BALLARI INSTITUTE OF TECHNOLOGY AND MANAGEMENT,BALLARI

(Autonomous Institute under VTU,Belagavi | Approved by AICTE,New Delhi)

“Jnana Gangotri" Campus, Hospet Rd, near Allipura, Ballari, 583104

COMPUTER SCIENCE AND ENGINEERING-DATA SCIENCE

Academic Year: 2023-2024

Semester	:	III (CSE-DS)
Subject	:	Data Structures and Application Laboratory
Subject Code	:	22CDL35
Faculty Name	:	Dr. Jagadeesh R M/Umadevi B E
Designation	:	Asst.Professor/Asst.Professor

Introduction about Data Structures and Application

1. **What is Data?**

   Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.

   There are different kinds of data such as

   Sound

   Video

   Single character

   Picture

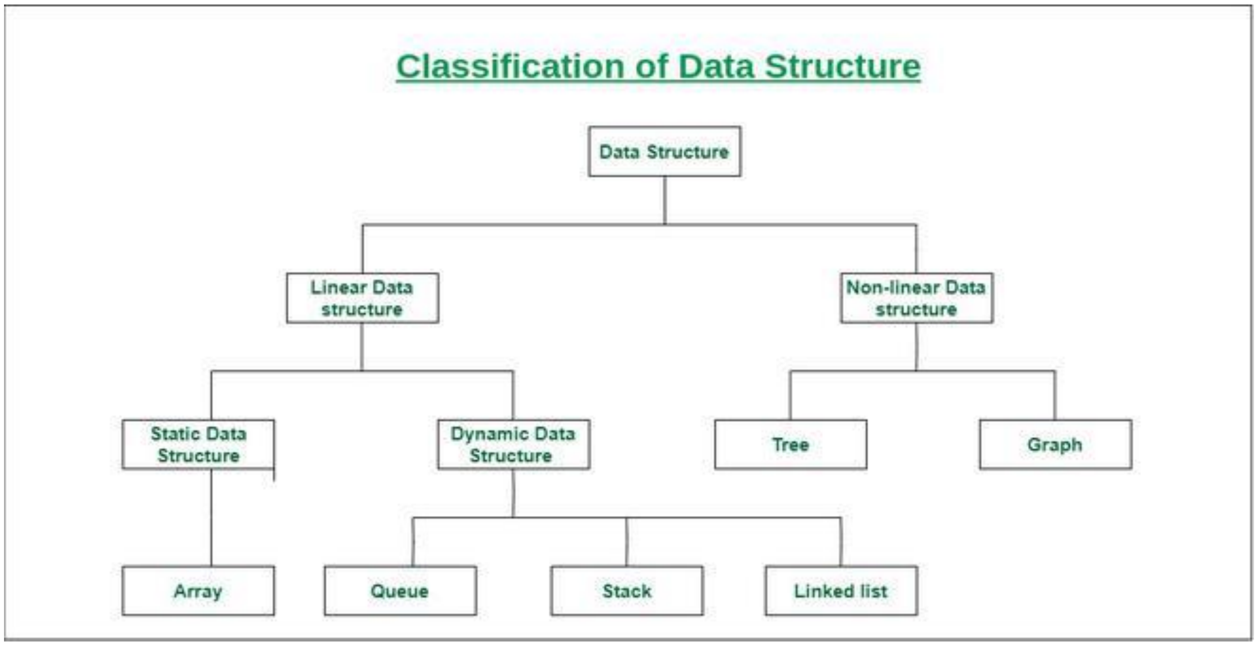
   Boolean(True/False)

   Text(String)

2. **What is Data Structure?**

   A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

   A data structure is not only used for organizing the data. It is also used for processing, retrieving, and storing data.



Linear data structure: Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.
Examples of linear data structures are array, stack, queue, linked list.

- **Static data Structure:** Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
  Ex: Array
- **Dynamic data Structure:** In Dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory(space) complexity of the code.
  Ex: queue, stack

Non-linear data structure: Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can't traverse all the elements in a single run only.
Examples of non-linear data structures are trees and graphs.

Array: An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).
Syntax: int Array[size];

Linked List: A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. rather they are linked using pointers. Linked List forms a series of connected nodes, where each node stores the data and the address of the next node
Syntax: Struct node * next;

Stack: A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (**front and rear**). It contains only one pointer **top pointer** pointing to the topmost element of the stack.

Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack.

The following are some common operations implemented on the stack:

- **Push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **Pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty():** It determines whether the stack is empty or not.
- **isFull():** It determines whether the stack is full or not.'
- **Peek():** It returns the element at the given position.
- **Count():** It returns the total number of elements available in a stack.
- **Change():** It changes the element at the given position.
- **Display():** It prints all the elements available in the stack.

Queue: A **Queue** is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

There are four different types of queue that are listed as follows -

- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

Binary Tree: Binary Tree is defined as a tree data structure where each node has at most 2 children. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

Basic Operation On Binary Tree:

- Inserting an element.
- Removing an element.
- Searching for an element.
- Traversing the tree.

Heap: A Heap is a special Tree-based data structure in which the tree is a complete binary tree.

Operations of Heap Data Structure:

- **Heapify:** a process of creating a heap from an array.
- **Insertion:** process to insert an element in existing heap time complexity O(log N).
- **Deletion:** deleting the top element of the heap or the highest priority element, and then organizing the heap and returning the element with time complexity O(log N).
- **Peek:** to check or find the first (or can say the top) element of the heap.

Heaps can be of two types:

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.

Hashing: Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Components of Hashing

There are majorly three components of hashing:

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

Graph: A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(E, V).

<u>Components of a Graph</u>

- **Vertices:** Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.
- **Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way. There are no rules. Sometimes, edges are also known as arcs. Every edge can be labeled/unlabelled.

Advantages of data structure:

1. Improved data organization and storage efficiency.
2. Faster data retrieval and manipulation.
3. Facilitates the design of algorithms for solving complex problems.
4. Eases the task of updating and maintaining the data.
5. Provides a better understanding of the relationships between data elements.

Disadvantage of Data Structure:

1. Increased computational and memory overhead.
2. Difficulty in designing and implementing complex data structures.
3. Limited scalability and flexibility.
4. Complexity in debugging and testing.
5. Difficulty in modifying existing data structures.

EXPERIMENT-1

Program 1: Design, Develop and Implement a menu driven Program in C for the following Array operations
a. Creating an 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<stdlib.h>
#define MAX 5

int a[MAX], pos, elem;
int n = 0;

/*Function Prototype*/
void create();
void display();
void insert();
void delete();

void main()
{
        int choice;
```

```
	while(1)
	{
		printf("\n\n~~~~MENU~~~~");
		printf("\n=>1. Create an array of N integers");
		printf("\n=>2. Display of array elements");
		printf("\n=>3. Insert ELEM at a given POS");
		 printf("\n=>4. Delete an element at a given POS");
		 printf("\n=>5. Exit");
		 printf("\n Enter your choice: ");
		scanf("%d", &choice);
		switch(choice)
		{
			case 1:		create();
					break;
			case 2:		display();
					break;
			case 3:		insert();
					break;
			case 4:		delete();
					break;
			case 5:		exit(1);
					break;
			default:	printf("\nPlease enter a valid choice:");
		}
	}
}
```

/*Creating an Array*/

```
void create()
{
	int i;
	printf("\n Enter the number of elements: ");
	scanf("%d", &n);
	printf("\n Enter the elements: ");
	for(i=0; i<n; i++)
	{
		scanf("%d", &a[i]);
	}
}
```

/*Displaying an array elements*/

```
void display()
{
	int i;
	if(n == 0)
```

```
        {
                printf("\n No elements to display");
                return;
        }
        printf("\nArray elements are: ");
        for(i=0; i<n;i++)
                printf("%d\t ", a[i]);
}
```

/*Inserting an element into an array*/

```
void insert()
{
        int i;

        if(n == MAX)
        {
                printf("\nArray is full. Insertion is not possible");
                return;
        }

        do
        {
                printf("\n Enter a valid position where element to be inserted:    ");
                scanf("%d", &pos);
        }while(pos > n);

        printf("\n Enter the value to be inserted:   ");
        scanf("%d", &elem);

        for(i=n-1; i>=pos ; i--)
        {
                a[i+1] = a[i];
        }
        a[pos] = elem;
        n = n+1;
        display();
}
```

/*Deleting an array element*/

```
void delete()
{
        int i;

        if(n == 0)
        {
                printf("\nArray is empty and no elements to delete");
                return;
        }
        do
        {
            printf("\n Enter a valid position from where element to be deleted:    ");
              scanf("%d", &pos);
        }
        while(pos>=n);
        elem = a[pos];
                printf("\n Deleted element is : %d \n", elem);
        for( i = pos; i< n-1; i++)
        {
                a[i] = a[i+1];
        }
        n = n-1;
        display();
        }
```

Sample Output 1

```
--------MENU-----------
1. CREATE
2.  DISPLAY
3. INSERT
4.  DELETE
5.  EXIT
-----------------------
ENTER YOUR CHOICE: 1
Enter the size of the array elements: 3
Enter the elements for the array: 10 25 30
ENTER YOUR CHOICE: 2
The array elements are: 10 25 30
ENTER YOUR CHOICE: 3
Enter the position for the new element: 1
Enter the element to be inserted : 20
 ENTER YOUR CHOICE: 2
```

```
The array elements are: 10 20 25 30
ENTER YOUR CHOICE: 4
Enter the position of the element to be deleted: 3
The deleted element is =30
enter your choice: 5
 Exit
```

Sample Output 2

```
--------MENU-----------
  1. CREATE
  2.  DISPLAY
  3.  INSERT
  4.  DELETE
  5. EXIT
 -----------------------
ENTER YOUR CHOICE: 1
Enter the size of the array elements: 3
Enter the elements for the array: 20 20 20
ENTER YOUR CHOICE: 2
The array elements are: 20 20 20
ENTER YOUR CHOICE: 3
Enter the position for the new element: 1
Enter the element to be inserted : 1
ENTER YOUR CHOICE: 2
The array elements are: 20 10 20 20
ENTER YOUR CHOICE: 4
Enter the position of the element to be deleted: 3
The deleted element is =20
enter your choice: 5
Exit
```

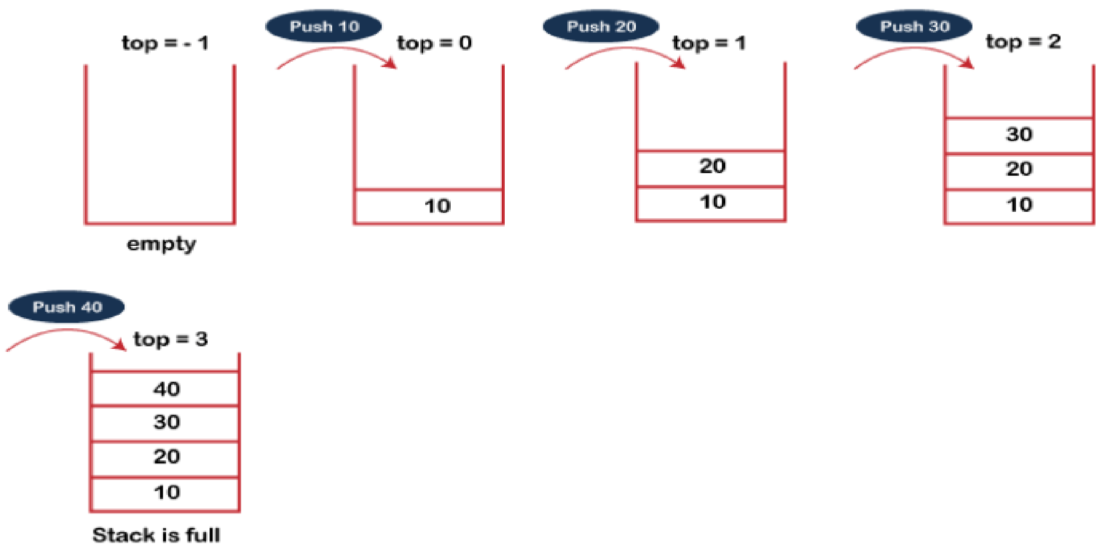
EXPERIMENT-2

Program 2: Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
a. *Push* an Element on to Stack
b. *Pop* an Element from Stack
c. Demonstrate how Stack can be used to check *Palindrome*
d. Demonstrate *Overflow* and *Underflow* situations on Stack
e. Display the status of Stack
f. Exit
Support the program with appropriate functions for each of the above operations

PUSH operation: -

The steps involved in the PUSH operation is given below:

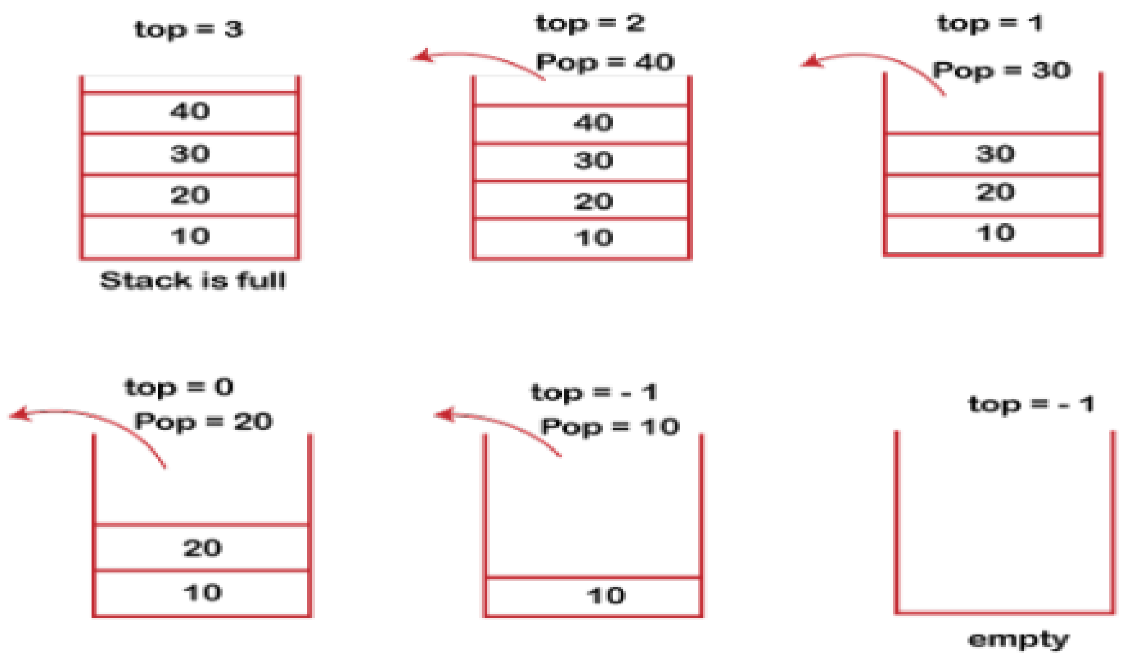
- Before inserting an element in a stack, we check whether the stack is full.
- If we try to insert the element in a stack, and the stack is full, then the overflow condition occurs.
- When we initialize a stack, we set the value of top as -1 to check that the stack is empty.
- When the new element is pushed in a stack, first, the value of the top gets incremented, i.e., top=top+1, and the element will be placed at the new position of the top.
- The elements will be inserted until we reach the max size of the stack.



POP operation:-

The steps involved in the POP operation is given below:

- Before deleting the element from the stack, we check whether the stack is empty.
- If we try to delete the element from the empty stack, then the underflow condition occurs.
- If the stack is not empty, we first access the element which is pointed by the top o Once the pop operation is performed, the top is decremented by 1, i.e., top=top-1.



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 4: Pop element from stack along with display the stack contents. if stack is empty give a message as „Stack is Underflow‟.

Step 5: Check whether the stack contents are Palindrome or not.

Step 6: Stop.

Program code:

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

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

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

void main()
{
  int choice, item;
    while(1)
        {
                printf("\n\n\n\n~~~~~~Menu~~~~~~ : ");
                printf("\n=>1.Push an Element to Stack and Overflow demo ");
                printf("\n=>2.Pop an Element from Stack and Underflow demo");
                printf("\n=>3.Palindrome demo ");
                printf("\n=>4.Display ");
                printf("\n=>5.Exit");
                printf("\n Enter your choice: ");
                scanf("%d", &choice);
                switch(choice)
                {
                        case 1:         printf("\n Enter an element to be pushed: ");
                                        scanf("%d", &item);
                                        push(item);
```

```
                                        break;
                        case 2:         item = pop();
                                        if(item != -1)
                                                printf("\n Element popped is: %d", item);
                                        break;
                        case 3:         palindrome();
                                        break;
                        case 4:         display();
                                        break;
                        case 5:         exit(1);
                        default:        printf("\n Please enter valid choice ") ;
                                        break;
                }
        }
}

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

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

int pop()
{
        int item;
```

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

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

void palindrome()
{
        int flag=1,i;
        printf("\n Stack content are:\n");
        for(i=top; i>=0 ; i--)
                printf("| %d |\n", s[i]);
        printf("\n Reverse of stack content are:\n");
        for(i=0; i<=top; i++)
```

```
            printf("| %d |\n", s[i]);

        for(i=0; i<=top/2; i++)
        {
            if( s[i] != s[top-i] )
            {
                flag = 0;
                break;
            }
        }
        if(flag == 1)
        {
            printf("\n It is palindrome number");
        }
        else
        {
            printf("\n It is not a palindrome number");
        }
}
```

Output:

```
~~~~~~Menu~~~~~~ :
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 11

~~~~~~Menu~~~~~~ :
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
```

```
Enter your choice: 1
Enter an element to be pushed: 12

~~~~~~Menu~~~~~~ :
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 13

~~~~~~Menu~~~~~~ :
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 14

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 15

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 16
~~~~Stack overflow~~~~

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 4
```

```
Stack elements are:
| 15 |
| 14 |
| 13 |
| 12 |
| 11 |

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 15

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 4
Stack elements are:
| 14 |
| 13 |
| 12 |
| 11 |
~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 14

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 13

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

```
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 12

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 11

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
~~~~Stack underflow~~~~

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 4
~~~~Stack is empty~~~~

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 11

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
```

```
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 22

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 11

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 3
Stack content are:
| 11 |
| 22 |
| 11 |

Reverse of stack content are:
| 11 |
| 22 |
| 11 |

It is palindrome number

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 11

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
```

```
Element popped is: 22

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 2
Element popped is: 11

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 11

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 22

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 1
Enter an element to be pushed: 33

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 3
Stack content are:
| 33 |
| 22 |
```

```
| 11 |

Reverse of stack content are:
| 11 |
| 22 |
| 33 |

It is not a palindrome number

~~~~~~Menu~~~~~
=>1.Push an Element to Stack and Overflow demo
=>2.Pop an Element from Stack and Underflow demo
=>3.Palindrome demo
=>4.Display
=>5.Exit
Enter your choice: 5
```

EXPERIMENT-3

Program 3 : Design, Develop and Implement a Program in C Language 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.

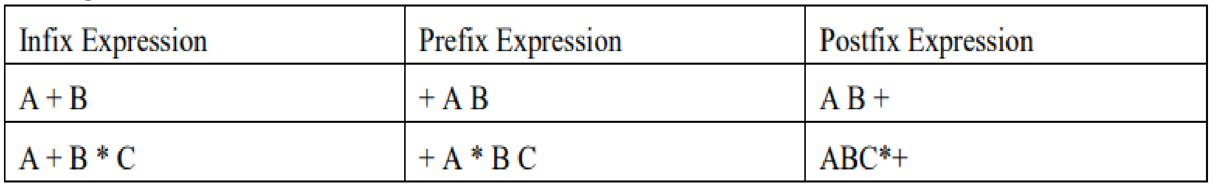
ABOUT THE EXPERIMENT:

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

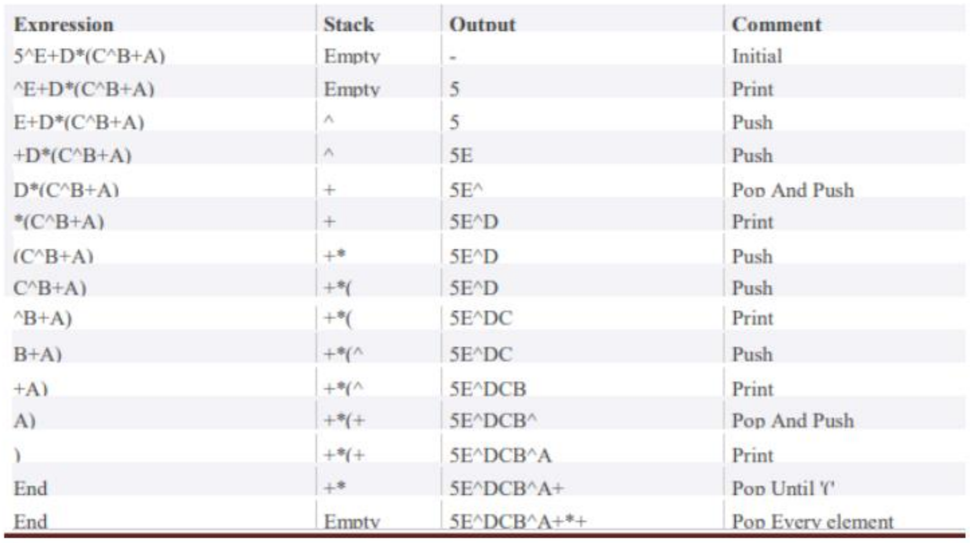
ALGORITHM:

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



Expression	Stack	Output	Comment
5^E+D*(C^B+A)	Empty	-	Initial
^E+D*(C^B+A)	Empty	5	Print
E+D*(C^B+A)	^	5	Push
+D*(C^B+A)	^	5E	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

Program code:

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

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

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

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

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

```
                                        j++;
                                }
                                push(symb);
                                break;
                default:        postfix[j] = symb;
                                j++;
            }
        }
        while(top > 0)
        {
                temp = pop();
                postfix[j] = temp;
                j++;
        }
        postfix[j] = '\0';
}

 void push(char item)
{
        top = top+1;
        stack[top] = item;
}

 char pop()
{
        char item;
        item = stack[top];
        top = top-1;
        return item;
}
```

```
int prec(char symb)
{
        int p;
        switch(symb)
        {
                case '#' :      p = -1;
                                break;

                case '(' :
                case ')' :      p = 0;
                                break;

                case '+' :
                case '-' :      p = 1;
                                break;

                case '*' :
                case '/' :
                case '%' :      p = 2;
                                break;

                case '^' :
                case '$' :      p = 3;
                                break;
        }
        return p;
}
```

Output:1

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

The entered infix expression is :
 (a+b)+c/d*e

The corresponding postfix expression is :
 ab+cd/e*+
```

Output:2

```
Enter the valid infix expression:          (4+2)+5/10*6


The entered infix expression is :

 (4+2)+5/10*6


The corresponding postfix expression is :

 42+510/6*+
```

EXPERIMENT-4

Program 4: Design, Develop and Implement a Program in C Language for Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

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.
```

Program Code

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

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

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

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

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

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

Output:1

```
Enter a valid postfix expression:
623+-382/+*2$3+
Result = 52
```

Output:2

```
Enter a valid postfix expression:
42$3*3-84/11+/+
Result = 46
```

EXPERIMENT-5

Program 5:Design, Develop and Implement a Program in C Language using Recursion for
a. generating Fibonacci series of n numbers
b. Solving Tower of Hanoi problem with n disks

Tower of Hanoi is a mathematical puzzle where we have three rods (**A**, **B**, and **C**) and **N** disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod **A**. The objective of the puzzle is to move the entire stack to another rod (here considered **C**), 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.

Tower of Hanoi using Recursion:

The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:

- Shift 'N-1' disks from 'A' to 'B', using C.
- Shift last disk from 'A' to 'C'.
- Shift 'N-1' disks from 'B' to 'C', using A.

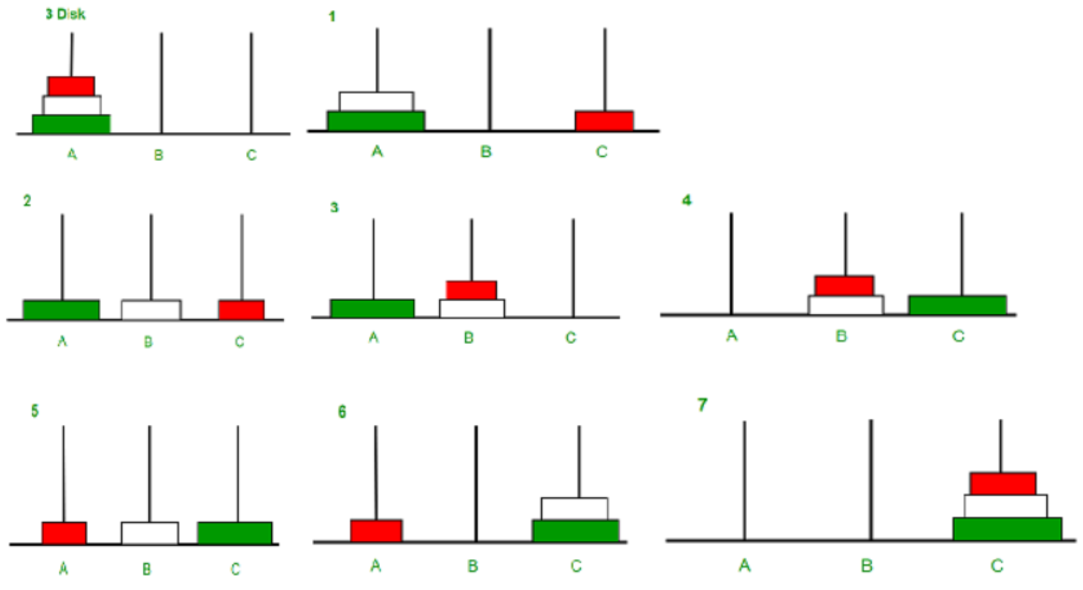


Image illustration for 3 disks

Follow the steps below to solve the problem:

Create a function **tower of Hanoi** where pass the N(current number of disk), **from_rod,to_rod,aux_rod**

Create a function **tower Of Hanoi** where pass the **N** (current number of disk), **from_rod**, **to_rod**, **aux_rod.**

- Make a function call for N – 1 th disk.
- Then print the current the disk along with **from_rod** and **to_rod**
- Again make a function call for N – 1 th disk.

ALGORITHM:

Step 1: Start.

Step 2: Read N number of discs.

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

Step 4: Stop.

Program Code:

```
#include<stdio.h>
#include<math.h>
void tower(int n, char from_peg,  char aux_peg, char to_peg);

void main()
{
        int n;
        printf("\nEnter the number of disks: ");
        scanf("%d", &n);
        tower(n, 'A', 'B', 'C');
 // A-> from_peg B->aux_peg C->to_peg
   printf("\nTotal number of moves = %0.0f", pow(2,n)-1 );
}


void tower(int n, char from_peg,  char aux_peg, char to_peg)
```

```
// A-> from_peg B->aux_peg C->to_peg
{
   if(n == 1)
      {
      printf("\nMove disk %d from %c peg to %c peg", n, from_peg,  to_peg);
            return;
      }
// move n-1 disks from A(from_peg) to B(to_peg) using C(aux_peg) as auxiliary
            tower(n-1, from_peg,  to_peg,  aux_peg);

            printf("\nMove disk %d from peg %c to %c peg", n, from_peg,  to_peg);
// move n-1 disks from B(aux_peg) to C(to_peg) using A(from_peg) as auxiliary
            tower(n-1, aux_peg, from_peg, to_peg);
}
```

Output:1

```
Enter the number of disks: 3
Move disk 1 from A peg to C peg
Move disk 2 from peg A to B peg
Move disk 1 from C peg to B peg
Move disk 3 from peg A to C peg
Move disk 1 from B peg to A peg
Move disk 2 from peg B to C peg
Move disk 1 from A peg to C peg
Total number of moves = 7
```

Output:2

```
Enter the number of disks: 5
Move disk 1 from A peg to C peg
Move disk 2 from peg A to B peg
Move disk 1 from C peg to B peg
Move disk 3 from peg A to C peg
Move disk 1 from B peg to A peg
Move disk 2 from peg B to C peg
Move disk 1 from A peg to C peg
Move disk 4 from peg A to B peg
Move disk 1 from C peg to B peg
Move disk 2 from peg C to A peg
Move disk 1 from B peg to A peg
Move disk 3 from peg C to B peg
Move disk 1 from A peg to C peg
Move disk 2 from peg A to B peg
Move disk 1 from C peg to B peg
Move disk 5 from peg A to C peg
Move disk 1 from B peg to A peg
Move disk 2 from peg B to C peg
Move disk 1 from A peg to C peg
Move disk 3 from peg B to A peg
Move disk 1 from C peg to B peg
Move disk 2 from peg C to A peg
Move disk 1 from B peg to A peg
Move disk 4 from peg B to C peg
Move disk 1 from A peg to C peg
Move disk 2 from peg A to B peg
Move disk 1 from C peg to B peg
Move disk 3 from peg A to C peg
Move disk 1 from B peg to A peg
Move disk 2 from peg B to C peg
Move disk 1 from A peg to C peg
Total number of moves = 31
```

EXPERIMENT-6

Program 6: Design, Develop and Implement a menu driven Program in C Language for the following operations on QUEUE of Integers (Array Implementation of Queue with maximum size MAX)

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

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

ABOUT THE EXPERIMENT:

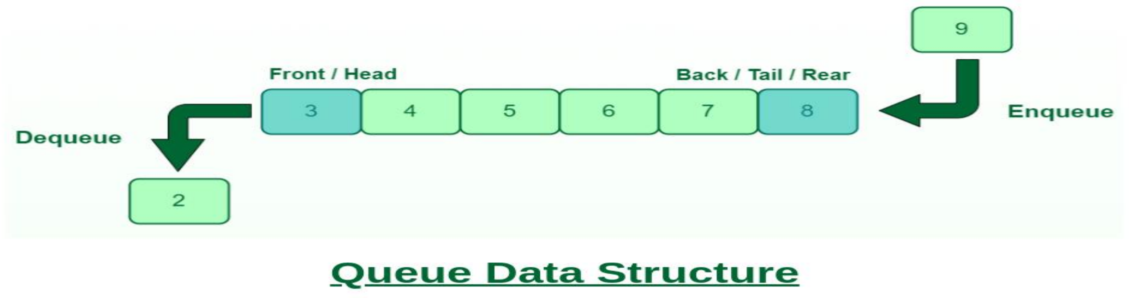
❖ A **Queue** is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out(FIFO) order.

❖ We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.

FIFO Principle of Queue:

❖ A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).

❖ Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the **front** of the queue(sometimes, **head** of the queue), similarly, the position of the last entry in the queue, that is, the one most recently added, is called the **rear** (or the **tail**) of the queue. See the below figure.



Queue Representation:

Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are

- **Queue:** the name of the array storing queue elements.
- **Front**: the index where the first element is stored in the array representing the queue.
- **Rear:** the index where the last element is stored in an array representing the queue.

ALGORITHM:

```
Step 1 : START
Step 2 : Check if the queue is full.
Step 3 : If the queue is full, produce overflow error and exit.
Step 4 : If the queue is not full, increment rear pointer to point the next empty space.
Step 5 : Add data element to the queue location, where the rear is pointing.
Step 6 : return success.
Step 7 :  END
```

Program Code:

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

int queue_arr[MAX];
int rear=-1;
int front=-1;

void insert(int item);
int del();
int peek();
void display();
int isFull();
int isEmpty();

int main()
{
	int choice,item;
	while(1)
	{
		printf("\n1.Insert\n");
		printf("2.Delete\n");
```

```
            printf("3.Display element at the front\n");
            printf("4.Display all elements of the queue\n");
            printf("5.Quit\n");
            printf("\n Enter your choice : ");
            scanf("%d",&choice);

           /*Switch Case*/
          switch(choice)
            {
            case 1:
                 printf("\n Input the element for adding in queue : ");
                 scanf("%d",&item);
                 insert(item);
                 break;
            case 2:
                 item=del();
                 printf("\n Deleted element is  %d\n",item);
                 break;
            case 3:
                 printf("\n Element at the front is %d\n",peek());
                 break;
            case 4:
                 display();
                 break;
            case 5:
                 exit(1);
            default:
                 printf("\nWrong choice\n");
            }/*End of switch*/
       }/*End of while*/

       return 0;

}/*End of main()*/

/*Insert Queue Element Function*/
void insert(int item)
{
/*Check the Overflow Condition*/
      if( isFull() )
       {
            printf("\n Queue Overflow\n");
            return;
       }
       if( front == -1 )
            front=0;
```

```
        rear=rear+1;
        queue_arr[rear]=item ;
}/*End of insert()*/

/*Queue Delete Function*/
int del()
{
        int item;
 /*Check the Underflow Condition*/
        if( isEmpty() )
        {
                printf("\n Queue Underflow\n");
                exit(1);
        }
        item=queue_arr[front];
        front=front+1;
        return item;
}/*End of del()*/

int peek()
{
   if( isEmpty() )
        {
                printf("\n Queue Underflow\n");
                exit(1);
        }
        return queue_arr[front];
}/*End of peek()*/

int isEmpty()
{
        if( front==-1 || front==rear+1 )
                return 1;
        else
                return 0;
}/*End of isEmpty()*/

int isFull()
{
        if( rear==MAX-1 )
                return 1;
        else
                return 0;
}/*End of isFull()*/
```

```
/*Queue Display Function*/
void display()
{
        int i;
        if ( isEmpty() )
        {
                printf("\n Queue is empty\n");
                return;
        }
        printf("\n Queue is :\n\n");
        for(i=front;i<=rear;i++)
                printf("%d  ",queue_arr[i]);
        printf("\n\n");
}/*End of display() */
```

Output:

```
1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 1
 Input the element for adding in queue : 5
 1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 1
 Input the element for adding in queue : 10
 1.Insert
2.Delete
```

```
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 1
 Input the element for adding in queue : 15
 1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 1
 Input the element for adding in queue : 20
 1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 1
 Input the element for adding in queue : 25
 1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 4
 Queue is :
5  10  15  20  25
```

```
1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 1
 Input the element for adding in queue : 30
 Queue Overflow

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 2
 Deleted element is  5

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 2
 Deleted element is  10

1.Insert
2.Delete
3.Display element at the front
```

```
4.Display all elements of the queue
5.Quit

 Enter your choice : 3
 Element at the front is 15

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 4
 Queue is :
15  20  25

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

 Enter your choice : 2
 Deleted element is  15

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit
```

```
Enter your choice : 2
Deleted element is  20

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

Enter your choice : 2
Deleted element is  25

1.Insert
2.Delete
3.Display element at the front
4.Display all elements of the queue
5.Quit

Enter your choice : 2
Queue Underflow
```

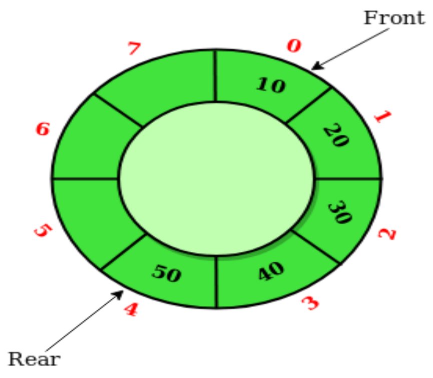
EXPERIMENT-7

Program 7: 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

ABOUT THE EXPERIMENT:

A circular queue is an Extended version of Normal queue where the last element of the queue is connected to the first element of the queue forming a circle.

The operations are performed based on FIFO (First In First Out) principle. It is also called **'Ring Buffer'**.



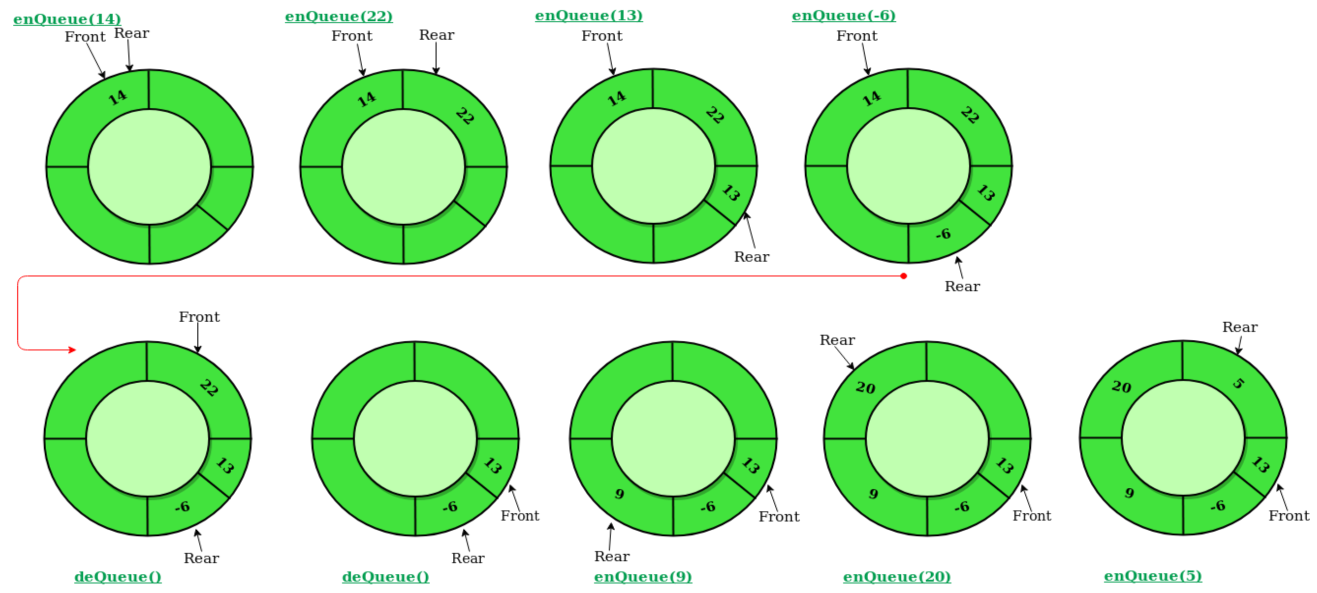
Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value**) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.
- Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
- If it is full then display Queue is full.

- If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
- Check whether the queue is Empty.
- If it is empty then display Queue is empty.
- If the queue is not empty, then get the last element and remove it from the queue.

Illustration of Circular Queue Operations:

Follow the below image for a better understanding of the enqueue and dequeue 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.

Program Code:

```
#include <stdio.h>
//#include <conio.h>
#include <stdlib.h>
#define SIZE 5
int CQ[SIZE];
int front=-1;
int rear=-1, ch;
int IsCQ_Full();
int IsCQ_Empty();
void CQ_Insert(int );
void CQ_Delet();
void CQ_Display();

void main()
{
 printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
while(1)
{
int ele;
printf("Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: if(IsCQ_Full())
printf("Circular Queue Overflow\n");
 else
{
printf("Enter the element to be inserted\n");
scanf("%d",&ele);
CQ_Insert(ele);
}
 break;
case 2: if(IsCQ_Empty())
printf("Circular Queue Underflow\n");
 else
CQ_Delet();
 break;
case 3: if(IsCQ_Empty())
printf("Circular Queue Underflow\n");
 else
CQ_Display();
 break;
case 4: exit(0);
}
}
}

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

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

```
void CQ_Display()
{
int i;
if(front==-1)
printf("Circular Queue is Empty\n");
else
{
printf("Elements of the circular queue are..\n");
for(i=front;i!=rear;i=(i+1)%SIZE)
{
printf("%d\t",CQ[i]);
}
printf("%d\n",CQ[i]);
}
}

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

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

Sample Output:

```
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice
1
Enter the element to be inserted
5
Enter your choice
1
Enter the element to be inserted
10
Enter your choice
1
Enter the element to be inserted
15
Enter your choice
1
Enter the element to be inserted
20
Enter your choice
1
Enter the element to be inserted
25
Enter your choice
3
Elements of the circular queue are..
5       10      15      20      25
Enter your choice
Enter your choice
1
Circular Queue Overflow
```

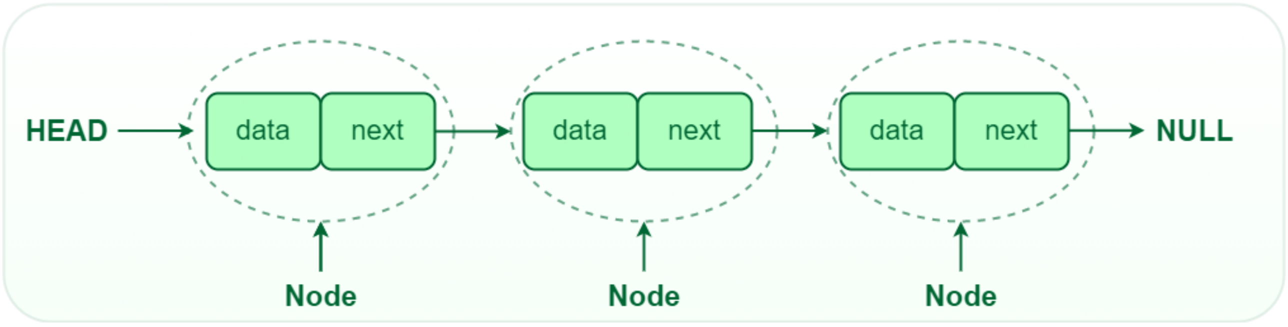
```
Enter your choice
2
Deleted element is: 5
Enter your choice
2
Deleted element is: 10
Enter your choice
2
Deleted element is: 15
Enter your choice
2
Deleted element is: 20
Enter your choice
2
Deleted element is: 25
Circular Queue Underflow
```

EXPERIMENT-8

Program 8: 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 / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit

ABOUT THE EXPERIMENT:

Linked List is a linear data structure, in which elements are not stored at a contiguous location, rather they are linked using pointers. Linked List forms a series of connected nodes, where each node stores the data and the address of the next node.



Node Structure: A node in a linked list typically consists of two components:
Data: It holds the actual value or data associated with the node.
Next Pointer: It stores the memory address (reference) of the next node in the sequence.
Head and Tail: The linked list is accessed through the head node, which points to the first node in the list. The last node in the list points to NULL or nullptr, indicating the end of the list. This node is known as the tail node.

ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student‟s information)
Step 3: Create a singly linked list. (SLL)

Step 4: Display the status of SLL.

Step 5: Count the number of nodes.

Step 6: Perform insertion at front of list.

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

Step 8: Perform insertion at end of the list.

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

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

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

Step 12: Stop.

Program code:

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

struct node
{
   char usn[25],name[25],branch[25];
   int sem;
   long int phone;
   struct node *link;
};
typedef struct node * NODE;
NODE start = NULL;
int count=0;

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

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

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

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

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

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

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

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

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

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

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

```
    switch(ch)
    {
      case 1: start = insertfront();
              break;
      case 2: start = deletefront();
              break;
      case 3: display();
             break;
     default : return;
    }
  }
  return;
}

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

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

      case 2: display();
            break;

      case 3: start = insertend();
            break;

      case 4: start = deleteend();
            break;

      case 5: stackdemo();
            break;

      case 6: exit(0);

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

      }
   }
}
```

Output:

```
~~~Menu~~~
Enter your choice for SLL operation

1:Create SLL of Student Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:Stack Demo using SLL(Insertion and Deletion at Front)
6:Exit

Enter your choice:1
Enter the no of students:    2
Enter the usn,Name,Branch, sem,PhoneNo of the student:11
UMA
CS
3
6845741422
Enter the usn,Name,Branch, sem,PhoneNo of the student:10
SUNANDA
CS
3
9874546547
~~~Menu~~~
Enter your choice for SLL operation

1:Create SLL of Student Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:Stack Demo using SLL(Insertion and Deletion at Front)
```

```
6:Exit

Enter your choice:2
The contents of SLL:

||1|| USN:10| Name:SUNANDA| Branch:CS| Sem:3| Ph:9874546547|
||2|| USN:11| Name:UMA| Branch:CS| Sem:3| Ph:6845741422|
 No of student nodes is 2

~~~Menu~~~
Enter your choice for SLL operation

1:Create SLL of Student Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:Stack Demo using SLL(Insertion and Deletion at Front)
6:Exit

Enter your choice:3
Enter the usn,Name,Branch, sem,PhoneNo of the student:14
POOJA
CS
3
87413512126
~~~Menu~~~
Enter your choice for SLL operation

1:Create SLL of Student Nodes
2:DisplayStatus
3:InsertAtEnd
```

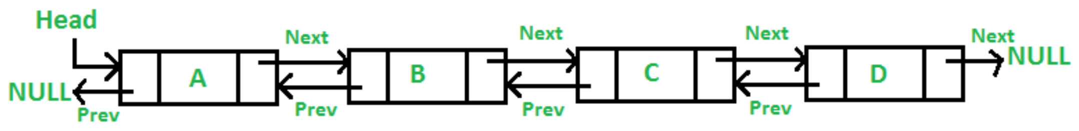
```
4:DeleteAtEnd
5:Stack Demo using SLL(Insertion and Deletion at Front)
6:Exit

Enter your choice:2
The contents of SLL:

||1|| USN:10| Name:SUNANDA| Branch:CS| Sem:3| Ph:9874546547|
||2|| USN:11| Name:UMA| Branch:CS| Sem:3| Ph:6845741422|
||3|| USN:14| Name:POOJA| Branch:CS| Sem:3| Ph:87413512126|
 No of student nodes is 3
```

EXPERIMENT-9

Program 9: 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:

A **doubly linked list** (DLL) is a special type of linked list in which each node contains a pointer to the previous node as well as the next node of the linked list.



Below are operations on the given DLL:

1. **Add a node at the front of DLL:** The new node is always added before the head of the given Linked List. And the newly added node becomes the new head of DLL & maintaining a global variable for counting the total number of nodes at that time.

2. Traversal of a Doubly linked list

3. **Insertion of a node:** This can be done in three ways:
- **At the beginning:** The new created node is insert in before the head node and head points to the new node.
- **At the end:** The new created node is insert at the end of the list and tail points to the new node.
- **At a given position:** Traverse the given DLL to that position(**let the node be X**) then do the following:
1. Change the next pointer of new Node to the next pointer of Node X.
2. Change the prev pointer of next Node of Node X to the new Node.
3. Change the next pointer of node X to new Node.
4. Change the prev pointer of new Node to the Node X.

4 **Deletion of a node:** This can be done in three ways:

- **At the beginning:** Move head to the next node to delete the node at the beginning and make previous pointer of current head to NULL .
- **At the last:** Move tail to the previous node to delete the node at the end and make next pointer of tail node to NULL.
- **At a given position:** Let the prev node of Node at position pos be Node X and next node be Node Y, then do the following:

1. Change the next pointer of Node X to Node Y.
2. Change the previous pointer of Node Y to Node X.

ALGORITHM:

Step 1: Start.

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

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

Step 4: Display the status of DLL.

Step 5: Count the number of nodes.

Step 6: Perform insertion at front of list.

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

Step 8: Perform insertion at end of the list.

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

Step 10: Demonstrate how doubly linked list can be used as double ended queue. Step 11: Stop

Program code:

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

struct node
{
        char ssn[25],name[25],dept[10],designation[25];
        int sal;
        long int phone;
        struct node *llink;
        struct node *rlink;
};
typedef struct node* NODE;

NODE first = NULL;
int count=0;


/*Create  Node Function*/
NODE create()
{
         NODE enode;
         enode = (NODE)malloc(sizeof(struct node));
         if( enode== NULL)
         {
                 printf("\nRunning out of memory");
                 exit(0);
          }
   printf("\nEnter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee: \n");
scanf("%s %s %s %s %d %ld", enode->ssn, enode->name, enode->dept, enode->designation,
&enode->sal, &enode->phone);
          enode->llink=NULL;
          enode->rlink=NULL;
          count++;
          return enode;
}

/*Insert  Node at Front Function*/
NODE insertfront()
{
        NODE temp;
        temp = create();
         if(first == NULL)
        {
               return temp;
         }
         temp->rlink = first;
         first->llink = temp;
         return temp;
}
```

/*Display Node Function*/

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

/*Delete Node at Front Function*/

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

/*Insert Node at End Function*/

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

          if(first == NULL)
          {
                    return temp;
          }
         cur= first;
         while(cur->rlink!=NULL)
        {
                  cur = cur->rlink;
         }

         cur->rlink = temp;
         temp->llink = cur;
          return first;
}
```

/*Delete a Node at End*/

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

        if(first->rlink == NULL)
        {
                  printf("\nThe employee node with the ssn:%s is deleted",first->ssn);
                  free(first);
                  count--;
                  return NULL;
        }

         prev=NULL;
         cur=first;
        while(cur->rlink!=NULL)
        {
            prev=cur;
            cur = cur->rlink;
        }

        cur->llink = NULL;
        printf("\nThe employee node with the ssn:%s is deleted",cur->ssn);
        free(cur);
        prev->rlink = NULL;
        count--;
        return first;
}

void deqdemo()
{
      int ch;
      while(1)
     {
            printf("\nDemo Double Ended Queue Operation");
     printf("\n1:InsertQueueFront\n      2:      DeleteQueueFront\n      3:InsertQueueRear\n
4:DeleteQueueRear\n 5:DisplayStatus\n 6: Exit \n");
            scanf("%d", &ch);

            switch(ch)
           {
                  case 1: first=insertfront();
                              break;
                 case 2: first=deletefront();
                            break;
                case 3: first=insertend();
                            break;
                case 4: first=deleteend();
                           break;
                 case 5: display();
                            break;
                default : return;
         }
     }
}

void main()
{
    int ch,i,n;
    while(1)
    {
        printf("\n\n~~~Menu~~~");
        printf("\n1:Create DLL of Employee Nodes");
```

```
        printf("\n2:DisplayStatus");
        printf("\n3:InsertAtEnd");
        printf("\n4:DeleteAtEnd");
        printf("\n5:InsertAtFront");
        printf("\n6:DeleteAtFront");
        printf("\n7:Double Ended Queue Demo using DLL");
        printf("\n8:Exit \n");
        printf("\nPlease enter your choice: ");
        scanf("%d",&ch);

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

         case 2:  display();
                  break;

          case 3: first = insertend();
                 break;

          case 4: first = deleteend();
                 break;

          case 5: first = insertfront();
                 break;

          case 6: first = deletefront();
                break;

          case 7: deqdemo();
                 break;

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

Output:

```
~~~Menu~~~
1:Create DLL of Employee Nodes
2:DisplayStatus
3:InsertAtEnd4:DeleteAtEnd
5:InsertAtFront
6:DeleteAtFront
7:Double Ended Queue Demo using DLL
8:Exit

Please enter your choice: 1
Enter the no of Employees:   2
Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
1
POO
CS
PROF
50000
654648674
Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
2
SWETHA
DS
PROF
64000
54766874

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

Please enter your choice: 2
ENode:1||SSN:1|Name:POO|Department:CS|Designation:PROF|Salary:50000|Phone
no:654648674
ENode:2||SSN:2|Name:SWETHA|Department:DS|Designation:PROF|Salary:64000|Phone
no:54766874
No of employee nodes is 2
```

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

Please enter your choice: 4
The employee node with the ssn:2 is deleted

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

Please enter your choice: 3
Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
3
UMA
CS
SE
45000
6546851534

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

Please enter your choice: 2
ENode:1||SSN:1|Name:POO|Department:CS|Designation:PROF|Salary:50000|Phone
no:654648674
```

```
ENode:2||SSN:3|Name:UMA|Department:CS|Designation:SE|Salary:45000|Phone
no:6546851534
No of employee nodes is 2

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

EXPERIMENT-10

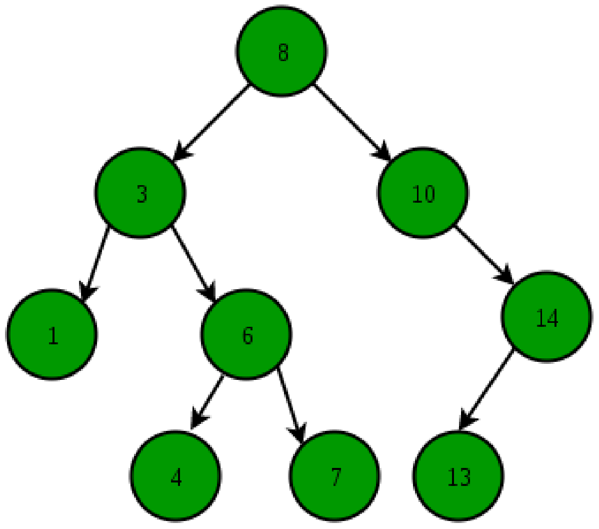
Program 10: Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
e. Exit

ABOUT THE EXPERIMENT:

- A Binary Search Tree (BST) is a special type of binary tree in which the left child of a node has a value less than the node’s value and the right child has a value greater than the node’s value.
- This property is called the BST property and it makes it possible to efficiently search, insert, and delete elements in the tree.
- The root of a BST is the node that has the largest value in the left subtree and the smallest value in the right subtree.
- Each left subtree is a BST with nodes that have smaller values than the root and each right subtree is a BST with nodes that have larger values than the root.

Binary Search Tree is a node-based binary tree data structure that has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.
- This means everything to the left of the root is less than the value of the root and everything to the right of the root is greater than the value of the root. Due to this performing, a binary search is very easy.
- The left and right subtree each must also be a binary search tree.
  There must be no duplicate nodes(BST may have duplicate values with different handling approaches).



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.

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 *lchild;
        struct BST *rchild;
};
typedef struct BST * NODE;

/*Create Root Node*/
NODE create()
{
        NODE temp;
        temp = (NODE) malloc(sizeof(struct BST));
        printf("\nEnter The value: ");
        scanf("%d", &temp->data);

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

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

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

/*Search Node*/

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

/*Tree In-Order Traversal Function*/

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

/*Tree Pre-Order Traversal Function*/

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

/*Tree Post-Order Traversal Function*/

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


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

```
                                        scanf("%d", &n);
                                        for(i=1;i<=n;i++)
                                        {
                                                newnode = create();
                                                if (root == NULL)
                                                        root = newnode;
                                                else
                                                        insert(root, newnode);
                                        }
                                        break;
                        case 2:          if (root == NULL)
                                                printf("\nTree Is Not Created");
                                        else
                                        {
                                                printf("\nThe Preorder display : ");
                                                preorder(root);
                                                printf("\nThe Inorder display : ");
                                                inorder(root);
                                                printf("\nThe Postorder display : ");
                                                postorder(root);
                                        }

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

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

Output:

```
~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 1
Enter the number of elements: 8
Enter The value: 12
Enter The value: 20
Enter The value: 15
Enter The value: 40
Enter The value: 32
Enter The value: 25
Enter The value: 5
Enter The value: 50

~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 2
The Preorder display : 12 5 20 15 40 32 25 50
The Inorder display : 5 12 15 20 25 32 40 50
The Postorder display : 5 15 25 32 50 40 20 12

~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
```

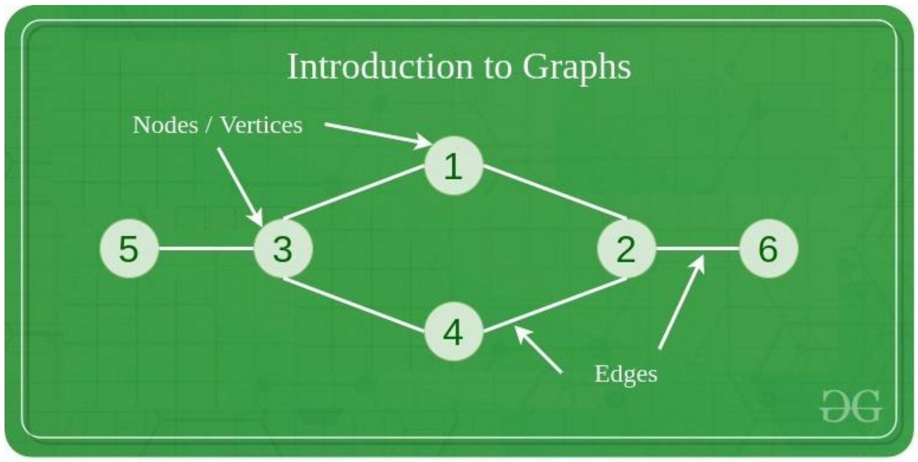
```
4.ExitEnter your choice: 3
Enter Element to be searched: 15
Key element is present in BST

~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 3
Enter Element to be searched: 10
Key element is not found in the BST
```

EXPERIMENT-11

Program 11: Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities
a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

ABOUT THE EXPERIMENT:

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(E, V).



Components of a Graph

- **Vertices:** Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.
- **Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way. There are no rules. Sometimes, edges are also known as arcs. Every edge can be labeled/unlabelled.

➢ Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.

➢ Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node).

Breadth First Search(BFS)

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

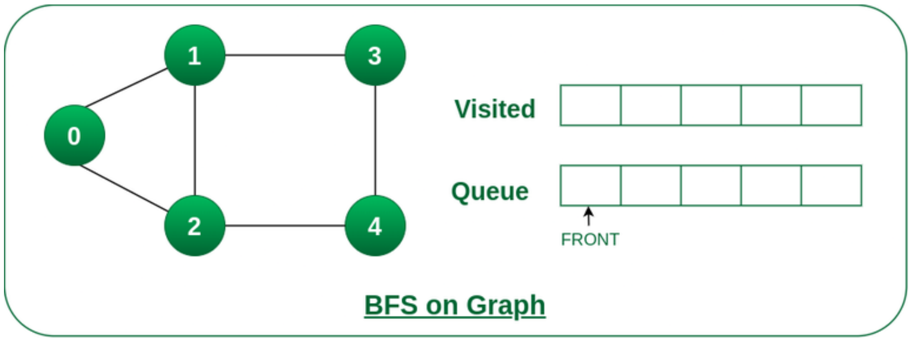
How does BFS work?

- Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.
- To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

illustration:

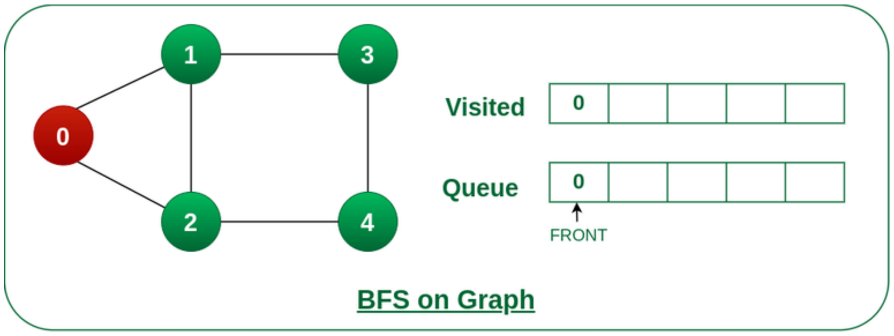
Let us understand the working of the algorithm with the help of the following example.

Step1: Initially queue and visited arrays are empty.



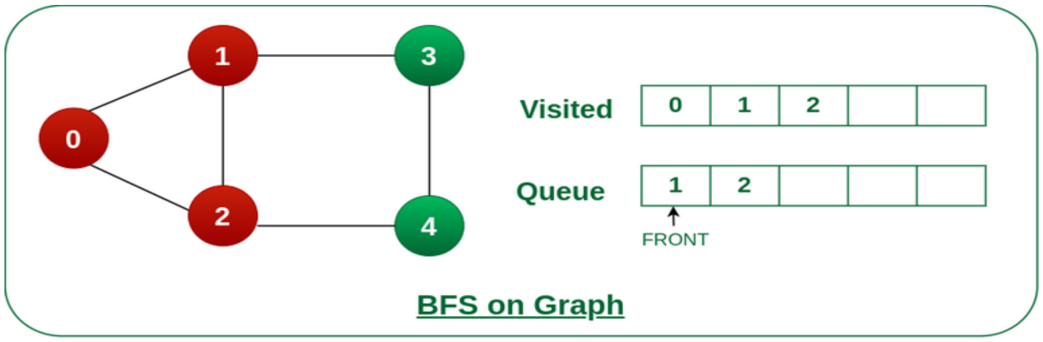
Queue and visited arrays are empty initially.

Step2: Push node 0 into queue and mark it visited



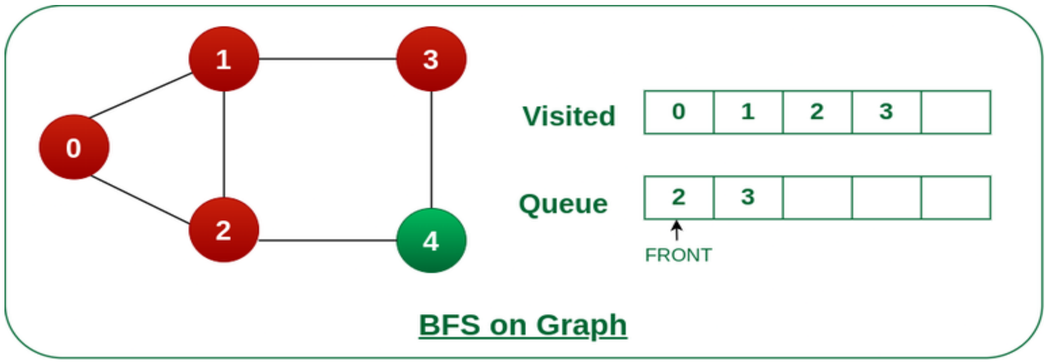
Push node 0 into queue and mark it visited.

Step 3: Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.



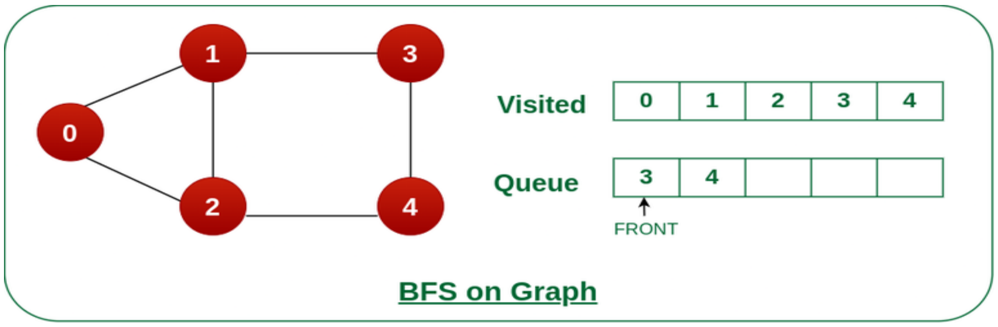
Remove node 0 from the front of queue and visited the unvisited neighbours and push into queue.

Step 4: Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



Remove node 1 from the front of queue and visited the unvisited neighbours and push

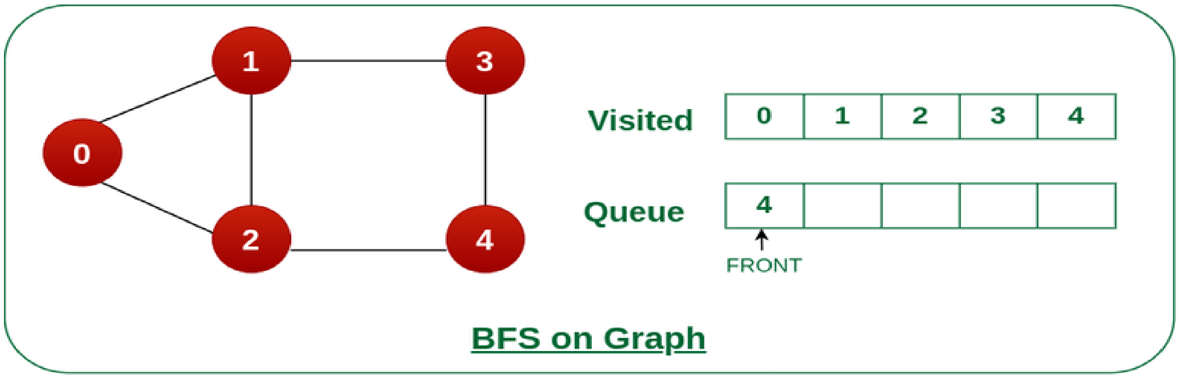
Step 5: Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.

Step 6: Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

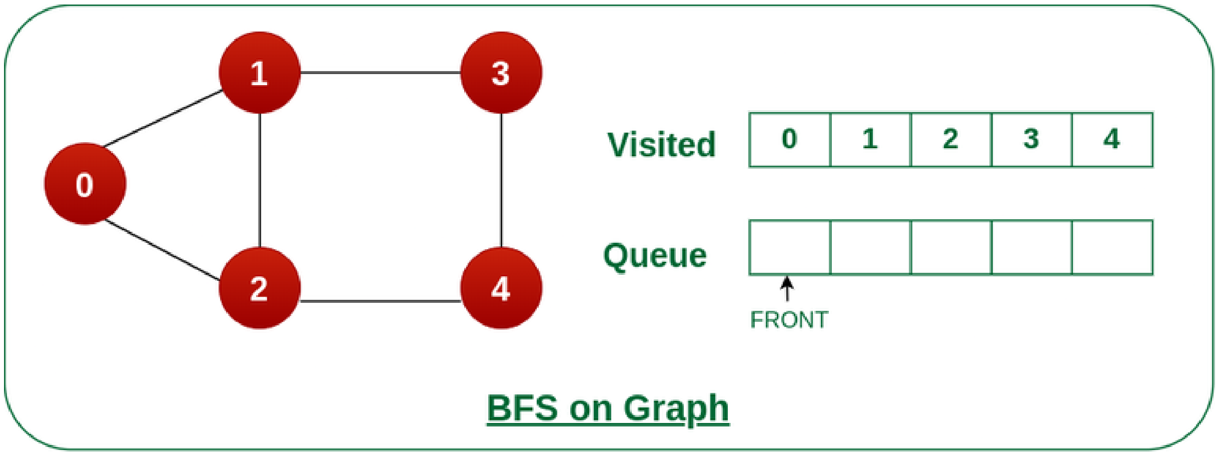
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

Steps 7: Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.

As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.



Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.

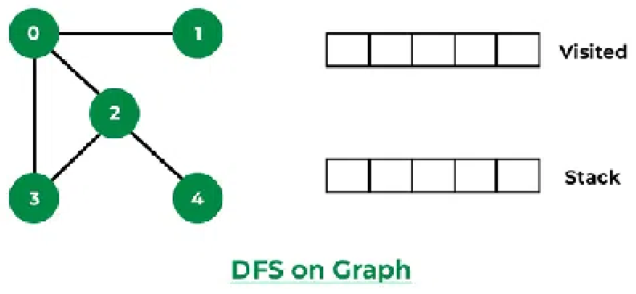
Depth First Search(DFS)

Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

How does DFS work?

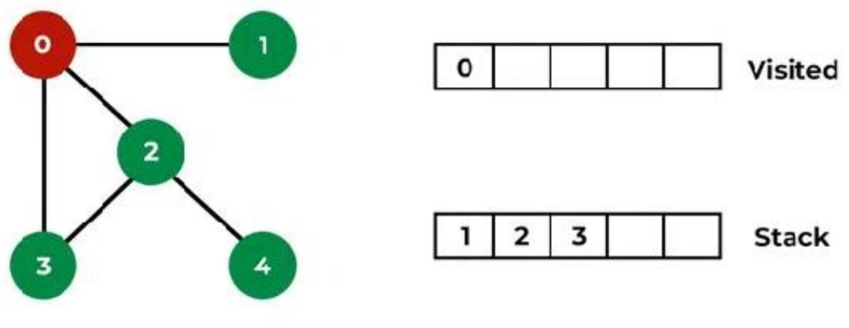
- Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.
- Let us understand the working of **Depth First Search** with the help of the following illustration:

Step1: Initially stack and visited arrays are empty.



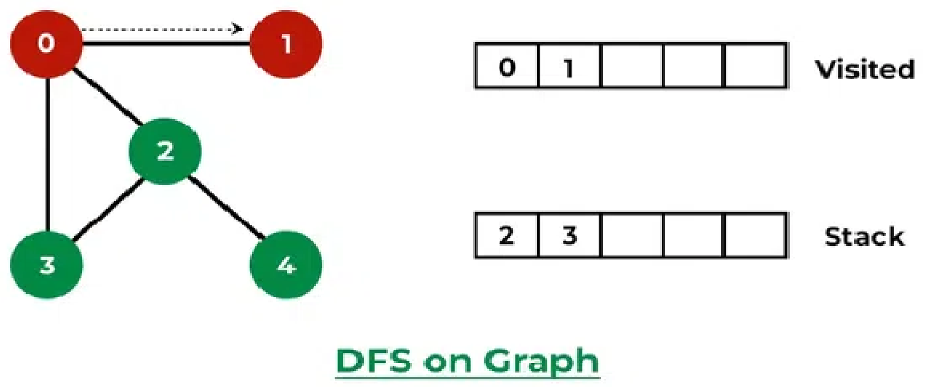
Stack and visited arrays are empty initially.

Step 2: Visit 0 and put its adjacent nodes which are not visited yet into the stack.



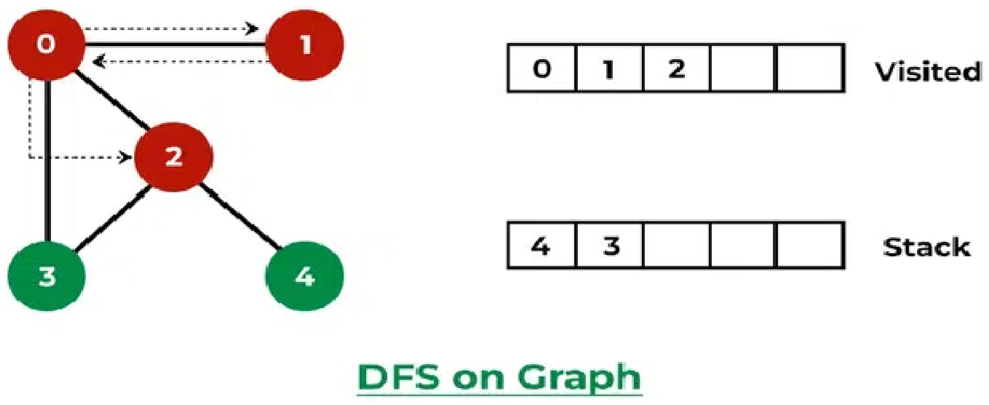
Visit node 0 and put its adjacent nodes (1, 2, 3) into the stack

Step 3: Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



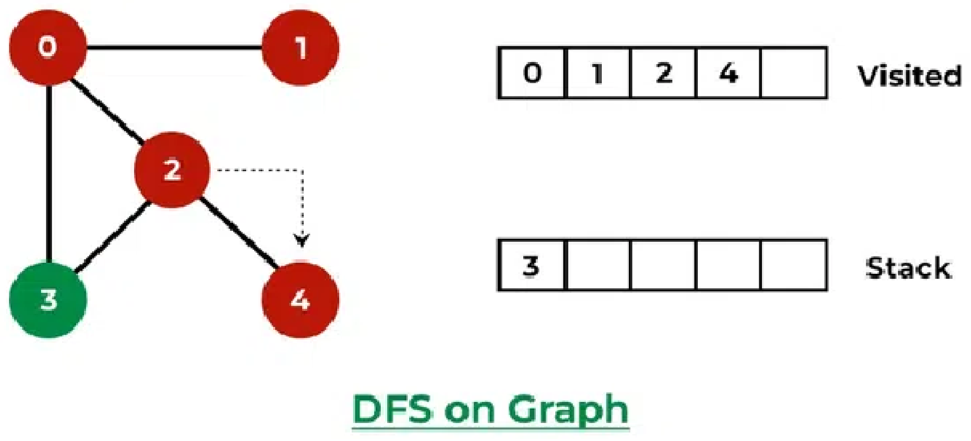
Visit node 1

Step 4: Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (i.e, 3, 4) in the stack.



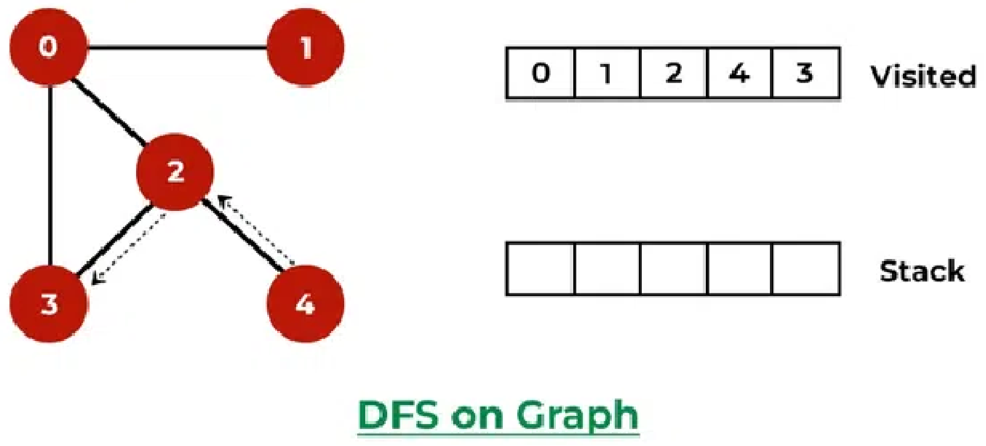
Visit node 2 and put its unvisited adjacent nodes (3, 4) into the stack

Step 5: Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



Visit node 4

Step 6: Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



Visit node 3

Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

ALGORITHM:

```
Step 1: Start.
Step 2: Input the value of N nodes of the graph
Step 3: Create a graph of N nodes using adjacency matrix representation.
Step 4: Print the nodes reachable from the starting node using BFS.
Step 5: Check whether graph is connected or not using DFS.
Step 6: Stop.
```

Program Code:

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

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

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

```
                    }
            }
        }
}

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

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

```
    for(i=1;i<=n;i++)
        visited[i]=0;
    printf("\nEnter the starting vertex: ");
    scanf("%d",&start);

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


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

Output:

Case-1 :

```
Enter the number of vertices in graph:  4
Enter the adjacency matrix:
0 1 0 1
0 0 1 0
0 0 0 1
0 0 0 0
Enter the starting vertex: 1
==>1. BFS: Print all nodes reachable from a given starting node
==>2. DFS: Print all nodes reachable from a given starting node
==>3:Exit
Enter your choice: 1
Nodes reachable from starting vertex 1 are: 2 4 3
```

Case-2:

```
Enter the number of vertices in graph:  4
Enter the adjacency matrix:
0 1 0 1
0 0 1 0
0 0 0 1
0 0 0 0
Enter the starting vertex: 2
==>1. BFS: Print all nodes reachable from a given starting node
==>2. DFS: Print all nodes reachable from a given starting node
==>3:Exit
Enter your choice: 2
Nodes reachable from starting vertex 2 are:
3 4
```

EXPERIMENT-12

Program 12: Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K $\rightarrow$L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

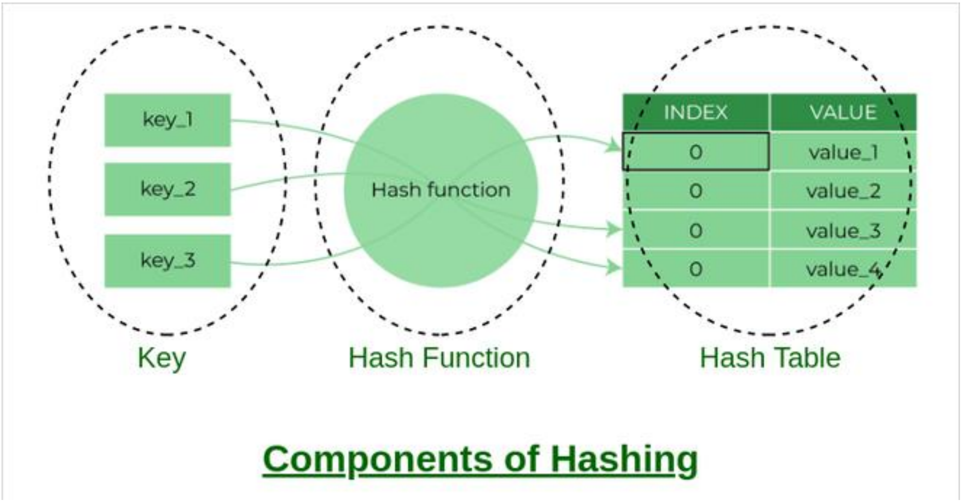
ABOUT THE EXPERIMENT

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

Components of Hashing

There are majorly three components of hashing:

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



How does Hashing work?

Suppose we have a set of strings {"ab", "cd", "efg"} and we would like to store it in a table.

Our main objective here is to search or update the values stored in the table quickly in O(1) time and we are not concerned about the ordering of strings in the table. So the given set of strings can act as a key and the string itself will act as the value of the string but how to store the value corresponding to the key?

- **Step 1:** We know that hash functions (which is some mathematical formula) are used to calculate the hash value which acts as the index of the data structure where the value will be stored.
- **Step 2:** So, let's assign
  "a" = 1,
  "b"=2, .. etc, to all alphabetical characters.
- **Step 3:** Therefore, the numerical value by summation of all characters of the string:
  "ab" = 1 + 2 = 3,
  "cd" = 3 + 4 = 7 ,
  "efg" = 5 + 6 + 7 = 18
- **Step 4:** Now, assume that we have a table of size 7 to store these strings. The hash function that is used here is the sum of the characters in **key mod Table size**. We can compute the location of the string in the array by taking the **sum(string) mod 7**.
- **Step 5:** So we will then store
  "ab" in 3 mod 7 = 3,
  "cd" in 7 mod 7 = 0, and
  "efg" in 18 mod 7 = 4.

0	1	2	3	4	5	6
cd			ab	efg		

Mapping key with indices of array

The above technique enables us to calculate the location of a given string by using a simple hash function and rapidly find the value that is stored in that location. Therefore the idea of hashing seems like a great way to store (key, value) pairs of the data in a table.

Hash Function:

the **hash function** creates a mapping between key and value, this is done through the use of mathematical formulas known as hash functions. The result of the hash function is referred to as a hash value or hash. The hash value is a representation of the original string of characters but usually smaller than the original.

Types of Hash functions:

1. Division Method.
2. Mid Square Method.
3. Folding Method.
4. Multiplication Method

ALGORITHM:

Step 1: Start.

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

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

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

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

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

Step 7: Stop.

Program Code:

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

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

/*Insert key Function*/

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

/*Display Hash Table Function*/

```
void display()
{
       int i;
       if(count == 0)
      {
              printf("\n Hash Table is empty");
                return;
       }
       printf("\n Hash Table contents are:\n ");
       for(i=0; i<m; i++)
```

```
                printf("\n T[%d] --> %d ", i, ht[i]);
}

void main()
{
      int i;
      printf("\n Enter the number of employee  records (N) :  ");
      scanf("%d", &n);
      printf("\n Enter the two digit memory locations (m) for hash table:  ");
      scanf("%d", &m);
      ht = (int *)malloc(m*sizeof(int));
      for(i=0; i<m; i++)
              ht[i] = -1;
      printf("\nEnter the four digit key values (K) for N Employee Records:\n  ");
      for(i=0; i<n; i++)
             scanf("%d", &key[i]);
      for(i=0;i<n;i++)
     {
             if(count == m)
             {
                printf("\n~~~Hash table is full. Cannot insert the record %d key~~~",i+1);
                break;
             }
             insert(key[i]);
   }
        //Displaying Keys inserted into hash table
         display();
}
```

Output:

```
Enter the number of employee  records (N) :   12
Enter the two digit memory locations (m) for hash table:   15
Enter the four digit key values (K) for N Employee Records:
  2541
  1487
  6547
  2165
  1563
  4521
  1354
  6524
  1356
  1564
  5142
  8542


  Hash Table contents are:
 T[0] --> -1
 T[1] --> -1
 T[2] --> 1487
 T[3] --> 1563
 T[4] --> 1354
 T[5] --> 2165
 T[6] --> 2541
 T[7] --> 6547
 T[8] --> 4521
 T[9] --> 1356
 T[10] --> 1564
 T[11] --> 8542
```

```
T[12] --> 5142
T[13] --> -1
T[14] --> 6524
```