Compiler Project - Part 1 Lexical Analyzer

T-603-Compilers, 2020

In this part of the project you will write a lexical analyser for a (slightly simplified) version of Turbo Pascal (TP), a once popular programming language. You will write two (almost) functionally equivalent lexical analysers, one hand-written in C++ and the other by using Flex.

You are provided with files *common.h*, *language.h/language.cpp*, and *lexer.h*, which implement helper routines and define the lexical class interfaces that you should use. **You are not allowed to change any of those files**. The abstract class *Lexer* provides an interface (and some minimal default implementation) for sub-classed lexical analysers. The file *language.h* provides information about the types of tokens to return and the information to store in each token. Note, that in addition to returning the defined tokens, the lexical analysers also need to remove white-spaces and comments from the input. At the end of the file a special *end-of-input* token should be returned, and if an unrecognized lexeme is encountered the lexical analyser should return a token of type *unknown*. Furthermore, it should throw an exception (defined in *lexer.h*) in the case of encountering a non-terminating string or comment.

You are also provided with a test-driver in file *main.cpp* that you can use to test your lexical analysers. Make sure that they work with the provided test-driver (we will also use it, as is, for testing your submission). You are furthermore provided with a sample TP program(s) in a sub-folder called *test*, which you can use for your testing. However, you also need to write additional test files to be able to thoroughly test your lexical analysers. The main program takes two optional command-line arguments: a flag indicating which parser to use (*-h* for hand-made and *-f* for flex-made) and a name of a TP program file to read.

Section 1

You are provided with a skeleton code of a hand-written lexical analyser for TP in files *hlexer.h* and *hlexer.cpp*. Implement a fully functional lexical analyser for the programming language TP using the two files as a starting point. You are free to change the *HLexer* class as you see fit (both in the .h and .cpp files) as long as you honor the interface of the abstract super-class *Lexer* and that of the current *HLexer* constructor (needs to work with the *main.cpp* test driver).

Section 2

You are provided with skeleton of a Flex-based lexer in the files *flexer.h*, *flexer.cpp*, and *flexer.l*. You are only allowed to change the *flexer.l* file. Extend the code in there by writing regular expressions that define all the required tokens. Note, there is no need for you to include a main program in the Flex file, as the *flexer.h* file provides a wrapper around the code generated by Flex that is consistent with the *Lexer* class interface. Instead simply use the test driver in *main.cpp* to test your code. Refer to the documentation for the Flex tool for details regarding Flex.

Language Details

The programming language is case-insensitive; for example, the identifiers X and x both refer to the same variable, and the program keyword can be written e.g. as *PROGRAM*, *Program*, *program*, or even *proGram*.

An *identifier* name consists of one or more letters (a-z,A-Z), an underscore (_), and digits (0-9), but with the restriction that its name cannot start with a digit.

An *integer* value consists of one or more digits (0-9).

A *real* value consists of an mandatory *integral part* (one or more digits), followed by an optional *fractional part* (a period followed by one or more digits), followed by an *exponent part* (the letter 'E' or 'e', an optional sign (+/-), and then one or more digits). If the fractional part is present, then the exponent part is optional but otherwise it is mandatory.

A *string* literal starts and end with the symbol ' (e.g., 'Hello world!') and must be written on a single line.

A *boolean* truth value is either *true* or *false*.

All keywords are reserved and a special token should be returned for each of them.

White-spaces consist of spaces, tabs, returns, and newlines.

Two forms of comments are supported, both block-comments (can span more than a single line). The former form starts with { and ends with a }, whereas the latter form starts with *(** and ends with a ***)*. Comments (of the same form) are not nested.

Hand-in Instructions

- You are expected to work in a group of **two students** (instructor permission needed if you want to work alone).
- Hand in the following files through *Canvas*:
  - *hlexer.h*, *hlexer.cpp*, and *flexer.l*.
  - A single TP test file called *mytest.pas* that uses all possible tokens and both type of comments.

  Keep in mind that your code will be tested by linking and running your code using the *main.cpp* program file on a variety of input. Make sure that your code compiles and links correctly with the provided main program.