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

February 13, 2013
KSL

Exam Project in Compiler Construction, part 3

Kim Skak Larsen
Spring 2013

Introduction

In this note, we describe one part of the exam project that must be solved in connection with the project ”A compiler for an imperative programming language”, Spring 2013. 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, 2013, 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 c99 ANSI standard as specified by the options below. This excludes C++, in particular. Your programs should be compiled using

```
gcc -std=c99 -Wall -Wextra -pedantic -m32
```

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-typprinter for KITTY programs.
- a makefile, connecting all of the above.

Additionally, you must hand in a report with program listings of all of the above, along 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.

Requirements

All material should be turned in on paper (referred to as *the report*) and electronically (a few exceptions are mentioned below). In addition, since this is an exam project, there are a number of important rules that will be detailed below.

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. Note that this includes test programs. The design of appropriate test programs is an important part of the development, verification, and documentation of your code. Thus, this is a task you must carry out yourselves, and acquiring and using test programs from another source is also a form of plagiarism.

There will be given what we judge to be more than sufficient time for each assignment and you are strongly encouraged to plan your work such that you will finish some days before the deadline.

Assignments 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.

Solutions

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. 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.

You must include the page at the end of this document as the front page of your report or attached in some way such that it is easily located. The report must be dated and signed by the members of the group.

For programs turned in as part of your solution, you must take care of the following:

The report must contain (possibly as an appendix) a printing of the entire program. This printing must be identical to the program that is turned in electronically. All the pages of your program print-out must contain your group number. One way of obtaining this is to use

```
a2ps -Pd3 --line-numbers=1 --tabsize=4 -g
  --header="Printed by group NN" file.c
```

where `NN` is your group number, or you can include the listing directly in your report using some LaTeX package.

The report must contain a description of the most important and relevant decisions that have been made in the process of answering the assignment and reasons must be given where this is appropriate.

You must also explain how the program has been tested. Test examples and test runs can and should be included to the extent that this is meaningful (really large test files can just be turned in electronically).

Programs

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. 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 and Linux.

Execution

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 `kitty`.
- Behavior of your compiler:
  - Your compiler must read from `stdin`.
  - In the final part, only correct assembler code may be written `stdout`.
  - If the compilation succeeds, the compiler must return zero.
  - If an error occurs during compilation, a value different from zero must be returned.
- Behavior of the code your compiler produces:
  - Only **print** statements may write to `stdout`.
  - 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.

Turning In

The report should be turned in at IMADA's secretaries' office. The office may be closed for very short periods of time. If, for some unexpected reason, the office must be closed for longer periods of time close to the deadline, an announcement will be made outside the office, giving instructions as to where you turn in your report.

For the first parts of the projects, you only need to turn in one copy of the report. For the final part, you must turn in two copies. For all parts, you must turn in all the material electronically.

Programs, test files, etc. should be turned in electronically. Your report should also be turned in electronically as a pdf file. As opposed to the paper version of your report, this version does not necessarily have to include programs and test files, since they are turned in separately. Also, signatures and the front page from the end of this document are not required in the pdf file.

The procedure for turning in electronically can be found via the project home page:

`http://www.imada.sdu.dk/~kslarsen/CC/Projekt/`

Avoid Danish (and other non-ascii) letters (such as æ, ø, and å) in your directory and file names (Blackboard does not handle this well).

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

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

CC, Spring 2013
Exam Project, part 3

Group	

Date	

Name	
Birthday	
Logins	
Signature	

Name	
Birthday	
Logins	
Signature	

Name	
Birthday	
Logins	
Signature	

This report contains a total of . . . . . pages.

Please write *very* clearly. Under Logins, give your IMADA followed by your student (`student.sdu.dk`) login.