

## Ch. 7 → arrays

An array is a collection of similar elements.

One variable  $\Rightarrow$  capable of storing multiple values.

$a \leftrightarrow a \leftrightarrow a[0]$

$a[3] \leftrightarrow *a + 3$

If  $a$  is a array  $a++$ ,  $a--$  is not allowed.

Date \_\_\_\_\_

Page \_\_\_\_\_

## Syntax

The syntax of declaration declaring an array looks like this.

int marks [90]  $\Rightarrow$  Integer Array

char name [20]  $\Rightarrow$  Character Array or String

float percentile [90]  $\Rightarrow$  Float array

The values can now be assigned to mark array like this.

marks [0] = 32 ;

marks [1] = 12 ;

Note: It is very important to note that the array index starts with 0 (0th element).

Marks $\rightarrow$	17	6	8	9	11	12	.....	13	14
	0	1	2	3	4	5	.....	88	89

Total 90 elements

## Accessing elements

Elements of an array can be accessed using

`scanf("%d", &marks[0]);`  $\Rightarrow$  Input 1<sup>st</sup> value

`printf("%d", marks[0]);`  $\Rightarrow$  Output 1<sup>st</sup> value

of the array

~~Quiz = Write a program to accept marks of 4 students in an array & print them to the screen~~

code # include <stdio.h>

int main () {

int marks [4]; // allocate space for 4 integers

printf ("Enter the value of marks for student 1: ");

scanf ("%d", &marks [0]);

printf ("Enter the value of marks for student 2: ");

scanf ("%d", &marks [1]);

printf ("Enter the value of marks for student 3: ");

scanf ("%d", &marks [2]);

printf ("Enter the value of marks for student 4: ");

scanf ("%d", &marks [3]);

printf ("You have entered %d %d %d &  
%d", marks [0], marks [1],  
marks [2], marks [3]);

return 0;

Output

Enter the value of marks for student 1 : 90 51  
2 : 90 52  
3 : 91 D3  
4 : 10 D4

You have entered 40, 90, 1, 10

Write a program to accept marks of 5 students & in an array & print them to the screen.

```
#include<stdio.h>
```

```
ent main()
```

```
ent marks[5];
```

```
for (ent i=0; i<5; i++)
```

```
{
```

printf("Enter the value of marks for student %d: ", i+1);

```
scanf("%d", &marks[i]);
```

```
for (ent i=0; i<5; i++)
```

```
{
```

printf("The value of marks for student %d is: %d\n", i+1, marks[i]);

```
return 0;
```

```
}
```

Output The value of marks for student 1 is: 8 DIS

11 2 : 6 6

11 3 : 8 8

11 4 : 10 10

11 5 : 9 9

INPUT:

8756 :

200 :

900 :

Generalization of an array

There are many other ways in which an array can be generalized.

`int cgpa[3] = { 9, 8, 8 };`  $\Rightarrow$  array can be

generalized while declaration

`float marks[] = { 33, 40 };`

arrays in memory

Consider this array

arrays in Memory

Consider this array:

`int arr[3] = { 1, 2, 3 };`  $\Rightarrow$  1 integer = 4 bytes

They will reserve  $4 \times 3 = 12$  bytes in memory 4 bytes for each integer.

1	2	3
G2802	G2306	G2310

$\Rightarrow$  arr in memory

## Pointer Arithmetic

A pointer can be incremented to point to the next memory location of that type.

Consider this example

`int p = 32;`

1
32

add. = 87994

`int *a = & p;`  $\Rightarrow$   $a = 87994$

`a += 1;`  $\Rightarrow$  Now  $a = 87998$

`char a = 'A';`

`char *b = &a;  $\Rightarrow b = 87994$`

`b++;`

$\Rightarrow$  Now `b = 87995`

`float p = 1.75;`

`float *a = &p;  $\Rightarrow$  Add. of p or a = 87994`

`a++;`

$\Rightarrow$  Now `a = 87998`

Following operations can be performed on a pointer:

- 1) Addition of a number to a pointer
- 2) Subtraction of a number from a pointer
- 3) Subtraction of one pointer from another
- 4) Comparison of two pointer variables

Accessing arrays using pointers

Consider the array

	7	9	2	8	1	5	1	8
Index	0	1	2	3	4	5	6	7

`ptr`

If `ptr` points to index 0, `ptr + 1` will point to index 1 & so on. It will point to next memory location.

This way we can have an integer pointer pointing to 1<sup>st</sup> element of the array like this:

`int *ptr = &arr[0];` for simply `arr`  
`ptr++;`

`*ptr` will have 9 as its value.

Code #include <stdio.h>

```
int main() {
    int marks[4];
    int *ptr;
    ptr = &marks[0];
```

```
for (int i=0; i<4; i++) {
    printf("Enter the value of marks for
           student %d : ", i+1);
    scanf("%d", ptr);
    ptr++;
}
```

```
for (int i=0; i<4; i++) {
    printf("The value of marks for
           student %d is %d \n", i+1, marks[i]);
}
return 0;
```

Output: Enter the value of marks for student 1:

82

88

11

2:

22

88

11

3:

23

88

11

4:

24

The value of marks for student 1 is 21  
 1 2 93 22

11 3 96 23

11 4 93 24

Code: #include <stdio.h>

```
void printArray(int *ptr, int n) {
    for (int i = 0; i < n; i++) {
        printf("The value of element %d is %d\n", i + 1, *(ptr + i));
    }
}
```

```
int main() {
    int arr[] = {1, 2, 3, 4, 5, 3, 6, 4, 5, 2, 3};
    printArray(arr, 7);
    return 0;
}
```

Output

The value of element 1 is 1

1	2	3	4	5	6	7
11	11	11	11	11	11	11

Passing arrays to functions

Arrays can be passed to the functions like this

printArray(arr, n);  $\Rightarrow$  function call

void printArray(Ent\* arr, Ent n);  $\Rightarrow$

function prototype  
or

void printArray(Ent arr[], Ent n);

## Multi-dimensional Arrays

An array can be of 2 dimensions/  
3 dimensions/n dimensions.

A 2 dimensional array can be defined  
as:

Ent arr[3][2] = {{1, 4}, {7, 9}, {11, 22}};

We can access the elements of this array  
as arr[0][0], arr[0][1] & so on.....

Value 1      Value 4

Ent arr[3][2] = {{1, 4},

{7, 9},

{11, 22}};

We can access the other elements of the  
array as arr[0][0], arr[0][1] & so on

Value 1      Value 4

```

Code: #include <stdio.h>

Ent main() {
    Ent n = students = 3;
    Ent n = subjects = 5;

    Ent marks [3][5];
    for (Ent i=0; i<n; student++) {
        for (Ent j=0; j<n; subjects++) {
            printf("Enter the marks of student %d in subject %d\n", i+1, j+1);
            scanf("%d", &marks[i][j]);
        }
    }

    for (Ent i=0; i<n; student++) {
        for (Ent j=0; j<n; subjects++) {
            printf("The marks of student %d in subject %d is: %d\n", i+1, j+1, marks[i][j]);
        }
    }

    return 0;
}

```

Output: `2 3 4  
5 6 7  
8 9 10`

Enter the marks of student 1 in subject 1 →   
 →   
 →   
 →   
 →

Enter the marks of student 1 in subject 1

(1)  $\text{float } \text{marks} = \text{input}(\text{"Enter marks of student 1 in subject 1"})$

The marks of student 1 in subject 1 is: 82

The marks of student 1 in subject 1 is: 82

(2)  $\text{float } \text{marks} = \text{input}(\text{"Enter marks of student 1 in subject 1"})$

if (marks > 80) {  
 print("Good")  
}

The marks of student 1 in subject 1 is: 82

## Array Declaration

The array has to be declared before it can be used in a program.

Syntax: `type variable_name [size]`

The type specifies the type of the elements that are to be stored in the array like int, float etc.

The variable name is any valid variable name in C. The size indicates the maximum number of elements that can be stored in array.

E.g. float per[50]

Accessing Accessing the elements of an array

In the above declared array the individual elements can be accessed by writing the subscript in the bracket after the array name. Thus marks[0] is the 1<sup>st</sup> element of the array, marks[1] is the second & so on.

E.g. for( $i=0$ ;  $i<10$ ;  $i++$ )

```

    {
        scanf("%d", &marks[i]);
    }

```

This code makes use of scanf to read elements into the array marks one by one.

Remember that the for loop should start with a 0, since the array subscript begins with 0. The for loop will execute until the value of  $i$  becomes 9.

It is also imp. to note here that C performs no bounds checking on arrays, i.e. there is no check on whether the subscript has exceeded the size of array.  $\therefore$  we should not enter data exceeding the array, otherwise results might be unpredictable.

E.g

#include <stdio.h>

int main () {

int num[5];

int sum, i;

float avg;

}

for(i=0; i<5; i++)

scanf("%d", &num[i]);

sum=0;

for(i=0; i<5; i++)

sum = sum + num[i];

avg = sum /

p) printf("The average of numbers is %f", avg);

# Initialization of Arrays

Syntax type array\_name[size] = {val1, val2...}

E.g. `int marks[5] = {40, 70, 32, 80, 76};`

`marks[0] = 40`

`marks[1] = 70`

`marks[2] = 32`

`marks[3] = 80`

`marks[4] = 76`

The size of the array may be omitted while initializing the array elements. The compiler will automatically allocate space for all elements.

E.g. `float num[7] = {43.8, 56.87, ..., 89.02};`

Each integer → 2 bytes memory

## Locations of Array Elements

10	20	5	2	8
5000	5002	5004	5006	5008

∴ 1<sup>st</sup> element at 5000, 2<sup>nd</sup> at 5002, 3<sup>rd</sup> at 5004

∴ array elements are stored in contiguous memory & since data type int occupies 2 bytes each element will be allocated 2 bytes.

If the array elements are not initialized at the point of their declaration they contain garbage values in the beginning. You can individually assign a value 0 to every element using a loop.

```
for(s=0; i<10; i++)
```

```
    mark[i]=0;
```

The storage class of array is by default as of type auto the elements of the array are not initialized to zero at the time of declaration. If the storage class is defined to be static all elements would have default initial value 0.

You cannot assign one array directly to another like ordinary variable i.e. if arr 1 & arr 2 are 2 arrays then

arr 1 = arr 2 is not valid enc

You have to explicitly assign individual elements of one array to corresponding elements of the other array.

## Two Dimensional Array

A 2D array is also called as matrix

<u>Student Id</u>	<u>Sub 1</u>	<u>Sub 2</u>	<u>Sub 3</u>
1100	40	50	67
2100	80	34	56
1330	90	98	89
1331	76	76	76
1485	80	70	65

2 D array can be declared as follows

type array name [row\_size][col\_size];

	C0	C1	C2	
R0	[0][0]	[0][1]	[0][2]	
R1	[1][0]	[1][1]	[1][2]	
R2	[2][0]	[2][1]	[2][2]	

## Initialising 2-D Arrays

Statement:  $\text{arr}[2][3] = \{10, 5, 3, 15, 20, 25\};$

$\text{arr}[0][0] = 10$        $\text{arr}[0][1] = 5$

$\text{arr}[0][2] = 3$        $\text{arr}[1][0] = 15$

$\text{arr}[1][1] = 20$        $\text{arr}[1][2] = 25$

∴ Elements are initialised row wise. There are 3 elements in each row of the array.

Alternative methods of generalization are:

`Ent arr[2][3] = {{10, 5, 3}, {15, 20, 25}};`

`q {{10, 5, 3}, {15, 20, 25}}`

In this way each row can be separated by braces. Commas are necessary after the closing of each row except the last. The foll. method may be used to generalize all the elements of a 2-D array to 0.

`Ent arr[2][3] = {{0}, {0}, {0}};`

This means that you are use a single zero for all elements of a column. Hence, you use 3 zeros to generalize all elements of each of the 3 columns.

→ In a 2D array the 1st dimension i.e. the row is optional.

But the 2nd dimension i.e. the column is optional. a must.

`Ent arr[1][3] = {10, 5, 3};`

However, `Ent arr[2][0]` & `Ent arr[0][0]` are invalid.

## Multidimensional Arrays

float arr1 [2][3][4];

float arr2 [4][3][2][2];

The 1<sup>st</sup> array would contain 24 elements whereas 2<sup>nd</sup> will contain 48 elements of type float.

E.g.

float arr1 [2][3][4] = {

{1,1,1,1},

{2,2,2,2},

{3,3,3,3}

};

{5,5,5,5},

{6,6,6,6},

{7,7,7,7},

{8,8,8,8},

{9,9,9,9},

{10,10,10,10},

{11,11,11,11},

{12,12,12,12},

{13,13,13,13},

{14,14,14,14},

{15,15,15,15},

{16,16,16,16},

{17,17,17,17},

{18,18,18,18},

{19,19,19,19},

{20,20,20,20},

{21,21,21,21},

{22,22,22,22},

{23,23,23,23},

{24,24,24,24},

{25,25,25,25},

{26,26,26,26},

{27,27,27,27},

{28,28,28,28},

{29,29,29,29},

{30,30,30,30},

{31,31,31,31},

{32,32,32,32},

{33,33,33,33},

{34,34,34,34},

{35,35,35,35},

{36,36,36,36},

{37,37,37,37},

{38,38,38,38},

{39,39,39,39},

{40,40,40,40},

{41,41,41,41},

{42,42,42,42},

{43,43,43,43},

{44,44,44,44},

{45,45,45,45},

{46,46,46,46},

{47,47,47,47},

{48,48,48,48},

{49,49,49,49},

{50,50,50,50},

{51,51,51,51},

{52,52,52,52},

{53,53,53,53},

{54,54,54,54},

{55,55,55,55},

{56,56,56,56},

{57,57,57,57},

{58,58,58,58},

{59,59,59,59},

{60,60,60,60},

{61,61,61,61},

{62,62,62,62},

{63,63,63,63},

{64,64,64,64},

{65,65,65,65},

{66,66,66,66},

{67,67,67,67},

{68,68,68,68},

{69,69,69,69},

{70,70,70,70},

{71,71,71,71},

{72,72,72,72},

{73,73,73,73},

{74,74,74,74},

{75,75,75,75},

{76,76,76,76},

{77,77,77,77},

{78,78,78,78},

{79,79,79,79},

{80,80,80,80},

{81,81,81,81},

{82,82,82,82},

{83,83,83,83},

{84,84,84,84},

{85,85,85,85},

{86,86,86,86},

{87,87,87,87},

{88,88,88,88},

{89,89,89,89},

{90,90,90,90},

{91,91,91,91},

{92,92,92,92},

{93,93,93,93},

{94,94,94,94},

{95,95,95,95},

{96,96,96,96},

{97,97,97,97},

{98,98,98,98},

{99,99,99,99},

{100,100,100,100},

{101,101,101,101},

{102,102,102,102},

{103,103,103,103},

{104,104,104,104},

{105,105,105,105},

{106,106,106,106},

{107,107,107,107},

{108,108,108,108},

{109,109,109,109},

{110,110,110,110},

{111,111,111,111},

{112,112,112,112},

{113,113,113,113},

{114,114,114,114},

{115,115,115,115},

{116,116,116,116},

{117,117,117,117},

{118,118,118,118},

{119,119,119,119},

{120,120,120,120},

{121,121,121,121},

{122,122,122,122},

{123,123,123,123},

{124,124,124,124},

{125,125,125,125},

{126,126,126,126},

{127,127,127,127},

{128,128,128,128},

{129,129,129,129},

{130,130,130,130},

{131,131,131,131},

{132,132,132,132},

{133,133,133,133},

{134,134,134,134},

{135,135,135,135},

{136,136,136,136},

{137,137,137,137},

{138,138,138,138},

{139,139,139,139},

{140,140,140,140},

{141,141,141,141},

{142,142,142,142},

{143,143,143,143},

{144,144,144,144},

{145,145,145,145},

{146,146,146,146},

{147,147,147,147},

{148,148,148,148},

{149,149,149,149},

{150,150,150,150},

{151,151,151,151},

{152,152,152,152},

{153,153,153,153},

{154,154,154,154},

{155,155,155,155},

{156,156,156,156},

{157,157,157,157},

{158,158,158,158},

{159,159,159,159},

{160,160,160,160},

{161,161,161,161},

{162,162,162,162},

{163,163,163,163},

{164,164,164,164},

{165,165,165,165},

{166,166,166,166},

{167,167,167,167},

{168,168,168,168},

{169,169,169,169},

{170,170,170,170},

{171,171,171,171},

{172,172,172,172},

{173,173,173,173},

{174,174,174,174},

{175,175,175,175},

{176,176,176,176},

{177,177,177,177},

{178,178,178,178},

{179,179,179,179},

{180,180,180,180},

{181,181,181,181},

{182,182,182,182},

{183,183,183,183},

{184,184,184,184},

{185,185,185,185},

{186,186,186,186},

{187,187,187,187},

{188,188,188,188},

{189,189,189,189},

{190,190,190,190},

{191,191,191,191},

{192,192,192,192},

{193,193,193,193},

{194,194,194,194},

{195,195,195,195},

{196,196,196,196},

{197,197,197,197},

{198,198,198,198},

{199,199,199,199},

{200,200,200,200},

{201,201,201,201},

{202,202,202,202},

{203,203,203,203},

{204,204,204,204},

{205,205,205,205},

{206,206,206,206},

{207,207,207,207},

{208,208,208,208},

{209,209,209,209},

{210,210,210,210},

{211,211,211,211},

{212,212,212,212},

{213,213,213,213},

{214,214,214,214},

{215,215,215,215},

{216,216,216,216},

{217,217,217,217},

{2

0<sup>th</sup> 2-D array having total 12 elements

Array elements in natural order  
[3][4] 2 2 2 2

3 3 3

5 5 5 5

1<sup>st</sup> 2-D

array pointing to 6th element of 6 static cells  
[3][4] 7 7 7 7

1<sup>st</sup> element is arr[0][0][0]. Then counting of elements on a 3-D array also begins with 0. ∴ 2<sup>nd</sup> element is arr[0][0][1]

### Ch. 7 → Practice Set

1) Create an array of 10 integers. Verify using pointer arithmetic that (ptr + 2) points to the third element where ptr is a pointer pointing to the 1<sup>st</sup> element of the array.

Code: #include <stdio.h>

```
int main()
```

```
int arr[10];
```

```
// int *ptr = &arr[0];
```

```
int *ptr = arr;
```

```
ptr = ptr + 2;
```

```
if (ptr == &arr[2]) {
```

```
printf("These point to the same location  
in memory\n"); }
```

else if

printf("These do not point to the  
same location in memory\n");

\*S

S

S

S

[ANSWER]

return 0;

S S

\*S

S S

S

S

\$ Output:

These point to the same location in  
memory.

(Q-8 P2)

Yell (8)

Q2) If S[3] is a 1D array of integers  
then \*(S+3) refers to the 3rd element.

(a) True

(b) False

(c) Depends

\*(S+2) points to 3rd element in S.

Q3) Write a program to create an  
array of ten integers & store multiplication  
table of 5 in it.

Code: #include <stdio.h>

```
int main() {  
    int mult[10];  
    for (int i=0; i<10; i++) {  
        mult[i] = 5 * (i+1);  
    }  
}
```

```
for (int i=0; i<10; i++) {
    printf("5 * %d = %d\n", i+1, mult[i]);
}
```

Output

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

$$5 \times 3 = 15$$

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

$$5 \times 6 = 30$$

$$5 \times 7 = 35$$

$$5 \times 8 = 40$$

$$5 \times 9 = 45$$

$$5 \times 10 = 50$$

Q) Write a program containing a function which reverses the array passed to it.

Code: #include <stdio.h>

```
void reverse(int* arr, int n) {
```

    int temp;

```
    for (int i=0; i<(n/2); i++) {
```

        temp = arr[i];

        arr[i] = arr[n-i-1];

        arr[n-i-1] = temp;

}

}

```

ent main() {
    ent arr[7] = {1, 2, 3, 4, 5, 6, 7};
    reverse Carr, 7);
    for (ent i=0; i<7; i++) {
        printf("The value of %d element is : %d\n",
               i, arr[i]);
    }
    return 0;
}

```

Output

The value of 0 element is : 27 = 8 × 2  
 1 } : 6 = 2 × 2  
 2 } : 12 = 2 × 2  
 3 } : 48 = 3 × 2  
 4 } : 32 = 4 × 2  
 5 } : 20 = 8 × 2  
 6 } : 12 = 2 × 2  
 7 } : 0 = 0 × 2

Q.5) Create an array of size 3×10 containing multiplication tables of the numbers 2, 7, 8, 9 respectively.

```
#include < stdio.h >
```

```

ent main() {
    ent mul_table[3][10];
    for (ent i=0; i<10; i++) {
        mul_table[0][i] = 2 * (i+1);
        mul_table[1][i] = 7 * (i+1);
        mul_table[2][i] = 8 * (i+1);
    }
}
```

```
for (int i=0; i<10; i++) {  
    printf("%d\t", i);  
}
```

Write same for statement for 7 8 9

return 0;

y

Output

$$2 \times 1 = 2$$

$$7 \times 1 = 7$$

$$9 \times 1 = 9$$

$$2 \times 2 = 4$$

$$7 \times 2 = 14$$

$$9 \times 2 = 18$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$2 \times 10 = 20$$

$$7 \times 10 = 70$$

$$9 \times 10 = 90$$

By function

```
#include <stdio.h>
```

```
void printTable(int *mulTable, int num, int n) {  
    printf("The multiplication table of %d is:\n",  
          num);  
    for (int i=0; i<n; i++) {  
        mulTable[i] = num*(i+1);  
    }  
}
```

```
for (int i=0; i<n; i++) {
```

```
    printf("%d\t", i);  
}
```

y

y

Ent math() {

Ent mulTable[3][10];

parentTable (mulTable[0], 2, 10);

parentTable (mulTable[1], 7, 10);

parentTable (mulTable[2], 9, 10);

return 0;

y

Output

The multiplication table of 2 is:

$$2 \times 1 = 2$$

$$2 \times 2 = 4$$

⋮

$$2 \times 10 = 10$$

$$0 = 0 \times 1$$

$$0 = 0 \times 1$$

2) The mult. table of 7 is:

$$7 \times 1 = 7$$

$$7 \times 2 = 14$$

$7 \times 3 = 21$  etc. up to  $7 \times 10 = 70$

same for 9 i.e.  $9 \times 0 = 0$  to  $9 \times 10 = 90$

Q.6 Create a 3D array & print the address of all elements in enc. order

$7 \times 3 \times 10$  i.e.  $210 = 7 \times 3 \times 10$

Ans.  $1 + 3 + 10 = 14$  i.e.  $14! \times 10!$  is required.

Code: `#include < stdio.h>`

```
Ent main () {  
    Ent arr [2][3][4];  
    for (Ent i=0; i<2; i++) {  
        for (Ent j=0; j<3; j++) {  
            for (Ent k=0; k<4; k++) {  
                printf ("The address of arr [%d][%d][%d] is  
                %u\n", i, j, k, &arr[i][j][k]);  
            }  
        }  
    }  
    return 0;  
}
```

The address of arr [0][0][0] is 6422116

Output:

The address of arr [0][0][0] is 6422116  
[0][0][1] — 6422120

[0][0][2] — 6422124  
[0][0][3] — 6422128  
[0][1][0] — 6422132  
[0][1][1] — 6422136  
[0][1][2] — 6422140  
[0][1][3] — 6422144

maximum address unprintable.