BMS Institute of Technology and Management

Yelahanka Bengaluru -560064

Department of Information Science and Engineering

DATA STRUCTURES LABORATORY Manual

(Subject Code: 17CSL38)

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

SEMESTER – III

2018-2019

BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

MISSION

Accomplish a stimulating learning environmentthrough high quality academic instruction, innovation and industry-institute interface.

ABOUT INSTITUTION

In view of the growing demand for technical education and with the goal of establishing a premier technical education on par with international standards, a new technical institution by name 'BMS Institute of Technology and Management' was established in 2002. Currently, BMSIT & M offers seven UG, three PG programs and Ph.D. /M.Sc. (Engg.) in seven disciplines. BMSIT & M considers research to be of equal importance as academics for the betterment of an institution. Research culture has been embraced well by the faculty members and research scholars at BMSIT and M. In this report, we present an overview of the research activities of Information Science and Engineering, BMSIT & M.

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

VISION

Emerge as centre of learning in the field of information science & engineering with technical competency to serve the society.

MISSION

To provide excellent learning environment through Balanced Curriculum, Best Teaching Methods, Innovation, Mentoring and Industry Institute Interaction.

ABOUT DEPARTMENT

The Department of Information Science and Engineering started in the Year 2010 with an approved intake of 60 and Enhanced to 120 from the academic year 2018-19. The Department has qualified and professionally dedicated faculty member practice OBE in the academic deliverables. The faculties have published research articles in various National, International, IEEE Conferences and Journals.

The department has modern laboratories to serve the teaching and research needs of the students as well as faculty members. The Department has been organizing conferences, workshops, expert lectures and student centric activities to encourage students and faculty to instil lifelong learning. Few of our students are working for consultancy projects along with few faculty members. The staffs are encouraged to attend the 10 days internship to bridge the gap between the academics and industry. The department has admirable research ambiance.

PROGRAMME EDUCATIONAL OBJECTIVES(PEO's)

PEO-1: Successful professional career in Information Science and Technology.

PEO-2: Pursue higher studies & research for advancement of knowledge in IT industry

PEO-3: Exhibit professionalism and team work with social concern.

PROGRAMME OUTCOMES(PO's)

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

PROGRAM SPECIFIC OUTCOMES(PSO's)

PSO-1: Apply the knowledge of information technology to develop software solutions.

PSO-2: Design and Develop hardware systems, manage and monitor resources in the product life cycle.

Table of Contents

/* 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.
***/**
Theory:-

Data Structure is defined as the way in which data is organized in the memory location. There are 2 types of data structures:

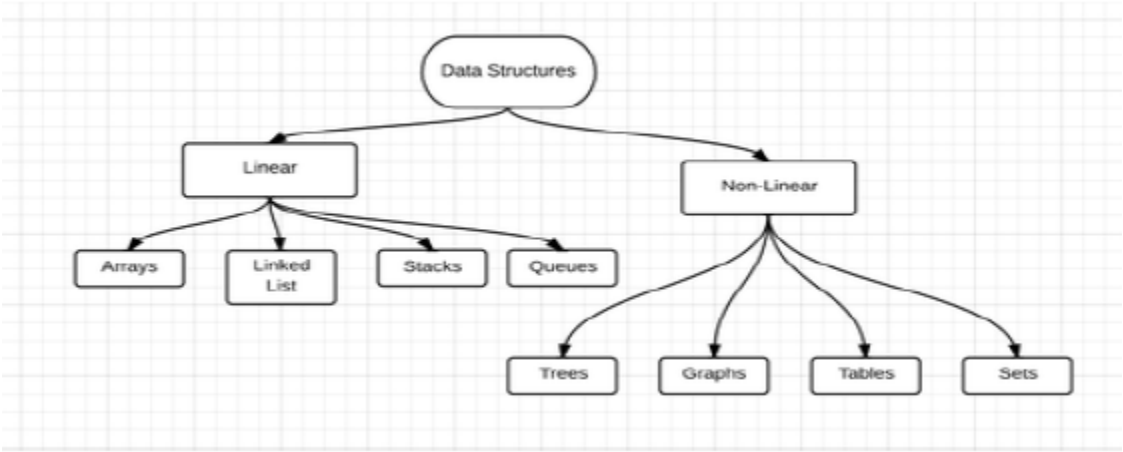


Fig 1.Classification of Data Structures

Linear Data Structure:

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

- Array
- Stack
- Queue
- Linked List

Non-Linear Data Structure:
Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.
The various types of non linear data structures are:

- Trees
- Graphs

//Implementation

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
int Array[MAX],n,pos=0,i;

void create_an_array()
{
        printf("Enter the size of the array: \n");
        scanf("%d",&n);
        printf("Enter %d elements \n",n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&Array[i]);
        }
        printf("\n Array Created \n");
}

//Diaplay the array elements with suitable headings////

void display(int a[])
{
        printf("Position\t Value\n");
        for(i=0;i<n;i++)
        {
                printf("\narray[%d] \t %d\n", i,a[i]);
        }

}
 // Insert an element at agiven position////
void insert()
{
  int element, pos;
  char ch;
  printf(" Enter the Position and Element where the element to be inserted \n");
  scanf("%d%d", &pos, &element);

        if( pos < 0 || pos > n-1)
                {
                        printf("Insertion is out of the boundary or filled");
                 }
        else
                 {
                          for(i=n-1;i>=pos;i--)
                          Array[i+1]=Array[i];

                    Array[pos]=element;
                    n=n+1;
                    printf("Element is inserted!");
```

```
                }
}

// deletion at a given posdition////

void deletion()
{
        int pos,val;
        printf("Enter the position where element need to be deleted :");
        scanf("%d", &pos);
        if( pos < 0 || pos > MAX-1)
        {
                printf("Insertion is out of the boundary or filled");
        }
         else
         {
                val=Array[pos];
                for(i=pos;i<n-1;i++)
                {
                        Array[i]=Array[i+1];
                }
        n=n-1;
        printf("Element %d is deleted from postion:%d \n",val, pos);
        }
}


 /// Main program////
void main()
{
        int choice;
        clrscr();
        while(1)
        {
                 printf("1.Create an Array\n2.Display\n3.Insert at position\n4.Delete at given
position\n5.Exit\n");
                 printf("Enter Your Choice:\n");
                 scanf("%d",&choice);
                 switch(choice)
                 {
                         case 1: create_an_array();break;
                         case 2: display(Array);break;
                         case 3: insert(); break;
                         case 4: deletion(); break;
                         case 5:exit(0);
                         default: printf("\n\n\n\tEnter proper Choice....");
                 }
        }
}
/* Output
Run 1:
```

```
1.Create an Array
2.Display
3.Insert at position
4.Delete at given position
5.Exit
Enter Your Choice:
1
Enter the size of the array:
3
Enter 3 elements
100 200 300
1.Create an Array
2.Display
3.Insert at position
4.Delete at given position
5.Exit
Enter Your Choice:
2

Position        Value

array[0]        100
array[1]        200
array[2]        300



*/
```

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

a. Read a main String (STR),
b. Pattern String (PAT) and a Replace String (REP)
c. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR.
Report suitable messages in case PAT does not exist in STR.
Support the program with functions for each of the above operations.
Don't use Built-in functions.
***/**

Theory:

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 s1<s2; greater than 0 if s1>s2.
- 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

//Implementation

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

char Mainstr[100],Pat[100],Replace[100],ans[100];
int i,j,c,m,k;

void read()
{
        printf("\nEnter a string \n");
        gets(Mainstr);
        printf("\nEnter a search string \n");
        flushall();
        gets(Pat);
        printf("\nEnter a replace string \n");
        flushall();
        gets(Replace);
}
void find_replace()
{
        i = m = c = j = 0;
        while ( Mainstr[c] != '\0')
        {
                if ( Mainstr[m] == Pat[i] ) // ..... matching
                {
                        i++;
                        m++;
                        if ( Pat[i] == '\0') //.....found occ
                        {
                                //.... copy replace string in ans string .....
                        for(k=0; Replace[k] != '\0';k++,j++) {
                                ans[j] = Replace[k];}
                        i=0;
                        c=m;
                        }
                }
                else //... mismatch
                {
                        ans[j] = Mainstr[c];
                        j++;
                        c++;
                        m = c;
                        i=0;
                }
        }
        ans[j] = '\0';
        printf("\nThe resultant string is\n%s" ,ans);
}
void main()
```

```
{
        clrscr();
        read();
        find_replace();
        getch();
}

/* Output

Enter a string
Respected Sir

Enter a search string
Respected

Enter a replace string
Dear

The resultant string is
Dear Sir
*/
```

/* 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
***/**

Theory:-

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


- 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.

//implementation

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 5
int Stack[MAX], top=-1;

void push()
{
        int ele;
        if(top<MAX-1)
        {
                printf("Enter the value to be inserted into the stack :");
                scanf("%d",&ele);
                Stack[++top] = ele;
        }
        else
         printf("Stack is Full");  // overflow condition

return;
}

void pop()
{
        if(top!=-1)
        {
                printf("the element deleted from the stack is : %d",Stack[top--]);
        }
        else
                 printf("Stack is Empty"); //Underflow condition

return;
}

void Palindrome()
{
        int i=0, len=top,flag=0,j=top;
        int stack1[5];

        while (j!=-1)
```

```
        {
            stack1[i]=Stack[j];
            i++;
            j--;
        }

        for (i = 0; i < len; i++)
        {
          if (stack1[i] != Stack[i])
          {       flag=1;
                  break;
          }
        }
         if(flag==0)
                   printf("It is a Palindrome \n");
         else
                   printf("It is not a palindrome");

}


void display()
{
        int i;
        if(top==-1)
        {
                printf("stack is empty");
        }
        else
        {
                printf("\ nElements in the stack are \n");
                for(i=0;i<=top;i++)
                printf("%d\n",Stack[i]);
        }
return;
}

void main()
{
        int choice;
        clrscr();
        while(1)
        {
                 printf("\n\n\n\t1.push...\t2.pop..\t3.palindrome...\t4.Display...\t5.Exit...");
                 printf("\n\n\n\tEnter Your Choice: ");
                 scanf("%d",&choice);
                 switch(choice)
                 {
                          case 1: push();break;
                          case 2: pop();break;
```

```
                        case 3: Palindrome(); break;
                        case 4: display(); break;
                        case 5:exit(0);
                        default: printf("\n\n\n\tEnter proper Choice....");
                }
        }
}
/* Output
Run 1:
********
1.push...       2.pop.. 3.palindrome... 4.Display...    5.Exit...
Enter Your Choice: 1
Enter the value to be inserted into the stack :1

1.push...       2.pop.. 3.palindrome... 4.Display...    5.Exit...
Enter Your Choice: 4
Elements in the stack are
1
2
1

Run 2:
*******
1.push...       2.pop.. 3.palindrome... 4.Display...    5.Exit...

Enter Your Choice: 2
The element deleted from the stack is : 1

1.push...       2.pop.. 3.palindrome... 4.Display...    5.Exit...
Enter Your Choice: 4
Elements in the stack are
1
2

Run 3:
*****

1.push...       2.pop.. 3.palindrome... 4.Display...    5.Exit...

Enter Your Choice: 4
Elements in the stack are
1
2
1

1.push...       2.pop.. 3.palindrome... 4.Display...    5.Exit...

Enter Your Choice: 3
It is a Palindrome
```

Run 4:
****

Enter Your Choice: 1
Enter the value to be inserted into the stack :1

1.push... 2.pop.. 3.palindrome... 4.Display... 5.Exit...

Enter Your Choice: 1
Enter the value to be inserted into the stack :2

1.push... 2.pop.. 3.palindrome... 4.Display... 5.Exit...

Enter Your Choice: 3
It is not a palindrome

1.push... 2.pop.. 3.palindrome... 4.Display... 5.Exit...
Enter Your Choice: 5

*/

/* 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.
***/**

Theory:-

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	A B C * +

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.

A+(B*C-(D/E-F)*G)*H

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

Step 4. Reverse the expression.
+*+A^BCD^E
Step 5. Result
+*+A^BCD^E5

//implementation

```
#define SIZE 50 /* Size of Stack */
#include <ctype.h>
#include <stdio.h>

char s[SIZE];

int top = -1; /* Global declarations */

void 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;
        clrscr();
        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);
getch();
}
/* Output
Run 1:

Read the Infix Expression ? a*b+c^d


Given Infix Expn: a*b+c^d Postfix Expn: ab*cd^+


*/
```

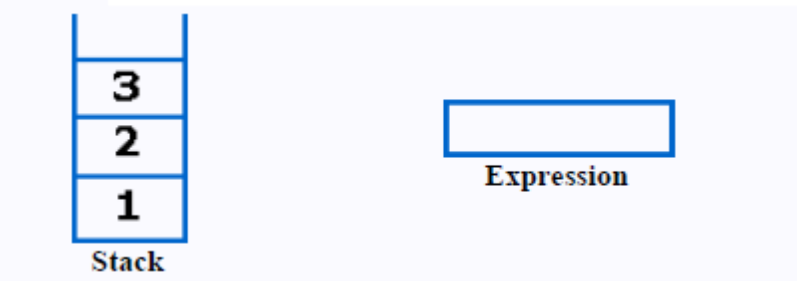
/* PROGRAM 5A
Design, Develop and Implement a Program in C for the following Stack Applications
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
***/**

Theory:-

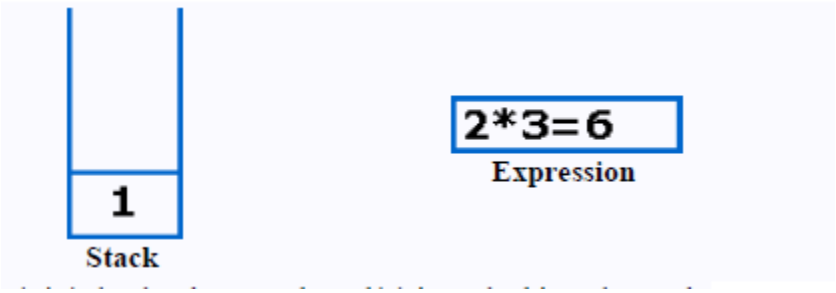
Postfix/Suffix Expression: Operators are written after their operands. Ex: XY+
In normal algebra we use the infix notation like **a+b*c**. The corresponding postfix notation is **abc*+**

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.

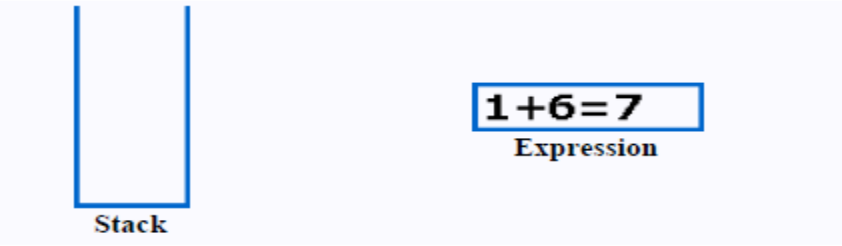


Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.

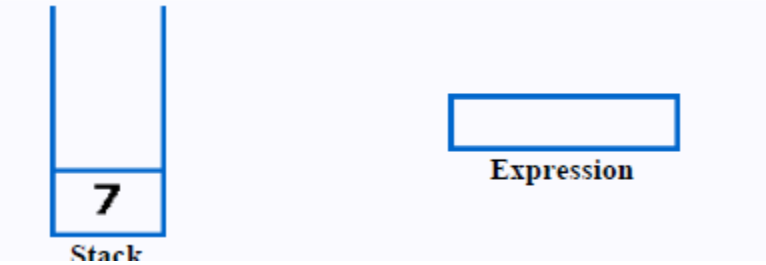


The value of the expression(2*3) that has been evaluated(6) is pushed into the stack.

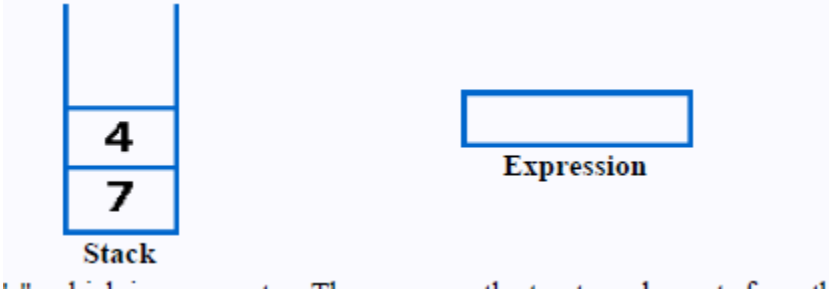
Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.



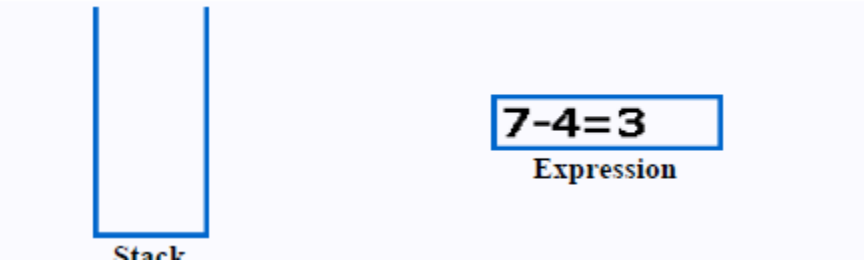
The value of the expression(1+6) that has been evaluated(7) is pushed into the stack.



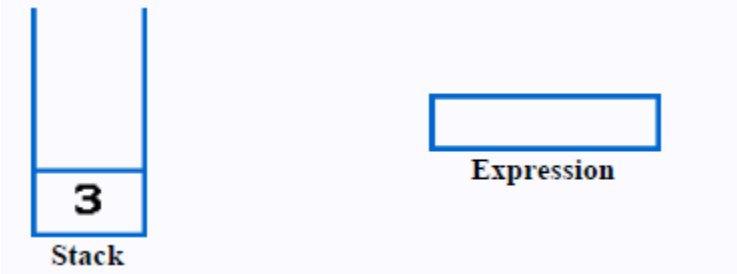
Next character scanned is "4", which is added to the stack.



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

//Implementation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#define MAX 50

int stack[MAX];
char post[MAX];
int top=-1;

void pushstack(int tmp);
void calculator(char c);
```

```
void main()
{
	int i;
	clrscr();
	printf("Insert a postfix notation :: ");
	scanf("%s",post);
	for(i=0;i<strlen(post);i++)
	{
		if(post[i]>='0' && post[i]<='9')
		{
			pushstack(i);
		}
		if(post[i]=='+' || post[i]=='-' || post[i]=='*' || post[i]=='/' || post[i]
	=='^')
		{
			calculator(post[i]);
		}
	}
	printf("\n\nResult :: %d",stack[top]);
getch();
}

void pushstack(int tmp)
{
	top++;
	stack[top]=(int)(post[tmp]-48);
}

void calculator(char c)
{
	int a,b,ans;
	b=stack[top];
	stack[top]='\0';
	top--;
	a=stack[top];
	stack[top]='\0';
	top--;
	switch(c)
	{
		case '+':
			ans=b+a;
		break;
		case '-':
			ans=b-a;
		break;
		case '*':
			ans=b*a;

		break;
		case '/':
```

```
                        ans=b/a;
                break;
                case '^':
                        ans=pow(b,a);
                break;
        default:
                ans=0;
        }
top++;
stack[top]=ans;
}
/* Output
Insert a postfix notation :: 22^32*+

Result :: 10
*/
```

/* 5b.c Tower of Honoi */

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.

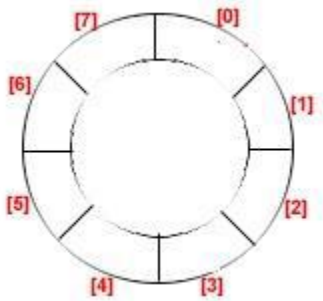
//implementation

```
#include<stdio.h>
void towers(int, char, char, char);
void main()
{
        int num;
        clrscr();
        printf("enter the number of disks : ");
        scanf("%d", &num);
        printf("the sequence of moves involved in the ower of honai are : \n");
        towers(num, 'A','C','B');
        getch();
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
        if(num == 1)
        {
                printf("\n MOVE DISK 1 FROM PEG %c TO PEG %c", frompeg, topeg);
        return;
        }
        towers( num -1,  frompeg,  auxpeg ,topeg);
        printf("\n MOVE DISK %d FROM PEG %c TO PEG %c", num,frompeg, topeg);
```

```
        towers( num -1,  auxpeg ,topeg ,frompeg );
}


/* Output
Run 1:

_________________
enter the number of disks : 2
the sequence of moves involved in the ower of honai are :

 MOVE DISK 1 FROM PEG A TO PEG B
 MOVE DISK 2 FROM PEG A TO PEG C
 MOVE DISK 1 FROM PEG B TO PEG C
*/
```

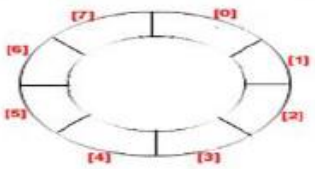
/*6 .Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for each of the above operations
***/**

Theory:-
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

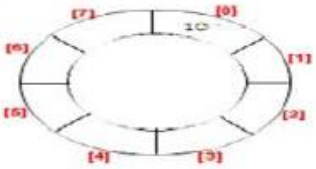


Consider the example with Circular Queue implementation:

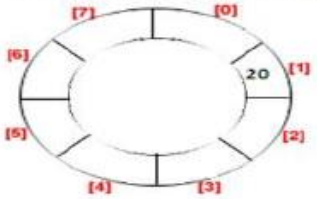
1) Initially: **Front = 0** and **rear = -1**



2) Add item 10 then **front = 0** and **rear =0.**

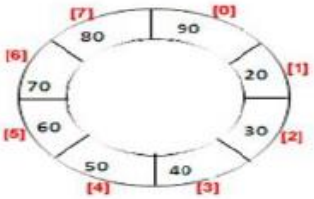


3) Now delete one item then **front = 1** and **rear =1.**



4) Like this now insert 30, 40, and 50,50,70,80 respectability then **front =1** and **rear = 7.**

5) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front =0** and **rear = 0.**



// Implementation

```
#include<stdio.h>
#include<conio.h>
#define size 5

char q[size];
int front=-1, rear=-1;
void insert();
void delet();
void display();

void main()
{
        int ch;
        clrscr();
        for(;;)
        {
                printf(" \n 1.Insert 2. Delete 3. Display 4.Exit\n");
                printf("\n Enter Your Choice \n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                insert();
                        break;
                        case 2:
                                delet();
```

```
                        break;
                        case 3:
                                display();
                        break;
                        case 4:
                                exit(0);
                        break;
                        default:
                                printf("Invalid Choice!");
                }
        }

}

void insert()
{
        char item;
        printf("Enter the element to be inserted to the queue \n");
        fflush(stdin);
        scanf("%c",&item);
        if(front==(rear+1)%size)
        printf(" \n Queue is Overflow ! \n");
        else
        {
                if(front==-1)
                {
                        front=rear=0;
                }
                else
                        rear=(rear+1)%size;
                        q[rear]=item;
        }
}
void display()
{
        int i;
        if(front==-1)
                printf("Queue is empty \n");
        else
        {
                for(i=front;i!=rear;i=(i+1)%size)
                {
                        printf("q[%d]=%c \n",i, q[i]);
                }
        printf("q[%d]=%c \n",i, q[i]);
        }
}
void delet()
{
        int del_item=0;
```

```
        //front=0;                                          '

        if(front==-1)
                printf("Queue is Empty!");
        else
        {
                //front=0;
                del_item=q[front];
                printf(" \nEleement deleted from the quees is '%c' at pos=%d \n",del_item,
front);
                if(front==rear)
                        front=rear=-1;
                else
                {

                front=(front+1)%size;
                }

        }
}
/*  Output
Enter the element to be inserted to the queue
d
 1.Insert 2. Delete 3. Display 4.Exit
 Enter Your Choice
1
Enter the element to be inserted to the queue
e
 1.Insert 2. Delete 3. Display 4.Exit
 Enter Your Choice
3
q[0]=a
q[1]=b
q[2]=c
q[3]=d
q[4]=e

Enter Your Choice
2
Eleement deleted from the quees is 'a' at pos=0
 1.Insert 2. Delete 3. Display 4.Exit
 Enter Your Choice
2
Eleement deleted from the quees is 'b' at pos=1
 1.Insert 2. Delete 3. Display 4.Exit

 Enter Your Choice
3
q[2]=c
q[3]=d
```

```
q[4]=e

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

 Enter Your Choice
1
Enter the element to be inserted to the queue
a
1.Insert 2. Delete 3. Display 4.Exit
 Enter Your Choice
1
Enter the element to be inserted to the queue
1
1.Insert 2. Delete 3. Display 4.Exit

 Enter Your Choice
3
q[2]=c
q[3]=d
q[4]=e
q[0]=a
q[1]=1

*/
```

/*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
e. Exit */

Theory:-

- 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.

//Implementation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct student
{
  char usn[12];
  char name[25];
  char branch[25];
  int sem;
  int phone_no;
  struct student *link;
};
typedef struct student STUD;

STUD *read_data()
{
    char usn[12],name[25],branch[25];
    int sem,phone_no;
    STUD *temp;
    temp=(STUD *)malloc(sizeof(STUD));
    printf("Enter the Students Details:\n");
    printf("Enter USN\n");
    scanf("%s",usn);
    strcpy(temp->usn,usn);
    printf("Enter Name\n");
    scanf("%s",name);
    strcpy(temp->name,name);
    printf("Enter Branch \n");
    scanf("%s",branch);
    strcpy(temp->branch,branch);
    printf("Enter Semester\n");
    scanf("%d",&sem);
    temp->sem=sem;
    printf("Enter Phone Number\n");
    scanf("%d",&phone_no);
    temp->phone_no=phone_no;
    temp->link=NULL;
    return temp;
```

```
}

STUD *insert_front(STUD *first)
{
        STUD *temp;
        temp=read_data();
        temp->link=first;
        return temp;
}

STUD *insert_end(STUD *first)
{
        STUD *temp,*prev;
        temp=read_data();
        if(first==NULL)
           return temp;
        prev=first;
        while(prev->link!=NULL)
           prev=prev->link;
        prev->link=temp;
        return first;
}

STUD *delete_front(STUD *first)
{
        STUD *cur;
        if(first==NULL)
        {
           printf("List is empty\n");
           return first;
        }
        cur=first;
        first=first->link;
        free(cur);
        return first;
}

STUD *delete_end(STUD *first)
{
        STUD *prev,*cur;
        if(first==NULL)
        {
           printf("List is empty\n");
           return first;
        }
        prev=NULL;
        cur=first;
        while(cur->link!=NULL)
        {
           prev=cur;
```

```
            cur=cur->link;
        }
        prev->link=NULL;
        free(cur);
        return first;
}

void display(STUD *first)
{
        STUD *temp;
        int count=0;
        if(first==NULL)
        {
            printf("List is empty\n");
            return;
        }
        printf("USN\tNAME\tBRANCH\tSEM\tPHONE NO.\n");
        temp=first;
        while (temp!=NULL)
        {
            printf("%s\t%s\t%s\t%d\t%d\n",temp->usn,temp->name,temp->branch,temp-
>sem,temp->phone_no);
            temp=temp->link;
            count++;
        }
        printf("The number of nodes in SLL=%d\n",count);
}


void main()
{
        int ch,i,n;
        STUD *first=NULL;
        clrscr();
        printf("Creation of SLL of N Students\n");
        printf("Enter the number of students\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
            first=insert_front(first);
        printf("SLL Created Successfully!!!\n");
        display(first);
        while(1)
        {
            printf("1.Display\t2.Insert End\t3:Delete End\nt.Insert Front\t5.Delete
Front\t6.Exit\n");
            printf("Enter the choice\n");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1: display(first);
```

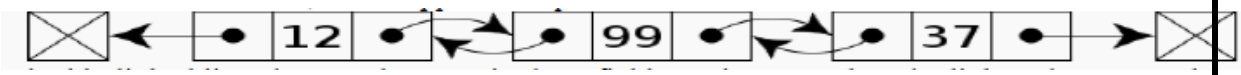
```
                        break;
                case 2: first=insert_end(first);
                        printf("Node Inserted at the End\n");
                        break;
                case 3: first=delete_end(first);
                        printf("Node deleted at the End\n");
                        break;
                case 4: first=insert_front(first);
                        printf("Node Inserted at Front\n");
                        break;
                case 5: first=delete_front(first);
                        printf("Node deleted at Front\n");
                        break;
                case 6: exit(0);
                        break;
                default:
                        printf("INVALID CHOICE !");
            }
        }
}
```

/* 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 */

Theory:-

- **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.

```
//Implementation
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct node
{
        char ssn[10],name[10],dept[15],desig[15];
        int phno;
        float sal;
        struct node *llink;
        struct node *rlink;
};
typedef struct node *NODE;
NODE temp, FIRST=NULL,END=NULL;
/*********************************/
NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        return x;
}
/*********************************/
void read()
{
        float sal; int phno;
        temp=getnode();
        temp->llink=NULL;
        temp->rlink=NULL;
        printf("Enter SSN");
        flushall();
        gets(temp->ssn);
        printf("Enter NAMe");
        flushall();
        gets(temp->name);
        printf("Enter dept:");
        flushall();
        gets(temp->dept);
        printf("Enter designation:");
        flushall();
        gets(temp->desig);
        printf("Enter phno");
        scanf("%d",&phno);
        temp->phno=phno;
        printf("Enter salary");
        scanf("%f",&sal);
        temp->sal=sal;
}
/*************Creation **********************/
void Create_DLL()
```

```
{
        int n,i=0;
        printf("enter the number of Employees \n");
        scanf("%d",&n);
        while(i!=n)
        {
                i++;
                printf("Enter the details of the %d employee\n", i);
                read();
                if(FIRST==NULL)
                {
                        FIRST=temp  ;
                        END=temp;
                }
                else{
                        END->rlink=temp;
                        temp->llink=END;
                        END=temp;
                }
        } //end of while
        printf("Creation of DLL for %d is done",i);
}
/**Display the status and count the number****/
void display_count()
{
        NODE temp1=FIRST;
        int count=1;
        printf("the employee details \n");
        if(temp1==NULL)
        {
                printf("the employee detail is NULL and count is %d\n",count-1);
        }
        else
        {
                printf("\nSSN\tName\tDept\tDesgn\tSal\tPhNo");
                while(temp1!=END)
                {
                        count++;
                        printf("\n%s\t%s\t%s\t%s\t%d\t%f",temp1->ssn,temp1->name,temp1-
>dept,temp1->desig,temp1->phno,temp1->sal);
                        temp1=temp1->rlink;
                 }
                  printf("\n%s\t%s\t%s\t%s\t%d\t%f",temp1->ssn,temp1->name,temp1-
>dept,temp1->desig,temp1->phno,temp1->sal);
                 printf("the student count is %d\n",count);
        }
        return;
}
  /**********************Insertion**************/
void  Insertionfront()
```

```
{
        printf("enetr the details of the employee\n");
        read();
        if(FIRST==NULL)
                FIRST=temp  ;
        else
        {
                temp->rlink=FIRST;
                FIRST->llink=temp;
                FIRST=temp;
        }
}
void Insertionend()
{
        temp=getnode();
        temp->llink=NULL;
        temp->rlink=NULL;
        printf("enter the deatils of the new employee\n");
        read();
        if(FIRST==NULL)
        {
                FIRST=temp;
                END=temp;
        }
        else
        {
                END->rlink=temp;
                temp->llink=END;
                END=temp;
        }
        return ;
}
/******************** Deletion********************/
void Deletionfront()
{
        NODE temp2 ;
        if(FIRST==NULL)
        {
                 printf("List is empty\n");
        }
        else if(FIRST==END)
        {
                temp2=FIRST;
                printf("Record with %s SSN is deleted\n", temp2->ssn);
                FIRST=NULL;
                END=NULL;
        }
        else {
                temp2=FIRST;
                printf("Record with %s SSN is deleted\n", temp2->ssn);
```

```
                FIRST =FIRST->rlink;
                temp2->llink=NULL;
                free(temp2);
        }
        return;
}
void Deletionend()
{
        NODE temp2 = END;
        if(temp2==NULL)
        {
                printf("List is empty\n");
        }
        else if(FIRST==END)
        {
                printf("Record with %s SSN is deleted\n", temp2->ssn);
                FIRST=NULL;
                END=NULL;
        }
        else
        {
                printf("Record with %s SSN is deleted\n", temp2->ssn);
                END=END->llink;
                END->rlink=NULL;
                free(temp2);
         }
        return ;
}
/**************Dqueue Demonstration**************/
void demonstration()
{
        int choice;
        clrscr();
        while(1)
        {
                 printf("\n1.Insertion at front...\t2.Insertion at end...\t...3.deletion at
front...\t4.deletion  at end....\t5.Display contents of dQueue...\t6.Exit...");
                 printf("\ntEnter Your Choice: ");
                 scanf("%d",&choice);
                 switch(choice)
                 {
                         case 1: Insertionfront(); break;
                         case 2: Insertionend(); break;
                         case 3:Deletionfront();break;
                         case 4:Deletionend();break;
                         case 5:display_count();break;
                         case 6:exit(0);
                          default: printf("\nEnter proper Choice....");
                 }
        }
```

```
}
void main()
{
        int choice;
        clrscr();
        while(1)
        {
                printf("\n\n\n\t1.create DLL...\t2.Display SLL..\t3.Insertion at
front...\t4.Insertion at end...\t...5.deletion at front...\t6.deletion  at end....\t7.Demonstration of
dQueue...\t8.Exit...");
                printf("\n\n\n\tEnter Your Choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: Create_DLL();          break;
                        case 2:display_count();        break;
                        case 3: Insertionfront();      break;
                        case 4: Insertionend();        break;
                        case 5:Deletionfront();        break;
                        case 6:Deletionend();          break;
                        case 7:demonstration();        break;
                        case 8:exit(0);
                        default: printf("\n\n\n\tEnter proper Choice....");
                }
        }
}
```

/* 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

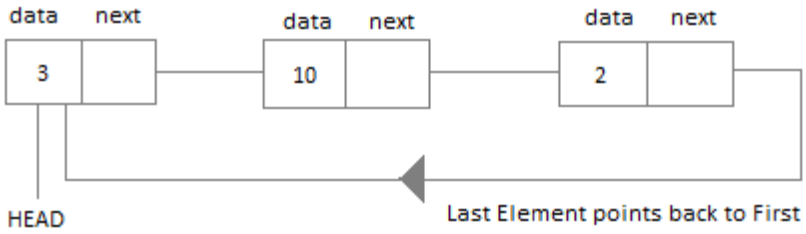
$P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$

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
***/**

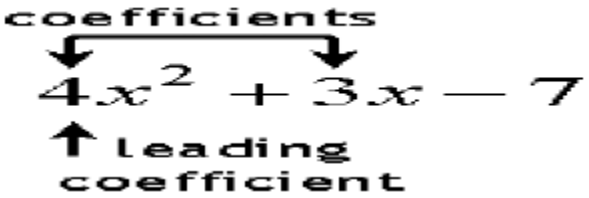
Theory:-

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. axn + bxn-1 + . . . + 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** x2 + 3y3for x = 7 and y = −2. Substitute the given values for x and y. **Evaluate**4x2y – 2xy2 + 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.

//Implementation

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
#include<math.h>
 struct node
{
        float cf;
        int px,py,pz;
        int flag;
        struct node *link;
};
typedef struct node *NODE;

NODE  getnode()
{
        NODE  x;
        x=(NODE) malloc (sizeof(struct node));
if(x==NULL)
{
        printf("out of memory\n");
        exit(0);
}
return x;
}

NODE insert_rear(float cf,float  x , float  y, float  z,NODE head)
{
        NODE temp,cur;
        temp=getnode();
        temp->cf=cf;
        temp->px=x;
        temp->py=y;
        temp->pz=z;
        temp->flag=0;
        cur=head->link;
        while(cur->link!=head)
        {
                cur=cur->link;}
        cur->link=temp;
        temp->link=head;
return head;
}


void display(NODE head)
```

```
{
        NODE  temp;
        if(head->link==head)
        {
                printf("polynominal doesn't exsits\n");
                return;
        }
temp=head->link;
while(temp!=head)
{
        printf("+ %  5.2fx^%dy^%dz^%d",temp->cf,temp->px,temp->py,temp->pz);
        temp=temp->link;
}
        printf("\n");
}
NODE  add_poly(NODE h1,NODE h2, NODE h3)
{
        NODE p1,p2;
        int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
        p1=h1->link;
        while(p1!=h1)
        {
                x1=p1->px;
                y1=p1->py;
                z1=p1->pz;
                cf1=p1->cf;
                p2=h2->link;
                while(p2!=h2)
                {
                         x2=p2->px;
                        y2=p2->py;
                        z2=p2->pz;
                        cf2=p2->cf;
                        if(x1==x2 &&  y1==y2 && z1==z2) break;
                                p2=p2->link;
                }
                if(p2!=h2)
                {
                        cf=cf1+cf2;
                        p2->flag=1;
                        if(cf!=0)
                                h3=insert_rear(cf,x1,y1,z1,h3);
                }
                else
                        h3=insert_rear(cf1,x1,y1,z1,h3);
                        p1=p1->link;
                }

p2=h2->link;
        while(p2!=h2)
```

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

NODE read_poly(NODE head)
{
        int i,n;
        int px,py,pz;
        float cf;
        printf("enter the number of terms\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("enter the %d term\n",i);
                printf("coeff=");
                scanf("%f",&cf);
                //if(cf==-999) break;
                printf("pow x=");
                scanf("%d",&px);
                printf("pow y=");
                scanf("%d",&py);
                printf("pow z=");
                scanf("%d",&pz);
                head=insert_rear(cf,px,py,pz,head);
        }
        return head;
}

void polysum()
{
        NODE h1,h2,h3;
        h1=getnode();
        h2=getnode();
        h3=getnode();
        h1->link=h1;
        h2->link=h2;
        h3->link=h3;
        printf("enter the first polynominal\n");
        h1=read_poly(h1);
        printf("enter the second polynominal\n");
        h2=read_poly(h2);
        h3=add_poly(h1,h2,h3);
        printf(" the first polynominal is\n");
        display(h1);
```

```
        printf("second polynominal is\n");
        display(h2);
        printf("the sum of two polynominal is\n");
        display(h3);
}
void represent_evaluate()
{
        NODE e1,temp;
        int x,y,z;
        float sum=0.0;
        e1=getnode();
        e1->link=e1;
        printf("enter the polynominal\n");
        e1=read_poly(e1);
        printf("polynominal i s \n");
        display(e1);
        printf("enter the values of coefficient\n");
        scanf("%d%d%d",&x,&y,&z);
         if(e1==NULL)
         {
                 printf("list is empty");
         }
         else
         {
                temp=e1->link;
                while(temp!=e1)
                {
                sum+=temp->cf*pow(x,temp->px)*pow(y,temp->py)*pow(z,temp->pz);
                temp=temp->link;
                }
         //     sum+=temp->cf*pow(x,temp->px)*pow(y,temp->py)*pow(z,temp->pz);
                printf("the total sum is %f\n",sum);
         }
return;
 }
void main()
{
         int choice;
        clrscr();
        while(1)
        {
                 printf("\n\n\n\t1.represent and evaluate...\t2.ADD TWO poly..\t3.Exit...");
                 printf("\n\n\n\tEnter Your Choice: ");
                 scanf("%d",&choice);
                 switch(choice)
                 {
                         case 1: represent_evaluate();break;
                         case 2:polysum();break;
                         case 3:exit(0);
                         default: printf("\n\n\n\tEnter proper Choice...."); }}}
```

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

Theory:-

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)

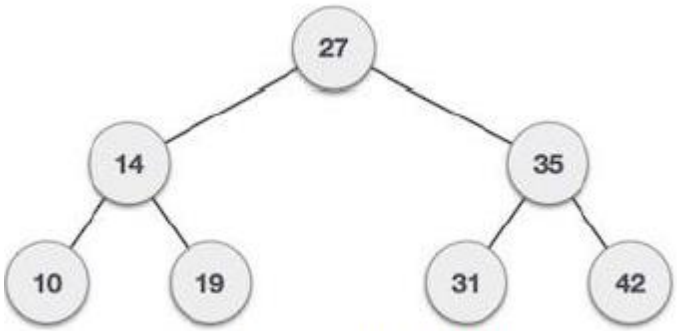


Fig: An example of BST

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.

//Implementation

```
#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);
        }
}

void main()
{
        int data, ch, i, n;
        NODE *root=NULL;
        clrscr();
        while (1)
        {
                printf("\n1.Insertion in Binary Search Tree");
                printf("\n2.Search Element in Binary Search Tree");
                printf("\n3.Delete Element in Binary Search Tree");
                printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1: printf("\nEnter N value: " );
                                scanf("%d", &n);
                printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                        for(i=0; i<n; i++)
                        {
                                scanf("%d", &data);
                                root=createtree(root, data);
                        }
                        break;
                        case 2: printf("\nEnter the element to search: ");
                                scanf("%d", &data);
                                root=search(root, data);
                        break;
                        case 3: printf("\nEnter the element to delete: ");
                                scanf("%d", &data);
                                root=del(root, data);
                        break;
                        case 4: printf("\nInorder Traversal: \n");
                                inorder(root);
```

```
                        break;
                        case 5: printf("\nPreorder Traversal: \n");
                                preorder(root);
                        break;
                        case 6: printf("\nPostorder Traversal: \n");
                                postorder(root);
                        break;
                        case 7: exit(0);
                                default:printf("\nWrong option");
                        break;
                }
        }
}
/* OUTPUT
```

```
5.Postorder
6.Exit
Enter your choice: 2

Enter the element to search: 24

Element found is: 24
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Inorder
4.Preorder
5.Postorder
6.Exit
Enter your choice: 3

Inorder Traversal:
2       2       5       5       6       7       8       8       9       14
15      24
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Inorder
4.Preorder
5.Postorder
6.Exit
Enter your choice: _
```

/*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 */

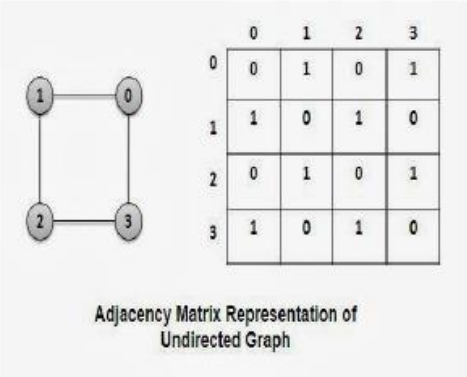
Theroy:-

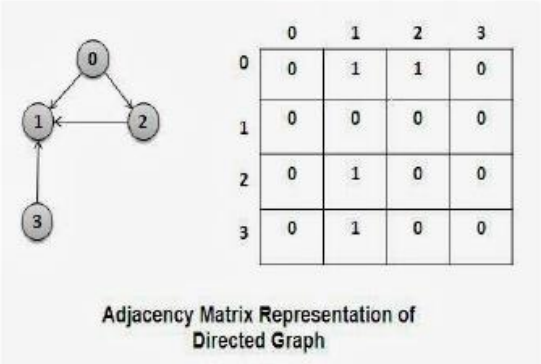
Adjacency Matrix

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.

//Implementation

```
#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,flag=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;
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        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]);
        for(;;)
        {
                printf("\n 1.BFS\n 2.DFS\n 3.Exit\n");
                printf("Enter your choice   :  ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("\n Enter the starting vertex:");
                                scanf("%d",&v);
        for(i=1;i<=n;i++)
        {
                q[i]=0;
```

```
                visited[i]=0;
        }

                        bfs(v);
                        if((v<1)||(v>n))
                        {
                                printf("\n Bfs is not possible");
                        }
                        else
                        {
                                printf("\n The nodes which are reachable from %d: ",v);
                                for(i=1;i<=n;i++)
                                if(visited[i])
                                printf("%d\t",i);
                        }
                        for(i=1;i<=n;i++)
                        {
                                if(visited[i])
                                {
                                        flag=1;
                                }
                        }
                        if(flag==0)
                                printf("\nFrom %d no vertex can be reached\n",v);
                break;
                case 2:
        for(i=1;i<=n-1;i++)
                reach[i]=0;

                dfs(1);
                        if(count==n-1)
                                printf("\n Graph is connected");
                        else
                                printf("\n Graph is not connected");
                break;
                case 3: exit(0);
                }
        }
}
/* Output
```

```
 Enter the number of vertices:5

 Enter graph data in matrix form:
0 0 1 1 0
0 1 0 1 1
0 0 0 0 1
0 0 1 1 0
0 0 0 0 1

 1.BFS
 2.DFS
 3.Exit
Enter your choice    :  1

 Enter the starting vertex:1

 The nodes which are reachable from 1: 3        4       5
 1.BFS
 2.DFS
 3.Exit
Enter your choice    :  _
```

```
 Enter the starting vertex:2

 The nodes which are reachable from 2: 2        3       4       5
 1.BFS
 2.DFS
 3.Exit
Enter your choice    :  1

 Enter the starting vertex:4

 The nodes which are reachable from 4: 2        3       4       5
 1.BFS
 2.DFS
 3.Exit
Enter your choice    :  2

 1->3
 3->5
 1->4
 Graph is not connected
 1.BFS
 2.DFS
 3.Exit
Enter your choice    :  3
```

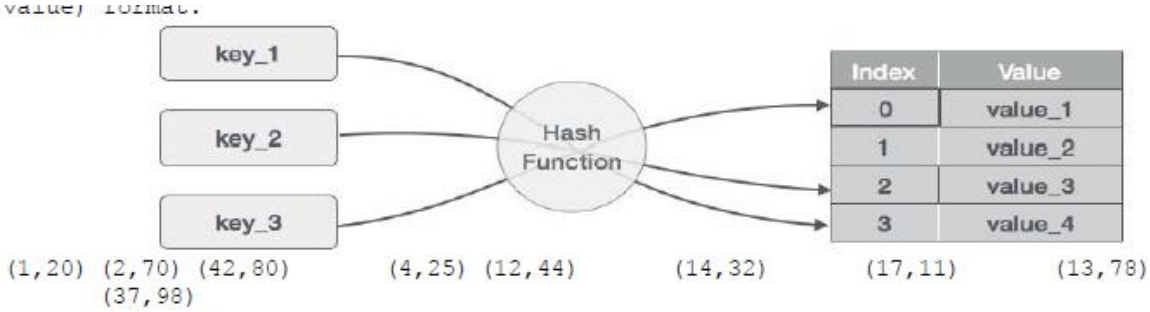
/* 12. Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses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

Theory:-

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

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

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



S.n.	Key	Hash	Array Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14

//Implementation

```
#include <stdio.h>
#define m 20
int HT[10];
int hash(int key)
{
   return key%m;
}
void linear_probe(int hk,int key)
{
   int i,flag=0;
   for (i=hk+1;i<m;i++)
   {
         if (HT[i]==999)
         {
            HT[i]=key;
            flag=1;
            break;
         }
   }

   for(i=0;i<hk&&flag==0;i++)
```

```
    {
        if (HT[i]==999)
        {
            HT[i]=key;
            flag=1;
            break;
        }
    }
    if (!flag)
        printf("HASH Table is Full!!!\n");
}
void main()
{
    FILE *fp;
    int N,i,key,hk;
    char name[100];
    for (i=0;i<m;i++)
        HT[i]=999;
    fp=fopen("emp.txt","r");
    while(!feof(fp))
    {
        fscanf(fp,"%d%s",&key,name);
        hk=hash(key);
        if (HT[hk]==999)
            HT[hk]=key;
      else
      {
            printf("Collision for key %d:\n",key);
            printf("Collision solved by Linear Probing\n\n");
            linear_probe(hk,key);
      }
    }
    printf("--------------------------------\n");
    printf("HASH TABLE\n");
    printf("--------------------------------\n");
    printf("Address\tKeys\n");
    for (i=0;i<m;i++)
        printf("%d\t%d\n",i,HT[i]);
}

/* Output
```

```
6       3012
7       999
8       999
9       999
Collision for key 2014:
Collision solved by Linear Probing

Collision for key 3012:
Collision solved by Linear Probing

------------------------------
HASH TABLE
------------------------------
Address Keys
0       999
1       999
2       1012
3       1013
4       1014
5       2014
6       3012
7       999
8       999
9       999
_
*/
```

VIVA QUESTIONS AND ANSWERS

1) What is data structure?

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

2) Differentiate file structure from storage structure.

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

3) When is a binary search best applied?

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

4) What is a linked list?

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

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

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

6) In what areas do data structures applied?

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

7) What is LIFO?

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

8) What is a queue?

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

9) What are binary trees?

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

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

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

11) What is a stack?

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

12) Explain Binary Search Tree

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

13) What are multidimensional arrays?

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

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

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

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

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

16) What is FIFO?

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

17) What is an ordered list?

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

18) What is merge sort?

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

19) Differentiate NULL and VOID.

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

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

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

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

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

22) What is a linear search?

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

23) How does variable declaration affect memory allocation?

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

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

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

25) What is a postfix expression?

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

26) What is a balanced tree?

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

27). Which data structure is needed to convert infix notations to post fix notations?

Stack

27). What is data structure or how would you define data structure?

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

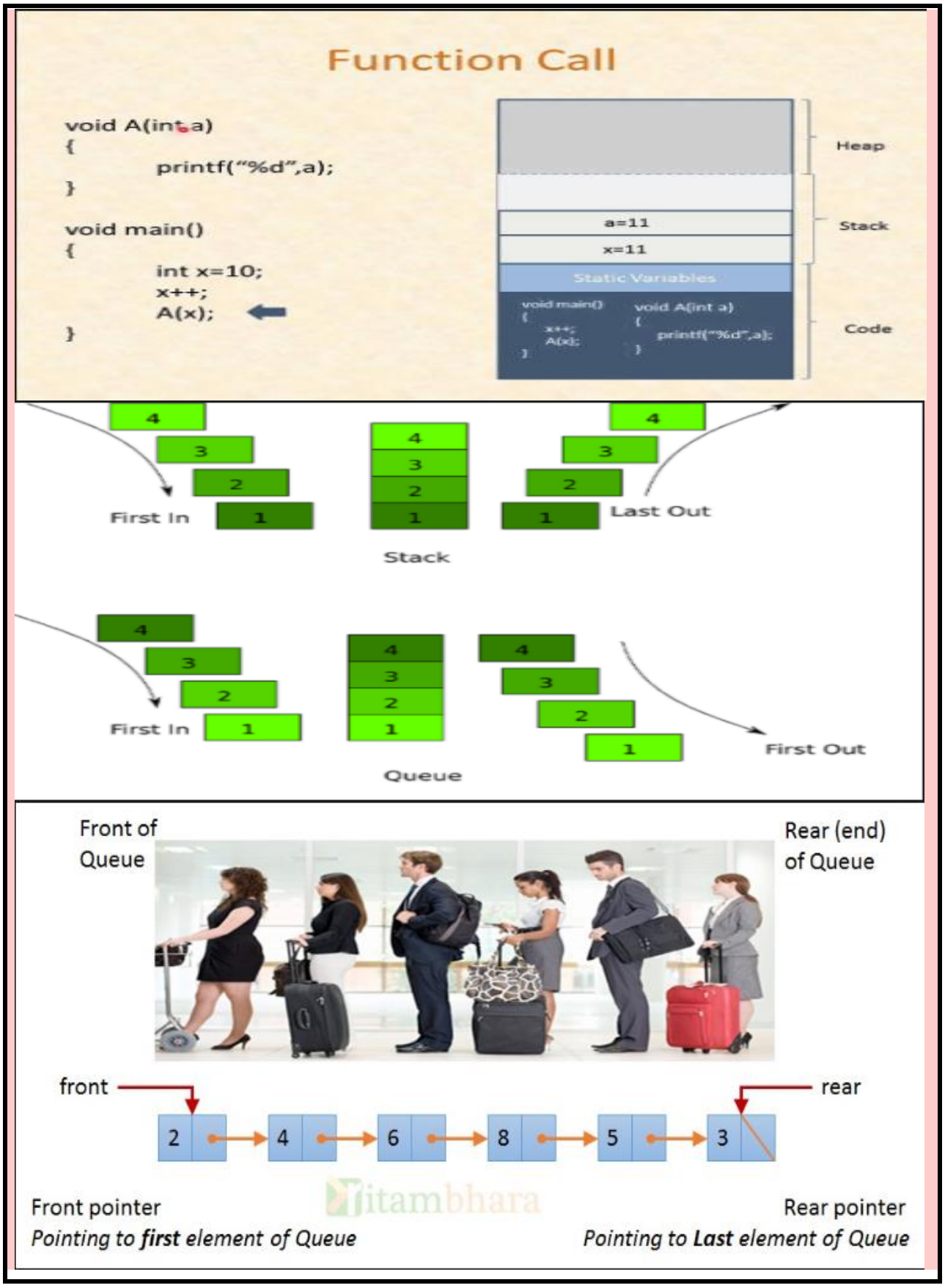
28. Which data structures we can implement using link list?

Queue and Stack

29. List different types of data structures?
Link list, queue, stack, trees, files, graphs

30) What is Hashing

Hashing is a technique to convert a range of key values into a range of indexes of an array.


Function Call
void A(int a)
{
printf("%d",a);
}
void main()
{
int x=10;
x++;
A(x);
}
Heap
Stack
a=11
x=11
Static Variables
Code
First In
Last Out
Stack
First In
First Out
Queue
Front of Queue
Rear (end) of Queue
front
rear
2
4
6
8
5
3
Front pointer
Pointing to first element of Queue
Rear pointer
Pointing to Last element of Queue

DATA STRUCTURES AND APPLCIATIONS

17CSL38

Success is not final, failure is not fatal: it is the courage to continue that counts. ...