CS 404: Lab 3: Calling FORTRAN from C

Getting Started

1. Boot into Linux. To do this, click on "shutdown" and then select "restart." The computer will begin restarting. You will eventually have the option to select either Win2K or Linux (use up and down arrows to switch, press enter to select).

2. Open a web browser and go to the course website[1]. Download RAD1Ditpack.tar.

3. To unpack the tar-file, type `tar xvf RAD1Ditpack.tar` in a terminal window. Cd into the directory "RAD1Ditpack."

Calling FORTRAN from C

1. RAD1Ditpack contains the source code for simulating 1-D reaction-advection-diffusion problems with periodic boundary conditions. This is essentially the model-problem from CS403 with some slight modifications[2]. As originally constructed, RAD1D solves for diffusion using an implicit scheme. Computationally, this means that we have to solve a system of linear equations:

$$A * C = RHS$$

for each time step. The matrix A has several important properties, chief among them, that it is sparse (most entries are zeros) and symmetric (the lower triangular portion is a mirror image of the upper triangular portion, or mathematically, $A(i, j) = A(j, i)$). Because A is sparse, we can save a tremendous amount of memory by only tracking the locations where $A \neq 0$ rather than storing all of the zeros. If n is the number of rows in A, then storing A as a full 2D array would take $8n^2$ bytes. Since there are only $3n$ nonzero elements, we can reduce the memory requirements to $24n$ bytes plus an additional $16n$ bytes for the integer arrays needed to remember where A is nonzero. This is what the original RAD1D did. Because A is symmetric, we can further

[1] www.cs.cornell.edu/Course/cs404/2002sp

[2] Check out www.cs.cornell.edu/Courses/cs403/Lecture08/note.html for more info on RAD1D

reduce the storage requirements by only storing the upper triangular portion of A. I've modified main.c and the routine BuildA in linalg.c to only store the nonzeros in the upper triangular portion. Take a look if you're interested.

2. Unfortunately, the routine to solve $Ac = b$ in the original version (pcgm in linalg.c) requires every nonzero entry in A to be represented. Perhaps more unfortunately, to use the routines in LAPACK with this problem, we would have to represent the entire upper triangular portion of A–both zeros and nonzeros. In order to take advantage of A's sparsity and symmetry, we need a new solver. Going through the GAMS tree, I found the package ITPACK which provides a variety of routines for solving sparse linear systems. The only problem: ITPACK is in FORTRAN, and RAD1D is in C, so we need to figure out how to call ITPACK from our C code. After reading the ITPACK documentation (a PDF version is on the course web site), I figured out that we need to call two functions: DFAULT which places the default parameters into two arrays IPARM and RPARM (integer and real parameters, respectively), and JCG which solves our system of equations using the "conjugate gradient method with a Jacobi preconditioner." I have replaced the call to the original pcgm routine (in SolveA in linalg.c) with calls to ITPACK, and I have also allocated the workspace arrays required by JCG (WKSP and IWKSP) and set the length of these arrays (NW). Your job is to get the calls to DFAULT and JCG to work. However, rather than talk you through it step-by-step as I usually do, I will only give you a checklist of things to do:

	Key	Description
1.	Call-by-value	Make sure all variables are called by reference (are either arrays or pointers) rather than by value. Remember, the reference to a C variable is obtained with the & operator: foo–value of variable foo, &foo–memory address (reference) of variable foo
2.	Case	Most FORTRAN compilers (including `g77` make all variables and function calls lower case; if they don't you can usually force everything lower case with a compiler option (usually -f). For everything to link correctly, you must call FORTRAN routines using a lower case name.
3.	Underscore	Most FORTRAN compilers append an underscore ("_") to each subroutine name. For everything to link correctly, you may need to add the underscore the subroutine call in the C program.
4.	Prototype	Strict ANSI C requires you to provide a prototype for every subroutine. The prototype gives the return type of the subroutine (void, if no values are explicitly returned), subroutine name, and the types of each variable. You need to provide prototypes for both DFAULT and JCG. There's a special "FORTRAN prototype" section at the top of linalg.c

3. When you think you've made the changes in linalg.c, you're ready to build the executable. I've modified the Makefile to compile the ITPACK routines in dsrc2c.f to dsrc2c.o. Typing "make" should create all of the objects and attempt to link them using `f77`. Why do we link with f77 rather than gcc? If you made the right changes in SolveA, `rad1d` should build correctly. You can run the problem by typing `rad1d basic.cmnd` (basic.cmnd is a "command" file that specifies the parameters for the simulation–see the online description for more details).