CS325 2017/18: Assessed Coursework

The CS325 module covers both theoretical and practical aspects of designing and building a compiler. In this programming project, students will learn how to examine a language specification and build a parser for programs in that language.

1. Goal of the project

You will develop a recursive descent parser for a subset of Cool (the Classroom Object-Oriented Language). Cool has been developed to support courses on compiler construction and abstracts many important concepts in typical programming languages. It has been used in many Universities (Stanford, Virginia, etc) but mostly with the aim of building a whole compiler with the use of parsing tools such as Bison. In this assignment, time limitations preclude building a whole compiler or interpreter; instead, you are asked to build a realistic top-down parser which identifies the names of classes and methods being declared.

By realistic, we mean a parser that can help diagnose many lexical and syntactic errors in source code. The reference interpreter for Cool has a very simple notion of error reporting, and stops at the first error. Your goal is to develop a front-end that both gives informative error messages and recovers enough after an error to provide information on more errors downstream.

First, of course, you need to build a parser. Rather than use a parser-building tool such as Yacc, Bison or JavaCC, you will build a parser from scratch, using a recursive-descent technique. In order to build a working parser methodically, you may need to adapt the grammar.

2. Programming Methodology

Your program will be written in Python. It will take as input a file consisting of Cool source code. If there are lexical and syntactical errors in the input file, your program will print on the first line the message “Errors found” followed on further lines by a list of errors with helpful diagnostic information. If there are no errors in the Cool file, your program will print on the first line the message “No errors found” followed on further lines where each line has the names of a Cool classes being defined and for each class the names of all the methods declared in it.

But first, you will need to build a simple parser. You may not use any existing parser construction library, instead you will craft the parser directly using the approach known as recursive descent parsing. Within this approach, there are two strategies you can use: (1) you can build a backtracking parser or (2) a predictive parser. Both have advantages and disadvantages, as discussed in lectures, and your task will be to apply modifications to the grammar in order to make programs parse-able.

The only external modules you are allowed to use are those identified in the Python (3.*) tutorial as part of the Standard Library, and the shlex module, which can facilitate the lexing phase. In particular, have a look at the `re` module in the Standard Library. You will not need to use Python classes in your program.

3. Source Language

The Cool language has been originally described in the manual available on the module webpage, and also at its original location:

http://theory.stanford.edu/~aiken/software/cool/cool-manual.pdf

This original document, however, has been superseded by a more up-to-date version available as online html at:

http://www.cs.virginia.edu/~cs415/cool.html

https://web.archive.org/web/20170121083058/http://www.cs.virginia.edu:80/~cs415/cool.html

You should read the whole manual and familiarize yourself with Cool, however, in terms of the coursework, you will be particularly concerned with Section 10. *Lexical Structure* and Section 11. *Cool Syntax*. One important aspect which has changed from the version available at Stanford to the version available in Virginia is Section 2. *Getting Started.* The original document mentions a `coolc` compiler, but this has been superseded by the `cool` interpreter, which can be downloaded from Virginia and which is available locally at `/local/java/cool/`. It can be run on the DCS Linux machines as `/local/java/cool/cool`.

Please note that you are not expected to catch type errors. This is often done by the front-end but outside the scope of this assignment.

Also, you can assume there will be no comments in the Cool source file. There are subtle difficulties in dealing with errors in comments. Thus, none of the marking examples will have any comments in them.

4. Deliverables

You will need to submit a zip file containing one or more Python source files. The main Python file must be called `coolparser.py` and it must provide a function called `parse(filename)` which will read and parse the file called `filename`.

At the bottom of file `coolparser.py`, please include the following code:

```
if __name__ == "__main__":
  import sys
  parse(sys.argv[1])
```

This will allow me to run your code by simply typing:

```
python coolparser.py <testfile>
```

The zip file should also include a two-page report in pdf form called `report.pdf`. Your report will complement your code, and suggested topics to be covered include:

- What are the main design decisions behind your implementation and what could you have done differently?
- What were the difficulties you faced? How did you overcome them?
- What help did you get? Which other sources did you consult?
- How did you test your solution?
- What are the known limitations of your solution?

Make sure that you use Python 3. Submissions with the parser written in Python 2 will not be accepted.

The deadline for submitting the zip file via Tabula is 06th November 2017 at noon. The time you submit to Tabula will be taken as the official time of submission.

Marking will be done anonymously, so please do not put your name in your program file or your report, instead include your University card number.

Remember that once you submit coursework you will not be allowed to de-register from this module.

5. Assessment Criteria

This coursework is worth 30% of the marks for CS325. It will be marked out of 100, and the marks will be allocated to the following aspects:

File contains code for recursive-descent parser and runs	10
Programming style, clarity of comments in code	10
Being able to differentiate programs with and without errors	10
Identifying and reporting classes and their methods	20
Quantity of errors detected	20
Quality of error reporting	10
Written report: reflective quality and writing skill	20

6. Plagiarism

By now you will know the Department policy on allowable collaboration and reliance on outside sources. You may well find bits of code that are relevant to the coursework, if you do make sure you acknowledge it and be aware that your submission will be marked taking this source into consideration. You may discuss the basic structure of your solution with colleagues, but the final code that is submitted must be your own.

Good luck with completing the coursework and contact me if you have any questions.