

The book was found

Evaluating Expressions

```
ExpressionNode ParseFactor( )
{
    Token lookahead = m_Lexer.Peek();
    ExpressionNode factorNode = null;
    switch( lookahead.Type )
    {
        case TokenType.UnaryNegation:
            Accept();
            factorNode = new UnaryNegateOperatorNode( ) { Target = ParseFactor( ) };
            break;
        case TokenType.Numbers:
            Accept();
            var factor = new NumberNode( ) { Token = lookahead };
            factorNode = ParseImplicitMultiplication( factor );
            break;
        case TokenType.Literal:
            factorNode = ParseLiteral( );
            break;
        case TokenType.OpenParen:
            factorNode = ParseSubExpression( );
            break;
    }
}
```

def defState1() { ... }
def defState2() { ... }
def defState3() { ... }
def defState4() { ... }
def defState5() { ... }
def defState6() { ... }
def defState7() { ... }
def defState8() { ... }
def defState9() { ... }
def defState10() { ... }
def defState11() { ... }
def defState12() { ... }
def defState13() { ... }
def defState14() { ... }
def defState15() { ... }
def defState16() { ... }
def defState17() { ... }
def defState18() { ... }
def defState19() { ... }
def defState20() { ... }
def defState21() { ... }
def defState22() { ... }
def defState23() { ... }
def defState24() { ... }
def defState25() { ... }
def defState26() { ... }
def defState27() { ... }
def defState28() { ... }
def defState29() { ... }
def defState30() { ... }
def defState31() { ... }
def defState32() { ... }
def defState33() { ... }
def defState34() { ... }
def defState35() { ... }
def defState36() { ... }
def defState37() { ... }
def defState38() { ... }
def defState39() { ... }
def defState40() { ... }
def defState41() { ... }
def defState42() { ... }
def defState43() { ... }
def defState44() { ... }
def defState45() { ... }
def defState46() { ... }
def defState47() { ... }
def defState48() { ... }
def defState49() { ... }
def defState50() { ... }
def defState51() { ... }
def defState52() { ... }
def defState53() { ... }
def defState54() { ... }
def defState55() { ... }
def defState56() { ... }
def defState57() { ... }
def defState58() { ... }
def defState59() { ... }
def defState60() { ... }
def defState61() { ... }
def defState62() { ... }
def defState63() { ... }
def defState64() { ... }
def defState65() { ... }
def defState66() { ... }
def defState67() { ... }
def defState68() { ... }
def defState69() { ... }
def defState70() { ... }
def defState71() { ... }
def defState72() { ... }
def defState73() { ... }
def defState74() { ... }
def defState75() { ... }
def defState76() { ... }
def defState77() { ... }
def defState78() { ... }
def defState79() { ... }
def defState80() { ... }
def defState81() { ... }
def defState82() { ... }
def defState83() { ... }
def defState84() { ... }
def defState85() { ... }
def defState86() { ... }
def defState87() { ... }
def defState88() { ... }
def defState89() { ... }
def defState90() { ... }
def defState91() { ... }
def defState92() { ... }
def defState93() { ... }
def defState94() { ... }
def defState95() { ... }
def defState96() { ... }
def defState97() { ... }
def defState98() { ... }
def defState99() { ... }

Evaluating Expressions

(1 + 2) * 3

```
graph TD
    Multiply --> Add
    Multiply --> Number3[3]
    Add --> Number1[1]
    Add --> Number2[2]
```

A Hands on Introduction
Lexing, Parsing, and Interpreters with C#

Richard Weeks



Synopsis

In Computer Science, the Art of Compiler writing is still considered somewhat of a Black Art; known only to the masters. Cryptic texts, enormous amounts of theory, more acronyms and notations than anyone should have to know are all barriers to entry in learning the basics of even implementing a simple language. Over the years, I've read just about every compiler book I could get my hands on and in each case I found myself somewhat surprised by the amount of assumed knowledge the reader was supposed to already have under their belt. Now, don't get me wrong, there are plenty of great authors and educators on the topic of Compiler design, and I've learned a tremendous amount in reading their works. Still, I personally feel there is a gap; a step 1 if you will, somewhere between basic parsing and full-blown Compiler design. It is this gap I hope to fill in the pages that follow. In order to provide a better understanding of what this book is about, let me first start by stating what this book is not about. This book is not an exploration of parsing techniques and error recover, formal language grammars, optimized code generation, or even efficient data structures. No, this book is about the basics; the core ideas required for parsing and interpreting a language. We'll use a language we are all familiar with – “ Math Expressions ” – a language that we have a common understanding of what the basic symbols mean and the order of operations are. To me, this eliminates having to learn the rules of a toy language first and then having to work out the strategy to implement said toy language. Additionally, we'll only be working with a small sub-set of the language, basic operators such as addition and subtraction, multiplication and division, grouping operators, negation, factorial and exponents. From there we'll move on to variables and functions. Once we've got the basics covered, we'll create a very simple math oriented language which allows for variable and function definitions, evaluation and printing out the results. I chose math expressions due to the sheer number of user posts on various public forums asking the very question: “ How do I evaluate a user supplied math expression? ” This question arises so frequently and so often with so many different answers and implementations, I decided to write this book as an attempt to hopefully ignite the spark in you to want to learn more about the subject of Compiler and Interpreter construction. Lastly, I want to point out that there are a number of ways in which to implement an evaluator for math expressions. These approaches range from Reverse Polish Notation (RPN) stack based evaluators or the Shunting-yard algorithm by Edsger Dijkstra, all the way to byte code generation and executed via virtual machines. In the end, your solution should be driven by the requirements, and not the other way around.

Book Information

File Size: 2448 KB

Print Length: 296 pages

Publication Date: December 31, 2012

Sold by: Digital Services LLC

Language: English

ASIN: B00AVNULRI

Text-to-Speech: Enabled

X-Ray: Not Enabled

Word Wise: Not Enabled

Lending: Enabled

Enhanced Typesetting: Enabled

Best Sellers Rank: #463,629 Paid in Kindle Store (See Top 100 Paid in Kindle Store) #45

in Books > Computers & Technology > Programming > Languages & Tools > Compiler Design

#129 in Books > Computers & Technology > Programming > Languages & Tools > Compilers

#283 in Kindle Store > Kindle eBooks > Computers & Technology > Programming > C & C++

Customer Reviews

Well written and provides a comprehensive description of the recursive decent parser. The step by step evolution of the parser and execution stage to evaluate more and more complex expressions is very well laid out.

Great content. Well presented and supported with examples.A+

[Download to continue reading...](#)

Evaluating Expressions Experiences of Special Education: Re-evaluating Policy and Practice through Life Stories Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research (2nd Edition) Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research, Enhanced Pearson eText with Loose-Leaf Version -- Access Card Package (5th Edition) Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research (5th Edition) Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research (4th Edition) Research Methods in Psychology: Evaluating a World of Information (Second Edition) Research Methods in Psychology: Evaluating a World of Information Negotiating Construction Law Disputes: Leading Lawyers on Evaluating Disputes, Assessing Risks, and Deciding the Best Course of Action (Inside the Minds)

Before and After Hinckley: Evaluating Insanity Defense Reform Interpreting Evidence: Evaluating Forensic Science in the Courtroom Evaluating and Resolving Wrongful Death and Personal Injury Cases Planning, Implementing, & Evaluating Health Promotion Programs: A Primer (7th Edition) Planning, Implementing, & Evaluating Health Promotion Programs: A Primer (6th Edition) From Cover to Cover (revised edition): Evaluating and Reviewing Children's Books Evaluating Drug Literature: A Statistical Approach Fundamentals of Government Information: Mining, Finding, Evaluating, and Using Government Resources The College Student's Research Companion: Finding, Evaluating, and Citing the Resources You Need to Succeed, Fifth Edition Snow Sense: A Guide to Evaluating Snow Avalanche Hazard Mastering Regular Expressions

[Dmca](#)