UNIT - III

* **Arrays** :- List of elements / collection of Homogeneous data items

Syntax :-

```
Datatype arrayname [size];
```

Ex :-

```
int a[10];
char b[5];
float c[10];
double d[5];
```

Rules :-

→ We can store 'many' data items'

→ All the data items must be 'same type'

Ex :- int a[4] → 1, 2, -3, 0 ✓

float b[4] → 2.2, 0.6, -3.1, 4.7 ✓

char c[3] → a, b, c ✓

int a[4] → 1, -3, 4.7, c ✗

→ We must specify the 'size'

Ex: int a[ ]; ✗ int a[3x2]; ✓

int a[-5]; ✗ int a[11/2]; ✓

int a[2+2]; ✓ int b; int a[b=11/2]; ✓

* **Initialization**

① At compile time.

② At Run time.

① **At compile time** :-

int a[5] = {0, -1, 11, 10, 2}; ✓

int a[ ] = {0, 1, 2, 0, -1, 6, 7}; ✓

int a[5] = {0, 1, -1}; ✓

int a[5] = {1, 2, 3, 4, 5, 6, 7}; error

int a[5]; ~~error~~ garbage values

int a[5] = {0}; ✓

int a[5] = {} error

char b[10] = {'v', 'e', 'd', 'a'}; ✓

(2) At run time :-

Ex:-

```
# include <stdio.h>
int main( )
{
   int i;
   int a[5];
   Printf(" Enter element of array : ");
   for(i=0; i<5, i++)
   {
     scanf("%d", &a[i]);
   }
}
```

i:	0	1	2	3	4
	10	0	11	-1	2

* Memory Representation and Accessing of Array

Ex:- int a[5] = {1, 10, 0, -1, 3};

4 x 5 = 20 Bytes

B + [Index x size of int]

	0	1	2	3	4
	1	0	10	-1	3
internal Pointer	2000	2004	2008	2012	2016

Base address :-

a[0] = 1 ⇒ First element → 2000 + [0*4] → 2000

a[1] = 0 ⇒ 2000 + [1*4] → 2004

a[2] = 10 ⇒ 2000 + [2*4] → 2008

a[3] = -1 ⇒ 2000 + [3*4] → 2012

a[4] = 3 ⇒ 2000 + [4*4] → 2016

B + (Index * size of int)

* Brief Notes :-

→ 'Array' is a 'collection' of more than one data item of same type

→ All the data items are stored in contiguous memory locations.

→ No. of data items array holds is 'size of array'

→ once size has declared, it can't be changed at run time [fixed size].

→ Index starts from '0'

→ Known as 'derived data type'.

→ Accessing of any element is faster using Index of array.

→ It allows to store data in multidimensional form.

→ 'Inserting' & 'deleting' elements from array is costly.

→ No bound checking in C.

* Multidimensional Arrays :-

2-D

```
datatype arrayname [rows][col];
```

3-D

```
datatype arrayname [block],[rows],[col];
```

* Examples :-

→ int a[2],[3]; →

	0	1	2
0	(0,1)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)

* 2-D Arrays :-

syntax :- datatype 'arrayname [row][col];

Ex:- int a[2][3];

* Initialization :-

① Direct :-

int a[2][3] = {10, 20, 30, 40, 50, 60};

int a[3][3] = {{10, 20, 30}, {40, 50, 60}};

int a[][3] = {{10, 20, 30}, {50, 60, 70}, {80, 90, 100}};

② Individual :-

```
int a[2][3];
    location  value
    a[0][2] = 34;
    a[0][0] = 10;
    a[0][1] = 26;
    a[1][2] = 40;
```

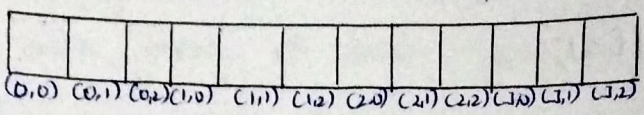
	0	1	2
0	10	26	34
1			40

③ Run-Time Initialization :-

```
int a[2][3], i, j;
for (i=0; i<2; i++)
{
  for (j=0; j<3; j++)
  {
    scanf ("%d", &a[i][j]);
  }
}
for (i=0; i<2; i++)
{
  for (j=0; j<3; j++)
  {
    printf ("%d", a[i][j]);
  }
  Printf ("\n");
}
```

* Row-major order :-

int a[4][3];

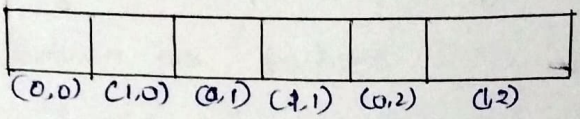


Formula :-

$$A[i][j] \longrightarrow B.A + W * ((i - Lr) * N + (j - Lc))$$

* column- major order :-

int a[2][3];



Formula :-

$$A[i][j] \longrightarrow BA + W * ((i - Lr) + M * (j - Lc))$$

* 'c' program to print

0 -1 -1
1 0 -1
1 1 0

A)

```
#include <stdio.h>
int main()
{
  int a[3][3], i,j;
  for(i=0; i<3; i++)
  {
    for(j=0; j<3; j++)
    { if(i==j)
      a[i][j]=0;
      else if(i<j)
       a[i][j] = -1;
      else
       a[i][j] =1;
    }
  }
  for(i=0; i<3; i++)
  { for(j=0; j<3;j++)
    { printf("%d", a[i][j]); } printf("\n"); } }
```

* operations in 1-d :-

(1) Insertion
(2) Deletion
(3) Traversing
(4) Searching
(5) sorting [A/D].

(1) Insertion & (3) Traversing :-

```
for (i=0; i<n; i++)
{
  scanf ("%d", &a[i]);
}
```

(2) Deletion :-

```
int pos;
scanf ("%d", &pos);
for (i=pos; i<n-1; i++)
{
  a[i] = a[i++]
}
```

(4) searching :-

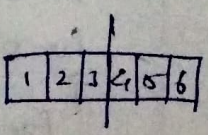
for Linear search

```
int flag=0;
for (i=0; i<n, i++)
if (flag==0)
  printf("element not found");
else
  printf ("element found");
```

```
if (key==a[i])
  flag=1;
  break;
```

Binary search

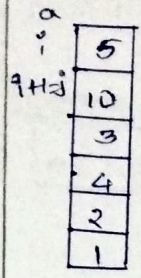
Rule :- All the elements must be in order.



⑤ Sorting :-

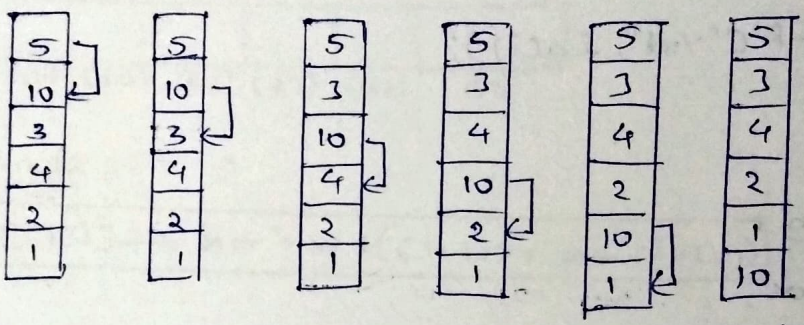
Types :- Bubble short, selection sort, Bucket sort etc.

Selection sort :-




```
for(i=0; i<n; i++)
{for(j=i+1; j<n; j++)
 { if(a[i]> a[j])
   { temp=a[i];
     a[i]=a[j];
     a[j]=temp;
   }
}}
```

Bubble sort :-



→ In Bubble sort, largest value comes to last position

✱ Operations in 2-d :-

① Addition :-

$$\overset{A}{[\ ]}_{m\times n} = \overset{B}{[\ ]}_{m\times n}$$

Ex :-

```
int A[5][5], B[5][5], S[5][5], r1, c1, r2, c2, i, j;
printf("Enter order of a matrix:");
scanf("%d.%d", &r1, &c1);
printf("Enter order of b matrix:");
scanf("%d %d", &r2, &c2);
if(r1==r2 & c1==c2)
{
  printf("Enter %d integers into A matrix\n", r1, c1);
  for(i=0; i<r1; i++)
  {
    for(j=0; j<c1; j++)
    { scanf("%d", &A[i][j]);
  }}
```

```
Printf("Enter %d Values into B matrix \n", r2, c2);
for(i=0; i<r2; i++)
{
  for(j=0; j<c2; j++)
  {
    scanf("%d", &B[i][j]);
  }
}
}
for (i=0; i<r1; i++)
{
  for(j=0; j<c1; j++)
  {
    S[i][j] = A[i][j] + B[i][j];
  }
  Printf("Enter Elements are %d", S[i][j]);
  printf("\n");
}
}
```

(2) Multiplication:-

$A[r_1][c_1]$, $B[r_2][c_2]$

if $(c_1 == r_2)$

Extra loop

```
for(i=0; i<r1; i++)
{
  for(j=0; j<c2; j++)
  {
    S[i][j]=0;
    for(k=0; k<c2; k++)
    {
      S[i][j]=S[i][j]+A[i][k] * B[k][j];
    }
  }
}
```

(3) Transpose (4) Diagonal (5) Substraction

* Arrays [3-Dimensional] :-

Ex :- int a[3][3][3];

block row col

* Initialization

① Direct :-

```
int a[3][3][3] = { { {1,2,3}, {4,5,6}, {7,8,9} },
                   { {10,11,12}, {1,2,3}, {4,5,6} },
                   { {7,8,9}, {4,5,6}, {1,2,3} } };
```

② Run time :-

```
int a[3][3][3];
for (i=0; i<3; i++)
{
  for (j=0; j<3; j++)
  {
    for (k=0; k<3; k++)
    {
      scanf ("%d", &a[i][j][k]);
    }
  }
}
for (i=0; i<3; i++)
{
  for (j=0; j<3; j++)
  {
    for (k=0; k<3; k++)
    {
      printf ("%d", a[i][j][k]);
    }
  }
}
```

STRINGS

* → String is defined as 'sequence of characters' and terminated with '\0' — null character.

Syntax :-

char array name [size].

Initialization :-

① Direct :-

Ex: char name [10] = {'a', 'b', 'c', 'd', 'e', '\0'};
↓ Null

char clg [ ] = "RGUKT";
char a [10] = "Welcome"; } No need of '\0'.

char b [ ] = {'a', 'b', 'c', 'd', 'e', '\0'};

② Run time :-

→ To 'Read'/'input' characters at Runtime three ways are possible.

① 'scanf (' ')' → %c, %s.

② 'gets ( )' → line of characters '\n'.

③ 'getchar ( )' → only one character at a time.

→ For 'output/write'

① printf ( )

② puts ( ) → line write

③ putchar ( ) → print/write only one character.

* Note :-

→ #include <stdio.h> is enough for all the functions used in strings. [gets, getchar ( ), puts ( ), putchar ( )]

→ strings are constants, we can't modify once it is initialized.

* String Operations :-

① string concatenation.
② Finding Length
③ string replace
④ string find
⑤ convertion from up to down (lower case)
⑥ convertion from lower to upper case.

* Note :-

→ We cannot use '+' to combine strings.

→ We have some functions for strings in "string.h" header file.

→ strcat ( )
strcpy ( )
strcmp ( )
strlen ( )
strrev ( )
strstr ( )
strlwr ( )
strupr ( )
strchr ( )

→ some predefined functions which gives 'True' or 'false' as result.

isalpha ( )
isdigit ( )
isalnum ( )
isspace ( )
islwr ( )
isuppr ( )
tolower ( )
toupper ( )

→ For conversions, we need to use 'stdlib.h' header file.

Ex:- a123

atoi( )
atol( )
atof( )

* Example :- Reading & printing the string.
~~Finding length of the string.~~

A)
```
#include<stdio.h>
void main( )
{
  char name[10]={'a','b','c','d','e','\0'};
  char clg[10]="RGUKT";
  char str[20],ch;
  int l,i;
  printf("Enter the string\n");
  gets(str);
                (or)
  ch=getchar( );
  while(ch!='*')
  {
    str[i]=ch;
    i++;
    ch=getchar( );
  }
  str[i]='\0';
  printf("%s",name);
  printf("%s",clg);
  printf("%s",str);  (or) puts(str);
                (or)
  while(str[i]!='\0')
  { putchar(str[i]);
    i++;
```

* String concatenation : using strcat.

```
# include <string.h>
void main( )
{
  char str1[50], str2[20];
  printf("Enter str1 & str2");
  gets (str1);
  gets(str2);
  strcat (str1, str2);
  printf ("%s", str1); (or) puts (str1);
}
```

* String concatenation : without using strcat.

```
# include <stdio.h>
void main( )
{
  int i=0; j=0;
  gets(str1);
  gets (str2);
  while (str1[i] != '\0')
  {
     i++;
  }
  while (str2[j] != '\0')
  {
     str1[i]=str2[j];
     i++;
     j++;
  }
  str2[i]='\0';
  puts (str2);
}
```

* Palindrome :- [using string handling functions].

```
#include <stdio.h>
#include <string.h>
void main( )
{
 char str1[20], str2[20];
 printf ("Enter str1 ");
 gets (str1);
 strcpy (str2, str1);
 strrev (str1);
 if (strcmp (str1, str2) ==0)
   printf ("palindrome");
 else
   printf ("Not a palindrome");
}
```