On leading underscores and names reserved by the C and C++ languages

devblogs.microsoft.com/oldnewthing/20230109-00

January 9, 2023

Raymond Chen

The C and C++ languages reserve certain categories of names for the implementation, which means that you cannot use them in your own code. Some are reserved unconditionally, precluding their use for for variable names, parameter names, classes, methods, macros, whatever. Others are reserved only in certain contexts.

The rules for C++ are collected in the **[lex.name]** chapter. The rules for C happen to match the C++ rules for the most part, (section 7.1.3 “Reserved identifiers“), so that makes things easier to remember.

Pattern	Conditions
Begins with two underscores	Reserved
Begins with underscore and uppercase letter	Reserved
Begins with underscore and something else	Reserved in global scope (includes macros)
Contains two consecutive underscores	Reserved in C++ (but okay in C)

Note that a popular convention of prefixing private members with an underscore runs afoul of these rules if the member name begins with an uppercase letter.

```
class Widget
{
public:
    Widget();

private:
    int _size; // okay
    void _Toggle(); // not okay
};
```

The C language does not have namespaces, so it also must reserve names in the global namespace for future expansion. Some names may not be used by symbols with external linkage. You can use them for type names, enumeration members, local variables, and functions or global variables declared with static storage class, but not for extern functions or extern global variables.

```
// Not allowed: Identifier with external linkage
// beginning with "str" and a lowercase letter.
int strategy;
void strafe() { /* ... */ }

// Allowed: Identifier with internal linkage beginning
// with "str" and a lowercase letter.
static int strawberry;
static void stream_video() { /* ... */ }
```

Furthermore, if you include the corresponding header file, the names are permitted to be shadowed by function-like macros. This means that if you intend to use a reserved name for someting without external linkage, you must first `#undef` it or enclose the name in parentheses to prevent it from being treated as a macro.

```
// Including this header may result in the definition
// of a function-like macro named "strategy".
#include <string.h>

// Must enclose in parentheses to prevent misinterpretation
// as function-like macro.
static void (strategy)();
```

As of C11, identifiers matching the following regular expressions may not be used for symbols with external linkage. (The list is given in section 7.31: “Future library directions”.) There are also reserved names for type definitions and macros, but I won’t list them here.

Pattern	**Header**
`cerfc?[fl]?`, `cexp2[fl]?`, `cexpm1[fl]?`, `clog1[0p][fl]?`, `clog2[fl]?`, `c[lt]gamma[fl]?`	`complex.h`
`is[a-z].*`, `to[a-z].*`	`ctype.h`, `wctype.h`
`atomic_[a-z].*`	`stdatomic.h`
`str[a-z].*`	`stdlib.h`, `string.h`
`mem[a-z].*`	`string.h`
`wcs[a-z].*`	`string.h`, `wchar.h`

`cnd_[a-z].*`, `mtx_[a-z].*`, `thrd_[a-z].*`, `tss_[a-z].*`	`thread.h`

It may come as a surprise that the C language reserves identifiers like `strong`, `island`, and `together`, but it does.

Bonus chatter: Windows header files have historically not been conscientious about avoiding these reserved names. We're trying to do better for new headers, but not everyone has gotten the memo.

Update: Added special "internal double-underscore" rule.