I. EXERCISES - SESSION 1

A. Comments on Sessions:

The goal of these hands-on sessions is to apply what you have learned during the lectures and also pick up C along the way. The sheet is due by the end of the lab session. Each lab is worth 10 points. The advanced programmer can skip the basic programming section (or skim through it to refresh his/her memory), but will have to attempt the problems section completely. For a quick programmer, there are problems at the end of the problem section that can be attempted, so that you get more out of this lab! If you need syntax, please ask.

NOTE on sessions: Any code that is written or modified by you should be commented. Please make sure you comment files too! Use separate lines for variable declarations (See my codes for coding ethics!). Please show your codes to the TA or your instructor so that you can get your grades.

B. Preparations:

1. Preparations: Download the session01.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/session01.tar.gz ./
   ```

   Untar the files using the following commands:

   ```
   tar xfvz session01.tar.gz
   ```

   This will automatically open a directory called session01. Move into that directory by typing the following command on the terminal:

   ```
   cd session01/
   ```

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

   ```
   ls
   ```

C. Warm-up

1. Basic Programing: Launch an editor using the following command on the terminal:

   ```
   gedit &
   ```

   The & symbol runs the editor in the background freeing the terminal for other use. (Try without the & and see if you understand what I mean!) Open the code area.c and make sure you understand it. Summarize what the code area.c does here:

   (a) Compile the code area.c using the following command:

   ```
   gcc -o area area.c
   ```

and run the executable using the following command:

```
./area
```

How would you check the correctness of the code? (Hint: Think of a special value of r where you know the answer right away!)

(b) Try compiling in the following way:

```
gcc area.c
```

This will generate an executable called a.out in your directory. Now you can run the executable using:

```
./a.out
```

(c) Can you explain the difference between the two ways of compiling?

2. More compilations: Open the code area_modified.c. Notice that the calculation of area has been shuffled to a subroutine and you will see that we have something called function prototype which declares the subroutine. Declaring function prototypes is essential so that the compiler knows how to handle the function area_calc. Now compile the code using the gcc command:

```
gcc -o area_modified area_modified.c
```

Next open the codes area_modified_rev2.c and area_calc.c. Notice that the subroutine that calculates the area is now a separate C code. Such standalone codes are really helpful for later use. This is something that we will see again. Let us compile the code using the following:

```
gcc -o area_modified_rev2 area_modified_rev2.c area_calc.c
```

Explain what you understand below:

3. Copy volume.c (this code calculates volume) into a another called volume_new.c using the following command:

```
cp volume.c volume_new.c
```

Open volume_new.c in the editor, gedit. Now modify the code such that the volume calculation is a subroutine (function), similar to what has been done in area_modified.c Compile and run the code.

D. Problems:

1. Floats:

(a) Open the code volume.c and compile it. Check the code at $r = 1$, where r is the radius. What do you expect and what does the code give? Can you figure out the reason?

(b) Repeat the exercise with the code volume_modified.c. How do the results in the two code compare?

(c) What happens if we code the volume as $4 * \text{radius} * \text{radius} * \text{radius}/3$?

(d) What is the result of 1/2 and 1./2. on a computer?

2. Precision and Accuracy: One can define machine precision as the largest number ϵ such that $1 + \epsilon = 1$.

(a) What are the steps you will need in order to calculate machine precision?

(b) Write a code to calculate precision and compile it. What is the machine precision for singles? and doubles?

3. (Advanced) Write a code to convert an integer in decimal to its binary equivalent.

4. (If you are really fast!) Write a code to implement calculating factorial functions for arbitrary values of an integer n. How large can n be?

5. (If you are really fast!) Round-off errors in subtractive cancellations:

(a) The solution to a quadratic equation:

$$ax^2 + bx + c = 0 \tag{1}$$

is

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

When would you be plagued by round-offs due to subtractive cancellations?

(b) Write a code that determines the roots of the quadratic equation above, for the following values of the coefficients: $a = 1$, $b = 1$ and $c = 10^{-n}$. Let n loop over the following values $(1, 2, \cdots 15)$. Calculate the roots compare against the expected answer. What happens and why? Can you think of a reliable way to estimate the roots in this case when $\sqrt{b^2 - 4ac} \approx \pm b$?