# ソースチェックに威力を発揮するCプリプロセッサ

Powerfull C Preprocessor for Source Checking

松井　潔
Kiyoshi MATSUI
陽和病院（〒178-0062 東京都練馬区大泉町 2-17-1） E-mail: kmatsui@t3.rim.or.jp

**ABSTRACT.** C preprocessing has a long historical background which contains some confusions. Although preprocessors have been converging to C Standard since C90, the so-called Standard-conforming preprocessors still sometimes behave wrong. Besides, the existing preprocessors are too reticent on unportable sources. MCPP, the free and portable C preprocessor develeped by the author, has been verified using Validation Suite, also developed by the author, which tests C/C++ preprocessor's conformance and quality comprehensively. MCPP not only has the highest conformance but also has plentiful and accurate diagnostics to check almost all the preprocessing problems in source code. The superiority of MCPP over the other preprocessors is shown using the Validation Suite.

## 1　背景

Cで書かれたソースプログラムには、プリプロセスのレベルでの問題を持っているものが少なくない。特定の処理系でコンパイルできることをもって良しとしてしまっているものの portability を欠いているもの、不必要にトリッキーな書き方をしているもの、C90 以前の特定の処理系の仕様をいまだにあてにしているもの、等々である。こうしたソースの書き方は portability と readability そしてメンテナンス性を損なうものであり、悪くすればバグの温床ともなりかねない。そうしたソースをより portable で明快な形で書き直すことは、多くの場合、簡単なことなのであるが、見過ごされている場合も多い。

そうしたソースが多く存在する背景となっているのは、一つには C90 以前のプリプロセス仕様がはなはだあいまいだったことである。これが、C99 が決まった今となっても尾を引いている。もう一つは、既存のプリプロセッサが寡黙すぎることである。プリプロセッサが怪しげなソースを黙って通すために、問題が見過ごされてしまうのである。

長い歴史を持つCのプリプロセス仕様には多くの混乱があった。C90 以降は各処理系の仕様が規格を中心に収束してきているが、「規格準拠」をうたう処理系が間違った動作をすることがいまだにみられる。これは処理系そのものの検証が不十分なためである。正確なCプリプロセッサの開発のためには、その検証をするソフトウェアの整備が必要である。

また、処理系固有の仕様はすべてドキュメントに記載することがC言語規格で定められているように、詳細で正確なドキュメントの作成は処理系の不可欠の部分を成す。しかし、この点も不十分な処理系が多い。この仕様のあいまいさが、portability

を欠いたソースプログラムを生む一つの背景ともなっていると考えられる。

さらに、規格自体にも、歴史的な事情からくる矛盾やあいまいさがいくつかあり、問題を複雑にしている。十分枯れた分野のようにも見えるCプリプロセスであるが、まだ問題は収束したとは言えないのである。

## 2 目的

私は久しく以前からCプリプロセッサを開発してきた。その成果はすでに 1998/08 に cpp V.2.0 として、1998/11 に cpp V.2.2 として公開している。このプリプロセッサは V.2.3 への update の途中で、情報処理振興事業協会 (IPA) の「平成１４年度未踏ソフトウェア創造事業」に採択された [1]。この cpp を他の cpp と区別するために MCPP と呼ぶ。Matsui CPP の意味である。

このプロジェクトの目的は、次のようなものである。

- 動作が正確で診断メッセージの豊富な、世界一優れたCプリプロセッサを開発する。
- Cプリプロセッサの動作と品質を徹底的にチェックする検証セットを充実させ、既存の主要な処理系を検証する。
- ことに GNU C を重視し、GNU C の testsuite において本開発の検証セットが利用できることを目指す。
- 詳細で正確なドキュメントを作成し、整備する。
- 英語版のドキュメントも作成して、成果を世界に問う。

## 3 MCPP の概要

MCPP は次のような特徴を持っている。

1. きわめて正確である。C, C++ のプリプロセスの reference model となるものを目指して作ってある。C90 [2–5] はもちろんのこと、C99 [6, 7], C++98 [8] に対応する実行時オプションも持っている。
2. C, C++ プリプロセッサそのものの詳細かつ網羅的なテストと評価をする検証セットが付属している。
3. 診断メッセージが豊富で親切である。診断メッセージは百数十種に及び、問題点を具体的に指摘する。それらは数種のクラスに分けられており、実行時オプションでコントロールすることができる。
4. デバッグ用の情報を出力する各種の #pragma ディレクティブを持っている。Tokenization をトレースしたり、 マクロ展開をトレースしたり、マクロ定義の一覧を出力したりすることができる。
5. 速度も遅いほうではないので、デバッグ時だけでなく日常的に使うことができる。16 ビットシステムでも使えるように作られているので、メモリが少なくても動作する。
6. Portable なソースである。MCPP をコンパイルする時に、ヘッダファイルにある設定を書き換えることで、UNIX 系、DOS/Windows 系のいくつかの処理系で、付属のプリプロセッサに代替して使えるプリプロセッサが生成されるようになっている。C90, C99, C++98 のどれに準拠する処理系でもコンパイルでき、C90 以前のいわゆる $K\&R^{1st}$ の処理系でさえもコンパイルできる広い portability を持っている。
7. 標準モード（C90, C99, C++98 対応）のプリプロセッサのほか、$K\&R^{1st}$ の仕様やいわゆる Reiser モデルのもの等、各種仕様のプリプロセッサを生成することができる。規格そのものの問題点を私が整理した自称 post-Standard モードまである。
8. Free software である。
9. 詳細なドキュメントが付属している。
   (a) サマリ文書。
   (b) 実行プログラム用マニュアル。使い方、仕様、診断メッセージの意味。ソースの書き方も示唆。
   (c) 実装用ドキュメント。任意の処理系に実装する方法。

表 1: 検証セット V.1.3 の項目数と配点

| | 項目数 | 最低点 | 最高点 |
|---|---|---|---|
| C90 規格合致度 | 173 | -162 | 448 |
| C99 規格合致度 | 20 | 0 | 98 |
| C++98 規格合致度 | 9 | 0 | 26 |
| 品質:診断メッセージ | 46 | 0 | 65 |
| 品質:その他 | 16 | -40 | 111 |
| 計 | 264 | -202 | 748 |

（d）検証セット解説。規格の解説を兼ねる。規格そのものの矛盾点も指摘し、代案を提示している。検証セットをいくつかの処理系に適用した結果を報告している。

## 4 プリプロセス検証セットによる各種プリプロセッサの検証

プリプロセッサの開発と同時にもう一つ問題となるのは、プリプロセッサの動作や品質の検証である。処理系が誤動作したり品質が悪かったりするのでは論外であるが、実際にテストしてみると、かなりの問題が見つかるものである。私は MCPP 開発の一環として、プリプロセス検証セットを作製し、MCPP とともに公開している。これはきわめて多面的な評価項目を持ち、プリプロセッサのできるだけ客観的で網羅的なテストをするものである。

検証セット V.1.3 は表 1 のようにテスト項目が 264 に及んでいる。うち動作テストが 230 項目、ドキュメントや品質の評価が 34 項目を占めている。各項目はウェイトを付けて配点されている。$K\&R^{1st}$ と C90 との共通仕様を正しく実装していれば 0 点、それさえも実装できていなければマイナス点、C90 以降の新しい仕様を正しく実装していればプラス点をつけるようになっている。「規格合致度」には診断メッセージとドキュメントの評価も含まれる。C99, C++98 の「規格合致度」というのは、C90 にない新たな規定に関するものである。また、「品質:診断メッセージ」というのは、規格で要求されていない診断メッセージに関する評価である。

検証セット V.1.3 をいくつかの処理系に適用した結果のサマリを表 2 に示す。処理系は古い順に並べてある。

*1 Martin Minow による DECUS cpp のオリジナル版 (1985/06) の shift-JIS に対応した OS-9/09 への移植版 (1989/04)。 [9]

*2 J. Roskind による UNIX, OS/2, MS-DOS 用 shareware である JRCPP の MS-DOS - OS/2 用試用版 (1990/03)。具体的な処理系には対応しない stand-alone のプリプロセッサ。 [10]

*3 1993 年のものの日本語版 (1994/12)。 [11]

*4 GNU C 2.7.1 / cpp (1995/12) を DJ Delorie が DOS extender である GO32 に移植したもの。日本語版への移植で shift-JIS に対応。 [12]

*5 LSI C-86 / cpp のきだあきらによる改造版 (1996/02)。 [13]

*6 筆者による free software の V.2.0 (1998/08)。DECUS cpp をベースとして書き直したもの。FreeBSD / GNU C 2.7, DJGPP V.1.12, WIN32 / Borland C 4.0, MS-DOS / Turbo C 2.0, LSI C-86 3.3, OS-9/09 / Microware C 等に対応。各種の動作モードのプリプロセッサを生成することができるが、このテストでは 32 ビットシステムでの標準版を使用。

*7 日本語版 (2000/08)。 [14]

*8 Jacob Navia 等による shareware (2000/09)。ソース付き。プリプロセス部分のソースは Dennis Ritchie その人が C90 対応のプリプロセッサとして書いたもの。 [15]

*9 Thomas Pornin による portable な free software (2000/10)。Stand-alone のプリプロセッサ。 [16]

*10 VineLinux 2.5, FreeBSD 4.4, CygWIN 1.3.10 で使われている GNU C 2.95.3 (2001/03)。 [17]

*11 GNU C 3.2R のソース (2002/08) を筆者が VineLinux 2.5, FreeBSD 4.7 上でコンパイルしたもの。 [17]

表 2: 各種プリプロセッサの検証結果

| ＯＳ | 処理系 | 実行プログラム (版数) | 規格合致度 | 総合評価 | 注 |
|---|---|---|---|---|---|
| OS-9/6x09 | Microware C/6809 | cpp | 256 | 317 | *1 |
| MS-DOS | | JRCPPCHK (V.1.00B) | 400 | 450 | *2 |
| WIN32 | Borland C++ V.4.02J | cpp32 | 412 | 458 | *3 |
| DJGPP V.1.12 M4 | GNU C 2.7.1 | cpp | 450 | 549 | *4 |
| MS-DOS | LSI C-86 V.3.30c | cpp (改造版 beta13) | 342 | 400 | *5 |
| FreeBSD, WIN32, etc. | GNU C, Borland C, etc. | cpp (V.2.0) | 502 | 656 | *6 |
| WIN32 | Borland C++ V.5.5 | cpp32 | 410 | 465 | *7 |
| WIN32 | LCC-Win32 V.3.6 | lcc | 392 | 479 | *8 |
| Linux, etc | | ucpp (V.0.7) | 423 | 491 | *9 |
| Linux, FreeBSD | GNU C 2.95.3 | cpp0 | 478 | 580 | *10 |
| Linux, FreeBSD | GNU C 3.2R | cpp0 | 540 | 655 | *11 |
| Linux, FreeBSD, etc. | GNU C, LCC-Win32, etc. | cpp (V.2.3) | 568 | 724 | *12 |

*12 MCPP V.2.3 (2003/02)。Linux / GNU C (2.95.3, 3.2), FreeBSD / GNU C (2.95.3, 3.2), CygWin 1.3.10, LCC-Win32 3.6, Borland C 5.5 等への対応が追加されている。 [18]

このように、MCPP はずば抜けた成績である。動作の正確さ、診断メッセージの豊富さと的確さ、ドキュメントの詳細さ、portability、どれをとっても抜群である。４年半前のバージョンである V.2.0 でさえも、まだそれを超えるものが見当たらない。V.2.3 ではさらに update され改良されている。自分で作って自分でテストしているのであるから当然とも言えるが、これだけ多角的なテストであればかなりの客観性があろう。

MCPP の次に優れているのは、このリストでは GNU C / cpp である。この cpp は C90 規格に合致した正しいソースを処理する分には、ほとんど問題がない。しかし、C99, C++98 の新しい仕様のいくつかが未実装であることは時間とともに解決されてゆくとしても、それ以外にも次のような問題がある。

1. 診断メッセージが不十分である。-pedantic -Wall オプションを指定することで多くの問題はチェックできるが、それでもまだかなり不足している。
2. デバッグ情報を出力する機能はほとんどない。
3. ドキュメントが少なく、仕様の不明確な部分や隠れ仕様が多い。ことに V.2 では -traditional オプションを指定しなくても traditional な仕様が隠れていることが問題である。
4. 規格と矛盾する独自仕様が多い（拡張仕様は #pragma で実装すべきものである）。

GNU C V.3 / cpplib は GNU C V.2 / cpp に比べるとこれらの点で大幅に改善されたが、まだ不十分である。

MCPP が GNU C / cpp に負けるのは速度くらいである。

他のプリプロセッサにはさらに問題が多い。

## 5 プリプロセッサによるソースチェックはなぜ必要か

ところで、プリプロセッサはＣ処理系の一部にすぎない。いくら世界一でも、プリプロセッサだけ作って何の役に立つのだろうか。

Cのプリプロセスは文字通りソースの「前処理」であるが、コンパイラ本体に対するオマケのようなものとして扱われてきたきらいがある。しかし、この「前処理」の存在理由は、readability とメンテナンス性を改善することであり、多様なシステムに対応するための portability の確保であり、要するにソースをより人間（プログラマ）にとって扱いやすくすることである。コーディングとメンテナンスの作業に対する影響は大きい。逆に、「前処理」が誤用されると、readability も portability もかえって損なわれることになる。この意味で、プリプロセッサによるソースチェックは重要なのである。

MCPP を処理系付属のプリプロセッサと置き換えて使うことで、ソースプログラムのプリプロセス上の問題点を、潜在的なバグや規格違反から portability の問題まで、ほぼすべて洗い出すことができる。

MCPP を FreeBSD 2.2.2R (1997/05) の kernel および libc ソースに適用した結果は、V.2.0 以来、そのドキュメントで報告している。Libc にはまったくと言ってよいほど問題がなかったが、kernel には全体からみればごく一部のソースではあるものの、いくつかの問題が発見された。問題のソースの多くは、4.4BSD-lite にあったものではなく、FreeBSD への実装と拡張の過程で新しく書かれたものであった。

その後、当時開発中であった MCPP V.2.3 を Linux の glibc 2.1.3 (2000/09) に適用してみたところ、こちらにも少なからぬ問題点のあることがわかった。これらの問題には、UNIX 系システムに古くから存在するいわゆる traditional なプリプロセス仕様を使ったものと、GNU C / cpp の独自の仕様や undocumented な仕様を前提としたものが多い。たとえば次のようなものである。

- 行をまたぐ文字列リテラル
- Cのプリプロセスを要するアセンブラソース
- ‘defined’ に展開されるマクロ
- 関数型マクロとして展開されるオブジェクト型マクロ
- Undocumented な環境変数を使うもの

GNU C / cpp がこれらを、少なくともデフォルトの設定では黙って通してしまうことが、こうした感心しない書法のソースを温存させ、そればかりか新たに生み出す結果になっていると考えられる。こうした書法は古いソースに多いとは限らず、むしろ新しいソースに時々見られることが問題である。システムのヘッダファイルにさえも時に問題がある。

他方で、コメントのネストは規格違反であるが、1990 年代中ごろまでは UNIX 系のオープンソースにしばしば見られたものの、その後は見かけなくなっている。これは GNU C / cpp がコメントのネストを認めなくなったためであろう。プリプロセッサはソースに大きな影響を与えるのである。

## 6　MCPP の実装方法

私が MCPP の実装にあたって目標としたのは「MCPP の概要」で述べた諸点である。これは完全主義的な目標であるため、かなりの時間がかかってしまったが、ほぼ達成できてきている。

正しいソースを正しく処理することはもちろんであるが、間違ったソースや怪しげなソースに的確な診断メッセージを出すことも重視し、十分なメッセージ群を用意した。

ドキュメントも重視し、undocumented な仕様のないよう、規格書にある共通仕様以外の全仕様をドキュメントに記載した。規格については、検証セットのドキュメントに網羅的な解説を記載した。

多くの処理系に実装してきたことも、MCPP 自身のソースの portability を広げ、動作チェックを徹底するために役立っている。

また、網羅的な検証セットの作製と並行して開発してきたことも、バグのないプリプロセッサを作る結果につながってきている。他のプリプロセッサを見ると、例えば LCC-Win32 のプリプロセス部分は Ritchie の書いたソースを使っているが、これには #if 式の評価などにかなりのバグがある。いかに Ritchie であっても、検証セットなくしてはバグは免れないのである。GNU C / cpp は長い間に多く

の人々にデバッグされてきてほぼ bug free となっているが、十分な検証セットがあればもっと早くバグがとれたはずである。それほどポピュラーでないプリプロセッサの場合は、検証セットなしにはデバッグは不可能と言って良い。

### 6.1 トークンベースの原則

さらに MCPP の内部的な実装方法に立ち入ると、私が重視したのはまず「トークンベース」の処理である。

MCPP で洗い出されるプリプロセス上の多くの問題の底流にあるのは、Cプリプロセスの原則に関する混乱である。Cのプリプロセスは「トークンベース」を原則とするものであるが、C90 以前にはあいまいであったために、文字ベースのテキスト処理の発想が入り込んでいた。C90 以降もプリプロセッサがそれを看過していたり、プリプロセッサ自身に文字ベースの処理が混入していたりすることによって、混乱が長く続いてきているのである。その上、規格自体にも中途半端な規定や矛盾がいくつか存在し、C99 でも改められていないことが、問題をいっそう複雑にしている。

MCPP のプログラム構造は「トークンベースのプリプロセス」という原則で組み立てられており、traditional な文字ベースのプリプロセスとは発想を異にする。他のプリプロセッサでは、トークンベースの処理を意図しながらも、そこに文字ベースの処理が紛れ込んでしまっていることが多いようである。プリプロセッサのバグの何割かはこれによるものだと思われる。

### 6.2 関数型マクロの関数的展開

MCPP の実装ではマクロ展開ルーチンも意を注いだところであり、既存のプログラムにとらわれず、明快なプログラム構造とすることを心掛けた

引数のないマクロの展開は単純なものであるが、引数付きマクロの展開には歴史的にさまざまな仕様があり、混乱があった。C90 で一応の整理がされたが、まだ収束したとは言えない状況にある。この問題については私の検証セットの cpp-test.txt 1.7.6 節で詳細に論じているところである。

C90 はこの混乱の続いていた引数付きマクロについて、「関数型マクロ 」という名前を付けて、関数呼び出しに似せて仕様を整理した。ところが C90 は同時に伝統的仕様に引きずられて矛盾した規定をしている。そして、その後、これについては corrigendum が出たり再訂正されたりしながら、C99 に至るまで迷走を続けてきている。

MCPP の関数型マクロの展開では、伝統的な実装方法を一掃し、また規格対応を第一とするのではなく明快な「関数的」展開を第一として、その上で規格の不規則な規則に対応する、という実装方法をとっている。

なお、MCPP の post-Standard モードというのは、「トークンベース」の原則と関数型マクロの「関数的」展開の原則を徹底させた、私の独自の仕様である。このモードで問題の検出されないソースは、プリプロセス上はきわめて portability の高いソースだと言える。

## 7 V.2.3 の成果

私は V.2.2 を公開した後、しばらくの間は時間がとれず、MCPP の開発は停滞してしまった。しかし、「未踏ソフトウェア」に新部 裕・プロジェクトマネージャによって採択されたのを機に、仕事を減らして時間を確保した。そして、2003/02 に MCPP V.2.3 と検証セット V.1.3 をリリースすることができた。このプロジェクトの期間中の主な成果としては、次のようなものがある。

### 7.1 GNU C 3.2 への対応

GNU C 3.2 のプリプロセッサを MCPP に置き換えた上で、GNU C 3.2 を使って GNU C 3.2 自身をリコンパイルし、testsuite を適用して結果を検証した。MCPP が GNU C 3 / cpp と実用上はほぼ十分な互換性を持っていることが確かめられた。

GNU C 3 のプリプロセッサは GNU C 2 と比べて、ソースが一新されるとともに、ドキュメントも一新され、testsuite の testcases も飛躍的に増えている。その結果、MCPP に接近する方向に大幅に改善されていることが認められた。また、GNU

C 3.2 自身のソースも glibc 2.1 等と比べると、はるかに問題の少ないことがわかった。これはプリプロセッサが一新されたことによる影響が大きいと考えられる。

### 7.2 検証セットの GNU C / testsuite への対応

私の検証セットのうち動作テストの testcases には、GNU C の testsuite として使える edition が追加された。従来の testsuite の testcases にはかなりの片寄りがあるので、私の検証セットのような網羅的な testcases が追加されるのは、意味のあることだと思われる。

また、従来の testsuite の testcases は単一の処理系しか想定しておらず、GNU C 3 の testcases は GNU C 2 / cpp にさえも適用できないものが多いが、私の testsuite 版検証セットは GNU C 2 / cpp, GNU C 3 / cpp, MCPP という 3 つのプリプロセッサのテストができるように書かれた。すなわち、testsuite で使われている DejaGnu, Tcl 等のツールの正規表現の機能を活用すれば、処理系によるプリプロセス出力の spacing の差異や診断メッセージの差異を吸収することができるのである。

### 7.3 英語版ドキュメントの作成

MCPP V.2.3 および検証セット V.1.3 の全ドキュメントの日英翻訳を、翻訳会社ハイウェル [19] に委託した。3 人のバイリンガルの担当者によって翻訳され、技術的な内容については私が修正を加え、テキストの整形を施して、英語版ドキュメントとして仕上げた。

## 8 V.2.4 への update 計画

MCPP V.2.4 では、V.2.3 でやり残したことを含めて、次のような updates を進めたい。

1. Multi-byte character, wide-character の encoding は V.2.0 以来対応してきた日本の EUC, shift-JIS、中国の GB 2312、台湾の Big5、韓国の KS C 5601 のほかに、新たに UTF-8, TRON-code, JIS にも対応させる。それによって、あらゆるプラットフォームのあらゆる処理系に移植可能なプリプロセッサとする。
2. 市販の主要な処理系に対応させる。ことに最もユーザの多い Visual C++ をとりあげ、これに検証セットを適用し、MCPP を対応させる。
3. すでに対応している処理系の新しいバージョンに対応させる。LCC-Win32, ucpp 等である。
4. MCPP のコンパイルを configure script によって自動的にできるようにする。
5. GNU C 3 / cpplib のソースプログラムと testsuite に検討を加える。
6. これらを通して MCPP と検証セットの完成度を高める。
7. これらに伴って、ドキュメントを日本語版・英語版ともに update する。
8. ドキュメントは従来のテキストファイルのほかに、UNIX 系システムのための info 形式のものと Windows 用の help 形式のものも用意する。

## 9 MCPP を知ってもらうために

MCPP は次のようにして、free software としての普及を図ってゆきたい。

1. CVS repository を開発の中心とし、web page でその案内をするとともに、他のＣプログラマの開発参加を呼び掛けてゆく。
2. 英語版ドキュメントとともに packaging して、GNU, Linux, FreeBSD 等に提起し、distribution への収録を期待する。
3. 英語の newsgroup である comp.std.c 等で紹介してゆく。
4. 雑誌で紹介する。

## 10 参加企業

全ドキュメントの日英翻訳を、有限会社ハイウェル（東京）に委託した。翻訳はすべて契約通り遂行された。

## 11 おわりに

MCPP は、DECUS cpp という古い public domain software から出発したものである。私が DECUS cpp をいじり始めたのは 1992 年のことである。実に１１年間もこの cpp をいじってきていることになる。しかし、いくら完成度を高めても、その成果を世に問う機会がなく、残念な思いをしてきた。

MCPP は「未踏ソフトウェア」に採択されて、ようやく世に出る機会を与えられた。８ヶ月間という開発期間は一人の開発者にとっては短く、大量のドキュメントの英訳の吟味にかなりの時間を要したこと等で、計画の一部をやり残す結果となったが、世界一正確で品質の優れたＣプリプロセッサを開発することができたつもりである。熟年のアマチュアプログラマとして、非力ながらもよくやったと自分では納得している。

MCPP と検証セットの旧版は vector と @nifty で公開してきた。

2002/08 には MCPP と検証セットの CVS repository が開設された [18]。開発中の revision を含めて最新版はここに置かれている。

次のところからは anonymous ftp できる。

```
ftp://ftp.m17n.org/pub/mcpp/
```

また、次のところに案内の web page がある。

```
http://www.m17n.org/mcpp/
```

多くのＣプログラマのコメントと開発参加をいただければ幸いである。

## 参考文献, URL


[1] 情報処理振興事業協会「平成１４年度未踏ソフトウェア創造事業」
http://www.ipa.go.jp/NBP/14nendo/14mito/

[2] ISO/IEC. *ISO/IEC 9899:1990(E) Programming Languages – C*. 1990.

[3] ISO/IEC. *ibid. Technical Corrigendum 1*. 1994.

[4] ISO/IEC. *ibid. Amendment 1: C integrity*. 1995.

[5] ISO/IEC. *ibid. Technical Corrigendum 2*. 1996.

[6] ISO/IEC. *ISO/IEC 9899:1999(E) Programming Languages – C*. 1999.

[7] ISO/IEC. *ibid. Technical Corrigendum 1*. 2001.

[8] ISO/IEC. *ISO/IEC 14882:1998(E) Programming Languages – C++*. 1998.

[9] Martin Minow, DECUS cpp.
http://sources.isc.org/devel/lang/cpp-1.0.txt

[10] J. Roskind, JRCPPCHK. 現在は所在不明

[11] Borland International Inc., ボーランド株式会社. *Borland C++ V.4.0*. 1994.

[12] DJ Delory, DJGPP 1.12 m4.
http://www.vector.co.jp/soft/dos/prog/se016978.html

[13] きだあきら, LSI C-86 / cpp 改造版 beta13. かつて *C Magazine* 誌の付録 CD-ROM に入っていた

[14] Borland Software Corp., ボーランド株式会社., Borland C++ Compiler 5.5
http://www.borland.co.jp/cppbuilder/freecompiler/

[15] Jacob Navia., LCC-Win32.
http://www.q-software-solutions.com/lccwin32/

[16] Thomas Pornin., ucpp.
http://pornin.nerim.net/ucpp/

[17] Free Software Foundation, GCC.
http://gcc.gnu.org/

[18] 松井　潔, MCPP V.2.3.
http://cvs.m17n.org/cgi-bin/viewcvs/?cvsroot=matsui-cpp

[19] 有限会社ハイウェル.
http://www.highwell.net/