I. EXERCISES - SESSION 2

A. Comments on Sessions:

The goals of this session are the following:

1. Learn how to use command line inputs for the main code
2. Read and write from FILES
3. Use makefiles
4. Wrap up with the initial error analysis

NOTE: Beginners attempt the Warm-up section entirely (qs 1 through 5) and Advanced attempt the warm-up section from qs 1 to qs 4 and do qs 6 from the problem section. As always the entire sheet is open to all and the above states the minimum that is needed for this lab.

B. Preparations:

1. Preparations: Download the session02.tar.gz file from

   ```
   www.physics.iitm.ac.in/~suna/nummethods.html
   ```

   under the section DCF sessions. The file usually downloads into the Desktop directory or the Downloads directory. Open the terminal (click on Applications → System tools → Konsole for Ubuntu running GNOME or search for Terminal on Ubuntu running Unity). Move the .tar.gz file to the home directory using the following command (remember to hit enter after each of the following commands):

   ```
   mv Desktop/session02.tar.gz ./
   ```

   Untar the files using the following commands:

   ```
   tar xfvz session02.tar.gz
   ```

   This will automatically open a directory called session02. Move into that directory by typing the following command on the terminal:

   ```
   cd session02/
   ```

   You can check the files in that directory by typing: (lists)

   ```
   ls
   ```

C. Warm-up

1. Basic Programing: Launch the editor gedit as you did in the previous session. Open the code area_new.c. This code illustrates how to use command-line arguments in the main code. We will compile this code using the makefile, which you need to create as follows: Copy the file make_area to make_area_new using the following command:

   ```
   cp make_area make_area_new
   ```

   Replace area with area_new. Compile using the the following command:

   ```
   make -f make_area_new
   ```

An interesting fact about makefiles: If you had saved this file as simply makefile, then you could compile it using the command make. But it is a good practise to have the names of files built in, so that we know the code for which a particular makefile is intended. Hence we will use the command given above. Now run the executable using the following command:

```
./area_new 1.0
```

Note in this case the first entry is the name of the executable, while the second (separated by a space) is the numerical value of the radius. Next try running the code for $r = 4.0$.

Go through the code area_new.c The main difference between this code and the area.c from the previous session is that this code takes arguments from the user from the command line (i.e. what you type at the prompt on the terminal). These arguments are stored as strings in the array argv[ ]. The first entry, i.e. argv[0] = ./area_new while the second argument is the numerical value of the radius, but this is a string. Hence we need to convert it to a real number which is done through the function atof, where we pass the second element of the array argv. argc keeps track of the number of command line variables.

2. You can also pass files through command line. Open the code area_input.c Note that this code can take inputs from command line, but it accesses a file that has been passed to the code through the argv array. Open the file area_input.inp This is the input file, although trivial in this example, this file could contain all the parameters that your code needs. Hence if you need to run your code for another radius, you do not have to compile it, you just need to change your input file! Create a makefile for area_input.c by copying make_area_new to make_area_input and replace area_new by area_input. Now compile using the make command (as you did in the previous part). Run the executable using the following command:

   ```
   ./area_input area_input.inp
   ```

   Change the radius to 4.0 in the input file and run again using the command above.

3. Makefiles: Open the code precision.c and compile it using the gcc command

   ```
   gcc -o precision precision.c
   ```

   and run the executable. What do you expect to see and what happens?

   Compile using the makefile make_precision using the following command:

   ```
   make -f make_precision
   ```

   Does your code compile? If not what are your errors?

4. Debugging: Open the code series.c. This code evaluates $\exp(-x)$ in three different ways: 1. using the brute force power series, 2. using a recursion relation and 3. by evaluating $\exp(x)$ and finding its inverse. Find the error in the code series.c. Compile using both the gcc command and using the makefile. (Hint: it is not due to the variable rel_err being declared and not used that you will get from the makefile. Remember in order to compile fresh with makefile after you have compiled with the gcc command, you will need to first remove the

.o files, which you can do by the command: make -f make_series clean, otherwise the makefile will tell you that the executable is up to date.)

5. (Beginners) Reading and Writing to files: Let us modify the code area_input.c to do the following: We would like to loop over r values starting from from minimum to some maximum and store the calculated area in an output file. Use makefiles to compile your code. Your task will be:
   (a) Pass $r_{\min}$ and $r_{\max}$ through the input file (therefore you need to read these in the code).
   (b) Loop between $r_{\min}$ and $r_{\max}$, calculate the area and dump the result to an output file. Remember that you need to open the output file in the "w" mode, which will create the file in the first place.
   (c) With the help of the tutorial from the class website, try to plot the area as a function of radius using gnuplot.

D. Problems

6. (Advanced) Summing Series: Write a code to sum the following series in two ways:

$$S_{\text{up}} = \sum_{n=1}^{N} \frac{1}{n}$$

and

$$S_{\text{down}} = \sum_{n=N}^{n=1} \frac{1}{n}$$

   (a) Calculate the relative error between the two ways of summing, defining the error as a function of N, which is the number of terms retained.

$$\epsilon_{\text{rel}} = 2 \left| \frac{S_{\text{up}} - S_{\text{down}}}{S_{\text{up}} + S_{\text{down}}} \right|$$

   You will need to work with single precision and go as large as $N = 10^{10}$ in order to see the results. This means you could increment the number of terms retained in powers of 10, starting from some minimum value and going all the way until the maximum value of 10^{10}. Store the relative erros as a function of N into a file.
   (b) Which way of summing will work well and why?
   (c) Plot the logarithm of the data using gnuplot (tutorial available on the class website) and exlain your plot.