Q2: Algorithm to evaluate Prefix Expression using Stack.

Initialisation: Read a token from input stream
a) If it is an operand, push it into operand stack
b) If it is an operator, push a marker onto operand stack & push the actual operator onto operator stack.

4: A-(B|C+(D%E*F)|G)*H

A-(|(BC|)+(DE%F*)|G)*H
A-((BC|)+(DE%F*G|))*H
A-(BC|DE%F*G|+*H
A-(BC|DE%F*G|+H*

ABC|DE%F*G|+H*A-

Q3: Recursion

- Recursion is a name given for expressing anything it terms of itself.
- Recursive function is a function which calls itself until a particular condition is met.

Properties of Recursive Algorithms:

- Recursive algo should terminate at some point, otherwise recursion will never end.
- Hence recursive algo should have stopping condition to terminate along with recursive calls.

Advantages: · Clearer and simpler versions of algorithm can be created using recursion.
- Recursive defination of a problem can be easily translated into a recursive function.
- Lot of bookkeeping activities such as initialisation is avoided.

Disadvantages: · When a function is called, the function saves formal parameter, local variables and return address and hence consumes a lot of memory.
- Lot of time is spent in pushing and poping and hence consumes more time to compute result.

INFIX TO POSTFIX

Q:4

1. Print operands as they arrive.
2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.
3. If the incoming symbol is a left parenthesis, push it on the stack.
4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.
5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.
6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.
7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.
8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

Q 5: What is an algorithm? Explain how you analyze the performance of it?

Performance analysis of an algorithm depends upon two factors i.e. amount of memory used and amount of compute time consumed on any CPU. Formally they are notified as complexities in terms of:

- **Space Complexity.**
- **Time Complexity.**

Process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

Space Complexity of an algorithm is the amount of memory it needs to run to completion i.e. from start of execution to its termination. Space need by any algorithm is the sum of following components:

1. **Fixed Component**: This is independent of the characteristics of the inputs and outputs. This part includes: Instruction Space, Space of simple variables, fixed size component variables, and constants variables.
2. **Variable Component**: This consist of the space needed by component variables whose size is dependent on the particular problems instances(Inputs/Outputs) being solved, the space needed by referenced variables and the recursion stack space is one of the most prominent components. Also this included the data structure components like Linked list, heap, trees, graphs etc.

Therefore the total space requirement of any algorithm 'A' can be provided as

Space(A) = Fixed Components(A) + Variable Components(A)

Example: Space Complexity

```
Algorithm Sum(number,size)\\ procedure will produce sum of all numbers
Provided in 'number' list
{
        result=0.0;
        for count = 1 to size do
1,2,3,4,....size times                          \\will repeat from
                result= result + number[count];
        return result;
}
```

the space complexity can be written as **Space(Sum) = 3 + n;**

Time Complexity of an algorithm(basically when converted to program) is the amount of computer time it needs to run to completion. The time taken by a program is the sum of the compile time and the run/execution time . The compile time is independent of the instance(problem specific) characteristics. following factors effect the time complexity:

1. Characteristics of compiler used to compile the program.
2. Computer Machine on which the program is executed and physically clocked.
3. Multiuser execution system.
4. Number of program steps.

Therefore the again the time complexity consist of two components fixed(factor 1 only) and variable/instance(factor 2,3 & 4),

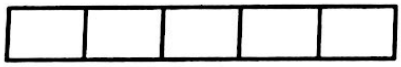
Time(A) = Fixed Time(A) + Instance Time(A)

Q 6: Discuss the problems associated with linear queue with suitable diagram.Give the algorithm for inserting an element into a circular queue?

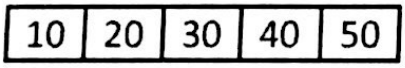
By the definition of a queue, when we add an element in Queue, rear pointer is increased by 1 whereas, when we remove an element front pointer is increased by 1. But in array implementation of queue this may cause problem as follows:

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

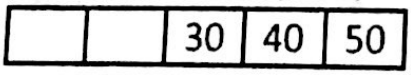
1. Initially empty Queue is there so,front = 0 and rear = -1



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



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



5. Now, actually we have deleted 2 elements from queue so, there should be space for another 2 elements in the queue, but as rear pointer is pointing at last position and Queue overflow condition
(Rear == SIZE-1) is true, we can't insert new element in the queue even if it has an empty space.
To overcome this problem there is another variation of queue called

[Inserts 'item' into the 'queue', 'F' is the Front end pointer and 'R' is the rare end pointer and 'N' is the number of elements in queue]

1. [Reset rare pointer]

```
if (R = N)
then R <-- 1
else
R <-- R + 1.
```

2. [Overflow ?]

```
if (F = R)
write Stack is full.
return.
```

3. [Insert element]

```
QUEUE [R] <-- item
```

4. [Setting front pointer]

```
if (F = 0)
then F <-- 1
return.
```

8. Templates are foundation of giveric programming which involves writing a code in a way that is independent of any particular type. A template is blue print for creating generaic class.
The presence of different class that are different in database of datamembers upon when member functions create huge difficulties in code maintaince.
Any change in one of the classes would have to be replicated to the other. Thus they come handy at this time.

```
#include <iostream>
using namespace std;
template <class T, int maxsize>
class st
{ int top;
  T a[maxsize];
  public: st() {top=-1}
          void push (int n)
        { if (top== maxsize-1)
        { cout <<"Overflow";
          return;
          }
          a [++top]=n;
          }
          T pop()
        { if (top==-1)
          { cout <<"Underflow";
            return -99;
          } return (a[top--]);
          } };
int main()
{ st <int, 4> stk;
  Stk.push(10);
  Stk.push(20);
  stk.push(30);
  Stk.push(40);
  Stk.push(50);
cout <<"Element popped"<<stk.pop();
cout <<"Element popped"<<stk.pop();
}
```

Templates is a facility provided by C++ language to write a common function that is independent of data type but which represents the common algorithm.
- Ease in code development
- Code maintainance.

iii) Now we have deleted 2 elements from queue there should be space for another 2 elements, as rear pointer is pointing at last position queue overflow condition is true, we cant any new element in the queue even if it ha empty space. To overcome this problem there is another variation of queue known as circ queue.

9 Big oh notation is a mathematical notation t that describes the limiting behaviour of function when argument tends towards a particular value or i infinity In computer science, big o notation is used t to classify algorithm by how they respond to ch change in input size such as how the processing ti time of an algorithm changes as the problem size extremely extremely large.

$f(n) = O(g(n))$ if there exists positive consta constants c and n_0 such that $f(n) \leq cg(n)$ for all n, $n \geq$

Proof:

By the big oh notation $f(n)$ is $O(n^3)$ if $f(n)$ if $f(n) \leq n_0$ for some $n \geq n_0$ let us check this condition

if $n^3 + 20n + 1 \leq cn^3$

then $1 + \frac{20}{n^2} + \frac{1}{n^3} \leq c$

then Big oh notation holds for $n \geq n_0 = 1$ and $c \geq 22$.

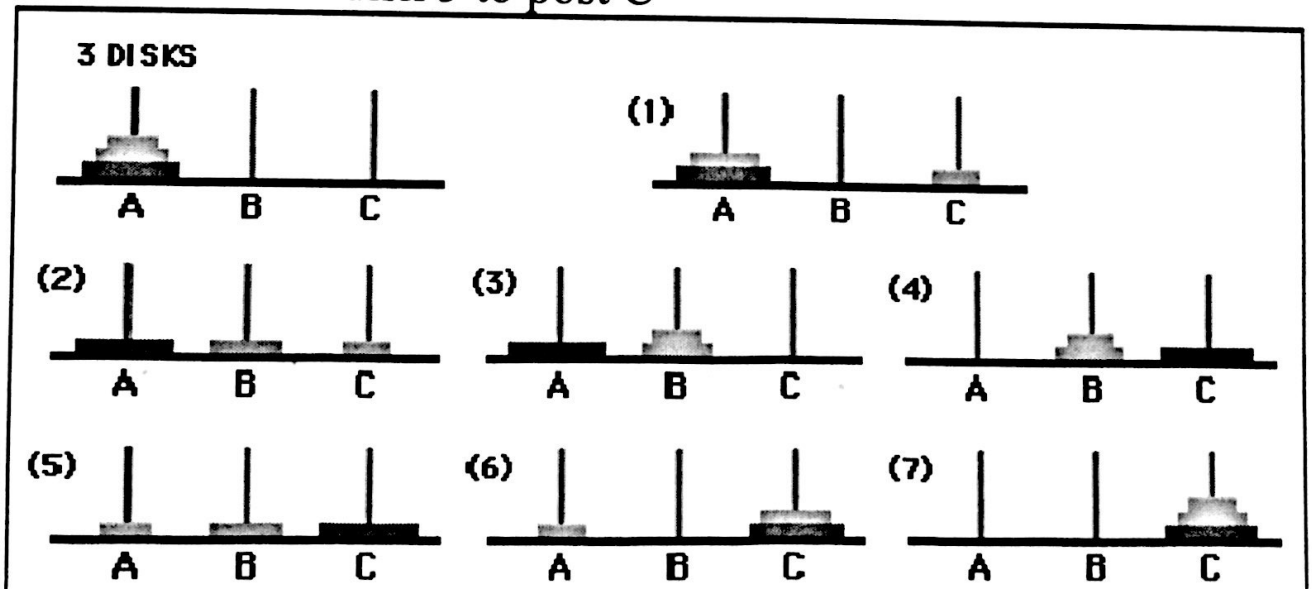
12) Performance of an algorithm can be measured in two ways

b. Remove the left parenthesis

Step 7. Exit

Q 10: Tower of henoi with three disks?

Move 1: move disk 3 to post C
Move 2: move disk 2 to post B
Move 3: move disk 3 to post B
Move 4: move disk 1 to post C
Move 5: move disk 3 to post A
Move 6: move disk 2 to post C
Move 7: move disk 3 to post C



Q:11

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.

A stack is a **recursive** data structure. Here is a structural definition of a Stack:

> a stack is either empty or
> it consistes of a top and the rest which is a stack;

Applications

- The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.
- Another application is an "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.

Backtracking. This is a process when you need to access the most recent data element in a series of elements. Think of a labyrinth or maze - how do you find a way from an entrance to an exit?

Once you reach a dead end, you must backtrack. But backtrack to where? to the previous choice point. Therefore, at each choice point you store on a stack all possible choices. Then backtracking simply means popping a next choice from the stack.

- Language processing:
  - space for parameters and local variables is created internally using a stack.
  - compiler's syntax check for matching braces is implemented by using stack.
  - support for recursion

Q12:-

```
int fact (int x)
{  int i, f = 1;          — 1
   for (i = 1; x! = 0; i++)   — x+1
      f = f * i;              — x
   return f;              — 1
}
```

$2x + 3$

Q:13: Application of Queue?

Typical uses of queues are in simulations and operating systems.

- Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
- Computer systems must often provide a "holding area" for messages between two processes, two programs, or even two systems. This holding area is usually called a "buffer" and is often implemented as a queue.

Our software queues have counterparts in real world queues.

Another important application of the queue data structure is to help us simulate and analyze such real world queues.

Step in each

14. Binary Search Time Complexity:

```
BST(mid, key)
{ if (mid == NULL || mid == key)
  return mid;
 if (key > a[mid])
 return (BST(left[mid], key));
  else
  return (BST(right[mid], key));
}
```

$T(n) = T(1) + T\left(\frac{n}{2}\right)$

$T(1) = 2$

$T\left(\frac{n}{2}\right) = 2 + T\left[\frac{n}{2^2}\right]$

$T(n) = 2+2+T\left[\frac{n}{4}\right]$

$T[n] = 4 + T\left[\frac{n}{4}\right]$

Instead of $n \to n/4 \Rightarrow T\left[\frac{n}{4}\right] = 2 + T\left[\frac{n/4}{2}\right]$

$T(n) = 4 + 2 + T\left[\frac{n}{8}\right]$

$T[n] = 6 + T\left[\frac{n}{8}\right]$

$T(n) = 2\times3 + T\left[\frac{n}{2^3}\right]$

$\therefore T(n) = 2\times4 \times T\left[\frac{n}{2^4}\right]$

$\therefore T(n) = 2k + T\left[\frac{n}{2k}\right]$

Taking $\frac{n}{2k} = 1$

$n = 2k$

$k \log 2 = \log n$

$\frac{\log n}{\log 2} = \log_2 n \Rightarrow \log(n/2)$

$T(n) = 2\log(n/2) + T\left(\frac{n}{2\log n/2}\right)$

$O(\log n)$

$\therefore$ Hence Time complexity is $O(\log n)$

}

16: There is a special type of ADT called the priority queue. This is a special type of table whose items are assigned priority values as they are added into the structure. This priority value is used to determine the order in which the items in the table are to be processed.

Methods:

create

This is the priority queue's class constructor. It will create a new empty priority queue.

isEmpty()

returns true if the priority queue is empty.

a new item to the priority queue.

e()

removes and returns the item in the priority queue with the highest priority.

Priority queue can be implemented in 4 ways

using arrays

The sort order will be from smallest to largest (ie the item with highest priority will always be the last item in the array)

limitations :

only a limited number of items may be added to the priority queue. If their limit is set to a large number, then this structure may waste a lot of memory

Although the delete() method is very efficient, the insert() method is very inefficient.

Using linked list

The sort order will be from largest to smallest (ie the item with highest priority is always the first item in the list).

limitations :

Although the delete() method is very efficient, the insert() method is inefficient.

Using a sorting algorithm to make a priority queue. A sorting algorithm can be used to implement a priority queue.

Limitations.

Although the insert() method is efficient, the delete() method is inefficient.

return.

Q:17

Algorithm of Infix to Prefix

```
1.          Step 1. Push ")" onto STACK, and add "(" to end of the A
2.          Step 2. Scan A from right to left and repeat step 3 to 6 for
    each element of A until the STACK is empty
3.          Step 3. If an operand is encountered add it to B
4.          Step 4. If a right parenthesis is encountered push it onto
    STACK
5.          Step 5. If an operator is encountered then:
6.                  a. Repeatedly pop from STACK and add to B each
    operator (on the top of STACK) which has same
7.                  or higher precedence than the operator.
8.              b. Add operator to STACK
9.          Step 6. If left parenthesis is encontered then
10.                 a. Repeatedly pop from the STACK and add to B (each
    operator on top of stack until a left parenthesis is encounterd)
```

11. b. Remove the left parenthesis
12. Step 7. Exit

else 'ch' operand) 'ch' of
in
oon

(5)
Symb
12

```
#include <stdlib.h>
using namespace std;
class twostack
{ int arr;
  int size;
  int top1, top2;
public:
   two stack(int n)
 {
   size = n;
   arr = new int [n];
   top1 = -1;
   top2 = size;
 }
   void push (int x)
  { if (top1 < top2-1)
   { top1 ++;
     arr [top1] = x;
   } else
   { cout << " Stack Overflow" << endl;
     exit (1);
   } }
  void push 2 (int x)
  { if (top1 < top 2-1)
   { top 2--;
     arr [top 2] = x;
   } else
   { Cout << " Stack Overflow \n";
     exit (0);
   } }
  int pop 1 ()
  { if (top1>=0)
   { int x = arr [top1];
     top --;
     return x;
else
{
 cout << "Stack Underflow" << endl;
 exit(1);
} }
 int Pop2 ()
 { if (top2 < size)
  { int x = arr [top2];
    top 2++;
    return x;
  }
  else
  { cout << "Stack Underflow" << endl;
    exit (1);
  } } };
int main ()
{
   twostacks ts (5);
   ts. push1 (5);
   ts. push 2 (10);
   ts. push 2 (15);
   ts. push1 (11);
   ts. push (7);
 Cout << "Propped element from stack1 is"
      << ts. pop1 () << endl;
 ts. push (40);
 Cout << "Popped element from stack2 is"
      << ts. pop2 () << endl;
   return 0;
}
```