Assignment 1: Flex Compilation Process and Corner Cases

Sarthak Kalankar

January 27, 2024

1 Introduction

This document focuses on the process of compiling the Flex file and handling specific corner cases that might arise during the compilation.

2 Compilation Steps

To compile the Flex file, follow these steps:

1. Open the directory "210935-assign1".
2. For problem<x>, do "cd problem<x>", then run the following command:

   ```
   flex -o prob<x>.c prob<x>.l
   ```

3. Compile the generated C source file using a C++ compiler like g++. The command is as follows:

   ```
   g++ -o prob<x> prob<x>.c -lfl
   ```

4. Run the generated executable to perform pattern matching on desired testcases:

   ```
   ./prob<x> < testcases/<testcasefilename.extension>
   ```

3 Handling Corner Cases - Problem 1

Below are some common issues and their solutions:

3.1 KEYWORDS, OPERATORS

If there are different occurrences of the same keyword multiple times but in different case variations, we maintain a single counter for it, although all the different variations will be shown in the output.

3.1.1 Input

The input consists of different variations of the word "Array":

```
Array
ARRAY
ArRay
arRAY
ARRAY
GEQ geq > <<
```

3.1.2 Output

The output demonstrates how each variation is identified as a keyword:

```
Illegal character '>' at line 6
OPERATOR 1 <<
KEYWORD 5 ARRAY
KEYWORD 5 ArRay
KEYWORD 5 Array
OPERATOR 2 GEQ
KEYWORD 5 arRAY
OPERATOR 2 geq
```

3.2 IDENTIFIERS

For invalid identifiers, we will tokenize them in the form of valid tokens.

3.2.1 Input

The input consists of different kinds of valid and invalid identifiers:

```
7abc82
g8reat
nice666
```

3.2.2 Output

The output demonstrates how each identifier is identified/tokenised:

```
INTEGER 1 7
IDENTIFIER 1 abc82
IDENTIFIER 1 g8reat
IDENTIFIER 1 nice666
```

3.3 STRINGS

For invalid strings,

3.3.1 Input

The input consists of different kinds of valid and invalid strings:

```
"hello'abc"
"Hello
World
Where"
'Hi guys'
"This is a
wrong thing

"No"This is the Kanpur City
>> ? CS335 Compilers
```

3.3.2 Output

The output demonstrates how each string is identified:

```
Illegal string "hello'abc" -> ending at line 1
Illegal string "This is the Kanpur City
>> ? CS335 Compilers  -> ending at line 10
STRING 1 "Hello
World
Where"
STRING 1 "This is a
wrong thing

"
STRING 1 'Hi guys'
IDENTIFIER 1 No
```

3.4 NUMERIC LITERALS

For invalid literals, we tokenize them to make the longest possible valid literals.

3.4.1 Input

The input consists of different kinds of valid and invalid literals:

```
.7
78.
65.893735213
-6
001;
0x0
0x455
0X7236
0x0012
0x0gagdj
0x056
0;
```

3.4.2 Output

The output demonstrates how each literal is tokenized:

```
Illegal character '.' at line 1
Illegal character '.' at line 2
OPERATOR 1 -
INTEGER 4 0
HEXADECIMAL 1 0X7236
HEXADECIMAL 4 0x0
HEXADECIMAL 1 0x455
INTEGER 1 1
INTEGER 1 12
INTEGER 1 213
INTEGER 1 56
INTEGER 1 6
FLOATING_POINT 1 65.893735
INTEGER 1 7
INTEGER 1 78
DELIMITER 2 ;
IDENTIFIER 1 gagdj
```

If the handling of literals like "0001" should be done by flagging them as invalid. We can add these rules to line 65 in the file prob1.l.

```
"0x0"{HEX_DIGIT}+                          { cout << "Illegal
    hexadecimal " << yytext <<" at line "<< yylineno << endl; }
"0"{DIGIT}+                                { cout << "Illegal
    integer " << yytext <<" at line "<< yylineno << endl; }
```

For floating points also, ".7" has been tokenized to "." and "7", same with "78.", for "65.893735213" floating point has been truncated to 6 decimal digits and "213" is identified as integer

3.5 COMMENTS

For unclosed comments, we tokenize it to make "{" a delimiter and then longest possible valid tokens.

3.5.1 Input

The input consists of different kinds of valid comments and as a delimiter:

```
{hello'abc}
{Hello
World
Where}
{Hi guys}
{This is a
wrong thing

}No{This is the Kanpur City
>> ? CS335 Compilers
```

3.5.2 Output

The output demonstrates how each comment is identified:

```
Illegal character '?' at line 10
OPERATOR 1 >>
IDENTIFIER 1 CS335
IDENTIFIER 1 City
IDENTIFIER 1 Compilers
IDENTIFIER 1 Kanpur
IDENTIFIER 1 No
IDENTIFIER 1 This
IDENTIFIER 1 is
IDENTIFIER 1 the
DELIMITER 1 {
```

4 Handling Corner Cases - Problem2

For names with more than 63 characters, they are reported as invalid names.