**Introduction to Using OSCER’s Linux Cluster Supercomputer**

This exercise will help you learn to use Sooner, the Linux cluster supercomputer administered by the OU Supercomputing Center for Education & Research (OSCER), a division of the University of Oklahoma (OU) Information Technology (IT).

Actions and commands that you should perform or enter are in the **computer boldface** font.

**Enter**

After each Unix command you type at the Unix prompt (explained below), press the key.

An account has been set up for you. Your user name is denoted here as yourusername, but may actually be of the form sipe####, where ### is a specific user ID number. Or, if you’re a permanent OSCER user, your user name may be tied to your name; for example, hneeman (Henry Neeman).

If you have any difficulty with this exercise, then please send e-mail to:

support@oscer.ou.edu

The steps for this exercise are listed on the following pages.

There are actually two versions of this exercise: a version in C, and a version in Fortran90. Where possible, descriptions of both will be given, but in some cases, only the C description will be given, with the Fortran90 version assumed.

**NOTE:** We don’t use Fortran77, because **Fortran77 is pure brain poison**, but Fortran90 is a very nice lan­guage. Also, we don’t use C++; instead, we assume that everyone who is comfortable in C++ will be served well enough by C.

**I. LOG IN**

1. From the PC of your choice (Windows, MacOS, Linux or whatever), bring up your web browser (Internet Explorer, Firefox, Opera, Safari, Chrome or whatever) and go to:

[**http://www.oscer.ou.edu/ssh\_install.php**](http://www.oscer.ou.edu/ssh_install.php)

**NOTICE** the underscore in the URL, between ssh and install.

Following the instructions on that page, log in to:

**sooner.oscer.ou.edu**

1. If you cannot log in to sooner.oscer.ou.edu, then try logging in to one of the following:

sooner1.oscer.ou.edu

sooner2.oscer.ou.edu

It turns out that sooner.oscer.ou.edu is an *alias* for these other computers; that is, when you log in to sooner.oscer.ou.edu, you’ll actually get logged into one of these.

1. Once you log in, you’ll get some text, and then a *Unix prompt* – probably but not necessar­ily a percent sign – with the cursor after it, like so:

%

There may be some information before the prompt character, such as the name of the com­puter that you’ve logged in to (which may be different from sooner.oscer.ou.edu), your user name, and so on. For purposes of these materials, we’ll generally use the percent sign % to indicate the Unix prompt.

1. Check the lines of text immediately above the Unix prompt.

If there are lines of text that read something like:

No directory /home/yourusername! Logging in with home = "/".

then you should log out immediately by entering

% **exit**

and then log back in. If you repeatedly have this problem, then please send e-mail to:

support@oscer.ou.edu

1. Check to be sure that you’re in your *home directory* (a *directory* in Unix is like a folder in Windows, and your *home directory* in Unix is like your desktop in Windows):

% **pwd**

/home/yourusername

This command is short for “Print working directory;” that is, “print the full name of the directory that I’m currently in.” The output is the name of the directory that you’re currently in.

If your current working directory is just a slash (which means the *root directory*, which is like C:\ in Windows), rather than something like /home/yourusername, then you should log out immediately (as above), then log back in.

If you repeatedly have this problem, then please send e-mail to:

support@oscer.ou.edu

**II. SET UP (FIRST TIME LOGGING IN ONLY)**

1. You should have been prompted to change your password when you first logged in.

If not, then please **IMMEDIATELY** do the following:

* 1. From the PC of your choice (Windows, MacOS, Linux or whatever), bring up your web browser (Internet Explorer, Firefox, Opera, Safari, Chrome or whatever) and go to:

[**http://www.oscer.ou.edu/password\_change.php**](http://www.oscer.ou.edu/password_change.php)

**(NOTICE** the underscore in the URL, between password and change.)

* 1. Follow the instructions on that page to change your OSCER password. Please note that this password change will affect only your accounts on OSCER computers, not anywhere else (including not affecting any other OU IT accounts if you’re at OU).

You **WON’T** have to do this for future logins.

1. You should have been prompted to set your e-mail address when you first logged in.

If not, then please **IMMEDIATELY** enter:

% **change\_email youremailaddress@yourinstitution.edu**

but replacing **youremailaddress@yourinstitution.edu** with your e-mail address.

You **WON’T** have to do this for future logins.

1. Please **IMMEDIATELY** enter the following:

% **echo youremailaddress@yourinstitution.edu > ~/.forward**

You **WON’T** have to do this for future logins.

**NOTES**

* + 1. You should replace **youremailaddress@yourinstitution.edu** with your e-mail address.
		2. After your e-mail address comes a blank space, then a greater-than symbol, then a blank space, then tilde slash period **forward**, with no spaces between them.
1. At the Unix prompt, enter exactly the bold text below, excluding the percent sign, which in­dicates the Unix prompt:

% **cp ~hneeman/.vimrc ~**

This command means: “Copy the file named .vimrc that’s in Henry Neeman’s home directory into my home directory.”

You **WON’T** have to do this for future logins.

**NOTICE**:

* The Unix copy command is cp.
* The first filename after cp is the *source* (the thing that you’re making a copy of); the second is the *destination* (the name and/or location of the copy).
* Henry Neeman’s username on OSCER supercomputers is hneeman.
* The filename .vimrc begins with a period (very important). The filename is pro­nounced “dot vim-are-see.”
* In Unix, filename s are *case sensitive*, meaning that it matters whether you use *upper case* (capital) or *lower case* (small) for each letter in a filename.
* In Unix, pieces of a filename (or actually of the directory that it’s in) are separated by slashes, **NOT** by backslashes as in Windows.
* The symbol ˜ (known as a *tilde*, pronounced “TILL-duh”) denotes your home direc­tory (another way to denote your home directory is ˜yourusername).
* The substring ˜hneeman means “the home directory of the user named hneeman.”
* There are no spaces between the slash and the .vimrc.
* If for some reason this doesn’t work, try:

% **cp /home/hneeman/.vimrc /home/yourusername**

1. Enter the following command:

% **cp ~hneeman/.nanorc ~**

You **WON’T** have to do this for future logins.

1. Create a *subdirectory* of your home directory named SIPE2011\_exercises, like so:

% **mkdir ~/SIPE2011\_exercises**

**NOTICE**: In the subdirectory name SIPE2011\_exercises, the SIPE **MUST BE CAPITALIZED**; that is, the directory name is “capital-O capital-C capital-U capital-C capital-S underscore exercises” with no spaces or other characters in between.

This command means: “Create a directory named SIPE2011\_exercises as a subdirectory inside my home directory” (it’s like creating a new folder named SIPE2011\_exercises on your desktop in Windows).

You **WON’T** have to do this for future logins.

1. Confirm that you have successfully created your SIPE2011\_exercises directory by listing the contents of the current working directory:

% **ls**

SIPE2011\_exercises

This command means: “List the names of the files and subdirectories in my current working directory.”

**NOTICE** that the command is “ell ess” — that is, small-L small-S — rather than “one ess” and that ls is short for “list.”

1. Set the *permissions* on your SIPE2011\_exercises directory so that only you can ac­cess it:

% **chmod u=rwx,go= SIPE2011\_exercises**

This command means: “Change the *mode* (list of permissions) on my subdirectory named SIPE2011\_exercises so that I (the user) can read files in it, write files in it, and go into (execute) it, but nobody else can.”

Your SIPE2011\_exercises directory is now ac­cessible only to you. The only other people who can access it are the *system* administrator*s* (*sysadmins* for short) of this Linux cluster supercomputer; that is, OSCER operations staff (excluding Henry).

You **WON’T** have to do this for future logins.

1. Log out of the Linux cluster supercomputer by entering the following command:

% **exit**

Once you have completed the setup steps in this section, you **WON’T** have to do them again when you log in later.

**III. COPY HENRY’S Intro DIRECTORY INTO YOUR SIPE2011\_exercises DIRECTORY**

1. Log in again.
2. Conﬁrm that you’re in your home directory:

% **pwd**

/home/yourusername

1. Check that you have a SIPE2011\_exercises subdirectory inside your home directory:

% **ls**

SIPE2011\_exercises

1. Go into your SIPE2011\_exercises subdirectory:

% **cd SIPE2011\_exercises**

This command means: “Change the working directory to SIPE2011\_exercises, which is a subdirectory of my current working directory.” (This is like double-clicking a folder in Windows.)

1. Conﬁrm that you’re in your SIPE2011\_exercises subdirectory:

% **pwd**

/home/yourusername/SIPE2011\_exercises

1. See what files or subdirectories (if any) are in the current working directory:

% **ls**

You may get no output, just the Unix prompt; if so, that indicates that your current working directory has no files or subdirectories in it.

1. **SIDEBAR**: To learn more about a particular Unix command, enter:

% **man *commandname***

for some command. For example, try

% **man chmod**

which will give you the online *manual page* for the chmod command.

The output of man goes through another command, more, which shows one screenful at a time. To get the next screenful, press the spacebar; to get the next line, press the **Enter** key. To quit the more command, press the **Q** key.

1. Copy the subdirectory named Intro from Henry’s SIPE2011\_exercises directory into your SIPE2011\_exercises directory:

% **cp -r ~hneeman/SIPE2011\_exercises/Intro ~/SIPE2011\_exercises/**

This command means: “Copy the subdirectory named Intro inside the directory named SIPE2011\_exercises under the home directory of user hneeman into my directory SIPE2011\_exercises under my home directory.”

1. Conﬁrm that the Intro subdirectory was copied into your SIPE2011\_exercises directory:

% **ls**

**Intro**

1. Go into your Intro subdirectory:

% **cd Intro**

1. Conﬁrm that you’re in your Intro subdirectory:

% **pwd**

/home/yourusername/SIPE2011\_exercises/Intro

1. See what files or subdirectories (if any) are in the current working directory (Intro):

% **ls**

C Fortran90

1. Go into either your C subdirectory or your Fortran90 subdirectory (**BUT NOT BOTH**):

% **cd C**

OR

% **cd Fortran90**

1. Conﬁrm that you’re in your C or Fortran90 subdirectory:

% **pwd**

/home/yourusername/SIPE2011\_exercises/Intro/C

OR the output of the pwd command might be:

% **pwd**

/home/yourusername/SIPE2011\_exercises/Intro/Fortran90

1. See what files or subdirectories (if any) are in the current working directory:

% **ls**

makefile my\_number.bsub my\_number.c my\_number\_input.txt

OR the source file might be named my\_number.f90 instead of my\_number.c.

**IV. EDIT THE BATCH SCRIPT FILE TO CREATE YOUR OWN UNIQUE VERSION**

1. Before you can run the original version of the program, you need to modify your copy of the batch script file my\_number.bsub to create a version that’s uniquely yours.

Using your preferred Unix text editor (whether nano, pico, vim, vi, emacs or whatever), edit your copy of my\_number.bsub.

For example, if you’re using nano, then the edit command would be:

% **nano my\_number.bsub**

This command means: “Edit the text in the file named my\_number.bsub that’s in my current working directory, using the text editor program named nano.”

If you need help using nano, please send e-mail to support@oscer.ou.edu.

1. In nano, notice the little help messages at the bottom of the screen:

^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg ^K Cut Text ^C Cur Pos

^X Exit ^J Justify ^W Where is ^V Next Pg ^U UnCut Text ^T To Spell

For example, consider

ˆW Where is

This means that you should press **Ctrl-W** (the caret ^ indicates the Ctrl key) to search for a particular string of characters.

Another example:

ˆC Cur Pos

This is short for “Cursor Position” and causes nano to tell you what line number the cursor is located at.

Another example:

ˆK Cut Text

This means “delete the line that the cursor is currently on.”

1. Using the text editor, make the following changes to my\_number.bsub:
2. Everywhere throughout the file, change yourusername to your user name (which might be of the form sipe####, or perhaps is based on your name). **THIS IS EXTREMELY IMPORTANT!**
3. Everywhere throughout the file, change

youremailaddress@yourinstitution.edu

to your full e-mail address. **THIS IS EXTREMELY IMPORTANT!**

1. **IMPORTANT!** Every few minutes while you’re editing, you should save the work that you’ve done so far, in case your work is interrupted by a computer crashing. In nano, enter **Ctrl-O** (the letter oh), at which point nano will ask you, near the bottom of the screen:

File Name to write : my\_number.bsub

That is, nano wants to know what filename to save the edited text into, with a de­fault filename of my\_number.bsub). Press **Enter** to save to the default filename my\_number.bsub.

1. The lines of text in the batch script file my\_number.bsub should be less than 80 char­acters long, and ideally at most 72 characters long. (Your PuTTY window should be 80 characters wide.)
2. Some text editors, including nano, try to help keep text lines short, by breaking a long line into multiple short lines. For example, nano might break a line like the following into two separate lines:

#BSUB –o

/home/yourusername/SIPE2011\_exercises/Intro/C/my\_number\_%J\_stdout.txt

That is, nano automatically puts a carriage return when the line starts getting too long for its taste.

Unfortunately, the batch scheduler (LSF, for Load Share Facility) will consider this to be an error. Why? Because the batch scheduler cannot allow an individual batch directive – that is, a line starting with #BSUB – to use more than one line.

For example, the batch script directive above should be on a single line:

#BSUB –o /home/yourusername/SIPE2011\_exercises/Intro/C/my\_number\_%J\_stdout.txt

So, you’ll need to correct any such occurrences.

1. After you’ve finished editing, go back up to the top of the batch script file, and **CAREFULLY READ THE ENTIRE BATCH SCRIPT FILE FROM START TO FINISH.** This will give you a much clearer understanding of what batch computing is and how it works.
2. **Understanding batch computing**:

As an analogy, imagine that you’re at a football game and you want a drink. You get up and walk to the concession stand. If there are a lot of people at the concession stand, then you’re going to have to wait a while before a server serves you, but if you’re the only person in line, or more generally if there are at least as many servers behind the counter as customers lined up to buy, then you’ll be served quickly.

Batch computing is analogous, except that instead of food and drink, you and the other users want your jobs to be run, and instead of food servers, the servers are computers that can run jobs. Typically, for a production cluster supercomputer, the number of resources requested by the users – that is, total servers requested – is much larger than the number of available resources (servers).

In the case of OSCER’s Linux cluster supercomputer, Sooner, the number of users is roughly 750, and the number of servers (computers) is roughly 500 – but most users want to use many servers at the same time. The only way to make this work is for a program known as a *scheduler* – in this case, LSF (Load Share Facility) – to decide whose jobs run on which servers, and when.

Compare getting food at a football game to getting food at home, where you just walk up to your fridge or cupboard or whatever, and take out what you want. But if you’ve got hundreds of people getting food, that method won’t work: it doesn’t *scale* to hundreds of people sharing one source of food, because you can’t fit all of them in front of the one fridge; instead, everyone has to wait their turn at the counter, and work with a server to get served.

Likewise with computing: your normal way of interacting with your laptop won’t work when hundreds of people are sharing one source of computing.

1. After you’ve finished editing and reading the batch script file, exit the text editor.

For example, in nano, enter **Ctrl-X**. If you have made any changes since the last time you entered Ctrl-O, then nano will ask you, near the bottom of the screen,

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES)?

To save your most recent changes to the file (which is probably what you want to do), press the **Y** key; to avoid saving your most recent changes, press the **N** key.

After that, nano will behave the same as if you had entered **Ctrl-­O**.

**V. LOOK AT, MAKE (COMPILE) AND RUN THE ORIGINAL VERSION**

1. For your own understanding, look at the contents of the source file:

% **cat my\_number.c**

OR:

% **cat my\_number.f90**

This command means: “Output the contents of the text file named my\_number.c (or my\_number.f90) to the terminal screen.”

**NOTICE** that the command to output the con­tents of a text file to the terminal screen without using the more command is cat, which is short for “concatenate,” a word that means “output one text file after another in sequence.”

The output of the cat command goes to the terminal screen (known as “standard output,” or “standard out” for short, abbreviated stdout), and in this case, we are only concatenating a single text file, so we’re simply outputting the text file’s contents to the ter­minal screen.

If you’re using PuTTY as your SSH client, and the contents of the file exceeds the height of the PuTTY window, then you can scroll up or down using the scrollbar on the right side of the window; most other SSH clients have similar capability.

1. For your own understanding, look at the contents of the input file:

% **cat my\_number\_input.txt**

1. For your own understanding, look at the contents of the makefile:

% **cat makefile**

1. *Make* (compile) the *executable* program for Henry’s original version of my\_number.c (or my\_number.f90):

% **make my\_number**

gcc -O -c my\_number.c

gcc -O -o my\_number my\_number.o

(It could be the case that the compiler is gfortran and the source file is my\_number.f90.)

**NOTICE**:

* In the make command, the *command line argument* my\_number is the name of the *executable* (the file that can actually be run) that you are making.
* The make command runs the C compiler gcc (OR the Fortran90 compiler gfortran) to compile the source file named my\_number.c (OR my\_number.f90).
* In the compile command, the *command line option*

-o my\_number

indicates that my\_number is to be the name of the executable.

If that option had been left out, then by default the name of the executable would be a.out (“the output of the assembler”), **WHICH WOULD BE BAD**, because then the executable’s filename wouldn’t explain the executable’s purpose.

1. Submit the batch script file my\_number.bsub to the batch scheduler:

% **bsub < my\_number.bsub**

**NOTICE** the less than symbol < which is **EXTREMELY IMPORTANT**.

You should get back output something like this:

Job <######> is submitted to queue <normal>.

where ###### is replaced by the batch job ID for the batch job that you’ve just submitted.

1. Check the status of your batch job:

% **bjobs**

You’ll get one of the following outputs, either:

No unfinished job found

(if you get this right after the bjobs command, try it several more times, because sometimes there’s a pause just before the batch job starts showing up, as below),

OR:

JOBID USER STAT QUEUE FROM\_HOST EXEC\_HOST JOB\_NAME SUBMIT\_TIME

4081250 hneeman PEND normal sooner1 my\_number Oct 17 14:58

where ###### is replaced by a batch job ID number, and yourusername is replaced by your user name, and where PEND is short for “pending,” meaning that your job is waiting to start,

OR:

JOBID USER STAT QUEUE FROM\_HOST EXEC\_HOST JOB\_NAME SUBMIT\_TIME

4081250 hneeman RUN normal sooner1 c127 my\_number Oct 17 14:58

1. You may need to check the status of your batch job repeatedly, using the bjobs command, until it runs to completion. **This may take several minutes (occasionally much longer).**

You’ll know that the batch job has finished running when it no longer appears in the list of your batch jobs, in which case it may say:

No unfinished job found

1. Once your job has finished running, ﬁnd the *standard output* and *standard error* files from your job:

% **ls -ltr**

**NOTICE** that the command is “ell ess space hyphen ell tee are” – that is, small-L small-S blank hyphen small-L small-T small-R – rather than “one ess” or “one tee are” and that ls is short for “list” and -ltr is short for “long detailed listing, sorted by time of most recent modification, in reverse order so that the most recently modified file is at the bottom.”

Using this command, you should see files named

my\_number\_######\_stdout.txt

and

my\_number\_######\_stderr.txt

(where ###### is replaced by the batch job ID).

These files should contain the output of my\_number. Ideally, the file length of my\_number\_######\_stderr.txt should be zero.

1. Look at the contents of the standard output file:

% **cat my\_number\_######\_stdout.txt**

(where ###### is replaced by the batch job ID).

You may want to look at the stderr file as well:

% **cat my\_number\_######\_stdout.txt**

1. If this run had **ANY** problems, then send e-mail to:

support@oscer.ou.edu

which reaches all OSCER staff (including Henry), and attach the following files:

makefile

my\_number.c

my\_number.bsub

my\_number\_######\_stdout.txt

my\_number\_######\_stderr.txt

10. Congratulations! You’ve just run your first batch job. Now continue on.

**VI. EDIT THE C SOURCE FILE TO CREATE YOUR OWN UNIQUE VERSION**

1. Now that you’ve run the original version of the program, it’s time to modify your copy of the source file my\_number.c (or my\_number.f90) to create a version that’s uniquely yours.

Using your preferred Unix text editor (whether nano, pico, vim, vi, emacs or whatever), edit your copy of my\_number.c (or my\_number.f90).

For example, if you’re using nano, then the edit command would be:

% **nano my\_number.c**

OR

% **nano my\_number.f90**

1. Using the text editor, make the following changes to my\_number.c (or my\_number.f90):
	1. In the *declaration section*, change the constant values assigned to minimum\_number, maximum\_number, close\_distance and computers\_number.

You may se­lect any integer values you want, which must be different from 1, 5, 10 and 1 re­spectively, and minimum\_number < computers\_number < maximum\_number, and they are sufficiently spread out that you can actually do the runs properly.

**DON’T CHANGE THE VALUE OF maximum\_guesses!!!**

* 1. In the *execution* sectio*n* (also known as the *body* of the program), change the following sequences of character text to your own words:
	2. Hey!
	3. That’s amazing!
	4. Close, but no cigar.
	5. Bzzzt! Not even close.
1. Every few minutes while you’re editing, you should save the work that you’ve done so far, in case your work is interrupted by a computer crashing.

In nano, enter **Ctrl-O** (the letter oh), at which point nano will ask you, near the bottom of the screen:

File Name to Write [Backup]: my\_number.c

That is, nano wants to know what filename to save the edited text into, with a default filename of my\_number.c (or my\_number.f90). Press **Enter** to save to the default filename my\_number.c (or my\_number.f90).

1. A *character string literal constant*, also known as a *character string literal* or a *string literal* for short, is a sequence of characters between a pair of double quotes.

For example, in the C printf statement

printf("This is a printf statement.\n");

the following is a string literal:

"This is a printf statement.\n"

Likewise, in the Fortran90 PRINT statement

PRINT \*, "This is a PRINT statement."

the following is a string literal:

"This is a PRINT statement."

We say that the pair of double quotes *delimits* the sequence of characters in the string literal. Note that, in C, the \n at the end of the string literal tells the program to output a carriage return (also known as a *newline*) at the end of the line of output text. (In Fortran90, the carriage return is implied by the end of the PRINT statement.)

1. The lines of text in the source file my\_number.c (or my\_number.f90) should be less than 80 characters long, and ideally no more than 72 characters long. (Your PuTTY window should be 80 characters wide.)
2. Some text editors, including nano, try to help keep text lines short, by breaking a long line into multiple short lines. For example, nano might break a line like:

printf("This is a long line and nano will probably break part of it off.\n");

into two separate lines:

printf("This is a long line and nano will probably break part

of it off.\n");

That is, nano automatically puts a carriage return when the line starts getting too long for nano’s taste.

Unfortunately, the C compiler (or the Fortran90 compiler) will consider this to be an error. Why? Because C (or Fortran90) cannot allow an individual string literal to use more than one line (in Fortran90, there’s a goofy way to do it, but it’s bad practice). So, the correct way to write the above example is:

printf("This is a long line and nano will probably");

printf(" break part of it off.\n");

OR:

PRINT \*, "This is a long line and nano will probably"

PRINT \*, " break part of it off."

1. Like the lines of source text, the lines of output text should be less than 80 characters long, and ideally no more than 72 characters long. You can break a long line of output text into shorter pieces by making it into two printf (or PRINT) statements.

For example:

printf("Why you big old stinker! That’s not between %d and %d!\n",

 minimum\_number, maximum\_number);

This single printf statement can be converted into two printf statements, like so:

printf("Why you big old stinker! That’s not between\n");

printf(" %d and %d!\n", minimum\_number, maximum\_number);

OR:

PRINT \*, "Why you big old stinker! That’s not between"

PRINT \*, minimum\_number, " and ", maximum\_number, "!"

1. After you’ve finished editing, exit the text editor. For example, in nano, enter **Ctrl-X**.

If you have made any changes since the last time you entered **Ctrl-O**, then nano will ask you, near the bottom of the screen:

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES)?

To save your most recent changes to the file (which is probably what you want to do), press the **Y** key; to avoid saving your most recent changes, press the **N** key. After that, nano will behave the same as if you had entered **Ctrl-O**.

1. Edit the input file my\_number\_input.txt to replace the original input values with input values relevant to your new unique version of the program, speciﬁcally:
	1. an integer value less than your value for minimum\_number;
	2. an integer value greater than your value for maximum\_number;
	3. an integer value between your value for minimum\_number and your value for maximum\_number (inclusive), but far from your value for computers\_number;
	4. an integer value close to your value for computers\_number (that is, within your value for close\_distance of your value for computers\_number);
	5. your value for computers\_number.

**VII. MAKE (COMPILE), RUN AND DEBUG YOUR OWN UNIQUE VERSION**

1. Make (compile) your own unique version of the executable program:

% **make my\_number**

gcc -O -c my\_number.c

gcc -O -o my\_number my\_number.o

(It could be the case that the compiler is gfortran and the source file is my\_number.f90.)

1. If the program doesn’t compile, then you’ll need to edit it and figure out where things went wrong. In the worst case, if you’re totally stumped, then copy the original from Henry’s directory again, and start editing from the beginning.
2. Repeat the instructions in section V, items 5-9, above.
3. Congratulations! You’ve just completed this exercise.