CIS*4650 (Winter 2016) Compilers

Implementation Project - Checkpoint Two

Due time: Mar. 23, 2016 by 11:59 pm.

Functionality:

For the second checkpoint, the following must also be implemented in your project:

Symbol Table
Type Checking (i.e., the major task of semantic analysis)

After Checkpoint One, your program should generate an abstract syntax tree if your input is valid; otherwise, it should detect and report syntactic errors. With the implementation of Checkpoint Two, your compiler can now detect and report semantic errors such as mismatched types in expressions, undeclared/redefined identifiers, etc. You should handle errors in the most reasonable way possible, always attempting to recover from an error and continue the entire process, without segmentation faults or core dumps. Whenever possible your program should attempt to recover from a particular error and continue the process, possibly detecting multiple errors in both syntax and semantics.

Execution and Output

Your compiler should be able to handle any possible C- programs to the point of building abstract syntax trees and construct the symbol tables that permit semantic analysis (i.e., you are not required to produce assembly language output, or any intermediate data structure beyond symbol tables and abstract syntax trees).

Errors detected by your program should be reported to stderr in a consistent and relatively meaningful way. You will find it useful to consider how other compilers you may have used report syntax errors during parsing and semantic analysis.

For this checkpoint, you are also responsible for implementing the -s command-line option (i.e., display symbol tables). Note that symbol tables change as you go through different scopes and normally we can gather all symbols at the ends of each scope. Accordingly, you can write messages when you enter and leave a scope and display all symbols for each scope just before you leave the scope. In addition, the scopes are fully nested, and thus, you should indent the information for each scope according to the nesting levels.

Documentation

Your submission should include a document describing what you or your group has done as a whole, and the contribution of each group member to the work, with emphasis on any modification made to your compiler since Checkpoint One. This does not mean you cut and

paste from the textbook or class notes. Instead, we are looking for insights into your design and implementation process, as well as an assessment of the work produced by each group member where appropriate. The group member assessment should be brief, but specific about what each member did, and how it fit in with the group effort.

The document should be reasonably detailed, about four to five double-spaced pages (12pt font) or equivalent, and should be organized with suitable headings. Some marks will be given to the organization and presentation of this document.

Test Programs

Each submission will include five original C- language programs. These test programs may be used to evaluate your own implementation. Obviously, you should submit programs that work well with your own implementations.

The test files should be named [12345].cm. Program 1.cm should compile without errors. 2.cm through 4.cm should exhibit various semantic errors (but no more than 3 per program and each file should test different aspects of your compiler). For 5.cm, anything goes and there is no limit to the number and types of errors in it.

All test files should have a comment header clearly describing the errors it contains, and what aspect(s) of the errors that your compiler is testing against.

Makefile

You are responsible for providing a Makefile to compile your program. Typing "make" should result in the compilation of all required components in order to produce an executable program: *cm*. Typing "make clean" should remove all generated files so that you can type "make" again to rebuild your program. You should ensure that all required dependencies are specified correctly in your Makefile.

Deliverables

(1) Documentation (hard-copy) should be submitted at the time of your demonstration.

(2) Electronic submission:
- All source code required to compile and execute your "*cm*" compiler program
- Makefile
- The required README file for build-and-test instructions.
- Tar and gzip all the files above before uploading it to the related dropbox in our CourseLink account by the due time.

(3) Demo: You should schedule a brief meeting of about 15 minutes with the instructor for a demo on March 24. This meeting will allow you to demonstrate your implementation and receive feedback as you embark on the next stage of your compiler construction.