CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2010
Assignment #2 Due: September 7, 2010

Part 1: Reading and Exercises
Compilers - Principles, Techniques, & Tools, 2nd Edition, Sections 2.1, 2.2, 2.3, 2.6
Homework Exercises – problems 2.2.1, 2.2.2, 2.2.3, 2.3.1, 2.6.1 and laboratory assignment.

Introduction
In this assignment, you will learn the basics of using lex and yacc. You will download a self extracting file and you will create the basic components of a C compiler. The code starts as just the scanner and parser for an ANSI C compiler and in each assignment you will build upon the code base. Over each assignment you will be learning about compiler features by adding a symbol table, parser functions and code generator for the PIC architecture.

Part 2: Laboratory

Make a directory on your computer in your EECS337 directory under your Case ID and call it hw02.
mkdir EECS337/caseid/hw02/ ; where caseid is YOUR Case ID, enter all in lower case

Download a copy of the self extracting shell script "c-grammar" and hw02_test.sh files to the hw02 directory from http://blackboard.case.edu/ in the EECS337 homework assignment area. You are now ready to solve the laboratory assignment.

Part 3: Laboratory Assignment
From a console window, change directory to the hw02 directory (cd ~/EECS337/caseid/hw02/) and type ***./c-grammar***. The shell script is self extracting and will create the following files.
main.c
scan.l
gram.y
Makefile
README

Look at the README file and acknowledge the contribution from Jeff Lee for providing this sample code. This example code will be the foundation of the ANSI C compiler for this semester.

Edit the Makefile to change the mistake in creating the output file. Change the line with $(OBJ)/ansi_c to $(OBJ) –o ansi_c. Then add into the rm -f y.tab.h y.output *.o line the file names scan.c and gram.c. You are now ready to run ***make***. The files should build and create an executable file. To clean the build type ***make clean*** and the old objects will be removed. Then type make to build everything again.

There are times when you may want to look at the output files created by lex and yacc. To create the C version of the scanner or the parser type:
lex -t scan.l > scan.c ; to create the C version of the lex file
yacc -dv gram.y ; to create the C version of the yacc file (y.tab.c)
mv -f y.tab.c gram.c ; rename it gram.c

This version of the compiler uses the standard input (stdin) for reading inputs and writes data to the standard output (stdout) and the error messages to the standard error (stderr). In a future assignment the program will be changed to open the files directly.

Notice on the build of scan.l there are three warning messages.
"scan.l", line 54: warning, rule cannot be matched
"scan.l", line 56: warning, rule cannot be matched
"scan.l", line 58: warning, rule cannot be matched
Edit the scan.l file and notice the warning messages are due to duplicate patterns being in the lex file. Remove the three duplicate lines and build again to show the warnings are now gone.

Edit main.c and add a standard header to the top of the file and change the main() function to properly include the normal command line arguments.

```
/*****************************************************************************
*
* FILE:             main.c
*
* DESC:                     EECS 337 Assignment 2
*
* AUTHOR:                   caseid
*
* DATE:                     September 7, 2010
*
* EDIT HISTORY:
*
*****************************************************************************/
int         main( int argc, char *argv[])
```

Edit the gram.y file and add a standard header to the top of the file along with the missing declarations section. YYDEBUG features will be demonstrated in the next assignment.

```
/*****************************************************************************
*
* FILE:             gram.y
*
* DESC:                     EECS 337 Assignment 2
*
* AUTHOR:                   caseid
*
* DATE:                     September 7, 2010
*
* EDIT HISTORY:
*
*****************************************************************************/
%{
#include "y.tab.h"
#define YYDEBUG       1
%}
```

Edit the scan.l file and add a standard header to the top of the file.

```
/*****************************************************************************
*
* FILE:             scan.l
*
* DESC:                     EECS 337 Assignment 2
*
* AUTHOR:                   caseid
*
* DATE:                     September 7, 2010
*
* EDIT HISTORY:
*
*****************************************************************************/
```

No need to edit the Makefile and add a standard header because we will be using a new Makefile in the next assignment.

Build the files again and test the ANSI C compiler by using the current main.c program and the previous example programs from assignment 1.
./ansi_c < main.c
./ansi_c < ../hw01/Code_1_6_1.c
./ansi_c < ../hw01/Code_1_6_2.c
./ansi_c < ../hw01/Code_1_6_4.c
Notice the last example stops echo of the source output with a message:
```
define  a
        ^
syntax error
```
This is due to the preprocessor # operations in the C program which are normally processed by the C precompiler (cpp). Run the Linux C precomiler and generate a version of the .c file using the .cpp file extension.
cpp ../hw01/Code_1_6_4.c > Code_1_6_4.cpp

Now run the differences utility (diff) to see the changes between the two files.
diff Code_1_6_4.cpp ../hw01/Code_1_6_4.c

Notice the #define macro is changed to substitute the final expression but the C preprocessor also adds # characters into the source code which are not handled by the scanner. Add a pattern and action to the scanner to allow # to end of line (EOL) to be treated the same as a comment. Add a line just below the "/*" { comment(); } line in scan.l
```
"#"                             { pdefine(); }
```

And below the yywrap() function add the function
```
pdefine()
{
        char c;

        while ((c = input()) != '\n' && c != 0);
}
```

Build the compiler again and run the test command:
./ansi_c < Code_1_6_4.cpp

The file should now correctly echo to the screen, without syntax errors, showing the #define macro in the final expression format and the # lines skipped over as comments. Notice this output is not the same as the output generated without the C preprocessor but now the previous syntax error is missing.
./ansi_c < ../hw01/Code_1_6_4.c ; also no syntax error but #define macro is missing

When all your lab assignments have been completed execute the homework script file "./hw02_test.sh". You may need to chance the mode of the file to an executable using the command "chmod 755 *.sh". The script file shall remove the old object files, make all source programs and run the tests. To redirect the standard error (stderr) and standard output (stdout) to a file use the following command.
./hw02_test.sh &> hw02_test.out

Print out the hw02_test.out file, put your name, assignment number, date on it and turn it in with your homework exercises. Consider using the **a2ps** utility for printing your output to help minimize the number of sheets.

Your final directory structure should be as below (using your Case ID):
EECS337/caseid/hw02/ansi_c
EECS337/caseid/hw02/c-grammar
EECS337/caseid/hw02/Code_1_6_4.cpp
EECS337/caseid/hw02/gram.y
EECS337/caseid/hw02_test.out
EECS337/caseid/hw02_test.sh
EECS337/caseid/hw02/main.c
EECS337/caseid/hw02/Makefile
EECS337/caseid/hw02/README
EECS337/caseid/hw02/scan.l

Along with the previous assignment.
EECS337/caseid/hw01/Code_1_6_1
EECS337/caseid/hw01/Code_1_6_1.c
EECS337/caseid/hw01/Code_1_6_2
EECS337/caseid/hw01/Code_1_6_2.c
EECS337/caseid/hw01/Code_1_6_4
EECS337/caseid/hw01/Code_1_6_4.c
EECS337/caseid/hw01_test.out
EECS337/caseid/hw01_test.sh
EECS337/caseid/hw01/Makefile