New Mexico Tech — Fall 2018
Dept. of Computer Science & Engineering — CSE 423

Programming Assignment 1

(Based on Material By: R. Heckendorn)

Posted: Jan. 20, Due: Sunday, Jan. 28 @ 11:59pm

Throughout this semester, you will build a compiler for a C like language called C-. The C- language is rather complex including recursive functions and arrays. You will work in groups of three students (unless explicitly allowed by the instructor to work in groups of more than three students.) You will be guided toward a successful completion by working on incremental development and testing of the compiler.

The output of your program will first be preprocessed by an automatic comparison program before being examined by the instructor and the TA. Please follow formatting instructions/examples very carefully. The results your program produces will need to look exactly like the target. Do not embellish with extra titles or other text such as "run complete" or "CSE 423 output". I will take off points for this. (This is realistic. Most companies run test suites and breaking the test suites is not looked upon well.) The testing facility of the submit script will help you get this annoying detail right. Thanks for your patience.

In this first assignment, use a combination of Flex and Bison to build and test a *scanner* for the C- language. The scanner will be named c- (note the lowercase. That is, c- will ultimately be the compiler for the language C-.). It will read and process a stream of characters representing tokens from a file. The filename may be given as the first argument to the c- command OR the input can come from standard input if the filename argument is not present. This means any of the three following calls to `c-` is a valid way to compile the C- code in file `filename`:

```
c- {filename}
```
or
```
cat filename | c-
```
or
```
c- < filename
```

Any of these calls will produce a stream of tokens as its output as described below. (Pretesting your answer will help assure compliance.) It will be constructed using both Flex and Bison to run on the CSE department Linux machines.

IMPORTANT: That is where the grading will occur and it must compile and run there!

The Flex Part

Build a Flex scanner that returns a token class for each token in the C- grammar. For numbers it should also "return" a numerical value and the string the user typed. For ids it should also return a string. It should treat the Booleans "true" and "false" as keywords but internally treat them as a BOOLCONST with values 0 and 1 for false and true respectively. The scanner should ignore comments and whitespace and not return.

Your compiler will generate an error if there is an illegal character in the input. That occurs when none of the token patterns match the top of input. See the example output for the exact wording of the error message. No token will be returned in the case of an error.

HINT: The scanner should keep track of the line number of each token and return it with the token and a string and/or numeric representation of the token, if appropriate, in a struct or class instance. This can be done by using yylval to contain a pointer to a struct or class instance you create.

CODING RESTRICTIONS: DO NOT USE YYSTYPE!!!!!! I will take off points. This is bad programming practice. Flex/Bison provides %union to make this association possible and that provides type checking so we should use that.

Note that in C-, like C and C++, newline is not an element of the grammar and is merely whitespace.

The Bison Part

Build a Bison parser that accepts any stream of legal tokens from the scanner. This is a simple grammar for just a stream of any legal tokens. It is NOT the grammar for C-!
The Bison part prints out the line number, the token type, and any extra information returned by the scanner. See example output to see what it should look like. Again, this first program will not recognize C-. It will only recognize C- tokens. One of the goals of this assignment is to get the basic build and communication between flex and Bison up and running.

Test Data
Use the test data provided to decide the format for test output.
Note that IDs are not what you think they are. They are not exactly like in C++.
Character constants may be the null character and so print in a way that can't be seen if you use a %c format. That is OK. We will be checking for an actual null character to be output with the %c. Use %c for chars for this assignment.

Note the class of any single special character token is printed as the characters itself. The class of any multicharacter token is printed as an all uppercase string.

Build and Test
You will have at least:

a file parser.l that contains the flex code
a file parser.y that contains the bison code
a file scanType.h that contains the declaration of either a struct or class that is used to pass your token information back from the scanner. This file will be included in the right place in both the .l and .y files.
a makefile (note the all lowercase) that we will execute to build your c-.

We will then run several files containing tokens through your c- and compare the results. We will do multiple runs, running them by both piping data into the file AND by using the filename as an argument.

Submission
The assignment will be submitted as a single uncompressed tar file. Name your file as follows:

```
GroupName_PA1.tar
```

Further instructions regarding the use of the submission script will be given later.

You can submit as many times as you like. The LAST file you submit BEFORE the deadline will be the one graded.

The grading program will use extensive tests, so thoroughly test your program with inputs of your own.

If you have tests you really think are important or just cool please send them to me and TA and we will consider adding them to the test suite.