C9X Revision Proposal

Title: Addition of predefined identifier `__func`.

Author:	Dave Mooney	David R. Tribble
Author Affiliation:	IBM	(self)
Postal Address:	1150 Eglinton Ave E North York ON M3C 1H7 CANADA	6004 Cave River Dr. Plano, TX 75093-6951 USA
E-mail Address:	dmm@vnet.ibm.com	drt@wcwcen.wf.com
Telephone Number:	+1 416 448 4474	+1 214 9641720
Fax Number:	+1 416 448 4414	(None)

Sponsor: Dave Mooney
Date: 1996-09-12
Document History: N419 was the Stage 1 proposal. N611 brings this to Stage 2.
Proposal Category:

- __ Editorial change/non-normative contribution
- __ Correction
- X_ New feature
- __ Addition to obsolescent feature list
- __ Addition to Future Directions
- __ Other (please specify) ______________________________

Area of Standard Affected:

- __ Environment
- X_ Language
- __ Preprocessor
- __ Library
  - __ Macro/typedef/tag name
  - __ Function
  - __ Header
- __ Other (please specify) ______________________________

Prior Art: I think this was implemented by Aztec C some years ago. GNU C and IBM VisualAge C++ currently use `__FUNCTION__` and Metaware uses `__function()`.
Target Audience: Programmers using debugging statements.
Related Documents (if any): (None)
Proposal Attached: X_ Yes __ No, but what's your interest?

Abstract: The use of the predefined identifier `__func` enables programmers to specify the name of the enclosing function in debugging statements. It also may be used by the `assert()` macro.

Following are the changes to Draft 6 ofC9X needed to reflect the ideas contained in N419.

(Add the following subclause.)

6.1.2.7 Predefined identifiers

[1] The following identifier shall be defined by the application:

`__func` The name of the lexically-enclosing function.

Forward references: The identifier `__func` (6.3.1.1).

(Add the following subclause.)

6.3.1.1 The identifier __func

Semantics

[1] The identifier `__func` is implicitly declared by the translator as if, immediately following the opening brace of each function definition, the declaration

```
static char __func[] = “function-name”;
```

appeared, where *function-name* is the name of the lexically-enclosing function [*]. This name is the unadorned name of the function, without extraneous linkage characters (such as leading underscores) or trailing parentheses.

[2] This name is encoded as if the implicit declaration had been written in the source character set and then translated into the execution character set as indicated in translation phase 5.

[3] If the program attempts to modify the contents of the array `__func[]`, the behaviour is undefined.

Example

[4] Consider the code fragment:

```
#include <stdio.h>
void spuzzum(void)
{
    printf(“%s\n”, __func);
    // ...
}
```

Each time this function is called, it will print to the standard output stream:

```
spuzzum
```

Footnote

[*] Note that since the name `__func` is reserved for any use by the implementation (7.1.3), if any other identifier is explicitly declared using the name `__func`, the behaviour is undefined.

7.2.1.1 The `assert` macro

(Replace the current paragraph [3] with:)

[3] The `assert` macro puts diagnostics into programs. When it is executed, if `expression` is false (this is, compares equal to 0), the `assert` macro writes information about the particular call that failed (including the text of the argument, the name of the source file, the source line number, and the name of the function containing the call -- the latter are respectively the values of the preprocessing macros `__FILE__` and `__LINE__` and the identifier `__func`) on the standard error file in an implementation-defined format. [*] It then calls the `abort` function.

Footnote

[*] The message written might be of the form

Assertion failed: "*expression*", function *abc*, file *xyz*, line *nnn*.

Rationale

Programmers who use debugging statements often use the `__LINE__` and `__FILE__` predefined macro names ([6.8.8]). However, displaying the name of the function that contains a given debugging statement is not as easy. Programmers often have to add explicit strings to their debugging statements (which are typically calls to `printf()`). This leads to clutter and wasted data space. It also leads to confusion if function names are changed or function code is duplicated, and the programmer, in haste, forgets to change the affected debugging statements.

A number of vendors have added a predefined macro with a name such as `__FUNCTION__` into their implementations that programmers can use to refer to the name of the current function. Using such a macro provides certain benefits, such as the ability to use the text in string-pasting, but since it forces the preprocessor (in translation phase 4) to have knowledge of the lexically-current function being translated (which is not known until translation phase 8), it was not feasible to add the feature to the Standard in that particular form.

The Committee considered a couple of other implementations of `__func`, such as adding a new keyword, but eventually decided that defining `__func` implicitly inside each function body was the technique that would cause the smallest number of changes to the current Standard and was closest in approach to other similar features already in the language.

In particular, the way that `__func` is implicitly declared parallels the way that functions are implicitly declared if they are called without a prototype visible. The fact that undefined behaviour is invoked if the `__func[]` array is written to allows the implementation to treat `__func` as if it were a string constant (by, for example, putting the array in read-only storage).

Declaring `__func` as an array that is declared once at the beginning of each function rather than a magic identifier that is replaced by a string each time it is seen guarantees that the relation

```
__func == __func
```

is always true.

The Committee considered whether `__func` should be well-defined at file scope or in the parameter list of a function definition, but found that the added functionality that this would bring would be outweighed by the additional complexity in its specification; consequently `__func` is defined only inside function bodies.