CPSC/ECE Fall 2018
Software Design Exercise #1

Canvas submission only
Assigned 9/5/2018
Due 10/3/2018 11:59PM

1 Overview

The overall objective of SDE #1 is to implement a scanner/parser for the lambda calculus using **both** `flex` and `bison`. This also necessitates the use of `c`. All work is to be done on a linux platform. Significant in-class discussion will accompany this SDE.

2 The Effort

We base the scanner parser on the grammar productions used previously, i.e.,

<expression> ::=
<variable>
| **<constant>**
| **(<expression> <expression>)**
| **(lambda <variable> . <expression>)**

Notice, due to ASCII/keyboard limitations, we're using the keyword `lambda` instead of the symbol λ.

The effort includes:

1. Building a `flex`-based scanner for the lexical part of the lambda calculus;
2. Building a `bison`-based parser for the lambda calculus productions;

3. Integrating the 2 in `c` source and building an executable version;
4. Testing the result; and
5. Adhering to the specified output format. (Don't improvise.)

3 Resources

It would be foolish to attempt this SDE without carefully exploring:

1. The text, especially the many examples in Chapter 6;
2. The `flex` and `bison` manuals; and
3. In-class discussions.

4 Constraints

The constraints for the lambda calculus scanner/parser are as follows:

1. The productions are those indicated in Section 2. Do not change them.
2. Variables are arbitrary-length strings of (all) lower-case alphabetic characters
3. Constants are numbers or (built-in) function names
4. A number is any length (unsigned) string comprised of the digits 0-9
5. (Built-in) function names are restricted to `fun<number>` where number is defined above.
6. The parser either succeeds or fails *with the output shown in the examples.*
7. The parser executable is named `lambda1`.
8. The token `lambda` may not be used as a variable or a constant (this includes as a function name).

9. In the case of a successful parse, the parser reports the form of the overall expression. See the examples.

10. Note: The scanner/parser does just what the name implies; **it is not a reduction machine**. In other words, input is scanned and parsed but reduction of a valid expression is neither necessary nor desired.

If you have questions about these constraints, be sure to ask. The examples show many of these required features.

5 Desired Interface/Output Format and Examples

The desired interface is shown below. This is the only acceptable output format. Each candidate string to be scanned and parsed is stored in a text file (*.lc) and input to the parser as shown.

```
/* first we build the executable */
$./build-lambda1.sh

/* aside: Anybody see any errors or warnings in the above build? */

$cat t0.lc
fun100

$./lambda1 <t0.lc

Parse for syntactically correct lambda-calculus expression was successful:

    The overall expression is a constant

$cat t1.lc
((lambda x.(fun1 x)) 100)

$./lambda1 <t1.lc

Parse for syntactically correct lambda-calculus expression was successful:

    The overall expression is a combination

$cat t1bad.lc
((lambda x.(fun1 x)) .x)
```

```
$./lambda1 <t1bad.lc

Sorry, Charlie: Not everybody can be a lambda expression!

$cat t2.lc
(((lambda x.(lambda y.(x y))) fun2) 3)

$./lambda1 <t2.lc

Parse for syntactically correct lambda-calculus expression was successful:

    The overall expression is a combination

$cat t2bad.lc
(((lambda x.(lambda y.(x y))) f2) 3)

$./lambda1 <t2bad.lc

Sorry, Charlie: Not everybody can be a lambda expression!

$cat t3.lc
(lambda x . (fun9 x))

$./lambda1 <t3.lc

Parse for syntactically correct lambda-calculus expression was successful:

    The overall expression is an abstraction

$cat t3bad.lc
(lambda (x . (fun9 x)))

$./lambda1 <t3bad.lc

Sorry, Charlie: Not everybody can be a lambda expression!

$cat t4.lc
(lambda x . (lambda y . (lambda z . (fun0 z))))

$./lambda1 <t4.lc

Parse for syntactically correct lambda-calculus expression was successful:

    The overall expression is an abstraction
```

```
$cat t4bad.lc
(lambda x . (lambda y . (fun0 z))))

$./lambda1 <t4bad.lc

Sorry, Charlie: Not everybody can be a lambda expression!

$cat t5.lc
((fun1 fun2) (fun3 fun4))

$./lambda1 <t5.lc

Parse for syntactically correct lambda-calculus expression was successful:

    The overall expression is a combination
```

6 Building the Scanner/Parser

6.1 Build Script

Below is the shell script (file: `build-lambda1.sh`) we will use to build your project. You should use it too.

```
#!/bin/bash
## to build 3520 sde1 project
bison -d lambda1.y
flex lambda1.in
gcc lambda1.c -Wall -lfl -o lambda1
```

There should not be any errors or warnings reported in the building process.

6.2 Required Use of Our Error Handling (`yyerror`) Function

For uniformity, **you are to use our version of the error handling function**, `yyerror()`, as shown below (and supplied by us in file `yyerror1.c`):

```
/* error handling function specifically for 3520 sde1 */
/* must be used */
/* called by yyparse on error */
```

```
int yyerror (s)
     char *s;
{
  return(-1);
}
```

7 Remarks

Please note:

1. This Assignment assesses *your* effort (not mine). I will not debug or design your code, nor will I install software for you. You (and only you) need to do these things.

2. It is never too early to get started on this effort. This will, unfortunately, become evident to some on or about 10/3/2018.

8 Additional Notes and Constraints

8.1 Notes

1. Only solutions employing `flex`, `bison` and `c` are acceptable.

2. Use this document as a checklist to be sure you have responded to all parts of the SDE, and have included the required files

3. Submit your solution by the deadline. After the deadline, a grade of 0 is assigned. See Section 9.

4. **This is an individual effort.**

8.2 Format of the Electronic Submission

The final **zipped** archive is to be named **<yourname>-sde1.zip**, where <yourname> is your (CU) assigned user name. You must upload this to Canvas prior to the deadline.

The minimal contents of this archive are as follows:

1. A `readme.txt` file listing the contents of the archive and a brief description of each file. Include 'the pledge' here. Here's the pledge:

Pledge:
On my honor I have neither given nor received aid on this exam.

This means, among other things, that the code you submit is **your** code.

2. Section 6 shows how we will attempt to build your executable. *This is the only way we will attempt to build your executable* from `flex`, `bison` and `c` sources for your implementation. Name the source files `lambda1.*`, where * is `in`, `y` and `c`, respectively.

   We will provide function `yyerror()` in file `yyerror1.c`, as shown in Section 6. The script will name the executable `lambda1`. We will attempt to build, run and test the executable. As indicated in the course syllabus, most of an SDE grade is based upon a **syntactically and semantically correct working solution which meets the given specifications and is submitted before the deadline**.

3. Include a log file (named `lambda1.log`) of sample operation. Show 10 example parses with syntactically correct lambda expressions and another 10 with incorrect syntax.

9 Final Remark: Deadlines Matter

Since multiple submissions to Canvas are allowed[1], if you have not completed all predicates, you should submit a freestanding archive of your current success before the deadline. This will allow the possibility of partial credit. **Do not attach any late submissions to email and send to either me or the graders.**

[1]But we will only download and grade the latest (or most recent).