Variadic Macros and Placemarkers

Paul Mensonides

Document number: N1545
Date: 29 October 2003
Project: Programming Language C++, Evolution Working Group
Reply-to: Paul Mensonides <pmenso57@comcast.net>

Abstract

Two facilities were added to the macro expansion mechanism of the C preprocessor as of C99. These facilities are *variadic macro parameters* and *placemarker preprocessing tokens* (ISO/IEC 9899 - 6.10). They are outstanding incompatibilities with C++ and have already been implemented in the preprocessors of many compilers (including, but not limited to, EDG, IBM, Metrowerks, and GCC). Use of these facilities can result in a significant reduction in the number of macros in modern libraries such as Boost. This paper proposes that these facilities be adopted by C++ exactly as defined by C99.

Placemarker preprocessing tokens

Placemarker preprocessing tokens (a.k.a. placemarkers) are a notion used to specify how macro arguments that consist of no preprocessing tokens should be handled.

```
#define A(x) x

A() // okay, expands to nothing
```

In current C++, an empty macro argument constitutes undefined behavior (16.3/10). This facility should be adopted by C++ for compatibility with C and because it fixes an unintuitive blemish in C++.

Variadic macros

Variadic macros are function-like macros that can invoked with an arbitrary number of arguments. They are similar to variadic functions in the core language.

```
#define B(...) __VA_ARGS__

B(1, 2, 3) // 1, 2, 3
```

The primary rationale for the adoption of variadic macros in C++ is compatibility with C in an area that breaks no currently conforming code. They are also useful in C++ because of the existence of types that contain commas that are no enclosed in parentheses.

```
#define C(x) x

C( std::pair<int, int> ) // error

#define D(...) __VA_ARGS__

D( std::pair<int, int> ) // std::pair<int, int>
```