Pointers in C

In this tutorial you will learn all about pointers in C- What pointers are, how to manipulate them in C, how useful pointers are, and different types of pointers with the help of some examples.

Addresses in C

We know that everything in programming is stored in a memory and each and every memory cell comprises two entities-- one is the **address location** and the other is the **value** stored in it.

Suppose that we declare a variable,v, of type int and initialize it with value 1.

```
int v = 1;
```

The value 1 will be stored in the address location, say here 2000, and 'v' will point to or contain the value 1. The next integer type value will be allocated in the successive memory location as given above on declaration. This indicates that each variable will have a unique address location irrespective of its values. To get the address of a variable we use the ampersand(&) prior to the variable.

```
&v;
```

Try the below example and see how v and &v is different.

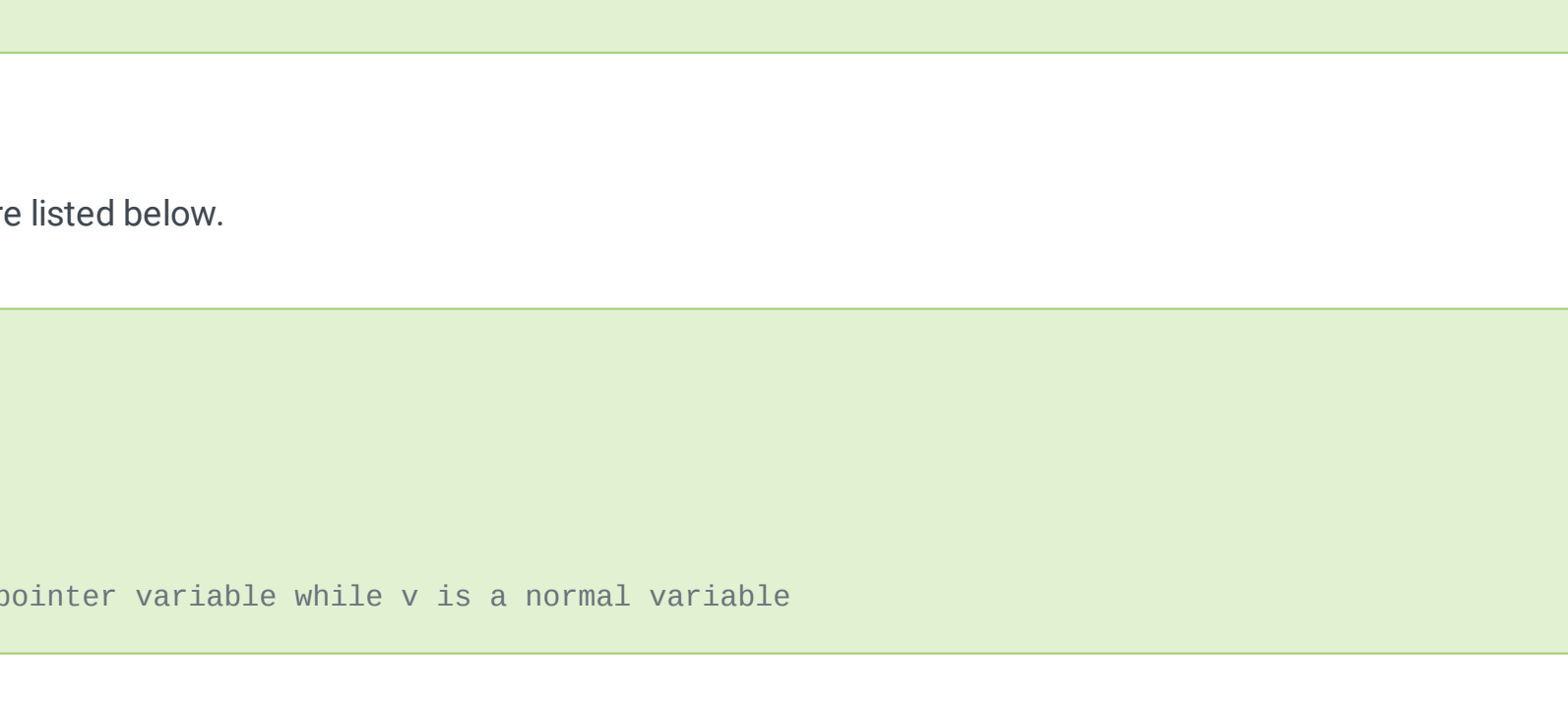
```
#include
int main()
{
  int v = 1;
  printf("variable value is : %d\n", v);

  // & to retrieve the address
  printf("address of variable is : %p", &v);
  return 0;
}
```

Output:

```
variable value is : 1
address of variable is : 000000000061FE1C
```

What are Pointers in C

A pointer is a special type of variable which stores the address location of the other cell instead of the value. Hence, in the case of pointers, we have an address location of the pointer itself and that of another cell stored in it.



How to declare pointers in c

To use pointer variables we must declare the data type of it, which should be the same as that of the value, whose address is going to be stored in it. Pointer variables are denoted by an **asterisk (*)** symbols.

```
data_type * pointer_variable_name;
```

Valid ways of declaring pointers are listed below.

```
int* p;
int * p;
int *p;

int* p1, v;   // p1 is a pointer variable while v is a normal variable
```

How to initialize pointers in c

Initialization of a pointer is just like initialization of a standard variable except the address is initialized to the pointer variable instead of value. the syntax is as follows:

```
Pointer_variable = &variable;
```

Consider the code fragment given below

```
int v=1;  //variable initialization
int *p;   //declaration of pointer variable
p=&v;     //telling the compiler to store v's address in p
```

In this example, the address of variable v is stored in the p, which is a pointer variable.

To access the value of the variable the pointer points to, we use the **asterisk(*)** symbol which is regarded as a Dereference operator while working with the pointer.

Note: The data type of variable and pointer are the same.

Pointer initialization during declaration

So far we have learned to declare and initialize a pointer in C. Now we can combine these two steps into a single one and it does not make any difference, just like what we did for standard variables. The syntax is as follows;

```
data_type * pointer_variable = &variable;
```

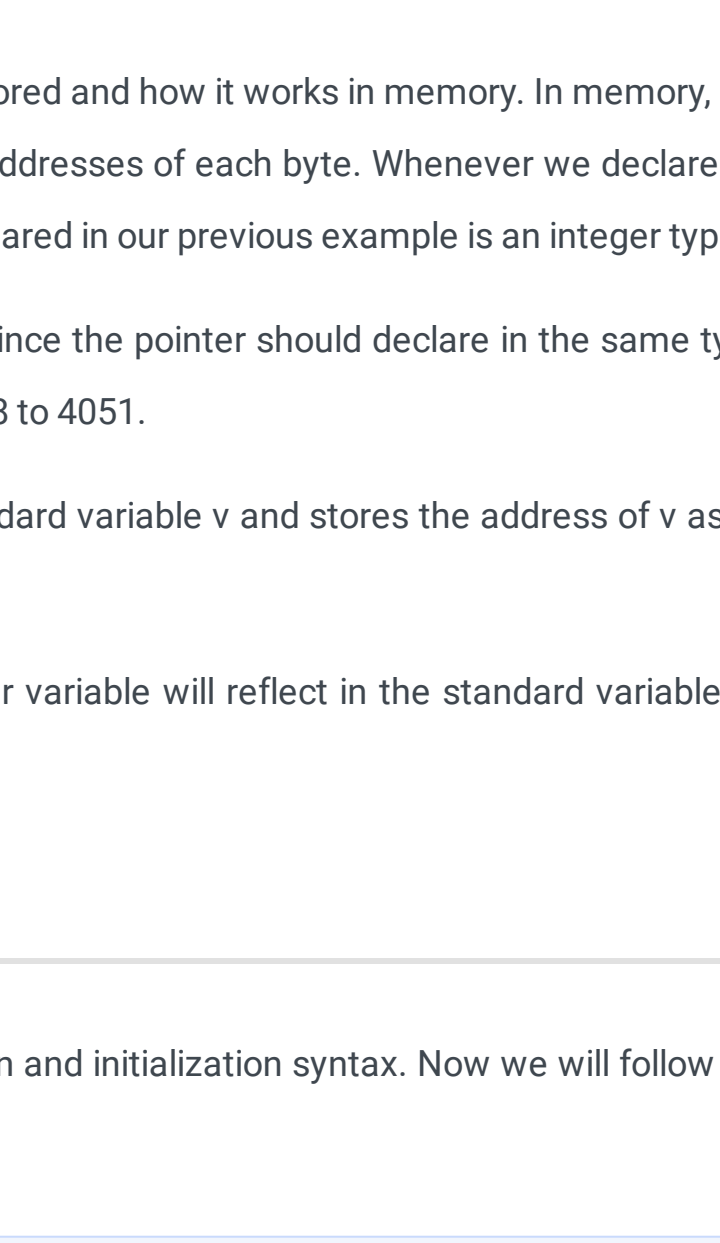
The above code fragment can be changed as :

```
int v=1;  //variable initialization
int *p = &v;  //declaration and initialization of pointer variable
```

This syntax is often confusing to beginners when you don't understand the perfect meaning of pointer. When you encounter `int *p = &v` , you may interpret that `*p` is the pointer variable but in reality, `p` is the pointer variable (* symbol helps to differentiate pointer variables from standard variables). Hence, here, a pointer variable `p` holds the address of a standard variable, `v`.

How pointers in c works

Pointer in memory can be best visualized as follows :



From this visual, we can see how variables and pointers are stored and how it works in memory. In memory, each segment or partition is 1 byte of memory and each byte has an address where we can say 2000, 2001, 2002, etc are addresses of each byte. Whenever we declare a standard variable memory is immediately allocated based on its datatype on program execution. Since the variable v declared in our previous example is an integer type it allocates 4 bytes of memory ie from 2000 to 2003.

The same happens when we declare a pointer variable too. Since the pointer should declare in the same type of variable, here pointer variable p is an integer type and allocates 4 bytes of memory in different locations i.e from 4048 to 4051.

And we can observe that the pointer variable p points to standard variable v and stores the address of v as its value. The pointer variable holds the address 2000 as its value and has its own address at 4048.

Since we work on addresses any change made by the pointer variable will reflect in the standard variable. So with the help of pointers, we can manage or work with addresses.

How to print pointers in c

We have learned all the basics of pointers including declaration and initialization syntax. Now we will follow this program to understand the printing format of pointers in C.

```
#include
int main()
{
   int v=1;    //variable declaration
   int *p;      //pointer variable declaration
   p=&v;        // p stores the address of v

   //Standard variable
   printf("---Standard Variable Details---\n");
   printf("Address of variable v is :%x\n",&v);
   printf("Value stored in v:%d\n",v);

   //Pointer Variable
   printf("\n---Pointer Variable Details---\n");
   printf("Own Address of Pointer Variable p is: %x\n",&p);
   printf("Value stored in P :%x\n",p);  //accessing the address of v
   printf("Value of v accessing with p is:%d\n",*p);//accessing the value of v


   // Changing value in variable using pointer
     *p = 50;
     printf("\n Value stored in v changed to :%d\n",v);


   return 0;
}
```

Output:

```
---Standard Variable Details---
Address of variable v is :61fe1c
Value stored in v:1

---Pointer Variable Details---
Own Address of Pointer Variable p  is: 61fe10

Value stored in P :61fe1c

Value of v accessing with p is:1
```

In this program, the address of v and value stored in p is the same which implies the pointer variable always holds the address of the cell it points to.

Note: '*' implies ' value at' and '&' implies 'address of'

What are pointers in c used for

Pointers are not an essential part of a program whereas it is a powerful feature supported by c to closely interact with hardware. Since C is a low-level language it is mostly machine-dependent and works with memory a lot. Hence pointers are an efficient way to work with memory as it provides the following features.

1. The first and foremost advantage of using pointers is that pointers save memory space to a huge extent by not creating a local copy of variables or functions that are passed.
2. Pointers allow dynamic allocation of memory. To be specific at any time of process based on demand you can create and destroy space in memory.
3. With the use of pointers, one can refer to the same space in memory from numerous locations. Any changes made to one location will reflect in all other locations too.
4. Since pointers have direct access to memory(address), execution time with pointers is faster.
5. Pointers allow call by reference which enables a function to alter the value of a variable in another function.

These are the main features that make pointers an inevitable part of the C language. Now let's see where the pointers are used often.

- Arrays use pointers for navigating effectively, also useful for representing 2D and multi-dimensional arrays.
- Data structures like linked lists and trees use pointers for memory allocation and deallocation.
- Pointers make file handling hassle-free.
- Since Pointers interact well with hardware, it is a vital part of embedded systems.
- Pointers are highly efficient in memory management.

Pointers are not limited to C or C++ but are used in most of all high-level languages implicitly for backend interactions. In C and C++, pointers are defined explicitly.

What is a pointer to pointer?

Now we are pretty clear that a pointer stores the address of a variable it points to. But it has its own address too. In c pointers can also be used to store the address of another pointer. This type of pointer is referred to as pointer to pointer (double pointer) in C.

This can be viewed as follows :

Here, the first pointer, ptr1, stores the address of the variable, var, while the second pointer, ptr2, holds the address of the first pointer (ptr1). ptr2 is the double-pointer. Declaration syntax of the double-pointer is similar to the declaration of pointer except for the requirement of an additional asterisk symbol.

```
datatype **pointer_variable;
```

Following is the example which illustrates the meaning of a double-pointer.

```
//double pointer
#include
void main ()
{
    int var = 1;
    int *ptr1;
    int **ptr2;
    ptr1 = &var; // pointer ptr1 is pointing to the address of var
    ptr2 = &ptr1; // pointer ptr2 is a double pointer pointing to the address of pointer ptr1
    printf("Address of var: %x\n",ptr1); // Address of variable will be printed
    printf("Address of ptr1: %x\n",ptr2); // Address of ptr1 will be printed
    printf("value stored at ptr1: %d\n",*ptr1); // value stored at the address contained by ptr
    printf("value stored at ptr2: %d\n",**ptr2); // value stored at the address contained by the pointer stored at ptr2

    printf("address of ptr2: %x\n",&ptr2); // Address of ptr2 will be printed
}
```

Output:

```
Address of var: 61fe1c
Address of ptr1: 61fe10
value stored at ptr1: 1
value stored at ptr2: 1
address of ptr2: 61fe08
```

When to use double pointers in c

To understand when to use double pointers in C a little explanation is needed. In a single statement, we can define the use of Double pointer as :

Double pointers are used in cases where you want to change a pointer which is passed as an argument in the function.

Suppose that you have a pointer `p1` with the address of a variable `v1`, as its value, and also you have another pointer `p2` with the address of `v2`. Now you wish to change the address of `p1` with `p2`. Is it possible to change the address stored in two pointers? Of Course possible, you can achieve this by assigning `p1= p2`, meaning the address of `p2` is now stored in `p1`. You can witness this in the following example:

```
#include

int main()
{

    int v1 = 10;
    int v2 = 20;

    int * p1 = &v1;
    int * p2 = &v2;

    printf(" p1's value: %x \n", p1);
    printf(" p2's value: %x \n", p2);

    printf("\n\n Can we change values of two pointers , say p1 and p2 ?");
    p1 = p2;
    printf("\n Value of p1 changed to: %x, same as 'p2' ", p1);
}
```

Output:

```
p1's value: 61fe04
p2's value: 61fe00

Can we change values of two pointers , say p1 and p2 ?
Value of p1 changed to: 61fe00, same as 'p2'
```

Now we will see whether the **same can be done with function** and what changes will reflect in the program. Yes, it can be done with the function, but the only thing matters is will the change made to pointers exist even after exiting the function. This can be spotted by examining the below given example:

```
#include

int main()
{
    int v1 = 10;
    int v2 = 20;

    int * p1 = &v1;
    int * p2 = &v2;

    printf(" p1's value: %x \n", p1);
    printf(" p2's value: %x \n", p2);

   printf("\n\n Can we change values of two pointers using function? \n");
   change_fun(p1, p2);
   printf("\n Outside Function : Value of 'p1' is : %x, same as 'p1' not 'p2'\n", p1);
  return 0;
}

void change_fun(int * x, int * z){
    x = z;
    printf(" Inside Function : Value of 'p1' is: %x  same as 'p2'", x);
}
```

Output:

```
p1's value: 61fe0c
p2's value: 61fe08

Can we change values of two pointers using function?
Inside Function : Value of 'p1' is: 61fe08  same as 'p2'
Outside Function : Value of 'p1' is : 61fe0c, same as 'p1' not 'p2'
```

In this program, the value of the pointer changes from `p1` to `p2` inside the function, and that change exists within the function. Outside the function value of the pointer, `p1` remains the same regardless of the change. This indicates that any change made to a pointer inside a function exists locally to that function.

This will cause errors and is not a fair practice in programming. So to solve this issue we can make **use of double pointers**. Let's observe the program and understand the tactics.

```
#include

int main()
{
    int v1 = 10;
    int v2 = 20;

    int * p1 = &v1;
    int * p2 = &v2;
    int ** pp = &p1;  // pointer to pointer 'v1'

    printf(" p1's value: %x \n", p1);
    printf(" p2's value: %x \n", p2);

    printf("\n\n Can we change values of two pointers with the help of double pointers? \n");
    change_doublepointer(pp, p2);
   printf("\n Outside Function : Value of 'p1' is : %x, same as  'p2'\n", p1);
  return 0;
}
void change_doublepointer(int ** x, int * z){
    *x = z;
    printf(" Inside Function: Value of 'p1' is: %x , same as 'p2', \n", *x);

}
```