UNIVERSITÉ DE LIÈGE

[INFO0085] COMPILERS

Compilers Project

Authors:
LA ROCCA Lionel (s161284) & HOGGE Louis (s192814)

August 9, 2023

Contents

1 Introduction

This report provides a broad overview of the implementation of a compiler using the Flex and Bison tools in C++. We discussed first the tools used, then the organization of the compiler and our implementation choices.

2 Tools Used

For the compiler, the following tools are used :

- Flex : A lexical analyzer generator that generates code for scanning input streams.
- Bison: A parser generator that generates code for syntax analysis and parser construction.
- C++: The programming language used for the compiler.

3 Organization of the compiler

3.1 Lexer

The lexer is defined in **lexer.lex**, for the lexer and parser we used the example provided.

We added a particular token: *EOFERROR* to return an error and stop the program when EOF.

3.2 Parser

For the parser, the AST is defined in **ast.hpp**. We used a *visitor* pattern. All the nodes are created in **parser.y**.

For the nodes:

1. We have first defined an **AstNode** that extended by **ProgramNode** the root of the AST. It is also extended by **TypeNode**, **FormalNode**, **FormalsNode ExprNode**, **BlockNode**, **ArgsNode**, **FieldNode**,**MethodNode**, **ClassBodyNode** and **ClassNode**.
2. For the expression node like the *If Then* or *While do*, they extends the **ExprNode**.

We try to respect the naming and hierarchy of the VSOP manual when defining our nodes.

To print the AST, the evaluation of the nodes start from the **ProgramNode** and go through all the AST.

There is also a boolean *checkMode*, when the Checker mode is selected, is used to print the type of the corresponding expression.

3.3 Checker

Concerning the checker part, the code is divided by pass, each pass is defined in the corresponding class.

1. *FirstPass*: The first pass go over the AST, considers class declarations (ignoring their fields and methods), records which classes are defined, and what their parents are. Then checks for cycles.
2. *SecondPass*: The second pass, go over the AST to report any use of an undefined class and to record the types of methods and fields for each class, without inspecting their body or initializer.

3. *ThirdPass*: The third Pass pass reports any overridden methods or fields, check main method, report any undefined types and record the types of formals for each class.

4. *FourthPass*: This pass performs type checking.

3.4 LLVM

The *CodeGeneration* class is responsible for translating the abstract syntax tree (AST) of a given program into LLVM IR, an intermediate representation that can be further compiled to executable machine code.

A sequence of visitor methods make up the majority of the class. The LLVM IR code for a particular section of the program is created whenever a node is visited, using data from both that node and previously visited nodes.

4 Limitation of the compiler

We didn't manage to finish the code generation part in time. The LLVM structure is well filled and almost hits the mark but the implementation of the visitors is in progress.

5 Time estimation

We estimate our work time to be a little over 80 hours each. The first part has taken less time comparing to the three others.