Decibel

Lexical analysis phase

Lexical Analyzer

- We used Flex to create Decibel's lexical analyzer
- **`lexer.l`** contains all the rules to identify the lexemes
- Using these rules, It currently identifies the lexemes and returns the corresponding token
- All the tokens have been defined in **`parser.y`**

```
{alpha}({alpha}|{digit}|_)* { return IDENTIFIER; }
{digit}+ { return INT_LITERAL; }
({digit}*"."{digit}+|{digit}+"."{digit}*) { return FLOAT_LITERAL; }
({digit}*"."{digit}+|{digit}+"."{digit}*|{digit}+)(hz|HZ|hZ|Hz) { return FLOAT_LITERAL; }
({digit}*"."{digit}+|{digit}+"."{digit}*|{digit}+)(ms|Ms|MS|mS) { return FLOAT_LITERAL; }
({digit}*"."{digit}+|{digit}+"."{digit}*|{digit}+)(s|S) { return FLOAT_LITERAL; }
\"[^"]*\" { return STRING_LITERAL; }
\'[^']*\' { return STRING_LITERAL; }
```

Lexical Patterns

- **Keywords, operators and syntax tokens:** They are matched to the corresponding tokens
- **Identifiers:** Any token that starts with an alphabet and contains only alphabets, digits or underscores which is not a keyword will return IDENTIFIER
- **Integer literals:** A token that contains only digits. Returns INT_LITERAL
- **Float literals**: A token that contains digits, and optionally a single ‘ . ’ . It can also have a case-insensitive suffix of “hz”, “s” or “ms”. Returns FLOAT_LITERAL

Lexical Patterns Cont'd

- **String literals:** A token starting with double quotes(“) and ending with double quotes(“), with no double quote(“) in between. Similar rule follows for single quotes(‘). It returns STRING_LITERAL
- **Comments:** Single line comments start with // and multiline comments are enclosed within /* and */. Comments are skipped
- **Newlines, spaces and tabs** are ignored. Any other character that does not match with any rule will result in INVALID_SYMBOL

Architecture

- We use makefile to build, run and test the lexical analyzer
- All the files related to lexical analyzer are present in `Lex` folder
- `lexer.l` is the main lexer file
- `make build` command builds the lexical analyzer and creates the binary
- `make run` command runs the binary generated by `make build`
- `Tests` subfolder contains all the tests for lexical analyzer
- `tester.c` in `Tests` runs the lexical analyzer and prints the tokens
- Each test consists of one input file(.in) and one expected output(.out) file

Architecture Cont'd

- `make test` command runs all the tests and checks the output against expected output using diff command and prints the results
- `make test FILE=<testName>` command runs the particular test and reports any differences between lexical analyzer output and expected output

```
 1  test1 [ PASSED ]
 2  test2 [ PASSED ]
 3  test3 [ PASSED ]
 4  test4 [ PASSED ]
 5  test5 [ PASSED ]
 6  test6 [ PASSED ]
 7  test7 [ PASSED ]
 8  test8 [ PASSED ]
 9  test9 [ PASSED ]
10  test10 [ PASSED ]
11  test11 [ PASSED ]
12  test12 [ PASSED ]
13  test13 [ PASSED ]
14  test14 [ PASSED ]
15  test15 [ PASSED ]
16  test16 [ PASSED ]
17  test17 [ PASSED ]
18  test18 [ PASSED ]
19  test19 [ PASSED ]
20  test20 [ PASSED ]
21  test21 [ PASSED ]
22  test22 [ PASSED ]
```

Changelog

- A new keyword, ‘to’ has been added to enable range specification in loops instead of ‘..’
- If a user types “0..10” the lexical analyser recognises at “0.” and “.10” treating them as FLOAT_NUMBER
- As a result a new token ‘**to**’ was introduced to avoid such problem

```
// Previous syntax
loop over <identifier> <start>..<end> [@<step_integer>] {
    <statements>
}

// New syntax
loop over <identifier> <start> to <end> [@<step_integer>] {
    <statements>
}
```

Contributions

Name	Lines of code
Aditya Garg	152
Anirudh Saikrishnan	298
Bonthu Mani Hemanth Reddy	397
Edward Nathan Varghese	171
Kaipa Venkata Tuhil	385
Shaik Police Parvez	116
Syed Abrar	142
Total	**1661**

Thank You