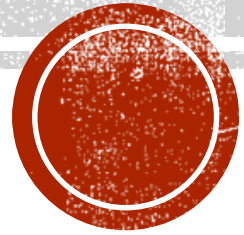
CS 101 LAB #5
FUNCTIONS



Mythili Vutukuru
IIT Bombay

Reference: "C How to Program", Deitel and Deitel, 8th Edition, Chapter 5

GOAL OF LAB #5

- In this lab, you will write programs to understand the following concepts
  - Writing functions
  - Using functions from the math library
  - Using the random number generator
  - Understanding recursion, and the tradeoff between recursion vs iteration
- Submit any two programs testing any two different concepts
- Please write code neatly, with proper indentations and comments

WRITING FUNCTIONS

5.11 *(Rounding Numbers)* Function `floor` may be used to round a number to a specific decimal place. The statement

```
y = floor(x * 10 + .5) / 10;
```

rounds `x` to the tenths position (the first position to the right of the decimal point). The statement

```
y = floor(x * 100 + .5) / 100;
```

rounds `x` to the hundredths position (the second position to the right of the decimal point). Write a program that defines four functions to round a number `x` in various ways

a) `roundToInteger(number)`
b) `roundToTenths(number)`
c) `roundToHundreths(number)`
d) `roundToThousandths(number)`

For each value read, your program should print the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth, and the number rounded to the nearest thousandth.

WRITING FUNCTIONS

5.17 *(Sides of a Right Triangle)* Write a function that reads three nonzero integers and determines whether they are the sides of a right-angled triangle. The function should take three integer arguments and return `1` (true) if the arguments comprise a right-angled triangle, and `0` (false) otherwise. Use this function in a program that inputs a series of sets of integers.

5.18 *(Even or Odd)* Write a program that inputs a series of integers and passes them one at a time to function `isEven`, which uses the remainder operator to determine whether an integer is even. The function should take an integer argument and return `1` if the integer is even and `0` otherwise.

5.26 *(Perfect Numbers)* An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number because 6 = 1 + 2 + 3. Write a function `isPerfect` that determines whether parameter `number` is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the factors of each perfect number to confirm that the number is indeed perfect. Challenge the power of your computer by testing numbers much larger than 1000.

WRITING FUNCTIONS

5.19 *(Rectangle of Asterisks)* Write a function that displays a solid rectangle of asterisks whose sides are specified in the integer parameters `side1` and `side2`. For example, if the sides are 4 and 5, the function displays the following

```
*****
*****
*****
*****
```

5.20 *(Displaying a Rectangle of Any Character)* Modify the function created in Exercise 5.19 to form the rectangle out of whatever character is contained in character parameter `fillCharacter`. Thus if the sides are 5 and 4, and `fillCharacter` is "@", then the function should print the following:

```
@@@@
@@@@
@@@@
@@@@
@@@@
```

WRITING FUNCTIONS

5.28 *(Sum of Digits)* Write a function that takes an integer and returns the sum of its digits. For example, given the number 7631, the function should return 17.

5.29 *(Lowest Common Multiple)* The *lowest common multiple (LCM)* of two integers is the smallest positive integer that is a multiple of both numbers. Write a function `lcm` that returns the lowest common multiple of two numbers.

5.30 *(Quality Points for Student's Grades)* Write a function `toQualityPoints` that inputs a student's average and returns 4 it's 90–100, 3 if it's 80–89, 2 if it's 70–79, 1 if it's 60–69, and 0 if the average is lower than 60.

MATH LIBRARY FUNCTIONS

5.27 *(Roots of a Quadratic Equation)* A quadratic equation is any equation of the form $ax^2 + bx + c = 0$ where a, b, and c are the coefficients of x. The roots of a quadratic equation can be calculated by the formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. If the expression, $b^2 - 4ac$, which is also called the discriminant, is positive then the equation has real roots. If the discriminant is negative, the equation has imaginary (or complex) roots. Write a function that accepts the coefficients of an equation as parameters, checks if the roots are real, and calculates the roots of the equation. Write a program to test this function.

5.45 *(Testing Math Library Functions)* Write a program that tests the math library functions in Fig. 5.2. Exercise each of these functions by having your program print out tables of return values for a diversity of argument values.

RANDOM NUMBERS

5.32 *(Guess the Number)* Write a C program that plays the game of "guess the number" as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1 to 1000. The program then types:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
```

The player then types a first guess. The program responds with one of the following:

```
1. Excellent! You guessed the number!
   Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.
```

If the player's guess is incorrect, your program should loop until the player finally gets the number right. Your program should keep telling the player `Too high` or `Too low` to help the player "zero in" on the correct answer. [*Note:* The searching technique employed in this problem is called binary search. We'll say more about this in the next problem.]

RANDOM NUMBERS

5.33 *(Guess the Number Modification)* Modify the program of Exercise 5.32 to count the number of guesses the player makes. If the number is 10 or fewer, print `Either you know the secret or you got lucky!` If the player guesses the number in 10 tries, then print `Ahah! You know the secret!` If the player makes more than 10 guesses, then print `You should be able to do better!` Why should it take no more than 10 guesses? Well, with each "good guess" the player should be able to eliminate half of the numbers. Now show why any number 1 to 1000 can be guessed in 10 or fewer tries.

5.31 *(Coin Tossing)* Write a program that simulates coin tossing. For each toss of the coin the program should print `Heads` or `Tails`. Let the program toss the coin 100 times, and count the number of times each side of the coin appears. Print the results. The program should call a separate function `flip` that takes no arguments and returns `0` for tails and `1` for heads. [*Note:* If the program realistically simulates the coin tossing, then each side of the coin should appear approximately half the time for a total of approximately 50 heads and 50 tails.]

RANDOM NUMBERS

5.47 *(Craps Game Modification)* Modify the craps program of Fig. 5.14 to allow *wagering*. Package as a function the portion of the program that runs one game of craps. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a `wager`. Use a `while` loop to check that `wager` is less than or equal to `bankBalance`, and if not, prompt the user to reenter `wager` until a valid `wager` is entered. After a correct `wager` is entered, run one game of craps. If the player wins, increase `bankBalance` by `wager` and print the new `bankBalance`. If the player loses, decrease `bankBalance` by `wager`, print the new `bankBalance`, check whether `bankBalance` has become zero, and if so print the message, `"Sorry. You busted!"` As the game progresses, print various messages to create some "chatter" such as, `"Oh, you're going for broke, huh?"` or `"Aw cmon, take a chance!"` or `"You're up big. Now's the time to cash in your chips!"`

RECURSION

5.34 *(Recursive Exponentiation)* Write a recursive function `power(base, exponent)` that when invoked returns

$$base^{exponent}$$

For example, `power(3, 4)` = `3 * 3 * 3 * 3`. Assume that `exponent` is an integer greater than or equal to 1. *Hint:* The recursion step would use the relationship

$$base^{exponent} = base * base^{exponent-1}$$

and the terminating condition occurs when `exponent` is equal to `1` because

$$base^{1} = base$$

5.39 *(Recursive Greatest Common Divisor)* The greatest common divisor of integers `x` and `y` is the largest integer that evenly divides both `x` and `y`. Write a recursive function `gcd` that returns the

greatest common divisor of `x` and `y`. The `gcd` of `x` and `y` is defined recursively as follows: If `y` is equal to `0`, then `gcd(x, y)` is `x`; otherwise `gcd(x, y)` is `gcd(y, x % y)`, where % is the remainder operator.

RECURSION VS. ITERATION

5.35 *(Fibonacci)* The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, …

begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms. a) Write a *nonrecursive* function `fibonacci(n)` that calculates the n^{th} Fibonacci number. Use `unsigned int` for the function's parameter and `unsigned long long int` for its return type. b) Determine the largest Fibonacci number that can be printed on your system.

c) Write a recursive program to calculate fibonacci(n), and observe which of the two (iteration vs recursion) is faster for large n.

COLLATZ CONJECTURE

- Consider the following game. Given a number, we do the following: if the number is even, divide it by 2; else if the number is odd, multiply it by 3 and add 1. Repeat this process again with the new number you get. Keep doing this and you will eventually reach 1. Why this series converges to 1 is not proven yet, but it seems to happen always. This is called the Collatz conjecture. Read up more here: https://en.wikipedia.org/wiki/Collatz_conjecture
- Write a program to take a number as input from the user, and print out the number of steps required for the sequence of numbers to converge to 1 using the method described above. You must solve this question using both recursion and iteration, and verify that both give the same answer.