# Fundamentals of Array and Pointers

Dr. Rahul Das Gupta

# Array:

**An array in C is a collective name given to a group of similar variables. Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type.**

# Few Important Points on Array to Remember:

1. **Name of the Array is the 'Address of the Starting Element of the Array' or the 'Base Address'.**

| | |
|---|---|
| **int *:** | **Pointer to an integer or address of an integer.** |
| **float *:** | **Pointer to a float or address of a float.** |
| **char *:** | **Pointer to a character or address of a character.** |

```
int A[10];
int *p;
p=A;
```

**p+7=??**

| | |
|---|---|
| p = A=1074 | A[0] |
| A+1=1076 | A[1] |
| A+2=1078 | A[2] |
| A+3=1080 | A[3] |
| A+4=1082 | A[4] |
| A+5=1084 | A[5] |
| A+6=1086 | A[6] |
| A+7=1088 | A[7] |
| A+8=1090 | A[8] |
| A+9=1092 | A[9] |

**Wrong Answer**

**p+7 = 1074 + 7 = 1081**

**Correct Answer**

**p+7 = 1074 + 7*sizeof(int) =1074 +7*2 = 1088**

**Hence, p+7 = A+7 = 1074+7*sizeof(int) = 1074+7*2 = 1088**

2. **Data Type of the Name of the Array is a Constant Pointer.**

**int A[10];**
**int x,y,z;**
**int * p;**
**Data type of A: const int * (Pointer to an integer or Address of an integer, which is fixed.)**

**p=&x;**
**p=&y;**
**p=&z;**
**p=A;**
**p=NULL;**

**Hence, p is not a Constant Pointer because you can assign any address at p.**

**A=&x;** ✗
**A=&y;** ✗
**A=&z;** ✗
**A=NULL;** ✗

**None of these are permitted.**

**But, all these above assignments for A are not permitted as A is a Constant Pointer.**

3. **Two Different Meanings of the Quantity Inside '[ ]':**
   i. **Case 1 (Declarative Statement): Quantity Inside '[ ]' is the Array Size.**

**int A[100]; /*It is a Declarative Statement. */**

**Array size = No. elements in the array = 100**
**Elements are: A[0], A[1], A[2],…, A[99]**
**Positional index varies from 0 to 99 = Array Size - 1.**

ii. Case 2 (Non-Declarative Statement): Quantity Inside '[ ]' is the Position Index.

/*It is not a Declarative Statement.*/
A[20]=-8; /* (20+1) that is 21st element of the array is -8. */

Position index can not be negative.

# Pointer:

## The pointer in C, is a variable that stores address of another variable.

Case-1: * is associated to *data type* in case of any *declaration statement*.

int *p,*q, *r; /* Declaration Statement*/

Data Type of p: int *

Data Type of q: int *

Data Type of r: int *

int *: Pointer to an Integer or Address of an Integer.

Case-2: * is a '*content of*' operator in case of any *non-declaration statement*. Here * is associated with variable.

int *p , n=3;

```
p=&n;
*p=5; /* Non-Declaration Statement*/

printf("\n n=%d ", n);
```

**Output:**

n=5

**&: Address of operator**

***: Content of operator**

***p: Content at the address p**

```
int A[10];
int *p;
p=A;
```

p+7=??

| | |
|---|---|
| p = A=1074 | A[0] |
| A+1=1076 | A[1] |
| A+2=1078 | A[2] |
| A+3=1080 | A[3] |
| A+4=1082 | A[4] |
| A+5=1084 | A[5] |
| A+6=1086 | A[6] |
| A+7=1088 | A[7] |
| A+8=1090 | A[8] |
| A+9=1092 | A[9] |

Wrong Answer

p+7 = 1074 + 7 = 1081

Correct Answer

p+7 = 1074 + 7*sizeof(int) =1074 +7*2 = 1088

```
int A[10],*p;
/* Data Type of p: int*
   Data Type of A: const int*
```

[ A= Base Address of the Array (Address of the first element of the array) it is a constant pointer to integer.]
*/

/*Address of the 1st element is assigned to p. */

p=A; /* Here, *p = A[0] */

/*Address of the (i+1)th element is assigned to p. */

p=A + i; /* Here, *p= A[i] */

*(p + i) = p[i] = *(i + p) = i[p]
p+i=&(p[i])=&(i[p])=i+p

# Few Important Points on Pointer to Remember:

(i) Two different meaning of *.

Case-1: * is associated to *data type* in case of any *declaration statement*.

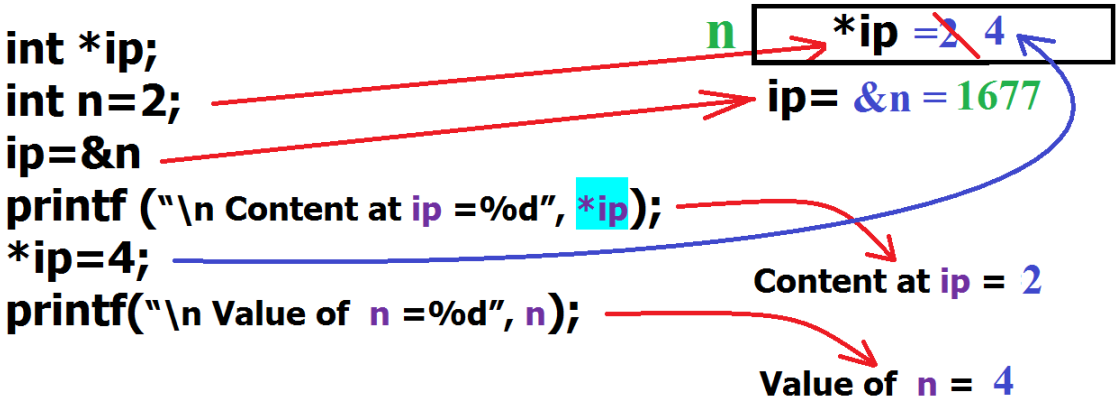
```
int *ip;
/*ip is a pointer to an integer or address of an integer.*/
float *fp;
/*fp is a pointer to a float or address of a float.*/
char *cp;
```

/*cp is a pointer to a character or address of a character.*/

| Variable | Data Type |
|---|---|
| ip | int *(pointer to an integer or address of an integer) |
| fp | float *(pointer to a float or address of a float) |
| cp | char * (pointer to a character or address of a character) |

**Case-2: * is a *'content of'* operator in case of any *non-declaration statement*. Here * is associated with variable.**

```
int *ip;
int n=2;
ip=&n; /* Here, & is a 'address of operator'.*/
printf("\n Content at ip =%d", *ip);
*ip=4;
/* Meaning of *ip : Content at the address ip
Where ip is a pointer to an integer or address of an integer. */
printf("\n Value of  n =%d", n);
```




**Output:**

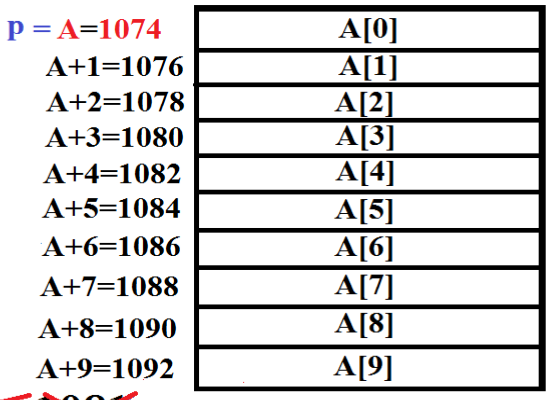
```
Content at ip = 2
Value of n = 4
```

**(ii) Actual value of (p+i) = p + i*sizeof(*data_type*)**

**Actual value of (p-i) = p - i*sizeof(*data_type*)**

```
int A[10];
int *p;
p=A;
```

**p+7=??**




**Wrong Answer**

~~p+7 = 1074 + 7 = 1081~~

**Correct Answer**

**p+7 = 1074 + 7*sizeof(int) =1074 +7*2 = 1088**

**(iii) Actual value of (p-q) = (p–q)/sizeof(*data_type*)**

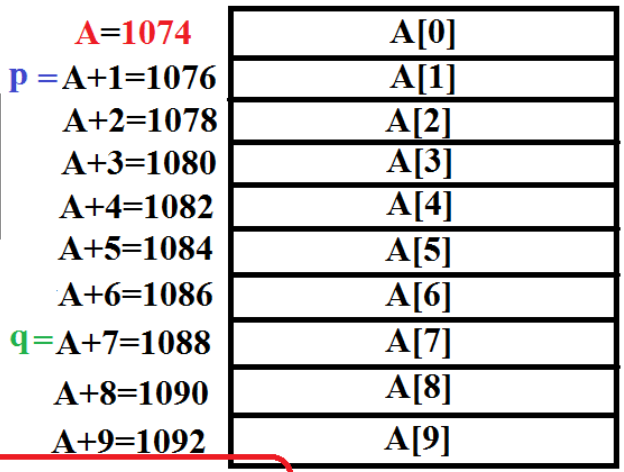
**Example:**

```
int A[10];
int *p,*q;
p=&A[1];
q=&A[7];
printf("\n %d",p-q);
```

**Output:**

**Wrong Answer**

~~p-q = 1076-1088 = -12~~



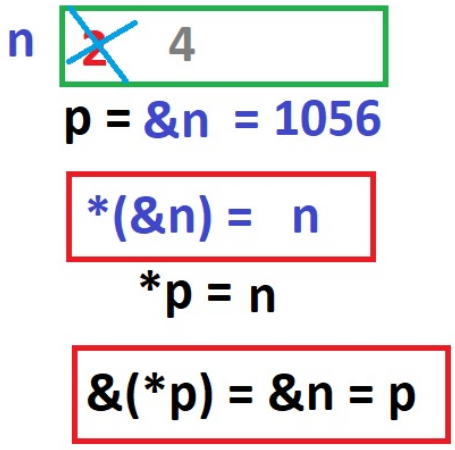

**Correct Answer**

$$p-q = \frac{(1076-1088)}{sizeof(int)} = \frac{(1076-1088)}{2} = -6$$

**(iv) p[i]=*(p+i)=*(i+p)=i[p]**

**(v)** **p+i=&(p[i])=&(i[p])=i+p**
**(vi)** ***(&x)=x**
**(vii)** **&(*p)=p**

```
int  n =2 , *p;
 p = &n;
*p = 4;
printf ("\n n = %d", n);
```




```
Output:
 n = 4
```

1. **To change the value of a variable through a function, the address of the variable must be passed as an input argument to the function.**

**Example:**

```
#include<stdio.h>

void swapping1 (int, int);
void swapping2(int *,int *);

void main ( )
{
  int  m=2, n=3;

   swaping1 (m, n);

   /* Swapping  will not  be effective.*/

   printf ("\n m = %d, n = %d ",  m, n);
```

```
   swapping2 (&m, &n);

  /* Actual swapping.  */

   printf ("\n m = %d, n = %d ",  m, n);
}
```

```
void swapping1 (int x, int y)
{
   int  t=x;
   x=y;
   y=t;
}
```

```
void swapping2(int *p, int *q)
{
    int  t=*p;
    *p=*q;
    *q=t;
}
```

## Output

m=2, n=3
m=3, n=2