Assignment 2
Parsing (CPSC411)
Due date November 10

1. Description:

In this assignment, your task is to create a parser for the C- programming language building on the scanner that you developed in assignment 1. The parser should read in a source file and determine whether it is valid given C-'s grammar. In the case of success an Abstract Syntax Tree (AST) should be produced, otherwise failure should be indicated. Evaluate the given C- grammar for ambiguities and work to resolve problems such as the dangling-else. Create a set of complete tests to exercise all of the tricky corner cases.

2. Requirements:

You should pay attention to the following requirements while implementing your parser:

I. You can use any language for this assignment as long as it can be executed on the undergraduate Linux servers. You should build on the scanner that you produced for assignment 1.
II. The assignment submission must include:
    a. Instructions on how to run the project and how the output can be interpreted.
    b. Explanation of what the test cases are and what they are testing for.
    c. All files needed to run your code, including the files used to run assignment 1.
III. You are allowed to use parser generating tools such as bison, or cup, so long as they are available on the undergraduate Linux Server. Otherwise, you may build a hand-made parser using recursive-descent or table-driven parsing methods.
IV. You may need to alter the C- grammar into an equivalent grammar to allow you to implement a parser with your chosen method. You may also alter the grammar to catch some semantic errors at this stage, such as variables and parameters having types of either "int" or "void" (It is later specified in the semantics that they may only be of type "int").
V. Your AST should follow the precedence and associativity rules for the C- language, this structure should be clear from the output format.

For other examples with valid and invalid inputs, please use the reference compiler with your own test files. Instructions on accessing the reference compiler can be found in the “ACCESSING REFERENCE COMPILER” file on d2l.

EXAMPLE INPUT:

```
/* A program to perform Euclid's
   Algorithm to compute gcd. */

int gcd (int u, int v)
{ if (v == 0) return u ;
  else return gcd(v,u-u/v*v);
  /* u-u/v*v == u mod v */
}

void main(void)
{ int x; int y;
  x = input(); y = input();
  output(gcd(x,y));
}
```

EXAMPLE OUTPUT:

```
program
    func @ line 4
        type [int] @ line 4
        new_id [gcd] @ line 4
        parameters
            var_param @ line 4
                type [int] @ line 4
                new_id [u] @ line 4
            var_param @ line 4
                type [int] @ line 4
                new_id [v] @ line 4
        block
            declarations
            statements
                if_else @ line 5
                    == @ line 5
                        var [v] @ line 5
                        int [0] @ line 5
                    return @ line 5
                        var [u] @ line 5
                    return @ line 6
                        funccall @ line 6
                            var [gcd] @ line 6
                            arguments
                                var [v] @ line 6
                                - @ line 6
                                    var [u] @ line 6
                                    * @ line 6
                                        / @ line 6
                                            var [u] @ line 6
                                            var [v] @ line 6
                                        var [v] @ line 6
```

```
func @ line 10
    type [void] @ line 10
    new_id [main] @ line 10
    parameters
    block
        declarations
            var_decl @ line 11
                type [int] @ line 11
                new_id [x] @ line 11
            var_decl @ line 11
                type [int] @ line 11
                new_id [y] @ line 11
        statements
            exprstmt @ line 12
                = @ line 12
                    var [x] @ line 12
                    funccall @ line 12
                        var [input] @ line 12
                        arguments
            exprstmt @ line 12
                = @ line 12
                    var [y] @ line 12
                    funccall @ line 12
                        var [input] @ line 12
                        arguments
            exprstmt @ line 13
                funccall @ line 13
                    var [output] @ line 13
                    arguments
                        funccall @ line 13
                            var [gcd] @ line 13
                            arguments
                                var [x] @ line 13
                                var [y] @ line 13
```