# Implement record.l (Part of GPL P2)

## Objective

The objective of this lab is to implement the specification file, **record.l**, for the lexical analyzer generator, **flex**. The completed **record.l** can be reused for GPL **P2**.

## Prerequisites

- GNU **g++**. `$g++ --version`
- Flex. `flex --version`
- Bison. `bison --version`
- Latest version of the course repository.

```
$cd <path>/<to>/CSC355_Student
$git pull origin
```

- Lab2 directory in your repository (e.g., `$CSC355_telim/Labs/Lab2$`).

## Detail

In `$CSC355_Student/Labs/Lab2$` directory, you will find the following files: **record.l**, **Makefile**, **error.cpp**, **error.h**, **parser.cpp**, **parser.h**, **y.output**, **y.tab.c**, and **y.tab.h**. From these files, the only file that you will edit in this lab is **record.l** (Note that **y.output**, **y.tab.c**, and **y.tab.h** are the files generated from compiling **record.y**, parser generator specification, with **bison** and **g++**. In **P2**, you will generate the listed files by completing **record.y**).

## Task

The first thing you will do is to copy all files in `$CSC355_Student/Labs/Lab2$` to your `Lab2` directory. Then, push these files to your GitHub repository.

The only section that you will edit in the **record.l** file is the **rule** section, i.e., under **%%**. Some of the rules have already been added, but some need your fix so the generated lexer works correctly. You will add/fix the rules as follows:

1. Add a rule to handle double constants returning `T_DOUBLE_CONSTANT` token. A double constant is a number with decimal points (we will not handle scientific `e` notation). A decimal constant can have digits before and after the period (e.g., 3.1) , digits only after the period (e.g., .42 instead of 0.42), or digits only before the period (e.g., 42. instead of 42.0) Tests: t003.in/t003.out, t004.in/t004.out.

2. Add a rule to handle string constants returning `T_STRING_CONSTANT` token. A string constant is any number of characters except `"` wrapped with the double quotations `""`, e.g., "hello world" is a valid string constant while """ is not. Tests: t005.in/t005.out, t006.in/t006.out

3. Add a rule to handle month acronym, i.e., always the first three letters in uppercase (e.g., FEB, AUG), returning `T_MONTH` token. Tests: t008.in/t008.out.

4. Fix the rule so that identifiers (the names) can...
   (a) contain underscores [tests: t009.in/t009.out]
   (b) contain digits if the first character is not a digit [tests: t010.in/t010.out t007.in/t007.out]

Your rule should have the following syntax:

```
<Regex>   <function to handle current token>  return <T_TOKEN>
```

For example, the following rule recognizes one or more digits and calls `get_int_value()` function to convert the digit (note `yytext` returns `char *`, C-style string) to integer, and returns `T_INT_CONSTANT` token.

```
[0-9]+          get_int_value(); return T_INT_CONSTANT;
```

## How to Compile and Test Your Code

In the `Lab2` directory, you will execute `$make` command (make sure you have a **Makefile** in your directory) to compile the program. If successfully compiled, you will see an executable binary file: **parser**. You can run your parser like below:

```
$./parser test/t003.in
parser.cpp::main() reading file tests/t003.in
parser.cpp::main() about to call yyparse()
type zero or more record declarations followed by ^D:


---------------------------------------------------
```

```
record George
{

  height = 68.2 (double)
  age = 42.5 (double)
  weight = 175.42 (double)

} 3 fields were declared.

1 record was declared.
------------------------------------------------------


parser.cpp::main() after call to yyparse()

no errors found by yyparse()

parser.cpp::main() end of program.
```

To test all test files, you can run the `p2_tester.py` python program. Do not move this out from `Lab2` directory. You run the program by executing `python3 p2_tester.py`. Some input files will lead to an error, which is intentional. The parser will print only up to the line it can handle, then halts with an error message. My code handles errors. Take a look at t024.in, t024.out, and t024.err.

## How to Submit Your Code

You will only add your **record.l** file.

```
$cd <path>/<to>/Lab2
$git add record.l
$git commit -m "your message, e.g., lab 2 - record.l"
$git push origin
```