CSc 453 : Programming Assignment 3

CSc 453 : Programming Assignment 3 (Type Checking and Semantic Actions)

Start Date: Tue Oct 13, 2015

Due Date: 11:59 PM, Mon Oct 26, 2015

1. General
   This assignment involves augmenting your parser to carry out some semantic actions, which involve processing declarations and managing symbol tables; checking type consistency of programs; and constructing syntax trees. The rules for processing declarations and checking type consistency are given in the C– Language Specification.
   Your program will silently accept correct inputs, and report error messages for inputs containing errors.

2. Syntax Trees

Syntax trees should be constructed a function at a time. The syntax tree for each function starts with an entry node, followed by a tree of nodes representing actions to be taken when executing that function. Each node has a field (the node descriptor) describing the kind of computation corresponding to that node, and a number of children. For example, an if-statement may translate to a node that has, apart from the node descriptor ("IF"), three children: an expr-node describing the condition to be evaluated, and two stmt-nodes representing the alternatives. An expr-node may have a field describing the operator at that node, and the appropriate number of children corresponding to subexpressions of the expression under consideration. The syntax tree node for a variable may contain a pointer to the symbol table entry for that variable. For more details regarding syntax trees, see Sec. 3.3.2 of the textbook [Loudon].

In this assignment, you will not actually do anything useful with the syntax trees you construct: the syntax tree for a function will be discarded once you leave that function. In future assignments, you will process the syntax tree to carry out code generation, dataflow analysis, and code optimization. You should strive to make your syntax tree construction as clean and modular as possible, so that future extensions are simplified.

2. Invoking Your Program
   Your program will be called compile. It will read all input from stdin, and report errors to stderr.

3. Turnin
   You should turn in the sources to your code on lectura. These should include:

The sources and headers for your scanner and parser. If you use tools such as lex, yacc, etc., you should turn in the specifications given to these tools.

A main routine that calls your parser.

A make file called Makefile: this should support at least the following:

make compile

creates an executable file in the current directory called compile that implements your parser together with auxiliary routines such as the lexical analyser, symbol table and syntax tree management routines, error handling routines, etc. This command should result in the creation of your scanner and parser from scratch, by invoking the appropriate tools on the input specifications.

Any additional material you wish to turn in. Any documentation or comments may be turned in in a file README.

To turn in your files, use the command

turnin cs453f15-assg3 files

For more information on the turnin command, try man turnin.

Note: The turnin command copies the files submitted into another directory. Because of this, programs that compile and execute without problems in your directory may not work correctly once they are turned in, because of problems with relative path names in include files and make files. Such problems are considered to be sloppiness inappropriate in a graduate course, and are liable to be penalized heavily.

The output of your program will be compared with our output using diff utility (see diff(1)). With the exception of error messages, your output must follow the specification exactly. For this reason it is recommended that you follow the specification, and instructions for turnin, closely.