Details on the SPL Lexer
($Revision: 1.29 $)

Gary T. Leavens
Leavens@ucf.edu

October 6, 2024

1 Purpose

This document gives some details about the lexical analyzer for the SPL language, and building it with the `flex` lexical analyzer generator [2].

To implement this assignment you will need to write:

1. the input for the `flex` lexical analyzer generator in a file named `spl_lexer.l` (see below for more about this), and
2. a main program (e.g., in a file named `lexer_main.c`) that has the behavior specified in this document.

2 What to Read

You may want to refer to *The flex Manual* [2].

You might also want to read *Systems Software: Essential Concepts* [1] in which we recommend reading chapter 5 (pages 81–91).

3 Overview

The SPL language itself is described in the *SPL Manual*, which is available in the files section of webcourses. The *SPL Manual* defines the lexical grammar of SPL (and its context-free grammar and its semantics).

The following subsections specify the interface between the Unix operating system (as on `eustis.eecs.ucf.edu`) and the lexer as a program.

3.1 Inputs

The lexer is passed a single file name as its only command line argument; this file should be the name of a (readable) text file containing the program that the lexer should produce tokens for. Note that this program file is not necessarily legal according to the grammar for SPL, and part of the task you have is to produce appropriate error messages for lexical errors. (Furthermore, this homework is *not* about checking the context-free grammar of programs, it is only checking the lexical grammar. Thus parsing or syntax errors according to the context-free grammar of SPL are out of scope for this homework.) For example, if the file name argument is `hw2-test1.spl` (and both the lexer executable, `./lexer`, and the

hw2-test1.spl file are in the current directory), then the lexer should work on hw2-test1.spl and send all its output to standard output, except for error messages, which are sent to standard error output. The follow command redirects both output streams to the file hw2-test1.myo:

```
./lexer hw2-test1.spl >hw2-test1.myo 2>&1
```

The same thing can also be accomplished using the make command on Unix (using the provided Makefile):

```
make lexer hw2-test1.myo
```

3.2 Outputs

The lexer prints its normal output, as specified below, to standard output (stdout). However, all error messages (e.g., for unreadable files or illegal tokens) should be sent to standard error output (stderr). See subsection 4.6 for more details about error messages.

3.3 Exit Code

When the lexer finishes without any errors, it should exit with a zero error code (which indicates success on Unix). However, when the lexer encounters an error it should terminate with a non-zero exit code (which indicates failure on Unix).

3.4 Tokens

The primary function that the parser will use (in future assignments) to obtain tokens is yylex, which has the following interface:

```
extern int yylex();
```

The function will be automatically generated from the specifications given in the file spl_lexer.l, which the flex tool uses to generate code in the files spl_lexer.c and spl_lexer.h.

Although the interface specified above says that yylex returns an **int** value, the tokens that the lexer returns are actually members of an enumerated type (yytokentype) defined in the provided file spl.tab.h.

(The spl.tab.h header file was generated by the (context-free) parser generator bison, and helps define the communication between the parser and lexer. (The spl.tab.h file is provided in hw2-tests.zip and should not need to be regenerated, although the parser generator tool, bison, can regenerate it. However, if make says that spl.tab.h is out of date, then it is best to extract spl.tab.h again from the hw2-tests.zip file and make sure that its modification time is current; on Eustis this can be done by using the command touch spl.tab.h.)

4 Output

The output consists of a listing of the tokens read from the input file, as printed by the lexer_output function shown in Figure 1. This function, along with the others defined in lexer.h should be put into the "user code" section of the flex input file spl_lexer.l for this project. This code is also found in the file spl_lexer_user_code.c, from where it can be copied into the user code section of the flex input file spl_lexer.l. Some of this code must be in the user code section of the lexer, because it uses some variable names that are only available in the generated lexer (especially those whose names start with the characters "yy" such as yytext, yylval, and yyin).

```
// Print information about the token t to stdout
// followed by a newline
void lexer_print_token(enum yytokentype t, unsigned int tline,
                       const char *txt)
{
    printf("%-6d %-4d \"%s\"\n", t, tline, txt);
}


/* Read all the tokens from the input file
 * and print each token on standard output
 * using the format in lexer_print_token */
void lexer_output()
{
    lexer_print_output_header();
    AST dummy;
    yytoken_kind_t t;
    do {
        t = yylex(&dummy);
        if (t == YYEOF) {
            break;
        }
        lexer_print_token(t, yylineno, yytext);
    } while (t != YYEOF);
}
```

Figure 1: The provided functions `lexer_print_token` and `lexer_output` that print tokens from the input file. This code can be copied from the file `spl_lexer_user_code.c`, which is included with the `hw2-tests.zip` file. Note that the function `lexer_print_output_header` is also provided in the same way.

4.1 A Simple Example

Consider the input in the file `hw2-errtest1.spl`, as shown in Figure 2, which is included in the `hw2-tests.zip` file in the files section of webcourses.

```
% $Id: hw2-errtest1.spl,v 1.1 2024/10/06 12:06:48 leavens Exp $
% Some illegal characters
a_bad_id
<>
procedure
!
```

Figure 2: The lexer test file `hw2-errtest1.spl`.

This is expected to produce the output found in the (provided file `hw2-errtest1.out`), as shown in Figure 3.

4.2 Starting `spl_lexer.l` File

To start writing the `spl_lexer.l` file, first execute the command (on a Unix command line):

```
Tokens from file hw2-errtest1.spl
Number Line  Text
258    3    "a"
hw2-errtest1.spl:3: invalid character: '_' ('\0137')
258    3    "bad"
hw2-errtest1.spl:3: invalid character: '_' ('\0137')
258    3    "id"
288    4    "<"
290    4    ">"
258    5    "procedure"
hw2-errtest1.spl:6: invalid character: '!' ('\041')
```

Figure 3: Expected lexer output on stdout and stderr from running the lexer on `hw2-errtest1.spl`, which produced the file shown (which is `hw2-errtest1.out`).

```
make start-flex-file
```

which will concatenate the provided files `spl_lexer_definitions_top.l` and `spl_lexer_user_code.c` together into an outline for the file `spl_lexer.l`. See the comments in that file for where to write any definitions and rules for flex.

4.3 Provided Code

We provide some code for printing the tokens (which is in the provided file `spl_lexer_user_code.c` and should be copied into your `spl_lexer.l` file) to run tests with the proper output formatting. After your program opens the given file in the lexer, by calling `lexer_init`, your program should call `lexer_output` to run the lexer. The `lexer_output` function asks for each token (in the lexer's file), until the lexer reaches an end-of-file, and it also prints the tokens in a standard format.

4.4 Provide Test Files

Tests for the lexer are found in the files named `hw2-test*.spl` and `hw2-errtest*.spl`, where `*` is replaced by a number (or letter). The expected output of each test is found in a file named the same as the test input but with the suffix `.out`. For example, the expected output of `hw2-test3.spl` is in the file `hw2-test3.out`.

You can check your own lexer by running the tests using the following Unix command on Eustis:

```
make check-outputs
```

Running the above command will generate files with the suffix `.myo`; for example your output from test `hw2-test3.spl` will be put into `hw2-test3.myo`.

4.5 Do Not Change the Provided Files

You must not change any of the provided `.h` or `.c` files. You must use the provided files in your program.

4.6 Errors that Must be Detected

Your code must detect the following errors:

1. A number's value that is too large to be contained in a **`int`** variable (use `INT_MAX` from the standard header file `limits.h` to determine if the number's value is outside the range that the implementation of C allows).

2. A character in the input is not one of the characters permitted by the lexical grammar (i.e., the character cannot be part of a token and is not one of the recognized whitespace characters). However, note that any character is allowed inside a comment.

Lexical error messages should be sent to `stderr`; they should start with a file name, a colon, and the line number where the error occurred, followed by a colon and a space.

Use the provided function `yyerror` (found in `spl\_lexer_User_code.c`, which should be copied into your `spl_lexer.l` file) to produce such error messages.

There are examples of programs with lexical errors in the files named `hw2-errtest*.spl`, where `*` is replaced by a number (or letter). The expected output of each test is found in a file named the same as the test input but with the suffix `.out`. For example, the expected output of `hw2-errtest3.spl` is in the file `hw2-test3.out`.

A Hints

You need to write the following files:

`spl_lexer.l`, which is the specification of the lexer for the flex tool [2]. You can start doing this by copying the provided parts into the file. First put the contents of the provided file named `spl_lexer_definitions_top.l` at the top and the contents of the provided file `spl_lexer_user_code.c` at the bottom. This process can be automated by using the `make` command with the target `start-flex-file` to put the top and bottom together. Then you need to fill in any definitions you want and the regular expressions and actions in the rules section.

`lexer_main.c`, which is a main program that calls `lexer_init` and then `lexer_output`.

For an example of what the input to the flex tool looks like, see the file `asm_lexer.l` from the first homework (this was provided in hw1-tests.zip) or the flex input in `float_lexer.l` available from this course's example code page.

We are providing several files for this homework, all of which are in the `hw2-tests.zip` file in the files section's hw2 directory on webcourses. These files include the following.

- A `Makefile` with targets `create-outputs` and `submission.zip` among others.
- `spl.tab.h` and `spl.tab.c`, which are generated by the parser generator (bison) and used for communication with the generated lexer. For this assignment, the `spl.tab.c` file is not useful, but your lexer's rules will need to return elements of the `yytokentype` enumeration defined in `spl.tab.h`, as described in class.
- `spl_lexer_definitions_top.l` and `pl_lexer_user_code.c`, which provide the top and bottom sections of the file `spl_lexer.l` that you need to write. You can use the `make` command with the target `start-flex-file` to put these together and then begin editing `spl_lexer.l`
- `utilities.h` and `utilities.c`, which provide the `bail_with_error` function you can use to write error messages about situations that are not lexical errors to `stderr` and then exit.

- `file_location.h` and `file_location.c`, which provide a type (`file_location` that stores a file name and line number).

The following modules are also provided, but are of less concern for this homework.

- `ast.h` and `ast.c`, which provide abstract syntax trees used for identifiers and numbers by the lexer.
- `machine_types.h` and `machine_types.c`, which as in the first homework, provide some type definitions that will be used later in code generation.
- `lexer.h` and `lexer.c` which provide declarations for the functions exported by the generated lexer to the parser. (The `lexer.c` file is mostly empty, as the code is all in the `spl_lexer.c` file that flex generates.)

References

[1] Euripides Montagne. *Systems Software: Essential Concepts*. Cognella Academic Publishing, 2021.

[2] Vern Paxson, Will Estes, and John Millaway. The flex manual. `https://westes.github.io/flex/manual/`, Oct 2016. For Flex version 2.6.2.