

# Channabasaveshwara Institute of Technology

(An ISO 9001:2008 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

# LAB MANUAL

(2016-17)

## 10CSL38 DATA STRUCTURES WITH C LABORATORY

III Semester

Name: \_\_\_\_\_

USN: \_\_\_\_\_

Batch: \_\_\_\_\_

**VISVESVARAYA TECHNOLOGICAL  
UNIVERSITY** Jnana Sangama, Belagavi, Karnataka –  
590018



**III SEMESTER / B.E.**

**DATA STRUCTURES WITH C  
LABORATORY (15CSL38)**

**LAB MANUAL**

**Prepared By**

**Pradeep M** *B.E., M.Tech*

**Assistant Professor**

*E mail: pradeep.m@cittumkur.org*

**Anil Kumar** *B.E., M.Tech*

**Assistant Professor**

*E mail: anilkumar9964@gmail.com*

## DATA STRUCTURES WITH C LABORATORY

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2015 -2016)

### SEMESTER - III

<b>Laboratory Code</b>	<b>15CSL38</b>	<b>IA Marks</b>	<b>20</b>
<b>Number of Lecture Hours/Week</b>	<b>01I + 02P</b>	<b>Exam Marks</b>	<b>80</b>
<b>Total Number of Lecture Hours</b>	<b>40</b>	<b>Exam Hours</b>	<b>03</b>

**CREDITS – 02**

**Course objectives:** This laboratory course enable students to get practical experience in design, develop, implement, analyze and evaluation/testing of

- Asymptotic performance of algorithms.
- Linear data structures and their applications such as Stacks, Queues and Lists
- Non-Linear Data Structures and their Applications such as Trees and Graphs
- Sorting and Searching Algorithms

**Descriptions (if any)**

**Implement all the experiments in C Language under Linux / Windows environment.**

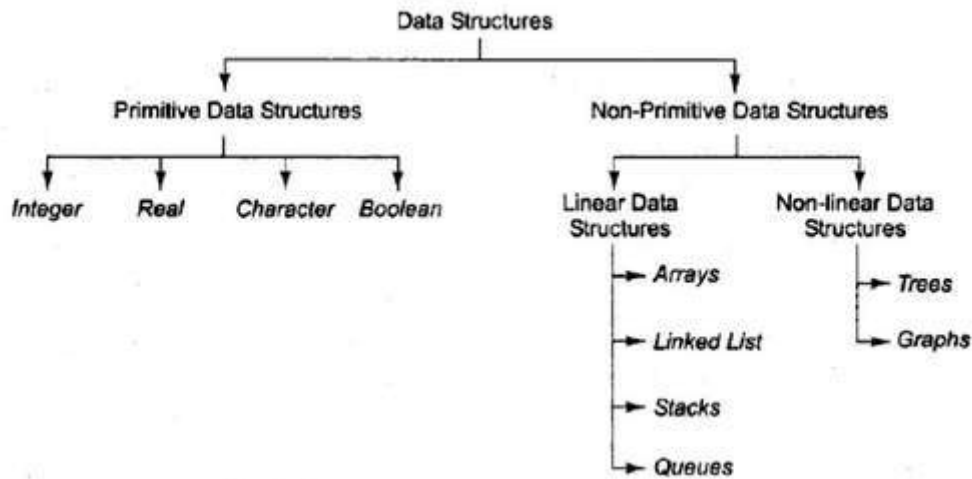
Ex. No.	Laboratory Experiments Description	Page No.
	<b>Introduction to Data Structures</b>	<b>1</b>
<b>1</b>	Design, Develop and Implement a menu driven Program in C for the following <b>Array</b> operations a. Creating an Array of <b>N</b> Integer Elements b. Display of Array Elements with Suitable Headings c. Inserting an Element ( <b>ELEM</b> ) at a given valid Position ( <b>POS</b> ) d. Deleting an Element at a given valid Position( <b>POS</b> ) e. Exit. Support the program with functions for each of the above operations.	<b>2</b>
<b>2</b>	Design, Develop and Implement a Program in C for the following operations on <b>Strings</b> a. Read a main String ( <b>STR</b> ), a Pattern String ( <b>PAT</b> ) and a Replace String ( <b>REP</b> ) b. Perform Pattern Matching Operation: Find and Replace all occurrences of <b>PAT</b> in <b>STR</b> with <b>REP</b> if <b>PAT</b> exists in <b>STR</b> . Report suitable messages in case <b>PAT</b> does not exist in <b>STR</b> Support the program with functions for each of the above operations. Don't use Built-in functions.	<b>7</b>
<b>3</b>	Design, Develop and Implement a menu driven Program in C for the following operations on <b>STACK</b> of Integers (Array Implementation of Stack with maximum size <b>MAX</b> ) a. <b>Push</b> an Element on to Stack b. <b>Pop</b> an Element from Stack c. Demonstrate how Stack can be used to check <b>Palindrome</b> d. Demonstrate <b>Overflow</b> and <b>Underflow</b> situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations	<b>10</b>
<b>4</b>	Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, %( <b>Remainder</b> ), ^ ( <b>Power</b> ) and <b>alphanumeric</b> operands.	<b>17</b>
<b>5</b>	Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of <b>Suffix expression</b> with single digit operands and operators: +, -, *, /, %, ^ b. Solving <b>Tower of Hanoi</b> problem with <b>n</b> disks	<b>21</b>
<b>6</b>	Design, Develop and Implement a menu driven Program in C for the following operations on <b>Circular QUEUE</b> of Characters (Array Implementation of Queue with maximum size <b>MAX</b> ) a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate <b>Overflow</b> and <b>Underflow</b> 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	<b>27</b>

7	Design, Develop and Implement a menu driven Program in C for the following operations on <b>Singly Linked List (SLL)</b> of Student Data with the fields: <i>USN, Name, Branch, Sem, PhNo</i> a. Create a <b>SLL</b> of <b>N</b> Students Data by using <i>front insertion</i> . b. Display the status of <b>SLL</b> and count the number of nodes in it c. Perform Insertion and Deletion at End of <b>SLL</b> d. Perform Insertion and Deletion at Front of <b>SLL</b> e. Demonstrate how this <b>SLL</b> can be used as <b>STACK</b> and <b>QUEUE</b> f. Exit	32
8	Design, Develop and Implement a menu driven Program in C for the following operations on <b>Doubly Linked List (DLL)</b> of Employee Data with the fields: <i>SSN, Name, Dept, Designation, Sal, PhNo</i> a. Create a <b>DLL</b> of <b>N</b> Employees Data by using <i>end insertion</i> . b. Display the status of <b>DLL</b> and count the number of nodes in it c. Perform Insertion and Deletion at End of <b>DLL</b> d. Perform Insertion and Deletion at Front of <b>DLL</b> e. Demonstrate how this <b>DLL</b> can be used as <b>Double Ended Queue</b> f. Exit	43
9	Design, Develop and Implement a Program in C for the following operations on <b>Singly Circular Linked List (SCLL)</b> 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 <b>POLY1(x,y,z)</b> and <b>POLY2(x,y,z)</b> and store the result in <b>POLYSUM(x,y,z)</b> Support the program with appropriate functions for each of the above operations	52
10	Design, Develop and Implement a menu driven Program in C for the following operations on <b>Binary Search Tree (BST)</b> of Integers a. Create a BST of <b>N</b> 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 ( <b>KEY</b> ) and report the appropriate message d. Delete an element( <b>ELEM</b> ) from BST e. Exit	61
11	Design, Develop and Implement a Program in C for the following operations on <b>Graph(G)</b> of Cities a. Create a Graph of <b>N</b> cities using Adjacency Matrix. b. Print all the nodes <b>reachable</b> from a given starting node in a digraph using <b>BFS</b> method c. Check whether a given graph is <b>connected</b> or not using <b>DFS</b> method.	70
12	Given a File of <b>N</b> employee records with a set <b>K</b> of Keys(4-digit) which uniquely determine the records in file <b>F</b> . Assume that file <b>F</b> is maintained in memory by a Hash Table(HT) of <b>m</b> memory locations with <b>L</b> as the set of memory addresses (2-digit) of locations in HT. Let the keys in <b>K</b> and addresses in <b>L</b> are Integers. Design and develop a Program in C that uses Hash function <b>H: K @L</b> as $H(K)=K \text{ mod } m$ ( <b>remainder</b> method), and implement hashing technique to map a given key <b>K</b> to the address space <b>L</b> . Resolve the collision (if any) using <b>linear probing</b> .	78
	<b>Viva Questions And Answers</b>	86
<p><b>Course outcomes:</b> On the completion of this laboratory course, the students will be able to:</p> <ul style="list-style-type: none"> <li>Analyze and Compare various linear and non-linear data structures</li> <li>Code, debug and demonstrate the working nature of different types of data structures and their applications</li> <li>Implement, analyze and evaluate the searching and sorting algorithms</li> <li>Choose the appropriate data structure for solving real world problems</li> </ul>		
<p><b>Graduate Attributes (as per NBA)</b></p> <ol style="list-style-type: none"> <li>Engineering Knowledge</li> <li>Problem Analysis</li> <li>Design/Development of Solutions</li> <li>4. Modern Tool Usage</li> </ol>		
<p><b>Conduction of Practical Examination:</b></p> <ol style="list-style-type: none"> <li>All laboratory experiments (<b>TWELVE</b> nos ) are to be included for practical examination.</li> <li>Students are allowed to pick one experiment from the lot.</li> <li>Strictly follow the instructions as printed on the cover page of answer script</li> <li>Marks distribution: Procedure + Conduction + Viva: <b>20 + 50 +10 (80)</b></li> <li><b>Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.</b></li> </ol>		

## INTRODUCTION TO DATA STRUCTURES

Data Structure is defined as the way in which data is organized in the memory location.

There are 2 types of data structures:



### Linear Data Structure:

In linear data structure all the data are stored linearly or contiguously in the memory. All the data are saved in continuously memory locations and hence all data elements are saved in one boundary. A linear data structure is one in which we can reach directly only one element from another while travelling sequentially. The main advantage, we find the first element, and then it's easy to find the next data elements. The disadvantage, the size of the array must be known before allocating the memory.

The different types of linear data structures are:

- Array
- Stack
- Queue
- Linked List

### Non-Linear Data Structure:

Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.

The various types of non linear data structures are:

- Trees
- Graphs

## EXPERIMENT - 01

Design, Develop and Implement a menu driven program in C for the following Array operations

- a. Creating Array of N Integer elements.
- b. Display of Array elements with suitable headings.
- c. Inserting an element (**ELEM**) at a given valid position (**POS**).
- d. Deleting an element at a given valid position (**POS**).
- e. Exit.

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

### ABOUT THE EXPERIMENT:

An **Array** is a collection of similar / same elements. In this experiment the array can be represented as one / single dimensional elements.

Menu driven program in c - language to perform various array operations are implemented with the help of user defined functions as followings;

- a. create()
- b. display()
- c. insert()
- d. del ()
- e. exit()

### ALGORITHM:

Step 1: Start.

Step 2: Read N value.

Step 3: Read Array of N integer elements

Step 4: Print array of N integer elements.

Step 5: Insert an element at given valid position in an array.

Step 6: Delete an element at given valid position from an array.

Step 7: Stop.

### PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
/* Global variables declaration */
int a[4], n, elem, i, pos;

/*Function Prototype and Definitions*/

void create()          //creating an array
{
    printf("\nEnter the size of the array elements: ");
    scanf("%d", &n);
    printf("\nEnter the elements for the array:\n");
    for(i=0; i<n; i++)
```

```
        scanf("%d", &a[i]);
    }    //end of create()

void display()    //displaying an array elements
{
    int i;
    printf("\nThe array elements are:\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t", a[i]);
    }
}    //end of display()

void insert()    //inserting an element in to an array
{
    printf("\nEnter the position for the new element: ");
    scanf("%d", &pos);
    printf("\nEnter the element to be inserted: ");
    scanf("%d", &elem);
    for(i=n-1; i>=pos; i--)
    {
        a[i+1] = a[i];
    }
    a[pos] = elem;
    n = n+1;
}    //end of insert()

void del()    //deleting an array element
{
    printf("\nEnter the position of the element to be deleted: ");
    scanf("%d", &pos);
    elem = a[pos];
    for(i=pos; i<n-1; i++)
    {
        a[i] = a[i+1];
    }
    n = n-1;
    printf("\nThe deleted element is = %d", elem);
} //end of delete()

void main()
{
```

```
int ch;
clrscr();
do{
    printf("\n\n-----Menu-----\n");
    printf("1.Create\n 2.Display\n 3.Insert\n 4.Delete\n 5.Exit\n");
    printf("-----");
    printf("\nEnter your choice: ");
    scanf("%d", &ch);
    switch(ch)
    {
        case 1: create();
                break;
        case 2: display();
                break;
        case 3: insert();
                break;
        case 4: del();
                break;
        case 5: exit(0);
                break;
        default: printf("\nInvalid choice:\n");
                break;
    }
}while(ch!=5);
getch();
} // end of main
```

**SAMPLE OUTPUT:**

```
-----Menu-----
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice: 1
Enter the size of the array elements: 5
Enter the elements for the array:
10 20 30 40 50
```

```
-----Menu-----
1.Create
2.Display
3.Insert
4.Delete
5.Exit
```



Enter your choice: 2

The array elements are:

10 20 30 40 50

-----Menu-----

1.Create

2.Display

3.Insert

4.Delete

5.Exit

Enter your choice: 3

Enter the position for the new element: 2

Enter the element to be inserted: 90

-----Menu-----

1.Create

2.Display

3.Insert

4.Delete

5.Exit

Enter your choice: 2

The array elements are:

10 20 **90** 30 40 50

-----Menu-----

1.Create

2.Display

3.Insert

4.Delete

5.Exit

Enter your choice: 4

Enter the position of the element to be deleted: 5

The deleted element is = **50**

-----Menu-----

1.Create

2.Display

3.Insert

4.Delete

5.Exit

Enter your choice: 2

The array elements are:

10 20 90 30 40

-----Menu-----

1.Create

2.Display

3.Insert

4.Delete

5.Exit

Enter your choice: 5

## EXPERIMENT - 02

Design, Develop and Implement a program in C for the following operations on Strings

- a. Read a Main String (STR), a Pattern String (PAT) and a Replace String (REP).
- b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Repost 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.

### ABOUT THE EXPERIMENT:

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

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows:

```
char greeting[] = "Hello";
```

C language supports a wide range of built-in functions that manipulate null-terminated strings as follows:

<code>strcpy(s1, s2);</code>	Copies string s2 into string s1.
<code>strcat(s1, s2);</code>	Concatenates string s2 onto the end of string s1.
<code>strlen(s1);</code>	Returns the length of string s1.
<code>strcmp(s1, s2);</code>	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
<code>strchr(s1, ch);</code>	Returns a pointer to the first occurrence of character ch in string s1.
<code>strstr(s1, s2);</code>	Returns a pointer to the first occurrence of string s2 in string s1.

### ALGORITHM:

Step 1: Start.

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

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

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

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

Step 6: Stop.

### PROGRAM CODE:

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

```
//Declarations
```

```
char str[100], pat[50], rep[50], ans[100];
int i, j, c, m, k, flag=0;

void stringmatch()
{
    i = m = c = j = 0;
    while(str[c] != '\0')
    {
        if(str[m] == pat[i])    // ..... matching
        {
            i++; m++;
            if(pat[i] == '\0')    //.....found occurrences.
            {
                flag = 1;
                //.... copy replace string in ans string.
                for(k = 0; rep[k] != '\0'; k++, j++)
                    ans[j] = rep[k];
                i = 0;
                c = m;
            }
        }    // if ends.
        else    //... mismatch
        {
            ans[j] = str[c];
            j++;
            c++;
            m = c;
            i = 0;
        }    //else ends
    }    //end of while
    ans[j] = '\0';
}    //end stringmatch()

void main()
{
    clrscr();
    printf("\nEnter a main string \n");
    gets(str);
    printf("\nEnter a pattern string \n");
    fflush();
    gets(pat);
    printf("\nEnter a replace string \n");
    fflush();
    gets(rep);
    stringmatch();
    if(flag == 1)
```

```
        printf("\nThe resultant string is\n %s" , ans);
    else
        printf("\nPattern string NOT found\n");
    getch();
} // end of main
```

**SAMPLE OUTPUT:****RUN 1:**

Enter a main string  
Test  
Enter a pattern string  
Te  
Enter a replace string  
Re

The resultant string is  
Rest

**RUN 2:**

Enter a main string  
This is Data Structure lab  
Enter a pattern string  
Data Structure  
Enter a replace string  
Data structure with C

The resultant string is  
This is Data structure with C lab

**RUN 3:**

Enter a main string  
This is Data Structure lab  
Enter a pattern string  
Date  
Enter a replace string  
DATA

Pattern string NOT found

## EXPERIMENT - 03

Design, Develop and Implement a menu driven program in C for the following operations on **STACK** of integers (Array implementation of stack with maximum size **MAX**)

- Push an element on to stack
- Pop an element from stack.
- Demonstrate how stack can be used to check palindrome.
- Demonstrate Overflow and Underflow situations on stack.
- Display the status of stack.
- Exit.

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

### ABOUT THE EXPERIMENT:

A stack is an abstract data type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack.

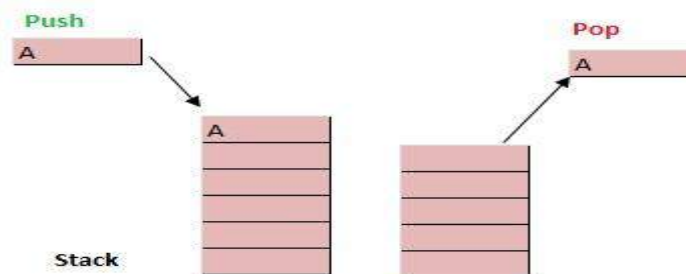
### EXAMPLES OF STACK:



A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO 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.

Below given diagram tries to depict 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:

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

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

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

### ALGORITHM:

Step 1: Start.

Step 2: Initialize stack size MAX and top of stack -1.

Step 3: Push integer element on to stack and display the contents of the stack.  
if stack is full give a message as ‘Stack is Overflow’.

Step 3: Pop element from stack along with display the stack contents.  
if stack is empty give a message as ‘Stack is Underflow’.

Step 4: Check whether the stack contents are Palindrome or not.

Step 5: Stop.

### PROGRAM CODE:

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

#define MAX 4

int stack[MAX], item;
int ch, top = -1, count = 0, status = 0;

/*PUSH FUNCTION*/
void push(int stack[], int item)
{
    if (top == (MAX-1))
        printf("\n\nStack is Overflow");
    else
    {
        stack[++top] = item;
        status++;
    }
}
```

```
/*POP FUNCTION*/
int pop(int stack[])
{
    int ret;
    if(top == -1)
        printf("\n\nStack is Underflow");
    else
    {
        ret = stack[top--];
        status--;
        printf("\nPopped element is %d", ret);
    }
    return ret;
}

/* FUNCTION TO CHECK STACK IS PALINDROME OR NOT */
void palindrome(int stack[])
{
    int i, temp;
    temp = status;
    for(i=0; i<temp; i++)
    {
        if(stack[i] == pop(stack))
            count++;
    }
    if(temp==count)
        printf("\nStack contents are Palindrome");
    else
        printf("\nStack contents are not palindrome");
}

/*FUNCTION TO DISPLAY STACK*/
void display(int stack[])
{
    int i;
    printf("\nThe stack contents are:");
    if(top == -1)
        printf("\nStack is Empty");
    else{
        for(i=top; i>=0; i--)
            printf("\n -----\n| %d |", stack[i]);
        printf("\n");
    }
}

/*MAIN PROGRAM*/
void main()
```



```

{
    clrscr();
    do{
        printf("\n\n---MAIN MENU---\n");
        printf("\n1. PUSH (Insert) in the Stack");
        printf("\n2. POP (Delete) from the Stack");
        printf("\n3. PALINDROME check using Stack");
        printf("\n4. Exit (End the Execution)");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch(ch){
        case 1: printf("\nEnter a element to be pushed: ");
                scanf("%d", &item);
                push(stack, item);
                display(stack);
                break;
        case 2: item=pop(stack);
                display(stack);
                break;
        case 3:
                palindrome(stack);
                break;
        case 4:
                exit(0); break;
        default:
                printf("\nEND OF EXECUTION");
        }//end switch
    }while (ch != 4);
    getch();
}

```

**SAMPLE OUTPUT:**

---MAIN MENU---

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 1

Enter an element to be pushed: 1

The stack contents are:

----

| 1 |

---MAIN MENU---

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack

```

3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:      1
Enter an element to be pushed:      2
The stack contents are:
-----
| 2 |
-----|
1 |

(----- AFTER THE 4 TIMES PUSH OPERATION -----)
---MAIN MENU---
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:      1
Enter an element to be pushed:      9
Stack is Overflow
The stack contents are:
-----
| 1 |
-----
| 2 |
-----|
2 |
-----|
1 |

---MAIN MENU---
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:      2
Popped element is      1

The stack contents are:
-----
| 2 |
-----|
2 |
-----|
1 |

(----- AFTER THE 4 TIMES POP OPERATION -----)
---MAIN MENU---
1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)
Enter Your Choice:      2
Stack is Underflow
The stack contents are:
Stack is Empty

```

(----- CHECKING FOR PALINDROME OR NOT USING STACK-----)

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 1  
Enter an element to be pushed: 1

The stack contents are:

-----  
| 1 |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 1  
Enter an element to be pushed: 2

The stack contents are:

-----  
2
1 |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 1  
Enter a element to be pushed: 1

The stack contents are:

-----  
1
2 |  
-----|  
1 |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 3  
**Stack contents are is Palindrome**

(----- CHECKING FOR PALINDROME OR NOT USING STACK-----)

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 1

Enter an element to be pushed: 1  
The stack contents are:  
-----|  
1 |

(AFTER 3 TIMES PUSH)

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 1  
Enter an element to be pushed: 3

The stack contents are:

-----  
3
2 |  
-----|  
1 |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Exit (End the Execution)

Enter Your Choice: 3

**Stack contents are not Palindrome**

## EXPERIMENT - 04

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

### ABOUT THE EXPERIMENT:

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

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

**postfix:** Operators are written after their operands. Ex: XY+

Examples of Infix, Prefix, and Postfix

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

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

**Step 1.** Reverse the infix expression.

5^E+D\*(C^B+A)

**Step 2.** Make Every '(' as ')' and every ')' as '('

5^E+D\*(C^B+A)

**Step 3.** Convert expression to postfix form.

**Step 4.** Reverse the expression.

+\*+A^BCD^E

**Step 5. Result**

+\*+A^BCD^E5

Expression	Stack	Output	Comment
A+(B*C-(D/E-F)*G)*H			
5^E+D*(C^B+A)	Empty	-	Initial
^E+D*(C^B+A)	Empty	5	Print
E+D*(C^B+A)	^	5	Push
+D*(C^B+A)	^	5E	Push
D*(C^B+A)	+	5E^	Pop And Push
*(C^B+A)	+	5E^D	Print
(C^B+A)	+*	5E^D	Push
C^B+A)	+*(	5E^D	Push
^B+A)	+*(	5E^DC	Print
B+A)	+*(^	5E^DC	Push
+A)	+*(^	5E^DCB	Print
A)	+*(+	5E^DCB^	Pop And Push
)	+*(+	5E^DCB^A	Print
End	+*	5E^DCB^A+	Pop Until '('
End	Empty	5E^DCB^A+*+	Pop Every element

**ALGORITHM:**

Step 1: Start.

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

Step 3: convert the infix expression to postfix expression.

Step 4: Stop

**PROGRAM CODE:**

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

int F(char symbol)
{
    switch(symbol)
    {
        case '+':
        case '-':
            return 2;
        case '*':
        case '/':
            return 4;
        case '^':
        case '$': return 5;
        case '(':          return 0;
        case '#': return -1;
        default: return 8;
    }
}

int G(char symbol)
{
    switch(symbol)
    {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 3;
        case '^':
        case '$': return 6;
        case '(':          return 9;
        case ')':          return 0;

        default: return 7;
    }
}
```

```
void infix_postfix(char infix[], char postfix[])
{
    int top, j, i;
    char s[30], symbol;
    top = -1;
    s[++top] = '#';
    j = 0;
    for(i=0; i < strlen(infix); i++)
    {
        symbol = infix[i];
        while(F(s[top]) > G(symbol))
        {
            postfix[j] = s[top--];
            j++;
        }
        if(F(s[top]) != G(symbol))
            s[++top] = symbol;
        else
            top--;
    }
    while(s[top] != '#')
    {
        postfix[j++] = s[top--];
    }
    postfix[j] = '\0';
}

void main()
{
    char infix[20], postfix[20];
    clrscr();
    printf("\nEnter a valid infix expression\n");
    fflush();
    gets(infix);
    infix_postfix(infix,postfix);
    printf("\nThe infix expression is:\n");
    printf ("%s",infix);
    printf("\nThe postfix expression is:\n");
    printf ("%s",postfix);
    getch();
}
```

**SAMPLE OUTPUT:**

Enter a valid infix expression

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

The infix expression is:

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

The postfix expression is:

abc-d\*+



## EXPERIMENT - 05

Design, Develop and Implement 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.

### ABOUT THE EXPERIMENT:

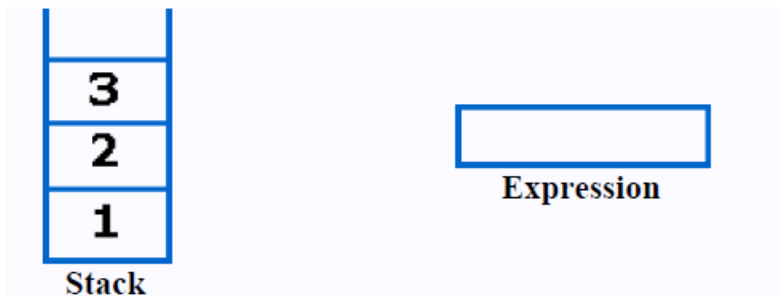
#### a. Evaluation of Suffix expression with single digit operands and operators: +, -, \*, /, %, ^

**Postfix/Suffix Expression:** Operators are written after their operands. Ex: XY+

In normal algebra we use the infix notation like  $a+b*c$ . The corresponding postfix notation is  $abc*+$

**Example:**      **Postfix String:** 123\*+4-

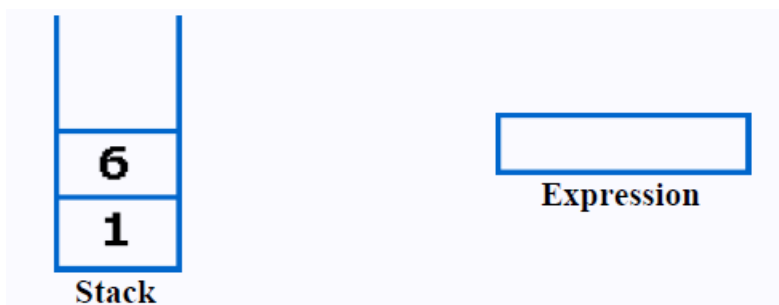
Initially the Stack is empty. Now, the first three characters scanned are 1,2 and 3, which are operands. Thus they will be pushed into the stack in that order.



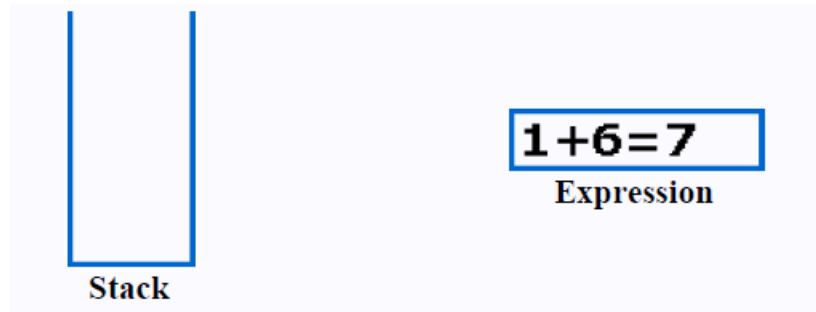
Next character scanned is "\*", which is an operator. Thus, we pop the top two elements from the stack and perform the "\*" operation with the two operands. The second operand will be the first element that is popped.



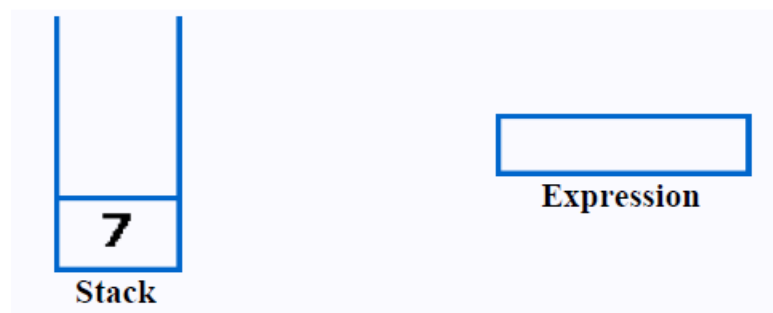
The value of the expression(2\*3) that has been evaluated(6) is pushed into the stack.



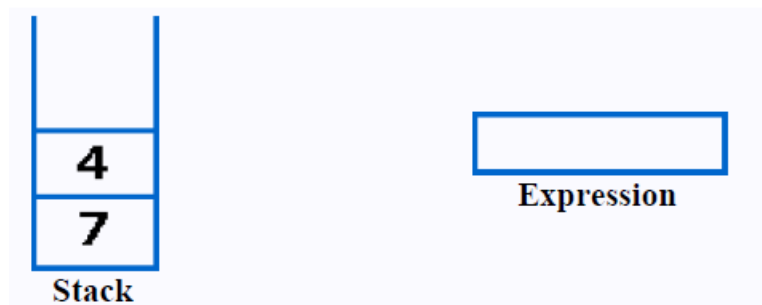
Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.



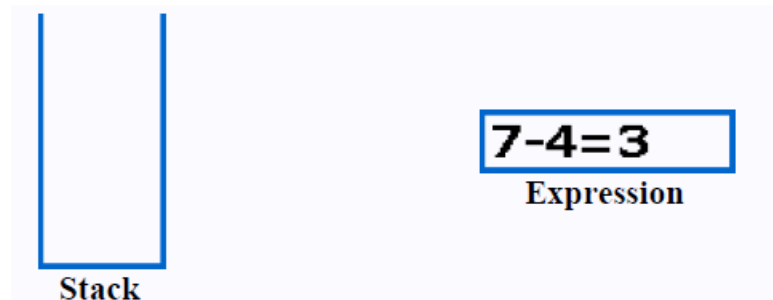
The value of the expression(1+6) that has been evaluated(7) is pushed into the stack.



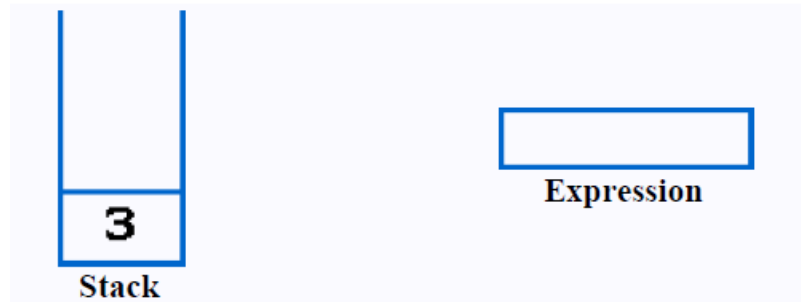
Next character scanned is "4", which is added to the stack.



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



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



Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned.

**End result:**

**Postfix String:** 123\*+4-

**Result:** 3

### ALGORITHM:

Step 1: Start.

Step 2: Read the postfix/suffix expression.

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

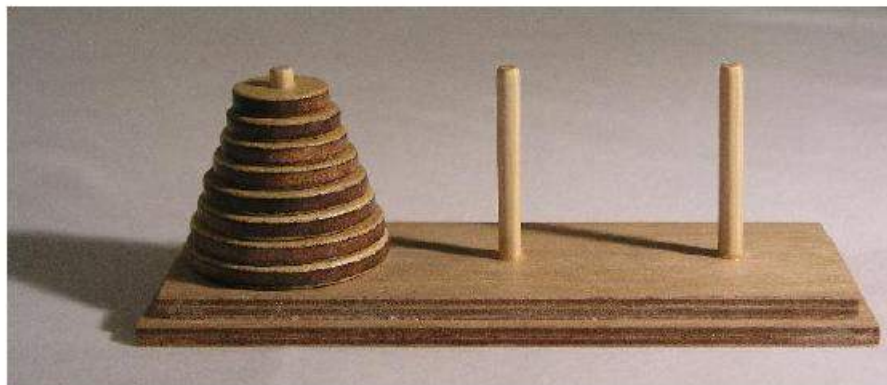
Step 4: Stop.

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

The **Tower of Hanoi** is a **mathematical game** or **puzzle**. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a **conical** shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is  $2^n - 1$ , where  $n$  is the number of disks.



**ALGORITHM:**

Step 1: Start.

Step 2: Read N number of discs.

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

Step 4: Stop.

**PROGRAM CODE:****PROGRAM 5A:**

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

double compute(char symbol, double op1, double op2)
{
    switch(symbol)
    {
        case '+':    return op1 + op2;
        case '-':    return op1 - op2;
        case '*':    return op1 * op2;
        case '/':    return op1 / op2;
        case '$':
        case '^':    return pow(op1,op2);
        default:    return 0;
    }
}

void main()
{
    double s[20], res, op1, op2;
    int top, i;
    char postfix[20], symbol;
    clrscr();
    printf("\nEnter the postfix expression:\n");
    fflush();
    gets(postfix);
    top=-1;
    for(i=0; <strlen(postfix); i++)
    {
        symbol = postfix[i];
        if(isdigit(symbol))
            s[++top] = symbol - '0';
        else
        {
            op2 = s[top--];
            op1 = s[top--];
```

```

        res = compute(symbol, op1, op2);
        s[++top] = res;
    }
}
res = s[top--];
printf("\nThe result is : %f\n", res);
getch();
}

```

**SAMPLE OUTPUT:**

RUN1:

Enter the postfix expression:

23+

The result is: 5.000000

RUN2:

Enter the postfix expression:

23+7\*

The result is: 35.000000

**PROGRAM 5B:**

```

#include<stdio.h>
#include<conio.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;
    clrscr();
    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);
        getch();
    }
```

**SAMPLE 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"

## EXPERIMENT - 06

Design, Develop and Implement a menu driven Program in C for the following operations on **Circular QUEUE** of Characters (Array Implementation of Queue with maximum size **MAX**)

- a. Insert an Element on to Circular QUEUE
- b. Delete an Element from Circular QUEUE
- c. Demonstrate *Overflow* and *Underflow* situations on Circular QUEUE
- d. Display the status of Circular QUEUE
- e. Exit

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

### ALGORITHM:

Step 1: Start.

Step 2: Initialize queue size to MAX.

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

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

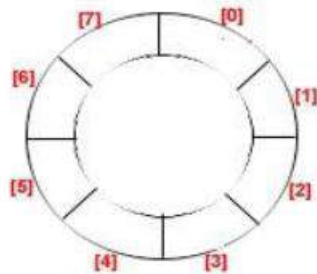
Step 5: Display the contents of the queue.

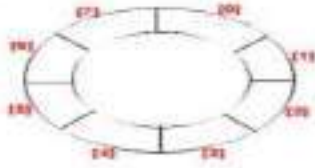
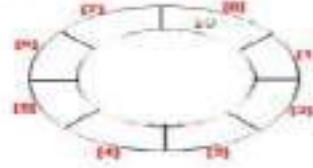
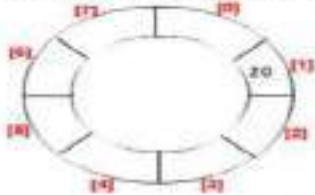
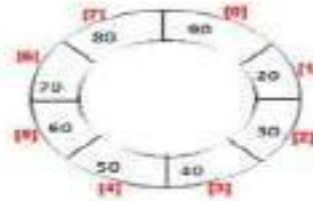
Step 6: Stop.

### ABOUT THE EXPERIMENT:

**Circular queue** is a linear data structure. It follows FIFO principle. In **circular queue** the last node is connected back to the first node to make a **circle**. **Circular** linked list follow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the **queue**. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

A circular queue looks like



**Consider the example with Circular Queue implementation:**1) Initially: **front = 0** and **rear = -1**.2) Add item 10 then **front = 0** and **rear = 0**.3) Now delete one item then **front = 1** and **rear = 1**. 4) Like this now insert 30, 40, and 50, 50, 70, 80 respectability then **front = 1** and **rear = 7**.5) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front = 0** and **rear = 0**.**PROGRAM CODE:**

```

#include<stdio.h>
#include<conio.h>
#define MAX 4

int ch, front = 0, rear = -1, count=0;
char q[MAX], item;

void insert()
{
    if(count == MAX)
        printf("\nQueue is Full");
    else {
        rear = (rear + 1) % MAX;
        q[rear]=item;
        count++;
    }
}

void del()
{
    if(count == 0)
        printf("\nQueue is Empty");
    else

```



```
{
    if(front > rear && rear==MAX-1)
    {
        front=0;    rear=-1;    count=0;
    }

    else
{
        item=q[front];
        printf("\nDeleted item is: %c",item);
        front = (front + 1) % MAX;
        count--;
    }
}

void display()
{
    int i, f=front, r=rear;
    if(count == 0)
        printf("\nQueue is Empty");
    else
    {
        printf("\nContents of Queue is:\n");
        for(i=f; i<=r; i++)
        {
            printf("%c\t", q[i]);
            f = (f + 1) % MAX;
        }
    }
}

void main()
{
    clrscr();
    do
    {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");
        printf("\nEnter the choice: ");
        scanf("%d", &ch);
        fflush();
        switch(ch)
        {
            case 1: printf("\nEnter the character / item to be inserted: ");
                    scanf("%c", &item);
                    insert();
                    break;
```

```

        case 2: del();
            break;
        case 3: display();
            break;
        case 4: exit(0);
            break;
    }
}while(ch!=4);
getch();
}

```

**SAMPLE POUTPUT:**

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 1

Enter the character / item to be inserted:      A

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 1

Enter the character / item to be inserted:      B

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 1

Enter the character / item to be inserted:      C

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 1

Enter the character / item to be inserted:      D

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 3

**Contents of Queue is:**

**A      B      C      D**

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 1

Enter the character / item to be inserted:      F

**Queue is Full**

1. Insert      2. Delete      3. Display      4. Exit

Enter the choice: 2

Deleted item is:      **A**

1. Insert      2. Delete      3. Display      4. Exit  
Enter the choice: 2

Deleted item is:      **B**

1. Insert      2. Delete      3. Display      4. Exit  
Enter the choice: 3

**Contents of Queue is:**  
**C      D**

1. Insert      2. Delete      3. Display      4. Exit  
Enter the choice: 1  
Enter the character / item to be inserted:      **K**

1. Insert      2. Delete      3. Display      4. Exit  
Enter the choice: 3

**Contents of Queue is:**  
**C      D      K**

1. Insert      2. Delete      3. Display      4. Exit  
Enter the choice: 4

## EXPERIMENT - 07

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

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

### ABOUT THE EXPERIMENT:

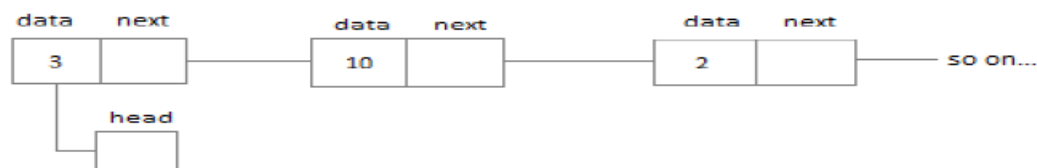
Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.



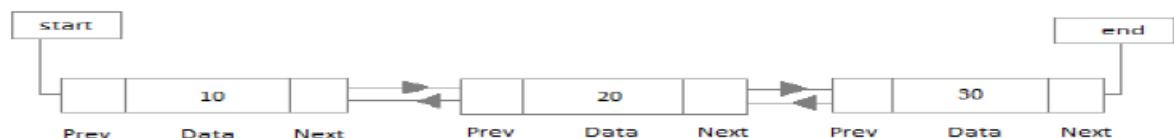
- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

### Types of Linked List:

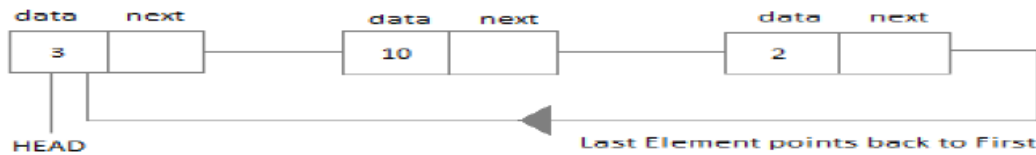
**Singly Linked List:** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



**Doubly Linked List:** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.



- **Circular Linked List:** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



### ALGORITHM:

Step 1: Start.

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

Step 2: Create a singly linked list. (SLL)

Step 3: Display the status of SLL.

Step 4: Count the number of nodes.

Step 5: Perform insertion at front of list.

Step 6: Perform deletion at the front of the list.

Step 7: Perform insertion at end of the list.

Step 8: Perform deletion at the end of the list.

Step 9: Demonstrate how singly linked list can be used as stack.

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

Step 11: Stop.

### PROGRAM CODE:

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

int MAX=4, count;

struct student
{
    char usn[10];
    char name[30];
    char branch[5];
    int sem;
    char phno[10];
    struct student *next;        //Self referential pointer.
};

typedef struct student NODE;

int countnodes(NODE *head)
{
    NODE *p;
    count=0;
```

```

    p=head;
    while(p!=NULL)
    {
        p=p->next;
        count++;
    }
    return count;
}

```

```

NODE* getnode(NODE *head)
{
    NODE *newnode;
    newnode=(NODE*)malloc(sizeof(NODE));           //Create first NODE
    printf("\nEnter USN, Name, Branch, Sem, Ph.No\n");
    fflush();
    gets(newnode->usn);
    fflush();
    gets(newnode->name);
    fflush();
    gets(newnode->branch);
    scanf("%d",&(newnode->sem));
    fflush();
    gets(newnode->phno);
    newnode->next=NULL;           //Set next to NULL...
    head=newnode;
    return head;
}

```

```

NODE* display(NODE *head)
{
    NODE *p;
    if(head == NULL)
        printf("\nNo student data\n");
    else
    {
        p=head;
        printf("\n----STUDENT DATA----\n");
        printf("\nUSN\tNAME\tBRANCH\tSEM\tPh.NO.");
        while(p!=NULL)
        {
            printf("\n%s\t%s\t%s\t%d\t%s", p->usn, p->name, p->branch, p->sem, p->phno);
            p = p->next;           //Go to next node...
        }
        printf("\nThe no. of nodes in list is: %d",countnodes(head));
    }
}

```

```
    return head;
}
```

```
NODE* create(NODE *head)
{
    NODE *newnode;
    if(head==NULL)
    {
        newnode=getnode(head);
        head=newnode;
    }
    else
    {
        newnode=getnode(head);
        newnode->next=head;
        head=newnode;
    }
    return head;
}
```

```
NODE* insert_front(NODE *head)
{
    if(countnodes(head)==MAX)
        printf("\nList is Full / Overflow!!");
    else
        head=create(head); //create()insert nodes at front.
    return head;
}
```

```
NODE* insert_rear(NODE *head)
{
    NODE *p, *newnode;
    p=head;
    if(countnodes(head) == MAX)
        printf("\nList is Full(Queue)!!");
    else
    {
        if(head == NULL)
        {
            newnode=getnode(head);
            head=newnode; //set first node to be head
        }
        else
        {
```

```

        newnode=getnode(head);
        while(p->next!=NULL)
        {
            p=p->next;
        }
        p->next=newnode;
    }
}
return head;
}

NODE* insert(NODE *head)
{
    int ch;
    do
    {
        printf("\n1.Insert at Front(First)\t2.Insert at End(Rear/Last)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: head=insert_front(head); break;
            case 2: head=insert_rear(head); break;
            case 3: break;
        }
        head=display(head);
    }while(ch!=3);
    return head;
}

NODE* delete_front(NODE *head)
{
    NODE *p;
    if(head==NULL)
        printf("\nList is Empty/Underflow (STACK/QUEUE)");
    else
    {
        p=head;
        head=head->next;    //Set 2nd NODE as head
        free(p);
        printf("\nFront(first)node is deleted");
    }
    return head;
}

```



```
NODE* delete_rear(NODE *head)
{
    NODE *p, *q;
    p=head;
    while(p->next!=NULL)
    {
        p=p->next; //Go upto -1 NODE which you want to delete
    }
    q=p->next;
    free(q);//Delete last NODE...
    p->next=NULL;
    printf("\nLast(end) entry is deleted");
    return head;
}

NODE* del(NODE *head)
{
    int ch;
    do{
        printf("\n1.Delete from Front(First)\t2. Delete from End(Rear/Last)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: head=delete_front(head);
                    break;
            case 2: head=delete_rear(head);
                    break;
            case 3: break;
        }
        head=display(head);
    }while(ch!=3);
    return head;
}

NODE* stack(NODE *head)
{
    int ch;
    do
    {
        printf("\nSSL used as Stack...");
        printf("\n 1.Insert at Front(PUSH) \t 2.Delete from Front(POP)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
```

```

        case 1: head=insert_front(head); break;
        case 2: head=delete_front(head); break;
        case 3: break;
    }
    head=display(head);
}while(ch!=3);
return head;
}

```

```

NODE* queue(NODE *head)
{
    int ch;
    do
    {
        printf("\nSSL used as Queue...");
        printf("\n 1.Insert at Rear(INSERT) \t 2.Delete from Front(DELETE))\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: head=insert_rear(head); break;
            case 2: head=delete_front(head); break;
            case 3: break;
        }
        head=display(head);
    }while(ch!=3);
    return head;
}

```

```

void main()
{
    int ch, i, n;
    NODE *head;
    head=NULL;
    clrscr();
    printf("\n*-----Studednt Database-----*");
    do
    {
        printf("\n 1.Create\t 2.Display\t 3.Insert\t 4.Delete\t 5.Stack\t 6.Queue\t 7.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: printf("\nHow many student data you want to create: ");
                    scanf("%d", &n);

```

```

        for(i=0;i<n;i++)
            head=create(head);//Call to Create NODE...
        break;
    case 2: head=display(head); //Call to Display...
        break;
    case 3: head=insert(head); //Call to Insert...
        break;
    case 4: head=del(head); //Call to delete
        break;
    case 5: head=stack(head);
        break;
    case 6: head=queue(head);
        break;
    case 7: exit(); //Exit...
    }
}while(ch!=7);
}

```

**SAMPLE OUTPUT:**

1. Create 2. Display 3. Insert 4. Delete 5. Stack 6.Queue 7. Exit

Enter your choice: 1

How many student data you want to create: 2

Enter USN, Name, Branch, Sem, Ph.No

1kt12cs001 kumar cs 3 9900099000

Enter USN, Name, Branch, Sem, Ph.No

1kt12is002 ravi is 3 9900099111

1. Create 2. Display 3. Insert 4. Delete 5. Stack 6.Queue 7. Exit

Enter your choice: 2

----STUDENT DATA----

USN	NAME	BRANCH	SEM	Ph.NO.
1kt12is002	ravi	is	3	9900099111
1kt12cs001	kumar	cs	3	9900099000

The no. of nodes in list is: 2

1. Create 2. Display 3. Insert 4. Delete 5. Stack 6.Queue 7. Exit

Enter your choice: 3

1.Insert at Front(First) 2.Insert at End(Rear/Last) 3.Exit

Enter your choice: 1

Enter USN, Name, Branch, Sem, Ph.No

1kt12cs003 suresh cs 3 99000992222

----STUDENT DATA----

USN	NAME	BRANCH	SEM	Ph.NO.
-----	------	--------	-----	--------

1kt12cs003	suresh	cs	3	99000992222
------------	--------	----	---	-------------

1kt12is002	ravi	is	3	9900099111
------------	------	----	---	------------

1kt12cs001	kumar	cs	3	9900099000
------------	-------	----	---	------------

The no. of nodes in list is: 3

1.Insert at Front(First) 2.Insert at End(Rear/Last) 3.Exit

Enter your choice: 2

Enter USN, Name, Branch, Sem, Ph.No

1kt12is004 naresh is 3 99000993333

----STUDENT DATA----

USN	NAME	BRANCH	SEM	Ph.NO.
-----	------	--------	-----	--------

1kt12cs003	suresh	cs	3	99000992222
------------	--------	----	---	-------------

1kt12is002	ravi	is	3	9900099111
------------	------	----	---	------------

1kt12cs001	kumar	cs	3	9900099000
------------	-------	----	---	------------

1kt12is004	naresh	is	3	99000993333
------------	--------	----	---	-------------

The no. of nodes in list is: 4

1.Insert at Front(First) 2.Insert at End(Rear/Last) 3.Exit

Enter your choice: 3

1. Create 2. Display 3. Insert 4. Delete 5. Stack 6.Queue 7. Exit

Enter your choice: 4

1. Delete from Front (First) 2. Delete from End (Rear/Last) 3.Exit

Enter your choice: 1

**Front (first) node is deleted**

----STUDENT DATA----

USN	NAME	BRANCH	SEM	Ph.NO.
-----	------	--------	-----	--------

1kt12is002	ravi	is	3	9900099111
------------	------	----	---	------------

1kt12cs001	kumar	cs	3	9900099000
------------	-------	----	---	------------

1kt12is004	naresh	is	3	99000993333
------------	--------	----	---	-------------

The no. of nodes in list is: 3

1. Delete from Front (First) 2. Delete from End (Rear/Last) 3.Exit

Enter your choice: 2

**Last (end) node is deleted**

----STUDENT DATA----

USN	NAME	BRANCH	SEM	Ph.NO.
1kt12is002	ravi	is	3	9900099111
1kt12cs001	kumar	cs	3	9900099000

The no. of nodes in list is: 2

1. Delete from Front (First) 2. Delete from End (Rear/Last) 3.Exit

Enter your choice: 3

1. Create 2. Display 3. Insert 4. Delete 5. Stack 6.Queue 7. Exit

Enter your choice: 5

SSL used as Stack...

1.Insert at Front(PUSH) 2.Delete from Front(POP)) 3.Exit

Enter your choice: 1

Enter USN, Name, Branch, Sem, Ph.No

1kt12is010 vikky is 3 9900011111

----STUDENT DATA----

USN	NAME	BRANCH	SEM	Ph.NO.
1kt12is010	vikky	is	3	9900011111
1kt12is002	ravi	is	3	9900099111
1kt12cs001	kumar	cs	3	9900099000

1. Create 2. Display 3. Insert 4. Delete 5. Stack 6.Queue 7. Exit

Enter your choice: 7



## EXPERIMENT - 08

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

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

### ABOUT THE EXPERIMENT:

**Doubly Linked List:** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.

In computer science, a **doubly linked list** is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called *links*, that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.

A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.

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

### ALGORITHM:

- Step 1: Start.
- Step 2: Read the value of N. (N student's information)
- Step 3: Create a doubly linked list. (DLL)
- Step 4: Display the status of DLL.
- Step 5: Count the number of nodes.
- Step 6: Perform insertion at front of list.
- Step 7: Perform deletion at the front of the list.
- Step 8: Perform insertion at end of the list.
- Step 9: Perform deletion at the end of the list.

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

Step 11: Stop.

**PROGRAM CODE:**

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

int MAX=4, count;

struct emp
{
    int ssn;
    char name[20];
    char dept[10];
    char desig[15];
    int sal;
    char phno[10];
    struct emp *left;
    struct emp *right;
};

typedef struct emp NODE;

int countnodes(NODE *head)
{
    NODE *p;
    count=0;
    p=head;
    while(p!=NULL)
    {
        p=p->right;
        count++;
    }
    return count;
}

NODE* getnode(NODE *head)
{
    NODE *newnode;
    newnode=(NODE*)malloc(sizeof(NODE));
    newnode->right=newnode->left=NULL;
    printf("\nEnter SSN, Name, Dept, Designation, Sal, Ph.No\n");
    scanf("%d",&newnode->ssn);
    fflush();
    gets(newnode->name);
    fflush();
}
```



```

    gets(newnode->dept);
    flushall();
    gets(newnode->desig);
    scanf("%d",&newnode->sal);
    flushall();
    gets(newnode->phno);
    head=newnode;
    return head;
}

```

```

NODE* display(NODE *head)
{
    NODE *p;
    if(head==NULL)
        printf("\nNo Employee data\n");
    else
    {
        p=head;
        printf("\n---EMPLOYEE DATA---\n");
        printf("\nSSN\tNAME\tDEPT\tDESINGATION\tSAL\tPh.NO.");
        while(p!=NULL)
        {
            printf("\n%d\t%s\t%s\t%s\t%d\t%s", p->ssn, p->name, p->dept, p->desig,
p->sal, p->phno);
            p = p->right;        //Go to next node...
        }
        printf("\nThe no. of nodes in list is: %d",countnodes(head));
    }
    return head;
}

```

```

NODE* create(NODE *head)// creating & inserting at end.
{
    NODE *p, *newnode;
    p=head;
    if(head==NULL)
    {
        newnode=getnode(head);
        head=newnode;
    }
    else
    {
        newnode=getnode(head);
        while(p->right!=NULL)
        {

```

```
        p=p->right;
    }
    p->right=newnode;
    newnode->left=p;
}
return head;
}
```

```
NODE* insert_end(NODE *head)
{
    if(countnodes(head)==MAX)
        printf("\nList is Full!!");
    else
        head=create(head);
    return head;
}
```

```
NODE* insert_front(NODE *head)
{
    NODE *p, *newnode;
    if(countnodes(head)==MAX)
        printf("\nList is Full!!");
    else
    {
        if(head==NULL)
        {
            newnode=getnode(head);
            head=newnode;    //set first node to be head
        }
        else
        {
            newnode=getnode(head);
            newnode->right=head;
            head->left=newnode;
            head=newnode;
        }
    }
    return head;
}
```

```
NODE* insert(NODE *head)
{
    int ch;
    do
    {
        printf("\n 1.Insert at Front(First) \t 2.Insert at End(Rear/Last)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: head=insert_front(head); break;
            case 2: head=insert_end(head); break;
            case 3: break;
        }
        head=display(head);
    }while(ch!=3);
    return head;
}
```

```
NODE* delete_front(NODE *head)
{
    NODE *p;
    if(head==NULL)
        printf("\nList is Empty (QUEUE)");
    else
    {
        p=head;
        head=head->right;
        head->right->left=NULL;
        free(p);
        printf("\nFront(first)node is deleted");
    }
    return head;
}
```

```
NODE* delete_end(NODE *head)
{
    NODE *p, *q;
    p=head;
    while(p->right!=NULL)
    {
        p=p->right; //Go upto -1 node which you want to delete
    }
    q=p->left;
```

```

    q->right=NULL;
    p->left=NULL;
    free(p);//Delete last node...
    printf("\nLast(end) entry is deleted");
    return head;
}

```

```

NODE *del(NODE *head)

```

```

{
    int ch;
    do {
        printf("\n1.Delete from Front(First)\t2. Delete from End(Rear/Last)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: head=delete_front(head);
                    break;
            case 2: head=delete_end(head);
                    break;
            case 3: break;
        }
        head=display(head);
    }while(ch!=3);
    return head;
}

```

```

NODE* queue(NODE *head)

```

```

{
    int ch, ch1, ch2;
    do
    {
        printf("\nDLL used as Double Ended Queue");
        printf("\n1.QUEUE- Insert at Rear & Delete from Front");
        printf("\n2.QUEUE- Insert at Front & Delete from Rear");
        printf("\n3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: do{
                printf("\n1.Insert at Rear\t2.Delete from From Front\t3.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch1);
                switch(ch1)

```

```

        {
            case 1: head=insert_end(head); break;
            case 2: head=delete_front(head); break;
            case 3: break;
        }
    }while(ch1!=3);
    break;
case 2: do{
    printf("\n1.Insert at Front\t2.Delete from Rear\t3.Exit");
    printf("\nEnter your choice: ");
    scanf("%d", &ch2);
    switch(ch2)
    {
        case 1: head=insert_front(head); break;
        case 2: head=delete_end(head); break;
        case 3: break;
    }
    }while(ch2!=3);
    break;
case 3: break;
    }
}while(ch!=3);
head=display(head);
return head;
}

void main()
{
    int ch, i, n;
    NODE *head;
    head=NULL;
    clrscr();
    printf("\n-----Employee Database-----");
    do
    {
        printf("\n1.Create\t2.Display\t3.Insert\t4.Delete\t5.Queue\t6.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: printf("\nHow many employees data you want to create: ");
                    scanf("%d", &n);
                    for(i=0;i<n;i++)
                        head=create(head);//Call to Create node...
                    break;
            case 2: head=display(head); //Call to Display...
                    break;
        }
    }
}

```

```

        case 3: head=insert(head); //Call to Insert...
                break;
        case 4: head=del(head); //Call to delete
                break;
        case 5: head=queue(head);
                break;
        case 6: exit(0); //Exit...
                break;
    }
}while(ch!=6);
}

```

**SAMPLE OUTPUT:**

-----Employee Database-----

1. Create    2. Display    3. Insert    4. Delete    5. Queue    7. Exit  
 Enter your choice: 1

How many employees data you want to create: 2

Enter SSN, Name, Dept, Designation, Sal, Ph.No

1	KUMAR	CSE	INSTRUCTOR	8000	900099000
---	-------	-----	------------	------	-----------

Enter SSN, Name, Dept, Designation, Sal, Ph.No

2	RAVI	ISE	INSTRUCTOR	9000	900099111
---	------	-----	------------	------	-----------

1. Create    2. Display    3. Insert    4. Delete    5. Queue    7. Exit  
 Enter your choice: 2

----EMPLOYEE DATA----

SSN	NAME	DEPT	DESINGATION	SAL	Ph.NO.
1	KUMAR	CSE	INSTRUCTOR	8000	900099000
2	RAVI	ISE	INSTRUCTOR	9000	900099111

The no. of nodes in list is: 2

1. Create    2. Display    3. Insert    4. Delete    5. Queue    7. Exit  
 Enter your choice: 3

1. Insert at Front (First)                    2.Insert at End (Rear/Last)    3.Exit  
 Enter your choice: 1

Enter SSN, Name, Dept, Designation, Sal, Ph.No

3	JIM	CSE	ATTENDER	6000	900099333
---	-----	-----	----------	------	-----------

----EMPLOYEE DATA----

SSN	NAME	DEPT	DESINGATION	SAL	Ph.NO.
3	JIM	CSE	ATTENDER	6000	900099333
1	KUMAR	CSE	INSTRUCTOR	8000	900099000
2	JIM	CSE	INSTRUCTOR	9000	900099111

The no. of nodes in list is: 3

1. Insert at Front (First)                      2.Insert at End (Rear/Last)    3.Exit  
Enter your choice: 3

1. Create      2. Display      3. Insert      4. Delete      5. Queue      7. Exit  
Enter your choice: 5

DLL used as Double Ended Queue

1.QUEUE- Insert at Rear & Delete from Front  
2.QUEUE- Insert at Front & Delete from Rear  
3.Exit

Enter your choice: 3

1. Create      2. Display      3. Insert      4. Delete      5. Queue      7. Exit  
Enter your choice: 7

## EXPERIMENT - 09

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

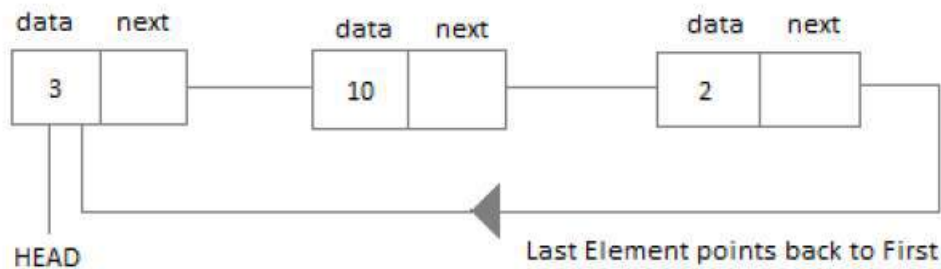
- a. Represent and Evaluate a Polynomial  $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$
- b. Find the sum of two polynomials  $POLY1(x,y,z)$  and  $POLY2(x,y,z)$  and store the result in  $POLYSUM(x,y,z)$

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

### ABOUT THE EXPERIMENT:

#### Circular Linked List:

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



#### Polynomial:

A **polynomial equation** is an **equation** that can be written in the form.  $ax^n + bx^{n-1} + \dots + rx + s = 0$ , where  $a, b, \dots, r$  and  $s$  are constants. We call the largest exponent of  $x$  appearing in a non-zero term of a **polynomial** the degree of that **polynomial**.

As with **polynomials** with one variable, you must pay attention to the rules of exponents and the order of operations so that you correctly **evaluate** an expression with two or more variables. **Evaluate**  $x^2 + 3y^3$  for  $x = 7$  and  $y = -2$ . Substitute the given values for  $x$  and  $y$ . **Evaluate**  $4x^2y - 2xy^2 + x - 7$  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.

$$\begin{array}{c}
 \text{coefficients} \\
 \downarrow \quad \downarrow \\
 4x^2 + 3x - 7 \\
 \uparrow \\
 \text{Leading} \\
 \text{coefficient}
 \end{array}$$

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**

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

© mathwarehouse.com

**ALGORITHM:**

Step 1: Start.

Step 2: Read a polynomial.

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

Step 3: Evaluate the given polynomial

Step 4: Read two polynomials and find the sum of the polynomials.

Step 5: Stop

**PROGRAM CODE:**

```
#include<stdio.h>
#include<alloc.h>
#include<math.h>

struct node
{
    int cf, px, py, pz; int flag;
    struct node *link;
};
typedef struct node NODE;

NODE* getnode()
{
    NODE *x;
    x=(NODE*)malloc(sizeof(NODE));
    if(x==NULL)
    {
        printf("Insufficient memory\n"); exit(0);
    }
    return x;
}

void display(NODE *head)
{
    NODE *temp;
    if(head->link==head)
    {
        printf("Polynomial does not exist\n"); return;
    }
    temp=head->link;
    printf("\n");
```

```

while(temp!=head)
{
    printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);
    if(temp->link != head)
        printf(" + ");
    temp=temp->link;
}
printf("\n");
}

```

```

NODE* insert_rear(int cf,int x,int y,int z,NODE *head)
{
    NODE *temp,*cur;
    temp=getnode();
    temp->cf=cf;
    temp->px=x;
    temp->py=y;
    temp->pz=z;
    cur=head->link;
    while(cur->link!=head)
    {
        cur=cur->link;
    }
    cur->link=temp;
    temp->link=head;
    return head;
}

```

```

NODE* read_poly(NODE *head)
{
    int px, py, pz, cf, ch;
    printf("\nEnter coeff: ");
    scanf("%d",&cf);
    printf("\nEnter x, y, z powers(0-indicate NO term): ");
    scanf("%d%d%d", &px, &py, &pz);
    head=insert_rear(cf,px,py,pz,head);
    printf("\nIf you wish to continue press 1 otherwise 0: ");
    scanf("%d", &ch);
    while(ch != 0)
    {

```

```

        printf("\nEnter coeff: ");
        scanf("%d",&cf);
        printf("\nEnter x, y, z powers(0-indicate NO term): "); scanf("%d%d%d",
        &px, &py, &pz); head=insert_rear(cf,px,py,pz,head);
        printf("\nIf you wish to continue press 1 otherwise 0: "); scanf("%d", &ch);
    }
    return head;
}

```

```

NODE* add_poly(NODE *h1,NODE *h2,NODE *h3)
{
    NODE *p1,*p2;
    int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
    p1=h1->link;
    while(p1!=h1)
    {
        x1=p1->px;
        y1=p1->py;
        z1=p1->pz;
        cf1=p1->cf;
        p2=h2->link;
        while(p2!=h2)
        {
            x2=p2->px;
            y2=p2->py;
            z2=p2->pz;
            cf2=p2->cf;
            if(x1==x2 && y1==y2 && z1==z2)
                break;
            p2=p2->link;
        }
        if(p2!=h2)
        {
            cf=cf1+cf2;
            p2->flag=1;
            if(cf!=0)
                h3=insert_rear(cf,x1,y1,z1,h3);
        }
        else
            h3=insert_rear(cf1,x1,y1,z1,h3);
        p1=p1->link;
    }
}

```

```

    }
    p2=h2->link;
    while(p2!=h2)
    {
        if(p2->flag==0)
            h3=insert_rear(p2->cf,p2->px,p2->py,p2->pz,h3);
        p2=p2->link;
    }
    return h3;
}

void evaluate(NODE *h1)
{
    NODE *head; int
    x, y, z;
    float result=0.0;
    head=h1;
    printf("\nEnter x, y, z, terms to evaluate:\n");
    scanf("%d%d%d", &x, &y, &z);
    while(h1->link != head)
    {
        result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
        h1=h1->link;
    }
    result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
    printf("\nPolynomial result is: %f", result);
}

void main() {
    NODE *h1,*h2,*h3; int
    ch; clrscr();
    h1=getnode();
    h2=getnode();
    h3=getnode();
    h1->link=h1;
    h2->link=h2;
    h3->link=h3;
    while(1)
    {
        printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");

```

```

printf("Enter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
    printf("\nEnter polynomial to evaluate:\n");
    h1=read_poly(h1);
    display(h1);
    evaluate(h1);
    break;
case 2:
    printf("\nEnter the first polynomial:");
    h1=read_poly(h1);
    printf("\nEnter the second polynomial:");
    h2=read_poly(h2);
    h3=add_poly(h1,h2,h3);
    printf("\nFirst polynomial is: ");
    display(h1);
    printf("\nSecond polynomial is: ");
    display(h2);
    printf("\nThe sum of 2 polynomials is: ");
    display(h3);
    break;
case 3:
    exit(0); break;
default:
    printf("\nInvalid entry"); break;
} getch();
}
}

```

**SAMPLE OUTPUT:**

1. Evaluate polynomial  $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$
2. Add two polynomials
3. Exit

Enter your choice: 1

Enter polynomial to  
evaluate: Enter coeff:

6

Enter x, y, z powers (0-indiacate NO term:

2 2 1

If you wish to continue press 1 otherwise 0:

1 Enter coeff: -4

Enter x, y, z powers (0-indiacate NO term: 0

1 5

If you wish to continue press 1 otherwise 0:

1

Enter coeff: 3

Enter x, y, z powers (0-indiacate NO term: 3 1 1

If you wish to continue press 1

otherwise 0: 1

Enter coeff: 2

Enter x, y, z powers (0-indiacate NO term: 1 5 1

If you wish to continue press 1

otherwise 0: 1

Enter coeff: -2

Enter x, y, z powers (0-indiacate NO term: 1 1 3

If you wish to continue press 1 otherwise 0:0

**Polynomial is:**

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

Enter x, y, z, terms to

evaluate: 1 1 1

**Polynomial result is: 5.000000**

1. Evaluate polynomial  $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$
2. Add two polynomials
3. Exit

Enter your choice: 2

Enter 1st polynomial:

Enter coeff: 4

Enter x, y, z powers (0-indiacate NO term: 2 2 2

If you wish to continue press 1

otherwise 0: 1 Enter coeff: 3

Enter x, y, z powers (0-indiacate NO term: 1 1 2

If you wish to continue press 1 otherwise 0:0

Enter 2nd polynomial:

Enter coeff: 6

Enter x, y, z powers (0-indicate NO term:

2 2 2

If you wish to continue press 1 otherwise 0:

0

**Polynomial is:**

**1st Polynomial is:**

**$4x^2y^2z^2 + 3x^1y^1z^2$**

**2nd Polynomial**

**is:  $6x^2y^2$**

**$z^2$**

**Added polynomial:  $10x^2y^2z^2 + 3x^1y^1z^2$**

1. Evaluate polynomial  $P(x,y,z) = 6x^2y^2z^2 -$

$4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$  2. Add two polynomials

3. Exit

Enter your choice: 3



## EXPERIMENT - 10

Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers

- Create a BST of N Integers: **6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2**
- Traverse the BST in Inorder, Preorder and Post Order
- Search the BST for a given element (**KEY**) and report the appropriate message
- Delete an element (**ELEM**) from BST
- Exit

### ABOUT THE EXPERIMENT:

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

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

Thus, a binary search tree (BST) divides all its sub-trees into two segments; *left* sub-tree and *right* sub-tree and can be defined as

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

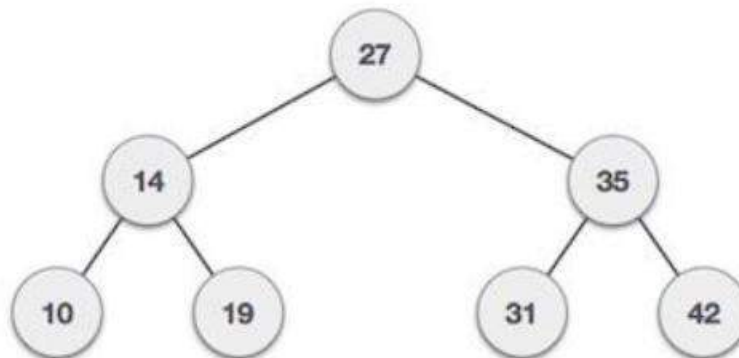


Fig: An example of BST

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

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

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

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

**ALGORITHM:**

Step 1: Start.  
Step 2: Create a Binary Search Tree for N elements.  
Step 3: Traverse the tree in inorder.  
Step 4: Traverse the tree in preorder  
Step 6: Traverse the tree in postorder.  
Step 7: Search the given key element in the BST.  
Step 8: Delete an element from BST.  
Step 9: Stop

**PROGRAM CODE:**

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

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

typedef struct BST NODE;

NODE *node;

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

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

```
NODE* search(NODE *node, int data)
{
    if(node == NULL)
        printf("\nElement not found");
    else if(data < node->data)
    {
        node->left=search(node->left, data);
    }
    else if(data > node->data)
    {
        node->right=search(node->right, data);
    }
    else
        printf("\nElement found is: %d", node->data);
    return node;
}
```

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

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

```

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

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

NODE* del(NODE *node, int data)
{
    NODE *temp;
    if(node == NULL)
    {
        printf("\nElement not found");
    }
    else if(data < node->data)
    {
        node->left = del(node->left, data);
    }
    else if(data > node->data)
    {
        node->right = del(node->right, data);
    }
    else
    { /* Now We can delete this node and replace with either minimum element in the right sub tree or maximum element in the left subtree */
        if(node->right && node->left)
        {
            /* Here we will replace with minimum element in the right sub tree */
            temp = findMin(node->right);
            node -> data = temp->data;
            /* As we replaced it with some other node, we have to delete that node */
            node -> right = del(node->right,temp->data);
        }
    }
}

```

```

    }
    else
    {
        /* If there is only one or zero children then we can directly remove it from the tree and connect its parent to its child */
        temp = node;
        if(node->left == NULL)
            node = node->right;
        else if(node->right == NULL)
            node = node->left;
        free(temp);          /* temp is longer required */
    }
}
return node;
}

void main()
{
    int data, ch, i, n;
    NODE *root=NULL;
    clrscr();
    while (1)
    {
        printf("\n1.Insertion in Binary Search Tree");
        printf("\n2.Search Element in Binary Search Tree");
        printf("\n3.Delete Element in Binary Search Tree");
        printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1: printf("\nEnter N value: ");
                    scanf("%d", &n);
                    printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                    for(i=0; i<n; i++)
                    {
                        scanf("%d", &data);
                        root=createtree(root, data);
                    }
                    break;
            case 2: printf("\nEnter the element to search: ");
                    scanf("%d", &data);
                    root=search(root, data);
                    break;
            case 3: printf("\nEnter the element to delete: ");
                    scanf("%d", &data);
                    root=del(root, data);
                    break;
        }
    }
}

```

```

        case 4: printf("\nInorder Traversal: \n");
                inorder(root);
                break;
        case 5: printf("\nPreorder Traversal: \n");
                preorder(root);
                break;
        case 6: printf("\nPostorder Traversal: \n");
                postorder(root);
                break;
        case 7: exit(0);
        default:printf("\nWrong option");
                break;
    }
}
getch();
}

```

**SAMPLE OUTPUT:**

1. Insertion in Binary Search Tree  
 2. Search Element in Binary Search Tree  
 3. Delete Element in Binary Search Tree  
 4. Inorder

5.

Preorder

6.

Postorder

7. Exit

Enter your choice: 1

Enter N value: 12

Enter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)

**6 9 5 2 8 15 24 14 7 8 5 2**

1. Insertion in Binary Search Tree  
 2. Search Element in Binary Search Tree  
 3. Delete Element in Binary Search Tree  
 4. Inorder

5.

Preorder

6.

Postorder

7. Exit

Enter your choice: 4

**Inorder Traversal:**

**2 5 6 7 8 9 14 15 24**

1. Insertion in Binary Search Tree

2. Search Element in Binary Search Tree 3. Delete Element in Binary Search Tree 4. Inorder  
5.  
Preorder  
6.  
Postorder  
7. Exit  
Enter your choice: 5

**Preorder Traversal:**

**6 5 2 9 8 7 15 14 24**

1. Insertion in Binary Search Tree  
2. Search Element in Binary Search Tree 3. Delete Element in Binary Search Tree 4. Inorder  
5.  
Preorder  
6.  
Postorder  
7. Exit  
Enter your choice: 6

**Postorder Traversal:**

**2 5 7 8 14 24 15 9 6**

1. Insertion in Binary Search Tree  
2. Search Element in Binary Search Tree 3. Delete Element in Binary Search Tree 4. Inorder  
5.  
Preorder  
6.  
Postorder  
7. Exit  
Enter your choice: 2

Enter the element to search: 24 **Element**

**found is: 24**

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

Postorder

7. Exit

Enter your choice: 2

Enter the element to  
search: 50

**Element not found**

1. Insertion in Binary Search Tree

2. Search Element in Binary

Search Tree 3. Delete Element in

Binary Search Tree 4. Inorder

5.

Preorder

6.

Postorder

7. Exit

Enter your choice: 3

Enter the element to  
search: 15

1. Insertion in Binary Search Tree

2. Search Element in Binary

Search Tree 3. Delete Element in

Binary Search Tree 4. Inorder

5.

Preorder

6.

Postorder

7. Exit

Enter your choice: 4

**Inorder Traversal:**

**2 5 6 7 8 9 14 24**

1. Insertion in Binary Search Tree

2. Search Element in Binary

Search Tree 3. Delete Element in

Binary Search Tree 4. Inorder

5.

Preorder

6.

Postorder

7. Exit

Enter your choice: 5

**Preorder Traversal:**

**6 5 2 9 8 7 24 14**



1. Insertion in Binary Search Tree
  2. Search Element in Binary Search Tree
  3. Delete Element in Binary Search Tree
  4. Inorder
  5. Preorder
  6. Postorder
  7. Exit
- Enter your choice: 7

## EXPERIMENT - 11

Design, Develop and Implement a Program in C for the following operations on **Graph(G)** of Cities

- Create a Graph of **N** cities using Adjacency Matrix.
- Print all the nodes **reachable** from a given starting node in a digraph using **BFS** method
- Check whether a given graph is **connected** or not using **DFS** method.

### ABOUT THE EXPERIMENT:

#### Adjacency Matrix

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

A graph  $G = (V, E)$  where  $v = \{0, 1, 2, \dots, n-1\}$  can be represented using two dimensional integer array of size  $n \times n$ .

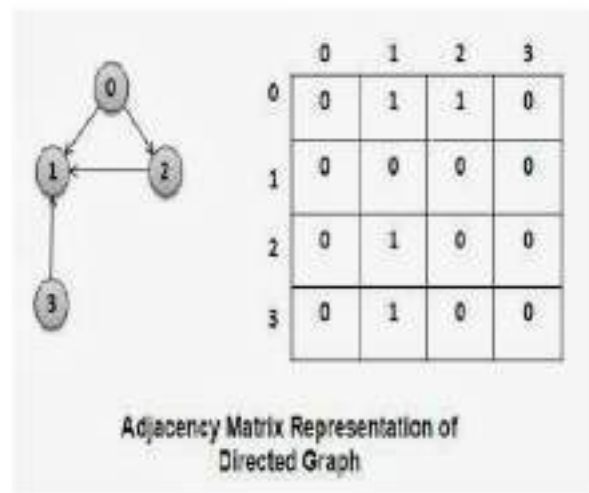
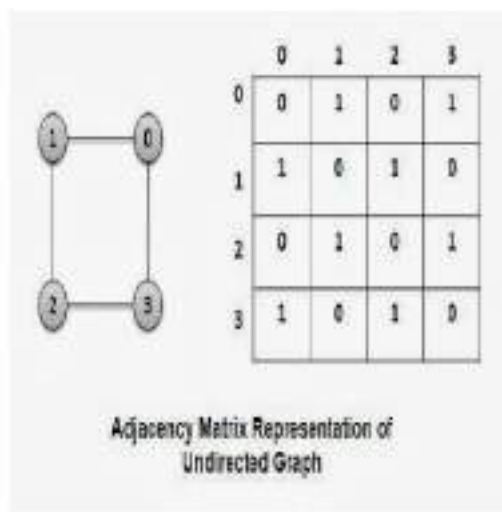
$a[20][20]$  can be used to store a graph with 20 vertices.

$a[i][j] = 1$ , indicates presence of edge between two vertices  $i$  and  $j$ .

$a[i][j] = 0$ , indicates absence of edge between two vertices  $i$  and  $j$ .

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

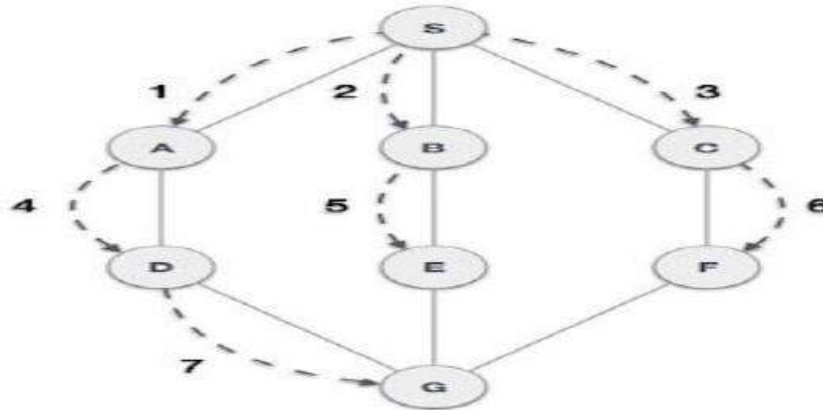
An example of adjacency matrix representation of an undirected and directed graph is given below:



**BFS**

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

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



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

- **Rule 1** – Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
- **Rule 2** – If no adjacent vertex found, remove the first vertex from queue.
- **Rule 3** – Repeat Rule 1 and Rule 2 until queue is empty.

**DFS**

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

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

***Algorithm***

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



white: vertex is unvisited;



gray: vertex is in progress;



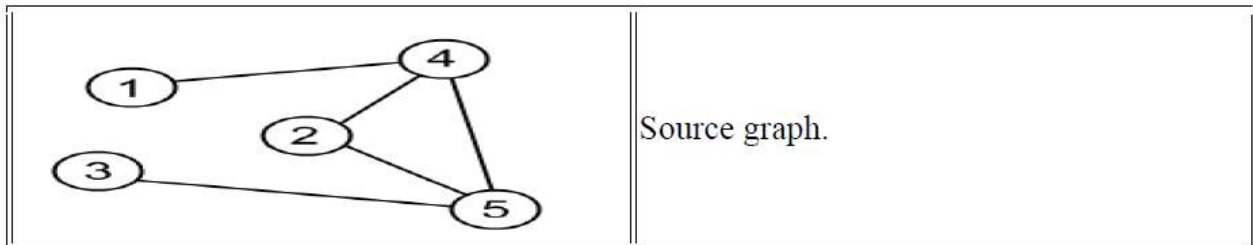
black: DFS has finished processing the vertex.




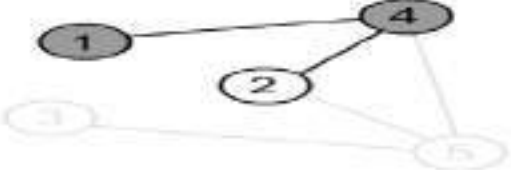
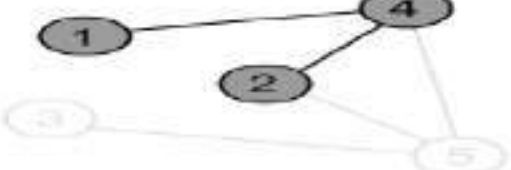
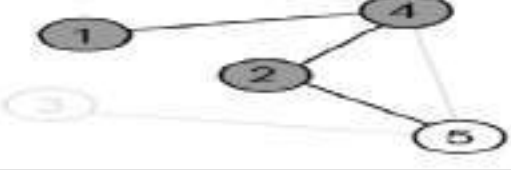
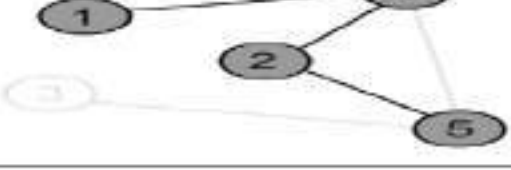

*NB.* For most algorithms Boolean classification *unvisited* / *visited* is quite enough, but we show general case here.

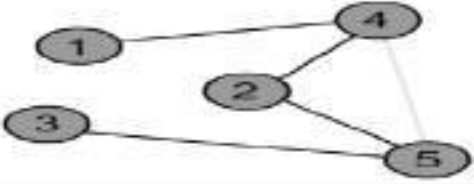
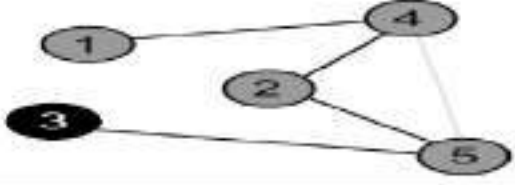
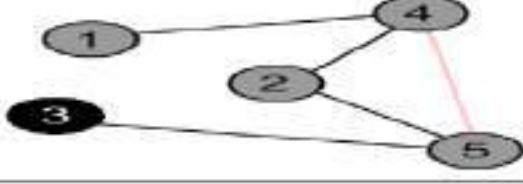
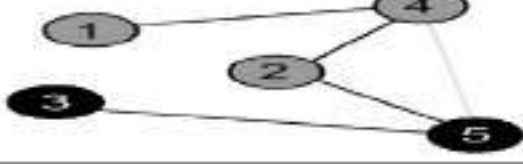
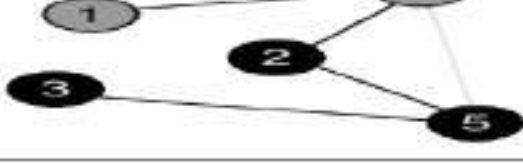
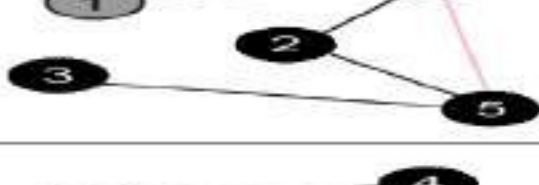


Initially all vertices are white (unvisited). DFS starts in arbitrary vertex and runs as follows:

- Mark vertex **u** as gray (visited).
- For each edge (**u**, **v**), where **u** is white, run depth-first search for **u** recursively.
- Mark vertex **u** as black and backtrack to the parent.

**Example.** Traverse a graph shown below, using DFS. Start from a vertex with number 1.



	Mark a vertex <b>1</b> as gray.
	There is an edge <b>(1, 4)</b> and a vertex <b>4</b> is unvisited. Go there.
	Mark the vertex <b>4</b> as gray.
	There is an edge <b>(4, 2)</b> and vertex a <b>2</b> is unvisited. Go there.
	Mark the vertex <b>2</b> as gray.
	There is an edge <b>(2, 5)</b> and a vertex <b>5</b> is unvisited. Go there.
	Mark the vertex <b>5</b> as gray.
	There is an edge <b>(5, 3)</b> and a vertex <b>3</b> is unvisited. Go there.

	Mark the vertex 3 as gray.
	There are no ways to go from the vertex 3. Mark it as black and backtrack to the vertex 5.
	There is an edge (5, 4), but the vertex 4 is gray.
	There are no ways to go from the vertex 5. Mark it as black and backtrack to the vertex 2.
	There are no more edges, adjacent to vertex 2. Mark it as black and backtrack to the vertex 4.
	There is an edge (4, 5), but the vertex 5 is black.
	There are no more edges, adjacent to the vertex 4. Mark it as black and backtrack to the vertex 1.
	There are no more edges, adjacent to the vertex 1. Mark it as black. DFS is over.

**ALGORITHM:**

Step 1: Start.

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

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

Step 3: Print the nodes reachable from the starting node using BFS.

Step 4: Check whether graph is connected or not using DFS.

Step 5: Stop.

**PROGRAM CODE:**

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

int a[10][10], n, m, i, j, source, s[10], b[10];
int visited[10];

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

void bfs()
{
    int q[10], u, front=0, rear=-1;
    printf("\nEnter the source vertex to find other nodes reachable or not: ");
    scanf("%d", &source);
    q[++rear] = source;
    visited[source] = 1;
    printf("\nThe reachable vertices are: ");
    while(front<=rear)
    {
        u = q[front++];
        for(i=1; i<=n; i++)
        {
            if(a[u][i] == 1 && visited[i] == 0)
            {
                q[++rear] = i;
                visited[i] = 1;
                printf("\n%d", i);
            }
        }
    }
}
```

```

    }
}

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

void main()
{
    int ch;
    clrscr();
    while(1)
    {
        printf("\n1.Create Graph\n2.BFS\n3.Check graph connected or not(DFS)\n4.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: create(); break;
            case 2: bfs();
                    for(i=1;i<=n;i++)
                        if(visited[i]==0)
                            printf("\nThe vertex that is not reachable %d", i);
                    break;
            case 3: printf("\nEnter the source vertex to find the connectivity: ");
                    scanf("%d",&source);
                    m=1;
                    dfs(source);
                    for(i=1;i<=n;i++) {
                        if(b[i]==0)
                            m=0;
                    }
                    if(m==1)
                        printf("\nGraph is Connected");
                    else

```



```

                printf("\nGraph is not Connected");
                break;
            default: exit(0);
        }
    }
}

```

**SAMPLE OUTPUT:**

1. Create Graph      2.BFS      3.Check graph connected or not (DFS)      4.Exit  
Enter your choice: 1

Enter the number of vertices of the digraph: 4

Enter the adjacency matrix of the graph:

```

0    0    1    1
0    0    0    0
0    1    0    0
0    1    0    0

```

1. Create Graph      2.BFS      3.Check graph connected or not (DFS)      4.Exit  
Enter your choice: 2

Enter the source vertex to find other nodes reachable or not: 1

```

3
4
2

```

1. Create Graph      2.BFS      3.Check graph connected or not (DFS)      4.Exit  
Enter your choice: 3

Enter the source vertex to find the connectivity: 1

```

1 -> 3
3 -> 2
1 -> 4

```

**Graph is Connected**

1. Create Graph      2.BFS      3.Check graph connected or not (DFS)      4.Exit  
Enter your choice: 4

## EXPERIMENT - 12

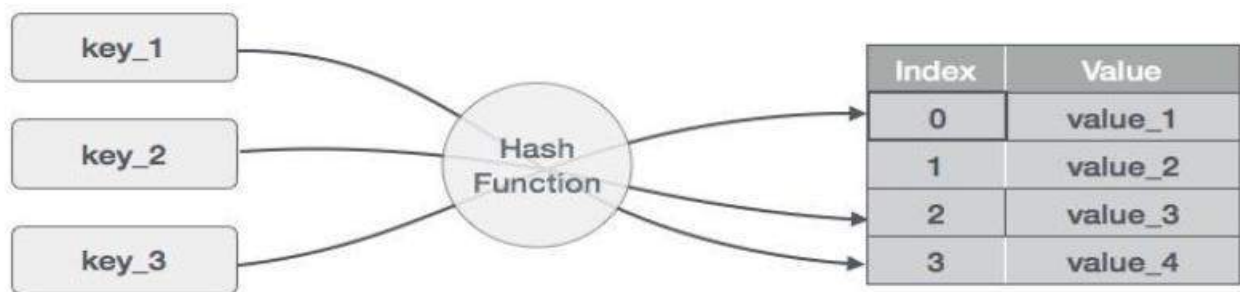
Given a File of **N** employee records with a set **K** of Keys(4-digit) which uniquely determine the records in file **F**. Assume that file **F** is maintained in memory by a Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses (2-digit) of locations in HT. Let the keys in **K** and addresses in **L** are Integers. Design and develop a Program in C that uses Hash function **H: K @L** as  $H(K)=K \text{ mod } m$  (**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**.

### ABOUT THE EXPERIMENT:

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

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of size of data. Hash Table uses array as a storage medium and uses hash technique to generate index where an element is to be inserted or 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 hashtable of size 20, and following items are to be stored. Item are in (key, value) format.



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

S.n.	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 used create already used index of the array. In such case, we can search the next empty location in the array by looking into the next cell until we found an empty cell. This technique is called linear probing.

S.n.	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

### Basic Operations

Following are basic primary operations of a hashtable which are following.  Search – search an element in a hashtable.

- Insert – insert an element in a hashtable.
- delete – delete an element from a hashtable.

### DataItem

Define a data item having some data, and key based on which search is to be conducted in hashtable.

```
struct DataItem
{
    int data;
    int key;
};
```

### HashMethod

Define a hashing method to compute the hash code of the key of the data item.

```
int hashCode(int key)
{
    return key % SIZE;
}
```

### ALGORITHM:

Step 1: Start.

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

Step 3: Assume that file **F** is maintained in memory by a Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses (2-digit) of locations in HT.

Step 3: Let the keys in **K** and addresses in **L** are Integers

Step 4: Hash function **H: K ®L** as  $H(K)=K \bmod m$  (**remainder** method)

Step 5: Hashing as to map a given key **K** to the address space **L**, Resolve the collision (if any) is using **linear probing**.

Step6: Stop.

**PROGRAM CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

struct employee {
    int id;
    char name[15]; };
typedef struct employee EMP;
EMP emp[MAX];
int a[MAX];

int create(int num) {
    int key;
    key = num % 100;
    return key;
}

int getemp(EMP emp[],int key)
{
    printf("\nEnter emp id: ");
    scanf("%d",&emp[key].id);
    printf("\nEnter emp name: ");

    fflush();
    gets(emp[key].name);
    return key;
}

void display() {
    int i, ch;
    printf("\n1.Display ALL\n2.Filtered Display");
    printf("\nEnter the choice:  ");
    scanf("%d",&ch);
    if(ch == 1)
    {
        printf("\nThe hash table is:\n");
        printf("\nHTKey\tEmpID\tEmpName");
        for(i=0; i<MAX; i++)
            printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
    }
}
```

```
else
{
    printf("\nThe hash table is:\n");
    printf("\nHTKey\tEmpID\tEmpName");
    for(i=0; i<MAX; i++)
        if(a[i] != -1)
            {
                printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
                continue;
            }
    }
}
```

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

```
        a[i] = num;
        flag = 1;
        break;
    }
    i = 0;
    while((i < key) && (flag == 0))
    {
        if(a[i] == -1)
        {
            a[i] = num;
            flag=1; break;
        }
        i++;
    } // end while
} // end else
} // end linear_prob()

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

**SAMPLE OUTPUT:**

RUN1:

Enter the data: 2

Enter emp id: 100

Enter emp name: Anand

Do you wish to continue?

(1/0): 1

Enter the data: 4

Enter emp id: 101

Enter emp name:

Kumar

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

1.Display ALL

2.Filtered

Display Enter

the choice: 1

The hash table is:

HTKey	EmpID	EmpName
0	0	
1	0	Anand
2	100	
3	0	Kumar
4	101	
5	0	
6	0	
7	0	
8	0	
9	0	

RUN2:

Enter the data: 2

Enter emp id: 100

Enter emp name: Anand

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

Enter the data: 4

Enter emp id: 101

Enter emp name: Kumar



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

The hash Table is:

HTKey	EmpID	EmpName
2	100	Anand
4	101	Kumar

---

## VIVA QUESTIONS AND ANSWERS

### 1) What is data structure?

Data structures refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with data structure, we not only focus on one piece of data, but rather different set of data and how they can relate to one another in an organized manner.

### 2) Differentiate file structure from storage structure.

Basically, the key difference is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

### 3) When is a binary search best applied?

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is search starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

### 4) What is a linked list?

A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link of data storage.

### 5) How do you reference all the elements in a one-dimension array?

To do this, an indexed loop is used, such that the counter runs from 0 to the array size minus one. In this manner, we are able to reference all the elements in sequence by using the loop counter as the array subscript.

### 6) In what areas do data structures applied?

Data structures is important in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

### 7) What is LIFO?

LIFO is short for Last In First Out, and refers to how data is accessed, stored and retrieved. Using this scheme, data that was stored last , should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

**8) What is a queue?**

A queue is a data structures that can simulates a list or stream of data. In this structure, new elements are inserted at one end and existing elements are removed from the other end.

**9) What are binary trees?**

A binary tree is one type of data structure that has two nodes, a left node and a right node. In programming, binary trees are actually an extension of the linked list structures.

**10) Which data structures is applied when dealing with a recursive function?**

Recursion, which is basically a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

**11) What is a stack?**

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

**12) Explain Binary Search Tree**

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

**13) What are multidimensional arrays?**

Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using a single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

**14) Are linked lists considered linear or non-linear data structures?**

It actually depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

**15) How does dynamic memory allocation help in managing data?**

Aside from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

**16) What is FIFO?**

FIFO is short for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

**17) What is an ordered list?**

An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key values form an increasing sequence, as the list is traversed.

**18) What is merge sort?**

Merge sort takes a divide-and-conquer approach to sorting data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continues until you have one single sorted list.

**19) Differentiate NULL and VOID.**

Null is actually a value, whereas Void is a data type identifier. A variable that is given a Null value simply indicates an empty value. Void is used to identify pointers as having no initial size.

**20) What is the primary advantage of a linked list?**

A linked list is a very ideal data structure because it can be modified easily. This means that modifying a linked list works regardless of how many elements are in the list.

**21) What is the difference between a PUSH and a POP?**

Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being "pushed" into the stack. On the other hand, a pop denotes data retrieval, and in particular refers to the topmost data being accessed.

**22) What is a linear search?**

A linear search refers to the way a target key is being searched in a sequential data structure. Using this method, each element in the list is checked and compared against the target key, and is repeated until found or if the end of the list has been reached.

**23) How does variable declaration affect memory allocation?**

The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

**24) What is the advantage of the heap over a stack?**

Basically, the heap is more flexible than the stack. That's because memory space for the heap can be dynamically allocated and de-allocated as needed. However, memory of the heap can at times be slower when compared to that stack.

**25) What is a postfix expression?**

A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

**26) What is the difference between the HEAP and the STACK?**

(Solution: HEAP is used to store dynamically allocated memory (malloc). STACK stores static data (int, const).)

**27) Where in memory are HEAP and the STACK located relative to the executing program?**

(Solution: The STACK and HEAP are stored "below" the executing program. The HEAP "grows" toward the program executable while the STACK grows away from it.)

**28) Describe the data structures of a double-linked list.**

(Solution: A double-linked list structure contains one pointer to the previous record in the list and a pointer to the next record in the list plus the record data.)

**29) How do you insert a record between two nodes in double-linked list?**

(Solution: Previous R; Data R; Next R; To insert a record (B) between two others (A and C): Previous.B = A; Next.B = C; Next.A = B; Previous.C = B;)

**30) In which data structure, elements can be added or removed at either end, but not in the middle?**

(Solution: queue)

**31) Which one is faster? A binary search of an ordered set of elements in an array or a sequential search of the elements.**

(Solution: binary search)

**32) What is a balanced tree?**

(Solution: A binary tree is balanced if the depth of two subtrees of every node never differ by more than one)

**33) Which data structure is needed to convert infix notations to post fix notations?**

(Solution: stack)

**34) What is data structure or how would you define data structure?**

(Solution: In programming the term data structure refers to a scheme for organizing related piece of information. Data Structure = Organized Data + Allowed Operations.)

**35) Which data structures we can implement using link list?**

(Solution: queue and stack)

**36) List different types of data structures?**

(Solution: Link list, queue, stack, trees, files, graphs)

**37) Define a linear and non linear data structure.**

Linear data structure: A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. Ex: Arrays, Linked Lists

Non-Linear data structure: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure. Ex: Trees, Graphs