Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DATA STRUCTURES LABORATORY

[BCSL305]

Prepared by
Mr. Dharaneshkumar M L
Asst. Prof. Dept. of AI & DS,
CIT, Gubbi

Department of Artificial Intelligence and Data Science

III Semester

Academic Year: 2025-26

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 lists
- 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.

Programs List:

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

15.09.2023

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 $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
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: The student should be able to:

- Analyze various linear and non-linear data structures
- Demonstrate the working nature of different types of data structures and their applications
- Use appropriate searching and sorting algorithms for the give scenario.
- Apply the appropriate data structure for solving real world problems

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)*
    c) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
    d) 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

<u>Execution Steps</u>:

Step 1: Open Terminal and type the following commands

Step 2: gedit pgmname.c (To open an editor and for typing the program)

Step 3: gcc pgmname.c (To Compile the Program)

Step 4: ./a.out (To Run the Program)

PROGRAM 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).
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

THEORY:

Here is a C code to declare a calendar as an array of 7 elements, each element of which is a structure having three fields: name of the day (a dynamically allocated string), date of the day (an integer), and description of the activity for a particular day (a dynamically allocated string), and 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

- **malloc( ):**

To dynamically allocate memory we use malloc() which is the standard library function in C . It takes the size of the memory block that you want to allocate as an argument and returns a pointer to the allocated memory block.

The syntax for using malloc() is as follows:

```
void *malloc(size_t size);
```

The size argument is the size of the memory block that you want to allocate in bytes. The function returns a pointer to the allocated memory block, or NULL if the allocation fails.

- **typedef struct:**

The typedef struct keyword in C is used to create a new type name for a structure. This can be useful for making your code more readable and maintainable.

ALGORITHM :

STEP 1: Define a structure to represent a day in the calendar.

```
typedef struct
```

```
{
  char *name;
  int date;
  char *description;
} Day;
```

STEP 2:Create a create() function to dynamically create the array of 7 elements of the Day structure.It returns the calendar array.

```
    Day *calendar =(Day *) malloc(sizeof(Day) * 7);
```

STEP 3: Create a read() function to read the data from the keyboard, iterate over the calendar array and prompt the user to enter the name, date, and description for each day of the week:

```
void read(Day *calendar) {
  for (int i = 0; i < 7; i++) {
    printf("Enter the day name: ");
    scanf("%s", calendar[i].name);
    printf("Enter the date: ");
    scanf("%d", &calendar[i].date);
    printf("Enter the description: ");
    scanf("%s", calendar[i].description);
  }}
```

STEP 4: To create a display() function to print the calendar on the screen, iterate over the calendar array and print the name, date, and activity for each day of the week:

```
void display(Day *calendar) {
  printf("\n\nCalendar for the week:\n\n");
  for (int i = 0; i < 7; i++) {
    printf("%s %d: %s\n", calendar[i].name, calendar[i].date, calendar[i].description);
  }}
```

STEP 5: Use the free() function to deallocate the memory space

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct
{
        char *name;
        int date;
        char *description;
} Day;
// Function to create a calendar
 Day *create()
 {
 Day *calendar = (Day *)malloc(sizeof(Day) * 7);
 if (calendar = = NULL)
{
   printf("Error allocating memory for calendar\n");
   exit(1);
 }
 return calendar;
}
// Function to read data from the keyboard
void read(Day *calendar)
{
 for (int i = 0; i < 7; i++)
{
   printf("Enter the name of day %d: ", i + 1);
   char *name = (char *)malloc(sizeof(char) * 100);
   if (name == NULL)
   {
     printf("Error allocating memory for day name\n");
     exit(1);
   }
```

```
    scanf("%s", name);
    calendar[i].name = name;
    printf("Enter the date of day %d: ", i + 1);
    int date;
    scanf("%d", &date);
    calendar[i].date = date;
    printf("Enter the description of activity for day %d: ", i + 1);
    char *description = (char *)malloc(sizeof(char) * 100);
    if (description == NULL) {
      printf("Error allocating memory for activity description\n");
      exit(1);
    }
    scanf("%s", description);
    calendar[i].description = description;
  }
}
// Function to print weeks activity details report on screen
void display(Day *calendar)
{
  printf("\nWeek Activity Details Report\n");
  printf("----------------------------------------\n");
  for (int i = 0; i < 7; i++)
{
    printf("%s %d: %s\n", calendar[i].name, calendar[i].date, calendar[i].description);
  }
}
int main( )
{
  // Create a calendar
  Day *calendar = create();

  // Read data from the keyboard
  read(calendar);
```

```
 // Display weeks activity details report on screen
 display(calendar);

 // Free the memory allocated for the calendar
 for (int i = 0; i < 7; i++)
{
   free(calendar[i].name);
   free(calendar[i].description);
 }
 free(calendar);
 return 0;
}
```

OUTPUT:

```
Enter the name of day 1: Monday
Enter the date of day 1: 10
Enter the description of activity for day 1: Work from home

Enter the name of day 2: Tuesday
Enter the date of day 2: 11
Enter the description of activity for day 2: Meet with client

Enter the name of day 3: Wednesday
Enter the date of day 3: 12
Enter the description of activity for day 3: Go to gym

Enter the name of day 4: Thursday
Enter the date of day 4: 13
Enter the description of activity for day 4: Work on project

Enter the name of day 5: Friday
Enter the date of day 5: 14
Enter the description of activity for day 5: Team lunch

Enter the name of day 6: Saturday
Enter the date of day 6: 15
Enter the description of activity for day 6: Relax and watch TV

Enter the name of day 7: Sunday
Enter the date of day 7: 16
Enter the description of activity for day 7: Go out with friends

Week Activity Details Report
-------------------------------
Monday 10: Work from home
Tuesday 11: Meet with client
Wednesday 12: Go to gym
Thursday 13: Work on project
Friday 14: Team lunch
Saturday 15: Relax and watch TV
Sunday 16: Go out with friends
```

Viva Questions:

1. Define Dynamic Allocation in C?

An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used. These functions are defined in the <stdlib.h> header file.

2. Define malloc() function , Give the syntax of malloc function

The name "malloc" stands for memory allocation.The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be casted into pointers of any form.

Syntax:

```
ptr = (castType*) malloc(size);
```

Example

```
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes. And, the pointer ptr holds the address of the first byte in the allocated memory.The expression results in a NULL pointer if the memory cannot be allocated.

3. What is the difference between malloc() and calloc() functions?

The name "calloc" stands for contiguous allocation.

The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.

Syntax of calloc():

```
ptr = (castType*)calloc(n, size);
```

Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

The above statement allocates contiguous space in memory for 25 elements of type float

4. Explain the purpose of using free function

Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.

Syntax of free()

```
free(ptr);
```

This statement frees the space allocated in the memory pointed by ptr.

5. Explain typedef struct with with an example

The typedef struct keyword in C is used to create a new data type that is a synonym for a struct. This can be useful for making code more readable and maintainable.

For example, the following code defines a new data type called Person that is a synonym for the struct person struct:

```
typedef struct person {
  char name[20];
  int age;
} Person;
```

Once this definition has been made, the Person type can be used to declare variables and function parameters, just like any other data type.

PROGRAM 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.

THEORY:

- This program works by first reading the main string, pattern string, and replace string from the user. Then, it calls the stringmatch() function to find and replace all occurrences of the pattern string in the main string with the replace string.

- The stringmatch() function works by iterating over the main string and comparing each character to the first character of the pattern string. If a match is found, the function then compares the remaining characters of the main string to the remaining characters of the pattern string. If all of the characters match, then the function replaces the pattern string in the main string with the replace string.

- Once the stringmatch() function has returned, the main program checks the return value of the function to see if the pattern string was found in the main string. If it was, then the main program prints a message to the user and prints the new main string. Otherwise, the main program prints a message to the user indicating that the pattern string was not found in the main string.

- The program first declares four character arrays: str, pat, rep, and ans. The str array will store the main string, the pat array will store the pattern string, the rep array will store the replacement string, and the ans array will store the resultant string.

- The program then initializes three integer variables: c, m, and i,j,flag. The c variable will keep track of the current index in the str array, the m variable will keep track of the current index in the pat array, and the i variable will keep track of the current index in the rep array,j variable will keep track of the current index in the ans array.if flag value is changed to 1, then resultant string is displayed.

- The program then defines a function called stringmatch(). This function will find and replace all occurrences of the pattern string in the main string with the replacement string.

- The stringmatch() function works by iterating over the str array. For each character in the str array, the function compares the character to the first character of the pattern string.
- If the characters match, then the function compares the remaining characters in the str array to the remaining characters in the pattern string. If all of the characters match, then the function replaces the pattern string in the main string with the replacement string.
- After the stringmatch() function has finished executing, the ans array will contain the resultant string.
- Finally, the main program checks to see if the pattern string was found in the main string. If it was, then the main program prints the resultant string. Otherwise, the main program prints a message indicating that the pattern string was not found.

ALGORITHM:

STEP 1: Read the main string into str, pattern into pat and replacement into string

STEP 2: Call stringmatch() function

a) Iterate over the str array. For each character in the str array, the stringmatch() function compares the character to the first character of the pattern string. If the characters match, then the function compares the remaining characters in the str array to the remaining characters in the pattern string.

```
  while(str[c] !='\0')  //Iterate through the main string
          {
                    if(str[m] = = pat[i]) //compare character from main string with pattern string
                    {
                              i++;
                              m++;
```

b) If all of the characters match, then the function replaces the pattern string in the main string with the replacement string.

```
      for(k=0; rep[k]!='\0'; k++, j++)
     {
         ans[j] = rep[k]; //replace the pattern string in the main string with the replacement string.
```

```
            }
```

STEP 3: The main program checks to see if the pattern string was found in the main string. If the flag variable is set to 1 then , prints the resultant string. Otherwise, the main program prints a message indicating that the pattern string was not found.

```
 if(flag = = 1)

                printf("\nResultant string is %s", ans);

        else

                printf("\nPattern string is not found");
```

PROGRAM:

```
#include<stdio.h>
char str[50], pat[20], rep[20], ans[50];
int     c=0, m=0, i=0, j=0, k, flag=0;
void stringmatch()
{
         while(str[c] !='\0')
        {
                if(str[m] == pat[i])
                {
                        i++;
                        m++;
                        if(pat[i] == '\0')
                        {
                                flag = 1;
                                for(k=0; rep[k]!='\0'; k++, j++)
                                {
                                        ans[j] = rep[k];
                                }
                                i = 0;
                                c = m;
                        }
                }
                else
                {
                        ans[j]= str[c];
                        j++;
                        c++;
                        m=c;
                        i=0;
                }
        }
        ans[j]='\0';
}
void main()
{
        printf("\nEnter the main string:");
        gets(str);
        printf("\nEnter the pat string:");
        gets(pat);
        printf("\nEnter the replace string:");
        gets(rep);
        stringmatch();
        if(flag == 1)
                printf("\nResultant string is %s", ans);
        else
                printf("\nPattern string is not found");
}
```

OUTPUT 1:

```
Enter the main string:Mangalore

Enter the pat string:alo

Enter the replace string:alu

Resultant string is Mangalure
```

OUTPUT 2:

```
Enter the main string:Mangalore

Enter the pat string:test

Enter the replace string:best

Pattern string is not found
```

Viva Questions:

1. What is an Array & how many types of arrays can be represented in memory?

An array is a structured data type made up of a finite, fixed size, collection of homogeneous ordered elements. Types of array: One-Dimensional array, Two Dimensional array, Multi-Dimensional array

2. What is the syntax for array declaration?

data type var_name[Expression];

3. What are the different ways in which elements can be inserted in an array?

Insertion of a new element in an array can be done in two ways: * Insertion at the end of array * Insertion at required position

4. What are the different ways in which elements can be deleted from the array?

Deleting an element at the end of an array presents no difficulties, but deleting element somewhere in the middle of the array requires us to rearrange the remaining elements.

5. How many types of implementation can be done for a two-dimensional array?

- Row-major implementation
- Column-major implementation

6. What are the limitations of arrays?

- These are static structures.
- It is very time consuming

7. What are the operations than can be performed on an array?

Insertion, Deletion, Traversing, Merging, Display

8. Define Strings

A String in C programming is a sequence of characters terminated with a null character '\0'. The C String is stored as an array of characters. The difference between a character array and a C string is the string is terminated with a unique character '\0'**.**

PROGRAM 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

THEORY:

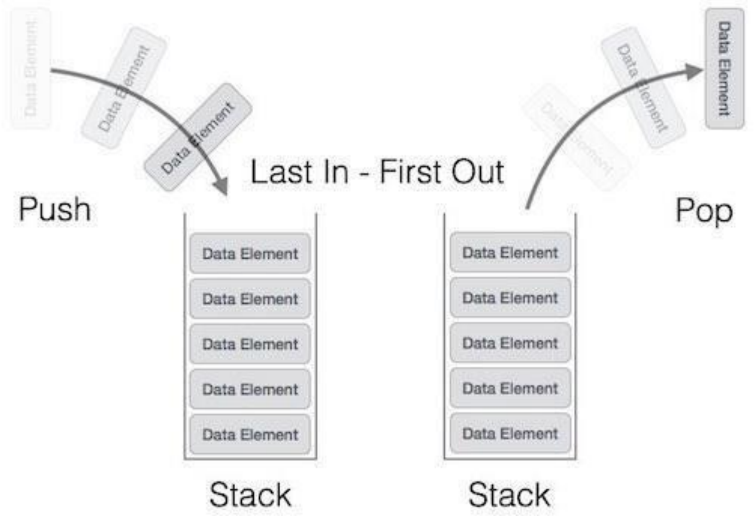
A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

Stack Examples

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the 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 stands for 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.

Stack Representation

The following diagram depicts a stack and its operations –



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 resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations − · **push()** − Pushing (storing) an element on the stack.

· **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

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

· **peek()** − get the top data element of the stack, without removing it.

· **isFull()** − check if stack is full.

· **isEmpty()** − check if stack is empty.

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

ALGORITHM:

Algorithm stack_operation

Step1:Start
Step2:Push( )
Step3:Pop( )
Step4:Display( )
Step5:Exit
Step6: Stop

Algorithm Push ( )

```
Step1:Check for stack overflow
        if top == maxsize – 1
Step2:then display stack overflow.
Step3:else
Step4:read the element to be pushed
        stack[++top]=c
```

Algorithm Pop ( )

```
Step1:Check for stack underflow
        if top = -1
Step2:then display stack underflow.
Step3:else
:pop the element from stack and display stack[top--]
```

Algorithm Display ( )

```
Step1:Check for stack underflow
        if top = -1
Step2:then display stack underflow.
Step3:else
Step4:repeat top to 0
Step5:display stack elements.
```

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 5

int s[MAX];
int top = -1;

void push(int item);
int pop();
void palindrome();
void display();

void main()
{
        int choice, item;
        while(1)
        {
                printf("\n\n\n\n~~~~~~Menu~~~~~~ : ");
                printf("\n=>1.Push an Element to Stack and Overflow demo ");
                printf("\n=>2.Pop an Element from Stack and Underflow demo");
                printf("\n=>3.Palindrome demo ");
                printf("\n=>4.Display ");
                printf("\n=>5.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &choice);
                switch(choice)
                {
                        case 1:         printf("\nEnter an element to be pushed: ");
                                        scanf("%d", &item);
                                        push(item);
                                        break;
                        case 2:         item = pop();
                                        if(item != -1)
                                                printf("\nElement popped is: %d", item);
                                        break;
                        case 3:         palindrome();
                                        break;
                        case 4:         display();
                                        break;
                        case 5:         exit(1);
                        default:        printf("\nPlease enter valid choice ") ;
                                        break;
            }
        }
}

void push(int item)
{
        if(top == MAX-1)
        {
                printf("\n~~~~Stack overflow~~~~");
                return;
        }

        top = top + 1 ;
        s[top] = item;
}

int pop()
{
```

```
        int item;
        if(top == -1)
        {
                printf("\n~~~~Stack underflow~~~~");
                return -1;
        }
        item = s[top];
        top = top - 1;
        return item;
}

void display()
{
        int i;
        if(top == -1)
        {
                printf("\n~~~~Stack is empty~~~~");
                return;
        }
        printf("\nStack elements are:\n ");
        for(i=top; i>=0 ; i--)
                printf("| %d |\n", s[i]);
}

void palindrome()
{
        int flag=1,i;
        printf("\nStack content are:\n");
        for(i=top; i>=0 ; i--)
                printf("| %d |\n", s[i]);

        printf("\nReverse of stack content are:\n");
        for(i=0; i<=top; i++)
                printf("| %d |\n", s[i]);

        for(i=0; i<=top/2; i++)
        {
                if( s[i] != s[top-i] )
                {
                        flag = 0;
                        break;
                }
        }
        if(flag == 1)
        {
```

```
                printf("\nIt is palindrome number");
        }
        else
        {
                printf("\nIt is not a palindrome number");
        }
}
```

OUTPUT:

```
~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 10

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 20
```

```
~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 30

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 40

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1
```

```
Enter an element to be pushed: 50

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 60

~~~~Stack overflow~~~~

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 4

Stack elements are:

 | 50 |

| 40 |

| 30 |

| 20 |

| 10 |
```

```
~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

Element popped is: 50

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 4

Stack elements are:

 | 40 |

| 30 |

| 20 |

| 10 |

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo
```

```
=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

Element popped is: 40

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 4

Stack elements are:

 | 30 |

| 20 |

| 10 |

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2
```

```
Element popped is: 30

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

Element popped is: 20

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 4

Stack elements are:

 | 10 |

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display
```

```
=>5.Exit

Enter your choice: 2

Element popped is: 10

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 2

~~~~Stack underflow~~~~

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 10

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo
```

```
=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 20

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 1

Enter an element to be pushed: 10

~~~~~~Menu~~~~~~ :

=>1.Push an Element to Stack and Overflow demo

=>2.Pop an Element from Stack and Underflow demo

=>3.Palindrome demo

=>4.Display

=>5.Exit

Enter your choice: 3

Stack content are:

| 10 |

| 20 |

| 10 |
```

```
Reverse of stack content are:

| 10 |

| 20 |

| 10 |

It is palindrome number
```

Viva Questions:

1. **What is a stack?**
   Stack is an ordered collection of elements like an array, it is a non- primitive, linear data structure and is often called as LIFO (Last In First Out), as the element to be inserted first is the last element to be removed from the stack.

2. **What are the operations that can be performed on stack?**
   Push for insertion and pop for deletion.

3. **What are the conditions to be checked before inserting / deleting elements from the stack?**
   Stack Overflow for insertion Stack Underflow for deletion

4. **How are stacks are implemented?**
   Stacks are implemented in two ways: Static implementation – array Dynamic implementation – pointers

5. **What are the applications of stack?**
   Infix , Post fix and Prefix expression evaluations and conversions are some of the applications of stack

6. **What is recursion?**
   The phenomenon of a function calling itself is called Recursion.

7. **Define “Top of stack”**
   It is a pointer indicating the top element of the stack

PROGRAM 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.

THEORY:

Converting an infix expression to a postfix expression is a fundamental operation in computer science and is often used in evaluating mathematical expressions. The postfix notation, also known as Reverse Polish Notation (RPN), has the advantage of being unambiguous and easy to evaluate using a stack. Here's an algorithm to convert an infix expression to postfix notation

Algorithm for Infix to Postfix Conversion:

STEP 1: Initialize an empty stack for operators. You can also use a list or array to store the output postfix expression.

STEP 2: Initialize an empty string to store the postfix expression.

STEP 3: Scan the infix expression from left to right.

STEP 4: For each symbol in the infix expression:

a. If it's an operand (a number or variable), add it to the output string.

b. If it's an open parenthesis '(', push it onto the stack.

c. If it's a closing parenthesis ')', pop and append operators from the stack to the output string until an open parenthesis '(' is encountered. Pop and discard the open parenthesis.

d. If it's an operator (+, -, *, /, etc.), then:

STEP 4.1: While the stack is not empty and the precedence of the operator at the top of the stack is greater or equal to the current operator's precedence (or it's left-associative and has equal precedence), pop operators from the stack and append them to the output string.

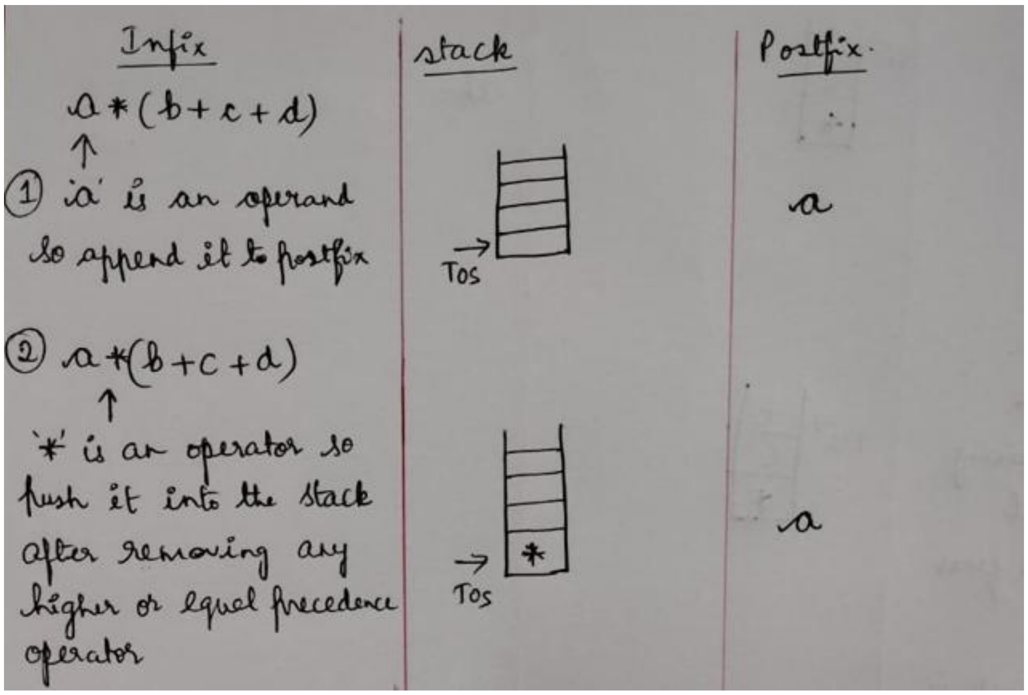
STEP 4.2 : Push the current operator onto the stack.

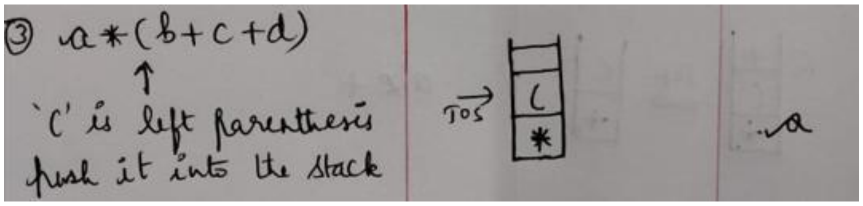
STEP 5: After processing all the symbols, pop any remaining operators from the stack and append them to the output string.

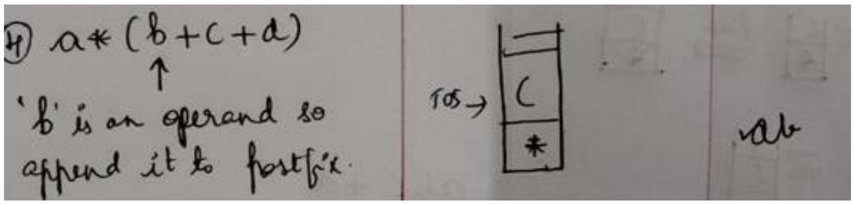
STEP 6 : The resulting string is the postfix expression.

Example:

Infix Expression: a*(b+c+d)

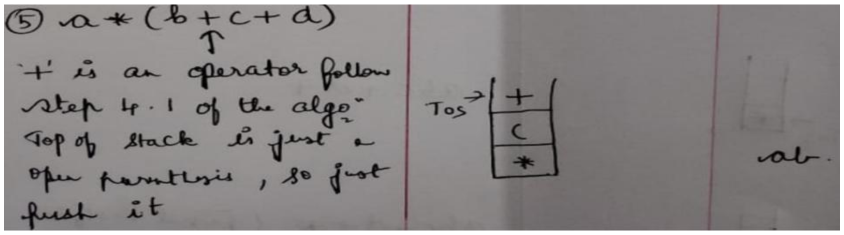

Infix
stack
Postfix.
a*(b+c+d)
① 'a' is an operand so append it to postfix
Tos
a
② a*(b+c+d)
'*' is an operator so push it into the stack after removing any higher or equal precedence operator
*
Tos
a





4) a*(b+c+d)

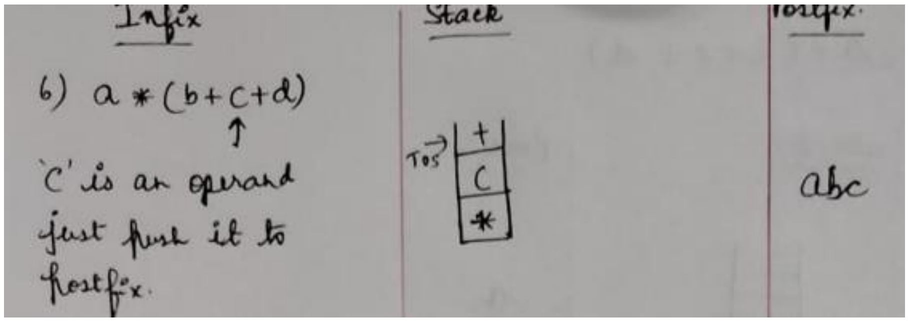
'b' is an operand so append it to postfix.



5) a*(b+c+d)

'+' is an operator follow step 4.1 of the algo

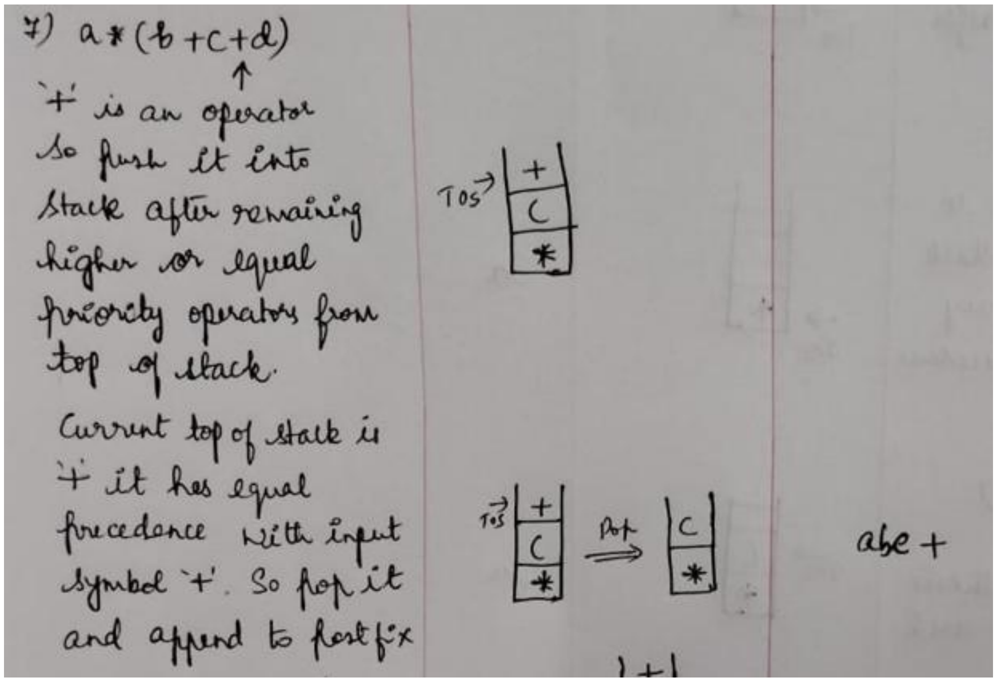
Top of stack is just a open paranthesis, so just push it



Infix Stack Postfix.

6) a*(b+c+d)

'c' is an operand just push it to postfix.

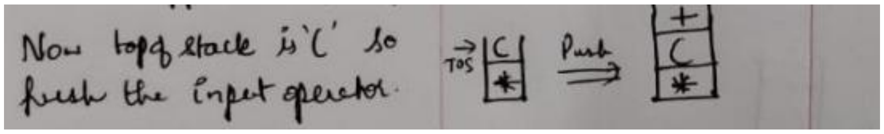


7) a*(b+c+d)

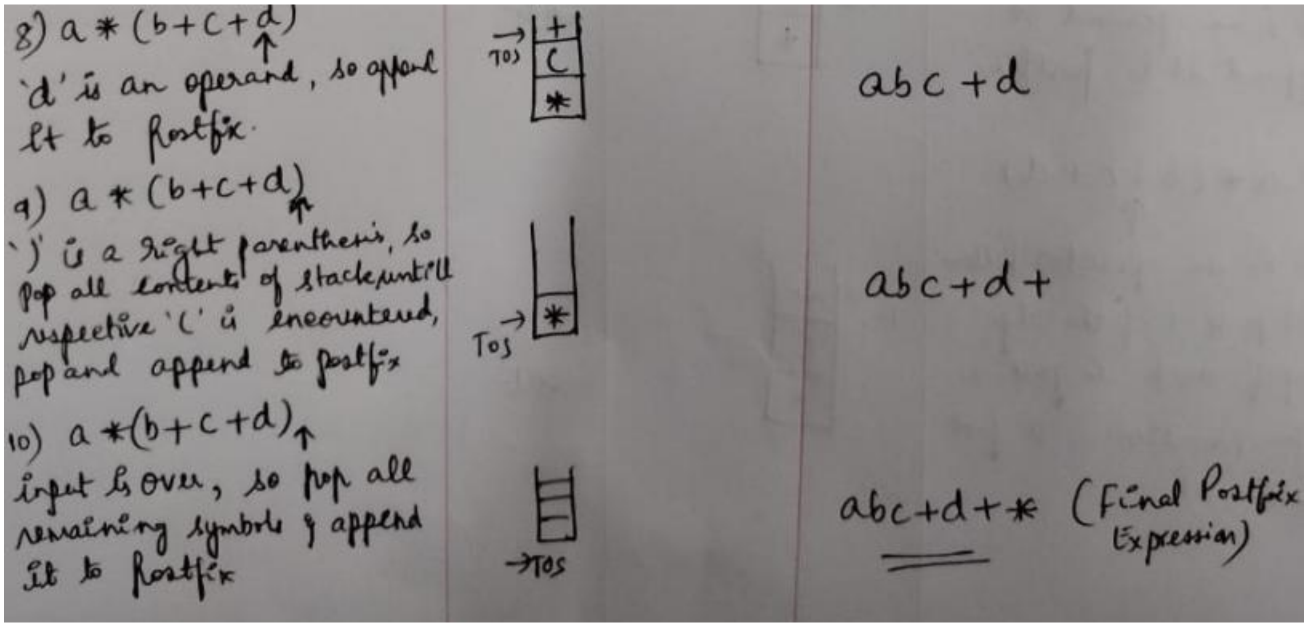
'+' is an operator so push it into stack after remaining higher or equal priority operators from top of stack.

Current top of stack is '+' it has equal precedence with input symbol '+'. So pop it and append to postfix

TOS
+
(
*
TOS
+
(
*
Pop
(
*
abc+



Now top of stack is '(' so push the input operator.



8) a * (b+c+d) 'd' is an operand, so append it to Postfix.	TOS → + (*	abc+d
9) a * (b+c+d) ')' is a right parenthesis, so pop all contents of stack until respective '(' is encountered, pop and append to postfix	TOS → *	abc+d+
10) a *(b+c+d) input is over, so pop all remaining symbols & append it to Postfix	→ TOS	abc+d+* (Final Postfix Expression)

Program:

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

void  evaluate();
void push(char);
char pop();
int prec(char);

char infix[30], postfix[30], stack[30];
int top = -1;
void main()
{
        printf("\nEnter the valid infix expression:\t");
        scanf("%s", infix);
        evaluate();
        printf("\nThe entered infix expression is :\n %s \n", infix);
        printf("\nThe corresponding postfix expression is :\n %s \n", postfix);
}
void evaluate()
{
```

```
int i = 0, j = 0;
char symb, temp;
push('#');
for(i=0; infix[i] != '\0'; i++)
{
        symb = infix[i];
        switch(symb)
        {
                case '(' :      push(symb);
                                break;

                case ')' :      temp = pop();
                                while(temp != '(' )
                                {
                                        postfix[j] = temp;
                                        j++;
                                        temp = pop();
                                }
                                break;
                case '+' :
                case '-' :
                case '*' :
                case '/' :
                case '%' :
                case '^' :
                case '$'  :     while( prec(stack[top]) >= prec(symb) )
                                {
                                        temp = pop();
                                        postfix[j] = temp;
                                        j++;
                                }
                                push(symb);
                                break;
```

```
                        default:        postfix[j] = symb;
                                        j++;
                }
        }
        while(top > 0)
        {
                temp = pop();
                postfix[j] = temp;
                j++;
        }
        postfix[j] = '\0';
}
 void push(char item)
{
        top = top+1;
        stack[top] = item;
}
 char pop()
{
        char item;
        item = stack[top];
        top = top-1;
        return item;
}

int prec(char symb)
{
        int p;
        switch(symb)
        {
                case '#' :      p = -1;
                                break;
                case '(' :
                    case ')' :          p = 0;
                                        break;
                    case '+' :
                    case '-' :          p = 1;
                                        break;
                    case '*' :
                    case '/' :
                    case '%' :          p = 2;
                                        break;
                    case '^' :
                    case '$' :          p = 3;
                                        break;
          }
          return p;
}
```

Output 1:

```
Enter the valid infix expression:        a*(b+c+d)

The entered infix expression is :

 a*(b+c+d)

The corresponding postfix expression is :

 abc+d+*
```

Output 2:

```
Enter the valid infix expression:        (a+b)+c/d*e

The entered infix expression is :

 (a+b)+c/d*e

The corresponding postfix expression is :

 ab+cd/e*+
```

Viva Questions:

1. Convert the infix expression a/b-c +d*e-a*c to a postfix expression.

Ans = ab/c-de*+ac*-

2. Expalin the steps to convert infix expression to postfix

STEP 1: Initialize an empty stack for operators. You can also use a list or array to store the output postfix expression.

STEP 2: Initialize an empty string to store the postfix expression.

STEP 3: Scan the infix expression from left to right.

STEP 4: For each symbol in the infix expression:

a. If it's an operand (a number or variable), add it to the output string.

b. If it's an open parenthesis '(', push it onto the stack.

c. If it's a closing parenthesis ')', pop and append operators from the stack to the output string until an open parenthesis '(' is encountered. Pop and discard the open parenthesis.

d. If it's an operator (+, -, *, /, etc.), then:

STEP 4.1: While the stack is not empty and the precedence of the operator at the top of the stack is greater or equal to the current operator's precedence (or it's left-associative and has equal precedence), pop operators from the stack and append them to the output string.

STEP 4.2 : Push the current operator onto the stack.

STEP 5: After processing all the symbols, pop any remaining operators from the stack and append them to the output string.

STEP 6 : The resulting string is the postfix expression.

PROGRAM 5:

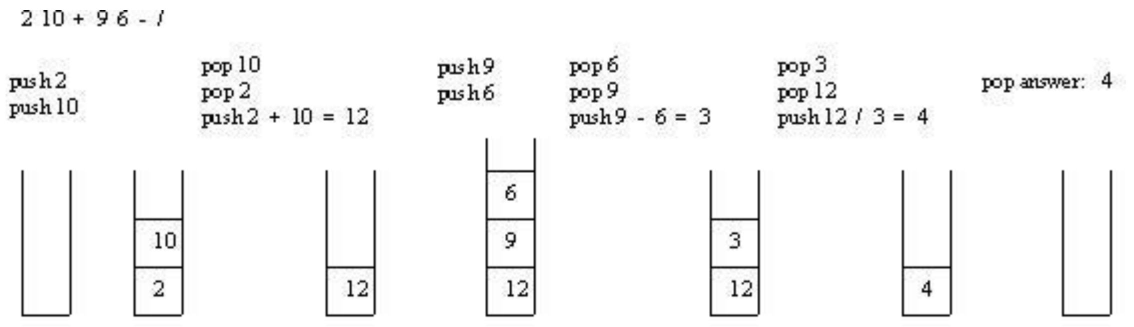
Develop a Program in C for the following Stack Applications

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

b. Solving Tower of Hanoi problem with n disks

THEORY:

Evaluation of a postfix expression using a stack is explained in below example:



Algorithm:

To evaluate a suffix expression with single digit operands and operators: +, -, *, /, %, ^, we can use the following algorithm:

STEP 1:Create a stack.
STEP 2: Scan the suffix expression from left to right.
STEP 3: If the current token is an operand, push it onto the stack.
STEP 4: If the current token is an operator, pop the top two operands from the stack and apply the operator to them. Push the result onto the stack.
STEP 5: Repeat steps 3 and 4 until the end of the suffix expression is reached.
STEP 6: The top operand on the stack is the result of the suffix expression.

Program 5a: **Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^**

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>

int i, top = -1;
int op1, op2, res, s[20];
char postfix[90], symb;

void push(int item)
{
        top = top+1;
        s[top] = item;
}

int pop()
{
        int item;
        item = s[top];
        top = top-1;
        return item;
}

void main()
{
        printf("\nEnter a valid postfix expression:\n");
        scanf("%s", postfix);
        for(i=0; postfix[i]!='\0'; i++)
        {
                symb = postfix[i];
                if(isdigit(symb))
                {
                        push(symb - '0');
                }
                else
                {
                        op2  =  pop();
                        op1  =  pop();
                        switch(symb)
                        {
                                case '+':       push(op1+op2);
                                                break;
                                case '-':       push(op1-op2);
                                                break;
                                case '*':       push(op1*op2);
                                                break;
```

```
                                        case '/':       push(op1/op2);
                                                        break;
                                        case '%':       push(op1%op2);
                                                        break;
                                        case '$':
                                        case '^':       push(pow(op1, op2));
                                                        break;
                                        default :  push(0);
                                }
                        }
                }
                res = pop();
                printf("\n Result = %d", res);
        }
```

OUTPUT:

```
Enter a valid postfix expression:
39+96-/

 Result = 4
```

5 b. Solving Tower of Hanoi problem with n disks.

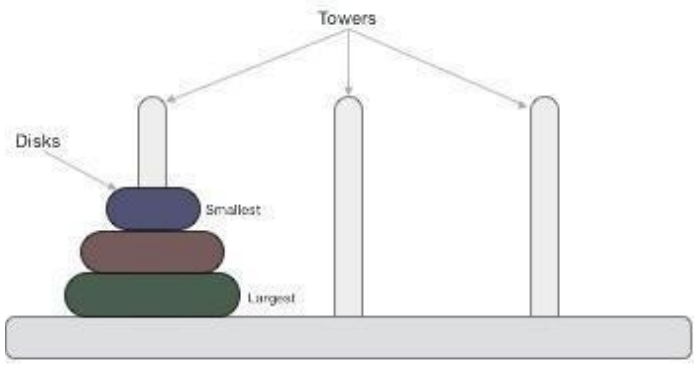
THEORY:

Recursion:

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

Tower of Hanoi

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted –



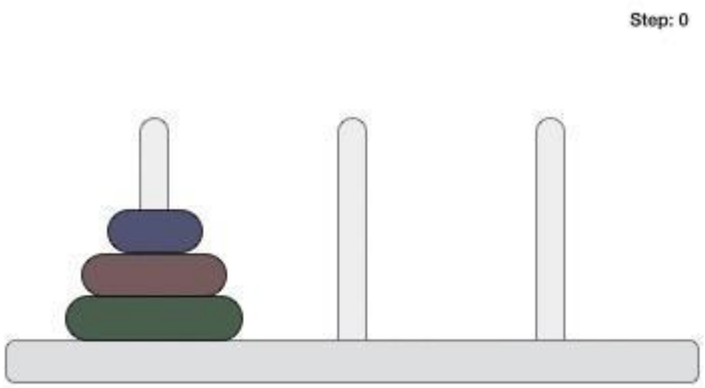
These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Following is an animated representation of solving a Tower of Hanoi puzzle with three disks.



Tower of Hanoi puzzle with n disks can be solved in minimum **2^n-1** steps. This presentation shows that a puzzle with 3 disks has taken **2^3 - 1 = 7** steps.

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 5b:

```
#include <stdio.h>
void tower(int n, int source, int temp,int destination)
{
   if(n == 0)
       return;
   tower(n-1, source, destination, temp);
   printf("\nMove disc %d from %c to %c", n, source, destination);
   tower(n-1, temp, source, destination);
}
```

```
void main()
{
    int n;
    printf("\nEnter the number of discs: \n");
    scanf("%d", &n);
    tower(n, 'A', 'B', 'C');
    printf("\n\nTotal Number of moves are: %d", (int)pow(2,n)-1);
}
```

OUTPUT:

```
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
```

VIVA QUESTIONS: (Program 5a and Program 5b)

1. **Evaluate the postfix expression: 4325*-+**
   Ans = -3

2. **Explains the rules to solve the Tower of Hanoi problem?**
   Suppose three pegs labelled A, B, C are given and suppose on peg A there are placed a finite number n of disks with decreasing size. The objective of the game is to move the disks from peg A to peg C using peg B as an auxilary. The rules of the game are as follows:
   • Only one disk may be moved at a time. Specifically, only the top disk on any peg may be moved to any other peg.
   • At no time can a larger disk be placed on a smaller disk.

4. **How many moves are required to solve the Tower of Hanoi problem with n disks?**
   Ans = f(n) = 2n -1 moves

PROGRAM 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

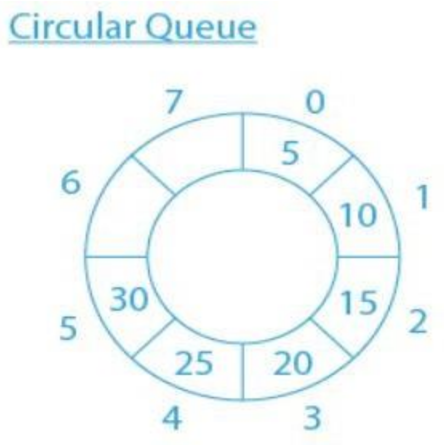
d. Display the status of Circular QUEUE

e. Exit

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

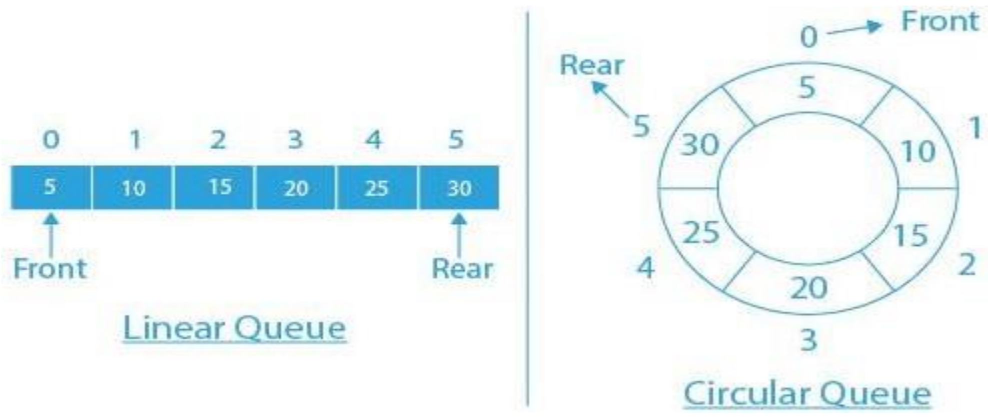
THEORY:

A circular queue is a data structure that behaves like a queue but wraps around itself, forming a circle instead of a straight line. It has a fixed size and consists of a front and rear pointer. When an element is added, it is inserted at the rear and when an element is removed, it is removed from the front. If the rear pointer reaches the end of the queue, it wraps around to the beginning, allowing for efficient use of memory. In this type of queue, the element can be added in any position or can be deleted from any position in the array but we have to maintain the pointers which will point towards the front and rear end of the queue. In this queue, the rear end can be at any point in the array.

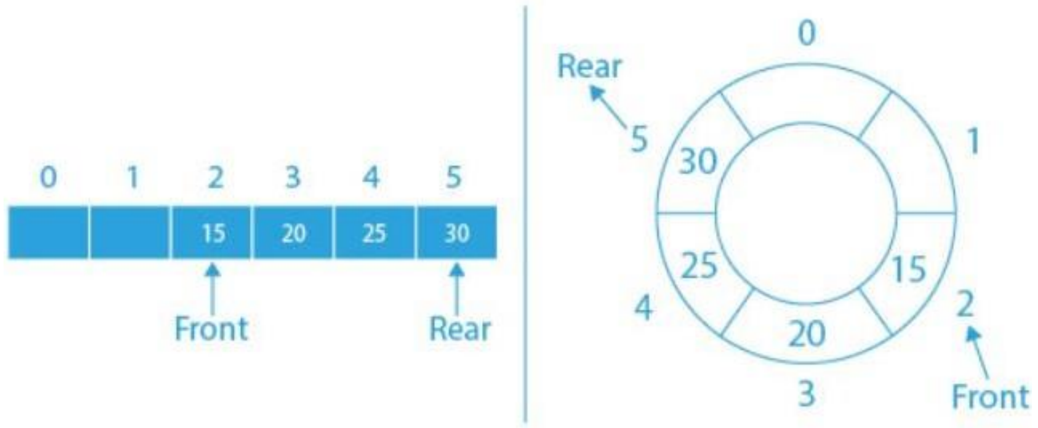


Example of Circular Queue over Linear Queue

Here, are the example of circular queue over linear queue: When Enqueue operation is performed on both types of queues: Let the queue is of size 6 having elements {5, 10, 15, 20, 25, 30}. In both the queues the front points at the first element 5 and the rear points at the last element 30 as shown in the below image:

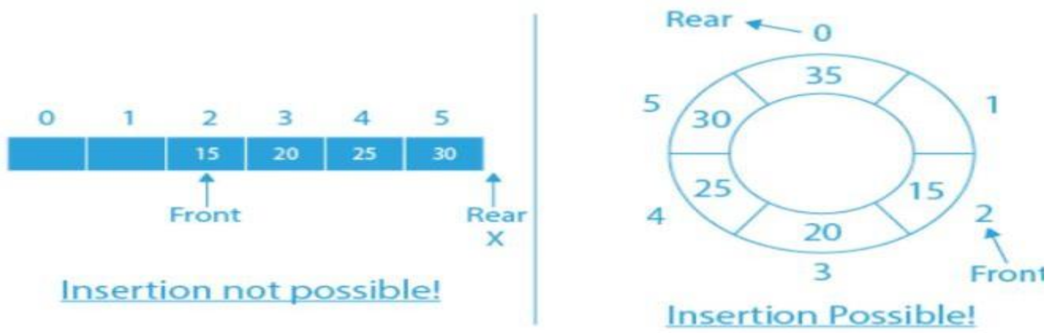


When the dequeue operation is performed on both the queues:When the dequeue operation is performed on both queues the first 2 elements are deleted from both queues. In both the queues the front points at the element with value 15 and the rear points at the element with value 30 as shown in the below image:



Now again enqueue operation is performed:

When enqueue operation is performed an element with a value of 35 is inserted in both the queues. The insertion of element 35 is not possible in Linear Queue but in the Circular Queue, the element with a value of 35 is possible as shown in the below image:



Explanation:

The insertion in the queue is done from the rear end and in case of Linear Queue of fixed size insertion is not possible when rear reaches the end of the queue.

But in the Circular Queue, the rear end moves from the last position to the front position circularly.

Operations of a Circular Queue in Data Structure:

front(): front is used to track the first element of the queue.

rear(): rear is used to track the last element of the queue.

Enqueue: This operation is used to Insert an element at the end of the queue.

Dequeue: This operation is used to remove and return an element from the front of the queue.

Steps for Enqueue Operation

1. Check if the queue is full.
2. If the queue is full then, the print queue is full.
3. If not, check the condition(front == -1)
4. If it is true then set the front and rear = 0
5. If not , set rear= (rear + 1) % max and insert the new element

Algorithm to Insert :

```
Step 1: If  front = (rear+1)%max
                Write " Overflow "
                Goto step 3
        [end of if]
Step 2: If front = -1
                Set front = rear = 0
        Else
                Set rear = (rear + 1) % max
                Set queue[rear] = val
        [end of if]
Step 3: Exit
```

Steps for Dequeue Operations:

1. Firstly, check whether the queue is empty or not.
2. If the queue is empty then display the queue is empty.
3. If not, check the condition(rear == front)
4. If the condition is true then set front = rear = -1.
5. Else, front == size-1 and return the element.

Algorithm to Delete:

```
Step 1: If front = -1
        Write " Underflow "
        Goto step 3
        [end of if]
Step 2: Set val = queue[front]
Step 3: If front = rear
        Set front = rear = -1
        Else
        Set front = (front+1)%max
        [end of if]
        [end of if]
Step 4: Exit
```

PROGRAM:

```
#include <stdio.h>
#include<stdlib.h>
#include<stdio_ext.h>
#define MAX 3
```

```
char cq[MAX];
int front = -1, rear = -1;

void insert(char);
void delete();
void display();

void main()
{
        int ch;
        char item;
        while(1)
        {
                printf("\n\n~~Main Menu~~");
                printf("\n==> 1. Insertion and Overflow Demo");
                printf("\n==> 2. Deletion and Underflow Demo");
                printf("\n==> 3. Display");
                printf("\n==> 4. Exit");
                printf("\nEnter Your Choice: ");
                scanf("%d", &ch);
                __fpurge(stdin);
               switch(ch)
               {
                        case 1:     printf("\n\nEnter the element to be inserted: ");
                                    scanf("%c", &item);
                                    insert(item);
                                    break;
                        case 2:     delete();
                                    break;
                        case 3:     display();
                                    break;
                         case 4:    exit(0);
                         default:   printf("\n\nPlease enter a valid choice");
                }
         }
}

void insert(char item)
{
          if(front == (rear+1)%MAX)
          {
                  printf("\n\n~~Circular Queue Overflow~~");
          }
          else
```

```
                {
                        if(front == -1)
                                front = rear = 0;
                        else
                                rear = (rear+1)%MAX;
                        cq[rear] = item;
                }
}

void delete()
{
        char item;
        if(front == -1)
        {
                printf("\n\n~~Circular Queue Underflow~~");
        }
        else
        {
                item = cq[front];
                printf("\n\nDeleted element from the queue is: %c ",item );


                if(front == rear) //only one element
                        front = rear = -1;
                else
                        front = (front+1)%MAX;
        }
}


void display ()
{
        int i ;
        if(front == -1)
        {
                printf("\n\nCircular Queue Empty");
        }
        else
        {
                printf("\nCircular Queue contents are:\n");
                printf("Front[%d]-> ", front);
                for(i = front; i != rear ; i = (i+1)%MAX)
                {
                        printf(" %c", cq[i]);
                }
                printf(" %c", cq[i]);
                printf(" <-[%d]Rear", rear);
        }
}
```

OUTPUT:

```
~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 1

Enter the element to be inserted: A

~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 1

Enter the element to be inserted: B

~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit
```

```
Enter Your Choice: 1

Enter the element to be inserted: C

~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 3

Circular Queue contents are:

Front[0]->  A B C <-[2]Rear



~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 1

Enter the element to be inserted: D

~~Circular Queue Overflow~~



~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo
==> 3. Display

==> 4. Exit

Enter Your Choice: 2

Deleted element from the queue is: A



~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 3

Circular Queue contents are:

Front[1]->  B C <-[2]Rear



~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 2

Deleted element from the queue is: B



~~Main Menu~~
```

```
==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 3

Circular Queue contents are:

Front[2]->  C <-[2]Rear

~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 2

Deleted element from the queue is: C

~~Main Menu~~

==> 1. Insertion and Overflow Demo

==> 2. Deletion and Underflow Demo

==> 3. Display

==> 4. Exit

Enter Your Choice: 3

Circular Queue Empty
```

Viva Questions:

1. Define Circular Queue.

A circular queue is a linear data structure that follows FIFO principle. In circular queue, the last node is connected back to the first node to make a circle.

2. What is the advantage of using a circular queue?

Circular queues efficiently manage cyclic data or processes and utilize memory effectively by reusing available space.

3. How is a circular queue implemented?

A circular queue can be implemented using arrays or linked lists. In arrays, modulo arithmetic is used to wrap around the queue, while linked lists are modified to form a circular arrangement.

4. What is the significance of circular increment in a circular queue?

Circular increment ensures that when the end of the queue is reached, the next element is positioned at the beginning of the queue, forming a circular structure.

5. What are the main operations supported by a circular queue?

The primary operations of a circular queue include enqueue (addition of an element), dequeue (removal of an element), front (retrieve the front element), rear (retrieve the rear element), and isEmpty (check if the queue is empty).

6. What are the applications of circular queues in data structures?

Circular queues are used in resource allocation, buffering, scheduling, simulation systems, data transmission, print spooling, real-time systems, networking, memory management, and more

PROGRAM 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)

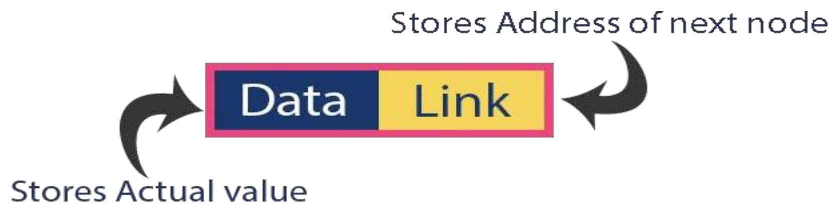
e. Exit

THEORY:

When we want to work with an unknown number of data values, we use a linked list data structure to organize that data. The linked list is a linear data structure that contains a sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called "Node".

What is Single Linked List?

Simply a list is a sequence of data, and the linked list is a sequence of data linked with each other.
The formal definition of a single linked list is as follows...
Single linked list is a sequence of elements in which every element has link to its next element in the sequence.
In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data field, and the next field. The data field is used to store actual value of the node and next field is used to store the address of next node in the sequence.
The graphical representation of a node in a single linked list is as follows…

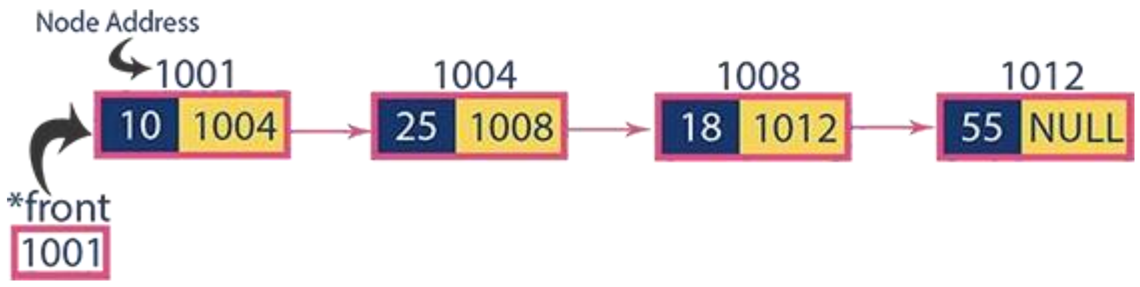


Important Points to be Remembered:

- In a single linked list, the address of the first node is always stored in a reference node known as "front" (Some times it is also known as "head").

- Always next part (reference part) of the last node must be NULL.

Example:



Operations on Single Linked List

The following operations are performed on a Single Linked List

- Insertion
- Deletion
- Display

Before we implement actual operations, first we need to set up an empty list. First, perform the following steps before implementing actual operations.

Step 1 - Include all the header files which are used in the program.

Step 2 - Declare all the user defined functions.

Step 3 - Define a Node structure with two members data and next

Step 4 - Define a Node pointer 'head' and set it to NULL.

Step 5 - Implement the main method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

ALGORITHM FOR INSERTION:

Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

[Note : According to the question we will be seeing only insert at the beginning or front and insert at the end or last. So the 3 option is not explained here]

1. Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)
Step 3 - If it is Empty then, set newNode→next = NULL and head = newNode.
Step 4 - If it is Not Empty then, set newNode→next = head and head = newNode.

2.Inserting At End of the list
We can use the following steps to insert a new node at end of the single linked list...

Step 1 - Create a newNode with given value and newNode → next as NULL.
Step 2 - Check whether list is Empty (head == NULL).
Step 3 - If it is Empty then, set head = newNode.
Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).
Step 6 - Set temp → next = newNode.

ALGORITHM FOR DELETION:

Deletion
In a single linked list, the deletion operation can be performed in three ways. They are as follows...
1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node
[Note : According to the question we will be seeing only delete at the beginning or front and delete at the end or last. So the 3 option is not explained here]

1.Deleting from Beginning of the list
We can use the following steps to delete a node from beginning of the single linked list...

Step 1 - Check whether list is Empty (head == NULL)
Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
Step 4 - Check whether list is having only one node (temp → next == NULL)
Step 5 - If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)
Step 6 - If it is FALSE then set head = temp → next, and delete temp.

2.Deleting from End of the list
We can use the following steps to delete a node from end of the single linked list...

Step 1 - Check whether list is Empty (head == NULL)
Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == NULL)
Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate the function. (Setting Empty list condition)
Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until it reaches to the last node in the list. (until temp1 → next == NULL)
Step 7 - Finally, Set temp2 → next = NULL and delete temp1.

ALGORITHM TO DISPLAY A SINGLE LINKED LIST

We can use the following steps to display the elements of a single linked list...

Step 1 - Check whether list is Empty (head == NULL)
Step 2 - If it is Empty then, display 'List is Empty!!!' and terminate the function.
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node
Step 5 - Finally display temp → data with arrow pointing to NULL (temp → data ---> NULL).

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    char usn[25],name[25],branch[25];
    int sem;
    long int phone;
    struct node *link;
};
typedef struct node * NODE;
NODE start = NULL;
int count=0;

NODE create()
{
    NODE snode;
    snode = (NODE)malloc(sizeof(struct node));

    if(snode == NULL)
```

```
   {
      printf("\nMemory is not available");
      exit(1);
   }
   printf("\nEnter the usn,Name,Branch, sem,PhoneNo of the student:");
   scanf("%s %s %s %d %ld",snode->usn, snode->name, snode->branch, &snode->sem, &snode-
>phone);
   snode->link=NULL;
   count++;
   return snode;
}

NODE insertfront()
{
   NODE temp;
   temp = create();
   if(start == NULL)
   {
         return temp;
   }

   temp->link = start;
   return temp;
}

NODE deletefront()
{
   NODE temp;
   if(start == NULL)
   {
      printf("\nLinked list is empty");
      return NULL;
   }
```

```
    if(start->link == NULL)
    {
            printf("\nThe Student node with usn:%s is deleted ",start->usn);
            count--;
            free(start);
            return NULL;
    }
    temp = start;
    start = start->link;
    printf("\nThe Student node with usn:%s is deleted",temp->usn);
    count--;
    free(temp);
    return start;
}

NODE insertend()
{
    NODE cur,temp;
    temp = create();

    if(start == NULL)
    {
      return temp;
    }
    cur = start;
    while(cur->link !=NULL)
    {
         cur = cur->link;
    }
    cur->link = temp;
    return start;
}
```

```
NODE deleteend()
{
    NODE cur,prev;
    if(start == NULL)
    {
      printf("\nLinked List is empty");
      return NULL;
    }

    if(start->link == NULL)
    {
      printf("\nThe student node with the usn:%s is deleted",start->usn);
      free(start);
      count--;
      return NULL;
    }

    prev = NULL;
    cur = start;
    while(cur->link!=NULL)
    {
       prev = cur;
       cur = cur->link;
    }

     printf("\nThe student node with the usn:%s is deleted",cur->usn);
     free(cur);
     prev->link = NULL;
     count--;
     return start;
}
```

```
void display()
{
    NODE cur;
    int num=1;
    if(start == NULL)
    {
        printf("\nNo Contents to display in SLL \n");
        return;
    }
    printf("\nThe contents of SLL: \n");
    cur = start;
    while(cur!=NULL)
    {
       printf("\n||%d|| USN:%s| Name:%s| Branch:%s| Sem:%d| Ph:%ld|",num,cur->usn, cur->name,cur-
>branch, cur->sem,cur->phone);
       cur = cur->link;
       num++;
    }
    printf("\n No of student nodes is %d \n",count);
}

void stackdemo()
{
   int ch;
   while(1)
   {
     printf("\n~~~Stack Demo using SLL~~~\n");
     printf("\n1:Push operation \n2: Pop operation \n3: Display \n4:Exit \n");
     printf("\nEnter your choice for stack demo");
     scanf("%d",&ch);

     switch(ch)
     {
```

```
        case 1: start = insertfront();
                break;
        case 2: start = deletefront();
                break;
        case 3: display();
               break;
       default : return;
      }
    }
   return;
}
int main()
{
    int ch,i,n;
    while(1)
    {
        printf("\n~~~Menu~~~");
        printf("\nEnter your choice for SLL operation \n");
        printf("\n1:Create SLL of Student Nodes");
        printf("\n2:DisplayStatus");
        printf("\n3:InsertAtEnd");
        printf("\n4:DeleteAtEnd");
        printf("\n5:Stack Demo using SLL(Insertion and Deletion at Front)");
        printf("\n6:Exit \n");
        printf("\nEnter your choice:");
        scanf("%d",&ch);

        switch(ch)
        {
        case 1 : printf("\nEnter the no of students:    ");
                 scanf("%d",&n);
                 for(i=1;i<=n;i++)
                    start = insertfront();
```

```
                 break;

        case 2: display();
                break;

        case 3: start = insertend();
                break;

        case 4: start = deleteend();
                break;

        case 5: stackdemo();
                break;

        case 6: exit(0);

        default: printf("\nPlease enter the valid choice");

        }
    }
}
```

<u>OUTPUT</u>:

```
~~~Menu~~~

Enter your choice for SLL operation



1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd
```

```
5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit

Enter your choice:1

Enter the no of students:    2

Enter the usn,Name,Branch, sem,PhoneNo of the student:4AJ21CS050

Ajay

CSE

3

9999999999

Enter the usn,Name,Branch, sem,PhoneNo of the student:4AJ22IS015

Vijay

ISE

1

9898989898

~~~Menu~~~

Enter your choice for SLL operation



1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit
```

```
Enter your choice:2

The contents of SLL:

||1|| USN:4AJ22IS015| Name:Vijay| Branch:ISE| Sem:1| Ph:9898989898|

||2|| USN:4AJ21CS050| Name:Ajay| Branch:CSE| Sem:3| Ph:9999999999|

 No of student nodes is 2



~~~Menu~~~

Enter your choice for SLL operation

1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit



Enter your choice:3

Enter the usn,Name,Branch, sem,PhoneNo of the student:4AJ20EC011

Jack

ECE

5

7878787878

~~~Menu~~~

Enter your choice for SLL operation
```

```
1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit

Enter your choice:2

The contents of SLL:

||1|| USN:4AJ22IS015| Name:Vijay| Branch:ISE| Sem:1| Ph:9898989898|

||2|| USN:4AJ21CS050| Name:Ajay| Branch:CSE| Sem:3| Ph:9999999999|

||3|| USN:4AJ20EC011| Name:Jack| Branch:ECE| Sem:5| Ph:7878787878|

 No of student nodes is 3

~~~Menu~~~

Enter your choice for SLL operation

1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit

Enter your choice:4

The student node with the usn:4AJ20EC011 is deleted

~~~Menu~~~
```

```
Enter your choice for SLL operation

1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit

Enter your choice:2

The contents of SLL:

||1|| USN:4AJ22IS015| Name:Vijay| Branch:ISE| Sem:1| Ph:9898989898|

||2|| USN:4AJ21CS050| Name:Ajay| Branch:CSE| Sem:3| Ph:9999999999|

 No of student nodes is 2

~~~Menu~~~

Enter your choice for SLL operation

1:Create SLL of Student Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:Stack Demo using SLL(Insertion and Deletion at Front)

6:Exit

Enter your choice:5

~~~Stack Demo using SLL~~~

1:Push operation
```

```
2: Pop operation

3: Display

4:Exit

Enter your choice for stack demo1

Enter the usn,Name,Branch, sem,PhoneNo of the student:4AJ22CS005

Reena

CSE

1

7676767676

~~~Stack Demo using SLL~~~

1:Push operation

2: Pop operation

3: Display

4:Exit

Enter your choice for stack demo3

The contents of SLL:

||1|| USN:4AJ22CS005| Name:Reena| Branch:CSE| Sem:1| Ph:7676767676|

||2|| USN:4AJ22IS015| Name:Vijay| Branch:ISE| Sem:1| Ph:9898989898|

||3|| USN:4AJ21CS050| Name:Ajay| Branch:CSE| Sem:3| Ph:9999999999|

 No of student nodes is 3

~~~Stack Demo using SLL~~~

1:Push operation

2: Pop operation
```

```
3: Display

4:Exit

Enter your choice for stack demo2

The Student node with usn:4AJ22CS005 is deleted

~~~Stack Demo using SLL~~~

1:Push operation

2: Pop operation

3: Display

4:Exit

Enter your choice for stack demo3

The contents of SLL:

||1|| USN:4AJ22IS015| Name:Vijay| Branch:ISE| Sem:1| Ph:9898989898|

||2|| USN:4AJ21CS050| Name:Ajay| Branch:CSE| Sem:3| Ph:9999999999|

 No of student nodes is 2
```

Viva Questions:

1. What is Linked lists?
A linked list is a data structure that can store a collection of items. In other words, linked lists can be utilized to store several objects of the same type. Each unit or element of the list is referred as a node. Each node has its own data and the address of the next node. It is like a chain. Linked Lists are used to create graph and trees.

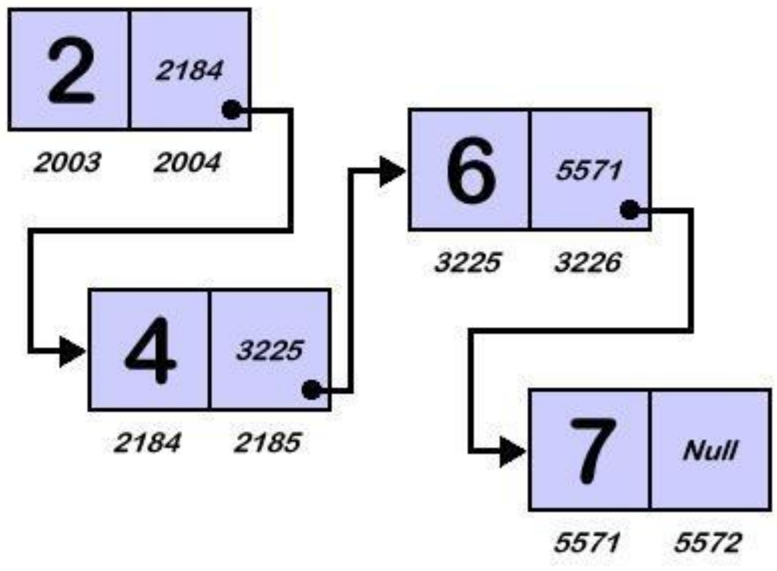
2. What type of memory allocation is referred for Linked lists?
Dynamic memory allocation is referred for Linked lists.

3. Mention what is traversal in linked lists?
Term Traversal is used to refer the operation of processing each element in the list.

4. Describe what is Node in link list? And name the types of Linked Lists?
Together (data + link) is referred as the Node. Types of Linked Lists are,

1. Singly Linked List
2. Doubly Linked List
3. Multiply Linked List
4. Circular Linked List

5. Mention what is Singly Linked list?
Singly Linked list are a type of data structure. In a singly linked list, each node in the list stores the contents of the node and a reference or pointer to the next node in the list. It does not store any reference or pointer to the previous node.



6. Mention what are the applications of Linked Lists?
Applications of Linked Lists are,

1. Linked lists are used to implement queues, stacks, graphs, etc.
2. In Linked Lists you don't need to know the size in advance.
3. Linked lists let you insert elements at the beginning and end of the list.

7. What does the dummy header in linked list contain?
In linked list, the dummy header contains the first record of the actual data

8. Mention the steps to insert data at the starting of a singly linked list?
Steps to insert data at the starting of a singly linked list include,

1. Create a new node
2. Insert new node by allocating the head pointer to the new node next pointer
3. Updating the head pointer to the point the new node.

PROGRAM 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.

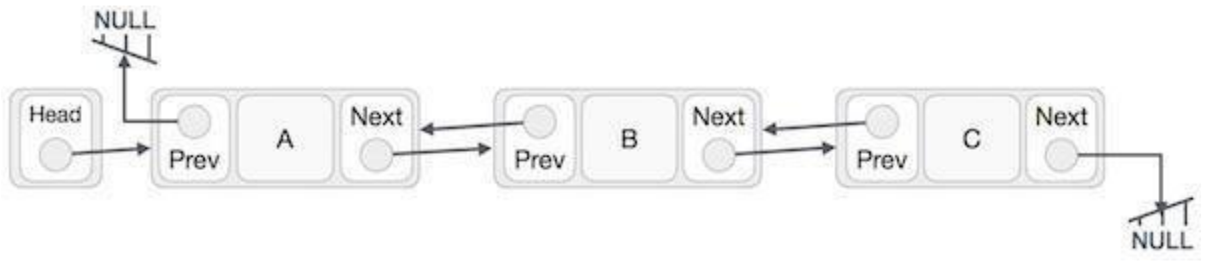
f. Exit

THEORY:

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward or backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **Prev** − Each link of a linked list contains a link to the previous link called Prev.
- **LinkedList** − A Linked List contains the connection link to the first link called First and to the last link called Last.

Doubly Linked List Representation



As per the above illustration, following are the important points to be considered.

- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

Basic Operations

Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Insert Last** − Adds an element at the end of the list.
- **Delete Last** − Deletes an element from the end of the list.
- **Insert After** − Adds an element after an item of the list.

- **Delete** − Deletes an element from the list using the key.
- **Display forward** − Displays the complete list in a forward manner.
- **Display backward** − Displays the complete list in a backward manner.

<u>ALGORITHM:</u>

Operations on Double Linked List

In a double linked list, we perform the following operations...

- Insertion
- Deletion
- Display
- Insertion

In a double linked list, the insertion operation can be performed in three ways as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

[Note : According to the question we will be seeing only insert at the beginning or front and insert at the end or last. So the 3 option is not explained here]

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the double linked list...

Step 1 - Create a newNode with given value and newNode → previous as NULL.
Step 2 - Check whether list is Empty (head == NULL)
Step 3 - If it is Empty then, assign NULL to newNode → next and newNode to head.
Step 4 - If it is not Empty then, assign head to newNode → next and newNode to head.

Inserting At End of the list

We can use the following steps to insert a new node at end of the double linked list...

Step 1 - Create a newNode with given value and newNode → next as NULL.
Step 2 - Check whether list is Empty (head == NULL)
Step 3 - If it is Empty, then assign NULL to newNode → previous and newNode to head.
Step 4 - If it is not Empty, then, define a node pointer temp and initialize with head.
Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).
Step 6 - Assign newNode to temp → next and temp to newNode → previous.

ALGORITHM FOR DELETION:

In a double linked list, the deletion operation can be performed in three ways as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

[Note : According to the question we will be seeing only delete at the beginning or front and delete at the end or last. So the 3 option is not explained here]

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the double linked list...

Step 1 - Check whether list is Empty (head == NULL)
Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
Step 3 - If it is not Empty then, define a Node pointer 'temp' and initialize with head.
Step 4 - Check whether list is having only one node (temp → previous is equal to temp → next)
Step 5 - If it is TRUE, then set head to NULL and delete temp (Setting Empty list conditions)
Step 6 - If it is FALSE, then assign temp → next to head, NULL to head → previous and delete temp.

Deleting from End of the list

We can use the following steps to delete a node from end of the double linked list...

Step 1 - Check whether list is Empty (head == NULL)
Step 2 - If it is Empty, then display 'List is Empty!!! Deletion is not possible' and terminate the function.
Step 3 - If it is not Empty then, define a Node pointer 'temp' and initialize with head.
Step 4 - Check whether list has only one Node (temp → previous and temp → next both are NULL)
Step 5 - If it is TRUE, then assign NULL to head and delete temp. And terminate from the function. (Setting Empty list condition)
Step 6 - If it is FALSE, then keep moving temp until it reaches to the last node in the list. (until temp → next is equal to NULL)
Step 7 - Assign NULL to temp → previous → next and delete temp.

ALGORITHM TO DISPLAY A DOUBLE LINKED LIST

We can use the following steps to display the elements of a double linked list...

Step 1 - Check whether list is Empty (head == NULL)
Step 2 - If it is Empty, then display 'List is Empty!!!' and terminate the function.
Step 3 - If it is not Empty, then define a Node pointer 'temp' and initialize with head.
Step 4 - Display 'NULL <--- '.
Step 5 - Keep displaying temp → data with an arrow (<===>) until temp reaches to the last node
Step 6 - Finally, display temp → data with arrow pointing to NULL (temp → data ---> NULL).

Program:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

        char ssn[25],name[25],dept[20],designation[25];

        int sal;

        long int phone;

        struct node *llink;

        struct node *rlink;

};

typedef struct node* NODE;

NODE first = NULL;

int count=0;

NODE create()

{

         NODE enode;

         enode = (NODE)malloc(sizeof(struct node));

         if( enode== NULL)

         {

                  printf("\nRunning out of memory");

                  exit(0);

          }

          printf("\nEnter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee: \n");

        scanf("%s  %s  %s  %s  %d  %ld", enode->ssn,  enode->name,  enode->dept,  enode->designation,
&enode->sal, &enode->phone);

          enode->llink=NULL;

          enode->rlink=NULL;

          count++;
          return enode;
}
NODE insertfront()
{
        NODE  temp;
        temp = create();
        if(first == NULL)
        {
               return temp;
         }
         temp->rlink = first;
         first->llink = temp;
         return temp;
}

void display()
{
         NODE cur;
         int nodeno=1;
         cur = first;
         if(cur == NULL)
                   printf("\nNo Contents to display in DLL");
         while(cur!=NULL)
        {
       printf("\nENode:%d||SSN:%s|Name:%s|Department:%s|Designation:%s|Salary:%d|Phone no:%ld",
nodeno, cur->ssn, cur->name,cur->dept, cur->designation, cur->sal, cur->phone);
               cur = cur->rlink;
                  nodeno++;
        }
         printf("\nNo of employee nodes is %d",count);
```

```
}


NODE deletefront()
{
        NODE temp;
        if(first == NULL)
        {
                printf("\nDoubly Linked List is empty");
                return NULL;
        }
        if(first->rlink== NULL)
       {
               printf("\nThe employee node with the ssn:%s is deleted", first->ssn);
              free(first);
              count--;
               return NULL;
        }
        temp = first;
        first = first->rlink;
        temp->rlink = NULL;
        first->llink = NULL;
        printf("\nThe employee node with the ssn:%s is deleted",temp->ssn);
        free(temp);
        count--;
        return first;
}
NODE insertend()
{
         NODE cur, temp;
         temp = create();
         if(first == NULL)
         {
                 return temp;
         }
        cur= first;
        while(cur->rlink!=NULL)
        {
                cur = cur->rlink;
        }

        cur->rlink = temp;
        temp->llink = cur;
        return first;
}
NODE deleteend()
{
        NODE prev,cur;
        if(first == NULL)
        {
                printf("\nDoubly Linked List is empty");
                return NULL;
        }

        if(first->rlink == NULL)
        {
                 printf("\nThe employee node with the ssn:%s is deleted",first->ssn);
                 free(first);
                 count--;
                return NULL;
        }
         prev=NULL;
         cur=first;
        while(cur->rlink!=NULL)
        {
                prev=cur;
                cur = cur->rlink;
         }
         cur->llink = NULL;
         printf("\nThe employee node with the ssn:%s is deleted",cur->ssn);
         free(cur);
         prev->rlink = NULL;
         count--;
         return first;
}
void deqdemo()
{
       int ch;
       while(1)
      {
             printf("\nDemo Double Ended Queue Operation");
      printf("\n1:InsertQueueFront\n 2: DeleteQueueFront\n 3:InsertQueueRear\n 4:DeleteQueueRear\n
5:DisplayStatus\n 6: Exit \n");
             scanf("%d", &ch);
             switch(ch)
            {
                    case 1: first=insertfront();
                                break;
                   case 2: first=deletefront();
```

```
                    break;
            case 3: first=insertend();
                    break;
            case 4: first=deleteend();
                   break;
             case 5: display();
                    break;
            default : return;
        }
    }
}

void main()
{
    int ch,i,n;
    while(1)
    {
        printf("\n\n~~~Menu~~~");
        printf("\n1:Create DLL of Employee Nodes");
        printf("\n2:DisplayStatus");
        printf("\n3:InsertAtEnd");
        printf("\n4:DeleteAtEnd");
        printf("\n5:InsertAtFront");
        printf("\n6:DeleteAtFront");
         printf("\n7:Double Ended Queue Demo using DLL");
         printf("\n8:Exit \n");
         printf("\nPlease enter your choice: ");
        scanf("%d",&ch);
        switch(ch)
```

```
          {
           case 1 : printf("\nEnter the no of Employees: ");
                    scanf("%d",&n);
                    for(i=1;i<=n;i++)
                    first = insertend();
                    break;

           case 2:  display();
                    break;

            case 3: first = insertend();
                   break;

            case 4: first = deleteend();
                   break;

            case 5: first = insertfront();
                   break;

            case 6: first = deletefront();
                 break;

            case 7: deqdemo();
                   break;

            case 8 : exit(0);
          default: printf("\nPlease Enter the valid choice");
          }
      }
```

```
}
```

Output:

```
~~~Menu~~~
1:Create DLL of Employee Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:InsertAtFront
6:DeleteAtFront
7:Double Ended Queue Demo using DLL
8:Exit

Please enter your choice: 1

Enter the no of Employees:  2

Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
S011
Ameya
Development
Developer
90000
9090909090

Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
S022
Deeksha
Sales
SalesRep
```

```
50000

8908908900



~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 2


ENode:1||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:2||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900

No of employee nodes is 2


~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL
```

```
8:Exit


Please enter your choice: 3


Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:

S033

Vishal

Operations

Admin

80000

8798798799




~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 2


ENode:1||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:2||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900
```

```
ENode:3||SSN:S033|Name:Vishal|Department:Operations|Designation:Admin|Salary:80000|Phone
no:8798798799

No of employee nodes is 3


~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 5


Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:

S004

Abhay

Devops

Devops

95000

9569569569



~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd
```

```
5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 2


ENode:1||SSN:S004|Name:Abhay|Department:Devops|Designation:Devops|Salary:95000|Phone
no:9569569569

ENode:2||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:3||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900

ENode:4||SSN:S033|Name:Vishal|Department:Operations|Designation:Admin|Salary:80000|Phone
no:8798798799

No of employee nodes is 4


~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 4


The employee node with the ssn:S033 is deleted
```

```
~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 2


ENode:1||SSN:S004|Name:Abhay|Department:Devops|Designation:Devops|Salary:95000|Phone
no:9569569569

ENode:2||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:3||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900

No of employee nodes is 3


~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit
```

```
Please enter your choice: 6


The employee node with the ssn:S004 is deleted


~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL

8:Exit


Please enter your choice: 2


ENode:1||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:2||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900

No of employee nodes is 2


~~~Menu~~~

1:Create DLL of Employee Nodes

2:DisplayStatus

3:InsertAtEnd

4:DeleteAtEnd

5:InsertAtFront

6:DeleteAtFront

7:Double Ended Queue Demo using DLL
```

```
8:Exit


Please enter your choice: 7


Demo Double Ended Queue Operation

1:InsertQueueFront

 2: DeleteQueueFront

 3:InsertQueueRear

 4:DeleteQueueRear

 5:DisplayStatus

 6: Exit

5


ENode:1||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:2||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900

No of employee nodes is 2

Demo Double Ended Queue Operation

1:InsertQueueFront

 2: DeleteQueueFront

 3:InsertQueueRear

 4:DeleteQueueRear

 5:DisplayStatus

 6: Exit

1


Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:

S002

Akash
```

```
Application

Developer

98000

7867867868


Demo Double Ended Queue Operation

1:InsertQueueFront

 2: DeleteQueueFront

 3:InsertQueueRear

 4:DeleteQueueRear

 5:DisplayStatus

 6: Exit



ENode:1||SSN:S002|Name:Akash|Department:Application|Designation:Developer|Salary:98000|Phone no:7867867868

ENode:2||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone no:9090909090

ENode:3||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone no:8908908900

No of employee nodes is 3

Demo Double Ended Queue Operation

1:InsertQueueFront

 2: DeleteQueueFront

 3:InsertQueueRear

 4:DeleteQueueRear

 5:DisplayStatus

 6: Exit

3
```

```
Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
S044
john
operations
Admin
98000
9239239233

Demo Double Ended Queue Operation
1:InsertQueueFront
 2: DeleteQueueFront
 3:InsertQueueRear
 4:DeleteQueueRear
 5:DisplayStatus
 6: Exit
5

ENode:1||SSN:S002|Name:Akash|Department:Application|Designation:Developer|Salary:98000|Phone no:7867867868
ENode:2||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone no:9090909090
ENode:3||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone no:8908908900
ENode:4||SSN:S044|Name:john|Department:operations|Designation:Admin|Salary:98000|Phone no:9239239233
No of employee nodes is 4
Demo Double Ended Queue Operation
1:InsertQueueFront
 2: DeleteQueueFront
 3:InsertQueueRear
 4:DeleteQueueRear
 5:DisplayStatus
 6: Exit
2
The employee node with the ssn:S002 is deleted
Demo Double Ended Queue Operation
1:InsertQueueFront
 2: DeleteQueueFront
 3:InsertQueueRear
 4:DeleteQueueRear
 5:DisplayStatus
 6: Exit
5

ENode:1||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone no:9090909090
ENode:2||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone no:8908908900
ENode:3||SSN:S044|Name:john|Department:operations|Designation:Admin|Salary:98000|Phone no:9239239233
No of employee nodes is 3
Demo Double Ended Queue Operation
1:InsertQueueFront
 2: DeleteQueueFront
 3:InsertQueueRear
 4:DeleteQueueRear
 5:DisplayStatus
 6: Exit
4

The employee node with the ssn:S044 is deleted
```

```
Demo Double Ended Queue Operation

1:InsertQueueFront

 2: DeleteQueueFront

 3:InsertQueueRear

 4:DeleteQueueRear

 5:DisplayStatus

 6: Exit

5


ENode:1||SSN:S011|Name:Ameya|Department:Development|Designation:Developer|Salary:90000|Phone
no:9090909090

ENode:2||SSN:S022|Name:Deeksha|Department:Sales|Designation:SalesRep|Salary:50000|Phone
no:8908908900

No of employee nodes is 2

Demo Double Ended Queue Operation

1:InsertQueueFront

 2: DeleteQueueFront

 3:InsertQueueRear

 4:DeleteQueueRear

 5:DisplayStatus

 6: Exit
```

Viva Questions:

1. **Define Double Linked List**

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward or backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

• Link − Each link of a linked list can store a data called an element.
• Next − Each link of a linked list contains a link to the next link called Next.
• Prev − Each link of a linked list contains a link to the previous link called Prev.
• LinkedList − A Linked List contains the connection link to the first link called First and to the last link called Last.

2. **What are the different operations supported by a double linked list?**

Following are the basic operations supported by a list.

- Insertion − Adds an element at the beginning of the list.
- Deletion − Deletes an element at the beginning of the list.
- Insert Last − Adds an element at the end of the list.
- Delete Last − Deletes an element from the end of the list.
- Insert After − Adds an element after an item of the list.
- Delete − Deletes an element from the list using the key.
- Display forward − Displays the complete list in a forward manner.
- Display backward − Displays the complete list in a backward manner

PROGRAM 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) = $6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$

b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

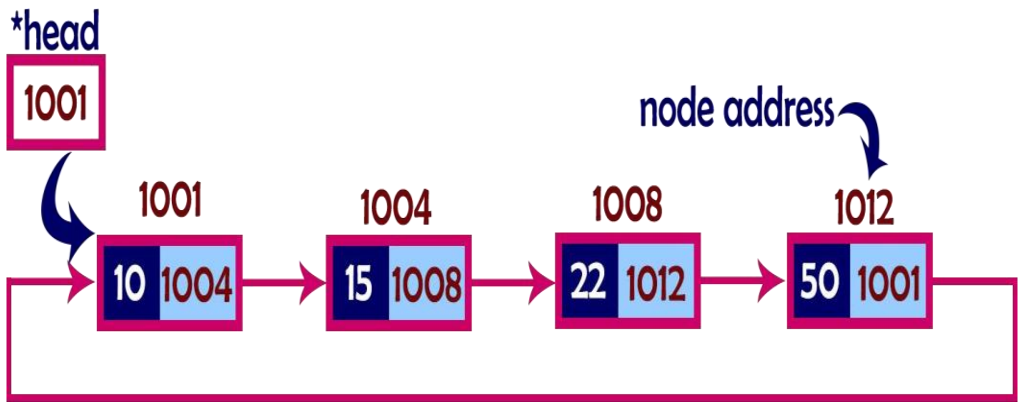
Support the program with appropriate functions for each of the above operations

<u>THEORY</u>:

In single linked list, every node points to its next node in the sequence and the last node points NULL. But in circular linked list, every node points to its next node in the sequence but the last node points to the first node in the list.

A circular linked list is a sequence of elements in which every element has a link to its next element in the sequence and the last element has a link to the first element.That means circular linked list is similar to the single linked list except that the last node points to the first node in the list

Example:



Operations

In a circular linked list, we perform the following operations...

1. Insertion
2. Deletion
3. Display

Before we implement actual operations, first we need to setup empty list. First perform the following steps before implementing actual operations.

Step 1 - Include all the header files which are used in the program.

Step 2 - Declare all the user defined functions.

Step 3 - Define a Node structure with two members data and next

Step 4 - Define a Node pointer 'head' and set it to NULL.

Step 5 - Implement the main method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

ALGORITHM FOR INSERTION:

Insertion

In a circular linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the circular linked list..

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set head = newNode and newNode→next = head .

Step 4 - If it is Not Empty then, define a Node pointer 'temp' and initialize with 'head'.

Step 5 - Keep moving the 'temp' to its next node until it reaches to the last node (until 'temp → next == head').

Step 6 - Set 'newNode → next =head', 'head = newNode' and 'temp → next = head'.

Inserting At End of the list

We can use the following steps to insert a new node at end of the circular linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL).

Step 3 - If it is Empty then, set head = newNode and newNode → next = head.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next == head).

Step 6 - Set temp → next = newNode and newNode → next = head.

ALGORITHM FOR DELETION:

In a circular linked list, the deletion operation can be performed in three ways those are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the circular linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize both 'temp1' and 'temp2' with head.

Step 4 - Check whether list is having only one node (temp1 → next == head)

Step 5 - If it is TRUE then set head = NULL and delete temp1 (Setting Empty list conditions)

Step 6 - If it is FALSE move the temp1 until it reaches to the last node. (until temp1 → next == head)

Step 7 - Then set head = temp2 → next, temp1 → next = head and delete temp2.

Deleting from End of the list

We can use the following steps to delete a node from end of the circular linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == head)

Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate from the function. (Setting Empty list condition)

Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until temp1 reaches to the last node in the list. (until temp1 → next == head)

Step 7 - Set temp2 → next = head and delete temp1.

ALGORITHM TO DISPLAY THE CIRCULAR LINKED LIST:

We can use the following steps to display the elements of a circular linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty, then display 'List is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node

Step 5 - Finally display temp → data with arrow pointing to head → data.**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 nonzero 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** $4x^2y - 2xy^2 + x - 7$ for $x = 3$ and $y = -1$.

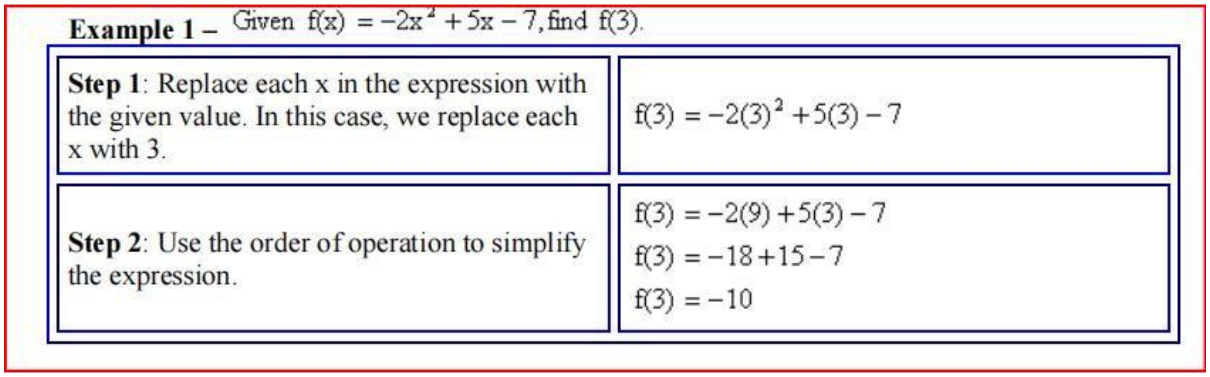
When a term contains both a number and a variable part, the number part is called the "coefficient". The coefficient on the leading term is called the "leading" coefficient.

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 given value.

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



Example 1 – Given $f(x) = -2x^2 + 5x - 7$, 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

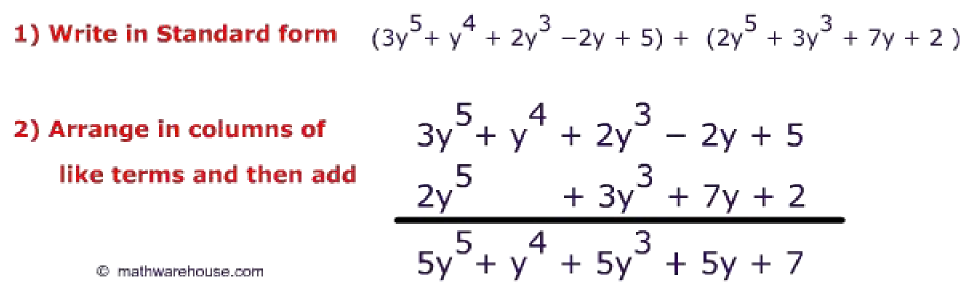
Standard form of a polynomial just means that the term with highest degree is first and each of the following terms

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*

$(3y^5 - 2y + y^4 + 2y^3 + 5)$ and $(2y^5 + 3y^3 + 2 + 7)$



1) Write in Standard form $(3y^5 + y^4 + 2y^3 - 2y + 5) + (2y^5 + 3y^3 + 7y + 2)$

2) Arrange in columns of like terms and then add

$$\begin{array}{rl} & 3y^5 + y^4 + 2y^3 - 2y + 5 \\ & 2y^5 + 3y^3 + 7y + 2 \\ \hline & 5y^5 + y^4 + 5y^3 + 5y + 7 \end{array}$$

© mathwarehouse.com

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define COMPARE(x, y)     ( (x == y) ? 0 : (x > y) ? 1 : -1)

struct node
{
            int coef;
            int xexp, yexp, zexp;
            struct node *link;
};
typedef struct node *NODE;

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

NODE attach(int coef, int xexp, int yexp, int zexp, NODE head)
{
           NODE temp, cur;
           temp = getnode();
           temp->coef = coef;
           temp->xexp = xexp;
           temp->yexp = yexp;
           temp->zexp = zexp;
           cur = head->link;
           while(cur->link != head)
           {
                      cur = cur->link;
           }
           cur->link = temp;
           temp->link = head;
           return head;
}

NODE read_poly(NODE head)
{
           int i, j, coef, xexp, yexp, zexp, n;
           printf("\nEnter the no of terms in the polynomial: ");
           scanf("%d", &n);
           for(i=1; i<=n; i++)
           {
                      printf("\n\tEnter the %d term: ",i);
                      printf("\n\t\tCoef = ");
                      scanf("%d", &coef);
                      printf("\n\t\tEnter Pow(x) Pow(y) and Pow(z): ");
                      scanf("%d", &xexp);
                      scanf("%d", &yexp);
                      scanf("%d", &zexp);
                      head = attach(coef, xexp, yexp, zexp, head);
           }
            return head;
}

void display(NODE head)
{
           NODE temp;
           if(head->link == head)
           {
                    printf("\nPolynomial does not exist.");
                    return;
           }
           temp = head->link;

           while(temp != head)
           {
                    printf("%dx^%dy^%dz^%d", temp->coef, temp->xexp, temp->yexp, temp->zexp);
                    temp = temp->link;
                    if(temp != head)
                            printf(" + ");
           }
}

int poly_evaluate(NODE head)
{
           int x, y, z, sum = 0;
           NODE poly;

           printf("\nEnter the value of x,y and z: ");
           scanf("%d %d %d", &x, &y, &z);
           poly = head->link;
           while(poly != head)
           {
                      sum += poly->coef * pow(x,poly->xexp)* pow(y,poly->yexp) * pow(z,poly->zexp);
                      poly = poly->link;
           }
           return sum;
}
,,,
NODE poly_sum(NODE head1, NODE head2, NODE head3)
{
   NODE a, b;
   int coef;
   a = head1->link;
   b = head2->link;

   while(a!=head1 && b!=head2)
   {
           while(1)
           {
                      if(a->xexp == b->xexp && a->yexp == b->yexp && a->zexp == b->zexp)
                      {
                              coef = a->coef + b->coef;
                              head3 = attach(coef, a->xexp, a->yexp, a->zexp, head3);
                              a = a->link;
                              b = b->link;
                              break;
                      } //if ends here
                      if(a->xexp!=0 || b->xexp!=0)
                      {
                              switch(COMPARE(a->xexp, b->xexp))
                              {
                            case -1 :  head3 = attach(b->coef, b->xexp, b->yexp, b->zexp, head3);
                                       b = b->link;
                                       break;

                              case 0 :    if(a->yexp > b->yexp)
                                       {
                                               head3 = attach(a->coef, a->xexp, a->yexp, a->zexp, head3);
                                               a = a->link;
                                               break;
                                        }
                                        else if(a->yexp < b->yexp)
                                        {
                                                head3 = attach(b->coef, b->xexp, b->yexp, b->zexp, head3);
                                                b = b->link;
                                                break;
                                        }
                                       else if(a->zexp > b->zexp)
                                       {
```

```
                        head3 = attach(a->coef, a->xexp, a->yexp, a->zexp, head3);
                        a = a->link;
                        break;
                    }
                    else if(a->zexp < b->zexp)
                    {
                        head3 = attach(b->coef, b->xexp, b->yexp, b->zexp, head3);
                        b = b->link;
                        break;
                    }
        case 1 :    head3 = attach(a->coef,a->xexp,a->yexp,a->zexp,head3);
                    a = a->link;
                    break;
        }  //switch ends here
        break;
} //if ends here
if(a->yexp!=0 || b->yexp!=0)
{
        switch(COMPARE(a->yexp, b->yexp))
        {
            case -1 :  head3 = attach(b->coef, b->xexp, b->yexp, b->zexp, head3);
                       b = b->link;
                       break;
            case 0 :  if(a->zexp > b->zexp)
                       {
                           head3 = attach(a->coef, a->xexp, a->yexp, a->zexp, head3);
                           a = a->link;
                           break;
                       }
                       else if(a->zexp < b->zexp)
                       {
                           head3 = attach(b->coef, b->xexp, b->yexp, b->zexp, head3);
                           b = b->link;
                           break;
                       }
            case 1 :    head3 = attach(a->coef, a->xexp, a->yexp, a->zexp, head3);
                        a = a->link;
                        break;
        }
        break;
}
if(a->zexp!=0 || b->zexp!=0)
{
        switch(COMPARE(a->zexp,b->zexp))
        {
                case -1 :  head3 = attach(b->coef,b->xexp,b->yexp,b->zexp,head3);
                           b = b->link;
                           break;
                case 1 :    head3 = attach(a->coef, a->xexp, a->yexp, a->zexp, head3);
                           a = a->link;
```

```
                                        break;
                            }
                            break;
                    }
            }
    }
    while(a!= head1)
    {
             head3 = attach(a->coef,a->xexp,a->yexp,a->zexp,head3);
            a = a->link;
    }
    while(b!= head2)
    {
            head3 = attach(b->coef,b->xexp,b->yexp,b->zexp,head3);
            b = b->link;
    }
    return head3;
}



void main()
{
  NODE head, head1, head2, head3;
  int res, ch;
  head =  getnode();     /* For polynomial evalaution */
  head1 = getnode();    /* To hold POLY1 */
  head2 = getnode();    /* To hold POLY2 */
  head3 = getnode();    /* To hold POLYSUM */

  head->link=head;
  head1->link=head1;
  head2->link=head2;
  head3->link= head3;

  while(1)
  {
     printf("\n~~~Menu~~~");
     printf("\n1.Represent and Evaluate a Polynomial P(x,y,z)");
     printf("\n2.Find the sum of two polynomials POLY1(x,y,z)");
     printf("\nEnter your choice:");
     scanf("%d",&ch);

     switch(ch)
     {
        case 1:          printf("\n~~~~Polynomial evaluation P(x,y,z)~~~\n");
                         head = read_poly(head);
                         printf("\nRepresentation of Polynomial for evaluation: \n");
                         display(head);
                         res = poly_evaluate(head);
                         printf("\nResult of polynomial evaluation is : %d \n", res);
```

```
                        break;
        case 2:         printf("\nEnter the POLY1(x,y,z):  \n");
                        head1 = read_poly(head1);
                        printf("\nPolynomial 1 is: \n");
                        display(head1);

                        printf("\nEnter the POLY2(x,y,z):  \n");
                        head2 = read_poly(head2);
                        printf("\nPolynomial 2 is: \n");
                        display(head2);

                        printf("\nPolynomial addition result: \n");
                        head3 = poly_sum(head1,head2,head3);
                        display(head3);
                        break;
          case 3:       exit(0);
      }
    }
}
```

Output:

```
~~~Menu~~~

1.Represent and Evaluate a Polynomial P(x,y,z)

2.Find the sum of two polynomials POLY1(x,y,z)

Enter your choice:1

~~~~Polynomial evaluation P(x,y,z)~~~



Enter the no of terms in the polynomial: 5

Enter the 1 term:

            Coef =  6

            Enter Pow(x) Pow(y) and Pow(z): 2 2 1

            Enter the 2 term:

            Coef =  -4

            Enter Pow(x) Pow(y) and Pow(z): 0 1 5
```

```
                    Enter the 3 term:

                    Coef =  3

                    Enter Pow(x) Pow(y) and Pow(z): 3 1 1

                    Enter the 4 term:

                    Coef =  2

                    Enter Pow(x) Pow(y) and Pow(z): 1 5 1

                    Enter the 5 term:

Coef =  -2

Enter Pow(x) Pow(y) and Pow(z): 1 1 3

Representation of Polynomial for evaluation:

6x^2y^2z^1 + -4x^0y^1z^5 + 3x^3y^1z^1 + 2x^1y^5z^1 + -2x^1y^1z^3

Enter the value of x,y and z: 1 1 1

Result of polynomial evaluation is : 5



~~~Menu~~~

1.Represent and Evaluate a Polynomial P(x,y,z)

2.Find the sum of two polynomials POLY1(x,y,z)

Enter your choice:2

Enter the POLY1(x,y,z):



Enter the no of terms in the polynomial: 5

Enter the 1 term:

                    Coef =  6
```

```
                    Enter Pow(x) Pow(y) and Pow(z): 4 4 4

                    Enter the 2 term:

                    Coef =  3

                    Enter Pow(x) Pow(y) and Pow(z): 4 3 1

                    Enter the 3 term:

                    Coef =  5

                    Enter Pow(x) Pow(y) and Pow(z): 0 1 1

                    Enter the 4 term:

                    Coef =  10

                    Enter Pow(x) Pow(y) and Pow(z): 0 1 0

                    Enter the 5 term:

                    Coef =  5

                    Enter Pow(x) Pow(y) and Pow(z): 0 0 0

                    Polynomial 1 is:

6x^4y^4z^4 + 3x^4y^3z^1 + 5x^0y^1z^1 + 10x^0y^1z^0 + 5x^0y^0z^0

Enter the POLY2(x,y,z):



Enter the no of terms in the polynomial: 5

Enter the 1 term:

                    Coef =  8

                    Enter Pow(x) Pow(y) and Pow(z): 4 4 4

                    Enter the 2 term:

                    Coef =  4
                    Enter Pow(x) Pow(y) and Pow(z): 4 2 1

                    Enter the 3 term:

                    Coef =  30

                    Enter Pow(x) Pow(y) and Pow(z): 0 1 0

                    Enter the 4 term:

          Coef =  20

          Enter Pow(x) Pow(y) and Pow(z): 0 0 1

          Enter the 5 term:

                    Coef =  3

                    Enter Pow(x) Pow(y) and Pow(z): 0 0 0

                    Polynomial 2 is:

8x^4y^4z^4 + 4x^4y^2z^1 + 30x^0y^1z^0 + 20x^0y^0z^1 + 3x^0y^0z^0

Polynomial addition result:

14x^4y^4z^4 + 3x^4y^3z^1 + 4x^4y^2z^1 + 5x^0y^1z^1 + 40x^0y^1z^0 + 20x^0y^0z^1 + 8x^0y^0z^0
```

Viva Questions:

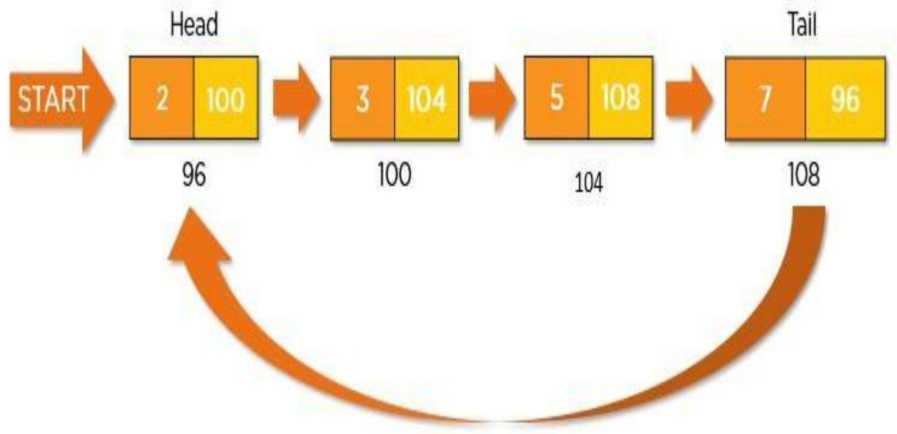
1. What is a Polynomial?

Algebraic expressions with non-negative power are called polynomials. Polynomials are the addition of monomials, binomials, and others. For example, f(x) = 3x2 + 4x + 5 is a polynomial with a degree of 2. The polynomial formula has both like terms and unlike terms.

- Like terms have the same powers of the same variable.
- Unlike terms have different variables and different powers.

2. What is Circular Linked List?

A circular Linked list is a unidirectional linked list. So, you can traverse it in only one direction. But this type of linked list has its last node pointing to the head node. So while traversing, you need to be careful and stop traversing when you revisit the head node.



3. What are the two types of circular linked list?

There are two types of circular linked lists: singly linked and doubly linked. In a singly linked circular linked list, each node has a pointer that points to the next node in the list. The last node in the list points back to the first node. In a doubly linked circular linked list, each node has pointers that point to both the next node and the previous node.

4. Define Singly Circular List

A Singly Circular Linked List is a data structure where each element in the list, known as a node, has two components: data and a reference (or link) to the next node in the sequence. In a circular linked list, the last node's reference points back to the first node, creating a closed loop. It's called "singly" because each node has a link to the next node, but not to the previous one (unlike a doubly linked list).

5. Define the basic structure for a node in a Singly Circular Linked List

```
struct Node {

   int data;           // Data stored in the node

   struct Node* next;    // Reference (pointer) to the next node

};
```

PROGRAM 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

THEORY:

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than or equal to its parent node's key.

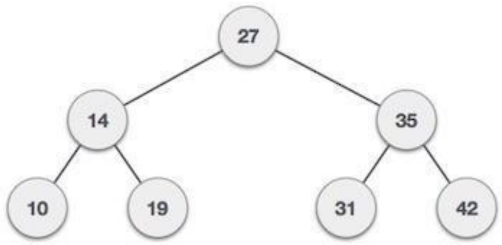
Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as –

```
left_subtree (keys)  ≤  node (key) ≤  right_subtree (keys)
```

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST –



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

Basic Operations

Following are the basic operations of a tree –

- **Search** – Searches an element in a tree.
- **Insert** – Inserts an element in a tree.
- **Pre-order Traversal** – Traverses a tree in a pre-order manner.
- **In-order Traversal** – Traverses a tree in an in-order manner.
- **Post-order Traversal** – Traverses a tree in a post-order manner.

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree –

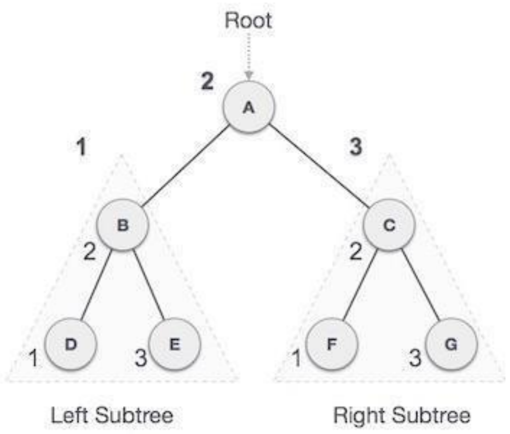
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.



We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be –

D → B → E → A → F → C → G

Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be –

A → B → D → E → C → F → G

Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

We start from **A**, and following pre-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be –

D → E → B → F → G → C → A

ALGORITHM:

Algorithm: Binary_Search_Tree_operations

```
Step1: Start
Step2: Read choice for BST operations
Step3: Switch ch
        case 1: Read n elements one by one and insert them into tree insert(item,root)
        case 2: Traversal(root)
        case 3: Search(root)
        case 4: Exit Step4:Stop
```

Algorithm: insert(item,root)

```
Step1: Start
Step2: Create newnode temp=getnode() temp->info=item
Step3: if root is NULL set temp as root otherwise compare temp->info with root if it is lesser move left
otherwise move towards right insert_node
Step4: Repeat step 2 until you find leaf node
Step5: Stop
```

Algorithm:Pre(root)

```
Step1: Start
Step2: if root!=NULL print root pre(root->llink) pre(root->rlink)
Step3: Stop
```

Algorithm:Post(root)

```
Step1: Start
Step2: if root!=NULL post(root->llink) post(root->rlink) print root
Step3: Stop
```

Algorithm:In(root)

```
Step1: Start
Step2: if root!=NULL in(root->llink) print root in(root->rlink)
Step3: Stop
```

Algorithm:Traversal(root)

```
Step1: Start
Step2: pre(root)
Step3: post(root)
Step4: in(root)
Step5: Stop
```

Algorithm:Search(root)

```
Step1: Read key element to be inserted
Step2: if key=root->info
```

```
                print successful search
        else if key>root->info
                root=root->rlink
        else if keyinfo
                root=root->llink
        else
                print unsuccessful search
```

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct BST
{
        int data;
        struct BST *lchild;
        struct BST *rchild;
};
typedef struct BST * NODE;
NODE create()
{
        NODE temp;
        temp = (NODE) malloc(sizeof(struct BST));
        printf("\nEnter The value: ");
        scanf("%d", &temp->data);


        temp->lchild = NULL;
        temp->rchild = NULL;
        return temp;
}

void insert(NODE root, NODE newnode);
void inorder(NODE root);
void preorder(NODE root);
void postorder(NODE root);
void search(NODE root);

void insert(NODE root, NODE newnode)
{
   /*Note: if newnode->data ==  root->data it will be skipped. No duplicate nodes are allowed */

        if (newnode->data < root->data)
        {
                if (root->lchild == NULL)
                        root->lchild = newnode;
                else
                        insert(root->lchild, newnode);
        }
        if (newnode->data > root->data)
        {
                if (root->rchild == NULL)
                        root->rchild = newnode;
                else
                        insert(root->rchild, newnode);
        }
}

void search(NODE root)
{
        int key;
        NODE cur;
        if(root == NULL)
        {
                printf("\nBST is empty.");
                return;
        }
        printf("\nEnter Element to be searched: ");
        scanf("%d", &key);
        cur = root;
        while (cur != NULL)
        {
                if (cur->data == key)
                {
                        printf("\nKey element is present in BST");
                        return;
                }
                if (key < cur->data)
                        cur = cur->lchild;
                else
                        cur = cur->rchild;
        }
        printf("\nKey element is not found in the BST");
}

void inorder(NODE root)
{
        if(root != NULL)
        {
                inorder(root->lchild);
                printf("%d ", root->data);
                inorder(root->rchild);
        }
}

void preorder(NODE root)
{
        if (root != NULL)
        {
                printf("%d ", root->data);
                preorder(root->lchild);
                preorder(root->rchild);
        }
}
```

```
void postorder(NODE root)
{
        if (root != NULL)
        {
                postorder(root->lchild);
                postorder(root->rchild);
                printf("%d ", root->data);
        }
}

void main()
{
        int ch, key, val, i, n;
        NODE root = NULL, newnode;
        while(1)
        {
                printf("\n~~~~BST MENU~~~~");
                printf("\n1.Create a BST");
                printf("\n2.Search");
                printf("\n3.BST Traversals: ");
                printf("\n4.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1:         printf("\nEnter the number of elements: ");
                                        scanf("%d", &n);
                                        for(i=1;i<=n;i++)
                                        {
                                                newnode = create();
                                                if (root == NULL)
                                                        root = newnode;
                                                else
```

```
                                        insert(root, newnode);
                                }
                                break;
                case 2:          if (root == NULL)
                                     printf("\nTree Is Not Created");
                                else
                                {
                                     printf("\nThe Preorder display : ");
                                     preorder(root);
                                     printf("\nThe Inorder display : ");
                                     inorder(root);
                                     printf("\nThe Postorder display : ");
                                     postorder(root);
                                }

                                break;
                case 3:         search(root);
                                break;

               case 4:     exit(0);
          }
     }
}
```

Output:

```
~~~~BST MENU~~~~
1.Create a BST
2.Search
3.BST Traversals:
4.Exit
Enter your choice: 1

Enter the number of elements: 12
Enter The value: 6
Enter The value: 9
Enter The value: 5
Enter The value: 2
Enter The value: 8
Enter The value: 15
Enter The value: 24
```

```
Enter The value: 14
Enter The value: 7
Enter The value: 8
Enter The value: 5
Enter The value: 2

~~~~BST MENU~~~~
1.Create a BST
2.Search
3.BST Traversals:
4.Exit
Enter your choice: 3

The Preorder display:
 6     5     2     9     8     7     15    14    24
The Inorder display:
2     5     6     7     8     9     14    15    24
The Postorder display:
2     5     7     8     14    24    15    9     6

~~~~BST MENU~~~~
1.Create a BST
2.Search
3.BST Traversals:
4.Exit
Enter your choice: 2

Enter Element to be searched: 66
Key element is not found in the BST

~~~~BST MENU~~~~
1.Create a BST
2.Search
3.BST Traversals:
4.Exit
Enter your choice: 2

Enter Element to be searched: 14
Key element is present in BST

~~~~BST MENU~~~~
1.Create a BST
2.Search
3.BST Traversals:
4.Exit
Enter your choice: 4
```

Viva Questions:

1. What is Binary Tree?

A binary tree is a 2-ary tree in which each node(N) has atmost 2 children (either 0 or 1). The node with 2 children are called internal nodes, and the nodes with 0 children are called external nodes or leaf nodes.

2. What is Complete Binary Tree?

A Binary tree is complete binary tree if all levels are completely filled except possibly the last/lowest level are full and in the last/lowest level all the items are on the left.

3. What is Perfect Balanced Binary Tree?

A perfect balanced binary tree is a binary tree where each node has same number of nodes in both subtrees.

4. Define Height in a tree?

Height is a general measured as number of edges from the root to deepest node in the tree.

5. What is tree traversal?

Traversal is used to visit each node in the tree exactly once. A full traversal of a binary tree gives a linear ordering of the data in the tree. There are 3 different type of tree traversal:

Inorder Traversal

Preorder Traversal

Postorder Traversal

6. How does Inorder Traversal work?

Traverse the left sub tree of the root node R in inorder.

Visit the root node R.

Traverse the right sub tree of the root node R in inorder.

7. How does Preorder Traversal work?

Traverse the left sub tree of the root node R in preorder.

Traverse the right sub tree of the root node R in preorder.

Visit the root node R.

8. How does Postorder Traversal work?

Visit the root node R.

Traverse the left sub tree of the root node R in postorder.

Traverse the right sub tree of the root node R in postorder.

9. What is a Binary Search Tree?

A BST(Binary Search Tree) is a binary tree in which each node satisfies search property. Each node's key value must be greater than all values in its left subtree and must be less than all values in its right subtree.

19. What is a AVL Tree?

AVL tree is a self-balancing binary search tree with height-balanced condition. For every node the height of left subtree and right subtree can be differ by at most 1.

20. What is a B-tree?

A B-Tree is a tree data structure that keeps data sorted and allows searches, insertions, deletions, and sequential access in logarithmic amount of time.

In B-Tree, internal nodes can have a variable number of child nodes within some pre-defined range. A B-tree is kept balanced by requiring that all leaf nodes are at the same depth.

A B-tree of order m is a tree which satisfies the following properties:

Every node has atmost 2m children.

Every node (except root and leaves) has atleast m children.

The root has atleast two children if it is not a leaf node.

All leaves appear in the same level, and carry information.

A non-leaf node with k children contain k-1 keys.

Leaf nodes contain atleast m-1 children and atmost 2m-1 keys.

PROGRAM 11:

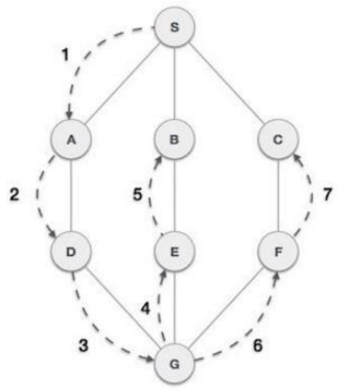
Develop a Program in C for the following operations on Graph(G) of Cities

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

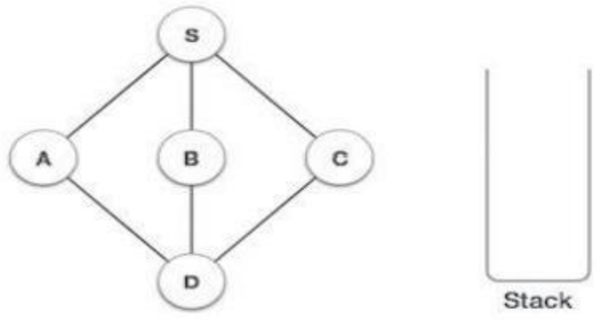
THEORY:

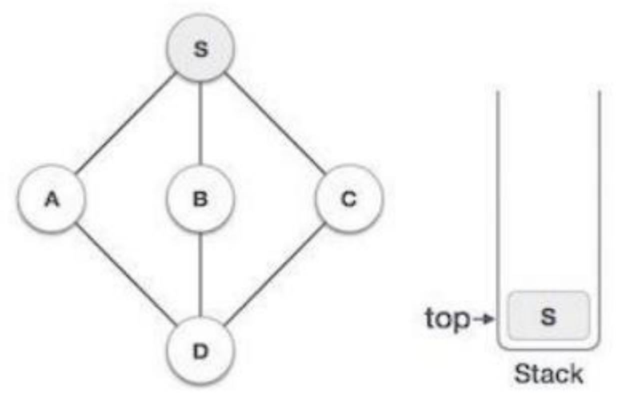
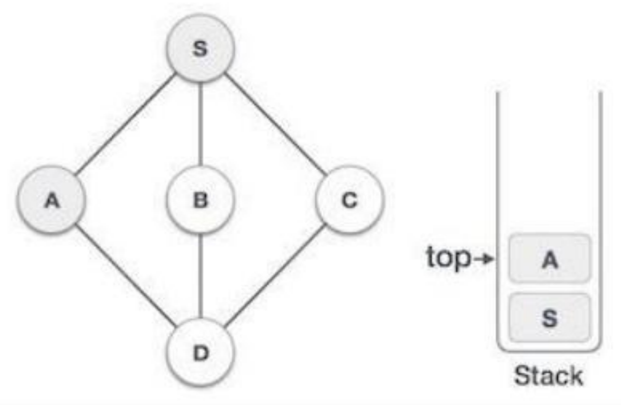
Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

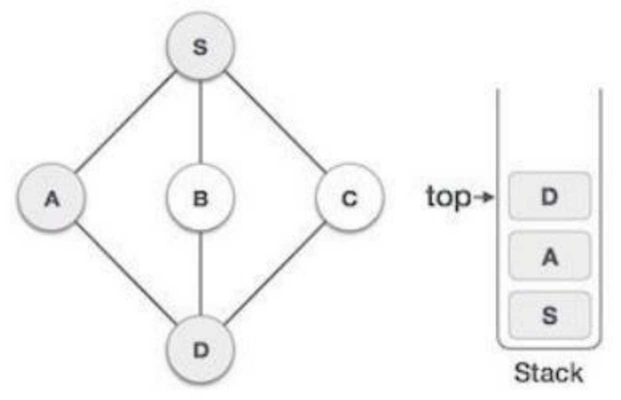
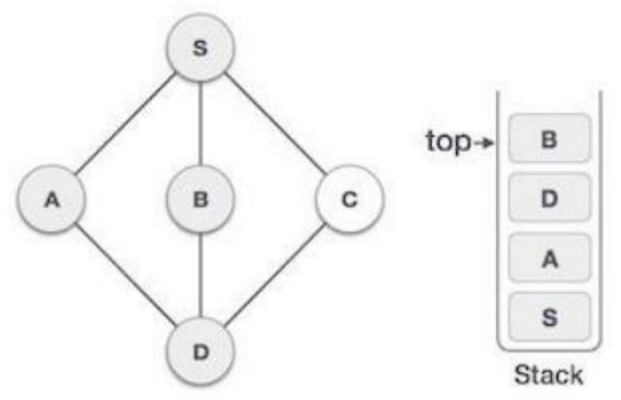


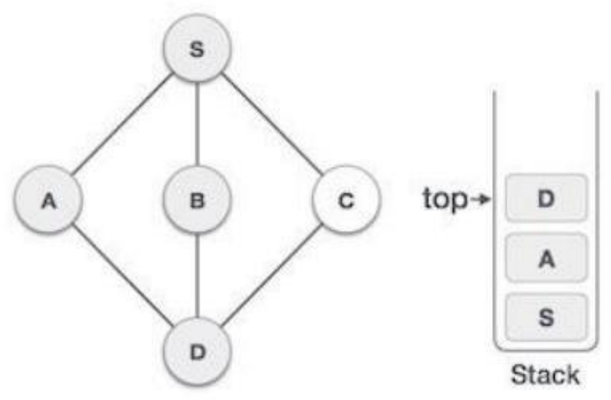
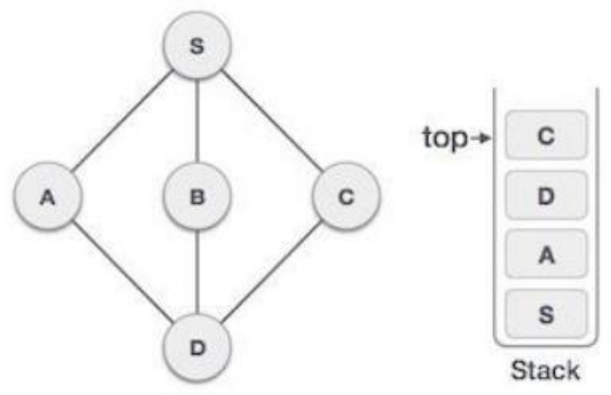
As in the example given above, DFS algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.

- Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

Step	Traversal	Description
1.		Initialize the stack.

2.		Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3.		Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S**and **D** are adjacent to **A** but we are concerned for unvisited nodes only.

4.		Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.
5.		We choose **B**, mark it as visited and put onto the stack. Here **B**does not have any unvisited adjacent node. So, we pop **B**from the stack.

6.		We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.
7.		Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

ALGORITHM:

Algorithm: Graph-Of-Cities

```
Step1: Start
Step2: Read no of cities
Step3: Read adjacent matrix for cities
Step4: Read choice for graph operations
Step5: Switch choice
        Case 1: Read source vertex
                Reachability(v)
                Print reachable nodes from source vertex
        Case 2: Connectivity(1)
                If all vertex are visited then print graph is connected
                Else graph is disconnected
        Case 3: Exit
Step6: Stop
```

Algorithm: Reachability(v)

```
Step1: Start
Step2: Repeat until n
        if (a[v][i] and ! visited [i])
```

```
            push s[++top]=i;
            if (top>=0)
            then visited (s[top])=1 reachability(s[top--])
Step3: Stop
```

Algorithm: Connectivity(r)

```
Step1: Start
Step2: Reapt until n.
            if (a[v][i] and !reach[i])
            print v to i connectivity
            connectivity (i)
Step3: Stop.
```

PROGRAM:

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


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


void bfs(int v)
{
int i, cur;
visited[v] = 1;
q[++rear] = v;
while(front!=rear)
{
            cur = q[++front];
for(i=1;i<=n;i++)
{
if((a[cur][i]==1)&&(visited[i]==0))
{
                     q[++rear] = i;
                    visited[i] = 1;
                    printf("->%d ", i);
}
}
}
}

void dfs(int v)
{
        int i;
visited[v]=1;
s[++top] = v;
for(i=1;i<=n;i++)
{
if(a[v][i] == 1&& visited[i] == 0 )
{
                printf("->%d ", i);
                dfs(i);
}
}
}

int main()
{

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

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


         case 2: printf("\nNodes reachable from starting vertex %d are:\n",start);
              dfs(start);
              break;
        case 3: exit(0);
        default: printf("\nPlease enter valid choice:");
      }
    }
}
```

Output 1:

```
Enter the number of vertices in graph: 9

Enter the adjacency matrix:

0 1 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0

0 0 0 1 1 1 0 0 1

0 0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 1 0

0 0 1 0 0 0 1 0 0

0 0 0 0 0 1 0 1 1

0 0 0 0 1 0 1 0 0

1 0 1 0 0 0 1 0 0

Enter the starting vertex: 1

==>1. BFS: Print all nodes reachable from a given starting node

==>2. DFS: Print all nodes reachable from a given starting node

==>3:Exit

Enter your choice: 1

Nodes reachable from starting vertex 1 are: ->2 ->9 ->3 ->7 ->4 ->5 ->6 ->8
```

Output 2:

```
Enter the number of vertices in graph: 9
Enter the adjacency matrix:

0 1 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0

0 0 0 1 1 1 0 0 1

0 0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 1 0

0 0 1 0 0 0 1 0 0

0 0 0 0 0 1 0 1 1

0 0 0 0 1 0 1 0 0

1 0 1 0 0 0 1 0 0

Enter the starting vertex: 1

==>1. BFS: Print all nodes reachable from a given starting node

==>2. DFS: Print all nodes reachable from a given starting node

==>3:Exit

Enter your choice: 2

Nodes reachable from starting vertex 1 are:

->2->9->3->4->5->8->7->6
```

Viva Questions:

1. Define graph?
Graph is a data structure that consists of following two components: A finite set of vertices also called as nodes. A finite set of ordered pair of the form (u, v) called as edge.

2. Define BFS?

In a breadth first search, you start at the root node, and then scan each node in the first level starting from the leftmost node, moving towards the right. Then you continue scanning the second level (starting from the left) and the third level, and so on until you've scanned all the nodes, or until you find the actual node that you were searching for.

3. Define DFS?

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

4. What is an adjacency matrix?

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

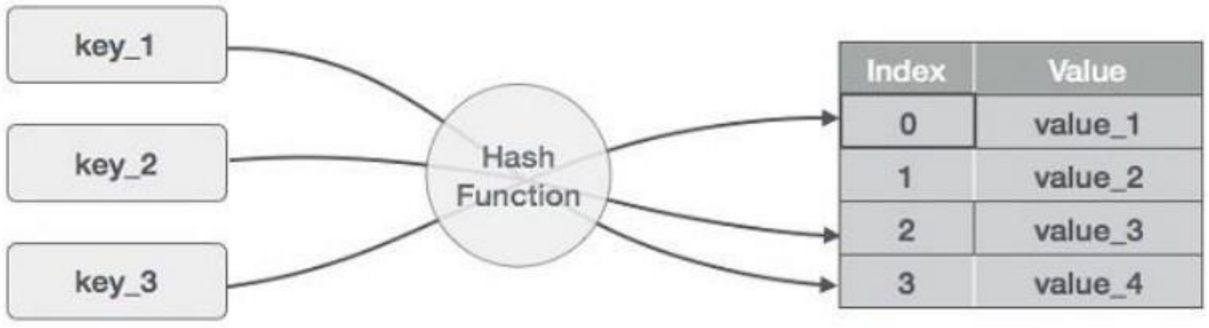
PROGRAM 12:

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

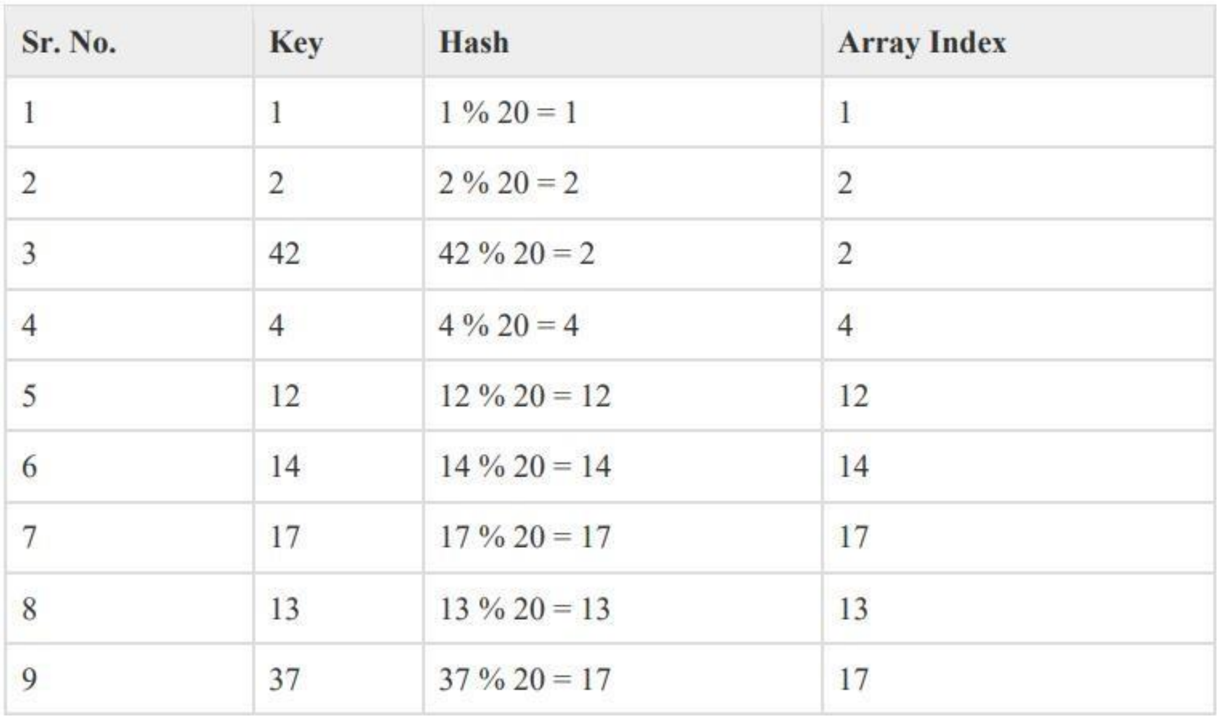
THEORY:

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data. Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

Hashing
Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.



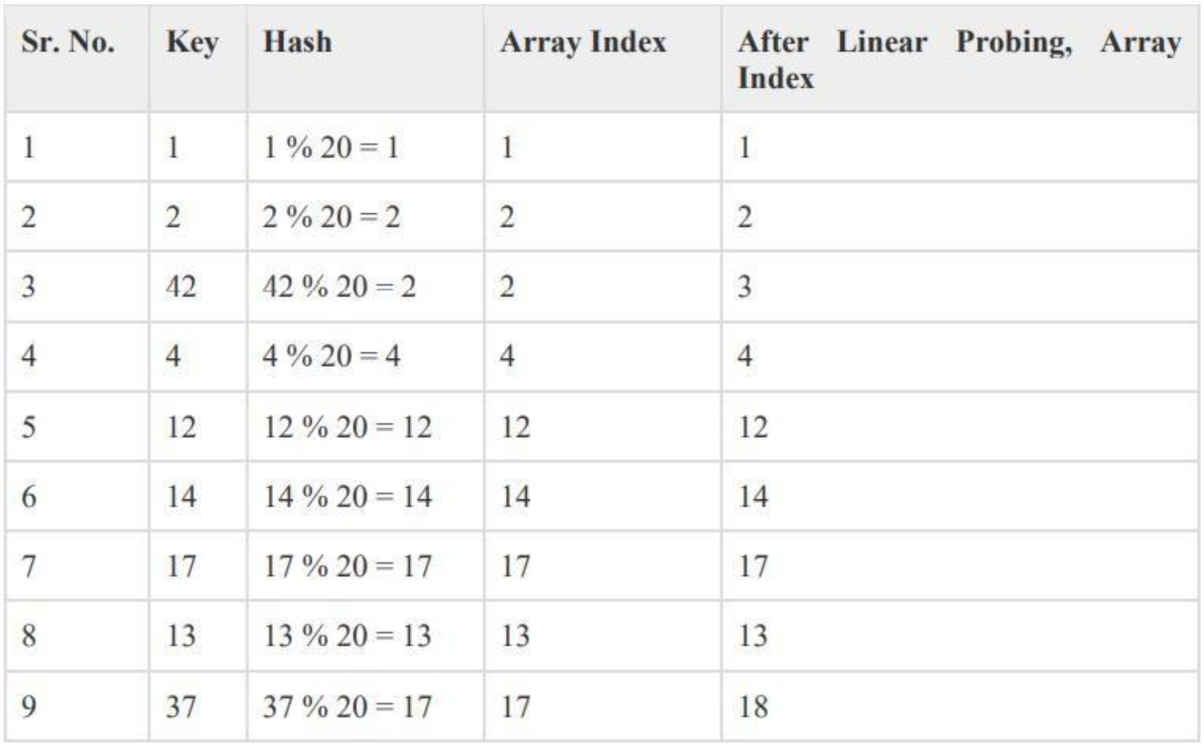
- (1,20)
- (2,70)
- (42,80)
- (4,25)
- (12,44)
- (14,32)
- (17,11)
- (13,78)
- (37,98)



Sr. No.	Key	Hash	Array Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14
7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

Linear Probing

As we can see, it may happen that the hashing technique is used to create an already used index of the array. In such a case, we can search the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called linear probing.



Sr. No.	Key	Hash	Array Index	After Linear Probing, Array Index
1	1	1 % 20 = 1	1	1
2	2	2 % 20 = 2	2	2
3	42	42 % 20 = 2	2	3
4	4	4 % 20 = 4	4	4
5	12	12 % 20 = 12	12	12
6	14	14 % 20 = 14	14	14
7	17	17 % 20 = 17	17	17
8	13	13 % 20 = 13	13	13
9	37	37 % 20 = 17	17	18

Basic Operations

Following are the basic primary operations of a hash table.

- **Search** − Searches an element in a hash table.
- **Insert** − inserts an element in a hash table.
- **delete** − Deletes an element from a hash table.

Algorithm for Linear Probing in Hash Table:

STEP 1: Start with an array (hash table) of fixed size, typically a prime number, where each element is a key-value pair. When a collision occurs, you need to resolve it using linear probing.

STEP 2: Compute the initial hash index for the key using a hash function, for example, hash(key) % array_size.

STEP 3: Check if the slot at the computed index is empty. If it's empty, insert the key-value pair at that location and finish the operation.

STEP 4: If the slot at the computed index is occupied, move linearly forward in the array (probe) by checking the next slot.

STEP 5: Keep probing linearly by checking the next slot until you find an empty slot.

STEP 6 : Insert the key-value pair into the first empty slot you find.

STEP 7: When searching for an existing key, follow the same probing process until you find the key or an empty slot. If you find the key, return its associated value; otherwise, the key is not in the hash table.

STEP 8: When deleting a key, follow the same probing process to locate the key. Once found, you can mark it as deleted (which is important to ensure that searching stops at the deleted slot) or replace it with a special "deleted" marker, depending on the implementation.

STEP 9: Linear probing can lead to clustering, where consecutive slots are filled, making it less efficient. To mitigate this, you can use techniques like double hashing, quadratic probing, or rehashing to spread out the elements more evenly.

Program:

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

int key[20],n,m;
int *ht,index;
int count = 0;

void insert(int key)
{
            index = key % m;
            while(ht[index] != -1)
            {
                        index = (index+1)%m;
            }
            ht[index] = key;
            count++;
 }

void display()
{
           int i;
           if(count == 0)
          {
                       printf("\nHash Table is empty");
                       return;
           }

           printf("\nHash Table contents are:\n ");
           for(i=0; i<m; i++)
                     printf("\n T[%d] --> %d ", i, ht[i]);
}


void main()
{
         int i;
         printf("\nEnter the number of employee  records (N) :  ");
         scanf("%d", &n);

         printf("\nEnter the two digit memory locations (m) for hash table:  ");
         scanf("%d", &m);

         ht = (int *)malloc(m*sizeof(int));
         for(i=0; i<m; i++)
                    ht[i] = -1;

         printf("\nEnter the four digit key values (K) for N Employee Records:\n  ");
         for(i=0; i<n; i++)
                   scanf("%d", &key[i]);

         for(i=0;i<n;i++)
        {
                  if(count == m)
                  {
```

```
                printf("\n~~~Hash table is full. Cannot insert the record %d key~~~",i+1);
                break;
            }
            insert(key[i]);
    }

        //Displaying Keys inserted into hash table
         display();
}
```

Output:

```
Enter the number of employee records (N) :   12

Enter the two digit memory locations (m) for hash table: 15

Enter the four digit key values (K) for N Employee Records:

1234

  5678

  3456

  2345

  6799

  1235

  7890

  3214

  3456

  1235

  5679

  2346

  Hash Table contents are:
```

```
T[0] --> 7890

T[1] --> -1

T[2] --> -1

T[3] --> -1

T[4] --> 1234

T[5] --> 2345

T[6] --> 3456

T[7] --> 6799

T[8] --> 5678

T[9] --> 1235

T[10] --> 3214

T[11] --> 3456

T[12] --> 1235

T[13] --> 5679

T[14] --> 2346
```

Viva Questions:

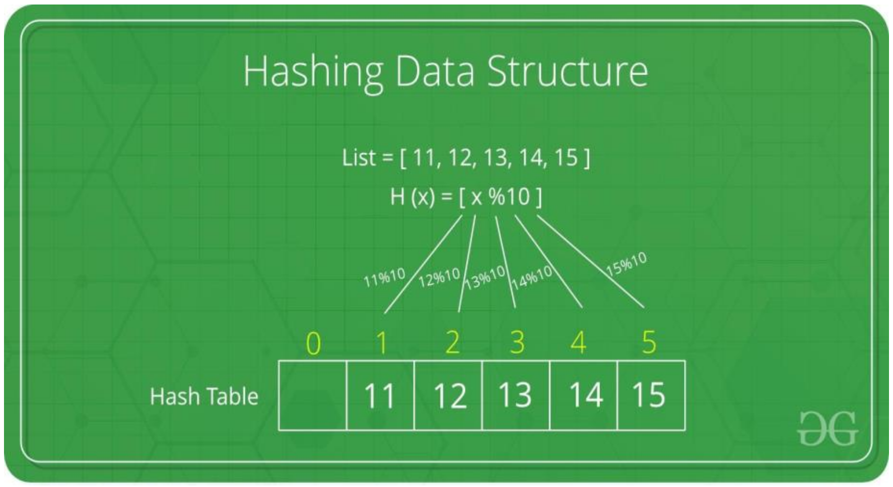
1. What is a Hash Table?

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

2. What is hashing?

Hashing refers to the process of generating a fixed-size output from an input of variable size using the mathematical formulas known as hash functions. This technique determines an index or location for the storage of an item in a data structure.



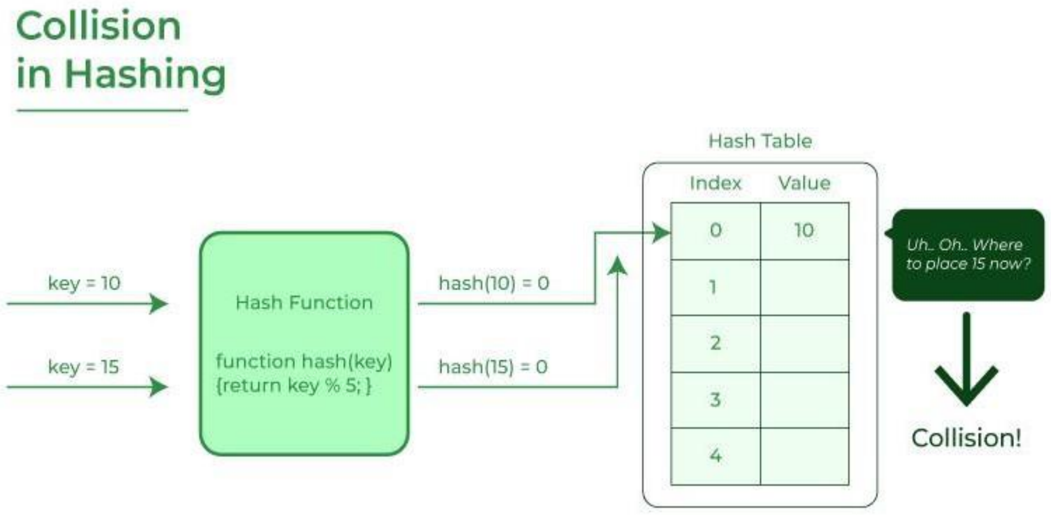
3. What are the components of Hashing?

There are majorly three components of hashing:

1. Key: A Key can be anything string or integer which is fed as input in the hash function the technique that determines an index or location for storage of an item in a data structure.
2. Hash Function: The hash function receives the input key and returns the index of an element in an array called a hash table. The index is known as the hash index.
3. Hash Table: Hash table is a data structure that maps keys to values using a special function called a hash function. Hash stores the data in an associative manner in an array where each data value has its own unique index.

4. What is Collision?

The hashing process generates a small number for a big key, so there is a possibility that two keys could produce the same value. The situation where the newly inserted key maps to an already occupied, and it must be handled using some collision handling technology.



5. What is Linear Probing technique?

Linear probing is a technique for handling collisions in hash tables. It works by placing the element in the next available slot in the hash table when a collision occurs. This process is repeated until a slot is found.

Linear probing is a simple and efficient technique for handling collisions. However, it can lead to clustering, which can degrade the performance of the hash table. Clustering occurs when elements are stored in consecutive slots in the hash table.

Here is an **example** of how linear probing works:

Hash table: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Element to insert: 10

Hash value of 10: 0

Slot 0 is occupied, so we move to the next slot.

Slot 1 is empty, so we place the element in slot 1.
Now, let's say we want to insert the element 11:

Hash table: [10, 11, 2, 3, 4, 5, 6, 7, 8, 9]

Element to insert: 11

Hash value of 11: 1

Slot 1 is occupied, so we move to the next slot.

Slot 2 is empty, so we place the element in slot 2.
As you can see, the elements 10 and 11 are stored in consecutive slots in the hash table. This is an example of clustering.

Clustering can degrade the performance of the hash table because it can increase the average search time. To avoid clustering, there are a number of other collision handling techniques that can be used, such as quadratic probing and double hashing.

Linear probing is a good choice for applications where performance is not critical and memory is limited. It is also a good choice for applications where the hash table is expected to be lightly loaded.

****