Inlining Proposal
WG14/N700 (X3J11/97-063)

Bill Homer
Tom MacDonald

Cray Research, Inc.
655F Lone Oak Drive
Eagan, MN 55121
homer@cray.com
tam@cray.com

21 May 1997

Introduction

This proposal adds the **`inline`** keyword to Standard C in such a way that it can be implemented with existing linker technology and also is compatible with C++. This is achieved by requiring that one and only one of the translation units containing the definition of an inline function be specified as the one that provides an external definition for the function. Because that specification consists simply of a declaration that lacks the inline keyword, it will also be accepted by a C++ translator.

This proposal does extend the C++ specification in two ways. First, if a function is declared inline in one translation unit, it need not be declared inline in every other translation unit. This allows, for example, a library function that is to be inlined within the library but to be available only through an external definition elsewhere. (The alternative of using a wrapper function for the external function both requires an additional name and may adversely impact performance, if a translator does not actually do inline substitution.)

Second, the requirement that all definitions of an inline function be ‘‘exactly the same’’ is replaced by the requirement that the behavior of the program should not depend on whether a call is implemented with a visible inline definition or the external definition of a function. This allows an inline definition to be specialized for its use within a particular translation unit. For example, the external definition of a library function might include some argument validation that is not needed for calls made from other functions in the same library.

These extensions do offer some advantages, and programmers who are concerned about compatibility can simply abide by the stricter C++ rules. Nevertheless, the extensions are not central to the proposal and the C++ rules could easily be substituted.

Edits to draft C9X Draft 10

6.1.1.1 Keywords

In the **Syntax** section, add to the list of keywords: **`inline`**

6.5 Declarations

In the **Syntax** section, append to the list of alternatives for the definition of *declaration-specifiers*:

function-specifiers

6.5.x Function-specifiers

Add a new subsection, following 6.5.3.

Syntax

function-specifier:
    `inline`

Constraints

Function-specifiers shall be used only in function declarations.

The inline function specifier shall appear only in a file scope function declaration. It shall not appear in a declaration of **`main`**.

Semantics

A function declaration with an **`inline`** function specifier declares an *inline function*. The inline function specifier requests, but does not require, that each call to the function be implemented through inline substitution of its defining compound statement.

For a function with external linkage, the following rules apply. If a function is declared with an inline function specifier, then it shall also be defined in the same translation unit. If all of the file scope declarations for a function in a translation unit include the inline function specifier, then the definition in that translation unit does not provide an external definition for the function. In this case, the defining *compound-statement* for the function is available to a translator only for the purpose of inline substitution, but it may also be ignored by the translator. If the behavior of the program depends on whether a call of the function is implemented using inline substitution or calling the external definition, then the behavior is undefined. (Footnote: Examples of operations in an inline function that can lead to undefined behavior include modifying an object declared with the static storage class, or any operation that is present in one but not another definition of the function.)

Example

The declaration of an inline function can result in either an external definition, or a definition available for use only within the translation unit. A file scope declaration without **`inline`** creates an external definition. The following example shows an entire translation unit.

```
inline double fahr(double t) {
   return (9.0 * t) / 5.0 + 32.0;
}

extern double fahr(double); /* creates an external definition */

inline double cels(double t) {
   return (5.0 * (t - 32.0)) / 9.0;
}

double convert(int is_fahr, double temperature) {
   /* A translator may perform inline substitutions. */
   return is_fahr ? cels(temperature) : fahr(temperature);
}
```

Note that the declarations of inline function **`fahr`** result in the creation of an external definition, but the declaration of **`cels`** requires an external definition in another translation unit.