CS 403: Lab 2: Comparing C and FORTRAN

Getting Started

1. Boot into Linux and log in.

2. Open Netscape (from the "K" menu at the lower left). Navigate to the course website[1] and download the file "CFtest.tar" to your home directory. Tar stands for "Tape ARchive" and is a standard Unix format for bundling several files as single file (similar to Zip file on Windows). To unpack the file, open a terminal and type tar xvf CFtest.tar. You should now have two directories: "Cbasic" and "Fbasic" which contain the sample problem implemented in C and FORTRAN.

C

1. Cd into Cbasic. You should have three files with C code (main.c, io.c, and system.c), two header files (io.h and system .h), and a single command file basic.cmnd. The program described by these files will read four parameters for the model problem (m, L, dt, T) from the .cmnd file and will create a length m array called C where it will place the location of the grid points. The program will then save C in a file called "C.txt." Use gcc to build the program from the .c files. I'll assume you've named it "cbasic."

2. Typing cbasic basic.cmnd will start the program and tell it to read from basic.cmnd. You can look at basic.cmnd by opening it in an editor or by typing more basic.cmnd on the command line (more is a UNIX command that displays text files–if the file has many lines, you may need to press the spacebar to page through the file). What happens if you just type cbasic?

3. Open basic.cmnd in an editor. Try changing some of the values and then running cbasic (Hint: use more to look at C.txt).

4. Let's look at how the program is structured. Type nedit *.c *.h & to view all of the files in an editor. main.c contains the main subroutine (where the program starts). All of the other subroutines are found in

[1] www.cs.cornell.edu/Courses/cs403/2002sp

the other .c files: io.c contains the routines for reading the .cmnd file and for displaying the parameters and saving C.txt; system.c contains a routine for generating an error message and another for dynamically allocating arrays. The .h files are header files, and they contain the prototypes for the subroutines in the corresponding .c files. C requires you to define a prototype for all subroutines. The prototype describes how the routine will be called (its input and output). In order to use the routines in io.c and system.c, the main file must import the prototypes. This is the function of the header files, and they are imported using the "#include" directive. Look at the files. What is the relationship betweeen system.c and system.h? Which files include system.h? io.h?

5. We're now going to check out two features in C. C was really designed to build command line tools, and consequently, there is a well-established mechanism for getting command line arguments into a C program. Look in main.c. How does the program decide whether or not to ask the user for a command file?

6. The size of a C array can be set at compile time (static allocation) or at run time (dynamic allocation). Compare how the character array commfilename is created vs. the double array C. Which allocation method is easier to program?

FORTRAN

1. Cd into Fbasic (cd ../Fbasic–".." indicates the directory one level above the working directory). Copy basic.cmnd into this directory (cp ../Cbasic/*.cmnd .), then list the files. The .f files contain the source code for a FORTRAN implementation of cbasic. Compared with the C version, what's missing?

2. Build fbasic. by typing g77 *.f -ofbasic (g77 is the GNU FORTRAN 77 compiler). Run the program by typing fbasic. What happens if you type fbasic basic.cmnd?

3. Open the .f files in an editor. Is there a main subroutine?

4. Look for the array C. What determines the size of C? Edit basic.cmnd so that m (the first line) is greater than MMAX in main.f, then run the

program. What happens? Fix the program so that it will work with your new m.

5. In question 1 above, you should've realized that FORTRAN has no concept of prototypes or header files. We'll now see an example of why prototypes are good. Remove m from the call to *PrintArray*. Rebuild fbasic and run it. What happens? If you'd done the same thing in main.c, would gcc have caught the error? Try it.