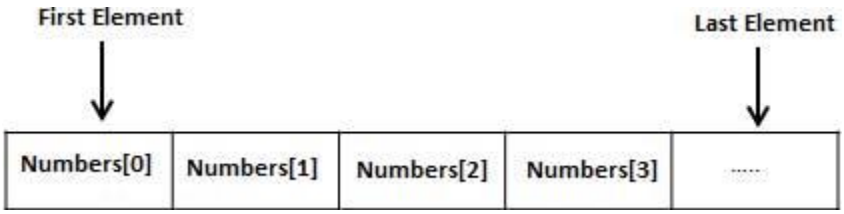
Advanced Variable Types

Array

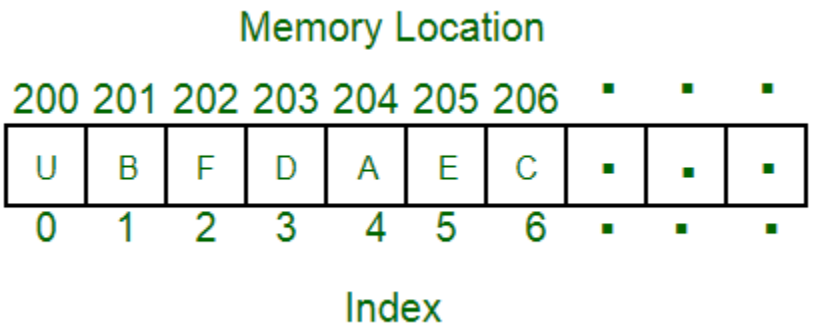
An array is collection of items stored at contiguous memory locations. The idea is to store multiple items of same type together.

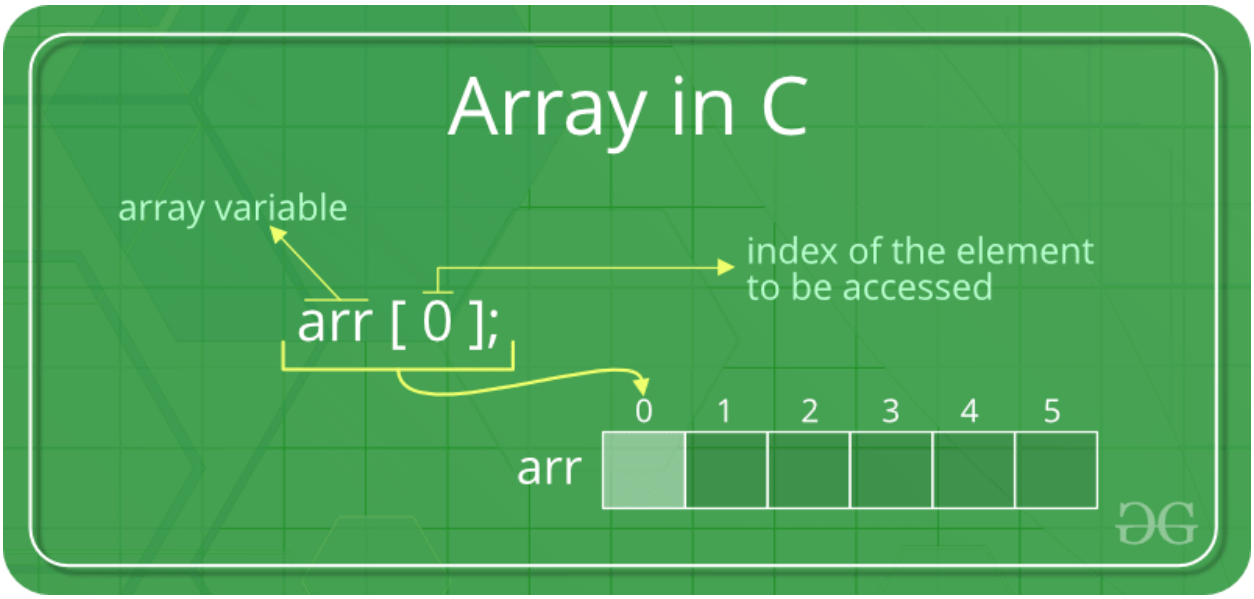
Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

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



An array is a series or list of variables in computer memory, all of which have the same name but are differentiated with special numbers called subscripts. A subscript is a number that indicates the position of a particular item within an array. Whenever you require multiple storage locations for objects, you are using a real-life counterpart of a programming array. For example, if you store important papers in a series of file folders and label each folder with a consecutive letter of the alphabet, then you are using the equivalent of an array.





Traversing an Array

Our basic operation starts with visiting each element of the array exactly once and in order. We can write that in pseudocode in two ways:

FOR i = 0 to array length - 1

FOR each element in the array

In the first example we are going by position starting at the first position 0 and going to the last position. In the second example we are going element by element.

Summing the Array

In any summing problem, we need a place or variable to store the sum. We have to initialize this variable to zero to start with and then traverse the array adding each element to the sum. Here is the pseudocode:

SET Sum to 0

FOR each element in the array

  ADD element to Sum

ENDFOR

PRINT Sum

Here is another version of the same code:

INITIALIZE Sum to 0

```
FOR i = 0 to array length - 1
  ADD array[i] to Sum
ENDFOR
WRITE Sum
```

Find Maximum Value in the Array

To find the maximum value, you initialize a placeholder called *max* with the value of the first element in the array. Then you go through the array element by element. If any element is greater than *max* you replace *max* with that element. Here is the pseudocode:

```
SET Max to array[0]
FOR i = 1 to array length - 1
  IF array[i] > Max THEN
    SET Max to array[i]
  ENDIF
ENDFOR
PRINT Max
```

Sequential Search for an Element in the Array

Let the element that we are searching for be X. We need to know if that element occurs in the array. One way is to return the position of the first occurrence of that element and if that element does not exist return -1 which is an invalid index. This is the pseudocode:

```
FOR i = 0 to array length - 1
  IF X = array[i] THEN
    RETURN i
  ENDIF
ENDFOR
RETURN -1
```

Another variation of this problem is to return the number of occurrences of X in the array. Here is a modification of the above code:

```
SET Count to 0
FOR i = 0 to array length - 1
```

```
  IF X = array[i] THEN
    INCREMENT Count
  ENDIF
ENDFOR
RETURN Count
```

How to declare an array?

dataType arrayName[arraySize];

For example,

float mark[5];

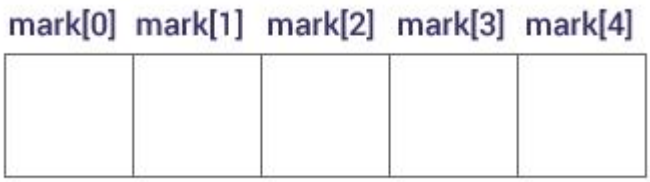
Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

It's important to note that the size and type of an array cannot be changed once it is declared.

Access Array Elements

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



Few keynotes:

- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]
- Suppose the starting address of mark[0] is **2120**. Then, the address of the mark[1] will be **2124**. Similarly, the address of mark[2] will be **2128** and so on. This is because the size of a float is 4 bytes.

- **Formula**

  LOC(A[i])= Base Address + size*i

How to initialize an array?

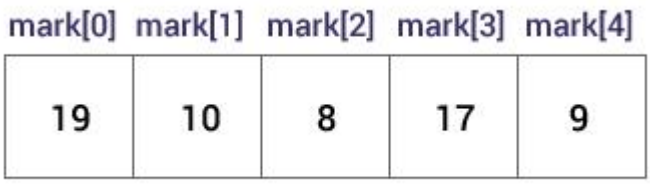
It is possible to initialize an array during declaration. For example,

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

You can also initialize an array like this.

```
1. int mark[] = {19, 10, 8, 17, 9};
```

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9

Change Value of Array elements

```
1. int mark[5] = {19, 10, 8, 17, 9}
2.
3. // make the value of the third element to -1
4. mark[2] = -1;
5.
6. // make the value of the fifth element to 0
```

```
7. 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.

```
1. // take input and store it in the 3rd element
2. scanf("%d", &mark[2]);
3. 
4. // take input and store it in the ith element
5. scanf("%d", &mark[i-1]);
```

Two-dimensional Array

Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

```
1. data_type array_name[rows][columns];
```

Consider the following example.

```
1. int twodimen[4][3];
```

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

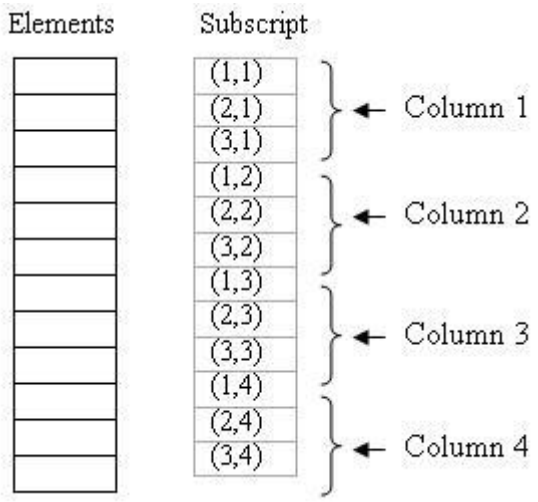
```
1. int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

Memory representation in 2D array

A 2D array's elements are stored in continuous memory locations. It can be represented in memory using any of the following two ways:

1. Column-Major Order
2. Row-Major Order

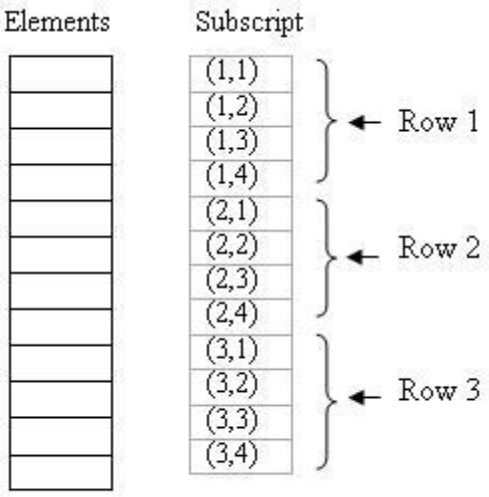
1. Column-Major Order:
In this method the elements are stored column wise, i.e. m elements of first column are stored in first m locations, m elements of second column are stored in next m locations and so on. E.g.
A 3 x 4 array will stored as below:



2. Row-Major Order:

In this method the elements are stored row wise, i.e. n elements of first row are stored in first n locations, n elements of second row are stored in next n locations and so on. E.g.

A 3 x 4 array will stored as below:



In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example,

```
1. float x[3][4];
```

Here, *x* is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Similarly, you can declare a three-dimensional (3d) array. For example,

```
1. float y[2][4][3];
```

Working with Files

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure.

A File is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard. So Files are used for storing information that can be processed by the programs. Files are not only used for storing the data, programs are also stored in files. In order to use files, we have to learn file input and output operations. That is, how data is read and how to write into a file.

Typical operations on files:

1. Open
2. Read
3. Write
4. Close

How is a file stored?

Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).

The last byte of a file contains the end-of-file character (EOF), with ASCII code 1A (hex).

While reading a text file, the EOF character can be checked to know the end.

Two kinds of files:

Text: contains ASCII codes only

Binary: can contain non-ASCII characters

- Image, audio, video, executable, etc.
- To check the end of file here, the file size value (also stored on disk) needs to be checked.

C provides a number of functions that helps to perform basic file operations. Following are the functions,

Function	**description**
fopen()	create a new file or open a existing file
fclose()	closes a file

getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

filename is the name of the file to be opened and **mode** specifies the purpose of opening the file. Mode can be of following types,

mode	**description**
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode

The fclose() function is used to close an already opened file.

Here fclose() function closes the file and returns **zero** on success, or **EOF** if there is an error in closing the file. This **EOF** is a constant defined in the header file **stdio.h**.

Difference between Append and Write Mode

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.

The only difference they have is, when you **open** a file in the **write** mode, the file is reset, resulting in deletion of any data already present in the file. While in **append** mode this will not happen. Append mode is used to append or add data to the existing data of file(if any). Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file.

`fseek()`, `ftell()` and `rewind()` functions

- fseek(): It is used to move the reading control to different positions using fseek function.
- ftell(): It tells the byte location of current position of cursor in file pointer.
- rewind(): It moves the control to beginning of the file.

References:

https://www.geeksforgeeks.org/array-data-structure/

https://www.cs.utexas.edu/~mitra/csFall2017/cs303/lectures/basic_algo.html

https://www.programiz.com/c-programming/c-arrays

http://www.xpode.com/ShowArticle.aspx?ArticleId=499

https://www.javatpoint.com/two-dimensional-array-in-c