DAYANANDA SAGAR COLLEGE OF ENGINEERING
An Autonomous Institute Affiliated to VTU, Belagavi Approved by AICTE; ISO 9001:2015
Certified Accredited by National Assessment Accreditation Council (NAAC) with 'A' grade
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

22CS33 - DATA STRUCTURES AND ITS APPLICATIONS LABORATORY MANUAL

Objectives:

1. Introduce the concept of data structures through ADT including List, Stack, Queues
2. To design and implement various data structure algorithms.
3. To introduce various techniques for representation of the data in the real world
4. To develop application using data structure algorithms

Outcomes: At the end of the course, student will be able to:

CO1	Select appropriate data structures to be used in the programming solution for a given problem.
CO2	Implement operations like searching, insertion, and deletion, traversing mechanism etc. on various data structures.
CO3	Implement Linear and Non-Linear data structures efficiently.
CO4	Implement various operations on Linked Lists
CO5	Design and Implement applications of Non-linear data structure.
CO6	Implement graph traversal algorithms

The student should be able to:

1	Analyze and Compare various linear and non-linear data structures
2	Code, debug and demonstrate the working nature of different types of data structures and their applications
3	Implement, analyze and evaluate the searching and sorting algorithms
4	Choose the appropriate data structure for solving real world problems

Laboratory exercises

Q.no	Problem Statement
1.	Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit. Support the program with appropriate functions for each of the above operations
2	Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.
3	Design, Develop and Implement a Program in C for the following Stack Applications. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
4.	Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE

	d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations
5.	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List of Student Data with the fields: USN, Name, Programme, Sem, PhNo a. Create a SLL of N Students Data by using front insertion. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion/Deletion at End of SLL d. Perform Insertion/Deletion at Front of SLL e. Exit
6.	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo a. Create a DLL of N Employees Data by using end insertion. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit
7.	Design, Develop and Implement Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z - 4yz^5+3x^3yz+2xy^5z- 2xyz^3$ b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
8.	Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers. a. Create a BST of N Integers b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit
9.	Design, Develop and Implement a Program in C for the operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all nodes reachable from a given source node in a graph using DFS/BFS method
10.	Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K→L as H (K) =K mod m, and implement hashing technique to map a given key K to the address space L. Resolve the collision if any using linear probing.

Integrated Laboratory Pre-Requisites:

1.	Certification: Pointers In C Programming
2.	Course Time: 01 hr 45 mins, Advanced level. Student can earn a certificate on completing the content https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013842032404848640104 70_shared/overview

Evaluation Scheme [Integrated]

Theory				
Sl.no	**Particulars**	**Max. Marks**	**Scale down to 10**	**Max. Marks**
a.	Continuous Internal Assessment -1	30	Average of 03 Assessments	10
b.	Continuous Internal Assessment -2	30		
c.	Continuous Internal Assessment -3	30		
d.	Assignment [Problem Based]	10	10	10
e.	Alternative Assessment	30	10	10

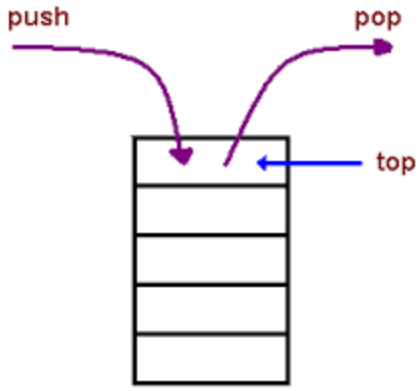
			Total	**30**
			Minimum Requirement	**12**
2. Laboratory				
Sl.no	**Particulars**	**Max. Marks**	Scale down to 20	**Max. Marks**
a.	Continuous Internal Evaluation	30		
b.	Laboratory Internal Examination	50		**20**
			Minimum requirement	**08**
NOTE: The student is not allowed take semester end examination, if fails to score minimum requirement marks either in theory or laboratory				

LAB MANUAL

1.Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate how Stack can be used to check Palindrome
d. Demonstrate Overflow and Underflow situations on Stack
e. Display the status of Stack
f. Exit
Support the program with appropriate functions for each of the above operations

Description:

A **stack** is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown **stacks** only two operations are allowed: push the item into the **stack**, and pop the item out of the **stack**.



Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- **push()** − pushing (storing) an element on the stack.
- **pop()** − removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks −

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

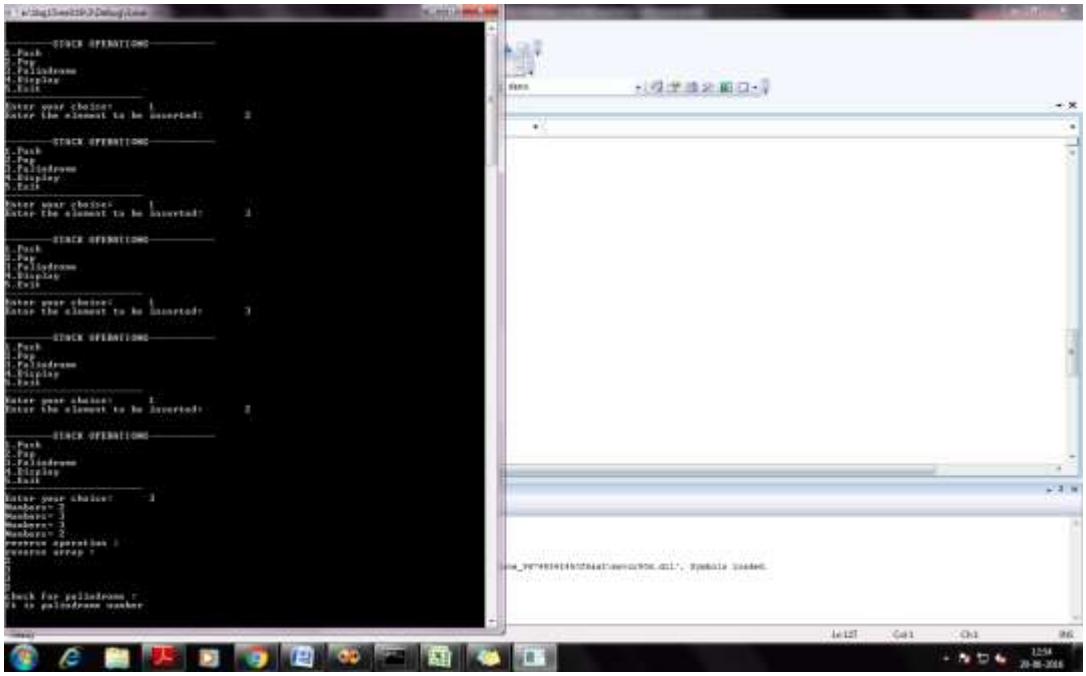
At all times, we maintain a pointer to the last Pushed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

Program:

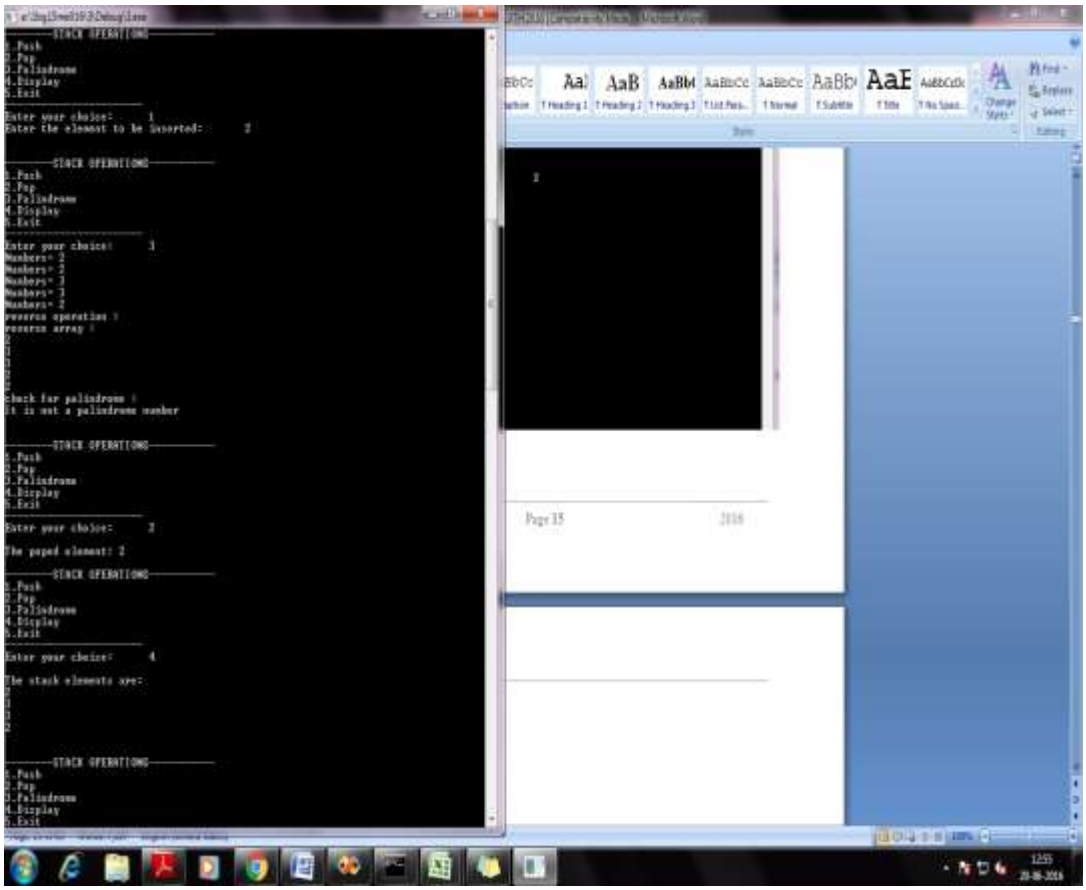
```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#define max_size 5
int stack[max_size],top=-1;
int a[10],n;
void push() //Inserting element into the stack
{
int item;
if(top==(max_size-1))
printf("\nStack Overflow:");
else
{
printf("Enter the element to be inserted:\t");
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
}
void pop() //deleting an element from the stack
{
int item;

if(top==-1)
printf("Stack Underflow:");
else
{
item=stack[top];
top=top-1;
printf("\nThe poped element: %d\t",item);
}
}
void display()              //Display function
{
int i;
if(top == -1)
printf("Stack Empty\n");
else
{
printf("Elements Are:\n");
for(i=top;i>=0;i--)
          printf("%d\n",stack[i]);
}
}
void palindrome()                   //Palindrome
{
int k,len=top+1,rev[max_size],i,j=0;
if(top == -1)
printf("Stack Empty\n");
else
```

```
{
printf("Array elements are : \n");       //displaying array elements
for(i=top;i>=0;i--)
        printf("%d\t",stack[i]);
for(k=len-1;k>=0;k--,j++)      //performing reverse operation
        rev[k]=stack[j];
printf("\n reverse array : \n");
for(k=0;k<len;k++)
        printf("%d\t",rev[k]);
for(i=0;i<len;i++)                //check for palindrome
        if(stack[i]!=rev[i])
                break;
if(i==len)
        printf("\n It is palindrome number\n");
else
        printf("\n It is not a palindrome number\n");
}
getch();
}
int main()
{
int choice;
while(1)
{
printf("\n\n--------STACK OPERATIONS----------\n");
printf("1.Push\n");
printf("2.Pop\n");
printf("3.Palindrome\n");
printf("4.Display\n");
printf("5.Exit\n");
printf("----------------------");
printf("\nEnter your choice:\t");
scanf("%d",&choice);
switch(choice)
{       case 1:push(); break;
        case 2:pop();  break;
        case 3:palindrome(); break;
        case 4:display(); break;
        case 5:exit(0);
        default:printf("\nInvalid choice:\n");
}
}
}
```

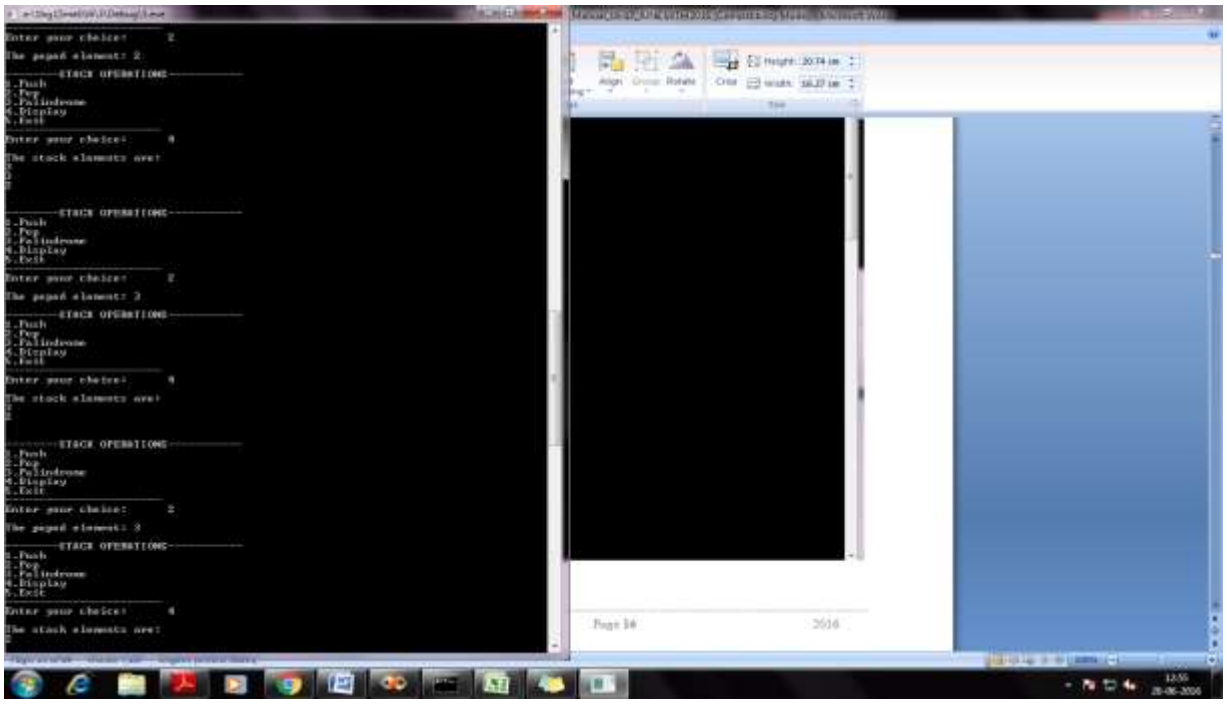
Output:

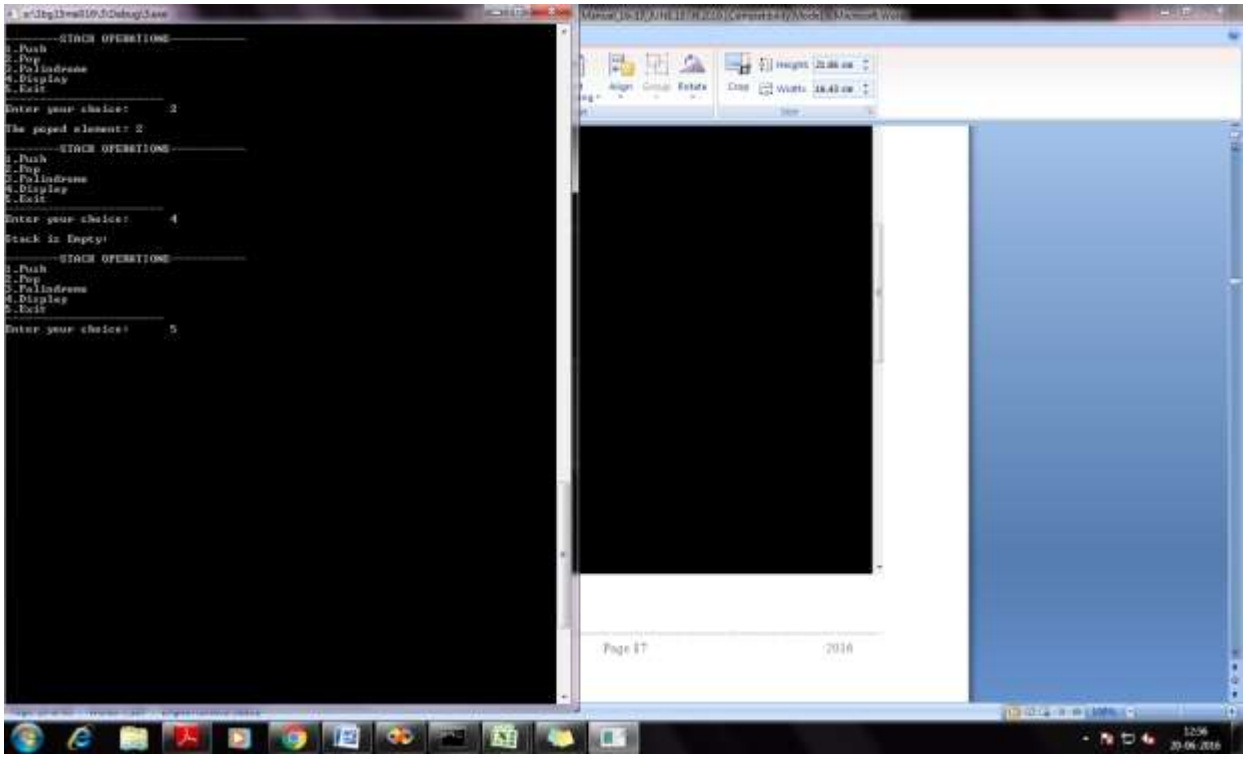


```
--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      1
Enter the element to be inserted:       2

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      1
Enter the element to be inserted:       1

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      1
Enter the element to be inserted:       3

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      1
Enter the element to be inserted:       2

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      3
Numbers= 2
Numbers= 3
Numbers= 1
Numbers= 3
Numbers= 2
reverse operation :
reverse array :
2
3
1
3
2
Check for palindrome :
It is palindrome number
```



```
--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      1
Enter the element to be inserted:       2

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      1
Numbers= 2
Numbers= 3
Numbers= 1
Numbers= 3
Numbers= 2
reverse operation :
reverse array :
2
3
1
3
2
Check for palindrome :
It is not a palindrome number

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      2

The poped element: 2

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
-----------------------
Enter your choice:      4

The stack elements are:
3
1
3
2

--------STACK OPERATIONS----------
1.Push
2.Pop
3.Palindrome
4.Display
5.Exit
```





1. **Viva Questions:**

1. What is stack and how to implement it?
2. What are the basic stack operations?
3. Which strategy Stack follows? (LIFO or FIFO)
4. What is the difference between a PUSH and a POP?
5. Why do we use Stacks?

2. Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, %(Remainder), ^(Power) and alphanumeric operands.

Description:

Steps to be followed in conversion of Infix to Postfix expression.

1. Print operands as they arrive.

2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

3. If the incoming symbol is a left parenthesis, push it on the stack.

4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.

6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.

7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.

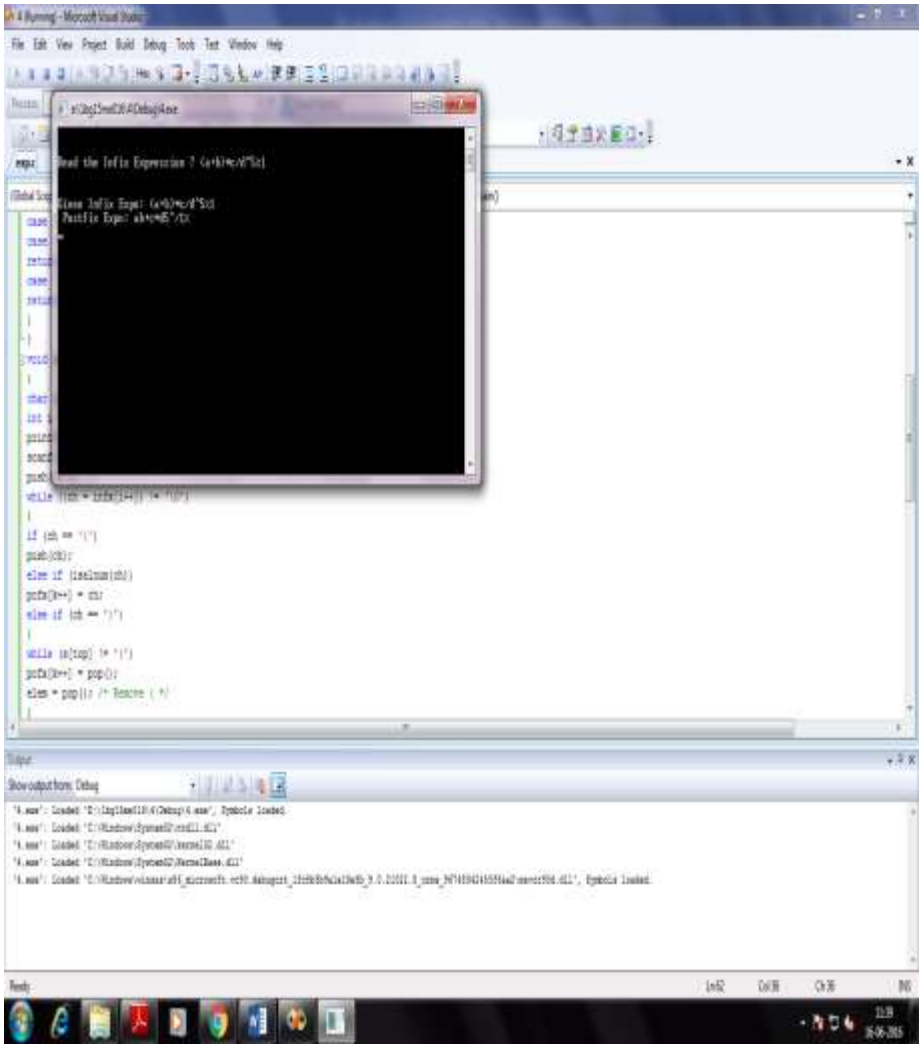
8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

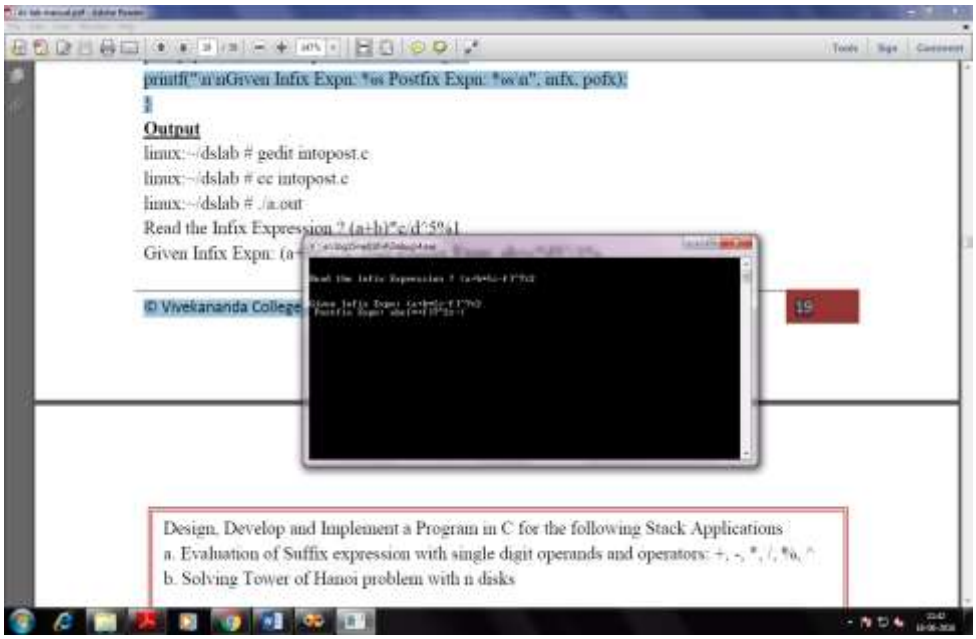
Program:

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50 /* Size of Stack */
char s[SIZE];
int top = -1; /* Global declarations */

push(char elem) /* Function for PUSH operation */
{
s[++top] = elem;
}

char pop() /* Function for POP operation */
{
return (s[top--]);
}

int pr(char elem) /* Function for precedence */
{
switch (elem)
{
case '#':
return 0;
case '(':
return 1;
```

```
case '+':
case '-':
return 2;
case '*':
case '/':
case '%':
return 3;
case '^':
return 4;
}
}

void main() /* Main Program */
{
char infx[50], pofx[50], ch, elem;
int i = 0, k = 0;
printf("\n\nRead the Infix Expression ? ");
scanf("%s", infx);
push('#');
while ((ch = infx[i++]) != '\0')
{
if (ch == '(')
push(ch);
else if (isalnum(ch))
pofx[k++] = ch;
else if (ch == ')')
{
while (s[top] != '(')
pofx[k++] = pop();
elem = pop(); /* Remove ( */
}
else /* Operator */
{
while (pr(s[top]) >= pr(ch))
pofx[k++] = pop();
push(ch);
}
}
while (s[top] != '#') /* Pop from stack till empty */
pofx[k++] = pop();
pofx[k] = '\0'; /* Make pofx as valid string */
printf("\n\nGiven Infix Expn: %s \n Postfix Expn: %s\n", infx, pofx);
getch();
}
```

Output:





Viva Questions:

1. What is Reversed Polish Notation?
2. Differentiate infix, postfix and prefix expressions.
3. How to evaluate postfix expression using stack.
4. Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.

ANS: Prefix Notation: - * +ABC ^ - DE + FG
Postfix Notation: AB + C * DE - FG + ^ -

5. Use the conversion algorithm to change the following infix expression into postfix.

- (A * B - (C - D)) / (E + F)
- A/B^C+D*E-A*C
- (B^2 – 4 * A * C)^(1/2)
- (4+8)*(6-5)/((3-2)*(2+2))

3. Design, Develop and Implement a Program in C for the following Stack Applications
Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

Description:
The changes to the position of the operator with respect to the operands create two new expression formats, **prefix** and **postfix**. Prefix expression notation requires that all operators precede the two operands that they work on. Postfix, on the other hand, requires that its operators come after the corresponding operands.

Table : Examples of Infix, Prefix, and Postfix

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

Table :An Expression with Parentheses

Infix Expression	Prefix Expression	Postfix Expression
(A + B) * C	* + A B C	A B + C *

Table : Additional Examples of Infix, Prefix, and Postfix

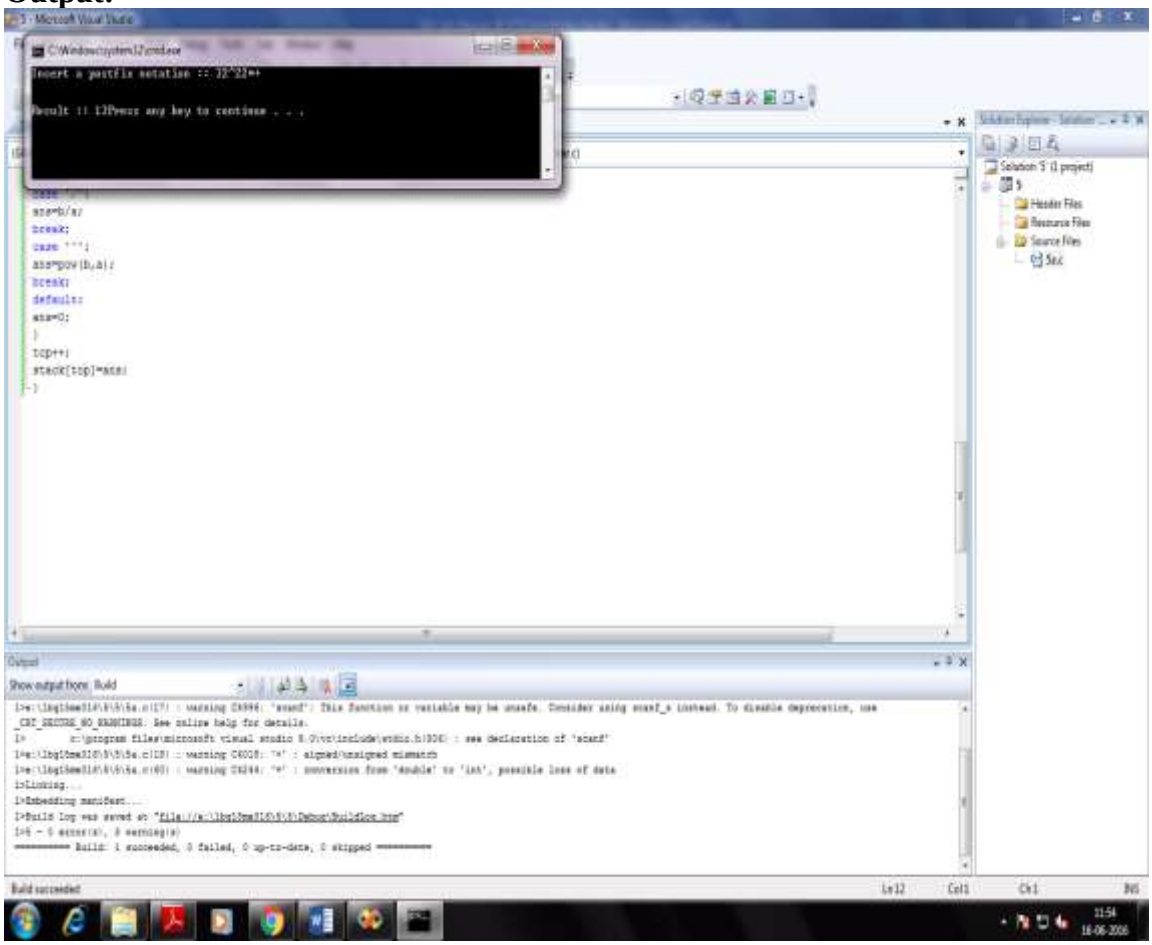
Infix Expression	Prefix Expression	Postfix Expression
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Program:

```
// Evaluation of Suffix Expression
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#define MAX 50
int stack[MAX];
char post[MAX];
int top=-1;
void pushstack(int tmp);
void calculator(char c);
void main()
{
int i;
printf("Insert a postfix notation :: ");
//gets(post);
scanf("%s",post);
for(i=0;i<strlen(post);i++)
{
if(post[i]>='0' && post[i]<='9')
{
pushstack(i);
}
if(post[i]=='+' || post[i]=='-' || post[i]=='*' || post[i]=='/' || post[i]=='^')
{
calculator(post[i]);
}
}
printf("\n\nResult :: %d",stack[top]);
}
void pushstack(int tmp)
{
top++;
stack[top]=(int)(post[tmp]-48);
}
void calculator(char c)
{
int a,b,ans;
a=stack[top];
stack[top]='\0';
top--;
b=stack[top];
stack[top]='\0';
top--;
switch(c)
{
case '+': ans=b+a;
break;
case '-':
ans=b-a;
break;
case '*':
ans=b*a;
```

```
break;
case '/':
ans=b/a;
break;
case '^':
ans=pow(b,a);
break;
default: ans=0;
}
top++;
stack[top]=ans;
getch ();
}
```

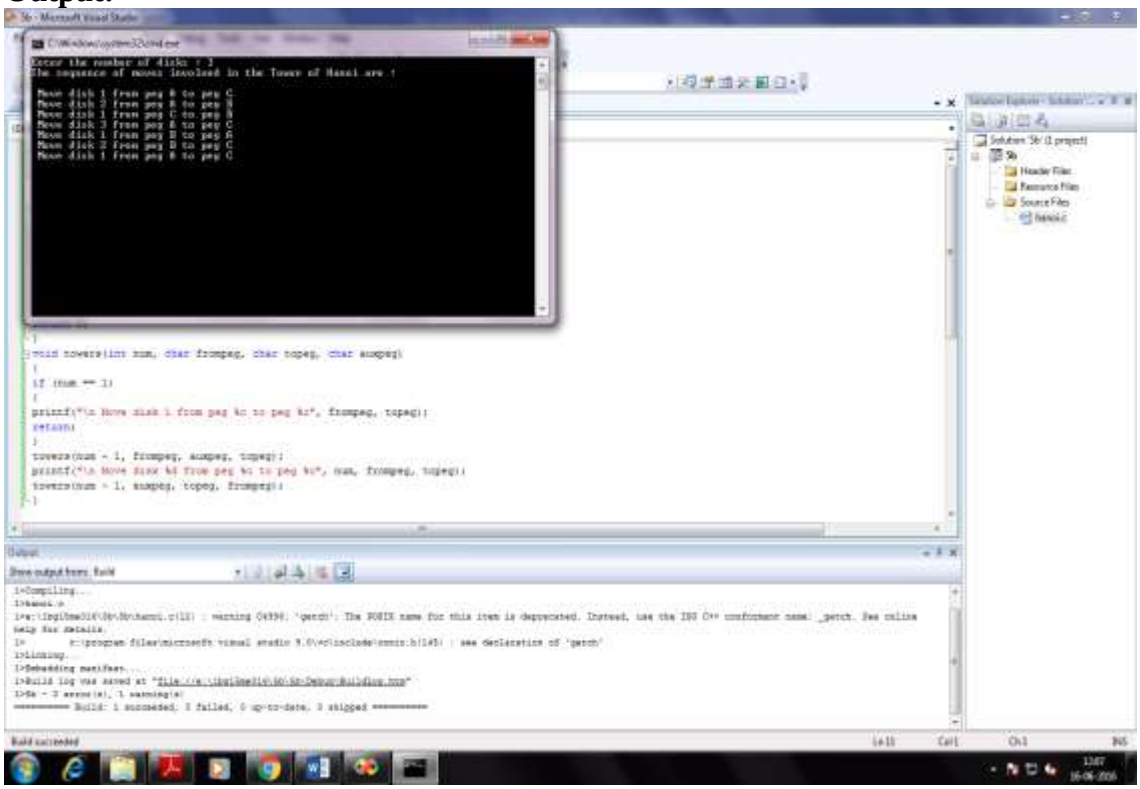
Output:



```
}
towers(num - 1, frompeg, auxpeg, topeg);
printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
towers(num - 1, auxpeg, topeg, frompeg);
}
int main()
{
int num;
printf("Enter the number of disks : ");
scanf_s("%d", &num);
printf("The sequence of moves involved in the Tower of Hanoi are :\n");
towers(num, 'A', 'C', 'B');
getch();
```

return 0;
}

Output:



4. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE e. Exit
Support the program with appropriate functions for each of the above operations

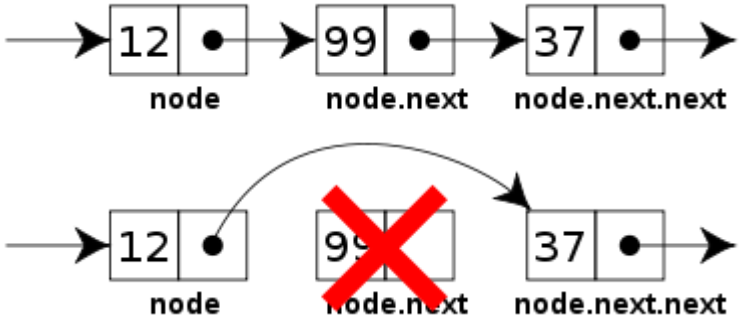
```
#include<stdio.h>
#include <stdlib.h>
#define sz 5
int i;
void enqueue(int q[sz], int *f, int *r, int item)
{
if(*f == -1 && *r ==-1)
{
  *f = *r = 0;
  q[*r] = item;
}
else if(((*r+1)%sz)== *f)
{
        printf("Queue is full.Enqueue not possible\n");
}
else
{
```

```
        *r = (*r+1)%sz;
        q[*r] = item;
}
}
void dequeue(int q[sz], int *r, int *f)
{
if( *f == -1 && *r == -1)
printf("Queue is empty.Dequeue not possible\n");
else if(*f == *r)
{
        *f = *r = -1;
}
else
{
printf("the item dequeued is %d\n", q[*f] );
*f = (*f +1)%sz;
}
}
void display(int q[sz], int *r, int *f)
{
int i = *f;
if( *f == -1 && *r == -1)
printf("Queue is empty.Nothing to display\n");
else
{
        printf("The elements in the queue are\n");
        while(i != *r)
        {
        printf("Queue[%d] ======> %d\n",i,q[i] );
        i = (i+1)%sz;
        }
        printf("Queue[%d] ======> %d\n",i, q[*r]);
}
}
void main()
{
int q[sz], r=-1, f= -1, item, ch;
for(;;)
{
        printf("Enter the option for circular queue 1: Enqueue 2:Dequeue 3:Display\n");
        scanf("%d", &ch);
        switch(ch)
        {
                case 1: printf("Enter the element you want to enqueue\n");
                                scanf("%d", &item);
                                enqueue(q,&f,&r, item);break;
                case 2: dequeue(q,&r,&f);
                                break;
                case 3: display(q,&r,&f);
                                break;
                default: exit(0);
        }
}
}
```

5. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo

a. Create a SLL of N Students Data by using front insertion.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of SLL
d. Perform Insertion and Deletion at Front of SLL
e. Demonstrate how this SLL can be used as STACK and QUEUE
f. Exit

Description:
Many programming languages such as Lisp and Scheme have **singly linked lists** built in. In many functional languages, these **lists** are constructed from nodes, each called a cons or cons cell. The cons have two fields: the car, a reference to the data for that node, and the cdr, a reference to the next node.



Basic operations of a singly-linked list are:
Insert – Inserts a new element at the end of the list.
Delete – Deletes any node from the list.
Find – Finds any node in the list.
Print – Prints the list.
Functions
1. Insert – This function takes the start node and data to be inserted as arguments. New node is inserted at the end so, iterate through the list till we encounter the last node. Then, allocate memory for the new node and put data in it. Lastly, store the address in the next field of the new node as NULL.

2. Delete - This function takes the start node (as pointer) and data to be deleted as arguments. Firstly, go to the node for which the node next to it has to be deleted, If that node points to NULL (i.e. pointer->next=NULL) then the element to be deleted is not present in the list. Else, now pointer points to a node and the node next to it has to be removed, declare a temporary node (temp) which points to the node which has to be removed. Store the address of the node next to the temporary node in the next
field of the node pointer (pointer->next = temp->next). Thus, by breaking the link we removed the node which is next to the pointer (which is also temp). Because we deleted the node, we no longer require the memory used for it, free() will deallocate the memory.

3. Find - This function takes the start node (as pointer) and data value of the node (key) to be found as arguments. First node is dummy node so, start with the second node. Iterate through the entire linked list and search for the key. Until next field of the pointer is equal to NULL, check if pointer->data = key. If it is then the key is found else, move to the next node and search (pointer = pointer -> next). If key is not found return 0, else return 1.
4. Print - function takes the start node (as pointer) as an argument. If pointer = NULL, then there is no element in the list. Else, print the data value of the node (pointer->data) and move to the next node by recursively calling the print function with pointer->next sent as an argument.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
int count = 0;
struct node
{
char usn[10], name[10], branch[3];
int sem, ph;
struct node *next;
}*first = NULL, *temp = NULL, *temp1 = NULL, *temp2 = NULL;
void create()
{
int sem, ph;
char usn[10], name[10], branch[10];
temp = (struct node *)malloc(sizeof(struct node));
printf("Enter Student details: \n");
printf("Enter USN:");
scanf(" %s", usn);
printf("Enter Name:");
scanf("%s", name);
printf("Enter Branch:");
scanf("%s", branch);
printf("ENTER Semester:");
scanf("%d", &sem);
printf("ENTER Phone no:");
scanf("%d", &ph);
strcpy(temp->usn, usn);
strcpy(temp->branch, branch);
strcpy(temp->name, name);
temp->sem = sem;
temp->ph = ph;
temp->next = NULL;
count++;
}
void insertfront()
{
create();
if (first == NULL)
first = temp;
else
{
temp->next = first;
first = temp;
}
}
void insertlast()
{
create();
if (first == NULL)
{
first = temp;
}
```

```
else
{
temp1 = first;
while (temp1->next != NULL)
{
        temp1 = temp1->next;
}
temp1->next = temp;
}
}
void deletefront()
{
if (first == NULL)
{
printf("\nNo nodes in the list\n");
}
else
{
temp1 = first;
first = first->next;
free(temp1);
count--;
}
}
void deletelast()
{
if (first == NULL)
{
printf("\nNo nodes in the list\n");
}
else
{
temp1 = first;
while (temp1->next != NULL)
{
        temp2 = temp1;
        temp1 = temp1->next;
}
temp2->next = NULL;
free(temp1);
count--;
}
}
void display()
{
if (first == NULL)
{
printf("\nNo nodes to display\n");
}
else
{
temp1 = first;
printf("There are %d number of nodes in the list\n ", count);
printf("Student Details are:\n");
```

```
printf("USN \tNAME \tBRANCH \tSEM \tPHONE NUMBER\n");
printf("\n--------------------------------------------------\n");
while (temp1 != NULL)
{


        printf("%-3s\t %-10s %-3s\t %-3d %-10d\n", temp1->usn, temp1->name, temp1->branch, temp1->sem,
temp1->ph);
        temp1 = temp1->next;
}
}
}
void main()
{
int choice, m, i;
printf("\n-->MENU<--\n");
while (1)
{
printf("1.Create a SLL\n2.Display the records\n3.Insert at the last\n4.Delete at the last\n5.Insert at the
front\n6.Delete at the front\n7.Exit\n");
printf("Enter ur choice: ");
scanf("%d", &choice);
switch (choice)
{
case 1:printf("\nEnter the number of student record to be created : ");
        scanf("%d", &m);
        for (i = 0; i<m; i++)
        {
                insertfront();
        }
        break;
case 2: display();
        break;
case 3: insertlast();
        break;
case 4: deletelast();
        break;
case 5: insertfront();
        break;
case 6: deletefront();
        break;
case 7: exit(1);
default: printf("Invalid choice \n");
        break;
}
}
_getch();
}
```

Output:

```
-->MENU<--
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 1

Enter the number of student record to be created : 1
Enter Student details:
Enter USN:1bg16cs12
Enter Name:aaa
Enter Branch:cse
ENTER Semester:3
ENTER Phone no:12345
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 2
There are 1 number of nodes in the list
 Student Details are:
USN      NAME     BRANCH   SEM      PHONE NUMBER

-------------------------------------------------------
1bg16cs12         aaa          cse   3    12345
```

```
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 3
Enter Student details:
Enter USN:1bg16cs13
Enter Name:ssss
Enter Branch:cse
ENTER Semester:4
ENTER Phone no:123456
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 2
There are 2 number of nodes in the list
 Student Details are:
USN      NAME     BRANCH   SEM      PHONE NUMBER

-------------------------------------------------------
1bg16cs12         aaa          cse   3    12345
1bg16cs13         ssss         cse   4    123456
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 5
Enter Student details:
Enter USN:1bg16cs10
Enter Name:eeee
Enter Branch:cse
ENTER Semester:5
ENTER Phone no:123456
```

```
Enter ur choice: 2
There are 3 number of nodes in the list
 Student Details are:
USN     NAME    BRANCH  SEM     PHONE NUMBER

-------------------------------------------------
1bg16cs10       eeee       cse  5   123456
1bg16cs12       aaa        cse  3   12345
1bg16cs13       ssss       cse  4   123456
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 4
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 2
There are 2 number of nodes in the list
 Student Details are:
USN     NAME    BRANCH  SEM     PHONE NUMBER

-------------------------------------------------
1bg16cs10       eeee       cse  5   123456
1bg16cs12       aaa        cse  3   12345
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 6
1.Create a SLL
2.Display the records
3.Insert at the last
4.Delete at the last
5.Insert at the front
6.Delete at the front
7.Exit
Enter ur choice: 2
There are 1 number of nodes in the list
 Student Details are:
USN     NAME    BRANCH  SEM     PHONE NUMBER

-------------------------------------------------
1bg16cs12       aaa        cse  3   12345
```

Viva Questions:

1. What is a Linked list?
2. Whether Linked List is linear or Non-linear data structure?
3. Can you represent a Linked list graphically?
4. How many pointers are required to implement a simple Linked list?
5. How many types of Linked lists are there?
6. What are the applications that use Linked lists?
7. What does the following function do for a given Linked List ?

```
void fun2(struct node* head)
{
  if(head== NULL)
    return;
  printf("%d  ", head->data);
```

```
  if(head->next != NULL )
    fun2(head->next->next);
  printf("%d  ", head->data);
}
```

Ans: fun2 () prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has even number of nodes, then fun2() skips the last node.

For Linked List 1->2->3->4->5, fun2 () prints 1 3 5 5 3 1. For Linked List 1->2->3->4->5->6, fun2() prints 1 3 5 5 3 1.

8. Suppose cursor points to a node in a linked list (using the node definition with member functions called data and link). What statement changes cursor so that it points to the next node?

A. cursor++;
B. cursor = link( );
C. cursor += link( );
D. cursor = cursor->link( );

9. Suppose cursor points to a node in a linked list (using the node definition with member functions called data and link). What Boolean expression will be true when cursor points to the tail node of the list?

A. (cursor == NULL)
B. (cursor->link( ) == NULL)
C. (cursor->data( ) == NULL)
D. (cursor->data( ) == 0.0)
E. None of the above

10.Why does our node class have two versions of the link member function?

A. One is public, the other is private.
B. One is to use with a const pointer, the other with a regular pointer.
C. One returns the forward link, the other returns the backward link.
D. One returns the data, the other returns a pointer to the next node.

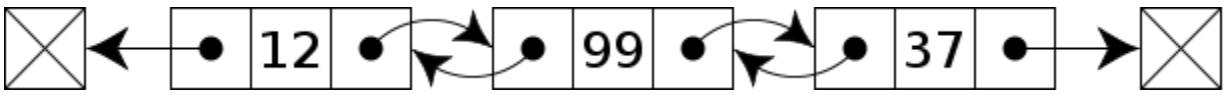
6. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo

a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue
f. Exit

Description:

A self referential data structure. A list of elements, with a head and a tail; each element points to another of its own kind in front of it, as well as another of its own kind, which happens to be behind it in the sequence.

The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.



Basic Operations

Following are the basic operations supported by a list.

Insertion − add an element at the beginning of the list.
Deletion − delete an element at the beginning of the list.
Insert Last − add an element in the end of the list.
Delete Last − delete an element from the end of the list.
Insert After − add an element after an item of the list.
Delete − delete an element from the list using key.
Display forward − displaying complete list in forward manner.
Display backward − displaying complete list in backward manner.

Program:

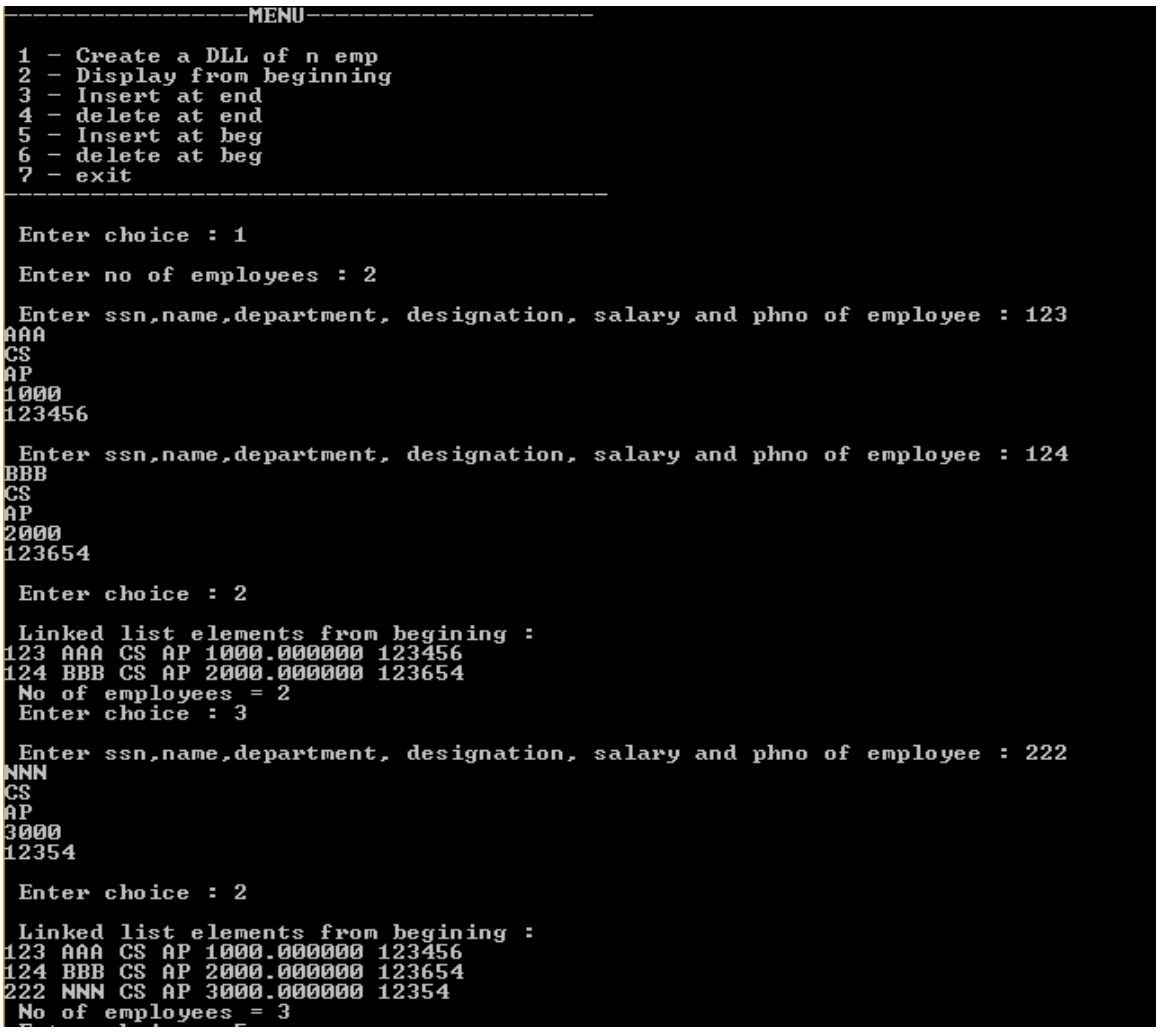
```
#include<stdio.h>
#include"stdafx.h"
#include<stdlib.h>
#include<string.h>
#include<conio.h>
int count = 0;
struct node
{
struct node *prev;
int ssn, phno;
float sal;
char name[20], dept[10], desg[20];
struct node *next;
}*h, *temp, *temp1, *temp2, *temp4;


void create()
{
int ssn, phno;
float sal; /.x
char name[20], dept[10], desg[20];
temp = (struct node *)malloc(sizeof(struct node));
temp->prev = NULL;
temp->next = NULL;
printf("\n Enter ssn,name,department, designation, salary and phno of employee : ");
scanf("%d %s %s %s %f %d", &ssn, name, dept, desg, &sal, &phno);
temp->ssn = ssn;
strcpy(temp->name, name);
strcpy(temp->dept, dept);
strcpy(temp->desg, desg);
temp->sal = sal;
temp->phno = phno;
count++;
}
void insertbeg()
{
if (h == NULL)
{
create();
```

```
h = temp;
temp1 = h;
}
else
{
create();
temp->next = h;
h->prev = temp;
h = temp;
}
}
void insertend()
{
if (h == NULL)
{
create();
h = temp;
temp1 = h;
}
else
{
create();
temp1->next = temp;
temp->prev = temp1;
temp1 = temp;
}
}
void displaybeg()
{
temp2 = h;
if (temp2 == NULL)
{
printf("List empty to display \n");
return;
}
printf("\n Linked list elements from begining : \n");
while (temp2 != NULL)
{
printf("%d %s %s %s %f %d\n", temp2->ssn, temp2->name, temp2->dept, temp2->desg, temp2->sal, temp2-
>phno);
temp2 = temp2->next;
}
printf(" No of employees = %d ", count);
}
int deleteend()
{
struct node *temp;
temp = h;
if (temp->next == NULL)
{
free(temp);
h = NULL;
return 0;
}
```

```
else
{
temp2 = temp1->prev;
temp2->next = NULL;
printf("%d %s %s %s %f %d\n", temp1->ssn, temp1->name, temp1->dept,
        temp1->desg, temp1->sal, temp1->phno);
free(temp1);
}
count--;
return 0;
}
int deletebeg()
{
struct node *temp;
temp = h;
if (temp->next == NULL)
{
free(temp);
h = NULL;
}
else
{
h = h->next;
printf("%d %s %s %s %f %d", temp->ssn, temp->name, temp->dept,
        temp->desg, temp->sal, temp->phno);
free(temp);
}
count--;
return 0;
}
int main()
{
int ch, n, i;
h = NULL;
temp = temp1 = NULL;
printf("-----------------MENU--------------------\n");
printf("\n 1 - Create a DLL of n emp");
printf("\n 2 - Display from beginning");
printf("\n 3 - Insert at end");
printf("\n 4 - delete at end");
printf("\n 5 - Insert at beg");
printf("\n 6 - delete at beg");
printf("\n 7 - exit\n");
printf("-----------------------------------------\n");
while (1)
{
printf("\n Enter choice : ");
scanf("%d", &ch);
switch (ch)
{
case 1:printf("\n Enter no of employees : ");
        scanf("%d", &n);
        for (i = 0; i<n; i++)
                insertend();
```

```
        break;
case 2:displaybeg();
        break;
case 3:insertend();
        break;
case 4:deleteend();
        break;
case 5:insertbeg();
        break;
case 6:deletebeg();
        break;
case 7: exit(1);
default: printf("wrong choice\n");
}
}
}
```

Output:



```
 Enter choice : 5

 Enter ssn,name,department, designation, salary and phno of employee : 666
FFF
CSE
AP
2000
12365

 Enter choice : 2

 Linked list elements from begining :
666 FFF CSE AP 2000.000000 12365
123 AAA CS AP 1000.000000 123456
124 BBB CS AP 2000.000000 123654
222 NNN CS AP 3000.000000 12354
 No of employees = 4
 Enter choice : 4
222 NNN CS AP 3000.000000 12354

 Enter choice : 2

 Linked list elements from begining :
666 FFF CSE AP 2000.000000 12365
123 AAA CS AP 1000.000000 123456
124 BBB CS AP 2000.000000 123654
 No of employees = 3
 Enter choice : 6
666 FFF CSE AP 2000.000000 12365
 Enter choice : 2

 Linked list elements from begining :
123 AAA CS AP 1000.000000 123456
124 BBB CS AP 2000.000000 123654
 No of employees = 2
 Enter choice : 7
```

Viva Questions:

1.What is doubly linked list?
2.How you represent doubly linked list?

3.How many pointers are required to implement a simple Linked list?
4.Which of the following statements (is)are true for double linked list?

a.the next pointer of last node points to null
b.the pre pointer of first node points to null
c.all of the above
d.none of the above
5.For double linked list, insertion of a new node can be done

a.after the 1st node
b.before the last node
c.after the 2nd node
d.all of the above
6.What is the difference between singly and doubly linked lists?

7.What are the applications that use Linked lists?

8.How to remove loops in a linked list (or) what are fast and slow pointers used for?

9.Header linked lists are frequently used for maintaining ..... in memory.
a.Polynomials

b.Binomial

c.Trinomial

d.Quadratic equation

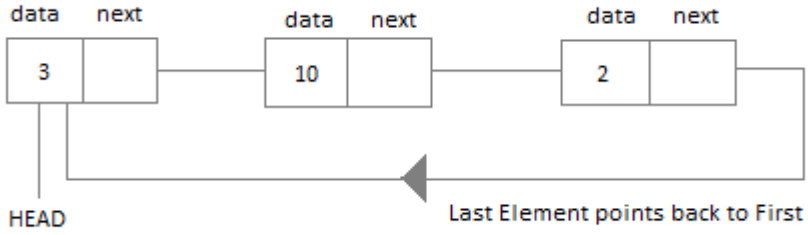
10.doubly linked list is also called as .....
A. linked list B. one way chain C. two way chain D. right link
11.RLINK is the pointer pointing to the ...
A. successor node B. predecessor node C. head node D. last node

7. Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes

a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)
Support the program with appropriate functions for each of the above operations

Description:

Circular Linked List is little more complicated linked data structure. In the circular linked list we can insert elements anywhere in the list whereas in the array we cannot insert element anywhere in the list because it is in the contiguous memory. In the circular linked list the previous element stores the address of the next element and the last element stores the address of the starting element. The elements points to each other in a circular way which forms a circular chain. The circular linked list has a dynamic size which means the memory can be allocated when it is required.



Application of Circular Linked List

- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.
- Another example can be Multiplayer games. All the Players are kept in a Circular Linked List and the pointer keeps on moving forward as a player's chance ends.
- Circular Linked List can also be used to create Circular Queue. In a Queue we have to keep two pointers, FRONT and REAR in memory all the time, where as in Circular Linked List, only one pointer is required.

Program:

```
#include<process.h>
#include<math.h>
#include<ctype.h>
#include<stdio.h>
#include"stdafx.h"
#include<stdlib.h>
#include<string.h>
#include<conio.h>


struct poly
{
int coef;                /* structure of each node in the circular list */
int powx;
int powy;
int powz;
```

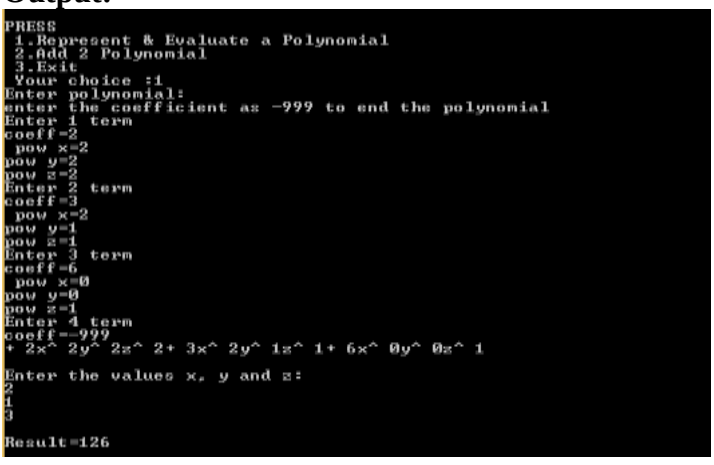
```
int flag;
struct poly *link;
};
typedef struct poly *term;


term getnode()          /* Dynamic memory allocation for each term of the polynomial inserted */
{
term x;
x = (term)malloc(sizeof(struct poly));
if (x == NULL)
{
printf("out of memory\n");
exit(1);
}
return x;
}


void display(term head)
{
term temp;
if (head->link == head)                    /* List is empty*/
{
printf("polynomial does not exist\n");
return;
}
for (temp = head->link; temp != head; temp = temp->link)
{
if (temp->coef<0)
        printf("%2dx^%2dy^%2dz^%2d", temp->coef, temp->powx, temp->powy, temp->powz);
else
        printf("+%2dx^%2dy^%2dz^%2d", temp->coef, temp->powx, temp->powy, temp->powz);
}


printf("\n");
}


/* inserting every new term of the polynomial at the rear end and linking it back to the head  node to form a
circular representation */


term insertrear(int coef, int x, int y, int z, term head)
{
term temp, cur;
temp = getnode();
temp->coef = coef;
temp->powx = x;
temp->powy = y;
temp->powz = z;
temp->flag = 0;
cur = head->link;
while (cur->link != head)
cur = cur->link;
```

```
cur->link = temp;
temp->link = head;
return head;
}


/* reading the structure of every term to be inserted */


term readpoly(term head)
{
int i;
int  powx, powy, powz, coef;
head = getnode();
head->link = head;
printf("enter the coefficient as -999 to end the polynomial\n");
for (i = 1;; i++)
{
printf("Enter %d term\n", i);
printf("coeff=");
scanf("%d", &coef);
if (coef == -999)
          break;
printf(" pow x=");
scanf("%d", &powx);
printf("pow y=");
scanf("%d", &powy);
printf("pow z=");
scanf("%d", &powz);
head = insertrear(coef, powx, powy, powz, head);
}
return head;
}
int power(int x, int n)
{
if (n == 0)
return 1;
return x * power(x, n - 1);
}
void evaluate(term poly)
{
term temp;
int x, y, z, xval, yval, zval, res = 0;
temp = poly->link;
printf("\nEnter the values x, y and z:\n");
scanf("%d%d%d", &x, &y, &z);
while (temp != poly)
{
xval = power(x, temp->powx);
yval = power(y, temp->powy);
zval = power(z, temp->powz);
res += temp->coef*xval*yval*zval;
temp = temp->link;
}
```

```
printf("\nResult=%d\n", res);
}


term addpoly(term head1, term head2, term head3) /* adding two polynomials */
{
term poly1, poly2;
int x1, x2, y1, y2, z1, z2, coef1, coef2, coef;
poly1 = head1->link;
while (poly1 != head1)
{
x1 = poly1->powx;
y1 = poly1->powy;
z1 = poly1->powz;
coef1 = poly1->coef;
poly2 = head2->link;
while (poly2 != head2)
        /* search for the matching powers for term in 2nd polynomial */
{
        x2 = poly2->powx;
        y2 = poly2->powy;
        z2 = poly2->powz;
        coef2 = poly2->coef;
        if (x1 == x2 && y1 == y2 && z1 == z2)
                break;
        poly2 = poly2->link;
}
if (poly2 != head2)
{
        coef = coef1 + coef2;
        poly2->flag = 1;
        if (coef != 0)
                head3 = insertrear(coef, x1, y1, z1, head3);
}
else
        head3 = insertrear(coef1, x1, y1, z1, head3);
poly1 = poly1->link;
}


/* Add the remaining terms of second polynomial to the result */


poly2 = head2->link;
while (poly2 != head2)
{
if (poly2->flag == 0)
{
        head3 = insertrear(poly2->coef, poly2->powx, poly2->powy, poly2->powz, head3);
}
poly2 = poly2->link;
}
return head3;
}
void main()
```

```
{
int ch;
term poly, head1, head2, head3;
poly = getnode();
head1 = getnode();
head2 = getnode();
head3 = getnode();
head3->link = head3;
do
{
printf("\n\nPRESS \n 1.Represent & Evaluate a Polynomial\n 2.Add 2 Polynomial\n 3.Exit\n Your choice :");
scanf("%d", &ch);
switch (ch)
{
case 1: printf("Enter polynomial:\n");
        poly = readpoly(poly);
        display(poly);
        evaluate(poly);
        break;
case 2: printf("Enter the first polynomial\n");
        head1 = readpoly(head1);
        printf("Enter the second polynomial\n");
        head2 = readpoly(head2);
        head3 = addpoly(head1, head2, head3);
        printf("The first polynomial is\n");
        display(head1);
        printf("The second polynomial is\n");
        display(head2);
        printf(" The sum of the polynomial is\n");
        display(head3);
        break;
case 3: printf("\n Terminating");
        break;
default: printf("Wrong Choice\n");
}
} while (ch != 3);
_getch();
}
```

Output:



```
PRESS
 1.Represent & Evaluate a Polynomial
 2.Add 2 Polynomial
 3.Exit
 Your choice :2
Enter the first polynomial
enter the coefficient as -999 to end the polynomial
Enter 1 term
coeff=5
 pow x=2
pow y=2
pow z=1
Enter 2 term
coeff=6
 pow x=3
pow y=2
pow z=1
Enter 3 term
coeff=6
 pow x=0
pow y=0
pow z=1
Enter 4 term
coeff=-999
Enter the second polynomial
enter the coefficient as -999 to end the polynomial
Enter 1 term
coeff=6
 pow x=2
pow y=2
pow z=2
Enter 2 term
coeff=5
 pow x=2
pow y=3
pow z=1
Enter 3 term
coeff=-999
The first polynomial is
+ 5x^ 2y^ 2z^ 1+ 6x^ 3y^ 2z^ 1+ 6x^ 0y^ 0z^ 1
The second polynomial is
+ 6x^ 2y^ 2z^ 2+ 5x^ 2y^ 3z^ 1
 The sum of the polynomial is
+ 5x^ 2y^ 2z^ 1+ 6x^ 3y^ 2z^ 1+ 6x^ 0y^ 0z^ 1+ 6x^ 2y^ 2z^ 2+ 5x^ 2y^ 3z^ 1
```

Viva questions:

1. what is singly circular linked list? How you represent it?
2. The disadvantage in using a circular linked list is .....

A. it is possible to get into infinite loop B. last node points to fist node.
C. time consuming D. requires more memory space.

3.A variation of linked list is circular linked list, in which the last node in the list points to first node of the list. One problem with this type of list is?

a) It waste memory space since the pointer head already points to the first node and thus the list node does not need to point to the first node.
b) It is not possible to add a node at the end of the list.
c) It is difficult to traverse the list as the pointer of the last node is now not NULL
d) All of above

4. In a circular linked list, the pointer of which of the following nodes points to null?
A. first node B) Last node C) can be any of the nodes D) None of the above

5.In a circular linked list, the pointer of the last node points to

A) first node B) Null C) second last node D) None of the above

6.You are given the head pointer of a circular linked list of size n. You then start traversing the linked list and stop when you encounter null pointer.How many nodes would you have covered?

A). n-1 nodes B) n nodes C).it can be any number between 1 to 10 nodes D).none of these

7. Which of the following things are best represented by circular linked lists?
   A). a given set of integers which need to be sorted
   B). vertices of a polygon
   C). names of students according to roll numbers
   D). None of these

8. Which of the following problems can occur with circular linked list?

A). not dynamic data structure B). infinite loop C). cannot insert in Θ(1) time
D). None of these

9. Which of the following statements are true?
A). there is no null pointer in circular linked list
B).in circular linked list, from every node we can reach every other node
C).All of the above
D).None of the above

10. To convert a simple linked list into a cicular linked list,we need to
A). make the head pointer point to the last node and then reverse the list
B). make the last node point to the first node
C). arrange the nodes of the list in a circle in the computer memory
D). anyone of the above methods can be used

8. Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Exit

Description:
Binary search trees (BST), sometimes called ordered or sorted binary trees, are a particular type of containers: data structures that store "items" (such as numbers, names etc.) in memory. They allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its *key* (e.g., finding the phone number of a person by name).
Binary search trees keep their keys in sorted order, so that lookup and other operations can use the principle of binary search: when looking for a key in a tree (or a place to insert a new key), they traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, based on the comparison, to continue searching in the left or right subtrees. On average, this means that each comparison allows the operations to skip about half of the tree, so that each lookup, insertion or deletion takes time proportional to the logarithm of the number of items stored in the tree.

Program:
```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>

  typedef struct BST {
  int data;
  struct BST *lchild, *rchild;
  } node;

  void insert(node *, node *);
  void inorder(node *);
  void preorder(node *);
  void postorder(node *);
  node* search(node *, int);
```

```
void main() {
int choice, n, i;
char ans = 'N';
int key;
node *new_node, *root, *tmp, *parent;
node *get_node();
parent = NULL;
root = NULL;
printf("\nProgram For Binary Search Tree ");
do {
        printf("\n1.Create");
        printf("\n2.Recursive Traversals");
        printf("\n3.Search");
        printf("\n4.Exit");
        printf("\nEnter your choice :");
        scanf("%d", &choice);


        switch (choice) {
        case 1:
                printf("\n How many nodes you want to enter : ");
                scanf("%d", &n);
                printf("\nEnter The %d Elements :  ", n);
                for (i = 0; i<n; i++)
                {
                        new_node = get_node();
                        scanf("%d", &new_node->data);


                        if (root == NULL) // Tree is not Created
                                root = new_node;
                        else
                                insert(root, new_node);
                }
                break;


        case 2:
                if (root == NULL)
                        printf("Tree Is Not Created");
                else {
                        printf("\nThe Inorder display : ");
                        inorder(root);
                        printf("\nThe Preorder display : ");
                        preorder(root);
                        printf("\nThe Postorder display : ");
                        postorder(root);
                }


                break;


        case 3:
```

```
                printf("\nEnter Element to be searched :");
                scanf("%d", &key);

                tmp = search(root, key);
                if (tmp)
                        printf("\nThe %d Element is Present", tmp->data);

                else
                        printf("\n The Key %d is not present in the BST", key);
                break;

        default:
                printf("\n Terminating");
                exit(0);

        }
} while (choice != 5);
}
// Get new Node

node *get_node() {
node *temp;
temp = (node *)malloc(sizeof(node));
temp->lchild = NULL;
temp->rchild = NULL;
return temp;
}
// This function is for creating a binary search tree

void insert(node *root, node *new_node)
{
if (new_node->data < root->data)
{
        if (root->lchild == NULL)
                root->lchild = new_node;
        else
                insert(root->lchild, new_node);
}

if (new_node->data > root->data)
{
        if (root->rchild == NULL)
                root->rchild = new_node;
        else
                insert(root->rchild, new_node);
}
}
// This function is for searching the node from binary Search Tree
```

```
node *search(node *root, int key)
{
node *temp;
temp = root;
if (root == NULL)
{
        printf("\n BST is Empty ");
        return root;
}
while (temp != NULL)
{
        if (temp->data == key)
                return temp;
        if (key < temp->data)
                temp = temp->lchild;
        else
                temp = temp->rchild;
}
return NULL;
}
// This function displays the tree in inorder fashion


void inorder(node *temp) {
if (temp != NULL) {
        inorder(temp->lchild);
        printf("%d , ", temp->data);
        inorder(temp->rchild);
}
}
// This function displays the tree in preorder fashion


void preorder(node *temp) {
if (temp != NULL) {
        printf("%d , ", temp->data);
        preorder(temp->lchild);
        preorder(temp->rchild);
}
}


// This function displays the tree in postorder fashion
void postorder(node *temp) {
if (temp != NULL) {
        postorder(temp->lchild);
        postorder(temp->rchild);
        printf("%d , ", temp->data);
}
}
```

Output:

```
Program For Binary Search Tree
1.Create
2.Recursive Traversals
3.Search
4.Exit
Enter your choice :1

 How many nodes you want to enter : 5

Enter The 5 Elements :  10
20
30
90
60

1.Create
2.Recursive Traversals
3.Search
4.Exit
Enter your choice :2

The Inorder display : 10 , 20 , 30 , 60 , 90 ,
The Preorder display : 10 , 20 , 30 , 90 , 60 ,
The Postorder display : 60 , 90 , 30 , 20 , 10 ,
1.Create
2.Recursive Traversals
3.Search
4.Exit
Enter your choice :3

Enter Element to be searched :50

 The Key 50 is not present in the BST
1.Create
2.Recursive Traversals
3.Search
4.Exit
Enter your choice :3

Enter Element to be searched :20

The 20 Element is Present
```

Viva Questions:

1. What is BST?
2. What are the basic operations that can be performed on binary search tree data structure?
3. What is binary search tree and how to traverse inorder, preorder and postorder?
4. What is recursion?
5. Explain the working of binary search technique.
6. What is tree?
7. What is min heap?
8. Illustrate max heap with an example.
9. What is the maximum height of a binary tree with N nodes?
10. What is the minimum height of a binary tree?
11. What is the maximum number of external nodes (or leaves) for a binary tree with height H?
12. What is the maximum number of internal nodes (or leaves) for a binary tree with height H?
13. . Discuss advantages of using hash tables versus binary search trees
14. Draw the binary tree which would be created by inserting the following numbers in the order given. We insert them as follows. Start from root, if number is less go left, else go right. When you get to a leaf, insert the new node.

50 30 25 75 82 28 63 70 4 43 74 35

15. Implement a non-recursive method of finding the minimum node in the binary search tree. The method should return a node.
16. Here is a small binary tree:

```
   14
  / \
```

```
    2    11
   / \   / \
  1  3  10  30
       / /
      7  40
```

Write the order of the nodes visited in:
A. An in-order traversal:
B. A pre-order traversal:
C. A post-order traversal:

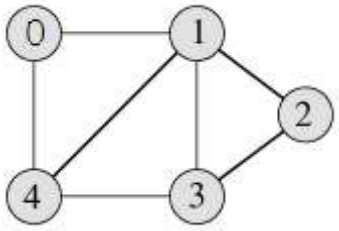
9. **Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities**
a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a digraph using BFS/DFS method

Description:

Graph is a data structure that consists of following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Graphs are used to represent many real life applications:
Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, facebook. For example, in facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender and locale.

Following is an example undirected graph with 5 vertices.



Following two are the most commonly used representations of graph.
1. Adjacency Matrix
2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Adjacency Matrix Representation of the above graph

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex 'u' to vertex 'v' are efficient and can be done O(1).
Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.

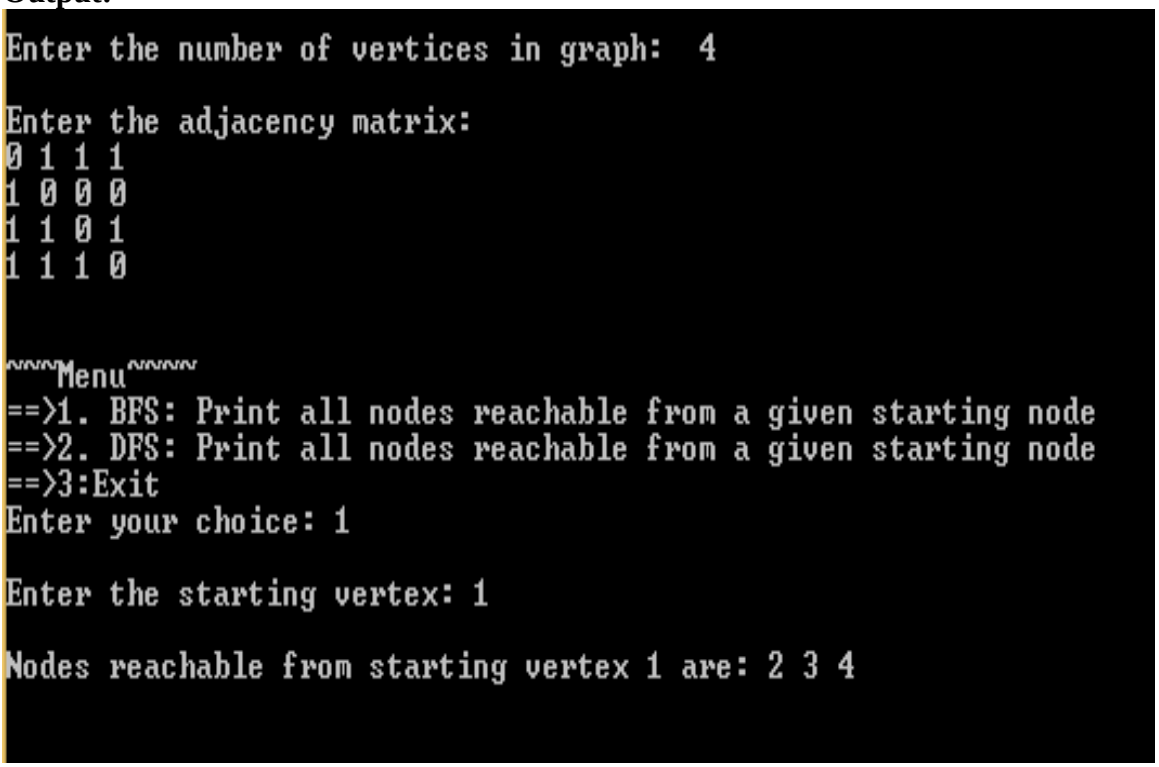
Program:

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
void bfs(int v);
void dfs(int v);

int a[50][50], n, visited[50];
int q[20], front = -1, rear = -1;
int s[20], top = -1, count = 0;

void creategraph()
{
int i, j;
printf("\nEnter the number of vertices in graph:  ");
scanf("%d", &n);
printf("\nEnter the adjacency matrix:\n");
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
        scanf("%d", &a[i][j]);
}

void bfs(int v)
{
int i, cur;
visited[v] = 1;
q[++rear] = v;
printf("\nNodes reachable from starting vertex %d are: ", v);
while (front != rear)
{
cur = q[++front];
for (i = 1; i <= n; i++)
{
        if ((a[cur][i] == 1) && (visited[i] == 0))
```

```
        {
                q[++rear] = i;


                visited[i] = 1;
                printf("%d ", i);
        }
}
}
}

void dfs(int v)
{
int i;
visited[v] = 1;
s[++top] = v;
for (i = 1; i <= n; i++)
{
if (a[v][i] == 1 && visited[i] == 0)
{
        dfs(i);
        count++;
}
}
}

int main()
{
int ch, start, i, j;
creategraph();
printf("\n\n~~~Menu~~~~");
printf("\n==>1. BFS: Print all nodes reachable from a given starting node");
printf("\n==>2. DFS: Print all nodes reachable from a given starting node");
printf("\n==>3:Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch (ch)
{
case 1:
        for (i = 1; i <= n; i++)
visited[i] = 0;
printf("\nEnter the starting vertex: ");
scanf("%d", &start);
bfs(start);
for (i = 1; i <= n; i++)
{
        if (visited[i] == 0)
                printf("\nThe vertex that is not reachable is %d", i);
}
break;
case 2:
        for (i = 1; i <= n; i++)
visited[i] = 0;
printf("\n Enter the starting vertex:\t");
```

```
scanf("%d", &start);
dfs(start);
printf("\nNodes reachable from starting vertex %d are:\n", start);
for (i = 1; i <= count; i++)
        printf("%d\t", s[i]);
break;
case 3: return 0;
default: printf("\nPlease enter valid choice:");
}
getch();
}
```

Output:



Viva Questions:

1. What is graph?
2. Define adjacent nodes.
3. What is a directed graph?
4. What is an undirected graph?
5. What is a loop?
6. What is a simple graph?
7. What is a weighted graph?
8. Define outdegree of a graph?
9. Define indegree of a graph?
10. Define path in a graph?
11. What is a simple path?
12. What is a cycle or a circuit?
13. What is an acyclic graph?
14. What is meant by strongly connected in a graph?
15. When is a graph said to be weakly connected?
16. Namethe different ways of representing a graph?
17. What is an undirected acyclic graph?
18. What is DFS and BFS?

19. What are the different algorithms used to implement DFS?
20. What are the different algorithms used to implement BFS?
21. Which of the following algorithms can be used to most efficiently determine the presence of a cycle in a given graph ?

A).DFS B) BFS C) Prim's Minimum Spanning Tree Alg D). Kruskal' Minimum Spanning Tree Algo

22.What is the space complexity of Depth-first search?

a) O(b) b) O(bl) c) O(m) d) O(bm)

23.Which search algorithm imposes a fixed depth limit on nodes?

a) Depth-limited search b) Depth-first search
c) Iterative deepening search d) Bidirectional search

24..DFS is _____ efficient and BFS is _____ efficient.

a) Space, Time b) Time, Space c) Time, Time d) Space, Space

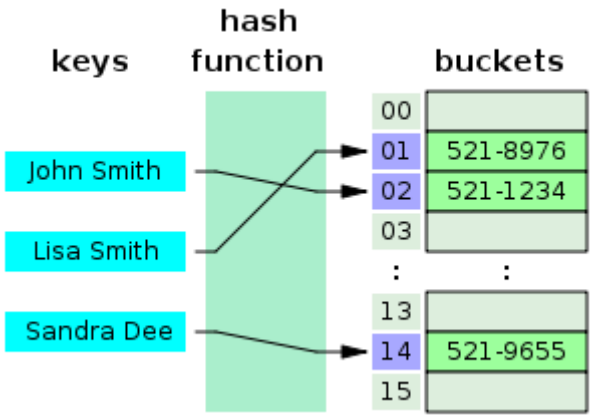
25.The data structure required for Breadth First Traversal on a graph is

A)queue (B) stack (C) array (D) tree

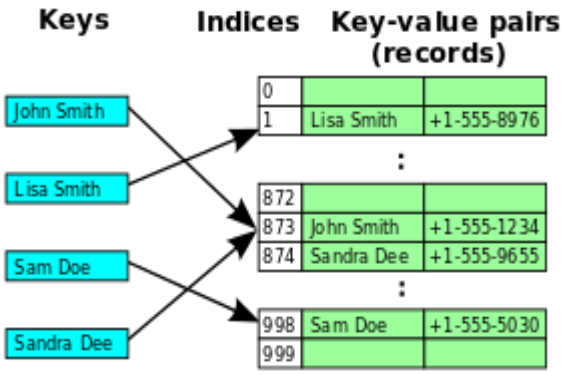
10. Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method) and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Description:

In computing, a hash table (hash map) is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.



Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key. In these schemes, each cell of a hash table stores a single key–value pair. When the hash function causes a collision by mapping a new key to a cell of the hash table that is already occupied by another key, linear probing searches the table for the closest following free location and inserts the new key there. Lookups are performed in the same way, by searching the table sequentially starting at the position given by the hash function, until finding a cell with a matching key or an empty cell.



Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct records
{
int flag, usn;// usn is the key- K with 4 digits
char name[20];
}rec[15];


int hash(int m)
{
int slots = 15; //assume number of slots available in hash table is 10
int r;
r = m % slots;
return r;
}
void main()
{
int m, k, usn, loc, i, n, j;
char name[20];
FILE *in, *outp;
printf("\nEnter no of records to read from file: ");
scanf("%d", &n);
in = fopen("input.txt", "r");//create input.txt file inside the source file of the Project created before executing and
add min 10 records
if (n <= 15)
{
for (k = 0; k<15; k++)
        rec[k].flag = 0;
for (i = 0; i<n; i++)
{
        fscanf(in, "%s%d", name, &usn);
        loc = hash(usn);
        printf("\naddress: %d", loc);
        if (rec[loc].flag == 0)
        {
```

```
                strcpy(rec[loc].name, name);
                rec[loc].usn = usn;
                rec[loc].flag = 1;
        }
        else//linear probing is taking place
        {
                for (j = loc + 1; j % 15<15; j++)
                {
                        if (rec[j % 15].flag == 0)
                        {
                                strcpy(rec[j % 15].name, name);
                                rec[j % 15].usn = usn;
                                rec[j % 15].flag = 1;
                                break;
                        }
                }
        }
}
fclose(in);
printf("\n\nThe Hash Table Content is: ");
for (i = 0; i<15; i++)
{
        if (rec[i]. flag == 1)
                printf ("\n%s     %d ", rec[i].name, rec[i].usn);
        else
                printf("\n##");
}
outp = fopen("output.txt", "w");//create output.txt file inside the source file of the Project created before executing
for (i = 0; i<15; i++)
{
        if (rec[i]. flag == 0)
                fprintf(outp, "\n# | #");
        else
                fprintf(outp, "\n%s | %d ", rec[i].name, rec[i].usn);
}
fclose(outp);
}
else
printf("\n Please Provide less than 10 Records !! ");
getch();
}
```

Output:

```
Enter no of records to read from file: 5

address: 13
address: 14
address: 3
address: 8
address: 10

The Hash Table Content is:
##
##
##
63      588
##
##
##
##
52      698
##
58      700
##
##
48      358
25      569
```

Viva Questions:

1. Define Hashing.
2. What do you mean by hash table?
3. What do you mean by hash function?
4. Write the importance of hashing.
5. What do you mean by collision in hashing?
6. What are the collision resolution methods?
7. What do you mean by separate chaining?
8. Write the advantage of separate chaining.
9. Write the disadvantages of separate chaining.
10. What are the types of collision resolution strategies in open addressing?
11. What do you mean by Probing?
12. What do you mean by linear probing?
13. What do you mean by primary clustering?
14. List the limitations of linear probing.
15. Mention one advantage and disadvantage of using quadratic probing
16. Why would I want to use a hash table rather than an array?
17. What are different techniques for making hash function? Explain with example.
18. What are different methods of collision resolution in hashing

0030 Latha
0012 Kavya
0026 Rakshit
0074 Deepa
0029 Asha
0026 Priya
0004 Chandra
0030 Neetu