PROGRAMACIÓN BÁSICA – Examen
Evaluación Continua – Final
7 de enero de 2021- SOLUCIÓN

PREGUNTA 1. Especificaciones y casos de prueba

1.1 Especifica el procedimiento *Obtener_Digito_Romano* que, dado un número entero positivo, devuelva el dígito romano equivalente de acuerdo con la siguiente tabla, o un carácter espacio en blanco si no es uno de esos siete números.

Decimal	1	5	10	50	100	500	1000
Romano	I	V	X	L	C	D	M

```
procedure Obtener_Digito_Romano (num: in Integer; resul: out Character)
   -- pre: num es uno de la lista (1, 5, 10, 50, 100, 500, 1000)
   -- post: resul es uno de la lista ('I', 'V', 'X', 'L', 'C', 'D', 'M'),
   --       y la posición de resul en la lista se corresponde con la posición
   --       de num en su lista. Es decir, resul corresponde al número romano num
```

1.2 Especifica la función *Son_Equivalentes* que, dado un valor entero y una cadena de caracteres, devuelva si los dos valores son equivalentes (Por ejemplo, 2257 y *MMCCLVII* son equivalentes, así como 10 y *X*, también 50 y *L*, pero no lo son 20 y *III*).

```
function Son_Equivalentes (num: in Integer; cadena: in String) return Boolean;
   -- pre: num > 0
   -- post: devuelve TRUE sii los caracteres contenidos en cadena están en la
   --       lista ('I', 'V', 'X', 'L', 'C', 'D', 'M') y además la expresión de
   --       num en notación romana se corresponde con cadena.
```

1.3 Describe casos de prueba de la función *Romano_A_Capicúa* que, dado un número romano, devuelve el número entero capicúa más cercano a su equivalente decimal. Si hubiera dos igual de cercanos, devuelve el menor de ellos. Por ejemplo, para el numero romano MMCCLVII (en decimal 2257), los dos números capicúas más cercanos son 2222 (N-35) y 2332 (N+75); y el más cercano de ellos es el 2222.

Caso 1: El propio número es capicúa
Entrada: "XXII" (22) → salida: "XXII" (22) [22 es capicúa]

Caso 2: De un dígito, son capicúas
Entrada: "VII" (7) → salida: "VII" (7)

Caso 3: El capicúa de abajo es el más cercano
Entrada: "XII" (12) → salida: "XI" (11) [el otro es "XXII" (22)]

Caso 4: El capicúa de arriba es el más cercano
Entrada: "XX" (20) → salida: "XXII" (22) [el otro es "XI" (11)]

PREGUNTA 2. Tipos Básicos

2.1 Implementa en Ada la función *Es_Desigual* que, dado un número entero, nos indique si el número es desigual. En un número desigual los dígitos no se repiten y, además, no pueden tener un valor mayor que la longitud del número (la cantidad de dígitos). Por ejemplo, 20, 102 y 1234 son desiguales porque todos los dígitos son distintos y de valor menor o igual a 2, 3 y 4, respectivamente; 1235 no es desigual porque hay un dígito con valor 5 siendo 4 el número de dígitos; 333 tampoco es desigual porque tiene dígitos repetidos.

```
function Es_Desigual (num1: in Natural) return Boolean;
```

PROGRAMACIÓN BÁSICA – Examen
Evaluación Continua – Final
7 de enero de 2021- SOLUCIÓN

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.
```

```
function Es_Desigual (num: in Natural) return Boolean is
   Cont, i, j, digito : Integer;
   desigual: Boolean;
begin
   desigual := true;
   Contar_Digitos(num, Cont);

   i := 1;
   while (i <= Cont and desigual) loop
      digito := Digito_I (num, i);
      if (digito > Cont) then
         desigual := false;
      ELSE
         j:= 1;  -- esta parte podría ser un subprograma
         while j <i and desigual loop
            if digito = Digito_I (num, j) then
               desigual := false;
            END IF;
            j:= j+1;
         end loop;
         i := i+1;
      end if;
   end loop;
   return desigual;

end Es_Desigual;
```

2.2. Implementa en Ada el programa *Secuencia_Desigual* que, lea del teclado una serie de números positivos terminada en cero y escriba en pantalla un mensaje indicando la mayor cantidad de números consecutivos desiguales encontrados y el número inicial de dicha secuencia, si es que existe (ver ejemplo). En la implementación se debe utilizar el subprograma implementado en el ejercicio 2.1.

Por ejemplo, si la secuencia de entrada es <33, **20**, **21**, 22, 4367, **12**, **102**, **1342**, 233, 0> (los números que aparecen en negrita son desiguales). La primera secuencia de números desiguales consecutivos comienza con el 20 y es de longitud 2, y la segunda secuencia comienza con el 12 y tiene una longitud de 3. El programa debe escribir **"la secuencia más larga de desiguales es de longitud 3 y comienza con 12".** Si no hubiera desiguales, el mensaje sería **"No se ha detectado ningún número desigual en la entrada."**

PROGRAMACIÓN BÁSICA – Examen
Evaluación Continua – Final
7 de enero de 2021- SOLUCIÓN

```
PROCEDURE Secuencia_Desigual IS
   N        : Integer;
   Cont, Cont_Aux, Prim, Prim_Aux : Integer := 0;

BEGIN
   Put_Line("Introduce varios numeros positivos (0 para terminar)");
   Get(N);
   WHILE N/= 0 LOOP
      IF Es_Desigual (N) THEN
         Cont_Aux:= Cont_Aux+ 1;
         IF Prim_Aux = 0 THEN
            Prim_Aux := N;
         END IF;
      ELSE
         IF Cont_Aux> Cont THEN
            Cont := Cont_Aux;
            Prim := Prim_Aux;
         END IF;
         Cont_Aux:= 0;
         Prim_Aux:= 0;
      END IF;
      Get(N);
   END LOOP;
   IF Cont=0 THEN
      Put_Line("No se ha detectado ningún número desigual en la entrada.");
   ELSE
      Put_Line("La secuencia más larga de desiguales es de longitud ");
      Put(Cont, 0);
      Put(" y comienza con ");
      Put(Prim, 0);
   END IF;
END Secuencia_Desigual;
```

PREGUNTA 3. Vectores – Salas de cine

En un *sistema de gestión de entradas para salas de cine* se almacena información sobre las reservas realizadas sobre las butacas de la sala y sobre las peticiones offline para reserva de butacas. Usamos el tipo **T_Sala_de_Cine** para representar las reservas realizadas. A cada butaca se le asocia el valor de si está ocupada o no y, si lo está, se indica el localizador de la reserva realizada.

El Tipo **T_Reservas_Off** representa una lista de peticiones de reserva offline que se han realizado para una sesión. Cada petición de reserva tiene un número de localizador y el número de butacas solicitadas. En esa misma estructura se almacena el resultado de la petición, indicando (con su fila y columna) cuál es la butaca inicial de la reserva, o bien el valor (0,0) si la reserva no se puede satisfacer.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
F1		True 222				True 232	True 595			
F2	True 469	True 469	True 469	True 469	True 422	True 411	True 111	True 321	True 543	
F3	True 333	True 333	True 333			True 247				True 567

PROGRAMACIÓN BÁSICA – Examen
Evaluación Continua – Final
7 de enero de 2021- SOLUCIÓN

F4	True 987		True 814							True 589
F5		True 123	True 288	True 288	True 288	True 288	True 288	True 288		True 259
F6	True 555			True 898	True 898					

Figura 1. Estado inicial de reservas

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

```
Num_Billete_Max: constant Integer := 1_000_000;
subtype T_Localizador is Integer range 1..Num_Billete_Max;


--Ejercicio 3.1
type T_Butaca is record
   Ocupada: Boolean;   -- True si ocupada y False si libre
   Localizador: T_Localizador;-- solo si la butaca está ocupada
end record;

type T_Sala_de_Cine is array (Integer range <>, Integer range <>) of T_Butaca;


--Ejercicio 3.2
type T_Reserva is record
   Localizador: T_Localizador;
   Num_butacas: Natural;   --butacas contiguas solicitadas
   Fila, columna: Integer; --fila y columna de la butaca inicial asignada
end record;

type T_Reservas_off is array (Integer range <>) of T_Reserva;
```

3.1. Implementa el procedimiento *Obtener_Hueco* que, dados el estado de reservas de la sala de cine y un número de butacas que se quieren ocupar, devuelva la posición (fila y columna) más adelantada y más a la izquierda posible para tantas butacas. Si hay varios huecos con esa capacidad, se debe devolver el de menor longitud de todos ellos. Si no hay hueco para tantas butacas en una misma fila, la posición que se devuelve es (0,0). Recuerda que los huecos se computan por fila, de manera que no se consideran contiguos huecos de distintas filas.

Con el estado de la figura anterior, cuando el número de butacas solicitado es 1, la posición del hueco es (1,1), porque, aunque en todos los huecos cabe una butaca, no todos son los de menor tamaño; solo son los de las posiciones (1,1), (2,10), (4,2), (5,1) y (5,9), y el más adelantado de todos ellos es el de la fila 1: (1,1). Cuando se solicitan 2 butacas, se obtiene el hueco que empieza por (3,4), porque es el más adelantado de los de dos posiciones (el otro está en (6, 2)); para 3 es (1,3), siguiendo el mismo razonamiento; Cuando se solicitan 4 butacas, no hay huecos exactos, así que se localiza un hueco mayor: (4,4) [6 butacas] y (6,6) [5 butacas], y se devuelve (6,6), que corresponde al hueco más pequeño de ellos; igualmente, para 5 butacas solicitadas, también se devuelve (6,6); para 6 es (4,4), y finalmente, para 7 es (0,0), puesto que no hay suficientes butacas en ninguna fila.

```
procedure Obtener_Hueco (Sala: in T_Sala_de_Cine; NButacas : in Positive;
                         Fil, Col : out Integer);
```

```
PROCEDURE Obtener_Hueco (
      Sala     : IN     T_Sala_De_Cine;
      NButacas : IN     Positive;
      Fil,
      Col      :    OUT Integer) IS
   Hueco       : Integer;
   Prim_F,
   Prim_C,
   Cont_Huecos : Integer := 0;
BEGIN
   Fil := 0;
   Col := 0;
   Hueco := Sala'length(2)+1; -- un valor más grande que la fila
   FOR I IN Sala'RANGE(1) LOOP
      FOR J IN Sala'RANGE(2) LOOP
         IF Sala(I,J).Ocupada OR ELSE
                                 J=Sala'Last THEN--asiento ocupado/último libre
            IF NOT Sala(I,J).Ocupada AND J=Sala'Last THEN -- último libre
               Cont_Huecos:= Cont_Huecos+1;
            END IF;
            IF Cont_Huecos >= NButacas THEN
               IF Cont_Huecos < Hueco THEN
                  Hueco := Cont_Huecos;
                  Fil := Prim_F;
                  Col := Prim_C;
               END IF;
            END IF;
            Cont_Huecos := 0;
         ELSE -- asiento libre
            IF Cont_Huecos = 0 THEN
               Prim_F := I;
               Prim_C := J;
               Cont_Huecos := 1;
            ELSE
               Cont_Huecos := Cont_Huecos +1;
            END IF;
         END IF;
      END LOOP;

   END LOOP;

END Obtener_Hueco ;
```

3.2. Implementa el procedimiento *Procesar_Espectadores* que, dado el estado de reservas de una sala de cine y un vector de tipo **T_Reservas_off** con personas que han solicitado offline reservas para esa sesión. El procedimiento debe asignar, a cada petición, butacas consecutivas utilizando el procedimiento *Obtener_Hueco.* Para ello, actualiza ambas estructuras de datos; la sala recoge la ocupación y el localizador, y cada reserva offline se completa con la fila y la columna asignadas o bien (0,0) si no se le pueden asignar butacas.

Por ejemplo, con el estado inicial indicado en la figura superior, si en el vector de solicitudes offline se tienen las peticiones siguientes (localizador, butacas solicitadas): [(899,5), (900,7), (901,6), (902,3), (903,2), (904,2), (905,2), (906,3)], el programa debe actualizar el estado de las reservas de la sala (ver figura inferior). Asigna las butacas por orden de petición y deja sin asignar solo aquellas peticiones con el número de butacas solicitado demasiado alto o cuando

no quedan más butacas para reservar. Las solicitudes de reserva que se devuelven son (se añade la fila y columna asignada en negrita) [(899,5,**6,6**), (900,7,**0,0**), (901,6,**4,4**), (902,3,**1,3**), (903,2,**3,4**), (904,2,**6,2**), (905,2,**1,8**), (906,3,**3,7**)].

```
procedure Procesar_Espectadores (Sala : in out T_Sala_de_Cine;
                                 R    : in out T_Reservas_off)
```

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
F1		True 222	True 902	True 902	True 902	True 232	True 595	True 905	True 905	
F2	True 469	True 469	True 469	True 469	True 422	True 411	True 111	True 321	True 543	
F3	True 333	True 333	True 333	True 903	True 903	True 247	True 906	True 906	True 906	True 567
F4	True 987		True 814	True 901	True 901	True 901	True 901	True 901	True 901	True 589
F5		True 123	True 288	True 288	True 288	True 288	True 288	True 288		True 259
F6	True 555	True 904	True 904	True 898	True 898	True 899	True 899	True 899	True 899	True 899

Figura 2. Estado final de la sala

```
PROCEDURE Procesar_Espectadores (
      Sala : IN OUT T_Sala_De_Cine;
      R    : IN OUT T_Reservas_Off) IS
   F,
   C : Integer;
BEGIN
   FOR I IN R'RANGE LOOP
      Obtener_Hueco(Sala, R(I).Num_Butacas, F, C);
      R(I).Fila := F;
      R(I).Columna := C;
      IF F /= 0 THEN
         FOR J IN C..C+R(I).Num_Butacas-1 LOOP
            Sala(F, J).Ocupada := True;
            Sala(F, J).Localizador := R(I).Localizador;
         END LOOP;
      END IF;
   END LOOP;
END Procesar_Espectadores;
```

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;
```

PROGRAMACIÓN BÁSICA – Examen
Evaluación Continua – Final
7 de enero de 2021- SOLUCIÓN

Implementa en Ada la función *Es_Sublista* que, dadas dos listas estáticas L1 y L2 cuyos elementos no se repiten y están ordenados ascendentemente, devuelva *true* si todos los elementos de L1 se encuentran en L2.

```
function Es_Sublista (L1, L2: in T_Lista_Estatica) return Boolean;
```

Como ilustración del funcionamiento de la función *Es_Sublista,* dadas las listas L0 a L7, en la parte derecha se representa cuál es el resultado que debe devolver la función es_sublista entre todas sus combinaciones. En la primera fila se muestra si L0 es sublista de todas las demás listas, la segunda los resultados de L1 con el resto y así sucesivamente.

- L0 ()
- L1 (10)
- L2 (10 20)
- L3 (5 17 20)
- L4 (10 13 20)
- L5 (15 16 20)
- L6 (5 10 17 20)
- L7 (5 10 15 17 20)

	L0	L1	L2	L3	L4	L5	L6	L7
L0	✓	✓	✓	✓	✓	✓	✓	✓
L1	×	✓	✓	×	✓	×	✓	✓
L2	×	×	✓	×	✓	×	✓	✓
L3	×	×	×	✓	×	×	✓	✓
L4	×	×	×	×	✓	×	×	×
L5	×	×	×	×	×	✓	×	×
L6	×	×	×	×	×	×	✓	✓
L7	×	×	×	×	×	×	×	✓

```
function Es_Sublista ( L1, L2 : in T_Lista_Estatica) return Boolean is
   i, j: Natural ;
   esSublista  : Boolean ;
begin
   i := 1;
   j := 1;
   esSublista := True;
   while i <= L1.Cont and j <= L2.Cont and esSublista loop
      if L1.Dato(i) = L2.Dato(j) then
         i := i+1;
         j := j+1;
      elsif L1.Dato(i) > L2 Dato(j) then
         j := j+1;
      else
         esSublista := false;
      end if;
   end loop;
   if j > L2.Cont and i <= L1.Cont then
      esSublista := false;
   end if;
   return esSublista;
end Es_Sublista;
```

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;
```

PROGRAMACIÓN BÁSICA – Examen
Evaluación Continua – Final
7 de enero de 2021- SOLUCIÓN

Implementa la función *Es_Sublista* que, dadas dos listas dinámicas L1 y L2 cuyos elementos no se repiten y están ordenados ascendentemente, devuelva true si todos los elementos de L1 se encuentran en L2. Véanse los ejemplos de la pregunta 4.

```
function Es_Sublista (L1, L2: in T_Lista_Dinamica) return Boolean;
```

```
function Es_Sublista ( L1, L2 : in T_Lista_Dinamica ) return Boolean is
   Aux1, Aux2: T_Lista_Dinamica ;
   esSublista  : Boolean ;
begin
   Aux1:= L1;
   Aux2:= L2;
   esSublista := true;
   while Aux1/= null and Aux2/= null and esSublista loop
      if Aux1.Info= Aux2.Info then
         Aux1:= Aux1.Sig;
         Aux2:= Aux2.Sig;
      elsif Aux1.Info > Aux2.Info then
         Aux2:= Aux2.Sig;
      else
         esSublista := false;
      end if;
   end loop;
   if Aux2= null and Aux1/= null then
      esSublista := false;
   end if;
   return esSublista;
end Es_Sublista;
```

Puntos preguntas	P1	P2	P3	P4	P5	Total	Tiempo
Continua(P4-P5)	-	-	-	5.00	5.00	10.00	90 min
Global (P1-P5)	0.75	1.75	2.5	2.5	2.5	10.00	210 min