**CSE 340 Principles of Programming Languages**

**Spring 2014**

**Programming Assignment #2**

**Due Date: Friday, February 26th, 2014**

**OBJECTIVE**

Create a syntax analyzer for the grammar listed below (and described in class). You will require to use the lexical analyzer (developed in assignment #1) as part of this project.

**INSTRUCTIONS**

1. Modify your lexical analyzer as follows:
2. Rename the file that contains the main() method of your lexer as “lexer.c” or “lexer.cpp”.
3. Move the main() method to a new file. Name this new file “parser.c” or “parser.cpp”
4. Create a structure or class Token with three attributes: tokenName (string), tokenWord (string), and tokenLineNumber (integer).
5. Inside your parser.c or parser.cpp file, create a LinkedList of Token’s objects/records.
6. Modify your Lexer to store in the LinkedList the information generated by the Lexer instead of writing it on a file. For instance, after analyzing the input below,

Hello world 2013;

0xcse340

The LinkedList must contain 5 objects of type Token with tokenName, tokenWord, and tokenLineNumber as follow:

(“identifier”, “Hello”, 1)

(“identifier”, “world”, 1)

(“integer”, “2013”, 1)

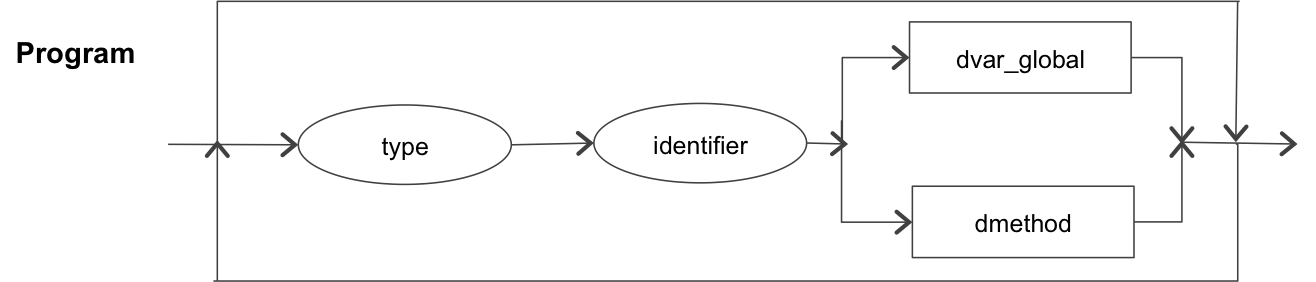
(“delimiter”, “;”, 1)

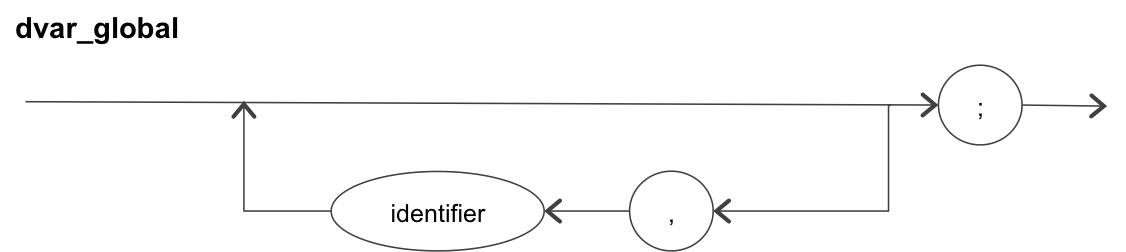
(“error”, “0xcse340”, 2)

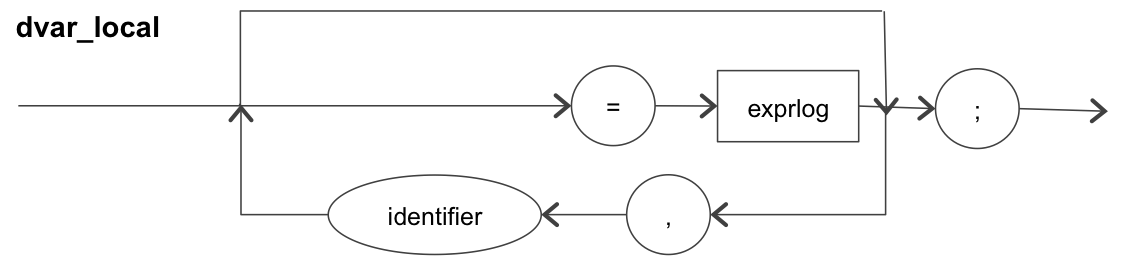
1. Update your Lexer to ensure that the following Tokens are identified:

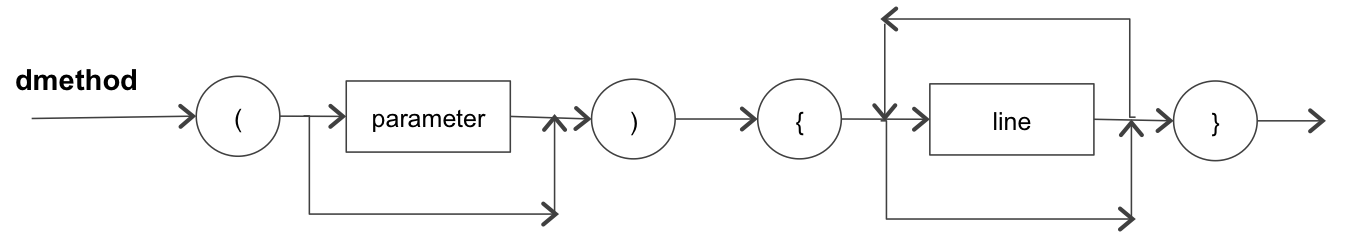
|  |  |  |
| --- | --- | --- |
|  | | **Token** |
| Arithmetic Operators | Addition | + |
| Subtraction | - |
| Multiplication | \* |
| Division | / |
| Reminder | % |
| Assignation | = |
| Logic Operators | And | & |
| Or | | |
| Not | ! |
| Relational Operators | Less than | < |
| Greater than | > |
| Keywords | | read |
| print |
| if |
| else |
| while |
| void |
| int |
| char |
| string |
| float |
| boolean |
| true |
| false |
| Delimiters | Colon | : |
| Semi-colon | ; |
| Coma | , |
| Open parenthesis | ( |
| Close parenthesis | ) |
| Open curly bracket | { |
| Close curly bracket | } |
| Open bracket | [ |
| Close bracket | ] |

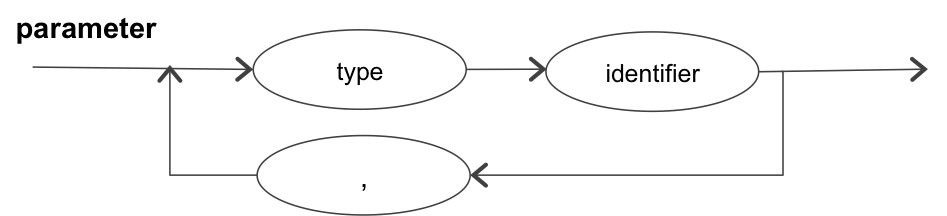
1. Write a recursive predictive descendent parser for the grammar listed below (and described in class). The code must be in the file “parser.c” or “parser.cpp. One method needs to be coded for each of the following rules:

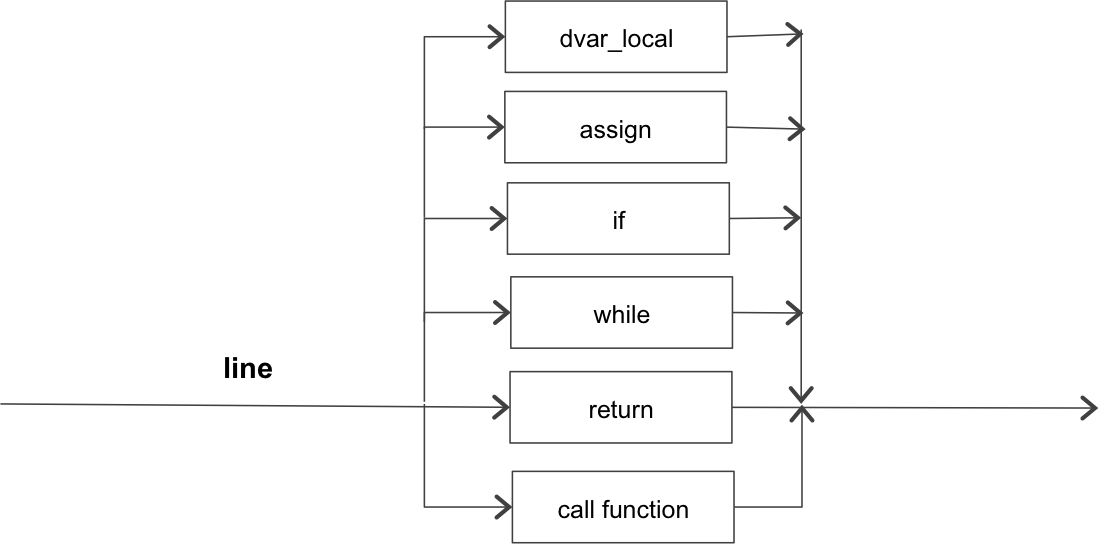


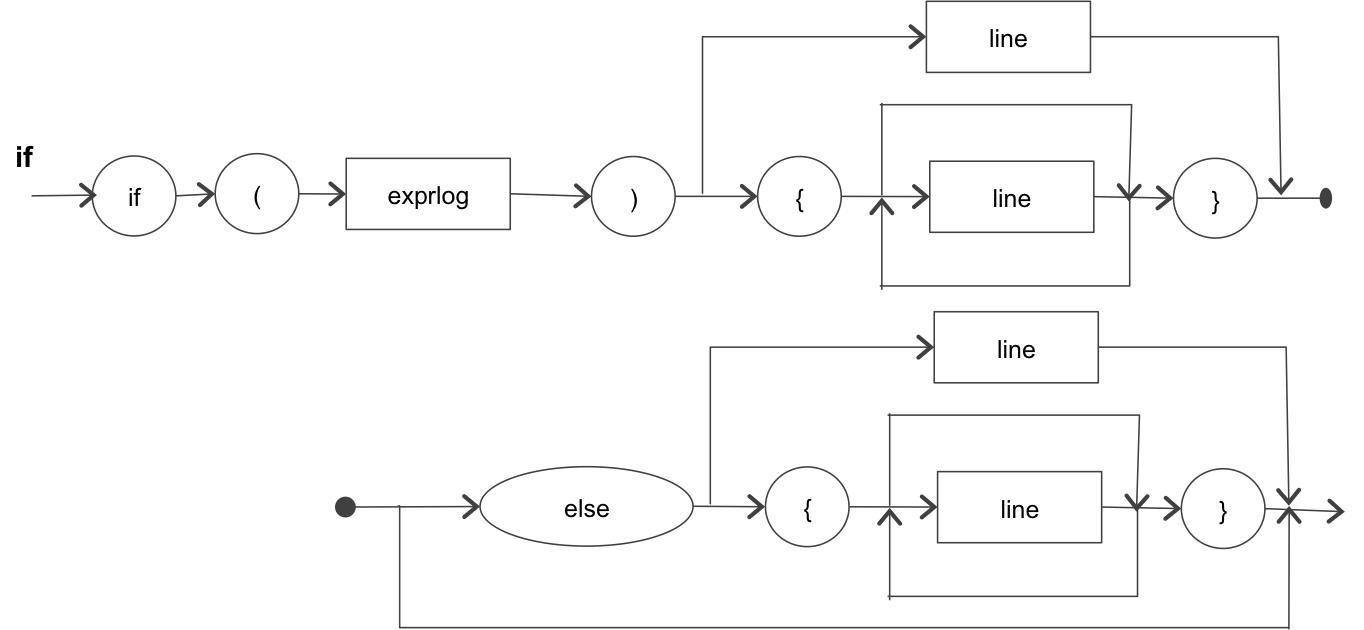


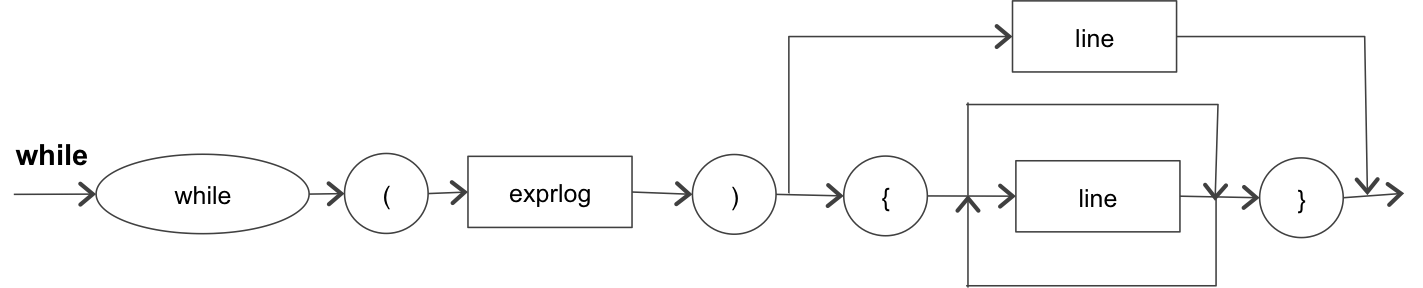


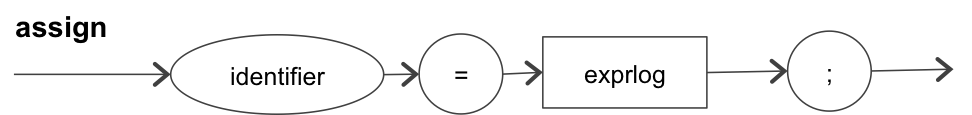


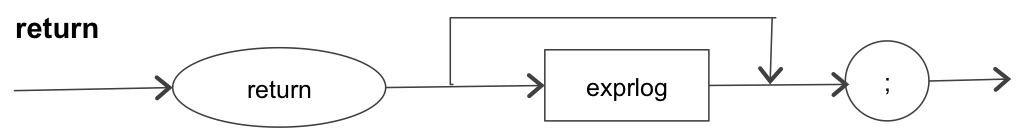


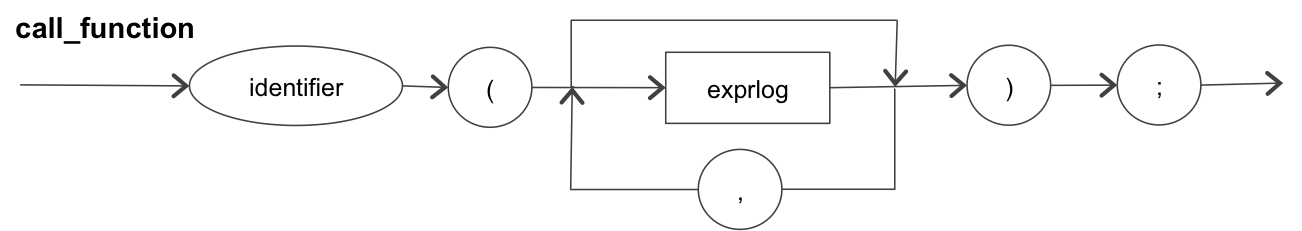


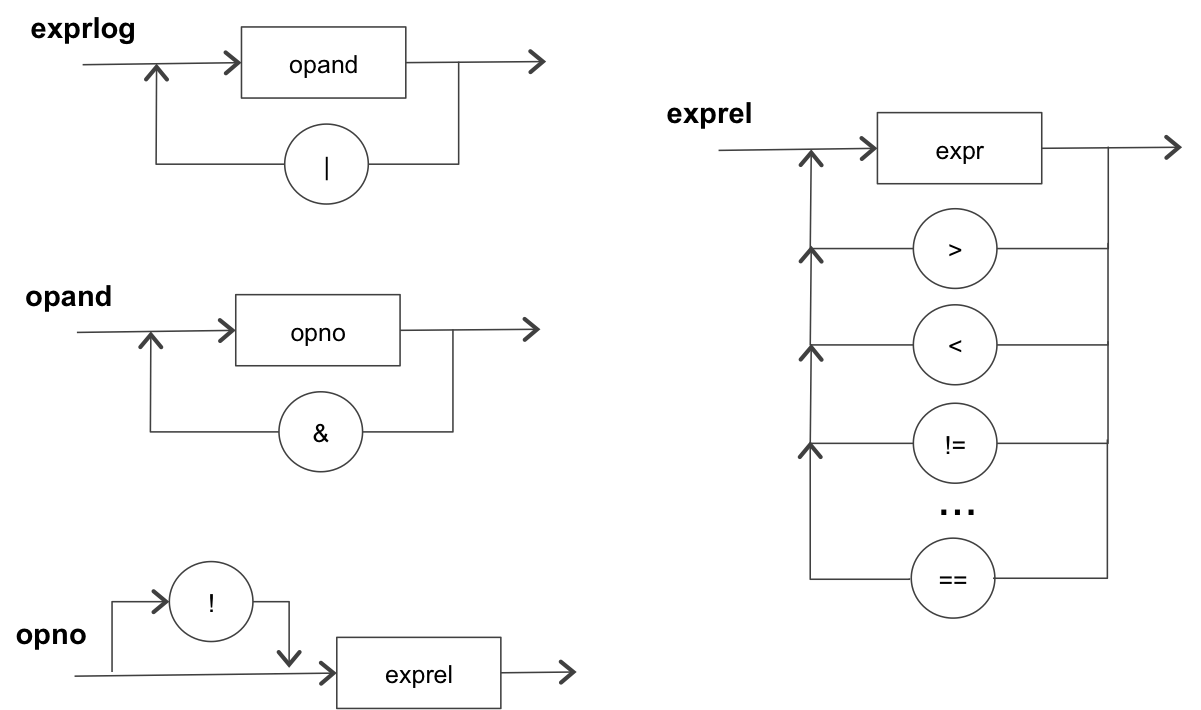


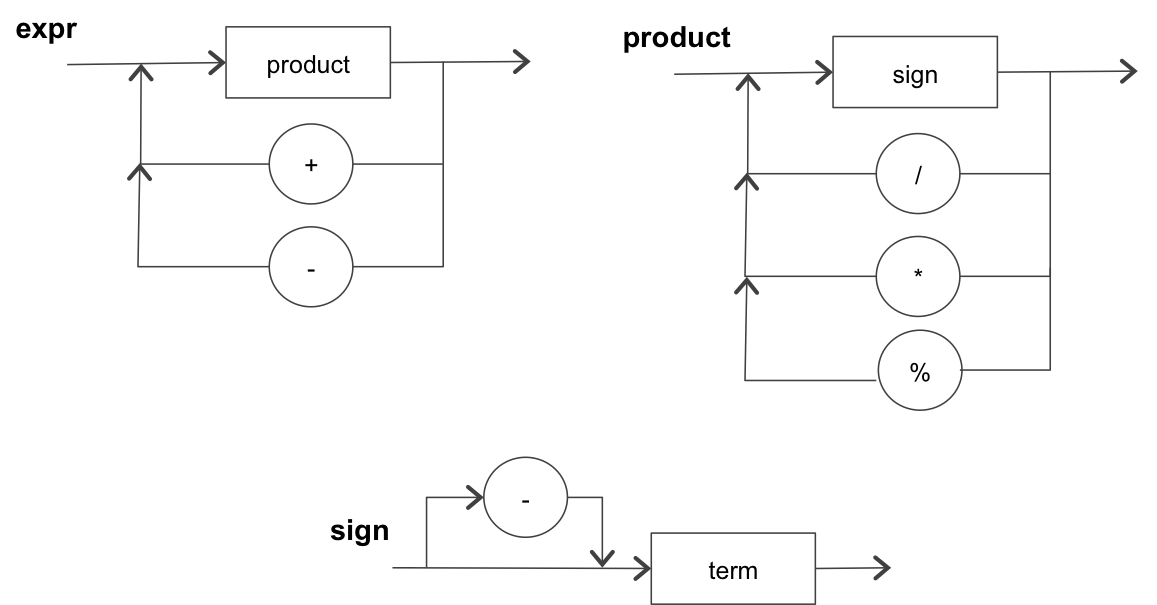


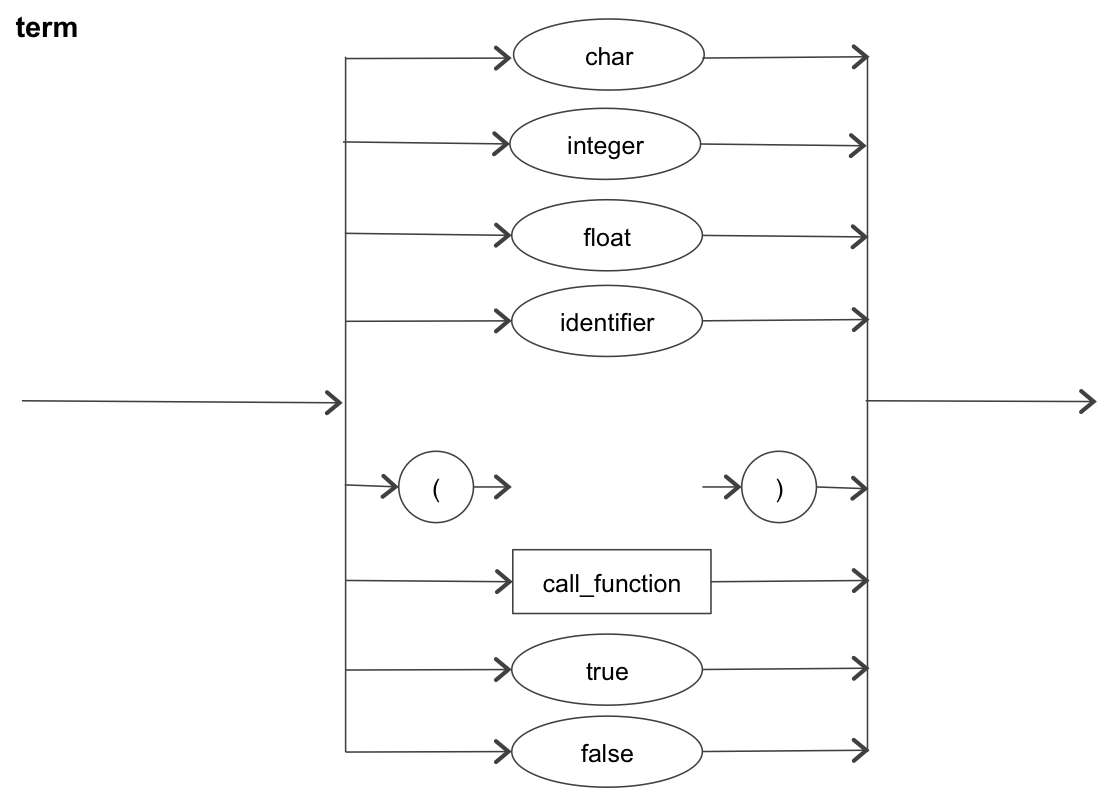












1. The parser must report the errors and the number of the line in which they occur, such as: Delimiter expected, Identifier expected, Type expected, Identifier or value expected, etc. For instance, for this source code:

|  |  |
| --- | --- |
| 1: | int x; |
| 2: | int main ( ) { |
| 3: | int y = x + 0xCSE; |
| 4: | myMethod(1, 2, ); |
| 5: | } |
| 6: | void myMethod(int one, int two) { |
| 7: | print (“hello world”) |
| 8: | } |

In a first step, the Lexer analyzes it and the LinkedList of tokens should be filled as follows:

|  |  |  |
| --- | --- | --- |
| **tokenName** | **tokenValue** | **tokenNumberLine** |
| keyword | int | 1 |
| identifier | x | 1 |
| delimiter | ; | 1 |
| keyword | int | 2 |
| identifier | main | 2 |
| delimiter | ( | 2 |
| delimiter | ) | 2 |
| delimiter | { | 2 |
| keyword | int | 3 |
| identifier | y | 3 |
| operator | = | 3 |
| identifier | x | 3 |
| operator | + | 3 |
| error | 0xCSE | 3 |
| delimiter | ; | 3 |
| identifier | myMethod | 4 |
| delimiter | ( | 4 |
| integer | 1 | 4 |
| delimiter | , | 4 |
| integer | 2 | 4 |
| delimiter | , | 4 |
| delimiter | ) | 4 |
| delimiter | ; | 4 |
| delimiter | } | 5 |
| keyword | void | 6 |
| identifier | myMethod | 6 |
| delimiter | ( | 6 |
| keyword | int | 6 |
| identifier | one | 6 |
| delimiter | , | 6 |
| keyword | int | 6 |
| identifier | two | 6 |
| delimiter | ) | 6 |
| delimiter | { | 6 |
| keyword | print | 7 |
| delimiter | ( | 7 |
| string | “hello world” | 7 |
| delimiter | ) | 7 |
| delimiter | } | 8 |

Using that LinkedList, the parser must report the following:

* Line 3: expected identifier or value
* Line 4: expected identifier or value
* Line 7: expected delimiter

The output should be saved in an output file. The name of the output file is passed as argument to the program when it is executed.

The format for the output file should be:

Line <line\_number>:\t<error\_message>

where “\t” represents a tab, the <line\_number> is the line where the error happens, and the <error\_message> could be:

* expected identifier or value
* expected identifier expected
* expected delimiter
* expected type

1. Read and actively participate in the discussion board about common errors in grammar analysis and use those discussions to improve your implementation.
2. Create a .zip file for your submission, you MUST follow the naming convention Firstname\_Lastname\_Projnumber.zip, this file should contain the following:

* Lexer.c or Lexer.cpp
* Parser.c or Parser.cpp – which should be the class with your main method.
* One new input file and its corresponding output file. You should name the files as **Firstname\_Lastname\_Projnumber\_input.txt** and **Firstname\_Lastname\_Projnumber\_output.txt.**
* A **makefile** to control the compilation of your code. The makefile should have at least a default target that builds your project. Google “makefile tutorial” for how to write makefiles. Before submitting your project, don’t forget to try the makefile at general.asu.edu and make sure it is fully working (able to compile and run).

**Remember that:**

* No credit will be given to programs that do not compile on general.
* No credit will be given to programs that do not execute correctly on general.
* No credit will be given if your program does not create an output file.
* No credit will be given if any file (source files, makefile, or test cases) is missing.