LABORATORY MANUAL

DATA STRUCTURES LABORATORY
18CSL38

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANA SANGAMA, BELGAVI-590018, KARNATAKA

Prepared By:

Mr. Narayan H M
Associate Professor
Dept. of CSE, MSEC

Mrs. Vinutha M
Assistant Professor
Dept. of CSE, MSEC

Department of Computer Science and Engineering

M.S. Engineering College

(An ISO 9001:2015 Certified Institution)
(Affiliated to Visvesvaraya Technological University Belgaum and approved by AICTE, New Delhi)
Navarathna Agrahara, Sadahalli Post, Bengaluru- 562 110, Tel: 080-3252 9939, 3252 957

DATA STRUCTURES LABORATORY (Effective from the academic year 2018 -2019) SEMESTER-III			
Subject Code	18CPL38	**CIE Marks**	40
Number of Contact Hours/Week	0:2:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	3 Hrs
Credits – 2			

Descriptions (if any):

- Implement all the programs in 'C / C++' Programming Language and Linux / Windows as OS.

Course Learning Objectives: This course (18CSL38) will enable students to:

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

- Asymptotic performance of algorithms.
- Linear data structures and their applications such as stacks, queues and lists
- Non-Linear data structures and their applications such as trees and graphs • Sorting and searching algorithms

Laboratory Outcomes: The student should be able to:

- Analyze and Compare various linear and non-linear data structures
- Code, debug and demonstrate the working nature of different types of data structures and their applications
- Implement, analyze and evaluate the searching and sorting algorithms
- Choose the appropriate data structure for solving real world problems

Programs List:

1.	Design, Develop and Implement a menu driven Program in C for the following array operations. a. Creating an array of N Integer Elements b. Display of array Elements with Suitable Headings c. Inserting an Element (ELEM) at a given valid Position (POS) d. Deleting an Element at a given valid Position (POS) e. Exit. Support the program with functions for each of the above operations.
2.	Design, Develop and Implement a Program in C for the following operations on Strings. a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP) b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in functions.
3.	Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations

4.	Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.
5.	Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks
6.	Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations
7.	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Branch, Sem, PhNo* a. Create a SLL of N Students Data by using *front insertion*. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion / Deletion at End of SLL d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e. Exit
8.	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo* a. Create a DLL of N Employees Data by using *end insertion*. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit
9.	Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
10.	Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers . a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 b. Traverse the BST in Inorder, Preorder and Post Order Search the BST for a given element (KEY) and report the appropriate message d. Exit

11.	Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method
12.	Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Execution Steps:

Step 1: Open Terminal and type the following commands
Step 2: linux:~/dslab # gedit pgmname.c (To open an editor and for typing the program)
Step 3: linux:~/dslab # cc pgmname.c (To Compile the Program)
Step 4: linux:~/dslab # ./a.out (To Run the Program)

Conduct of Practical Examination:

- All laboratory experiments, excluding the first, are to be included for practical examination.
- Experiment distribution ○ For questions having only one part: Students are allowed to pick one experiment from the lot and are given equal opportunity.
  - ○ For questions having part A and B: Students are allowed to pick one experiment from part A and one experiment from part B and are given equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure part to be made zero.
- Marks Distribution *(Subjected to change in accoradance with university regulations)*
  - c) For questions having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - d) For questions having part A and B
    - i. Part A – Procedure + Execution + Viva = 4 + 21 + 5 = 30 Marks ii.
    - Part B – Procedure + Execution + Viva = 10 + 49+ 11 = 70 Marks

Prepared/Updated By:	**Verified By:**
Mr. Narayan H M & Mrs.Vinutha M	Mrs. Dipti Patnayak

EXPERIMENT: 1

1.Design, Develop and Implement a menu driven Program in C for the following Array operations

a. Creating an Array of N Integer Elements

b. Display of Array Elements with Suitable Headings

c. Inserting an Element (ELEM) at a given valid Position (POS)

d. Deleting an Element at a given valid Position(POS)

e. Exit.

Support the program with functions for each of the above operations

ABOUT THE EXPERIMENT:

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

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

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

ALGORITHM:

Step 1: Start.

Step 2: Read N value.

Step 3: Read Array of N integer elements

Step 4: Print array of N integer elements.

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

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

Step 7: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
int a[20];
int n,val,i,pos;

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

int main()
{
int choice;
while(choice)
{
printf("\n\n--------MENU          \n");
printf("1.CREATE\n");
printf("2.DISPLAY\n");
printf("3.INSERT\n");
printf("4.DELETE\n");
printf("5.EXIT\n");
printf(" ");
```

```
printf("\nENTER YOUR CHOICE:\t");
scanf("%d",&choice);
switch(choice)
{
case 1: create(); break;
case 2: display(); break;
case 3: insert(); break;
case 4: delete(); break;
case 5: exit(0); break;
default: printf("\nInvalid choice:\n");
         break;
}
}
return 0;
}
//creating an array
void create()
{
printf("\nEnter the size of the array elements:\t");
scanf("%d",&n);
printf("\nEnter the elements for the array:\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
}
//displaying an array elements void display()
{
int i;
printf("\nThe array elements are:\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
}
//inserting an element into an array

void insert()
{
printf("\nEnter the position for the new element:\t");
scanf("%d",&pos);
printf("\nEnter the element to be inserted :\t");
scanf("%d",&val);
for(i=n-1;i>=pos;i--)
{
a[i+1]=a[i];
}
a[pos]=val;
n=n+1;
}

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

```
a[i]=a[i+1];
}
n=n-1;
printf("\nThe deleted element is =%d",val);
}
```

Sample Output 1

```
--------MENU-----------
1.      CREATE
2.      DISPLAY
3.      INSERT
4.      DELETE
5.      EXIT
----------------------
ENTER YOUR CHOICE: 1
Enter the size of the array elements: 3 Enter the elements for the array:
10 25 30

ENTER YOUR CHOICE: 2
The array elements are:
10 25 30

ENTER YOUR CHOICE: 3
Enter the position for the new element: 1 Enter the element to be inserted : 20

ENTER YOUR CHOICE: 2
The array elements are:
10 20 25 30

ENTER YOUR CHOICE: 4
Enter the position of the element to be deleted: 3 The deleted element is =30

enter your choice: 5
Exit
```

Sample Output 2

```
--------MENU-----------
        1. CREATE
        2. DISPLAY
        3. INSERT
        4. DELETE
        5. EXIT
----------------------
ENTER YOUR CHOICE: 1
Enter the size of the array elements: 3 Enter the elements for the array:
    20  20 20

ENTER YOUR CHOICE: 2
The array elements are:
20 20 20

ENTER YOUR CHOICE: 3
Enter the position for the new element: 1 Enter the element to be inserted : 10
```

ENTER YOUR CHOICE: 2
The array elements are:
20 10 20 20

ENTER YOUR CHOICE: 4
Enter the position of the element to be deleted: 3 The deleted element is =20

enter your choice: 5
Exit

EXPERIMENT: 2

2. Design, Develop and Implement a Program in C for the following operations on Strings

a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)

b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR.

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

ABOUT THE EXPERIMENT:

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null. The following declaration and initialization create a string consisting of the word "Hello".

To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

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

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

char greeting[] = "Hello";

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

strcpy(s1, s2);	Copies string s2 into string s1.
strcat(s1, s2);	Concatenates string s2 onto the end of string s1.
strlen(s1);	Returns the length of string s1.
strcmp(s1, s2);	Returns 0 if s1 and s2 are the same; less than 0 if s1s2.
strchr(s1, ch);	Returns a pointer to the first occurrence of character ch in string s1.
strstr(s1, s2);	Returns a pointer to the first occurrence of string s2 in string s1.

ALGORITHM:

Step 1: Start.

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

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

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

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

Step 6: Stop.

PROGRAM CODE:

```
#include<stdio.h>
 void main()
{
char STR[100],PAT[100],REP[100],ans[100];
int i,j,c,m,k,flag=0;
```

```
printf("\nEnter the MAIN string: \n"); gets(STR);
printf("\nEnter a PATTERN string: \n"); gets(PAT);
printf("\nEnter a REPLACE string: \n"); gets(REP);
i = m = c = j = 0; while ( STR[c] != '\0') {

// Checking for Match
if ( STR[m] == PAT[i]
) { i++; m++;
flag=1;
if ( PAT[i] == '\0')
{
//copy replace string in ans string
for(k=0; REP[k] != \0';k++,j++)
ans[j] = REP[k];
i=0;
c=m;
}
}
else //mismatch
{
ans[j] = STR[c];
j++; c++;
m = c; i=0;
}
}
if(flag==0)
{
printf("Pattern doesn't found!!!");
}
else
{
ans[j] = '\0';
printf("\nThe RESULTANT string is:%s\n" ,ans);
}
}
```

Sample Output 1

Enter the MAIN string:
good morning

Enter a PATTERN string: morning

Enter a REPLACE string:
evening
The RESULTANT string is: good evening

Sample Output 2

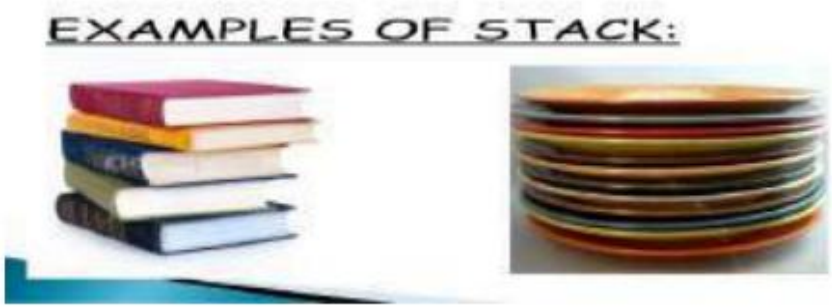
Enter the MAIN string: hi vcet

Enter a PATTERN string: bye
Enter a REPLACE string: hello
Pattern doesn't found!!

EXPERIMENT: 3

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

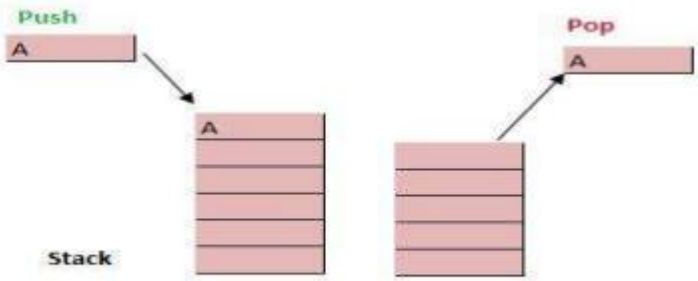
ABOUT THE EXPERIMENT: A stack is an abstract data type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack.



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

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

Below given diagram tries to depict a stack and its operations –



A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.

Basic Operations:

- push() - pushing (storing) an element on the stack.
- pop() -removing (accessing) an element from the stack. To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks;
- peek() − get the top data element of the stack, without removing it.
- isFull() − check if stack is full.
- isEmpty() − check if stack is empty.

ALGORITHM:

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

PROGRAM CODE:

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#define max_size 5
int stack[max_size],top=-1,flag=1;
int i,temp,item,rev[max_size],num[max_size];
void push();
void pop();
void display();
void pali();
int main()
{
int choice;
printf("\n\n--------STACK OPERATIONS--------");
printf("1.Push\n");
printf("2.Pop\n"); printf("3.Palindrome\n"); printf("4.Display\n"); printf("5.Exit\n");
printf(" ");
while(1)
{
printf("\nEnter your choice:\t");
scanf("%d",&choice);
switch(choice)
{
case 1:  push();break;
case 2:  pop();
         if(flag)
         printf("\nThe poped element: %d\t",item);
         temp=top; break;
case 3:  pali();
         top=temp; break;
case 4:  display(); break;
case 5:  exit(0); break;
```

```
default: printf("\nInvalid choice:\n"); break;
}
}
//return 0;
}
void push() //Inserting element into the stack
{
if(top==(max_size-1))
{
printf("\nStack Overflow:");
}
else
{
printf("Enter the element to be inserted:\t");
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
temp=top;
}
void pop()        //deleting an element from the stack
{
if(top==-1)
{
printf("Stack Underflow:");
flag=0;
}
else
{
item=stack[top];
top=top-1;
}
}
void pali()
{ i=0;
if(top==-1)
{
printf("Push some elements into the stack first\n");
}
else
{
while(top!=-1)
{
rev[top]=stack[top]; pop();
}
top=temp; for(i=0;i<=temp;i++)
{
if(stack[top--]==rev[i])
{
if(i==temp)
{
printf("Palindrome\n"); return;
}
}
}
printf("Not Palindrome\n");
}
}
```

```
void display()
{
int i; top=temp;

if(top==-1)
{
printf("\nStack is Empty:");
}
else
{
printf("\nThe stack elements are:\n" );
for(i=top;i>=0;i--)
{
printf("%d\n",stack[i]);
}
}
}
```

Sample Output 1

linux:~/dslab # gedit stack.c linux:~/dslab # cc stack.c

--------STACK OPERATIONS----------
1.Push 2.Pop
3.Palindrome 4.Display 5.Exit

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

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

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

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

Enter your choice: 2 The poped element: 5

Enter your choice: 4 The stack elements are: 1
2
1
Enter your choice: 3 Numbers= 1
Numbers= 2
Numbers= 1 reverse operation : reverse array : 1
2
1
check for palindrome : It is palindrome number

Enter your choice: 5
Exit

Sample Output 2

--------STACK OPERATIONS----------
1.Push 2.Pop

3.Palindrome 4.Display 5.Exit

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

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

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

Enter your choice: 1
Enter the element to be inserted: 50

Enter your choice: 2 The poped element: 50

Enter your choice: 4 The stack elements are: 10
20
10
Enter your choice: 3 Numbers= 1
Numbers= 2
Numbers= 1 reverse operation : reverse array : 1
2
1
check for palindrome : It is palindrome number

Enter your choice: 5
Exit

EXPERIMENT: 4

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

ABOUT THE EXPERIMENT:

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

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

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

Examples of Infix, Prefix, and Postfix

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

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

Step 1. Reverse the infix expression.

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

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

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

Step 3. Convert expression to postfix form.

Step 4. Reverse the expression.

+*+A^BCD^E

Step 5. Result

+*+A^BCD^E5

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

ALGORITHM:

Step 1: Start.
Step 2: Read an infix expression with parenthesis and without parenthesis.
Step 3: convert the infix expression to postfix expression.
Step 4: Stop

PROGRAM CODE:

```
#define SIZE 50 /* Size of Stack */
#include <ctype.h>
#include <stdio.h>
char s[SIZE];
int top = -1; /* Global declarations */
push(char elem) /* Function for PUSH operation */
{
s[++top] = elem;
}

char pop() /* Function for POP operation */
{
return (s[top--]);
}

int pr(char elem) /* Function for precedence */
{
switch (elem)
{
case '#': return 0;
 case '(': return 1;
case '+':
case '-': return 2;
case '*': case '/':
case '%': return 3;
case '^': return 4;
}
}
void main() /* Main Program */
{
char infx[50], pofx[50], ch, elem;

int i = 0, k = 0;
printf("\n\nRead the Infix Expression ? ");
scanf("%s", infx);
push('#');
while ((ch = infx[i++]) != '\0')
{
if (ch == '(') push(ch);
else if (isalnum(ch))
pofx[k++] = ch;
 else if (ch == ')')
{
while (s[top] != '(')
pofx[k++] = pop();
elem = pop(); /* Remove ( */
}
else /* Operator */
```

```
{
while (pr(s[top]) >= pr(ch))
pofx[k++] = pop();
push(ch);
}
}
while (s[top] != '#') /* Pop from stack till empty */
 pofx[k++] = pop();
pofx[k] = '\0'; /* Make pofx as valid string */
printf("\n\nGiven Infix Expn: %s Postfix Expn: %s\n", infx, pofx);
}
```

Sample Output 1

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

Sample Output 2

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

EXPERIMENT: 5

5. Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks

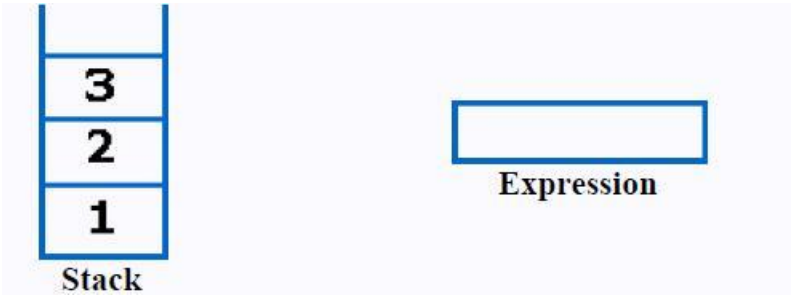
ABOUT THE EXPERIMENT:

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

Postfix/Suffix Expression: Operators are written after their operands. Ex: XY+
In normal algebra we use the infix notation like **a+b*c**. The corresponding postfix notation is **abc*+**

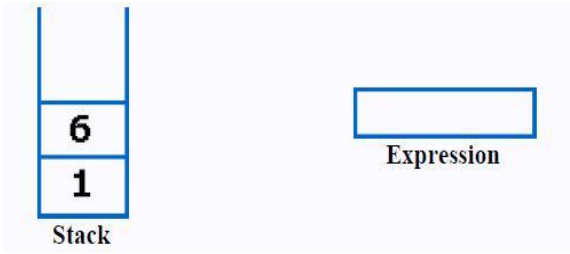
Example: Postfix String: 123*+4-

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

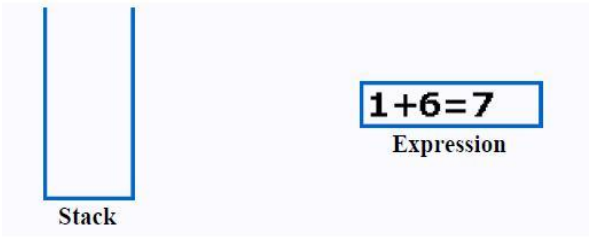


Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.

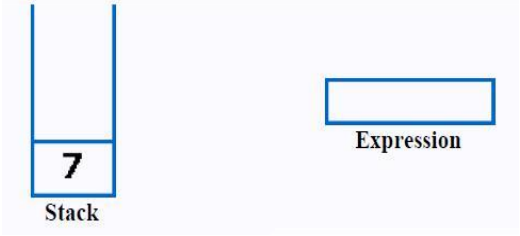
The value of the expression(2*3) that has been evaluated(6) is pushed into the stack.



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



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



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

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

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

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

End result: Postfix String:123*+4- Result: 3

ALGORITHM:

Step 1: Start.

Step 2: Read the postfix/suffix expression.

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

Step 4: Stop.

PROGRAM CODE 5A:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define MAX 20
struct stack
{
int top;
float str[MAX];
}s;//stack
char postfix[MAX];//postfix
void push(float);
float pop();
int isoperand(char);
float operate(float,float,char);
int main()
{
int i=0;
printf("Enter Expression:");
scanf("%s",postfix);
float ans,op1,op2;
while(postfix[i]!='\0')
{
if(isoperand(postfix[i]))
push(postfix[i]-48);
else
{
op1=pop();
op2=pop();
ans=operate(op1,op2,postfix[i]);
push(ans);
printf("%f %c %f = %f\n",op2,postfix[i],op1,ans);
}
i++;
}
printf("%f",s.str[s.top]);
getch();
}

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

```
void push(float x)
{
if(s.top==MAX-1)
printf("Stack is full\nStack overflow\n");
else
{
s.top++;
s.str[s.top]=x;
}
}
float pop()
{
if(s.top==-1)
{
printf("Stack is emplty\nSTACK UNDERFLOW\n");
getch();
}
else
{
s.top--;
return s.str[s.top+1];
}
}
float operate(float op1,float op2,char a)
{
switch(a)
{
case '+' : return op2+op1;
case '-' : return op2-op1;
case '*' : return op2*op1;
case '/' : return op2/op1;
case '^' : return pow(op2,op1);
}

}
```

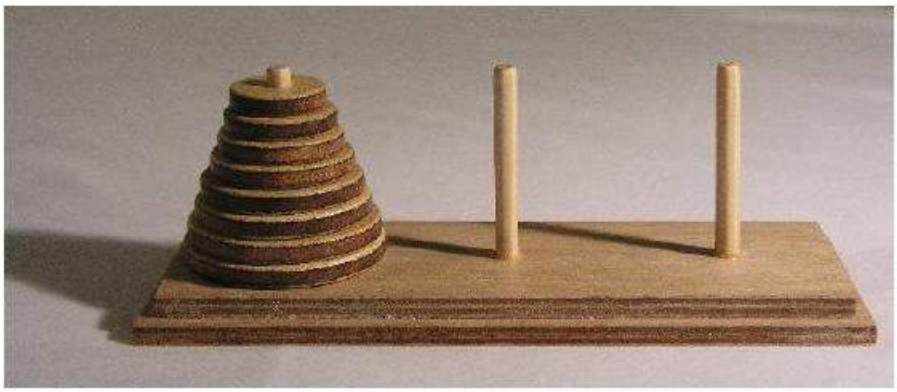
// **5b. Towers of Hanoi**

Solving Tower of Hanoi problem with n disks.

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

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.
  With three disks, the puzzle can be solved in seven moves. The minimum number of moves

required to solve a Tower of Hanoi puzzle is 2^n - 1, where *n* is the number of disks.



ALGORITHM:

Step 1: Start.
Step 2: Read N number of discs.
Step 3: Move all the discs from source to destination by using temp rod.
Step 4: Stop.

PROGRAM CODE: 5B

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

Sample Output 1

Insert a postfix notation :: 22^32*+
Result :: 10

Sample Output 2

Insert a postfix notation :: 23+

Result :: 5

Sample Output 1

Enter the number of discs: 3

Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total Number of moves are: 7

EXPERIMENT: 6

6. Design, Develop and Implement a menu driven Program in C for the following operations

on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

a. Insert an Element on to Circular QUEUE

b. Delete an Element from Circular QUEUE

c. Demonstrate*Overflow*and*Underflow*situations on Circular QUEUE d. Display the status of Circular QUEUE

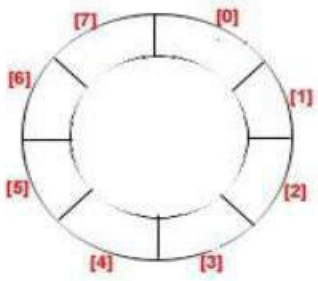
e. Exit

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

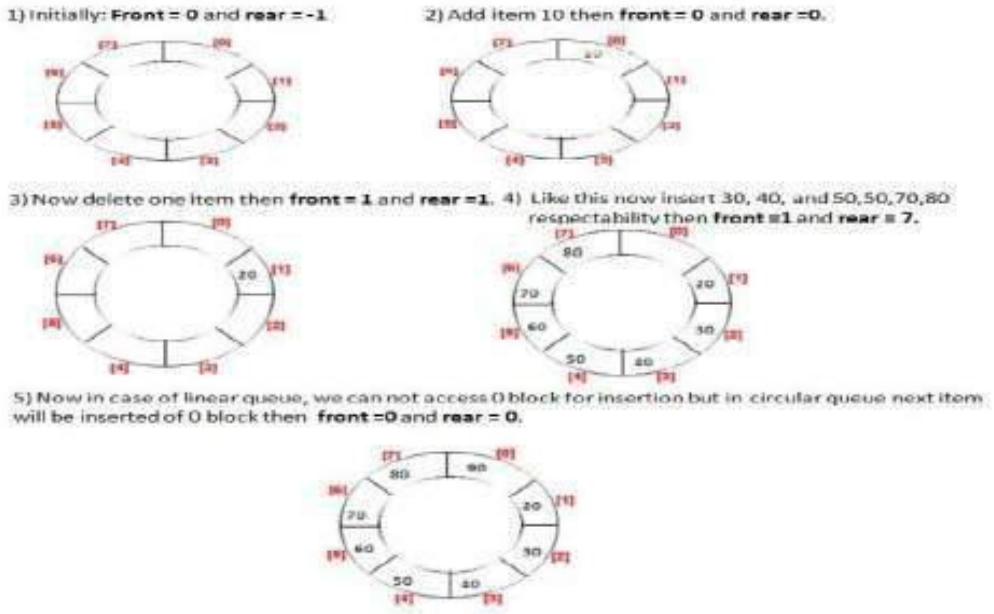
ABOUT THE EXPERIMENT:

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

A circular queue looks like



Consider the example with Circular Queue implementation:



ALGORITHM:

Step 1: Start.

Step 2: Initialize queue size to MAX.

Step 3: Insert the elements into circular queue. If queue is full give a message as ‘queue is overflow” Step 4: Delete an element from the circular queue. If queue is empty give a message as ‘queue is underflow’.

Step 5: Display the contents of the queue.

Step 6: Stop.

PROGRAM CODE:

```
#include <stdio.h>
 #include <conio.h>
#define SIZE 5
int CQ[SIZE];
int front=-1;
int rear=-1, ch;
int IsCQ_Full();
int IsCQ_Empty();
void CQ_Insert(int );
void CQ_Delet();
void CQ_Display();
void main()
{
printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
while(1)
{
int ele;
printf("Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: if(IsCQ_Full())
          printf("Circular Queu Overflow\n");
          else
         {
           printf("Enter the element to be inserted\n");
           scanf("%d",&ele); CQ_Insert(ele);
         }
          break;
case 2: if(IsCQ_Empty())
           printf("Circular Queue Underflow\n");
           else
           CQ_Delet();
          break;
case 3: if(IsCQ_Empty())
            printf("Circular Queue Underflow\n");
           else
           CQ_Display();
          break;
case 4: exit(0);
}
}
```

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

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

Sample Output 1

```
Circular Queue operations
1.insert
2.delete
3.display
4.exit
Enter your choice:1
```

Enter element to be insert:10
Enter your choice:1
Enter element to be insert:20
Enter your choice:1

Enter element to be insert:30

Enter your choice:3 10 20 30
rear is at 30 front is at 10
Enter your choice:2 Deleted element is:10

Enter your choice:3 20 30
rear is at 30 front is at 20
Enter your choice:4
Exit

Sample Output 2

Circular Queue operations
1.insert
2.delete
3.display
4.exit
Enter your choice:1
Enter element to be insert:1000
Enter your choice:1
Enter element to be insert:2000
Enter your choice:1

Enter element to be insert:3000

Enter your choice:3
1000 2000 3000
rear is at 3000
front is at 1000
Enter your choice:2
Deleted element is:1000

Enter your choice:3
2000 3000
rear is at 3000
front is at 2000
Enter your choice:4
Exit

EXPERIMENT: 7

7. Design, Develop and Implement a menu driven Program in C for the following operations on

Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Branch, Sem, PhNo*

a. Create a SLL of N Students Data by using *front insertion*.

b. Display the status of SLL and count the number of nodes in it c. Perform Insertion and Deletion at End of SLL d. Perform Insertion and Deletion at Front of SLL

f. **Demonstrate how this SLL can be used as STACK and QUEUE f. Exit**

ABOUT THE EXPERIMENT:

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

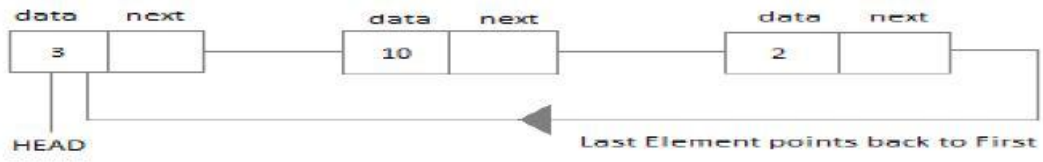
- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let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

Types of Linked List:

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

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

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



ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student's information)
Step 3: Create a singly linked list. (SLL)
Step 4: Display the status of SLL.
Step 5: Count the number of nodes.
Step 6: Perform insertion at front of list.
Step 7: Perform deletion at the front of the list.
Step 8: Perform insertion at end of the list.
Step 9: Perform deletion at the end of the list.
Step 10: Demonstrate how singly linked list can be used as stack.
Step 11: Demonstrate how singly linked list can be used as queue.
Step 12: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int count=0;
struct stud
{
long long int ph;
int sem;
char name[15],usn[15],brnch[8];
struct stud *next;
}
*head=NULL,*tail=NULL,*temp=NULL,*temp1;
void create(long long int n,int s,char na[20],char u[15],char b[5])
{
if(head==NULL)
{
head=(struct stud*)malloc(1*sizeof(struct stud));
head->ph=n;
head->sem=s;
strcpy(head->name,na);
strcpy(head->usn,u);
strcpy(head->brnch,b);
head->next=NULL;
 tail=head;
count++;
}
else
```

```
{
temp=(struct stud*)malloc(1*sizeof(struct stud));
temp->ph=n;
temp->sem=s;
strcpy(temp->name,na);
strcpy(temp->usn,u);
strcpy(temp->brnch,b);
temp->next=NULL;
tail->next=temp;
tail=temp;
count++;
}
}
void display()
{
temp1=head;
if(temp1==NULL)
{
printf("\nlist is empty\n");
}
else
{
printf("student details are as follows:\n");

while(temp1!=NULL)
{
printf(" \n");
printf("NAME:%s\nUSN:%s\nBRANCH:%s\nSEM:%d\nPHONE NO.:%lld\n",temp1-
>name,temp1->usn,temp1->brnch,temp1->sem,temp1->ph);
printf(" \n");
temp1=temp1->next;
}
printf("no. of nodes=%d\n",count);
}
}
void insert_head(long long int n,int s,char na[15],char u[15],char b[8])
{
temp=(struct stud*)malloc(1*sizeof(struct stud));
temp->ph=n;
temp->sem=s;
strcpy(temp->name,na);
strcpy(temp->usn,u);
strcpy(temp->brnch,b);
temp>next=hed;
head=temp;
count++;
}
void insert_tail(long long int n,int s,char na[15],char u[15],char b[8])
{
temp=(struct stud*)malloc(1*sizeof(struct stud));
 temp->ph=n;
temp->sem=s;
strcpy(temp->name,na);
strcpy(temp->usn,u);
strcpy(temp->brnch,b);
tail->next=temp;
temp->next=NULL;
tail=temp;
```

```
count++;
}
void delete_head()
{
temp1=head;
 if(temp1==NULL)
{
printf("list is empty\n");
}
else
 {
head=head->next;
printf("deleted node is:\n");
printf(" \n");
printf("NAME:%s\nUSN:%s\nBRANCH:%s\nSEM:%d\nPHONE NO.:%lld\n",temp1->name,
temp1->usn,temp1->brnch,temp1->sem,temp1->ph);
printf(" \n");
free(temp1);
count--;
}
}
void delete_tail()
{
temp1=head;
if(temp1==NULL)
{
printf("list is empty\n");
}
while(temp1->next!=tail)
{
temp1=temp1->next;
}
printf("deleted node is:\n"); printf("        \n");
printf("NAME:%s\nUSN:%s\nBRANCH:%s\nSEM:%d\nPHONE NO.:%lld\n",tail->name,tail-
>usn,tail->brnch,tail->sem,tail->ph); printf("       \n");
free(tail);
tail=temp1;
tail->next=NULL;
count--;
}
void main()
{
int choice;
long long int ph; int sem;
char name[20],usn[15],brnch[5]; printf("--------MENU    \n");
printf("1.create\n2.Insert from head\n3.Insert from tail\n4.Delete from head\5.Delete from
tail\n6.display\n7.exit\n");
printf(" \n");
while(1)
{
printf("enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
        printf("enter the name usn branch sem phno. of the student respectively\n");
        scanf("%s%s%s%d%lld",name,usn,brnch,&sem,&ph);
        create(ph,sem,name,usn,brnch);
```

```
        break;

case 2:
        printf("enter the name usn branch sem phno. of the student respectively\n");
        scanf("%s%s%s%d%lld",name,usn,brnch,&sem,&ph);
        insert_head(ph,sem,name,usn,brnch);
        break;

case 3:
        printf("enter the name usn branch sem phno. of the student respectively\n");
        scanf("%s%s%s%d%lld",name,usn,brnch,&sem,&ph);
        insert_tail(ph,sem,name,usn,brnch);
        break;

case 4:
        delete_head();
        break;
case 5:
        delete_tail();
        break;

case 6:
        display();
        break;

case 7:
        exit(0);
        default:printf("invalid option\n");
}
}
}
```

Sample Output 1

```
–----------MENU----------------------
1 – create a SLL of n emp 2 - Display from beginning 3 - Insert at end
4 - delete at end 5 - Insert at beg 6 - delete at beg 7 - exit
-------------------------------------------------

Enter choice : 1
Enter no of students : 2
Enter usn,name, branch, sem, phno of student : 007 vijay CSE 3 121 Enter usn,name, branch, sem,
phno of student : 100 yashas CSE 3 911

Enter choice : 2
Linked list elements from begining : 100 yashas CSE 3
911 007 vijay CSE 3 121 No of students = 2


Enter choice : 3
Enter usn,name, branch, sem, phno of student : 001 raj CSE 3 111

Enter choice : 2
Linked list elements from begining :
100 yashas CSE 3 911
007 vijay CSE 3 121
001 raj CSE 3 111
No of students = 3
```

Enter choice : 4 001 raj CSE 3 111
Enter choice : 2
Linked list elements from begining :
100 yashas CSE 3 911
007 vijay CSE 3 121 No of students = 2

Enter choice : 5
Enter usn,name, branch, sem, phno of student : 003 harsh cse 3 111

Enter choice : 2
Linked list elements from begining : 003 harsh cse 3 111
100 yashas CSE 3 911
007 vijay CSE 3 121 No of students = 3
Enter choice : 6 003 harsh cse 3 111

Enter choice : 2
Linked list elements from begining : 100 yashas CSE 3
911 007 vijay CSE 3 121 No of students = 2

Enter choice : 7
Exit

Sample Output 2

–----------MENU----------
1 – create a SLL of n emp 2 - Display from beginning 3 - Insert at end
4 - delete at end 5 - Insert at beg 6 - delete at beg 7 - exit
---

Enter choice : 1
Enter no of students : 1
Enter usn,name, branch, sem, phno of student : 009 suhas ISE 8 9854125422

Enter choice : 2
Linked list elements from begining : 009 suhas ISE 8
9854125422
No of students = 1

Enter choice : 3
Enter usn,name, branch, sem, phno of student : 001 raj CSE 3 111

Enter choice : 2
Linked list elements from begining :
009 suhas ISE 8 9854125422
001 raj CSE 3 111
No of students = 2

Enter choice : 4
001 raj CSE 3 111
Enter choice : 2
Linked list elements from begining :
009 suhas ISE 8 9854125422
No of students = 1

Enter choice : 5
Enter usn,name, branch, sem, phno of student : 009 suhas ISE 8 9854125422

003 harsh cse 3 111

Enter choice : 2
Linked list elements from begining : 009 suhas ISE 8 9854125422
003 harsh cse 3 111

No of students = 2
Enter choice : 6
003 harsh cse 3 111

Enter choice : 2
Linked list elements from begining : 003 harsh cse 3 111
No of students = 1

Enter choice : 7
Exit

EXPERIMENT: 8

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

a. Create a DLL of N Employees Data by using *end insertion*.

b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL

g. **Demonstrate how this DLL can be used as Double Ended Queue f. Exit**

ABOUT THE EXPERIMENT:

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

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

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

ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student's information)
Step 3: Create a doubly linked list. (DLL)
Step 4: Display the status of DLL.
Step 5: Count the number of nodes.
Step 6: Perform insertion at front of list.
Step 7: Perform deletion at the front of the list.
Step 8: Perform insertion at end of the list.
Step 9: Perform deletion at the end of the list.
Step 10: Demonstrate how doubly linked list can be used as double ended queue.
Step 11: Stop.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
 struct Enode
{
char ssn[15];
char name[20];
char dept[5];
char designation[10];
 int salary;
long long int phno;
struct Enode *left;
struct Enode *right;
}*head=NULL;
struct Enode *tail,*temp1,*temp2;
void create(char [],char [],char [],char [],int ,long long int);
void ins_beg(char [],char [],char [],char [],int ,long long int);
void ins_end(char [],char [],char [],char [],int ,long long int);
void del_beg();
void del_end();
void display();
int count=0;
void main()
{
int choice;
char s[15],n[20],dpt[5],des[10];
int sal;
long long int p;

printf("1.Create\n2.Display\n3.Insert at beginning\n4.Insert at End\n5.Delete at beginning\n6.Delete
at End\n7.Exit\n");
while(1)
{
printf("\nEnter your choice\n");
scanf("%d",&choice); switch(choice)
{
case 1:
        printf("Enter the required data(Emp no,Name,Dept,Desig,sal,phone\n");
        scanf("%s%s%s%s%d%lld",s,n,dpt,des,&sal,&p);
        create(s,n,dpt,des,sal,p);
        break;

case 2:
        display();
        break;

case 3:
        printf("Enter the required data (Emp no,Name,Dept,Desig,sal,phone\n");
        scanf("%s%s%s%s%d%lld",s,n,dpt,des,&sal,&p);
        ins_beg(s,n,dpt,des,sal,p);
        break;

case 4:
        printf("Enter the required data(Emp no,Name,Dept,Desig,sal,phone\n");
        scanf("%s%s%s%s%d%lld",s,n,dpt,des,&sal,&p);
        ins_end(s,n,dpt,des,sal,p);
```

```
        break;

case 5:
        del_beg();
        break;

case 6:
        del_end();
        break;

case 7:
        exit(0);
}
}
}
void create(char s[15],char n[20],char dpt[5],char des[10],int sal,long long int p)
{
if(head==NULL)
{
head=(struct Enode *)malloc(1*sizeof(struct Enode));
strcpy(head->ssn,s);
strcpy(head->name,n);
strcpy(head->dept,dpt);
strcpy(head->designation,des);
head->salary=sal;

head->phno=p;
head->left=NULL;
head->right=NULL;
 tail=head;
}
else
{
temp1=(struct Enode *)malloc(1*sizeof(struct Enode));
strcpy(temp1->ssn,s);
strcpy(temp1->name,n);
strcpy(temp1->dept,dpt);
strcpy(temp1->designation,des);
temp1->salary=sal;
temp1->phno=p;
tail->right=temp1;
temp1->right=NULL;
temp1->left=tail;
tail=temp1;

}
}
void display()
{
temp1=head;
printf("Employee Details          \n");
while(temp1!=NULL)
{
printf(" \n");
printf("%s\n%s\n%s\n%s\n%d\n%lld\n",temp1->ssn,temp1->name,temp1->dept,temp1-
>designation,temp1->salary,temp1->phno); printf("        ");
temp1=temp1->right;
}
```

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

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

Sample Output 1

```
----------------MENU---------------------
1.Create 2.Display
3.Insert at beginning 4.Insert at End 5.Delete at beginning 6.Delete at End 7.Exit
-----------------------------------------
Enter choice : 1
Enter no of employees : 2
Enter ssn,name,department, designation, salary and phno of employee : 1 RAJ SALES
MANAGER 15000 911
Enter ssn,name,department, designation, salary and phno of employee : 2 RAVI HR ASST 10000 123
```

Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
No of employees = 2 Enter choice : 3
Enter ssn,name,department, designation, salary and phno of employee : 3 RAM
MARKET MANAGER 50000 111

Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
3 RAM MARKET MANAGER 50000.000000 111
No of employees = 3 Enter choice : 4
3 RAM MARKET MANAGER 50000.000000 111
Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
No of employees = 2

Enter choice : 5
Enter ssn,name,department, designation, salary and phno of employee : 0 ALEX EXE TRAINEE
2000 133
Enter choice : 2
Linked list elements from begining :
0 ALEX EXE TRAINEE 2000.000000 133
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
No of employees = 3

Enter choice : 6

0 ALEX EXE TRAINEE 2000.000000 133

Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123

No of employees = 2
Enter choice : 7
Exit

Sample Output 2

----------MENU----------
1.Create 2.Display
3.Insert at beginning 4.Insert at End 5.Delete at beginning 6.Delete at End 7.Exit
---
Enter choice : 1
Enter no of employees : 1
Enter ssn,name,department, designation, salary and phno of employee : 1 RAJ SALES
MANAGER 15000 911

Enter choice : 2
Linked list elements from begining :

```
1 RAJ SALES MANAGER 15000.000000 911

No of employees = 1
Enter choice : 3
Enter ssn,name,department, designation, salary and phno of employee :
3 RAM MARKET MANAGER 50000 111

Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAM MARKET MANAGER 50000.000000 111
No of employees = 2

Enter choice : 4
3 RAM MARKET MANAGER 50000.000000 111

Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
No of employees = 2

Enter choice : 5
Enter ssn,name,department, designation, salary and phno of employee : 0 ALEX EXE TRAINEE
2000 133
Enter choice : 2
Linked list elements from begining :
3       ALEX EXE TRAINEE 2000.000000 133
4       RAJ SALES MANAGER 15000.000000 911
5 RAVI HR ASST 10000.000000 123
No of employees = 3

Enter choice : 6

0       ALEX EXE TRAINEE 2000.000000 133

Enter choice : 2
Linked list elements from begining :
3       RAJ SALES MANAGER 15000.000000 911
4 RAVI HR ASST 10000.000000 123

No of employees = 2
Enter choice : 7
Exit
```

EXPERIMENT: 9

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

a. Represent and Evaluate a Polynomial P(x,y,z) = $6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$

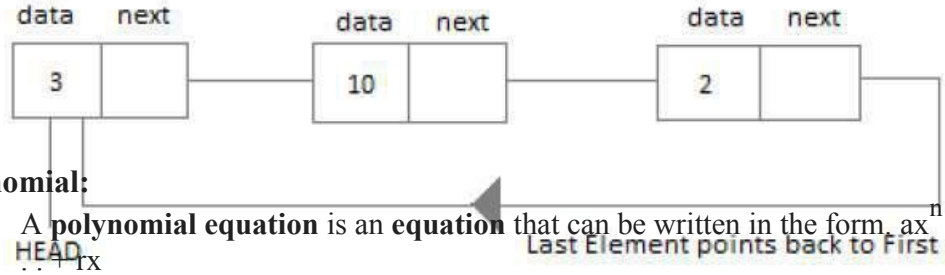
b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

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

ABOUT THE EXPERIMENT:

Circular Linked List:

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

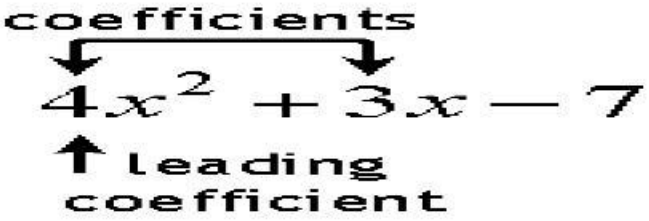


Polynomial:

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

As with **polynomials** with one variable, you must pay attention to the rules of exponents and the order of operations so that you correctly **evaluate** an expression with two or more variables. **Evaluate** $x^2 + 3y^3$ for x = 7 and y = −2. Substitute the given values for x and y. **Evaluate** $4x^2y – 2xy^2 + x – 7$ for x = 3 and y = −1.

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



In the above example, the coefficient of the leading term is 4; the coefficient of the second term is 3; the constant term doesn't have a coefficient.

Here are the steps required for Evaluating Polynomial Functions:

Step 1: Replace each x in the expression with the given value.

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

Example 1 – Given $f(x) = -2x^2 + 5x - 7$, find $f(3)$.

Step 1: Replace each x in the expression with

the given value. In this case, we replace each x with 3.

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

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

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

Here are the steps required for addition of two polynomials.

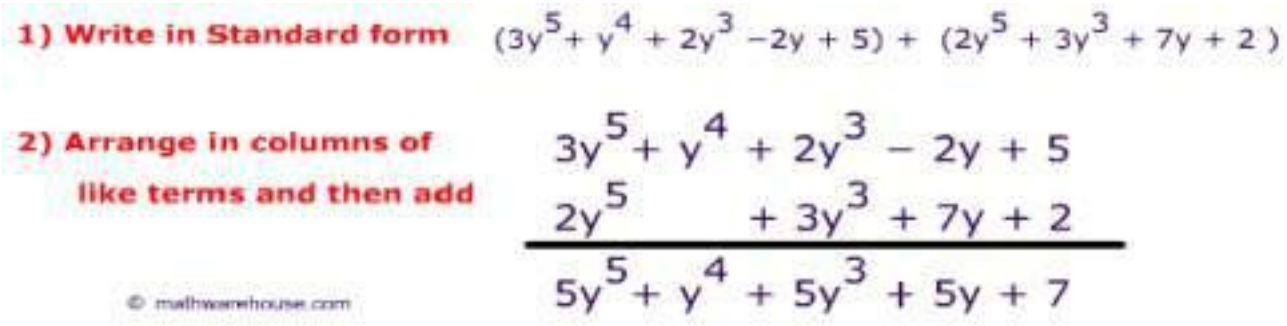
Step 1

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

Step 2

- Arrange the like terms in columns and add the like terms

Example 1: *Let's find the sum of the following two polynomials* $(3y^5 - 2y + y^4 + 2y^3 + 5)$ and $(2y^5 + y^3 + 2 + 7)$



ALGORITHM:

Step 1: Start.
Step 2: Read a polynomial.
Step 3: Represent the polynomial using singly circular linked list.
Step 4: Evaluate the given polynomial
Step 5: Read two polynomials and find the sum of the polynomials.
Step 6: Stop

PROGRAM CODE:

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

```
struct node
{
int coeff;
int expo;
struct node *ptr;
};
struct node *head1,*head2,*head3, *temp,*temp1,*temp2,*temp3,*list1,*list2,*list3;
struct node *dummy1,*dummy2;
void create_poly1(int , int);
void create_poly2(int , int);
void display();
void add_poly();
void eval_poly(int );
 int n,ch;
int c,e,i;
void main()
{
int x; list1=list2=NULL;
printf("1.Create first polynomial\n2.Create Second Polynomial\n3.Display both the
polynomials\n"); printf("4.Add Polynomials\n5.Evaluate a Polynomial\n6.Exit\n");
while(1)
{
printf("Enter choice\n");
scanf("%d",&ch); s
witch(ch)
{
case 1: printf("Enter the number of terms\n");
          scanf("%d",&n);
          printf("Enter coefficient & power of each term\n");
          for(i=0;i<n;i++)
          {
          scanf("%d%d",&c,&e);
          create_poly1(c,e);
          }
          break;

case 2: printf("Enter the number of terms\n");
          scanf("%d",&n);
          printf("Enter coefficient & power of each term\n");
          for(i=0;i<n;i++)
          {
          scanf("%d%d",&c,&e);
          create_poly2(c,e);
          }
          break;

case 3:
          display();
          break;

case 4:
          add_poly();
          break;

case 5:
          printf("Enter the value for x\n");
          scanf("%d",&x);
          eval_poly(x);
```

```
            break;

case 6:exit(0);
}
}
}
void create_poly1(int c, int e)
{
Dummy           1=(struct node*)malloc(1*sizeof(struct node));
 dummy1->coeff=0;
dummy1->expo=0;
dummy1->ptr=list1;
if(list1==NULL)
{
list1->coeff=c;
list1->expo=e;

list1->ptr=list1; head1=list1;
head1->ptr=dummy1;
}
else
{
temp=(struct node*)malloc(1*sizeof(struct node));
temp->coeff=c;
temp->expo=e;
head1->ptr=temp;
temp->ptr=dummy1;
 head1=temp;
}
}
void create_poly2(int c, int e)
{
dummy2=(struct node*)malloc(1*sizeof(struct node));
dummy2->coeff=0;
dummy2->expo=0;
dummy2->ptr=list2;
if(list2==NULL)
{
list2=(struct node*)malloc(1*sizeof(struct node));
list2->coeff=c;
list2->expo=e;
list2->ptr=list2;
head2=list2;
head2->ptr=dummy2;
}
else
{
temp=(struct node*)malloc(1*sizeof(struct node));
temp->coeff=c;
temp->expo=e;
head2->ptr=temp;
temp->ptr=dummy2;
head2=temp;
}
}
void add_poly()

{
```

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

if(temp1==dummy1)
{
while(temp2!=dummy2)
{
temp=(struct node*)malloc(1*sizeof(struct node));
 temp->coeff=temp2->coeff;
temp->expo=temp2->expo;
 temp->ptr=list3;
head3->ptr=temp;
head3=temp;
temp2=temp2->ptr;
}
}
if(temp2==dummy2)
{
while(temp1!=dummy1)
{
temp=(struct node*)malloc(1*sizeof(struct node));
```

```
temp->coeff=temp1->coeff;
temp->expo=temp1->expo;
temp->ptr=list3;
head3->ptr=temp;
 head3=temp;
temp1=temp1->ptr;
}
}
}
void display()
{
temp1=list1;
temp2=list2;
temp3=list3;
printf("\nPOLYNOMIAL 1:");
while(temp1!=dummy1)
{
printf("%dX^%d+",temp1->coeff,temp1->expo);
temp1=temp1->ptr;
}
printf("\b ");
printf("\nPOLYNOMIAL 2:");

while(temp2!=dummy2)
{
printf("%dX^%d+",temp2->coeff,temp2->expo);
temp2=temp2->ptr;
}
printf("\b ");
printf("\n\nSUM OF POLYNOMIALS:\n"); while(temp3->ptr!=list3)
{
printf("%dX^%d+",temp3->coeff,temp3->expo);
temp3=temp3->ptr;
}
printf("%dX^%d\n",temp3->coeff,temp3->expo);
}
void eval_poly(int x)
{
int result=0;
temp1=list1;
 temp2=list2;
while(temp1!=dummy1)
{
result+=(temp1->coeff)*pow(x,temp1->expo);
temp1=temp1->ptr;
}
printf("Polynomial 1 Evaluation:%d\n",result);
result=0;
while(temp2!=dummy2)
{
result+=(temp2->coeff)*pow(x,temp2->expo);
temp2=temp2->ptr;
}
printf("Polynomial 2 Evaluation:%d\n",result);
}
```

Sample Output 1

```
----------<< MENU >>----------
Polynomial Operations : 1.Add    2.Evaluate   3.Exit
-------------------------------------------------
Enter your choice==>1
Enter no of terms of polynomial==>3
Enter coef & expo==>
4
3
Enter coef & expo==> 2
2
Enter coef & expo==> 5
1
The polynomial is==>5x^(1) + 2x^(2) + 4x^(3)
Enter no of terms of polynomial==>3
Enter coef & expo==> 4
1
Enter coef & expo==> 3
2
Enter coef & expo==> 5
3
The polynomial is==>4x^(1) + 3x^(2) + 5x^(3)
Addition of polynomial==>
The polynomial is==>9x^(1) + 5x^(2) + 9x^(3)
Enter your choice==>2
Enter no of terms of polynomial==>3
Enter coef & expo==>
1
Enter coef & expo==> 4
2
Enter coef & expo==> 5
4
The polynomial is==>3x^(1) + 4x^(2) + 5x^(4)

Enter the value of x=2 Value of polynomial=102
Enter your choice==>3 exit
```

EXPERIMENT: 10

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

ABOUT THE EXPERIMENT:

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

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

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

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

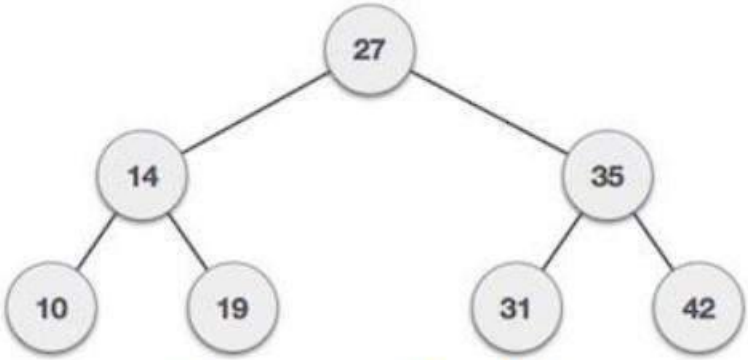


Fig: An example of BST

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

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

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

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

ALGORITHM:

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

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct BST
{
int data;
struct BST *left;
struct BST *right;
};
typedef struct BST NODE;
NODE *node;
NODE* createtree(NODE *node, int data)
{
if (node == NULL)
{
NODE *temp;
temp= (NODE*)malloc(sizeof(NODE));
temp->data = data;
temp->left = temp->right = NULL;
return temp;
}
if (data < (node->data))
{
node->left = createtree(node->left, data);
}
else if (data > node->data)
{
node -> right = createtree(node->right, data);
}
return node;
}
NODE* search(NODE *node, int data)
{
if(node == NULL)
printf("\nElement not found");
else if(data < node->data)
{
node->left=search(node->left, data);
}
else if(data > node->data)
{
node->right=search(node->right, data);
}
else
printf("\nElement found is: %d", node->data);
 return node;
```

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

}
NODE* findMin(NODE *node)
{
if(node==NULL)
{
return NULL;
}
if(node->left)
return findMin(node->left);
else
return node;
}
NODE* del(NODE *node, int data)
{
NODE *temp;
if(node == NULL)
{
printf("\nElement not found");
}
else if(data < node->data)
{
node->left = del(node->left, data);
}
else if(data > node->data)
{
node->right = del(node->right, data);
}
else
{
/* Now We can delete this node and replace with either minimum element in the right sub tree or
```

```
maximum element in the left subtree */
if(node->right && node->left)
{
/* Here we will replace with minimum element in the right sub tree */
temp = findMin(node->right);
node -> data = temp->data;
/* As we replaced it with some other node, we have to delete that node */
node -> right = del(node->right, temp->data);
}
else
{
/* If there is only one or zero children then we can directly remove it from the tree and connect its
parent to its child */
temp = node;
if(node->left == NULL)
node = node->right;
else if(node->right == NULL)
node = node->left;
free(temp);    /* temp is longer required */
}
}
return node;
}
void main()
{
int data, ch, i, n;
NODE *root=NULL;
clrscr();
while (1)
{
printf("\n1.Insertion in Binary Search Tree");
printf("\n2.Search Element in Binary Search Tree");
printf("\n3.Delete Element in Binary Search Tree");
printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
printf("\nEnter your choice: "); scanf("%d", &ch);
switch (ch)
{
case 1:
        printf("\nEnter N value: " );
        scanf("%d", &n);
        printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
        for(i=0; i<n; i++)
        {
        scanf("%d", &data);
        root=createtree(root, data);
        }
        break;

case 2:
        printf("\nEnter the element to search: ");
        scanf("%d", &data);
        break;

case 3:
        printf("\nEnter the element to delete: ");
        scanf("%d", &data);
        root=del(root, data);
        break;
```

```
case 4:
        printf("\nInorder Traversal: \n");
        inorder(root);
        break;
case 5:
        printf("\nPreorder Traversal: \n");
        preorder(root);
        break;

case 6:
        printf("\nPostorder Traversal: \n");
        postorder(root);
        break;

case 7:
        exit(0);

        default : printf("\nWrong option");
        break;
}
}
}
```

Sample Output 1

Program For Binary Search Tree
1. Create
2. Search
3. Recursive Traversals
4. Exit

Enter your choice :1 Enter The Element 15
Want To enter More Elements?(1/0)1 Enter The Element 25

Want To enter More Elements?(1/0)1 Enter The Element 35

Want To enter More Elements?(1/0)1

Enter The Element 45

Want To enter More Elements?(1/0)1 Enter The Element 5
Want To enter More Elements?(1/0)1 Enter The Element 7
Want To enter More Elements?(1/0)0 Enter your choice :2
Enter Element to be searched :7

The 7 Element is Present Parent of node 7 is 5
1. Create
2. Search
3. Recursive Traversals 4.Exit

Enter your choice :2
Enter Element to be searched :88 The 88 Element is not Present

Enter your choice :3

The Inorder display : 5 7 15 25 35 45
The Preorder display : 15 5 7 25 35 45
The Postorder display : 7 5 45 35 25 15 Enter your choice :4

EXPERIMENT: 11

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

a. Create a Graph of N cities using Adjacency Matrix.

b. Print all the nodes reachable from a given starting node in a digraph using BFS method

c. Check whether a given graph is connected or not using DFS method

ABOUT THE EXPERIMENT:

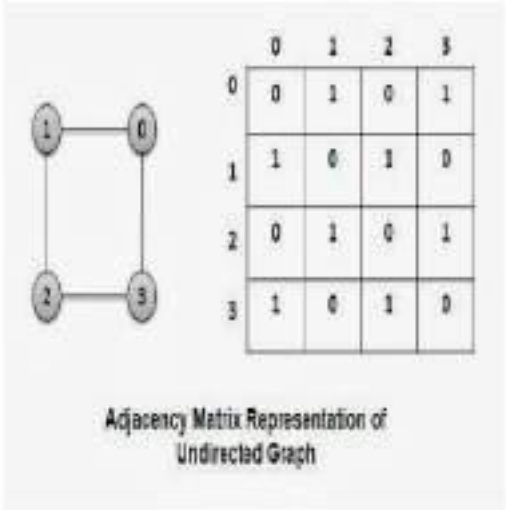
Adjacency Matrix In graph theory, computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a (0, 1)-matrix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

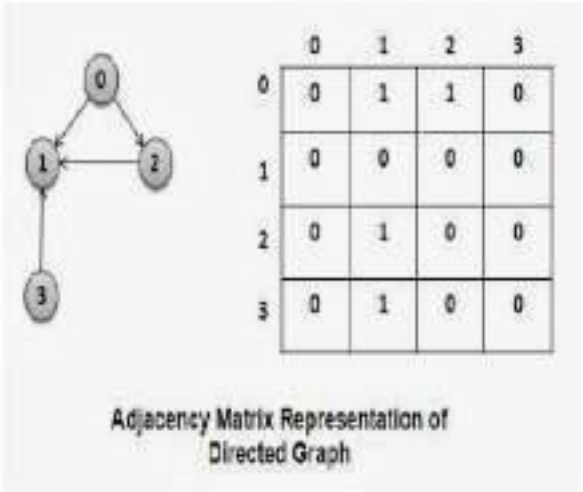
A graph G = (V, E) where v= {0, 1, 2, . . .n-1} can be represented using two dimensional integer array of size n x n.

a[20][20] can be used to store a graph with 20 vertices.
a[i][j] = 1, indicates presence of edge between two vertices i and j.
a[i][j] = 0, indicates absence of edge between two vertices i and j.

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

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

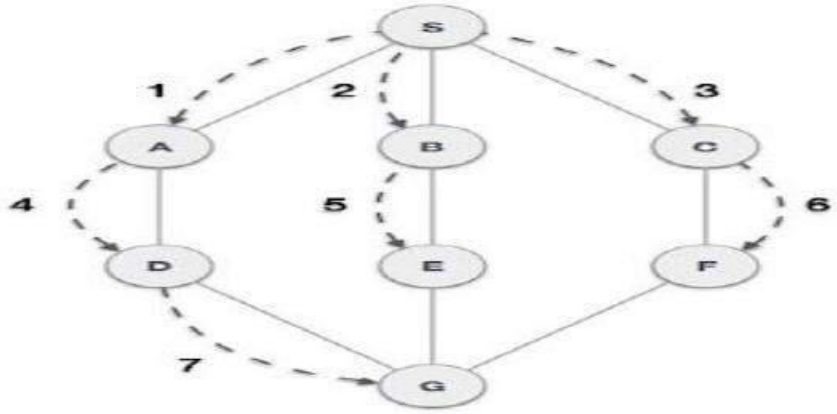




BFS

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

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



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

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

DFS

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

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

Algorithm

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

white: vertex is unvisited;

gray: vertex is in progress;

black: DFS has finished processing the vertex.

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

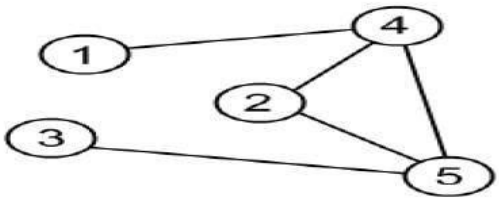
Initially all vertices are white (unvisited). DFS starts in arbitrary vertex and runs as follows:

5.Mark vertex **u** as gray (visited).

6.For each edge **(u, v)**, where **u** is white, run depth-first search for **u** recursively.

7.Mark vertex **u** as black and backtrack to the parent.

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



Source graph.

ALGORITHM:

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

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
int a[20][20],q[20],visited[20],reach[10],n,i,j,f=0,r=-1,count=0;
void bfs(int v)
{
for(i=1;i<=n;i++)
 if(a[v][i] && !visited[i])
q[++r]=i;
if(f<=r)
{
visited[q[f]]=1;
bfs(q[f++]);
}
}
void dfs(int v)
{
int i; reach[v]=1;
for(i=1;i<=n;i++)
{
if(a[v][i] && !reach[i])
{
printf("\n %d->%d",v,i);
count++;
dfs(i);
}
}
}
void main()
{
int v, choice;
printf("\n Enter the number of vertices:");
scanf("%d",&n);

for(i=1;i<=n;i++)
{
q[i]=0;
visited[i]=0;
```

```
}
for(i=1;i<=n-1;i++)
reach[i]=0;

printf("\n Enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("1.BFS\n 2.DFS\n 3.Exit\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
        printf("\n Enter the starting vertex:");
        scanf("%d",&v);
        bfs(v);
        if((v<1)||(v>n))
        {
        printf("\n Bfs is not possible");
        }
        else
        {
        printf("\n The nodes which are reachable from %d:\n",v);
        for(i=1;i<=n;i++)
        if(visited[i])
        printf("%d\t",i);
        }
        break;

case 2:
        dfs(1);
        if(count==n-1)
        printf("\n Graph is connected");
        else
        printf("\n Graph is not connected");
        break;

case 3: exit(0);
        }
}
```

Sample Output 1

```
Enter the number of vertices:5
Enter graph data in matrix form:
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 0
0 1 0 0 0
1.      BFS
2.      DFS
3.      Exit 2

1->2
2->3
3->4
2->5
```

Graph is connected
Enter the number of vertices:5
Enter graph data in matrix form:
0 1 0 1 0
1 0 1 0 0
0 1 0 1 0
1 0 1 0 0
0 0 0 0 0

1. BFS
2. DFS
3. Exit 2

1->2
2->3
3->4
Graph is not connected
Enter the number of vertices:5

Enter graph data in matrix form:
0 1 1 0 0
0 0 0 1 0
0 0 0 0 0
0 0 1 0 0
0 0 1 0 0
1. BFS
2. DFS
3. Exit 1
Enter the starting vertex:1
The nodes which are reachable from 1: 2 3 4

Enter graph data in matrix form: 0 1 1 0 0
0 0 0 1 0
0 0 0 0 0
0 0 1 0 0
0 0 1 0 0
1. BFS
2. DFS
3. Exit 1
Enter the starting vertex:0 BFS is not possible

EXPERIMENT: 12

12. Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a

Program in C that uses Hash function H: K → L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing

ALGORITHM:

Step 1: Start.
Step 2: Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F.
Step 3: Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.
Step 4: Let the keys in K and addresses in L are Integers
Step 5: Hash function H: K ®L as H(K)=K mod m (remainder method)
Step 6: Hashing as to map a given key K to the address space L, Resolve the collision (if any) is using linear probing.
Step 7: Stop.

PROGRAM CODE:

```
#include <stdio.h>
 #include <stdlib.h>
 #define MAX 100
int create(int);
void display (int[]);
void main()
{
int a[MAX],num,key,i;
int ans=1;
printf(" collision handling by linear probing : \n");
for (i=0;i<MAX;i++)
{
a[i] = -1;
}
do
{
printf("\n Enter the data");
scanf("%4d", &num);
key=create(num);
linear_prob(a,key,num);
printf("\n Do you wish to continue ? (1/0) ");
scanf("%d",&ans);
}while(ans);
display(a);
}
int create(int num)
{
int key;
key=num%100;
 return key;
```

```
}

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

a[i] = num;
flag=1;
break;
}
 i++;
}
}
}
void display(int a[MAX])
{
int i,choice;
printf("1.Display ALL\n 2.Filtered Display\n");
scanf("%d",&choice);
if(choice==1)
{
printf("\n the hash table is\n");
for(i=0; i<MAX; i++)
```

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

Sample Output 1

collision handling by linear probing :

Enter the data1234

Do you wish to continue ? (1/0) 1 Enter the data2548
Do you wish to continue ? (1/0) 1 Enter the data3256
Do you wish to continue ? (1/0) 1 Enter the data1299
Do you wish to continue ? (1/0) 1 Enter the data1298
Do you wish to continue ? (1/0) 1 Enter the data1398
Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING

Do you wish to continue ? (1/0) 0 1.Display ALL
2.Filtered Display 2
the hash table is 0 1398
34 1234
48 2548
56 3256
98 1298
99 1299

MODEL VIVA QUESTIONS & ANSWERS

Title of the Practical: Program to search an element of array using linear search

Q1 Define searching process?
A1 searching is the process of finding an element within the list of elements stored in any order or randomly.

Q2 How many types of searching are there?
A2 There is basically two types of searching:-linear search and Binary search.

Q3Define: linear search?
A3 In linear search, we access each elements of an array one by one sequentially and see weather it is desired element or not.

Q4 Why binary search method is more efficient then liner search?
A4 It is because less time is taken by linear search to search an element from the sorted list of elements.

Q5 Efficiency of linear search ?
A5 The time taken or the number of comparisons made in searching a record in a search table determines the efficiency of the technique.

Q6 What do you understand by the term "linear search is unsuccessful"?
A6 search will be unsuccessful if all the elements are accessed and the desired elements are not found.

Q7 What is worse case?
A7 In the worse case, the no. of average case we may have to scan half of the size of the array (n/2).

Q8 What is the drawback of linear search?
A8 There is no requisite for the linear Search.

Q9 During linear search, when the record is present in first position then how many comparisons are made ?
A9 Only one comparison.

Q10 During linear search, when the record is present in last position then how many comparisons are made?
A10 n comparisons have to be made.

Title of the Practical: Program to reverse the element of array. Insertion and deletion on array at specified position.

Q1 What is an array & how many types of arrays represented in memory?
A1 An array is a structured datatype made up of a finite, fixed size, collection of homogeneous ordered elements. Types of array: One-Dimentional array, Two-Dimentional array, Multi-Dimentional array.

Q2 How can be declared an array?
A2 data_type var_name[Expression];

Q3 How can be insert an element in an array?
A3 Insertion of a new element in an array can be done in two ways:
* Insertion at the end of array. * Insertion at required position.

Q4 How can be delete an element in an array?
A4- Deleting an element at the end of an array presents no difficulties, but deleting element somewhere in the middle of the array.

Q5 How many types of implementation of a two-dimentional array?
A5 * Row-major implementation * Column-major implementation

Q6 What is array of pointers?
A6 Array of pointer refers to homogeneous collection of pointer that is collection of pointer of same datatype.

Q7 What are limitations of array?
A7 * These are static structures. * It is very time consuming.

Q8 Where the elements of the array are stored respectively in successive?

A8 Memory locations.

Q9 How can merge two arrays?
A9 Simplest way of merging two arrays is that first copy all elements of one array into a third empty array and the copy all the elements of other array into third array.

Q10 What is the operations of array?
A10 Insertion, Deletion, Traversing, Merging.

Title of the Practical: Program based on structure union

Q1 What are structures?
A1 Structures are used to store different types.

Q2 What are arrays?
A2 Arrays are used to store similar data types.

Q3 Is array of structures possible?
A3 Is array of structures are possible.

Q4 How structures are declared?
A4 Declaration: struct book { char name; Float price; Int pages; };

Q5 Why we use functions?
A5 Writing functions avoids rewriting the same code over and over.

Q6 Name some library functions?
A6 printf(), scanf() are examples of library function.

Q7 What are user defined functions?
A7 Functions declared by the user are called user defined functions.

Q8 Explain call by value?
A8 When the value is passed in the function is called call by value.

Q9 Explain call by reference?
A9 When the address of the value is passed is called call by reference.

Q10 Why we used keyword „break"?
A10 When break is encountered inside any loop, control automatically passes to the first statement after the loop.

Title of the Practical: Program to implement PUSH and POP operation on stack.

Q1 What is stack?
A1 Stack is an ordered collection of elements like array.

Q2 What are the operations performed on stack?
A2 Push for insertion and pop for deletion.

Q3 What is push operation?
A3 When an element is inserted into the stack is called push operation

Q4 What is pop operation.
A4 When an element is deleted from the stack is called pop operation.

Q5 How stacks are implemented?
A5 Stacks are implemented in two ways: static implementation –array Dynamic implementation –pointers.

Q6 What are the applications of stack?
A6 infix , post fix and prefix notations are the applications of stack.

Q7 What is recursion.
A7 Recursion is defined as function calling itself.

Q8 What are the different types of stack
A8 Direct: a system calls itself from within itself. Indirect: two functions mutually calls one another

Q9 Define "Top of stack"
A9 Pointer indicating the top element of the stack.

Q10 Is stack is prinmitive or non primitive data structure ?
A10 Non primitive data structure.

Title of the Practical: Program based on infix to prefix and post fix notation.

Q1 What is Stack ?
A1 Stack is ordered collection of element like arrays out it has a special feature that deletion and insertion of element can be done only from one end called top of stack .It is also called LIFO(last in first out).

Q2 Application of Stack ?
A2 Infix Prefix Postfix

Q3 Terms used in Stack ?
A3 > Context > Stack frames > Maxsize

Q4 Explain infix in Stack ?
A4 The operator is written in between the operands. Ex:- A+B

Q5 Explain Postfix in Stack ?
A5 The operator is written ofter the operands.It is also called suffix notation. Ex:- AB+

Q6 Define operations on Stack ?
A6 The basic operation that can be performed on Stack are as follows: >PUSH >POP

Q7 Give postfix form for (A+B)*C/D
A7 AB+C*D/

Q8 Give postfix form for A+B/C-D
A8 ABC/+D

Q9 Give prefix form for A/B^C+D
A9 +/A^BCD

Q10 Give prefix form for A*B+C
A10 +*ABC

Title of the Practical: Program based on queue & their operations for an application

Q1 Define queue.
A1Queue is an homogeneous collection of elements. It is logically a first in first out (FIFO) type of list.Queue means a line.

Q2 In how many ways queue is implemented?
A2 Queues can be implemented in two ways: 1. static implementation (using array) 2. dynamic implementation (using pointers)

Q3 What is the rear end in a queue?
A3 in a queue new elements are added at one end called the rear end .

Q4 What is a front end?
A4 The existing elements in a queue are deleted from an end called the front end.

Q5 What is the value of front and rear end in an empty queue?
A5 Front =-1 and rear =-1.

Q6 Write an algorithm for deleting a node from a queue.
A6 1. If (front == null) Write queue is empty and exit Else Temp = start Value = temp ->no Start = start -> next Free (temp) Return(value) 2. exit.

Q7 What are the different variarions in a queue?
A7 Major variations are: 1. circular queue 2. double ended queue 3. priority queue.

Q8 What is the full form of dequeue?
A8 DEQUEUE : Double ended queue.It is another form of queue in which both insertion and deletion are performed at the either end.

Q9 When an element is added to the dequeue with n memory cells ,what happens to LEFT or RIGHT.
A9 If the element is added an the left , then LEFT is decreases by 1 (mod n) . IF the element is added on the right , then RIGHT is increase by 1 (mod n)

Q10 What are the applications of queue.
A10 Applications are: 1. round robin technique for processor scheduling 2. All types of customer services(eg. RTC) 3. Printer server routines.

Title of the Practical: Program based on the implementation of circular queue.

Q1 What is Circular Queue?
A1 A Circular Queue is one in which the insertion of a new element is done at the very first variation of the Queue if the last location of the Queue is full.

Q2 Why use of Circular Queue?
A2 A Circular Queue over comes the problem of un utilized space in linear Queue implemented as Array.

Q3 Explain Dequeue ?
A3 It is also a homogenous list of element in which insertion deletion of element are perform both the ends we insert element from the rear or from the front end.

Q4 What is Queue ?
A4 It is a non primitive linear data structure. In a Queue new element are added at the one end called rear end and the element are removed from another end called front end.

Q5 Variation in a Queue ?
A5 Circular Queue -> Dequeue ->Priority Queue

Q6 What is Priority Queue ?
A6 Priority Queue determines the order in which the exist in the Queue the highest Priority items are removed first.

Q7 Application of Queue ?
A7 1> Customer service center 2> Ticket counter 3> Queue is used for determine space complexity & time complexity.

Q8 What is Queue implemention?
A8 >> Static implemention(Array) >>Dynamic implemention(Pointer)

Q9 Define operations on queue?
A9 The basic operation that can be performed on queue are – 1>To insert an element in a Queue 2>To delete an element from the Queue

Q10 What is Stack ?
A10 Stack is ordered collection of element like arrays out it has a special feature that deletion and insertion of element can be done only from one end called top of stack .It is also called LIFO(last in first out).

Title of the Practical: Program based on list operations and its applications.

Q1 Define linked list.
A1 Linked list are special list of some data elements linked to one another .The logical ordering is represented by having each element pointing to the next element. Each element is called a node.

Q2 What does a node define?
A2 Node has two parts: INFO – it stores the information and POINTER – which points to the next element.

Q3 What are the different types of linked list?
A3 Linked list are of four types: 1. singly linked list 2. doubly linked list 3. circular linked list 4. circular doubly linked list.

Q4 What are the different operations performed on a linked list?
A4 Basic operations are: creation, insertion, deletion , traversing, searching , concatenation and display.

Q5 What are the advantages of linked list?
A5 Advantages are: 1. linked lists are dynamic data structures 2. Efficient memory utilizations 3. Insertion and deletions are easier and efficient

Q6 What is null pointer?

A6 The link field of the last node contains NULL rather than a valid address. It is a null pointer and indicates the end of the list.

Q7 What is external pointer?
A7 It is a pointer to the very first node in the linked list, it enables us to access the entire linked list.

Q8 What are the different notations used in a linked list.
A8 Node(p) : a node pointed to by the pointer p Data (p) : data of the node pointed by p Link (p) : address of the next node that follows yhe node pointed to by the pointer p.

Q9 What are advantages of circular linked list.
A9 1.Nodes can be accessed easily. 2. deletion of node is easier 3. concatenation and splitting of circular list is more efficient.

Q10 A doubly linked list contains how many fields?
A10 Doubly linked list consists of three fields: Data : contains the information Next: contains the address of the next node Prev: contains the address of the previous node.

Title of the Practical: Program based on pointers in C.

Q1 Which of the following abstract data types are NOT used by Integer Abstract Data type group?
A1. A)Short B)Int C)float D)long

Q2 What pointer type is used to implement the heterogeneous linked list in C?
A2 The answer is the void pointer. The heterogeneous linked list contains different data types in it's nodes and we need a link, pointer, to connect them. Since we can't use ordinary pointers for this, we use the void pointer. Void pointer is a generic pointer type, and capable of storing pointer to any type.

Q3: What issue do auto_ptr objects address?
A3: If you use auto_ptr objects you would not have to be concerned with heap objects not being deleted even if the exception is thrown.

Q4.: What is a dangling pointer?
A4: A dangling pointer arises when you use the address of an object after its lifetime is over. This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed.

Q5: What is the difference between a pointer and a reference?
A5: A reference must always refer to some object and, therefore, must always be initialized; pointers do not have such restrictions. A pointer can be reassigned to point to different objects while a reference always refers to an object with which it was initialized.

Q6: What is the difference between const char *myPointer and char *const myPointer?
A6: Const char *myPointer is a non constant pointer to constant data; while char *const myPointer is a constant pointer to non constant data.

Q7: From which is the pointer to object of a base class type compatible ?
A7: Pointers to object of a base class type is compatible with pointer to onject of a derived class . Therefore , we can use single pointer variable to point to objects of base class as well as derived class.

Q8: What is a this pointer?
A8: A special pointer known as this pointer stores the address of the object that is currently invoking a member function.

Q9:How are objects passed to functions?
A9: Objects can be passed to functions through call-by –value as well as call-by- reference mechanism.

Q10: Why is Arrow operator (“-> “) used?
A10: The arrow operator is used to access the public members of the class with a pointer to an object.

Title of the Practical: Implementation of tree using linked list.

Q1 What is a tree?
A1 A tree is a non linear data structure in which items are arranged in a sorted sequence.It is a finite set of one or more data items.

Q2 What is a root in a tree?

A2 A root is the first node in the hierarchical arrangement of data items.

Q3 What do you mean by degree of a tree?
A3 It is a maximum degree of nodes in a given tree .

Q4 What are non terminal nodes?
A4 Any node(Except the root node) is not zero is called non terminal node. Non terminal nodes are the intermediate nodes in traversing the given tree.

Q5 Who are called siblings in a tree?
A5 The children nodes of a given parent node are called siblings. They are also called brothers.

Q6 . During linear search, when the record is present somewhere in search table, then how many comparisons are made?
Ans- (n+1)/2.

Q7 How can access one-dimentional aray elements?
A7 array_name[index or subscript];

Q8 What is the traversing of an array?
A8 Traversing means to access all the elements of the array, starting from first element upto the last element in the array one-by-one.

Q9Explain the types of searching?
A9There are two types of searching: - 1> linear searching, 2> binary searching.

Q10 What is the linear search?
A10 We access each element of and see whether. It is desire element or not a search will be unsuccessful. If the all elements are access and the desired element is not found.