Programming Language Design
Department of Computer Science
Dr. Francisco Ortin
francisco.ortin@cit.ie

Programming Language Design

Laboratory 1 – Architecture of a Compiler

This document is used for self-evaluation. Its purpose is to check that the student fully understands the mini-language implementation. You do not need to send me or upload your answers. Should you have any questions, you either send me a message via Canvas Inbox or write it in the module discussion forum.

Lab 01 is different from the other labs. In the subsequent labs, you will be required to design and implement components of your compiler for evaluation. You may, of course, use the mini-compiler from Lab 01 as a reference, so you won't be starting entirely from scratch.

Please, notice that

- First, analyze the source code of the mini-compiler. Then, answer all the questions. Only after that, read the answers for self-evaluation.
- The purpose is not to just know if the student's answers are correct, but to make sure you understand the language implementation.
- If your answer is not correct, please, reopen the implementation and make sure you realize why the correct answer is not the one you thought.
- Please, notice that, for the rest of labs, the correct answer (design / implementation) will **not** be provided (including those evaluated).

Answers

1. Identify the packages that implement the different phases of the compiler (lexical, syntax and semantic analysis, and code generation).

   Lexical = `parser`

   Syntax = `parser`

   Semantic = `semantic`

   Code generation = `codegeneration`

2. Which file describes the lexical specification of the language?

   `Cmm.g4` file in the `parser` package (the `parser/CmmLexer.java` file is the implementation of the lexer, generated by ANTLR when `Cmm.g4` is passed as an input).

3. Which file describes the syntax specification of the language?

   `Cmm.g4` file in the `parser` package (the `parser/CmmParser.java` file is the implementation of the parser, generated by ANTLR when `Cmm.g4` is passed as an input).

4. In which package(s) is (are) the abstract syntax tree implemented

   In the `ast` and `types` packages.

5. What is the type (i.e., class or interface) that generalizes any node of the AST?

   `ast.ASTNode`

6. What is the concrete (non-abstract) AST node that represents the whole source program?

   `ast.Program`

7. Do you recognize the design pattern used to model ASTs?

   It is the Composite design pattern.

8. How many passes (AST traversals) are performed in the semantic-analysis phase?

   Two: Identification(Visitor) and TypeChecking(Visitor).

9. What general type models any AST traversal? Do you know the design pattern used?

   The class is `visitor.Visitor`. The design pattern is called Visitor.

10. Enumerate the different traversals defined in this particular implementation?

    Identification, TypeChecking, ExecuteCG, ValueCG and AddressCG.

11. Which is the pass (traversal) that checks that variables must be defined before their use? Which phase does that traversal belong to?

    Identification, which belongs to the semantic analysis.

12. Where is the main responsibility to infer the type of an addition expression placed? (i.e., what method should be modified if we want to change the current behavior)

    The `arithmetic` method of the `Type` class in the `ast.types` package.

13. Where is the responsibility of computing the MAPL size in bytes for each type in the source language placed?

    The `numberOfBytes` method of the `Type` class in the `types` package.

14. In code generation, why are there two different classes for pushing values and addresses of expressions?

    Because, while traversing a `Variable` node, we must know what to push. If the variable is the left-hand side of an assignment, its address is pushed; otherwise, we must push its value. Since that information (left- or right-hand side) is not stored in the `Variable` node, we define two different traversals that the parent node calls depending on the position (left or right) of the `Variable` node.

15. Which traversal directs code generation of the whole program (using other traversals)?

    The `ExecuteCG` class in the `codegeneration` package (you can see how it is the only one called in the `Main` class; i.e., `ValueCG` and `AddressCG` are not called).