P2

Reid Chen, Gaoyi Hu

October 6, 2021

Contents

1 JLex

The crux of this project is to come up with a way to distinguish the 4 scenarios of string literals. When set a state called `OKSTR` when JLex sees a `"` to indicate that the JLex is currently looking at a legal string literal. We let JLex to match one character at a time. If JLex sees another `"`, then this string is terminated. If JLex sees a \, we enter a state called `BACKSLASH`. Now, there are two cases. If JLex sees one of $\{n, t, ?, \backslash, ”\}$, then it is a valid escape, so we go back to state `OKSTR`. On the other hand, if the character after \ is not one of the valid escape character, the JLex enters `BAD_ESCAPED` state, indicating that this string literal now is bad. JLex check if a string literal in `BAD_ESCAPED` is terminated or not using a similar logic as `OKSTR`.

2 Testing

We have created 3 directories to store the testing files. `inputs` stores all the inputs, with extension `.in`. `outputs` stores the standard outputs, with extension `.out`. `expects` stores the expected standard outputs, with extension `.ex`. We keep `allTokens.in` and `eof.txt` in the root of this project, but we did not use them. Test files are located in these 3 directories described above.

The correctness of standard errors, i.e. the message produced by the `ErrMsg` are checked in the main of P2, instead of using `diff` to compare the

expected standard outputs and the actual standard outputs. To standard error (error messages), we redirect 'System.Err' to a customized stream. And compare the expected String with the String of the stream.

The comparision of standard outputs is done using `diff TEST_CASE.out TEST_CASE.ex` in the Makefile.