Advanced Programming languages
Programming Project 2

Project #2: A low level Code Generator.

This project involves building the code generator for the compiler. Also, extend your grammar to include simple read and write capabilities and variable declarations. This part will entail possibly using an intermediate representation such as a syntax tree and the symboltable to make static semantic checks and to generate low-level code. The static semantic checks should be to ensure that a variable has been declared before it is used and to ensure there are no redefinition errors.

You will extend the parser from project 1 to generate a register-based intermediate representation similar to the following:

Given a source code program such as this:

```
PROGRAM gcd;
VAR i, j: Integer;
BEGIN
  read (i);
  read (j);
  WHILE i <> j DO
    if i > j THEN
      i = i - j;
    ELSE
      j = j - i;
  ENDWHILE
  write (i);
END.
```

Your code generator will produce output in a text file similar to this:

```
-- generated by symbol table traversal
.data   -- begin static data section
.word i
.word j
.text - text of the code itself
main:
    a1 := &input
    call readint
    i := rv
    a1 := &input
    call readint
    j := rv
    goto L1
L2: r1 := i    -- body of while loop
    r2 := j
    r1 := r1 > r2
    if r1 goto L3
    r1 := j    -- else part of loop
    r2 := i
    r1 := r1 - r2
    j := r1
```

```
    goto L4
L3: r1 := i
    r2 := j
    r1 := r1 - r2
    i := r1
L4:
L1: r1 := i
    r2 := j
    r1 := r1 <> r2  -- while loop test
    if r1 goto L2
    a1 := &output
    r1 := i
    a2 := r1
    call writeint
    a1 := &output
    call writeln
    goto exit     -- return to OS
```

Program Input and Output
There will be no user interaction with the program. The program will be invoked 8 times from a shell script as demonstrated in class (see below in deliverables). Input will be via commandline parameter of the file to be parsed, specified in the script file. Output will be 4 files containing the intermediate representation code for the four legal programs. Additionally output will go to the console reporting either a successful parse or a syntax error and the line on which the error was detected.

Other Policies
All projects will be individual.

Deliverables:

You will submit all your work bundled up in WinZip (on PC) to the DropBox for the course on `elearning.uwf.edu`. Please make your name is very clear on the zipfile you upload and inside all files contained within the zipfile. Place all your files in a single directory named `<firstInitial><lastname>-p<projNumber>`. For instance, my submission would be `jcoffey-p2`. zip that folder into a .zip file and upload it to the appropriate dropbox.

So, submit a single zip file containing a single folder, with the following items in the folder:

- An updated version of the grammar of your language.
- Source code for the entire program
- A User's manual regarding exactly what this submission does and how to run it
- Design documents reflecting the complete program.
- Four examples each of both legal and illegal programs. The programs should be of increasing complexity, with the last legal and illegal program being the ***gcd program*** from above.
- A script that attempts to compile and run all legal and illegal programs as a batch, producing separate compiled files or appropriate error messages.
- A makefile to build your project.
