En cada ejercicio se evaluará la eficiencia del código, el uso de identificadores significativos, la indentación, escritura correcta de llaves {} y el uso adecuado de la palabra reservada `const`. Se dispone de tres horas para entregar la prueba y debe realizarse en forma estrictamente individual.

Problema 1: Paréntesis balanceados [55%]

Para un editor de textos orientado a programadores y matemáticos, se necesita una función que reciba una cadena de caracteres de cualquier longitud y determine si todos los paréntesis en ella están balanceados. Se consideran paréntesis a las tres parejas de caracteres: `()`, `{}` y `[]`.

La función recibe por parámetro una cadena de caracteres y la longitud de la misma como un entero largo. La función debe retornar un entero que indica la posición del primer paréntesis no balanceado en la cadena, con el fin de que el editor pueda posicionar el cursor del usuario donde deba realizar la corrección. Si la cadena se termina y quedan paréntesis abiertos, debe retornar la posición del último paréntesis abierto. Si los paréntesis en la cadena están balanceados, la función debe retornar la posición `-1`.

Dado que la función se va a invocar muchas veces mientras el usuario edita un archivo de texto, la función sólo debe hacer un único recorrido por la cadena. La cadena tampoco debe ser modificada por la función, ya que el editor le pasará directamente el texto que edita el usuario, y no se quiere realizar copias de la misma, ya que consumirían tiempo y espacio.

Los desarrolladores del editor requieren que usted demuestre que la función haga bien su trabajo. Provea un pequeño programa que lea líneas de la entrada estándar, y para cada una de ellas indique en la salida estándar la posición donde se encuentre el primer paréntesis no balanceado, o el texto "ok" si están balanceados, como se muestra a la derecha.

1. [5%] Implementa un programa de prueba correctamente.
2. [20%] Implementa correctamente la función que encuentra el primer paréntesis no balanceado o el último que quedó abierto.
3. [10%] Realiza un único recorrido por la cadena sin modificarla. No genera fallos aún con cadenas muy extensas.
4. [15%] Rastree la memoria del programa. Su dibujo debe ilustrar el estado del programa en el momento inmediato en que ha detectado que la línea 3 del ejemplo de entrada no tiene los paréntesis balanceados. Debe indicar el número de línea en su programa donde esto ocurre. Si hace su dibujo a lápiz, no borre memoria modificada, sino tache para dejar rastro de lo ocurrido.
5. [5%] Documenta el código explicando el algoritmo implementado.
6. [10% opcional] Escriba una rutina que lea líneas de un archivo o la entrada estándar y al mismo tiempo cuente la cantidad de caracteres leídos. Sugerencia: dado que las cadenas que puede leer son de longitudes arbitrarias no conocidas, haga a su función leer en un búfer al que pueda hacer crecer con `realloc(ptr,size)`.

Ejemplo de entrada:

```
:(practica para examen):

x=-(-2 - -3)^4 / [(5 - 2]^10)
valor de x: [seleccion unica]
( ) x=(-3)^-10
( } x=3^-10
) ) x=1/3^10
( ) x=-(3)^-10
( ) x={-(3)^10,x>0|(-3)^10,x<0
( ) no se :{

[=](){a?(a-(--b)):(b-(--a))}
es? [seleccion multiple]
[ ] puntero a funcion
[ [ funcion lambda
[ ] macro (preprocesador
[ ] plantilla
...
```

Ejemplo de salida:

```
ok::(practica para examen):
ok:
24:x=-(-2 - -3)^4 / [(5 - 2]^10)
ok:valor de x: [seleccion unica]
ok:( ) x=(-3)^-10
02:( } x=3^-10
00:) ) x=1/3^10
ok:( ) x=-(3)^-10
06:( ) x={-(3)^10,x>0|(-3)^10,x<0
11:( ) no se :{
ok:
ok:[=](){a?(a-(--b)):(b-(--a))}
ok:es? [seleccion multiple]
ok:[ ] puntero a funcion
02:[ [ funcion lambda
10:[ ] macro (preprocesador
ok:[ ] plantilla
```

Problema 2: Validador de Sudoku [45%]

Implemente un validador de Sudoku, el cual podría ser usado para saber si juegos digitalizados de las primeras revistas no tienen errores, o para ayudar a personas a crear nuevos retos de este juego. El Sudoku tiene sólo tres reglas:

1. Cada fila debe tener los números de 1 a 9 sin que se repitan.
2. Cada columna debe tener los números de 1 a 9 sin que se repitan.
3. Cada caja o región (resaltada en borde negro grueso) debe tener los números de 1 a 9 sin que se repitan.

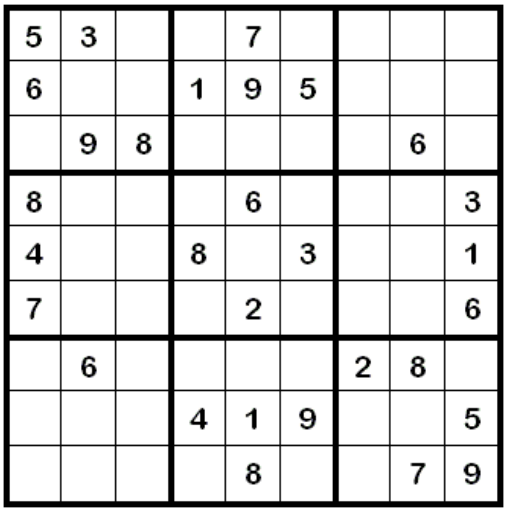
El programa debe leer tableros de Sudoku de la entrada estándar. Los tableros podrían estar parcialmente llenos, donde las celdas llenas se indican con el número que contienen y las vacías con el carácter punto (`'.'`).

Su programa debe determinar si el tablero de Sudoku leído de la entrada estándar es válido, y en tal caso imprimir el texto "`valid`" en la salida estándar. Si el tablero no es válido, se debe indicar la celda donde se detectó el error, antecedido por una de las letras: `'r'` para fila, `'c'` para columna, ó `'b'` para caja. En el ejemplo de la derecha se detectó que el valor de la celda en la fila 5 y columna 4 genera una caja (`b`) inválida. En caso de que hayan múltiples errores en un tablero, sólo se reporta el primero de ellos, siguiendo el orden de evaluación de las tres letras indicadas, y el orden de lectura izquierda-derecha y arriba-abajo.

Nota: un tablero de Sudoku válido (parcialmente lleno) no es necesariamente resoluble. Su programa sólo debe validar que las celdas llenas cumplan con las tres reglas anteriores, indiferentemente de si el tablero es resoluble o no.

Los tableros de Sudoku en la entrada deben constar de 9 filas de 9 caracteres separadas por cambios de línea. Sin embargo, por errores de digitación o de reconocimiento de caracteres a partir de documentos históricos, podrían aparecer caracteres no esperados (por ejemplo un `'0'` en lugar de un `'8'`) o tableros incompletos. En tal caso, se debe imprimir la celda donde se detecta el error usando la notación anterior.

1. [5%] Diseña una solución y la documenta en el código. Modulariza la solución en funciones que podrían ser reutilizadas.
2. [5%] Lee tableros de Sudoku correctamente de la entrada estándar. Reporta errores de lectura.
3. [20%] Implementa correctamente la validación del tablero de Sudoku, aplicando las tres reglas a las celdas llenas (con valores).
4. [5%] Reporta errores de validez en la salida estándar usando la notación solicitada y en el orden de evaluación solicitado.
5. [10%] Se quiere que la función que realiza la evaluación del tablero sea reutilizable. Esta función debe indicar al llamador si el tablero es válido o no, y en caso de no serlo, dónde se encuentra el error. Invocar a esta función no debe reportar errores en la salida estándar (el cuál debe ser trabajo de otra función). Dado que el programa debe apegarse a las buenas prácticas de programación y de eficiencia, tampoco debe usar variables globales.



Ejemplo de entrada:

```
53..7....
6..195...
.98....6.
8...6...3
4..6.3..1
7...2...6
.6....28.
...419..5
....8..79
```

Ejemplo de salida:

```
b5,4
```