PROGRAMACIÓN BÁSICA – Examen
Evaluación GLOBAL
19 de enero de 2022

PARTE 1

PREGUNTA 1. (1 punto) Especificaciones y casos de prueba (30 min)

1.1 Especifica el procedimiento *Sugerencias_de_Propina*, que dada la cantidad que tiene que pagar un cliente de un bar, devuelva tres sugerencias de propina basadas en esa cantidad: una propina del 10%, una del 15% y otra del 20% de dicha cantidad. Por ejemplo, si la cantidad a pagar son 100.00€, se debe sugerir propinas de 10.00, 15.00 y 20.00€ respectivamente. Si fueran 13.50€, las sugerencias deben ser 1.35, 2.03 y 2.70€.

1.2 Especifica la función *Valor_imc*, que, dada la altura en cm de una persona y su peso en Kg, devuelva la clasificación del valor del índice de masa corporal de dicha persona (una letra entre A y D), que se calcula según el resultado de la siguiente fórmula:

$\text{imc} = \frac{\text{peso(kg)}}{\text{altura(m)}^2}$	• **A** – Extrema Delgadez (menos de 18.5) • **B** – Normal (18.5-24.9) • **C** – Sobrepeso (25.0-29.9) • **D** – Obesidad (30.0 o más)

1.3 Describe 4 casos de pruebas para el subprograma *Valor_imc* indicado en el punto anterior.

PROGRAMACIÓN BÁSICA – Examen
Evaluación GLOBAL
19 de enero de 2022

PREGUNTA 2. Tipos Básicos (t= 45min)

2.1 (1.25 punto) Implementa en Ada la función *Mas_2_al_reves* que, dado un número natural N, devuelve el valor de N + 2 con sus dígitos en orden inverso (ver ejemplos). De los ejemplos cabe destacar que si al sumar 2 e invertir el resultado quedan ceros a la izquierda estos no cuentan. Mirar el caso del 8 que, sumando 2, daría 10 y el número invertido sería 1. O el caso del 1998 que, sumando 2, daría 2000 y el valor invertido sería 2.

Ejemplos:

N	Resultado
0	2
1	3
2	4

N	Resultado
4	6
6	8
8	1 (0**1**)

N	Resultado
12	41
30	23
41	34

N	Resultado
300	203
624	626
1998	2 (000**2**)

```
function Mas_2_al_reves (N: in Natural) return Natural;
```

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

```
function Contar_digitos (Num: in Integer) return Natural;
-- Post: devuelve el número de dígitos de Num

procedure Digito_I (Num: in Positive; I: in Positive; Digito: out Natural);
-- pre: 1<= I <= cantidad de dígitos de Num
-- post: Digito es el I-ésimo dígito de Num (contados desde la izq)
-- Ejemplo: para el número 53, el primer dígito es 5 y el segundo es 3.
```

2.2 (1.75 puntos) Implementa en Ada el programa *Cuantos_Mayores* que, dada una secuencia de números enteros de la entrada estándar terminada en uno negativo (siendo ninguno de los anteriores negativo), con cada uno escriba por pantalla cuántas veces se puede aplicar consecutivamente la función *Mas_2_al_revés* obteniendo un número aún mayor.

Por ejemplo, al número 0 **se le puede aplicar 4 veces** la función *Mas_2_al_reves* obteniendo cada vez un número mayor que el original (el número al que se le aplica la función). Partiendo del 0, 0+2 devuelve 2 (inverso 2, 2>0, 1ª vez), para 2 (el resultado anterior), 2**+2** devuelve 4 (inverso 4, 4>2, 2ª vez), para 4, 4**+2** devuelve 6 (inverso 6, 6>4, 3ª vez), para 6, 6+2 devuelve 8 (inverso 8, 8>6, 4ª vez) y para 8, 8+2 devuelve 10 (inverso 1, 1<8) devuelve 1, que no es mayor que 8, y se termina.

Si la secuencia de entrada es <**0 12 30 -1**>, el programa deberá escribir por pantalla **<4 1 0 >**. Por el 0 de entrada escribirá el 4 en la salida (ejemplo anterior). Para el 12 escribirá el 1, porque solo se puede aplicar una vez (del 12 la función *Mas_2_al_reves* devuelve 41 y para 41 devuelve 34, que no es mayor que 41). Para el 30 escribirá el 0 porque el resultado de aplicar la función *Mas_2_al_reves* a 30 es el 23, que es menor que 30.

PROGRAMACIÓN BÁSICA – Examen
Evaluación GLOBAL
19 de enero de 2022

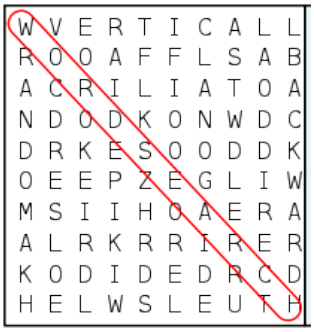
PARTE 2

PREGUNTA 3. Vectores (t= 45 min.)

Vamos a representar una sopa de letras (ver imagen A) con el tipo `T_Sopa_Letras`. Cada posición de la sopa de letras contendrá una letra mayúscula.

```
Max: constant Natural := 6;
type T_Sopa_Letras is array (INTEGER range <>) of STRING(1..Max);
```

(A)



(B)

	1	2	3	4	5	6
1						
2	E	V	I	L	C	
3					R	
4		G			U	
5		E	S	T	E	
6		T			L	

(C)

	1	2	3	4	5	6
1	T	R	U	E		
2	E	V	I	L	C	
3	D			T	R	U
4	I	G	T	R	U	E
5	T	E	S	T	E	
6		T	R	U	L	

Inicialmente cuando se va a proceder a construir la sopa de letras, ésta se inicializa con espacios en blanco (' ') y luego mientras se está construyendo se van añadiendo las letras poco a poco. La figura (B y C) representan estados intermedio, "en construcción". En (B) se tiene cuatro palabras incluidas (EVIL, CRUEL, GET, ESTE), donde el String "EVIL" está en la segunda fila, con el carácter 'E' colocado en la primera columna. Recordad que un String es un array de caracteres, y que para la longitud de un String P, se utiliza la expresión *P'Length*, (p.e. para "EVIL", P'l*ength* devolvería 4), y *P'First* y *P'Last* (P'range) para los índices del primer y último carácter (rango de los índices).

3.1. (1.75 puntos) Implementa la función *Encaja_Este* que, dada una sopa de letras que está en construcción (o sea, todavía con algunos espacios en blanco), una palabra P (String), una fila F y una columna C, indica si la palabra encaja en la sopa de letras partiendo de la fila y columna indicadas hacia la derecha (o sea, hacia el Este). Los valores de F y C están dentro del rango de la sopa de letras. Se considera que una palabra encaja en la sopa de letras si hay espacio suficiente para colocar todas sus letras y si las letras de la palabra no difieren con las ya existentes en la sopa de letras. Esta función no modifica la sopa de letras, tan solo detecta si encaja o no en la posición indicada.

Por ejemplo, partiendo de la sopa de letras de la figura (B), si se pide comprobar (ver figura C) si encaja "TRUE" en F=1 y C=1, la respuesta es cierto (todas las casillas están en blanco), si preguntara en F=4 y C=3, también sería cierto (aunque no todos son blancos, la letra 'U' de la posición 4, 5, coincide con la de la palabra), pero si se preguntara en F=6 y C=2 la respuesta sería falsa (ya que, aunque el primer carácter 'T' coincide y los dos siguientes están libres, el último no coincide, ya que el carácter 'E' no coincide con 'L'). Si indicáramos en la posición F=3 y C=4, tampoco encajaría ya que no entra la palabra.

```
function Encaja_Este (SL: in T_SOPA_LETRAS; P: in STRING;
                                  F,C: in Positive) return Boolean;
```

3.2. (0.75 puntos) Implementa el procedimiento *Colocar_Sur* que modifica una sopa de letras en construcción y añade una palabra P a partir de la fila F y la columna C hacia abajo. Se considera que F y C están dentro del rango y que la palabra encaja en dicha posición (no realizar comprobaciones).

Por ejemplo, a partir de la sopa de letras de la imagen (B), si se pide incrustar la palabra "EDIT" en la posición F=2 y C=1, se modifica la sopa de letras tal y como se muestra en la imagen (C).

```
procedure Colocar_Sur(SL: in out T_SOPA_LETRAS; P in String;
                                                  F,C: in Positive);
```

PROGRAMACIÓN BÁSICA – Examen
Evaluación GLOBAL
19 de enero de 2022

PREGUNTA 4. (1.5 puntos) Listas estáticas (30 min.)
Dadas las siguientes estructuras de datos que representan:

- Una lista (estática) de palabras (**T_Lista_Est_Pal**), que contiene Strings.
- Un vector para clasificar palabras (**T_Todas_Palabras**). En la posición 1 estará la lista de palabras (**T_Lista_Est_Pal**) de 1 letra, en la 2 la lista de palabras de 2 letras, las de 3 letras en la posición 3, etc.

```
Max: constant Natural := 30;

type T_Vector_Palabras is array (1..Max) of String(1..Max);
type T_Lista_Est_Pal is record
  palabras: T_Vector_Palabras;
  cont: Natural;       --Número de palabras en la lista
end record;

type T_Todas_Palabras is array (1..Max) of T_Lista_Est_Pal;
```

Para cada palabra se han reservado el máximo número de caracteres (*Max*), pero solo se utilizan los caracteres los realmente necesarios. La palabra "FIND" es un String de 4 caracteres.

En un vector de *T_Todas_Palabras* se quieren almacenar palabras clasificadas por número de caracteres. Por ejemplo, en la posición 4 del vector sólo habrá Strings de longitud 4 (en el ejemplo, "SEEK" y "FIND").

Recordad que un String es un array de caracteres, y que para conocer la longitud de un String P, se utiliza la expresión *P'Length*. (p.e. para el String "FIND", la expresión *Length* devolvería 4), y *P'First* y *P'Last* para los índices del primer y último carácter.

Supongamos el vector de todas las palabras clasificadas (*T_Todas_Palabras):*

Long	Lista de palabras
1..3	()
4	(FIND, SEEK)
5	()
6	(RANDOM, SLEUTH)
7	()
8	(DIAGONAL, VERTICAL)
9	(WIKIPEDIA)
10	(HORIZONTAL, WORDSEARCH)

Significa que: No hay palabras de tamaños 1, 2 y 3, de 4 caracteres hay dos palabras (SEEK y FIND), de 5 tampoco hay palabras.

Se puede comprobar que, para cada posición del vector, sólo tratamos la palabra con el tamaño expresado en el índice del vector.

Implementa el procedimiento *Insertar_Palabra* que, dados un vector de palabras clasificadas y una palabra concreta, inserta en la lista estática de palabras correspondiente (que está ordenada alfabéticamente) la palabra indicada. Si la palabra ya existiera en la lista, la deja intacta.

```
procedure Insertar_Palabra (VP : in out T_Todas_Palabras; P: in String);
```

Por ejemplo, con el vector anterior y la palabra 'HOME', el vector modificado sería:

Long	Lista de palabras
1..3	()
4	(FIND, **HOME**, SEEK)
…	…

PROGRAMACIÓN BÁSICA – Examen
Evaluación GLOBAL
19 de enero de 2022

PARTE 3

PREGUNTA 5. (2 puntos) Listas dinámicas (t= 45 min.)

En un vector (**T_Todas_Palabras**) se quieren almacenar palabras clasificadas (**A_Lista_Din_Pal**) por número de caracteres. Las estructuras de datos que lo representa son:

- Una lista (dinámica) de palabras (**A_Lista_Din_Pal**), que contiene Strings de 30 caracteres.

```
Max: constant Natural := 30;

type T_Nodo_Pal;
type A_Lista_Din_Pal is access T_Nodo_Pal;
type T_Nodo_Pal is record
  Pal: String(1..Max);
  Sig: A_Lista_Din_Pal;
end record;
```

- Un vector con todas las palabras clasificadas (**T_Todas_Palabras**). En la posición 1 estará la lista dinámica de palabras de 1 letra, en la 2 la lista dinámica de palabras de 2 letras, las de 3 letras en la posición 3, etc.

```
type T_Todas_Palabras is array (1..Max) of A_Lista_Din_Pal;
```

Por ejemplo, supongamos el vector de todas las palabras clasificadas (*T_Todas_Palabras*):

Long	Lista de palabras
1..3	null
4	(FIND, SEEK)
5	null
6	(RANDOM, SLEUTH)
7	null
8	(DIAGONAL, VERTICA)
9	(WIKIPEDIA)
10	(HORIZONTAL, WORDSEARCH)

Significa que: No hay palabras de tamaños 1, 2 y 3, de 4 caracteres hay dos (FIND, SEEK), de 5 tampoco hay palabras, etc.

Se puede comprobar que, para cada posición del vector, sólo tratamos el String con el tamaño expresado en el índice del vector.

Para cada palabra se han reservado el máximo número de caracteres (*Max*), pero solo se utilizan los caracteres realmente necesarios. Por ejemplo, en la posición 4 del vector sólo se tratarán Strings de longitud 4 (en el ejemplo, "FIND" y "SEEK").

Implementa el procedimiento *Modificar_Palabra*, que, dados un vector de palabras clasificadas y una palabra concreta, si la palabra ya existe en la lista dinámica de palabras correspondiente, la borra de ésta. Si la palabra no existe en la lista, entonces la inserta ordenadamente. Tener en cuenta que las listas dinámicas están ordenadas alfabéticamente. Resolver con un único recorrido de la lista correspondiente.

```
procedure Modificar_Palabra (VP : in out T_Todas_Palabras; P: in String);
```

Recordad que un String es un array de caracteres, y para conocer la longitud del String P, se utiliza la expresión *P'Length*. (p.e. para el String "HOME", la expresión *P'length* devolvería 4), y *P'First* y *P'Last (P'range)* para los índices del primer y último carácter (rango de los índices). Por ejemplo, con el vector inicial VP, considerando los casos:

(A) P= "FIND", la elimina de VP

Long	Lista de palabras
1..3	null
4	(SEEK)

(B) P= "HOME", la inserta en VP

Long	Lista de palabras
1..3	null
4	(FIND, **HOME**, SEEK)