



Using Pointers in C

Pointers seem to confuse everybody, let's get our heads around these incredibly useful elements in the C language.

How pointers work, how to use them, and how to understand them.

James Walmsley
2/1/2007

Introduction

Pointers seem to cause the most brain strain more than any other element of the C programming language. I believe that the word pointer causes a lot of confusion as to what they actually do and how they work.

Over the past few weeks I've seen dozens if not hundreds of incorrect attempts of pointer use. Even from some experienced programmers.

So what is a pointer?

In simple terms, a pointer is simply an alias to a piece of memory already defined in another function.

Pointers play a slightly different role when it comes to an array, although these are quite similar devices of C.

Remember when thinking of pointers and how to implement them, replace the word pointer with alias.

Why use pointers?

One of the main problems with the mis-understanding of pointers, is because a lot of people don't fully understand why we need to use them. Or when it is appropriate to do so.

For example, we have a function, in general we can pass it a number of arguments and it can return a **SINGLE** answer.

```
int add(int a, int b)
{
    return a + b;
}

void main(void)
{
    int answer;
    answer = add(1,2);    /* Answer becomes 3 */
}
```

Ok, so the above example is very straight forward and easy to understand. However what happens if we need a single function to do lots of things? **If we need to return more than a single answer or value, then we use a pointer.**

The above example can be re-written using pointers as follows:

```
void add(int *answer, int a, int b)
{
    answer = a + b;
}

void main(void)
{
    int answer;
    add(&answer, 1, 2);    /* Answer becomes 3 */
}
```

We define a pointer in a function as:

type *name e.g:

int *answer, char *name, etc etc.

We can reference any kind of variable or memory like this. To set the alias (pointer) we simply pass it the address of a variable we want it to modify.

In the above example, this is done by passing &answer, which means the address of answer, to the function.

For example:

A function called foo defines a pointer...

```
void foo(int *mypointer) { }
```

This is passed the address of an integer called original:

```
int original;
foo(&original);
```

Whenever in function foo we want to modify the **VALUE** or **DATA** stored in original, we simple reference the name mypointer. So to set original to = 3 in foo() :

```
void foo(int *mypointer) {
    mypointer = 3;
}
```

NOTE: there is no need use the * in the name, just mypointer this modifies the value.

Now when the function foo finishes all its routines, the number 3 is left in original.

This is the basic use of the pointer. We can return as many values as we like back to main in this way.

Quick Info (I'll be more complete in these areas soon).

Once the pointer is initialised we can change the address it's pointing to, (change which variable it references) by simply assigning a new address.

```
*mypointer = &newint;
```

To pass the address of the original integer into another function we'd simply send mypointer (without the &).

So e.g. (inside foo)

```
Foo(){  
  
    Otherfunction(mypointer);  
  
}
```

Otherfunction receives the address of original and can modify it too.

[I shall continue my guide on pointer use later.]