**Introduction:**

A Pointer in C language is a variable which holds the address of another variable of same data type.

Pointers are used to access memory and manipulate the address.

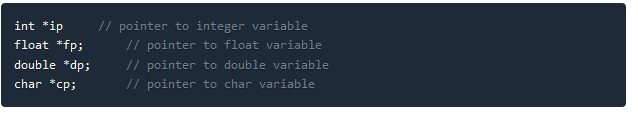
Pointers are one of the most distinct and exciting features of C language. It provides power and flexibility to the language. Although pointers may appear a little confusing and complicated in the beginning, but trust me, once you understand the concept, you will be able to do so much more with C language.

## **Declaration of C Pointer variable**

General syntax of pointer declaration is,

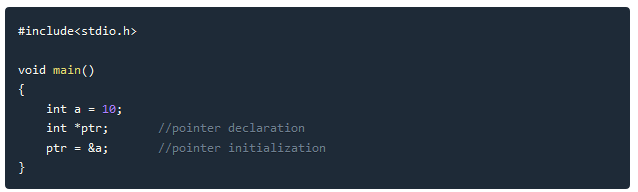
datatype \*pointer\_name;

Here are a few examples:

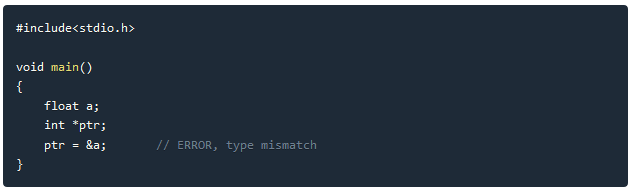


## **Initialization of C Pointer variable**

**Pointer Initialization** is the process of assigning address of a variable to a **pointer** variable. Pointer variable can only contain address of a variable of the same data type. In C language **address operator** & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.



Pointer variable always point to variables of same datatype. Let's have an example to showcase this:



If you are not sure about which variable's address to assign to a pointer variable while declaration, it is recommended to assign a NULL value to your pointer variable. A pointer which is assigned a NULL value is called a **NULL pointer**.

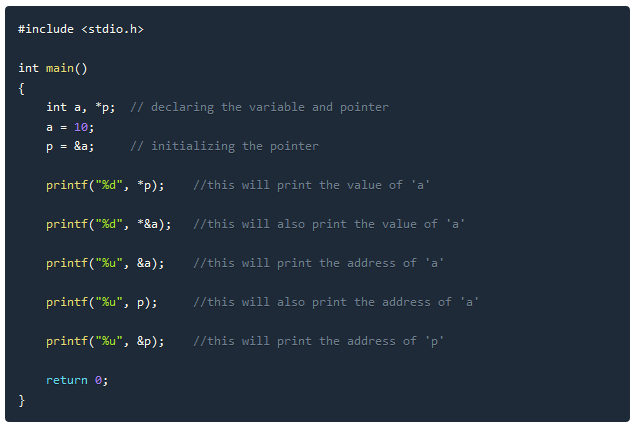


### 

### 

### **Using the pointer or Dereferencing of Pointer**

Once a pointer has been assigned the address of a variable, to access the value of the variable, pointer is **dereferenced**, using the **indirection operator** or **dereferencing operator** \*.



# 

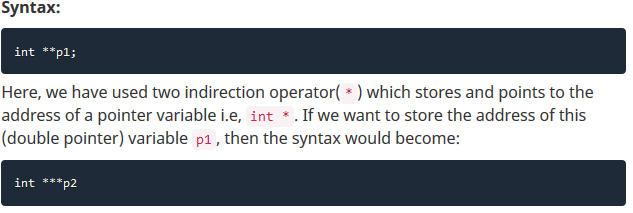
# 

# 

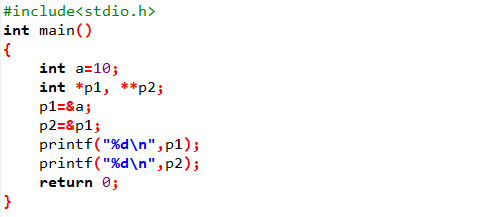
# 

# **Pointer to a Pointer in C(Double Pointer)**

Pointers are used to store the address of other variables of similar datatype. But if you want to store the address of a pointer variable, then you again need a pointer to store it. Thus, when one pointer variable stores the address of another pointer variable, it is known as **Pointer to Pointer** variable or **Double Pointer**.



**Example:**



**Output:**

6487572

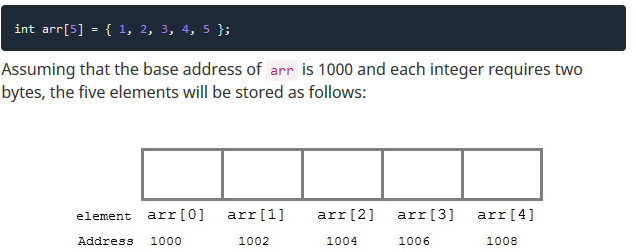
6487560

Note: The addresses may differ…

# **Pointer and Arrays in C**

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address i.e address of the first element of the array is also allocated by the compiler.

Suppose we declare an array arr,



Here, variable arr will give the base address, which is a constant pointer pointing to the first element of the array, arr[0]. Hence arr contains the address of arr[0] i.e 1000. In short, arr has two purpose - it is the name of the array and it acts as a pointer pointing towards the first element in the array.

**arr=&arr[0]** by default

We can also declare a pointer of type int to point to the array arr.

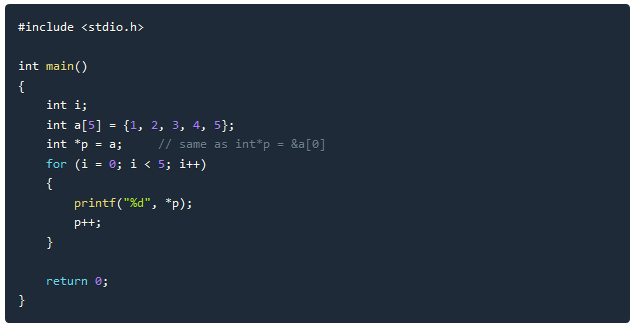


Now we can access every element of the array arr using p++ to move from one element to another.

**NOTE:** You cannot decrement a pointer once incremented. p-- won't work.

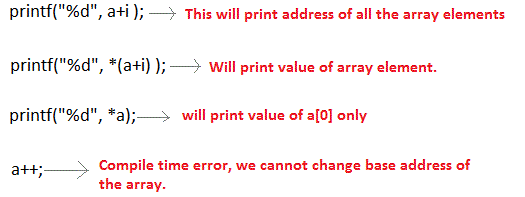
**Example of pointer to array**

As studied above, we can use a pointer to point to an array, and then we can use that pointer to access the array elements. Lets have an example,

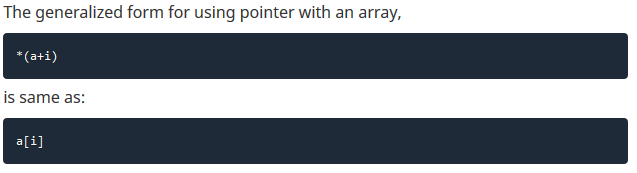


In the above program, the pointer \*p will print all the values stored in the array one by one. We can also use the Base address (a in above case) to act as a pointer and print all the values.

**What will happen if ?**



**Conclusion:**



**Accessing two dimensional array using pointer**

Let us suppose a two-dimensional array

int matrix[3][3];

For the above array,

matrix =>Points to base address of two-dimensional array.

Since array decays to pointer.

\*(matrix) =>Points to first row of two-dimensional array.

\*(matrix + 0) =>Points to first row of two-dimensional array.

\*(matrix + 1) => Points to second row of two-dimensional array.

\*\*matrix => Points to matrix[0][0]

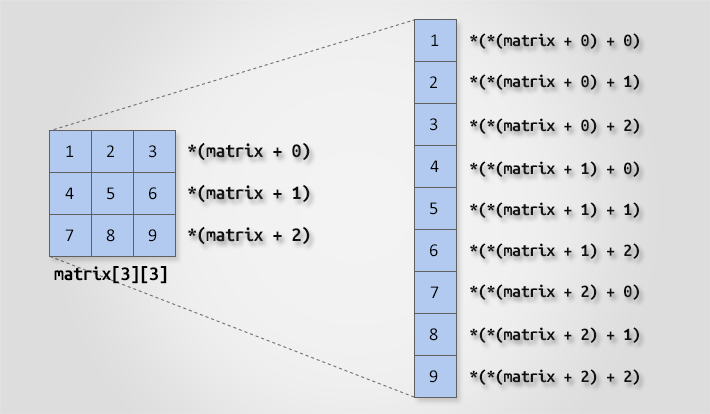
\*(\*(matrix + 0)) => Points to matrix[0][0]

\*(\*(matrix + 0) + 0) => Points to matrix[0][0]

\*(\*matrix + 1) => Points to matrix[0][1]

\*(\*(matrix + 0) + 1) => Points to matrix[0][1]

\*(\*(matrix + 2) + 2) => Points to matrix[2][2]



**Arrays of Pointer in C**

Just like we can declare an array of int, float or char etc, we can also declare an array of pointers, here is the syntax to do the same.

**Syntax:** datatype \*array\_name[size];

Let’s take an example:

int \*array[5];

Here array is an array of 5 integer pointers. It means that this array can hold the address of 5 integer variables. In other words, you can assign 5 pointer variables of type pointer to int to the elements of this array.

The following program demonstrates how to use an array of pointers.

#include<stdio.h>

#define SIZE 10

int main()

{

int \*arrop[3];

int a = 10, b = 20, c = 50, i;

arrop[0] = &a;

arrop[1] = &b;

arrop[2] = &c;

for(i = 0; i < 3; i++)

{

printf("Address = %d\t Value = %d\n", arrop[i], \*arrop[i]);

}

return 0;

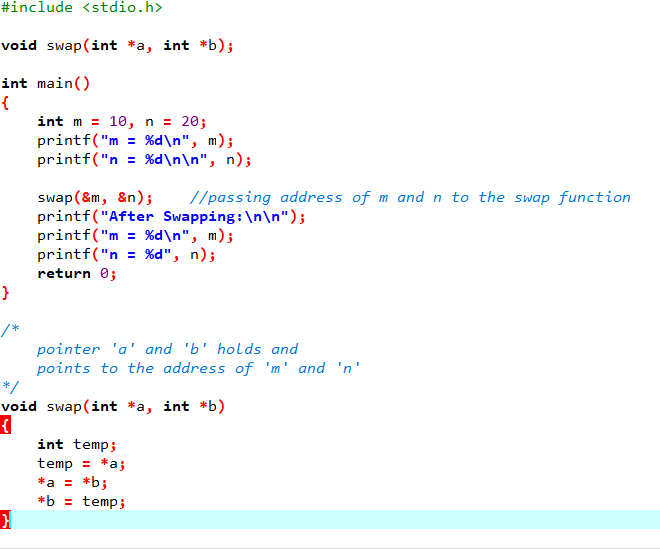
}



# Pointers as Function Argument in C

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as **call by reference**. When a function is called by reference any change made to the reference variable will effect the original variable.

### Example: Swapping two numbers using Pointer



m = 10

n = 20

After Swapping:

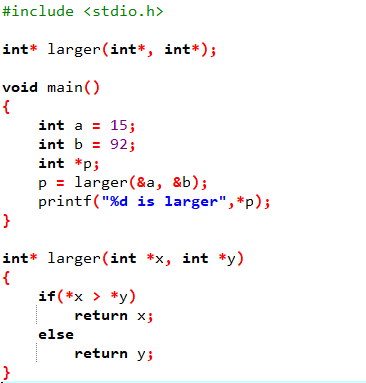
m = 20

n = 10

### 

### Functions returning Pointer variables

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function. They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.



92 is larger

**Dynamic Memory Allocation(DMA) in C**

The process of allocating and freeing memory at runtime is known as Dynamic Memory Allocation. This reserves the memory required by the program and returns this valuable resource to the system once the use of reserved space is utilized. Dynamic memory is managed and served with pointers that point to the newly allocated space of memory.

Why DMA?

In most situations , it is not possible to know the size of the memory required until runtime. For example, let us consider a program which asks users for number of students. According to a given number of students, it asks marks of each student and stores in an array to process the marks. The size of the array used to store the marks of students is fixed. Suppose the size of the array has been fixed to 100. The program wouldn’t work if the number of students is more than 100. If the number of students is less than 100 say 10, only 10 memory locations are used and rest 90 locations are reserved but nor used i.e. wastage of memory occurs. In this situation DMA will be useful.

There are four library functions: malloc(), calloc(),free() and realloc() for memory management. These functions are defined within header file stdlib.h and alloc.h

malloc():

**“malloc”** or **“memory allocation”** method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.

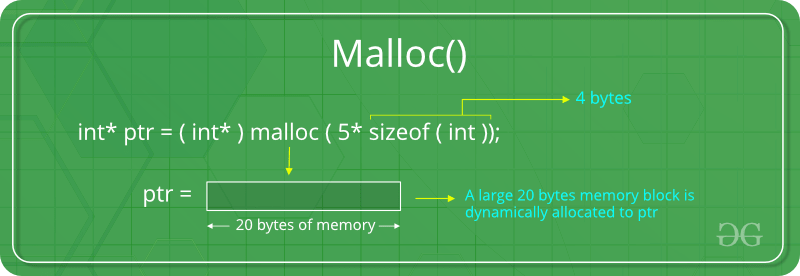
**Syntax:**

ptr = (cast-type\*) malloc(byte-size)

**For Example:**

***ptr = (int\*) malloc(100 \* sizeof(int));***

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

If space is insufficient, allocation fails and returns a NULL pointer.

calloc()

**“calloc”** or **“contiguous allocation”** method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value ‘0’.

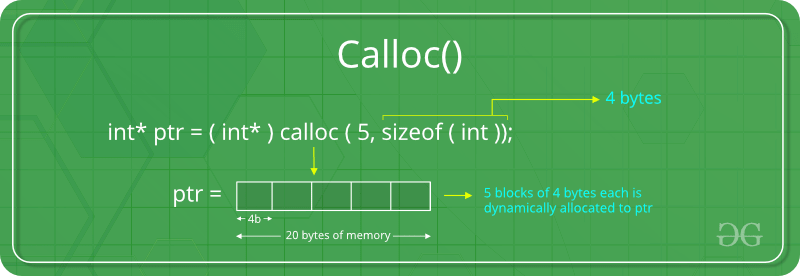
**Syntax:**

ptr = (cast-type\*)calloc(n, element-size);

**For Example:**

***ptr = (float\*) calloc(25, sizeof(float));***

*This statement allocates contiguous space in memory for 25 elements each with the size of the float.*

**

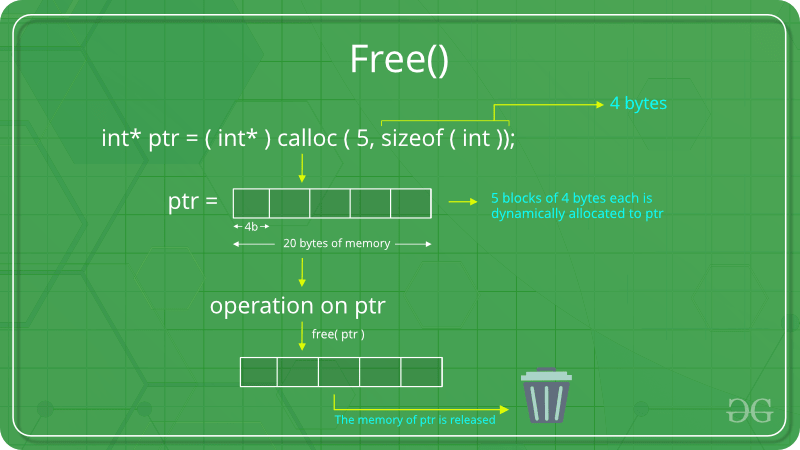
If space is insufficient, allocation fails and returns a NULL pointer.

free()

**“free”** method in C is used to dynamically **deallocate** the memory. The memory allocated using functions malloc() and calloc() is not deallocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

**Syntax:**

free(ptr);



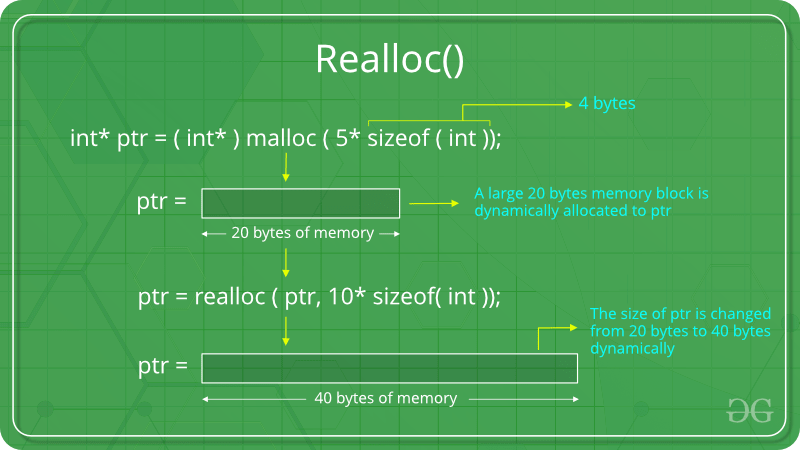
realloc()

**“realloc”** or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**.

**Syntax:**

ptr = realloc(ptr, newSize);

where ptr is reallocated with new size 'newSize'.

****

**Example: A program to read the number of students and then marks of each student and then display sum and average of marks.**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n,i;

float \*ptr,avg,sum=0;

printf("How many students are there?");

scanf("%d",&n);

ptr=(float\*)malloc(n\*sizeof(float));

printf("Enter the marks of each student:");

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

{

scanf("%f",ptr+i);

sum=sum+\*(ptr+i);

}

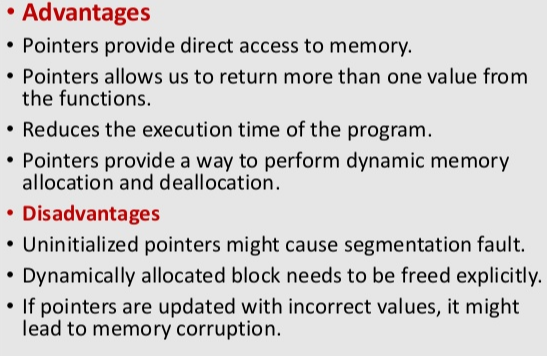
avg= (float)sum/n;

printf("Sum=%f \t Average=%f",sum,avg);

return 0;

}

**Advantage and Disadvantage of Pointer**

****