CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2013
Assignment #1 30 points
Due: September 3, 2013

Part 1: Reading and Exercises

Compilers - Principles, Techniques, & Tools, 2nd Edition, Chapter 1.
Homework Exercises – problems 1.6.1, 1.6.2, 1.6.3, 1.6.4 and laboratory assignment.

Introduction

All homework assignments and grades are posted on blackboard at http://blackboard.case.edu. Download the cref.pdf file for a quick reference guide to programming in C. Notice comments are defined from slash-star (/*) to star-slash (*/) or from slash-slash (//) to the end of the line ("\n"). If you do not know how to program in C, you may also want to get the C Programming Language book by Brian W. Kernighan, Dennis M. Ritchie, publication Date: April 1, 1988 | ISBN-10: 0131103628 | ISBN-13: 978-0131103627 | Edition: 2nd.

In this assignment, you will learn the basics of C, makefile and Unix shell scripts. The Jennings Unix laboratory has a computer that is Linux based. You should be able to access the lab with your CaseID card, user name and password. Unix is the preferred operating system for this class. Mac computers are Unix based and **the assignments shall build and run under Unix**. You may need to install the user development software. Microsoft Windows machines can use Cygwin for simulating the Unix operations but have a number of inconsistencies. The automated shell scripts running the homework assignments will not work for both environments. The best solution is to use a Mac computer or a computer in the Jennings lab. Another method is to use your computer (Windows, Mac, Linux, etc) and ssh into the Jennings lab computer for the assignments.

The ability to quickly write and edit source code is key to designing compilers. You must be able to edit text files and maintain the correct file formats. Software editors under Microsoft Windows may change the end of line format (CR-LF) and break the ability of the software compiler tools (gcc, lex, yacc and makefile) to process your files. There were utilities for changing Windows (DOS) to Unix file (dos2unix?) formats but finding the right Unix/Linux based text editor will save you effort in the long run. You should consider using textedit, emacs, vi, gedit or some other Unix based text editor. You should get good at using your text editor and if possible learn to do keyboard marcos. Macros provide you with the ability to quickly generate new source code based on previous work.

Homework assignments are handed out on Tuesday and due the following week. Turn in your book exercises on paper and print out the results of your laboratory assignments. There normally is a Unix script file provided for generating the output results.

Introduction

The assignments are written using the C programming language with the flex/lex and bison/yacc software tools. The output of these tools is C source code that is compiled and linked with a C main program. The call into the lex/yacc parser will be covered in the next assignment. A C main program has two arguments for passing command line arguments. The first variable (int argc) is an integer type and defines the number of string items in the second variable (char *argv[]). The second variable is an array of pointers to the character type. In C strings are terminated with a null character (0). Just about every C main program passes the name of the program itself as the first parameter. In the Echo.c program the command line parameters are printed to the standard output (stdout). The main program is declared as an integer and must return an integer value to the operating system on exit. A zero (0) normally indicates success and anything else is an error condition. The Echo.c program is shown on the next page.

The main program is the entry point called by the operating system. It passes the command line arguments to the function for processing. In the C programming language the only two things you can do is declare memory (variables) and writing logic (statements) as part of a function. If the variables are declared outside a function (global/static) the scope is from the declaration to the end of the file. If the variable is declared inside a function (local/stack) the scope is from the declaration to the end of the block (closing }). The next part of the program reads each character from the standard input (stdin) and writes the character to the standard output (stdout). The error messages normally go to the standard error (fprintf(stderr, “error\n”);).

```
/*
 *     Echo.c main program
 */
int    main( int argc, char *argv[])
{
/*
 *     declare variables
 */
       int i;
       int c;
/*
 *     read from the command line and write out parameters
 */
       for( i = 0; i < argc; i++)
            printf( "argv[ argc: %d]: %s\n", i, argv[ i]);
/*
 *     read from stdin and write to stdout
 */
       while(( c = getchar()) != EOF)    // while input character from stdin
does not equal end of file (-1)
              putchar( c); // put the character to the stdout
       return 0;
}
```

Part 2: Laboratory

Log onto the UNIX operating system and start a terminal window. Type “pwd” to show your current working directory and use “ls –la” to display the files in your directory. Use “cd” to change directories and if you type “cd” without a directory path then you will be returned to your home directory (~/). The tab key will auto-complete your commands. Create the following directory structures in your home directory. Be sure to use your Case ID to create the assignment directory. For example my Case ID is dxo4.

mkdir EECS337/	; create a directory structure with EECS337 at the top
mkdir EECS337/caseid/	; where caseid is YOUR Case ID, entered in lower case
mkdir EECS337/caseid/hw01/	; each home work assignment will sequence in number
cd EECS337/caseid/hw01/	; change to the new assignment 01 directory

Download a copy of the hw01_caseid.tar file to your hw01 directory from blackboard 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 hw01_caseid.tar

Your directory structure should now contain the files below (using your Case ID):
EECS337/caseid/hw01/Code_1_6_1.c
EECS337/caseid/hw01/Code_1_6_2.c
EECS337/caseid/hw01/Code_1_6_4.c
EECS337/caseid/hw01/Echo.c
EECS337/caseid/hw01/hw01_test.sh
EECS337/caseid/hw01/Makefile

Part 3: Laboratory C Assignment
As you edit the source code files change the headers to use your Case ID instead of the word “caseid”.

1) From exercise 1.6.1, edit file Code_1_6_1.c and enter the source code example from the book into the body of the main program and save the file. Then type “make Code_1_6_1” to compile the program into an executable format. Compare the output values with the results obtained from your book problems and fix any errors or warnings. To execute the program you need to have the current directory “.” in your path. Type “./” for the current working directory so type “./Code_1_6_1” to execute the program. The standard output will be displayed to the console window.

2) From exercise 1.6.2, enter the code example into the body of file Code_1_6_2.c and then type “make Code_1_6_2” to compile the program and “./Code_1_6_2” to run the program. Check your output against the book solution.

3) From exercise 1.6.4, enter the code example into the body of file Code_1_6_4.c and then type “make Code_1_6_4” to compile the program and “./Code_1_6_4” to run the program. This example may have an error based on your version of the book. Find and correct the compiler error. You will get a warning message that main does not return an integer (int). Notice the main routine is declared a void but is required to return an integer. Fix the program to remove the warning message and build and run the program again. Check your output against the book solution.

Part 4: Laboratory Echo Assignment
1) Build the Echo.c file by typing the command “make Echo”. A C program reads from the standard input (stdin) and writes to the standard output (stdout). Error messages are directed to the standard error (stderr). Input includes reading from the console or from a redirected file. To echo the Echo.c file using redirection type the command “./Echo < Echo.c”. The Echo.c file should be echoed back to the screen, character by character. If you do not supply an input file using the redirect (<) operator then the program will read from the console input. In this case to terminate the program type a Control^C and the program will be interrupted and halted. If you type Control^Z then the program is suspended and can be restarted in the background by typing the background (bg) command. Then use ps to find the thread ID and kill -9 thread_id to stop the background process.

You can compare the two versions by using the “diff” command. First capture the output by typing “./Echo < Echo.c > Echo.txt”. This command uses two redirects, use < for the input and > for the output. If you use >> then the output is appended to the end of the output file. Then type “diff Echo.c Echo.txt”. This will show you the difference between the two files. The files should be the same except for the command line printout: `argv[ argc: 0]: ./Echo`

Part 5: Laboratory Echo2 Assignment
Create a copy of your Echo.c file and call it Echo2.c (cp Echo.c Echo2.c).

Part 5A: Command Line parameters
Edit the Echo2.c code to process the command line arguments. We want to use one-hot state assignment to represent flags and the following macros are useful for implementing flags in C. First we define the flag value (0x00000001) and then three macros to test, set and clear the flag. The flags variable must also be declared inside the C program. Next flag is at 0x00000002.

```
/*
 *     define flags: [+/-]echo
 */
#define       FLAGS_ECHO          0x00000001
#define       IS_FLAGS_ECHO(a)    (a & FLAGS_ECHO)
#define       SET_FLAGS_ECHO(a)   (a |= FLAGS_ECHO)
#define       CLR_FLAGS_ECHO(a)   (a &= ~FLAGS_ECHO
```

Now replace the command line source code so we can loop over the command line arguments to check for our flags instead. From the command line, we want to set (+echo) or reset (-echo) the flags.

```
int flags = 0;

for( i = 0; i < argc; i++)
{
     if( !strcmp( argv[ i], "+echo"))
             SET_FLAGS_ECHO( flags);
     else if( !strcmp( argv[ i], "-echo"))
             CLR_FLAGS_ECHO( flags);
}
```

When using a real time operating system (RTOS) we normally use tasks/threads, semaphores, queues and flags. Flags are normally used by one (1) thread to signal many threads (N) of a state or condition. For example to quit, exit or power-down the flags variable is cleared `(flags = 0;)` to signal all the other threads to clean up and exit.

Part 5B: Remove // Comments to EOL
In Echo2.c, implement a state machine to remove the comments that go from // to end of line. There are many ways to code this operation. Do not worry about comments inside string literals ("\\") or character constants ('\'). You will need to look for the '/' and '/' characters and not output the characters until you see the end of line ('\n'). **When you code the state machine you should only use one call to gethchar()** and switch on the state to decide your action. When you run the program again (./Echo2 < Echo.c) the comment lines from the // to end of line should now be missing from the output. Use the same method of testing found in Part 4 for testing the original Echo program to see if these comments are correctly removed.

Part 5C: Remove /* Comments */
In Echo2.c, now update your state machine to remove the comments that go from /* to */. When you run the program again the comment lines from the /* to */ should now be missing from the output. We will cover the removal of comments from C source programs using lex and regular expressions in a future assignment. You will be reusing some of this code so be sure to save your work.

Part 5D: Add the +echo –echo Operation
In Echo2.c, now update your state machine to use the +echo flag to `(if( ! IS_FLAGS_ECHO( flags)))` not remove the comments when the flag is entered on the command line. When you run the program again `(./Echo2 +echo < Echo2.c > Echo2.txt)` the program should now be complete and have no differences. Notice you can now turn the flag on and off between programs on the command line `(./Echo2 +echo < Echo.c –echo < Echo2.c)`.

Part 6: Laboratory Output Generation
When your C lab assignments have been completed then execute the homework script file "./hw01_test.sh". You may need to change 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 example programs and run the tests.** To redirect the standard error (stderr) and standard output (stdout) to a single file use the command "./hw01_test.sh &> hw01_test.txt".

Print out the **hw01_test.txt** results file, put your name, assignment number, date on it and turn it in with your book homework exercises. Notice that the test script runs your Code*.c files, the Echo and Echo2 program. Your source code for Echo2.c is part of the printout.

Your directory structure should now contain the files below (using your Case ID):

```
EECS337/caseid/hw01/Code_1_6_1
EECS337/caseid/hw01/Code_1_6_1.c
EECS337/caseid/hw01/Code_1_6_1.o
EECS337/caseid/hw01/Code_1_6_2
EECS337/caseid/hw01/Code_1_6_2.c
EECS337/caseid/hw01/Code_1_6_2.o
EECS337/caseid/hw01/Code_1_6_4
EECS337/caseid/hw01/Code_1_6_4.c
EECS337/caseid/hw01/Code_1_6_4.o
EECS337/caseid/hw01/Echo
EECS337/caseid/hw01/Echo.c
EECS337/caseid/hw01/Echo.o
EECS337/caseid/hw01/Echo.txt
EECS337/caseid/hw01/Echo2
EECS337/caseid/hw01/Echo2.c
EECS337/caseid/hw01/Echo2.o
EECS337/caseid/hw01/Makefile
EECS337/caseid/hw01/hw01_caseid.tar
EECS337/caseid/hw01/hw01_test.sh
EECS337/caseid/hw01/hw01_test.txt
```