# CS 445 – Spring 2023

Homework 1

Due on or before Jan 29 at 11:59 PM Pacific Time

**WARNING**

Because the output of your program for this assignment will first be processed by an automatic comparison program before being examined by a human being, please follow formatting instructions/examples very carefully. The results your program produces will need to **look exactly like the examples**. Do not embellish with extra titles or other text such as "run complete" or "CS445 output" or even any extra spaces. Points will be deducted for this. (This is realistic. Most companies run test suites on their products and breaking the test suites is highly frowned upon in industry.) The testing facility of the submit script will help you refine your software to the point that it is completely correct if you use it as intended.

## 1. The problem

Use a combination of Flex and Bison code as instructed in class to build and drive a scanner for the C- programming language. A grammar for C- is available on the course website. Your scanner will be named **c-** (note the lowercase). That is, **c-** will ultimately be the compiler for the language **C-**. Your scanner will read and process a stream of characters representing tokens from a file. The filename may be given as an argument to the **c-** command, or the input can come from standard input if the filename argument is not present. This means that the user can pass the C- code in file `filename.c-` to **c-** in any of the three following ways:

```
$bash> c- filename.c-
$bash> cat filename.c- | c-
$bash> c- < filename.c-
```

To get this to work will require that you be able to optionally read a file from the command line. You will need to define arguments to main and use the yacc/bison variable named `yyin` to read from the file that the user has specified. You should use `fopen` to get a `FILE*` that you then assign to `yyin`.

For this assignment you are to build a scanner using lex/flex and a driver for the scanner using yacc/bison. We will be using both lex and yacc for all other assignments as well, so figuring out how to get them to work together will be worth your effort. The machine cs-445.cs.uidaho.edu is available for class use using your UI credentials. **That is where the grading will occur and your homework must compile and run on this machine. There are no exceptions to this.**

### 1.1 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 that the user typed. For identifiers (ids) it should also return a string. It should treat the Booleans "true" and "false" as keywords, but internally treat them as members of the token class BOOLCONST with values 0 and 1 for false and true respectively. The scanner should ignore comments and whitespace characters.

Your scanner will generate an error if there is an illegal character in the input. This occurs when none of the token patterns match the the current location in the input stream. See the example output for the exact wording of the error message to be used in this case. No token will be returned in the case of this error and scanning will continue.

Your scanner will generate an warning if a character constant is given with more than one character (i.e. 'dogs'). If this happens the first character will be used, the remaining ignored, and a warning will be issued. See the example output for the exact wording of the error message to be used in this case.

Your scanner will generate an error if a character constant is given that contains no characters (i.e. ‘’). No token will be returned and the token ignored. See the example output for the exact wording of the error message to be used in this case.

Your scanner will keep track of the line number of each token encountered and return it with the token and a string and and/or numeric representation of the token, if appropriate, in a struct or class instance. This can be accomplished by using `yylval` to contain a pointer to a struct or class instance that you create and want to return.

You may not use YYSTYPE for this assignment. Points will be deducted if you choose to ignore this warning. Directly accessing YYSTYPE is considered poor programming practice. The flex and bison tools provide the `%union` construct to facilitate defining a symbol type for `yylval` without stripping away the insulation provided by YYSTYPE. Please use `%union` and not YYSTYPE.

### 1.2 The bison Part

Build a simple bison parser that accepts any stream of legal tokens from the scanner. You will have to come up with the simple grammar for this. **This is a grammar for just a stream of any legal tokens, it is not the grammar for C-!** This will serve as a “driver” to drive your scanner (the flex part). 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.

Your program for this homework will just recognize tokens in C- and will not recognize grammatical constructs of C-. One of the goals of this assignment is to get the basic build and communication between flex and bison up and running. A template for constructing your bison file for this assignment is contained on the course website.

### 1.3 The Test Data

Sample test inputs and their corresponding outputs are provided for this assignment. These are available on the course website. You should examine these inputs and their corresponding outputs very carefully and ensure that your program behaves precisely like these examples.

Character constants in test files may be the NUL character (‘\0’) and will therefore print in a way that cannot be seen if you use a “%c” format string. This is expected. Be sure to use the “%c” format string for all characters on this homework. Failure to do so will result in output that does not match the examples, which will result in points being deducted from your homework.

Note in the provided test data that the class of any single special character token is printed as the characters itself, and the class of any multi-character token is printed as an uppercase string. Follow this example in your program to produce results that are the same as the provided examples.

## 2. Deliverables and Submission

Your homework will be submitted as an *uncompressed* .tar file that contains no subdirectories. Your .tar file will contain at least the following items:

1. A file named parser.l that contains your flex code.
2. A file named parser.y that contains your bison code.
3. A file named 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 **must** be #included in the right place in your flex and bison files. An example of the declaration of a struct that can be used to pass token information back from the scanner is shown below:

```
struct TokenData {
    int  tokenclass;        // token class
    int  linenum;           // line where found
    char *tokenstr;         // what string was actually read
    char cvalue;            // any character value
    int  nvalue;            // any numeric value or Boolean value
    char *svalue;           // any string value e.g. an id
};
```

4. A makefile (note the all lowercase) that will be executed to build your **c-**. The name of this file must be in all lowercase letters in order to work correctly on the test machine.

The uncompressed .tar file is submitted to the class submission page. You can submit as many times as you like. **The last .tar file you submit before the deadline will be the only one graded.** No late submissions are accepted for this assignment. For all submissions you will receive email at your uidaho address showing how your file performed on the pre-grade tests. The grading program will use more extensive tests, so thoroughly test your program with inputs of your own.

Your code must compile successfully and have no shift/reduce or reduce/reduce conflicts in bison. Your code must run with no runtime errors such as segmentation violations (SIGSEGVs).