ECOR 1606 Lab #2

Convert the following algorithms into C--. Run the programs using the test values from Lab #1 and confirm that the same results are obtained (note that you can "view" your Lab #1 file using the submit program). You should also experiment with single stepping and breakpoints (see notes after the algorithms). Note that you may find the C-- Mini-User Guide ("*cmmguide.pdf*") helpful.

Submit your two files "*lab21.cmm*" and "*lab22.cmm*" as lab #2 for your lab section to prove your attendance at the lab. **Both files must be submitted at the same time.** As long as they are in the same folder, when you select "*lab21.cmm*" the submit program will find them both. If you click "submit" and there are still files missing, you will be asked to add the missing file(s).

1. Finding the square root of a number:

create three variables: *number*, *x0*, and *x1*

output "Enter a number: "
read a value into *number*
set *x0* to 1
set x1 to 0.5 * (1 + *number*)
while (the absolute value of *x1* - *x0* is more than 0.00001 [**see note below**]) **do**
    set *x0* to *x1*
    set *x1* to 0.5 * (*x1* + (*number* / *x1*))
endwhile
output "The square root of ", the value of *number*, " is ", the value of *x1*, and an endl

Note: to write "the absolute value of *x1* - *x0* is more than 0.00001" in C-- use:

fabs(x1 - x0) > 0.00001

Single stepping is both a useful debugging tool and a good way for beginners to gain a better understanding of how computer programs are executed. The basic idea is that the program is executed one step at a time. This allows the programmer to follow the process and observe every change in variable values.

To single step through a program, select "step through" from the "execute" menu. At each step of the execution process an explanatory message is displayed and the user is given the choice of "step" (go on to next step), "run" (begin executing continuously), and "stop" (stop executing altogether"). Except when program input is required, it is only necessary to keep on selecting "step" in order to single step through an entire program.

Single stepping through an entire program rapidly becomes unattractive as programs become larger. Usually single stepping is restricted to those parts of a program where problems are suspected. This particular program, however, is sufficiently short that single stepping through all of it is quite practical. Do this. When input is required enter "3" (don't forget to hit ENTER) in the input/output window. Once this has been done the input/output window should be minimized or just moved out of the way.

2. Converting decimal (base 10) numbers to binary (base 2):

Algorithm:

create two variables: *number* and *divider*

output "Enter a number: "
read a value into *number*

Output "The binary equivalent of ", value of *number*, and " is "

set *divider* to 1
while (2 * *divider* <= *number*) **do**
    set *divider* to *divider* * 2
endwhile

while (*divider* >= 1) **do**
    if (*number* >= *divider*) **then**
        Output "1"
        set *number* to *number* - *divider*
    else
        Output "0"
    endif
    set *divider* to *divider* / 2
endwhile
Output an endl

"Breakpoints" allow the programmer to specify that execution should be temporarily halted when a selected statement is about to be executed. This makes it possible to examine the values of variables before execution is resumed (either in single step or continuous mode, as desired). Breakpoints are a valuable debugging tool. If a loop is not working properly, for example, a breakpoint at the beginning of a loop makes it possible to quickly see how variables are changing from one loop iteration to the next. In the C-- environment, breakpoints are set (and cleared) by left clicking on the little "traffic lights" that appear at the beginning of every statement. Set a breakpoint at "if (number > divider)" and execute the program. You will find that execution stops every time the statement is reached. To simply continue from the breakpoint, select "run". You should also try single stepping for a few instructions (select "step") before resuming continuous execution. This is a common debugging technique. A breakpoint is placed at the beginning of a suspected problem area and the statements of interest are then executed in single step mode.