CS330 - Computer Organization & Assembly Language
Assignment # 2

Individual Work Only

Problems:

cubeOfOdd (n)

Write the function "**cubeOfOdd**" that takes an int **n** as input. The function will print **i**^{3} value for *all non-negative odd integers* **i<n**. More simply: print the cube of all odd integers less than the input.

Sample Input:	**Expected Output:**	***Hints***
n1 = 5	1 27	1^3=1 3^3=27
n1 =3	1	1^3=1
n1 =8	1 27 125 343	1^3=1 3^3=27 5^3=125 7^3=343

introToCS330 (n)

Write the function introToCS330 that takes a positive integer **n** and prints a string according to the following conditions

- if n is divisible by 7, it should print "UAB"
- if n is divisible by 3, it should print "CS"
- if n is divisible by both 3 and 7, it should print "UAB CS 330"
- if n is a prime number other than 3 or 7, it should print "Go Blazers"
- otherwise, it should print the cube of n

Sample Inputs	Expected Outputs
n=3	"CS"
n=70	"UAB"
n=4	64
n=17	"Go Blazers"
n=42	"UAB CS 330"

printHello (n)

Write the function `printHELLO` that takes an integer **n** (n is equal or greater than 0) and returns a string with the integers 0 through n, inclusive, except that every power of 2 is replaced by *HELLO*.

Sample Inputs	Expected Outputs
n=3	0HELLOHELLO3
n=7	0HELLOHELLO3HELLO567
n=10	0HELLOHELLO3HELLO567HELLO910
n=1	0HELLO

* Note that there is no whitespace in the string.
* Two hints for a simpler, cleaner implementation: you actually do not need to do any type of division or exponentiation.
* Context: the powers of 2 are naturally of fundamental importance to computer science, since computers store integers as binary numbers (base 2). An understanding of the function 2^n is important for an appreciation of exponential-time algorithms (and its associated exponential explosion).

`paintGallons(length, width, height)`

Write the function **`paintGallons (length, width, height)`** that takes three floats for **`length`**, **`width`**, and **`height`** of the rectangular room, all in feet. Using this information and the nominal coverage for a given paint, compute the amount of paint needed to cover the four walls and ceiling, assuming no doors or windows. The function returns the whole number of gallons of paint needed. Assume one gallon of paint will cover 400 square feet. You need to *round up* the required gallon number.

Sample Code Run

Assume a room of `length 10 feet, width 12 feet and height 8 feet.`
The area of the walls would be:
10 ft x 8 ft = 80 sq. ft. (2 walls this size)
12 ft x 8 ft = 96 sq. ft. (2 walls this size)
The area of the ceiling would be:
10 ft x 12 ft = 120 sq. ft.
The total area would then be:
2 x 80 sq ft. + 2 x 96 sq. ft. + 120 sq. ft = 472 sq. ft.
472 sq. ft / 400 sq. ft per gallon = **1.18 gallons thus 2**

`grader (avg_exams, avg_hw, attendance)`

Write the function "**`grader`**" that takes two floats: **`avg_exams`** and **`avg_hw`**, and one integer **`attendance`** as input and prints "PASS" or "FAIL" depending on the below criteria. The function will use the student's grades and attendance to decide if the student passes or fails.

- Attendance must be greater than 20
- `avg_exams` and the `avg_hw` must both be greater than 70.
- At least one of `avg_exams` or `avg_hw` must be greater than 85.

Example Function Calls

`grader(72,88,22)` prints "PASS"

`grader(66,100,24)` prints “FAIL”

`grader(100,90,18)` prints “FAIL”

TURN IN:

- asgn2.c containing all functions, which compiles and runs on Vulcan, and is documented appropriately
- a Makefile, where (a) the first rule compiles your code, and (b) there is a rule called “run” that will run the code. Therefore, the command “make” will compile the code, and the command “make run” will run the code.
- Independent completion form
- All files should be inside a .zip file

RUBRIC:

- 75%: 15% for each working function (each missing function is -15 points)
- 25%: Document all code
  - Comments must be thorough and correct but not redundant.
  - Explain both what you’re doing and why you’re doing it.
- **NOTE: Code that does not compile on Vulcan will result in an automatic zero. Submissions not in a .zip file, missing a Makefile, or missing an Independent Completion form will result in an automatic zero.**
- Bonus: 5% bonus for user input