isOperator(*it) != -1) {
      a = stk.top();
      stk.pop();
      b = stk.top();
      stk.pop();
      stk.push(operation(a, b, *it));
    }else if(isOperand(*it) > 0) {
      stk.push(scanNum(*it));
    }
  }
  return stk.top();
}
main() {
  string post = "53+62/*35*+";
  cout << "The result is: "<<postfixEval(post);
}
```

OUTPUT:

The result is: 39

4. Write a program to ensure the parentheses are nested correctly in an arithmetic expression.

```
#include <iostream>
#include <stack>
using namespace std;
bool isBalancedExp(string exp) {
  stack<char> stk;
  char x;
  for (int i=0; i<exp.length(); i++) {
    if (exp[i]=='('||exp[i]=='['||exp[i]=='{') {
      stk.push(exp[i]);
      continue;
    }
    if (stk.empty())
      return false;
    switch (exp[i]) {
    case ')':
      x = stk.top();
      stk.pop();
      if (x=='{' || x=='[')
        return false;
      break;
    case '}':
      x = stk.top();
      stk.pop();
      if (x=='(' || x=='[')
        return false;
      break;
    case ']':
      x = stk.top();
      stk.pop();
      if (x =='(' || x == '{')
        return false;
      break;
    }
  }
```

```
  return (stk.empty());
}
int main() {
  string expresion = "()[(){()}]";
  if (isBalancedExp(expresion))
    cout << "This is Balanced Expression";
  else
    cout << "This is Not Balanced Expression";
}
```

OUTPUT:

This is Balanced Expression

5. **Write a program to find following using Recursion.**

a) Factorial of +ve Integer

```
// Factorial of positive integer
#include<conio.h>
#include<iostream.h>
void main()
{
clrscr();
int n, i;
unsigned long  f=1;
cout<<"Emter the Number: ";
cin>>n;
if(n<=0)
cout<<"Please Enter valid number";
else
{
for (i=n; i>=1; i--)
{
f=f*i;
}
cout<<"Factorial of "<<n<<" = "<<f;
}
getch();
}
```

OUTPUT:

Enter the Number: <u>5</u>
Factorial of <u>5</u> = <u>120</u>

b) nth term of the Fibonacci Sequence

```
//Fibonacci Series using Recursion
#include<bits/stdc++.h>
using namespace std;

int F(int N)
{
   if (N <= 1)
      {
       return N;
      }

   return F(N-1) + F(N-2);
}

int main ()
{
   int N = 5;
   cout << F(N);
   return 0;
}
```

OUTPUT:

5

c) GCD of two +ve integers

```
//GDC of two positive integers

#include <iostream>
using namespace std;
int gcd(int a, int b) {
  if (b == 0)
  return a;
  return gcd(b, a % b);
}
int main() {
  int a = 105, b = 30;
  cout<<"GCD of "<< a <<" and "<< b <<" is "<< gcd(a, b);
  return 0;
}
```

OUTPUT:

GCD of 105 and 30 is 15

6. Write a program to create a single linked list and write functions to implement the following
operations:
a) Insert an element at a specified position
b) Delete a specified element in the list
c) Search for an element and find its position in the list
d) Sort the elements in the list ascending order

```
// C++ program for the above approach
#include <iostream>
using namespace std;
// Node class to represent a node of the linked list.
class Node {
public:
	int data;
	Node* next;
	// Default constructor
	Node()
	{
		data = 0;
		next = NULL;
	}
	// Parameterised Constructor
	Node(int data)
	{
		this->data = data;
		this->next = NULL;
	}
};
// Linked list class to implement a linked list.
class Linkedlist {
	Node* head;
public:
	// Default constructor
	Linkedlist() { head = NULL; }
	// Function to insert a node at the end of the  linked list.
	void insertNode(int);
	// Function to print the linked list.
```

```
	void printList();
	// Function to delete the node at given position
	void deleteNode(int);
};
// Function to delete the  node at given position
void Linkedlist::deleteNode(int nodeOffset)
{
	Node *temp1 = head, *temp2 = NULL;
	int ListLen = 0;
	if (head == NULL) {
		cout << "List empty." << endl;
		return;
	}
	// Find length of the linked-list.
	while (temp1 != NULL) {
		temp1 = temp1->next;
		ListLen++;
	}
	// Check if the position to be  deleted is greater than the length
	// of the linked list.
	if (ListLen < nodeOffset) {
		cout << "Index out of range"
			<< endl;
		return;
	}
	// Declare temp1
	temp1 = head;
	// Deleting the head.
	if (nodeOffset == 1) {
		// Update head
		head = head->next;
		delete temp1;
		return;
	}
	// Traverse the list to  find the node to be deleted.
	while (nodeOffset-- > 1)
	{
```

```
            // Update temp2
            temp2 = temp1;
            // Update temp1
            temp1 = temp1->next;
        }
        // Change the next pointer  of the previous node.
        temp2->next = temp1->next;
        // Delete the node
        delete temp1;
}
// Function to insert a new node.
void Linkedlist::insertNode(int data)
{
        // Create the new Node.
        Node* newNode = new Node(data);
        // Assign to head
        if (head == NULL) {
            head = newNode;
            return;
        }
        // Traverse till end of list
        Node* temp = head;
        while (temp->next != NULL) {
            // Update temp
            temp = temp->next;
        }
        // Insert at the last.
        temp->next = newNode;
}
// Function to print the nodes of the linked list.
void Linkedlist::printList()
{
        Node* temp = head;
        // Check for empty list.
        if (head == NULL) {
            cout << "List empty" << endl;
```

```
            return;
        }
        // Traverse the list.
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
}
// Driver Code
int main()
{
        Linkedlist list;
        // Inserting nodes
        list.insertNode(1);
        list.insertNode(2);
        list.insertNode(3);
        list.insertNode(4);
        cout << "Elements of the list are: ";
        // Print the list
        list.printList();
        cout << endl;
        // Delete node at position 2.
        list.deleteNode(2);
        cout << "Elements of the list are: ";
        list.printList();
        cout << endl;
        return 0;
}
```

OUTPUT:

Elements of the list are: 1 2 3 4
Elements of the list are: 1 3 4

7. **Write a program to create a double linked list and write functions to implement the following operations:**
a)Insert an element at a specified position
b) Delete a specified element in thelist
c) Search for an element and find its position in the list
d) Sort the elements in the list ascending order

```
#include<iostream>
using namespace std;
class Node {
  public:
    int key;
  int data;
  Node * next;
  Node * previous;

  Node() {
    key = 0;
    data = 0;
    next = NULL;
    previous = NULL;
  }
  Node(int k, int d) {
    key = k;
    data = d;
  }
};
class DoublyLinkedList {
  public:
    Node * head;
  DoublyLinkedList() {
    head = NULL;
  }
  DoublyLinkedList(Node * n) {
    head = n;
  }
  // 1. CHeck if node exists using key value
```

```
Node * nodeExists(int k) {
  Node * temp = NULL;
  Node * ptr = head;
  while (ptr != NULL) {
    if (ptr - > key == k) {
      temp = ptr;
    }
    ptr = ptr - > next;
  }
  return temp;
}
```

// 2. **Append a node to the list**

```
void appendNode(Node * n) {
  if (nodeExists(n - > key) != NULL) {
    cout << "Node Already exists with key value : " << n - > key << ".
Append another node with different Key value" << endl;
  } else {
    if (head == NULL) {
      head = n;
      cout << "Node Appended as Head Node" << endl;
    } else {
      Node * ptr = head;
      while (ptr - > next != NULL) {
        ptr = ptr - > next;
      }
      ptr - > next = n;
      n - > previous = ptr;
      cout << "Node Appended" << endl;
    }
  }
}
```

// 3. **Prepend Node - Attach a node at the start**

```
void prependNode(Node * n) {
  if (nodeExists(n - > key) != NULL) {
    cout << "Node Already exists with key value : " << n - > key << ".
Append another node with different Key value" << endl;
```

```
  } else {
    if (head == NULL) {
      head = n;
      cout << "Node Prepended as Head Node" << endl;
    } else {
      head - > previous = n;
      n - > next = head;
      head = n;
      cout << "Node Prepended" << endl;
    }

  }
}
// 4. Insert a Node after a particular node in the list
void insertNodeAfter(int k, Node * n) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
    cout << "No node exists with key value: " << k << endl;
  } else {
    if (nodeExists(n - > key) != NULL) {
      cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key value" << endl;
    } else {
      Node * nextNode = ptr - > next;
      // inserting at the end
      if (nextNode == NULL) {
        ptr - > next = n;
        n - > previous = ptr;
        cout << "Node Inserted at the END" << endl;
      }
      //inserting in between
      else {
        n - > next = nextNode;
        nextNode - > previous = n;
        n - > previous = ptr;
        ptr - > next = n;
        cout << "Node Inserted in Between" << endl;
```

```
      }
    }
  }
}
// 5. Delete node by unique key. Basically De-Link not delete
void deleteNodeByKey(int k) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
    cout << "No node exists with key value: " << k << endl;
  } else {

    if (head - > key == k) {
      head = head - > next;
      cout << "Node UNLINKED with keys value : " << k << endl;
    } else {
      Node * nextNode = ptr - > next;
      Node * prevNode = ptr - > previous;
      // deleting at the end
      if (nextNode == NULL) {
        prevNode - > next = NULL;
        cout << "Node Deleted at the END" << endl;
      }
      //deleting in between
      else {
        prevNode - > next = nextNode;
        nextNode - > previous = prevNode;
        cout << "Node Deleted in Between" << endl;

      }
    }
  }
}
// 6th update node
void updateNodeByKey(int k, int d) {
  Node * ptr = nodeExists(k);
  if (ptr != NULL) {
```

```
    ptr - > data = d;
    cout << "Node Data Updated Successfully" << endl;
   } else {
    cout << "Node Doesn't exist with key value : " << k << endl;
   }
  }
  // 7th printing
  void printList() {
   if (head == NULL) {
    cout << "No Nodes in Doubly Linked List";
   } else {
    cout << endl << "Doubly Linked List Values : ";
    Node * temp = head;

    while (temp != NULL) {
     cout << "(" << temp - > key << "," << temp - > data << ") <--> ";
     temp = temp - > next;
    }
   }
  }
};
int main() {
  DoublyLinkedList obj;
  int option;
  int key1, k1, data1;
  do {
   cout << "\nWhat operation do you want to perform? Select Option
number. Enter 0 to exit." << endl;
   cout << "1. appendNode()" << endl;
   cout << "2. prependNode()" << endl;
   cout << "3. insertNodeAfter()" << endl;
   cout << "4. deleteNodeByKey()" << endl;
   cout << "5. updateNodeByKey()" << endl;
   cout << "6. print()" << endl;
   cout << "7. Clear Screen" << endl << endl;
   cin >> option;
   Node * n1 = new Node();
```

```
//Node n1;
switch (option) {
case 0:
  break;
case 1:
  cout << "Append Node Operation \nEnter key & data of the Node to be
Appended" << endl;
  cin >> key1;
  cin >> data1;
  n1 - > key = key1;
  n1 - > data = data1;
  obj.appendNode(n1);
  //cout<<n1.key<<" = "<<n1.data<<endl;
  break;
case 2:
  cout << "Prepend Node Operation \nEnter key & data of the Node to
be Prepended" << endl;
  cin >> key1;
  cin >> data1;
  n1 - > key = key1;
  n1 - > data = data1;
  obj.prependNode(n1);
  break;
case 3:
  cout << "Insert Node After Operation \nEnter key of existing Node
after which you want to Insert this New node: " << endl;
  cin >> k1;
  cout << "Enter key & data of the New Node first: " << endl;
  cin >> key1;
  cin >> data1;
  n1 - > key = key1;
  n1 - > data = data1;

  obj.insertNodeAfter(k1, n1);
  break;
case 4:
```

```
    cout << "Delete Node By Key Operation - \nEnter key of the Node to be
deleted: " << endl;
    cin >> k1;
    obj.deleteNodeByKey(k1);
    break;
  case 5:
    cout << "Update Node By Key Operation - \nEnter key & NEW data to
be updated" << endl;
    cin >> key1;
    cin >> data1;
    obj.updateNodeByKey(key1, data1);
    break;
  case 6:
    obj.printList();
    break;
  case 7:
    system("cls");
    break;
  default:
    cout << "Enter Proper Option number " << endl;
  }
 } while (option != 0);
 return 0;
}
```

8. Write programs to implement the following using a single linked list:
a) Stack ADT

```
// C++ program to Implement a stack using singly linked list
#include <bits/stdc++.h>
using namespace std;
// creating a linked list;
class Node {
public:
	int data;
	Node* link;

	// Constructor
	Node(int n)
	{
		this->data = n;
		this->link = NULL;
	}
};
class Stack {
	Node* top;
public:
	Stack() { top = NULL; }
	void push(int data)
	{
		// Create new node temp and allocate memory in heap
		Node* temp = new Node(data);
		//Check if stack (heap) is full, then inserting an element would
		// lead to stack overflow
		if (!temp) {
			cout << "\nStack Overflow";
			exit(1);
		}
		// Initialize data into temp data field
		temp->data = data;
		// Put top pointer reference into temp link
		temp->link = top;
```

```
        // Make temp as top of Stack
        top = temp;
}
// Utility function to check if the stack is empty or not
bool isEmpty()
{
// If top is NULL it means that there are no elements are in stack
        return top == NULL;
}

// Utility function to return top element in a stack
int peek()
{
        // If stack is not empty , return the top element
        if (!isEmpty())
                return top->data;
        else
                exit(1);
}

// Function to remove a key from given queue q
void pop()
{
        Node* temp;

        // Check for stack underflow
        if (top == NULL) {
                cout << "\nStack Underflow" << endl;
                exit(1);
        }
        else {
                // Assign top to temp
                temp = top;
                // Assign second node to top
                top = top->link;
```

```
                /* This will automatically destroy the link between first
                node and second node Release memory of top node  i.e
                delete the node*/
                free(temp);
            }
        }
        // Function to print all the elements of the stack
        void display()
        {
            Node* temp;
            // Check for stack underflow
            if (top == NULL) {
                cout << "\nStack Underflow";
                exit(1);
            }
            else {
                temp = top;
                while (temp != NULL) {
                    // Print node data
                    cout << temp->data;
                    // Assign temp link to temp
                    temp = temp->link;
                    if (temp != NULL)
                        cout << " -> ";
                }
            }
        }
    };

    // Driven Program
    int main()
    {
        // Creating a stack
        Stack s;
        // Push the elements of stack
        s.push(11);
        s.push(22);
```

```
        s.push(33);
        s.push(44);
        // Display stack elements
        s.display();
        // Print top element of stack
        cout << "\nTop element is " << s.peek() << endl;
        // Delete top elements of stack
        s.pop();
        s.pop();
        // Display stack elements
        s.display();
        // Print top element of stack
        cout << "\nTop element is " << s.peek() << endl;
        return 0;
}
```

OUTPUT:

44 -> 33 -> 22 -> 11

Top element is 44

22 -> 11

Top element is 22

8. Write programs to implement the following using a single linked list:
b) Queue ADT

```
#include <bits/stdc++.h>
using namespace std;
struct QNode {
	int data;
	QNode* next;
	QNode(int d)
	{
		data = d;
		next = NULL;
	}
};
struct Queue {
	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 from given queue q
	void deQueue()
	{
		// 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);
	}
};
// Driven Program
int main()
{
	Queue q;
	q.enQueue(10);
	q.enQueue(20);
	q.deQueue();
	q.deQueue();
	q.enQueue(30);
	q.enQueue(40);
	q.enQueue(50);
	q.deQueue();
	cout << "Queue Front : " << (q.front)->data << endl;
	cout << "Queue Rear : " << (q.rear)->data;
}
```

OUTPUT:

Queue Front : 40
Queue Rear : 50

9. Write a program to create singular circular linked lists and function to implement the following operations:
a) Insert an element at a specifiedposition
b) Delete a specified element in thelist
c) Search for an element and find its position in thelist
// C++ program for creating a node for singular circular linked list // C++ program for the above methods

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
	int data;
	struct Node* next;
};
struct Node* addToEmpty(struct Node* last, int data)
{
	// This function is only for empty list
	if (last != NULL)
		return last;
	// Creating a node dynamically.
	struct Node* temp
		= (struct Node*)malloc(sizeof(struct Node));
	// Assigning the data.
	temp->data = data;
	last = temp;
	// Creating the link.
	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 list is empty, return.
        if (last == NULL) {
                cout << "List is empty." << endl;
                return;
        }
        // Pointing to first Node of the list.
        p = last->next;

        // Traversing the list.
        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);

        // Function call
        traverse(last);
        return 0;
}
```

OUTPUT: 2 4 6 8 10 12

10. Write a program to implement Binary search technique using Iterative method and Recursive method

```
// C++ program to implement recursive Binary Search
#include <bits/stdc++.h>
using namespace std;
/* A recursive binary search function. It returns location of x in given array
arr[l..r] is present, otherwise -1*/
int binarySearch(int arr[], int l, int r, int x)
{
	if (r >= l) {
		int mid = l + (r - l) / 2;
		// If the element is present at the middle  itself
		if (arr[mid] == x)
			return mid;
// If element is smaller than mid, then  it can only be present in left
subarray
		if (arr[mid] > x)
			return binarySearch(arr, l, mid - 1, x);
		// Else the element can only be present in right sub array
		return binarySearch(arr, mid + 1, r, x);
	}
	return -1;
}
int main(void)
{
	int arr[] = { 2, 3, 4, 10, 40 };
	int x = 10;
	int n = sizeof(arr) / sizeof(arr[0]);
	int result = binarySearch(arr, 0, n - 1, x);
	(result == -1)
		? cout << "Element is not present in array"
		: cout << "Element is present at index " << result;
	return 0; // We reach here when element is not present in array
}
```

OUTPUT:

Element is present at index 3

11. Write a program for sorting the given list numbers in ascending order using the following technique:
a)Bubble sort

```
#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 bubbleSort(int *array, int size) {
  for(int i = 0; i<size; i++) {
    int swaps = 0;         //flag to detect any swap is there or not
    for(int j = 0; j<size-i-1; j++) {
      if(array[j] > array[j+1]) {       //when the current item is bigger than
next
        swapping(array[j], array[j+1]);
        swaps = 1;    //set swap flag
      }
    }
    if(!swaps)
      break;       // No swap in this pass, so array is sorted
  }
}
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);
  bubbleSort(arr, n);
  cout << "Array after Sorting: ";
  display(arr, n);
}
```

OUTPUT:

Enter the number of elements: 6
Enter elements:
56 98 78 12 30 51
Array before Sorting: 56 98 78 12 30 51
Array after Sorting: 12 30 51 56 78 98

11. Write a program for sorting the given list numbers in ascending order using the following technique:
b) Selection Sort

```
// Selection Sort
#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 bubbleSort(int *array, int size) {
  for(int i = 0; i<size; i++) {
    int swaps = 0;        //flag to detect any swap is there or not
    for(int j = 0; j<size-i-1; j++) {
      if(array[j] > array[j+1]) {      //when the current item is bigger than
next
        swapping(array[j], array[j+1]);
        swaps = 1;    //set swap flag
      }
    }
    if(!swaps)
      break;      // No swap in this pass, so array is sorted
  }
}
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);
  bubbleSort(arr, n);
  cout << "Array after Sorting: ";
  display(arr, n);
}
```

OUTPUT:

Enter the number of elements: 6
Enter elements:
56 98 78 12 30 51
Array before Sorting: 56 98 78 12 30 51
Array after Sorting: 12 30 51 56 78 98

12. **Write a program for sorting the given list numbers in ascending order using the following technique:**

a. Insertion sort

```
#include<iostream>
using namespace std;
void display(int *array, int size) {
  for(int i = 0; i<size; i++)
    cout << array[i] << " ";
  cout << endl;
}
void insertionSort(int *array, int size) {
  int key, j;
  for(int i = 1; i<size; i++) {
    key = array[i];//take value
    j = i;
    while(j > 0 && array[j-1]>key) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = key;   //insert in right place
  }
}
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);
  insertionSort(arr, n);
  cout << "Array after Sorting: ";
  display(arr, n);
}
```

OUTPUT:

Enter the number of elements: 6
Enter elements:
9 45 23 71 80 55
Array before Sorting: 9 45 23 71 80 55
Array after Sorting: 9 23 45 55 71 80

12. **Write a program for sorting the given list numbers in ascending order using the following technique:**

b. Quick sort

```
// C++ Implementation of the Quick Sort Algorithm.
#include <iostream>
using namespace std;
int partition(int arr[], int start, int end)
{
	int pivot = arr[start];
	int count = 0;
	for (int i = start + 1; i <= end; i++) {
		if (arr[i] <= pivot)
			count++;
	}
	// Giving pivot element its correct position
	int pivotIndex = start + count;
	swap(arr[pivotIndex], arr[start]);
	// Sorting left and right parts of the pivot element
	int i = start, j = end;
	while (i < pivotIndex && j > pivotIndex) {
		while (arr[i] <= pivot) {
			i++;
		}
		while (arr[j] > pivot) {
			j--;
		}
		if (i < pivotIndex && j > pivotIndex) {
			swap(arr[i++], arr[j--]);
		}
	}
	return pivotIndex;
}
void quickSort(int arr[], int start, int end)
{
	// base case
	if (start >= end)
		return;
```

```
        // partitioning the array
        int p = partition(arr, start, end);
        // Sorting the left part
        quickSort(arr, start, p - 1);
        // Sorting the right part
        quickSort(arr, p + 1, end);
}
int main()
{
        int arr[] = { 9, 3, 4, 2, 1, 8 };
        int n = 6;
        quickSort(arr, 0, n - 1);
        for (int i = 0; i < n; i++) {
                cout << arr[i] << " ";
        }
        return 0;
}
```

OUTPUT:

1 2 3 4 8 9

13.Write a program for sorting the given list numbers in ascending order using the following technique:
a)Merge sort

```
//Merg Sort:
#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;
  //marge 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 << "Array after Sorting: ";
  display(arr, n);
}
```

OUTPUT:

Enter the number of elements: 6
Enter elements:
14 20 78 98 20 45
Array before Sorting: 14 20 78 98 20 45
Array after Sorting: 14 20 20 45 78 98

13.Write a program for sorting the given list numbers in ascending order using the following technique:
b) Heapsort

```
// C++ program for implementation of Heap Sort
#include <iostream>
using namespace std;
/* To heapify a subtree rooted with node i which is an index in arr[]. n is
size of heap*/
void heapify(int arr[], int n, int i)
{
	int largest = i; // Initialize largest as root
	int l = 2 * i + 1; // left = 2*i + 1
	int r = 2 * i + 2; // right = 2*i + 2
	// If left child is larger than root
	if (l < n && arr[l] > arr[largest])
		largest = l;
	// If right child is larger than largest so far
	if (r < n && arr[r] > arr[largest])
		largest = r;
	// If largest is not root
	if (largest != i) {
		swap(arr[i], arr[largest]);
		// Recursively heapify the affected sub-tree
		heapify(arr, n, largest);
	}
}
// main function to do heap sort
void heapSort(int arr[], int n)
{
	// Build heap (rearrange array)
	for (int i = n / 2 - 1; i >= 0; i--)
		heapify(arr, n, i);
	// One by one extract an element from heap
	for (int i = n - 1; i >= 0; i--) {
		// Move current root to end
		swap(arr[0], arr[i]);
		// call max heapify on the reduced heap
```

```
            heapify(arr, i, 0);
        }
}
/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
        for (int i = 0; i < n; ++i)
                cout << arr[i] << " ";
        cout << "\n";
}
// Driver program
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int n = sizeof(arr) / sizeof(arr[0]);
        heapSort(arr, n);
        cout << "Sorted array is \n";
        printArray(arr, n);
}
```

OUTPUT:

Sorted array is
5 6 7 11 12 13

14. Write a program to traverse a binary tree in following way:
a) Pre-order

```
#include<iostream>
using namespace std;
struct node {
  int data;
  struct node *left;
  struct node *right;
};

struct node *createNode(int val) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->data = val;
  temp->left = temp->right = NULL;
  return temp;
}

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

struct node* insertNode(struct node* node, int val) {
  if (node == NULL) return createNode(val);
  if (val < node->data)
  node->left = insertNode(node->left, val);
  else if (val > node->data)
  node->right = insertNode(node->right, val);
  return node;
}

int main() {
  struct node *root = NULL;
```

```
  root = insertNode(root, 4);
  insertNode(root, 5);
  insertNode(root, 2);
  insertNode(root, 9);
  insertNode(root, 1);
  insertNode(root, 3);
  cout<<"Pre-Order traversal of the Binary Search Tree is: ";
  preorder(root);
  return 0;
}
```

OUTPUT:

Pre-Order traversal of the Binary Search Tree is: 4 2 1 3 5 9

14. Write a program to traverse a binary tree in following way:
b)In-order

```
/* C++ program to construct tree  from inorder traversal */
#include <bits/stdc++.h>
using namespace std;
/* A binary tree node has data, pointer to left child and a pointer to right
child */
class node
{
	public:
	int data;
	node* left;
	node* right;
};
/* Prototypes of a utility function to get the maximum value in
inorder[start..end] */
int max(int inorder[], int strt, int end);
/* A utility function to allocate memory for a node */
node* newNode(int data);
/* Recursive function to construct binary of size len from Inorder traversal
inorder[]. Initial values of start and end should be 0 and len -1. */
node* buildTree (int inorder[], int start, int end)
{
	if (start > end)
		return NULL;
	/* Find index of the maximum element from Binary Tree */
	int i = max (inorder, start, end);
	/* Pick the maximum value and make it root */
	node *root = newNode(inorder[i]);
	/* If this is the only element in inorder[start..end], then return it */
	if (start == end)
		return root;
	/* Using index in Inorder traversal, construct left and  right subtress
*/
	root->left = buildTree (inorder, start, i - 1);
	root->right = buildTree (inorder, i + 1, end);
```

```
        return root;
}
/* UTILITY FUNCTIONS */
/* Function to find index of the maximum value in arr[start...end] */
int max (int arr[], int strt, int end)
{
        int i, max = arr[strt], maxind = strt;
        for(i = strt + 1; i <= end; i++)
        {
                if(arr[i] > max)
                {
                        max = arr[i];
                        maxind = i;
                }
        }
        return maxind;
}
/* Helper function that allocates a new node with the given data and
NULL left and right pointers. */
node* newNode (int data)
{
        node* Node = new node();
        Node->data = data;
        Node->left = NULL;
        Node->right = NULL;
        return Node;
}
/* This function is here just to test buildTree() */
void printInorder (node* node)
{
        if (node == NULL)
                return;
        /* first recur on left child */
        printInorder (node->left);
        /* then print the data of node */
        cout<<node->data<<" ";
        /* now recur on right child */
```

```
	printInorder (node->right);
}
/* Driver code*/
int main()
{
	/* Assume that inorder traversal of following tree is given
		40
		/ \
		10 30
		/	\
		5	28 */
	int inorder[] = {5, 10, 40, 30, 28};
	int len = sizeof(inorder)/sizeof(inorder[0]);
	node *root = buildTree(inorder, 0, len - 1);
	/* Let us test the built tree by printing Inorder traversal */
	cout << "Inorder traversal of the constructed tree is \n";
	printInorder(root);
	return 0;
}
```

OUTPUT:

```
Inorder traversal of the constructed tree is
5  10   40  30  28
```

14.Write a program to traverse a binary tree in following way:
c)Post-order

```
#include<iostream>
using namespace std;
struct node {
  int data;
  struct node *left;
  struct node *right;
};

struct node *createNode(int val) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->data = val;
  temp->left = temp->right = NULL;
  return temp;
}

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

struct node* insertNode(struct node* node, int val) {
  if (node == NULL) return createNode(val);
  if (val < node->data)
  node->left = insertNode(node->left, val);
  else if (val > node->data)
  node->right = insertNode(node->right, val);
  return node;
}

int main() {
  struct node *root = NULL;
```

```
  root = insertNode(root, 4);
  insertNode(root, 5);
  insertNode(root, 2);
  insertNode(root, 9);
  insertNode(root, 1);
  insertNode(root, 3);
  cout<<"Post-Order traversal of the Binary Search Tree is: ";
  postorder(root);
  return 0;
}
```

OUTPUT:

Post-Order traversal of the Binary Search Tree is: 1 3 2 9 5 4

15. Write a program to the implementation graph traversals – BFS and DFS.

```
 #include<iostream>
#include<vector>
#include<queue>
#include<stack>
using namespace std;
//add the edge in graph
void edge(vector<int>adj[],int u,int v){
  adj[u].push_back(v);
}
//function for bfs traversal
void bfs(int s,vector<int>adj[],bool visit[]){
  queue<int>q;//queue in STL
  q.push(s);
  visit[s]=true;
  while(!q.empty()){
    int u=q.front();
    cout<<u<<" ";
    q.pop();
//loop for traverse
    for(int i=0;i<adj[u].size();i++){
      if(!visit[adj[u][i]]){
        q.push(adj[u][i]);
        visit[adj[u][i]]=true;
      }
    }
  }
}
//function for dfs traversal
void dfs(int s,vector<int>adj[],bool visit[]){
  stack<int>stk;//stack in STL
  stk.push(s);
  visit[s]=true;
  while(!stk.empty()){
    int u=stk.top();
    cout<<u<<" ";
```

```
    stk.pop();
//loop for traverse
    for(int i=0;i<adj[u].size();i++){
      if(!visit[adj[u][i]]){
        stk.push(adj[u][i]);
        visit[adj[u][i]]=true;
      }
    }
  }
}
int main(){
  vector<int>adj[5];//vector of array to store the graph
  bool visit[5];//array to check visit or not of a node
  //initially all node are unvisited
  for(int i=0;i<5;i++){
    visit[i]=false;
  }
   edge(adj,0,2); //input for edges
  edge(adj,0,1); //input for edges
  edge(adj,1,3); //input for edges
  edge(adj,2,0); //input for edges
  edge(adj,2,3); //input for edges
  edge(adj,2,4); //input for edges
  cout<<"BFS traversal is"<<"  ";
  bfs(0,adj,visit);  //call bfs function //1 is a starting point
  cout<<endl;
  //again initialise all node unvisited for dfs
  for(int i=0;i<5;i++){
    visit[i]=false;
  }
  cout<<"DFS traversal is"<<"  ";
    dfs(0,adj,visit); //call dfs function //1 is a starting point
}
```

OUTPUT:

BFS traversal is 0 2 1 3 4

DFS traversals is 0 1 3 2 4

16. Write a program to find the minimum spanning tree for a weighted graph using: a) Prim'sAlgorithm

```
// A program Demonstrating Prim's Algorithm
#include <iostream>
 #include<bits/stdc++.h>
 #include <cstring>
 using namespace std;

 // number of vertices in graph
 #define V 7

 // create a 2d array of size 7x7
 //for adjacency matrix to represent graph

 int main () {
   // create a 2d array of size 7x7
 //for adjacency matrix to represent graph
   int G[V][V] = {
   {0,28,0,0,0,10,0},
 {28,0,16,0,0,0,14},
 {0,16,0,12,0,0,0},
 {0,0,12,22,0,18},
 {0,0,0,22,0,25,24},
 {10,0,0,0,25,0,0},
 {0,14,0,18,24,0,0}
 };

   int edge;            // number of edge

   // create an array to check visited vertex
   int visit[V];

   //initialise the visit array to false
  for(int i=0;i<V;i++){
    visit[i]=false;
}
```

```
// set number of edge to 0
edge = 0;

// the number of edges in minimum spanning tree will be
// always less than (V -1), where V is the number of vertices in
//graph

// choose 0th vertex and make it true
visit[0] = true;

int x;          //  row number
int y;          //  col number

// print for edge and weight
cout << "Edge" << " : " << "Weight";
cout << endl;
while (edge < V - 1) {//in spanning tree consist the V-1 number of edges

//For every vertex in the set S, find the all adjacent vertices
// , calculate the distance from the vertex selected.
// if the vertex is already visited, discard it otherwise
//choose another vertex nearest to selected vertex.

    int min = INT_MAX;
    x = 0;
    y = 0;

    for (int i = 0; i < V; i++) {
      if (visit[i]) {
          for (int j = 0; j < V; j++) {
            if (!visit[j] && G[i][j]) { // not in selected and there is an edge
                if (min > G[i][j]) {
                    min = G[i][j];
                    x = i;
                    y = j;
                }
```

```
              }
          }
        }
      }
      cout << x <<  " ---> " << y << " :  " << G[x][y];
      cout << endl;
      visit[y] = true;
      edge++;
    }

  return 0;
}
```

OUTPUT:

```
Edge : Weight
0 ---> 5 : 10
5 ---> 4 : 25
4 ---> 3 : 22
3 ---> 2 : 12
2 ---> 1 : 16
1 ---> 6 : 14
```

16. Write a program to find the minimum spanning tree for a weighted graph using: b) Kruskal'sAlgorithm

```
#include <iostream>
 #include <vector>
 #include <utility>
 #include <algorithm>
 using namespace std;
 const int MAX = 1000;
 int id[MAX], nodes, edges;  //array id is use for check the parent of
vertex;
 pair <long long, pair<int, int> > p[MAX];

   //initialise the parent array id[]
 void init()
 {
    for(int i = 0;i < MAX;++i)
        id[i] = i;
 }

 int root(int x)
 {
    while(id[x] != x)  //if x is not itself parent then update its parent
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x; //return the parent
 }

//function for union
 void union1(int x, int y)
 {
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
 }
```

```
//function to find out the edges in minimum spanning tree and its cost
long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        if(root(x) != root(y))
        {
            minimumCost += cost;
    cout<<x<<" ----> "<<y<<" :"<<p[i].first<<endl;//print the edges contain
in spanning tree
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    init();
    cout <<"Enter Nodes and edges"<<endl;
    cin >> nodes >> edges;

    //enter the vertex and cost of edges
    for(int i = 0;i < edges;++i)
    {
        cout<<"Enter the value of X, Y and edges"<<endl;
    cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }
```

```
    //sort the edges according to their cost
    sort(p, p + edges);
    minimumCost = kruskal(p);
    cout <<"Minimum cost is "<< minimumCost << endl;
    return 0;
}
```

OUTPUT:

```
Enter Nodes and edges
7
9
Enter the value of X, Y and edges
0
5
10
Enter the value of X, Y and edges
5
4
25
Enter the value of X, Y and edges
4
3
22
Enter the value of X, Y and edges
3
2
12
Enter the value of X, Y and edges
2
1
16
Enter the value of X, Y and edges
1
0
28
```

Enter the value of X, Y and edges
1
6
14
Enter the value of X, Y and edges
6
4
24
Enter the value of X, Y and edges
3
6
18
0 ----> 5 :10
3 ----> 2 :12
1 ----> 6 :14
2 ----> 1 :16
4 ----> 3 :22
5 ----> 4 :25
Minimum cost is 99