UNIT 1

Prof. Chaitali R Patil

- Algorithm Design Tools:
  - Pseudocode and Flowchart

The algorithm is to sort the array A of size N.

```
Algorithm sort(ref A<integer>, val N<integer>)
Pre array A to be sorted
Post sorted array A
Return None
1. if(N < 1) goto step (4)
2. M = N - 1
3. For I = 1 to M do
    For J = I + 1 to N do
    begin
          if(A(I) > A(J))
                    then
                    Begin
                              T = A(I)
                              A(I) = A(J)
                              A(J) = T
                    end
          end if
    end
4. stop
```

Algorithm to searche for an element in an array

Algorithm search (val list<array>,val X<integer>)

Pre list containing data array to be searched and

argument containing data to be located

Post None

Return Location

1.Let list be the array and X be the element to be searched

2.For I = 1 to N do

```
    begin
          if(List(I) = X)
          then
                    Return I
          End if
    end
```

3.Return -1

4.stop

Write an algorithm to compute the following:

$$P = n!/(n - r)!$$

Pre None **Post None**

Return Result

1. Read n and r

2. Let

(a) A = FACT(n) and

(b) B = FACT(n – r)

3. Result = A / B

4. Print Result

5. Stop

Here FACT is the subalgorithm to compute the factorial of a number as

$$n! = n \times (n - 1) \times (n - 2) \times \ldots \times 1$$

subalgorithm FACT

1. Read n

2. Let Result = 1

3. while(n not equal to 1) do

Result = Result × n

n = n – 1

end while

4. Return Result

```
class Array
{   private:
            int MaxSize;
            int A[20];
            int Size;
    public:
            Array() // constructor
            {
                        MaxSize = 20;
                        Size = 0;
            }
    void Read_Array();
    void Display(); // Traverse_Forward()
    void Traverse_Backward();
    void Insert(int Location, int Element);
    void Delete(int Location);
    int Search(int Element);
};
```

```
void Array :: Read_Array()
{
    int i, N;
    cout << "Enter size of array";
    cin >> N;
    if(N > MaxSize)
    {
        cout << "Array of this size cannot be created";
        cout << "Maximum size is" << MaxSize;
        return;
    }
    else
    {
        for(i = 0; i < N; i++)
        {
            cin >> A[i];
        }
        Size = N;
    }
}
```

```
void Array :: Display()
{
    int i;
    for(i = 0; i < Size; i++)
    cout << A[i] << "\t";
    cout << endl;
}
```

```
void Array :: Traverse_Backward()
{
    int i;
    for(i = Size - 1; i >= 0; i––)
    Cout << A[i] << "\t";
    cout << endl;
}
```

```
int Array :: Search(int Element)
{
    int i;
    for(i = 0; i < Size - 1; i++)
    {
        if(Element == A[i])
        return(i);
    }
    return(-1);
}
```

Inserting an Element into an Array

Data shifting can be performed using the following function

```
void Array :: Insert(int Location, int Element)
{
    int i;
    if(Size >= MaxSize)
    {
        cout << "Sorry, Array Overfl ow";
        return;
    }
    For(i = Size - 1; i >= Location - 1; i--)
    {
        A[i + 1] = A[i]; // shifting element to right by 1 position
    }
    A[Location - 1] = Element;
    Size = Size + 1;
}
```

Deleting an Element

```
void Array :: Delete(int Location)
{
    int i;
    for(i = Location; i < Size; i++)
    {
        A[i - 1] = A[i];
        // shifting elements to the left by 1 position
    }
    A[Size - 1] = 0;
    // Store 0 at the last location to mark it empty
    Size = Size - 1;
}
```

```
void main()
{
    Array A;
    A.Read_Array();
    A.Display(); // Traverse_Forward()
    A.Traverse_Backward();
    A.Insert(3, 66); // insert at position 3
    A.Display();
    cout << endl;
    A.Delete(3); // delete 4th element
    A.Display();
    cout << endl;
    cout << A.Search(66);
    cout << A.Search(3);
}
```

Abstract Data Type

We can define *data structures as follows:*

A data structure is a set of domains D, a designated domain d E D, a set of functions F, and a set of axioms A. The triple structure (D, F, A) denotes the data structure with the

following elements:

Domain (D) This is the range of values that the data may have.

Functions (F) This is the set of operations for the data. We must specify a set of operations for a data structure to operate on.

Axioms (A) This is a set of rules with which the different operations belonging to F can **actually be implemented.**

d = Integer

```
Integer
Domain D = {Integer, Boolean}
Set of functions F = {zero, ifzero, add, increment}
Set of axioms A = {
                    ifzero(zero()) → true;
                    ifzero(increment(zero()) → false
                    add(zero(), x) → x
                    add(increment(x), y) = increment(add(x, y))
                    equal(increment(x), increment(y) = equal(x, y)
                 }
end Integer
```

```
Abstract data type Integer
  Operations
     zero() → int
     ifzero(int) → boolean

     increment(int) → int
     add(int, int) → int
     equal(int, int) → boolean
  Rules/axioms for operations
     for all x, y ∈ integer let
       ifzero(zero()) → true;
       ifzero(increment(zero()) → false
       add(zero(), x) → x
     add(increment(x), y) → increment(add(x, y))
     equal(increment(x), increment(y) → equal(x, y)
  end Integer
```

This is an example of the `Integer` data structure; five basic functions are defined on a set of integer data object. These functions are as follows:

1. `zero()` $\rightarrow$ `int`—It is a function which takes no input but generates the integer zero as result. That is, its output is 0.
2. `ifzero(int)` $\rightarrow$ `Boolean`—This function takes one integer input and checks whether that number is 0 or not. It generates output of type True/False, that is, of the Boolean type.
3. `increment(int)` $\rightarrow$ `int`—This function reads one integer and produces its incremented value, that is, (integer + 1), which is again an integer.
   For example, `increment(3)` $\rightarrow$ `4`
4. `add(int, int)` $\rightarrow$ `int`—This function reads two integers and adds them producing another integer.
5. `equal(int, int)` $\rightarrow$ `Boolean`—This function takes two integer values and checks whether they are equal or not. Again, it gives output of the True/False type. So its output is of Boolean type.

The set of axioms which describes the rules of operations is as follows:

1. `ifzero(zero)` → `true`—This axiom says that the `zero()` function which produces an integer zero, is checked by the `ifzero()` function, and ultimately the result is true.
2. `ifzero(increment(zero()))` → `false`—The value of `increment(zero)` is 1 and hence `ifzero(1)` is false.
3. `add(zero(), x)` → `x` —This means that `0 + x = x`.
4. `add(increment (x), y)` → `increment (add(x, y))`—Assuming `x = 3` and `y = 5`, this means that `add(increment (3), 5) = increment(add(3, 5)) = add(4, 5) = increment(8) = 9`.
5. `equal(increment(x), increment(y))` → `equal(x, y)`— This axiom specifies that if x and y are equal, then x + 1 and y + 1 are also equal.