Document number: N3801
Date: 2013-10-14
Working groups: SG12, CWG
Reply to: gdr@microsoft.com

Removing Undefined Behavior from the Preprocessor

Gabriel Dos Reis
Microsoft

Abstract

This paper recommends removal of *undefined behavior* from the C++ preprocessor. In all cases, the erroneous constructs are identified as violations of diagnosable rules.

1 Introduction

The recommendations contained in this document implement the consensus of SG-12's first meeting in Chicago on Thursday September 26, 2013. Those attending that meeting felt strongly that erroneous preprocessor constructs should be diagnosed, and none of them should lead to unrestricted runtime behavior.

It turns out that most changes were straightforward. A brief discussion of a couple of suggested changes is offered in §3.

2 Proposed wording

The removal of undefined behavior from the preprocessor calls for altering clauses 2 and 16. These changes are relative to committee document N3691, the latest Wording Draft prior to the Chicago meeting.

2.1 Modification of Clause 2

1. Modify second bullet of §2.2/1 as follows:

> [...] If, as a result, a character sequence that matches the syntax of a universal-character-name is produced, ~~the behavior is undefined~~the logical source line is conditionally supported with an implementation-defined semantics.

2. Modify fourth bullet of §2.2/1 as follows:

   [...] If a character sequence that matches the syntax of a universal-character-name is produced by token concatenation (16.3.3), ~~the behavior is undefined~~the program is ill-formed.

3. Modify §2.5/2 as follows:

   If a `'` or a `"` character matches the last category, ~~the behavior is undefined~~the program is ill-formed.

2.2 Modification of Clause 16

1. Modify §16.1/4 as follows:

   [...] If the token `defined` is generated as a result of this replacement process or use of the `defined` unary operator does not match one of the two specified forms prior to macro replacement, ~~the behavior is undefined~~the program is ill-formed.

2. Modify §16.2/4 as follows:

   [...] If the directive resulting after all replacements does not match one of the two previous forms, ~~the behavior is undefined~~the program is ill-formed.

3. Modify §16.3/11 as follows:

   [...] If there are sequences of preprocessing tokens within the list of arguments that would otherwise act as preprocessing directives, ~~the behavior is undefined~~the program is ill-formed.

4. Modify §16.3.2/2 as follows:

   [...] If the replacement that results is not a valid character string literal, ~~the behavior is undefined~~the program is ill-formed.

5. Modify §16.3.3/3 as follows:

   [...] If the result is not a valid preprocessing token, ~~the behavior is undefined~~the program is ill-formed.

6. Modify §16.4/3 as follows:

> [...] If the digit sequence specifies zero or a number greater than ~~2147483647, the behavior is undefined~~an implementation-defined limit, which shall be no less than 2147483647, the program is ill-formed.

7. Modify §16.4/5 as follows:

> [...] If the directive resulting after all replacements does not match one of the two previous forms, ~~the behavior is undefined~~the program is ill-formed; otherwise, the result is processed as appropriate.

8. Modify §16.8/4 as follows:

> If any of the pre-defined macro names in this subclause, or the identifier `defined`, is the subject of a `#define` or a `#undef` preprocessing directive, ~~the behavior is undefined~~the program is ill-formed.

2.3 Implementation-defined limits

The modification of §16.4/3 clearly acknowledge the existence of an implementation-defined limit for the line number in a `#line` preprocessing directive. Consequently, add the following to Annex B

> — the line number in a `#line` directive [2147483647]

3 Discussion

Existing implementations offer the diagnosis suggested in §2.5/2.
There is no proposal to amend §2.14.5/12 because it is a description of the effect of potentially evaluated expression that is outside the realm of the preprocessor.
The implementation-defined limit on the line-number recognizes an implicit constraint on programs, the violation of which is best diagnosed at translation time.

4 Acknowledgment

I would like to thank the audience at the SG12 meeting in Chicago, members of the UB mailing list, and Jens Maurer for their feedback.