The scrim compiler

Monday 12 May 2003

CS623, Spring 2003

Marcel Levy,
Stan Sexton,
Reid Weber

http://scrim.sourceforge.net/

Table of Contents

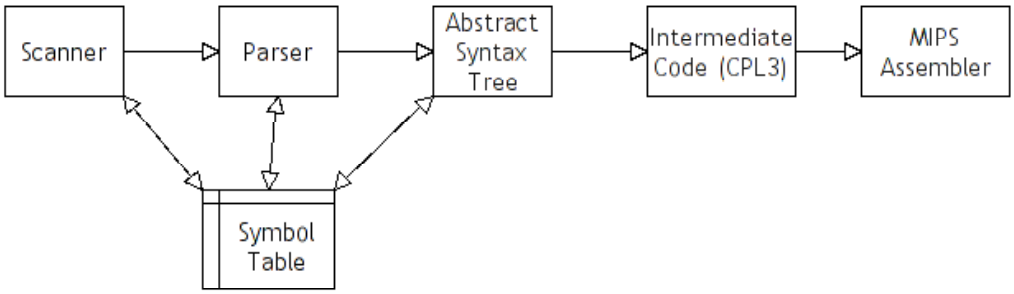
1. Overview

This document describes *scrim,* a C compiler project for CS623, Spring 2003. The scrim team members are:

- Marcel Levy
- Stan Sexton
- Reid Weber

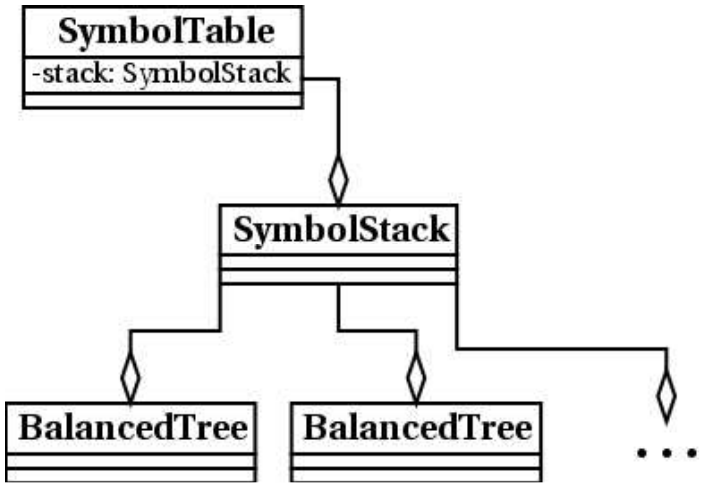
Scrim takes valid C code and produces MIPS assembler as output. we have completed all the components in **Fig. 1.**

1.1. Fig. 1: Compiler components:



The scanner is constructed using the scanner generator *flex,* and the parser is generated using *bison,* the GNU version of *yacc.*The symbol table is implemented in C++ using the class structure shown in **Fig. 2.** The symbol table stores objects subclassed from the SymbolObject class: PrimitiveSymbolObject, ArraySymbolObject, etc.

1.2. Fig. 2: Symbol table class structure



2. How to obtain and run scrim

The project is hosted at http://sourceforge.net/projects/scrim/. Tarballs containing the source code will be released in the "Files" section. The executable is built by typing "make scrimc" at the command line.

2.1. scrim usage

```
scrimc -f <file name> -d[lLys] -o <lex debug file name>  -S -q -h

    -f [filename]   Input name of file to scan. (test.c)

    -dl             Lexer debugging output consists of a
                    list of tokens and their values as
                    returned by the scanner (lexer).

    -dL             Same as -dl but with comments passed through
                    to the output file.

    -ds             Symbol table debugging output consists
                    of symbol table dumps at key points
                    during execution (for example, at the
                    beginning and end of blocks). (debug/st.scrim.xml)

    -dy             Parser debugging which outputs bison debugging info

    -o [filename]   Specify what file to direct lexer/parser debugging
                    to. (debug/lex.scrim.debug)

     -q          Generate intermediate code (debug/scrim.cpl3)
                    and AST XML output (debug/ast.xml)

    -h            Prints out this information.
```

The cpl3asm program is designed to be used as the last step in the compilation process. It operates on the three address code (cpl3) computed by scrimc, or any other cpl3 generation tool and outputs MIPS assembly language. If you run this program on a .c file your results will be suboptimal and largely undefined. The options for the cpl3asm translator are below:

```
-h Prints out the this information.
-f [filename] File name to be converted from cpl3 to assembly.
-o [filename] Specifies where the output code will be placed.
-l [language] Selects the output assembly language. The only current
              option for this is MIPS assembly. Leave blank
              to signal MIPS.
-d Dumps all cpl3 tokens to the output file.
-v Verbose. Lists input and output files.
```

2.2. File names

test.c: Default input C source code, unless overridden by the -f flag.
debug/lex.scrim.debug: Scanner output triggered by -dl flag.
debug/t1.list: Parser reductions output triggered by -dy flag.
debug/st.scrim.xml: Symbol table output triggered by -ds flag.
debug/ast.xml: AST output triggered by -q flag.
debug/scrim.cpl3: CPL3 output triggered by -q flag.

2.3. Assumptions and restrictions

Scrim will either run until it reaches the end of the source code, or until it encounters either an illegal character or a syntax error, at which point it reports the error and terminates execution. Identifiers are limited to 128 characters, and line lengths in the source code must be limited to 4096 characters.
Also, only the following items from the grammar are implemented:

1. declaring constants/types and variables, including array initializations and assigning strings to arrays.
2. generating code for simple assignments and expressions
3. generating code for the conditional (IF) statement
4. generating code for the WHILE loop
5. generating code for the array accesses and operations
6. handling declarations, generating code for procedures with single nesting levels and value parameters.

2.4. Interesting features

- Type checking for constants and variables
- Errors and warnings show line with column marked
- While loops implemented using IF and JMP statements to speed assembly generation.
- Extensive symbol class hierarchy
- Reasonably sophisticated CPL3 scoping and code-generation mechanism
- CPL3 comments preserved in assembly code
- Used AST XML output to generate SVG visualizations

3. Source Code

4. Sample Runs