A New Book Proposal: C Compiler Design and Implementation

Hi Rachel/David,

I have another book proposal for you, this time about compiler construction. It describes the design and implementation of a C compiler (including the preprocessor, linker, and standard library) in Java, with all code given. I have some vacation coming up, and I will try to work as much as possible. In the long-term view, I also have plans to write two similar books with the code translated into C++ and C#.

Current status:

The code is (at least almost) completely finished. What remains to be done is to document the code; that is, to write the book (in MS Word). However, my experience from the previous book projects is that the major part of the job is to finish the code.

Other books in the field:

The books below are the latest editions of three classic books. They describe in detail the basic features of compiler construction and to some extent advanced features. The structure of these books is similar to mine, one can say that I have followed these books. However, while these books give a large number of small examples, my book describes in each chapter the features necessary to be included in a C compiler.

[Alfred V. Aho](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Alfred%20V%20Aho), Lam, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers – Principles, Techniques, and Tools, 2nd edition. [Prentice Hall](http://www.bokus.com/cgi-bin/product_search.cgi?publisher=Prentice%20Hall), 2006.

[Keith Cooper](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Keith%20Cooper), Linda Torczon. Engineering a Compiler, 2nd edition. Morgan Kaufman, 2011.

[Charles N. Fischer](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Charles%20N%20Fischer), Ron K. Cytron, Richard J. LeBlanc. Crafting a Compiler. [Pearson Education](http://www.bokus.com/cgi-bin/product_search.cgi?publisher=Pearson%20Education), 2009.

These books describe compiler construction in Java, C, and ML. They describe a compiler for a smaller language with excerpt from the code given in the book (the complete code is downloadable). However, these books are briefer than the books above, and describe the compiler features in less detail.

[Andrew W Appel](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Andrew%20W%20Appel). Modern Compiler Implementation in Java, 2nd edition. [Cambridge University Press](http://www.bokus.com/cgi-bin/product_search.cgi?publisher=Cambridge%20University%20Press), 2002.

[Andrew W Appel](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Andrew%20W%20Appel). Modern Compiler Implementation in C, 2nd edition. [Cambridge University Press](http://www.bokus.com/cgi-bin/product_search.cgi?publisher=Cambridge%20University%20Press), 2004.

[Andrew W Appel](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Andrew%20W%20Appel). Modern Compiler Implementation in ML, 2nd edition. [Cambridge University Press](http://www.bokus.com/cgi-bin/product_search.cgi?publisher=Cambridge%20University%20Press), 2004.

The following book could be said to be closely related to mine, as it also described the code for a C compiler. However, this book is more or less unreadable. It presents the code, which (in my option) is unstructured, and it does not include much text describing the code. Hopefully, my book will describe the topic in a clearer way. I also plan to describe the theory of a compiler, not just the code to implement it. Moreover, this book does not include the preprocessor, linker, or standard library, and the code is written in C, not Java.

Christopher W. Fraser, David R. Hanson. A Retargetable C Compiler : Design and Implementation. Benjamin Cummings, 1995.

This books deals with the more advanced parts of compiler construction; that is, compiler optimization. Even though I will include some optimization at the end of my book, it does not compete with this book.

[Steven S. Muchnick](http://www.bokus.com/cgi-bin/product_search.cgi?authors=Steven%20Muchnick). Advanced Compiler Design & Implementation. [Morgan Kaufmann](http://www.bokus.com/cgi-bin/product_search.cgi?publisher=MORGAN%20KAUFMANN), 2003.

The chapters:

1. Introduction. Introduces the compiler phases by demonstrating a compiler for a small toy language generating MIPS-code executable in the SPIM simulator.

2. The Scanner. The scanner is a relatively small part of the compiler. Its task is to put together characters into tokens, the smallest significant parts of the source code. Examples are as key words, operators, and numerical values. The scanner is written in JLex, a lexical generator for Java, based on Lex for C.

3. The Parser. The parser, on the other hand, is a large part of the compiler. Its task is to confirm that the given tokens (generated by the scanner) agree with the syntax of the programming language, which is represented by a series of grammatical rules. The parser is defined in CUP, a syntactical generator for Java, based on Yacc for C. Each rule can also be equipped with code dealing with type checking and target code generation. However, I have tried to omit as much as possible of the code in this chapter. Most of the code is made up of calls to methods defined in later chapters.

4. Declarations and the Symbol Table. C has a rather complicated declaration system with aggregated types such as structs and arrays with a corresponding complicated syntax. All defined variables and functions are stored in the symbol table, which is a hierarchical structure matching the program structure.

5. Type Checking. C has a rather large set of operators with complicated rules that need to be checked.

6. Intermediate Code Generation. When the types of expressions and statements are checked, three-address-code are generated, which is a simple intermediate language used to represent the code internally. Type conversation is also included in this chapter.

7. Static and Dynamic Variable Initialization. In C, variables can be initialized. If the variable is static, the data shall be generated and placed in the static area of the final target code. If it is dynamic (auto) the initialization will result in a series of assignments. One thing that complicates the issue is that it is possible to initialize hierarchical structures made up by structs and arrays with one flat list.

8. Intermediate Code Optimization. The intermediate code can be optimized. For instance, code that is never reached and assignment of variables that are never used shall be removed.

9. Target Code Generation. The target code of the C compiler of this book is Intel x86, which is harder to deal with than the MIPS code in chapter 1. It holds a few registers and registers of different sizes overlap. Therefore, the register allocation process needs to closely keep track on which variable values that are currently stored in the registers.

10. The Preprocessor. Before the actually compilation starts, the source code has been traversed by the preprocessor, which replaces macros with text, includes header files, and provides conditional programming.

11. The Linker. When the target code has finally been generated, it becomes stored into an object file. As the source code can be distributed over several files, the target code need to be merged into one executable file. This is the task of the linker, it merges the code and data area, resolve the static and extern references and generate a file of executable code.

12. The Standard Library. Finally, the C standard includes a standard library of functions and macros, which can be included by the linker in the final executable file.

Best Regards,

Stefan Björnander

Pettersbergsgatan 14, 10 tr

724 63 Västerås

Sweden

stefan.bjornander@crosscontrol.com

Phone +46 701 645406, +46 705 195198