



Shree Rahul Education Society's (Regd.)

# **SHREE L. R. TIWARI COLLEGE OF ENGINEERING**

(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)  
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423  
Minority Status (Hindi Linguistic)

**DEPARTMENT OF COMPUTER ENGINEERING**

## **Data Structures Lab. (CSL301)**

**Subject Code: CSL301**

**Conducted by: Mrs. Uma Goradiya**

**Academic Year: 2024-25**

**Semester: III**

**Branch: Computer Engineering**



Shree Rahul Education Society's (Regd.)  
**SHREE L. R. TIWARI**  
**COLLEGE OF ENGINEERING**  
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)  
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423  
Minority Status (Hindi Linguistic)

## DEPARTMENT OF COMPUTER ENGINEERING



### A Laboratory Journal for Data Structures Lab. (CSL301)

**ACADEMIC YEAR: 2024-2025**

**Course Name:** Data Structures Lab.

**Course Code:** CSL301

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

**Semester:** III (Third)

**Roll No. :** \_\_\_\_\_



Shree Rahul Education Society's (Regd.)

# SHREE L. R. TIWARI COLLEGE OF ENGINEERING

(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)  
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423  
Minority Status (Hindi Linguistic)

## DEPARTMENT OF COMPUTER ENGINEERING

### Institution's VISION AND MISSION

<b>Vision</b>	To be a world class institute and a front runner in educational and socioeconomic development of the nation by providing high quality technical education to students from all sections of society.
<b>Mission</b>	To provide superior learning experiences in a caring and conducive environment so as to empower students to be successful in life & contribute positively to society.
<b>Quality Policy</b>	We, at SHREE L. R. TIWARI COLLEGE OF ENGINEERING, shall dedicate and strive hard to continuously achieve academic excellence in the field of Engineering and to produce the most competent Engineers through objective & innovative teaching methods, consistent updating of facilities, welfare & quality improvement of the faculty & a system of continual process improvement.

### Computer Engineering Department's

<b>Vision</b>	<b>To be a world class institute and a front runner in educational and socioeconomic development of the nation by providing high quality technical education to students from all sections of society.</b>
<b>Mission</b>	<b>1) To develop - technical, analytical, theoretical competencies, managerial skills and practical exposure. 2) Over all development of students, faculty and staff by providing encouraging environment and infrastructure for learning, skill development and research. 3) To strengthen - versatility, adaptability and chase for excellence amongst students with highest ethical values as their core strength.</b>
<b>Program Educational Objectives</b>	<b>1) Be employed in industry, government, or entrepreneurial endeavors to demonstrate professional advancement through significant technical achievements and expanded leadership responsibility by exhibiting ethical attitude and good communication skills. 2) Demonstrate the ability to work effectively as a team member and/or leader in an ever-changing professional environment. 3) To pursue higher studies, engage in professional development, research and entrepreneurship and adapt to emerging technologies.</b>



Shree Rahul Education Society's (Regd.)

# SHREE L. R. TIWARI COLLEGE OF ENGINEERING

(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)  
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423  
Minority Status (Hindi Linguistic)

## DEPARTMENT OF COMPUTER ENGINEERING

### Certificate

*This is to certify that Mr. /Ms. \_\_\_\_\_*

*Class \_\_\_\_\_ Roll No. \_\_\_\_\_ Exam Seat No. \_\_\_\_\_*

*of Third Semester of Degree in Computer Engineering has completed the required number of Practical's / Term Work / Session in the subject **Data Structures Lab** from the Department of Computer Engineering during the academic year of 2024 -2025 as prescribed in the curriculum.*

Course in-Charge

Head of the Department

**Date:**

Seal of  
Institution



Shree Rahul Education Society's (Regd.)

# **SHREE L. R. TIWARI COLLEGE OF ENGINEERING**

(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)  
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423  
Minority Status (Hindi Linguistic)

## **INSTRUCTION FOR STUDENTS**

Students shall read the points given below for understanding the theoretical concepts and practical applications.

- 1) Listen carefully to the lecture given by teacher about importance of subject, curriculum philosophy learning structure, skills to be developed, information about equipment, instruments, procedure, method of continuous assessment, tentative plan of work in laboratory and total amount of work to be done in a semester.
- 2) Student shall undergo study visit of the laboratory for types of equipment, instruments, software to be used, before performing experiments.
- 3) Read the write up of each experiment to be performed, a day in advance.
- 4) Organize the work in the group and make a record of all observations.
- 5) Understand the purpose of experiment and its practical implications.
- 6) Write the answers of the questions allotted by the teacher during practical hours if possible or afterwards, but immediately.
- 7) Student should not hesitate to ask any difficulty faced during conduct of practical/exercise.
- 8) The student shall study all the questions given in the laboratory manual and practice to write the answers to these questions.
- 9) Student shall develop maintenance skills as expected by the industries.
- 10) Student should develop the habit of pocket discussion/group discussion related to the experiments/exercises so that exchanges of knowledge/skills could take place.
- 11) Student shall attempt to develop related hands-on-skills and gain confidence.
- 12) Student shall focus on development of skills rather than theoretical or codified knowledge.
- 13) Student shall visit the nearby workshops, workstation, industries, laboratories, technical exhibitions, trade fair etc. even not included in the Lab manual. In short, students should have exposure to the area of work right in the student hood.
- 14) Student shall insist for the completion of recommended laboratory work, industrial visits, answers to the given questions, etc.
- 15) Student shall develop the habit of evolving more ideas, innovations, skills etc. those included in the scope of the manual.
- 16) Student shall refer technical magazines, proceedings of the seminars, refer websites related to the scope of the subjects and update his knowledge and skills.
- 17) Student should develop the habit of not to depend totally on teachers but to develop self-learning techniques.
- 18) Student should develop the habit to react with the teacher without hesitation with respect to the academics involved.
- 19) Student should develop habit to submit the practical's, exercise continuously and progressively on the scheduled dates and should get the assessment done.
- 20) Student should be well prepared while submitting the write up of the exercise. This will develop the continuity of the studies and he/she will not be over loaded at the end of the term.



Shree Rahul Education Society's (Regd.)

# **SHREE L. R. TIWARI COLLEGE OF ENGINEERING**

(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)  
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423  
Minority Status (Hindi Linguistic)

## **GUIDELINES FOR TEACHERS**

Teachers shall discuss the following points with students before start of practical's of the subject.

- 1) Learning Overview: To develop better understanding of importance of the subject. To know related skills to be developed such as Intellectual skills and Motor skills.
- 2) Learning Structure: In this, topic and sub topics are organized in systematic way so that ultimate purpose of learning the subject is achieved. This is arranged in the form of fact, concept, principle, procedure, application and problem.
- 3) Know you're Laboratory Work: To understand the layout of laboratory, specifications of equipment/Instruments/Materials, procedure, working in groups, planning time etc. Also to know total amount of work to be done in the laboratory.
- 4) Teaching shall ensure that required equipment's are in working condition before start of experiment, also keep operating instruction manual available.
- 5) Explain prior concepts to the students before starting of each experiment.
- 6) Involve students' activity at the time of conduct of each experiment.
- 7) While taking reading/observation each student shall be given a chance to perform or observe the experiment.
- 8) If the experimental set up has variations in the specifications of the equipment, the teachers are advised to make the necessary changes, wherever needed.
- 9) Teacher shall assess the performance of students continuously as per norms prescribed by university of Mumbai and guidelines provided by IQAC.
- 10) Teacher should ensure that the respective skills and competencies are developed in the students after the completion of the practical exercise.
- 11) Teacher is expected to share the skills and competencies are developed in the students.
- 12) Teacher may provide additional knowledge and skills to the students even though not covered in the manual but are expected from the students by the industries.
- 13) Teachers shall ensure that industrial visits if recommended in the manual are covered.
- 14) Teacher may suggest the students to refer additional related literature of the Technical papers/Reference books/Seminar proceedings, etc.
- 15) During assessment teacher is expected to ask questions to the students to tap their achievements regarding related knowledge and skills so that students can prepare while submitting record of the practical's. Focus should be given on development of enlisted skills rather than theoretical /codified knowledge.
- 16) Teacher should enlist the skills to be developed in the students that are expected by the industry.
- 17) Teacher should organize Group discussions /brain storming sessions / Seminars to facilitate the exchange of knowledge amongst the students.
- 18) Teacher should ensure that revised assessment norms are followed simultaneously and progressively.
- 19) Teacher should give more focus on hands on skills and should actually share the same.
- 20) Teacher shall also refer to the circulars related to practical's supervise and assessment for additional guidelines.





**DEPARTMENT OF COMPUTER ENGINEERING**

**DATA STRUCTURES LAB**  
**List of Experiments (A)**

Sr. no.	Name of Experiments	CO Covered
1	Implement Stack using ADT using array.	CO1
2	Convert an Infix expression to Postfix expression using stack ADT.	CO1
3	Evaluate Postfix Expression using stack ADT.	CO1
4	Implement Linear Queue ADT using array.	CO1
5	Implement Circular Queue ADT using array.	CO1
6	Implement Singly Linked List ADT.	CO2
7	Implement Doubly Linked List ADT.	CO1
8	Implement Stack ADT using Linked List	CO3
9	Implement Binary Tree ADT using Linked List.	CO3
10	Implement Binary Search	CO4
11	Implement Graph Traversal (DFS)	CO4
12	Implement Circular Linked List	CO4
13	Implement Binary Search Tree.	CO4



## DEPARTMENT OF COMPUTER ENGINEERING

### Program Outcome (POs & PSOs)

Program Outcomes are the skills and knowledge which the students have at the time of graduation. This will indicate what student can do from subject-wise knowledge acquired during the program .

PO	Graduate Attributes	Description of the Program outcome as defined by the NBA
PO-1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO-2	Problem analysis	Identify, formulate, review research literature, and analyses complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO-3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO-4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO-5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO-6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO-7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO-8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO-9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO-10	Communication	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO-11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO-	Life-long	



12	learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
<b>Program Specific Outcomes (PSOs) defined by the programme. Baseline-Rational Unified Process(RUP)</b>		
PSO-1	Computing solution to solve real life problem	The graduate must be able to develop, deploy, test and maintain the software or computing hardware solutions to solve real life problems using state of the art technologies, standards, tools and programming paradigms.
PSO-2	Computer Engineering knowledge and skills	The graduate should be able to adapt Computer Engineering knowledge and skills to create career paths in industries or business organizations or institutes of repute.

**Student's Signature**



## Course Objectives and Outcomes

**Academic Year:** 2024-25

**Class:** SE

**Course Code:** CSL301

**Program:** Computer Engineering

**Course Name:** Data Structures Lab.

**Department:** Computer Engineering

**Sem.:** III

**Faculty:** Ms Uma Goradiya

### Course Objectives:

Sr. No.	Statement
1	To implement basic data structures such as arrays, linked lists, stacks and queues.
2	Solve problem involving graphs, and trees.
3	To develop application using data structure algorithms
4	Compute the complexity of various algorithms.

### Course Outcomes:

CO's No.	Abbre.	Statement
CSL301.1	CO1	Students will be able to implement linear data structures & be able to handle operations like insertion, deletion, searching and traversing on them.
CSL301.2	CO2	Students will be able to implement nonlinear data structures & be able to handle operations like insertion, deletion, searching and traversing on them.
CSL301.3	CO3	Students will be able to choose appropriate data structure and apply it in various problems
CSL301.4	CO4	Students will be able to select appropriate searching techniques for given problems.

### Course Prerequisite:

Sr. No.	Pre-requisite
1	Basic of C programming .

### Teaching and Examination Scheme:

Teaching Scheme (Hrs)			Credits Assigned				Examination Scheme								
Theory	Pract	Tut	Theory	TW/Pract	Tut	Total	Theory			T W	Ora l	Oral & Pract	Tota l		
-	2	-	-	1	-	1	Internal Assessment							End Sem. Exam	Exam Duration ( in Hrs)
							Test 1	Test 2	Avg.						
							-	-	-	-	25	--	25	50	

**Term Work** (Total 25 Marks) = (Experiments: 15 mark + Assignments: 05 mark + Attendance: 05 marks (TH+PR)).

**Oral exam** will be based on the above mentioned experiment list and CSC303: Data Structures system

# EXPERIMENT :- 1

**Aim:** To study STACK Data structure.

**Title:** To implement stack using array.

**Objective:** To design and implement stack data structures.

**Outcome:** Students will be able to implement stack data structure.

**Theory:**

- Stack is a linear data structure which follows a particular order in which operations are performed.
- The order may be LIFO (Last In First Out) or FILO (First In Last Out).
- The data structure is like stack of dishes. In stack of dishes, the dish which was put in last (on top) will come out first and the dish which was put first (at bottom) will come out last. Same thing is implemented on data.
- Stack has only one end call 'Top' which points to the top element of the stack. Insertion and deletion of item is done on top. If stack is empty then Top points to -1.
- Mainly the following three basic operations are performed in stack:
  1. isEmpty: Returns true if stack is empty else returns false.
  2. Push: Adds an element at the top of Stack. If stack is full then it is called stack-overflow condition.
  3. Pop: Deletes top element of the stack. The items are popped in reverse order in which they were pushed. If stack is empty then it is called as stack-underflow condition.
- Applications of Stack:
  1. Conversion of infix to prefix/postfix
  2. Redo-Undo features at many places like editors, photoshop
  3. Forward and backward features in web browsers
  4. Recursion

**Algorithm :**

Here, STACK is an array with maximum size SIZE. TOP points to the top most element of stack STACK.

- Algorithm to check if stack is empty or not:
  1. If (TOP==-1) Then
  2. Return 1
  3. Else
  4. Return 0
- Algorithm to check if stack is full or not:
  1. If (TOP==MAX-1) Then
  2. Return 1
  3. Else
  4. Return 0
- Algorithm to push ():
  1. If (TOP==MAX-1) Then [Check Stack Overflow condition]
  2. Print "Stack Overflow"
  3. Else
  4. TOP=TOP+1 [Increment TOP by one]
  5. STACK[TOP]=ITEM [Assign ITEM to TOP of STACK]
  6. EXIT

- Algorithm to pop ():
  1. If (TOP== -1) Then [Check stack underflow condition]
  2. Print "Stack is Empty"
  3. Else
  4. TOP=TOP-1
  5. EXIT
  
- Algorithm to display ():
  1. Loop: From i=TOP till i!= -1 and i++
  2. Print STACK[i];
  3. EXIT
  
- Algorithm to main():
  1. TOP=-1
  2. FLAG=0
  3. Input CHOICE
  4. If (CHOICE==1) Then
  5. Input ITEM
  6. Push(ITEM)
  7. Else If (CHOICE==2) Then
  8. Pop(ITEM)
  9. Else If (CHOICE==3) Then
  10. Display()
  11. Else If (CHOICE==4) Then
  12. FLAG=1
  13. If (FLAG==0) Then
  14. Goto STEP 3
  15. EXIT

**Program:**

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 4

int stack_arr[MAX];

int top = -1;

int isEmpty()
{
    if(top== -1)
        return 1;
    else
        return 0;
}

int peek()
```

```

{
    if(isEmpty())
    {
        printf("Stack underflow\n");
        exit(1);
    }
    return stack_arr[top];
}

```

```

void print()
{
    int i;
    if(top==-1)
    {
        printf("stack underflow\n");
    }
    for(i=top;i>=0;i--)
        printf("%d\n", stack_arr[i]);
    printf("\n");
}

```

```

int pop()
{
    int value;
    if(isEmpty())
    {
        printf("Stack underflow");
        exit(1);
    }
    value = stack_arr[top];
    top = top - 1;
    return value;
}

```

```
int isFull()
{
    if(top==MAX-1)
        return 1;
    else
        return 0;
}
```

```
void push(int data)
{
    if(isFull())
    {
        printf("Stack Overflow");
    }
    top = top + 1;
    stack_arr[top] = data;
}
```

```
int main()
{
    int choice, data;
    while(1)
    {
        printf("\n");
        printf("1.Push\n");
        printf("2.Pop\n");
        printf("3.Print the top element\n");
        printf("4.Print all the elements of the stack\n");
        printf("5.Quit\n");
        printf("Please enter your choice:\n");
        scanf("%d",&choice);

        switch(choice)
```



```

{
    case 1:
        printf("Enter the element to be pushed:");
        scanf("%d",&data);
        push(data);
        break;

    case 2:
        data = pop();
        printf("Deleted element is %d\n",data);
        break;

    case 3:
        printf("The topmost element of the stack is %d\n",peek());
        break;

    case 4:
        print();
        break;

    case 5:
        exit(1);

    default:
        printf("Wrong choice\n");
}
}
return 0;
}

```

### **Output:**

1.Push

2.Pop

- 3.Print the top element
- 4.Print all the elements of the stack
- 5.Quit

Please enter your choice: 1

Enter the element to be pushed:12

- 1.Push
- 2.Pop
- 3.Print the top element
- 4.Print all the elements of the stack
- 5.Quit

Please enter your choice: 1

Enter the element to be pushed:23

- 1.Push
- 2.Pop
- 3.Print the top element
- 4.Print all the elements of the stack
- 5.Quit

Please enter your choice: 1

Enter the element to be pushed:34

- 1.Push
- 2.Pop
- 3.Print the top element
- 4.Print all the elements of the stack
- 5.Quit

Please enter your choice: 1

Enter the element to be pushed:56

- 1.Push
- 2.Pop
- 3.Print the top element
- 4.Print all the elements of the stack

5.Quit

Please enter your choice: 4

56

34

23

12

1.Push

2.Pop

3.Print the top element

4.Print all the elements of the stack

5.Quit

Please enter your choice: 3

The topmost element of the stack is 56

1.Push

2.Pop

3.Print the top element

4.Print all the elements of the stack

5.Quit

Please enter your choice:

5

.....Program finished with exit code 1

**Conclusion:** Thus we have implemented stack using array.

**Viva Questions:**

1. **What is Stack and where it can be used?**
2. **What is the difference between a PUSH and a POP?**
3. **What is the advantage of the heap over a stack?**
4. **How does variable declaration affect memory allocation?**
5. **What is LIFO?**
6. **What is a stack?**
7. **Which data structures is applied when dealing with a recursive function?**

## EXPERIMENT :- 2

**Aim:** To study application of STACK

**Title:** To convert infix expression to postfix.

**Objective:** To implement application of stack.

**Outcome:** Students will be able to implement infix to postfix.

**Theory:**

**Infix expression:**

The expression of the form a op b. When an operator is in-between every pair of operands.

**Postfix expression:**

The expression of the form a b op. When an operator is followed for every pair of operands.

### Why postfix representation of the expression?

The compiler scans the expression either from left to right or from right to left.

Consider the below expression: a op1 b op2 c op3 d

If op1 = +, op2 = \*, op3 = +

The compiler first scans the expression to evaluate the expression b \* c, then again scan the expression to add a to it. The result is then added to d after another scan.

The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix(or prefix) form before evaluation.

The corresponding expression in postfix form is:abc\*+d+. The postfix expressions can be evaluated easily using a stack.

### Algorithm:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
  - .....3.1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty), push it.
  - .....3.2 Else, Pop the operator from the stack until the precedence of the scanned operator is less-equal to the precedence of the operator residing on the top of the stack. Push the scanned operator to the stack.
4. If the scanned character is an '(', push it to the stack.
5. If the scanned character is an ')', pop and output from the stack until an '(' is encountered.
6. Repeat steps 2-6 until infix expression is scanned.
7. Pop and output from the stack until it is not empty.

### Program:

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

```
#define MAX 100
```

```
char stack[MAX];
```

*Dept. of Computer Engineering, Shree L. R. Tiwari College of Engineering, Thane-401107.*

```

char infix[MAX], postfix[MAX];
int top = -1;

void push(char);
char pop();
int isEmpty();
void inToPost();
int space(char);
void print();
int precedence(char);
int isOperand(char n)

int space(char c)
{
    //if symbol is a blank space or a tab
    if(c == ' ' || c == '\t')
        return 1;
    else
        return 0;
}

int precedence(char symbol)
{
    switch(symbol)
    {
        //higher value means greater precedence

        case '^':
            return 3;
        case '/':
        case '*':
            return 2;
        case '+':
        case '-':
            return 1;
        default:
            return 0;
    }
}

void print()
{
    int i = 0;
    printf("The equivalent postfix expression is:");
    while(postfix[i])
    {
        printf("%c", postfix[i++]);
    }
    printf("\n");
}

```

```

void push(char c)
{
    if(top == MAX - 1)
    {
        printf("Stack Overflow");
    }
    top = top + 1;
    stack[top] = c;
}

char pop()
{
    char c;
    if(top == -1)
    {
        printf("Stack underflow");
        exit(1);
    }
    c = stack[top];
    top = top - 1;
    return c;
}

int isEmpty()
{
    if(top == -1)
        return 1;
    else
        return 0;
}

int isOperand(char n)
{
    if((n>='a'&& n<='z')||(n>='A'&& n<='Z')||(n>=0 && n<=9))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void inToPost ()
{
    int i, j = 0;
    char next;
    char symbol;
    for(i=0; i<strlen(infix); i++)
    {
        symbol = infix[i];

```



```

//if symbol is not a whitespace or tab
if(!space(symbol))
{
    switch(symbol)
    {
        case '(':
            push(symbol);
            break;

        case ')':
            while((next=pop()) != '(')
                postfix[j++] = next;
            break;

        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while(!isEmpty() && precedence(stack[top]) >= precedence(symbol))
                postfix[j++] = pop();
            push(symbol);
            break;

        default: //if the symbol is an operand
            if(isOperand(symbol)==1)
                postfix[j++] = symbol;
    }
}

while(!isEmpty())
    postfix[j++] = pop();
postfix[j] = '\0';
}

```

```

int main()
{
    printf("Enter the infix expression: ");
    gets(infix);

    inToPost();
    print();
    return 0;
}

```

**OUTPUT:**

Enter the infix expression:  $7+(9-5)*2$   
The equivalent postfix expression is:  $795-2*+$

**Conclusion:** Thus, we converted infix expression to postfix using stack.

**Viva Questions:**

1. **What is a postfix expression?**
2. Convert the expression  $((A + B) * C - (D - E) ^ (F + G))$  to equivalent Prefix and Postfix notations.
3. What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

## EXPERIMENT :- 3

**Aim:** To study application of STACK

**Title:** To evaluate postfix expression using stack.

**Objective:** To design and implement evaluation of postfix..

**Outcome:** Students will be able to implement Evaluation of postfix.

**Theory:**

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix.

**Example:**

Let the given expression be “2 3 1 \* + 9 -”. We scan all elements one by one.

- 1) Scan ‘2’, it’s a number, so push it to stack. Stack contains ‘2’
- 2) Scan ‘3’, again a number, push it to stack, stack now contains ‘2 3’ (from bottom to top)
- 3) Scan ‘1’, again a number, push it to stack, stack now contains ‘2 3 1’
- 4) Scan ‘\*’, it’s an operator, pop two operands from stack, apply the \* operator on operands, we get  $3*1$  which results in 3. We push the result ‘3’ to stack. Stack now becomes ‘2 3’.
- 5) Scan ‘+’, it’s an operator, pop two operands from stack, apply the + operator on operands, we get  $3+2$  which results in 5. We push the result ‘5’ to stack. Stack now becomes ‘5’.
- 6) Scan ‘9’, it’s a number, we push it to the stack. Stack now becomes ‘5 9’.
- 7) Scan ‘-’, it’s an operator, pop two operands from stack, apply the – operator on operands, we get  $5-9$  which results in -4. We push the result ‘-4’ to stack. Stack now becomes ‘-4’.
- 8) There are no more elements to scan, we return the top element from stack (which is the only element left in stack).

**Algorithm:**

- 1) Create a stack to store operands (or values).
- 2) Scan the given expression and do following for every scanned element.
  - a) If the element is a number, push it into the stack
  - b) If the element is an operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
- 3) When the expression is ended, the number in the stack is the final Answer

**Program:**

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

```
#define MAX 100
```

```
int stack[MAX];
char postfix[MAX];
int top = -1;
```

```

void push(int);
int pop();
int post_eval();

void push(int val)
{
    if(top == MAX - 1)
    {
        printf("Stack Overflow");
    }
    top = top + 1;
    stack[top] = val;
}

int pop()
{
    int val;
    if(top == -1)
    {
        printf("Stack underflow");
        exit(1);
    }
    val = stack[top];
    top = top - 1;
    return val;
}

int post_eval()
{
    int i,a,b;
    for(i=0;i<strlen(postfix);i++)
    {
        //if the symbol is an operand
        if(postfix[i] >= '0' && postfix[i] <= '9')
        {
            push(postfix[i] - '0');
        }
        else
        {
            //pop the topmost symbols
            a = pop();
            b = pop();
            switch(postfix[i])
            {
                case '+':
                    push(b+a);
                    break;

```

```

        case '-':
            push(b-a);
            break;

        case '*':
            push(b*a);
            break;

        case '/':
            push(b/a);
            break;

        case '^':
            push(pow(b,a));
            break;
    }
}
return pop();
}

int main()
{
    int result;
    printf("Enter the postfix expression: ");
    gets(postfix);

    result = post_eval();
    printf("The result obtained after postfix evaluation is: ");
    printf("%d\n",result);
    return 0;
}

```

### OUTPUT:

Enter the postfix expression: 795-2\*+  
 The result obtained after postfix evaluation is: 15

**Conclusion:** Thus, we evaluated postfix expression using stack.

## EXPERIMENT :- 4

**Aim:** To study Queue Data Structure

**Title:** To implement queue using array.

**Objective:** To design and implement Queue data structures.

**Outcome:** Students will be able to implement Queue data structure.

**Theory:**

Like Stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

The difference between stacks and queues is in removing. In a stack, we remove the item the most recently added; in a queue, we remove the item the least recently added.

**Operations on Queue:**

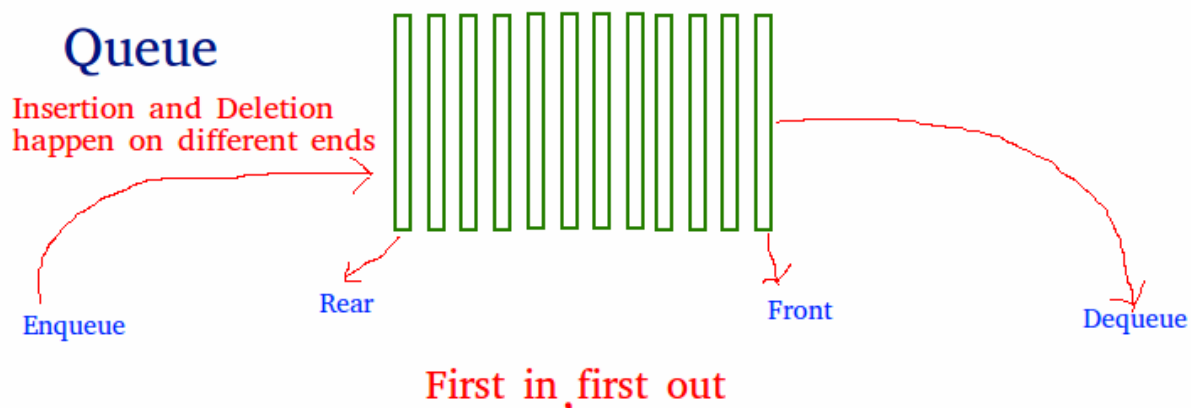
Mainly the following four basic operations are performed on queue:

**Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

**Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

**Front:** Get the front item from queue.

**Rear:** Get the last item from queue.



**Applications of Queue:**

Queue is used when things don't have to be processed immediately, but have to be processed in First In First Out order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

- 1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
- 2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Array implementation Of Queue

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front. If we simply increment front and rear indices, then there may be problems, front may reach end of the array. The solution to this problem is to increase front and rear in circular manner.

**Algorithm:**

*Dept. of Computer Engineering, Shree L. R. Tiwari College of Engineering, Thane-401107.*



Here, Q is an array with maximum size SIZE. FRONT points to the front most element of Queue Q.

- Algorithm to check if queue is empty or not:
  1. If (FRONT == -1) Then
  2.     Return 1
  3. Else
  4.     Return 0
  
- Algorithm to check if queue is full or not:
  1. If (FRONT == SIZE-1) Then
  2.     Return 1
  3. Else
  4.     Return 0
  
- Algorithm for ENQUEUE (insert element in Queue):
  1. If (REAR = SIZE-1) Then //Queue is full
  2.     print "Queue is full"
  3.     Exit
  4. Else
  5.     If (FRONT = -1) Then //Queue is empty
  6.         FRONT = FRONT + 1
  7.     Else
  8.         REAR = REAR + 1 // increment REAR
  9.         Q[REAR] = ITEM
  10. End if
  11. Stop
  
- Algorithm for DEQUEUE (delete element from Queue)
  1. If (FRONT == -1) then
  2.     print "Queue is empty"
  3.     Exit
  4. Else
  5.     If (FRONT = REAR)
  6.         REAR = -1
  7.         FRONT = -1
  8.     Else
  9.         FRONT = FRONT + 1
  10.     End if
  11. End if
  12. End if
  13. Stop
  
- Algorithm to display elements:
  1. Loop: From i=FRONT till i!= REAR and i++
  2.     Print Q[i];
  3. EXIT
  
- Algorithm for main():
  1. REAR=FRONT=-1
  2. FLAG=0
  3. Input CHOICE

4. If (CHOICE==1) Then
5. Input ITEM
6. ENQUEUE(ITEM)
7. Else If (CHOICE==2) Then
8. DEQUEUE(ITEM)
9. Else If (CHOICE==3) Then
10. Display()
11. Else If (CHOICE==4) Then
12. FLAG=1
- 13.If (FLAG==0) Then
- 14.Goto STEP 3
- 15.EXIT

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 4
```

```
int queue[MAX];
int front = -1;
int rear = -1;
```

```
void enqueue(int);
int dequeue();
int peek();
void print();
int isFull();
int isEmpty();
```

```
void enqueue(int data)
{
    if(isFull())
    {
        printf("Queue overflow\n");
        exit(1);
    }
    if(front == -1)
        front = 0;
    rear = rear + 1;
    queue[rear] = data;
}
```

```
int dequeue()
{
    int data;
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    data = queue[front];
    front = front + 1;
    return data;
}
```

```

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

int isFull()
{
    if(rear == MAX-1)
        return 1;
    else
        return 0;
}

int peek()
{
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    return queue[front];
}

void print()
{
    int i;
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    printf("Queue:\n");
    for(i=front; i<=rear; i++)
        printf("%d",queue[i]);
    printf("\n");
}

int main()
{
    int choice, data;
    while(1)
    {
        printf("\n");
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Print the first element\n");
        printf("4.Print all the elements of the queue\n");
        printf("5.Quit\n");
        printf("Please enter your choice:\n");
        scanf("%d",&choice);
    }
}

```

```

switch(choice)
{
    case 1:
        printf("Enter the element to be added in the queue:");
        scanf("%d",&data);
        enqueue(data);
        break;

    case 2:
        data = dequeue();
        printf("Deleted element is %d\n",data);
        break;

    case 3:
        printf("The first element of the queue is %d\n",peek());
        break;

    case 4:
        print();
        break;

    case 5:
        exit(1);

    default:
        printf("Wrong choice\n");
}
}
return 0;
}

```

### Output:

```

1.Insert
2.Delete
3.Print the first element
4.Print all the elements of the queue
5.Quit
Please enter your choice:
1
Enter the element to be added in the queue:10

```

```

1.Insert
2.Delete
3.Print the first element
4.Print all the elements of the queue
5.Quit
Please enter your choice:

```

1

Enter the element to be added in the queue:15

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

1

Enter the element to be added in the queue:45

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

3

The first element of the queue is 10

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

4

Queue:

101545

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

2

Deleted element is 10

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

2

Deleted element is 15

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

4

Queue:

45

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

2

Deleted element is 45

1.Insert

2.Delete

3.Print the first element

4.Print all the elements of the queue

5.Quit

Please enter your choice:

4

Queue Underflow

**Conclusion:** Thus, we implemented queue using array.

#### **Viva Questions:**

1. **What is a queue?**
2. **What is a dequeue?**
3. **What is a Queue, how it is different from stack and how is it implemented?**
4. **Which data structures are used for BFS and DFS of a graph?**
5. **How to implement a stack using queue?**



## EXPERIMENT :- 5

**Aim:** To overcome drawback of linear Queue

**Title:** To implement circular queue using array.

**Objective:** To design and implement Circular Queue data structures.

**Outcome:** Students will be able to implement Circular Queue data structure.

**Theory:**

Like Stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

**Drawback of queue:**

By the definition of a queue, when we add an element in Queue, rear pointer is increased by 1 whereas, when we remove an element front pointer is increased by 1. But in array implementation of queue this may cause problem as follows:

Consider operations performed on a Queue (with SIZE = 5) as follows:

1. Initially empty Queue is there so, front = -1 and rear = -1

--	--	--	--	--

2. When we add 5 elements to queue, the state of the queue becomes as follows with front = 0 and rear = 4

10	20	30	40	50
----	----	----	----	----

3. Now suppose we delete 2 elements from Queue then, the state of the Queue becomes as follows, with front = 1 and rear = 4

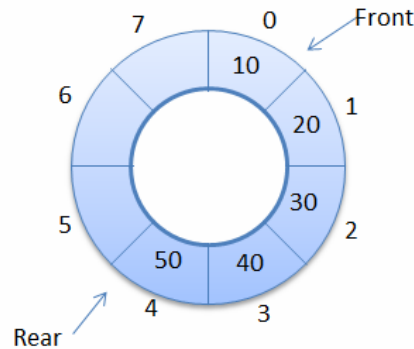
		30	40	50
--	--	----	----	----

4. Now, actually we have deleted 2 elements from queue so, there should be space for another 2 elements in the queue, but as rear pointer is pointing at last position and Queue overflow condition ( $\text{Rear} == \text{SIZE}-1$ ) is true, we can't insert new element in the queue even if it has an empty space.

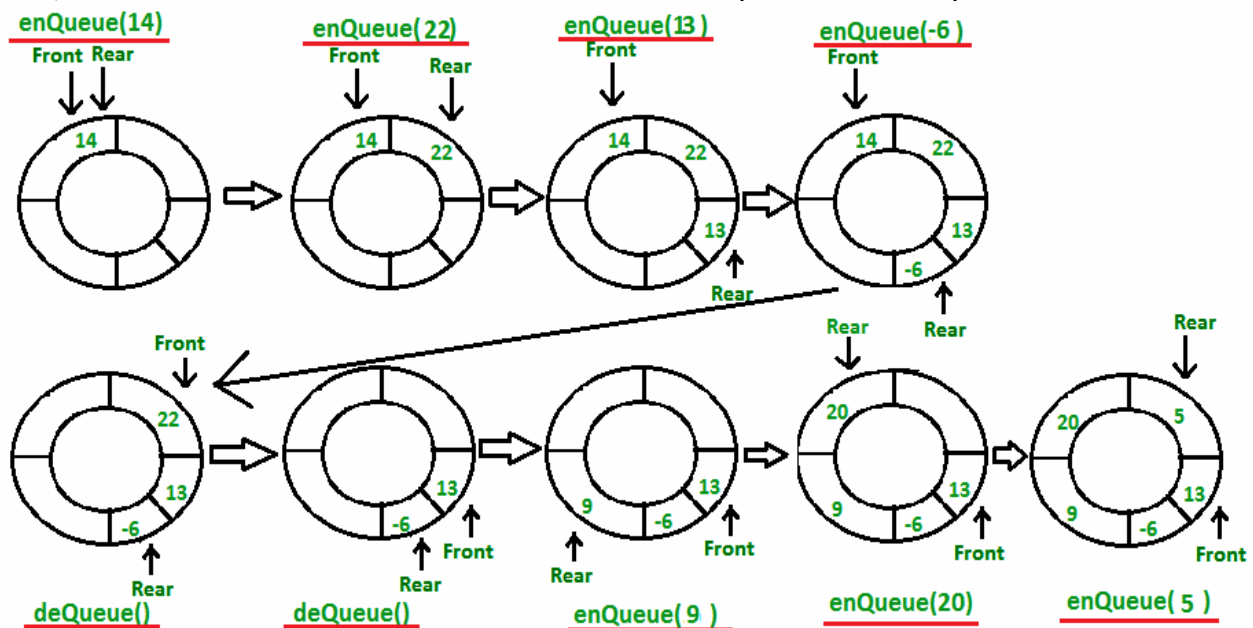
To overcome this problem there is another variation of queue called CIRCULAR QUEUE.

**Circular Queue:**

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'.



In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue.



### Operations on Circular Queue:

**Front:** Get the front item from queue.

**Rear:** Get the last item from 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 Rear position.

**deQueue():** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

### Applications:

1. Memory Management: The unused memory locations in the case of ordinary queues can be utilized in circular queues.
2. Traffic system: In computer controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.
3. CPU Scheduling: Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

### Algorithm:

Here, CQ is an array with maximum size SIZE. FRONT points to the front most element of Circular QueueCQ.

- Algorithm to check if Circular Queue is empty or not:  
5. If (RAER== -1) Then

6. Return 1
  7. Else
  8. Return 0
- Algorithm to check if Circular Queue is full or not:
    5. If (FRONT==(REAR+1) %SIZE) Then
    6. Return 1
    7. Else
    8. Return 0
  - Algorithm for ENQUEUE (insert element in Circular Queue):
    1. If (FRONT==(REAR+1) %SIZE) Then //Circular Queue is full
    2. print "Circular Queue is full"
    3. Exit
    4. Else
    5. If (FRONT = -1) Then //Circular Queue is empty
    6. FRONT=FRONT+1
    7. Else
    8. REAR==(REAR+1) %SIZE // increment REAR
    9. CQ[REAR] = ITEM
    10. End if
    11. Stop
  - Algorithm for DEQUEUE (delete element from Circular Queue)
    1. If (FRONT=-1) then
    2. print "Circular Queue is empty"
    3. Exit
    4. Else
    5. If (FRONT = REAR)
    7. REAR = -1
    8. FRONT = -1
    9. Else
    10. FRONT==(FRONT+1) %SIZE
    11. End if
    12. End if
    13. Stop
  - Algorithm to display elements:
    4. Loop: From i=FRONT till i!= REAR and i=(i+1) %SIZE
    5. Print CQ[i];
    6. EXIT Loop
    7. Print CQ[REAR];
    8. EXIT
  - Algorithm for main():
    13. REAR=FRONT=-1
    14. FLAG=0
    15. Input CHOICE
    16. If (CHOICE==1) Then
    17. Input ITEM

```
18. ENQUEUE(ITEM)
19. Else If (CHOICE==2) Then
20. DEQUEUE(ITEM)
21. Else If (CHOICE==3) Then
22. Display()
23. Else If (CHOICE==4) Then
24. FLAG=1
13.If (FLAG==0) Then
14.Goto STEP 3
15.EXIT
```

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 4
```

```
int queue[MAX];
int front = -1;
int rear = -1;
```

```
void enqueue(int);
int dequeue();
int peek();
void print();
int isFull();
int isEmpty();
```

```
void enqueue(int data)
{
    if(isFull())
    {
        printf("Queue overflow\n");
        exit(1);
    }
    if(front == -1)
        front = 0;
    rear = rear + 1;
    queue[rear] = data;
}
```

```
int dequeue()
{
    int data;
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    data = queue[front];
    front = front + 1;
    return data;
}
```

```

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

int isFull()
{
    if(rear == MAX-1)
        return 1;
    else
        return 0;
}

int peek()
{
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    return queue[front];
}

void print()
{
    int i;
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    printf("Queue:\n");
    for(i=front; i<=rear; i++)
        printf("%d",queue[i]);
    printf("\n");
}

int main()
{
    int choice, data;
    while(1)
    {
        printf("\n");
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Print the first element\n");
        printf("4.Print all the elements of the queue\n");
        printf("5.Quit\n");
        printf("Please enter your choice:\n");
        scanf("%d",&choice);

        switch(choice)

```

```

{
    case 1:
        printf("Enter the element to be added in the queue:");
        scanf("%d",&data);
        enqueue(data);
        break;

    case 2:
        data = dequeue();
        printf("Deleted element is %d\n",data);
        break;

    case 3:
        printf("The first element of the queue is %d\n",peek());
        break;

    case 4:
        print();
        break;

    case 5:
        exit(1);

    default:
        printf("Wrong choice\n");
}
}
return 0;
}

```

### Output:

```

1.Insert
2.Delete
3.Print the first element
4.Print all the elements of the queue
5.Quit
Please enter your choice:
1
Enter the element to be added in the queue:10

```

```

1.Insert
2.Delete
3.Print the first element
4.Print all the elements of the queue
5.Quit
Please enter your choice:
1
Enter the element to be added in the queue:15

```

1.Insert  
2.Delete  
3.Print the first element  
4.Print all the elements of the queue  
5.Quit  
Please enter your choice:  
1  
Enter the element to be added in the queue:45

1.Insert  
2.Delete  
3.Print the first element  
4.Print all the elements of the queue  
5.Quit  
Please enter your choice:  
3  
The first element of the queue is 10

1.Insert  
2.Delete  
3.Print the first element  
4.Print all the elements of the queue  
5.Quit  
Please enter your choice:  
4  
Queue:  
101545

1.Insert  
2.Delete  
3.Print the first element  
4.Print all the elements of the queue  
5.Quit  
Please enter your choice:  
2  
Deleted element is 10

1.Insert  
2.Delete  
3.Print the first element  
4.Print all the elements of the queue  
5.Quit  
Please enter your choice:  
2  
Deleted element is 15

1.Insert  
2.Delete  
3.Print the first element  
4.Print all the elements of the queue  
5.Quit  
Please enter your choice:  
4  
Queue:

45

- 1.Insert
- 2.Delete
- 3.Print the first element
- 4.Print all the elements of the queue
- 5.Quit

Please enter your choice:

2

Deleted element is 45

- 1.Insert
- 2.Delete
- 3.Print the first element
- 4.Print all the elements of the queue
- 5.Quit

Please enter your choice:

4

Queue Underflow

**Conclusion:** Thus, we implemented circular queue using array.

**Viva Questions:**

1. What is drawback of linear Queue?
2. **Which Data Structure Should be used for implementing LRU cache?**
3. The data structure required for Breadth First Traversal on a graph is?
4. If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they be removed?
5. A data structure in which elements can be inserted or deleted at/from both the ends but not in the middle is?



## EXPERIMENT :- 6

### Aim: To study Linked List

**Title:** To implement Singly Linked List

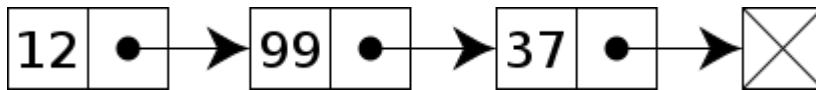
**Objective:** To teach various storage mechanisms of data.

**Outcome:** Students will be able to handle operations like insertion, deletion, searching and traversing on various data structures.

### Theory:

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.

Singly linked lists contain nodes which have a data field as well as a 'next' field, which points to the next node in line of nodes. Operations that can be performed on singly linked lists include insertion, deletion and traversal.



A singly linked list whose nodes contain two fields: an integer value and a link to the next node

### Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have following limitations.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.

2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to be shifted.

For example, in a system if we maintain a sorted list of IDs in an array `id[]`.

`id[] = [1000, 1010, 1050, 2000, 2040]`.

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in `id[]`, everything after 1010 has to be moved.

The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk, while an array has to be declared in the source code, before compiling and running the program. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal.

### Advantages

---

- Linked lists are a dynamic data structure, which can grow and be pruned, **allocating and deallocating memory** while the program is running.
- Insertion and deletion node operations are easily implemented in a linked list.
- Dynamic data structures such as stacks and queues can be implemented using a linked list.
- There is no need to define an initial size for a linked list.
- Items can be added or removed from the middle of list.
- Backtracking is possible in two way linked list.

### **Disadvantages**

---

- They use more memory than **arrays** because of the storage used by their **pointers**.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently **sequential access**.
- Nodes are stored incontiguously, greatly increasing the time required to access individual elements within the list, especially with a **CPU cache**.
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards<sup>[1]</sup> and while doubly linked lists are somewhat easier to read, memory is consumed in allocating space for a **back-pointer**.

### **Applications:**

- 1) Representation of a sparse matrix
- 2) Addition of polynomials

### **Algorithm:**

#### **To insert new node at the beginning**

```

Step 1: Create memory for temp.
Step 2: SET temp->link = NULL
Step 3: SET temp->data = num
Step 4: IF q=NULL
PRINT(Creating)
        SET p = temp
        SET q = temp
Step 5: ELSE
        SET temp->link = p
Step 7: EXIT

```

#### **To insert new node at the end**

```

Step 1: Create memory for temp.
Step 2: SET temp->link=NULL
Step 3: SET temp->data = num
Step 4: IF q=NULL
PRINT(Creating)
        SET p = temp
        SET q = temp GO TO Step 10

```

Step 5: ELSE  
    SET r = q  
Step 6: SET r = r->link  
Step 7: Repeat step 6 while r->link!=NULL  
Step 8: SET r->link = temp  
Step 9: End Of IF  
Step 10: EXIT

### **Delete element from list**

Step 1: SET found = 0  
Step 2: SET prev = NULL  
Step 3: IF q = NULL  
    PRINT(List does not exist) GO TO Step 13  
Step 4: ELSE  
    SET curr = q  
Step 5: IF curr->data = num  
    SET p = curr->link  
    SET found = 1  
  
Step 6: Repeat Step 5 while prev = NULL  
Step 7: ELSE  
    SET prev->link = curr ->link  
    SET found = 1  
Step 8: End Of IF  
Step 9: End Of IF  
Step 10: IF found = 1  
    PRINT(Number deleted)  
Step 11: ELSE  
    PRINT(Number does not exist)  
Step 12: End Of IF  
Step 13: EXIT

### **To count the number of nodes**

Step 1: SET c = 0  
Step 2: IF q = NULL  
    PRINT(List does not exist)  
Step 3: ELSE  
    SET q = q->link  
    SET c = c + 1  
Step 4: repeat Step 3 while q!=NULL  
Step 5: End Of IF  
Step 6: PRINT(c)  
Step 7: EXIT

### **To Display nodes in the list**

Step 1: IF q = NULL  
    PRINT(List does not exit)  
Step 2: ELSE

```
    PRINT(q->data)
Step 3: SET q = q->link
Step 4: Repeat step 2 and step 3 while q!=NULL
Step 5: EXIT
```

**Program:**

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

struct node
{
    int data;
    struct node *link;
};

struct node *head;

void beg_insert ();
void last_insert ();
void random_insert();
void begin_delete();
void last_delete();
void random_delete();
void display();

void main ()
{
    int choice =0;

    while(choice != 5)
    {
        printf("\n\n*****Main Menu*****\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n===== \n");
        printf("\n1.Insert in beginning\n2.Insert at last\n3.Insert at any random
location\n4.Delete from Beginning\n5.Delete from last\n6.Delete node after specified
location\n7.Search for an element\n8.Show\n9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);

        switch(choice)
        {
            case 1: beg_insert();
                    break;
            case 2: last_insert();
                    break;
            case 3: random_insert();
                    break;
            case 4: begin_delete();
                    break;
```

```

    case 5: last_delete();
        break;
    case 6: random_delete();
        break;
    case 7: search();
        break;
    case 8: display();
        break;
    case 9: exit(0);
        break;
    default: printf("Please enter valid choice..");
}
}
}

```

```

void beg_insert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->link = head;
        head = ptr;
        printf("\nNode inserted");
    }
}

```

```

void last_insert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {

```

```

    ptr -> link = NULL;
    head = ptr;
    printf("\nNode inserted");
}
else
{
    temp = head;
    while (temp -> link != NULL)
    {
        temp = temp -> link;
    }
    temp->link = ptr;
    ptr->link = NULL;
    printf("\nNode inserted");
}
}
}

void random_insert()
{
    int i, loc, item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d", &item);
        ptr->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d", &loc);
        temp = head;
        for(i=0; i<loc; i++)
        {
            temp = temp->link;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }
        }
        ptr ->link = temp ->link;
        temp ->link = ptr;
        printf("\nNode inserted");
    }
}

```

```

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->link;
        free(ptr);
        printf("\nNode deleted from the beginning ...\n");
    }
}

void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> link == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->link != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->link;
        }
        ptr1->link = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}

void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion\n");
}

```

```

scanf("%d",&loc);
ptr=head;
for(i=0;i<loc;i++)
{
    ptr1 = ptr;
    ptr = ptr->link;

    if(ptr == NULL)
    {
        printf("\nCan't delete");
        return;
    }
}
ptr1 ->link = ptr ->link;
free(ptr);
printf("\nDeleted node %d ",loc+1);
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
    }
    if(flag==1)
    {
        printf("Item not found\n");
    }
}

```



```

}

void display()
{
    struct node *ptr; ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values . . . .\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> link;
        }
    }
}

```

## OUTPUT

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

1

Enter value

12

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

1

Enter value

13

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

2

Enter value?

34

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning

- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

2

Enter value?

34

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

3

Enter element value445 5

Enter the location after which you want to insert 1

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn

- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

8

printing values . . . . .

13  
12  
45  
34  
34

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

2

Enter value?

55

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn

- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

3

Enter element value67

Enter the location after which you want to insert 5

Node inserted

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

8

printing values . . . . .

13  
12  
45  
34  
34  
55  
67

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

4

Node deleted from the begining ...

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

8

printing values . . . . .

12  
45  
34  
34  
55  
67

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning

- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

5

Deleted Node from the last ...

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

8

printing values . . . . .

12  
45  
34  
34  
55

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn

- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

6

Enter the location of the node after which you want to perform deletion

1

Deleted node 2

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

8

printing values . . . . .

12  
34  
34  
55

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning



- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

7

Enter item which you want to search?

55

item found at location 4

\*\*\*\*\*Main Menu\*\*\*\*\*

Choose one option from the following list ...

=====

- 1.Insert in beginning
- 2.Insert at last
- 3.Insert at any random locationn
- 4.Delete from Beginning
- 5.Delete from last
- 6.Delete node after specified location
- 7.Search for an elementn
- 8.Show
- 9.Exit

Enter your choice?

9

**Conclusion:** Thus we have implemented singly linked list.

**Viva Questions:**

1. What is a linked list?
2. What is the primary advantage of a linked list?
3. What are the parts of a linked list?
4. How do you search for a target key in a linked list?

## EXPERIMENT :- 7

**Aim:** To study Doubly Linked List

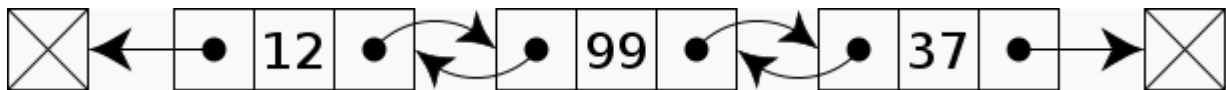
**Title:** To implement Doubly Linked List.

**Objective:** To teach various storage mechanisms of data.

**Outcome:** Students will be able to handle operations like insertion, deletion, searching and traversing on various data structures.

### Theory:

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.

Following are the important terms to understand the concept of doubly linked list.

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

### Basic Operations:

Following are the basic operations supported by a list.

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

#### **Advantages :**

- 1) A DLL can be traversed in both forward and backward direction.
- 2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given. In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

#### **Disadvantages :**

- 2) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though (See [this](#) and [this](#)).
- 2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

#### **Algorithm:**

##### **To insert new node at the beginning**

```

Step 1: Create memory for temp.
Step 2: SET temp->prev = NULL
Step 3: SET temp->data = num
Step 4: IF q=NULL
temp-> next= NULL
        SET p = temp
        SET q = temp
Step 5: ELSE
        SET temp->next = p
        SET p->prev = temp
Step 7: End Of IF
Step 8: EXIT

```

### **To insert new node at the end**

Step 1: Create memory for temp.  
Step 2: SET temp->next=NULL  
Step 3: SET temp->data = num  
Step 4: IF q=NULL  
PRINT(Creating)  
    SET temp->prev = NULL  
    SET p = temp  
    SET q = temp GO TO Step 10  
Step 5: ELSE  
    SET r = q  
Step 6: SET r = r->next  
Step 7: Repeat step 6 while r->next!=NULL  
Step 8: SET r->next = temp  
    SET temp->prev = r  
Step 9: End Of IF  
Step 10: EXIT

### **Delete element from list**

Step 1: SET found = 0  
Step 2: IF q = NULL  
    PRINT(List does not exist)  
Step 5: ELSE  
    SET curr = q  
Step 6: IF curr!=NULL Then  
Step 7: IF curr->data = num Then  
Step 8: IF curr = p Then  
    SET p = curr->next  
Step 9: End Of IF  
Step 10: IF p!=NULL  
    SET p->prev = NULL  
Step 11: ELSE  
    SET (curr->prev)->next = curr->next  
    SET (curr->next)->prev = curr->prev  
    SET found = 1  
Step 12: End Of IF  
Step 13: End Of IF  
Step 13: IF found = 1  
    PRINT(Number deleted)  
Step 11: ELSE  
    PRINT(Number does not exist)  
Step 12: End Of IF  
Step 13: EXIT

### **To count the number of nodes**

Step 1: SET c = 0

Step 2: IF q = NULL  
    PRINT(List does not exist)  
Step 3: ELSE IF q->next!=NULL  
    SET q = q->next  
    SET c = c +1  
Step 5: Repeat Step 3  
Step 6: End Of IF  
Step 7: PRINT(c)  
Step 8: EXIT

### **To Display nodes in the list**

Step 1: IF q = NULL  
    PRINT(List does not exist)  
Step 2: ELSE  
    PRINT(q->data)  
Step 3: SET q = q->next  
Step 4: Repeat step 2 and step 3 while q!=NULL  
Step 5: EXIT

### **Program:**

```
#include<stdio.h>

#include<stdlib.h>

struct node
{
    int data;

    struct node *prev,*next;
};

struct node *p;

void append(struct node *q,int num)
{
    struct node *temp,*r;

    if(q==NULL)
    {
        printf("Creating List\n");
```

```

        temp=(struct node*)malloc(sizeof(struct node));

        temp->data=num;

        temp->prev=NULL;

        temp->next=NULL;

        p=q=temp;
    }
else
    {
        temp=(struct node*)malloc(sizeof(struct node));

        temp->data=num;

        temp->next=NULL;

        r=q;

        while(r->next!=NULL)

            r=r->next;

        r->next=temp;

        temp->prev=r;
    }
}

void inbegin(struct node *q,int num)
{
    struct node *temp;

    if(q==NULL)

        printf("Link List Does Not Exists\n");

    else
    {
        temp=(struct node *)malloc(sizeof(struct node));

        temp->data=num;

        temp->next=q;
    }
}

```

```

        temp->prev=NULL;

        q->prev=temp;

        p=q=temp;

    }

}

void delet(struct node *q,int num)
{
    struct node *curr;

    int found=0;

    if(q==NULL)
        printf("Link List Does Not Exists\n");
    else
    {
        for(curr=q;curr!=NULL;curr=curr->next)
        {
            if(curr->data==num)
            {
                if(curr==p)
                {
                    p=(curr->next);
                    if(p!=NULL)
                    {
                        p->prev=NULL;
                    }
                    found=1;
                }
            }
            else
            {

```

```

        (curr->prev)->next=curr->next;
        (curr->next)->prev=curr->prev;
        found=1;
    }
}
}
if(found==1)
    printf("Number Deleted\n");
else
    printf("Number Not Found\n");
}

}

void count(struct node *q)
{
    int c;
    c=0;
    if(q==NULL)
        printf("Link List Does Not Exists\n");
    else
    {
        while(q!=NULL)
        {
            c++;
            q=q->next;
        }
        printf("Number Of Nodes %d\n",c);
    }
}

```



```

    }
}

void display(struct node *q)
{
    if(q==NULL)
        printf("Link List Does Not Exists\n");

    else

    {
        printf("Contents:\n");
        while(q!=NULL)
        {
            printf("%d\n",q->data);
            q=q->next;
        }
    }
}

int main()
{
    int n,c;
    p=NULL;

    do
    {
        printf("Enter Your Choice:\n");
        printf("1.Create\\Append\t");
        printf("\t2.Inbegin\t");
        printf("\t3.Delete\t");
        printf("\n4.Count\t");
    }
}

```

```

printf("\t\t5.Display\t");
printf("\t6.Exit\n");
scanf("%d",&c);
switch(c)
{
    case 1:
        printf("Enter A Value:\n");
        scanf("%d",&n);
        append(p,n);
        break;
    case 2:
        printf("Enter A Value:\n");
        scanf("%d",&n);
        inbegin(p,n);
        break;
    case 3:
        printf("Enter A Value:\n");
        scanf("%d",&n);
        delet(p,n);
        break;
    case 4:
        count(p);
        break;
    case 5:
        display(p);
        break;
}
}

```

```

while(c!=6);

return 0;

}

```

### Output:

Enter Your Choice:

1.Create\Append	2.Inbegin	3.Delete
4.Count	5.Display	6.Exit

1

Enter A Value:

1

Creating List

Enter Your Choice:

1.Create\Append	2.Inbegin	3.Delete
4.Count	5.Display	6.Exit

1

Enter A Value:

5

Enter Your Choice:

1.Create\Append	2.Inbegin	3.Delete
4.Count	5.Display	6.Exit

2

Enter A Value:

4

Enter Your Choice:

1.Create\Append	2.Inbegin	3.Delete
4.Count	5.Display	6.Exit

5

Contents:

4

1

5

**Conclusion:** Thus we have implemented doubly linked list.

### Viva Questions:

- 1.What are the advantages of doubly linked list over single linked list?
2. What are the disadvantages of doubly linked list over single linked list?

## EXPERIMENT :- 8

**Aim:** To study stack using linked list

**Title:** To implement stack using Linked List

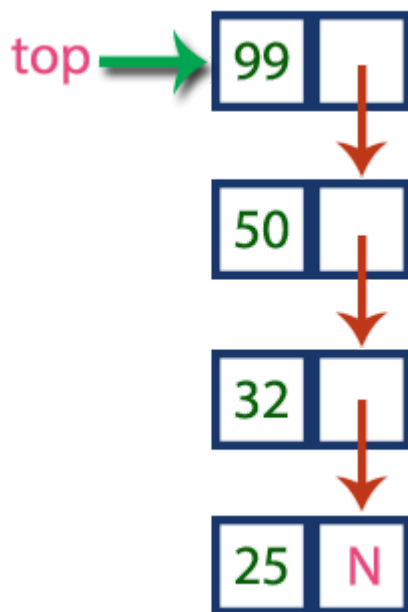
**Objective:** To perform Push, Pop operation on stack using Linked List

### Theory:

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

Example



In above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32, 50 and 99.

### Algorithm:

To implement stack using linked list, we need to set the following things before implementing

*Dept. of Computer Engineering, Shree L. R. Tiwari College of Engineering, Thane-401107.*

actual operations.

Step 1: Include all the header files which are used in the program. And declare all the user defined functions.

Step 2: Define a 'Node' structure with two members data and next.

Step 3: Define a Node pointer 'top' and set it to NULL.

Step 4: Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

push(value) - Inserting an element into the Stack

We can use the following steps to insert a new node into the stack...

Step 1: Create a newNode with given value.

Step 2: Check whether stack is Empty (top == NULL)

Step 3: If it is Empty, then set newNode → next = NULL.

Step 4: If it is Not Empty, then set newNode → next = top.

Step 5: Finally, set top = newNode.

pop() - Deleting an Element from a Stack

We can use the following steps to delete a node from the stack...

Step 1: Check whether stack is Empty (top == NULL).

Step 2: If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function

Step 3: If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.

Step 4: Then set 'top = top → next'.

Step 7: Finally, delete 'temp' (free(temp)).

display() - Displaying stack of elements

We can use the following steps to display the elements (nodes) of a stack...

Step 1: Check whether stack is Empty (top == NULL).

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

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

Step 4: Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack (temp → next != NULL).

Step 4: Finally! Display 'temp → data ---> NULL'.

### **Program:**

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

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

```
struct node  
{
```

```
int data;  
struct node *link;  
};
```

```
struct node *p;  
struct node *top;
```

```
void push(struct node *q,int num)
```

```

{

struct node *temp;
if(q==NULL)
{
    printf("Creating stack\n");
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    temp->link=NULL;
    p=temp;
    top=p;

}
else
{
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    temp->link=p;
    p=temp;
    top=p;
}
}

```

```

void pop(struct node *q)
{

```

```

    if(q==NULL)
        printf("stack does not exist");

```

```

    else
    {
        p=q->link;
        top=p;

```

```

    }

```

```

}

```

```

void count(struct node *q)
{

```

```

    int c; c=0;
    if(q==NULL)
        printf("Link List Does Not Exists");
    else
    {

```

```

        while(q!=NULL)
        {

```

```

            c++;
            q=q->link;
        }

```

```
printf("Number Of Nodes %d\n",c);
}

}
```

```
void display(struct node *q)
{

if(q==NULL)
printf("Link List Does Not Exists\n");
else
{

printf("Contents:\n");
while(q!=NULL)
{

printf("%d\n" ,q->data);
q=q->link;
}
}
}
```

```
int main()
{

int n,c; p=NULL;

do
{

printf("Enter Your Choice:\n");
printf("1.Create/push\t");
```

```
printf("\t2.pop\t");
printf("\t3.Count\t");
printf("\t4.Display\t");
printf("\t5.Exit\n");
scanf("%d",&c);
switch(c)
{
```

```
case 1:
printf("Enter A Value:\n");
scanf("%d",&n);
push(p,n);
break;
case 2:
pop(p);
break;
```

```
case 3:
```

```

count(p);
break;
case 4: display(p);
break;
}
}
while(c!=5);
return 0;

}

```

### Output:

```

Enter Your Choice:
1.Create/push      2.pop      3.Count      4.Display      5.Exit
1
Enter A Value:
5
Creating stack
Enter Your Choice:
1.Create/push      2.pop      3.Count      4.Display      5.Exit
1
Enter A Value:
8
Enter Your Choice:
1.Create/push      2.pop      3.Count      4.Display      5.Exit
2
Enter Your Choice:
1.Create/push      2.pop      3.Count      4.Display      5.Exit
4
Contents:
5

```

**Conclusion:** Thus we have implemented stack using linked list.

### Viva Questions:

1. Linked lists considered linear or non-linear data structures?
2. What is the primary advantage of a linked list?



## EXPERIMENT :- 9

**Aim: To study Binary Tree**

**Title:** To implement Binary tree.

**Objective:** To perform insertion and traversal operation on Binary Tree

**Outcome:** Students will be able to handle operations like insertion, deletion, searching and traversing on Binary Tree..

### Theory:

A binary tree is a method of placing and locating files (called records or keys) in a database, especially when all the data is known to be in random access memory (RAM). The algorithm finds data by repeatedly dividing the number of ultimately accessible records in half until only one remains.

In a tree, records are stored in locations called leaves. This name derives from the fact that records always exist at end points; there is nothing beyond them. Branch points are called nodes. The order of a tree is the number of branches (called children) per node. In a binary tree, there are always two children per node, so the order is 2. The number of leaves in a binary tree is always a power of 2. The number of access operations required to reach the desired record is called the depth of the tree.

### TYPES OF BINARY TREE:-

#### 1. Strictly Binary Tree

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follows...A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree. Strictly binary tree is also called as Full Binary Tree or Proper Binary Tree or 2-Tree. Strictly binary tree data structure is used to represent mathematical expressions.

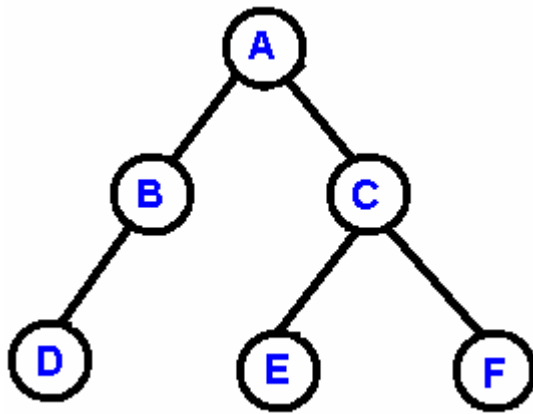
#### 2. Complete Binary Tree

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be  $2^{\text{level}}$  number of nodes. For example at level 2 there must be  $2^2 = 4$  nodes and at level 3 there must be  $2^3 = 8$  nodes. A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree. Complete binary tree is also called as Perfect Binary Tree.

#### 3. Extended Binary Tree

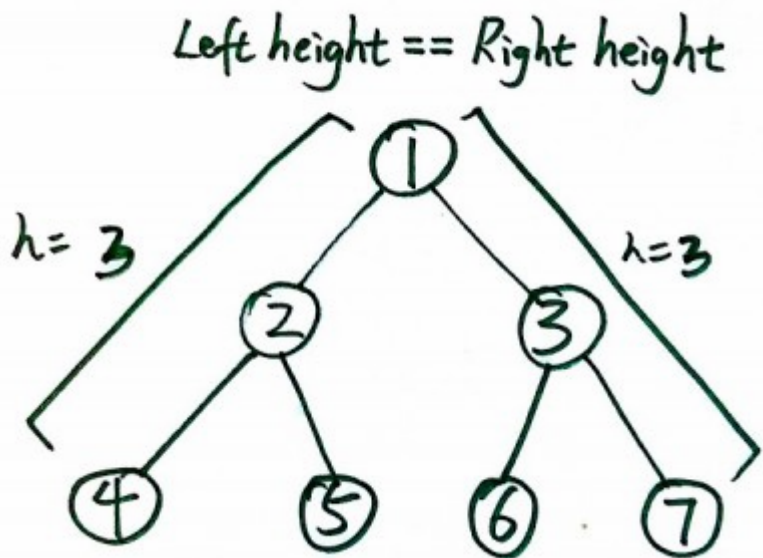
A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required. The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.

### Example of binary tree



Here A is the root node ,B&C are its left and right subtree respectively,further E&F are C's left and right subtree respectively and lastly D is the left subtree of B.

Counting of nodes in binary tree, and also measuring of height in binary tree..



Applications of binary trees:-

- Binary Search Tree - Used in *many* search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- Binary Space Partition - Used in almost every 3D video game to determine what objects need to be rendered.
- Binary Tries - Used in almost every high-bandwidth router for storing router-tables.
- Hash Trees - used in p2p programs and specialized image-signatures in which a hash needs to be verified, but the whole file is not available.
- Heaps - Used in implementing efficient priority-queues, which in turn are used for scheduling processes in many operating systems, Quality-of-Service in routers, and A\* (*path-finding algorithm used in AI applications, including robotics and video games*). Also used in heap-sort.
- Huffman Coding Tree (Chip Uni) - used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.

- GGM Trees - Used in cryptographic applications to generate a tree of pseudo-random numbers.
- Syntax Tree - Constructed by compilers and (implicitly) calculators to parse expressions.
- Treap - Randomized data structure used in wireless networking and memory allocation.
- T-tree - Though most databases use some form of B-tree to store data on the drive, databases which keep all (most) their data in memory often use T-trees to do so.

#### Advantages:-

- We have an ordering of keys stored in the tree. Any time we need to traverse the increasing (or decreasing) order of keys, we just need to do the in-order (and reverse in-order) traversal on the tree.
- We can implement order statistics with binary search tree - Nth smallest, Nth largest element. This is because it is possible to look at the data structure as a sorted array.
- We can also do range queries - find keys between N and M ( $N \leq M$ ).
- BST can also be used in the design of memory allocators to speed up the search of free blocks (chunks of memory), and to implement best fit algorithms where we are interested in finding the smallest free chunk with size greater than or equal to size specified in allocation request.

#### Disadvantages:-

- The main disadvantage is that we should always implement a balanced binary search tree - AVL tree, Red-Black tree, Splay tree. Otherwise the cost of operations may not be logarithmic and degenerate into a linear search on an array.

#### Algorithm:

##### To insert a node :

Step 1: PRINT(Press 'L' to add left and 'R' to add right)

Step2: INPUT(ans)

Step 3: IF ans = 'R' Then

Step 4: IF r->right=NULL Then

r->right=q

Step 5: ELSE

CALL insert(r->right , q) GO TO Sep 1

Step 6: End Of IF

Step 7: ELSE

IF r->left=NULL Then

r->left = q

Step 8: ELSE

CALL insert(r->left,q) GO TO Step 1

Step 9: End Of IF

Step 10: EXIT

### **To Display in Preorder ,Inorder and Postorder**

Step 1: IF r!=NULL Then  
Step 2: IF n=1 Then  
    PRINT(r->data)  
    CALL tree(r->left , n) GO TO Step 1  
    CALL tree(r->right , n) GO TO Step 1  
Step 3: End Of IF  
Step 4: IF n=2 Then  
    CALL tree(r->left , n) GO TO Step 1  
    PRINT(r->data)  
    CALL tree(r->right , n) GO TO Step 1  
Step 5: End Of IF  
Step 6: IF n=3 Then  
    CALL tree(r->left , n) GO TO Step 1  
    CALL tree(r->right , n) GO TO Step 1  
    PRINT(r->data)  
Step 7: End Of IF  
Step 8: EXIT

### **Program:**

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

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

```
struct node
```

```
{
```

```
int data;
```

```
struct node *left,*right;
```

```
};
```

```
struct node *root,*q;
```

```
void insert(struct node *,struct node *);
```

```
void tree(struct node * ,int );
```

```

int main()

{

struct node *temp;

int n,x,c;

root=NULL;

do

{

printf("Enter Your Choice\n");

printf("1.Create\\Add\t2.Display\t3.Exit\n");

scanf("%d",&c);

switch(c)

{

case 1:

temp=(struct node *)malloc(sizeof(struct node));

temp->left=NULL;

temp->right=NULL;

printf("Enter The Data\n");

scanf("%d",&x);

temp->data=x;

if(root==NULL)

{

root=temp;

printf("Value Added\n");

}

else

insert(root,temp);

break;

case 2: printf("1.Preorder\t2.Inorder\t3.Postorder\n");

```

```

        scanf("%d",&n);
        if(root==NULL)

        printf("Tree Does Not Exists\n");
        else
        tree(root,n);
        break;
    }

    }while(c!=3);
    return 0;
}

void insert(struct node *root,struct node *q)

{

int ans;

printf("Press 1 To Insert In Left\n");
printf("Press 2 To Insert In Right\n");
scanf("%d",&ans);

if(ans==2)
{
    if(root->right==NULL)
    {
        root->right=q;

        printf("Value Added To Right Successfully\n");

    }
    else
    insert(root->right,q);
}

```

```

else
{
    if(root->left==NULL)
    {
        root->left=q;

        printf("Value Added To Right Successfully\n");
    }
    else
        insert(root->left,q);
}
}

```

```

void tree(struct node *root,int n)

```

```

{
    if(root!=NULL)
    {
        if(n==1)
        {
            printf("%d\n",root->data);
            tree(root->left,n);
            tree(root->right,n);
        }
        if(n==2)
        {
            tree(root->left,n);
            printf("%d\n",root->data);
            tree(root->right,n);
        }
    }
}

```

```

if(n==3)
{

    tree(root->left,n);

    tree(root->right,n);

    printf("%d\n",root->data);

}
}
}

```

### Output:

```

Enter Your Choice
1.Create\Add  2.Display    3.Exit
1
Enter The Data
5
Value Added
Enter Your Choice
1.Create\Add  2.Display    3.Exit
1
Enter The Data
8
Press 1 To Insert In Left
Press 2 To Insert In Right
1
Value Added To Right Successfully
Enter Your Choice
1.Create\Add  2.Display    3.Exit
1
Enter The Data
8
Press 1 To Insert In Left
Press 2 To Insert In Right
2
Value Added To Right Successfully
Enter Your Choice
1.Create\Add  2.Display    3.Exit
2
1.Preorder    2.Inorder    3.Postorder
1
5
8
8

```

**Conclusion:** Thus we have studied and implemented binary tree.



**Viva Questions:**

- 1.What is an AVL tree?**
- 2.What are binary rees?**
- 3. Differentiate linear from non linear data structure.**
- 4.What is threaded binary tree?**

# Experiment 10

**Aim:** To study searching technique

**Title:** To implement binary search.

**Objective:** To search given element in a list using Binary Search.

**Outcome:** Students will be able to search for element in a list using Binary Search.

## Theory:

Binary search is one of the fundamental algorithms in computer science. In order to explore it, we'll first build up a theoretical backbone, then use that to implement the algorithm properly and avoid those nasty off-by-one errors everyone's been talking about.

### Finding a value in a sorted sequence

In its simplest form, binary search is used to quickly find a value in a sorted sequence (consider a sequence an ordinary array for now). We'll call the sought value the *target* value for clarity. Binary search maintains a contiguous subsequence of the starting sequence where the target value is surely located. This is called the *search space*. The search space is initially the entire sequence. At each step, the algorithm compares the median value in the search space to the target value. Based on the comparison and because the sequence is sorted, it can then eliminate half of the search space. By doing this repeatedly, it will eventually be left with a search space consisting of a single element, the target value.

For example, consider the following sequence of integers sorted in ascending order and say we are looking for the number 55:

0	5	13	19	22	41	55	68	72	81	98
---	---	----	----	----	----	----	----	----	----	----

We are interested in the location of the target value in the sequence so we will represent the search space as indices into the sequence. Initially, the search space contains indices 1 through 11. Since the search space is really an interval, it suffices to store just two numbers, the low and high indices. As described above, we now choose the median value, which is the value at index 6 (the midpoint between 1 and 11): this value is 41 and it is smaller than the target value. From this we conclude not only that the element at index 6 is not the target value, but also that no element at indices between 1 and 5 can be the target value, because all elements at these indices are smaller than 41, which is smaller than the target value. This brings the search space down to indices 7 through 11:

55	68	72	81	92
----	----	----	----	----

Proceeding in a similar fashion, we chop off the second half of the search space and are left with:

5	6
5	8

Depending on how we choose the median of an even number of elements we will either find 55 in the next step or chop off 68 to get a search space of only one element. Either way, we conclude that the index where the target value is located is 7.

### Algorithm:

STEP 1: get the middle element;

STEP 2: if the middle element equals to the searched value, the algorithm stops;

STEP 3: otherwise, two cases are possible:

- searched value is less, than the middle element. In this case, go to the step 1 for the part of the array, before middle element.
- searched value is greater, than the middle element. In this case, go to the step 1 for the part of the array, after middle element

### Program:

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

```
int main()
```

```
{
```

```
int a[100],n,key,low,mid,high,i,chk=0;
```

```
printf("How Many Elements:\n");
```

```
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
```

```
{
```

```
printf("Enter Element %d:\n",(i+1));
```

```
scanf("%d",&a[i]);
```

```
}
```

```
printf("Enter The Element To Be Searched:\n");
```

```
scanf("%d",&key);
```

```
low=0; high=n-1;
```

```
while(low<=high)
```

```

{

mid=(low+high)/2;
if(a[mid]==key)
{

printf("Value Found At Position %d\n",(mid+1)); chk=1;
break;

}

if(a[mid]>key)
high=mid-1;
else low=mid+1;
}

if(chk==0)

printf("Value Not Found\n");
return 0;
}

```

### **Output:**

How Many Elements:

5

Enter Element 1:

10

Enter Element 2:

20

Enter Element 3:

30

Enter Element 4:

45

Enter Element 5:

50

Enter The Element To Be Searched:

45

Value Found At Position 4

**Conclusion:** Thus, we implemented binary search.

**Viva Questions:**

*Dept. of Computer Engineering, Shree L. R. Tiwari College of Engineering, Thane-401107.*

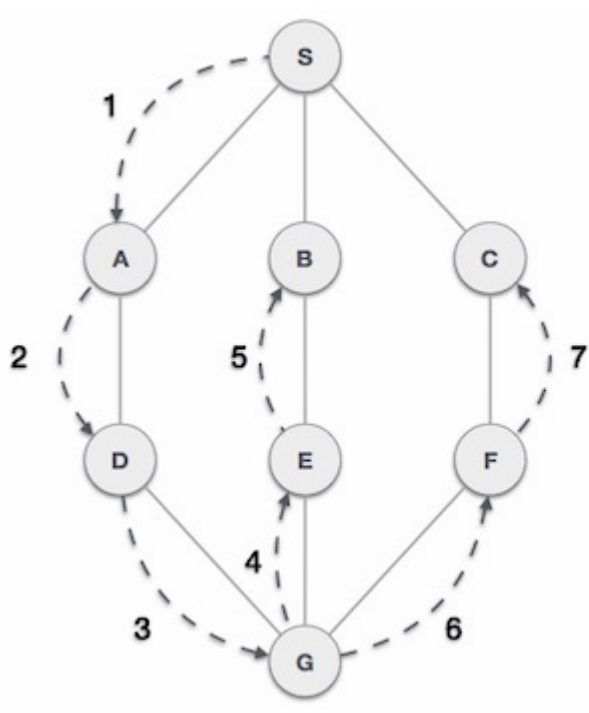
- 1.What is time complexity of Binary Search?
- 2.What are different searching techniques?
- 3.What is searching?

## **EXPERIMENT NO. 11**

### **Aim:To study Graph Traversal Technique**

**Title:** Write A Program To implement depth first search Algorithm.

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



### **Algorithm:**

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.

4. Keep repeating steps 2 and 3 until the stack is empty.

### Program:

```
#include<stdio.h>

int a[20][20],reach[20],n;
void dfs(int v) {
    int i;
    reach[v]=1;
    for (i=1;i<=n;i++)
        if(a[v][i] && !reach[i]) {
            printf("\n %d->%d",v,i);
            dfs(i);
        }
}
void main() {
    int i,j,count=0;

    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
        reach[i]=0;
        for (j=1;j<=n;j++)
            a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for (i=1;i<=n;i++) {
        if(reach[i])
            count++;
    }
    if(count==n)
        printf("\n Graph is connected"); else
        printf("\n Graph is not connected");
}
```

### Output:

Enter number of vertices:4

Enter the adjacency matrix:

1

1

0

1

1

0

1

1

1

0

0

0

1

1

1

1

1->2

2->3

2->4

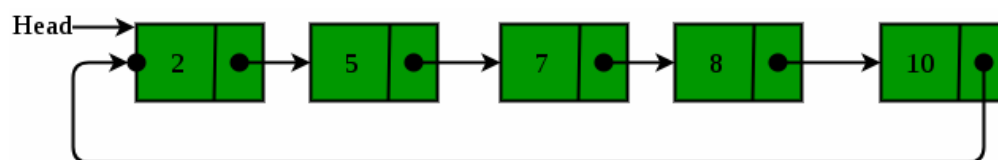
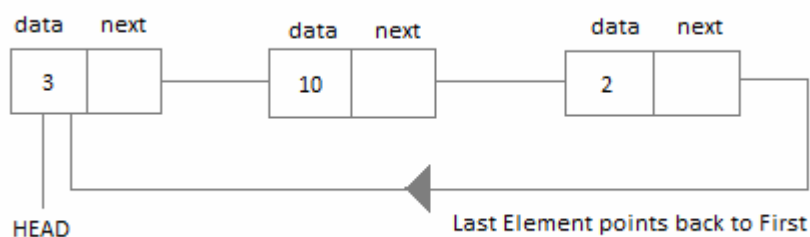
Graph is connected

## EXPERIMENT NO. 12

**Aim:** To study Circular Linked List

**Title:** Write A Program to Implement Circular Linked List.

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



Advantages of Circular Linked Lists:

- 1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
- 2) Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.
- 3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.



## **Application of Circular Linked List**

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

### **FOR APPEND OPERATION:**

1. If the Linked List is empty then we simply, add the new Node as the Head of the Linked List.
2. If the Linked List is not empty then we find the last node, and make it' next to the new Node, and make the next of the Newly added Node point to the Head of the List.

### **FOR INBEGIN OPERATION:**

1. The first Node is the Head for any Linked List.
2. When a new Linked List is instantiated, it just has the Head, which is Null.
3. Else, the Head holds the pointer to the first Node of the List.
4. When we want to add any Node at the front, we must make the head point to it.
5. And the Next pointer of the newly added Node, must point to the previous Head, whether it be NULL(in case of new List) or the pointer to the first Node of the List.
6. The previous Head Node is now the second Node of Linked List, because the new Node is added at the front.

### **FOR DELETE OPERATION:**

- If the Node to be deleted is the first node, then simply set the Next pointer of the Head to point to the next element from the Node to be deleted. And update the next pointer of the Last Node as well.
- If the Node is in the middle somewhere, then find the Node before it, and make the Node before it point to the Node next to it.

- If the Node is at the end, then remove it and make the new last node point to the head.

### **FOR COUNT OPERATION:**

1. Start
2. Declare C =0
3. while (Q != NULL)
4. Display "List Does Not Exists"
5. Else (Step 5 To Step 10)
6. Repeat till Step 8 while (Q != rear)
7. Increment C by 1
8. Q=Q's Link Section
9. End while
10. Increment c by 1
11. Display Value of  
C
12. Stop

### **FOR DISPLAY OPERATION:**

1. Start
2. if (Q != NULL)
3. Display "List Does Not Exists"
4. Else (Step 4 To Step 8)
5. Display "Contents"
6. Move to A New Line
7. Repeat till Step 8 while (Q != rear)
8. Display Value of (Q's Data Section)
9. Move to A New  
Line
10. Q=Q's Link Section
11. Display Value  
of(Q's Data section)
12. Stop

### **Program:**

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

```

#include<stdlib.h>

struct node
{

int data;
struct node *link;
};

struct node *p;

void append(struct node *q,int num)
{

struct node *temp,*r;

if(q==NULL)
{
printf("Creating List\n");
temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;
temp->link=temp;
p=temp;
q=temp;

}
else
{
r=q;
while(r->link!=p)
r=r->link;

temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;
temp->link=p;
r->link=temp;
}
}

void inbegin(struct node *q,int num)
{

struct node *temp,*r;
if(q==NULL)
printf("Link List Does Not Exists\n");
else
{
r=q;
while(r->link!=p)
r=r->link;

temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;
temp->link=p;
p=temp;
}
}

```

```

    r->link=p;
}
}

void delet(struct node *q,int num)
{
    struct node *prev,*curr,*r;
    int found=0;
    prev=NULL;

    if(q==NULL)
        printf("list does not exist");

    else if(q->data=num)
    {
        r=q;
        while(r->link!=p)
            r=r->link;

        q=q->link;
        r->link=p;
    }

    else
    {
        for(curr=q->link;curr!=p;prev=curr,curr=curr->link)
        {
            if(curr->data==num)
            {
                if(prev==NULL)
                {
                    p=curr->link;
                    found=1;
                }

                else
                {
                    prev->link=curr->link;
                    found=1;
                }
            }
        }
    }

    if(found==1)
        printf("no deleted");

    else
        printf("not found");

}

}

void count(struct node *q)
{

```

```

int c;
c=0;
if(q==NULL)
printf("Link List Does Not Exists");
else
{
    c++;
    q=q->link;
    while(q!=p)
    {

c++;
q=q->link;
}

printf("Number Of Nodes %d\n",c);
}

}

void display(struct node *q)
{

if(q==NULL)
printf("Link List Does Not Exists\n");
else
{

printf("Contents:\n");
printf("%d\n",q->data);
q=q->link;
while(q!=p)
{

printf("%d\n" ,q->data);
q=q->link;
}
}
}

int main()
{

int n,c; p=NULL;

do
{

printf("Enter Your Choice:\n");
printf("1.Create\\Append\t");

printf("\t2.Begin\t");
printf("\t3.Delete\t"); printf("\t4.Count\t");

```

```

printf("\t5.Display\t");
printf("\t6.Exit\n");
scanf("%d",&c);
switch(c)
{

case 1:
printf("Enter A Value:\n");
scanf("%d",&n); append(p,n);
break;
case 2:
printf("Enter A Value:\n");
scanf("%d",&n);
inbegin(p,n);
break;
case 3:
printf("Enter A Value:\n");
scanf("%d",&n);
delet(p,n);
break;
case 4:

count(p);
break;
case 5: display(p);
break;
}
}
while(c!=6);
return 0;

}

```

Output:

Enter Your Choice:

1.Create\Append	2.Begin	3.Delete	4.Count
5.Display	6.Exit		

1

Enter A Value:

5

Creating List

Enter Your Choice:

1.Create\Append  
5.Display

2.Begin  
6.Exit

3.Delete

4.Count

2

Enter A Value:

5

Enter Your Choice:

1.Create\Append  
5.Display

2.Begin  
6.Exit

3.Delete

4.Count

1

Enter A Value:

9

Enter Your Choice:

1.Create\Append  
5.Display

2.Begin  
6.Exit

3.Delete

4.Count

4

Number Of Nodes 3

Enter Your Choice:

1.Create\Append  
5.Display

2.Begin  
6.Exit

3.Delete

4.Count

### **Conclusion:**

Thus we have studied circular linked list is more efficient than normal linked list.

## **Aim: To study Binary Search Tree**

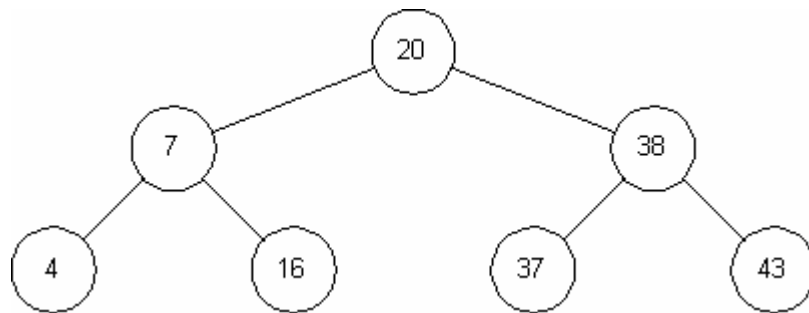
**Title:** Write A Program to Implement Binary Search Tree.

**Theory:** A **Binary Tree** consists of **Nodes** and **Branches**. A **Node** is a place where data is stored, there is one **Special Node** the **root** which is the **starting point** of the tree. **Branches** connect the nodes to each other. A branch is either a **left branch** or a **right branch**. Accordingly, a node is said to have a **left child** and a **right child**. A node without children is called a **leaf**. Non-leaf nodes are called **internal nodes**.

## **BINARY SEARCH TREE**

A **Binary Search Tree** is a **Binary Tree** whose entries can be arranged in order. For every node  $x$  in the tree, the value of the entry of  $x$  is *greater than the values of all the entries in the left subtree of  $x$ , and smaller than the values of all the entries in the right subtree*.

They Are a Particular Type of Containers: Data Structure that store "**items**" (such as numbers, names etc.) in Memory. They allow Fast Lookup, Addition & Removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its *key*.



## **Balanced Binary Tree Search:**

A search of the balanced binary tree is equivalent to a binary search of an ordered list. In both cases, each check eliminates half of the remaining items. Hence searching is  $O(\log N)$ .

## **Binary Search Tree Expectations:**

Insertions & Deletions should be faster than  $O(N)$ , and searching should not be slower than  $O(\log N)$ .



## **Algorithm:**

### **FOR INSERT OPERATION**

- Start
- If the Root is NULL
  - Replace the Empty Tree with a New Tree with the Item at the Root
- Else if item.data equals root.data
  - Item Already There Implies Error
- Else if item.data is less than root.data
  - Recursively Insert the Item in the Left Subtree
- Else
  - Recursively Insert the Item in the Right Subtree

### **FOR SEARCH OPERATION**

1. Start
2. Declare Return Type of Search Operation as Int and Pass Parameters  
Root Address and The Data to be Found
3. If Data to be found is present at root node
  - ✓ Print “Value Found at Root Node”
  - ✓ Return 0
  - ✓ If Data is Found at current Root
  - ✓ Print “Value Found”
  - ✓ Return 0
4. If Data to be found is greater than the value of root node
  - ✓ If root’s Right Child is NULL
    - Print “Value Not Found”
    - Return 0
  - ✓ Else
    - Recursively Call Search Function with Root’s Right Child and Data to be found
5. If Data to be found is less than the value of root node
  - ✓ If root’s Left Child is NULL
    - Print “Value Not Found”
    - Return 0
  - ✓ Else
    - Recursively Call Search Function with Root’s Left Child and Data to be found

6. Return 0

7. Stop

### **Program:**

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node *lc,*rc;

};

struct node *p;

void insert(struct node *, struct node *);

int search(struct node* ,int);

void main()

{

    struct node *temp;

    int c,i,x,n;

    p=NULL;

    clrscr();

    printf("Enter Number of Element:\t");

    scanf("%d",&c);

    for(i=1;i<=c;i++)

    {

        temp=(struct node *)malloc(sizeof(struct node));

        temp->lc=NULL;

        temp->rc=NULL;
```

```
printf("Enter Value %d:\t",i);  
scanf("%d",&n);  
temp->data=n;  
if(p==NULL)  
    p=temp;  
else  
    insert(p,temp);  
  
}
```

```

printf("Enter Value to Search :\t");
scanf("%d",&x);

    search(p,x);

    getch();

}

void insert(struct node *r,struct node *t)
{
    if(t->data>r->data)
    {
        if(r->rc==NULL)
            r->rc=t;
        else
            insert(r->rc,t);
    }
    else if(t->data<r->data)
    {
        if(r->lc==NULL)
            r->lc=t;
        else

```

```

        insert(r->lc,t);
    }
}

int search(struct node *r,int t)
{
    if(p->data==t)
    {
        printf("Value Found At Root Node\n");
        return 0;
    }
    else if(r->data==t)
    {
        printf("Value Found\n");
        return 0;
    }
    if(t>r->data)
    {
        if(r->rc==NULL)
        {
            printf("Value Not Found\n");
            return 0;
        }
        else
            search(r->rc,t);
    }
    else if(t<r->data)

```

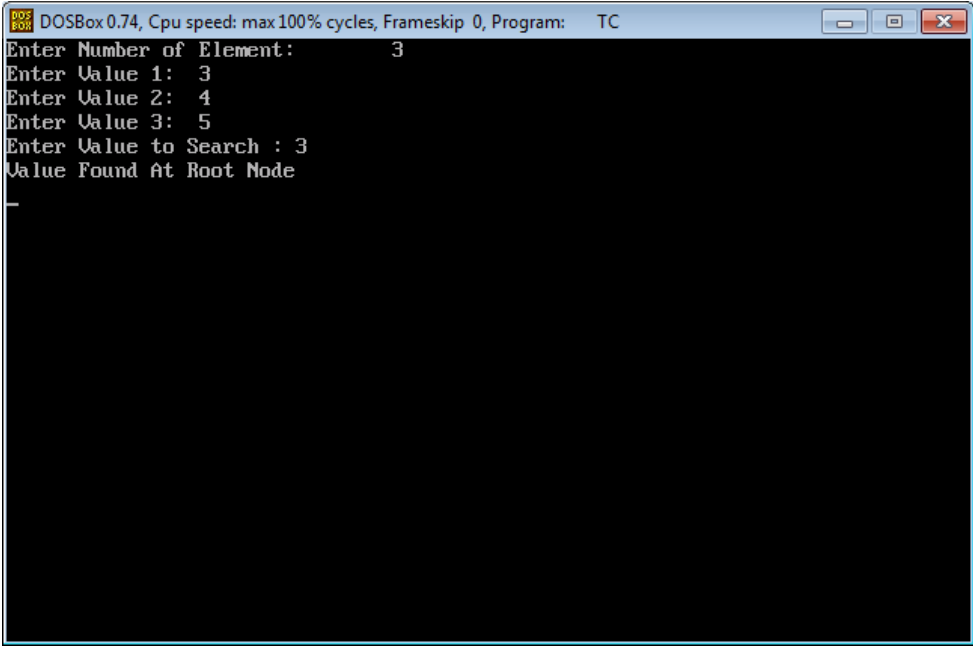
```

    {
        if(r->lc==NULL)
        {
            printf("Value Not Found\n");return 0;

        }
        search(r->lc,t);
    }
    else
    {
        return 0;
    }
}

```

## **Output:**



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter Number of Element: 3
Enter Value 1: 3
Enter Value 2: 4
Enter Value 3: 5
Enter Value to Search : 3
Value Found At Root Node

```

