C - The C1X Charter

Document: WG14 N1250
Author: John Benito, Convener
Date: 2007/06/29

Introduction

At the WG14/INCITS J11 meeting in London, UK, April 2007 there was general agreement the committee should start thinking about what was next for a revision of the C Standard. I accepted an action item to modify the charter used for the development of C9X. As in the original charter used for C9X, the intention of this charter is to present a statement of principles and a plan of attack. This charter does not identify any technical issues since those are immaterial at this stage.

Although the committee is not required to begin work on revising the current standard, there is much happening that can or does influence C directly. Examples are the evolution of C-like programming languages (C#, Java and C++), the rising threat of internet security, the increased awareness of programming language vulnerabilities , and emphasis on safety-critical systems to name a few.

Original Principles

Before embarking on a revision of the C Standard, it is useful to reflect on the charter of the original drafting committee, according to the original Rationale Document in the section entitled *Purpose:*, and the version of the charter that was used for the C99 revision.

The work of the Committee was in large part a balancing act. The Committee has tried to improve portability while retaining the definition of certain features of C as machine-dependent. It attempted to incorporate valuable new ideas without disrupting the basic structure and fabric of the language. It tried to develop a clear and consistent language without invalidating existing programs. All of the goals were important, and each decision was weighed in the light of sometimes contradictory requirements in an attempt to reach a workable compromise.

In specifying a standard language, the Committee used several guiding principles, the most important of which are:

1. Existing code is important, existing implementations are not. A large body of C code exists of considerable commercial value. Every attempt has been made to ensure that the

bulk of this code will be acceptable to any implementation conforming to the Standard. The Committee did not want to force most programmers to modify their C programs just to have them accepted by a conforming translator.

On the other hand, no one implementation was held up as the exemplar by which to define C: It is assumed that all existing implementations must change somewhat to conform to the Standard.

2. C code can be portable. Although the C language was originally born with the UNIX operating system on the DEC PDP-11, it has since been implemented on a wide variety of computers and operating systems. It has also seen considerable use in cross-compilation of code for embedded systems to be executed in a free-standing environment. The Committee has attempted to specify the language and the library to be as widely implementable as possible, while recognizing that a system must meet certain minimum criteria to be considered a viable host or target for the language.

3. C code can be non-portable. Although it strove to give programmers the opportunity to write truly portable programs, the Committee did not want to force programmers into writing portably, to preclude the use of C as a "high-level assembler;" the ability to write machine-specific code is one of the strengths of C. It is this principle which largely motivates drawing the distinction between strictly conforming program and conforming program.

4. Avoid "*quiet changes*." Any change to widespread practice altering the meaning of existing code causes problems. Changes that cause code to be so ill-formed as to require diagnostic messages are at least easy to detect. As much as seemed possible, consistent with its other goals, the Committee has avoided changes that quietly alter one valid program to another with different semantics, which cause a working program to work differently without notice. In important places where this principle is violated, the Rationale points out a *QUIET CHANGE*.

5. A standard is a treaty between implementer and programmer. Some numerical limits have been added to the Standard to give both implementers and programmers a better understanding of what must be provided by an implementation, of what can be expected and depended upon to exist. These limits are presented as minimum maxima (i.e., lower limits placed on the values of upper limits specified by an implementation) with the understanding that any implementer is at liberty to provide higher limits than the Standard mandates. Any program that takes advantage of these more tolerant limits is not strictly conforming, however, since other implementations are at liberty to enforce the mandated limits.

6. Keep the spirit of C. The Committee kept as a major goal to preserve the traditional spirit of C. There are many facets of the spirit of C, but the essence is a community sentiment of the underlying principles upon which the C language is based. For the Cx1 revision there is consensus to add a new facet **f** to the original list of facets. The new spirit of C can be summarized in phrases like:

(a) Trust the programmer.
(b) Don't prevent the programmer from doing what needs to be done.
(c) Keep the language small and simple.
(d) Provide only one way to do an operation.
(e) Make it fast, even if it is not guaranteed to be portable.
(f) Make support for safety and security demonstrable.

Proverb **e** needs a little explanation. The potential for efficient code generation is one of the most important strengths of C. To help ensure that no code explosion occurs for what appears to be a very simple operation, many operations are defined to be how the target machine's hardware does it rather than by a general abstract rule. An example of this willingness to live with what the machine does can be seen in the rules that govern the widening of char objects for use in expressions: whether the values of char objects widen to signed or unsigned quantities typically depends on which byte operation is more efficient on the target machine.

One of the goals of the Committee was to avoid interfering with the ability of translators to generate compact, efficient code. In several cases the Committee has introduced features to improve the possible efficiency of the generated code; for instance, floating point operations may be performed in single-precision if both operands are float rather than double.

Additional Principles for C9X

At the WG14 meeting in Tokyo, Japan, in July 1994, the original principles were re-endorsed and the following new ones were added:

7. Support international programming. During the initial standardization process, support for internationalization was something of an afterthought. Now that internationalization has proved to be an important topic it should have equal visibility with other topics. As a result, all revision proposals submitted shall be reviewed with regard to their impact on internationalization as well as for other technical merit.

8. Codify existing practice to address evident deficiencies. Only those concepts that have some prior art should be accepted[1]. Unless some proposed new feature addresses an evident deficiency that is actually felt by more than a few C programmers, no new inventions should be entertained.

9. Minimize incompatibilities with C90 (ISO/IEC 9899:1990). It should be possible for existing C implementations to gradually migrate to future conformance, rather than requiring a replacement of the environment. It should also be possible for the vast majority of existing conforming C programs to run unchanged.

[1] Prior art may come from implementations of languages other than C or C++.

10. Minimize incompatibilities with C++. The committee recognizes the need for a clear and defensible plan with regard to how it intends to address the compatibility issue with C++. The committee endorses the principle of maintaining the largest common subset clearly and from the outset. Such a principle should satisfy the requirement to maximize overlap of the languages while maintaining a distinction between them and allowing them to evolve separately.

Regarding our relationship with C++, the committee is content to let C++ be the "big" and ambitious language. While some features of C++ may well be embraced, it is not the committee's intention that C become C++.

11. Maintain conceptual simplicity. The committee prefers an economy of concepts that do the job. Members should identify the issues and prescribe the minimal amount of machinery that will solve them. The committee recognizes the importance of being able to describe and teach new concepts in a straightforward and concise manner.

Additional Principles for C1X (new)

At the WG14 meeting in London, England, in June 2007, the original principles and the principles that were used for C9X were reviewed, and the following observations were added[2]:

12. *Trust the programmer*, as a goal, is outdated in respect to the security and safety programming communities. While it should not be totally disregarded as a facet of the spirit of C, the C1Xversion of the C Standard should take into account that programmers need the ability to check their work.

13. Unlike for C9X, the consensus at the London meeting was that there should be no invention, without exception. Only those features that have a history and are in common use by a commercial implementation should be considered. Also there must be care to standardize these features in a way that would make the Standard and the commercial implementation compatible.

14. Migration of an existing code base is an issue. The ability to mix and match C89, C99, and C1X based code is a feature that should be considered for each proposal.

Important Observations

During the revision process, it will be important to consider the following observations:

- Regarding the 14 principles, there is a tradeoff between them — none is absolute. However, the more the committee deviates from them, the more rationale is needed to explain the deviation.

[2] Not in any particular order.

- The original standard had a very positive reception from both the user and vendor communities. However, C99 has been not so widely received.
- With the C1X revision the Committee agreed that it might discuss the sub-sectioning of the Standard. As the Standard grows in size and complexity the Committee should not force the vendors that supply the small machine market to implement features that bloat the compiler and are not used in the environment for which the implementation is targeted.
- The standard is not considered to be broken. Rather, a revision is needed to track emerging and/or changing technologies.
- Most users of C view it as a general-purpose high-level language. While *higher level* constructs can be added, they should be done so only if they don't contradict the basic principles.
- There are a good number of useful suggestions to be found from the public comments and defect-report processing.

Sources of Influence

Areas to which the committee should look when revising C include:

- All Technical Corrigenda and Records of Response.
- Future directions in current standard.
- Features currently labeled obsolescent or deprecated.
- Requirements resulting from JTC1/SC2 (character sets).
- The evolution of safety critical software development.
- All known software security issue (programming language vulnerabilities)
- All Type II Technical Reports developed by WG14.
- The evolution of C++ and other C based programming languages.
- The evolution of implementations, including compilers, libraries and operation systems.
- Other papers and proposals from member delegations.
- Cross-language standards groups' work.
- Other comments from the public at large.
- Sub-setting the standard.
- Other prior art.

Submission Guidelines

Without a set of acceptance criteria, judging any technical proposal becomes a highly subjective, and definitely emotional, exercise. It also wastes a lot of time and energy. Therefore, submitters are encouraged to keep all the guiding principles in mind when making submissions.

Guidelines for the submission of proposals will be provided. Each submission shall contain a cover page containing responses to a number of questions and further summary information, enabling the essence of a submission to be distilled simply by reading that cover. The information requested will include such things as: title, author, author affiliation, date, document number, abstract, proposal category (e.g., editorial, correction, new feature), prior art[3], and target audience. All proposals must have a WG14 document number and be in a format[4] suitable for electronic distribution, with the cover page as the first page of the document.

Submissions must be sponsored in the same way as Defect Reports; that is, either by the convener of WG14, WG14 itself, or by a WG14 national member body. This provides a filtering process and allows submissions to be rejected early in the process if they violate the revision principles. It also allows substantially incomplete or disjoint proposals to be returned for further refinement.

Documentation

The current Rationale document will be updated.

The Editor for the C9X standard has agreed to take on the task of acting as the Editor for the C1X effort. The initial job of the editor will be to integrate Technical Corrigenda I, II, and II into a single base document against which the committee can work when considering and/or preparing technical papers as well as in handling current and future Defect Reports.

Revision Schedule

The milestones for the revision process are:

- CD Registration -- December 2009
- CD Ballot -- December 2010
- DIS Ballot -- December 2011

This schedule allows for the formal adoption of a revised standard by the end of 2011, with a publication date of 2012.

The purpose of this schedule is twofold:

- To provide the public with a reasonably accurate idea of when a revised C Standard is likely to appear.
- To keep the revision process focused.

[3] This should include all commercial implementations and the history of these implementations. An experimental implementation is not considered a commercial implementation.
[4] HTML, PDF, and MS Word are currently the formats WG14 is using.