5.4.6 Constant Types and Formats

A constant is used to represent an immediate value in the source code, as opposed to a variable that could hold the same value. For example 123 is a constant.

Like any value, a constant must have a C type. In addition to a constant's type, the actual value can be specified in one of several formats.

5.4.6.1 INTEGRAL CONSTANTS

The format of integral constants specifies their radix. MPLAB XC8 supports the ANSI standard radix specifiers, as well as ones which enables binary constants to be specified in C code.

The formats used to specify the radices are given in Table 5-7. The letters used to specify binary or hexadecimal radices are case insensitive, as are the letters used to specify the hexadecimal digits.

TABLE 5-7: RADIX FORMATS

Radix	Format	Example
binary	`0b` *number* or `0B` *number*	0b10011010
octal	`0` *number*	0763
decimal	*number*	129
hexadecimal	`0x` *number* or `0X` *number*	0x2F

Any integral constant will have a type of `int`, `long int` or `long long int`, so that the type can hold the value without overflow. Constants specified in octal or hexadecimal can also be assigned a type of `unsigned int`, `unsigned long int` or `unsigned long long int` if the signed counterparts are too small to hold the value.

The default types of constants can be changed by the addition of a suffix after the digits; e.g., `23U`, where `U` is the suffix. Table 5-8 shows the possible combination of suffixes and the types that are considered when assigning a type. So, for example, if the suffix `l` is specified and the value is a decimal constant, the compiler will assign the type `long int`, if that type will hold the constant; otherwise, it will assigned `long long int`. If the constant was specified as an octal or hexadecimal constant, then unsigned types are also considered.

TABLE 5-8: SUFFIXES AND ASSIGNED TYPES

Suffix	Decimal	Octal or Hexadecimal

`u` or `U`	`unsigned int`
	`unsigned long int`
	`unsigned long long int`
		`unsigned int`
		`unsigned long int`
		`unsigned long long int`

`l` or `L`	`long int`
	`long long int`
		`long int`
		`unsigned long int`
		`long long int`
		`unsigned long long int`

`u` or `U`, and `l` or `L`	`unsigned long int` `unsigned long long int`	`unsigned long int` `unsigned long long int`
`ll` or `LL`	`long long int`	`long long int` `unsigned long long int`
`u` or `U`, and `ll` or `LL`	`unsigned long long int`	`unsigned long long int`

Here is an example of code that can fail because the default type assigned to a constant is not appropriate:

```
unsigned long int result;
unsigned char shifter;

void main(void)
{
    shifter = 20;
    result = 1 << shifter;
    // code that uses result
}
```

The constant `1` (one) will be assigned an `int` type, hence the result of the shift operation will be an `int`. Even though this result is assigned to the `long` variable, `result`, it can never become larger than the size of an `int`, regardless of how much the constant is shifted. In this case, the value 1 shifted left 20 bits will yield the result 0, not 0x100000.

The following uses a suffix to change the type of the constant, hence ensure the shift result has an `unsigned long` type.

```
    result = 1UL << shifter;
```

5.4.6.2 FLOATING-POINT CONSTANT

Floating-point constants have `double` type unless suffixed by `f` or `F`, in which case it is a `float` constant. The suffixes `l` or `L` specify a `long double` type which is considered an identical type to `double` by MPLAB XC8.

5.4.6.3 CHARACTER AND STRING CONSTANTS

Character constants are enclosed by single quote characters, `'`, for example `'a'`. A character constant has `int` type, although this can be later optimized to a `char` type by the compiler.

To comply with the ANSI C standard, the compiler does not support the extended character set in characters or character arrays. Instead, they need to be escaped using the backslash character, as in the following example.

```
const char name[] = "Bj\370rk";
printf("%s's Resum\351", name);   \\ prints "Bjørk's Resumé"
```

Multi-byte character constants are not supported by this implementation.

String constants, or string literals, are enclosed by double quote characters “, for example “`hello world`”. The type of string constants is `const char *` and the character that make up the string are stored in the program memory, as are all objects qualified `const`.

A common warning relates to assigning a string literal to a pointer that does not specify a `const` target, for example:

```
char * cp = "hello world\n";
```

The string characters cannot be modified, but this type of pointer allows writes to take place, hence the warning. To prevent yourself from trying to overwrite the string, qualifier the pointer target as follows. See also **Section 5.4.5.1 “Combining Type Qualifiers and Pointers”**.

```
const char * cp = "hello world\n";
```