C For Dummies Blog

The *_Generic* Keyword

Posted on **May 8, 2021**

The C11 standard added the “underscore” bunch to the C language’s traditional keywords:

_Alignas
_Alingof
_Atomic
_Bool
_Complex
_Generic
_Imaginary
_Noreturn
_Static_assert
_Thread_local

I don’t routinely use any of these in my programs, beyond trying a few out to see how they work. The *_Bool* keyword comes in handy. The rest? Well, they’re worth exploring from a curiosity standpoint. For this week’s Lesson, I reveal the mysteries of the *_Generic* keyword.

It would be neat if computer scientists just threw in these new keywords for fun. They’d say, “This new underscore word will really drive the coders nuts!” But no, these keywords have their roots in problems that were originally solved in other ways. This tale tells true for the *_Generic* keyword, which has its roots in C’s various high-level math functions.

For example, different math functions are required for specific data types. You may not be aware of these differences because the math function is really a macro. Internally, the macro calls a specific function based on the data type used as an argument. This process explains how the *_Generic* keyword came about; its job is to help determine a data type.

Here is the format for *_Generic*, which behaves more like a function than a keyword:

```
r = _Generic( expression, list... );
```

r is an *int* value generated by the *_Generic* keyword.

expression is the variable or constant to be evaluated.

list is a menu of options. It works similar to *case* statements in a *switch-case* structure. The *list* presents a series of data types followed by a colon and an expression. These items are separated by commas, and a *default* item is present to handle no-matches:

```
_Generic( var, char: 1, int: 2, long: 3, default: 0 );
```

Above, assume the data type for `var` is *int*. Therefore, the value 2 is generated by the *_Generic* keyword.

Here's another, more practical example from a math function standpoint:

```
#define cbrt(X) _Generic((X), long double: cbrtl, float: cbrtf,
default: cbrt )
```

In this declaration, the `cbrt()` macro can evaluate to either the *cbrtl()*, *cbrtf()* or *cbrt()* function depending on the data type of argument `X`. In this manner, *_Generic* helps the code to implement the proper cubed root function based on the argument's data type.

Because I'm mathematically disinclined, here is some sample code you can play with:

2021_05_08-LESSON.C

```
#include <stdio.h>

int main()
{
    char a;
    int r;

    r = _Generic( a, char: 1, int: 2, long: 3, default: 0 );
    printf("'a' is ");
    switch(r)
    {
        case 1:
            puts("a char");
            break;
        case 2:
            puts("an int");
            break;
        case 3:
            puts("a long");
            break;
        default:
            puts("unknown");
    }

    return(0);
}
```

At Line 8, the *_Generic* keyword evaluates the data type of variable `a`. In this case, it's a *char*, so *int* value 1 is returned. This value is saved in variable `r` and used in the

switch-case structure starting at Line 10. Here is the output:

```
'a' is a char
```

You can change the declaration of variable `a` at Line 5 to *int*, *long*, and so on to see how output is affected. If you change the data type to *float* or a pointer, the *default* value of 0 is generated.

This Lesson may not convince you to use *_Generic* in your code, but I hope you appreciate its worth. Perhaps you can work it in somehow, but like the most of the underscore keywords, it remains a curiosity.

This entry was posted in **Lesson** by **dgookin**. Bookmark the **permalink [https://c-for-dummies.com/blog/?p=4746]** .