Sequential statement ⇒ Here the statements are executed serially.

⇒ Write an algorithm to find area of circle.

[It is an algorithm for finding area of a circle. It is written by xyz]

```
→ Step 1 : start
Step 2 : Let a, r
Step 3 : Display "Enter Radius:"
Step 4 : Input r
Step 5 : a = 3.141 * r * r
Step 6 : Display "The area of circle is", a
Step 7 : Stop
```

⇒ Write an algorithm to swap two numbers using a 3rd variable.

[Swapping algorithm]

```
→ step 1 : start
step 2 : Let a, b, temp
step 3 : Display "enter two numbers:"
step 4 : Input a, b
step 5 : temp = a
step 6 : a = b
step 7 : b = temp
step 8 : Display " a contains ", a
step 9 : Display " b contains ", b
step 10 : stop
```

(2) Selection control statements ⇒

If statement : It allows to select a block of statements for execution based on a condition.

If can be used in 4 ways:

1) simple if
2) If .. else
3) Else if ladder
4) Nested if

1. if statement :

```
if (condition) then
    statements
    - - - - - - - .     } → available inside the if statement
[end of if]
```

2. if ... else statement :

```
if (condition) then
block1 statements
- - - - - - - - -
else
block2 statements
- - - - - -
[end of if] - - - -
```

⇒ Write an algorithm to find the greatest among two unequeal numbers.
[finding greatest among two numbers]

```
→ Step 1: start
  Step 2: Let A,B
  Step 3: Display "Enter two unequeal numbers"
  Step 4: Input A,B
  Step 5: if (A>B) then
          Step 5.1: Display "The greatest is", A
  Step 6: Else
          Step 6.1: Display "The greatest is", B
          [end of if]
  Step 7: stop
```

⇒ Write an algorithm to find the greatest among 3 unequeal numbers.
[finding the greatest among 3 numbers]

```
→ Step 1 : start
  Step 2 : Let A,B,c
  Step 3 : Display "Enter three unequeal numbers"
  Step 4 : Input A,B,c
  Step 5 : if (A>B AND A>c) then
          Step 5.1: Display "The greatest is", A
  Step 6 : Else if(B>c) then
          Step 6.1: Display "The greatest is", B
  Step 7 : Else
          Step 7.1: Display "The greatest is", c
          [end of if]
  Step 8 : stop
```

(3) Loop control statements ⇒
They are:
1) While statement and
2) for statement

1. **While statement:**

```
while (condition)
    statements
    - - - - - - -
[end of while]
```

⇒ Write an algorithm to find factorial of a number.
[finding factorial]

```
→ step 1: start
  step 2: Let i, fact, n
  step 3: Display "Enter a number"
  step 4: Input n
  step 5: i=1, fact =1
  step 6: while (i<=n)
```

```
    Step6.1 :fact=fact*i
    step6.2: i=i+1
     [end of while]
Step7: Display "Factorial value is",facct
step8: stop
```

⇒ Write an algorithm to test a number is prime or not.

```
→ Step1 : start [prime/not]
Step2 : Let n, i, j
step3 : Display "Enter a number"
step4 : Input n
step5: i= 1, j=0
Step6: while (i<=n)
~~Step 6.1: no = n%10~~
~~Step 6.2: n = n/10~~
~~Step 6.3: j=j+1~~
    step 6.1 : if (n%i=0) then
        Step 6.1.1 : j=j+1
        [end of if]
    Step 6.2 : i=i+1
    [end of while]
Step7 : if (j=2) then
   Step7.1: Display "prime number"
Step8 : Else
    Step8.1: Display "not prime number"
          [end of if]
Step9: stop
```

⇒ Write an algorithm to test a number is Armstrong or not.

⇒ Write an algorithm to find GCD of two numbers

⇒ Write an algorithm to test a number is perfect or not.

⇒ Write an algorithm to test a number is strong or not.

⇒ Write an algorithm to generate Fibonacci series of N numbers.

[Armstrong number /not]

```
→ Step1: start
Step2: Let sum, temp, n, r
step3: Display "Enter a number"
step4: Input n
step5: sum=0 , temp=n
step6: while (n>0)
      step6.1: r=n%10
      Step6.2: sum = sum+(r×r×r)
      step6.3: n= n/10
        [end of while]
step7: if (sum=temp) then
      step 7.1: Display "It is an armstrong number"
step8: Else
      step8.1: Display "It is not an armstrong number"
      [end of if]
step9: stop
```

```
→ [finding GCD]
step1: start
step2: Let n1, n2, gcd, i
step3: Display "Enter 2 numbers"
step4: Input n1 and n2
step5: gcd = 1, i = 1
step6: while (i<=n1 AND i<=n2)
        Step6.1: if (n1 % i = 0 AND n2 % i = 0) then
            Step6.1.1: gcd = i
                [end of if]
        Step 6.2: i = i+1
                [end of while]
Step7: Display "gcd is ", gcd
Step8: stop
→ [perfect number/not]
Step1: start
Step2: Let n, s, i
Step3: Display "enter a number"
Step4: Input n
Step5: s = 0, i = 1
Step6: while (i<n)
        Step6.1: if (n%i=0) then
            Step6.1.1: s = s+i
                [end of if]
        Step6.2: i = i+1
                [end of while]
Step7: if (s = n) then
        Step7.1: Display "It is a perfect number"
Step8: Else
        Step8.1: Display "It is not a perfect number"
            [end of if]
Step 9: stop
→ [strong number/not]
Step1: start
Step2: Let n, no, sum, fact, i, r
Step3: Display "Enter a number"
Step4: Input n
Step5: no = n, sum = 0
Step6: while (n>0)
        Step6.1: r = n%10
            Step6.2: fact = 1, i = 1
            Step6.3: while (i<=r)
                    Step 6.2.1: fact = fact*i
            Step 6.2.2:i =i+1
                  [end of while]
            Step 6.3: sum = sum+fact
            Step 6.4: n=n/10
                  [end of while]
step 7: If (sum = no) then
        Step 7.1: Display "This is a strong number"
step8: Else
        Step 8.1: Display " This is not a strong number"
        [end of if]
step9: stop
```

→ [Fibonacci Series algo]

```
step 1: start
step 2: Let a,b,n,s,i
step 3: Display " Enter the no. of term up to which to be printed"
step 4: Input n
step 5: a=0, b=1, s=0, i=1
step 6: Display a,b
step 7: while (i<=n)
        step 7.1: s= a+b
        Step 7.2: a=b
        Step 7.3: b=s
        Step 7.4: i= i+1
        step 7.5: Display s
              [end of while]
Step8: stop
```

* For loop:

Syntax

```
For(initial value to final value), increment by 1
    Statements block
    [end of for]
```

ex: Let t

```
For (i= 1 to 100), increment by 1
    Display "Hello World"
    [end of for]
```

↙ In while loop

```
Let i
  i=1
while (i<=n)
    Display" Hello"
    i= i+1
    [end of while]
```

→ Write an algorithm to test a number is prime or not using for loop.

→ [prime/not algo]

```
Step 1: Start
Step 2: Let n, i, count
Step 3: Display "Enter a no"
Step 4: Input n
Step 5: count = 0
Step 6: For (i=1 to n), increment by 1
    Step 6.1: if (n%i=0) then
        Step 6.1.1: count = count + 1
            [end of if]
    [end of for]
Step 7: if (count=2) then
    Step 7.1: Display "It is a prime"
Step 8: Else
    Step 8.1: Display "It is not a prime"
        [end of if]
Step 9: Stop
```

Array

Types of array:

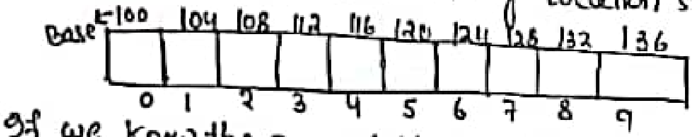
① 1D Array

② Multi Dimensional Array

One - Dimensional array:

The memory representation will be:

ex: Let a [10], Assume memory location size = 4 bytes



If we know the Base Address, then we can find the address of

$\boxed{a[i] = Base + w * i}$

where w is size of memory location.
i is the index. position.

ex: for a[10], Base = 100
w = 4 bytes

To find address of
A[4] = 100 + 4 * 4 = 116

The operations on 1D array are:

① Traversion
② Incertion
③ Deletion
④ Searching
⑤ Sorting
⑥ contafenating
⑦ Merging
⑧ creation

Assume an array A[100] declared, We need to input N data into it.

Creation Operation ⇛

- Here we define an array with large size.
- Input N elements into it.
- Specify the lower bound (LB) and upper bound (UB).
- All the array from LB to UB for other operations.
- Here we create individual subroutine or procedure or module for each operation similar to independent function.

→ Algorithm →

```
Creation (la[100], lb, ub, n)
Step1 : start
Step2 : Let i
Step3 : Display " how many numbers to store "
Step4 : Input n
Step5 : i=0
Step6: while (i<=n-1)
        Step 6.1 : input la[i]
        Step 6.2 : i= i+1
             [end of while]
Step7: Lb=0 ,ub= n-1
Step8 : exit
```

(1) Traversion Operation ⇛

- Visiting all the elements in an array is called traversion.
- It can be displaying the elements, counting the elements etc.

→ Algorithm →

```
Traversion( la[100], lb ,ub)
Step1:
Step2 : Let i
Step 3 : Display "the elements in array are :"
Step4: i= lb
Step5 : while ( i<=ub)
        Step5.1 : Display la[i]  OR  Traverse la[i]
        Step5.2 : i= i+1
             [end of while]
Step6 : exit
```

Using for loop ⇛

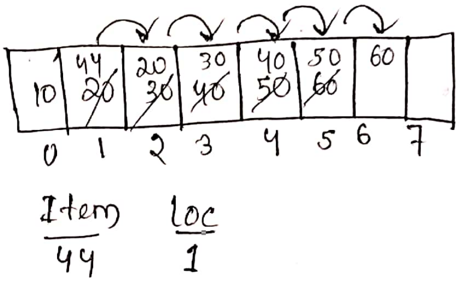
```
Step4: For (i=lb to ub), incr by 1
     Step4.1 : Display la[i] OR Traverse  la[i]
          [end of for]
Step 5 : exit
```

(2) Insertion Operation ⇒

When we need to insert a data within the given boundaries LB and UB then we need to vacant the location before storing the data. It is possible by shifting the elements from UB to the location.

→ Algorithm→

```
Insertion (la[100], lb,ub , item, loc)
step1 : start
step2 : let i
step3: Display "enter item and location"
step4: intput item, loc
steps : for (i=ub to loc), decr by 1
        step 5.1 : la[i+1]=la[i]
              [end of for]
step6: la[ loc ]= item
step7: ub= ub+1
step 8: Stop
```

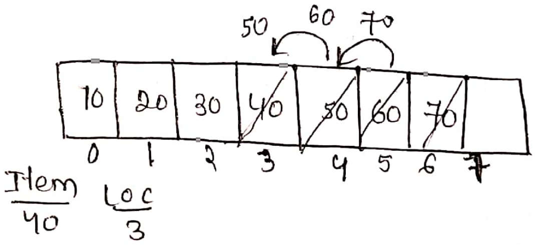


(3) Deletion operation ⇒

When we need to delete an element within the given boundaries then after deleting the value we cannot keep the vacant memory within the list. so, we must follow shifting process to fill the vacant location. so, for deletion we need to shift the elements from the location to ub.

→ Algorithm→

```
Deletion (la[100], lb,ub ,item, loc)
step1: start
step2: let i
step3: Display "Enter item to be deleted"
step4: input item
Steps: for (i= lb to ub), incr by 1
      steps5.1: if (item = la[i]) then
            step 5.1.1: Break
                  [end of if]
            [end of for]
step6: if (i>ub) then
      step 6.1 : Display "element to be deleted is not found"
step7: else
      step7.1: loc=i
      step7.2: for (i=loc to ub), incr by 1
            step7.2.1 : la[i] = la[i+1]
                  [end of for]
      step 7.3 : ub =ub-1
            [end of if]
step 8: exit
```



* Deletion based on location given:

```
Deleteion (la[100], lb ,ub, loc)
step 1 :start
step 2 : let i
step 3: Display 'enter location to be deleted"
step4: input loc
steps: for (i = loc to ub), incr by 1
```

```
    Step 5.1 : la[i]= la[i+1]
             [end of for]
Step 6: ub = ub-1
Step7: exit
```