University of Amsterdam

Compiler Construction

CiviC Compiler Report

April 1, 2021

Students:
René Kok
13671146

Aram Mutlu
13574116

Lecturer:
Dhr. dr. C.U. Grelck

Course:
Compiler Construction

Course code:
5062COMP6Y

1 Introduction

This report provides information regarding the CiviC compiler we build in the course Compiler Construction by dr. C.U. Grelck. The goal of the CiviC compiler is to create a CiviC compiler that transforms CiviC code into CiviC assembly code that a computer can understand. In this report we will focus on different aspects of our compiler and what we did and how we did it. We will do this by following the order of the milestones, from scanning/parsing to code generation and also use the milestones to further explain the compiler.

2 Lexicographic Analysis

The first step for building a CiviC compiler is to create (or extend in this case) the scanner that reads the CiviC code (stream of characters) that recognizes the symbols and words and outputs a (finite) stream of tokens. By recognizing the words and symbols the Scanner can use pattern matching to find see what the next action will be for the compiler. For the scanner to know which symbols are available in the CiviC language, we need to define these ourselves. We did this by defining all the possible symbols (parenthesis, brackets and binary operators etc.) which are supported by the CiviC language. The CiviC language only supports integers, floats and booleans so there is no need to define other types in the compiler.

In our scanner we added a validator that validates if an integer is too large or too small by comparing it to the max int and min int. If the integer is too high or too low the compiler returns an CTIerror telling the integer is out of range.

```
[0-9]+          { long integer = strtol(yytext, NULL, 10);
                    if (integer > INT_MAX || integer < INT_MIN) {
                        CTIerror("Integer %s out of range", yytext);
                    }
                    else
                    {
                        yylval.cint=atoi(yytext);
                        FILTER( INTVAL);
                    }
                }
```

Listing 1: Integer range check

3 Syntatic Analysis

After the scanner is done and creates a (finite) stream of tokens, the parser comes into play and will start its job. The parser will use the output from the scanner and use it to create and output a abstract syntax tree (AST). The parser uses the stream of tokens to do pattern matching with the pre defined casus in the compiler.

4 Semantic Analysis

5 Optimization

6 Code Generation

7 Conclusion/Discussion/Reflection

The last section of your report should be reflective and could be called anything along the lines of Conclusions, Discussion or Reflection.