Compiler Construction

Project Report

Sam van Kampen (11874716) and R.A.J. Wacanno (11741163)

Bachelor Informatica

April 5th, 2020

1 Introduction

2 Scanning and Parsing

2.1 Preprocessing

As indicated in the language specification, CiviC supports including files with C preprocessor syntax. The simplest way of implementing this is, of course, to simply run the C preprocessor over the input file.

The base framework implemented this by having the C preprocessor output a hidden, preprocessed file in the same directory, which would then be read in. This caused a number of problems. Firstly, it never cleaned it up, so your directory would be littered by preprocessed files, and secondly, it completely failed when compiling something in a different directory (as it simply prepended a dot to the file path). Our more elegant solution was to create a temporary file using `tmpfile()`, which returns a `FILE*`, and using shell redirection to make CPP redirect the output into that file through its file descriptor.

2.2 Abstract Syntax Tree

3 Semantic Analysis

3.1 The Symbol Table

Our symbol table representation went through a number of iterations. We thought of using a hashtable, or even simply a vector, but at some point realised that something we'd really like to have was persistent references - a pointer to a symbol table entry that would be valid for as long as that entry existed. That ruled out vectors and maps because of their resize requirements. So, we went back to the age-old linked list.

3.2 Type checking

4 Internal Representation Transformations

5 Code Generation

6 Extensions

When it comes to extensions, we built the compiler with local/nested functions in mind, and added arrays all the way at the end of development. This late addition turned out to not be ideal, but did illustrate nicely what the issues with not factoring in features during the design phase are.

7 Reflection