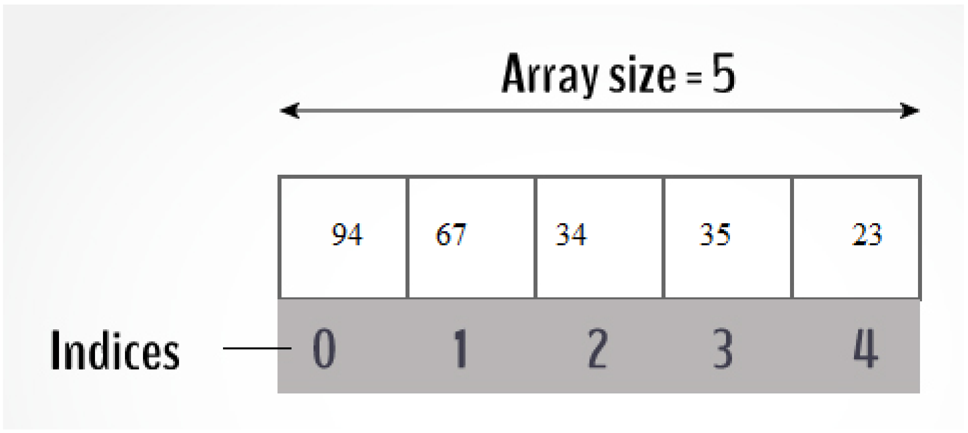
Array and String

Array is a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables 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};
int mark[5] = {19, 10, 8, 17, 9};
```

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

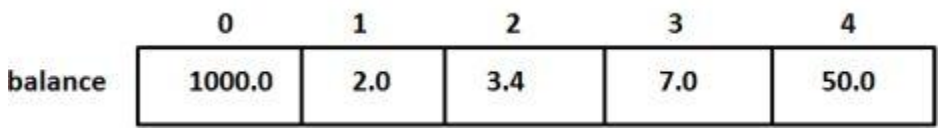
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
int mark[] = {19, 10, 8, 17, 9};
```

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
```

Change Value of Array elements

```
int mark[5] = {19, 10, 8, 17, 9}


// make the value of the third element to -1
```

```c
mark[2] = -1;


// make the value of the fifth element to 0
mark[4] = 0;
```

Input and Output Array Elements

Here's how you can take input from the user and store it in an array element.

```c
// take input and store it in the 3rd element
scanf("%d", &mark[2]);

// take input and store it in the ith element
scanf("%d", &mark[i-1]);
```

Here's how you can print an individual element of an array.

```c
// print the first element of the array
printf("%d", mark[0]);

// print the third element of the array
printf("%d", mark[2]);

// print ith element of the array
printf("%d", mark[i-1]);
```

Example 1: Array Input/Output

```c
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>


int main() {
  int values[5];
```

```
 printf("Enter 5 integers: ");

 // taking input and storing it in an array
 for(int i = 0; i < 5; ++i) {
   scanf("%d", &values[i]);
 }

 printf("Displaying integers: ");

 // printing elements of an array
 for(int i = 0; i < 5; ++i) {
   printf("%d\n", values[i]);
 }
 return 0;
}
```

Output

```
Enter 5 integers: 1
-3
34
0
3
Displaying integers: 1
-3
34
0
3
```

Here, we have used a for loop to take 5 inputs from the user and store them in an array. Then, using another for loop, these elements are displayed on the screen.

Example 2: Calculate Average

```c
// Program to find the average of n numbers using arrays

#include <stdio.h>
int main()
{
    int marks[10], i, n, sum = 0, average;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    for(i=0; i<n; ++i)
    {
        printf("Enter number%d: ",i+1);
        scanf("%d", &marks[i]);

        // adding integers entered by the user to the sum variable
        sum += marks[i];
    }

    average = sum/n;
    printf("Average = %d", average);

    return 0;
}
```

Output

```
Enter n: 5
Enter number1: 45
Enter number2: 35
Enter number3: 38
Enter number4: 31
Enter number5: 49
Average = 39
```

Here, we have computed the average of n numbers entered by the user.

Access elements out of its bound!

Suppose you declared an array of 10 elements. Let's say,

```
int testArray[10];
```

You can access the array elements from testArray[0] to testArray[9]. Now let's say if you try to access testArray[12]. The element is not available. This may cause unexpected output (undefined behavior). Sometimes you might get an error and some other time your program may run correctly. Hence, you should never access elements of an array outside of its bound.

Multidimensional arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −

DataType ArrayName[size1][size2]...[sizeN];

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

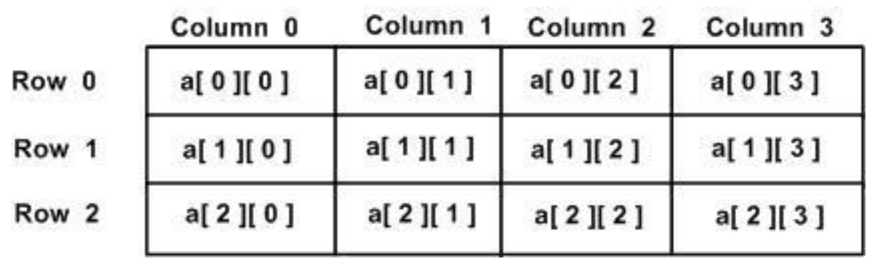
int threedim[5][10][4];

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 –

```c
#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
```

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

Initialization of a 3D array

You can initialize a three-dimensional array in a similar way like a two-dimensional array. Here's an example,

```
int test[2][3][4] = {{{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},{{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

Example 1: Two-dimensional array to store and print values

```
// C program to store temperature of two cities of a week and display it.
#include <stdio.h>
const int CITY = 2;
const int WEEK = 7;
int main()
{
  int temperature[CITY][WEEK];

  // Using nested loop to store values in a 2d array
  for (int i = 0; i < CITY; ++i)
  {
    for (int j = 0; j < WEEK; ++j)
    {
      printf("City %d, Day %d: ", i + 1, j + 1);
      scanf("%d", &temperature[i][j]);
    }
  }
  printf("\nDisplaying values: \n\n");
```

```
 // Using nested loop to display vlues of a 2d array
 for (int i = 0; i < CITY; ++i)
 {
   for (int j = 0; j < WEEK; ++j)
   {
     printf("City %d, Day %d = %d\n", i + 1, j + 1, temperature[i][j]);
   }
 }
 return 0;
}
```

Output

```
City 1, Day 1: 33
City 1, Day 2: 34
City 1, Day 3: 35
City 1, Day 4: 33
City 1, Day 5: 32
City 1, Day 6: 31
City 1, Day 7: 30
City 2, Day 1: 23
City 2, Day 2: 22
City 2, Day 3: 21
City 2, Day 4: 24
City 2, Day 5: 22
City 2, Day 6: 25
City 2, Day 7: 26

Displaying values:

City 1, Day 1 = 33
City 1, Day 2 = 34
City 1, Day 3 = 35
City 1, Day 4 = 33
City 1, Day 5 = 32
City 1, Day 6 = 31
City 1, Day 7 = 30
City 2, Day 1 = 23
City 2, Day 2 = 22
City 2, Day 3 = 21
City 2, Day 4 = 24
City 2, Day 5 = 22
City 2, Day 6 = 25
City 2, Day 7 = 26
```

Example 2: Sum of two matrices

```
// C program to find the sum of two matrices of order 2*2

#include <stdio.h>
int main()
{
  float a[2][2], b[2][2], result[2][2];

  // Taking input using nested for loop
  printf("Enter elements of 1st matrix\n");
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      printf("Enter a%d%d: ", i + 1, j + 1);
      scanf("%f", &a[i][j]);
    }
```

```c
  // Taking input using nested for loop
  printf("Enter elements of 2nd matrix\n");
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      printf("Enter b%d%d: ", i + 1, j + 1);
      scanf("%f", &b[i][j]);
    }

  // adding corresponding elements of two arrays
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      result[i][j] = a[i][j] + b[i][j];
    }

  // Displaying the sum
  printf("\nSum Of Matrix:");

  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      printf("%.1f\t", result[i][j]);

      if (j == 1)
        printf("\n");
    }
  return 0;
}
```

Output

```
Enter elements of 1st matrix
Enter a11: 2;
Enter a12: 0.5;
Enter a21: -1.1;
Enter a22: 2;
Enter elements of 2nd matrix
Enter b11: 0.2;
Enter b12: 0;
Enter b21: 0.23;
Enter b22: 23;

Sum Of Matrix:
2.2     0.5
-0.9    25.0
```

Example 3: Three-dimensional array

```
// C Program to store and print 12 values entered by the user

#include <stdio.h>
int main()
{
  int test[2][3][2];

  printf("Enter 12 values: \n");

  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 3; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        scanf("%d", &test[i][j][k]);
      }
    }
  }

  // Printing values with proper index.

  printf("\nDisplaying values:\n");
  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 3; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        printf("test[%d][%d][%d] = %d\n", i, j, k, test[i][j][k]);
      }
    }
  }

  return 0;
}
```

Output

```
Enter 12 values:
1
2
3
4
5
6
7
```

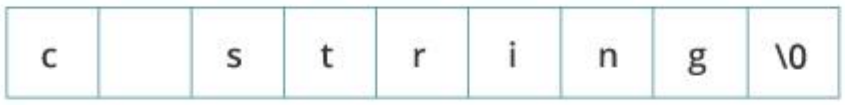
```
8
9
10
11
12

Displaying Values:
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```

String

In C programming, a string is a sequence of characters terminated with a null character \0. For example:

```
char c[] = "c string";
```

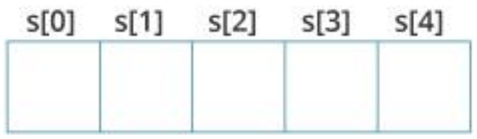
When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.



How to declare a string?

Here's how you can declare strings:

```c
char s[5];
```

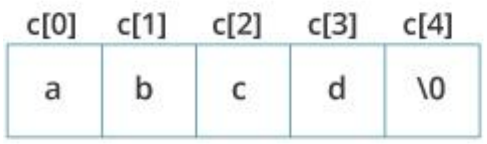


Here, we have declared a string of 5 characters.

How to initialize strings?

You can initialize strings in a number of ways.

```c
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



Let's take another example:

```c
char c[5] = "abcde";
```

Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters. This is bad and you should never do this.

Read String from the user

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab etc.).

Example 1: scanf() to read a string

```
#include <stdio.h>
int main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;
}
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Even though Dennis Ritchie was entered in the above program, only "Ritchie" was stored in the name string. It's because there was a space after Dennis.

How to read a line of text?

You can use the fgets() function to read a line of string. And, you can use puts() to display the string.

Example 2: fgets() and puts()

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Here, we have used fgets() function to read a string from the user.

fgets(name, sizeof(name), stdlin); // read string

The sizeof(name) results to 30. Hence, we can take a maximum of 30 characters as input which is the size of the name string.

To print the string, we have used puts(name);.

Note: The gets() function can also be to take input from the user. However, it is removed from the C standard. It's because gets() allows you to input any length of characters. Hence, there might be a buffer overflow.

Example

```
/* C program to Convert String to Uppercase without using strupr() */

#include <stdio.h>
#include <string.h>

int main()
{
	char Str1[100];
	int i;

	printf("\n Please Enter a String that you want to Convert into Uppercase :  ");
	fgets(Str1,sizeof(Str1),stdin);

	for (i = 0; Str1[i]!='\0'; i++)
	{
		if(Str1[i] >= 'a' && Str1[i] <= 'z')
		{
			Str1[i] = Str1[i] -32;
```

```
                    }
          }

          printf("\n The given String in Upper Case = %s", Str1);

          return 0;
}
```

Example

```
#include <stdio.h>
int main() {
    char s[] = "Programming is fun";
    int i;
    for (i = 0; s[i] != '\0'; ++i);

    printf("Length of the string: %d", i);
    return 0;
}
```

Output

```
Length of the string: 18
```

Common String Functions (String.h)

strlen()

The strlen() function takes a string as an argument and returns its length. The returned value is of type long int.

Example: C strlen() function

```
#include <stdio.h>
#include <string.h>
int main()
{
   char a[20]="Program";
   char b[20]={'P','r','o','g','r','a','m','\0'};
   printf("Length of string a = %ld \n",strlen(a));
   printf("Length of string b = %ld \n",strlen(b));
```

```
    return 0;
}
```

Output

```
Length of string a = 7
Length of string b = 7
```

Note that the strlen() function doesn't count the null character \0 while calculating the length.

strcpy()

Function prototype

```
char* strcpy(char* destination, const char* source);
```

The strcpy() function copies the string pointed by source (including the null character) to the character array destination. The function also returns the copied array.

Example: C strcpy()

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

int main()
{
    char str1[10]= "awesome";
    char str2[10];
    char str3[10];

    strcpy(str2, str1);
    strcpy(str3, "well");
    puts(str2);
    puts(str3);
```

```
   return 0;
}
```

Output

```
awesome
well
```

It is important to note that the destination array should be large enough to copy the array. Otherwise, it may result in undefined behavior.

strcmp()

The strcmp() function compares two strings and returns 0 if both strings are identical.

C strcmp() Prototype

```
int strcmp (const char* str1, const char* str2);
```

The `strcmp()` compares two strings character by character.
If the first character of two strings is equal, the next character of two strings are compared. This continues until the corresponding characters of two strings are different or a null character `'\0'` is reached.

Return Value from strcmp()

Return Value	Remarks
0	if both strings are identical (equal)
Negative	if the ASCII value of the first unmatched character is less than second.
positive integer	if the ASCII value of the first unmatched character is greater than second.

Example: C strcmp() function

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

int main()
{
    char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
    int result;

    // comparing strings str1 and str2
    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);

    // comparing strings str1 and str3
    result = strcmp(str1, str3);
    printf("strcmp(str1, str3) = %d\n", result);

    return 0;
}
```

Output

```
strcmp(str1, str2) = 32

strcmp(str1, str3) = 0
```

The first unmatched character between string str1 and str2 is third character. The ASCII value of 'c' is 99 and the ASCII value of 'C' is 67. Hence, when strings str1 and str2 are compared, the return value is 32.

When strings str1 and str3 are compared, the result is 0 because both strings are identical.

C strcat()

This function concatenates (joins) two strings.

strcat() Prototype

```
char *strcat(char *dest, const char *src)
```

It takes two arguments, i.e, two strings or character arrays, and stores the resultant concatenated string in the first string specified in the argument. The pointer to the resultant string is passed as a return value.

Example: C strcat() function

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "This is ", str2[] = "Year 2020";

    //concatenates str1 and str2 and resultant string is stored in str1.
    strcat(str1,str2);
    puts(str1);
    puts(str2);

    return 0;
}
```

Output

```
This is Year 2020

Year 2020
```