Instructions

- Use Matlab to create a folder called `lastname-initial-ppcp02` (where as usual, `lastname` is YOUR last name and `initial` is YOUR first initial).
- Enter that folder in Matlab, and do all your work in that folder.
- When you finish, use Matlab to create a `.zip` file of that folder and submit that `.zip` file to the PPCP02 dropbox on D2L.

Problems

1. (10 points) The purpose of this problem is to demonstrate that you can change a program which uses a `for` loop into a program which uses a `while` loop instead.

   I have provided you with a Matlab function called `oddsarefor.m` which will take a vector of integers `vec` and return the product of only its odd positive entries as the value `oddprod`. All negative entries, zero entries, and positive even entries will be ignored while computing `oddprod`. If the vector contains no odd positive entries, then the function will simply return the value `1`.

   You must modify the function so that it uses a `while` loop instead of a `for` loop, but still acts in the same fashion. Save your modified function as a file called `oddsarewhile.m`. Function specifications and some sample function calls are given below.

input parameter	`vec`	vector of integer values
output parameter	`oddprod`	product of only odd positive entries

   sample function calls

`oddsarewhile([1,2,3,4,5,6,7])`	produces `105`
`oddsarewhile([-2,3,0,9,4,-5])`	produces `27`
`oddsarewhile([-8,-1,0,2])`	produces `1`
`oddsarewhile([2,4,6,8,10])`	produces `1`

2. (15 points) Written in expanded form, the usual factorial function is

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 2 \cdot 1.$$

One possible generalization is the **skip factorial** function, denoted by $n!_k$. Like the ordinary factorial, the skip factorial is a product of a decreasing sequence of positive integers, except that instead of decreasing each time by 1, the elements in the product decrease by k. Some examples are shown below:

$$\begin{aligned} 12!_2 &= 12 \cdot 10 \cdot 8 \cdot 6 \cdot 4 \cdot 2 = 46080, \\ 23!_3 &= 23 \cdot 20 \cdot 17 \cdot 14 \cdot 11 \cdot 8 \cdot 5 \cdot 2 = 96342400, \\ 193!_{37} &= 193 \cdot 156 \cdot 119 \cdot 82 \cdot 45 \cdot 8 = 105765791040. \end{aligned}$$

Write a Matlab function called `skiptomyloop.m` which will calculate the value of the skip factorial when given two positive integers `nval` and `kval`. If either of `nval` or `kval` is not an integer, the function should return the flag value `-1`. If either of `nval` or `kval` is nonpositive, the function should return the flag value `-2`.

Even though it would be more efficient in this case to avoid a loop by using a mask, your function **must use a loop**. It may be either a `for` loop or a `while` loop... either type of loop will work for this function. Program specifications and sample function calls are given below.

input parameter	`nval`	a positive integer
input parameter	`kval`	a positive integer
output parameter	`skipfact`	the value of $n!_k$

sample function calls

`skiptomyloop(7,1)`	produces `5040`
`skiptomyloop(23,3)`	produces `96342400`
`skiptomyloop(193,37)`	produces `105765791040`
`skiptomyloop(9.2,3)`	produces `-1`
`skiptomyloop(-9.2,3)`	produces `-1`
`skiptomyloop(-8,3)`	produces `-2`

3. (15 points) Evaluating a polynomial can be an "expensive" thing for a computer. The "cheapest" way to evaluate a polynomial is to rewrite the polynomial
$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$
in nested (or Hörner) form as
$$p(x) = a_0 + x\left(a_1 + x\left(a_2 + \cdots x\left(a_{n-1} + x\left(a_n\right)\right)\right)\right).$$
For example, if $p(x) = 7 - 6x + 4x^2 + 5x^3 - 3x^4$, begin by rewriting it in nested form as follows:
$$p(x) = 7 + x\left(-6 + x\left(4 + x\left(5 + x(-3)\right)\right)\right).$$
Then we can evaluate any value of $p(x)$ from the inside outward in a sequence of basically identical operations. To evaluate $p(2)$ we calculate as follows:
$$\begin{aligned} -3 \cdot 2 + 5 &= -1, \\ -1 \cdot 2 + 4 &= 2, \\ 2 \cdot 2 - 6 &= -2, \\ -2 \cdot 2 + 7 &= 3. \end{aligned}$$
Therefore, $p(2) = 3$. Write a Matlab function called `evalpoly.m` to calculate the value of any polynomial function given a list of its coefficients (in descending order) and a value of x. Program specifications and sample function calls are discussed below.

input parameter	`clist`	a vector of coefficients (in descending order by power of x)
input parameter	`xval`	value(s) at which to evaluate the polynomial
output parameter	`pval`	value(s) of the polynomial at `xval`

Matlab has a built-in function called `polyval` which accomplishes the same job as your function `evalpoly.m`, but you are forbidden to use it in your program. However, you may use it to test your code: `evalpoly(clist,xval)` should produce the same value as the built-in Matlab function `polyval(clist,xval)` for any value `xval` and any vector of coefficients `clist`.

NOTE: if your function can accept a vector of values for `xval` and correctly produce the corresponding vector of `pval`, you may earn up to 18 points