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.

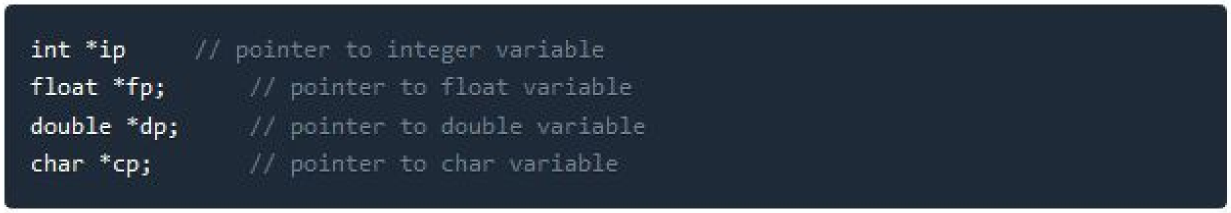
Pointers are one of the most distinct and exciting features of C language. It provides power and flexibility to the language. Although pointers may appear a little confusing and complicated in the beginning, but trust me, once you understand the concept, you will be able to do so much more with C language.

Declaration of C Pointer variable

General syntax of pointer declaration is,

datatype *pointer_name;

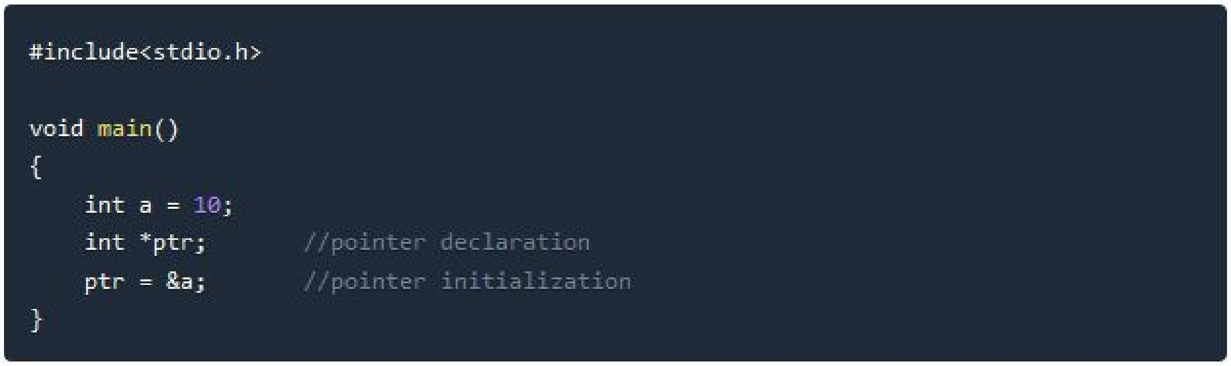
Here are a few examples:



```
int *ip     // pointer to integer variable
float *fp;      // pointer to float variable
double *dp;     // pointer to double variable
char *cp;       // pointer to char variable
```

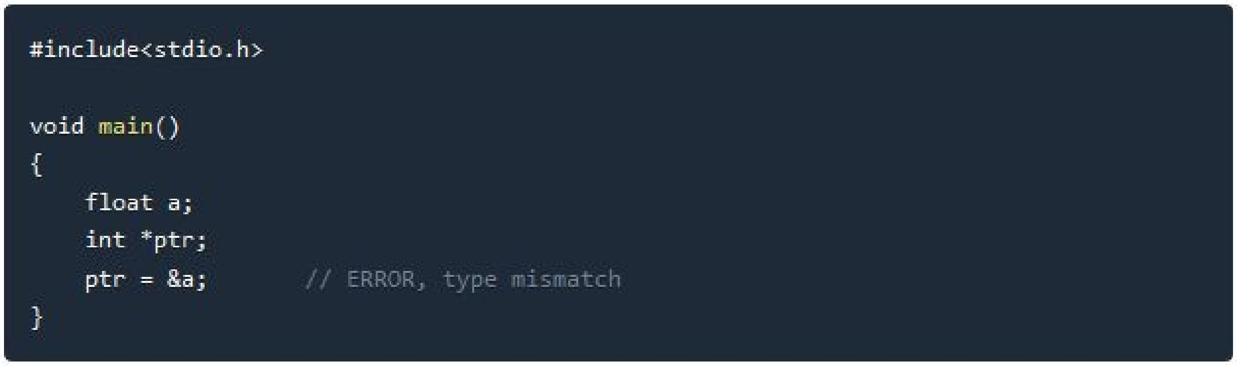
Initialization of C Pointer variable

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.



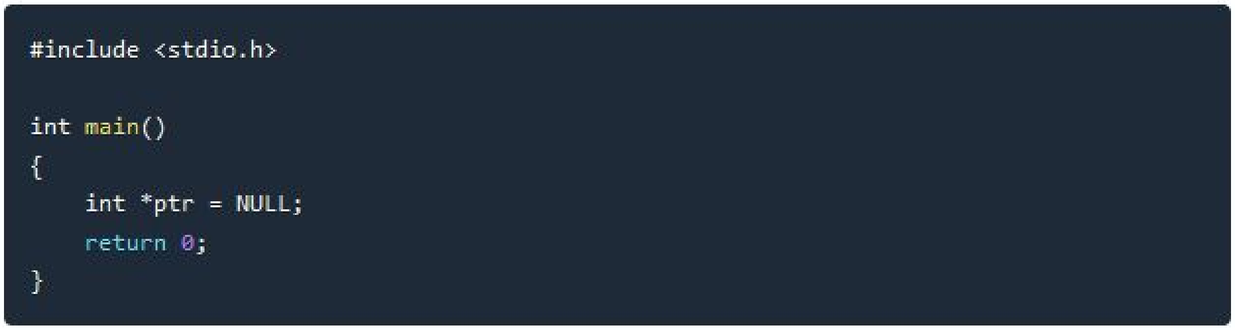
```c
#include<stdio.h>

void main()
{
    int a = 10;
    int *ptr;       //pointer declaration
    ptr = &a;       //pointer initialization
}
```

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



```c
#include<stdio.h>

void main()
{
    float a;
    int *ptr;
    ptr = &a;       // ERROR, type mismatch
}
```

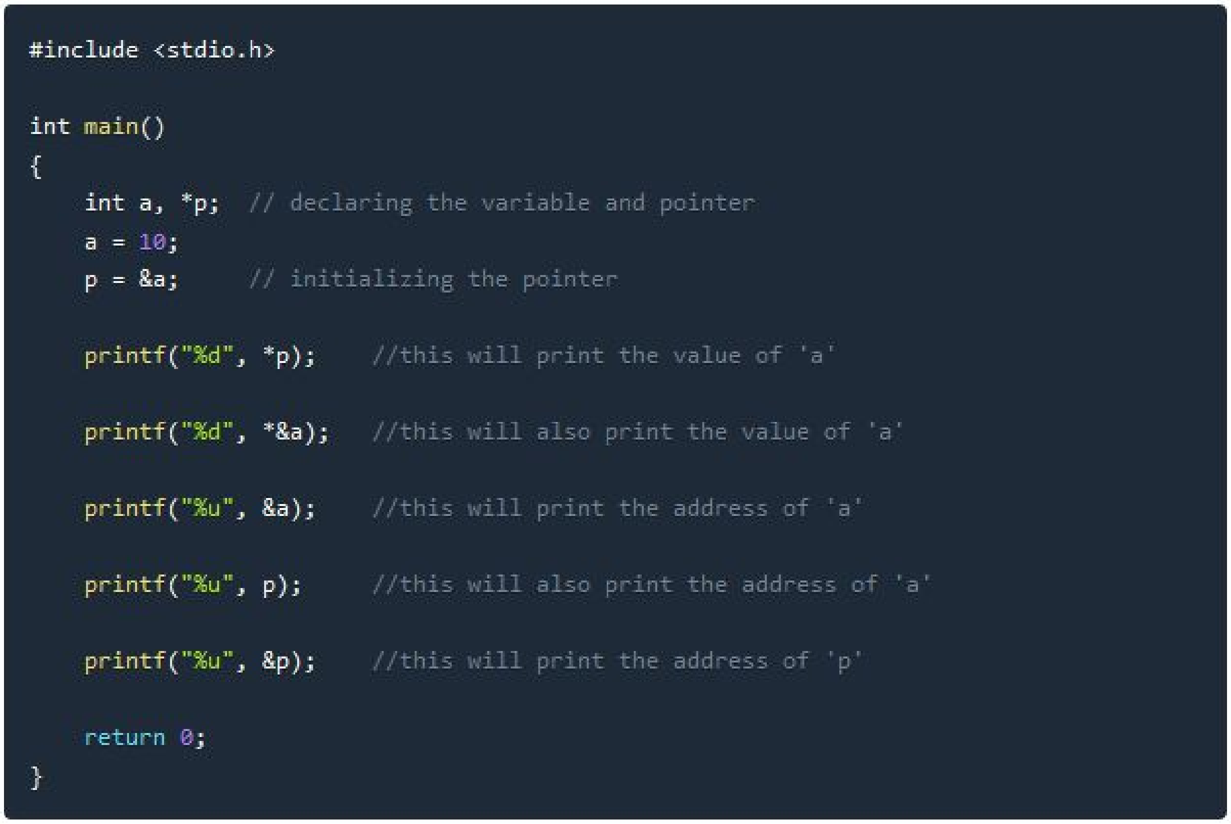
If you are not sure about which variable's address to assign to a pointer variable while declaration, it is recommended to assign a NULL value to your pointer variable. A pointer which is assigned a NULL value is called a **NULL pointer**.



```c
#include <stdio.h>

int main()
{
    int *ptr = NULL;
    return 0;
}
```

Using the pointer or Dereferencing of Pointer

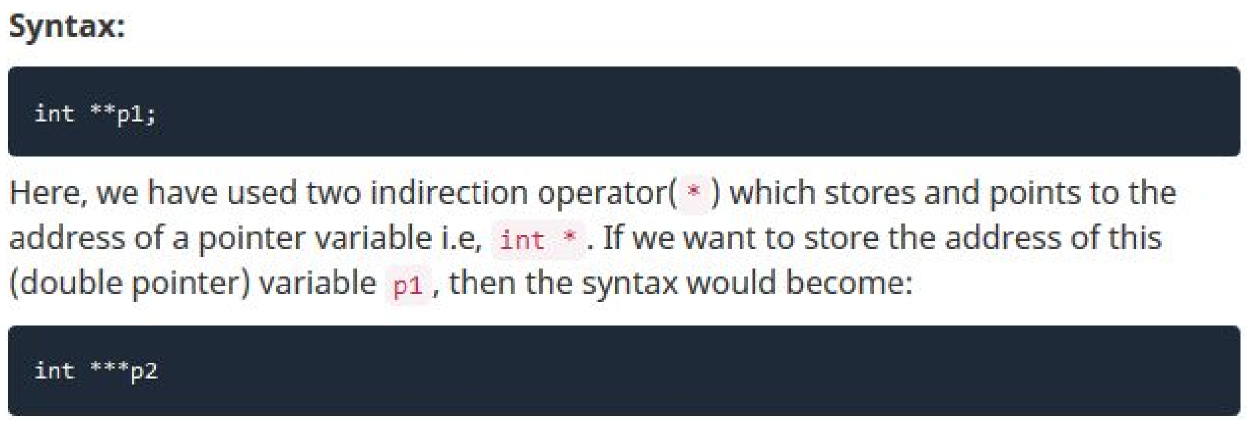
Once a pointer has been assigned the address of a variable, to access the value of the variable, pointer is **dereferenced**, using the **indirection operator** or **dereferencing operator** *.



```c
#include <stdio.h>

int main()
{
    int a, *p;  // declaring the variable and pointer
    a = 10;
    p = &a;     // initializing the pointer

    printf("%d", *p);    //this will print the value of 'a'

    printf("%d", *&a);   //this will also print the value of 'a'

    printf("%u", &a);    //this will print the address of 'a'

    printf("%u", p);     //this will also print the address of 'a'

    printf("%u", &p);    //this will print the address of 'p'

    return 0;
}
```

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**.



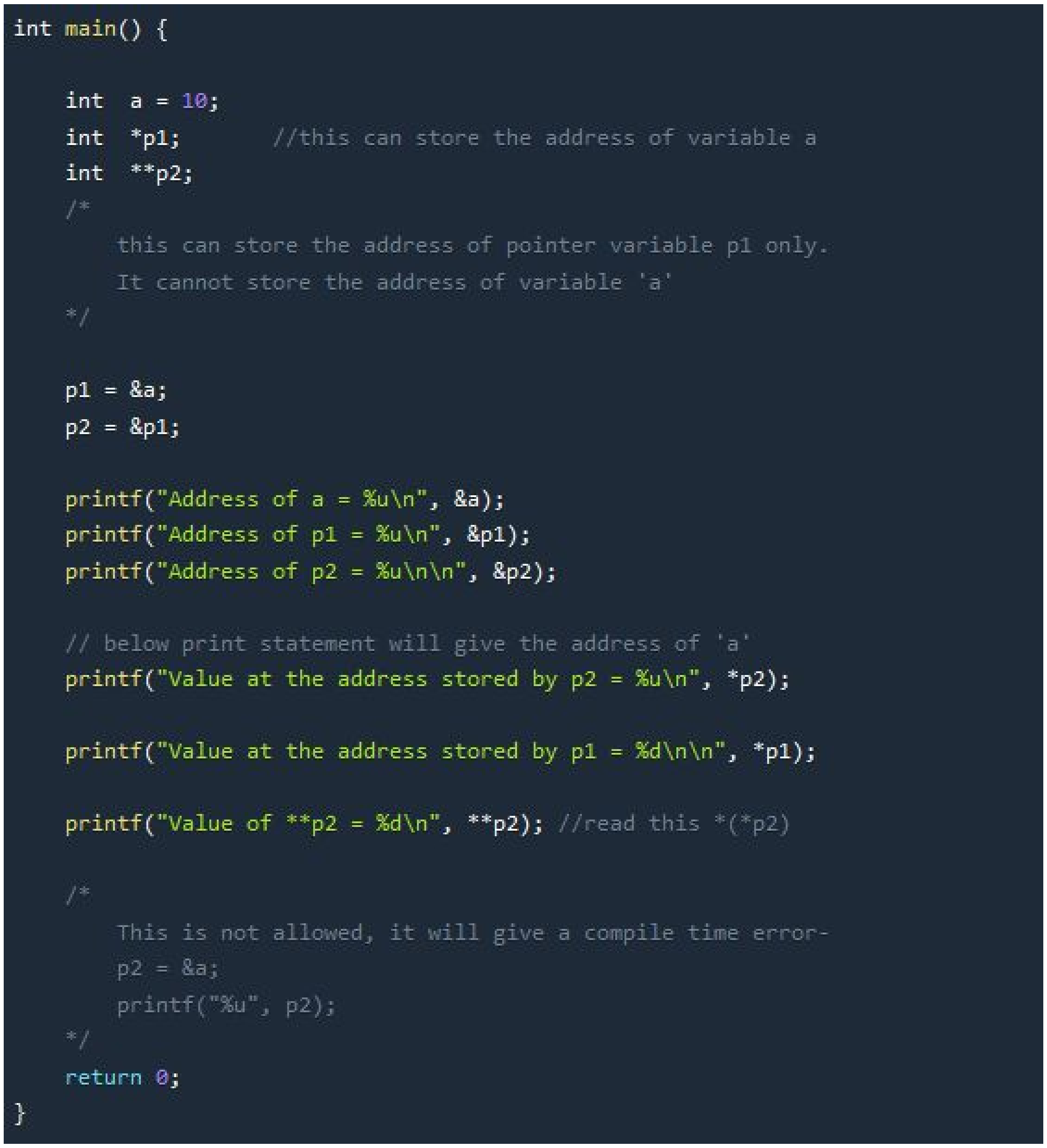
Syntax:

```
int **p1;
```

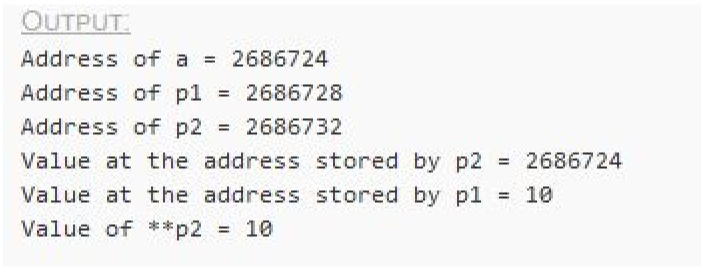
Here, we have used two indirection operator(`*`) which stores and points to the address of a pointer variable i.e, `int *`. If we want to store the address of this (double pointer) variable `p1`, then the syntax would become:

```
int ***p2
```

Example:



```c
int main() {

    int  a = 10;
    int  *p1;       //this can store the address of variable a
    int  **p2;
    /*
        this can store the address of pointer variable p1 only.
        It cannot store the address of variable 'a'
    */

    p1 = &a;
    p2 = &p1;

    printf("Address of a = %u\n", &a);
    printf("Address of p1 = %u\n", &p1);
    printf("Address of p2 = %u\n\n", &p2);

    // below print statement will give the address of 'a'
    printf("Value at the address stored by p2 = %u\n", *p2);

    printf("Value at the address stored by p1 = %d\n\n", *p1);

    printf("Value of **p2 = %d\n", **p2); //read this *(*p2)

    /*
        This is not allowed, it will give a compile time error-
        p2 = &a;
        printf("%u", p2);
    */
    return 0;
}
```



OUTPUT:

```
Address of a = 2686724
Address of p1 = 2686728
Address of p2 = 2686732
Value at the address stored by p2 = 2686724
Value at the address stored by p1 = 10
Value of **p2 = 10
```

Note: The addresses may differ...