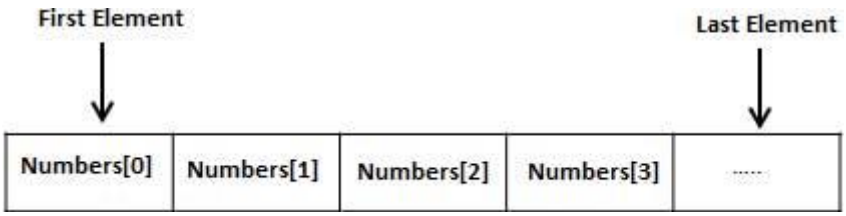
C - Arrays

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the **same type**.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [arraySize];

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement −

double balance[10];

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

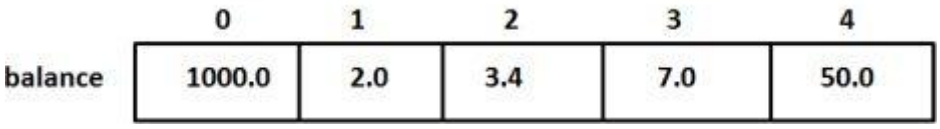
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

balance[4] = 50.0;

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above –



Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

double salary = balance[9];

The above statement will take the 10th element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays –

```
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
```

```
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Multi-dimensional Arrays in C: C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration –

type name[size1][size2]...[sizeN];

For example, the following declaration creates a three dimensional integer array –

int threedim[5][10][4];

Two-dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows –

type arrayName [x][y];

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows –

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
  {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example –

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array. You can verify it in the above figure. Let us check the following program where we have used a nested loop to handle a two-dimensional array –

```
#include <stdio.h>

int main () {

  /* an array with 5 rows and 2 columns*/
  int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
  int i, j;

  /* output each array element's value */
  for ( i = 0; i < 5; i++ ) {

    for ( j = 0; j < 2; j++ ) {
      printf("a[%d][%d] = %d\n", i,j, a[i][j] );
    }
  }

  return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Passing Arrays as Function Arguments in C

If you want to pass a single-dimension array as an argument in a function, you would have to declare a formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similarly, you can pass multi-dimensional arrays as formal parameters.

Way-1

Formal parameters as a pointer –

```
void myFunction(int *param) {
  .
  .
  .
}
```

Way-2

Formal parameters as a sized array –

```
void myFunction(int param[10]) {
  .
  .
  .
}
```

Way-3

Formal parameters as an unsized array –

```
void myFunction(int param[]) {
  .
  .
  .
}
```

Example

Now, consider the following function, which takes an array as an argument along with another argument and based on the passed arguments, it returns the average of the numbers passed through the array as follows –

```
double getAverage(int arr[], int size) {

  int i;
  double avg;
```

```
  double sum = 0;

  for (i = 0; i < size; ++i) {
    sum += arr[i];
  }

  avg = sum / size;

  return avg;
}
```

Now, let us call the above function as follows –

```
#include <stdio.h>

/* function declaration */
double getAverage(int arr[], int size);

int main () {

  /* an int array with 5 elements */
  int balance[5] = {1000, 2, 3, 17, 50};
  double avg;

  /* pass pointer to the array as an argument */
  avg = getAverage( balance, 5 ) ;

  /* output the returned value */
  printf( "Average value is: %f ", avg );

  return 0;
}
```

When the above code is compiled together and executed, it produces the following result –

Average value is: 214.400000

String and Character Array

String is a sequence of characters that is treated as a single data item and terminated by null character '\0'. Remember that C language does not support strings as a data type. A string is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

For example: The string "hello world" contains 12 characters including '\0' character which is automatically added by the compiler at the end of the string.

Declaring and Initializing a string variables

There are different ways to initialize a **character array** variable.

```
char name[13] = "StudyTonight";      // valid character array initialization

char name[10] = {'L','e','s','s','o','n','s','\0'};    // valid initialization
```

Remember that when you initialize a character array by listing all of its characters separately then you must supply the '\0' character explicitly.

String Input and Output

Input function scanf() can be used with %s format specifier to read a string input from the terminal. But there is one problem with scanf() function, it terminates its input on the first white space it encounters. Therefore if you try to read an input string "Hello World" using scanf() function, it will only read Hello and terminate after encountering white spaces.

However, C supports a format specification known as the edit set conversion code %[..] that can be used to read a line containing a variety of characters, including white spaces.

```
#include<stdio.h>

#include<string.h>

void main()

{

   char str[20];

   printf("Enter a string");

   scanf("%[^\n]", &str);  //scanning the whole string, including the white spaces

   printf("%s", str);

}
```

Another method to read character string with white spaces from terminal is by using the gets() function.

```
char text[20];

gets(text);

printf("%s", text);
```

String Handling Functions

C language supports a large number of string handling functions that can be used to carry out many of the string manipulations. These functions are packaged in string.h library. Hence, you must include string.h header file in your programs to use these functions.

The following are the most commonly used string handling functions.

Method	Description
strcat()	It is used to concatenate(combine) two strings
strlen()	It is used to show length of a string
strrev()	It is used to show reverse of a string
strcpy()	Copies one string into another
strcmp()	It is used to compare two string
strcat()	It is usedto add the strings

strcat("hello", "world");

strcat() function will "world" to "hello" i.e it will ouput helloworld.

strlen() function

strlen() function will return the length of the string passed to it.

```
int j;

j = strlen("studytonight");

printf("%d",j);
```

12

strcmp() function

strcmp() function will return the ASCII difference between first unmatching character of two strings.

```
int j;
```

```
j = strcmp("study", "tonight");
printf("%d",j);
```

-1

strcpy() function

It copies the second string argument to the first string argument.

```
#include<stdio.h>
#include<string.h>
int main()
{
   char s1[50];
   char s2[50];
   strcpy(s1, "StudyTonight");     //copies "studytonight" to string s1
   strcpy(s2, s1);     //copies string s1 to string s2
   printf("%s\n", s2);
   return(0);
}
```

StudyTonight

strrev() function

It is used to reverse the given string expression.

```
#include<stdio.h>
int main()
{
   char s1[50];

   printf("Enter your string: ");
```

```
    gets(s1);
    printf("\nYour reverse string is: %s",strrev(s1));
    return(0);
}
```

Enter your string: **studytonight**

Your reverse string is: **thginotyduts**