

ANSI DOC NO: X3J16/92-0101R
ISO DOC NO: WG21/N0178
DATE: Nov 16, 1992

**C++ Translation Limits
The Boston Proposal**

Paul Stone
Perennial
paul@peren.com

Table of Contents

Introduction
Abstract
Legend and Rationale
Proposed Wording
Table of Translation Limits
Glossary with Notations
References

Introduction

At the Boston X3J16/WG21 meeting two proposals relating to translation limits were presented for straw vote. The first proposal was a standalone proposal on translation limits. The second proposal, Twofold Conformance, was a qualifier on the first proposal but was considered premature for formal vote. This paper is a presentation of what was presented at Boston for formal vote, therefore the Twofold Conformance section has been removed.

Abstract

The advocates of this proposal believe that the notion of a portable C++ program is impossible without specification of translation limits.

ANSI/ISO C specified translation limits for C implementations. This C++ proposal expands on the C specification in two ways: Individual (solo) limits specification, and C++ specific parameters.

Legend and Rationale

1. You may notice the temporary introduction of what look like macro names, ie, NEST_COMPOUNDS, as abbreviations for each item. This is only to aid in our discussion, which may last quite a while. The names need not appear in the Draft Document. A glossary is attached. The numbers preceding (leftmost column) are also for the ease of the reader, but are more subject to change than the macro names.
2. The third column "C combo" holds the value defined for ANSI/ISO C 2.2.4.1/5.2.4.1. These values are given for historic reference, are not subject to debate, and will not appear in the Draft Document. The abbreviation "combo" stands for "combined limits" --- the "rubber teeth" test program that contains an instance of each of the limits within one strictly conforming C program.
3. The fourth column "C++ combo" lists the proposed values for the same kind of combined limits, rubber teeth test, as defined for ANSI C, but as applied to C++. All of these values are subject to committee scrutiny --- we propose that they appear in the Draft Document. The "C++ combo" values, en masse, define a "least common denominator" of program portability. For sake of discussion, the values shown currently mirror the ANSI/ISO C standard, and may be too low if individual testing is rejected by committee (see "C++ solo").
4. The fifth column "C++ solo" lists the proposed values for separately tested translation limits. This column is provided in order to fulfill the second criterion of Andrew Koenig's Dallas proposal.

The purpose of the "C++ solo" tests is twofold.

"Minima become maxima", Bjarne has observed of the ANSI C translation limits. For instance, corporate policy may dictate that all C programs not exceed the translation limits (be strictly conforming programs), for fear of non-portability. We cannot not dismiss such a policy as being totally misguided.

"We should pick unreasonably large values for the individual tests [C++ solo] such that an implementation may not impose arbitrary fixed limits."

The use of the words minimum and maximum has been avoided because it is misleading. All values shown in all three columns should be thought of as minima.

Proposed Wording

Add a new section to the Draft on Translation Limits containing the following text:

The implementation shall be able to translate and execute at least one program that contains one instance of every "C++ combo" limit.

For each of the "C++ solo" limits, the implementation shall be able to translate and execute at least one program that contains an instance of that limit.

and including the table of translation limits that follows.

Notes about the table:

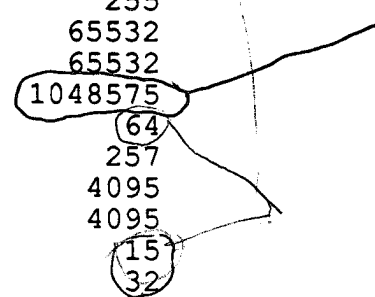
1. The column "C combo" is for reference only, and is not part of this proposal per se.
2. The first three entries were treated as one limit by ANSI C.

Table of Translation Limits

Item Name	C combo	C++ combo	C++ solo
01 NEST_COMPOUNDS	15	15	255
02 NEST_ITERATIONS	15	15	255
03 NEST_SELECTIONS	15	15	255
04 NEST_CONDITIONAL_INCLUSION	8	8	32
05 DECL_PTR_ADR_FNC	12	12	15
06 NEST_PAREN_DECL	31	31	31
07 NEST_PAREN_EXPR	32	32	32
08 SIGNIFICANT_INTERNAL	31	31	1021
09 SIGNIFICANT_EXTERNAL	6	6	1021
10 EXTERNAL_IDENTIFIERS	511	511	65532
11 BLOCK_IDENTIFIERS	127	127	511
12 MACRO_IDENTIFIERS	1024	1024	1024
13 FUNCTION_PARAMETERS	31	31	255
14 FUNCTION_ARGUMENTS	31	31	255
15 MACRO_PARAMETERS	31	31	255
16 MACRO_ARGUMENTS	31	31	255
17 LINE_LENGTH	509	509	65532
18 LITERAL_LENGTH	509	509	65532
19 OBJECT_SIZE	32767	32767	1048575
20 NEST_INCLUDES	8	8	64
21 CASE_LABELS	257	257	257
22 STRUCT_MEMBERS	127	127	4095
23 ENUM_CONSTANTS	127	127	4095
24 NEST_STRUCTS	15	15	15
25 AT_EXIT_FUNCTIONS	32	32	32

C++-specific limits:

Item Name	N/A C combo	C++ combo	C++ solo
26 ALL BASES	-	1024	16384
27 DIRECT_BASE_CLASSES	-	1024	1024
28 NEST_CLASSES	-	15	15
29 CLASS_MEMBERS	-	127	4095
30 ABSTRACT_FUNCTIONS	-	1024	4096
31 CONVERSION_FUNCTIONS	-	256	1024
32 OVERLOADED_FUNCTIONS	-	256	1024
33 OVERLOADED_CONSTRUCTORS	-	256	1024
34 VIRTUAL_FUNCTIONS	-	1024	1024
35 VIRTUAL_BASE_SUBOBJECTS	-	1024	1024
36 STATIC_MEMBERS	-	256	1024
37 FRIENDS	-	1024	4096
38 ACCESS_DECLARATIONS	-	1024	4096
39 MEM_INITIALIZERS	-	1024	32768
40 SCOPE_QUALIFIERS	-	1024	4096
41 NEST_EXTERNS	-	256	1024
42 TEMPLATE_ARGUMENTS	-	256	256
43 HANDLERS_PER_TRY_BLOCK	-	256	256
44 EXCEPTION_SPECS	-	256	256



Glossary with Notations (order of previous appearance)

NEST_COMPOUNDS

Nesting levels of compound statements.

Note: NEST_COMPOUNDS, NEST_ITERATIONS & NEST_SELECTIONS entries were treated as one limit by ANSI C.

NEST_ITERATIONS

Nesting levels of iteration control structures.

NEST_SELECTIONS

Nesting levels of selection control structures.

NEST_CONDITIONAL_INCLUSION

Nesting levels of conditional inclusion.

DECL_PTR_ADR_FNC

Pointer, array, and function declarators (in any combinations) modifying an arithmetic, a structure, a union, or an incomplete type in a declaration.

NEST_PAREN_DECL

Nesting levels of parenthesised declarators within a full declarator.

NEST_PAREN_EXPR

Nesting levels of parenthesised expressions within a full expression.

SIGNIFICANT_INTERNAL

Significant initial characters in an internal identifier or macro name.

SIGNIFICANT_EXTERNAL

Significant initial characters in an external identifier.

EXTERNAL_IDENTIFIERS

External identifiers in one translation unit.

BLOCK_IDENTIFIERS

Identifiers with block scope declared in one block.

MACRO_IDENTIFIERS

Macro identifiers simultaneously defined in one translation unit.

FUNCTION_PARAMETERS

Parameters in one function definition.

FUNCTION_ARGUMENTS

Arguments in one function call.

MACRO_PARAMETERS

Parameters in one macro definition.

MACRO_ARGUMENTS

Arguments in one macro invocation.

LINE_LENGTH

Characters in a logical source line.

LITERAL_LENGTH

Characters in a character string literal or wide string literal (after concatenation).

Glossary with Notations, cont.

OBJECT_SIZE
Bytes in an object (in a hosted environment only).

NEST_INCLUDES
Nesting levels for #included files.

CASE_LABELS
Case labels for a switch statement (excluding those for any nested switch statements).

STRUCT_MEMBERS
Members in a single structure or union.

ENUM_CONSTANTS
Enumeration constants in a single enumeration.

NEST_STRUCTS
Levels of nested structure or union definitions in a single struct-declaration-list.

AT_EXIT_FUNCTIONS
Functions registered by atexit().
See ANSI C X3.159-1989, 4.10.4.4.
Note: This is a runtime, rather than translation, limit.

C++-specific limits:

ALL_BASES
Direct and indirect base classes
(count of edges in the inheritance graph).

DIRECT_BASE_CLASSES
Direct bases classes per class.

NEST_CLASSES
Depth of nested class definitions, ie,
class S1 { class S2 { class S3 { int i; }; }; };
Note: NEST_CLASSES may be redundant with NEST_STRUCTS.

CLASS_MEMBERS
Class members in a single class object.
Note: May be redundant with STRUCT_MEMBERS.

ABSTRACT_FUNCTIONS
Abstract functions in one class.

CONVERSION_FUNCTIONS
Type conversions 'operator T()' in one class.

OVERLOADED_FUNCTIONS
Overloaded functions for a given name.

OVERLOADED_CONSTRUCTORS
Overloaded constructors in one class.

Glossary with Notations, cont.

VIRTUAL_FUNCTIONS

Virtual functions per class.

VIRTUAL_BASE_SUBJECTS

Virtual base subobjects per class object.

STATIC_MEMBERS

Static members of one class.

FRIENDS

Friend declarations in one class.

ACCESS_DECLARATIONS

Access control declarations in one class.

MEM_INITIALIZERS

mem-initializers. Initializations of base classes or members in a constructor definition, e.g.,
T::T() : a(1), b(2), ... { }

SCOPE_QUALIFIERS

Scope qualifications of one identifier, e.g.,
BASE1::BASE2::BASE3::id

NEST_EXTERNS

``extern "lang" { }'' nesting levels.

TEMPLATE_ARGUMENTS

Template arguments in a template declaration.

HANDLERS_PER_TRY_BLOCK

Handlers per try block.

EXCEPTION_SPECS

Throw specifications on a single function declaration; that is, the number of type-id's in the type-id-list of an exception-specification.

References

0. Minutes of X3J16/WG21 Boston meeting, where both straw and formal votes were taken (and formal vote reconsidered).
1. Minutes of X3J16 Dallas meeting, X3J16/91-0136, pg 16-18. Sets mandate for inclusion of translation limits.
2. ANSI C Definition, X3.159-1989 2.2.4.1. (ISO 9899, 5.2.4.1.)
3. NIST/FIPS-160 ANSI C Validation Suite, ACVS, especially test P20031.c (aka rubber teeth).
4. Email traffic on env reflector, beginning with x3j16-env-289.