Compiler Report

Ryan Child
Compiler Theory, Winter quarter 2012

Description

This compiler compiles programs written in the language specified in the file “project.pdf”, included in this package. It is written in C/C++. The scanner sequentially fetches tokens one at a time while the parser performs syntax analysis, lexical analysis, code generation and lexical / syntax error reporting.

Building

To build, verify maketools is installed and type “make” from the project directory. This will build an executable called “compiler”

To build on x64_64 architectures, the following additional packages must be installed:

libc6-dev-i386
lib32stdc++6
g++-multilib

The compiler only builds with the gcc –m32 flag. This is because the main memory and register file are an array of integers (4 bytes on 32 and 64 bit architectures) which must be able to also store pointers (8 bytes on 64-bit architectures).

Running

The compiler takes one argument, which is the source file to build. If there are no errors, the compiler generates a C-like assembly file, “<file_base>.c”. It then uses gcc to compile and link this file to the runtime, runtime.c and produces the final compiled executable, “a.out”.

Four test programs are provided. One is a comprehensive test of the compiler’s support for the language, named good.w. To demonstrate the error-handling capabilities of the compiler, three bad programs with different types of errors are also provided. They are named bad1.w, bad2.w, and bad3.w.

Implementation Notes

The grammar in the language specification (project.pdf) was re-written to eliminate left recursion, which would have caused infinite loops as my parser uses left-recursive descent. Frame and stack pointers were maintained to allow easy access to function parameters and local variables. C labels and goto statements were used for loops, if statements and function calls. Since the output of the compiler is C and not assembly, the addresses of functions (labels) are not known at compile time. They are found at runtime by prefixing “&&” to labels, which gives the address of a label. The addresses of global functions are found by the runtime before executing the program, and the addresses of local functions are pushed onto the stack each function call. To support simple error recovery, an error flag is set when there is an error so that the parser can recover and continue processing declarations and statements.