Ch. 7 → Arrays

An array is a collection of similar elements.

One variable ⇒ Capable of storing multiple values.

a ⇔ a ⇔ a[0]
a[3] ⇔ *(a+3)
If a is a array a++, a-- is not allowed.

Date
Page

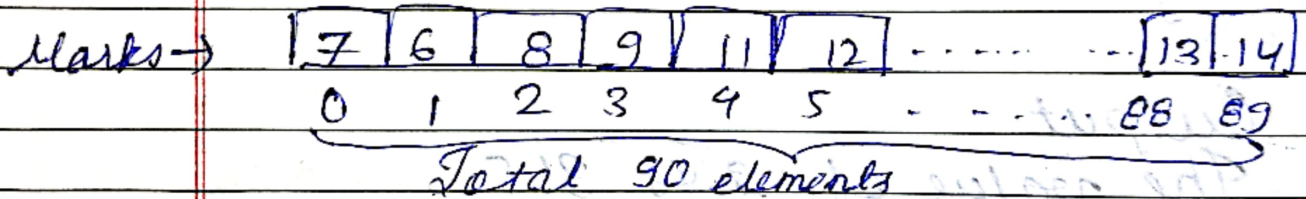
Syntax
The syntax of ~~declaration~~ declaring an array looks like this.

```
int marks[90] ⇒ Integer Array
char name[20] ⇒ Character Array or Str
float percentile[90] ⇒ 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.



Accessing elements
Elements of an array can be accessed using

```
scanf("%d", &marks[0]); ⇒ Input 1st value
printf("%d", marks[0]); ⇒ Output 1st 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 student1: ");
    scanf("%d", &marks[0]);
    printf("Enter the value of marks for student2: ");
    scanf("%d", &marks[1]);
    printf("Enter the value of marks for student3: ");
    scanf("%d", &marks[2]);
    printf("Enter the value of marks for student4: ");
    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
2 : 90
3 : 10
4 : 10

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>

int main(){
    int marks[5];

    for (int i=0; i<5; i++)
    {
    printf("Enter the value of marks for
                                student %d: ", i+1);
    scanf("%d", &marks[i]);
    }

    for(int 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
2 : 6
3 : 8
4 : 10
5 : 9
```

Initialization of an array

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

int cgpa[3]={9,8,8}; ⇒ array can be initialized while declaration

float marks[]={33,40};

- Arrays in memory

Consider the array

Arrays in Memory

Consider the array:

int arr[3]={1,2,3} ⇒ 1 integer = 4 bytes

It will reserve 4x3 = 12 bytes in memory 4 bytes for each integer.

1	2	3	⇒ arr in memory
62302	62306	62310	

Pointer Arithmetic

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

Consider the example

int i = 32;

int *a = &i; ⇒ a = 87994

a++; ⇒ Now a = 87998

i

32

add = 87994

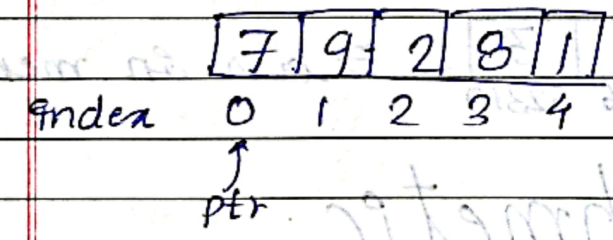
```
char a = 'A';
char *b = &a; ⇒ b = 87994
b++;          ⇒ Now b = 87995

float i = 1.7;
float *a = &i; ⇒ Add. of i or a = 87994
a++;           ⇒ 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



If ptr point to index 0, ptr++ will point to index 1 & so on...

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

```
int* ptr = &arr[0]; → or 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: \n", 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:
21
Enter the value of marks for student 2:
22
Enter the value of marks for student 3:
23
Enter the value of marks for student 4:
24

The value of marks for student 1 is 21
" " " 2 is 22
" " " 3 is 23
" " " 4 is 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, 3543, 34, 3, 645, 23};
    printArray(arr, 7);
    return 0;
}
```

Output
```
The value of element 1 is 1
" " " 2 — 2
" " " 3 — 3543
" " " 4 — 34
" " " 5 — 3
" " " 6 — 645
" " " 7 — 23
```

Passing arrays to functions
Arrays can be passed to the functions like this

printArray(arr, n); ⇒ function call

Void printArray(int *i, int n); ⇒

function prototype

or

Void printArray(int i[], int n);

Multi-dimensional Arrays

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

A 2 dimensional array can be defined as:

~~int 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

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

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

Value 1 value 4

Code:

```
#include <stdio.h>

int main() {
    int n_students = 3;
    int n_subjects = 5;

    int marks[3][5];
    for(int i=0; i<n_students; i++){

    for(int j=0; j<n_subjects; j++){

    printf("Enter the marks of student
        %d in subject %d \n", i+1, j+1);
    scanf("%d", &marks[i][j]);
    }
    }

    for(int i=0; i<n_students; i++){
    for(int j=0; j<n_subjects; j++){

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

    }

    return 0;
```

Output:

Enter the marks of student 1 in subject 1 → 2
" 2 → 3
" 3 → 4
" 4 → 5
" 5 → 6

Enter
Enter the marks of student 2 on subject 1

1 ——— 5

same for 3

The marks of student 1 on subject 1 is: 2

The marks of student 3 on subject 5 is -

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 on 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 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 1st 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 begin 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. ∴ we should not enter data exceeding the array, otherwise results might be unpredictable.

Date
Page

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/5;
printf("The average of numbers is %f", avg);
}
```

Initialization of Arrays

Syntax → type array_name[size] = {val1, val2 ... valn};

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

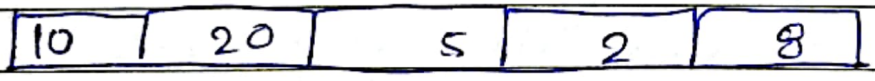
marks[0] = 40
marks[1] = 70
marks[2] = 32
marks[3] = 80
marks[4] = 76

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[] = {43.8, 56.87, ... 89.024}

Each integer → 2 bytes memory

Locations of Array elements



1st element at 5000, 2nd at 5002 & 3rd 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 using initialised 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(i=0; i<10; i++)

    mark[i]=0;
```

∵ storage class of array is by default is 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 in C

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

Student_Id	Sub 1	Sub 2	Sub 3
1100	40	50	67
2100	80	34	56
1330	90	98	89
1331	76	76	76
1485	80	70	65

2D 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]

Initializing 2-D Arrays

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

arr[0][0] = 10 arr[0][1] = 5

arr[0][2] = 3 arr[1][0] = 15

arr[1][1] = 20 arr[1][2] = 25

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

Alternative methods of initialisation are:

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

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

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

```
int 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 initialize 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.

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

However
int arr[2][0] & int arr[0][0] are invalid

Multidimensional Arrays

```
int arr1 [2][3][4];
float arr2[4][3][2][2];
```

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

E.g.

```
int 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},
                    }
                   };
```

A 3D array can be considered as a 2D array or an another 2-D array. The outermost array is an array with 2-elements where each element itself is a 2-D array whose dimensions are [3][4].

Thus the above 2D array can be represented as a series of 2D arrays as follows.

Date
Page

0th 2-D Array [3][4]

1 1 1 1

2 2 2 2

3 3 3 3

5 5 5 5

1st 2-D Array [3][4]

6 6 6 6

7 7 7 7

∴ 1st element is arr1[0][0][0]. The counting of elements in a 3-D array also begins with 0. ∴ 2nd element is arr1[0][0][1]

Ch. 7 → Practice Set

1) Create an array of 10 nos. & verify using pointer arithmetic that (ptr+2) points to the third element where ptr is a pointer pointing to the 1st 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("They point to the same location in memory\n");
    }
```

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

    return 0;

    }
```

// Output:
These point to the same location in memory.

Q2) If S[3] is a 1D array of integers then *(S+3) refers to the third element:
(a) True
(b) False
(c) Depends

*(S+2) points to 3rd element

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

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

return 0;

}
```

Output

5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50

(4) 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;
}
}
```

```
int main(){
    int arr[7]={1,2,3,4,5,6,7};
    reverse(arr,7);
    for(int 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: 7
             1          : 6
             2          : 5
             3          : 4
             4          : 3
             5          : 2
             6          : 1
```

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

```
#include<stdio.h>

int main(){
    int mul table[3][10];
    for(int i=0;i<10;i++){
        mul table[0][i]=2*(i+1);
    }
```

```
for(int i=0; i<10; i++){
    printf("2 x %d \n", i+1, mul table[0][i]);
```

Write same for statement for 7 & 9

```
return 0;
}
```

Output

2 x 1 = 2	7 x 1 = 7	9 x 1 = 9
2 x 2 = 4	7 x 2 = 14	9 x 2 = 18
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
2 x 10 = 20	7 x 10 = 70	9 x 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 x %d = %d \n", num, i+1, mulTable
                                    mulTable[i]);
    }
}
```

```
int main(){
int mulTable[3][10];
printTable(mulTable[0], 2, 10);
printTable(mulTable[1], 7, 10);
printTable(mulTable[2], 9, 10);

return 0;
}
```

Output

The multiplication table of 2 is:

2×1 = 2
2×2 = 4
.
.
.
2×10 = 10

The mult. table of 7 is:

7×1 = 7
7×2 = 14
.
.
.
7×10 = 70

same for 9

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

Code:

```
#include<stdio.h>

int main(){
  int arr[2][3][4];
  for(int i=0;i<2;i++){
  for(int j=0;j<3;j++){
  for(int 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

Output:

The address of arr[0][0][0] is 6422116
arr[0][0][1] — 6422120
arr[0][0][2]
arr[0][0][3]
arr[0][1][0]
arr[0][1][1]
arr[0][1][2]
arr[0][1][3]