**DS LAB MANUEL**

**List of Programs:**

**1. Write a C program for sorting a list using Bubble sort and then apply binary search.**

**2. Write a C program for implementing the operations of a queue.**

**3. Write a C program to implement the operations on priority queues.**

**4. Write a C program to implement the operations on circular queues.**

**5. Write a C program to implement the operations on stacks.**

**6. Write a C program for evaluating a given postfix expression using stack.**

**7. Write a C program for converting a given infix expression to postfix form using stack.**

**8. Write a C program to implement the operations on single linked list.**

**9. Write a C program for demonstrates operations on double linked list.**

**10. Write a C program for the representation of polynomials using circular linked list and for the addition of two such polynomials**

**11. Write a C program to create a binary search tree and for implementing the in order, Pre order, post order traversal using recursion**

**12. a) Write a C program for finding the transitive closure of a digraph**

**b) Write a C program for finding the shortest path from a given source to any vertex in a digraph using Dijkstra’s algorithm.**

**13. a) Write a C program for finding the Depth First Search of a graph.**

**b) Write a C program for finding the Breadth First Search of a graph.**

**1. Write a C program for sorting a list using Bubble sort and then apply binary search.**

***/\*C program to accept N numbers sorted in ascending orderand to search for a given number using Binary Search. Report success or failure.\*/***

**#include <stdio.h>**

**void main()**

**{**

**int array[10];**

**int i, j, num, temp, keynum;**

**int low, mid, high;**

**printf("Enter the value of num \n");**

**scanf("%d", &num);**

**printf("Enter the elements one by one \n");**

**for (i = 0; i < num; i++)**

**{**

**scanf("%d", &array[i]);**

**}**

**printf("Input array elements \n");**

**for (i = 0; i < num; i++)**

**{**

**printf("%d\n", array[i]);**

**}**

***/\* Bubble sorting begins \*/***

**for (i = 0; i < num; i++)**

**{**

**for (j = 0; j < (num - i - 1); j++)**

**{**

**if (array[j] > array[j + 1])**

**{**

**temp = array[j];**

**array[j] = array[j + 1];**

**array[j + 1] = temp;**

**}**

**}**

**}**

**printf("Sorted array is...\n");**

**for (i = 0; i < num; i++)**

**{**

**printf("%d\n", array[i]);**

**}**

**printf("Enter the element to be searched \n");**

**scanf("%d", &keynum);**

***/\* Binary searching begins \*/***

**low = 1;**

**high = num;**

**do**

**{**

**mid = (low + high) / 2;**

**if (keynum < array[mid])**

**high = mid - 1;**

**else if (keynum > array[mid])**

**low = mid + 1;**

**} while (keynum != array[mid] && low <= high);**

**if (keynum == array[mid])**

**{**

**printf("SEARCH SUCCESSFUL \n");**

**}**

**else**

**{**

**printf("SEARCH FAILED \n");**

**}**

**}**

**OUTPUT:**

**1. Enter the value of num**

**5**

**Enter the elements one by one**

**8**

**2**

**4**

**6**

**5**

**Input array elements**

**8**

**2**

**4**

**6**

**5**

**Sorted array is...**

**2**

**4**

**5**

**6**

**8**

**Enter the element to be searched**

**5**

**SEARCH SUCCESSFUL**

**2. Enter the value of num**

**5**

**Enter the elements one by one**

**8**

**2**

**6**

**4**

**1**

**Input array elements**

**8**

**2**

**6**

**4**

**1**

**Sorted array is...**

**1**

**2**

**4**

**6**

**8**

**Enter the element to be searched**

**5**

**SEARCH FAILED**

**2. Write a C program for implementing the operations of a queue.**

**Program:**

**#include<stdio.h>**

**#define n 5**

**int main()**

**{**

**int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;**

**printf("Queue using Array");**

**printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");**

**while(ch)**

**{**

**printf("\nEnter the Choice:");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**if(rear==x)**

**printf("\n Queue is Full");**

**else**

**{**

**printf("\n Enter no %d:",j++);**

**scanf("%d",&queue[rear++]);**

**}**

**break;**

**case 2:**

**if(front==rear)**

**{**

**printf("\n Queue is empty");**

**}**

**else**

**{**

**printf("\n Deleted Element is %d",queue[front++]);**

**x++;**

**}**

**break;**

**case 3:**

**printf("\nQueue Elements are:\n ");**

**if(front==rear)**

**printf("\n Queue is Empty");**

**else**

**{**

**for(i=front; i<rear; i++)**

**{**

**printf("%d",queue[i]);**

**printf("\n");**

**}**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("Wrong Choice: please see the options");**

**}**

**}**

**}**

**return 0;**

**}**

**OUTPUT:**

**Queue using Array**

**1. Insertion**

**2. Deletion**

**3. Display**

**4. Exit**

**Enter the Choice: 1**

**Enter no 1:10**

**Enter the Choice: 1**

**Enter no 2:54**

**Enter the Choice: 1**

**Enter no 3:98**

**Enter the Choice: 1**

**Enter no 4:234**

**Enter the Choice: 3**

**Queue Elements are:**

**10**

**54**

**98**

**234**

**Enter the Choice: 2**

**Deleted Element is 10**

**Enter the Choice: 3**

**Queue Elements are:**

**54**

**98**

**234**

**Enter the Choice: 4**

[**https://www.programming9.com/programs/c-programs/303-implementation-of-queue-using-array-in-c**](https://www.programming9.com/programs/c-programs/303-implementation-of-queue-using-array-in-c)

[**https://www.sanfoundry.com/c-program-queue-using-array/**](https://www.sanfoundry.com/c-program-queue-using-array/)

**3. Write a C program to implement the operations on priority queues.**

**Program:**

***/\****

***\* C Program to Implement Priority Queue to Add and Delete Elements***

***\*/***

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 5**

**void insert\_by\_priority(int);**

**void delete\_by\_priority(int);**

**void create();**

**void check(int);**

**void display\_pqueue();**

**int pri\_que[MAX];**

**int front, rear;**

**void main()**

**{**

**int n, ch;**

**printf("\n1 - Insert an element into queue");**

**printf("\n2 - Delete an element from queue");**

**printf("\n3 - Display queue elements");**

**printf("\n4 - Exit");**

**create();**

**while (1)**

**{**

**printf("\nEnter your choice : ");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**printf("\nEnter value to be inserted : ");**

**scanf("%d",&n);**

**insert\_by\_priority(n);**

**break;**

**case 2:**

**printf("\nEnter value to delete : ");**

**scanf("%d",&n);**

**delete\_by\_priority(n);**

**break;**

**case 3:**

**display\_pqueue();**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("\nChoice is incorrect, Enter a correct choice");**

**}**

**}**

**}**

***/\* Function to create an empty priority queue \*/***

**void create()**

**{**

**front = rear = -1;**

**}**

***/\* Function to insert value into priority queue \*/***

**void insert\_by\_priority(int data)**

**{**

**if (rear >= MAX - 1)**

**{**

**printf("\nQueue overflow no more elements can be inserted");**

**return;**

**}**

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

**{**

**front++;**

**rear++;**

**pri\_que[rear] = data;**

**return;**

**}**

**else**

**check(data);**

**rear++;**

**}**

***/\* Function to check priority and place element \*/***

**void check(int data)**

**{**

**int i,j;**

**for (i = 0; i <= rear; i++)**

**{**

**if (data >= pri\_que[i])**

**{**

**for (j = rear + 1; j > i; j--)**

**{**

**pri\_que[j] = pri\_que[j - 1];**

**}**

**pri\_que[i] = data;**

**return;**

**}**

**}**

**pri\_que[i] = data;**

**}**

***/\* Function to delete an element from queue \*/***

**void delete\_by\_priority(int data)**

**{**

**int i;**

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

**{**

**printf("\nQueue is empty no elements to delete");**

**return;**

**}**

**for (i = 0; i <= rear; i++)**

**{**

**if (data == pri\_que[i])**

**{**

**for (; i < rear; i++)**

**{**

**pri\_que[i] = pri\_que[i + 1];**

**}**

**pri\_que[i] = -99;**

**rear--;**

**if (rear == -1)**

**front = -1;**

**return;**

**}**

**}**

**printf("\n%d not found in queue to delete", data);**

**}**

***/\* Function to display queue elements \*/***

**void display\_pqueue()**

**{**

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

**{**

**printf("\nQueue is empty");**

**return;**

**}**

**for (; front <= rear; front++)**

**{**

**printf(" %d ", pri\_que[front]);**

**}**

**front = 0;**

**}**

**OUTPUT:**

1 - Insert an element into queue

2 - Delete an element from queue

3 - Display queue elements

4 - Exit

Enter your choice: 1

Enter value to be inserted: 20

Enter your choice: 1

Enter value to be inserted: 45

Enter your choice: 1

Enter value to be inserted: 89

Enter your choice: 3

89 45 20

Enter your choice: 1

Enter value to be inserted: 56

Enter your choice: 3

89 56 45 20

Enter your choice: 2

Enter value to delete: 45

Enter your choice: 3

89 56 20

Enter your choice: 4

[**https://www.sanfoundry.com/c-program-priority-queue/**](https://www.sanfoundry.com/c-program-priority-queue/)

**4. Write a C program to implement the operations on circular queues.**

**Program:**

**#include<stdio.h>**

**# define MAX 5**

**int cqueue\_arr[MAX];**

**int front = -1;**

**int rear = -1;**

**void insert(int item)**

**{**

**if((front == 0 && rear == MAX-1) || (front == rear+1))**

**{**

**printf("Queue Overflow n");**

**return;**

**}**

**if(front == -1)**

**{**

**front = 0;**

**rear = 0;**

**}**

**else**

**{**

**if(rear == MAX-1)**

**rear = 0;**

**else**

**rear = rear+1;**

**}**

**cqueue\_arr[rear] = item ;**

**}**

**void deletion()**

**{**

**if(front == -1)**

**{**

**printf("Queue Underflown");**

**return ;**

**}**

**printf("Element deleted from queue is : %dn",cqueue\_arr[front]);**

**if(front == rear)**

**{**

**front = -1;**

**rear=-1;**

**}**

**else**

**{**

**if(front == MAX-1)**

**front = 0;**

**else**

**front = front+1;**

**}**

**}**

**int display()**

**{**

**int front\_pos = front,rear\_pos = rear;**

**if(front == -1)**

**{**

**printf("Queue is emptyn");**

**}**

**printf("Queue elements :n");**

**if( front\_pos <= rear\_pos )**

**while(front\_pos <= rear\_pos)**

**{**

**printf("%d ",cqueue\_arr[front\_pos]);**

**front\_pos++;**

**}**

**else**

**{**

**while(front\_pos <= MAX-1)**

**{**

**printf("%d ",cqueue\_arr[front\_pos]);**

**front\_pos++;**

**}**

**front\_pos = 0;**

**while(front\_pos <= rear\_pos)**

**{**

**printf("%d ",cqueue\_arr[front\_pos]);**

**front\_pos++;**

**}**

**}**

**printf("n");**

**}**

**int main()**

**{**

**int choice,item;**

**do**

**{**

**printf("1.Insert");**

**printf("2.Delete");**

**printf("3.Display");**

**printf("4.Quit");**

**printf("Enter your choice : ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1 :**

**printf("Input the element for insertion in queue : ");**

**scanf("%d", &item);**

**insert(item);**

**break;**

**case 2 :**

**deletion();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**break;**

**default:**

**printf("Wrong choicen");**

**}**

**}while(choice!=4);**

**return 0;**

**}**

**OUTPUT:**

**1. Insert 2.Delete 3.Display 4.Quit Enter your choice: 1**

**Input the element for insertion in queue: 4**

**1. Insert 2.Delete 3.Display 4.Quit Enter your choice: 1**

**Input the element for insertion in queue: 5**

**1. Insert 2.Delete 3.Display 4.Quit Enter your choice: 1**

**Input the element for insertion in queue: 1**

**1. Insert 2.Delete 3.Display 4.Quit Enter your choice: 1**

**Input the element for insertion in queue: 8**

**1. Insert 2.Delete 3.Display 4.Quit Enter your choice: 3**

**Queue elements: 4 5 1 8 1.Insert 2.Delete 3.Display 4.Quit Enter your choice: 2**

**Element deleted from queue is: 4 1.Insert 2.Delete 3.Display 4.Quit Enter your choice: 2**

**Element deleted from queue is: 5 1.Insert 2.Delete 3.Display 4.Quit Enter your choice: 3**

**Queue elements: 1 8 1.Insert 2.Delete 3.Display 4.Quit Enter your choice: 4**

[**https://www.edureka.co/blog/circular-queue-in-c/**](https://www.edureka.co/blog/circular-queue-in-c/)

[**https://onecompiler.com/c/3xfj665a8**](https://onecompiler.com/c/3xfj665a8)

**5. Write a C program to implement the operations on stacks.**

**Aim:** To write C program to implement the operations on stacks.

**Program:**

#include<stdio.h>

#include<stdlib.h>

#define MAXSTK 100

int top=-1;

int items[MAXSTK];

int Isempty();

int Isfull();

void Push(int);

int Pop();

void Display();

void main()

{

int x;

char ch='1';

while(ch!='4')

{

printf("\n 1-PUSH");

printf("\n 2-POP");

printf("\n 3-DISPLAY");

printf("\n 4-QUIT");

printf("\n Enter your choice:");

fflush(stdin);

ch=getchar();

switch(ch)

{

case '1':

printf("\n Enter the element to be pushed:");

scanf("%d",&x);

Push(x);

break;

case '2':

x=Pop();

printf("\n Pop element is %d\n:",x);

break;

case '3':

Display();

break;

case '4':

break;

default:

printf("\n Wrong choice!Try again:");

}

}

}

int Isempty()

{

if(top==-1)

return 1;

else

return 0;

}

int Isfull()

{

if(top==MAXSTK-1)

return 1;

else

return 0;

}

void Push(int x)

{

if(Isfull())

{

printf("\n Stack full");

return;

}

top++;

items[top]=x;

}

int Pop()

{

int x;

if(Isempty())

{

printf("\n Stack empty");

exit(0);

}

x=items[top];

top--;

return x;

}

void Display()

{

int i;

if(Isempty())

{

printf("\n Stack empty");

return;

}

printf("\n Elements in the Stack are :\n");

for(i=top;i>=0;i--)

printf("%d\n",items[i]);

}

**OUTPUT:**

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 1**

Enter the element to be pushed: 5

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 1**

Enter the element to be pushed: 7

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 1**

Enter the element to be pushed: 3

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 3**

Elements in the Stack are:

3

7

5

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 2**

Pop element is 3

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 2**

Pop element is 7

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 3**

Elements in the Stack are: 5

1-PUSH

2-POP

3-DISPLAY

4-QUIT

**Enter your choice: 4**

**6. Write a C program for evaluating a given postfix expression using stack.**

**Aim: To write a C program for evaluating a given postfix expression using stack**

**Algorithm:**

1. **Add ) to postfix expression.**
2. **Read postfix expression Left to Right until ) encountered**
3. **If operand is encountered, push it onto Stack  
   [End If]**
4. **If operator is encountered, Pop two elements**
   1. **A -> Top element**
   2. **B-> Next to Top element**
   3. **Evaluate B operator A**
   4. **Push B operator A onto Stack**
5. **Set result = pop**
6. **END**

**PROGRAM:**

**/\* This program is for evaluation of postfix expression This program assumes that there are only four operators (\*, /, +, -) in an expression and operand is single digit only further this program does not do any error handling e.g. it does not check that entered postfix expression is valid or not. \*/**

**#include <stdio.h>**

**#include <ctype.h>**

**#define MAXSTACK 100 /\* for max size of stack \*/**

**#define POSTFIXSIZE 100 /\* define max number of charcters in postfix expression \*/**

**/\* declare stack and its top pointer to be used during postfix expression evaluation\*/**

**int stack[MAXSTACK];**

**int top = -1; /\* because array index in C begins at 0 \*/**

**/\* can be do this initialization somewhere else \*/**

**/\* define push operation \*/**

**void push(int item)**

**{**

**if (top >= MAXSTACK - 1) {**

**printf("stack over flow");**

**return;**

**}**

**else {**

**top = top + 1;**

**stack[top] = item;**

**}**

**}**

**/\* define pop operation \*/**

**int pop()**

**{**

**int item;**

**if (top < 0) {**

**printf("stack under flow");**

**}**

**else {**

**item = stack[top];**

**top = top - 1;**

**return item;**

**}**

**}**

**/\* define function that is used to input postfix expression and to evaluate it \*/**

**void EvalPostfix(char postfix[])**

**{**

**int i;**

**char ch;**

**int val;**

**int A, B;**

**/\* evaluate postfix expression \*/**

**for (i = 0; postfix[i] != ')'; i++) {**

**ch = postfix[i];**

**if (isdigit(ch)) {**

**/\* we saw an operand,push the digit onto stack ch - '0' is used for getting digit rather than ASCII code of digit \*/**

**push(ch - '0');**

**}**

**else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') {**

**/\* we saw an operator**

**\* pop top element A and next-to-top elemnet B**

**\* from stack and compute B operator A**

**\*/**

**A = pop();**

**B = pop();**

**switch (ch) /\* ch is an operator \*/**

**{**

**case '\*':**

**val = B \* A;**

**break;**

**case '/':**

**val = B / A;**

**break;**

**case '+':**

**val = B + A;**

**break;**

**case '-':**

**val = B - A;**

**break;**

**}**

**/\* push the value obtained above onto the stack \*/**

**push(val);**

**}**

**}**

**printf(" \n Result of expression evaluation : %d \n", pop());**

**}**

**int main()**

**{**

**int i;**

**/\* declare character array to store postfix expression \*/**

**char postfix[POSTFIXSIZE];**

**printf("ASSUMPTION: There are only four operators(\*, /, +, -) in an expression and operand is single digit only.\n");**

**printf(" \nEnter postfix expression,\npress right parenthesis ')' for end expression : ");**

**/\* take input of postfix expression from user \*/**

**for (i = 0; i <= POSTFIXSIZE - 1; i++) {**

**scanf("%c", &postfix[i]);**

**if (postfix[i] == ')') /\* is there any way to eliminate this if \*/**

**{**

**break;**

**} /\* and break statement \*/**

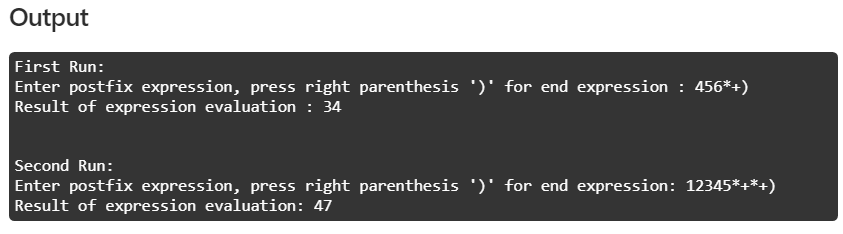
**}**

**/\* call function to evaluate postfix expression \*/**

**EvalPostfix(postfix);**

**return 0;**

**}**

****

[**https://www.includehelp.com/c/evaluation-of-postfix-expressions-using-stack-with-c-program.aspx**](https://www.includehelp.com/c/evaluation-of-postfix-expressions-using-stack-with-c-program.aspx)

**7. Write a C program for converting a given infix expression to postfix form using stack.**

**Program:**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c ",pop());

}return 0;

}

**Output-1:**

**Enter the expression: a+b\*c**

**a b c \* +**

**Output-2:**

**Enter the expression: (a+b)\*c+(d-a)**

**a b + c \* d a - +**

**Output-3:**

**Enter the expression: ((4+8)(6-5))/((3-2)(2+2))**

**4 8 + 6 5 - 3 2 - 2 2 + /**

[**https://www.programming9.com/programs/c-programs/230-c-program-to-convert-infix-to-postfix-expression-using-stack**](https://www.programming9.com/programs/c-programs/230-c-program-to-convert-infix-to-postfix-expression-using-stack)

**8. Write a C program to implement the operations on single linked list.**

**Program:**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Node;**

**typedef struct Node \* PtrToNode;**

**typedef PtrToNode List;**

**typedef PtrToNode Position;**

**struct Node**

**{**

**int e;**

**Position next;**

**};**

**void Insert(int x, List l, Position p)**

**{**

**Position TmpCell;**

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

**if(TmpCell == NULL)**

**printf("Memory out of space\n");**

**else**

**{**

**TmpCell->e = x;**

**TmpCell->next = p->next;**

**p->next = TmpCell;**

**}**

**}**

**int isLast(Position p)**

**{**

**return (p->next == NULL);**

**}**

**Position FindPrevious(int x, List l)**

**{**

**Position p = l;**

**while(p->next != NULL && p->next->e != x)**

**p = p->next;**

**return p;**

**}**

**void Delete(int x, List l)**

**{**

**Position p, TmpCell;**

**p = FindPrevious(x, l);**

**if(!isLast(p))**

**{**

**TmpCell = p->next;**

**p->next = TmpCell->next;**

**free(TmpCell);**

**}**

**else**

**printf("Element does not exist!!!\n");**

**}**

**void Display(List l)**

**{**

**printf("The list element are :: ");**

**Position p = l->next;**

**while(p != NULL)**

**{**

**printf("%d -> ", p->e);**

**p = p->next;**

**}**

**}**

**void Merge(List l, List l1)**

**{**

**int i, n, x, j;**

**Position p;**

**printf("Enter the number of elements to be merged :: ");**

**scanf("%d",&n);**

**for(i = 1; i <= n; i++)**

**{**

**p = l1;**

**scanf("%d", &x);**

**for(j = 1; j < i; j++)**

**p = p->next;**

**Insert(x, l1, p);**

**}**

**printf("The new List :: ");**

**Display(l1);**

**printf("The merged List ::");**

**p = l;**

**while(p->next != NULL)**

**{**

**p = p->next;**

**}**

**p->next = l1->next;**

**Display(l);**

**}**

**int main()**

**{**

**int x, pos, ch, i;**

**List l, l1;**

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

**l->next = NULL;**

**List p = l;**

**printf("LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");**

**do**

**{**

**printf("\n\n1. INSERT\t 2. DELETE\t 3. MERGE\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");**

**scanf("%d", &ch);**

**switch(ch)**

**{**

**case 1:**

**p = l;**

**printf("Enter the element to be inserted :: ");**

**scanf("%d",&x);**

**printf("Enter the position of the element :: ");**

**scanf("%d",&pos);**

**for(i = 1; i < pos; i++)**

**{**

**p = p->next;**

**}**

**Insert(x,l,p);**

**break;**

**case 2:**

**p = l;**

**printf("Enter the element to be deleted :: ");**

**scanf("%d",&x);**

**Delete(x,p);**

**break;**

**case 3:**

**l1 = (struct Node \*) malloc(sizeof(struct Node));**

**l1->next = NULL;**

**Merge(l, l1);**

**break;**

**case 4:**

**Display(l);**

**break;**

**}**

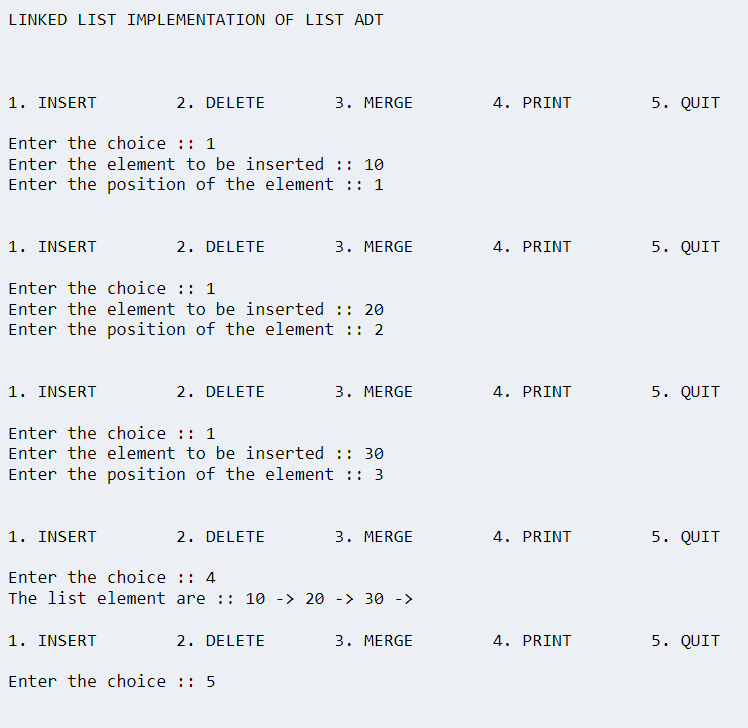
**}**

**while(ch<5);**

**return 0;**

**}**

**OUTPUT:**

****

[**https://www.programming9.com/programs/c-programs/226-c-program-to-implement-single-linked-list-operations**](https://www.programming9.com/programs/c-programs/226-c-program-to-implement-single-linked-list-operations)

**9. Write a C program for demonstrates operations on double linked list.**

**Program:**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Node;**

**typedef struct Node \* PtrToNode;**

**typedef PtrToNode List;**

**typedef PtrToNode Position;**

**struct Node**

**{**

**int e;**

**Position previous;**

**Position next;**

**};**

**void Insert(int x, List l, Position p)**

**{**

**Position TmpCell;**

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

**if(TmpCell == NULL)**

**printf("Memory out of space\n");**

**else**

**{**

**TmpCell->e = x;**

**TmpCell->previous = p;**

**TmpCell->next = p->next;**

**p->next = TmpCell;**

**}**

**}**

**int isLast(Position p)**

**{**

**return (p->next == NULL);**

**}**

**Position Find(int x, List l)**

**{**

**Position p = l->next;**

**while(p != NULL && p->e != x)**

**p = p->next;**

**return p;**

**}**

**void Delete(int x, List l)**

**{**

**Position p, p1, p2;**

**p = Find(x, l);**

**if(p != NULL)**

**{**

**p1 = p -> previous;**

**p2 = p -> next;**

**p1 -> next = p -> next;**

**if(p2 != NULL) // if the node is not the last node**

**p2 -> previous = p -> previous;**

**}**

**else**

**printf("Element does not exist!!!\n");**

**}**

**void Display(List l)**

**{**

**printf("The list element are :: ");**

**Position p = l->next;**

**while(p != NULL)**

**{**

**printf("%d -> ", p->e);**

**p = p->next;**

**}**

**}**

**void main()**

**{**

**int x, pos, ch, i;**

**List l, l1;**

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

**l->previous = NULL;**

**l->next = NULL;**

**List p = l;**

**printf("DOUBLY LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");**

**do**

**{**

**printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");**

**scanf("%d", &ch);**

**switch(ch)**

**{**

**case 1:**

**p = l;**

**printf("Enter the element to be inserted :: ");**

**scanf("%d",&x);**

**printf("Enter the position of the element :: ");**

**scanf("%d",&pos);**

**for(i = 1; i < pos; i++)**

**{**

**p = p->next;**

**}**

**Insert(x,l,p);**

**break;**

**case 2:**

**p = l;**

**printf("Enter the element to be deleted :: ");**

**scanf("%d",&x);**

**Delete(x,p);**

**break;**

**case 3:**

**p = l;**

**printf("Enter the element to be searched :: ");**

**scanf("%d",&x);**

**p = Find(x,p);**

**if(p == NULL)**

**printf("Element does not exist!!!\n");**

**else**

**printf("Element exist!!!\n");**

**break;**

**case 4:**

**Display(l);**

**break;**

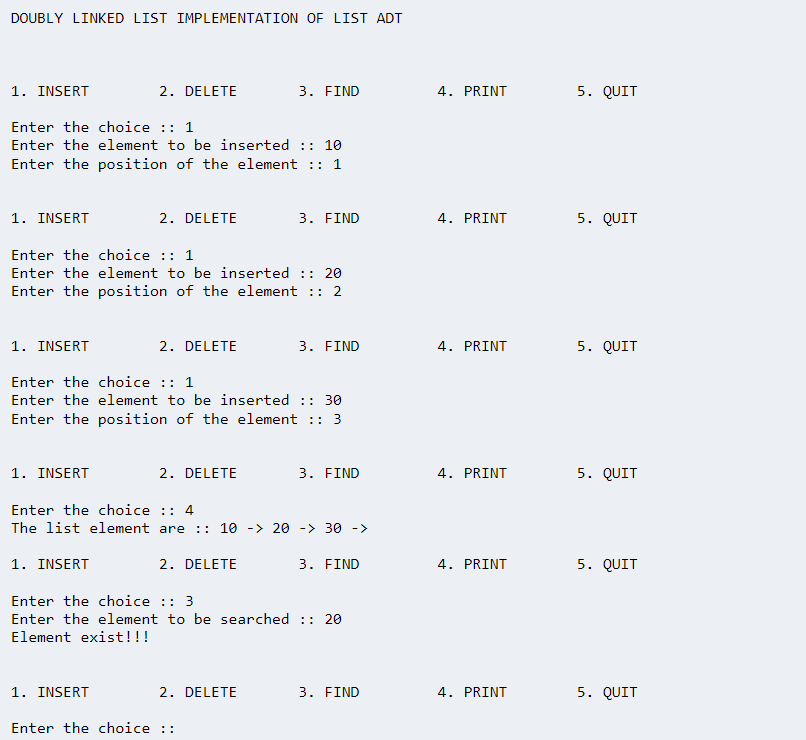
**}**

**}**

**while(ch<5);**

**}**

**OUTPUT:**

****

[**https://www.programming9.com/programs/c-programs/227-c-program-to-implement-doubly-linked-list-operations**](https://www.programming9.com/programs/c-programs/227-c-program-to-implement-doubly-linked-list-operations)

**10. Write a C program for the representation of polynomials using circular linked list and for the addition of two such polynomials**

**11. Write a C program to create a binary search tree and for implementing the in order, Pre order, post order traversal using recursion.**

**Program:**

**#include <stdio.h>**

**struct tnode**

**{**

**int data;**

**struct tnode \*right;**

**struct tnode \*left;**

**};**

**struct tnode \*CreateBST(struct tnode \*, int);**

**void Inorder(struct tnode \*);**

**void Preorder(struct tnode \*);**

**void Postorder(struct tnode \*);**

**int main()**

**{**

**struct tnode \*root = NULL;**

**int choice, item, n, i;**

**do**

**{**

**printf("\n\nBinary Search Tree Operations\n");**

**printf("\n1. Creation of BST");**

**printf("\n2. Traverse in Inorder");**

**printf("\n3. Traverse in Preorder");**

**printf("\n4. Traverse in Postorder");**

**printf("\n5. Exit\n");**

**printf("\nEnter Choice : ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:**

**root = NULL;**

**printf("\n\nBST for How Many Nodes ? ");**

**scanf("%d",&n);**

**for(i = 1; i <= n; i++)**

**{**

**printf("\nEnter data for node %d : ", i);**

**scanf("%d",&item);**

**root = CreateBST(root,item);**

**}**

**printf("\nBST with %d nodes is ready to Use!!\n", n);**

**break;**

**case 2:**

**printf("\nBST Traversal in INORDER \n");**

**Inorder(root);**

**break;**

**case 3:**

**printf("\nBST Traversal in PREORDER \n");**

**Preorder(root);**

**break;**

**case 4:**

**printf("\nBST Traversal in POSTORDER \n");**

**Postorder(root);**

**break;**

**case 5:**

**printf("\n\n Terminating \n\n");**

**break;**

**default:**

**printf("\n\nInvalid Option !!! Try Again !! \n\n");**

**break;**

**}**

**} while(choice != 5);**

**return 0;**

**}**

**struct tnode \*CreateBST(struct tnode \*root, int item)**

**{**

**if(root == NULL)**

**{**

**root = (struct tnode \*)malloc(sizeof(struct tnode));**

**root->left = root->right = NULL;**

**root->data = item;**

**return root;**

**}**

**else**

**{**

**if(item < root->data )**

**root->left = CreateBST(root->left,item);**

**else if(item > root->data )**

**root->right = CreateBST(root->right,item);**

**else**

**printf(" Duplicate Element !! Not Allowed !!!");**

**return(root);**

**}**

**}**

**void Inorder(struct tnode \*root)**

**{**

**if( root != NULL)**

**{**

**Inorder(root->left);**

**printf(" %d ",root->data);**

**Inorder(root->right);**

**}**

**}**

**void Preorder(struct tnode \*root)**

**{**

**if( root != NULL)**

**{**

**printf(" %d ",root->data);**

**Preorder(root->left);**

**Preorder(root->right);**

**}**

**}**

**void Postorder(struct tnode \*root)**

**{**

**if( root != NULL)**

**{**

**Postorder(root->left);**

**Postorder(root->right);**

**printf(" %d ",root->data);**

**}**

**}**

**OUTPUT:**

**Binary Search Tree Operations**

**1. Creation of BST**

**2. Traverse in Inorder**

**3. Traverse in Preorder**

**4. Traverse in Postorder**

**5. Exit**

**Enter Choice : 1**

**BST for How Many Nodes ? 5**

**Enter data for node 1 : 10**

**Enter data for node 2 : 2**

**Enter data for node 3 : 8**

**Enter data for node 4 : 15**

**Enter data for node 5 : 6**

**BST with 5 nodes is ready to Use!!**

**Binary Search Tree Operations**

**1. Creation of BST**

**2. Traverse in Inorder**

**3. Traverse in Preorder**

**4. Traverse in Postorder**

**5. Exit**

**Enter Choice : 2**

**BST Traversal in INORDER**

**2 6 8 10 15**

**Binary Search Tree Operations**

**1. Creation of BST**

**2. Traverse in Inorder**

**3. Traverse in Preorder**

**4. Traverse in Postorder**

**5. Exit**

**Enter Choice : 3**

**BST Traversal in PREORDER**

**10 2 8 6 15**

**Binary Search Tree Operations**

**1. Creation of BST**

**2. Traverse in Inorder**

**3. Traverse in Preorder**

**4. Traverse in Postorder**

**5. Exit**

**Enter Choice : 4**

**BST Traversal in POSTORDER**

**6 8 2 15 10**

**Binary Search Tree Operations**

**1. Creation of BST**

**2. Traverse in Inorder**

**3. Traverse in Preorder**

**4. Traverse in Postorder**

**5. Exit**

**Enter Choice: 5**

**Terminating**

[**https://www.cprogrammingnotes.com/question/bst-recursion.html**](https://www.cprogrammingnotes.com/question/bst-recursion.html)

**12. a) Write a C program for finding the transitive closure of a digraph.**

**Program:**

**#include<stdio.h>**

**#include<conio.h>**

**#include<math.h>**

**int max(int, int);**

**void warshal(int p[10][10], int n) {**

**int i, j, k;**

**for (k = 1; k <= n; k++)**

**for (i = 1; i <= n; i++)**

**for (j = 1; j <= n; j++)**

**p[i][j] = max(p[i][j], p[i][k] && p[k][j]);**

**}**

**int max(int a, int b) {**

**;**

**if (a > b)**

**return (a);**

**else**

**return (b);**

**}**

**void main() {**

**int p[10][10] = { 0 }, n, e, u, v, i, j;**

**printf("\n Enter the number of vertices:");**

**scanf("%d", &n);**

**printf("\n Enter the number of edges:");**

**scanf("%d", &e);**

**for (i = 1; i <= e; i++) {**

***//printf("\n Enter the end vertices of edge %d:", i);***

**scanf("%d%d", &u, &v);**

**p[u][v] = 1;**

**}**

**printf("\n Matrix of input data: \n");**

**for (i = 1; i <= n; i++) {**

**for (j = 1; j <= n; j++)**

**printf("%d\t", p[i][j]);**

**printf("\n");**

**}**

**warshal(p, n);**

**printf("\n Transitive closure: \n");**

**for (i = 1; i <= n; i++) {**

**for (j = 1; j <= n; j++)**

**printf("%d\t", p[i][j]);**

**printf("\n");**

**}**

**getch();**

**}**

**OUTPUT:**

**Enter the number of vertices: 5**

**Enter the number of edges: 11**

**1 1**

**1 4**

**3 2**

**3 3**

**3 4**

**4 2**

**4 4**

**5 2**

**5 3**

**5 4**

**5 5**

**Matrix of input data:**

**1 0 0 1 0**

**0 0 0 0 0**

**0 1 1 1 0**

**0 1 0 1 0**

**0 1 1 1 1**

**Transitive closure:**

**1 1 0 1 0**

**0 0 0 0 0**

**0 1 1 1 0**

**0 1 0 1 0**

**0 1 1 1 1**

[**https://www.sanfoundry.com/c-program-construct-transitive-closure-warshalls-algorithm/**](https://www.sanfoundry.com/c-program-construct-transitive-closure-warshalls-algorithm/)

[**https://ghrtutorials9.blogspot.com/2014/05/c-program-to-compute-transitive-closure.html**](https://ghrtutorials9.blogspot.com/2014/05/c-program-to-compute-transitive-closure.html)

**b) Write a C program for finding the shortest path from a given source to any vertex in a digraph using Dijkstra’s algorithm.**

**Program:**

**#include <stdio.h>**

**#include <limits.h>**

***// Number of vertices in the graph***

**#define V 9**

***// A utility function to find the vertex with minimum distance value, from***

***// the set of vertices not yet included in shortest path tree***

**int minDistance(int dist[], int sptSet[]) {**

***// Initialize min value***

**int min = INT\_MAX, min\_index;**

**int v;**

**for (v = 0; v < V; v++)**

**if (sptSet[v] == 0 && dist[v] <= min)**

**min = dist[v], min\_index = v;**

**return min\_index;**

**}**

***// A utility function to print the constructed distance array***

**void printSolution(int dist[], int n) {**

**printf("Vertex Distance from Source\n");**

**int i;**

**for (i = 0; i < V; i++)**

**printf("%d \t\t %d\n", i, dist[i]);**

**}**

***// Funtion that implements Dijkstra's single source shortest path algorithm***

***// for a graph represented using adjacency matrix representation***

**void dijkstra(int graph[V][V], int src) {**

**int dist[V]; *// The output array. dist[i] will hold the shortest***

***// distance from src to i***

**int sptSet[V]; *// sptSet[i] will 1 if vertex i is included in shortest***

***// path tree or shortest distance from src to i is finalized***

***// Initialize all distances as INFINITE and stpSet[] as 0***

**int i, count, v;**

**for (i = 0; i < V; i++)**

**dist[i] = INT\_MAX, sptSet[i] = 0;**

***// Distance of source vertex from itself is always 0***

**dist[src] = 0;**

***// Find shortest path for all vertices***

**for (count = 0; count < V - 1; count++) {**

***// Pick the minimum distance vertex from the set of vertices not***

***// yet processed. u is always equal to src in first iteration.***

**int u = minDistance(dist, sptSet);**

***// Mark the picked vertex as processed***

**sptSet[u] = 1;**

***// Update dist value of the adjacent vertices of the picked vertex.***

**for (v = 0; v < V; v++)**

***// Update dist[v] only if is not in sptSet, there is an edge from***

***// u to v, and total weight of path from src to v through u is***

***// smaller than current value of dist[v]***

**if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u]**

**+ graph[u][v] < dist[v])**

**dist[v] = dist[u] + graph[u][v];**

**}**

***// print the constructed distance array***

**printSolution(dist, V);**

**}**

***// driver program to test above function***

**int main() {**

***/\* Let us create the example graph discussed above \*/***

**int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},**

**{4, 0, 8, 0, 0, 0, 0, 11, 0},**

**{0, 8, 0, 7, 0, 4, 0, 0, 2},**

**{0, 0, 7, 0, 9, 14, 0, 0, 0},**

**{0, 0, 0, 9, 0, 10, 0, 0, 0},**

**{0, 0, 4, 0, 10, 0, 2, 0, 0},**

**{0, 0, 0, 14, 0, 2, 0, 1, 6},**

**{8, 11, 0, 0, 0, 0, 1, 0, 7},**

**{0, 0, 2, 0, 0, 0, 6, 7, 0}**

**};**

**dijkstra(graph, 0);**

**return 0;**

**}**

[**https://www.sanfoundry.com/c-program-find-shortest-path-between-two-vertices-using-dijkstras-algorithm/**](https://www.sanfoundry.com/c-program-find-shortest-path-between-two-vertices-using-dijkstras-algorithm/)

**13. a) Write a C program for finding the Depth First Search of a graph.**

**Program:**

**// DFS algorithm in C**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int vertex;**

**struct node\* next;**

**};**

**struct node\* createNode(int v);**

**struct Graph {**

**int numVertices;**

**int\* visited;**

**// We need int\*\* to store a two dimensional array.**

**// Similary, we need struct node\*\* to store an array of Linked lists**

**struct node\*\* adjLists;**

**};**

**// DFS algo**

**void DFS(struct Graph\* graph, int vertex) {**

**struct node\* adjList = graph->adjLists[vertex];**

**struct node\* temp = adjList;**

**graph->visited[vertex] = 1;**

**printf("Visited %d \n", vertex);**

**while (temp != NULL) {**

**int connectedVertex = temp->vertex;**

**if (graph->visited[connectedVertex] == 0) {**

**DFS(graph, connectedVertex);**

**}**

**temp = temp->next;**

**}**

**}**

**// Create a node**

**struct node\* createNode(int v) {**

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

**newNode->vertex = v;**

**newNode->next = NULL;**

**return newNode;**

**}**

**// Create graph**

**struct Graph\* createGraph(int vertices) {**

**struct Graph\* graph = malloc(sizeof(struct Graph));**

**graph->numVertices = vertices;**

**graph->adjLists = malloc(vertices \* sizeof(struct node\*));**

**graph->visited = malloc(vertices \* sizeof(int));**

**int i;**

**for (i = 0; i < vertices; i++) {**

**graph->adjLists[i] = NULL;**

**graph->visited[i] = 0;**

**}**

**return graph;**

**}**

**// Add edge**

**void addEdge(struct Graph\* graph, int src, int dest) {**

**// Add edge from src to dest**

**struct node\* newNode = createNode(dest);**

**newNode->next = graph->adjLists[src];**

**graph->adjLists[src] = newNode;**

**// Add edge from dest to src**

**newNode = createNode(src);**

**newNode->next = graph->adjLists[dest];**

**graph->adjLists[dest] = newNode;**

**}**

**// Print the graph**

**void printGraph(struct Graph\* graph) {**

**int v;**

**for (v = 0; v < graph->numVertices; v++) {**

**struct node\* temp = graph->adjLists[v];**

**printf("\n Adjacency list of vertex %d\n ", v);**

**while (temp) {**

**printf("%d -> ", temp->vertex);**

**temp = temp->next;**

**}**

**printf("\n");**

**}**

**}**

**int main() {**

**struct Graph\* graph = createGraph(4);**

**addEdge(graph, 0, 1);**

**addEdge(graph, 0, 2);**

**addEdge(graph, 1, 2);**

**addEdge(graph, 2, 3);**

**printGraph(graph);**

**DFS(graph, 2);**

**return 0;**

**}**

**OUTPUT:**

**Adjacency list of vertex 0**

**2 -> 1 ->**

**Adjacency list of vertex 1**

**2 -> 0 ->**

**Adjacency list of vertex 2**

**3 -> 1 -> 0 ->**

**Adjacency list of vertex 3**

**2 ->**

**Visited 2**

**Visited 3**

**Visited 1**

**Visited 0**

[**https://www.programiz.com/dsa/graph-dfs**](https://www.programiz.com/dsa/graph-dfs)

**b) Write a C program for finding the Breadth First Search of a graph.**

**Program:**

**// BFS algorithm in C**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define SIZE 40**

**struct queue {**

**int items[SIZE];**

**int front;**

**int rear;**

**};**

**struct queue\* createQueue();**

**void enqueue(struct queue\* q, int);**

**int dequeue(struct queue\* q);**

**void display(struct queue\* q);**

**int isEmpty(struct queue\* q);**

**void printQueue(struct queue\* q);**

**struct node {**

**int vertex;**

**struct node\* next;**

**};**

**struct node\* createNode(int);**

**struct Graph {**

**int numVertices;**

**struct node\*\* adjLists;**

**int\* visited;**

**};**

**// BFS algorithm**

**void bfs(struct Graph\* graph, int startVertex) {**

**struct queue\* q = createQueue();**

**graph->visited[startVertex] = 1;**

**enqueue(q, startVertex);**

**while (!isEmpty(q)) {**

**printQueue(q);**

**int currentVertex = dequeue(q);**

**printf("Visited %d\n", currentVertex);**

**struct node\* temp = graph->adjLists[currentVertex];**

**while (temp) {**

**int adjVertex = temp->vertex;**

**if (graph->visited[adjVertex] == 0) {**

**graph->visited[adjVertex] = 1;**

**enqueue(q, adjVertex);**

**}**

**temp = temp->next;**

**}**

**}**

**}**

**// Creating a node**

**struct node\* createNode(int v) {**

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

**newNode->vertex = v;**

**newNode->next = NULL;**

**return newNode;**

**}**

**// Creating a graph**

**struct Graph\* createGraph(int vertices) {**

**struct Graph\* graph = malloc(sizeof(struct Graph));**

**graph->numVertices = vertices;**

**graph->adjLists = malloc(vertices \* sizeof(struct node\*));**

**graph->visited = malloc(vertices \* sizeof(int));**

**int i;**

**for (i = 0; i < vertices; i++) {**

**graph->adjLists[i] = NULL;**

**graph->visited[i] = 0;**

**}**

**return graph;**

**}**

**// Add edge**

**void addEdge(struct Graph\* graph, int src, int dest) {**

**// Add edge from src to dest**

**struct node\* newNode = createNode(dest);**

**newNode->next = graph->adjLists[src];**

**graph->adjLists[src] = newNode;**

**// Add edge from dest to src**

**newNode = createNode(src);**

**newNode->next = graph->adjLists[dest];**

**graph->adjLists[dest] = newNode;**

**}**

**// Create a queue**

**struct queue\* createQueue() {**

**struct queue\* q = malloc(sizeof(struct queue));**

**q->front = -1;**

**q->rear = -1;**

**return q;**

**}**

**// Check if the queue is empty**

**int isEmpty(struct queue\* q) {**

**if (q->rear == -1)**

**return 1;**

**else**

**return 0;**

**}**

**// Adding elements into queue**

**void enqueue(struct queue\* q, int value) {**

**if (q->rear == SIZE - 1)**

**printf("\nQueue is Full!!");**

**else {**

**if (q->front == -1)**

**q->front = 0;**

**q->rear++;**

**q->items[q->rear] = value;**

**}**

**}**

**// Removing elements from queue**

**int dequeue(struct queue\* q) {**

**int item;**

**if (isEmpty(q)) {**

**printf("Queue is empty");**

**item = -1;**

**} else {**

**item = q->items[q->front];**

**q->front++;**

**if (q->front > q->rear) {**

**printf("Resetting queue ");**

**q->front = q->rear = -1;**

**}**

**}**

**return item;**

**}**

**// Print the queue**

**void printQueue(struct queue\* q) {**

**int i = q->front;**

**if (isEmpty(q)) {**

**printf("Queue is empty");**

**} else {**

**printf("\nQueue contains \n");**

**for (i = q->front; i < q->rear + 1; i++) {**

**printf("%d ", q->items[i]);**

**}**

**}**

**}**

**int main() {**

**struct Graph\* graph = createGraph(6);**

**addEdge(graph, 0, 1);**

**addEdge(graph, 0, 2);**

**addEdge(graph, 1, 2);**

**addEdge(graph, 1, 4);**

**addEdge(graph, 1, 3);**

**addEdge(graph, 2, 4);**

**addEdge(graph, 3, 4);**

**bfs(graph, 0);**

**return 0;**

**}**

**OUTPUT:**

**Queue contains**

**0 Resetting queue Visited 0**

**Queue contains**

**2 1 Visited 2**

**Queue contains**

**1 4 Visited 1**

**Queue contains**

**4 3 Visited 4**

**Queue contains**

**3 Resetting queue Visited 3**

[**https://www.programiz.com/c-programming/online-compiler/**](https://www.programiz.com/c-programming/online-compiler/)