Tarea - Ejercicios básicos de programación

La siguiente tarea consta de aplicar los conceptos básicos de programación aprendidos hasta el momento.

Se necesita que usted ejercite su capacidad de crear algoritmos, recuerde que lo que importa es el algoritmo.

La tarea es para realizarla de manera individual. Por favor realizar la tarea a conciencia, es decir, **abstenerse de usar inteligencia artificial** o de pedirle ayuda a alguien más, salvo al guía o a los asistentes de curso, para cuestiones técnicas. Si se sospecha algo “extraño” entonces será etiquetado para futuras evaluaciones.

Para ciertos ejercicios, se necesita que utilice listas de tamaño fijo, es decir arreglos. No debe utilizar las listas dinámicas que le ofrecen las bibliotecas del lenguaje.

“Calibración”

Realice un programa para procesar “documentos de calibración”. Un documento de calibración consta de líneas de texto; cada línea contenía originalmente un valor de calibración específico que fue “descalibrado” por la interferencia de caracteres. En cada línea, el valor de calibración se puede encontrar combinando el primer dígito y el último dígito (en ese orden) para formar un solo número de dos dígitos.

Por ejemplo:

1abc2
pqr3stu8vwx
a1b2c3d4e5f
treb7uchet

Para este ejemplo, los valores de calibración de las 4 líneas son 12, 38, 15, y 77. Sumando todos juntos se obtiene 142.

Entonces, su programa (o método) recibe las líneas descalibradas y produce las líneas calibradas.

“Cubos”

Un juego de adivinanzas trata sobre revelar cubos de colores que están dentro de una bolsa. Por ejemplo, en una partida del juego se pudieron haber revelado 3 cubos rojos, 5 verdes y 4 azules.

Considere los siguientes registros de partidas:

Juego 1: 3 azules, 4 rojos**;** 1 rojo, 2 verdes, 6 azules**;** 2 verdes
Juego 2: 1 azul, 2 verdes**;** 3 verdes, 4 azules, 1 rojo**;** 1 verde, 1 azul
Juego 3: 8 verdes, 6 azules, 20 rojos**;** 5 azules, 4 rojos, 13 verdes**;** 5 verdes, 1 rojo
Juego 4: 1 verde, 3 rojos, 6 azules**;** 3 verdes, 6 rojos**;** 3 verdes, 15 azules, 14 rojos
Juego 5: 6 rojos, 1 azul, 3 verdes**;** 2 azules, 1 rojo, 2 verdes

En el juego 1, se revelan tres conjuntos de cubos de la bolsa (y luego se vuelven a colocar). El primer conjunto consta de 3 cubos azules y 4 cubos rojos; el segundo conjunto consta de 1 cubo rojo, 2 cubos verdes y 6 cubos azules; el tercer conjunto consta de solo 2 cubos verdes.

Se necesita saber qué juegos habrían sido posibles dependiendo de la distribución de cubos. Por ejemplo, si la bolsa contuviera sólo 12 cubos rojos, 13 cubos verdes y 14 cubos azules:

- Los juegos 1, 2 y 5 habrían sido posibles si la bolsa se hubiera cargado con esa configuración.
- El juego 3 habría sido imposible porque en un momento dado se mostró 20 cubos rojos a la vez.
- De manera similar, el juego 4 también habría sido imposible porque se mostró 15 cubos azules a la vez.
- Al sumar los ID de los juegos que hubieran sido posibles, se obtiene 8.

Entonces, determine qué juegos hubieran sido posibles si la bolsa hubiera estado llena con solo 12 cubos rojos, 13 cubos verdes y 14 cubos azules. ¿Cuál es la suma de los ID de esos juegos?

“Raspaditas”

Recibimos varias raspaditas como regalo, pero no logramos descifrar qué ganamos.

Al coger una, parece que cada raspadita tiene dos listas de números separados por una barra vertical (|): una lista de números ganadores y luego una lista de números que tenemos.

Por lo que se puede deducir, hay que averiguar cuáles de los números que tenemos aparecen en la lista de números ganadores. La primera coincidencia hace que la tarjeta valga un punto y cada coincidencia después de la primera duplica el valor en puntos de esa tarjeta.

Por ejemplo:

Raspa 1: 41 48 83 86 17 | 83 86 6 31 17 9 48 53
Raspa 2: 13 32 20 16 61 | 61 30 68 82 17 32 24 19
Raspa 3: 1 21 53 59 44 | 69 82 63 72 16 21 14 1
Raspa 4: 41 92 73 84 69 | 59 84 76 51 58 5 54 83
Raspa 5: 87 83 26 28 32 | 88 30 70 12 93 22 82 36
Raspa 6: 31 18 13 56 72 | 74 77 10 23 35 67 36 11

En el ejemplo anterior, la raspa 1 tiene cinco números ganadores (41, 48, 83, 86 y 17) y ocho números que tenemos (83, 86, 6, 31, 17, 9, 48 y 53). De los números que tenemos, ¡cuatro de ellos (48, 83, 17 y 86) son números ganadores! Eso significa que la carta 1 vale 8 puntos (1 por el primer acierto, luego se duplica tres veces por cada uno de los tres aciertos posteriores al primero).

Entonces, su programa debe analizar las raspas y por cada una producir la siguiente salida (sustituya).

La tarjeta [N] tiene [N] números ganadores ([N, N y N]), por lo que vale N puntos.
La tarjeta [N] tiene ...

Finalmente, diga cuánto es el total de puntos de la pila de raspaditas.

“Ordenar palabras”

Suponga que tiene un conjunto de nodos que representan elementos de un diccionario. Cada nodo tiene una clave única (alfanumérica), que será una palabra dentro del diccionario, y su valor (alfanumérico), que será el significado de la palabra.

Por ejemplo:

Jugar: Hacer algo con alegría con el fin de entretenerse, divertirse o desarrollar determinadas capacidades.
Correr: Ir de prisa.

Comer: Masticar y deglutir un alimento sólido. Ingerir alimento.

El conjunto de nodos está almacenado de manera desordenada, ya que las palabras que se ingresan son totalmente aleatorias.

Debe crear un programa (método) que tome un conjunto de nodos (como el anterior de 3 palabras) y devuelva el conjunto ordenado:

Comer: Masticar y deglutir un alimento sólido. Ingerir alimento.
Correr: Ir de prisa.
Jugar: Hacer algo con alegría con el fin de entretenerse, divertirse o desarrollar determinadas capacidades.

* Java: usar ***compareTo***. Ej: str1.compareTo(str2)
* C++: usar ***compare***. Ej. str1.compare(str2)

“Transmutar AK”

Escriba un algoritmo (método) que reciba una lista cuyos elementos son letras (caracteres). Las letras que puede contener el listado van desde la **'A'** hasta la **'K'**. Sin embargo, algunas de las “celdas” del listado pueden venir “vacías” es decir, con un caracter **''** o **' '** (note el espacio).

Cada vez que su método es invocado, su algoritmo toma la lista y debe devolver otra lista, del mismo tamaño, pero cuyos elementos se transformen siguiendo las siguientes reglas:

- Dos celdas consecutivas que contengan el mismo valor “transmutan” y se produce la siguiente letra. El resultado se almacena en una de las celdas, por lo que la otra debe quedar vacía.
- Una celda que ya fue transmutada, no puede ser transmutada nuevamente.
- El listado final debe acomodar todas las letras al inicio de la lista sin dejar espacios vacíos, estos quedarán al final de la lista.

Si no ocurrió ninguna transmutación ni tampoco hubo un acomodo de letras, entonces su método debe devolver una lista vacía.

Los siguientes ejemplos muestran cuál es la entrada de su método y qué debería éste devolver:

['A', 'A', 'A', 'A'] -> ['B', 'B', ' ', ' ']
['A', 'B', 'C', 'B']-> [ ]
['A', 'C', 'C', 'B']-> ['A', 'D', 'B', ' ']
[' ', ' ', ' ', 'A'] -> ['A', ' ', ' ', ' ']
[' ', ' ', 'E', 'E'] -> ['F', ' ', ' ', ' ']
[' ', 'F', 'F', 'F'] -> ['G', 'F', ' ', ' ']
['A', ' ', 'A', 'C'] -> ['B', 'C', ' ', ' ']
['A', ' ', ' ', 'A'] -> ['B', ' ', ' ', ' ']
['B', ' ', 'B', 'B'] -> ['C', 'B', ' ', ' ']
[' ', ' ', 'I', 'I'] -> ['J', ' ', ' ', ' ']
[' ', ' ', ' ', ' '] -> [ ]