CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2013
Assignment #2 30 points
Due: September 10, 2013

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 (a,b,c), 2.2.3 (only for a,b,c), draw a state transition diagram for all the states implemented in this laboratory assignment.

Introduction
In this assignment, you will look at the C pre-compiler operations (cpp) and you will expand the state machine to remove the macro statements `(# to EOL)`. This is the same operation as the slash-slash type comments to EOL. Then you will then expand the state machine to handle a number of special characters `()*+,;=[]{},` single digit integer constants (CONSTANT), identifiers (IDENTIFIER), the C types `void, char, int` (VOID, CHAR, INT) and the keyword `return` (RETURN). From these states you will call a token_print function with a token and character buffer of the current input text. You need to terminate the text string with a null (`0`). You run a test script and submit the output file as the laboratory part of the assignment. Also add to the state machine the ability to handle a string literal `"w:%d,\tx:%d,\ty:%d,\tz:%d\n"`.

Part 2: Laboratory Assignment
From a console window, make a directory on your computer in your EECS337 directory under your Case ID and call it hw02. Change into that directory and copy the previous Echo2 and Code example programs from assignment 1 into the current working directory. (Note: if you did not develop a working version of Echo2 or the Code programs then consider re-using the solutions code posted to blackboard under the assignment 01 area.)

mkdir EECS337/caseid/hw02/ ; where caseid is YOUR Case ID, enter all in lower case
cd EECS337/caseid/hw02/
cp ../hw01/Echo2.c .
cp ../hw01/Code_1_6_1.c .
cp ../hw01/Code_1_6_2.c .
cp ../hw01/Code_1_6_4.c .

Download a copy of the hw02_caseid.tar file to the hw02 directory from http://blackboard.case.edu/ in the EECS337 homework assignment area. You are now ready to solve the laboratory assignment. To untar the tar file type the command.
tar xvf hw02_caseid.tar

Your directory structure should now contain the extra files shown below:
hw02_test.sh
tokens.c
Makefile

Part 3: Test and Update Echo2.c Include
Build the Echo2 program and test the execution using the commands below. The output should show the C program source code with the C comments removed from the output.
make clean
make
./Echo2 < ../hw01/Code_1_6_1.c

Edit the Echo2.c file and just below the #include <stdio.h> line add the additional include files and the additional C source code file (tokens.c). This is one method to include additional source code into a program file. Normally the Makefile is updated to include the extra file with an include file (.h) to "extern" the information between the files. You will see this other method in a future assignment.

```
#include     <stdlib.h>
#include     <string.h>
#include     <time.h>
#include     "tokens.c"
```

Inside the main function just below the declaration statements add the additional source code to the file. Change the word (caseid) to your Case ID. This code will capture the current time and print the information to the console window. The time and ctime functions need the time.h include file to define these functions. The graders check this information in your test script output file so be sure to use your Case ID.

```
       time_t t;
/*
 *     print start of test time
 */
       time( &t);
       fprintf( stdout, "for caseid start time: %s", ctime( &t));
```

Part 4: Update Code_1_6_?.c Files and C PreCompiler

Edit each of the example files (Code_1_6_?.c) and delete the line with #include <stdio.h>. Without the standard input/output definitions the files will still build and run but you will see a warning message.

make Code_1_6_1
Code_1_6_1.c: In function 'main':
Code_1_6_1.c:36: warning: incompatible implicit declaration of built-in function 'printf''

Notice the Code_1_6_4.c file uses the #define macro as part of the program. The preprocessor # operations in a C program are normally processed by the C precompiler (cpp). Run the C precompiler and generate a version of the .c file using the .cpp file extension.
cpp 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 Code_1_6_4.c

Notice the #define macro is changed to substitute the final expression for `a` by `(x+1)`. Also notice the C preprocessor adds # characters into the source code which are not handled by the Echo2 program.

Part 5 A: Update Echo2.c State Machine for # to EOL

Add to the Echo2.c state machine code to remove the pattern # to end of line (EOL). This is the same as the method to handle C slash-slash comments to EOL. Build the Echo2 program again (make clean, make) and run the test command:
./Echo2 < Code_1_6_4.cpp

The file should now correctly echo to the screen, showing the #define macro in the final expression format and the # lines skipped over as comments. Add to the Echo2.c state machine code to process these special characters

Part 5 B: Add the Special Characters

Add to the Echo2.c state machine the ability to process the special characters `()*+,;=[]{},.` Instead of using putchar(c) to output the character use the print_token function instead which is part of the tokens.h file. Also remove the white spaces and any special characters (NAK, ACK) so the characters are removed from the output. For example:

```
                    switch( c)
                    {
                    case '(':
                    case ')':
                    case '*':
                    case '+':
                    case ',':
                    case ';':
                    case '=':
                    case '[':
                    case ']':
                    case '{':
                    case '}':
                        text[ 0] = c;
                        text[ 1] = 0;
                        print_token( c, text);
                        break;
                    case ' ':
                    case '\t':
                    case '\v':
                    case '\f':
                    default:          // skip all these characters
                        break;
//                      putchar( c); // put the character to the stdout
```

Part 5 C: Add Integer Constants

Add to the Echo2.c state machine code to process the one digit immediate values used in the programs. This time use the token CONSTANT when you call the print_token function.

```
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                    case '8':
                    case '9':
                        text[ 0] = c;
                        text[ 1] = 0;
                        print_token( CONSTANT, text);
                        break;
```

Part 5 D: Add code to handle keywords, identifiers and strings

Add code to handle the keywords: `void,int,char,return,` identifiers and strings. Notice that `main, printf` are identifiers and not keywords. Do not worry about handling embedded double quotes inside a string. You may want to use the isdigit and isalpha functions. When you are finished your output should match the following code for the Code_1_6_4.cpp file:

```
for caseid start time: Tue Sep  3 09:04:02 2013
token: INT text: int
token: IDENTIFIER text: x
token: '=' text: =
token: CONSTANT text: 2
token: ';' text: ;
token: VOID text: void
token: IDENTIFIER text: b
token: '(' text: (
token: ')' text: )
token: '{' text: {
token: IDENTIFIER text: x
token: '=' text: =
token: '(' text: (
token: IDENTIFIER text: x
token: '+' text: +
token: CONSTANT text: 1
token: ')' text: )
token: ';' text: ;
token: IDENTIFIER text: printf
token: '(' text: (
token: STRING_LITERAL text: "%d\n"
token: ',' text: ,
token: IDENTIFIER text: x
token: ')' text: )
token: ';' text: ;
token: '}' text: }
token: VOID text: void
token: IDENTIFIER text: c
token: '(' text: (
token: ')' text: )
token: '{' text: {
token: INT text: int
token: IDENTIFIER text: x
token: '=' text: =
token: CONSTANT text: 1
token: ';' text: ;
token: IDENTIFIER text: printf
token: '(' text: (
token: STRING_LITERAL text: "%d\n"
token: ',' text: ,
token: '(' text: (
token: IDENTIFIER text: x
token: '+' text: +
token: CONSTANT text: 1
token: ')' text: )
token: ')' text: )
token: ';' text: ;
token: '}' text: }
token: INT text: int
token: IDENTIFIER text: main
token: '(' text: (
token: ')' text: )
token: '{' text: {
token: IDENTIFIER text: b
token: '(' text: (
token: ')' text: )
token: ';' text: ;
token: IDENTIFIER text: c
token: '(' text: (
token: ')' text: )
token: ';' text: ;
token: RETURN text: return
token: CONSTANT text: 0
token: ';' text: ;
token: '}' text: }
```

Part 6: Laboratory Output Generation
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.txt

Print out the hw02_test.txt file, put your name, assignment number, date on it and turn it in with your homework exercises.

Your final directory structure should be as below (using your Case ID):

```
EECS337/caseid/hw02/Code_1_6_1.c
EECS337/caseid/hw02/Code_1_6_2.c
EECS337/caseid/hw02/Code_1_6_4.c
EECS337/caseid/hw02/Code_1_6_4.cpp
EECS337/caseid/hw02/Echo2
EECS337/caseid/hw02/Echo2.c
EECS337/caseid/hw02/Echo2.o
EECS337/caseid/hw02/Makefile
EECS337/caseid/hw02/hw02_caseid.tar
EECS337/caseid/hw02/hw02_test.sh
EECS337/caseid/hw02/hw02_test.txt
EECS337/caseid/hw02/tokens.c
```