PROGRAMMING LANGUAGE CONCEPTS (COMP 2212)
SEMESTER TWO, 2024

exercise sheet four: using alex tool

The aim of this exercise class is to introduce the Alex tool. By the end of the class you should have a basic understanding of Alex file and how to generate a lexer.

We will be returning to these same examples next week for Parsing in the Happy tool.

Work on the exercises independently or in groups and if you run in to difficulty then use the Friday lab session to ask any questions you have about the them.

Before you get started, download the file Tokens.x. Read this and see if you can work out the lexemes used and how they correspond to tokens.

Task One

Modify the `Tokens.x` Alex file to allow for a new piece of syntax representing an exponentiation operator. You can choose whatever symbol you like to represent this.

Make sure that you compile your modified code using Alex to generate file `Tokens.hs`.

Write a Main module that

- uses a command line arguments using `getArgs` to read in a text file to obtain a `String`
- feeds this `String` to the function `alexScanTokens` generated by Alex to try obtain a `[Token]` value
- prints out a representation of this token list or an error message where there is an unrecognised symbol.

Make sure that you test your code on different input files.

Task Two

Modify your files from Task One to use the "posn" wrapper in the `Tokens.x` file instead of the "basic" wrapper. This will involve changing the token actions to have type `AlexPosn -> String -> Token`. See Wrappers for more detail.

Write a function `tokenPosn` in your Alex file that extracts the source code position (line:column) from a given token as a string e.g. "5:43".

Task Three

Design the syntax for a domain specific language called the **Maze Direction Language** (MDL). Programs written in MDL should describe a sequence of instructions for moving a MazeBot around in a maze. The maze will contain obstacles which block the MazeBot's progress.

Commands must allow the programmer to specify a move forward for a given number of steps or to rotate left/right. The language should allow sequences of such moves to be created. There should also be an operator that allows the MazeBot to check whether there is an obstacle located within N steps in the direction the MazeBot is facing. The value N is given to the check operator and must be in the range 1 to 9. Finally, there should be a conditional operator that allows if-then-else branching based on the results of obstacle checking.

An example program of the language might specify the following instructions:

- Go forward for 10 steps
- Rotate Left
- If there is an obstacle within the 3 steps ahead then
    - Rotate Left, Go Forward 1 step, Rotate Right
    - Otherwise Go Forward 3 spaces then Rotate Left

Write a BNF grammar for your concrete syntax of MDL, identify the lexemes in your grammar and then write an Alex file to generate a lexer for your language. You will need to define a data type of Tokens for MDL as a target for lexing.

Write a Main module that reads MDL programs and lexes them. You do not need to implement this language at this point. Just take a source program and produce a list of Tokens.