INSTITUTO TECNOLÓGICO DE COSTA RICA
Escuela de Computación - Bachillerato en Ingeniería en Computación
IC-1802 Introducción a la Programación - Examen Parcial
Prof. Mauricio Avilés

Total de puntos: 50 Tiempo: 3 horas

Instrucciones:
Lea cuidadosamente los casos proporcionados, y basándose en la teoría y práctica vista en clases, aplique su criterio para responder las preguntas indicadas de manera individual.
Este examen debe resolverse mediante el uso de la computadora.
Cualquier comportamiento deshonesto durante la prueba será sancionado con la nota mínima.
Debe entregarse un único archivo .ZIP con las soluciones de los problemas propuestos. La entrega se hacer por medio del TEC-Digital, en las evaluaciones del curso.

Nombre: _____

Carné: _____

Las respuestas de este examen serán resultado de mis decisiones individuales. No usaré, recibiré, ni ofreceré ayuda no autorizada. No copiaré de otros exámenes, ni permitiré que nadie copie parte alguna de este examen. No realizaré ninguna trampa ni procedimiento deshonesto. Juro por mi honor que todo lo anterior es cierto.

Firma: _____

Parte I – Problemas

Para cada uno de los problemas presentados debe cumplirse lo siguiente:

- Las subrutinas que utilicen recursión deben dividirse en función principal y auxiliar.
- No escriba código adicional al solicitado.
- No lea entradas o realice impresiones a menos que el enunciado se lo solicite explícitamente.
- Las subrutinas utilizadas deben lanzar excepciones en caso de que las restricciones no se cumplan.
- La documentación interna no será evaluada.

1. **Elementos repetidos en una lista (10 puntos)**

   Escriba una función llamada **repetidos(L)** que reciba como argumento una lista L no vacía de elementos y retorne la lista de elementos que están repetidos en la lista. Si el elemento se encuentra repetido más de una vez en L, sólo debe aparecer una vez en la lista de resultado. La función debe lanzar una excepción en caso de que no se cumpla alguna de las restricciones. No se permite el uso del tipo set, ni de ningún tipo de iteración (for o while). Utilice recursión de cola. No es necesario crear un programa principal para utilizar esta función.

   Ejemplos:

```
>>> repetidos([2, 5, 6, 8, 2])
[2]
>>> repetidos(['a', 'b', 'c', 'd'])
[]
>>> repetidos([1, 4, 6, 1, 4, 7, 9, 1])
[1, 4]
```

   Chequeo de restricciones: 2 puntos
   Función principal y auxiliar: 1 punto
   Resto del algoritmo: 7 puntos
   Total: 10 puntos

2. **Quick sort descendente (10 puntos)**

   Escriba una función llamada quick_sort(L) que reciba como argumento una lista L de elementos de cualquier tipo y retorne la lista ordenada en orden descendente. La función debe implementar el algoritmo Quicksort de forma que el resultado sea en orden descendente. No es necesario comprobar que todos los elementos sean del mismo tipo, sólo que la entrada sea una lista. No se permite el uso del método sort, ni ordenarla de forma ascendente para luego invertir el orden de los elementos. Debe utilizarse recursión. No es necesario crear un programa principal para utilizar esta función.

   Ejemplos:

```
>>> quick_sort([2, 5, 6, 8, 2])
[8, 6, 5, 2, 2]
>>> quick_sort (["gendo", "shinji", "rei", "asuka", "misato"])
["shinji", "rei", "misato", "gendo", "asuka"]
>>> quick_sort ([1, 4, 6, 1, 4, 7, 9, 1])
[9, 7, 6, 4, 4, 1, 1, 1]
```

   Chequeo de restricciones: 2 puntos
   Función principal y auxiliar: 1 punto
   Resto del algoritmo: 7 puntos
   Total: 10 puntos

3. **Validación de Sudoku (30 puntos)**

Este problema consiste en crear un programa robusto que lea el contenido de una matriz sudoku de un archivo en formato JSON e imprima en pantalla si corresponde a un sudoku válido o, en caso de que no, cuáles son los errores que contiene.

El programa debe iniciar solicitando un nombre de archivo al usuario. El programa debe intentar abrir dicho archivo y extraer los datos en formato JSON. Junto con el examen se proveen 10 archivos de prueba, 5 válidos y 5 inválidos.

No es obligatorio utilizar orientación a objetos, pero puede hacerlo si le resulta útil.

El programa debe ser robusto, utilice la estructura **try/except** para atrapar cualquier error que pueda ocurrir en el mismo. Los mensajes de error no deben ser exactamente iguales a los ejemplos, lo importante es que el programa no se interrumpa por un error.

Junto con el enunciado se provee una función llamada **imprimirSudoku(M)** que se encarga de imprimir en pantalla la matriz tal y como se muestra en los ejemplos. Puede utilizarla para centrarse en la lógica del problema. También se provee la función **esMatrizSudoku(M)** que es una función booleana que dice si M es una matriz numérica de 9x9 con valores enteros de 1 a 9.

A continuación, se muestran ejemplos de la salida del programa.

Ejemplo 1 – Sudoku válido:

```
Escriba el nombre del archivo a procesar: valido1.txt
   0 1 2 3 4 5 6 7 8
  ┌─────┬─────┬─────┐
0 │ 9 6 3 │ 4 2 8 │ 1 5 7 │
1 │ 2 5 8 │ 3 7 1 │ 6 9 4 │
2 │ 1 4 7 │ 9 6 5 │ 3 2 8 │
  ├─────┼─────┼─────┤
3 │ 4 8 9 │ 5 3 2 │ 7 6 1 │
4 │ 3 1 6 │ 7 4 9 │ 2 8 5 │
5 │ 5 7 2 │ 8 1 6 │ 4 3 9 │
  ├─────┼─────┼─────┤
6 │ 8 3 4 │ 6 5 7 │ 9 1 2 │
7 │ 6 9 1 │ 2 8 4 │ 5 7 3 │
8 │ 7 2 5 │ 1 9 3 │ 8 4 6 │
  └─────┴─────┴─────┘
Sudoku válido.
```

En el ejemplo anterior puede observarse que se cumple satisfactoriamente que cada fila, cada columna y cada región (recuadros de 3 x 3) contienen los números del 1 al 9 sin repetirse.

Ejemplo 2 – Sudoku inválido:

```
Escriba el nombre del archivo a procesar: invalido1.txt
   0 1 2 3 4 5 6 7 8
  ┌─────┬─────┬─────┐
0 │ 2 3 4 5 1 6 9 8 7 │
1 │ 8 7 9 3 2 4 1 6 5 │
2 │ 1 5 6 9 7 8 2 3 4 │
  ├─────┼─────┼─────┤
3 │ 4 9 4 7 8 3 6 5 1 │
4 │ 6 1 5 2 4 9 3 7 8 │
5 │ 7 8 9 6 5 1 4 2 9 │
  ├─────┼─────┼─────┤
6 │ 9 6 1 8 3 7 5 2 2 │
7 │ 5 4 8 1 6 2 7 9 3 │
8 │ 3 2 7 4 3 5 8 1 6 │
  └─────┴─────┴─────┘
Fila 3:       repetidos [4]
Fila 5:       repetidos [9]
Fila 6:       repetidos [2]
Fila 8:       repetidos [3]
Columna 2:    repetidos [4, 9]
Columna 4:    repetidos [3]
Columna 7:    repetidos [2]
Región 3:     repetidos [4, 9]
Región 7:     repetidos [3]
Región 8:     repetidos [2]
Sudoku no válido.
```

En este ejemplo puede observarse cómo el programa señala las filas, las columnas y las regiones en las que se encuentran elementos repetidos, pero además dice cuáles son dichos elementos.

Ejemplo 3:

```
Escriba el nombre del archivo a procesar: prueba.txt
Error al abrir y procesar el archivo. [Errno 2] No such file or
directory: 'prueba.txt'
```

En este ejemplo se observa cómo el programa muestra un mensaje de error en caso de que el archivo no se encuentre. El programa debe imprimir un mensaje y salir sin errores.

Ejemplo 4:

```
Escriba el nombre del archivo a procesar: hola.txt
Error al abrir y procesar el archivo. Expecting value: line 1 column 1
(char 0)
```

En este último ejemplo se observa cómo el programa muestra un mensaje de error en caso de que el archivo sea de formato JSON. El programa debe imprimir un mensaje y salir sin errores.

Manejo de archivo:	**4 puntos**
Manejo de errores:	**5 puntos**
Verificación de filas:	**7 puntos**
Verificación de columnas:	**7 puntos**
Verificación de regiones:	**7 puntos**
Total:	**30 puntos**