Compilers

Project overview

2nd Bachelor Computer Science
2022 - 2023

Kasper Engelen
kasper.engelen@uantwerpen.be

1 Introduction

In this document you will find practical information concerning the project that you will work on throughout the semester. The project will consist of implementing a compiler that compiles code written in (a subset of) the C language. The compiler will output LLVM IR code and MIPS assembly code. The project will be completed over the semester with weekly incremental assignments. During the project you will work in groups of two students. You can also work alone, but this will make the assignment obviously more challenging.

Concretely, you will construct a grammar and lexer specification, which is then turned into Python code using the ANTLR tool. The rest of the project will consist of developing custom code written in Python 3. This custom code will take the parse tree and turn it into an abstract syntax tree (AST). This AST will then be used to generate code in the LLVM IR language and in the MIPS assembly language. At various stages of the project you will also have to develop utilities to provide insight into the internals of the compiler, such as visualising the AST tree.

2 Reporting

At each evaluation, a version of your compiler should be submitted. Upload a zip file on Blackboard which contains the following:

- A minimal report that discusses your progress, discussing the implementation status of every required, and optional (if implemented), feature.

- ANTLR grammar.
- Python sources of the compiler.
- "build" and "test" scripts that can easily be used to demonstrate the functionality of your compiler.
- (if necessary) C example sources for the implemented functionality as well as the expected output for the respective examples.

No solutions will be accepted via e-mail; only timely submissions posted on BlackBoard will be accepted and assessed.

3 Reference Compiler

If you want to compare certain properties (output, performance, etc.) of your compiler to a real-world compiler, the reference is the GNU C Compiler with options `-ansi` and `-pedantic`. The `-ansi` and `-pedantic` flags will tell GCC to **strictly** adhere to the C89 standard. Compiling a C source code file called `test.c` using GCC can be done as follows:

```
gcc -ansi -pedantic test.c
```

You can then study the behaviour of the compiler: warnings, errors, behaviour of the compiled code, etc.

When in doubt over the behavior of a piece of code (syntax error, semantical error, correct code, etc.), GCC 4.6.2 is the reference. Apart from that, you can consult the ISO and IEC standards, although only with regards to the basic requirements.

4 Tools

The framework of your compiler is generated by specialized tools:

- In order to convert your grammar to parsing code, you use ANTLR. ANTLR has got several advantages compared to the more classic Lex/Yacc tools. On the one hand, your grammar specifications are shorter. On the other hand, the generated Python code is relatively readable.
- DO NOT edit generated files. Import and extend classes instead.

Make sure your compiler is platform independent. In other words, take care to avoid absolute file paths in your source code. Moreover, your compilation and test process

should be controlled by the "test" script.

5 Deadlines and Evaluation

Evaluation:

- Make sure your compiler has been thoroughly tested on a number of C files. Describe briefly (in the README file) which input files test which constructions.
- You should be able to demonstrate that you understand the relations between the different rules.
- You should understand the role of a symbol table. Make sure you can indicate which data structure you use and how this relates to the AST structure.
- Show that every rule instantiates an AST class.
- Show which rules fill the symbol table and which rules read from it.

Deadlines: The following deadlines are strict:

- By **Friday 24 February 2023**, you should send an e-mail with the members of your group (usually 2 people, recommended).
- By **Friday 31 March 2023**, you should be able to demonstrate that your compiler is capable of compiling a small subset of C to the intermediary LLVM language. This will be defined in project assignments 1 – 3.
- By **Friday 28 April 2023**, you should be able to demonstrate that your compiler is capable of compiling C to the intermediary LLVM language (project assignments 1 – 6).
- By **Tuesday 30 May 2023**, the final version of your project should be submitted. The semantic analysis should be complete now, and code generation to both LLVM and MIPS should be working. Indicate, in the README file, which optional requirements you chose to implement.

No solutions will be accepted via e-mail; only timely submissions posted on BlackBoard will be accepted and assessed.

6 Exam

The schedule for the final presentations will be available on Blackboard and discussed with all groups. In case you wish to report on the progress of your compiler at an earlier date than indicated, please let us know.