CSE340 Spring B 2021 Project 1: Lexical Analysis

Due: **Sunday, March 21, 2021 by 11:59 pm AZ time**

The goal of this project is to give you hands-on experience with lexical analysis. You will extend the provided lexical analyzer to support more token types. The next section lists all new token types that you need to implement.

1. Token Types

Modify the lexer to support the following 3 token types:

```
REALNUM   = NUM DOT digit digit*
BASE08NUM = ((pdigit8 digit8*) + 0) (x) (08)
BASE16NUM = ((pdigit16 digit16*) + 0) (x) (16)
```

Where

```
digit16  = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + A + B + C + D + E + F
pdigit16 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + A + B + C + D + E + F
digit    = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
pdigit   = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
digit8   = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7
pdigit8  = 1 + 2 + 3 + 4 + 5 + 6 + 7
```

Note that `NUM` and `DOT` are already defined in the lexer, but here are the regular expressions for the sake of completeness:

```
NUM = (pdigit digit*) + 0
DOT = '.'
```

Note that `DOT` is a single dot character, the quotes are used to avoid ambiguity.

The list of valid tokens including the existing tokens in the code would be as follows:

```
IF
WHILE
DO
THEN
PRINT
PLUS
MINUS
DIV
MULT
EQUAL
COLON
COMMA
SEMICOLON
LBRAC
RBRAC
LPAREN
RPAREN
NOTEQUAL
GREATER
LESS
LTEQ
GTEQ
DOT
NUM
ID
REALNUM
BASE08NUM
BASE16NUM
```

This list should be used to determine the token if the input matches more than one regular expression.

2. How-To

Follow these steps:

- Download the `lexer.cc` , `lexer.h` , `inputbuf.cc` and `inputbuf.h` files accompanying this project description. Note that these files might be a little different than the code you've seen in class or
elsewhere.
- Add your code to the files to support the token types listed in the
previous section.
- Compile your code using GCC compiler in `Ubuntu 19.04 or higher` . You will need to use the `g++` command to compile your code in a terminal window.

  Note that you are required to compile and test your code in Ubuntu 19.04 or higher using the GCC compilers. You are free to use any IDE or text editor on any platform, however, using tools available in Ubuntu 19.04 g++ version 4.9 (or tools that you could install on Ububntu 19.04 could save time in the development/compile/test cycle. See next section for more details on how to compile using GCC.
- Test your code to see if it passes the provided test cases. You will need to extract the test cases from the zip file and run the test script `test1.sh` . More details on this in the next section.
- Submit your code in canvas before the deadline:

  For this project you need to update lexer.cc and lexer.h. The updates that you need to do in lexer.h are minimal and are already implemented in the submission website. So you do not need to upload the lexer.h file in CANVAS. But you still do need to update lexer.h if you want to compile locally.

3. Compile & Test

3.1 Compiling Code with GCC

You should compile your programs with the GCC compilers which are available in g++ 4.9 in Ubuntu 19.04. The GCC is a collection of compilers for many programming languages. There are separate commands for compiling C and C++ programs:

- Use `gcc` command to compile C programs
- Use `g++` to compile C++ programs

Here is an example of how to compile a simple C++ program:

```
$ g++ test_program.cpp
```

If the compilation is successful, gcc will generate an executable file named `a.out` in the same folder as the program. You can change the output file name by specifying the `-o` switch:

```
$ g++ test_program.cpp -o hello.out
```

To enable all warning messages of the GCC compiler, use the `-Wall` switch:

```
$ g++ -Wall test_program.cpp -o hello.out
```

The same options can be used with `gcc` to compile C programs.

Compiling projects with multiple files

If your program is written in multiple source files that should be linked together, you can compile and link all files together with one command:

```
$ g++ file1.cpp file2.cpp file3.cpp
```

Or you can compile them separately and then link:

```
$ g++ -c file1.cpp
$ g++ -c file2.cpp
$ g++ -c file3.cpp
$ g++ file1.o file2.o file3.o
```

The files with the `.o` extension are object files but are not executable. They are linked together with the last statement and the final executable will be `a.out` .

NOTE: you can replace `g++` with `gcc` in all examples listed above to compile C programs.

3.2 Testing your code with I/O Redirection

Your programs should not explicitly open any file. You can only use the **standard input** e.g. `std::cin` in C++, `getchar()` , `scanf()` in C and **standard output** e.g. `std::cout` in C++, `putchar()` , `printf()` in C for input/output.

However, this restriction does not limit our ability to feed input to the program from files nor does it mean that we cannot save the output of the program in a file. We use a technique called standard IO redirection to achieve this.

Suppose we have an executable program `a.out` , we can run it by issuing the following command in a terminal (the dollar sign is not part of the command):

```
$ ./a.out
```

If the program expects any input, it waits for it to be typed on the keyboard and any output generated by the program will be displayed on the terminal screen.

Now to feed input to the program from a file, we can redirect the standard input to a file:

```
$ ./a.out < input_data.txt
```

Now, the program will not wait for keyboard input, but rather read its input from the specified file. We can redirect the output of the program as well:

```
$ ./a.out > output_file.txt
```

In this way, no output will be shown in the terminal window, but rather it will be saved to the specified file. Note that programs have access to another standard interface which is called standard error e.g. `std::cerr` in C++, `fprintf(stderr, ...)` in C. Any such output is still displayed on the terminal screen. However, it is possible to redirect standard error to a file as well, but we will not discuss that here.

Finally, it's possible to mix both into one command:

```
$ ./a.out < input_data.txt > output_file.txt
```

Which will redirect standard input and standard output to `input_data.txt` and `output_file.txt` respectively.

Now that we know how to use standard IO redirection, we are ready to test the program with test cases.

Test Cases

A test case is an input and output specification. For a given input there is an *expected* output. A test case for our purposes is usually represented by two files:

- `test_name.txt`
- `test_name.txt.expected`

The input is given in `test_name.txt` and the expected output is given in `test_name.txt.expected`.

To test a program against a single test case, first we execute the program with the test input data:

```
$ ./a.out < test_name.txt > program_output.txt
```

The output generated by the program will be stored in `program_output.txt`. To see if the program generated the expected output, we need to compare `program_output.txt` and `test_name.txt.expected`. We do that using a general purpose tool called `diff`:

```
$ diff -Bw program_output.txt test_name.txt.expected
```

The options `-Bw` tells `diff` to ignore whitespace differences between the two files. If the files are the same (ignoring the whitespace differences), we should see no output from `diff`, otherwise, `diff` will produce a report showing the differences between the two files.

We would simply consider the test passed if `diff` could not find any differences, otherwise we consider the test failed.

Our grading system uses this method to test your submissions against multiple test cases. There is also a test script accompanying this project `test1.sh` which will make your life easier by testing your code
against multiple test cases with one command.

Here is how to use `test1.sh` to test your program:

- Store the provided test cases zip file in the same folder as your project source files
- Open a terminal window and navigate to your project folder using `cd` command
- Unzip the test archive using the `unzip` command:

  ```
  $ unzip test_cases.zip
  ```

  NOTE: the actual file name is probably different, you should replace `test_cases.zip` with the correct file name.

- Store the `test1.sh` script in your project directory as well
- Mark the script as executable once you download it:

  ```
  $ chmod +x test1.sh
  ```

- Compile your program. The test script assumes your executable is called `a.out`
- Run the script to test your code:

  ```
  $ ./test1.sh
  ```

The output of the script should be self explanatory. To test your code after each change, you will just perform the last two steps afterwards.

4. Requirements

Here are the requirements of this project:

- You should submit **all your code on Gradescope.** No need to submit on canvas anymore.
- You should use C/C++, no other programming languages are allowed.

- You should familiarize yourself with the Ubuntu 19.04 environment and the GCC compiler. Programming assignments in this course might be very different from what you are used to in other classes.
- You **cannot use library methods for regular expression (regex) matching in projects**. You will be implementing them yourself.
- You can write helper methods or have extra files, but they **should have been written by you**

5. Evaluation

The submissions are evaluated based on the automated test cases on the Gradescope. Your grade will be proportional to the number of test cases passing. If your code does not compile on the submission website, you will not receive any points. On Gradescope, when you get the results back, ignore the “Test err” case, it is not counted toward the grade.

You can access the Gradescope through the left side bar in canvas, or using this link: (https://www.gradescope.com/courses/254734). You have already been enrolled in the grade scope class, and using the left side bar in canvas you will automatically get into the Gradescope course. If you want to used the link directly, than you should have received an email with title "Welcome to Gradescope for CSE 340 - Spring B" which provides info on setting up your account and password. If you already had a Gradescope account with your asu email, then most probably the course should show up in your Gradescope dashboard.