LAB04

Objetivos:

- Manejo de la instrucción condicional if
- Manejo de la instrucción condicional case
- Manejo de la instrucción iterativa loop
- Manejo de la instrucción iterativa while
- Manejo de la instrucción iterativa for

Lab 4. Instrucciones condicionales e iterativas

4.1. Lista de ejercicios

Estos ejercicios se deben resolver con las instrucciones que se han explicado hasta el momento en clase (asignación, instrucción nula, llamada a otros procedimientos o funciones, instrucciones condicionales e instrucciones iterativas).

4.1.1. Función condicional

Escribe una función *f* que, dados dos valores enteros *x* y *pos*, devuelva otro valor entero de acuerdo con la siguiente fórmula:

$$f(x, pos) = \begin{cases} x & si\ pos\ es\ par \\ 2x & si\ pos\ es\ impar\ y\ x < 5 \\ 2x - 9 & si\ pos\ es\ impar\ y\ x \geq 5 \end{cases}$$

4.1.2. Nota final de un alumno

La nota definitiva de un alumno se calcula a partir de las tres notas de evaluación continua. Implementa el procedimiento que calcule la nota final de un alumno a partir de las notas y un código que indica el cálculo que se quiere realizar sobre las tres notas. Si en algún momento, la nota no se puede calcular, entonces se devuelve el valor -1. Ten en cuenta, que, como notas que son, las tres notas pueden ser iguales o diferentes y su valor está entre el mínimo (0.0) y el máximo (10.0). La lista de códigos asociados a los distintos cálculos que se quieren hacer es la siguiente:

Código	Cálculo a realizar
1	La nota máxima
2	La nota menor de las tres
3	La nota intermedia (ni menor ni mayor)
4	La tercera nota (la última)
5	La media aritmética de todas las notas
6	La media geométrica de todas las notas

La media geométrica es de n valores (positivos) es la raíz enésima de su producto:

$$media\ geométrica = \sqrt[n]{x_1 \cdot x_2 \cdot \cdots x_n}$$

4.1.3. Trigonométrica a polar

Implementa el procedimiento polar que, dado un número complejo en formato trigonométrico (a, b), nos devuelve el mismo número complejo en forma polar. Para definir exactamente el ángulo, hace falta estudiar el signo de a y b y determinar el cuadrante en el que se encuentra (como si fuera una coordenada en el plano) y devolver un ángulo positivo en radianes.

La biblioteca *Ada.Numerics.Elementary_Functions* que ofrece la arcotangente (*arctan*), si la necesitas.

$$modulo = \left|\sqrt[2]{a^2 + b^2}\right|$$

Al analizar las fórmulas, ten cuidado con aquellos valores que hacen que el cálculo no sea posible (¿qué pasa si a contiene el valor 0?) pero que se puede obtener un resultado válido como solución. Se sugiere revisar las fórmulas por si hubiera algún caso más con las mismas características (que lo hay).

$$argumento = \begin{cases} arctan\left(\frac{b}{a}\right) & si\ a > 0\ y\ b > 0 \\ 2\pi + arctan\left(\frac{b}{a}\right) & si\ a > 0\ y\ b < 0 \\ \pi + arctan\left(\frac{b}{a}\right) & si\ a < 0 \\ 0 & si\ a > 0\ y\ b = 0 \\ +\frac{\pi}{2} & si\ a = 0\ y\ b > 0 \\ +\frac{3\pi}{2} & si\ a = 0\ y\ b < 0 \\ 0 & si\ a = 0\ y\ b = 0 \end{cases}$$

4.1.4. Resultado de la ONCE

Crea un algoritmo que, dados dos números de cuatro cifras (el número de cupón premiado y el número de cupón que hay que comprobar), obtenga el premio que le corresponde al cupón a comprobar, sabiendo que si tiene los cuatro dígitos iguales (y en el mismo orden) le corresponden 100000 euros, si coinciden los tres últimos dígitos le corresponden 50000 euros y si son los dos últimos el premio son 3 euros. El resto de los casos no tiene premio asociado.

4.1.5. Contar y sumar múltiplos

Dados dos números naturales N1 y N2, haz un procedimiento que devuelve la suma de los *múltiplos de 3* entre N1 y N2 (ambos inclusive). N1 es estrictamente menor que N2. Haz que el programa cuente también los múltiplos de 3.

Ejemplo: Para N1=5 y N2=15 dicha suma es 42 (es decir, 42=6+9+12+15) y la cuenta es 4 (los cuatro números mencionados).

4.1.6. Factorial

Crea una función que, dado un numero positivo, devuelva el número factorial de ese número. La fórmula que define el número factorial de un número es la siguiente:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 2 \cdot 1$$

4.1.7. Decimal a Binario

Crea un procedimiento que, dado un número decimal, lo convierta a un número binario haciendo divisiones sucesivas entre dos (si no conoces el método, busca en Internet, que hay muchas explicaciones sobre el tema).

Ejemplo: Para los datos 127 y 1022, los resultados deben ser 111_1111 y 11_1111_1110, respectivamente.

4.1.8. Pregunta de tipo test

Crea un procedimiento que, dada una pregunta y cuatro opciones haga la pregunta, escriba las cuatro opciones (numerándolas de 1 a 4) y lea la respuesta que da el usuario y lo devuelva como valor.

Por ejemplo, con la pregunta *"¿De qué color es el caballo blanco de Santiago?"* y las opciones *"Alazán" (rojizo), "Blanco", "Melado" (color miel)* y *"Picazo" (blanco y negro),* el procedimiento escribirá los mensajes y leerá el número correspondiente a la respuesta indicada, que será el valor que devuelva.

```
¿De qué color es el caballo blanco de Santiago?
1. Alazán
2. Blanco
3. Melado
4. Picazo
Elige tu respuesta [1-4]: 3
```

4.1.9. Segundo anterior

Crea el procedimiento *segundo_anterior* que, dada una hora válida (*horas*, *minutos* y *segundos*), calcule qué hora era un segundo antes de ese momento. Por ejemplo, la hora anterior a 10:30:40 es 10:30:39.

4.1.10. Ecuación de segundo grado

Crea el procedimiento *solucionar* que nos devuelva las soluciones de una ecuación de segundo grado. Solo estamos interesados en las soluciones reales (no las imaginarias). El procedimiento debe detectar cuántas soluciones reales tiene la ecuación (0, 1 ó 2) y devolver los valores adecuados para ello. Si no hay soluciones, los valores de las soluciones no tendrán sentido. Si solo hay una solución, solo se dará una solución y si hay dos las dos. La fórmula de las soluciones de una ecuación de segundo grado $ax^2 + bx + c$ es la que aparece más abajo.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Para calcular la raíz cuadrada, en la biblioteca de funciones *Ada.Numerics.Elementary_functions,* está la función *Sqrt.*

Fíjate que hay casos en los que existe solución para la ecuación, pero no usando la fórmula. Por ejemplo, cuando a=0 (la ecuación es $bx + c$), si se usa la fórmula general, al dividir entre 0 se producirá un error, sin embargo, la solución de la ecuación es una y se corresponde con la siguiente fórmula:

$$x = -\frac{c}{b}$$

Hay otro caso más que puede dar problemas y tiene que ver con la raíz cuadrada y la fórmula que incluye.

¿Cuándo se puede calcular una raíz cuadrada? Esta vez no se indica la solución.

4.1.11. Añadir duración a una tarea

Crea un algoritmo que, dada la hora de comienzo de una tarea y la duración en segundos de ésta, calcule la hora en que finalizará la tarea. La duración de la tarea puede ser de varias horas, pero nunca superior a media jornada (14400s). Las tareas incompletas continúan en la siguiente jornada. La jornada laboral es de 8:00 a 16:00.

4.1.12. Secuencia de Collatz

Escribe un procedimiento que muestre en pantalla la secuencia de Collatz de un número entero positivo dado, y devuelva, además, la longitud de la secuencia y la suma de sus números. La secuencia de Collatz de un número se construye de la siguiente forma: Si el número es par, se divide entre dos; si es impar, se multiplica por tres y se le suma uno; y así sucesivamente hasta llegar al número 1.

Ejemplo: Para el dato 18, la secuencia de Collatz es <18, 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1> La secuencia tiene 21 números y la suma es 357.

4.1.13. Fibonacci

Crea una función que calcule el número de la serie de Fibonacci. La serie de Fibonacci comienza con los números 0 y 1 y el resto de los elementos se corresponde con la suma de los dos anteriores.

F_i	Valor	Explicación
F_0	0	Por definición
F_1	1	Por definición

F_2	1	$F_0+F_1=0+1$
F_3	2	$F_1+F_2=1+1$
F_4	3	$F_2+F_3=1+2$
F_5	5	$F_3+F_4=2+3$
F_6	8	$F_4+F_5=3+5$
F_7	13	$F_5+F_6=5+8$
F_8	21	$F_6+F_7=8+13$
F_9	34	$F_7+F_8=13+21$
F_{10}	55	$F_8+F_9=21+34$
F_{11}	89	$F_9+F_{10}=34+55$
F_{12}	144	$F_{10}+F_{11}=55+89$
F_{13}	233	$F_{11}+F_{12}=89+144$
…	…	…

4.1.14. Divisores propios

Crea un procedimiento que indique cuántos divisores propios tiene un número entero N. Contamos entre los posibles divisores a aquellos números positivos entre 1 y N–1 cuya división es exacta.

4.1.15. Día anterior

Crea un procedimiento que, modifique una fecha dada (*día*, *mes* y *año*) y coloque en su lugar el día anterior. Comienza considerando que todos los años no son bisiestos. Luego añade la condición de que el año puede ser bisiesto. Los años bisiestos son los que son múltiplos de 4 y no ser múltiplos de 100. Los múltiplos de 400 son una excepción, ya que, aun siendo múltiplos de 100, son bisiestos.

4.1.16. Estación del año

Crea una función que dados dos números enteros día y mes, indique a qué estación del año corresponde dicha fecha. Se considera que la primavera va desde

el 21 de marzo hasta el 20 de junio; el verano se extiende desde el 21 de junio al 20 de septiembre; el otoño dura desde el 21 de septiembre hasta el 20 de diciembre; y el invierno ocurre entre el 21 de diciembre y el 20 de marzo.

4.1.17. Estación del año enumerada

Crea una función equivalente a la del ejercicio anterior que, en lugar de devolver un STRING, devuelva un valor del tipo T_Estacion (ya definido en el fichero de especificación).

4.1.18. Número perfecto

Crea la función *es_perfecto* que, dado un número entero positivo N, indique si N es perfecto. Se considera que un número es perfecto si el número es igual a la suma de sus divisores propios. Por ejemplo, 28 es un número perfecto porque sus divisores son 1, 2, 4, 7 y 14. Y la suma de ellos 1+2+4+7+14=28.

4.1.19. Contenido del texto

Construye un procedimiento que lea del teclado un texto que termina en punto (.) y devuelva (1) cuántos espacios, (2) cuántas letras y (3) cuántos dígitos hay en dicho texto. Cualquier otro carácter encontrado se ignora en la cuenta. Por ejemplo, si el texto es ***"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*** *Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."* el resultado debería ser el correspondiente hasta el primer punto encontrado (lo

que aparece en **negrita,** terminado en *aliqua*) y sería: 102 letras, 18 espacios y 0 dígitos.

La biblioteca *Ada.Characters.Handling* ofrece funciones para saber si un carácter es de control, un gráfico, una letra (mayúscula o minúscula), una mayúscula, alfanumérica, un carácter especial…

4.1.20. El método de Newton-Raphson

Es un algoritmo para encontrar aproximaciones de los ceros o raíces de una función real. También puede ser usado para encontrar el máximo o mínimo de una función, encontrando los ceros de su primera derivada. Sean dos funciones f(x) y su derivada f'(x). La idea consiste en ir calculando sucesivos valores de x_i siguiendo la fórmula siguiente, hasta que el error sea menor que una cierta cantidad.

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} \qquad Error = \frac{|x_i - x_{i-1}|}{|x_i|}$$

Ejemplo: Aplica el método para calcular x, de $f(x) = \cos(x) - x^3$, partiendo de x_0=0.5 con un error menor a 0.001. Necesitarás definir una función f(x), la función derivada $f'(x) = -sen(x) - 3x^2$.

x_{i-1}	x_i	error	Comentario
0.5	1,11214	0,55042	error>0.001 buscar nuevo x
1,11214	0,90967	0,22257	error>0.001 buscar nuevo x
0,90967	0,86548	0,00206	error>0.001 buscar nuevo x
0,86726	0,86548	0,00206	error>0.001 buscar nuevo x
0,86548	0,86547	0,00000	Parar aquí

4.1.21. Organizar las vacaciones

María quiere irse de vacaciones. Tiene exactamente V días para ello. Durante el año, se ha comprometido a ir con sus amigas de la Universidad y con las del conservatorio. Ambos grupos de amigas son flexibles, y tienen disponibilidad de fechas para hacerlo. Crea un programa que dados los días de vacaciones que tiene María, los días que pasará con las del conservatorio C y los que pasará con sus amigas de la universidad, U, diga si es posible que María haga las vacaciones con todas sus amigas como tenía previsto, sabiendo que María no tiene el don de la ubicuidad y la picaresca no es una característica de María, por lo que no iría simultáneamente al mismo sitio con sus dos grupos de amigas.

4.1.22. Mayor o menor

Hacer un programa que saque números aleatorios entre 1-100 y haga el juego Mayor menor para acertar el número pensado.

El ordenador debe pensar el número N y preguntar por el número a adivinar. Como respuesta a cada intento, se responde **Mas** si el número indicado es menor que N; **Menos** si el número indicado es mayor que N; y **Enhorabuena, ¡has acertado!** si el numero coincide con el pensado.

4.1.23. Mayor o menor con límite de intentos

Rehaz el programa anterior para que se permita un número máximo de intentos dependiendo del rango en el que se encuentra el número aleatorio. Por ejemplo, para números entre 1-100 dejar 4 intentos, para números entre 1-1000 dejar 9 intentos y para 1-10000 dejar 15 intentos. En el caso de llegar al número máximo de intentos, el programa debe indicar **Lo**

siento, has superado el número máximo de intentos.

4.1.24. Función de siete segmentos

Crea una función llamada *digito_7s,* que dados siete valores booleanos que representan los segmentos de un display de siete segmentos (ver 3.1.40, para ver una descripción de en qué consisten este tipo de displays), devuelva el dígito del 0 al 9 que le corresponde. Si no correspondiera a ningún dígito, devolverá el valor -1.

4.1.25. Veintiuno (21)

Este juego es de dos jugadores que alternativamente van restando una, dos o tres unidades. El jugador que llega a 0 es el que pierde. Implementa un programa que juegue a este juego. Como datos se pasa el nombre del oponente y se indica si comienza el ordenador o el humano. El programa debe ir pidiendo las sucesivas cantidades a ir eliminando hasta decidir el ganador de la partida, que es el resultado que se devuelve (un booleano: gana jugador o no).

Un ejemplo de funcionamiento es el siguiente (para dos jugadores comenzando el ordenador):

#	JUGADOR 1	ORDENADOR	SALDO
			21
1		-1	20
2	-2		18
3		-2	16
4	-1		15
5		-3	12
6	-3		9
7		-1	8
8	-1		7

9		-3	4
10	-2		2
11		-1	1
12	¡¡PIERDE!!		

Una estrategia ganadora con dos jugadores es la siguiente: intentar dejar como número restante de elementos un número múltiplo de 4 (pasar de 21 a 20, de 19 a 16…). El primer jugador que lo haga es el que va a ganar la partida (a no ser que haga un movimiento que le haga salirse de la estrategia). Si el ordenador no está en la estrategia, se quitará 1, para intentar tener más posibilidades de que el otro jugador no siga la estrategia.

4.1.26. Nim mísere

Este juego se parece al anterior, pero el punto de partida son tres montones con un número diferente de valores. El juego se hace entre dos jugadores en orden alternativo. Cada jugador puede eliminar todas las piezas que quiera, pero de un único montón. El que quita el último elemento del último montón, pierde. El procedimiento recibe el nombre del jugador, la cantidad en cada uno de los montones y quién empieza. A partir de ahí, se comienza el juego, se toman las jugadas alternativas y se decide quién es el ganador (un booleano: el ordenador o el jugador).

4.1.27. Estrategia para el juego de dados *Pig*

Pig es un juego de dados en el que la decisión principal es la de apostar las ganancias previas frente a potenciales ganancias mayores. Varios jugadores lanzan un dado en varios turnos. En un turno, un jugador lanza un dado hasta que obtiene un 1, se planta o alcanza la puntuación objetivo.

Si el jugador obtiene un 1, su puntuación de ese turno es 0 y se pasa al siguiente jugador.

Si el jugador obtiene cualquier otra puntuación, se añade a la puntuación previa de ese turno y el turno continua y lanza de nuevo el dado.

Si el jugador se planta, al total obtenido hasta el momento se añade la puntuación del turno y se pasa el turno al siguiente jugador.

El primer jugador que consigue más de 100 puntos gana.

Por ejemplo, Daniela comienza un turno sacando un 5. Daniela podría plantarse y conseguir 5 puntos, pero elige volver a tirar. Daniela saca un 2 y podría plantarse con un total de 7 puntos en el turno, pero elige volver a tirar de nuevo. Daniela saca un 1 y debe finalizar un turno sin puntos. El siguiente jugador lanza los dados y obtiene las siguientes tiradas: 4-5-3-5-6, ante lo que decide plantarse, añadiendo a su puntuación total los 23 puntos de este turno.

Crea cuatro funciones plantarse_20, plantarse_25, plantarse_4 o plantarse_sprint, que devuelva un booleano sugiriendo qué hacer en el juego Pig según las estrategias que se comentan en la siguiente sección. Un resultado *true* significa plantarse y *false* continuar lanzando los dados. Aunque en todas las estrategias no se usan todos los datos, por hacer las funciones consistentes, siempre se le pasan los mismos datos:

- Numero de turno
- Puntuación total hasta el momento de los jugadores (el que tiene el turno y el oponente)
- Puntuación en el turno actual

Estrategia óptima

Se ha hecho un estudio para calcular cuál es la estrategia óptima del juego para dos jugadores dependiendo de las puntuaciones que tiene cada uno de ellos y la cantidad obtenida hasta el momento en el turno. Para un humano es difícil de realizar el cálculo por lo que se sugieren alternativas cercanas:

- **Plantarse en 20**. El jugador tira los dados hasta obtener 20 o más. En ese momento se planta. (8% de desventaja frente a la estrategia óptima).
- **Plantarse en 25.** Se ha descubierto que esta estrategia es más eficaz (4.2% de desventaja frente a la estrategia óptima).
- **Cuatro turnos puntuando.** Se planta uno en 25 en el primer turno. En los siguientes turnos uno se planta cuando se alcanza la cantidad necesaria para llegar a 100 entre el número de turnos restantes (3.3% de desventaja frente a la estrategia óptima).
- **Sprint final o seguir el paso.** Si alguno de los jugadores ha obtenido 71 o más, lanzar dados para ganar. Si no, plantarse en 21 más la diferencia entre puntuaciones dividida entre 8 (0.9% de desventaja frente a la estrategia óptima).

4.1.28. Jugada de póker

Crea un programa que, a partir de 5 cartas de la baraja francesa, decida cuál es la jugada de póker más alta que se puede hacer con esas cinco cartas. La baraja de francesa tiene cuatro palos: dos de color negro (♠ - Picas y ♣ - Tréboles) y dos de color rojo (♥ - Corazones y ♦ - Diamantes). Dentro de cada palo, las cartas tienen 13 posibles valores que van del 2 al 10, y cuatro valores más: JQKA en orden creciente de valor.

Las jugadas de póker que se pueden crear son las siguientes en orden descendente de valor:

- **Escalera real.** Son las cinco cartas de más valor de mismo palo, como A♥ K♥ Q♥ J♥ 10♥. No hay jugada superior a ésta. Si hubiera varios con esta jugada, habría un empate entre ellos.
- **Escalera de color.** Son cinco cartas del mismo palo que van contiguas como Q♥ J♥ 10♥ 9♥ 8♥. En caso de empate entre dos escaleras, gana aquella que tenga la carta más alta.
- **Póker.** Es una mano que contiene cuatro cartas del mismo valor y otra distinta. Por ejemplo, 9♣ 9♠ 9♦ 9♥ J♥ es un póker de nueves.
- **Full.** También conocido como *full hand.* Es una mano que contiene un trío y una pareja. Por ejemplo, 3♣ 3♠ 3♦ 6♣ 6♥ es un full de treses y seises. En caso de empate, se mira primero el valor de los tríos y luego el de la pareja. Así, 8♠ 8♦ 8♥ 7♦ 7♣ es mejor que 4♦ 4♠ 4♣ 9♦ 9♣, que a su vez es mejor que 4♦ 4♠ 4♣ 5♣ 5♦. Las manos que solo difieren en los palos se consideran un empate, como las dos siguientes: K♣ K♠ K♦ J♣ J♠ y K♣ K♥ K♦ J♣ J♥.
- **Color.** Es una mano que contine 5 cartas todas del mismo palo, como K♣ 10♣ 7♣ 6♣ 4♣. Para desempatar tiene importancia la carta más alta, después la segunda más alta, después la tercera más alta, luego la cuarta y finalmente la quinta más alta. Por ejemplo, K♦ J♦ 9♦ 6♦ 4♦ es mejor que Q♣ J♣ 7♣ 6♣ 5♣, que, a su vez, es mejor que J♥ 10♥ 9♥ 4♥ 2♥, que, a su vez, es mejor que J♠ 10♠ 8♠ 6♠ 3♠, que, a su vez, es mejor que J♥ 10♥ 8♥ 4♥ 3♥, que, a su vez, es mejor que J♣ 10♣ 8♣ 4♣ 2♣. Las manos de color que solo

difieren en los palos, como 10♦ 8♦ 7♦ 6♦ 5♦ y 10♠ 8♠ 7♠ 6♠ 5♠, se consideran empate.

- **Escalera.** Es una mano, como 7♣ 6♠ 5♠ 4♥ 3♥, que contiene cinco cartas en secuencia en la que alguna tiene un palo diferente a las demás (si no sería escalera de color). El ejemplo previo es una escalera de siete (la carta más alta). El as se puede considerar como una carta alta (como en A♦ K♣ Q♣ J♦ 10♠) o como una carta baja (como en 5♣ 4♦ 3♥ 2♥ A♠), pero no puede actuar como alta y baja a la vez (como en Q♠ K♠ A♣ 2♥ 3♦, en este caso no se considera escalera, sino la carta más alta). Entre dos escaleras, se desempata según el valor de la carta más alta, por lo que J♥ 10♥ 9♣ 8♠ 7♥ es mejor que 10♠ 9♠ 8♣ 7♥ 6♠, que es mejor que 6♣ 5♠ 4♥ 3♠ 2♦. Si las manos solo difieren en los palos se considera empate.
- **Trío.** Es una mano, como 2♦ 2♠ 2♣ K♠ 6♥ (trío de doses) que contiene tres cartas del mismo valor y dos cartas más con diferentes valores cada una de ellas. El valor de un trío define el valor de la jugada. En caso de empate en el trío se usan los valores del resto de las cartas. Por ejemplo, 6♥ 6♦ 6♠ Q♣ 4♠ es mejor que 3♦ 3♠ 3♣ K♠ 2♠, que es mejor que 3♦ 3♠ 3♣ J♣ 7♥, que es mejor que 3♦ 3♠ 3♣ J♠ 5♦. Si solo difieren en el palo, se considera empate. Así, las dos siguientes manos empatan: 9♠ 9♥ 9♦ 10♦ 8♥ y 9♣ 9♠ 9♥ 10♦ 8♦.
- **Doble pareja.** Es una mano que tiene dos cartas con el mismo valor, otras dos cartas con otro valor diferente y una quinta carta con un valor distinto. Por ejemplo, J♥ J♣ 4♣ 4♠ 9♥ es una doble pareja de jotas y cuatros. Ante dos dobles parejas, se mira primero el valor de la pareja más alta y luego el de la más baja y finalmente el de la carta suelta. Así, 10♦ 10♠ 2♠ 2♣ K♣ es mejor que 5♣ 5♠ 4♦ 4♥ 10♥,

que es mejor que 5♣ 5♠ 3♣ 3♦ Q♠, que es mejor que 5♣ 5♠ 3♣ 3♦ J♠. Si dos dobles parejas solo difieren únicamente en sus palos, como K♦ K♠ 7♦ 7♥ 8♥ y K♣ K♠ 7♣ 7♥ 8♣, se considera un empate.

- **Pareja.** Es una mano que tiene dos cartas con el mismo valor, y tres cartas más con valores diferentes entre ellas y con ese valor, por ejemplo, 4♥ 4♠ K♠ 10♦ 5♠ es una pareja de cuatros. Entre dos parejas se mira primero el valor de las parejas (mejor cuanto más alto sea) y luego el valor de las tres cartas restantes, mirando primero la carta más alta, luego la segunda más alta, y finalmente la tercera más alta. Así, 9♣ 9♦ Q♠ J♥ 5♥ es mejor que 6♦ 6♥ K♠ 7♥ 4♣, que es mejor que 6♦ 6♥ Q♥ J♠ 2♣, que es mejor que 6♦ 6♥ Q♠ 8♣ 7♦, que es mejor que 6♦ 6♥ Q♦ 8♥ 3♠. Si solo hay diferencias en los palos y no en los valores se considera empate, como en 8♠ 8♦ 10♥ 6♣ 5♠ y 8♥ 8♣ 10♣ 6♠ 5.
- **Carta más alta.** A veces conocida como *nada.* Son las manos que no entran en ninguna de las categorías anteriores, como K♥ J♥ 8♣ 7♦ 4♠. Para comparar manos, se mira la carta con el mayor valor, luego la segunda con mayor valor, y así sucesivamente hasta la quinta carta con mayor valor. Por ejemplo, K♠ 6♣ 5♥ 3♦ 2♣ es mejor que Q♠ J♦ 6♣ 5♥ 3♣, mejor que Q♠ 10♦ 8♣ 7♦ 4♠, que es mejor que Q♥ 10♥ 7♣ 6♥ 4♠, que es mejor que Q♣ 10♣ 7♦ 5♣ 4♦, mejor que Q♥ 10♦ 7♠ 5♠ 2♥. Si solo hay diferencias en los palos y no en los valores se considera empate, como en 10♣ 8♠ 7♠ 6♥ 4♦ y 10♦ 8♦ 7♠ 6♣ 4.

4.2. Números aleatorios en Ada

Para obtener números enteros aleatorios en Ada su usa la biblioteca *Ada.Numerics.Discrete_Random*. A continuación, se presenta un pequeño trozo de código tomado del manual que se explica en los siguientes párrafos.

```
1   with ada.Text_IO; use Ada.Text_IO;
2   with ada.numerics.discrete_random;
3   procedure aleatorio is
4     type T_rang_Azar is
5       new Integer range 1..100;
6     package Azar_Int is new
7    ada.numerics.discrete_random(T_rang_Azar);
8     use Azar_Int;
9     gen : Generator;
10    num : T_rang_Azar;
11  begin
12    reset(gen);
13    num := random(gen);
14    put_line(Num'Img);
15  end aleatorio;
```

Primero, se necesita indicar que se va a usar dicha biblioteca de programas (línea 2).

Después, hace falta definir un (sub)tipo que define el rango de números enteros aleatorios (línea 4). El rango debe ser continuo. No se puede elegir como rango 1..10 y 20..30 (se podría elegir 1..30 y luego controlar que no salgan números entre 11..19). En el ejemplo se buscan números aleatorios entre 1 y 100.

A continuación, se crear una instancia del paquete con la biblioteca de programas específica para el rango indicado (líneas 5-6). Fíjate que se usa la biblioteca que teníamos, y se le indica el rango especificado *T_Rang_Azar*.

La instrucción *use* (línea 7) simplifica las instrucciones *reset* y *random* que aparecen más adelante (evitan tener que poner el prefijo *Azar*_Int por delante).

En la línea 8 se define un generador de valores aleatorios, que se inicializa en la línea 11. Este paso es importante porque sin él, el generador generaría siempre los mismos números en el mismo orden (lo cual podría ser conveniente para hacer pruebas en algún caso).

La función *random* de la línea 12 obtiene un nuevo valor aleatorio del tipo *T_Rang_Azar* a partir del generador *gen.* Las sucesivas llamadas producirán nuevos valores aleatorios dentro del rango.