INF1002 Programming Fundamentals – C Lab1

OBJECTIVES:

1. To choose, install, and configure a development environment suitable for C programming.
2. To comprehend C expressions and programs.
3. To write, compile, and debug C programs.

EXERCISE-1: HELLO WORLD!

Once you've chosen and installed your development environment, make sure that you can type in, compile, and execute the famous "Hello World!" program:

```
#include <stdio.h>

int main() {

    printf("Hello world!\n");
    printf("Welcome to INF1002!\n");

    return 0;
}
```

EXERCISE-2: HELLO WORLD!

Suppose that the following variable declarations have been made in a program:

```
int a = -1, b = 2;
float x = 0.1;
float y = 1.5;
char c = 'p';
```

1. Write a program to print the value for each of the following calculation to the screen.
    a) a / b
    b) a * b
    c) (b * 3) % 4
    d) x * a
    e) x * y
    f) y / x
    g) c – 3
2. Write a program to print the results.
    a) printf("%4d", a);
    b) printf("%4d", b);
    c) printf("a/0b = %d", a / b);
    d) printf("%x", b);
    e) printf("%.2f", y);
    f) printf("%10.1f", x);
    g) printf("c =\t%c", c);

EXERCISE - 3: READING INPUT AND PERFORMING CALCULATIONS

The Body Mass Index (BMI) is calculated as follow:

$$BMI = \frac{weightInKilograms}{heightInMetres \times heightInMetres}$$

Create a new program called "bmi" that asks the user to type in his or her weight in kilograms and height in metres, then calculates and displays the user's BMI to one decimal place. The application should also evaluate whether the user is underweight or overweight according to the following table:

BMI	Evaluation
Less than 18.5	Underweight
Between 18.5 and 24.9	Normal
Between 25.0 and 29.9	Overweight
30.0 or greater	Obese

The following shows some sample output for the program, with the user input shown in red:

```
Enter your weight in kilograms: 65
Enter your height in metres: 1.85
Your BMI is 19.0. That is within the normal range.
```

Lab Assignment:

To help you better practice, you need to perform a set of tasks in one auto-grading system, Gradescope in LMS/xSiTe. Gradescope will provide you immediate feedback of your program, such that you will know the issue of your program.

In this lab, you need to finish one task in Gradescope system. Gradescope system will provide you immediate feedback about the correctness of your program. You can view the feedback of your system by clicking the failed case. Gradescope adopts a test cases-based approach to check your program. Your score is depending on the number of test cases that you program can pass. Note that to train you to have a good programming practice, you must write your program strictly according to the requirement of the tasks, including your input and output format. If there is any difference (even one space), your program will fail on the test cases. SO, TRAIN YOURSELF TO BE AN EXACT THINKER!

To help you practice, you are allowed to do multiple attempts for each task. Enjoy your learning!

Task: GUESS INTEGER

Task Description:

Write a program called "*guessInteger.c*" that will play a simple two-player number-guessing game as follows:

1. The program will output "Player 1, enter a number between 1 and 1000:". If the player enters a number that is not in this range (inclusive), the program will output "That number is out of range." This repeats until Player 1 has entered a number that is in range.

2. Player 2 (who has not been watching Player 1) has 10 rounds in which to correctly identify the number. At the start of each round, the program will output "Player 2, you have *n* guess(es) remaining." where *n* is the number of rounds left
3. The program will output "Enter your guess:" and wait for Player 2 to enter a number.
   a. If Player 2's guess is too high, the program will output "Too high."
   b. If Player 2's guess is too low, the program will output "Too low."
   c. If Player 2's guess is correct, the program will output "Player 2 wins." and stop.
   d. If Player 2's guess is out of range, the program will output "That number is out of range." and return to Step 3. The number of guesses should *not* be incremented.
4. If Player 2 has not guessed the number at the end of the 10th round, the program will output "Player 1 wins." and stop.

Some sample output is shown over the page, with the user input shown in red:

```
Player 1, enter a number between 1 and 1000:
1500
That number is out of range.
Player 1, enter a number between 1 and 1000:
500
Player 2, you have 10 guesses remaining.
Enter your guess:
750
Too high.
Player 2, you have 9 guesses remaining.
Enter your guess:
250
Too low.
Player 2, you have 8 guesses remaining.
Enter your guess:
500
Player 2 wins.
```

Note:

- Use macros (*#define*) to represent all constants, so that it is easy to change things like the number of guesses allowed, the highest number allowed, and so on.
- Don't use *sys.argv[]* for user inputs. Use other functions such as *scanf(), fgets(), fgetc(),* etc.,
- There is no white space in the print after the colons (:).
- Include comment to your code that explains what each section of it does.

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "*guessInteger.c*" from the "C\Lab\Lab1" xSiTe folder.
2. Do not change file name or function names.
3. Add your code to the function "*main()*" and other required functions.
4. Don't use *sys.argv[]* for user inputs. Use other functions such as *scanf(), fgets(), fgetc(),* etc.,
5. Drag and drop your locally tested code to Gradescope. You can submit and auto graded any number of times. Last submission counts.
6. There are 5 test cases and maximum score for this task is 5.
7. Autograder will be timed out after 10 minutes.