Prog. Básica - Laboratorio 7
Extractos de ejercicios de examen

NOTAS:

1. Cuando uno de los ejercicios que se proponen carece de programa de prueba o de plantilla, significa que lo tenéis que hacer vosotros **desde cero**. Además, el hecho de que se proporcionen algunos casos de prueba no significa que no pueda faltar alguno. Tendréis que **añadir los casos de prueba que falten**.
2. En este laboratorio no hay que completar ningún informe. Solamente tendréis que subir a eGela los ficheros fuente (.adb exportados de Eclipse y *.py* copiados/pegados de CodeSkulptor) **comprimidos en un fichero** (.zip preferiblemente) que deberá ajustarse a las **reglas de nombrado** exigidas hasta ahora (por ejemplo, JSaez_PEtxebarria_lab7.zip).
3. Se presupone que los ejercicios son correctos, es decir, que no tienen errores de compilación y que funcionan correctamente. Esto significa que las soluciones de los ejercicios no puntúan nada por ser correctas, pero penalizan si tienen errores o no se ajustan a lo que se pide. Una vez que la solución es correcta, lo que se evalúa (lo que puntúa) son:
    a. **Los casos de prueba**: ¿Se han contemplado todos los casos de prueba, desde los generales a los más críticos? Se dan algunos pero faltan muchos otros.
    b. **La eficiencia**: ¿Se utilizan los "chivatos" cuando hace falta? ¿No hay asignaciones innecesarias? ¿No hay condicionales que no deberían ejecutarse?, ¿Se definen solamente los parámetros o variables necesarios?, etcétera.
    c. **La claridad**: ¿El código está tabulado? ¿Los nombres de las variables ayudan a entender el código? ¿Hay un único return al final de la función?, etcétera.

1º ejercicio: Eliminar repetidos

Este ejercicio únicamente hay que resolverlo en ADA.
Ficheros: *def_datos.ads* y *escribir_lista.adb* (No hay que modificarlos).
Plantillas: *esta.adb*, *eliminar_repetidos.adb* y *prueba_eliminar_repetidos.adb*.

Hacer un subprograma que, dada una lista de enteros, obtenga otra lista equivalente que no contenga elementos repetidos.

2º ejercicio: Lotería

Este ejercicio únicamente hay que resolverlo en ADA.
Fichero: *datos_loteria.ads* (No hay que modificarlo).
Plantillas: *calcular_aciertos.adb, total_dinero.adb* y *prueba_total_dinero.adb*.

La siguiente estructura de datos contiene los números de boleto premiados en una edición de la lotería primitiva. Se premia a los boletos de 3, 4, 5 y 6 aciertos, y por cada una de las 4 categorías de aciertos se tiene una lista de los boletos con premio en esa categoría. Un boleto puede aparecer más de una vez en una lista de premios.

```
Max_Premios_por_Categoria: constant Integer := 100;
type Tabla_Boletos is array(1.. Max_Premios_por_Categoria) of Integer;

subtype Valor_entre_0_y_Tope is Integer range 0 .. Max_Premios_por_Categoria;
type Lista_Boletos_Premiados is record
    Cuantos: Valor_entre_0_y_Tope;
    Boletos: Tabla_Boletos;
end record;

type T_Datos_Premios is array(3 .. 6) of Lista_Boletos_Premiados;
```

Diseñar y codificar el subprograma que, a partir de un número de boleto, el dinero recaudado y una estructura de tipo T_Datos_Premios, calcule cuánto dinero le corresponde a ese boleto en premios. La asignación de dinero se calcula de la siguiente manera: la mitad del dinero recaudado se destina a premios. Esta cantidad se distribuye a partes iguales entre cada una de las cuatro categorías, y dentro de cada categoría se distribuye de manera proporcional entre sus acertantes.

3º ejercicio: Resta de enteros grandes

Este ejercicio únicamente hay que resolverlo en PYTHON.
Plantilla: *resta.py*.

Se van a utilizar vectores de tamaño 100 para representar, dígito a dígito, valores enteros muy grandes. Por ejemplo, el valor 1223334445678556111 se representaría de la siguiente manera:

0	1	2	...	88	89	90	91	92	93	94	95	96	97	98	99
0	0	0	...	4	4	5	6	7	8	5	5	6	1	1	1

Escribir un subprograma que obtenga la resta (dígito a dígito y controlando el valor de las llevadas) de dos enteros grandes utilizando esta clase de representación.

4º ejercicio: Compilador

Este ejercicio únicamente hay que resolverlo en ADA.
Fichero: *datos.ads* (No hay que modificarlo).
Plantillas: *balanceado.adb y prueba_balanceado.adb*.

Un error de compilación muy común en los programas es el que proviene de abrir paréntesis o llaves y dejarlos sin cerrar o cerrar paréntesis o llaves que no se han abierto. Se pide hacer un subprograma en Ada que sea capaz de detectar este tipo de errores.

Por ejemplo:

```
if (a > b) then { b := a+n ) ; z := x+y }
                              ¡¡¡Error!!!
```

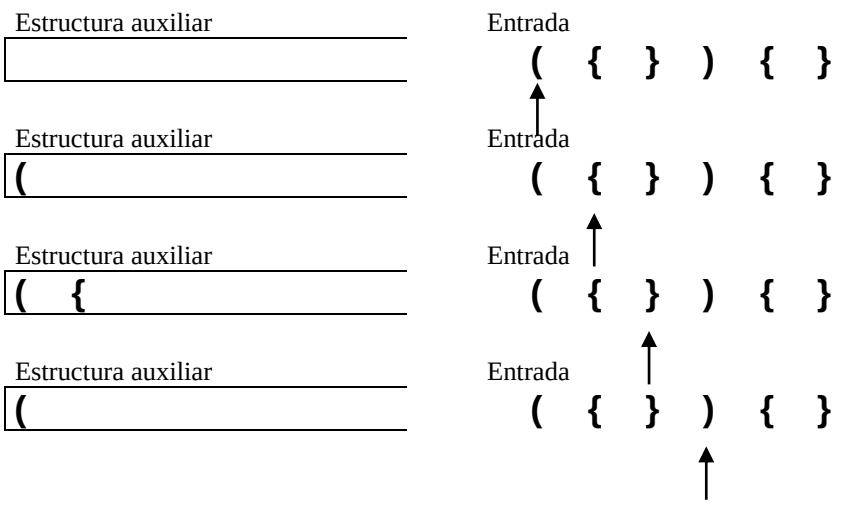
El proceso a seguir es usar un vector auxiliar y recorrer la cadena de entrada carácter a carácter, de manera que:

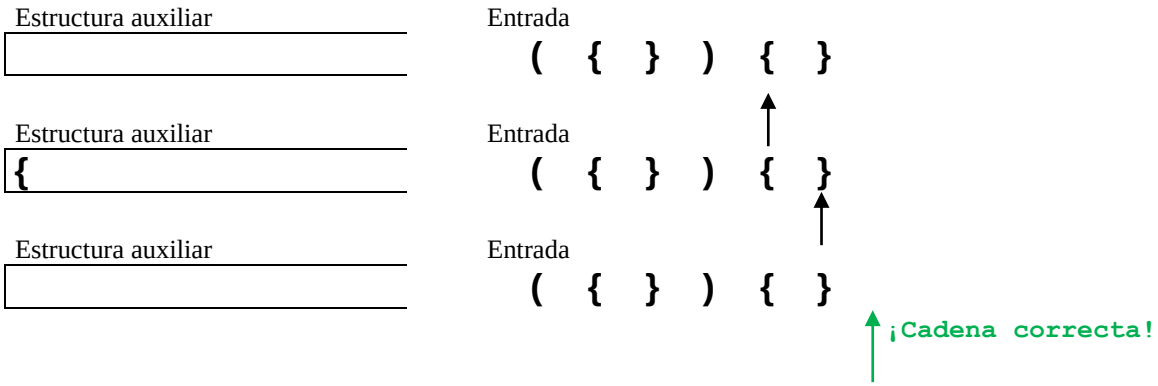
- Los caracteres que no son paréntesis o llaves se ignoran.
- Si se encuentra un paréntesis o una llave de apertura, entonces se introduce en la estructura auxiliar.
- Si se encuentra un paréntesis o una llave de cierre, entonces debe coincidir con el último elemento introducido en la estructura auxiliar, si lo hubiera. En este caso se debe eliminar el paréntesis o llave de apertura de la estructura auxiliar, ya que ha sido emparejado correctamente. Si no fuera el caso, o la estructura auxiliar estuviera vacía, entonces se ha detectado un error.

El proceso acaba cuando se encuentra uno de los siguientes casos:

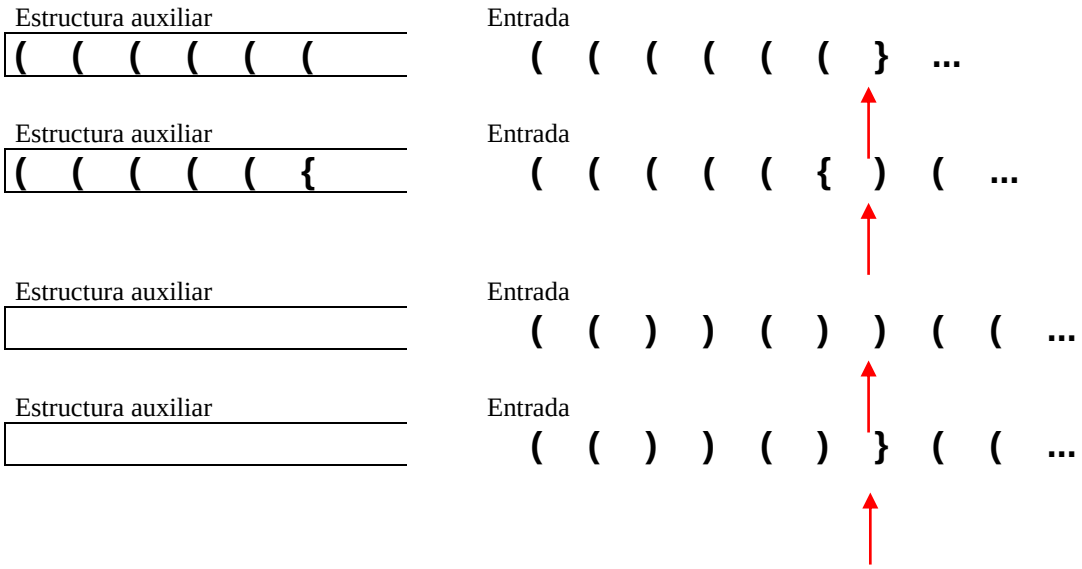
a) Acaba el recorrido de la cadena de entrada con éxito

Ejemplo: **if(a= {b*c }) then {a:=c }**





b) Se ha encontrado una de las siguientes configuraciones erróneas



También es una configuración errónea toda aquella en la que al finalizar el recorrido de la cadena de entrada se tiene aún algún elemento en la estructura auxiliar.

5º ejercicio: Amplitudes

Este ejercicio únicamente hay que resolverlo en PYTHON.
Plantilla: *prueba_calcular_amplitud_max.py*.

Escribir un subprograma que tome como entrada los datos de una función matemática representada por una lista de enteros. Esta lista estará formada por subsecuencias ascendentes y descendentes que representan amplitudes de segmentos, donde un segmento está formado por una serie ascendente de valores seguida de una serie descendente. El subprograma devolverá la longitud del segmento con mayor amplitud.

Ejemplo:

Dada la siguiente lista, la función devolverá 10, que es el tamaño del mayor segmento que contiene (posiciones 7 a 16). El otro segmento (posiciones 1 a 7) tiene una amplitud menor. Nótese que la lista contiene también una serie ascendente (posiciones 16 a 20) que no se considera segmento porque no desciende.