Dr. T. THIMMAIAH INSTITUTE OF TECHNOLOGY

[AFFILIATED TO VISVESWARAIAH TECHNOLOGICAL UNIVERSITY]

OORGAUM, KGF – 563120

NAAC Accredited 'A' Grade

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BE [CSE] - III SEMESTER

Data Structures Laboratory

Subject Code: BCSL305

NAME: ……………………………………………………………………………….	
BRANCH: ………………………	**USN: …………………………………..**

Dr. T. THIMMAIAH INSTITUTE OF TECHNOLOGY

[Affiliated To Visveswaraiah Technological University]

OORGAUM, KGF – 563120

NAAC Accredited 'A' Grade

Department of Computer Science and Engineering

2023-24

Vision of the Department:

- ❖ **To produce highly competent and innovative Computer Science professionals through excellence in teaching, training and research.**

Mission of the Department:

- ❖ **To provide appropriate infrastructure to impart need-based technical education through effective teaching and research.**
- ❖ **To involve the students in innovative projects on emerging technologies to fulfill the industrial requirements.**
- ❖ **To render leadership skills and ethical responsibilities in students that leads them to become globally competent professionals.**

DATA STRUCTURES LABORATORY
SEMESTER - III

Course Code:	**BCSL305**	**CIE Marks**	50
Number of Contact Hours/Week:	0:0:2	**SEE Marks**	50
Total Number of Lab Contact Hours:	28	**Exam Hours**	03
Credits - 1			

Course Learning Objectives:

This laboratory course enables students to get practical experience in design, develop, implement, analyze and evaluation/testing of

- ❖ Dynamic memory management
- ❖ Linear data structures and their applications such as stacks, queues and List
- ❖ Non-Linear data structures and their applications such as trees and graphs

Descriptions (if any):

- ❖ Implement all the programs in "C " Programming Language and Linux OS.

Program List:

Prog. No	Details
1	Develop a Program in C for the following: a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String). b) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.
2	Develop a Program in C for the following operations on Strings. a) Read a main String (STR), a Pattern String (PAT) and a Replace String (REP) b) Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in functions.
3	Develop 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
4	Develop 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.
5	Develop a Program in C for the following Stack Applications a) Evaluation of Suffix expression with single digit operands and operators: **+, -, *, /, %, ^** b) Solving Tower of Hanoi problem with n disks
6	Develop 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
7	Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) 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(Demonstration of stack) e) Exit
8	Develop 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
9	Develop a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes a) Represent and Evaluate a Polynomial P(x,y,z) = 6x 2 y 2 z-4yz 5 +3x 3 yz+2xy 5 z-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

10	Develop 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
11	Develop 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 DFS/BFS method
12	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. 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.

Laboratory Outcomes:
At the end of the course the students will be able to:

CO1	**Demonstrate** the working nature of different types of data structures.
CO2	**Design** and **Analyze** the performance of Stack, Queue, SLL, DLL and CLLL.
CO3	**Demonstrate** the non-linear tree, graph data structure.

Conduct of Practical Examination:

- ❖ Experiment distribution
  - ✓ For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - ✓ For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- ❖ Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- ❖ Marks Distribution *(Need to change in accordance with university regulations)*
  - a) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - b) For laboratories having PART A and PART B
    - i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    - ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

EXPERIMENT: 1

a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

PROGRAM CODE:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
// Define a structure to represent a day
struct Day
{
        char name[20];
        int date;
        char activity[100];
};
void main()
{
        int i;
        // Declare an array of 7 elements to represent the calendar
        struct Day calendar[7];
        clrscr();
        // Initialize the calendar with sample data
        strcpy(calendar[0].name, "Monday");
        calendar[0].date = 1;
        strcpy(calendar[0].activity, "Work from 9 AM to 5 PM");
        strcpy(calendar[1].name, "Tuesday");
        calendar[1].date = 2;
        strcpy(calendar[1].activity, "Meeting at 10 AM");

        strcpy(calendar[2].name, "Wednesday");
        calendar[2].date = 3;
        strcpy(calendar[2].activity, "Gym at 6 PM");

        strcpy(calendar[3].name, "Thursday");
        calendar[3].date = 4;
        strcpy(calendar[3].activity, "Dinner with friends at 7 PM");

        strcpy(calendar[4].name, "Friday");
        calendar[4].date = 5;
        strcpy(calendar[4].activity, "Movie night at 8 PM");

        strcpy(calendar[5].name, "Saturday");
        calendar[5].date = 6;
        strcpy(calendar[5].activity, "Weekend getaway");

        strcpy(calendar[6].name, "Sunday");
        calendar[6].date = 7;
        strcpy(calendar[6].activity, "Relax and recharge");

        // Print the calendar
        printf("Calendar for the week:\n");
        for (i = 0; i < 7; i++)
        {
                printf("%s (Date: %d): %s\n", calendar[i].name, calendar[i].date,
                calendar[i].activity);
        }
        getch();
}
```

SAMPLE OUTPUT 1:

```
Calendar for the week:

Monday (Date: 1): Work from 9 AM to 5 PM
Tuesday (Date: 2): Meeting at 10 AM
Wednesday (Date: 3): Gym at 6 PM
Thursday (Date: 4): Dinner with friends at 7 PM
Friday (Date: 5): Movie night at 8 PM
Saturday (Date: 6): Weekend getaway
Sunday (Date: 7): Relax and recharge
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 1

b) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

PROGRAM CODE

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

// Define a structure to represent a day
struct Day
{
        char name[20];
        int date;
        char activity[100];
};

// Function to create the calendar
void create(struct Day calendar[7])
{
        int i;
        for (i = 0; i < 7; i++)
        {
                printf("Enter details for %s:\n", calendar[i].name);
                printf("Date: ");
                scanf("%d", &calendar[i].date);
                printf("Activity: ");
                scanf(" %s", &calendar[i].activity);
       }
}
// Function to read data from the keyboard

void read(struct Day calendar[7])
{
        int i;
        FILE *file = fopen("calendar.txt", "r");
        if (file == NULL)
        {
                printf(" \n Error opening the file.\n");
                return;
        }
        for (i = 0; i < 7; i++)
        {
                fscanf(file, "%d", &calendar[i].date);
                fscanf(file, " %s", &calendar[i].activity);
        }
        fclose(file);
}

// Function to display the calendar
void display(struct Day calendar[7])
{
        int i;
        printf("Calendar for the week:\n");
        for (i = 0; i < 7; i++)
        {
              printf("%s (Date: %d): %s\n", calendar[i].name, calendar[i].date,
              calendar[i].activity);
        }
}
```

```
int main()
{
        struct Day calendar[7];
        int choice;
        clrscr();
        // Initialize the names of the days
        strcpy(calendar[0].name, "Monday");
        strcpy(calendar[1].name, "Tuesday");
        strcpy(calendar[2].name, "Wednesday");
        strcpy(calendar[3].name, "Thursday");
        strcpy(calendar[4].name, "Friday");
        strcpy(calendar[5].name, "Saturday");
        strcpy(calendar[6].name, "Sunday");
        printf("1. Create Calendar \n");
        printf("2. Read Calendar from File \n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
                create(calendar);
                break;
        case 2:
                read(calendar);
                break;
        default:
                printf("Invalid choice.\n");
                return 1;
        }
display(calendar);
return 0;
}
```

SAMPLE OUTPUT 1:

```
1. Create Calendar
2. Read Calendar from File

Enter your choice: 1
Enter details for Monday:
Date: 20/11/2023
Activity: C
Enter details for Tuesday:
Date: 21/11/2023
Activity: C++
Enter details for Wednesday:
Date: 22/11/2023
Activity: JAVA
Enter details for Thursday:
Date: 23/11/2023
Activity: PYTHON
Enter details for Friday:
Date: 24/11/2023
Activity: GAMING
Enter details for Saturday:
Date: 25/11/2023
Activity: APTITUDE
Enter details for Sunday:
Date: 26/11/2023
Activity: GENERAL KNOWLEDGE

Enter your choice: 2
Calendar for the week:
Monday (Date: 20/11/2023): C
Tuesday (Date: 21/11/2023): C++
Wednesday (Date: 22/11/2023): JAVA
```

```
Thursday (Date: 23/11/2023): PYTHON
Friday (Date: 24/11/2023): GAMING
Saturday (Date: 25/11/2023): APTITUDE
Sunday (Date: 26/11/2023): GENERAL KNOWLEDGE
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 2

Design, Develop and Implement a Program in C for the following operations on Strings a. Read a main String (STR),

a Pattern String (PAT) and a Replace String (REP)

b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT doesnot exist in STR.

Support the program with functions for each of the above operations.Don't use Built-in functions.

ABOUT THE EXPERIMENT:

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello." char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; If you follow the rule of array initialization then you can write the above statement as follows: char greeting[] = "Hello"; C language supports a wide range of built-in functions that manipulate null-terminated.strings as follows:

strcpy(s1, s2); Copies string s2 into string s1.

strcat(s1, s2); Concatenates string s2 onto the end of string s1.

strlen(s1); Returns the length of string s1.

strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1s2.

strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.

strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

ALGORITHM:

Step 1: Start the Program

Step 2: Read main string STR, pattern string PAT and replace string REP.

Step 3: Search / find the pattern string PAT in the main string STR.

Step 4: if PAT is found then replace all occurrences of PAT in main string STR with
REP string.

Step 5: if PAT is not found give a suitable error message.

Step 6: Stop the Program.

PROGRAM CODE:

```
#include<stdio.h>
void main()
{
        char STR[100], PAT[100], REP[100], ans[100];
        int i, j, c, m, k, flag=0;
        printf("\n Enter the MAIN string: \n");
        gets(STR);
        printf("\n Enter a PATTERN string: \n");
        gets(PAT);
        printf("\n Enter a REPLACE string: \n");
        gets(REP);
         i = m = c = j = 0;
        while ( STR[c] != '\0')
        {
                // Checking for Match
                if ( STR[m] == PAT[i])
                {
                        i++; m++;
                        flag=1;
                        if ( PAT[i] == '\0')
                        {
                                //copy replace string in ans string
                                for(k=0; REP[k] != '\0';k++,j++)
                                ans[j] = REP[k];
                                 i=0;
                                c=m;
                        }
                }
                else //mismatch
                {
                        ans[j] = STR[c];
                        j++;
                        c++;
                        m = c; i=0;
                }
        }
        if(flag==0)
        {
                printf("\n Pattern doesn't found!!!");
        }
        else
        {
                ans[j] = '\0';
                printf("\n The RESULTANT string is:%s\n" ,ans);
        }
}
```

SAMPLE OUTPUT 1:

```
Enter the MAIN string: good morning
Enter a PATTERN string: morning
Enter a REPLACE string: evening
The RESULTANT string is: good evening
```

SAMPLE OUTPUT 2:

```
Enter the MAIN string: hi Raja
Enter a PATTERN string: bye
Enter a REPLACE string: hello
Pattern not found!!
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 3

Design, Develop a menu driven Program in C for the following operations on STACK ofIntegers (Array Implementation of Stack with maximum size MAX)

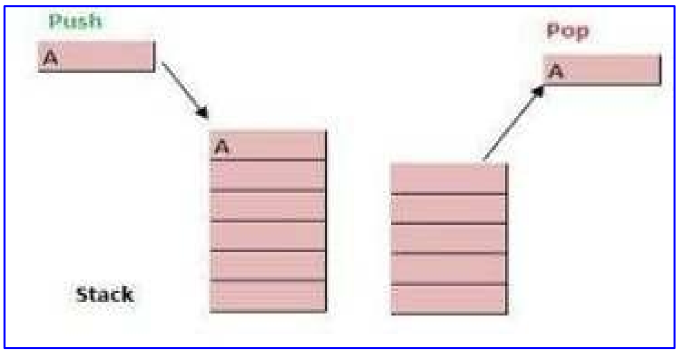
- **a. Push an Element on to Stack**
- **b. Pop an Element from Stack**
- **c. Demonstrate how Stack can be used to check Palindrome**
- **d. Demonstrate Overflow and Underflow situations on Stack**
- **e. Display the status of Stack**
- **f. Exit**

Support the program with appropriate functions for each of the above operations.

ABOUT THE EXPERIMENT:

A stack is an abstract data type (ADT), commonly used in most programming languages. It isnamed stack as it behaves like a real-world stack. A real-world stack allows operations at oneend only. For example, we can place or remove a card or plate from top of the stack only.

Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. LIFO standsfor Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation. A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic re-sizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.



Basic Operations:

push() - pushing (storing) an element on the stack.

pop() -removing (accessing) an element from the 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.

ALGORITHM:

```
Step 1: Start the Program.
Step 2: Initialize stack size MAX and top of stack -1.
Step 3: Push integer element on to stack and display the contents of the stack. if stack is
        full give a message as „Stack is Overflow‟.
Step 4: Pop element from stack along with display the stack contents. if stack is empty
        give a message as „Stack is Underflow‟.
Step 5: Check whether the stack contents are Palindrome or not.
Step 6: Stop the Program
```

PROGRAM CODE

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

#define max_size 5
int stack[max_size], top=-1, flag=1;
int i, temp, item, rev[max_size], num[max_size];

void push();
void pop();
void display();
void pali();

void main()
{
        int choice;
        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(" ");
        while(1)
        {
                printf("\n Enter your choice:\t");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:
                                push();
                                break;
                        case 2:
                                pop();
                                if(flag)
                                printf("\n The poped element: %d\t",item);
                                temp=top;
                                break;
                        case 3:
                                pali();
                                top=temp;
                                break;
                        case 4:
                                display();
                                break;
                        case 5:
                                exit(0);
                                break;
                        default:
                                printf("\n Invalid choice:\n");
                                break;
                }
        }
}

void push() //Inserting element into the stack
{
        if(top==(max_size-1))
        {
                printf("\n Stack Overflow:");
        }
        else
        {
                printf("Enter the element to be inserted:\t");
                scanf("%d",&item);
                top=top+1;
                stack[top]=item;
        }
        temp=top;
}

void pop() //deleting an element from the stack
{
        if(top==-1)
        {
                printf("Stack Underflow:");
                flag=0;
        }
        else
        {
                item=stack[top];
```

```
		top=top-1;
	}
}

void pali()
{
	i=0;
	if(top==-1)
	{
		printf(" Push some elements into the stack first\n");
	}
	else
	{
		while(top!=-1)
		{
			rev[top]=stack[top];
			pop();
		}
		top=temp;
		for(i=0;i<=temp;i++)
		{
			if(stack[top--]==rev[i])
			{
				if(i==temp)
				{
					printf(" Numbers in Stack is Palindrome\n");
					return;
				}
			}
		}
		printf("Numbers in Stack is not Palindrome\n");
	}
}

void display()
{
	int i;
	top=temp;
	if(top==-1)
	{
		printf("\n Stack is Empty:");
	}
	else
	{
		printf("\n The stack elements are:\n" );
		for(i=top;i>=0;i--)
		{
```

```
                        printf("%d\n",stack[i]);
                }
        }
}
```

SAMPLE OUTPUT 1:

```
STACK OPERATIONS

1. Push
2. Pop
3. Palindrome
4. Display
5. Exit


Enter your choice: 1
Enter the element to be inserted: 1

Enter your choice: 1
Enter the element to be inserted: 2

Enter your choice: 1
Enter the element to be inserted: 1

Enter your choice: 1
Enter the element to be inserted: 5

Enter your choice: 4
The stack elements are: 1 2 1 5

Enter your choice: 2
The poped element: 5

Enter your choice: 4
The stack elements are: 1 2 1

Enter your choice: 3
Numbers in Stack is Palindrome

Enter your choice: 5 Exit
```

SAMPLE OUTPUT 2:

```
STACK OPERATIONS

1. Push
2. Pop
3. Palindrome
4. Display
5. Exit

Enter your choice: 1
Enter the element to be inserted: 10

Enter your choice: 1
Enter the element to be inserted: 20

Enter your choice: 1
Enter the element to be inserted: 30

Enter your choice: 1
Enter the element to be inserted: 40

Enter your choice: 4
The stack elements are: 10 20 30 40

Enter your choice: 2
The poped element: 40

Enter your choice: 4
The stack elements are: 10 20 30

Enter your choice: 3
Numbers in Stack is not Palindrome

Enter your choice: 5
Exit
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 4

Develop 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.

ABOUT THE EXPERIMENT:

Infix: Operators are written in-between their operands. Ex: X + Y

Prefix: Operators are written before their operands. Ex: +X Y

Postfix: Operators are written after their operands.Ex: XY+

Examples of Infix, Prefix, and Postfix

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

Expression	**Stack**	**Output**	**Comment**
5^E+D*(C^B+A)	Empty	-	Initial
^E+D*(C^B+A)	Empty	5	Print
E+D*(C^B+A)	^	5	Push
+D*(C^B+A)	^	5E	Print
D*(C^B+A)	+	5E^	Pop and Push
*(C^B+A)	+	5E^D	Print
(C^B+A)	+*	5E^D	Push
C^B+A)	+*(	5E^D	Push
^B+A)	+*(	5E^DC	Print
B+A)	+*(^	5E^DC	Push
+A)	+*(^	5E^DC	Print
A)	+*(+	5E^DC^	Pop and Push
)	+*(+	5E^DC^A	Print
End	+*	5E^DC^A+	Pop until '('
End	Empty	5E^DC^A+* +	Pop until '('

Infix to prefix conversion Expression = (A+B^C)*D+E^5

Step 1. Reverse the infix expression. 5^E+D*)C^B+A(

Step 2. Make Every '(' as ')' and every ')' as '(' 5^E+D*(C^B+A)

Step 3. Convert expression to postfix form.

Step 4. Reverse the expression: +*+A^BCD^E

Step 5. Result : +*+A^BCD^E5

ALGORITHM:

Step 1: Start the program.

Step 2: Read an infix expression with parenthesis and without parenthesis.

Step 3: convert the infix expression to postfix expression.

Step 4: Stop the program.

PROGRAM CODE:

```
#include <conio.h>
#include <ctype.h>
#include <stdio.h>

#define SIZE 50 /* Size of Stack */
char s[SIZE];
int top = -1; /* Global declarations */

void 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;
        clrscr();
        printf("\n\n Read 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\n Given Infix Expn: %s Postfix Expn: %s\n", infx, pofx);
        getch();
}
```

SAMPLE OUTPUT 1:

```
        Read the Infix Expression (a+b)*c/d^5%1
        Given Infix Expn: (a+b)*c/d^5%1
        Postfix Expn: ab+c*d5^/1%
```

SAMPLE OUTPUT 2:

```
        Read the Infix Expression (a+(b-c)*d)
        Given Infix Expn: (a+(b-c)*d)
        Postfix Expn: abc-d*+
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 5

Develop a Program in C for the following Stack Applications

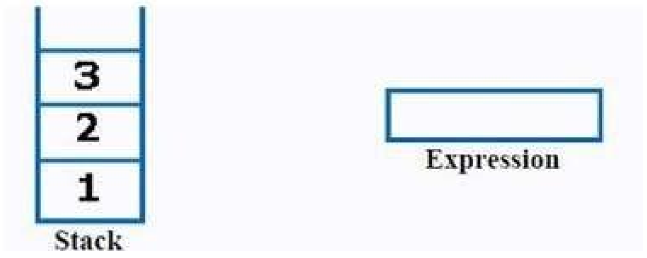
a. Evaluation of Suffixexpression with single digit operands and operators: +, -, *, /, %, ^

b. Solving Tower of Hanoi problem with n disks.

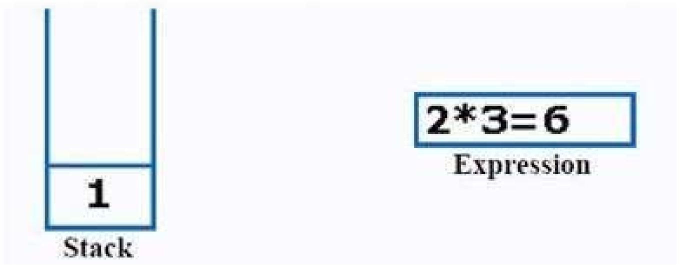
ABOUT THE EXPERIMENT:

Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^Postfix/Suffix Expression: Operators are written after their operands. Ex: XY+ In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+

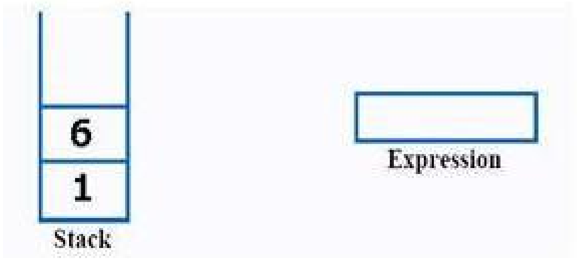
Example: Postfix String: 123*+4- Initially the Stack is empty. Now, the first three charactersscanned are 1,2 and 3, which are operands. Thus they will be pushed into the stack in that order.



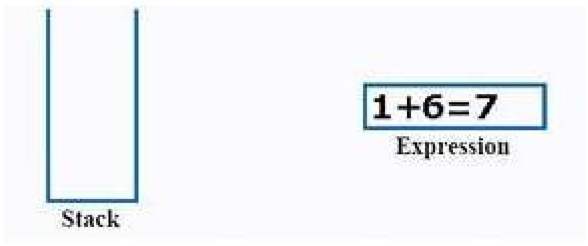
Next character scanned is "*", which is an operator. Thus, we pop the top two elements fromthe stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.



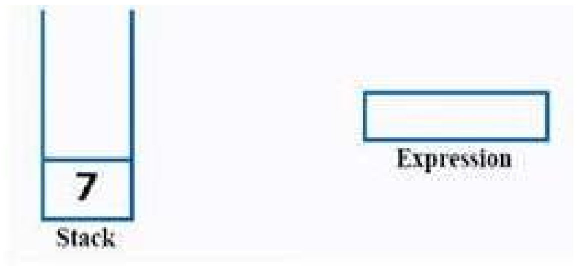
The value of the expression(2*3) that has been evaluated(6) is pushed into the stack.



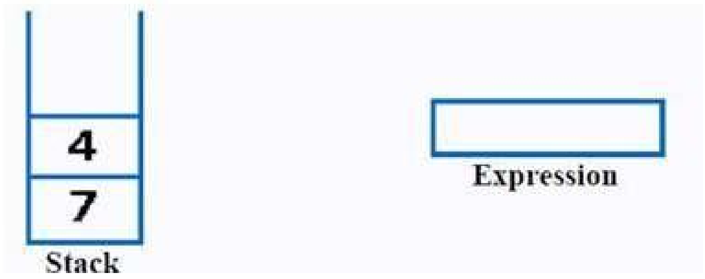
Next character scanned is "+", which is an operator. Thus, we pop the top two elements fromthe stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.



The value of the expression(1+6) that has been evaluated(7) is pushed into the stack.



Next character scanned is "4", which is added to the stack.



Next character scanned is "-", which is an operator. Thus, we pop the top two elements from the stack and perform the "-" operation with the two operands. The second operand will be the first element that is popped.

7-4=3
Expression
Stack

The value of the expression (7-4) that has been evaluated(3) is pushed into the stack.

Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned. End result: Postfix String:123*+4- Result: 3

ALGORITHM:

Step 1: Start the program.

Step 2: Read the postfix/suffix expression.

Step 3: Evaluate the postfix expression based on the precedence of the operator.

Step 4: Stop the program.

PROGRAM CODE:

```
#include <stdio.h>

#include <conio.h>

#include <math.h>


#define MAX 20

struct stack

{

        int top;

        float str[MAX];

}s; //stack


char postfix[MAX];//postfix void push(float);

float pop();

int isoperand(char);

float operate(float,float,char);
```

```
void push(float);
void main()
{
        int i=0;
        float ans,op1,op2;
        printf("\n Enter Expression: ");
        scanf("%s",postfix);

        while(postfix[i]!='\0')
        {
                if(isoperand(postfix[i]))
                        push(postfix[i]-48);
                else
                {
                        op1=pop();
                        op2=pop();
                        ans=operate(op1,op2,postfix[i]);
                        push(ans);
                        printf("%f %c %f = %f\n",op2,postfix[i],op1,ans);
                }
                i++;
        }
        printf("%f",s.str[s.top]);
        getch();
}

int isoperand(char x)
{
        if(x>='0' && x<='9') return 1;
        else return 0;
}

void push(float x)
{
        if(s.top==MAX-1)
                printf("Stack is full\n Stack overflow\n");
        else
        {
                s.top++;
                s.str[s.top]=x;
        }
}

float pop()
{
        if(s.top==-1)
        {
                printf(" Stack is emplty\n STACK UNDERFLOW\n");
                getch();
        }
        else
        {
                s.top--;
                return s.str[s.top+1];
        }
}

float operate(float op1,float op2,char a)
{
        switch(a)
        {
                case '+' : return op2+op1;
                case '-' : return op2-op1;
                case '*' : return op2*op1;
                case '/' : return op2/op1;
                case '^' : return pow(op2,op1);
        }
}
```

SAMPLE OUTPUT 1:

```
Enter Expression:123*+4-
2.000000 * 3.000000 = 6.000000
1.000000 + 6.000000 = 7.000000
7.000000 - 4.000000 = 3.000000
3.000000
```

SAMPLE OUTPUT 2:

```
Insert a postfix notation :: 22^32*+
2.000000 ^ 2.000000 = 4.000000
3.000000 * 2.000000 = 6.000000
4.000000 + 6.000000 = 10.000000
10.000000
```

SAMPLE OUTPUT 3:

```
Insert a postfix notation :: 23+
2.000000 + 3.000000 = 5.000000
5.000000
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully.

B) Towers of Hanoi

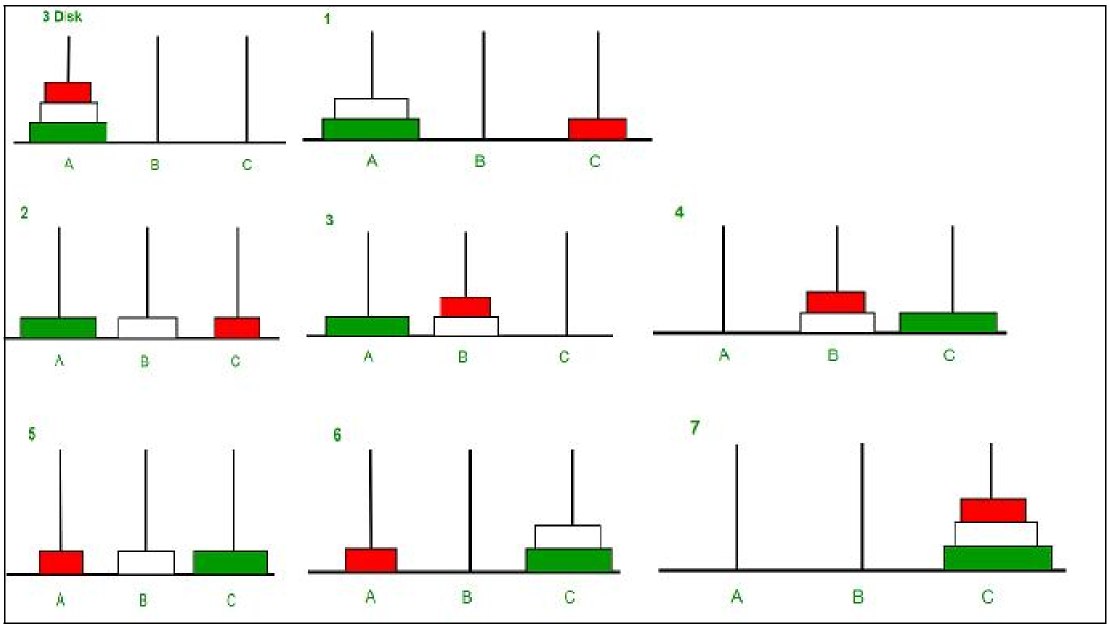
Solving Tower of Hanoi problem with n disks. The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the followingsimple rules:

• Only one disk can be moved at a time.

• Each move consists of taking the upper disk from one of the stacks and placing it on top ofanother stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

• No disk may be placed on top of a smaller disk. With three disks, the puzzle can be solvedin seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzleis **2^n - 1**, where n is the number of disks.



ALGORITHM:

Step 1: Start.

Step 2: Read N number of discs.

Step 3: Move all the discs from source to destination by using temp rod.

Step 4: Stop.

PROGRAM CODE:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void tower(int n, int source, int temp,int destination)
{
        if(n == 0) return;
        tower(n-1, source, destination, temp);
        printf("\n Move disc %d from %c to %c", n, source, destination);
        tower(n-1, temp, source, destination);
}
void main()
{
        int n;
        clrscr();
        printf("\n Enter the number of discs: \n");
        scanf("%d", &n);
        tower(n, 'A', 'B', 'C');
        printf("\n\n Total Number of moves are: %d", (int)(pow(2,n)-1));
        getch();
}
```

SAMPLE OUTPUT 1:

```
Enter the number of discs: 3
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C
Total Number of moves are: 7
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 6

Develop 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.

ABOUT THE EXPERIMENT:

Circular queue is a linear data structure. It follows FIFO principle. In circular queue the last node is connected back to the first node to make a circle. Circular linked list fallow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the queue. The queue is considered as a circular queue when the positions 0 and MAX1 are adjacent. Any position before front is also after rear. A circular queue looks like:

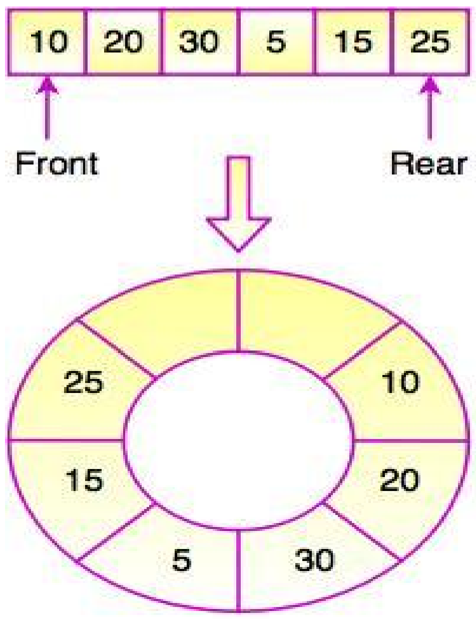
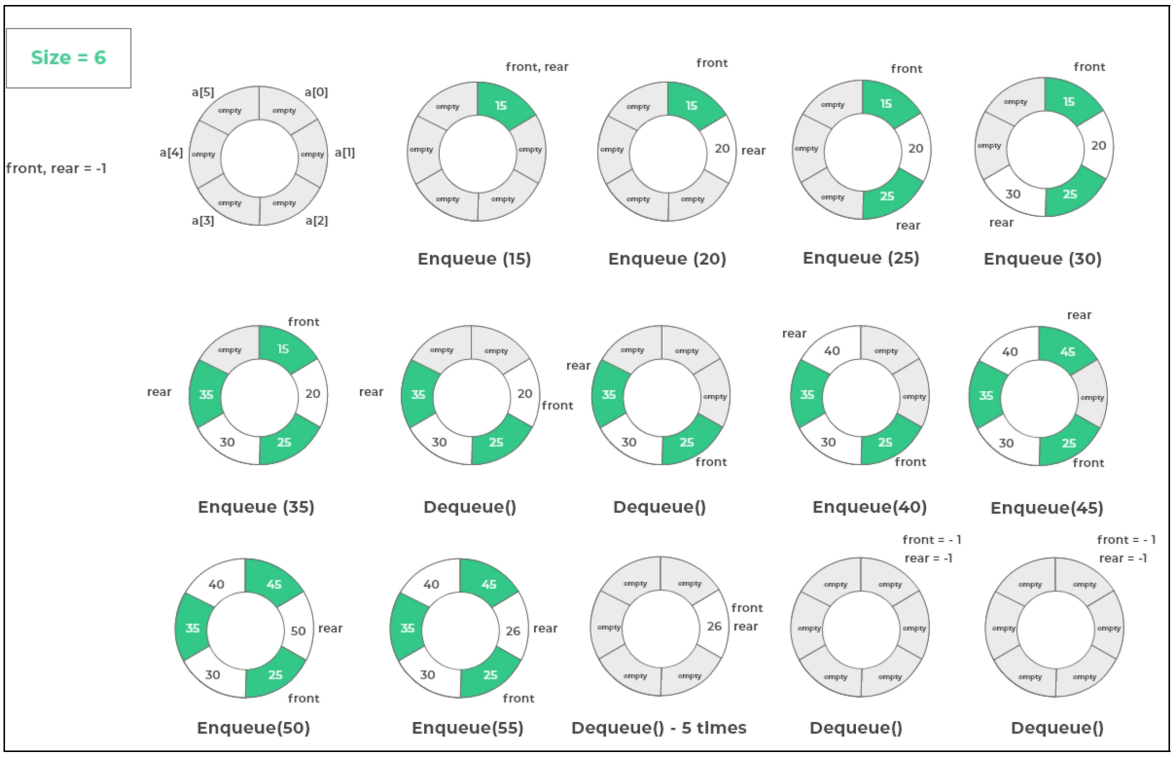


Fig. Circular Queue

Consider the example with Circular Queue implementation:



ALGORITHM:

Step 1: Start.

Step 2: Initialize queue size to MAX.

Step 3: Insert the elements into circular queue. If queue is full give a message as "queue is overflow"

Step 4: Delete an element from the circular queue. If queue is empty give a message as

"queue is underflow".

Step 5: Display the contents of the queue.

Step 6: Stop.

PROGRAM CODE:

```
#include <stdio.h>

#include <conio.h>


#define SIZE 5

int CQ[SIZE];

int front=-1;
int rear=-1, ch;
int IsCQ_Full();
int IsCQ_Empty();

void CQ_Insert(int );
void CQ_Delet();
void CQ_Display();

void main()
{
    int ch, ele;
    printf("\n 1.Insert \n 2.Delete \n 3.Display \n 4.Exit \n");
    while(1)
    {
        printf("\n Enter your choice \n");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            if(IsCQ_Full())
                printf("\n Circular Queue Overflow \n");
            else
            {
                printf("\n Enter the element to be inserted \n");
                scanf("%d",&ele);
                CQ_Insert(ele);
            }
            break;
        case 2:
            if(IsCQ_Empty())
                printf("\n Circular Queue Underflow \n");
            else
                CQ_Delet();
            break;
            case 3:
                    if(IsCQ_Empty())
                            printf("\n Circular Queue Underflow \n");
                    else
                            CQ_Display();
                    break;
            case 4:
                    exit(0);
            }
    }
}

void CQ_Insert(int item)
{
    if(front==-1)
            front++;
    rear = (rear+1)%SIZE;
    CQ[rear] =item;
}

void CQ_Delet()
{
    int item;
    item=CQ[front];
    printf("\n Deleted element is: %d",item);
    front = (front+1)%SIZE;
}

void CQ_Display()
{
    int front_pos = front,rear_pos = rear;
    if(front == -1)
    {
    printf("\n Queue is empty \n");
```

```
        return;
        }
        printf("\n Elements of the circular queue are.. : ");
        if( front_pos <= rear_pos )
        while(front_pos <= rear_pos)
        {
                printf("%d ",CQ[front_pos]);
                front_pos++;
        }
        else
        {
                while(front_pos <= SIZE-1)
                {
                        printf("%d ",CQ[front_pos]);
                        front_pos++;
                }
                front_pos = 0;
                while(front_pos <= rear_pos)
                {
                        printf("%d ",CQ[front_pos]);
                        front_pos++;
                }
        }
        printf("\n");
}

int IsCQ_Full()
{
        if(front ==(rear+1)%SIZE) return 1;
        return 0;
}

int IsCQ_Empty()
{
        if(front == -1)
        return 1;
        else if(front == rear)
        {
                printf("Deleted element is: %d",CQ[front]);
                front=-1;
                return 1;
        }
        return 0;
}
```

SAMPLE OUTPUT 1:

```
Circular Queue operations
        1. Insert
        2. Delete
        3. Display
        4. Exit
Enter your choice:1
Enter element to be insert:10
Enter your choice:1
Enter element to be insert:20
Enter your choice:1
Enter element to be insert:30

Enter your choice:3
Elements of the circular queue are.. :
10 20 30

Enter your choice:2
Deleted element is:10

Enter your choice:3
Elements of the circular queue are.. :
```

```
20 30
Enter your choice:4
Exit
```

SAMPLE OUTPUT 2:

```
Circular Queue operations
        1. Insert
        2. Delete
        3. Display
        4. Exit
Enter your choice:1
Enter element to be insert:1000
Enter your choice:1
Enter element to be insert:2000
Enter your choice:1
Enter element to be insert:3000

Enter your choice:3
Elements of the circular queue are.. :
1000 2000 3000

Enter your choice:2
Deleted element is:1000

Enter your choice:3
Elements of the circular queue are.. :
2000 3000

Enter your choice:4
Exit
```

Result:

Thus, the given 'C' program is written, executed and tested successfully

EXPERIMENT: 7

Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) 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

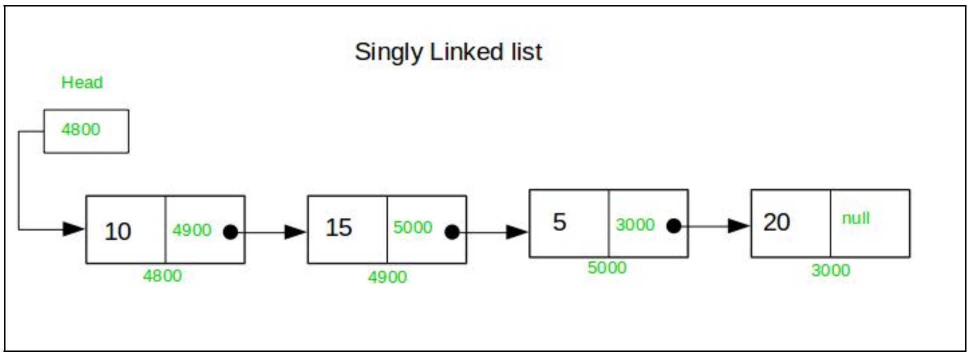
c. Perform Insertion / Deletion at End of SLL

d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)

e. Exit

ABOUT THE EXPERIMENT:

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create treesand graphs.

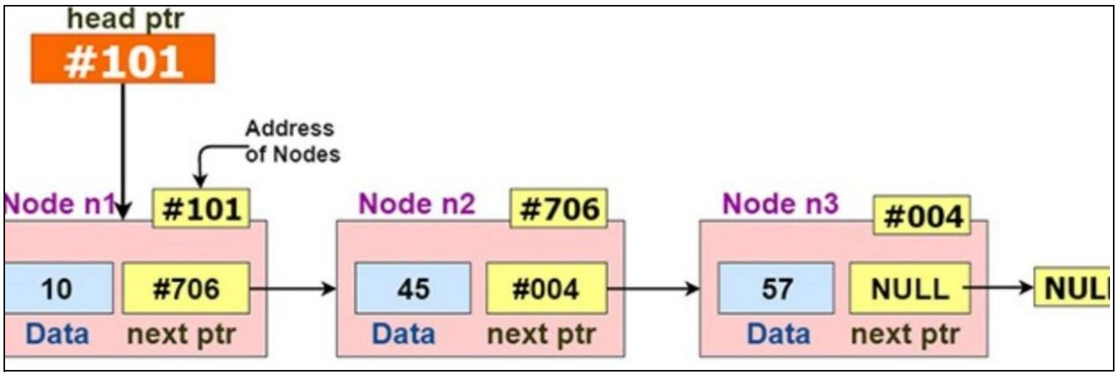


- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don"t need to know the size in advance.

Types of Linked List:

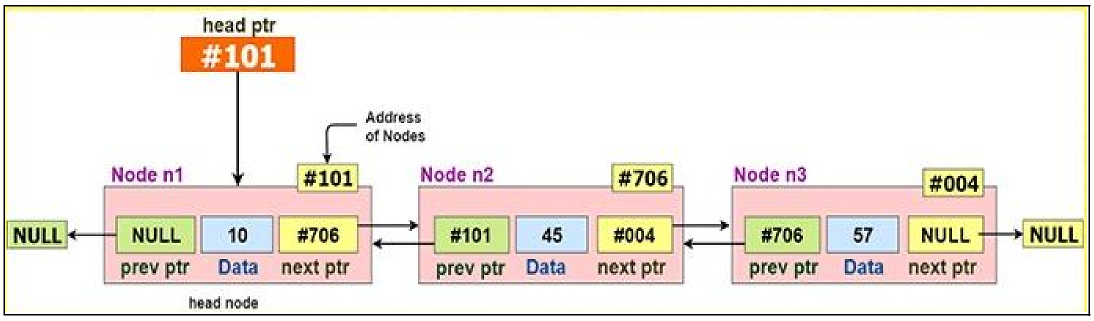
Singly Linked List:

Singly linked lists contain nodes which have a datapart as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



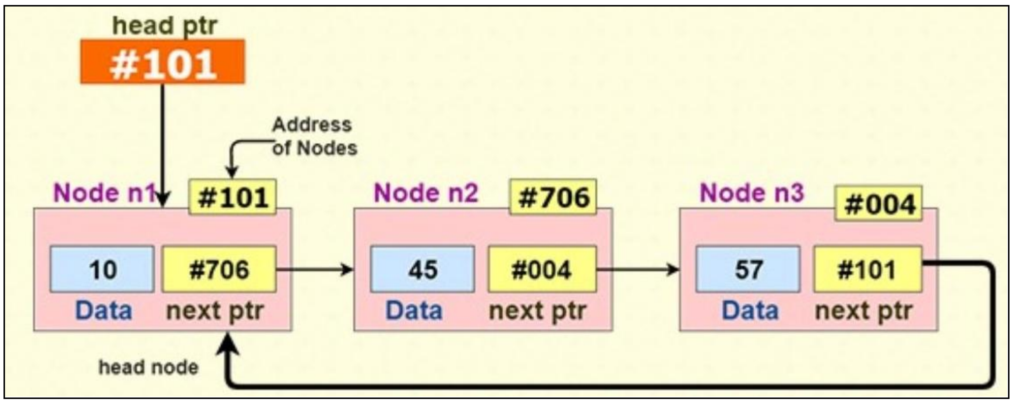
Doubly Linked List:

In a doubly linked list, each node contains two links the first link pointsto the previous node and the next link points to the next node in the sequence.



Circular Linked List:

In the circular linked list the last node of the list contains the address ofthe first node and forms a circular chain.



ALGORITHM:

Step 1: Start.

Step 2: Read the value of N. (N student"s information)

Step 3: Create a singly linked list. (SLL)

Step 4: Display the status of SLL.

Step 5: Count the number of nodes.

Step 6: Perform insertion at front of list.

Step 7: Perform deletion at the front of the list.

Step 8: Perform insertion at end of the list.

Step 9: Perform deletion at the end of the list.

Step 10: Demonstrate how singly linked list can be used as stack.

Step 11: Demonstrate how singly linked list can be used as queue.

Step 12: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>

typedef struct
{
        int usn;
        char name[20];
        char branch[20];
        int semester;
        char phone[20];
}STUDENT;

struct node
{
        int usn;
        char name[20];
        char branch[20];
        int semester;
        char phone[20];
        struct node *link;
};
```

```
typedef struct node*NODE;

NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        if(x==NULL)
        {
                printf("\n Out of memory\n");
                exit(0);
        }
        return x;
}

NODE insert_front(STUDENT item,NODE first)
{
        NODE temp;
        temp=getnode();
        temp->usn=item.usn;
        strcpy(temp->name,item.name);
        strcpy(temp->branch,item.branch);
        temp->semester=item.semester;
        strcpy(temp->phone,item.phone);
        temp->link=NULL;
        if(first==NULL)
                return temp;
        temp->link=first;
        return temp;
}

NODE insert_rear(STUDENT item,NODE first)
{
        NODE temp,cur;
        temp=getnode();
        temp->usn=item.usn;
        strcpy(temp->name,item.name);
        strcpy(temp->branch,item.branch);
        temp->semester=item.semester;
        strcpy(temp->phone,item.phone);
        temp->link=NULL;
        if(first==NULL)
                return temp;
        cur=first;
        while(cur->link!=NULL)
        {
                cur=cur->link;
        }
        cur->link=temp;
        return first;
}

NODE delete_front(NODE first)
{
        NODE temp;
        if(first==NULL)
        {
                printf("\n Student list is empty … \n");
                return NULL;
        }
        temp=first;
        temp=temp->link;
        printf("\n Delete student record: USN=%d\n",first->usn);
        free(first);
        return temp;
}

NODE delete_rear(NODE first)
{
        NODE cur, prev;
        if(first==NULL)
        {
                printf("\n Student list is empty … \n");
                return first;
        }
        if(first->link==NULL)
        {
                printf("\n Delete student record: USN=%d\n",first->usn);
                free(first);
                return NULL;
        }
        prev=NULL; cur=first;
        while(cur->link!=NULL)
        {
                prev=cur;
                cur=cur->link;
        }
        printf("\n Delete student record: USN=%d\n",cur->usn);
        free(cur);
        prev->link=NULL;
        return first;
}

void display(NODE first)
{
        NODE cur;
        int count=0;
        if(first==NULL)
        {
                printf("\n Student list is empty ...\n");
                return;
        }
        cur=first;
        while(cur!=NULL)
        {
                printf("%d\t%s\t%s\t%d\t%s\t\n",cur->usn,cur->name,cur->branch,cur->semester,cur->phone);
                cur=cur->link;
                count++;
        }
        printf("\n Number of students= %d\n",count);
}
```

```
void main()
{
        NODE first;
        int choice;
        STUDENT item;
        clrscr();
        first=NULL;
        for(;;)
        {
        printf("\n 1. Insert_front\n 2. Insert_rear\n 3. Delete_front\n 4. Delete_rear\n 5. Display \n 6. Exit\n");
        printf("\n Enter the choice : \n");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:
                        printf("USN :\n");
                        scanf("%d",&item.usn);
                        printf("Name :\n");
                        scanf("%s",item.name);
                        printf("Branch :\n");
                        scanf("%s",item.branch);
                        printf("Semester:\n");
                        scanf("%d",&item.semester);
                        printf("Phone :\n");
                        scanf("%s",item.phone);
                        first=insert_front(item,first);
                        break;
                case 2:
                        printf("USN :\n");
                        scanf("%d",&item.usn);
                        printf("Name :\n");
                        scanf("%s",item.name);
                        printf("Branch :\n");
                        scanf("%s",item.branch);
                        printf("Semester:\n");
                        scanf("%d",&item.semester);
```

```
                    printf("Phone :\n");
                    scanf("%s",item.phone);
                    first=insert_rear(item,first);
                    break;
            case 3:
                    first=delete_front(first);
                    break;
            case 4:
                    first=delete_rear(first);
                    break;
            case 5:
                    display(first);
                    break;
            default:
                    exit(0);
        }
        }
}
```

SAMPLE OUTPUT 1:

```
1. Insert_front
2. Insert_rear
3. Delete_front
4. Delete_rear
5. Display
6. Exit

Enter choice : 1
USN : 100
Name : Ram
Branch :  CSE
Semester: 3
Phone : 29292029

Enter choice : 2
USN : 200
Name : Siva
Branch :  ECE
```

```
Semester: 3
Phone : 32323232

Enter choice : 5
100  Ram   CSE  3  29292029
200 Siva   ECE  3  32323232
Number of students = 2

Enter choice : 1
USN : 50
Name : Ravi
Branch :  CSE
Semester: 3
Phone : 12121314

Enter choice : 5
50   Ravi  CSE  3  2121314
100  Ram   CSE  3  29292029
200 Siva   ECE  3  32323232
Number of students = 3

Enter choice : 2
USN : 300
Name : Rani
Branch :  EEE
Semester: 3
Phone : 40432567

Enter choice : 5
50   Ravi  CSE  3  2121314
100  Ram   CSE  3  29292029
200 Siva   ECE  3  32323232
300  Rani   EEE  3 40432567
Number of students = 4

Enter choice : 3
Delete student record: USN= 50
```

```
Enter choice : 5
100  Ram   CSE  3  29292029
200 Siva   ECE  3  32323232
300  Rani   EEE  3 40432567
Number of students = 3

Enter choice : 4
Delete student record: USN= 300

Enter choice : 5
100  Ram   CSE  3  29292029
200 Siva   ECE  3  32323232
Number of students = 2

Enter choice : 3
Delete student record: USN= 100

Enter choice : 5
200 Siva   ECE  3  32323232
Number of students = 1

Enter choice : 3
Delete student record: USN= 200

Enter choice : 5
Student list is empty …
Enter choice : 3
Student list is empty …
Enter choice : 4
Student list is empty …

Enter choice : 6
Exit
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 8

Develop 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

ABOUT THE EXPERIMENT:

Doubly Linked List: In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence. In computer science, a doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.

The beginning and ending nodes' previous and next links, respectively, point to some kind ofterminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders. A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.

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 firstnodes) because there is no need to keep track of the previous node during traversal or no needto traverse the list to find the previous node, so that its link can be modified.

ALGORITHM:

Step 1: Start.

Step 2: Read the value of N. (N student‟s information)

Step 3: Create a doubly linked list. (DLL)

Step 4: Display the status of DLL. Step 5: Count the number of nodes.

Step 6: Perform insertion at front of list.

Step 7: Perform deletion at the front of the list. Step 8: Perform insertion at end of the list.

Step 9: Perform deletion at the end of the list.

Step 10: Demonstrate how doubly linked list can be used as double ended queue.

Step 11: Stop.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Enode
{
        char ssn[15];
        char name[20];
        char dept[5];
        char designation[10];
        long int salary;
        long long int phno;
        struct Enode *left;
        struct Enode *right;
}*head=NULL;

struct Enode *tail,*temp1,*temp2;

void create(char [],char [],char [],char [],long int,long long int);
void ins_beg(char [],char [],char [],char [],long int,long long int);
void ins_end(char [],char [],char [],char [],long int,long long int);
void del_beg();
void del_end();
void display();
int count=0;

void main()
{
        int choice;
        char s[15],n[20],dpt[5],des[10];
        long int sal;
        long long int p;
        printf("\n 1. Create");
        printf("\n 2. Display");
        printf("\n 3. Insert at Beginning");
        printf("\n 4. Insert at End");
        printf("\n 5. Delete at Beginning");
        printf("\n 6. Delete at End");
        printf("\n 7. Exit\n");
        while(1)
        {
        printf("\n Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
                printf("\n Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): \n");
                scanf("%s%s%s%s%ld%lld",s,n,dpt,des,&sal,&p);
                create(s,n,dpt,des,sal,p);
                break;
        case 2:
                display();
                break;
        case 3:
                printf("\n Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone):\n");
                scanf("%s%s%s%s%ld%lld",s,n,dpt,des,&sal,&p);
                ins_beg(s,n,dpt,des,sal,p);
                break;
        case 4:
                printf("\n Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone):\n");
                scanf("%s%s%s%s%ld%lld",s,n,dpt,des,&sal,&p);
                ins_end(s,n,dpt,des,sal,p);
                break;
        case 5:
                del_beg();
                break;
        case 6:
                del_end();
```

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

void create(char s[15],char n[20],char dpt[5],char des[10],long int sal,long long int p)
{
        if(head==NULL)
        {
        head=(struct Enode *)malloc(1*sizeof(struct Enode));
        strcpy(head->ssn,s);
        strcpy(head->name,n);
        strcpy(head->dept,dpt);
        strcpy(head->designation,des);
        head->salary=sal;
        head->phno=p;
        head->left=NULL;
        head->right=NULL;
        tail=head;
        }
        else
        {
        temp1=(struct Enode *)malloc(1*sizeof(struct Enode));
        strcpy(temp1->ssn,s);
        strcpy(temp1->name,n);
        strcpy(temp1->dept,dpt);
        strcpy(temp1->designation,des);
        temp1->salary=sal;
        temp1->phno=p;
        tail->right=temp1;
        temp1->right=NULL;
        temp1->left=tail;
        tail=temp1;
        }
}
```

```
void display()
{
        temp1=head;
        if(temp1==NULL)
        {
                printf("\n No Employee Data ... ");
                return;
        }
        printf(" Employee Details \n");
        while(temp1!=NULL)
        {
        printf(" \n");
        printf(" Emp. No: %s\n Name: %s\n Dept : %s\n Desig : %s\n Sal: %ld\n Phone: %lld\n",
           temp1->ssn,temp1->name, temp1->dept,temp1->designation,temp1->salary,temp1->phno);
        printf(" ");
        temp1=temp1->right;
        }
}

void ins_beg(char s[15],char n[20],char dpt[5],char des[10],long int sal,long long int p)
{
        temp1=(struct Enode *)malloc(1*sizeof(struct Enode));
        strcpy(temp1->ssn,s);
        strcpy(temp1->name,n);
        strcpy(temp1->dept,dpt);
        strcpy(temp1->designation,des);
        temp1->salary=sal;
        temp1->phno=p;
        temp1->right=head;
        head->left=temp1;
        head=temp1;
        temp1->left=NULL;
}

void ins_end(char s[15],char n[20],char dpt[5],char des[10],long int sal,long long int p)
{
        temp1=(struct Enode *)malloc(1*sizeof(struct Enode));
        strcpy(temp1->ssn,s);
        strcpy(temp1->name,n);
        strcpy(temp1->dept,dpt);
        strcpy(temp1->designation,des);
        temp1->salary=sal;
        temp1->phno=p;
        tail->right=temp1;
        temp1->left=tail;
        temp1->right=NULL;
        tail=temp1;
}

void del_beg()
{
        temp1=head;
        if(temp1==NULL)
        {
                printf("\n No Employee Data ... ");
                return;
        }
        temp1=head->right;
        free(head);
        head=temp1;
        head->left=NULL;
        printf("\n One Data removed ... ");
}

void del_end()
{
        temp1=head;
        if(temp1==NULL)
        {
                printf("\n No Employee Data ... ");
                return;
        }
        temp1=tail->left;
        free(tail);
        tail=temp1;
        tail->right=NULL;
        printf("\n One Data removed ... ");
}
```

SAMPLE OUTPUT 1:

```
1. Create
2. Display
3. Insert at Beginning
4. Insert at End
5. Delete at Beginning
6. Delete at End
7. Exit

Enter your choice : 1
Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone):
101
Raju
CSE
AP
45000
23452456

Enter your choice : 2
Emp. No: 101
Name: Raju
Dept : CSE
Desig : AP
Sal: 45000
Phone: 23452456

Enter your choice : 3
Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone):
100
Ram
ECE
```

```
AP
40000
34252672

Enter your choice : 2
Emp. No: 100
Name: Ram
Dept : ECE
Desig : AP
Sal: 40000
Phone: 34252672

Emp. No: 101
Name: Raju
Dept : CSE
Desig : AP
Sal: 45000
Phone: 23452456

Enter your choice : 4
Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone):
102
Selvi
CSE
AP
43000
785768456

Enter your choice : 2
Emp. No: 100
Name: Ram
Dept : ECE
Desig : AP
Sal: 40000
Phone: 34252672
```

```
Emp. No: 101
Name: Raju
Dept : CSE
Desig : AP
Sal: 45000
Phone: 23452456

Emp. No: 102
Name: Selvi
Dept : CSE
Desig : AP
Sal: 43000
Phone: 785768456

Enter your choice : 5
One Data removed ...

Enter your choice : 2
Emp. No: 101
Name: Raju
Dept : CSE
Desig : AP
Sal: 45000
Phone: 23452456

Emp. No: 102
Name: Selvi
Dept : CSE
Desig : AP
Sal: 43000
Phone: 785768456

Enter your choice : 6
One Data removed ...
```

```
Enter your choice : 2
Emp. No: 101
Name: Raju
Dept : CSE
Desig : AP
Sal: 45000
Phone: 23452456

Enter your choice : 5
One Data removed ...

Enter your choice : 2
No Employee Data ...

Enter your choice : 5
No Employee Data ...

Enter your choice : 6
No Employee Data ...

Enter your choice : 7
<Exit>
```

Result:

Thus, the given 'C' program is written, executed and tested successfully

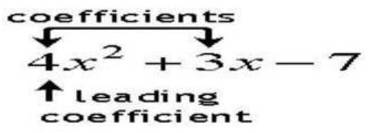
EXPERIMENT: 9

Develop 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) = 6x 2 y 2 z-4yz 5 +3x 3 yz+2xy 5 z-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.

ABOUT THE EXPERIMENT:

Circular Linked List: In the circular linked list the last node of the list contains the address of the first node and forms a circular chain. Polynomial: A polynomial equation is an equation that can be written in the form. ax n + bx n-1 + . . . + rx + s = 0, where a, b, . . . , r and s are constants.

We call the largest exponent of x appearing in a non-zero term of a polynomial the degree of that polynomial. As with polynomials with one variable, you must pay attention to the rules of exponents and the order of operations so that you correctly evaluate an expression with two or more variables. Evaluate x 2 + 3y 3 for x = 7 and y = −2. Substitute the given values for x and y. Evaluate 4x 2 y – 2xy 2 + x – 7 for x = 3 and y = −1. When a term contains both anumber and a variable part, the number part is called the "coefficient". The coefficient on the leading term is called the "leading" coefficient.



In the above example, the coefficient of the leading term is 4; the coefficient of the second term is 3; the constant term doesn't have a coefficient. Here are the steps required for Evaluating Polynomial Functions:

Step 1: Replace each x in the expression with the givenvalue.

Step 2: Use the order of operation to simplify the expression.

Example 1

$$\text{Given } f(x) = -2x^2 + 5x - 7, \text{find } f(3).$$

Step 1: Replace each x in the expression with the given value. In this case, we replace each x with 3.

$$f(3) = -2(3)^2 + 5(3) - 7$$

Step 2: Use the order of operation to simplify the expression.

$$f(3) = -2(9) + 5(3) - 7$$
$$f(3) = -18 + 15 - 7$$
$$f(3) = -10$$

Here are the steps required for addition of two polynomials.

Step 1

- Arrange the Polynomial in standard form
- Standard form of a polynomial and each of the following terms just means that the term with highest degree is first

Step 2

- Arrange the like terms in columns and add the like terms
- **Example 1:**

Let's find the sum of the following two polynomials

$(3x^5 + 2x + x^4 + 2x^3 + 5)$ and $(2x^5 + 3x^3 + 2 + 7x)$

Write in the standard form:

$(3x^5 + x^4 + 2x^3 + 2x + 5) + (2x^5 + 3x^3 + 7x + 2)$

Arrange in columns of like terms and then add:

```
3x^5 +  x^4 + 2x^3 + 2x + 5
2x^5        + 3x^3 + 7x + 2
--------------------------------
5x^5 +  x^4 + 5x^3 + 9x + 7
```

ALGORITHM:

Step 1: Start the program.

Step 2: Read a polynomial.

Step 3: Represent the polynomial using singly circular linked list.

Step 4: Evaluate the given polynomial

Step 5: Read two polynomials and find the sum of the polynomials.

Step 6: Stop the program

PROGRAM CODE:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
struct node
{
        int coeff;
        int expo;
        struct node *ptr;
};

struct node *head1,*head2,*head3, *temp,*temp1,*temp2,*temp3,*list1,*list2,*list3;
struct node *dummy1,*dummy2;

void create_poly1(int , int);
void create_poly2(int , int);
void display();
void add_poly();
void eval_poly(int );
int n,ch;
int c,e,i;
void main()
{
        int x;
        list1=list2=list3=NULL;
        printf("\n 1. Create first polynomial");
        printf("\n 2. Create Second Polynomial");
        printf("\n 3. Display both the polynomials");
        printf("\n 4. Add Polynomials");
        printf("\n 5. Evaluate a Polynomial");
        printf("\n 6. Exit");
        while(1)
        {
                printf("\n Enter choice: ");
                scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                        printf("\n Enter the number of terms: ");
```

```
                    scanf("%d",&n);
                    printf("\n Enter coefficient & power of each term : ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d %d",&c,&e);
                        create_poly1(c,e);
                    }
                    break;
                case 2:
                    printf("\n Enter the number of terms: ");
                    scanf("%d",&n);
                    printf("\n Enter coefficient & power of each term: ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d %d",&c,&e);
                        create_poly2(c,e);
                    }
                    break;
                case 3:
                    display();
                    break;
                case 4:
                    add_poly();
                    break;
                case 5:
                    printf("\n Enter the value for x : ");
                    scanf("%d",&x);
                    eval_poly(x);
                    break;
                case 6:
                    exit(0);
                }
        }
}

void create_poly1(int c, int e)
{
        dummy1=(struct node*)malloc(1*sizeof(struct node));
        dummy1->coeff=0;
        dummy1->expo=0;
        dummy1->ptr=list1;
        if(list1==NULL)
```

```
	{
		list1=(struct node*)malloc(1*sizeof(struct node));
		list1->coeff=c;
		list1->expo=e;
		list1->ptr=list1;
		head1=list1;
		head1->ptr=dummy1;
	}
	else
	{
		temp=(struct node*)malloc(1*sizeof(struct node));
		temp->coeff=c;
		temp->expo=e;
		head1->ptr=temp;
		temp->ptr=dummy1;
		head1=temp;
	}
}

void create_poly2(int c, int e)
{
	dummy2=(struct node*)malloc(1*sizeof(struct node));
	dummy2->coeff=0;
	dummy2->expo=0;
	dummy2->ptr=list2;
	if(list2==NULL)
	{
		list2=(struct node*)malloc(1*sizeof(struct node));
		list2->coeff=c;
		list2->expo=e;
		list2->ptr=list2;
		head2=list2;
		head2->ptr=dummy2;
	}
	else
	{
		temp=(struct node*)malloc(1*sizeof(struct node));
		temp->coeff=c;
		temp->expo=e;
		head2->ptr=temp;
		temp->ptr=dummy2;
		head2=temp;
```

```
        }
}

void add_poly()
{
        temp1=list1;
        temp2=list2;
        while((temp1!=dummy1)&&(temp2!=dummy2))
        {
                temp=(struct node*)malloc(1*sizeof(struct node));
                if(list3==NULL)
                {
                        list3=temp;
                        head3=list3;
                }
                if(temp1->expo==temp2->expo)
                {
                        temp->coeff=temp1->coeff+temp2->coeff;
                        temp->expo=temp1->expo;
                        temp->ptr=list3;
                        head3->ptr=temp;
                        head3=temp;
                        temp1=temp1->ptr;
                        temp2=temp2->ptr;
                }
                else if(temp1->expo>temp2->expo)
                {
                        temp->coeff=temp1->coeff;
                        temp->expo=temp1->expo;
                        temp->ptr=list3;
                        head3->ptr=temp;
                        head3=temp;
                        temp1=temp1->ptr;
                }
                else
                {
                        temp->coeff=temp2->coeff;
                        temp->expo=temp2->expo;
                        temp->ptr=list3;
                        head3->ptr=temp;
                        head3=temp;
                        temp2=temp2->ptr;
                }
        }
        if(temp1==dummy1)
        {
                while(temp2!=dummy2)
                {
                        temp=(struct node*)malloc(1*sizeof(struct node));
                        temp->coeff=temp2->coeff;
                        temp->expo=temp2->expo;
                        temp->ptr=list3;
                        head3->ptr=temp;
                        head3=temp;
                        temp2=temp2->ptr;
                }
        }
        if(temp2==dummy2)
        {
                while(temp1!=dummy1)
                {
                        temp=(struct node*)malloc(1*sizeof(struct node));
                        temp->coeff=temp1->coeff;
                        temp->expo=temp1->expo;
                        temp->ptr=list3;
                        head3->ptr=temp;
                        head3=temp;
                        temp1=temp1->ptr;
                }
        }

        temp3=list3;
        printf("\n\n Sum of Polynomials 1 and 2 : ");
        while(temp3->ptr!=list3)
        {
                printf(" %dX^%d+",temp3->coeff,temp3->expo);
                temp3=temp3->ptr;
        }
        printf(" %dX^%d\n",temp3->coeff,temp3->expo);
        list3=NULL;
}

void display()
{
        temp1=list1;
        temp2=list2;
        printf("\n \n Polynomial - 1: ");
        while(temp1!=dummy1)
        {
                printf(" %dX^%d+",temp1->coeff,temp1->expo);
                temp1=temp1->ptr;
        }
        printf("\n \n Polynomial - 2: ");
        while(temp2!=dummy2)
        {
                printf(" %dX^%d+",temp2->coeff,temp2->expo);
                temp2=temp2->ptr;
        }
}

void eval_poly(int x)
{
        int result=0;
        temp1=list1;
        temp2=list2;
        while(temp1!=dummy1)
        {
                result+=(temp1->coeff)*pow(x,temp1->expo);
                temp1=temp1->ptr;
        }
        printf("\n Polynomial - 1 Evaluation: %d\n",result);
        result=0;
        while(temp2!=dummy2)
        {
                result+=(temp2->coeff)*pow(x,temp2->expo);
                temp2=temp2->ptr;
        }
        printf("\n Polynomial - 2 Evaluation: %d\n",result);
}
```

SAMPLE OUTPUT:

Sample:

$$3x^5 + x^4 + 2x^3 + 2x + 5$$
$$2x^5 \qquad + 3x^3 + 7x + 2$$

```
1. Create first polynomial
2. Create Second Polynomial
3. Display both the polynomials
4. Add Polynomials
5. Evaluate a Polynomial
6. Exit

Enter choice: 1
Enter the number of terms: 5
Enter coefficient & power of each term :
3 5
1 4
2 3
2 1
5 0

Enter choice: 2
Enter the number of terms: 4
Enter coefficient & power of each term :
2 5
3 3
7 1
2 0

Enter choice: 3
Polynomial - 1: 3x^5 + 1x^4 + 2x^3 + 2x^1 + 5x^0
Polynomial - 1: 2x^5  + 3x^3 + 7x^1 + 2x^0
```

```
Enter choice: 4
Sum of Polynomials 1 and 2 : 5x^5 + 1x^4 + 5x^3 + 9x^1 + 7x^0

Enter choice: 5
Enter the value for x : 2
Polynomial - 1 Evaluation: 137
Polynomial - 2 Evaluation: 104

Enter choice: 6
<Exit>
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 10

Develop a menu driven Program in C for the following operations on Binary SearchTree (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 In-order, Pre-order and Post-order**
- **c. Search the BST for a given element (KEY) and report the appropriate message**
- **d. Exit**

ABOUT THE EXPERIMENT:

A binary search tree (BST) is a tree in which all nodes follows the below mentioned properties:

- ✓ The left sub-tree of a node has key less than or equal to its parent node's key.
- ✓ The right sub-tree of a node has key greater than or equal to its parent node's key.
- ✓ Thus, a binary search tree (BST) divides all its sub-trees into two segments;

  left sub-tree and rightsub-tree and can be defined as:

$$\text{left\_subtree (keys)} \leq \text{node (key)} \leq \text{right\_subtree (keys)}$$

Following are basic primary operations of a tree which are following:

- ✧ Search - search an element in a tree.
- ✧ Insert - insert an element in a tree.
- ✧ Preorder Traversal - traverse a tree in a preorder manner.
- ✧ Inorder Traversal - traverse a tree in an inorder manner.
- ✧ Postorder Traversal - traverse a tree in a postorder manner.

Node definition:

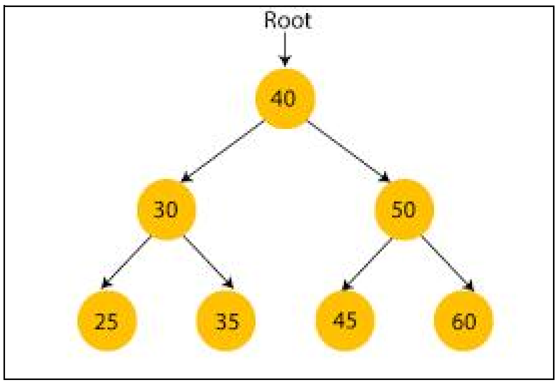
Define a node having some data, references to its left and right child nodes.

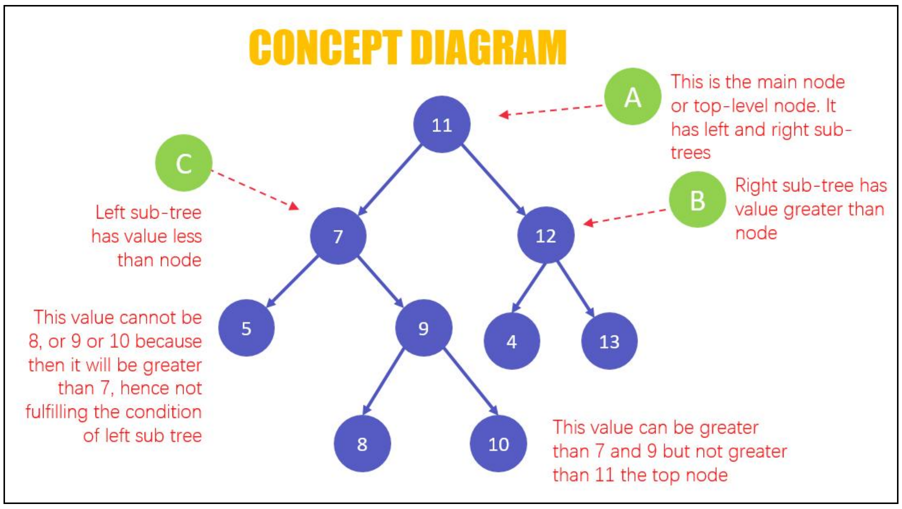
```
struct node
{
        int data;
        struct node *leftChild;

        struct node *rightChild;

    };
```

Sample image for Binary Search Tree:




CONCEPT DIAGRAM
A
This is the main node or top-level node. It has left and right sub-trees
B
Right sub-tree has value greater than node
C
Left sub-tree has value less than node
This value cannot be 8, or 9 or 10 because then it will be greater than 7, hence not fulfilling the condition of left sub tree
This value can be greater than 7 and 9 but not greater than 11 the top node
11
7
12
5
9
4
13
8
10

Attributes of Binary Search Tree:

A BST is made of multiple nodes and consists of the following attributes:

- Nodes of the tree are represented in a parent-child relationship
- Each parent node can have zero child nodes or a maximum of two subnodes or subtrees on the left and right sides.
- Every sub-tree, also known as a binary search tree, has sub-branches on the right and left of themselves.
- All the nodes are linked with key-value pairs.
- The keys of the nodes present on the left subtree are smaller than the keys of their parent node
- Similarly, The left subtree nodes' keys have lesser values than their parent node's keys.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Create a Binary Search Tree for N elements.
Step 3: Traverse the tree in inorder.
Step 4: Traverse the tree in preorder
Step 5: Traverse the tree in postorder.
Step 6: Search the given key element in the BST.
Step 7: Delete an element from BST.
Step 8: Stop the program
```

PROGRAM CODE

```c
#include <stdio.h>
#include <stdlib.h>

struct BST
{
	int data;
	struct BST *left;
	struct BST *right;
};

typedef struct BST NODE;
NODE *node;

NODE* createtree(NODE *node, int data)
{
	if (node == NULL)
	{
		NODE *temp;
		temp= (NODE*)malloc(sizeof(NODE));
		temp->data = data;
		temp->left = temp->right = NULL;
		return temp;
	}
	if (data < (node->data))
	{
		node->left = createtree(node->left, data);
```

```
        }
        else if (data > node->data)
        {
                node -> right = createtree(node->right, data);
        }
        return node;
}

NODE* search(NODE *node, int data)
{
        if(data == node->data)
        {
                printf("\n Element %d is found ", data);
        }
        /*
        if(node == NULL)
        {
                printf("\n No Data .... \n");
                return 0;
        }
        */

        else if(data < node->data)
        {
                node->left=search(node->left, data);
        }
        else if(data > node->data)
        {
                node->right=search(node->right, data);
        }
        else
        {
             printf("\n Element %d is not found ", data);
        //      return 0;
        }
        return node;
}

void inorder(NODE *node)
{
        if(node != NULL)
        {
                inorder(node->left);
                printf("%d\t", node->data);
                inorder(node->right);
        }
}

void preorder(NODE *node)
{
        if(node != NULL)
        {
                printf("%d\t", node->data);
                preorder(node->left);
                preorder(node->right);
        }
}

void postorder(NODE *node)
{
        if(node != NULL)
        {
                postorder(node->left);
                postorder(node->right);
                printf("%d\t", node->data);
        }
}

NODE* findMin(NODE *node)
{
        if(node==NULL)
        {
                return NULL;
        }
        if(node->left)
                return findMin(node->left);
        else
                return node;
}

NODE* del(NODE *node, int data)
{
        NODE *temp;
        if(data < node->data)
        {
                node->left = del(node->left, data);
        }
        else if(data > node->data)
        {
                node->right = del(node->right, data);
        }
        else
        {
                /* Now We can delete this node and replace with either minimum element
                in the right sub tree or maximum element in the left subtree */
                if(node->right && node->left)
                {
                        /* Here we will replace with minimum element in the
                           right sub tree */
                        temp = findMin(node->right);
                        node -> data = temp->data;
                        /* As we replaced it with some other node, we have to
                           delete that node */
                        node -> right = del(node->right, temp->data);
                }
                else
                {
                        /* If there is only one or zero children then we can
                                  directly remove it from the tree and connect its
                                parent to its child */
                        temp = node;
                        if(node->left == NULL)
                                node = node->right;
                        else if(node->right == NULL)
                                node = node->left;
                        free(temp); /* temp is longer required */
                        printf("\n Element is removed ... ");
                }

        }
        return node;
}

void main()
{
        int data, ch, i, n;
```

```
    NODE *root=NULL;
    while (1)
    {
        printf("\n 1. Insertion in Binary Search Tree");
        printf("\n 2. Search Element in Binary Search Tree");
        printf("\n 3. Delete Element in Binary Search Tree");
        printf("\n 4. Inorder");
        printf("\n 5. Preorder");
        printf("\n 6. Postorder");
        printf("\n 7. Exit");
        printf("\n Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\n Enter n value: " );
            scanf("%d", &n);
            printf("\n Enter the values to create BST like(6,9,5,2,8,15,24,14)\n");
            for(i=0; i<n; i++)
            {
                scanf("%d", &data);
                root=createtree(root, data);
            }
            break;
        case 2:
            printf("\n Enter the element to search: ");
            scanf("%d", &data);
            if(root == NULL)
                printf("\n No Data .... \n");
            else
                search(root, data);
            break;
        case 3:
            printf("\n Enter the element to delete: ");
            scanf("%d", &data);
            if(root == NULL)
                printf("\n No Data .... \n");
            else
                root=del(root, data);
            break;
        case 4:
            printf("\n Inorder Traversal: \n");
```

```
                if(root == NULL)
                        printf("\n No Data .... \n");
                else
                        inorder(root);
                break;
        case 5:

                printf("\n Preorder Traversal: \n");
                if(root == NULL)
                        printf("\n No Data .... \n");
                else
                        preorder(root);
                break;
        case 6:
                printf("\n Postorder Traversal: \n");
                if(root == NULL)
                        printf("\n No Data .... \n");
                else
                        postorder(root);
                break;
        case 7:
                exit(0);
        default :
                printf("\n Wrong option");
                break;
        }
    }
}
```

SAMPLE OUTPUT:

```
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit

Enter your choice: 1
Enter n value: 5
Enter the values to create BST like (6,9,5,2,8,15,24,14):
```

```
40 10 30 20 50


Enter your choice: 4
Inorder Traversal:
10    20       30       40       50

Enter your choice: 5
Preorder Traversal:
40    10       30       20       50

Enter your choice: 6
Postorder Traversal:
20    30       10       50


Enter your choice: 2
Enter the element to search: 10
Element 10 is found

Enter your choice: 2
Enter the element to search: 80
Element 80 is not found

Enter your choice: 3
Enter the element to delete: 50
Element is removed …

Enter your choice: 4
Inorder Traversal:
10    20       30       40

Enter your choice: 7
<Exit>
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully

EXPERIMENT: 11

Develop 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 DFS/BFS method

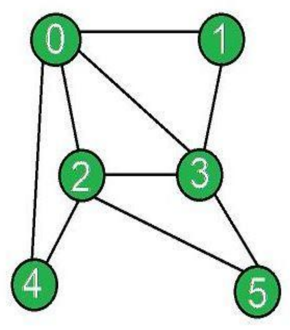
ABOUT THE EXPERIMENT:

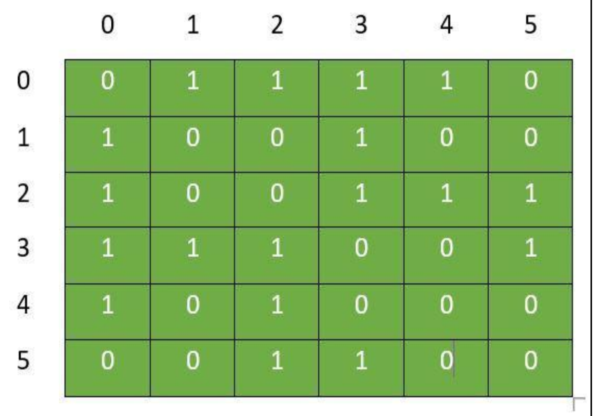
Adjacency Matrix In graph theory, computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of verticesare adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a (0, 1)-matrix with zeros on its diagonal.

A graph G = (V, E) where v= {0, 1, 2, . . .n-1} can be represented using two dimensional integer array of size n x n. a[20][20] canbe used to store a graph with 20 vertices. a[i][j] = 1, indicates presence of edge between two vertices i and j. a[i][j] = 0, indicates absence of edge between two vertices i and j.

- ✓ A graph is represented using square matrix.
- ✓ Adjacency matrix of an un-directed graph is always a symmetric matrix, i.e. an edge (i, j)implies the edge (j, i).
- ✓ Adjacency matrix of a directed graph is never symmetric, adj[i][j] = 1 indicates a directededge from vertex i to vertex j.

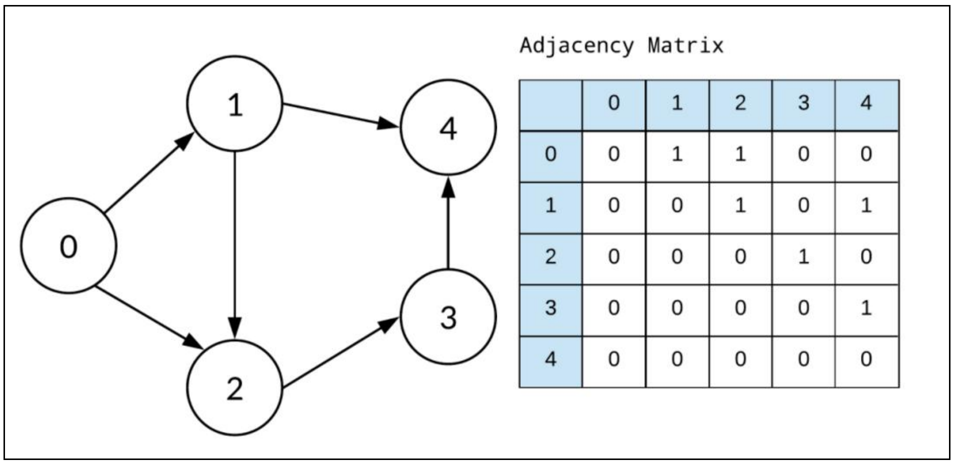
Example - Undirected Graph:





	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	1	1	1
3	1	1	1	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

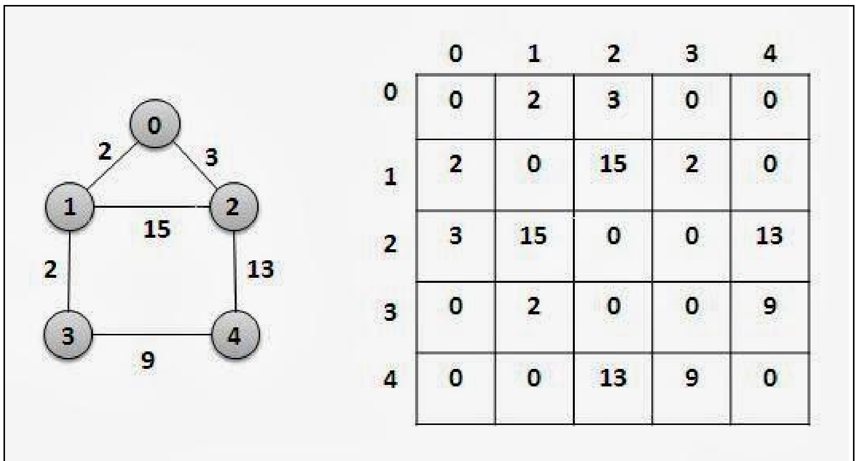
Example - Directed Graph:



Adjacency Matrix

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

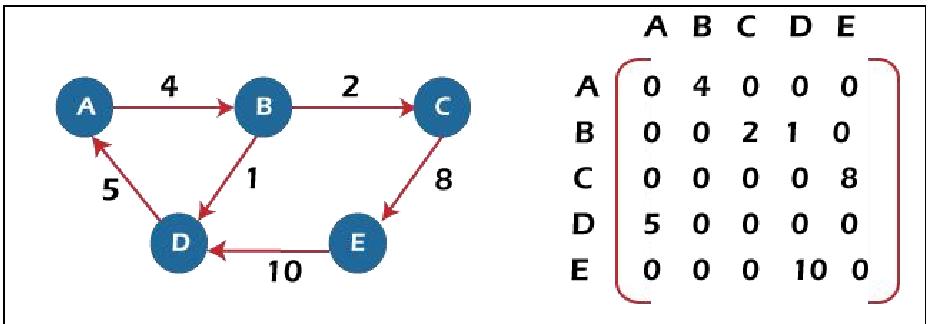
Example - Weighted Un-directed Graph:



	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Adjacency Matrix Representation of Weighted Graph

Example - Weighted Directed Graph:



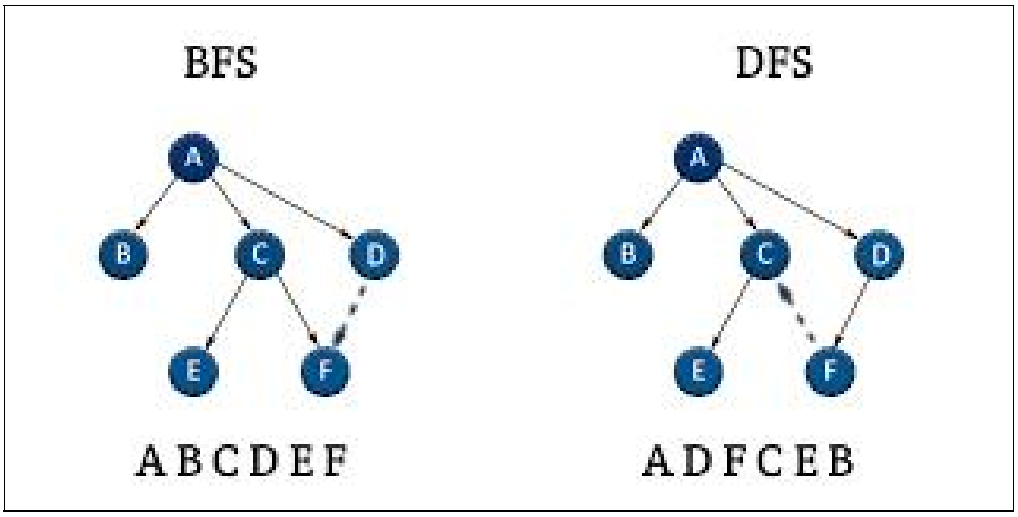
weighted Directed Graph

	A	B	C	D	E
A	0	4	0	0	0
B	0	0	2	1	0
C	0	0	0	0	8
D	5	0	0	0	0
E	0	0	0	10	0

Adjacency Matrix

BFS graph Breadth-first search (BFS) is an algorithm data structures. It starts at the tree root for traversing or searching tree or and explores the neighbor nodes first, before moving to thenext level neighbors.

Breadth First Search algorithm(BFS) traverses a graph in a breadth wards motion and uses aqueue to remember to get the next vertex to start a search when a dead end occurs in any iteration.



As in example given above, BFS algorithm traverses from A to F. It employs following rules.

Rule 1 − Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.

Rule 2 − If no adjacent vertex found, remove the first vertex from queue.

Rule 3 − Repeat Rule 1 and Rule 2 until queue is empty.

DFS Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

Depth-first search, or DFS, is a way to traverse the graph. Initially it allows visiting vertices of the graph only, but there are hundreds of algorithms for graphs, which are based on DFS. Therefore, understanding the principles of depth-first search is quite important to move ahead into the graph theory. The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack.

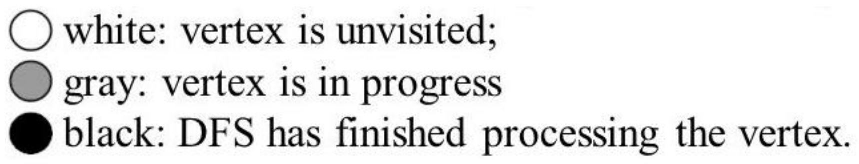
In DFS, each vertex has three possible colors representing its state:

white: vertex is unvisited;

gray: vertex is in progress;

black: DFS has finished processing the vertex.

Sample Colour:



○ white: vertex is unvisited;
◉ gray: vertex is in progress
● black: DFS has finished processing the vertex.

ALGORITHM:

Step 1: Start.

Step 2: Input the value of N nodes of the graph

Step 3: Create a graph of N nodes using adjacency matrix representation.

Step 4: Print the nodes reachable from the starting node using BFS.

Step 5: Check whether graph is connected or not using DFS.

Step 6: Stop.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int a[20][20], q[20], visited[20], reach[10], n, i, j, f=0, r=-1, count=0;

void bfs(int v)
{
        for(i=1;i<=n;i++)
        if(a[v][i] && !visited[i])
                q[++r]=i;
        if(f<=r)
        {
                visited[q[f]]=1;
                bfs(q[f++]);
        }
}

void dfs(int v)
{
        int i;
        reach[v]=1;
        for(i=1;i<=n;i++)
        {
                if(a[v][i] && !reach[i])
                {
                        printf("\n %d->%d",v,i);
                        count++;
                        dfs(i);
                }
        }
}

void main()
{
        int v, choice;
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                q[i]=0;
                visited[i]=0;
        }
        for(i=1;i<=n-1;i++)
                reach[i]=0;
        printf("\n Enter graph data in matrix form:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                scanf("%d",&a[i][j]);
        printf("\n 1. BFS \n 2. DFS \n 3. Exit\n");
        printf("\n Enter Choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
                printf("\n Enter the starting vertex:");
                scanf("%d",&v);
                bfs(v);
                if((v<1)||(v>n))
                {
                        printf("\n Bfs is not possible");
                }
                else
                {
                        printf("\n The nodes which are reachable from %d:\n",v);
                        for(i=1;i<=n;i++)
                        if(visited[i])
                        printf("%d\t",i);
                }
                break;
        case 2:
                dfs(1);
                if(count==n-1)
                        printf("\n Graph is connected ");
                else
                        printf("\n Graph is not connected ");
                break;
        case 3:
                exit(0);
        }
        getch();
}
```

Sample Output (1):

```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

1. BFS
2. DFS
3. Exit

Enter Choice : 1
Enter the starting vertex : 1
The nodes which are reachable from 1:
        1   2  3   4
```

ample Output (2):

```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 0 0
1 0 1 1
0 1 0 0
0 1 0 0

1. BFS
2. DFS
3. Exit

Enter Choice : 2
   1 - >  2
   2 - >  3
   2 - >  4
Graph is connected
```

Result:

Thus, the given 'C' program is written, executed and tested successfully

EXPERIMENT: 12

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. 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.

ALGORITHM:

Step 1: Start.

Step 2: Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F.

Step 3: 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.

Step 4: Let the keys in K and addresses in L are Integers

Step 5: Hash function H: K ®L as H(K)=K mod m (remainder method)

Step 6: Hashing as to map a given key K to the address space L, Resolve the collision (if any) is using linear probing.

Step 7: Stop.

Program Code:

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

#define MAX 10
```

```
int create(int);
void display (int[]);
void linear_prob(int [], int, int);

void main()
{
        int a[MAX],num,key,i;
        int ans=1;
        clrscr();
        printf(" \n Collision handling by linear probing : \n");
        for (i=0;i<MAX;i++)
        {
                a[i] = -1;
        }
        do
        {
                printf("\n Enter the data : ");
                scanf("%4d", &num);
                key=create(num);
                linear_prob(a,key,num);
                printf("\n Do you wish to continue ? (1/0) ");
                scanf("%d",&ans);
        }while(ans);
        display(a);
        getch();
}

int create(int num)
{
        int key;
        key=num%10;
        return key;
}

void linear_prob(int a[MAX], int key, int num)
{
    int flag, i, count=0;
    flag=0;
    if(a[key]== -1)
    {
        a[key] = num;
    }
    else
    {
        printf("\n Collision Detected...!!!\n");
        i=0;
        while(i<MAX)
        {
            if (a[i]!=-1)
            count++;
            i++;
        }
        printf("\n Collision avoided successfully using LINEAR PROBING\n");
        if(count == MAX)
        {
            printf("\n Hash table is full");
            display(a);
            exit(1);
        }
        for(i=key+1; i<MAX; i++)
        if(a[i] == -1)
        {
            a[i] = num;
            flag =1;
            break;
        }
        while((i<key) && (flag==0))
        {
            if(a[i] == -1)
            {
                a[i] = num;
                flag=1;
                break;
```

```
                    } i++;
                }
            }
}

void display(int a[MAX])
{
            int i,choice;
            printf("\n 1. Display ALL\n 2. Filtered Display\n");
            printf("\n Enter Choice : ");
            scanf("%d",&choice);

            if(choice==1)
            {
                        printf("\n The hash table is: \n");
                        for(i=0; i<MAX; i++)
                        printf("\n %d %d ", i, a[i]);
            }
            else
            {
                        printf("\n The hash table is:  \n");
                        for(i=0; i<MAX; i++)
                        if(a[i]!=-1)
                        {
                                    printf("\n %d %d ", i, a[i]);
                                    continue;
                        }
            }
}
```

Sample Output (1):

```
Collision handling by linear probing :

Enter the data : 1231
Do you wish to continue ? (1/0) 1

Enter the data : 2342
```

```
Do you wish to continue ? (1/0) 1

Enter the data : 4533
Do you wish to continue ? (1/0) 1

Enter the data : 6633
Collision Detected...!!!
Collision avoided successfully using Linear Probing

Do you wish to continue ? (1/0) 1

Enter the data : 7333
Collision Detected...!!!
Collision avoided successfully using Linear Probing

Do you wish to continue ? (1/0) 0

1. Display ALL
2. Filtered Display

Enter Choice : 1

The hash table is:
0       -1
1       1231
2       2342
3       4533
4       6633
5       4675
6       7333
7       -1
8       -1
9       -1
```

Sample Output (2):

```
Collision handling by linear probing :

Enter the data : 1231
Do you wish to continue ? (1/0) 1

Enter the data : 2342
Do you wish to continue ? (1/0) 1
```

```
Enter the data : 4533
Do you wish to continue ? (1/0) 1

Enter the data : 6633
Collision Detected...!!!
Collision avoided successfully using Linear Probing

Do you wish to continue ? (1/0) 1

Enter the data : 7333
Collision Detected...!!!
Collision avoided successfully using Linear Probing

Do you wish to continue ? (1/0) 0

1. Display ALL
2. Filtered Display
Enter Choice : 1

The hash table is:
1        1231
2        2342
3        4533
4        6633
5        4675
6        7333
```

Result:

Thus, the given ‘C’ program is written, executed and tested successfully