Be Careful Using Chars

Ashwin Menon

December 15, 2024

You might think that the *char* would be the perfect data type to represent byte sized variables, especially in an embedded system where storage space is at a premium. However, there are serious pitfalls with this, not least of which is the fact that it is implementation defined whether a *char* is a signed or an unsigned value. Consider this:

```
char c = 255; // Is c signed or unsigned?
int i  = c;   // Is i -1 or 255?
```

The moment you try to store the *char* in a larger numeric value, then the larger value becomes implementation defined. You might think you could get around this by using *unsigned char* or *signed char* variables to make the signedness explicit, and therefore under your control. But that has pitfalls as well. Consider this:

```
char ch;
signed char schar;
unsigned char uchar;

char *p_char = &uchar;           // error
signed char *p_schar = p_char;   // error
unsigned char *p_uchar = p_char; // error
p_schar = p_uchar;               // error
```

All four pointer assignments are illegal in C++ and will be flagged as invalid conversions. This makes it a problem when you try to pass a *signed char* or an *unsigned char* pointer to any of the standard library string functions, all of which take *char** arguments. Consider:

```
const char *str = "hello";
const signed char *s_str = "goodbye"; // error
const unsigned char *u_str = "ringo"; // error

int i = strlen(str);
int j = strlen(s_str); // error
int k = strlen(u_str); // error
```

The second and third string assignments are errors, because you can't convert a *char** to a *signed char** or an *unsigned char**. For the same reason (working in reverse), the second and third calls to *strlen* are errors, since *strlen* only accepts *char** arguments.

You can avoid all this bother by simply using *char* everywhere instead of the *signed* or *unsigned* variant, and by also making sure that you never store negative or too large values in these *char*'s. This makes it perfectly fine to use *char*'s to store ASCII strings, since ASCII characters use only the lower 127 values, and therefore can never be sign extended to negative values.

If you want to store 8-bit numeric values that don't represent ASCII characters (i.e. they really are numbers) then use one of the standard integer types, like *uint8_t* or *int8_t*.