Arrays

- **Definition of Array :**

  An array is a sequenced collection of related data items that share a common name. In other words, An array can be is used to represent a list of numbers or a list of names.

- An array is a derived data type.

- Examples: list of students, list of temperatures, list of employees, list of products etc.

- when we use array name salary to represent a set of salaries of a group of employee. We can refer to the individual salaries by writing a number called index or subscript in bracket after the array name

  for example : salary[10];

  It represents the salary of 1o employee.

- The complete set of values is refereed as array and individual values are called elements.

There are mainly three types of the array:

- One – Dimensional arrays
- Two – Dimensional arrays
- Multidimensional arrays

❑ One – Dimensional Arrays:

A list of items can be given one variable name using only one subscript and such a variable is called a single subscripted variable or a one dimensional array.

Syntax:

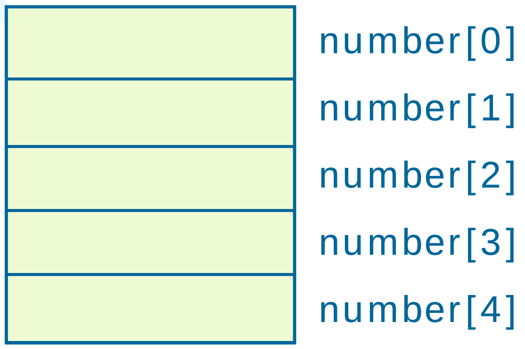
type variable-name[size];

The type specifies the data type of the element that will be contained in the array, such as int, float or char.

Ex: If we want to represent a set of five numbers say, (35,40,20,57,19) by an array variable number, then we may declare the variable number as follows.

int number[5];

And the computer reserves five storage location as shown below:



The value to the array element can be assigned as follow:

```
number[0] = 35 ;
number[1] = 40 ;
number[2] = 20 ;
number[3] = 57 ;
number[4] = 19 ;
```

This would case the array number to store the value as shown below:

35	number[0]
40	number[1]
20	number[2]
57	number[3]
19	number[4]

Same way,

```
float height[50];
```

Declares the height to be an array containing 50 real elements. Any subscript to 0 to 49 are valid.

- **Any references to the arrays outside the declared limits would not necessarily cause an error. Rather, it might result in unpredictable program results.**

- **The size should be either integer constant or symbolic constant.**

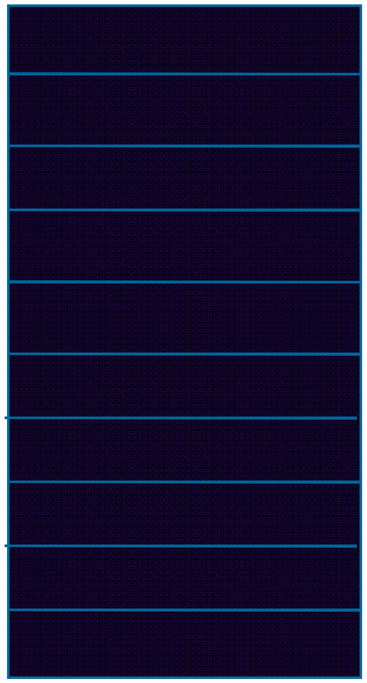
The C language treats character string simply as array of characters. The size in a character string represents the maximum number of characters that the string can hold.

```
char name[10] ;
```

Declare name as a character array variable that can hold a maximum of 10 characters. Suppose we read the following string constant into the string variable name.

“WELL DONE”

Each character of the string is treated as an element of the array name and is stored in the memory as follows:



- Character string terminates with an additional null character. Thus name[10] holds the null character ‘\0’. When declaring character array, we must allow one extra space for the null character.

INITIALIZATION OF ONE – DIMENSIONAL ARRAYS:

After an array is declared, its element must be initialized. Otherwise, they will contain “garbage”. An array can be initialized at either of the following stages:

At compile time
At run time

Compile Time Initialization:

Syntax:

Type array-name[size] = {list of values};

The value in the list are separated by commas.
Ex: int number[3] = {0,0,0} ;

Will declare the variable number as an array of size 3 and will assign zero to each element. If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically

```
float  total[5] = { 0.0 , 15.75 , -10};
```

Will initialize the first three elements to 0.0, 15.75 and -10.0 and the remaining two elements to zero

The size may be omitted. In such cases, the compiler allocate enough spaces for all initialized elements.

Ex: `int counter[] = {1,1,1,1}`

Counter array contain four element with initial value 1.

Character array may be initialized similar manner.

```
char name[] = {'R', 'A', 'M', '\0'};
```

Is equivalent to :

```
char name[] = "RAM";
```

If we have more than the declared size, the compiler will produce an error. That is the statement.

`int number[3] = {10, 20, 30, 40} ;` will not work. It is illegal in C.

Run Time Initialization :

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

Ex:

```
   for(i=0; i<100 ; i++)
   {
        if (i < 50 )
                sum[i] = 0.0;
        else
                sum[i] = 1.0;
  }
```

In the above case the first 50 elements are initialized to zero while the remaining elements are initialized to 1.0 are run time.

We can also use a read function such as scanf() to initialize an array.

```
Ex:     int  x[3];
        scanf (“%d  %d   %d”, &x[0], &x[1], &x[2] ) ;
```

Will initialize array elements with the values entered through the kwyboard.

```
/* program:-Write a C program to arrange the accepted numbers in
   ascending order and descending order: */

#include<stdio.h>
#include<conio.h>

void main()
{
    int i,j,ans,n,temp=0, no[20];
    clrscr();
    printf("Enter the size of the list:=");
    scanf("%d",&n);
          for(i=0;i<n;i++)
          {
                    printf("Enter the no:-");
                    scanf("%d",&no[i]);
          }
   printf("Enter the your choice ascending=1&descending=2:=");
   scanf("%d",&ans);
   for(i=0;i<n;i++)
   {
        for(j=i;j<n;j++)
        {
                if(ans==1)
                {
                        if(no[i]>no[j])
                        {
                           temp=no[i];
                           no[i]=no[j];
                           no[j]=temp;
                          }
                 }
                else

                if(no[i]<no[j])
                {
                        temp=no[i];
                        no[i]=no[j];
                        no[j]=temp;
                }
        }
  printf("%d",no[i]);
  printf("\n");
   }
 getch();
}
```

TWO – DIMENSIONAL ARRAYS:

If you want to store data into table or matrix form at that time you can use the two dimensional array. Consider the following data table, which shows the value of sales of three items by four sales man.

Ex:

	ITEM 1	**ITEM2**	**ITEM 3**
SALES MAN 1	**310**	**275**	**365**
SALES MAN 2	**210**	**190**	**325**
SALES MAN 3	**405**	**235**	**240**
SALES MAN 4	**260**	**300**	**380**

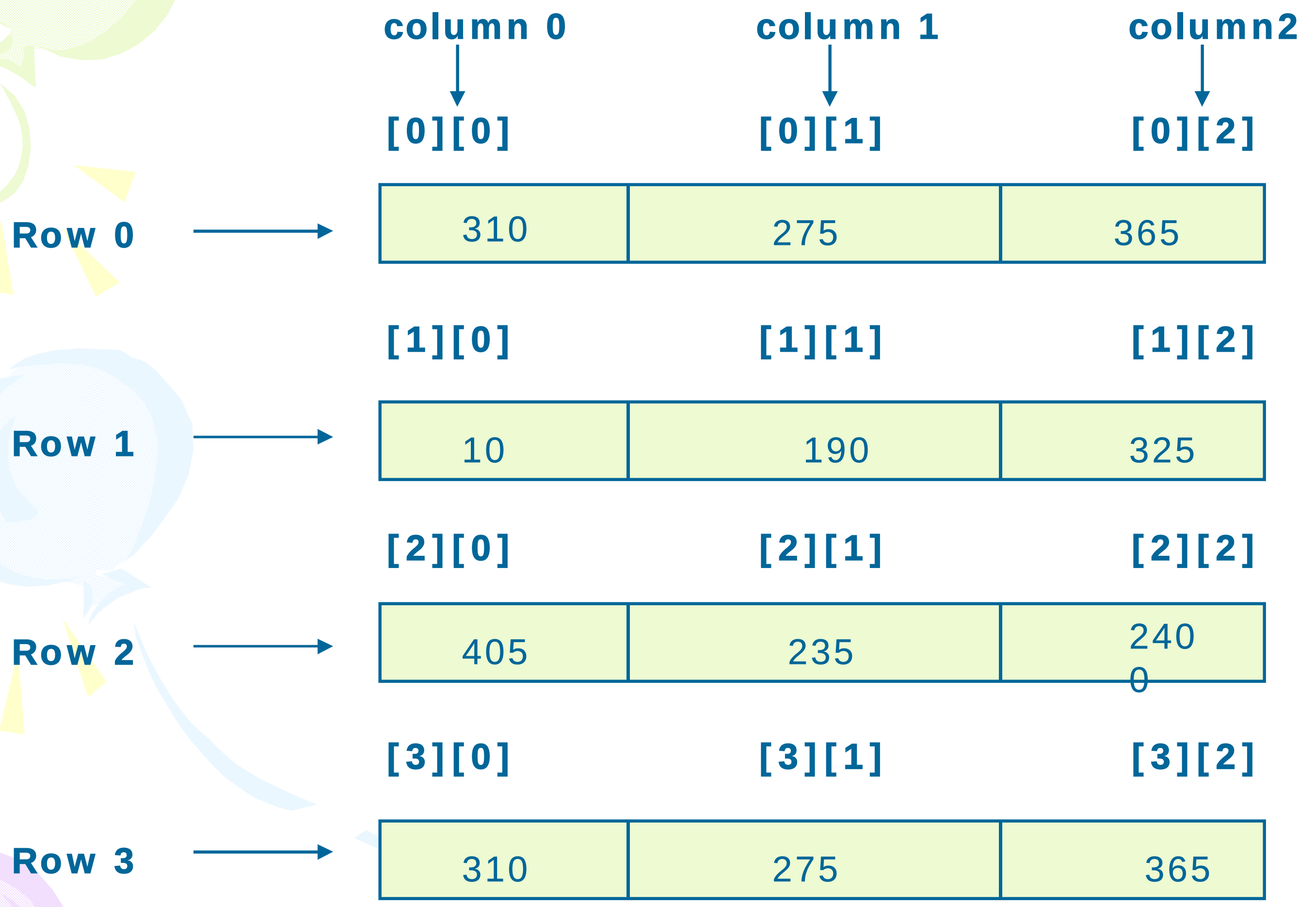
The table discussed above can be define as :

v[4][3] where 4 Rows and 3 columns.

Declaration of two dimensional array:

Syntax : **data_type   array-name[rowsize][colsize];**

Memory representation of the two dimensional array for v[4][3] .



INITIALIZING TWO DIMENSIONAL ARRAY:

Ex: `int table[2][3] = {0,0,0,1,1,1};`

Initialize elements of the first row to 0 and second row to 1. The initialization is done row by row.

The above statement can be equivalent written as :

```
int table[2][3] = {{0,0,0},{1,1,1}};
```

We can also initialize a two dimensional array in the form of a matrix as shown below:

```
int table[2][3] =     {
                                {0,0,0},
                                {1,1,1}
                      } ;
```

When the array is completely initialized with all values, we need not specify the size of the first dimension. That is, the statement is permitted.

```
int table[][3] =      {
                                {0,0,0} ,
                                {1,1,1)
                      };
```

If the value is missing in an initializes, they are automatically set to zero.

Ex:
```
int table[2][3] ={
                 {1,1} ,
                 {2}
              } ;
```

Will initialize the first two elements of the first row to one, the first element of the second row to two and all other elements to zero.

When all the elements are initialized to zero, the following short cut method may be used.

```
int m[3][5] = { 0, 0 ,0};
            or

int m[3][5] = { {0} , {0} ,{0} };
```

```
//program30 : addition of two matrix
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n,m,a[10][10],b[10][10],m1,n1;
    clrscr();

    printf("Enter No of Rows for first matrix:");
    scanf("%d",&n);

    printf("Enter No of columns for first Matrix:");
    scanf("%d",&m);

    printf("Enter No of Rows for second matrix:");
    scanf("%d",&n1);

    printf("Enter No of columns for second Matrix:");
    scanf("%d",&m1);
```

```
if(m!=m1 || n!=n1)
    printf("Sum not Possible.");

else
{
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        printf("Give value of a[%d][%d]:",i+1,j+1);
                        scanf("%d",&a[i][j]);
                }
        }

        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        printf("Give value of b[%d][%d]:",i+1,j+1);
                         scanf("%d",&b[i][j]);
                }
        }
printf("First Matrix:\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                         printf("%d ",a[i][j]);
                }
      printf("\n");

        }
printf("Second Matrix\n");
        for(i=0;i<n1;i++)
        {
                for(j=0;j<m1;j++)
                {
                        printf("%d ",b[i][j]);
                 }
      printf("\n");
        }
    printf("After Sum New Matrix\n");

        for(i=0;i<n1;i++)
        {
                for(j=0;j<m1;j++)
                {
                        printf("%d ",a[i][j]+b[i][j]);
                }
      printf("\n");
        }
   }
   getch();
}
```

❑ The scope, visibility and lifetime of variables :

✓ In C there are four types of storage classes :

1. Automatic variables
2. External variables
3. Static variables.
4. Register variables

❖ Automatic variables :

✓ Automatic variables declared inside a function in which they are to be utilized. so, automatic variables are private to that function.

✓ Automatic variables are referred as a local variable or internal variable.

✓ So you can declare this variable in more than one function without any confusion.

✓ Ex:
```
main()
   {
     int number ;
   }
```

You can also declared explicitly with <u>auto</u> keyword

Ex:
```
main()
    { auto int number ;
    }
```

- **External variable :**
- ✓ variables that are both active and alive throughout the entire program is known as external variables. It is also called global variable.
- ✓ unlike local variables global variables can be accessed by any function in the program.
- ✓ Ex:

```
int number;
main()
{
  number=10;
}
fun1()
{
  number=10;
}
fun2()
{
  number=10;
}
```

❖ Static variables:

- ✓ Static variables declared using static keyword.
  Ex: static int number;

- ✓ Static variables are initialized only once during the execution of the program. Static variables are generally used to retain values between function calls.

  Ex:

```
        void stat(void);
        void main()
      {
                int i;
                for(i=1;i<=3;i++)
                   stat();
      }
    void stat(void)
    {
              static int x=0;
              x++ ;
              printf("%d\n",x);
      }
```

- ✓ Output:

```
1
2
3
```

❖ Register Variables :

- ✓ Register Variables are generally used to store variable in machine’s register. so you can access that variable very fast than store in the memory.
- ✓ you can create your register variable using register keyword.
  Ex: register int x;
- ✓ register variables are generally used in looping.

❖ Static Functions :

- ✓ A static function in C is a function that has a scope that is limited to its object file. This means that the static function is only visible in its object file. A function can be declared as static function by placing the static keyword before the function name.
- ✓ Example:

```
#include <stdio.h>
static void staticFunc(void)
{
    printf("Inside the static function staticFunc() ");
}
int main()
{
   staticFunc();
   return 0;
}
```

- ✓ Output:

```
Inside the static function staticFunc()
```

POINTERS

❑ What is a pointer ?

- A Pointer is a derived Data Type.
- A pointer is nothing but a variable that contains the address which is a location of the another variable in memory.

Benefits of using the pointer :

1. Pointer are more efficient in handling array and data tables.
2. Pointer reduces the length and complexity of the program.
3. The use of pointer array to character string results in saving of data storage space in memory.
4. Pointer can be used to return multiple values from functions via functions arguments.
5. Pointers permit references to functions.
6. Pointers allow C to support dynamic memory management.
7. Pointers provide an efficient tool for manipulating Dynamic Data Structure such as Linked List , Queue, Stack and Tree.
8. They increase the execution speed.

❑ Accessing The Address of a Variable:

- ✓ The actual location of a variable in the memory is system dependent and therefore the address of a variable is not known to us immediately.
- ✓ We can access the address of a variable using & Operator.

```
    int quantity=179;
    int *p;
    p=&quantity;
```

quantity	variable
179	value
5000	Address

- ✓ p=&quantity; will assign the address the 5000 to the variable p.
- ✓ The & Operator can be used only with simple variable or array element.

❑ Declaring and initializing pointers :

- ✓ syntax : data_type *pt_name ;

- ✓ this tells the compiler three things about the variable pt_name.

  1. * tells the compiler that the variable pt_name is a pointer variable.
  2. pt_name needs a memory location.
  3. pt_name points to a variable of type data_type.

- ✓ Ex: int quantity=179 ;

```
int *p;
p=&quantity;
```

quantity	variable
179	value
5000	Address

- ✓ p=&quantity; using this statement you can store the address of the variable quantity to the pointer variable p.

- ✓ Note : you can know the address of variable using %u format specification.

- ✓ You can't assign an absolute address to a pointer variable directly.
  Ex: p = 5000; It is not possible

- **Accessing variables using pointers.**

```
void main()
{
        int x,y, *ptr;
        x=10;
        ptr=&x;
        y = *ptr;
        printf("Value of x is %d\n\n",x);
        printf("%d is stored at address %u\n", x, &x);
        printf("%d is stored at address %u\n", *&x, &x);
        printf("%d is stored at address %u\n", *ptr, ptr);
        printf("%d is stored at address %u\n",ptr,&ptr);
        printf("%d is stored at address %u\n", y, &y);

        *ptr=25;
        printf("\n Now x = %d\n",x)
        getch();
}
```

Output:

```
Value of X is 10
10 is stored at address 4104
10 is stored at address 4104
10 is stored at address 4104
4104 is stored at address 4106
10 is stored at address 4108

Now x = 25
```

❑ Pointer increments and scale factor :

- ✓ p1++ will cause the pointer p1 points to the next value of its type. Ex :- if p1 is an integer pointer with the initial value say 2800,then after the operations p1=p1+1,the value of p1 will be 2802,not 2801

- ✓ when we increment pointer its value is increased by the length of the data type that it points to. This length is called scale factor.

❑ Pointer and Arrays :-

- ✓ When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in memory location.

- ✓ Base address is the location of the first element of the array

  int x[5] = {1,2,3,4,5};

- ✓ Suppose the base address of x is 1000 and assuming that Each integer requires 2 bytes, the five elements stored as Follows :

Element	x[0]	x[1]	x[2]	x[3]	x[4]
Value	1	2	3	4	5
Address	1000	1002	1004	1006	1008

So, base address is,
x = &x[0] = 1000

- If we declare p as integer pointer, then we can make the pointer p to point to the array x by the following statements.

  ```
  p = x  or   p = &x[0]
  ```

  Now we can access every element of x using p++ to move from one element to another.

  ```
  p      =   &x[0] (=1000)
  p+1    =   &x[1] (=1002)
  p+2    =   &x[2] (=1004)
  p+3    =   &x[3] (=1006)
  p+4    =   &x[4] (=1008)
  ```

```
/***PRGRAMM FOR SUM OF N NUMBER ELEMENT *****/
#include<stdio.h>
main()
{
    int x[10],i,*p,n,sum=0;
    printf("enter the N:");
    scanf("%d",&n);
    printf("enter the the data:\n");
    for(i=0;i<n;i++)
    {
    scanf("%d",&x[i]);
    }
    p = x;
    for(i=0;i<n;i++)
    {
    printf("\n%d",*p);
    sum=sum + *p;
    p++;
    }
    printf("\nsum=%d",sum);
    getch();
}
```

❑ Pointer and character string:

- ✓ Strings are treated like character arrays and therefore, they are declared and initialized as follows:
- ✓ `char str[5] = “good”;`
- ✓ C supports an alternative method to create string using pointer variable of type char.

```
char *str = “good”;
```

- ✓ The pointer str now points to the first character of the string “good”.

❑ Array of Pointers:

- ✓ One important use of pointer is handling of a table of string. following is the array of string.

```
char name[3][25];
```

- ✓ This say that table containing three names ,each with a maximum length of 25 characters. So total storage requirement for the name table are 75 bytes

- ✓ We know that individual string may be unequal length. therefore instead of making each row a fixed number of characters, we can make it a pointer to a string of varying length.

```
char *name[3] = {
                        "New Zealand",
                            "Australia",
                            "India"
                        }
```

- ✓ Where name to be an array of three pointers to a character, each pointer points to a particular name.

```
name[0]="New Zealand",
name[1]="Australia",
name[2]="India"
```

- ✓ This declaration allocates only 28 bytes.

- ✓ The character arrays with the rows of varying length are called ragged array.

❑ Pointers As Function Arguments:

- ✓ When we pass addresses to a function, the parameter receiving the addresses should be pointers.
- ✓ The process of calling a function using pointer to pass the address of variable is known as call by reference. The function which is called by ‘reference’ can change the value of the variable used in the call.
- ✓ The process of passing the actual value of variable is known as call by value.
- ✓ Ex:

```
main()
{
        int x=20;;
        change(&x);
        printf("%d\n",x);
}
change(int *p)
{
        *p = *p + 10;
}
```

- ✓ Thus call by reference provides the mechanism by which the function can change the stored value.

Ex: / * * *PROGRAMM FOR INCHANGING THE VALUE* * * * * /

```
#include<stdio.h>
main()
{
int a,b;
printf("enter the number");
scanf("%d %d",&a,&b);
printf("\t\t\nA=%d B=%d\n\n",a,b);
swap(&a,&b);
printf("\t\t\nA=%d B=%d",a,b);
getch();
}
swap(int *p ,int *q)
{
int t;
t=*p;
*p=*q;
*q=t;
}
```

Call by Value :

If data is passed by value, the data is copied from the variable used in for example main() to a variable used by the function. So if the data passed is modified inside the function, the value is only changed in the variable used inside the function.

Call by value example:

```
#include <stdio.h>

void call_by_value(int x)
{
    printf("Inside call_by_value x = %d before adding 10.\n", x);
    x += 10;
    printf("Inside call_by_value x = %d after adding 10.\n", x);
}
```

```
int main()
{
    int a=10;
    printf("a = %d before function call_by_value.\n", a);
    call_by_value(a);
    printf("a = %d after function call_by_value.\n", a);
    return 0;
}
```

The output of this call by value code example will look like this:

```
a = 10 before function call_by_value.
Inside call_by_value x = 10 before adding 10.
Inside call_by_value x = 20 after adding 10.
a = 10 after function call_by_value.
```

- **Call by Reference :**

  If data is passed by reference, a pointer to the data is copied instead of the actual variable as is done in a call by value. Because a pointer is copied, if the value at that pointers address is changed in the function, the value is also changed in main().

Call by reference example:

```
#include <stdio.h>

void call_by_reference(int *y)
{
    printf("Inside call_by_reference y = %d before adding 10.\n",
    *y);
   (*y) += 10;
   printf("Inside call_by_reference y = %d after adding 10.\n", *y);
}
```

```
int main()
 {
    int b=10;
    printf("b = %d before function call_by_reference.\n", b);
    call_by_reference(&b);
    printf("b = %d after function call_by_reference.\n", b);
return 0;
}
```

The output of this call by reference code example will look like this:

```
b = 10 before function call_by_reference.
Inside call_by_reference y = 10 before adding 10.
Inside call_by_reference y = 20 after adding 10.
b = 20 after function call_by_reference.
```