CPSC 425 Compiler

C*13 Compiler

By Joel Knudsen and Bryan Lansdowne

September 20, 2013

Introduction

For this phase of the project we are submitting the Scanner code used to tokenize the source language.

Participation

Code was done jointly, Joel laid out the framework, Bryan produced a rough working copy, Joel reworked the rough copy and was responsible for much of the current version of the code.
Testing was done jointly with Bryan conducting the final round of testing.
Documentation was primarily written by Bryan while the images were provided by Joel.

Project Status

Our scanner is completed to the point of testing and is ready to be integrated with other components of the compiler. We have a parser module that simply invokes the Scanner. The Scanner has a tracer built in that when turned on will display the tokens and values found as it runs.

As this is the first phase no previous phases were altered. In the next phase we will produce the parser code such that the parser will produce a parse tree from the tokens obtained from the Scanner.

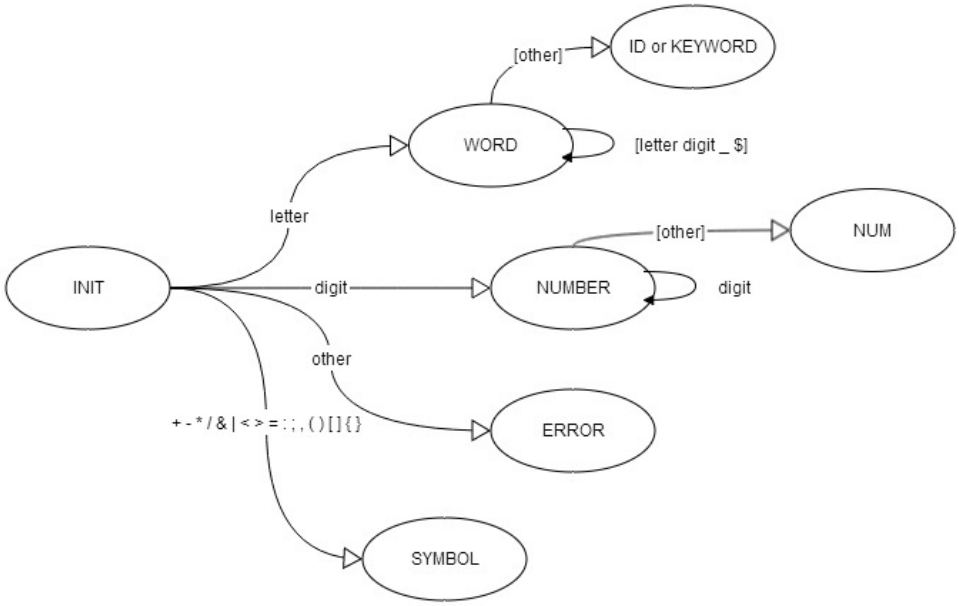
Architecture and Design

Our compiler is written in Java for familiarity of the language.

Our compiler currently consists of Main, Scanner, Parser, and Error/Utility modules. More modules will be added as the project continues.

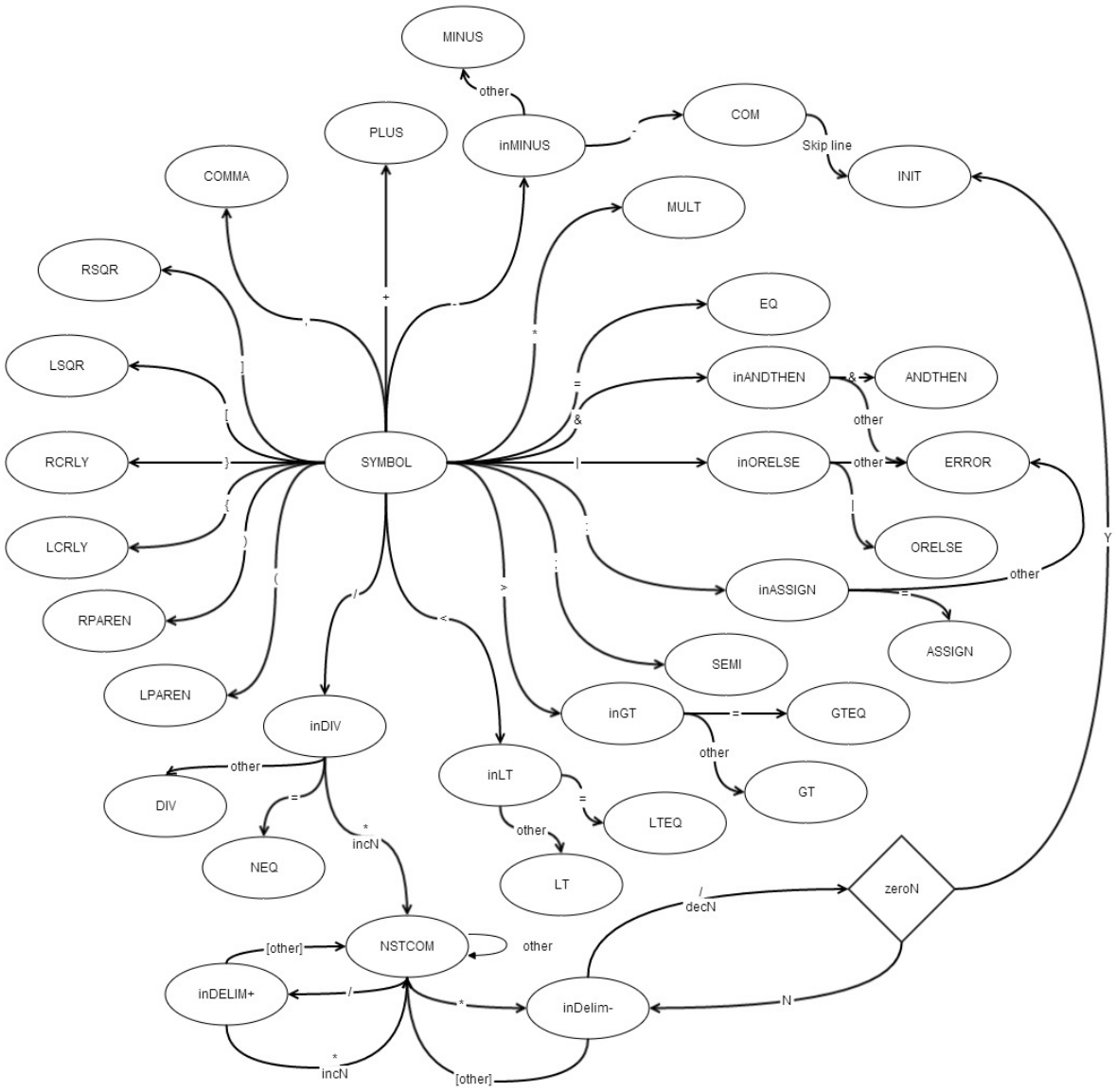
- The Main module is responsible for starting the compiler and preferences of the program, such as choosing source files to compile and target file names.
- The Scanner module is responsible for reading the source file and providing tokens upon request to the Parser.
- The Parser is currently only capable of requesting tokens from the Scanner. In the next stage the parser will construct a Parse tree to pass further up the chain of modules.
- The Error/Utility Module decodes and displays error messages from the other modules.

The scanner uses a state diagram as follows:



[Fig 1]

The WORD state uses a map to determine whether the word is a keyword or an ID. The symbol state is more in depth:



[Fig 2]

Implementation

Our compiler currently is only capable of tokenizing the C*13 language at this phase.

The compiler is capable of standard Java integer size, however it uses a signed integer to store an unsigned integer of max size 2,147,483,647.

The Scanner uses recursive methods to identify longest substrings, as such extremely long Identifier names may cause stack overflows.

Build and Use
Our compiler was coded in Java 7 and built using Eclipse Kepler.

The compiler is run from the console with the following 2 parameters.

- “s” turns on the trace in the Scanner which prints the tokens to standard output.
- “filename” selects the file to be compiled where filename is replaced by the path of the source file. Example “C:\test.cs13”

A typical run would look like this typed into the console.

- java -jar cpsc425.jar s basic.cs13

Code
Source Files
- Main.java - Main module
- Scanner.java - Scanner module
- Token.java - Token class
- TokenType.java - TokenType enums
- Parser.java - Parser module
- ErrorUtility.java - Error/Utility module

Refer to architecture and design for what each file does.

Tests and Observations
For the initial phase of testing we used an exhaustive approach of different combinations of white space, symbols, and strings. Note that the Scanner only recognises the difference between commented code and uncommented code and valid and non-valid symbols. It does not error check syntax.

Testing was done using a source file designed to test main functionality, edge cases, and intentional errors. The source file does not resemble propper c*13 code, but rather serves to test the the recognition capable from the Scanner. The Parser will ensure legitimate syntax.

The current version of the scanner is able to correctly identify all of the tokens and combinations of tokens in the test file. The test file will continue growing as the project continues but at this point we believe we have correctly working code.