CSC488 Assignment 5 Documentation

Ou Ye, yeou
Theresa Ma, matheres
Lawrence Wu, wulawre1
Zixuan Jaimie Jin, jinzi2
Grzegorz Wlodarek, wlodare1

New Classes

src/compiler488/codegen/Instructions.java

- This Class is used to keep track of all the instructions in list and later use to push the values into memory.

Code Generator Design

- Our code generator for A5 uses a similar recursive pattern as the semantic checker that we implemented in A3. Every AST subclass has a doCodeGeneration method that performs the code generation and calls doCodeGeneration on its children. The children will evaluate the declarations/statements/expressions and leave the result onto the stack for the parent to finish doing the code generation.
- We added additional fields and methods to symbol table related classes to accommodate the code generation extension, such as lexical level and order number to keep track of the scope of the content, a list of return statements' address in instruction memory to be filled with the address of the instruction after the routine, and lists of exit statements' address in instruction memory to be filled with the address of the instruction after each loop.
- We also modified the semantic checker to allow the symbol table to keep track of the number of variables declared in each scope during the semantic analysis phase. This allowed us to allocate the precise amount of memory at the beginning of each major scope (for both the major scope and the minor scopes enclosed in it) during the code generation phase.

Changes since A4

There are very few changes between the assignment 4 design template and the implementation of the code generation for assignment 5.

- Changed comparison operators (<= and >=) to use NOT and SWAP instead of evaluating the left and right expressions twice.

  <u>For left <= right:</u>
  % evaluate left
  % evaluate right
  SWAP
  LT
  PUSH MACHINE_FALSE
  EQ

<u>For left >= right:</u>
% evaluate left
% evaluate right
LT
PUSH MACHINE_FALSE
EQ

- Changed the not operator to use EQ and MACHINE_FALSE instead of using arithmetic operators.

  <u>For “not <operand>”:</u>
  % evaluate <operand>
  PUSH MACHINE_FALSE
  EQ

- Changed the and operator to use NOT, OR and De Morgan’s Law instead of using arithmetic operators.

  <u>For “<left> and <right>”:</u>
  % evaluate <left>
  PUSH MACHINE_FALSE
  EQ
  % evaluate <right>
  PUSH MACHINE_FALSE
  EQ
  OR
  PUSH MACHINE_FALSE
  EQ

Who did what

- Ou Ye - Test cases and implementation of the code generation methods.
- Zixuan - Implementation of the code generation methods
- Theresa - Implementation of the code generation methods
- Grzegorz - Worked on additional test cases and reviewed code for errors.
- Lawrence - Implementation of code generation methods