CISC 810: Fundamentals of Computational Science, Assignment 3

Set September Oct 18th due Oct Nov 6th (few extra days on this one)

Q1. Automated compilation & makefiles. Download the `testdir1.tar` file from the class website and install it in a directory of your own choosing. Untar the file using `tar -xvf testdir1.tar`. Once finished label the Makefiles as `Makefile.[abcde]`, and create a tar tar archive (`tar -c` *yourname*`_makefiles.tar Makefile.*`) and send it to me by email (thacker@astro.queensu.ca). You should include hard copies in your assignment as well.
(a) Write a makefile to create the executable `mytest`. Provide a rule for each function (*i.e.* `mytest.o, test1.o, test2.o, test3.o`). The c compiler will create an intermediate object file by using the `-c` flag. Note that to make the final executable you will need to link the math library.
(b) Create a new makefile that adds an additional target `mytest-debug` which compiles a version of the code with the compilers debugging flag (`-g`) turned on. Note that this will mean new rules for each object file. Also add a phony target `clean` to remove intermediate object files and executables.
(c) These makefiles are quite inefficient as written and there is a lot of repetition. Create a new makefile that uses macros (variables) to reduce repetition. For example, you can call the compiler flags in both the standard version of the code `CFLAGS`, and the flags for the debug version `CDFLAGS`.
(d) Create another makefile that uses automatic variables and pattern rules for the `mytest` and `mytest-debug` targets and any associated prerequisites. For this particular case you can include one header file per object in each rule, even though technically the functions may have a dependency on more than one.
(e) Lastly, create the simplest version of this makefile possible using the `.c.o` pattern rule and automatic variables. Do not include the debug target in this case.

Q2 Simple programming. We can evaluate an approximation to $\cos(x)$ using the Taylor series,

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + ...$$

by increasing the number of terms we can ensure the result is accurate to a prescribed tolerance value, although the precise number of terms depends on the value of x. In practice there are many numerical issues with this approach and optimized math libraries use a different techniques (you can find better polynomials in a given range, or use a table look-up method to interpolate). Write a program (in C or FORTRAN) that for a given terminal input x, evaluates the above series and prints its value, but ensure that the minimum number of instructions are used (HINT: you don't have to use the formula as written and for a polynomial of degree n you can evaluate it using $n-1$ additions and $n-1$ multiplications.) You may assume that x is floating point number. Submit your completed code to me via email. Marks will be assigned for (1) having the correct algorithm (2) sensible code structure & variable name choice (3) adequate but not excessive commenting. Send me a copy of your solution via email.

Q3. Debugging & pointers. Download the C code for assignment 3 from the class website and create an executable. In this question you are going to use the `script` command to save your terminal input so that I can see you are performing the steps correctly. If you use cygwin you can download the `script` command from the class website, you can either place it in your own bin directory and change your path to find it, or you can place it in the directory that you are going to work in. If you have any problems setting up the `script` program email me.

(a) This first program (`prog1.c`) calculates an average and needs input from the terminal which in this case is best provided by a list of integers (type four numbers, each on its own line and then save it in a file named `list`). Compile the program and create an executable. Now create a new shell at the command prompt and then type `script` *yourname.script_a*. This will save your output to a file name *yourname.script_a*. Start the debugger gdb by loading in the executable. Set breakpoints at line 6 and 11 and then begin running the program in the debugger using input from `list`. Once you get to line 6 turn on the display of values for `num, val` and `sum` by using the `display` command. Step or continue through until you get to second breakpoint - how do the values look is everything working OK? What are the values of `num, val` and `sum` at this point. What is the main bug in the program? Send me a copy of the scriptfile you have generated and the input list of numbers you used.
(b) Compile `prog2.c` and debug the code within gdb. Each time you run the code you will need to give it three numbers. Debug the segmentation fault by stepping through the execution and again create a scriptfile of your approach (with a name *yourname.script_b*). Why is there a segmentation fault and why doesn't the sum command give the expected result of x+y+z after you fix the segmentation fault? Again, send me a copy of the script file.
(c) Write a program that uses pointers to set each value of an array to zero. You need only provide a printout of the code in this question.

Q4 Programming & optimization. Write a code to do a matrix multiply of floating-point values (in C/FORTRAN, you can use pointers in C if you wish).
(a) As well as the multiply routine itself, create a routine which inputs values into the matrices (the precise values are up to you, but every value in the input matrices should be filled so you cannot make assumptions about sparseness). Also, ensure program correctness. Comment the code sensibly.
(b) Once you've prepared the code and tested it, now optimize it using the compiler and also see if there are any hand coded optimizations you can make. Bonus marks are available on this question. You can time how long the program takes to complete using the Unix `time` *yourprogram* command. Note that you will probably need to choose a fairly large matrix to make the code take a few seconds to complete. Tabulate the performance improvements you get as you make changes to the code and make a graph of the results. Send me a copy of your code by email and also provide hard copy in the assignment. Note there are bonus marks available on this question if you make optimizations we have not covered.