INTRODUCTION TO DATA STRUCTURE

Q.1 What is Data structure [5M MAY 15]

Data Structures

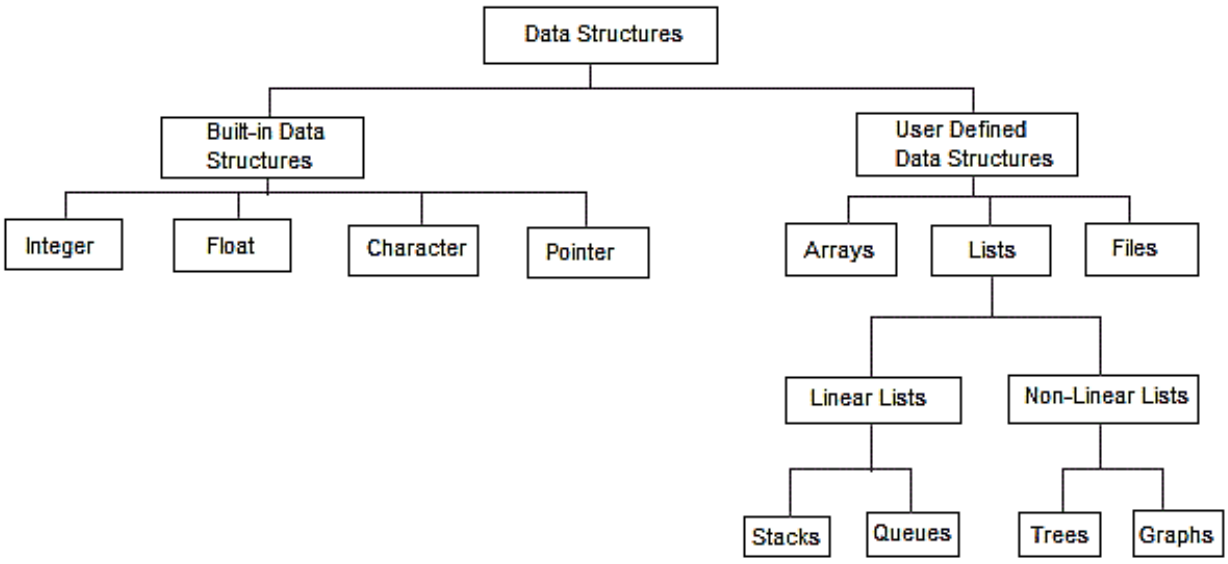
A *data structure* is a way of storing and organizing data in such way that we can perform operation on these data in an efficient way.

Areas in which data structures are applied extensively

- Compiler Design,
- Operating System,
- Database Management System,
- Statistical analysis package,
- Numerical Analysis,
- Graphics,
- Artificial Intelligence,

Q . 2 Explain different types of data structure with example [M-5 May-18]

Classification Data Structures



Types of Data structure [5M MAY17, DEC17]

Data structure is divided into two types

Primitive & Non-Primitive

1. **Primitive**: The integers, reals, logical data, character data, pointers and reference are primitive data structures. these data types are available in most programming language as build in type.
2. **Non-primitive**: These data structure is derived from primitive data structures.
   examples of non-primitive data structures: array, structure, union, linked list, stack, queue, tree, graph.

List data structure is divided into two types [5M DEC-16]

a. **Linear data structure:**
   Elements are arranged in a linear fashion, A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached, Linked list stores data in an organized linear fashion

   All one-one relation can be handled through linear data structure.

   e.g Stack , Queue

b. **Non-Linear data structure:**
   Element are arranged in nonlinear fashion, In this, the data elements can be attached to more than one element exhibiting the hierarchical relationship, all one-many, many-one or many-many relations are handled in Non-linear data structure.

   e.g Trees, Graphs

Q.3 What are various operations on data structure. [M-3 DEC-18]

Operations on Data Structures

- Traversing :
  Traversing a data structure is accessing each data and accessing only once.

- Searching :
  Searching is finding the location of a data in within the given data structure.

- Inserting :
  Inserting is adding a new data in the data structure.

- Deleting :
  Deleting is removing a data from the data structure.

- Sorting :
  Sorting is arranging of data in some logical order.

- Merging :
  Merging is combining of two similar data structures.

Different Approaches for Designing an Algorithm

Design of an algorithm is an important issue. There are several approaches for designing an algorithm. These approaches include:

1. Top-down approach.
2. Bottom-up approach.

Top-Down Approach

One has to write a program to solve a specific task.

A programmer tries to partition the solution into subtasks. Each subtask is similarly decomposed until all task are expressed within a programming language.

Top-down method of design is based on decomposing a large problem into sub problems and keep repeating this process till the resulting sub-problems are so simple that their solutions can be expressed in a few lines of a programming language.

Bottom-Up Approach

This approach is reverse of top-down approach. Here, the different parts of the problem are first solved using a programming language and then these pieces of programs are combined into a complete program.

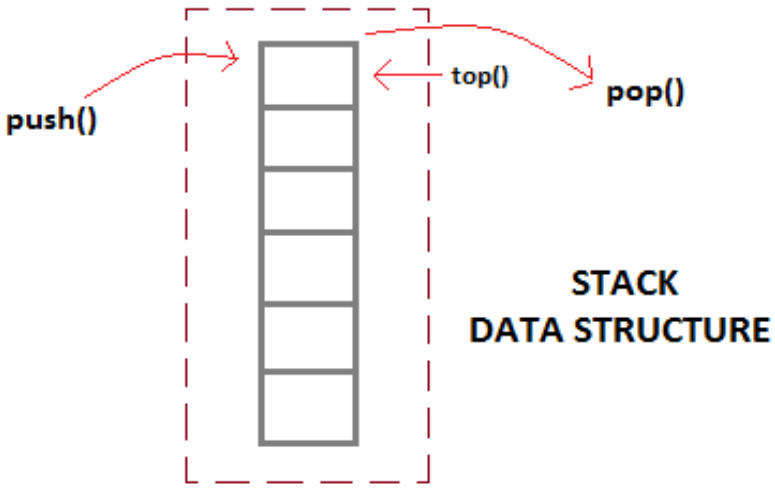
Define ADT (Abstract data type)

- Stack is an order list with insertion and deletion can be performed at only one position called top.
- A big program is never written as monolithic piece of program, instead it is broken down in smaller modules and each module is developed independently.
- Main program uses the services of next level function without knowing their implementation details. thus level of abstraction is created.
- Abstraction for primitive data type is provided by compiler.
- eg: we use integer data type and also perform various operation on them without knowing representation and how operations are performed on them.
- eg: z=x+y; meaning of operation '+' is defined by complier and its implementation details remain hidden from user.

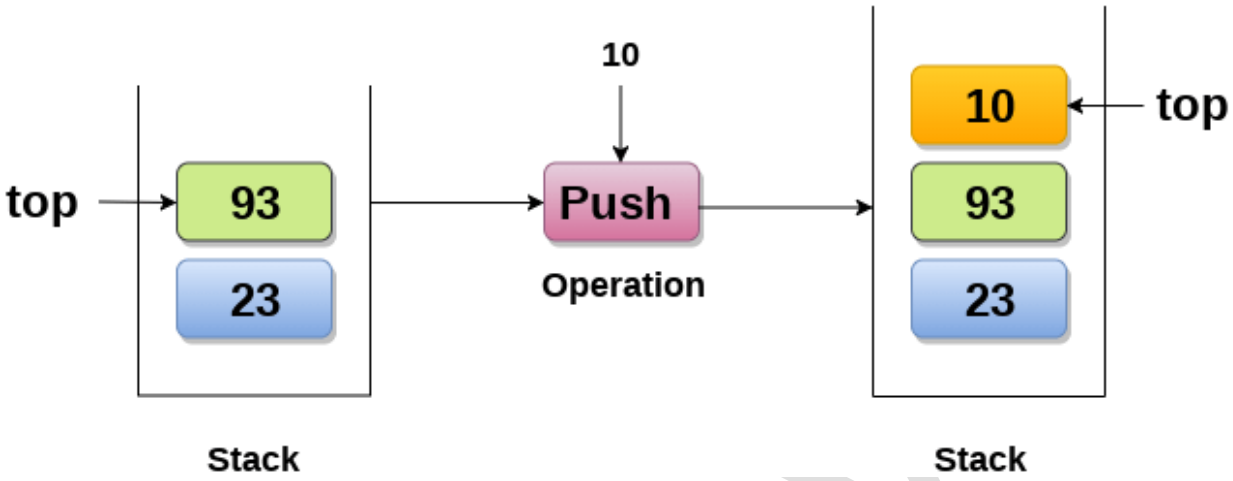
STACKS

What is stack

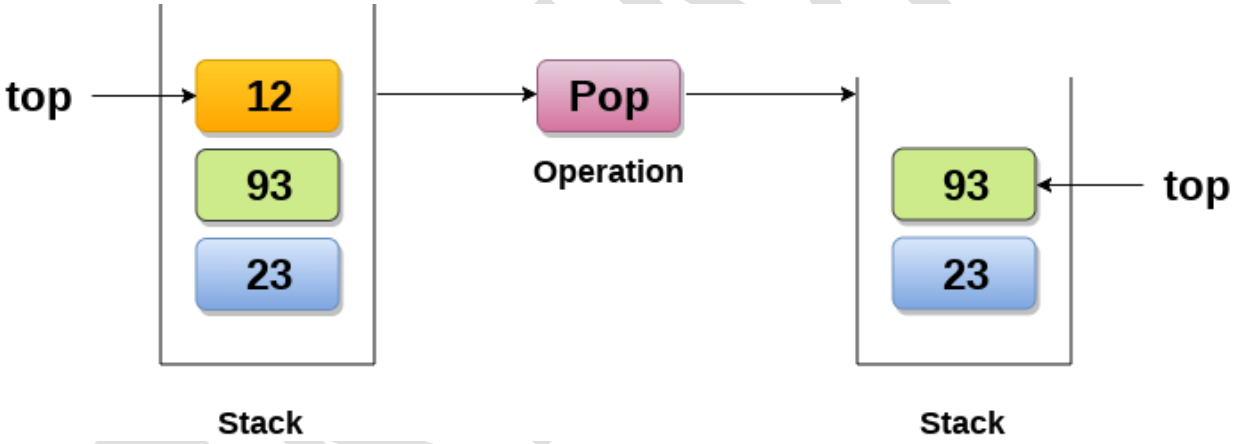
- Stack is a LIFO(Last In First Out) structure.
- Stack is a linear list where all insertion and deletion are permitted only at one end of list.
- Every time an element is added, it goes on the top of the stack, the only element that can be removed is the element that was at the top of the stack.



Inserting Data into Stack :-



Removing Data from Stack :-



STACK as an ADT

Stack can be represented using an array.

A one-dimensional array can be used to hold elements of stack.

Another variable “top” is used to keep track of the index of the top most element.

```
typedef struct stack
{
        int data[SIZE];
        int top;
}stack;
```

- **void initialize(stack *p):**
  It initializes a stack as an empty stack . Initial value of top is set to -1.

- **int empty(stack *p):**
  Functions checks whether the stack is empty, It returns 1 or 0 depending on whether the stack is empty or not.

- **int full (stack *p):**
  This function checks whether the stack is full. The function returns 1 or 0 depending on whether the stack is full or not.

- **int pop (stack *p):**
  The function deletes topmost element from the stack and also returns it to the calling program.

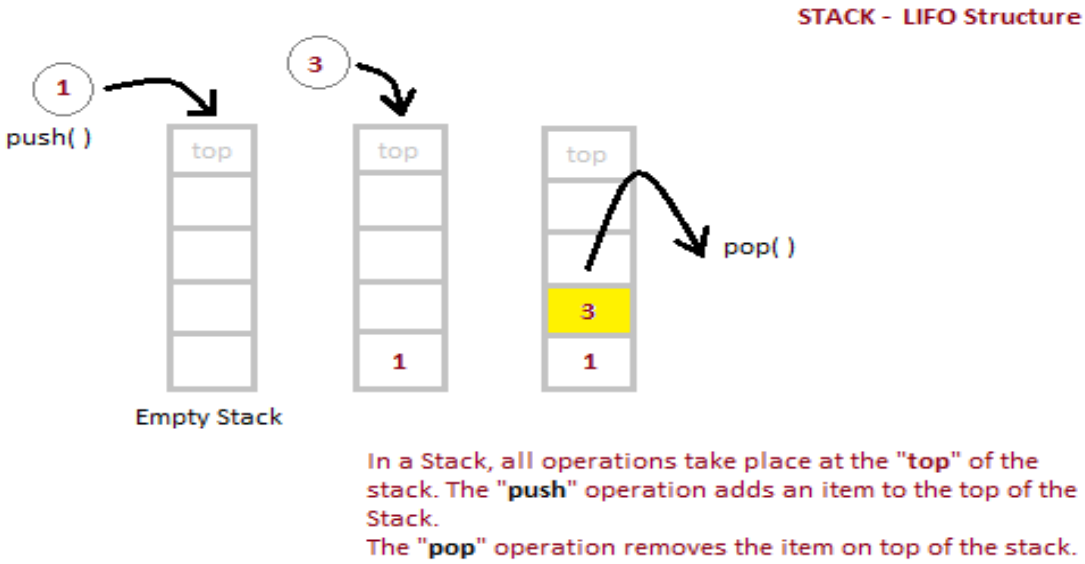
- **void push (stack *p , Int x):**
  The function inserts the element x onto the stack pointed by P. Insertion will cause an overflow if the stack is full.

Applications of Stack [5M DEC16]

- Expression Conversion
  a) Infix to postfix
  b) Infix to prefix
  c) postfix to infix
  d) prefix to infix
- Expression Evaluation
- Parsing
- Simulation of recursion
- Function Call

Implementation of Stack

- Stack can be easily implemented using an **Array** or a **Linked List**.
- Arrays are quick, but are limited in size and Linked List requires overhead to allocate, but is not limited in size.



One dimensional array can be used to hold element of a stack.

```
typedef struct stack
{
     int data[max];
     int top;
} stack;
```

a) **Initilize stack**

```
void initialize (stack*s)
{
  S→ top=  -1;
}
```

b) **Determine stack is empty**

```
int empty (stack*s)
{
  if (s→top = = -1)
  {
       return (1);
  }
  else
  {
```

```
        return (0);
    }
}
```

c) **Determine stack is full**

```
int full (stack*s)
{
  If (s→top = = max -1)
   {
     return (1);
   }
  else
  {
     return (0);
  }
}
```

d) **Insert element into stack**

```
void push (stack*s, int x)
{
  s→top = s→top + 1;
  s→data[s→top] = x;
}
```

e) **Remove element from stack**

```
int pop (stack*s)
{
  int x;
  x=s→data [s→top];
  s→top = s→top – 1;
  return(x);
}
```

WRITE A C PROGRAM FOR IMPLEMENTING STACK USING ARRAY.

```
#include<stdio.h>
#include<conio.h>
#define MAX 6
typedef struct stack
{
    int data[MAX];
    int top;
}stack;
void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *, int );
void print(stack *);
void main( )
{
    stack s;
    int x, op;
    init(&s);
    do
    {
        printf("\n\n1.Push\n2.Pop\n3.Print\n4.
        Quit");
        printf("\n Enter Your choice : ");
        scanf("%d", &op);
        switch(op)
        {
            case 1:print("\n enter a number:");
                   scanf("%d",&x);
                   if(!full(&s))
                     push(&s,x);
                   else
                     printf("\n Stack is full……");
                   break;
            case 2:
                   if(!empty (&s))
                   {
                     x=pop(&s);
                     printf("popped value = %d",x);
                   }
                   else
```

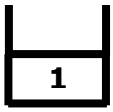
```
                        printf("\nStack is empty…….");
                        break;
                case 3:print(&s);
                         break;
            }
     }while( op != 4);
}
void init(stack *s)
{
         s->top = -1;
}
int empty(stack *s)
{
        if(s->top==-1)
           return(1);
        else
           return(0);
}
int full(stack *s)
{
     if(s->top==MAX-1)
         return(1);
     else
         return(0);
}
void push(stack *s,int x)
{
         s->top = s->top + 1;
         s->data[s->top] = x;
}
int pop(stack *s)
{
        int x;
        x=s->data[s->top];
        s->top=s->top-1;
        return(x);
 }
void print(stack *s)
{
     int i;
     for(i=s->top; i>=0; i--)
     {
       printf("%d",s->data[i]);
     }
```

}

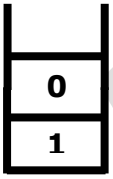
Convert Decimal to binary

$(21)_{10} = (\ ?\ )_2$

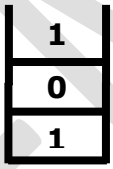
21 % 2 =1



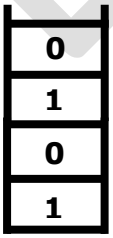
10 % 2 =0



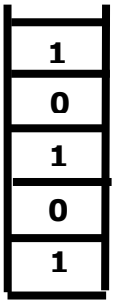
5 % 2 = 1



2 % 2 = 0



1 % 2 = 1



WRITE A PROGRAM IN C TO CONVERT DECIMAL TO BINARY USING STACK.

```
#include <stdio.h.>
#include<conio.h>
#include<ctype.h>
#define  MAX 50
typedef struct stack
{
  int data [MAX];
  int top;
}stack;
void init(stack*);
int empty(stack*);
int full(stack*);
int pop(stack*);
void push(stack*,int);
void main()
{
  stack s;
  int x;
  init(&s);
  printf("Enter a decimal number\n");
  scanf("%d", &x);
  while((x!=0)
  {
      if(!full(&s))
      {
          push(&s, x%2);
          x=x/2;
      }
      else
      {
          printf("Stack overflow");
      }
  }
  while (!empty(&s))
  {
      x=pop(&s);
      printf("%d",x);
  }
  getch();
}
```

```
void init(stack *s)
{
  S→top= -1;
}
int empty (stack *s)
{
  if(s→top ==-1)
      return(1);
  else
      return(0);
}
int full(stack *s)
{
  if(s→top = = MAX-1)
      return(1);
  else
      return(0);
}
void push(stack *s, intx)
{
  s→top=s→top+1;
  s→data[s→top]=x;
}
int pop (stack  *s)
{
  int x;
  x= s→data[s→top];
  s→top= s→top-1;
  return(x);
}
```

Expression representation

An Expression is a collection of operators and operands that represents a specific value.

There are three methods for representation of an expression.

- Infix (x+y) Operator between operands
- Prefix +xy Operator before operands
- Postfix xy+ Operator after operands

Evaluation of a postfix expression using a stack

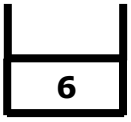
Given Expression

6 5 3 + 9 * +

Initially stack is empty.

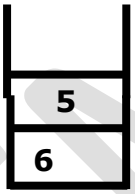
First token is an operand , push 6 on the push

5 3 + 9 * +



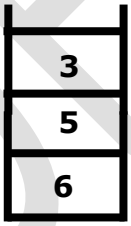
Next token is and operand ,push 5 on the stack

3+9*+



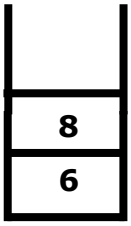
Next token is operand , push 3 on the stack .

+ 9 * +



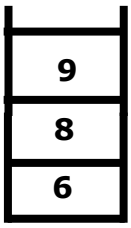
Next token is an operator "+" , pop two operands 3 and 5 ,
add them and push the result on the stack.

9 * +



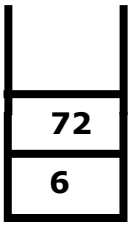
Next token is an operand , push 9 on the stack.

*** +**

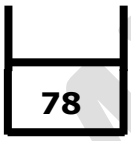


Next token is an operator " * ", pop two operands 9 and 8,multiply them and push the result on the stack.

+



Next token is an operator "+" , pop two operands 72 and 6 , add them and push the result back on the stack.



Value of the expression = 78

WRITE A PROGRAM TO EVALUATION A POSTFIX EXPRESSION [10M MAY 18]

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#define MAX 50
typedef struct stack
{
	int data [MAX];
	int top;
}stack;
void init(stack *s);
int empty(stack *s);
int full(stack *s);
int pop(stack *s);
void push(stack *s,int x);
int evaluate(int op1,int op2,char x);
void main()
{

	char x;
	int op1,op2,val;
	stack s;
	init(&s);
	printf("Enter a postfix expression\n");
	while((x=getchar())!='\n')
	{
		if(isdigit(x))
		{
			push(&s,x-48);
		}
		else
		{
```

```
                        op1=pop(&s);
                        op2=pop(&s);
                        val=evaluate(op1,op2,x);
                        push(&s,val);
                }
        }
        val=pop(&s);
        printf("After evaluation=%d",val);
        getch();
}
int evaluate(int op1,int op2,char x)
{
        if(x=='+')
                return(op1+op2);
        if(x=='-')
                return(op1-op2);
        if(x=='*')
                return(op1*op2);
        if(x=='/')
                return(op1/op2);
        if(x=='%')
                return(op1%op2);
}


void init(stack *s)
{
        s->top= -1;
}
int empty (stack *s)
{
        if(s->top ==-1)
                return(1);
        else
                return(0);
}
int full(stack *s)
{
        if(s->top == MAX-1)
                return(1);
        else
                return(0);
}
void push(stack *s, int x)
{
        s->top=s->top+1;
        s->data[s->top]=x;
0}
int pop (stack *s)
{
        int x;
        x= s->data[s->top];
        s->top= s->top-1;
        return(x);
}
```

Convert the following expression Q into the postfix form. Q =(A+B)*C-D/E*(F/G).
Show stack representation.

Sr. no	Input	Stack	Output
1	(A+B)*C-D/E*(F/G)	Empty	-
2	A+B)*C-D/E*(F/G)	(	-
3	+B)*C-D/E*(F/G)	(	A
4	B)*C-D/E*(F/G)	(+	A
5	)*C-D/E*(F/G)	(+	AB
6	*C-D/E*(F/G)	Empty	AB+
7	C-D/E*(F/G)	*	AB+
8	-D/E*(F/G)	*	AB+C
9	D/E*(F/G)	-	AB+C*
10	/E*(F/G)	-	AB+C*D
11	E*(F/G)	-/	AB+C*D
12	*(F/G)	-/	AB+C*DE
13	(F/G)	-*	AB+C*DE/
14	F/G)	-*(	AB+C*DE/
15	/G)	-*(	AB+C*DE/F
16	G)	-*(/	AB+C*DE/F
17	)	-*(/	AB+C*DE/FG
18	END	-*	AB+C*DE/FG/
19	END	Empty	AB+C*DE/FG/*-

Given Expression =(A+B)*C-D/E*(F/G).
Postfix Expression = AB+C*DE/FG/*-

Convert the following expression to postfix form
(f-g)*((a+b)*(c-d))/e [5M May-17]

WRITE A PROGRAM TO CONVERTS AN EXPRESSION FROM INFIX TO POSTFIX USING STACK.[10M DEC16,DEC17]

```
#include <stdio.h.>
#include<conio.h>
#include<ctype.h>
#define  MAX 50
Typedef struct stack
{
  int data [MAX];
  int top ;
}stack;
int precedence(char);
void init(stack*);
int empty(stack*);
int full(stack*);
int pop(stack*);
void push(stack*,int);
int top(stack*); //value of the top element
void infix_to_postfix(char infix[],char postfix[]);
void main()
{
  char infix[30],postfix[30];
  clrscr();
  printf("\n enter an infix expression:");
  gets(infix);
  infix_to_postfix(infix,postfix);
  printf("\npostfix:%s",postfix);
  getch();
}
void infix_to_postfix(char infix[],char postfix[])
{
  stack s;
  char x;
  inti,j;  //i-index for infix[],j-index for postfix
  char token;
  init(&s);
  j=0;
```

```
  for(i=0;infix[i]!='\0';i++)
  {
       token =infix[i];
       If(isalnum(token))
         postfix[j++]=x;
       else if(token =='(')
         push(&s, '(');
       else if(token ==')')
       {
         While((x=pop(&s))!='(')
              Postfix[j++]=x;
       }
       else
       {
         While(precedence(token)<=precedence(top(&s))&
         &!empty(&s))
         {
              x=pop(&s);
              postfix[j++]= x;
         }
           push(&s, token);
       }
  }
  While(!empty(&s))
  {
       x=pop(&s);
       postfix[j++]=x;
  }
  postfix[j]='\0';
}
int precedence(char x)
{
  if(X =='(')
       return(0);
  if (X= ='+' || X== '_')
       return(1);
  if (X= ='*' || X = ='%')
       return(2);
```

```
  return(3);
}
void init(stack *s)
{
     S ->top=-1;
}
int empty (stack *s)
{
  If(s ->top = = MAX-1)
           return(1);
  return(0);
}
void push(stack *s, intx)
{
  s-> top=s->top+1;
  s->data[s->top]=x;
}
int pop (stack  *s)
{
  int x;
  X= s-> data[s->top];
  s->top= s-> top -1;
  return(x);
}
int top (stack *p)
{
     return(p->data[p ->top]);
}
```

C program to convert:

1.Postfix to Infix

2.Postfix to Prefix

3.Prefix to Infix

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
# define MAX 20
typedefstruct stack
{
  int data [MAX];
  int top ;
}stack;

char str[MAX],stack[MAX];
int top=-1;
void push(char c)
{
   stack[++top]=c;
}
char pop()
{
   return stack[top--];
}
void post_in()
{
  inti,n;
  char a,b,op;
  printf("Enter the postfix expression\n");
  gets(str);
  gets(str);
  n=strlen(str);
  for(i=0;i<MAX;i++)
      stack[i]=NULL;
  printf("Infix expression is:\t");
  printf("%c",str[0]);
```

```
    for(i=1;i<n;i++)
    {
        if(str[i]=='+'||str[i]=='-
        '||str[i]=='*'||str[i]=='/')
        {
           b=pop();
           //a=pop();
           op=str[i];
          printf("%c%c",op,b);
        }
        else
        {
           push(str[i]);
        }
     }
     printf("%c",str[top--]);
    }
void post_pre()
{
inti,n;
printf("Enter the postfix expression\n");
  gets(str);
  gets(str);
  n=strlen(str);
printf("Prefix expression is:\t");
  for(i=n-1;i>=0;i--)
printf("%c",str[i]);
}

void pre_in()
{
    intn,i;
    char a,b,op;
    printf("Enter the prefix expression\n");
    gets(str);
    gets(str);
    n=strlen(str);
    for(i=0;i<MAX;i++)
```

```
    stack[i]=NULL;
      printf("Infix expression is:\t");
      for(i=0;i<n;i++)
    {
if(str[i]=='+'||str[i]==''||str[i]=='*'||str[i]=='/')
         {
            push(str[i]);
         }
         else
         {
            op=pop();
            a=str[i];
              printf("%c%c",a,op);
         }
      }
  printf("%c",str[top--]);
  }
  void main()
  {
  int opt;
  clrscr();
     while(1)
     {
  printf("\n.......MENU.......");
  printf("\n\t1.Postfix to Infix\n\t2.Postfix to
  Prefix\n\t3.Prefix to Infix\n\t4.Exit\n");
  scanf("%d",&opt);
        switch(opt)
        {
           case 1:post_in();break;
           case 2:post_pre();break;
           case 3:pre_in();break;
           case 4:exit();
            default:printf("Invalid choice");break;
        }
     }
  }
```

Introduction to Recursive Functions

- A function that calls itself is called as a recursive function.
- A recursive function must definitely have a condition that exit from calling the function again.
- Hence there must be a condition that calls the function itself if that condition is true. If the condition could also be implemented vice versa i.e. if the condition is false then the function calls itself else if the condition true, it will exit from the loop calling itself again.

Factorial (n) = 1
= n* factorial(n-1)

Write a program to find factorial of a number using recursion

```
#include <stdio.h>
#include<conio.h>
long int fact(int);
int main()
{
  int n,ans;
  printf("Enter a number \n);
  scanf("%d",&n);
  ans=fact(n);
  printf("Factorial of  %d is  %d" ,n,ans);
  return 0;
}

long int fact(int x)
{
  long int f;
  if(x= =1)
  {
     return(x);
  }
  else
  {
        f=x*fact(x-1);
        return (f);
```

```
  }
}
```

PROGRAM: WRITE A PROGRAM TO FIND VALUE OF Y USING A RECURSIVE FUNCTION, WHERE $Y = X^N$

```
#include<stdio.h>
#include<conio.h>
int exponential (int , int);
void main()
{
    int n , x , y;
    printf("Enter the value of x and n \n");
    scanf("%d %d",&x , &n);
    y = exponential(x, n);
    printf("the value of x raise to  is %d",y);
    getch();
  }
  int exponential (int x , int n)
  {
     int ans;
     if(n == 1)
     {
        return x ;
     }
     else
     {
        ans= x * exponential(x, n-1);
        return(ans);
     }
  }
```

OUTPUT:

```
enter the values of x and n
2
6
the value of x raise to n is 64
```

WRITE A RECURSIVE FUNCTION TO COMPUTE SUM OF FIRST N NUMBER BEGINNING WITH 1.

```
#include<stdio.h>
#include<conio.h>
   int compute(int);
void main()
{
  int no ,sum;
  printf("Enter a number \n");
  scanf("%d",&no);
  sum= compute (no);
  printf("sum of number from 1 to n %d",sum);
  getch();
}
int compute (int n)
{
    int ans;
    if(n = = 1)
    {
       return 1 ;
    }
    else
    {
       ans= n + compute(n-1);
       return(ans);
    }
}
```

Output :-

```
Enter a number
10
Sum of 1 to n is 55.
```

WRITE A RECURSIVE FUNCTION TO COMPUTE FIBONACCI SERIES.

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

```
int fibo(int);
Void main()
{
    int n,i,c,f;
    printf("Enter a number of terms");
    scanf("%d", &n);
    for(c=1;c<=n;c++)
    {
        f=fibo(i);
        printf("%d",f);
        i++
    }
    getch();
}
int fibo(int n)
{
    if(n==0)
    {
        return 0;
    }
    else if(n==1)
    {
        return 1;
    }
    else
    {
        return (fibo(n-1) + fibo (n-2));
    }
}
```

WRITE A PROGRAM FOR SUM OF DIGITS USE RECURSIVE FUNCTION TO CALCULATE SUM OF DIGITS OF AN INTEGER

```
#include  <stdio.h>
#include  <conio.h>
int  sum(int  n)
{
   if(n==0)
           return  0 ;
   else
      return( n%10 + sum(n/10) ) ;
}

void main()
{
   int  n,ans ;
   printf("Enter the number: ") ;
   scanf("%d" , &n) ;
   ans= sum(n);
   printf("Sum of digits of %d is %d" , n ,ans) ;
   getch() ;
}
```

Output:

```
Enter the number: 246
Sum of digits of 246 is 12
```

Important Questions

- Write a program in ‘C’ to evaluate a postfix expression. [M-10 Dec-13]
- Explain infix, postfix and prefix expressions with example[M-6 May-15]
- Write a c program to convert infix expression into postfix expr [M-10 May-15,May-14].
- Write a c program to convert a polish notation to reverse polish notation.[M-10 Dec-14].
- Explain ADT. list linear and Non-linear data structure [M-5 DEC 17,DEC 16].
- Explain Stack as an ADT. Give applications of stack [M-5 DEC 16]
- What are various operations on data structure. [M-5 DEC-18]
- Using data structure to check well formatted of parentheses in an algebraic expression. Write c program for same.
- Explain different types of data structure with example [M-5 May-18, May-17]
- Write a program in ‘C’ to evaluate postfix expression using stack ADT [M-10 May-18]
- Explain ADT. List the linear and non linear data structure with example. [M-5 DEC-17]
- Write a program to convert infix expression into postfix expression. [M-10, DEC-17, DEC-16]