Overview of Compiler

Prof. James L. Frankel
Harvard University

Version of 6:37 PM 23-Jan-2018

Lexer (PS1)

- Read input program
- Break it up into tokens
- Tokens are identifiers, reserved words, operators, separators, and constants

Parser (PS2)

- Using the stream of tokens from the Lexer, create a tree data structure that represents the complete input program
- At this point, identifiers will be simply strings; there is no symbol table and no checking of declarations

Symbol Table (PS3)

- A symbol table is created that stores all identifiers and their respective types
- The symbol table is actually a hierarchical data structure that has identifiers stored by scope (file scope, procedure/function scope, block scope)
- Your symbol table design must facilitate searching for an identifier from the point of reference through all enclosing scopes successively all the way up to global scope (that is, file scope)
- All references to identifiers (before PS3 they appear as strings) will be replaced by pointers to the appropriate entry in the symbol table
- Additionally, a statement label data structure needs to be created for each procedure so that references to statement label can be resolved
- Also, a string literal table needs to be created for all constant strings and references to those constant strings replaced by pointers to the appropriate entry in the string literal table
- Check for multiply defined identifiers in the same scope and also for references to identifiers that are not declared
- Procedure definitions must agree with prototype declarations

Semantic Analysis (PS4)

- In the semantic analysis phase, all semantic rules of the language are checked for correctness
- All appropriate conversions are applied
  - usual casting conversions
  - usual assignment conversions
  - usual unary conversions
  - usual binary conversions
  - etc.
- Implicit type conversions are made explicit
- Type errors will be output during this phase
- After this phase, the parse tree will be type-correct

Intermediate Representation Generation (PS5)

- Generate three-address IR instruction nodes
  - Nodes contain opcode, result, and two operands
- Store the IR instruction nodes in a doubly-linked list
- Assume infinite number of temporary registers
- IR instructions should assume a load/store architecture
  - The only IR instructions to access memory have some variant of load and store opcodes
- Type information should be reflected in the chosen opcode
- All expressions should first be evaluated as an lvalue, if possible, and then further evaluated to an rvalue, if an rvalue is needed

MIPS Assembly Language Generation (PS6)

- Generate MIPS assembly language from the IR nodes
- The MIPS assembly language should be acceptable to QtSpim (a MIPS emulator)
- Follow our standard calling convention
- Add predefined function declarations for the relevant system calls provided by QtSpim
  - These calls perform input and output and terminate execution
  - Special case calls to each of those functions so that the calls issue the appropriate system call

Optimization (Final Project)

- Optimizations will be performed at the IR level
  - Each optimization is an IR-to-IR transformation
  - New IR nodes may be created to enable certain optimizations
- Choose interesting optimizations that will significantly improve the MIPS code produced by your compiler
- Clean up code
- Add relevant comments that may not already be present
- All aspects of your compiler will be presented in the last class meeting
  - Ten minute prerecorded video is shown
  - Five minute Q&A session