Department of Mathematics and Computer Science
University of Southern Denmark, Odense

February 14, 2019
KSL

Exam Project in Compiler Construction, part 3

Kim Skak Larsen
Spring 2019

Introduction

In this note, we describe one part of the exam project that must be solved in connection with the compiler project, Spring 2019. It is important to read through the entire project description before starting the work on the project; also the sections on requirements and how to turn in your solution.

Deadline

Friday, March 15, 2019, at 12:00 (noon)

Correct KITTY Programs

Among other things, you must turn in a program which must be written in the programming language C. It must be the c11 ANSI standard as specified by the options below. This excludes C++, in particular. Your programs should be compiled using

```
gcc -std=c11 -Wall -Wextra -pedantic
```

The primary new tasks of this part of the project are to construct a weeder and a type checker. These phases must be combined with the symbol table and the scanner/parser from the previous parts of the project to form a complete front-end of a KITTY compiler. To test the front-end, a new pretty printer must be constructed which prints a representation of the abstract syntax tree where all expressions (and subexpressions) are annotated with their types. This must be the output for correct KITTY programs. For incorrect KITTY programs, the compiler must print an error message, informing the programmer of at least one error in the program along with its line number and a reasonable explanation of what the error is.

Weeder

There should be a separate weeder phase between the parsing and the type checking phases. As a minimum, the following must be handled:

- For function definitions, it must be verified that names after the keyswords **func** and **end** are identical.
- It must be verified that all function calls will result in the execution of a **return** statement. It is a part of the assignment to detail this requirement and describe the implemented rules in the report.

Type checking

This part can structurally be organized through the following three (abstract) traversals of the abstract syntax tree. You can consider whether or not some traversal could conveniently be merged with one of the other traversals.

1. Collection of variable, type, and function declarations.
2. Calculation of the types of all expressions and subexpressions. One possibility is to allocate space in the nodes of the abstract syntax tree for saving this information.
3. Verification of correct usage of all variables, types, and functions.

Prettyprinter

A prettyprinter is here a program which prints the abstract syntax tree with sufficient indentation and/or parantheses so that the structure of the tree can be verified.

Additionally, the type of all expressions and subexpressions must be indicated in the print-out. Find a way to do this without making the printed programs completely unreadable.

Testing

A sufficient collection of programs must be tested such that it is verified, via the prettyprinter, that all type information is computed correctly. Additionally, any error message should be provoked by some test program.

Turning in

Electronically, you must turn in

- All relevant files from the previous parts of the project.
- C-files for the weeder (presumably `weed.c` with header file).
- C-files for the type checker (presumably `typecheck.c` with header file).
- a C-program which, using the files above, implements a type annotating pretty-printer for KITTY programs.
- a makefile, connecting all of the above.

Additionally, you must hand in a report with brief descriptions of the most important choices made in the process of creating the weeder and type checker. You must include a sufficient and documented testing. See also the standard requirements.

General Requirements and Rules

Here we list general requirement, procedures for turning in, and exam rules.

Exam Rules

This is an exam project. Cooperation beyond what is explicitly permitted will be considered cheating and will be treated as such. You have a duty to keep your notes private and protect your files against reading and copying by others. Both parties involved in a possible plagiarism can be held responsible.

There will be given what we judge to be more than sufficient time for solving the project. Still, we strongly encourage you to plan your work such that you will finish some days before the deadline.

Solutions that are turned in after the deadline will not be accepted. Downtime on the system or the printers will not automatically result in an extension; not even if it is the last hours before the deadline. Neither will own or children's illness without a statement from your physician, etc.

The solution

The solution consists of a program, test material, and a report. Thus, we use the term "report" to mean your description of the solution to the project without the program listing and listing of test examples and results (other than what may have been merged into the report as examples, etc.).

All specific requirements posed in the project description must of course be fulfilled.

The Report

The report should in the best possible manner account for the entire solution, i.e., it must contain a description of the most important and relevant decisions that have been made in the process of developing the solution and reasons must be given where this is appropriate.

You must also explain how the program has been tested. Test examples or references to test examples and test runs can and should be included to the extent that this is meaningful.

Possible omissions, known errors, etc. should be described in the report. It is often a good idea to do this in a separate section instead of mixing it in with the rest of the report.

Programs

Files and directories should be named and organized logically. Programs must be well-structured with appropriately chosen names and indentation and tested sufficiently. The numbers of characters (including blanks and 4 times the number of tabs) on a program line is limited to 79. This is important for various tools used for inspecting, evaluating, and viewing your programs, and it is important for the print-out of parts of your own program that you will see at the exam.

Programs will often be tested automatically. This makes it extremely important to respect all interface-like demands, e.g., input/output formats.

Programs that are turned in must compile and run on IMADA's machines. In particular, they should be written in the programming language C. It must be the c11 ANSI standard as specified by the options below. This excludes C++, in particular. Your programs should be compiled using

```
gcc -std=c11 -Wall -Wextra -pedantic
```

In particular, no architecture-dependent option should be added, such as, for instance, `-m32` or `-m64`.

You are very welcome to develop your programs at home, but it is your responsibility. This includes technical problems at home, lack of access to relevant software, moving data to IMADA via e-mail, USB keys, etc. and converting to the correct format, e.g., between Windows, Mac, and Linux.

Execution

This section on execution does not apply to part 1, but starts applying gradually through the project parts until it applies fully at the end. It is included in every project description, so you are not surprised at the end.

In the following, we list execution requirements regarding your compiler as well as the code your compiler produces. In most cases, this is just to conform to default standards or to choose one among alternatives:

- Your compiler (executable) must be called `compiler`.
- Behavior of your compiler:
  - Your compiler must read from `stdin`.
  - In the final part, only correct assembler code may be written to `stdout`.
  - If the compilation succeeds, the compiler must return zero.
  - If an error occurs during compilation, then
    - *nothing* should be written to `stdout`,
    - an error message should be written to `stderr`, and
    - a value different from zero must be returned.
  - It is recommended that the beginning of each phase of the compilation is announced on `stderr`.
- Behavior of the code your compiler produces:
  - The code produced must be X86 Assembly/GAS Syntax, compilable with

    ```
    gcc -no-pie -m64
    ```

  - Only **write** statements may write to `stdout` and it should write its integer or boolean argument followed by a newline.
  - If no error occurs, the code must return zero.
  - If an error occurs (that you catch), the code must return a value different from zero. If you write an error message, it must go to `stderr`.

Turning In

You must turn in on paper *and* electronically. The details are given below. All material that is turned in both on paper and electronically must be identical.

On Paper

You must turn in your

- report.

The front page of your report should contain your group number and the full names and student logins of all group members.

Procedure for turning in on paper: The material on paper should be turned in by placing it in the lecturer's letterbox. You are also welcome to give it directly to the lecturer.

Electronically

Electronically, you must turn in

- the report as `report.pdf`,
- all relevant program and test files,
- a makefile, connecting the program files,
- the compiler as `compiler`, which should be an executable file.

Procedure for turning in electronically: The procedure for turning in electronically can be found via the project home page or this direct link:

`https://imada.sdu.dk/~kslarsen/CC/elafl.php`

However, it might be good to know already now that you should avoid Danish (and other non-ascii) characters (such as æ, ø, and å) in your directory and file names (Blackboard does not handle this well). To be safe, also avoid whitespace and special characters not normally occurring in file names.

You may upload your files individually or collect your files into one (archive) file (recommended) before uploading. If you choose to do the latter, you must use either `tar` (optionally also `gzip`'ed) or `zip` for this.