THE UNIVERSITY OF ARIZONA®
DEPARTMENT OF COMPUTER SCIENCE

CSc 453: Spring 2025

Assignment 4 Milestone 1

Start: April 16, 2025
Due: 11:59 PM, April 23, 2025

1. General

This assignment involves writing a parser for the full C-- language. Changes in this language relative to the G2 subset targeted in Assignment 3 are shown highlighted here.

2. Required functionality

2.1. Grammar transformations

Before you begin writing code, you should transform the C-- grammar to remove ambiguity and make it recursive-descent-parsable, as discussed in class. At a minimum, this requires transformations to the grammar rules for arithmetic and Boolean expressions, as discussed in class.

2.2. Programming requirements

For this milestone, you should extend your compiler from Assignment 3 milestone 2 to parse the full C-- language. This assignment does not require you to do anything beyond parsing, i.e., it is not necessary to do semantic checking or AST construction (those will be done in the next milestone).

If the input is syntactically correct, your parser should exit silently with exit status 0; if the input contains any syntax errors, your parser should print out an error message on **stderr** (see below) and exit with exit status 1. You are not required to implement any error recovery.

Driver code and program invocation

The driver code is unchanged from Assignment 3. Your program will be invoked as follows:

```
./compile < input_file >& output_file
```

Makefile

You should submit a Makefile that provides (at least) the following targets:

- **make clean**:
  Deletes any object files (*.o) as well as the file 'compile'
- **make compile**:
  Compiles all the files from scratch and creates an executable file named 'compile'.

FIRST and FOLLOW sets

You can use the **gff** tool to compute FIRST and FOLLOW sets for the grammar. Documentation on the use of gff is available here. You can access a gff executable in either of two ways:

1. it is available as an executable on host **lectura.cs.arizona.edu** at /home/cs453/spring25/bin/gff ; or
2. you can grab the source code (available as a zip file here) and build the executable yourself.

Error messages

Your error messages should satisfy the following requirements:

- Each error message should contain the string **`ERROR`**. The grading code will use this to identify error messages.
- Each error message should contain a string **`LINE`** *nnn*, where *nnn* is the line number where the error was detected. The grading code will use this to check whether errors are being detected at the right place.

The case (upper/lower/mixed) of the strings ERROR and LINE mentioned above is not important.

Apart from the two requirements above, you are free to craft your error message as you see fit. You should try to give error messages that are helpful to the user—imagine you were using your compiler "for real" and consider whether your error messages would be helpful for the user to understand and locate the problem.

See the assignment spec for Assg 1 Milestone 2 for common problems in handling syntax errors.

2.2. Interface requirements

Your program should be written to interface appropriately with the driver code mentioned above. To this end, your code should provide the following:

```
int parse();  /* the parser function */
```

3. Files you need to submit

- Makefile that supports the functionality described above;
- driver.c -- the driver file for this assignment (unchanged from Assignment 3);
- any additional files implementing your compiler.