1. Write a function named *eval* that returns the value of the polynomial $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0$.

If you are programming in Java or C#, the function signature is
double eval(double x, int[ ] a)

If you are programming in C or C++, the function signature is
double eval(double x, int a[ ], int len) where len is the number of elements in the array

Examples:

if x is	if the input array is	this represents	eval should return
1.0	{0, 1, 2, 3, 4}	$4x^4 + 3x^3 + 2x^2 + x + 0$	10.0
3.0	{3, 2, 1}	$x^2 + 2x + 3$	18.0
2.0	{3, -2, -1}	$-x^2 - 2x + 3$	-5.0
-3.0	{3, 2, 1}	$x^2 + 2x + 3$	6.0
2.0	{3, 2}	$2x + 3$	7.0
2.0	{4, 0, 9}	$9x^2 + 4$	40.0
2.0	{10}	10	10.0
10.0	{0, 1}	x	10.0

Copy and paste your answer here and click the "Submit answer" button

2. **A non-empty array *a* of length n is called an array of all possiblities** if it contains all numbers between 0 and a.length-1 inclusive. Write a method named **isAllPossibilities** that accepts an integer array and returns 1 if the array is an array of all possiblities, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isAllPossibilities(int[ ] a)

If you are programming in C or C++, the function signature is
int isAllPossibilities(int a[ ], int len) where len is the number of elements in the array

Examples

if the input array is	return
{1, 2, 0, 3}	1
{3, 2, 1, 0}	1
{1, 2, 4, 3}	0 (because 0 not included and 4 is too big)

{0, 2, 3}	0 (because 1 is not included)
{0}	1
{}	0

Copy and paste your answer here and click the "Submit answer" button

3. **An array is called *layered*** if its elements are in ascending order and each element appears two or more times. For example, {1, 1, 2, 2, 2, 3, 3} is layered but {1, 2, 2, 2, 3, 3} and {3, 3, 1, 1, 1, 2, 2} are not. Write a method named **isLayered** that accepts an integer array and returns 1 if the array is layered, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isLayered(int[ ] a)

If you are programming in C or C++, the function signature is
int isLayered(int a[ ], int len) where len is the number of elements in the array

Examples:

if the input array is	**return**
{1, 1, 2, 2, 2, 3, 3}	1
{3, 3, 3, 3, 3, 3, 3}	1
{1, 2, 2, 2, 3, 3}	0 (because there is only one occurence of the value 1)
{2, 2, 2, 3, 3, 1, 1}	0 (because values are not in ascending order)
{2}	0
{}	0

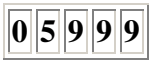
Copy and paste your answer here and click the "Submit answer" button

4. A mileage counter is used to measure mileage in an automobile. A mileage counter looks something like this

0	**5**	**9**	**9**	**8**

The above mileage counter says that the car has travelled 5,998 miles. Each mile travelled by the automobile increments the mileage counter. Here is how the above mileage counter changes over a 3 mile drive.

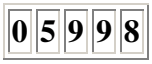
After the first mile



After the second mile

After the third mile

A mileage counter can be represented as an array. The mileage counter



can be represented as the array
int a[ ] = new int[ ] {8, 9, 9, 5, 0}
Note that the mileage counter is "backwards" in the array, a[0] represents ones, a[1] represents tens, a[2] represents hundreds, etc.

Write a function named updateMileage that takes an array representation of a mileage counter (which can be arbitrarily long) and adds a given number of miles to the array. Since arrays are passed by reference you can update the array in the function, you do not have to return the updated array.

You do not have to do any error checking. You may assume that the array contains non-negative digits and that the mileage is non-negative

If you are programming in Java or C#, the function signature is
void updateMileage counter(int[ ] a, int miles)

If you are programming in C or C++, the function signature is
void updateMileage counter(int a[ ], int miles, int len) where len is the number of elements in the array

Examples:

if the input array is	and the mileage is	the array becomes
{8, 9, 9, 5, 0}	1	{9, 9, 9, 5, 0}
{8, 9, 9, 5, 0}	2	{0, 0, 0, 6, 0}
{9, 9, 9, 9, 9, 9, 9, 9, 9, 9}	1	{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{9, 9, 9, 9, 9, 9, 9, 9, 9, 9}	13	{2, 1, 0, 0, 0, 0, 0, 0, 0, 0}

Note that the mileage counter wraps around if it reaches all 9s and there is still some mileage to add.

Hint: Write a helper function that adds 1 to the mileage counter and call the helper function once for each mile

Copy and paste your answer here and click the "Submit answer" button

5. An array is said to be *hollow* if it contains 3 or more zeros in the middle that are preceded and followed by the same number of non-zero elements. Furthermore, all the zeroes in the array must be in the middle of the array. Write a function named *isHollow* that accepts an integer array and returns 1 if it is a hollow array, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isHollow(int[ ] a)

If you are programming in C or C++, the function signature is
int isHollow(int a[ ], int len) where len is the number of elements in the array

Examples:

if the input array is	is hollow?	reason
{1,2,0,0,0,3,4}	yes	2 non-zeros precede and follow 3 zeros in the middle
{1,1,1,1,0,0,0,0,0,2,1,2,18}	yes	4 non-zeros precede and follow the 5 zeros in the middle
{1, 2, 0, 0, 3, 4}	no	There are only 2 zeroes in the middle; at least 3 are required
{1,2,0,0,0,3,4,5}	no	The number of preceding non-zeros(2) is not equal to the number of following non-zeros(3)
{3,8,3,0,0,0,3,3}	no	The number of preceding non-zeros(3) is not equal to the number of following non-zeros(2)

{1,2,0,0,0,3,4,0}	no	Not all zeros are in the middle
{0,1,2,0,0,0,3,4}	no	Not all zeros are in the middle
{0,0,0}	yes	The number of preceding non-zeros is 0 which equals the number of following non-zeros. And there are three zeros "in the middle".

Hint: Write three loops. The first counts the number of preceding non-zeros. The second counts the number of zeros in the middle. The third counts the number of following non-zeros. Then analyze the results.

Copy and paste your answer here and click the "Submit answer" button

6. A positive number n is *consecutive-factored* if and only if it has factors, i and j where i > 1, j > 1 and j = i + 1. Write a function named **isConsecutiveFactored** that returns 1 if its argument is consecutive-factored, otherwise it returns 0.

the function signature is
int isConsectiveFactored(int n)

Examples:

If n is	**return**	**because**
24	1	24 = 2*3*4 and 3 = 2 + 1
105	0	105 = 3*5*7 and 5 != 3+1 and 7 != 5+1
90	1	factors of 90 include 2 and 3 and 3 = 2 + 1
23	0	has only 1 factor that is not equal to 1
15	0	15 = 3*5 and 5 != 3 + 1
2	0	2 = 1*2, 2 = 1 + 1 but factor 1 is not greater than 1
0	0	n has to be positive
-12	0	n has to be positive

Copy and paste your answer here and click the "Submit answer" button

7. A twin prime is a prime number that differs from another prime number by 2. Write a function named **isTwinPrime** with an integer parameter that returns 1 if the parameter is a twin prime, otherwise it returns 0. Recall that a prime number is a number with no factors other than 1 and itself.

the function signature is
int isTwinPrime(int n)

Examples:

number	is twin prime?
5	yes, 5 is prime, 5+2 is prime
7	yes, 7 is prime, 7-2 is prime
53	no, 53 is prime, but neither 53-2 nor 53+2 is prime
9	no, 9 is not prime

8. Write a function named **largestAdjacentSum** that iterates through an array computing the sum of adjacent elements and returning the largest such sum. You may assume that the array has at least 2 elements.

If you are writing in Java or C#, the function signature is
int largestAdjacentSum(int[ ] a)

If you are writing in C or C++, the function signature is
int largestAdjacentSum(int a[ ], int len) where len is the number of elements in a

Examples:

if a is	return
{1, 2, 3, 4}	7 because 3+4 is larger than either 1+2 or 2+3
{18, -12, 9, -10}	6 because 18-12 is larger than -12+9 or 9-10
{1,1,1,1,1,1,1,1,1,1}	2 because all adjacent pairs sum to 2
{1,1,1,1,1,2,1,1,1}	3 because 1+2 or 2+1 is the max sum of adjacent pairs

9. An array is called *zero-balanced* if its elements sum to 0 and for each positive element n, there exists another element that is the negative of n. Write a function named **isZeroBalanced** that returns 1 if its argument is a zero-balanced array. Otherwise it returns 0.

If you are writing in Java or C#, the function signature is
int isZeroBalanced(int[ ] a)

If you are writing in C or C++, the function signature is
int isZeroBalanced(int a[ ], int len) where len is the number of elements in a

Examples:

if a is	return
{1, 2, -2, -1}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{-3, 1, 2, -2, -1, 3}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{3, 4, -2, -3, -2}	0 because even though this sums to 0, there is no element whose value is -4
{0, 0, 0, 0, 0, 0}	1 this is true vacuously; 0 is not a positive number
{3, -3, -3}	0 because it doesn't sum to 0. (Be sure your function handles this array correctly)
{3}	0 because this doesn't sum to 0
{}	0 because it doesn't sum to 0

10. A twin prime is a prime number that differs from another prime number by 2. Write a function named **isTwinPrime** with an integer parameter that returns 1 if the parameter is a twin prime, otherwise it returns 0. Recall that a prime number is a number with no factors other than 1 and itself.

the function signature is
int isTwinPrime(int a[ ])

Examples:

number	is twin prime?
5	yes, 5 is prime, 5+2 is prime
7	yes, 7 is prime, 7-2 is prime
53	no, 53 is prime, but neither 53-2 nor 53+2 is prime
9	no, 9 is not prime

11. Write a function named **largestAdjacentSum** that iterates through an array computing the sum of adjacent elements and returning the largest such sum. You may assume that the array has at least 2 elements.

If you are writing in Java or C#, the function signature is
int largestAdjacentSum(int[ ] a)

If you are writing in C or C++, the function signature is
int largestAdjacentSum(int a[ ], int len) where len is the number of elements in a

Examples:

if a is	return
{1, 2, 3, 4}	7 because 3+4 is larger than either 1+2 or 2+3
{18, -12, 9, -10}	6 because 18-12 is larger than -12+9 or 9-10
{1,1,1,1,1,1,1,1,1}	2 because all adjacent pairs sum to 2
{1,1,1,1,1,2,1,1,1}	3 because 1+2 or 2+1 is the max sum of adjacent pairs

12. An array is called *zero-balanced* if its elements sum to 0 and for each positive element n, there exists another element that is the negative of n. Write a function named **isZeroBalanced** that returns 1 if its argument is a zero-balanced array. Otherwise it returns 0.

If you are writing in Java or C#, the function signature is
int isZeroBalanced(int[ ] a)

If you are writing in C or C++, the function signature is
int isZeroBalanced(int a[ ], int len) where len is the number of elements in a

Examples:

if a is	return
{1, 2, -2, -1}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{-3, 1, 2, -2, -1, 3}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{3, 4, -2, -3, -2}	0 because even though this sums to 0, there is no element whose value is -4
{0, 0, 0, 0, 0, 0}	1 this is true vacuously; 0 is not a positive number
{3, -3, -3}	0 because it doesn't sum to 0. (Be sure your function handles this array correctly)
{3}	0 because this doesn't sum to 0

{}	0 because it doesn't sum to 0

sort a numeric array in ascending order if that is already sorted in descending order.//ans **DA-short.c**

1. accepts an array and return the percent of integers in the array that is divisible by 3.
2. test an array if that is symmetric. {1,2,3,2,1}, {4,5,6,6,5,4} are symmetric arrays.
3. accepts array of integers and returns an integer that represents the biggest difference between any two values in the array.
4. word game-------
   j, q, x, z---10 points
   k, v----5 points
   A to Z and other small case----1 point
   Other----0 points.
   Find the values of {a,r,t,f,z}
5. one word in an anagram of another word if it is a re-arrangement of all the letters of the second word. Write a function that accepts two character arrays and return 1 if they are anagrams of each other else return 0.
6. inputs an array of positive integer and a character array. The integers in the first arrays are indexes into the array of characters. The function should return an array of character containing the character referenced by the integer array.
   e.g.---input—{0,4,7}&{h,a,p,p,i,n,e,s,s} –output—{h,i,s}.
7. input two character arrays and returns the one that is greater in texic order (dictionary order).
   Returns first array---if both are equal
   Returns null ----if either of array is null.
8. write a function that accepts an array of positive integer as its argument and returns an array containing of the odd integers in the input array. The length of the input array should be equal to the number of odd integers that it contains.
   Input----{1,2,3,4,5} ---output---{1,3,5}---should not be---{1,3,5,0,0}
9. write a function that accepts an array of integer and returns the number of distinct integers in the array.
   Input—{1,2,3,10}---output—4
   Input---{5,5,5,5}-----output---1
10. write a function that accepts a character array that contains digits as input and returns its integer equivalent. If a non-digit character is found or the input array is null, or contains no element the function should return -1.
    Input---{1,3,9}—output—139
    Input---{0,4,7}—ouput---47
    Input---{1,+,-2}—output…..-1
    input{}----output…….-1
11. accepts an array and find if the array is happy. If each elements in the array exists exactly two times then the array is happy.
12. balanced array: even elements is even and odd elements is odd

e.g.{1,2,3,4,7,8}

13. odd heavy: array if it contains at least one odd element and every element whose value is odd is greater than every even-valued element{11,4,9,2,8}---11,9 are greater than all even elements\
14. accepts a character array and print the * for each character and also count the repeatable character and mark it by *.
    Input—{a,b,a,b,c,d,a,c,a,b,d,f}---
    output—
    a--****
    b--***
    c--**
    d--**
    f--*
15. Fibonacci series (0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …….)
16. prime no
17. factorial no
18. eliminating duplicate no from array
19. finding out largest and second largest.