Lab4 - Nested Cases Scanner

Due Oct 2 at 11:59pm **Points** 100 **Questions** 1
Available Sep 26 at 12am - Oct 2 at 11:59pm 7 days **Time Limit** None
Allowed Attempts Unlimited

This quiz was locked Oct 2 at 11:59pm.

Attempt History

	Attempt	Time	Score
KEPT	Attempt 2	less than 1 minute	85 out of 100
LATEST	Attempt 2	less than 1 minute	85 out of 100
	Attempt 1	148 minutes	0 out of 100 *

* Some questions not yet graded

Correct answers are hidden.

Score for this attempt: **85** out of 100
Submitted Sep 26 at 5:30pm
This attempt took less than 1 minute.

Question 1

85 / 100 pts

Consider the following grammar for a simple programming language:

```
<program> ::= <statement> | <program> <statement>
<statement> ::= <assignStmt> | <repeatStmt> |
<printStmt>
<assignStmt> ::= <id> = <expr> ;
<repeatStmt> ::= repeat ( <expr> ) <statement>
<printStmt> ::= print <expr> ;
<expr> ::= <term> | <expr> <addOp> <term>
<term> ::= <factor> | <term> <multOp> <factor>
<factor> ::= <id> | <number> | - <factor> | ( <expr> )
<id> ::= <letter> | <id> <letter>
```

```
<letter> ::= a | b | c | d | e | f | g | h | i | j
| k | l | m | n | o | p | r | s | t
| u | v | w | x | y | z
<number> ::= <digit> | <number> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<addOp> ::= + | -
<multOp> ::= * | / | %
```

Implement a scanner for the language over this grammar using the **nested cases** approach.

Start with determining what should be the tokens in this grammar. Then, using a pencil and paper, draw an FSM for the scanner. Verify that the FSM indeed accepts the tokens of the language over this grammar. When you convince yourself that the FSM is correct, start coding.

Write the scanner following the guidelines on the nested cases approach from the lectures and the code of the ad-hoc scanner given in files `scanner_ad_hoc.c` and `scanner.h` included in the **`lab4_scanner_1.zip`** archive.

Do not use any library-based scanning facilities like `strtok()` or `strsep()` in your implementation.

Test your scanner on the following program:

```
firstvar = 1;
secondvar = 2;
repeat (10)
 thirdvar = 2 * (firstvar + secondvar) / (firstvar +
2);
repeat (firstvar + 2 * secondvar)
 repeat (thirdvar)
  print firstvar;
```

Print the tokens as they are recognized by the scanner as follows:

```
{<token> value}
```

For example:

```
{<identifier> firstvar}, {<assignment>}, {<number> 1},
{<semicolon>}, {<identifier> secondvar},
{<assignment>}, {<number> 2}, {<semicolon>}, {<keyword>
repeat}, {<lparen>}, ...
```

Your scanner should be implemented in a separate file (`scanner.c`) and called for each token from the driver (that you should implement in `main.c`) that should take input either from a file or from `stdin` if no file is specified as a command line argument (recall `argv` and `argc` prameters of `main()`). The scanner should return a pointer to a structure `TOKEN` that includes token type and a union for any extra information. For example, numbers need their values, identifiers need their names, as do keywords, etc. Token types are already declared through `enum` in `scanner.h`.

The `main()` should print the token using the return value; one token at a time.

You may use C `...` extensions for switch statement that allow to specify ranges of values for cases; for example:

```
switch (character) {
 case 'a'...'z':
  ...
  break;
 case '0'...'9':
  ...
  break;
 default:
  ...
}
```

You can also use functions like `isdigit()`, etc.

You will also find it useful to get familiar with `ungetc()`, because it will allow you to put a character back into the stream, so that when you start the scanner next time it starts with the same character. That approach will lead to a simpler code, as you do not have to store any character for future tokens.

You may also want to explore `freopen()`, so your input can come from either a file or `stdin`.

Please submit a zip archive of your complete CLion project directory. Include a photo-scan of the pencil drawing showing your FSM.

[KeithSkinnerLab03.zip (https://cilearn.csuci.edu/files/611323/download)](https://cilearn.csuci.edu/files/611323/download)

- FSM not submitted - you should have followed the specifications literally; that is you should have used states independent from the token types - these are two different concepts that should not be mixed. There are states in FSM that are not final and only final states are associated with token types.

Quiz Score: **85** out of 100