

An Evaluation of the Usefulness of Compiler Error Messages

Michael W. Bigrigg*, Russell Bortz, Shyamal Chandra, David Reed,
Jared Sheehan, and Sara Smith

*Contact Information:
Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh PA 15217
bigrigg@cmu.edu

Abstract

One of the most important parts of computer programming is the compiler. It is the mechanism that enables a programmer to control the behavior of a computing system by means of a high level language, as opposed to having to write in machine code or even binary. The compiler is designed to be a language translator between the programmer and the machine. This report analyzes how the compiler conveys errors to programmer

Background

Modern compilers are typically divided into two phases, the front end and the back end. The front end involves machine independent processes that convert the input program into an intermediate representation. This intermediate representation is then fed into the back end, which performs optimizations and machine dependent code generation (either machine assembly for natively compiled programs or an interpreter format). [1] However, most of the compiler-user interaction is in the front end, and will be of the most interest in this discussion.

The front end itself contains three sub-phases. The first of these is lexical analysis where the program is scanned character by character and produces a list of tokens or terminals. These items represent the fundamental building blocks of a program's written form. This is also analogous to the vocabulary of a natural language. There are lexical specification forms that aid in making the scanner or lexical analyzer of a compiler. This specification is typically in the form of regular expressions, which provides only simple error checking such as constant integer values being out of bounds, un-terminated multi-line comments, and illegal characters (as seen in C/C++). [3]

The second sub-phase is known as syntax analysis or parsing. In this phase, the tokens generated from lexical analysis are fed into the parser, which formulates, based on a set of rules called a grammar, a binary parse tree representing valid language statements. This phase is most equivocal to the formulation of proper sentences in a natural language. An interesting point about parsers is that there are different methods used in performing the actual analysis. Two major branches of parsing are $LL(k)$ and $LR(k)$. LL , standing for left to right, leftmost derivation, is considered a more primitive form of parsing and can be traced by hand easily for verification purposes. LR , standing for left to right, rightmost derivation is an improvement on the LL parsers and permits the parsing of the most grammars, however, is typically difficult to trace by hand. The benefit of LR parsers is that they detect, early on, syntax errors, such as improper expression formation. [2]

The last sub-phase of importance is semantic analysis. In this part, the parse tree generated from the previous step is examined (typically by walking up and down the tree) and includes data type checking (verifying variables are assigned to correct type of expression), object protection enforcement, etc. Semantic analysis is typically implemented by hand and not through an intermediary specification and generation process. At any rate, all of these phases are capable of catching and notifying the programmer of incorrect code (errors) and potential problems (warnings).

Scope and Motivation

With C and C++ being one of the more commonly used programming languages, this report, therefore, focuses on the compilation of a set of standard C/C++ programs that have known errors in the three phases mentioned previously. What is of importance in this report is the quality of the errors reported

by different compilers. These messages are defined by the writer of a compiler in the lexical, syntax, and semantic phases, due to a lack of adherence to the rules of the language. This includes using incorrect keywords, programming incorrect syntax, and performing invalid type casting between objects or primitives. Ideally, these messages should provide enough information to assist the programmer in producing valid code that adheres to the rules defined by the language he/she is programming in.

There are multiple compilers for the C/C++ language for different platforms and different levels of development. During the build process of an application, it is a time consuming process (especially for large applications) to determine what caused a compiler failure. Many times, much of development is spent fixing code to be successfully compiled as well as time being spent understanding the compiler errors that arise. The motivation behind this report is to find what compilers prove the most useful when notifying the user of an error and which are the poorest and performing this task. A total of sixteen test programs were created, each with a unique error. These sixteen programs were compiled using a variety of different compilers on different platforms. An in depth analysis is performed to rigorously compare these compilers against one another with the ultimate result of finding the “best” compiler in terms of complete and useful error message reporting.

Compilers Tested

[Bloodshed Dev-C++ 5.0 beta 8 \(4.9.8.0\) \(12 MB\) with Mingw/GCC 3.2 Dev-C++](#)

[Bloodshed Dev-C++](#), authored by [Colin Laplace](#), [Hongli Lai](#), [Yiannis Mandravellos](#) and [Mike Berg](#) is a full-featured Integrated Development Environment (IDE) for the C/C++ programming language. It is free software, written in Delphi. It uses [Mingw](#) port of [GCC](#) as its compiler. It can also be used in combination with Cygwin. Dev-C++ creates console and GUI Win32 (The Dev-C++ Resource Site).

[Borland C++ Version 5.02, Copyright© 1991, 1997, Borland International, Inc.](#)

Borland C++ is a tool for developing C++ applications for Windows 3.1, Windows 95 and Windows NT, by targeting both 32- and 16-bit platforms. It includes a “fully programmable ObjectScripting IDE.” Users can choose from either OWL or MFC environments, with the source code included (Borland C++ 5.02 Questions and Answers).

[Borland C++Builder 5.5.1, 5, 5.5](#)

C++Builder is an ANSI C++ development environment for building Internet and distributed applications. C++Builder includes tools for creating systems to support e-commerce using HTML 4 and XML. C++Builder allows users to build, debug and deploy applications using the Visual Component Library with over 200 reusable components with source (C++Builder Studio Main Product Page).

[CC386/win32 Version 2.32.1.183 \(2003\)](#)

Ladsoft’s CC386, is a 32 bit free C compiler. CC386 is written by David Lindauer. It provides a Win32 IDE as well as a MSDOS version. While billed as a C compiler, it has all the tools needed to generate WIN32 assembly language programs including nasm, a linker, a librarian, a resource compiler, and import libraries (LadSoft cc386 Page).

[Comeau C/C++ 4.3.3 \(2002\) Comeau Computing](#)

Comeau C++ is a low cost C++ solution available on multiple platforms (Comeau’s C++ FAQ). Comeau C++ 4.3.0 is available for [MS-Windows](#) (XP, 2000, ME, NT, 98 and 95), [LINUX](#) (RedHat, SuSE, Debian, etc.), Solaris/SPARC, SunOS, NetBSD, SCO UNIX and all 3/486 SVR3’s. Some features and capabilities include Core language enhancements for all major and minor features of C++ and C, including export as well as support for additional C back ends on Windows: VC++ 7.0, Borland, Metrowerks, MinGW, lcc-win32 and Digital Mars (C++ Compilers from Comeau Computing).

[Digital Mars Compiler C/C++ Versions 8.35n, 8.36 Copyright 1995-2003 Digital Mars.](#)

The Digital Mars Compiler, formerly Zortech C++ and Symantec C++, by Walter Bright is one of the free C compilers not ported from GNU C. Digital Mars generates Win32, Win16, DOS16 and DOS32 exes in a Win32 development environment (Willus.com’s Win32 C/C++ Compilers Page: Digital Mars). Some of its features include a linker, a librarian, standard and Win32 header files, runtime linkable libraries, a standard template library, and the MicroEmacs editor (Digital Mars C/C++ Compiler – CNET-Asia).

GNU project C and C++ Compiler gcc Versions 2.95 (1999), 3.31(2003), G++ Version 3.2.3(2003)

The main Gnu Compiler Collection (GCC) distribution contains front ends for C (gcc), C++ (g++), Objective C, Fortran (g77), Java ([GCJ](#)), and Ada (GNAT) (GCC Front Ends). The GNU project's GCC version 2.95 is the first release of GCC since the April 1999 GCC/EGCS reunification and includes nearly a year's worth of new development and bugfixes (GCC 2.95). The GCC 3.3 release series includes numerous [new features, improvements, bug fixes, and other changes](#) (GCC 3.3 Release Series). The purpose of the G++ 3.2 series was released to provide a stable platform for OS distributors. A primary objective was to stabilize the C++ ABI (GCC 3.2 Release Series).

Intel C++ 7.1 for Windows and for Linux

The Intel C++ Compiler 7.1 for Windows provides compatibility with the Microsoft Visual Studio .NET environment. The compiler offers an integrated development environment as well as optimization features (Intel C++ Compiler for Windows). The Intel C++ Compiler 7.1 for Linux provides compatibility with GNU C/C++, C++ ABI conformance, gcc extensions support, and the ability to build the kernel with fewer modifications. Its optimization features provide a way to get optimal performance from applications (Intel C++ Compiler for Linux).

Lcc-win32 Wedit Win32 Version 3.3

Lcc-win32 is a C compiler system for Windows developed by Jacob Navia which can be downloaded for free. The download includes "Code generator (compiler, assembler, linker, resource compiler, librarian); integrated development environment with editor, debugger, make file generation, resource editor, etc.; User manual and technical documentation" (lcc-win32: A Compiler system for windows).

Microsoft® Visual C++® Version 6.0, Copyright© 1994-98 Microsoft® Corporation

Microsoft Visual C++ .NET, Microsoft Development Environment Version 7.1.3088 Copyright 1987-2002 Microsoft Corporation

Visual C++ is part of Microsoft Visual Studio. As part of this development tools suite it enables create applications and components for corporate use and the Internet. Visual Studio is a complete development tools suite that provides easy-to-use tools for building solutions (Microsoft® Visual C++® General FAQ, Product Information for Visual C++ .NET 2003).

Miracle C version 3.2

The Miracle C Workbench, is an IDE for the shareware Miracle C compiler written by Tadeusz Szocik. "The Miracle C Compiler runs under MS-Windows targeting the command line. All traditional C syntax is implemented, including record (struct/union) and enumerated data types, int, long and floating point data types, user type definition, bit fields in structs, initializers for all data types. There is a comprehensive library of functions; some example programs demonstrating compiler features; and Windows Helpfile documentation is supplied with the package" (Miracle C Compiler).

Open Watcom Version 1.1

"Open Watcom is a joint effort between [SciTech Software Inc](#), [Sybase®](#), and the Open Source development community to maintain and enhance the Sybase Watcom C/C++ and Fortran compiler products. Plans for Open Watcom include porting the compiler to the Linux and FreeBSD platforms, as well as updating the compilers to support the latest C and C++ ANSI standards (Open Watcom – Portable Compilers and Tools). Open Watcom C/C++ provides an integrated development environment (IDE) with the tools, SDK's, and libraries needed to create powerful 16- and 32- bit applications. Open Watcom C/C++ supports Windows 95/98/Me, Windows NT/2000/XP, OS/2, extended DOS and Novell Netware (Open Watcom C/C++ and FORTRAN).

Portland Group C/C++

"The Portland Group Compiler Technology team originates from the acquisition by STMicroelectronics of The Portland Group, Inc. (a.k.a. PGI). The Portland Group Compiler Technology team offers high performance scalar and parallel Fortran, C and C++ compilers and tools for 32-bit x86 (AMD AthlonXP, Intel Pentium 4 and Xeon) and 64-bit AMD64 technology (AMD Opteron) processor-based workstations, servers and clusters (About The Portland Group Compiler Technology)." "PGCC® Workstation includes The Portland Group Compiler Technology's native parallelizing/optimizing C++ (Linux-only) and ANSI C

compilers for 32-bit x86 and 64-bit AMD64 technology Linux and W98/WNT/W2K workstations. All C++ functions are compatible with Fortran and C functions, so you can compose programs from components written in all three languages. C++ function overloading, function in lining, multiple inheritance, and templates are all fully supported (PGCC Workstation).”

Reads51 Version 4.20, Rigel Corporation

“Reads51 is an integrated applications software development system, which runs on an IBM PC or compatible host. Reads51 allows writing, compiling, assembling, debugging, downloading, and running applications software in the MCS-51 language. Reads51 contains a C compiler, relative assembler, linker/locator, editor, chip simulator, assembly language debugger, and host-to-board communications in a user-friendly, menu-driven environment” (Reads51).

SoftIntegration CH 4.0 Standard (2003) SoftIntegration, Inc.

Ch is an embeddable C/C++ interpreter for cross-platform scripting, shell programming, 2D/3D plotting, numerical computing, and embedded scripting (Frequently Asked Questions for Ch). Ch 4.0, is SoftIntegration’s newest version. Ch Standard Edition offers a very high-level language (VHLL) environment for general-purpose computing supporting an increasing number of C/C++ libraries and software packages. Virtually any C programs can readily run in the Ch language environment across different platforms without lengthy compile/link/execute/debug cycles (The SoftIntegration Ch standard edition).

Web-based Compiler System University of North Texas

The Web-based Compiler System, developed by Don Retzlaff and Thomas Irby of the University of North Texas can be found at <http://www.cs.unt.edu/~irby/wbcRoden/wbcForms.html>. “The Web-based Compiler System is designed to simplify the beginning student's interface to developing simple C++ programs to help the student understand how to develop programs in C++, as well as provide an easy means to let the student experiment and play with the world of possibilities provided by the C++ programming language” (Web-based Compiler System). While this is more of a debugger than a compiler since it does not produce the compiled file if you are not part of the class that it is for, it is very helpful since the messages it gives are really meaningful.

Compiler Ratings

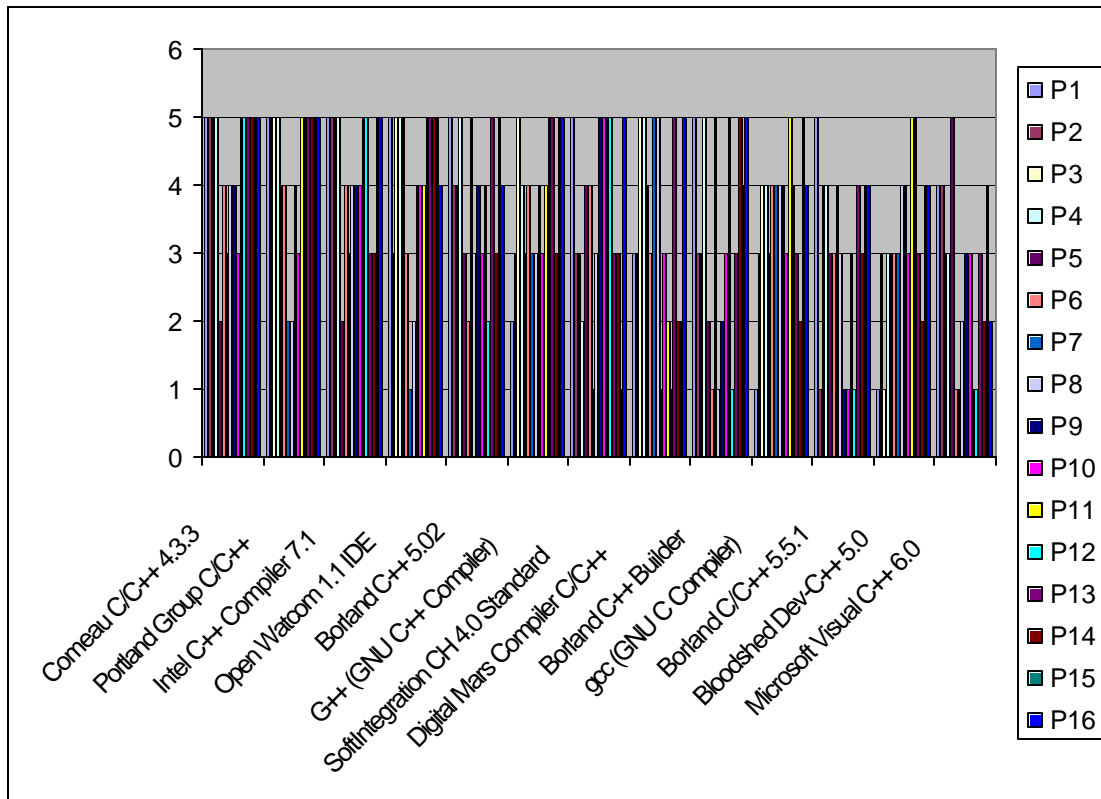
Clarity and Accuracy

This score signifies the comprehensibility of the error message. Sometimes this score will be good even if the error message is hard to understand if the only way to explain the error message is the way the compiler explained it. This score also takes into account the correctness of the error message. If the Error message is the correct one and at the correct place then the compiler should get a perfect score for that particular program. We will give a 4 for a correct answer without warnings and a 5 for a correct answer with warnings that are at other possible answers to the error message. For instance the first program is missing a ‘;’. If the compiler tells me that I am missing a ‘;’ but it could also be a missing ‘}’ then they get a 5.

Tilt

This score basically allows a reviewer sway the final score--either higher or lower--based on the reviewer's experience with a compiler. For example, a compiler might be accurate but is difficult to use.

Compiler	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	Tilt	Total	Rank
Comeau C/C++ 4.3.3	5	5	5	5	2	4	3	4	4	3	5	5	5	5	5	5	20	90	1
Portland Group C/C++	5	5	5	5	4	4	2	2	4	3	5	5	5	5	5	5	20	89	2
Intel C++ Compiler 7.1	5	5	5	5	2	4	3	4	4	4	5	5	3	3	5	5	20	87	3
Open Watcom 1.1 IDE	5	3	5	5	5	3	1	2	4	4	4	5	5	5	5	4	16	81	4
Borland C++ 5.02	5	4	4	5	3	2	5	3	4	3	4	2	5	3	5	4	15	76	5
G++ (GNU C++ Compiler)	2	3	5	4	3	4	3	3	4	3	4	5	5	3	5	5	6	67	6
SoftIntegration CH 4.0 Standard	5	3	3	2	4	4	1	3	5	5	5	5	3	3	1	5	10	67	7
Digital Mars Compiler C/C++	3	3	5	5	4	3	5	5	1	3	2	1	5	2	2	5	10	64	8
Borland C++ Builder	5	3	3	5	2	1	5	1	2	3	5	1	3	5	4	5	10	63	9
gcc (GNU C Compiler)	1	3	4	4	3	4	4	4	4	3	5	4	3	2	5	4	6	63	10
Borland C/C++ 5.5.1	5	1	4	4	3	3	4	3	1	1	3	1	4	3	4	4	7	55	11
Bloodshed Dev-C++ 5.0	1	3	1	3	3	3	3	4	4	3	5	5	3	2	4	4	2	53	12
Microsoft Visual C++ 6.0	4	4	3	3	5	1	1	2	3	3	1	1	3	2	4	2	8	50	13
Average	3.9	3.5	4	4.2	3.3	3.1	3.1	3.1	3.4	3.2	4.1	3.5	4	3.3	4.2	4.4	11.54		
Maximum	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5	20		
Minimum	1	1	1	2	2	1	1	1	1	1	1	1	3	2	1	2	2		



Conclusion

The best compiler may depend on the user's experience of that compiler. When clear and concise error messages are displayed, the compiler is much easier to use and this cuts down on the debugging time tremendously. As illustrated throughout these test cases, each compiler has at least one case where it out-

performs the others. Therefore, it may be useful to combine their debugging powers by using them all. Each of the compilers had particular strong points and proved to be the best compiler at catching certain errors in certain programs. It is difficult if not impossible to easily say which compiler is the best because the results were extremely varied, and each compiler did excellent in particular areas.

References

[1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles, Techniques and Tools. Addison-Wesley. 1986.

[2]] Sebesta, Robert W. Concepts of Programming Languages – 6th Edition. Addison-Wesley Publishing. Massachusetts. 2003.

[3] <http://en2.wikipedia.org/wiki/Compilers>

Web Bibliography

“About The Portland Group Compiler Technology.” <http://www.pgroup.com/about/index.htm> November 2003.

“Borland C++ 5.02 Questions and Answers.” <http://www.tietovayla.fi/BORLAND/CPLUS/BC5/qanda52.html>. November 2003.

“C++Builder Studio Main Product Page.” http://www.borland.com/cbuilder/cbuilder_5/index.html. November 2003.

“C++ Compilers from Comeau Computing.” <http://www.comeaucomputing.com> November 2003.

“Comeau’s C++ FAQ.” <http://www.comeaucomputing.com/faqs/genfaq.html#tellme>. November 2003.

“The Dev-C++ Resource Site.” <http://www.bloodshed.net/dev/index.html>. October 2003.

“Digital Mars C/C++ Compiler – CNET-Asia.” <http://asia.cnet.com/downloads/pc/swinfo/0,39000587,39035760s,00.htm> November 2003.

“Frequently Asked Questions for Ch.” <http://www.softintegration.com/support/faq/general.html>. November 2003.

“GCC 2.95.” <http://www.gnu.org/software/gcc/gcc-2.95>. November 2003.

“GCC 3.2 Release Series.” <http://gcc.gnu.org/gcc-3.2>. November 2003.

“GCC 3.3 Release Series.” <http://www.gnu.org/software/gcc/gcc-3.3>. November 2003.

“GCC Front Ends.” <http://gcc.gnu.org/frontends.html>. November 2003.

“Intel C++ Compiler for Linux.” <http://www.intel.com/software/products/compilers/clin>. November 2003.

“Intel C++ Compiler for Windows.” <http://www.intel.com/software/products/compilers/cwin>. November 2003.

“LadSoft cc386 Page.” <http://members.tripod.com/~ladsoft/frindx.htm?cc386.htm> October 2003.

“lcc-win32: A Compiler system for windows by Jacob Navia.” <http://www.cs.virginia.edu/~lcc-win32/>. October 2003.

“Microsoft Visual C++ General FAQ.” <http://msdn.microsoft.com/visualc/previous/vc6/faq.aspx>.
November 2003.

“Miracle C Compiler.” <http://www.c-compiler.com>. October 2003.

“Open Watcom C/C++ and FORTRAN.” http://www.openwatcom.org/product/features_content.html.
November 2003.

“Open Watcom – Portable Compilers and Tools.” <http://www.openwatcom.org>. November 2003.

“PGCC Workstation.” <http://www.pgroup.com/products/workpgcc.htm>. November 2003.

“Product Information for Visual C++ .NET 2003.” <http://msdn.microsoft.com/visualc/productinfo>.
November 2003.

“Reads51.” <http://www.rigelcorp.com/reads51.htm>. September 2003.

“The SoftIntegration Ch standard edition.” <http://www.softintegration.com/products/chstandard>.
November 2003.

“Web-based Compiler System.” <http://webcpp.csci.unt.edu>. October 2003.

“Willus.com's Win32 C/C++ Compilers Page: Digital Mars.” <http://www.willus.com/ccomp.shtml?p05>.
November 2003.

Test Programs

Test1.c

```
main()
{
    int i;
    i = 1 /* ; */
}
```

Test2.c

```
main()
{
    10 = 10;
}
```

Test3.c

```
main()
{
    int i;
    j = i;
}
```

Test4.c

```
main()
```

```
{
    int i, i, j;
    j = i;
}
```

Test5.cpp

```
class foo {
    private:
    int i;
};

main()
{
    foo f;
    f.i = 12;
}
```

Test6.c

```
int i, j[10];

main()
{
    j = i;
}
```

Test7.c

```
foo(int);

foo(char a) {
    return;
}

main()
{
    int i;
    foo(i);
}
```

Test8.c

```
int a[-10];

main()
{
}
```

Test9.c

```
struct {  
    int a;  
} foo;
```

```
main()  
{  
    foo.b = 12;  
}
```

Test10.c

```
int a[10];  
struct { int b; } foo;
```

```
main()  
{  
    a[foo] = 12;  
}
```

Test11.c

```
main()  
{  
    printf ("hi");  
}
```

Test12.c

```
main()  
{  
#if FOO  
  
}
```

Test13.c

```
main()  
{  
    int i;  
  
    switch (i) {  
        case 1: printf("one\n");  
        case 1: printf("two\n");  
    }  
}
```

Test14.c

```
main()  
{  
    int i;  
    struct { int a; } foo;
```

```
        switch (i) {
        case 1: printf("one\n");
        case foo: printf("two\n");
        }
}
```

Test15.cpp

```
#include <stdio.h>

void foo(short int i) {
    printf("one\n");
}

void foo (long int i) {
    printf ("two\n");
}

main()
{
    foo(1);
}
```

Test16.cpp

```
main()
{
    case 1: printf ("hi\n");
}
```