PROGRAMMING LANGUAGE CONCEPTS (COMP 2212)
SEMESTER TWO, 2018

Home

exercise sheet two: alex and happy

The aim of this exercise class is to introduce the Alex and Happy tools. By the end of the class you should have a basic understanding of Alex files, Happy files and how to generate a lexer and parser from them.

The demonstrators in the lab will help answer any questions you have about the tutorials you are reading or if you have difficulty understanding the solutions to the exercises.

Before you get started, download the files Tokens.x and Grammar.y. Read these and see if you can work out the grammar represented.

Task One

Modify the `Tokens.x` Alex file and the `Grammar.y` Happy file to allow for a new piece of syntax representing an exponentiation operator.

Make sure that you compile your modified code using Alex and Happy to generate files `Tokens.hs` and `Grammar.hs`.

Write a Main module that

- reads in a text file to obtain a `String`
- uses the function `alexScanTokens` generated by Alex to try obtain a `[Token]` value
- uses the function `parseCalc` generated by Happy to try obtain an `Exp` value.

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.

Modify your Happy file `Grammar.y` to work with this modified Token type and to report the location of any parse error in the input stream.

Task Three

Design 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 langua ge 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, then write Alex and Happy files to generate a lexer and parser for the language. You will need to define a data type of Abstract Syntax

Trees for MDL as a target for parsing. You do not need to write any sort of interpreter for MDL though.

Write a Main module that reads MDL programs and parses them.

Page maintained by Pawel Sobocinski, Julian Rathke.