C – Array

C Array is a collection of variables belonging to the same data type. You can store group of data of same data type in an array.
Def^{n}:- ***"Array is a collection of homogenous (similar) elements in which values of same data type can be store into a single variable."***

- Simple variable can store only one value. If you want to store n-no of value, there is need to declare n-no of variables.
- It increases length of the program and decreases program efficiency.
- All these n-variables store in random memory locations, therefore it requires more time for execution.
- Array might be belonging to any of the data types(int, float, char etc)
- Array size must be a constant value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

EXAMPLE FOR C ARRAYS:

```
int a[5];      // integer array of 5 elements
char b[5];    // character array   i.e. string with 5 characters
(this char array is also called as string.)
```

TYPES OF C ARRAYS:

There are 2 types of C arrays. They are,

1. One dimensional array ex. int a[3]; float b[7]; etc
2. Multi dimensional array
   - Two dimensional array ex. int a[2][3];
   - Three dimensional array ex. int a[2][3][4];
   - four dimensional array ex. int a[2][3][4][2];

1. ONE DIMENSIONAL ARRAY IN C:

When we use single subscript enclosd within square bracket, at the time of declaration, then such an array variable is called as one-dimensional array.

Array declaration syntax:

Before using array variable there is need to declare array avariable:-

data-type array_name [array_size/subscript];

Example int a [5];
It creates 5 integer variables as:- a[0], a[1], a[2], a[3], a[4]
Here index value always starts from 0.

Initialization of array:-

Initialization means assigning constant values to array elements.

Array initialization syntax:

data_type arr_name [arr_size]={value1, value2, value3,….};

Integer array example:

```
int  a[5];
int  a[5]={5,6,7,8,9};
```

It means a[0]=5, a[1]=6, a[2]=7, a[3]=8, a[4]=9

Character array example:

```
char c[3];
char c[3]={'H','a','i'};
        OR
char str[0] = 'H';
char str[1] = 'a';
char str[2] = 'i;
```

Array accessing syntax:

arr_name[index];

```
ex.   str[0]; /*H is accessed*/
      str[1]; /*a is accessed*/
      str[2]; /*i is accessed*/
```

EXAMPLE PROGRAM FOR ONE DIMENSIONAL (1-D)ARRAY IN C:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i;                          // declaring and Initializing array in C
   int arr[5] = {10,20,30,40,50};

   for (i=0;i<5;i++)
   {
         printf("value of array ",  arr[i]);
    }

}
```

OUTPUT:

```
value of array 10  20  30  40  50
```

```
      /* Above array can be initialized as below also
   arr[0] = 10;      arr[1] = 20;      arr[2] = 30;
    arr[3] = 40;     arr[4] = 50; */
```

2. TWO DIMENSIONAL ARRAY IN C:

- Two dimensional array is nothing but ***array of array.***
- It is also called as ***matrix***

syntax :

data_type array_name[num_of_rows][num_of_column];

Array declaration syntax:

data_type arr_name [rows][columns];

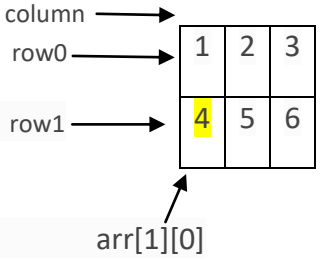
example:

int arr[2][3];

Array initialization :

int arr[2][3] = {1,2, 3, 4,5,6};

arr [0] [0] = 1;
arr [0]]1] = 2;
arr [0][2] = 3;
arr [1] [0] = 4;
arr [1] [1] = 5;
arr [1] [2] = 6;



EXAMPLE PROGRAM FOR TWO DIMENSIONAL ARRAY IN C:

```
#include<stdio.h>
 #include<conio.h>
void main()
{
  int  i, j;
  int arr[2][3] = {10,20,30,40,50,60};

  for (i=0;i<2;i++)              // i= for row no
  {
    for (j=0;j<3;j++)          // j for column no
    {
            printf("value of array",arr[i][j]);
    }
  }
}
```

OUTPUT:

```
value of array 10  20 30  40  50  60
```

```
/* Above array can be initialized as below also
    arr[0][0] = 10;      arr[0][1] = 20;    arr[0][1] = 30;
    arr[1][0] = 40;      arr[1][1] = 40;    arr[1][2] = 50;     */
```

C – Pointer

- Computer memory is sequential collection of storage cells as shown in figure, each cell known as ***byte.***
- Each cell is identified by using address, associated with it.
- The cell address starts from zero and last address depends on memory size.
- A computer system having 64K memory will have its last address as 65,535.

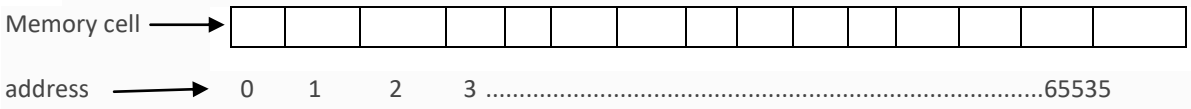
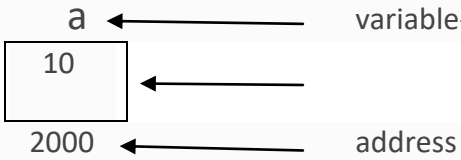


fig. Memory organisation

- When we declare a variable, a memory is allocated for it .
- Consider a following example

  int a = 10;

  suppose variable a stored at memory location 2000, then its representation as below:-



- ***"Pointers in C language is a variable that stores/points the address of another variable."***
- ***" A Pointer in C is used to allocate memory dynamically i.e. at run time. "***
- A pointer is derived data type in C.
- The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- ***&*** symbol is used to get the ***address*** of the variable.
- ******* symbol is used to get the ***value*** of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.

DECLARATION OF POINTER:-

In C variable must want to be declare before using it.

Syntax :

data_type *var_name;

Example :
```
int *p1;
float *p2;
char *p3;
```

Where, * is used to denote that "p" is pointer variable and not a normal variable.
p1 points to int data type.
P2 points to float data type.
P3 points to char data type.

INITIALIZATION OF POINTER:-

Once pointer is declared, it must be initialized as below:-

Syntax :

data_type *var_name = & variable-name;

Example :
```
int  a;
float  b;
char  c;
int *p1 = &a;
float *p2 = &b;
char *p3 = &c;
```

Where, * is used to denote that "p" is pointer variable and not a normal variable.
p1 stores the address of variable a.
P2 stores the address of variable b.
P3 stores the address of variable c.

EXAMPLE PROGRAM FOR POINTERS IN C:-

[Suppose q is stored at memory location 1000]

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int *ptr, q;
  q = 50;
   ptr = &q;                                  /* address of q is assigned to ptr */
   printf("value= %u", *ptr);                        /* display q's value using ptr variable */
  printf("\n  address= %d", ptr);              /*display address stored in pointer "ptr"*/
   getch();
}
```

OUTPUT*:*
value ***= 50***
address =1000

C - Pointer arithmetic

A pointer in c is an address, which is a numeric value.

Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers:

++, --, +, –

To understand pointer arithmetic,

let us consider that **ptr** is an integer pointer which points to the address 1000.

let us perform the following arithmetic operation on the pointer –

```
int   *ptr, q;                  (declaration of pointer)
ptr = &q;                       (stores address of q into ptr variable i.e ptr=1000)
ptr++;                          (increment the pointer variable i.e ptr=1002)
ptr--;                          (increment the pointer variable i.e ptr=1000)
ptr= ptr+2;                     (increment the pointer variable i.e ptr=1004)
ptr= ptr-2;                     (increment the pointer variable i.e ptr=1002)
```

After the above operation(ptr++), the **ptr** will point to the location 1002 because each time ptr is incremented,

it will point to the next integer location which is 2 bytes next to the current location.

This operation will move the pointer to the next memory location without impacting the actual value at the memory location.

If **ptr** points to a character data type(1 byte) whose address is 1000, then the above operation will point to the location 1001 because the next character will be available at 1001.

Examples for Pointer Arithmetic

Suppose address for ip=1000, dp=2000 and ch=3000

Then output of following program as below:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i = 12, *ip = &i;
    double d = 2.3, *dp = &d;
    char ch = 'a', *cp = &ch;

    printf("Value of ip = %d ", ip);
    printf("Value of dp = %d ", dp);
    printf("Value of cp = %d ", cp);

    printf("Value of ip + 1 = %d ", ip + 1);
    printf("Value of dp + 1 = %d ", dp + 1);
    printf("Value of cp + 1 = %d ", cp + 1);

    printf("Value of ip + 2 = %d ", ip + 2);
    printf("Value of dp + 2 = %d ", dp + 2);
```

```
    printf("Value of cp + 2 = %d ", cp + 2);

   getch();
}
```

Output:

```
Value of ip = 1000
Value of dp = 2000
Value of cp = 3000

Value of ip + 1 = 1002
Value of dp + 1 = 2004
Value of cp + 1 = 3001

Value of ip + 2 = 1004
Value of dp + 2 = 2008
Value of cp + 2 = 3002
```

Array of Pointers in C

Just like we can declare an array of int, float or char etc, we can also declare an array of pointers, here is the syntax to do the same.

Syntax: datatype *array_name[size];

Let's take an example:

int *ptr[5];

Here ptr is an array of 5 integer pointers. It means that this array can hold the address of 5 integer variables, or in other words, you can assign 5 pointer variables of type pointer to int to the elements of this array.

The following program demonstrates how to use an array of pointers.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
   int *ptr[3];
   int a = 10, b = 20, c = 50, i;

   ptr[0] = &a;
   ptr[1] = &b;
   ptr[2] = &c;

   for(i = 0; i < 3; i++)
   {
      printf("Address = %d ", ptr[i]);
      printf("Value = %d", *ptr[i]);
   }
   getch( );
```

}

Output:

Address = 1040 Value = 10
Address = 1038 Value = 20
Address = 1036 Value = 50

How it works ?

Notice how we are assigning the addresses of a, b and c. In line 9, we are assigning the address of variable a to the 0th element of the of the array. Similarly, the address of b and c is assigned to 1st and 2nd element respectively.

ptr[i] gives the address of ith element of the array. So ptr[0] returns address of variable a, ptr[1] returns address of b and so on. To get the value at address use indirection operator (*).

***ptr[i]**

So *ptr[0] gives value at address[0], Similarly *ptr[1] gives the value at address ptr[1] and so on.

/*Program to print square of "*" using 2-d array */

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int A[4][4],i,j;

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
        printf(" * ");
        }
      printf("\n");
    }
    getch();
}
```

Output:-

```
* * * *
* * * *
* * * *
* * * *
```

/*Program to find addition of two numbers using pointers*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
                int a=10, b=20, c;
               int  *p1 = &a;
               int  *p2 = &b;
               c = *p1 + *p2;
               printf("addition = %d"c);
                getch( );
}
```