PROGRAMACIÓN BÁSICA – Examen
Evaluación Extraordinaria – Final
14 de junio de 2021

PREGUNTA 1. Especificaciones y casos de prueba

1.1 Especifica el procedimiento *Dados* que, si se le pasa el número de caras de un dado y el de un segundo dado (que puede tener igual número de caras o diferente), devuelve el valor más probable sumando los números obtenidos al lanzar esos dos dados. Si hubiera varias sumas equiprobables, se devuelve la más alta de todas ellas. Por ejemplo, si los dos dados tienen 6 caras cada uno, la suma más probable es 7 (es la que más posibilidades tiene de salir, porque hay seis combinaciones que nos llevan a ella, como se ve en la tabla siguiente). Si un dado tuviera 6 caras y otro 4, las sumas más probables son 5, 6 y 7. Y la mayor de todas ellas es 7.

EJEMPLO 1				EJEMPLO 2
1 dado de 6 y otro de 6 caras	Comb.	**suma**	Comb.	1 dado de 6 y otro de 4 caras
--	0	**1**	0	--
1+1	1	**2**	1	1+1
1+2; 2+1	2	**3**	2	1+2; 2+1
1+3; 2+2; 3+1	3	**4**	3	1+3; 2+2; 3+1
1+4; 2+3; 3+2; 4+1	4	**5**	4	1+4; 2+3; 3+2; 4+1
1+5; 2+4; 3+3; 4+2; 5+1	5	**6**	4	2+4; 3+3; 4+2; 5+1
1+6; 2+5; 3+4; 4+3; 5+2; 6+1	**6**	**7**	**4**	**3+4; 4+3; 5+2; 6+1**
2+6; 3+5; 4+4; 5+3; 6+2	5	**8**	3	4+4; 5+3; 6+2
3+6; 4+5; 5+4; 6+3	4	**9**	2	5+4; 6+3
4+6; 5+5; 6+4	3	**10**	1	6+4
5+6; 6+5	2	**11**	0	--
6+6	1	**12**	0	--

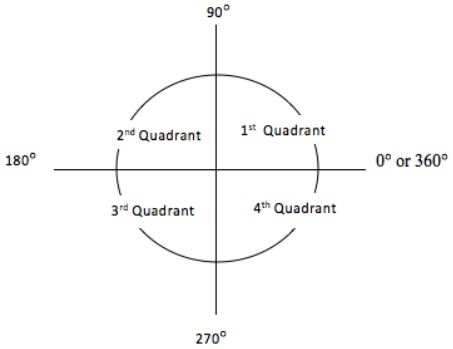
```
procedure Dados (Dado1, Dado2 : in Positive; Suma_Mas_Probable: out Positive);
--pre: Dado 1 y Dado2 representan el número de caras de cada dado respectivamente
--post: suma_mas_probable = la suma más probable que sale al tirar los dos dados,
--        dado1 y dado2
```

1.2 Especifica la función *suma_de_dígitos* que, dado un número natural devuelve la suma de sus dígitos. Por ejemplo, para el número 123 el resultado es **6**, para el número 4376 devuelve **20** o para el número 3 devuelve **3**.

```
function Suma_De_Digitos (N : in Natural)
  return Natural;
--post: resultado es la suma de digitos de N
```

1.3 Describe varios casos de prueba de la función *Cuadrante* que, dados un ángulo en grados devuelva el número de cuadrante en que se encuentra dicho ángulo. Por ejemplo, un ángulo de 60º está en el cuadrante 1.

PROGRAMACIÓN BÁSICA – Examen
Evaluación Extraordinaria – Final
14 de junio de 2021




```
Casos de prueba:
-- Num# Grados Descripción Resultado
--    1   30   Cuadrante 1    1
--    2  100   Cuadrante 2    2
--    3  200   Cuadrante 3    3
--    4  300   Cuadrante 4    4
```

PREGUNTA 2. Tipos Básicos

2.1 Implementa en Ada el procedimiento *números_que_suman* que, dados tres números naturales positivos *suma*, *min* y *max*, encuentre un par de números naturales *valor_min* y *valor_max* comprendidos entre min y max (incluyendo dichos valores), tales que la suma de sus dígitos sea *suma*. *Valor_min* es el menor de esos valores y *Valor_max* el mayor. Si no existiera dicho número se devuelve **0**.

Por ejemplo, si la suma es 4 y la búsqueda se hace entre 1 (*min*) y 100 (*max*), el número menor cuyos dígitos suman 4 es **4** (*valor_min*) y el mayor es **40** (*valor_max*). En cambio, entre 4 y 39, el número menor cuyos dígitos suman 4 es **4** y el máximo es **31**. Entre 7 y 16, siendo la suma de dígitos 4, el número menor y máximo serían el mismo, es decir, **13** y **13,** respectivamente. Y entre 5 y 9, como no hay números cuyos dígitos sumen 4, se debería devolver **0** y **0**.

```
procedure Numeros_Que_Suman (suma: in Natural; min, max: in Natural;
                             valor_min, valor_max: out Natural);
```

Se encuentran disponibles los siguientes subprogramas para obtener, respectivamente, el número de dígitos de un número dado y el i-ésimo dígito de un número dado:

```
procedure Contar_digitos (Num: in Integer; digits: out Natural);
-- Post: devuelve el número de dígitos de Num

function Digito_I (Num: in Integer; I: in Positive) return Integer;
-- pre: Num > 0, 1<= I <= cantidad de dígitos de Num
-- post: devuelve el I-ésimo dígito de Num
-- Ejemplo: para el número 53, el primer dígito es 5 y el segundo es 3.
```

Ejercicio resuelto por refinamientos:

```
PROCEDURE Numeros_Que_Suman (
      Suma,
      Min,
      Max       : IN     Natural;
```

```
      Valor_Min,
      Valor_Max :    OUT Natural) IS

BEGIN
   Calcular_Num(Suma, Min, Min, Max, 1, Valor_Min);
  -- nuevo subprograma calcular_Num: refinamiento
   IF Valor_Min/=0 THEN
      Calcular_Num(Suma, Max, Min, Max, -1, Valor_Max);
   END IF;

END Numeros_Que_Suman;
```

```
PROCEDURE Calcular_Num (
      Suma,
      Num   :        Natural;
      Min,
      Max   :        Natural;
      Mul   :        Integer;
      Valor :    OUT Natural) IS
   Encontrado : Boolean := False;
   Sum        : Natural := 0;
BEGIN
   Valor:= Num;
   Encontrado:=False;
   Sum:=0;
   WHILE NOT Encontrado AND Min<=Valor AND Valor<=Max LOOP
      Sumar_Digitos(Valor, Sum); -- nuevo subprograma sumar_digitos: refinamiento
      IF Sum=Suma THEN
         Encontrado:=True;
      ELSE
         Valor:=Valor+(1*Mul);
         Sum:=0;
      END IF;
   END LOOP;
   IF NOT Encontrado THEN
      Valor:=0;
   END IF;
END Calcular_Num;
```

```
PROCEDURE Sumar_Digitos (
      Valor :        Natural;
      Sum   :    OUT Natural) IS
   Dig : Natural;
BEGIN
   Sum := 0;
   Contar_Digitos(Valor, Dig);
   WHILE Dig/=0 LOOP -- sumar dígitos del número
      Sum:=Sum+Digito_I(Valor, Dig);
      Dig:=Dig-1;
   END LOOP;
END Sumar_Digitos;
```

PROGRAMACIÓN BÁSICA – Examen
Evaluación Extraordinaria – Final
14 de junio de 2021

2.2. Implementa en Ada el programa *Secuencia_Suma_Dados* que lea del teclado el resultado de una o más tiradas de un par de dados, terminadas en 0, y calcule el mayor número de puntos obtenidos en las tiradas realizadas y cuántas veces se ha obtenido en las tiradas leídas.

Por ejemplo, si la secuencia de entrada es <(2 1) **(4 5)** (6 1) **(5 4) (6 3)** (1 1) (6 1) **(5 4)** 0> (los paréntesis los usamos para diferenciar las tiradas en este ejemplo, pero no se escriben en el teclado y la negrita la usamos para mostrar las tiradas más altas). La respuesta esperada es un mensaje que indique que **"la tirada más alta es 9 y ha aparecido 4 veces".**

```
procedure secuencia_suma_dados is
   dado1, dado2: natural;
   suma, suma_max: natural:=0;
   cont:natural:=0;
begin
   loop
      Put_Line(" Valor del dado 1: ");
      get(dado1);
      exit when dado1 = 0;
      Put_Line(" Valor del dado 2: ");
      Get(dado2);
      suma:= dado1+ dado2;
      if (suma > suma_max) then
         suma_max:= suma;
         cont:=1;
      elsif (suma = suma_max) then
            cont:=cont+1;
      end if;
   end loop;

   Put("La tirada mas alta es "); put(suma_max);
   Put(" y ha aparecido "); put(cont);  Put(" veces.");
end secuencia_suma_dados;
```

PREGUNTA 3. Vectores – Cuatro en raya

El juego de las cuatro en raya usa un tablero en vertical formado por seis filas y siete columnas. La partida la juegan dos jugadores, con fichas de distinto color. Al comienzo el tablero está vacío y los jugadores alternativamente van colocando sus fichas en él. La única regla es que el jugador decide en qué columna colocar su ficha. La fila que le corresponde es la más inferior que no esté ocupada. La partida acaba cuando un jugador consigue colocar 4 fichas alineadas horizontal, vertical o diagonalmente, que es el ganador. Si no quedasen huecos para colocar fichas, entonces la partida acaba en tablas.
El tipo **T_Tablero** representa un tablero de dimensiones 6x7. Este es un tablero de tipo **T_Tablero**, con 8 fichas, cuatro por jugador, el número de la ficha indica el orden en que se han jugado.

6							
5							
4				⑧			
3				⑥			
2			❺	④			
1			②	❶	❸	❼	❾
	1	**2**	**3**	**4**	**5**	**6**	**7**

PROGRAMACIÓN BÁSICA – Examen
Evaluación Extraordinaria – Final
14 de junio de 2021

El tipo **T_Jugadas** representa una serie de jugadas donde se indica, en orden, en qué columnas los jugadores han ido insertando las fichas. En las posiciones impares están las jugadas de un jugador y en las pares las del otro jugador.

	1	2	3	4	5	6	7	8	9
Jugadas	4	3	5	4	3	4	6	4	7

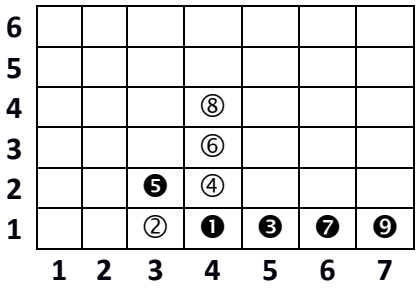
La declaración en Ada de los tipos indicados es la siguiente:

```
Max_Filas: constant Integer := 6;
Max_Cols: constant Integer := 7;
Max_jugadas: constant Integer := Max_filas * Max_Cols;

subtype T_Jugador is Integer range 0..2; -- 0–Vacio; 1-Blanco; 2-Rojo

type T_Pos_Tablero is record
   Ocupada: Boolean;  -- True si ocupada y False si libre
   Jugador: T_Jugador;-- número del jugador
end record;

type T_Tablero is array (Integer range 1..Max_Filas, Integer range 1..Max_Cols)
   of T_Pos_Tablero;

type T_Jugadas is array (Integer range 1..Max_Jugadas) of Positive;
```

3.1. Implementa el procedimiento *Inicializar_Tablero* que, dado un tablero, lo inicialice para empezar a jugar una partida. El tablero inicializado no tiene fichas.

```
procedure Inicializar_Tablero(T: out T_Tablero) is
begin
   for fila in T'first(1) .. T'last(1) loop
      for columna in T'first(2) .. T'last(2) loop
         T(fila, columna).Ocupada:=false;
      end loop;
   end loop;
end inicializar_tablero;
```

3.2. Implementa el procedimiento *Procesar_Partida* que, dado un tablero vacío y una lista de jugadas correctas (i.e. sin columnas fuera de rango, ni más jugadas en una columna de las permitidas, etc.), altera el tablero para reflejar el nuevo estado del tablero tras incorporar cada una de las jugadas que aparecen en dicha lista.




```
procedure Procesar_Partida (T: in out T_Tablero; J: in T_Jugadas);
```

Por ejemplo, con el tablero vacío, y la lista de jugadas (4, 3, 5, 4, 3, 4, 6, 4, 7), el jugador 1 coloca la ficha en la columna 4, luego el jugador 2 en la 3, después el primero en la columna 5, y el segundo en la 4. Y así, sucesivamente, en las columnas 3, 4, 6, 4 y 7. La primera ficha se coloca en la posición (1,4), la más inferior del tablero. Lo mismo

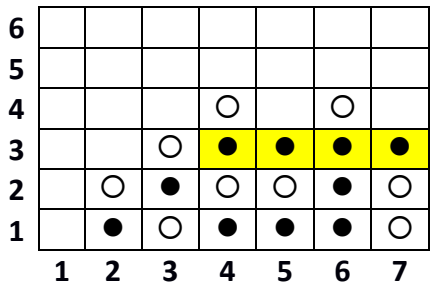
ocurre con las dos siguientes jugadas: (1,3) y (1,5). La siguiente ficha va en la columna 4, pero como la fila 1 está ocupada, entonces se coloca en la posición (2,4). Así sucesivamente hasta alcanzar la configuración del tablero de la figura de la derecha.

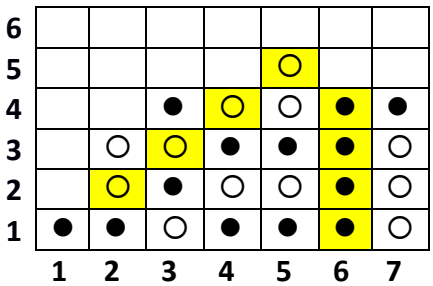
```
procedure procesar_partida(T: in out T_tablero; J: in T_jugadas) is
   fila:natural;
begin
   for i in J'range loop
      fila:= T'last(1); --lo más abajo de la matriz
      while T(fila, J(i)).Ocupada loop --busca primer hueco en columna J(i)
         fila:=fila-1;
      end loop;
      T(fila, J(i)).Ocupada:=true;
      if i rem 2 = 0 then calcular número jugador 1 o 2
         T(fila, J(i)).Jugador:=2; -- posición par el jugador 2
      else
         T(fila, J(i)).Jugador:=1; -- posición impar el jugador 1
      end if;
   end loop;

end procesar_partida;
```

3.3. Implementa la función *Final_Partida* que, dado un tablero, comprueba si hay cuatro fichas alineadas en él. Es suficiente con comprobar que hay cuatro fichas alineadas horizontalmente.

```
function Final_Partida (T: in T_Tablero) return Boolean;
```

Por ejemplo, con la situación del tablero de la figura de la izquierda, el resultado debe ser ***true*** porque en la tercera fila, columnas 4-7 hay 4 fichas del mismo color (o sea, del mismo jugador). En cambio, en la figura de la derecha no hay ninguna fila que cumpla dicha condición y, por tanto, el resultado debe ser ***false*****.**





```
FUNCTION Final_Partida (T: IN T_Tablero) RETURN Boolean IS
   Fin     : Boolean  := False;
   Cont    : Positive;
   F, C: Natural;

BEGIN
   F:= T'Last(1);
   WHILE F>= 1 AND NOT Fin LOOP
      C:=1;
      Cont:=1;
      WHILE C<T'Last(2) AND NOT Fin LOOP
         IF T(F,C).Jugador = T(F,C+1).Jugador THEN
            Cont:=Cont+1;
            IF Cont=4 THEN
               Fin:=True;
            END IF;
         ELSE
            Cont:=1;
         END IF;
      END LOOP;
      F:=F-1;
   END LOOP;
   RETURN Fin;

END Final_Partida;
```

PREGUNTA 4. Listas estáticas

Dadas las siguientes declaraciones de una lista estática:

```
type T_Vector_enteros is array(1..100) of Integer;

type T_lista_Estatica is record
   Dato: T_Vector_Enteros;
   Cont: Natural;
end record;
```

Implementa en Ada el procedimiento *Mezclar_Listas* que, dadas dos listas estáticas E1 y E2 sin ordenar y sin elementos repetidos, modifique E1 y añada al final de E1 todos los elementos de E2 que aún no se encuentran en E1 y elimine de E1 aquellos que sí que están.

```
procedure Mezclar_Listas (E1: in out T_Lista_Estatica; E2: in T_Lista_Estatica);
```

Como ilustración del funcionamiento del procedimiento *Mezclar_Listas,* dadas las listas L0 a L7, en la parte derecha se representa cuál es el resultado que debe devolver el procedimiento *Mezclar_Listas* entre todas sus combinaciones. Para facilitar la lectura, los números se han puesto ordenados, aunque **no es una precondición.** En la cuarta columna de la primera fila se muestra el resultado de *Mezclar_Listas* sobre una lista vacía (L0) y L5 como segundo parámetro. Como no existe ningún elemento en la lista inicial, se añaden los elementos de L5. En cambio, cuando se usa L5 y L5, como

todos los elementos de la primera lista están en la segunda, se van eliminando hasta quedar vacía. Finalmente, para L5 y L6, de la lista inicial, se elimina el 20 (que está en la segunda lista) y el resto se añade al final (o sea, se añaden el 5, el 10 y el 17).

Valor de E2		()	(15 16 20)	(5 10 17 20)	(5 10 15 17 20)
Valor de E1		**L0**	**L5**	**L6**	**L7**
()	**L0**	()	(15 16 20)	(5 10 17 20)	(5 10 15 17 20)
(10)	**L1**	(10)	(10 15 16 20)	(5 17 20)	(5 15 17 20)
(10 20)	**L2**	(10 20)	(10 15 16)	(5 17)	(5 15 17)
(5 17 20)	**L3**	(5 17 20)	(5 17 15 16)	(10)	(10 15)
(10 13 20)	**L4**	(10 13 20)	(10 13 15 16)	(13 5 17)	(13 5 15 17)
(15 16 20)	**L5**	(15 16 20)	()	(15 16 5 10 17)	(16 5 10 17)
(5 10 17 20)	**L6**	(5 10 17 20)	(5 10 17 15 16)	()	(15)
(5 10 15 17 20)	**L7**	(5 10 15 17 20)	(5 10 17 16)	(15)	()

Nota: Para facilitar la lectura, los números en estos ejemplos se han puesto ordenados, aunque **no es una precondición.**

```
PROCEDURE Mezclar_Listas (
      E1 : IN OUT T_Lista_Estatica;
      E2 : IN     T_Lista_Estatica) IS

   J : Natural;
BEGIN

   FOR I IN 1..E2.Cont LOOP
      J:=1;
      WHILE J<=E1.Cont AND THEN E1.Dato(J)/=E2.Dato(I) LOOP
         J:=J+1;
      END LOOP;
      IF J>E1.Cont THEN
         E1.Cont:=E1.Cont+1;
         E1.Dato(E1.Cont):=E2.Dato(I);
      ELSE
         E1.Dato(J..E1.Cont-1):=E1.Dato(J+1..E1.Cont);
         E1.Cont:=E1.Cont-1;
      END IF;
   END LOOP;


END Mezclar_Listas;
```

PROGRAMACIÓN BÁSICA – Examen
Evaluación Extraordinaria – Final
14 de junio de 2021

PREGUNTA 5. Listas dinámicas

Dadas las siguientes declaraciones de una lista dinámica:

```
type T_Nodo;

type T_Lista_Dinamica is access T_Nodo;
type T_Nodo is record
   Info: Integer;
   Sig: T_Lista_Dinamica;
end record;
```

Implementa el procedimiento *Mezclar_Listas* que, dadas dos listas dinámicas D1 y D2 sin ordenar y sin elementos repetidos, modifique D1 y añada al final de D1 todos los elementos de D2 que aún no se encuentran en D1 y elimine de D1 aquellos que sí están. Véanse los ejemplos de la pregunta 4.

```
PROCEDURE Mezclar_Listas (D1 : IN OUT T_Lista_Dinamica; D2: IN T_Lista_Dinamica) IS
   Ant1,
   Act1 : T_Lista_Dinamica;
   Aux2 : T_Lista_Dinamica := D2;

BEGIN

   WHILE Aux2/=NULL LOOP   --tratar cada elemento de D2
      Ant1:=D1; Act1:= D1;
      WHILE Act1/=NULL AND THEN Act1.Info/=Aux2.Info LOOP
         Ant1:= Act1;
         Act1:=Act1.Sig;
      END LOOP;
      IF D1= NULL THEN --lista D1 vacía: insertar como primero
         D1:=NEW T_Nodo'(Aux2.Info, NULL); -- D1 es vacío y hay que insertar primero
      ELSE
         IF Act1=NULL THEN -- no encontrado. Ant1 apunta al último: insertar ultimo
            Ant1.Sig:=NEW T_Nodo'(Aux2.Info, NULL);
         ELSE -- encontrado el elemento
            IF Ant1=Act1 THEN -- eliminar primero
               D1:= D1.Sig;
            ELSE -- eliminar intermedio/ultimo
               Ant1.Sig:= Act1.Sig;
            END IF;
         END IF;
      END IF;
      Aux2:=Aux2.Sig;
   END LOOP;
END Mezclar_Listas;
```

Puntos preguntas	P1	P2	P3	P4	P5	Total	Tiempo
Global (P1-P5)	0.75	1.75	2.5	2.5	2.5	10.00	180 min
Tiempo preguntas	30min	25+10min	10+15+15min	30min	25min		