Week 4 Assignment Specification
Module 1 Revision

This document contains the specification for all the assignments of the week.
Each question found here should have the following structure

- Question Preamble or Context (for application questions)
- Your Programming Task
- Input format
- Input constraints (i.e. the range of possible values of inputs to your program)
- Output format
- Starter Files (and how you can find them)
- Test Cases (at least one per question)

Because making a stress test or a test suite for each of the code is really too tedious, submit to me all the work you have completed and I will help you test and review your code.

There are **6** programming assignments for this week.
You are required to complete at least **3** out of **6** assignments for this week.

3 of these assignments highlighted in yellow, you are encouraged to do these first and you can complete the rest of the assignments as you revise in your own time.

Table of Contents

FOREWORD
Your programming assignments are getting longer, more demanding in terms of assignment specifications, so this would be the new format of your programming assignments. Remember to not shy away from Google, Stack Overflow, W3Resource, TutorialsPoint, Programiz.com and others– these websites are there to help you if you need specific targeted syntax as it is not efficient to read through the 100 slides just to find one specific thing. You will encounter issues along the way, don't give up, and I'll be one text away!

---

1 Approximating sin(x)

Preamble
The value of $\sin(x)$ for a value of x can be expressed in polynomial terms by using the Maclaurin Series. Formally, an approximation of $\sin(x)$ centered around $x = 0$ is given by

$$\sin(x) = \sum_{r=0}^{n} \frac{(-1)^r x^{2r+1}}{(2r+1)!}$$

Your Task:
Write a program in C that given a value of x and n, outputs an approximation for $\sin(x)$.

Input Format:
The input only consists of 1 line, a floating-point number x followed by an integer n.

Constraints:
You are not allowed to use any functions defined in the `math.h` header file.
$-25 \leq x \leq 25, 0 \leq n \leq 50$.

Output Format:
Output the approximation for $\sin(x)$.

Starter File:
`1_sine.c`
Using this starter code is optional.
This code file implements the input and output formats as specified above, and passes the input to an empty function which you have to implement. The code file would then take the result of your implementation and print it to the console in the specified format.

If you have a different idea in mind, feel free to create a new code file for this question.

[Turn Over for test cases]

TEST CASES FOR Q1

Test Case 1

Input

```
4.0000 5
```

Output:

```
-0.767
```

Test Case 2

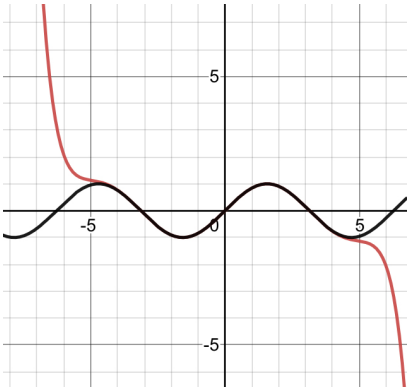
Input:

```
-5.10 5
```

Output:

```
1.151
```

Explanation:



The red graph is an approximation of sin(x) for the first 5 terms of the Maclaurin series. The black graph is the graph of sin(x). Roughly from the looks of the diagram, the domain which the Maclaurin series is a good approximation is within the interval of (-5, 5). The value of x in test case 1 lies within the range, which would give an accurate approximation, while the value of x in test case 2 does not.

Test Case 3

Input:

```
25.1327412287 37
```

Output:

```
0.000
```

Test Case 4

Input

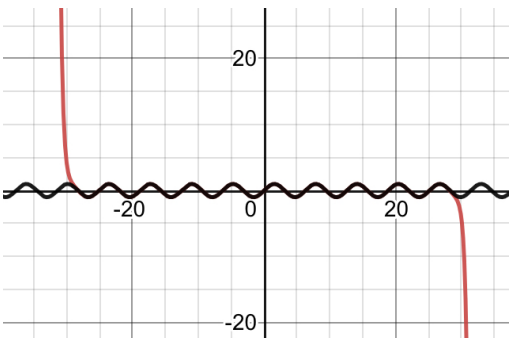
```
29.50 37
```

Output:

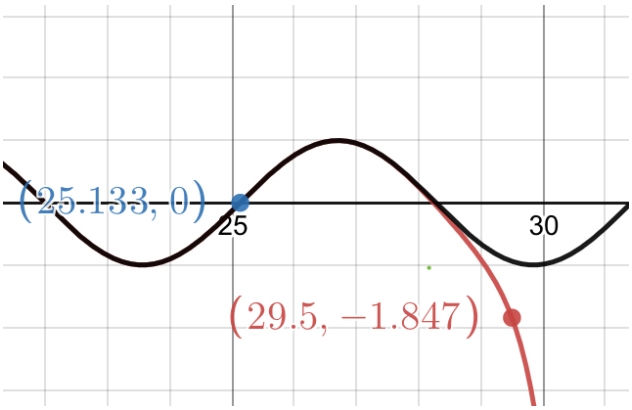
```
-1.847
```

Explanation:

When 37 terms of the Maclaurin series is used, then the graph of the approximation (red) and the graph of $sin(x)$ (black) would look like this.



The x value and its approximation of its sine for test case 3 is marked in blue, and for test case 4 it is marked in red.



END OF Q1

2 Improving Code Readability

Preamble
We learnt that functions are able to enhance code readability. Your previous implementation of the function sine( ) involves only involves 1 user defined function.

There are a few operations within the calculation that could be better encapsulated into functions on their own: such as the exponential component of the formula and the factorial component of the formula highlighted below.

$$\sin(x) = \sum_{r=0}^{n} \frac{(-1)^r x^{2r+1}}{(2r+1)!}$$

Your Task:
By reusing some code appropriately from Q1, implement **2 new** user defined functions for the exponential component and the factorial component of the formula. You will also be required to adapt your entry for the function sine, which will be used to call these new user defined functions.

Note:
Input, output specifics, and constraints are similar to Q1.

Starter File:
`2_sinefunc.c`
You are highly encouraged to use this starter file. Copy over code from Q1 to the functions here, and adapt function sine() accordingly to call these newly added user defined functions when necessary.

If you have a different idea in mind, feel free to create a new code file for this question.

Test Cases:
Reuse the test cases provided to you in Q1.

END OF Q2

3 Types of Approximations

Preamble
The expression $|x|$ has a value of x when $x \geq 0$ and $-x$ when $x < 0$.

There are 3 different kinds of approximations:

- Poor
- Good
- Excellent

In the given domain from $x \in [a, b]$, where a and b are rational,
Function f is considered an **excellent approximation** to a function g if $|f(x) - g(x)| \leq \frac{1}{320}$,
Function f is considered a **good approximation** to a function g if $|f(x) - g(x)| \leq \frac{1}{100}$,
and f is called a **poor approximation** to a function g otherwise, for all real values of x within the interval $a \leq x \leq b$.

For this question in particular,

$$f(x) = \sum_{r=0}^{n} \frac{(-1)^r (x^{2r+1})}{(2r+1)!}, \text{and } g(x) = \sin x$$

Your Task:
Given the number of terms n of the Maclaurin approximation for $\sin x$, and the values of a and b,
Write a program in C that outputs how good an approximation of f is to g within the interval $x \in [a, b]$.
Reuse code from Question 1 or Question 2 when you can.
Include the `math.h` header file and use the sine function provided to you in C for the function g.

Input Format:
The first line would contain the integer n. The second line contains two floating point numbers a and b.

Constraints:
$0 \leq n \in \mathbb{Z} \leq 50, -50 \leq a < b \leq 50$ and $a, b \in \mathbb{R}$

Output Format:
Output one of the three words:
If f is an excellent approximation to g, output the word "`EXCELLENT`". (without quotes)
If f is a good approximation to g, output the word "`GOOD`". (without quotes)
If f is a poor approximation to g, output the word "`POOR`". (without quotes)

Starter File:
`3_approximations.c`
This code file implements the input and output formats as specified above, and passes the input to an empty function which you have to implement. The code file would then take the result of your implementation and print it to the console in the specified format.
Only in this file you are to request to include the `math.h` file and use the in-built sin(x) function in C. You are to use this function to compare it with your approximation in Q1.

[Turn Over for test cases]

TEST CASES FOR Q3

Test Case 1

Input

```
1
-0.8244 0.8244
```

Output:

```
EXCELLENT
```

Test Case 2

Input:

```
1
-0.8246 0.8246
```

Output:

```
GOOD
```

Test Case 3

Input:

```
1
-1.0424 1.0424
```

Output:

```
GOOD
```

Test Case 4

Input:

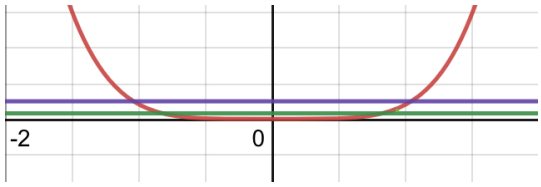
```
1
-1.0426 1.0426
```

Output:

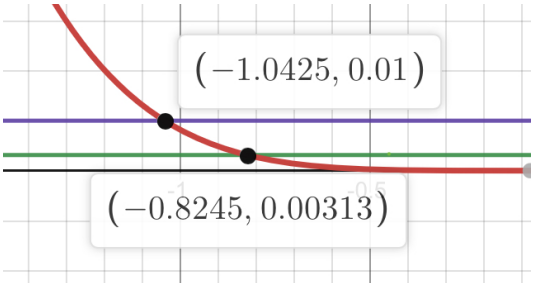
```
POOR
```

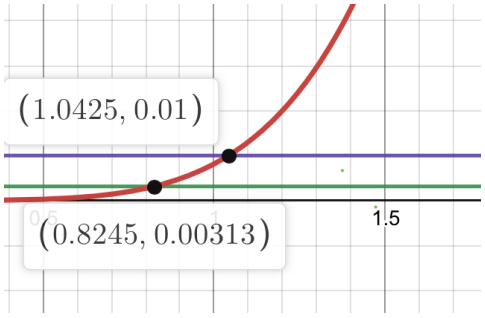
Explanation (for test cases 1 to 4):

In the image below, the red graph is the graph of $y = |f(x) - g(x)|$, the purple line is $y = \frac{1}{100}$ and the green line is $y = \frac{1}{320}$.



In the images that follow, an excellent approximation lies from $x \in [-0.8245, 0.8245]$ and a good approximation lies from $x \in [-1.0425, 1.0425]$. It is poor outside the latter range.





[Turn Over for more Test Cases]

Test Case 5

Input

```
1
-1.0 -0.5
```

Output:

```
GOOD
```

Test Case 6

Input:

```
1
0.5 0.8
```

Output:

```
EXCELLENT
```

Test Case 7

Input:

```
1
-0.83 0.5
```

Output:

```
GOOD
```

Hint:

Aren't there infinitely many real numbers within a given range? How is it possible to check all of them? Don't forget the principles behind Maclaurin series – that is - an approximation most accurate when values are closer to 0. What would this say about values further from zero? The hint is that you only need to **check one or at most two values** to determine whether the approximation is good or bad. But which one or which two? This I will leave you to find out.

END OF Q3

4 Your Name In Hashes

Preamble
Earlier this module, you actually played with the utility input and output functions in C to output your name in hashes.

You will be creating a new version of this program as follows.

Your Task:
Upon running the program, ask the user to enter an integer between 0 and 7 (both inclusive). For the integer i between 0 and 7, print out the first i characters of your name (instead of the full name.

You are also required to ensure that the number that the user enters is within this range. Reprompt the user if the user fails to comply.

Input Format:
An integer i

Constraints:
$0 \leq i \in \mathbb{Z} \leq 7$

Output Format:
The first i characters of your name in hashes.
Starter File:
`4_myprettyname2.c`
The starter code for this file is actually your endpoint for the assignment in Week 0. **Implement the input and output specifics** to match the input and output formats described above, and in the test cases below.

[Turn Over for test cases]

TEST CASES FOR Q4

Test Case 1

Input

```
0
```

Output:

```

```

Explanation:

No letters will be printed, since the number entered is 0.

Test Case 2

Input:

```
2
```

Output:

```
#       #    ############
  #   #            #
    #              #
  #   #            #
#       #    ############
```

Explanation:

The first two characters of the name are printed.

Test Case 3

Input:

```
6
```

Output:

```
#       #    ############          #        #       #        ######    #
  #   #            #             #   #      # #     #      #          #
    #              #            #######     #   #   #      #     ###   #
  #   #            #           #       #    #     # #      #      #    #
#       #    ############     #         #   #       #      ########    ######
```

Explanation:

Test Case 4

Input:

```
8
2
```

Output:

```
#       #    ############
  #   #            #
    #              #
  #   #            #
#       #    ############
```

Explanation:

A user enters an invalid number. As a result, the program waits for the user to enter a second number.

END OF Q4

5 Using Header Files in C

Watch the Assignment Code walkthroughs before attempting this code.

Preamble

To explore use of other header files in C other than the common ones we've seen such as `stdio.h` or `math.h`, in this question, we will draw from in built functions in C across three different header files.

Your Task:

Your task consists of three parts.

1) In Part 1, use the `sleep()` function in C defined in the `unistd.h` header file to create a lag time of **two seconds** between each printf() statement within this segment.

2) In Part 2, you will see something that resembles a portion of your vending machine application. Use the `assert()` function in C defined in the `assert.h` header file to do ensure that the value the user enters is non-negative. Failing the assertion would cause your program to abort (Part 3 will not run!).

3) In Part 3, you will find code calling a function which contains an infinite for loop which counts from 1 to infinity. **Modify the for loop** to stop the execution of it after 1 seconds. You will be required to use a couple of functions defined in the `time.h` header file.

Hint:

Use google! There are specific user friendly functions already provided and already defined.

Starter File:

`5_stdheaderfiles.c`

Because the assignment specifications very closely follow this starter file, you are required to use this starter file for the assignment.

Test Case for Part 1

Output:

```
Each (…2 seconds later)
word (…2 seconds later)
should (…2 seconds later)
appear (…2 seconds later)
after (…2 seconds later)
two (…2 seconds later)
seconds
```

Test Case A for Part 2

Input:

```
20
```

Output:

```
You have 2000 cents.
 Would you like any of these drinks?
```

Test Case B for Part 2

Input:

```
-1
```

Output: (It may be the following format or slightly different)

```
Assertion failed: (some condition it has falsified), function (function
name), file (filename), line (line number)
Abort trap: (number)
```

Test Case for Part 3

Output description:

Program will print 1 to some number that is above 100000. The for loop should suddenly terminate at different values of i each time.

END OF Q5

6 Approximation of Definite Integrals

Preamble

Integration is a fundamental, essential operation of calculus. A **definite integral** of a function $f(x)$ and it represents the area under the curve of the graph of $f(x)$ bounded by one of the axes. A **definite integral** of a function $f(x)$ within the domain $x \in [a, b]$ is given by the expression

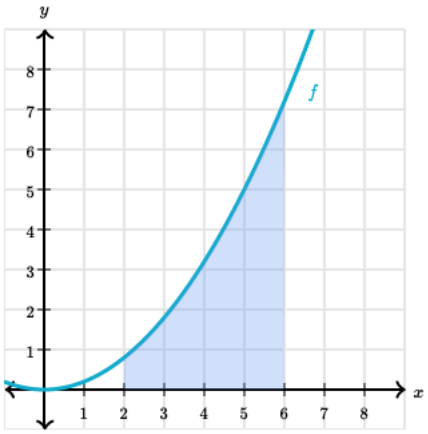
$$\int_a^b f(x)\,dx,$$

where a and b are rational and $a < b$.

The Riemann Sum

The value of the definite integral can be approximated via a technique called the **sum of "infinite" rectangles.**

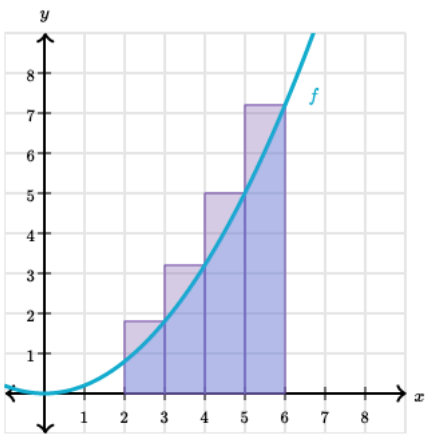
The example below shows the graph of $y = \frac{1}{5}x^2$. For this example, we want to find the area under the graph of $f(x) = \frac{1}{5}x^2$ between $x = 2$ and $x = 6$.



Using definite integral notation, we can represent the exact area as $\int_2^6 \frac{1}{5}x^2\,dx$.

We can approximate this area using **Riemann sums.** Let $R(n)$ be the **right**[1] Riemann sum approximation of our area under $\frac{1}{5}x^2$ using n equal subdivisions (i.e. n rectangles of equal width).

For instance, for $R(4)$, the area is overestimated as we can see in the diagram below:



The area under the curve of f between $x = 2$ and $x = 6$ is approximated using 4 rectangles of equal width.

[1] **Right** here suggests that each rectangle's **right upper corner** intersects the curve.

We can make our approximation better by dividing our area into further rectangles that are smaller in width, by using $R(n)$ for larger values of n.

Preamble Adapted from Khan Academy.org

Your Task:
Integration can be especially tedious for extremely complex expressions for $f(x)$. In this question,

$$f(x) = \sin(x \log_9 x) - \log_9(x \sin x) \text{ and } I = \int_{\frac{\pi}{10}}^{\frac{10\pi}{11}} f(x)\, dx$$

Given a value for n, find the approximation for I using the **right** Riemann sum using n rectangles.

Input Format:
Your input consists of 1 integer – which is the value of n.

Constraints:
$1 \leq n \in \mathbb{Z} \leq 10^6$

Output Format:
The value of $R(n)$, accurate up to 3 decimal places, but printed to the 10th decimal place.

Starter File:
`6_integrals.c`
Using this starter code is optional.
This code file implements the input and output formats as specified above, and passes the input to an empty function which you have to implement. The code file would then take the result of your implementation and print it to the console in the specified format.

If you have a different idea in mind, feel free to create a new code file for this question.

TEST CASES FOR Q6
Test Case 1
Input
```
1
```
Output:
```
2.739
```
Explanation:
Since there's only 1 rectangle, $f\left(\frac{11\pi}{10}\right) = 1.0776$ and the size of the interval is 2.5418. The integral is overestimated to a value of approximately $1.0776 \times 2.542 = 2.74$.
Test Case 2
Input:
```
1000000
```
Output: (Doesn't have to match after the third decimal place)
```
0.9176021176
```
Explanation:
As n gets large, the value of the area approaches $\int f(x)dx$. When computing on a graphing calculator, $I = \int f(x)dx = 0.91760188756$

END OF Q6

END OF WEEK 4 ASSIGNMENT