CMR
INSTITUTE OF
TECHNOLOGY

CMR

Internal Assessment Test 1 – April 2019

Sub:	**Data Structures using C++**							**Code:**	18MCA22
Date:	16-04-19	Duration :	90 mins	Max Marks:	50	**Sem:**	II	**Branch:**	**MCA**

Dr. Vakula Rani **Answer any five of the following** **5 x 10 = 50 Marks**

Q1 What is Data Structure? Explain the classification of Data Structures with an example. Give any four applications of Data Structures.

Sol :

The study of data structures deals with the study of how the data is organized in the memory, how efficiently the data can be retrieved from the memory, how the data is manipulated in the memory and the possible ways in which different data items are logically related.

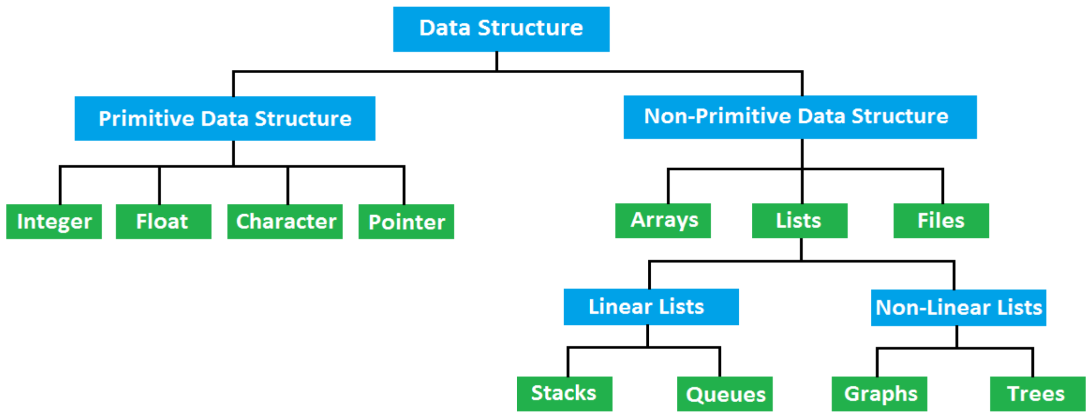
Def 1: The logical or mathematical model of a particular organization of data is known as Data Structure.

Def 2 : A Data structure 'd' is a triplet (D,F,A) where 'D' is a set of Data objects, 'F' is a set of functions and 'A' is a set of rules to implement the functions.

Classification of data structure :

The data structures are mainly classified into two categories :

- Primitive data structure
- Non-primitive data structure



i)**Primitive data structure :**

The primitive data structures are known as basic data structures. These data structures are directly operated upon by the machine instructions. Normally, primitive data structures have different representation on different computers.

Example of primitive data structure : Integer ,Float ,Character , Pointer

Integer :

The integers are signed or unsigned whole numbers with the specified range such as 5, 39, -1917, 0 etc. They have no fractional parts. Integers can be positive or negative but whether or not they can have negative values, it depends upon the integer types.

Float :

Float refers floating point or real number. It can hold a real number or a number having a fractional part like 3.112 or 588.001 etc. The decimal point signals that it is a floating point number, not an integer. The number 15 is an integer but 15.0 is a floating point number.

Character :

It can store any member of the basic character set. If a character from this set is stored in a character variable, its value is equivalent to the integer code of that character basically known as ASCII code. It can hold one letter/symbol like a, B, d etc. Characters can also be of different types.

Pointer :

A pointer is but a variable-like name points or represents a storage location in memory (RAM). RAM contains many cells to store values. Each cell in memory is 1 byte and has a unique address to identify it. The memory address is always an unsigned integer.

ii) Non-Primitive data structure :

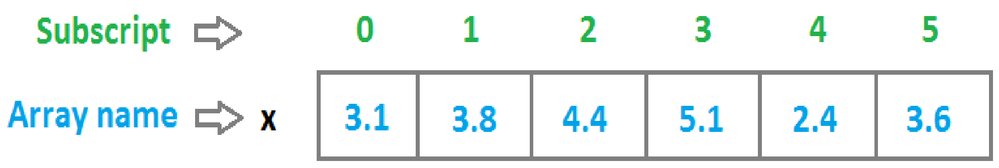
The non-primitive data structures are highly developed complex data structures. Basically, these are developed from the primitive data structure. The non-primitive data structure is responsible for organizing the group of homogeneous and heterogeneous data elements.

Non-primitive data structure :

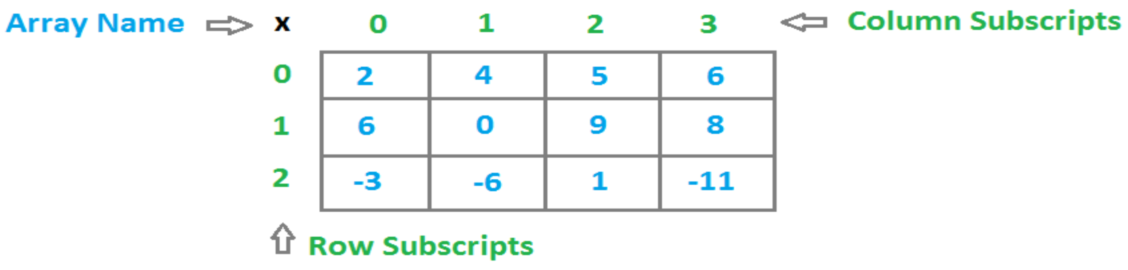
- Arrays
- Lists
- Files

Arrays :

Arrays are the set of homogeneous data elements stored in RAM. So, they can hold only one type of data. The data may be all integers, all floating numbers or all characters. Values in an array are identified using array name with subscripts. Single sub-scripted variables are known as a one-dimensional array or linear array; two sub-scripted variables are referred as a two-dimensional array.



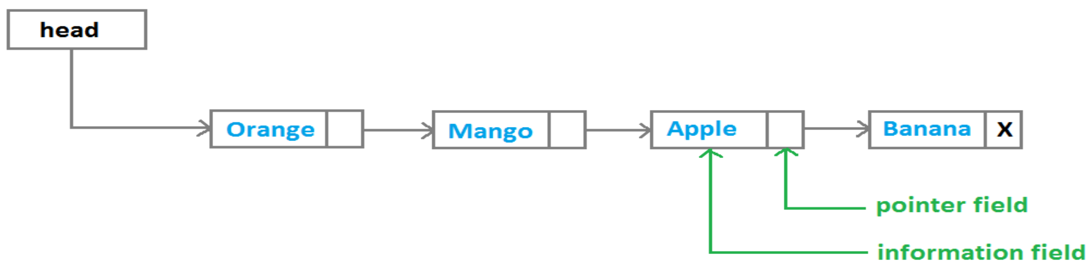
One Dimensional Array



Two Dimensional Array

Lists :

A list is a collection of a variable number of data items. Lists fall in the non-primitive type of data structure in the classification of data structure. Every element on a list contains at least two fields, one is used to store data and the other one is used for storing the address of next element.



Linear linked lists

Files

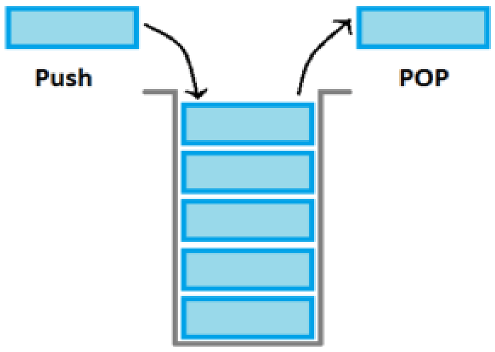
Stack :

Like arrays, a stack is also defined as an ordered collection of elements. A stack is a non-primitive linear data structure having a special feature that we can delete and insert elements from only one end, referred as TOP of the stack. The stack is also known as Last In First Out (LIFO) type of data structure for this behaviour.

When we perform insertion or deletion operation on a stack, its base remains unchanged but the top of the stack changes. Insertion in a stack is called Push and deletion of elements from the stack is known as Pop.

We can implement a stack using 2 ways:

- Static implementation (using arrays)
- Dynamic implementation (using pointers)



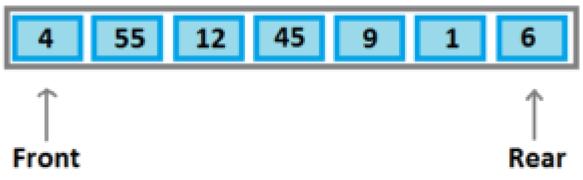
Stack

Queues :

Queues are also non-primitive linear data structure. But unlike stacks, queues are the First In First Out (FIFO) type of data structures. We can insert an element in a queue from the REAR end but we have to remove an element from the only FRONT end.

We can also implement queues using 2 ways :

- Using arrays
- Using pointers



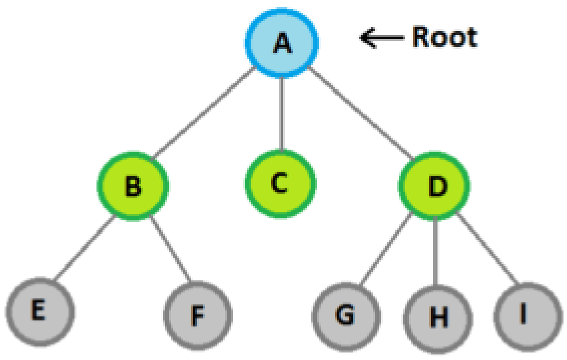
Queue

Trees :

Trees fall into the category of non-primitive non-linear data structures in the classification of data structure. They contain a finite set of data items referred as nodes. We can represent a hierarchical relationship between the data elements using trees.

A Tree has the following characteristics :

- The top item in a hierarchy of a tree is referred as the root of the tree.
- The remaining data elements are partitioned into a number of mutually exclusive subsets and they itself a tree and are known as the subtree.
- Unlike natural trees, trees in the data structure always grow in length towards the bottom.

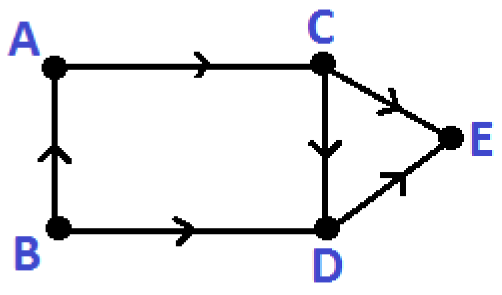


Trees

Graph :

Graph falls in the non-primitive non-linear type of data structure in the classification of data structure. Graphs are capable of representing different types of physical structures. Apart from computer science, they are used broadly in the fields of Geography, Chemistry & Engineering Sciences.

A graph normally a combination of the set of vertices V and set of edges E.



Graph

The different types of Graphs are :

- Directed Graph
- Non-directed Graph
- Connected Graph
- Non-connected Graph
- Simple Graph
- Multi-Graph

Applications of Data Structures

Data structures are applied extensively in :

- Compiler Design,
- Operating System,
- Database Management System,
- Statistical analysis packages,
- Numerical Analysis,
- Graphics,
- Simulation

(2) What is abstract data type give an example? Describes Stack ADT.

Sol :

The term abstract data type refers to the basic mathematical concept that defines the data type. ADT will specify the logical properties of a data type. It is a useful tool for implementers and programmers who wish to use the data type correctly.

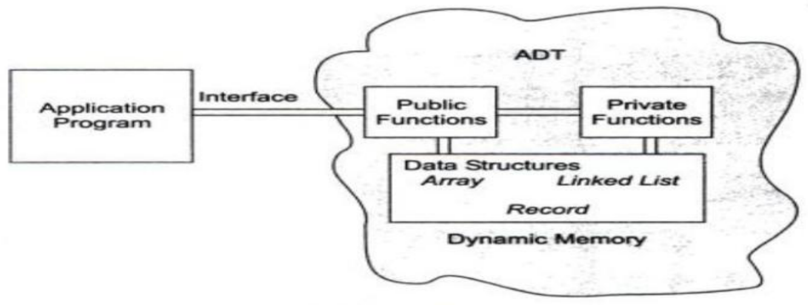


Fig. 1.13 ADT Model

Stack as ADT

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO. The stack operations are given below.

1) **Stack()** creates a new stack that is empty. It needs no parameters and returns an empty stack.
2) **push(item)** adds a new item to the top of the stack. It needs the item and returns nothing.
3) **pop()** removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
4) **peek()** returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
5) **isEmpty()** tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
6) **size()** returns the number of items on the stack. It needs no parameters and returns an integer.

For example, if s is a stack that has been created and starts out empty, then Table 1 shows the results of a sequence of stack operations. Under stack contents, the top item is listed at the far right.

Stack Operation	**Stack Contents**	**Return Value**
s.isEmpty()	[]	True
s.push(4)	[4]	
s.push(5)	[4,5]	
s.peek()	[4,5]	5
s.push(6)	[4,5,6]	
s.size()	[4,5,6]	3
s.isEmpty()	[4,5,6]	False
s.push(8)	[4,5,6,8]	
s.pop()	[4,5,6]	8
s.pop()	[4,5]	6
s.size()	[4,5]	2

Q3. What is an array? Write an algorithm to Insert and Delete an element from an array.

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Insert operation - Inserts one or more data elements into an array. New element can be added at the beginning, end, or any given index of array. Let **A** be a Linear Array (unordered) with **N** elements and Item is inserted into the K^{th} position of A

Algorithm for Insert operation

```
Step-1 :  Set J = N
Step-2 :  Set N = N+1
Step-3 :  Repeat steps 4 and 5 while J >= K
Step-4 :  Set A[J+1] = A[J]
Step-5 :  Set J = J-1
Step6 :  Set A[K] = Item
Step-7 : Stop
```

Delete operation - Deletion refers to removing an existing element from the array and re-organizing all elements of an array. Algorithm to delete an element available at the Kth position of A.

Algorithm for Delete operation

```
Step-1 : Set J = K
```

```
Step-2 : Repeat steps 3 and 4 while J < N
Step-3 : Set A[J] = A[J + 1]
Step-4 : Set J = J+1
Step-5 : Set N = N-1
Step-6 : Stop
```

```
void insert(int pos, int  item)
{
 for (i = n - 1; i >= pos - 1; i--) // Moving the elements Backward
 a[i+1] = a[i];
  a[pos-1] =  item;     // Inserting into the Location
  n=n+1;
}

int del(int pos)
{
  int temp;
   if (pos >= n+1)
        cout<<"Deletion not possible"<<endl;  // Checking for pos beyond the limit
 else                  {
              item=a[pos-1];
              for (i = pos - 1; i < n - 1; i++)   // Moving he elements forward
            a[i] = a[i+1];
       }
       n=n-1;
       return(item);
  }
```

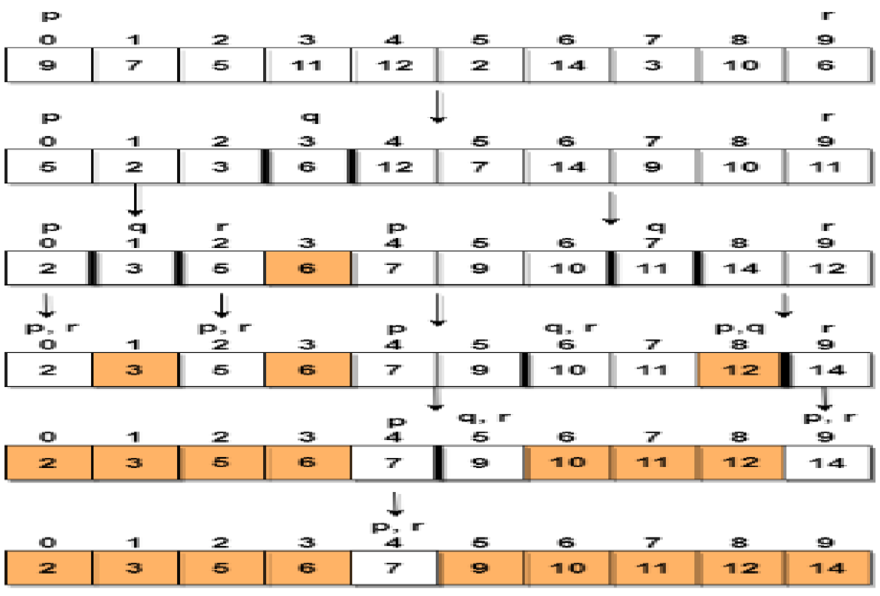
Q4. Describe Quicksort algorithm and sort the following set of items showing the items at each stage:
9,7,5,11,12,2,14,3,10,6

Sol :

The quicksort scheme developed by C. A. R. Hoare has the best average behavior among all the sorting methods . In Insertion Sort the key K_i currently controlling the insertion is placed into the right spot with respect to the sorted subfile ($R_1, ..., R_{i-1}$). Quicksort differs from insertion sort in that the key K_i controlling the process is placed at the right spot with respect to the whole file. Thus, if key K_i is placed in position $s(i)$, then K_j $K_{s(i)}$ for $j < s(i)$ and K_j $K_{s(i)}$ for j > $s(i)$. Hence after this positioning has been made, the original file is partitioned into two subfiles one consisting of records $R_1, ..., R_{s(i)-1}$ and the other of records $R_{s(i)+1}, ..., R_n$. Since in the sorted sequence all records in the first subfile may appear to the left of $s(i)$ and all in the second subfile to the right of $s(i)$, these two subfiles may be sorted independently. The method is best stated recursively as below and where INTERCHANGE (x,y) performs t x; x y; y t.

Algorithm for Quick Sort

Step 1 − Choose the highest index value has pivot
Step 2 − Take two variables to point left and right of the list excluding pivot
Step 3 − left points to the low index
Step 4 − right points to the high
Step 5 − while value at left is less than pivot move right
Step 6 − while value at right is greater than pivot move left
Step 7 − if both step 5 and step 6 does not match swap left and right
Step 8 − if left ≥ right, the point where they met is new pivot



```
void qsort(int f , int  l)
{       int i,j,m,pe;
        i=f;
        j=l+1;
        m=i;
        pe=a[f];
        if(f<l)
        {       do
                {       do      i++;    while(a[i]<pe);
                        do      j--;    while(a[j]>pe);

                        if(i<j)
                                swap(&a[i],&a[j]);
                }while(i<j);
                swap(&a[j],&a[m]);
                qsort(f,j-1);
                qsort(j+1,l);
        }
}
void  swap(int *x,  int *y)
{       int temp;
        temp=*x;
        *x=*y;
        *y=temp;
}
```

Q5. Explain the algorithm to convert infix expression to postfix expression? Illustrate with the example. A+(B*C-(D/E-F)*G)*H

Sol:

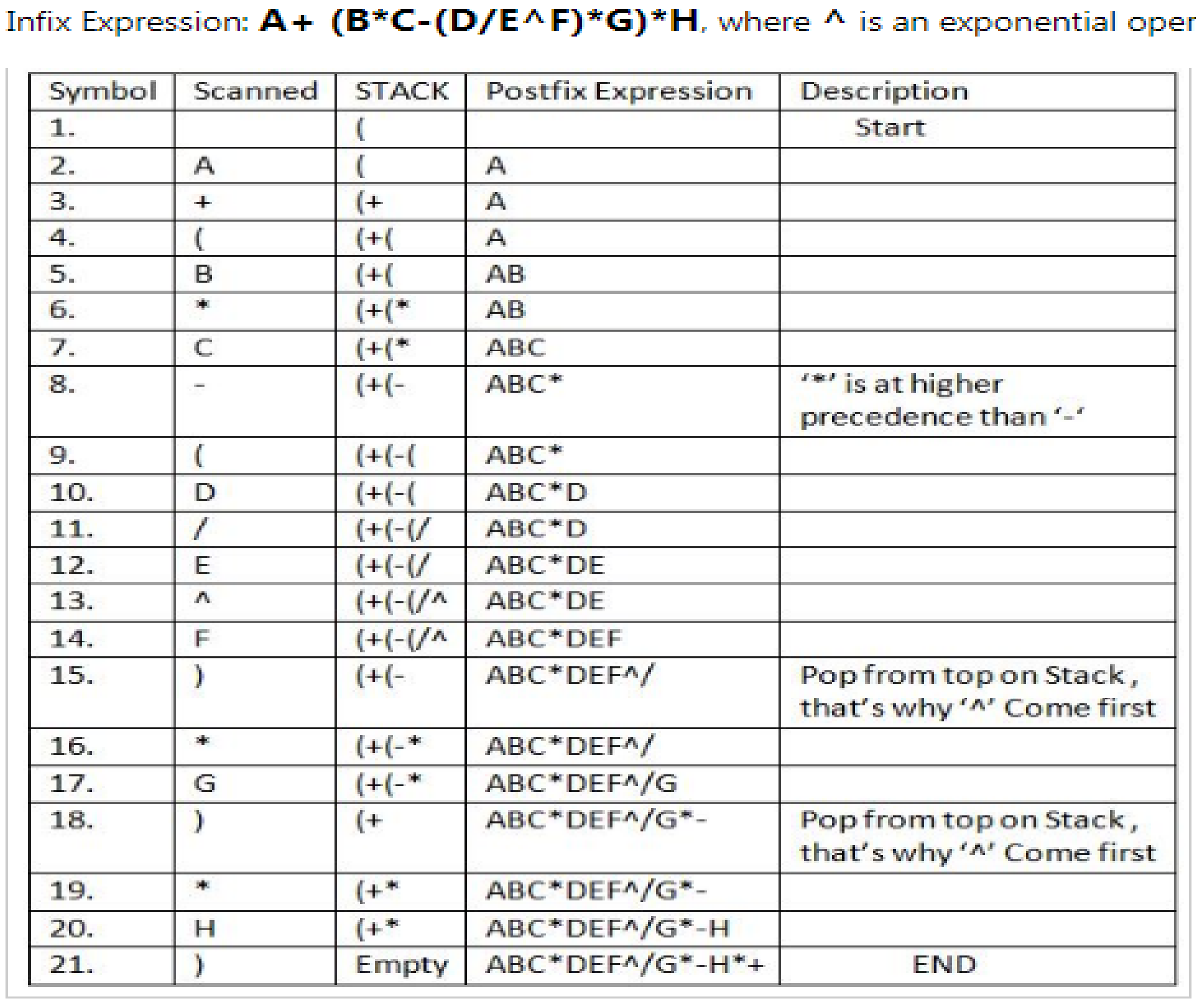
Algorithm to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push "("onto Stack, and add ")" to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.

3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
       [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
       [End of If]
       [End of If]
7. END.

Example : A+(B*C-(D/E-F)*G)*H --- Trace Table;



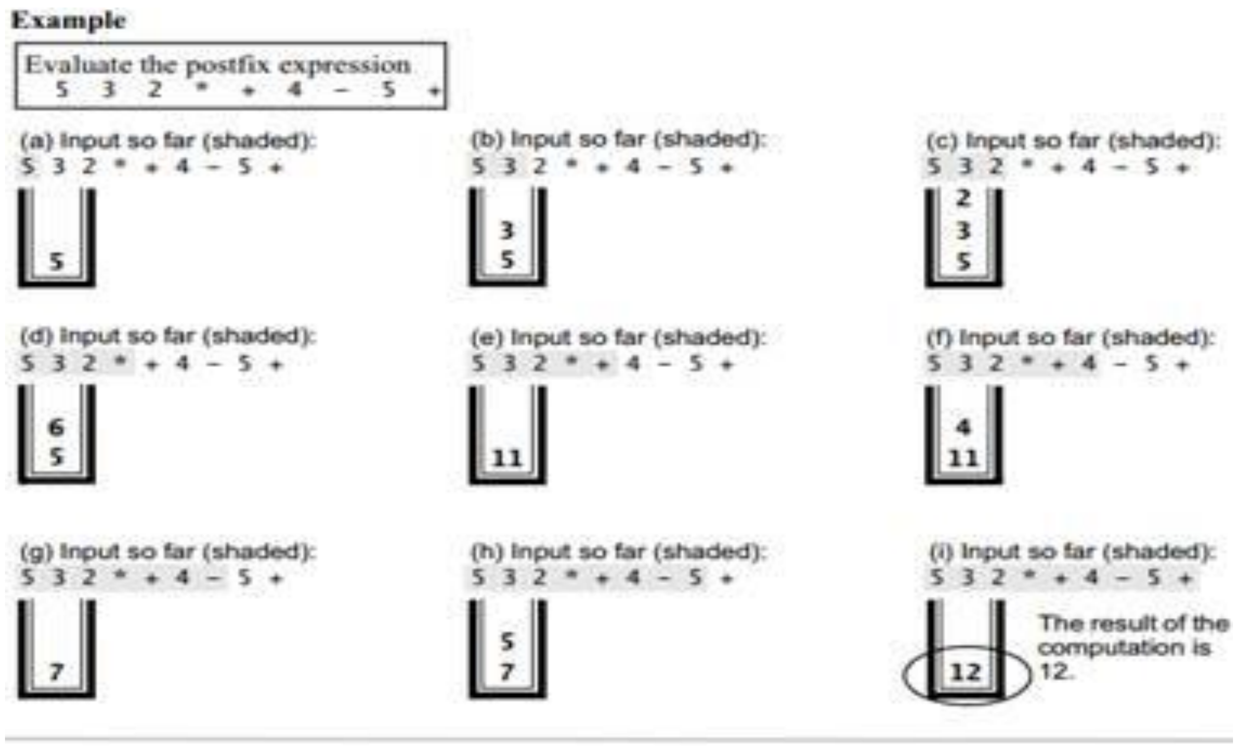
Infix Expression: **A+ (B*C-(D/E^F)*G)*H**, where ^ is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack, that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack, that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Q6. Write an algorithm to evaluation of a postfix expression . Explain with an example.

Sol :
The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared toinfix notation as parenthesis are not required in postfix. We have discussed infix to postfix conversion. In this post, evaluation of postfix expressions is discussed.
Following is algorithm for evaluation postfix expressions.
1) Create a stack to store operands (or values).

2) Scan the given expression and do following for every scanned element.
   a) If the element is a number, push it into the stack
   b) If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
3) When the expression is ended, the number in the stack is the final answer


Example
Evaluate the postfix expression
5 3 2 * + 4 - 5 +
(a) Input so far (shaded):
5 3 2 * + 4 - 5 +
5
(b) Input so far (shaded):
5 3 2 * + 4 - 5 +
3
5
(c) Input so far (shaded):
5 3 2 * + 4 - 5 +
2
3
5
(d) Input so far (shaded):
5 3 2 * + 4 - 5 +
6
5
(e) Input so far (shaded):
5 3 2 * + 4 - 5 +
11
(f) Input so far (shaded):
5 3 2 * + 4 - 5 +
4
11
(g) Input so far (shaded):
5 3 2 * + 4 - 5 +
7
(h) Input so far (shaded):
5 3 2 * + 4 - 5 +
5
7
(i) Input so far (shaded):
5 3 2 * + 4 - 5 +
12
The result of the computation is 12.

```
#include<iostream>
#include<string>
#include<bits/stdc++.h>
#define MAX 20
using namespace std;

class evaluation
{
int top;
        int s[20];
        int rank,i,l,val;
        int t1,t2,t3;
        char str[20];

        public:
        int push(int);
        int pop();
        void eval();
        void valid();
        void read();
        evaluation()
        {       rank=0;
                top=-1;
        }
};
```

```
void evaluation::read()
{       cout<<"\nEnter the Expression : ";
        cin>>str;
        l=strlen(str);
}


int evaluation::push(int val)
{       if(top >= MAX)
                cout<<"\nStack Overflow.";
        else
        {       top++;
                s[top]=val;
        }
        return 0;
}


int evaluation::pop()
{       int val;
        if(top==-1)
                cout<<"\nInvalid Expression.";
        else
                val=s[top--];
        return val;
}


void evaluation::valid()
{       for(i=0;i<l;i++)
        {       if( str[i]=='+' ||str[i]== '-'||
                str[i]=='*'||str[i]=='/'||str[i]=='^')
                        rank--;
                else
                        rank++;
        }
        if(rank != 1)
        {       cout<<"\nInvalid Expression.";
                exit(0);
        }
}


void evaluation::eval()
{       for(i=0;i<l;i++)
        {       if(str[i]!='+'&&str[i]!='-'&&str[i]!='*'
                &&str[i]!='/'&&str[i]!='^')
                {       cout<<"\nEnter the value of "<<str[i]<<" = ";
                        cin>>val;
                        push(val);
                }
                else
                {       t2=pop();
```

```
                        if(top==-1)
                        {       cout<<"\nInvalid Expression.";
                                exit(0);
                        }
                        t1=pop();

                        switch(str[i])
                        {       case '+':t3=t1+t2;      break;
                                case '-':t3=t1-t2;      break;
                                case '*':t3=t1*t2;      break;
                                case '/':t3=t1/t2;      break;
                                case '^':t3=pow(t1,t2);break;
                        }
                        push(t3);
                }
        }
        t3=pop();
        cout<<"\nThe evaluted value of expression " <<str<<" = "<<t3;
}



int main()
{       evaluation e1;
        cout<<"\nEvaluation of Postfix Expression : ";
        e1.read();
        e1.valid();
        e1.eval();
        return(0);

}
```

Q7a). What is recursion? Explain the process of recursion

Sol :

Recursion - Definition

- A function is recursive if a statement in the body of the function calls itself.
- Recursion is a name given for expressing anything in terms of itself.
- When a procedure contains a call to itself it is called ***direct recursion***.
- When a procedure calls another procedure, which in turn calls the first procedure it is called as ***indirect recursion***.
- In recursion calling function and called function are same.

Properties of Recursion

- It must not generate an infinite sequence of calls on itself since any algorithm which does that do not terminate.
- For at least one argument a recursive function f must be defined in terms that do not involve f.
- Each time it calls itself it must be some how nearer to the solution.
- Without a non recursive exit no recursive function can be computed.

Q7b). Write a recursive algorithm to find the nth term in Fibonacii series. Give the recursion tree

Sol :

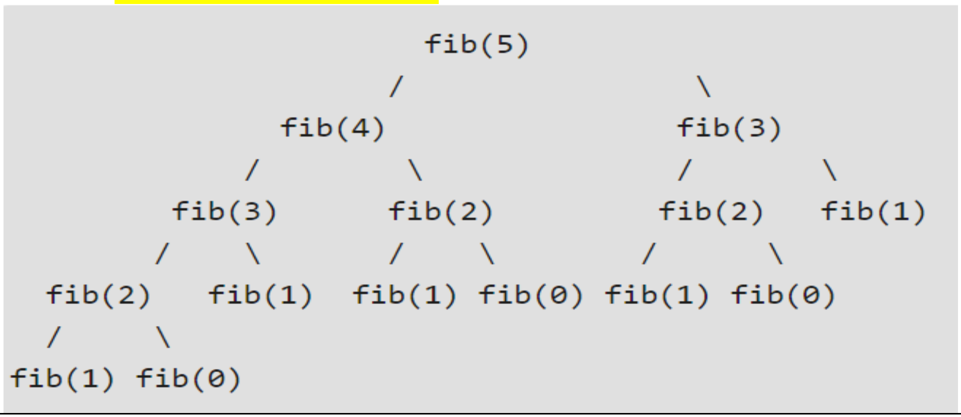
The Fibonacci numbers are the numbers in the following integer sequence.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..
In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

$$F_0 = 0 \quad \& \quad F_1 = 1$$

```
int  fibo( int  n )
{
   if ( n <= 1 )
      return n;
   return ( fibo( n-1 ) + fibo( n-2 ) ) ;
}
```

Recursion Tree for n=5



```
                          fib(5)
                      /              \
                fib(4)                fib(3)
             /        \              /       \
         fib(3)        fib(2)        fib(2)   fib(1)
        /     \       /     \       /     \
   fib(2)   fib(1)  fib(1) fib(0) fib(1) fib(0)
   /     \
fib(1) fib(0)
```

Q 8 a) Write an algorithm to convert Prefix expression to Postfix expression? Illustrate with an example..

Sol :
Algorithm for Prefix to Postfix:

Step-1 : Read the Prefix expression in reverse order (from right to left)
Step-2 : If the symbol is an operand, then push it onto the Stack
Step-3 : If the symbol is an operator, then pop two operands from the Stack
 Create a string by concatenating the two operands and the operator after them.
 string = operand1 + operand2 + operator
 And push the resultant string back to Stack
Step-4 : Repeat the above steps until end of Prefix expression.

Example :
Prefix : An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).
Example : *+AB-CD (Infix : (A+B) * (C-D))
Postfix: An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).
Example : AB+CD-* (Infix : (A+B * (C-D))

```
#include <iostream>
#include<string>
#include <stack>
using namespace std;

// funtion to check if character is operator or not
bool isOperator(char x) {
 switch (x) {
 case '+':
 case '-':
 case '/':
 case '*':
   return true;
```

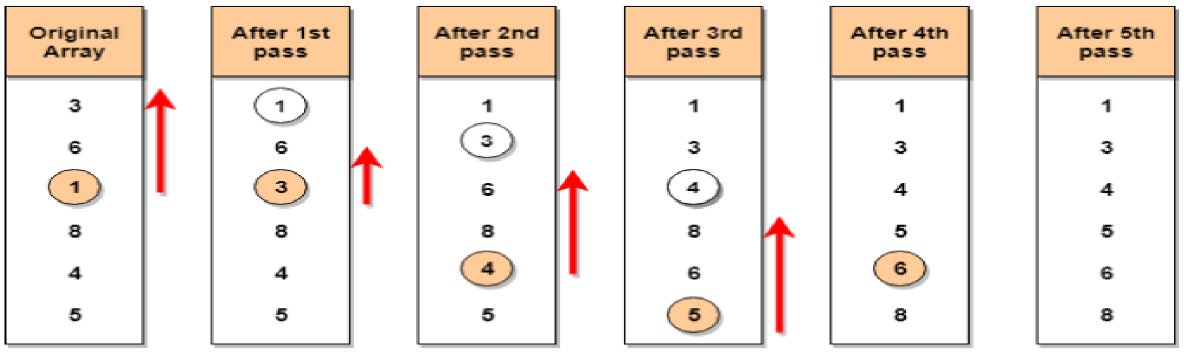
```
  }
  return false;
}

// Convert prefix to Postfix expression
string preToPost(string pre_exp)
{
   stack<string> s;

  int length = pre_exp.size(); // length of expression

  for (int i = length - 1; i >= 0; i--) // reading from right to left
   {

    if (isOperator(pre_exp[i]))  // check if symbol is operator
          {

      string op1 = s.top(); s.pop(); // pop two operands from stack
      string op2 = s.top(); s.pop();

      string temp = op1 + op2 + pre_exp[i];  // concat the operands and operator

      s.push(temp);           // Push string temp back to stack
    }

    else {   // if symbol is an operand

       s.push(string(1, pre_exp[i]));  // push the operand to the stack
       }
    }
  return s.top();   // stack contains only the Postfix expression
}
 // funtion to check expression is valid  or not
void valid( string str)
{ int i,rank=0;
    int l = str.size();
          for(i=0;i<l;i++)
          {         if( str[i]=='+' ||str[i]== '-'||
                    str[i]=='*'||str[i]=='/'||str[i]=='^')
                              rank--;
                    else
                              rank++;
          }
          if(rank != 1)
          {         cout<<"\nInvalid Expression.";
                    exit(0);
          }
}

int main() {
  string prefixexp;
  cout<<"\nEnter Prefix Expression : ";
  cin>>prefixexp;
  valid(prefixexp);
  cout << "\nPostfix : " << preToPost(prefixexp);
  return 0;
}
```

Q9a). Write an algorithm for selection sort. Explain with an example

Sol :

Algorithm

Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted



Q9 b) Write a recursive algorithm to find the GCD of two numbers.

Sol ;

GCD or Greatest Common Divisor of two or more integers is the largest positive integer that divides both the number without leaving any remainder.

Example: GCD of 20 and 8 is 4.

```
int gcd(int x, int y)
   {
   int  a, b;
   a = (x > y) ? x : y; // a is greater number
   b = (x < y) ? x : y; // b is smaller number
    if ( b != 0)
      return gcd(b, a%b);
   else
      return a;
}
```

Finding GCD:

```
14 ) 48 ( 3
     42
     -----
      6
```

Remainder is not yet zero, so we will now divide 14 by 6

```
14 ) 48 ( 3
    42
    -----
```

Q10 a) Write an algorithm to search an item in a given list using binary search . Illustrate with an example

Sol :

Binary search is a technique to search an item in a given sorted list of elements. First , it start searching data from middle of the list. If it is a match, return the position of the it , and exit Otherwise, If data is greater than middle, search in higher sub-list else If data is smaller than middle, search in lower sub-list. Repeat until match.

Algorithm for Binary Search:

```
int binarySearch(int a, int item,int n)
{      lb = 0
       ub = n
            while( lb < ub )
                 mid =  (lb + ub) / 2
              if a [mid] == item:
                     return (mid+1)
              else if ( a [mid] < item)
                        lb =  mid + 1
              else if a [mid] > item:
                      ub = mid – 1
          return (-1)
}
```

Q10 b) Write an algorithm for Bubble sort. Explain with an example.

Sol :

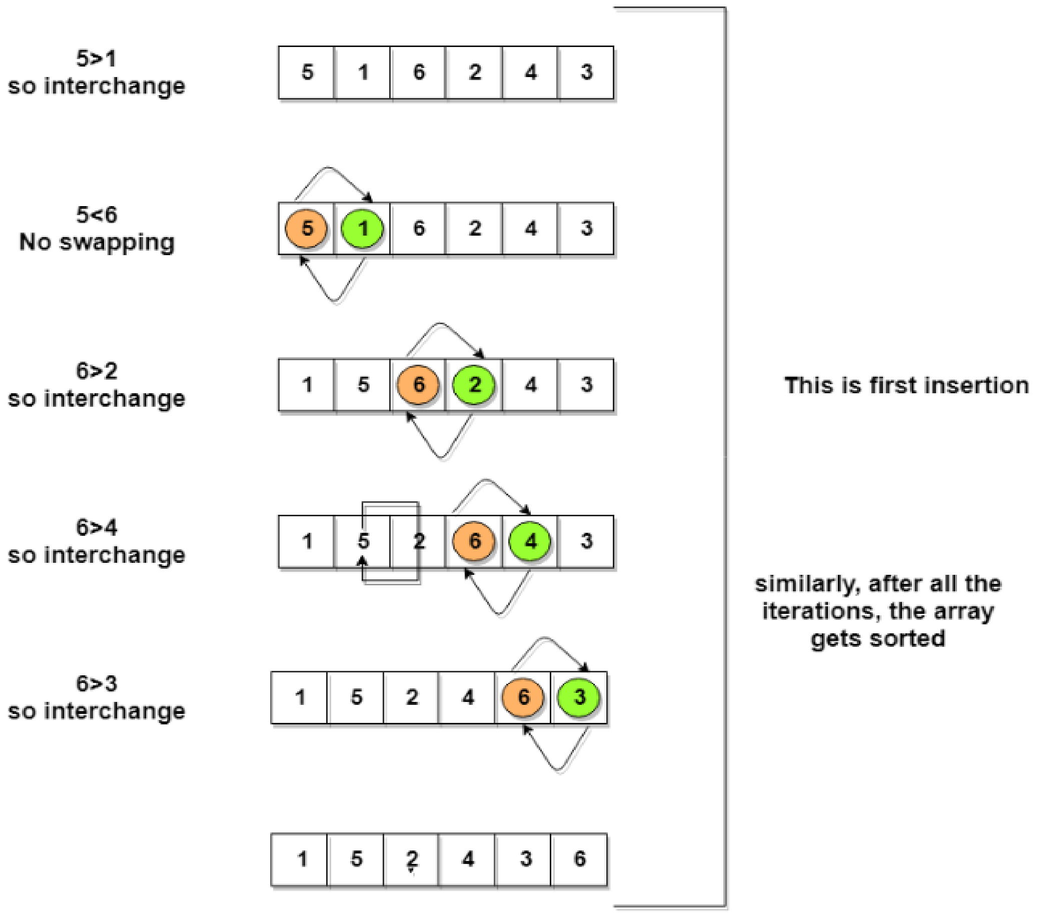
Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.

If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on.

If we have total n elements, then we need to repeat this process for n-1 times.

It is known as bubble sort, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.

Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.



```
#include <stdio.h>
void main()
{
   int a[10],n, i,j,temp;
   printf("\n Enter number of  elements of an array:\n");
            scanf("%d", &n);

   printf("\nEnter elements of an array:\n");
   for (i=0; i<n; i++)
      scanf("%d", &a[i]);
   printf("\n Original Array: \n");
    for (i=0; i < n; i++)
       printf("%d   ", a[i]);
  for (i = 0; i < n-1; i++)
      for (j = 0; j < n-i-1; j++)
          if (a[j] > a[j+1])
           {
                 temp = a[j];
             a[j] = a[j+1];
             a[j+1] = temp;
                           }
         printf("\n Sorted Array: \n");
   for (i=0; i < n; i++)
       printf("%d   ", a[i]);
}
```