Prog. Básica - Laboratorio 8
Ejercicios de examenes

NOTAS:

- 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**.
- En este laboratorio no hay que completar ningún informe. Solamente tendréis que subir a eGela los ficheros fuente (.adb exportados de Eclipse) **comprimidos en un fichero** (.zip preferiblemente) que deberá ajustarse a las **reglas de nombrado** exigidas hasta ahora (por ejemplo, JSaez_PEtxebarria_lab8.zip).
- 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: Ordenar pares e impares

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

Se pide escribir un subprograma que, dado un vector de enteros, recoloque sus valores de tal modo que los números pares queden al principio y los impares al final.

Por ejemplo, dado el siguiente vector:

10	3	13	4	6	3	5	2	9	7	8	18	12	2

El resultado obtenido podría ser:

10	4	6	2	8	18	12	2	3	13	3	5	9	7

NOTAS:

- NO importa en qué orden queden los valores pares entre sí y los impares entre sí, siempre que todos los pares estén seguidos al principio del vector y todos los impares estén juntos al final.
- En este sentido, es imprescindible que la solución sea eficiente. En concreto solamente deberá recorrerse el vector una única vez, por lo que se sugiere utilizar dos índices: uno que se moverá de la primera posición en adelante, y otro que se moverá desde la última posición hacia atrás.

2º ejercicio: Elecciones

Este ejercicio únicamente hay que resolverlo en ADA.

Ficheros: *tipos.ads, asignar_maximos.adb, calcular_cocientes_partido,adb, calcular_todos_cocientes.adb, distribuir_votos.adb, escribir_escainos.adb, escribir_lista_partidos_votos.adb, escribir_tabla_cocientes_partidos.adb, escribir_tabla_Dhont.adb, obtener_partidos_con_mas_de_un_3_por_ciento.adb, repartir_escainos.adb* (No hay que modificarlos).
Ficheros de apoyo: *construir_lista_partidos_votos_caso1.adb, prueba_asignar_maximos.adb, prueba_calcular_cocientes_partido.adb, prueba_calcular_todos_los_cocientes.adb, prueba_distribuir_votos.adb, prueba_obtener_partidos_con_mas_de_un_3_por_ciento.adb* y *prueba_repartir_escainos.adb* (Se pueden modificar/adaptar si se considera necesario).
Plantillas: *calcular_escainos.adb* y *prueba_calcular_escainos.adb.*

En el fichero *tipos.ads* se facilitan las siguientes definiciones de tipos de datos para representar los resultados de las elecciones en la comunidad autónoma:

Constantes establecidas

```
Max_Num_Partidos: constant integer :=10;
Num_Escainos: constant integer := 25;
```

Rangos

```
subtype T_Rango_Partidos is Integer range 0..Max_Num_Partidos;
subtype T_Nombre is String(1..15);
```

Lista de partidos con sus votos (Lista_Partidos_Votos)

```
type T_Info_Partido_Votos is record
    Nombre  : T_Nombre;
    Votos   : Natural;
end record;
type T_Tabla_Partido is array (1..Max_Num_Partidos) of T_Info_Partido_Votos;
type T_Lista_Partidos_Votos is record
    Num_Partidos:   T_Rango_Partidos;
    Tabla_Partidos: T_Tabla_Partido;
end record;
```

Lista de partidos con sus escaños (T_Lista_Escainos)

```
type T_Info_Partido_Escainos is record
    Nombre : T_Nombre;
    Escainos: Natural;
end record;
type T_Tabla_Escainos is array (1..Max_Num_Partidos) of T_Info_Partido_Escainos;
type T_Lista_Escainos is record
    Num_Partidos  : T_Rango_Partidos;
    Tabla_Escainos: T_Tabla_Escainos;
end record;
```

Se quiere implementar un subprograma que permita calcular cuántos escaños le corresponde a cada partido político que se ha presentado a las elecciones, cuya cabecera es la siguiente:

```
procedure calcular_escainos (
      LPV_Bizkaia, LPV_Araba, LPV_Gipuzkoa: in T_Lista_Partidos_Votos;
      Resultado: out T_Lista_Escainos)
-- Pre: las 3 variables de entrada contienen los resultados de las elecciones en
--       cada una de las 3 provincias.
-- Post: el resultado es el número de escainos de cada partido. Este resultado se
--        calcula asignando los 25 escainos a repartir en cada provincia, y
--        acumulándolos finalmente por cada partido.
```

Por ejemplo, dados los siguientes resultados electorales:

Bizkaia		**Araba**		**Gipuzkoa**	
PNV/EA	264774	PNV/EA	51601	PNV/EA	147498
PSE-PSOE	151347	PP	43765	EHAK	78088
PP	113867	PSE-PSOE	43765	PP	70577
EHAK	65431	EHAK	14180	PSE-PSOE	51163
EB-IU	36258	EB-IU	8395	EB-IU	20278
ARALAR	10187	ARALAR	2541	ARALAR	15273

Después de aplicar la asignación de escaños por el método D'Hont, el número de escaños por provincia sería:

Bizkaia		**Araba**		**Gipuzkoa**	
PNV/EA	11	PNV/EA	8	PNV/EA	10
PSE-PSOE	6	PP	7	EHAK	5
PP	5	PSE-PSOE	7	PP	5
EHAK	2	EHAK	2	PSE-PSOE	3
EB-IU	1	EB-IU	1	EB-IU	1
				ARALAR	1

Y el resultado final, esto es, el número total de escaños por partido, quedaría:

Total escaños	
PNV/EA	29
PSE-PSOE	16
PP	17
EHAK	9
EB-IU	3
ARALAR	1

Notas:

- El resultado no tiene por qué estar ordenado por número de escaños, aunque en el ejemplo se muestre así.
- Para resolver este ejercicio, se facilita el subprograma **repartir_escainos** junto con otros subprogramas que utiliza. No hay que implementar ninguno de ellos., Estos subprogramas se encargan de aplicar el método D'Hont para repartir los 25 escaños de acuerdo a los votos conseguidos en una determinada provincia.

3º ejercicio: Asistencia a tutorías

Este ejercicio únicamente hay que resolverlo en ADA.
Fichero: *datos.ads* (No hay que modificarlo).
Plantillas: *posicion_de_dni.adb, eliminar_repetidos.adb, ordenar.adb, obtener_alumnos_y_tutorias_totales.adb* y *prueba_obtener_alumnos_y_tutorias_totales.adb.*

Se dispone de las siguientes definiciones de datos para almacenar la información relativa a qué alumnos han acudido a tutorías durante el cuatrimestre: el calendario guarda la asistencia a tutorías de cada uno de los días laborables del cuatrimestre. Por cada día laborable se registra la fecha y los números de DNI de los estudiantes que han acudido a tutorías ese día.

```
NA: constant Integer := 60; -- Número máximo de alumnos
subtype Rango_Alumnos is Integer range 1..NA;

type T_DNIs is array (Rango_Alumnos) of Positive;

type Info_dia is record
   Fecha: String(1..10);  -- en formato "aaaa/mm/dd"
   Cuantos_alumn: Natural;
   DNIs: T_DNIs;
end record;

NDL: constant Integer := 200; -- Número máximo de días laborables
subtype Rango_dias_laborables is Integer range 1..NDL;

type T_dias is array (Rango_Dias_laborables) of Info_dia;

type Info_calendario is record
   Cuantos_Dias: Natural;
   Dias: T_dias;
end record;
```

También se dispone de las siguientes definiciones de datos para guardar la información correspondiente a cuántas veces ha acudido a tutorías cada alumno:

```
type Info_tutorias is record
   DNI: Positive;
   Tutorias_Totales: Natural;
end record;

type T_tutorias is array(Rango_Alumnos) of Info_tutorias;

type Lista_tutorias_alumnos is record
   Cuantas: Natural;
   Tutorias: T_tutorias;
end record;
```

Se quiere implementar un subprograma que, a partir de la información almacenada en el calendario, obtenga el listado del alumnado ordenado por número de asistencias a tutorías. Su cabecera es la siguiente:

```
procedure obtener_alumnos_y_tutorias_totales (Calendario: in Info_calendario;
                                              Lista: out Lista_tutorias_alumnos)
-- Pre: Calendario contiene información sobre la asistencia a tutorías (por cada día
--      laborable del cuatrimestre, la fecha y los números de DNI de quienes han acudido
--      a tutorías).
-- Post: Lista está ORDENADA por el número total de tutorías a las que han asistido los
--       estudiantes.
--       Si un alumno ha acudido más de una vez a tutorías el mismo día, solamente se le
--       contabilizará una vez por ese día para computar el número total de asistencias.
```

Por ejemplo, supongamos que el calendario dispone de la siguiente información:

- El día 2018/06/18 han acudido a tutorías los estudiantes cuyos números de DNI son 11, 22, 33, 11, 22, 33, 44 y 55.
- El día 2018/06/19 han acudido a tutorías los estudiantes con DNI 55, 22 y 55.
- El día 2018/06/20 no ha acudido nadie a tutorías.
- El día 2018/06/21 han acudido a tutorías los estudiantes con DNI 22 y 33.
- El día 2018/06/22 han acudido a tutorías los estudiantes cuyos números de DNI son 44, 33, 22, 44 y 11.

Como resultado de la ejecución del subprograma anterior se obtendrá un elemento de tipo Lista_tutorias_alumnos, que en este ejemplo debería almacenar la siguiente información.

- DNI 22 ha acudido 4 días.
- DNI 33 ha acudido 3 días.
- DNI 55 ha acudido 2 días.
- DNI 44 ha acudido 2 días.
- DNI 11 ha acudido 2 días.

Nótese que cuando un mismo DNI aparece más de una vez el mismo día, solamente se ha contabilizado una vez para ese día. Además, dado que los tres últimos DNIs (11, 44 y 55) tienen el mismo número de tutorías totales, no importa en qué orden aparezcan en la lista.