

School of Computer Science and Software Engineering

CS3322 Programming Languages and Implementation

Assignment 3

Due 5pm Friday 14th of October

(Deadline extension 1 week due to delay in the lecture schedule.)

The purpose of this assignment is to practice the implementation of scanners and parsers.

We will implement a tiny toy calculator language that calculates expressions for angles.

The core components of this calculator are a scanner (generated with ML-Lex) and a parser (implemented manually in ML). The actual evaluation can either take place directly in the parser (using an attribute grammar) or it can be implemented separately by traversal of the parse tree.

An *angular constant* takes the form “220deg12min10sec”. There are 1-3 digits preceding the *deg* part, 1-2 digits preceding the *min* part and 1-2 digits preceding the *sec* part. All of these parts are optional and it is permissible to have more than 360 degrees, more than 60 minutes or more than 60 seconds in a constant. For instance, “999deg80sec” is a valid angular constant.

Angular expressions consist of angular constants, ordinary numerical constants, such as 32 or 2.5, the infix operators PLUS and MINUS (spelt as words) which operate on angles and numerical constants, the infix operators DIV and TIMES, which operate on an angle and a numerical value, and brackets. For example,

2 TIMES (66deg12sec MINUS 1.5 TIMES 12min14sec)

is a valid expression. PLUS, MINUS, DIV and TIMES have their usual precedence. There is no unary minus. Note that it does not make sense to multiply or divide by an angle or to subtract a numerical constant from an angle or vice versa. However, your implementation does not need to check this. You only need to check for syntactic validity, whereas this is a semantic property (type correctness).

The evaluator should read a sequence of angular expressions from a file, evaluate them in order and print the value of any expressions prefixed with PRINT to `TextIO.stdOut`.

Expressions can also include variables. Identifiers for these can be arbitrary words of the form $[aA\dots zZ]^*$ but not the reserved words for the operators or for the units (“*deg*”, “*min*”, “*sec*”). Variable names are case-sensitive. All variables must be assigned values before they are used in an expression. An assignment statement takes the form “variable-name := expression ;”, e.g. `Test := 60deg ;`

For these variables, you will need to keep a simple symbol table in which the variable values are stored and where subsequent expression evaluations can look up the value.

Each variable can only be assigned once. A variable value is only valid in a single input file, i.e. every call to the evaluation function starts with an empty symbol table. The condition that a variable is assigned exactly once is a semantic condition and does not need to be checked by your program.

Example: consider the following input file:

```
Test := 320deg12min ;
PRINT Test DIV 2 MINUS 3 TIMES 20deg32min ;
Rest := 360deg MINUS Test;
PRINT Rest;
```

The evaluator should print the two lines “98deg30min00sec” and “39deg48min”. to `stdOut`. Note that the

output should be normalized in the sense that all degree values are in the range 0-360, minutes in the range 0-60 and seconds in the range 0-60.

1. Define a suitable datatype for the tokens to be returned by the lexer. Using ML-Lex implement a suitable scanner for the expression files described above. You can adapt the example lexer that was discussed in the tutorials (Lecture notes Implementation 2). This code has also been made available on the web site. Your lexer must signal lexical errors. **[3 mark]**
2. Write down an LL(1) grammar for this expression language. **[1 mark]**
3. In SML/NJ implement a recursive descent parser for the expression files described above. You can adapt the recursive descent expression parser discussed in the tutorials (Lecture notes Implementation 3). This code has also been made available on the web site. Your parser must signal syntactic errors. **[4 marks]**
4. Extend your code for the recursive descent parser with semantic actions and attributes so that it evaluates the expressions in the file. The toplevel-function should be

```
val evaluate = fn : string -> unit
```

which takes a filename as an argument, evaluates this file and prints the values of all expressions that have been prefixed with the PRINT command during the evaluation. **[2 marks]**

Submission Instructions

The above exercises contribute 10% to your total CSE3322 mark.

The assignment is due **Due 5pm Friday 14th of October**.

(Deadline extension 1 week due to delay in the lecture schedule.)

If possible you should get your assignment marked in one of the optional tutorials. This will give you a chance to comment on problems that your code may have.

If you cannot get your assignment marked in the tutorial you must submit electronically. You should use `/cs/cc/bin/submit` to submit your assignment. To do so create a directory that contains all the files which you have to submit (see below) and submit this directory as a whole. Login with ssh to `ra-clay.cc.monash.edu.au` or `sng.cc.monash.edu.au` and then type `/cs/cc/bin/submit`. Submit will ask you for: the course code which is `cse3322`, the assessment code which is `ass3`, your student I.D., and the name of the directory to be submitted. This must be `Ass3`.

You must submit the following items in the directory called `Ass3`:

1. The lexer specification in a file `angle-lex.lex`,
2. The output of ML-LEX for this specification in a file `angle-lex.lex.sml`,
3. The program for the parser in a file `angle-parse.sml`. **Note that when loading this file, the lexer must automatically be loaded.**
4. A Postscript or PDF file called `grammar.ps` or `grammar.pdf` containing the grammar for Question 2. If you are unable to generate such an electronic file you can submit the grammar in handwritten form to the general office.

Furthermore, each program file must include the following declaration at the top of the file (failure to include this will result in the submission receiving 0 marks):

```
Monash University,  
School of Computer Science and Software Engineering  
Student Declaration for CSE3322 Submission 2004 Sem 2  
I *your name*, ID: *your ID*, Email: *your Monash Email*  
declare that this submission is  
my own work and has not been copied from any other source  
without attribution. I acknowledge that severe penalties exist for  
any copying of code without attribution, including a mark of 0  
for his assessment.
```

If you have solved Part 4 of the assignment you can just submit the complete code in the file `angle-parse.sml`. You do not need to submit a separate solution to Part 3 in this case.

Your programs will be marked on correctness, style, efficiency, clarity and documentation.

Assignments handed in after the due date will attract a late penalty of 5% per day unless special consideration applies or there has been prior agreement in writing from the lecturer. No submission will be accepted later than one week after the due date.