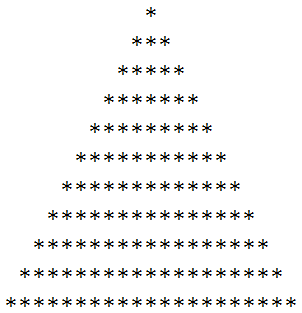
1. Write a program that inputs a positive integer ***N*** andoutputs the *Fibonacci**number FN*
2. Write a program that counts and outputs the number of digits in ***N!***

* Example:

***Please input a positive integer:*** 5

***Number of digits in 5! is: 3***

1. Write a program that inputs three real numbers and outputs whether they can be sides of a triangle.
2. Write a program that calculates and outputs the first ***N*** ***even*** *Fibonacci* numbers, separated by comma and space. ***N*** is entered from the standard input.
3. Write a program that calculates and outputs the total **sum** of the **digits** of the first ***N*** *Fibonacci* numbers.
4. Write a program that checks and prints whether two rectangles overlap. The rectangles’ sides are parallel to ***x*** and ***y*** axes. Each rectangle is determined by the coordinates of its upper-left and lower-right corner. Those coordinates need to be input (you may use integer numbers).
   1. Example output: ***Rectangles overlap***
5. Write a program that outputs the following triangle (***height = 11***):



1. The same triangle as previous, but having ***height*** and ***symbol*** input by the user.
2. Write a program that outputs the first ***N*** natural numbers in a rectangle with a given width (***W***). Numbers are printed in ascending order sequentially in horizontal direction until line is complete and the sequence continues on the next line, etc. The text on the console shall look as a rectangular block (the last line may be incomplete). ***N*** and ***W*** are entered by the user.
3. Write a program that inputs an integer and checks whether it is a prime number.
4. Write a program that calculates the following:

***R = p1 + 2p2 + 3p3 + 4p4… + NpN***

where***pi*** is ***i***-th *prime number.* ***N*** is input by the user.

1. Write a program that prints the length of a ***hailstone sequence***, given the ***seed N***. The seed is entered from the standard input.
2. Write a program to print the number of *digits* in the ***Nth*** ***palindromic prime*** number.
3. Write a program that is capable of inputting very long (i.e. 40-digit) numbers from the standard input and checks whether the number is a *palindrome****.*** Theprogram shall write *“Palindromic”* or *“Not palindromic”* to the standard output.
4. Write a program that calculates the sum of the first ***N*** *Mersenne primes* and prints it to the standard output. ***N*** is input by the user.
5. Write a program that makes simple quality test of the ***rand()*** library function. More precisely, it shall test how *uniformly* the generated random numbers are *distributed*. The guidelines for the program are the following:
   1. The well-known library function ***rand()*** shall be used in a suitable way to generate ***N*** integer numbers between ***0*** and (***M-1)***. ***M*** and ***N*** are entered from the standard input.
   2. ***M*** shall be in the interval ***[10, 50]***. The program shall ask the user to re-enter ***M*** if user input is not in this interval (input sanity check).
   3. ***N*** shall be in the interval ***[10 \* M, 40 \* M]*** in order to get good statistics with good graphical appearance (see below). The user input for ***N*** shall be sanity-checked, too.
   4. During the process of generation of random numbers, statistics of the *distribution* should be kept, counting the number of hits of each random number
   5. After all random numbers are generated and statistics measured, the result (called a ***histogram***) shall be displayed on to the console in a *row chart* format like that (i.e. ***M = 15, N = 600***):



1. Write a program that measures and displays the ***histogram*** of how many times each letter is used in a text entered by the user. Assume Latin input, measure statistics for the 26 Latin letters only. Don’t distinguish between small and capital letters! I.e. ‘**a**’ and ‘**A**’ shall be counted as the same character. Display the histogram in a *row chart* format on the console.
2. Write *functions* ***encode*** and ***decode*** that implement ***Run-length Encoding*** on ***byte*** buffers (not strings!).

Each of the functions shall have the following parameters:

* 1. Pointer to function’s input data
  2. Length of input data (in bytes)
  3. Pointer to where function will store its output (processed) data
  4. Maximum length (limit, size) of the output buffer

Each of the functions shall return the real length of the output (processed) data.

Write also ***main*** function that demonstrates (tests) how ***encode*** and ***decode*** work. Testing idea shall be to take a test sequence, ***encode*** it, ***decode*** it again and compare (***data*** and ***length***)to the original test sequence.

Let the ***main*** function print original, encoded and decoded data and its length for each test sequence in a way that makes it obvious for the user whether the program operates correctly.

Use the following test sequences:

***{0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 8, 8, -56, 0, 0, 0, 0, 0, 0}***

***{0}***

***\* {}*** (empty sequence with length 0)

***{-1, -2, -3, -4, -5}***

1. Write a program that demonstrates ***callback*** functionality using ***function pointers***. Here are the requirements/guidelines for the program:
   1. The program’s ***main()*** shall call a function ***isPrime*** that accepts two parameters: an ***unsigned long int*** parameter and a ***function pointer*** parameter (see below). The ***isPrime*** function shall return ***bool*** (whether the first parameter is a *prime* number or not).
   2. The ***function pointer*** parameter of ***isPrime*** function shall be of type that is suitable to point to functions with declaration(interface):

***void******callback****(****unsigned******long****,* ***unsigned******long****);*

* 1. The program’s ***main()*** shall simply input a number, call ***isPrime*** function and then display “***N is prime***” or ***“N is not prime”*** depending on the return value of ***isPrime*** function
  2. The ***callback*** feature is related to the second argument passed from ***main*** to the ***isPrime*** function. That argument shall be another function that is meant to display progress (i.e. “***checking divisor 73 of 127***” when first argument is ***73*** and the second ***127***). The function pointer shall be called from within ***isPrime*** once every iteration while checking whether the number is prime or not.
  3. In that program, new display functions can be written and “injected” as the second argument to ***isPrime*** functions. This keeps separation between pure calculation (***isPrime***) from display of progress (the display functions with prototype described in ***2.***)
  4. Write at least 1 *callback* function for progress display, with the given prototype that displays progress (preferably *in a single line on the console)* like that:

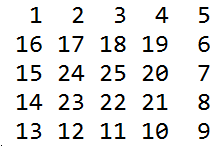
***Checking divisor 2 of 11***

* 1. \*Write a second progress display callback function, that prints progress using row chart, i.e.:

***Progress: [\*\*\*\*\*\*\* ]***

* 1. Test the program with large prime numbers (i.e. 982451653) so that the user can observe the progress displayed by the callback function.

1. **\***Write a program that inputs an integer number ***T*** and a set of ***N*** integers from the console and prints all subsets that have sum greater than ***T.***
2. **\***Write a program that outputs the first ***N*** natural numbers in a square spiral of size ***S*** like that:



1. **\*\***Write a program that inputs an IBAN number and calculates whether it is a valid Bulgarian IBAN.
   1. Example:

***Please input an IBAN:*** BG80 BNBG 9661 1020 3456 78

***The IBAN is valid!***