Homework Assignment

HW4: Multi-module programs and MAKE

Assigned: 31 October 2019 **Due:** 14 November 2019 by class time

The objective of this assignment is to get you familiar with multi-module programs and MAKE. Make a subdirectory "hw4" in your cs240 directory for this assignment and copy the files from /courses/cs240/f19/xiaohui/GROUP/hw4.

1. Compile and link multiple modules together into a single executable file named "calcit".

1.1 Read the provided programs main.c getop.c stack.c getch.c and calc.h.

1.2 Use gcc –m32 main.c getop.c stack.c getch.c –o calcit to generate the executable file calcit. Fix compiler errors. Run and test it. Then, draw a dependency tree for these files and write a makefile for the calcit, **with all the commands included**. Some hints are provided in Lecture 10. Try "make" and regenerate the executable file calcit. Run and test it.

1.2 MODIFY the calcit source programs to make it operate on integers instead of floating point variables and support BIT OPERATIONS (^, &, |, ~) and %. In order to do this, change the variable types and returns of functions such as "pop" so that type "int" is used in calculations. Then, add the operations ^, |, &, ~, %, to support these binary operations. For example, if the operation is ^, |, &, then it is programmed as following. You may want to observe the result and guess the meaning of these operators. I will introduce these operators later in class.

XOR ^: push(pop()^ pop()),
OR |: push(pop()| pop()),
AND &: push(pop()& pop())
NOT ~: ~pop()

Note that because we no longer allow double type returns, we can deal only with integers and must use the atoi() function instead of atof(). atoi() is a C library function, and you are not required to code it.

Use the infix string ~(((292 %16) +(292/16)*16) ^ 292) as the script test case. Your program need not understand infix arithmetic though -- you must convert the test expression given above to reverse polish before feeding it to your program (just create a file named "calc.in" with the proper sequence and use it as stdin). A convertor can be found here http://www.mathblog.dk/tools/infix-postfix-converter/.

For this problem, turn in a hardcopy of a "typescript1" file as usual.

2. Experiment with make and touch.

Prepare another file "makefile2" that is a copy of makefile except remove the command lists for the .o rules (we want to prove that they are superfluous). Leave the command list in the rule for calcit. In this case, make will try to build the .o files according to some built-in implicit rules which includes $(CC) and $(CFLAGS) (see https://www.gnu.org/software/make/manual/html_node/Implicit-Rules.html for some details).

Use "script typescript2" to create a file that shows you doing the following:

a. "make clean –f makefile2" to clear out old *.o files
b. "make –f makefile2" to do a full rebuild of calcit.
c. modify getop.c to add in a comment line with your name.
d. "make –f makefile2" to see just getop.c recompiled, plus the loadable calcit.
e. "touch calc.h" to simulate an edit to calc.h.
f. "make –f makefile2" to see two compiles based on dependencies on calc.h, plus the loadable calcit.
g. Use the diff -w command to show how the two makefiles are different.
h. Use exit to save the typescript2 file.

For this problem, turn in a hardcopy of a “typescript2” file as usual.