ISO/IEC JTC 1/SC 22 N 4484

2009-08-11

ISO/IEC JTC 1/SC 22
Programming Languages

Document Type:	**National Body Contribution**
Document Title:	**Netherlands Contribution on The Growth of Programming Language Standards**
Document Source:	**National Body of the Netherlands**
Document Status:	**This document is circulated to National Bodies for review and consideration. This document was submitted by the Netherlands before the due date for document submission but was inadvertently not posted. Therefore, it will be discussed under agenda item 12.5.**
Action ID:	**ACT**
Due Date:	
No. of Pages:	**4**

Secretariat, ISO/IEC JTC 1, American National Standards Institute, 25 West 43rd Street, New York, NY 10036; Telephone: 1 212 642 4932; Facsimile: 1 212 840 2298; Email: lrajchel@ansi.org

The growth of programming language standards

Author: Willem Wakker (NEN, Netherlands)
Date: July 2009
Status: For discussion at SC22 Plenary, Delft, September 2009

Over the past decades programming language standards have been augmented with many new features aimed at supporting modern hardware features (such as parallelism) or state-of-the-art programming concepts (like object orientation). At the same time backward compatibility was always a great concern. The result of these two, sometimes conflicting, approaches to PL development have resulted in ever growing standards: new features were added to the standard, while existing features remained.
For the large systems and implementations that make use of all the features (or nearly all the features) this is no problem. However for modern small, dedicated (embedded) processors that do not support all the new features (or even the older features such as for instance floating point) this is a problem: when (nearly) the whole standard is required to be supported by a conforming implementation it is practically impossible to deliver conforming implementations for such small systems.
Furthermore, for a growing number of programming language standards it is reported that the current (or even the previous) version of the standard is either not fully implemented at all, or only by very few organizations. This is a serious threat for the credibility of programming language standardization.

This document proposes a way forward by defining a set of requirements for what is called a Conforming Subset Implementation (CSI). It is the intention that these (generic) requirements can be added to any programming language standard thereby allowing for more implementations that conform, either to the standard or to a (well defined) CSI.
The aim is that any CSI specification describes a real subset of the full standard so that a full implementation can run any CSI conforming application without problems; consequently a full standard conforming implementation is also a CSI conforming implementation.
This document does not address the question of how a CSI should be documented, approved and registered, once they are made possible. That issue should be addressed in a later stage.

Requirements on a CSI:

- A Conforming Subset Implementation (CSI) specification shall never define new functionality. If a platform requires a subset of the programming language plus new additional functionality then the new functionality must be provided in a separate document. The platform can then conform to the combination of the subset of the existing standard plus the new specifications.
- A program that adheres to a CSI specification (i.e., does not rely on additional specifications) shall be a conforming program according to the full standard; such a program shall have the same effect on both (the full and the subset) implementations.
- A CSI may alter limits defined in the standard as follows:
  - If the limit is specified as having a fixed value, it shall not be changed by a CSI.
  - If a limit is specified as having a minimum or maximum acceptable value, it may be changed by a CSI as follows:
    - A CSI may increase a minimum acceptable value, but shall not make a minimum acceptable value smaller.
    - A CSI may reduce a maximum acceptable value, but shall not make a maximum acceptable value larger.
  - A CSI shall not change a limit specified as having a minimum or maximum value into a limit specified as having a fixed value.

- A CSI shall not create new limits.

- A CSI language processor shall produce error messages for all constructs from the full standard that are not supported by the subset.
- A CSI language processor may not redefine certain language features. Example: it is not allowed to implement fixed-point arithmetic using the keywords for floating-point. If a CSI supports keywords from the full standard, then those keywords shall have exactly the same meaning as defined by the full standard.
- If functionality is excluded by the CSI by declaring one or more keywords as not supported, the use of related but non keyword dependent functionality (think of format specifiers in printing or scanning routines) always results in undefined behavior. Whether the CSI language processor detects such cases is a matter of quality of implementation.

Methods of defining a CSI:

- The most common way to define a CSI is the exclusion of certain keywords from the main language; example: a CSI for the programming language C may exclude the keywords **`float`** and **`double`**. In such a CSI the excluded keywords remain keywords in the sense of the original programming language standard: in C those keywords may not be used in the CSI as variable names.
  Note: some language features cannot be excluded by the simple exclusion of one or more keywords (suppose one would exclude array's from a language …); in such cases the definition of the CSI will be somewhat more complex but the general idea remains the same.
- An alternative method to restrict a full standard is the definition of optional parts in the standard. This is the preferred method when the subset is already defined during the development of the standard.
- There is no requirement that a facility (such as a form of pre-processor variables) exists so that a program can make use of the presence of absence of language features in a CSI. However, individual programming languages may prescribe such features.
- When adding new functionality to a programming language, the developing organization (working group) is strongly encouraged to carefully consider whether the new functionality really belongs in the main body of the standard, or if the functionality can be defined in a separate non-mandatory annex or separate document (IS or TR). When the main body remains (relatively) small there is less need for subset implementations.
- When implementing the CSI requirements in programming language standard, additional programming language dependent requirements may be specified (example: a CSI specification is not allowed to exclude the integer type).
- When implementing the CSI requirements in programming language standard, the generic rules for a CSI should probably go into the conformance section of the main body, the definition for an individual CSI should go probably in an informative Annex, or in a separate document. Once the generic rules are included in the standard, also separate interest groups can define their own CSI (e.g., the Fortran CSI for washing machines).

Criticisms.
The most common objection against subsetting a standard is the fact that subsets make it difficult/unpractical/impossible to write truly portable applications and libraries: such programs should check for (all combinations of) subset definitions, contain possibly large portions of source code that should be compiled conditionally which makes maintenance and testing a nightmare.
The counter argument is that the program conforming to the Fortran CSI for washing machines might be used for (and hence should be portable to) a dishwasher environment, but this it is highly unlikely that this program will ever be used in a space telescope which will require either the full standard or its own CSI. But if there is no Fortran CSI for washing machines then any manufacturer can define its own nonconforming implementation ('what the hack, I am not conforming anyway') and the chances that his program might be reused in a different environment are strongly reduced.

So, yes, subsets may limit portability but general portability ('every program should be able to run anywhere') is a non-attainable goal anyway.
Furthermore: something need to be done quickly: see introduction.

Delft, 10 July 2009

Chairman Dutch SC22 mirrorcommittee: Willem Wakker