University of Amsterdam

Informatics Institute
Systems and Networking Lab

CiviC Compiler Project

Auxiliary Information

Document Version 0.1

Simon Polstra Clemens Grelck

Introduction

The purpose of this document is to provide you with additional organisational information regarding the CiviC project. All technical information can be found in the language description and in the milestones document.

CiviC Compiler Report

Together with your compiler you must submit a report in the form of a pdf file. This should have the form of a scientific report or paper. As such it should start with a title, the names of the authors, their student IDs and their affiliations. Normally the affiliation would be something like *University of Amsterdam*, but in our case that would be pretty boring, so rather use the programmme you're enrolled in as affiliation, e.g. *Bachelor Informatica* or *Pre-Master Software Engineering*.

Scientific papers typically continue with an abstract, but we skip that because all your abstracts would pretty much look alike and merely re-iterate (or worse: copy) what we had written in one of the various documents.

The first section of your paper is the *Introduction*, where you provide an overview of your work. Drop a few sentences describing the overall project, but keep this short. Do not copy the definition of the CiviC language or the various milestones. Expect your reader to be familiar with both the CiviC language and compilers in general. Focus on the specific aspects of **your** compiler. For example, which extensions you implement (and possibly to what extent) is of great interest to us. Do you support new compiler flags? Have you implemented some optimisations?

The following sections best follow the logical structure of compilers from scanning/parsing to code generation. However, also some of the milestones may make up for good (sub-)sections, e.g. how you implement logical disjunction and conjunction. In essence, explain the sequence of steps **your** compiler takes to compile CiviC programs into CiviC-VM assembly code. Focus in particular on non-obvious aspects: You all use LEX and YACC for scanner and parser generation, respectively. You can probably keep short on this, unless you figured out some particularly cool solutions and ideas. In contrast, optimisations for compact code size or execution speed are non-standard. Elaborate on what you have done in more detail here. Illustrate the (hopefully positive) effect of your code transformations on code size and/or execution speed by means of experiments. Use the corresponding VM flags to obtain the necessary data.

Writing a compiler is like fighting the dragon: did you encounter particular pitfalls, walls against progress, etc? Did you at a certain point fail to overcome them? Tell us. Note that blaming yourself never negatively affects your grade. We anyways find out via black box testing if you failed to implement certain features. In this sense your compiler and your report are graded independently. So, you can be free in telling us what does work and what does not work.

The last section of your report should be reflective and could be called anything along the lines of *Conclusions*, *Discussion* or *Reflection*. You do not need to write anything about related work, and you do not need a bibliography.

The concrete formatting is entirely up to you. All the guidelines here are literally meant to *guide* you. They are not meant to limit your creativity. Keep in mind the following rules of thumb:

- Keep short about anything box standard: don't copy/paste CiviC definitions or milestones.
- Invest your ink (and energy) into the unique aspects of your compiler and any smart decisions you took.
- Tell us about your design decisions.
- You're proud of something? Let us know!

Illustrations are always a plus. These can range from nice graphics to code (transformation) examples. Show us what you've learned: why not use a compilation scheme to formalise your compiler work at an abstract level.

There is no formal page limit, but **6–8 pages** should normally suffice.

Do not use fillers to get to a certain page number. And, do not attach the complete scource code of your compiler (or the like) as appendix.

Submission Guidelines

The report must be submitted as a single pdf file via the corresponding Canvas assignment. Construct the file name from the family names of the authors combined by an underscore.

The compiler should be submitted as 'clean' tar file that untars into its own subdirectory, again via Canvas.

You can create a tar in the correct format with the commands:

```
~/coco/framework$ make clean
~/coco/framework$ cd ..
~/coco$ tar czf framework.tgz framework
```

The framework should not contain any git repository files, so archive it or just remove the .git directory.

Submission is via the corresponding Canvas assignment.

Compiler Evaluation and Grading:

The evaluation of your compiler is predominantly based on black-box testing. For this purpose we have two test suites: the public test suite is made available via Canvas; the private test suite is secretly used for additional evaluation.

As a rule of thumb we will apply the following grading scheme:

decent compiler, 0 extensions	7.0 base grade
decent compiler, 1 extension	8.0 base grade
decent compiler, 2 extensions	9.0 base grade
decent compiler, 3 extensions	10.0 base grade
very good code size	0.5 bonus
very good cycle count	0.5 bonus
very decent code size	0.25 bonus
very decent cycle count	0.25 bonus
failing test cases	up to 2.0 malus

Although compiler evaluation is mainly based on black box testing, we do look at your source code as well. Unusually good or bad source code also affects your final grade with a bonus or malus along the lines of the above table.