

Part IV: Functions

- 1.- Write a function `integerPower(base, exponent)` that returns the value of `base` `exponent`. For example, `integerPower(3, 4) = 3 * 3 * 3 * 3`. Assume that `exponent` is a positive, nonzero integer, and `base` is an integer. Function `integerPower` should use `for` to control the calculation. Do not use any math library functions.
- 2.- Write a function `multiple` that determines for a pair of integers whether the second integer is a multiple of the first. The function should take two integer arguments and return `1` (true) if the second is a multiple of the first, and `0` (false) otherwise. Use this function in a program that inputs a series of pairs of integers.
- 3.- Write a program that inputs a series of integers and passes them one at a time to function `even`, which uses the remainder operator to determine if an integer is even. The function should take an integer argument and return `1` if the integer is even and `0` otherwise.
- 4.- Write a function that displays a solid square of asterisks whose side is specified in the integer parameter `side`. For example, if `side` is `4`, the function displays:

```
1  ****
2  ****
3  ****
4  ****
```

- 5.- Modify the function created in Exercise 4 to form the square out of whatever character is contained in the character parameter `fillCharacter`. Thus if `side` is `5` and `fillCharacter` is `"#"` then this function should print.

```
1  #####
2  #####
3  #####
4  #####
5  #####
```

- 6.- Write a function that takes the time as three integer arguments (for hours, minutes, and seconds) and returns the number of seconds since the last time the clock "struck 12." Use this function to calculate the amount of time in seconds between two times, both of which are within one 12-hour cycle of the clock.



Mandatory: (part of CasioCAN project)

- 7.- Write a function to validate time in 24 hour format, the function will receive as parameters the hour, minutes and seconds. It shall return a 1 if the parameters makes a valid time or zero if not


```
1  uint8_t Validate_Time( uint8_t hour, uint8_t minutes, uint8_t seconds );
```



Mandatory: (part of CasioCAN project)

8.- Write a function to validate date in day of the month, month (from 1 to 12) and year (four digits from 1900 to 2100) format, the function will receive as parameters the date. It shall return a 1 if the parameters make a valid date or zero if not. The leap years shall be taken into consideration

```
1 uint8_t Validate_Date( uint8_t days, uint8_t month, uint16_t year );
```

 **Mandatory:** (part of CasioCAN project)

9.- Write a function to calculate the day of the week from a valid day, month and year. It will receive a valid date and return the day of the week from Sunday (0) to Saturday (6)

```
1 uint8_t WeekDay( uint8_t days, uint8_t month, uint16_t year );
```

10.- 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:

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

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

```
1 1. Excellent! You guessed the number!
2 Would you like to play again (y or n)?
3 2. Too low. Try again.
4 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.]

11.- Modify the program of Exercise 10 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.

12.- The greatest common divisor (GCD) of two integers is the largest integer that evenly divides each of the two numbers. Write function `gcd` that returns the greatest common divisor of two integers.

13.- Write a function `qualityPoints` that inputs a student's average and returns 4 if a student's average is 90–100, 3 if the average is 80–89, 2 if the average is 70–79, 1 if the average is 60–69, and 0 if the average is lower than 60.

14.- The use of computers in education is referred to as computer-assisted instruction (CAI). Write a program that will help an elementary school student learn multiplication. Use the `rand` function to produce two positive one-digit integers. The program should then prompt the user with a question, such as `How much is 6 times 7?` The student then inputs the answer. Next, the program checks the student's

answer. If it's correct, display the message "Very good!" and ask another multiplication question. If the answer is wrong, display the message "No. Please try again." and let the student try the same question repeatedly until the student finally gets it right. A separate function should be used to generate each new question. This function should be called once when the application begins execution and each time the user answers the question correctly.