Manejo de Cadenas en Turbo Pascal.

1. Tipos de datos estáticos en Turbo Pascal (breve repaso).

Antes de comenzar haremos un breve repaso sobre los tipos de datos que soporta el Turbo Pascal, para ello veamos primero cómo se clasifican:

- Simples
  - Ordinales o Numerables.
    - Enteros (integer, longint, shortint, word, byte)
    - Lógicos (Boolean = (False, True))
    - Caracteres (Char)
    - Subrango (1..N, 1..M, 'a'..'z')
    - Enumerado
  - No Ordinales
    - Reales o de Punto Flotante (real, single, double, extended)

- Estructurados
  - Arreglos
  - Registros
  - Conjuntos
  - Archivos

- Cadenas de Caracteres (Strings)

Los tipos de datos Simples se caracterizan porque las variables pertenecientes a este tipo solamente pueden almacenar *un único* dato en forma simultánea.

Estos tipos se subdividen además en Ordinales y No Ordinales. El primer conjunto incluye todos los tipos en los cuales se puede establecer un orden y una numeración de los valores permitidos, y por lo tanto se puede determinar qué valor es el inmediato inferior o inmediato superior de otro valor conocido. Por ejemplo, supongamos que A es una variable de tipo integer, si A=2000, sabemos que el valor inmediato inferior es 1999 y el inmediato superior es 2001. Supongamos ahora que LETRA es una variable de tipo char (el cual analizaremos detalladamente más adelante), de valor 'F', intuitivamente sabemos que 'E' precede a LETRA y que el valor siguiente es 'G'.

En la subclase No Ordinales entran aquellos tipos en los que no se puede establecer qué valor es el inmediato superior o inferior de otro conocido. Por ejemplo, consideremos una variable de *punto flotante* B = 2000. Uno estaría tentado en decir que el valor que la precede es 1999, pero en este caso no es así. Como B es un dato perteneciente a los reales (matemáticamente hablando), entre 1999 y 2000 existen infinitos valores, por lo tanto no se puede establecer con precisión quién precede al 2000. (El número sería X=1999.999999999..... con infinitos decimales iguales a 9, aunque los matemáticos afirman que si un valor X tiene infinitas cifras decimales y todas iguales a nueve, el valor que se tiene es igual a la parte entera de X más uno, (int(X)+1), en nuestro caso sería 2000. Entonces estamos nuevamente en el comienzo del problema.)

En algunas ocasiones uno desea que una variable almacene más de un datos simultáneamente, por ello existen los tipos de datos que entran en la clase Estructurados. Dentro de esta clase se incluyen los arreglos y matrices, registros (o records), los conjuntos y los archivos.

Otra clase está formada por las variables de tipo Cadena o String, que nos permitirá almacenar y operar con cadenas de caracteres, es decir, con palabras y/o frases. En general la bibliografía la considera como una clase propia, pero, como veremos más adelante, se la puede considerar una subclase de los tipos estructurados.

2. TIPO DE DATO CHAR.

Antes de estudiar el tipo de dato String es conveniente analizar primero el tipo Char (o caracter). Este tipo de datos nos permite almacenar en una variable un caracter. Este caracter puede ser una letra (mayúscula o minúscula), un dígito ('0', '1',... '9'), un caracter especial ('@', '$', '%', etc.), un símbolo ('ø', '', '', etc.) o un caracter de control (Enter #13, Escape #27, Tabulación #9, etc.).

Como el idioma que "hablan" (o entienden) las computadoras está compuesto únicamente por números, los diseñadores de las primeras máquinas decidieron asociar cada caracter con un número. En un primer momento cada diseñador elegía una asociación arbitraria o en conveniencia con su proyecto, por lo tanto no existía una compatibilidad entre los distintos sistemas. Por ese motivó se optó por tomar una codificación estándar y que todos deberían respetar. Esta codificación se denomina ASCII (American Standard Code for Information Interchange; código estándar americano para intercambio de información). Actualmente se emplea el código ASCII extendido que permite la codificación de 256 caracteres.

En la tabla ASCII cada caracter tiene un orden fijo y preestablecido, por lo tanto se puede conocer qué caracter se encuentra antes o después de otro. Al final de este apunte se encuentra una copia de la tabla ASCII, pero si la resumimos obtendríamos algo como lo que sigue:

i)	Caracteres de Control	(beep, tabulación, retorno de carro, escape, etc.)
ii)	Espacio	(caracter #32)
iii)	Símbolos I	('!', ' " ', '#', '$'. ....)
iv)	Dígitos	('0' .. '9')
v)	Símbolos II	(':', ';', '<', ...)
vi)	Letras Mayúsculas	('A' .. 'Z')
vii)	Símbolos III	('[', '\', ']', ...)
viii)	Letras Minúsculas	('a' .. 'z')
ix)	Caracteres especiales	(vocales acentuadas, símbolos monetarios, 'ñ', 'Ñ', etc.)
x)	Caracteres Gráficos	
xi)	Símbolos matemáticos	(alfa, beta, gama, >=, <=, unión, intersección, etc.)

Si observamos detenidamente, podemos ver que existen dos grupos de Letras, uno con las letras mayúsculas y otro con las minúsculas, por lo tanto, por ejemplo, el caracter 'A' es distinto al caracter 'a' ya que se encuentran en distinta posición. En forma más general, 'A' < 'a'. Es decir, todas las letras mayúsculas son menores que las letras minúsculas, por ejemplo, 'Z' < 'a', es decir, 'Z' está antes que 'a', concepto que no coincide con lo que aprendimos en la escuela primaria. Esta observación la deberemos tener en cuenta más adelante cuando, por ejemplo, quisiésemos ordenar una lista de palabras en orden alfabético.

2.1. Manejo de variables de tipo Char.

- Función Chr() y operador #.

Como es lógico, el Turbo Pascal nos provee de una función que nos devuelve el caracter ASCII ubicado en una determinada posición de la tabla. Esta función es Chr(). Cuando se invoca la función se debe pasar como argumento un valor de tipo Byte que representará la posición del caracter dentro de la tabla.

Ejemplo:

```
{ El siguiente programa muestra todos los caracteres visibles de la tabla ASCII }
{ Los caracteres de Control no se muestran }
{ Si se desea ver toda la tabla, el ciclo for deberá comenzar en 0 (cero) }
```

```
program Funcion_Char;
var
      Caracter: char;
      i         : byte;
begin
      writeln(' Caracteres visibles de la tabla ASCII ');

      for i:=32 to 255 do
      begin
        Caracter := Chr(i);          {Caracter almacenará el caracter ASCII ubicado en la posición i }
        write(i:5,'=', Caracter:2);
      end;
      writeln;

end.  {program  Funcion_Char}
```

Existe otro modo de indicar que el contenido de una variable char sea el caracter ubicado en una determinada posición, y es por medio del operador #. Por ejemplo, si quisiéramos almacenar en una variable el símbolo que representa la tecla Enter (ASCII 13), se puede escribir:

```
Tecla_Enter := #13;
```

Como se ve, existe una semejanza entre la función Chr() y el operador #, ya que los dos nos devuelven el caracter ASCII ubicado en una determinada posición. La diferencia que existe entre ambos es que el operador # siempre debe ir acompañado con un número (ej. #1, #65, #255), por lo tanto se debe fijar cuando se hace el programa y no se podrá cambiar en tiempo de ejecución. En cambio, como vimos, la función Chr() es más genérica y nos permite cambiar su argumento a medida que se ejecuta el programa. Resumiendo, a continuación se muestra un código *NO VALIDO*:

```
for i:= 32 to 255 do
begin

      Caracter := #i;        {<---------- Esto NO está permitido}

      write(i:5,'=', Caracter:2);
end;
```

- Función Ord().

La función Ord() es la inversa de la función Chr(), dado un caracter como argumento nos devolverá su posición en la tabla ASCII. Por ejemplo:

```
Ord('A')  = 65
Ord('a')  = 97
Ord('á')  =160
Ord(#255)  =255
```

- Uso de Char como control de un ciclo For.

Como indicamos al comienzo los tipos char son Ordinales o Numerables, por lo tanto se pueden usar como variables de control en un ciclo For, es decir, podemos hacer que dicha variable se "desplace" por la tabla ASCII entre dos valores extremos dados como dato. Veamos dos ejemplos:

```
program Ciclo_For_con_Char;
var
    i          : char;
    Posicion   : byte;
begin

    for i:= 'A' to 'Z' do
        begin
          Posicion := Ord(i);
          writeln(i:2, '=', Posicion:4);
        end;
end.
```

```
program Ciclo_For_con_Char;
var
    i          : char;
    Posicion   : byte;
begin

    for i:= 'Z' downto 'A' do
        begin
          Posicion := Ord(i);
          writeln(i:2, '=', Posicion:4);
        end;
end.
```

En el ciclo for se hace variar a *i* (de tipo char) entre el caracter 'A' y el caracter 'Z', por lo tanto se recorrerán todas las letras mayúsculas del alfabeto inglés. En el interior del ciclo, se obtiene la posición de cada letra dentro de la tabla ASCII y se muestra en pantalla. En el primer programa la secuencia comienza en 'A' y finaliza en 'Z'. En el segundo caso comienza en 'Z' y finaliza en 'A'. Como 'A' < 'Z' se debe usar la instrucción *downto*.

- Función ReadKey.

La función ReadKey (unidad Crt) hace una pausa hasta que el usuario pulse una tecla y nos devuelve el(los) caracter(es) asociado(s) a la tecla que el usuario pulsó. Veamos un ejemplo en el cual hacemos un ciclo hasta que el usuario pulse la tecla Escape (#27). Dentro del ciclo mostramos el caracter que nos entrega el ReadKey (y su código).

```
program Codigo_tecla;
uses crt;
const
    Tecla_Escape = #27;       { Se define una constante que almacenará el ASCII 27 }
var
   Tecla: char;
begin
    clrscr;
    writeln('Pulse una tecla y le diré su código.');
    writeln('Pulse Escape para salir.');

    repeat
        Tecla := readkey;
        writeln(Tecla, '= ', Ord(Tecla));
   until Tecla = Tecla_Escape;
end.
```

NOTA: Algunas teclas tienen asociados dos caracteres, como por ejemplo, las flechas (←, →), las teclas F1..F12, Page Up, Page Down, etc. En estos casos el ciclo repeat se ejecuta dos veces, ya que el primer caracter siempre es igual al #0, por lo tanto distinto de #27. Como el readkey lee de a un caracter por vez, en la primer pasada leerá el primer caracter (#0), y luego leerá el segundo caracter que sigue almacenado en la memoria de teclado (o buffer de teclado).

- Función UpCase().

La función UpCase() nos permite cambiar letras minúsculas a letras mayúsculas. Esta función toma un argumento de tipo char y devuelve la correspondiente equivalencia en mayúsculas de la expresión. Si la

expresión ya está en mayúsculas o no tiene ningún caracter mayúscula equivalente se devuelve el caracter sin modificar.

Los dos ejemplos que se muestran a continuación son equivalentes. En ambos casos se realiza una tarea (ej. cálculo del cubo de un valor entero), pero el programa termina cuando el usuario pulsa la tecla 'S'.

```
program N_al_Cubo;
uses Crt;
var
    Tecla      : char;
    N          : integer;

begin

    repeat
        writeln('Ingrese N: ');
        read(N);
        writeln(N, '^3 = ', N*sqr(N));

        writeln('¿Desea Salir? (S/N)');
        Tecla := readkey;

    until (Tecla='s') or (Tecla='S');

end.
```

```
program N_al_Cubo;
uses Crt;
var
    Tecla      : char;
    N          : integer;

begin

    repeat
        writeln('Ingrese N: ');
        read(N);
        writeln(N, '^3 = ', N*sqr(N));

        writeln('¿Desea Salir? (S/N)');
        Tecla := readkey;

    until UpCase(Tecla)='S';

end.
```

Antes de terminar el ciclo se le indica al usuario que pulse la tecla 'S' para salir. La tecla pulsada se almacena en la variable Tecla. En el primer programa debemos considerar las dos alternativas posibles, ya que no sabemos de antemano qué caracter 'S' (mayúscula o minúscula) ingresará el usuario. En el segundo programa hacemos uso de la función Upcase() y evitamos la doble pregunta. Si el usuario ingresa la 's', UpCase la transformará en 'S', por lo tanto, deberemos consultar únicamente por esta alternativa.

3. TIPO DE DATO STRING o CADENA DE CARACTERES.

Como vimos, el tipo Char sólo nos permite almacenar un solo caracter en cada variable, por lo tanto, si deseamos almacenar nombres, frases o cualquier otro dato que contenga más de un caracter no lo podemos hacer. Por tal motivo, en Turbo Pascal se cuenta con un tipo estructurado STRING (o cadena).

Un String o cadena de caracteres es una secuencia de datos de tipo char cuya longitud física puede variar entre 1 y 255 caracteres. Se puede hacer una analogía entre el tipo string y un arreglo de variables de tipo char, ya que estructuralmente son similares, pero la diferencia principal reside en que el lenguaje provee un biblioteca de funciones y procedimientos para el manejo de string que no se pueden hacer con los arreglos. (Si se emplean arreglos el programador deberá crear cada una de las rutinas que el lenguaje provee para los tipos string).

3.1. Declaración de variables de tipo String.

Cuando se declara una variable de tipo string se deberá indicar la longitud máxima que tendrá esa cadena de caracteres. La longitud máxima que se puede emplear es de 255 caracteres.

Por ejemplo:

```
var
    Texto      : string[100];
    Nombre     : string[20];
    Apellido   : string[20];
```

En este caso se declararon cuatro variables de tipo string. La cadena más larga que puede contener la variable Texto es de 100 caracteres. En el caso de Nombre y Apellido almacenarán valores de hasta 20 caracteres.

3.2. Asignación de Valores.

Para cargar datos en una cadena se pueden emplear tres alternativas:

a) declarando una constante:

```
const
        Texto_Constante = 'A mi no me pueden modificar';
```

b) Asignado a una variable dentro del programa:

```
Texto := 'Pulse una tecla para continuar';
Frase := Texto;
```

c) Usando RealdLn o Read:

```
write('Ingrese su nombre: ');
Readln(Nombre);
write('Ingrese su Apellido: ');
Readln(Apellido);
writeln('Usted se llama :', Nombre, ' ', Apellido);
```

En los casos b) y c), si el dato que se pretende almacenar tiene una longitud mayor que la longitud física de la variable, se truncará en dato original y solo se almacenarán los caracteres que entren.

Ejemplo:

Consideremos las siguientes variables:

```
var
    Texto      : string[100];
    Materia    : string[20];
```

Ahora asignamos a ambas variables la misma frase:

```
Texto := 'Principios de Computadoras I';
Materia := 'Principios de Computadoras I';
```

La frase *'Principios de Computadoras I'* tiene una longitud de 28 caracteres, por lo tanto sólo usamos 28 de los 100 caracteres reservados para la variable Texto. Pero la variable Materia tiene una longitud máxima de 20 caracteres, entonces el contenido de la variable Materia es:

```
Materia = 'Principios de Comput'
```

El resto de la frase *'adoras I'* se perdió. Este detalle debe tenerse en cuenta en ciertos casos como el siguiente:

```
Materia := 'Principios de Computadoras I';
Texto := Materia;
```

Si se ejecutan estas dos líneas en el orden mostrado, la variable Texto, que tiene capacidad para almacenar la frase completa, sólo recibirá por parte de la variable Materia los primeros 20 caracteres, por lo tanto Texto también almacenará la frase truncada.

3.3. Longitud de las Cadenas - Función Length().

Hasta ahora solo hablamos de la *longitud física* de las cadenas, que coincide con la cantidad de caracteres reservados para la variable, o sea, su longitud máxima.

Como vimos, no siempre se usan todos los caracteres reservados. La *longitud lógica* es la que nos indica la cantidad de caracteres usados. En otras palabras, la longitud lógica es la cantidad de caracteres que posee el contenido de una variable de tipo string.

Consideremos nuevamente a las variables Texto y Materia, y la siguiente asignación:

```
Texto    := 'Principios de Computadoras I';
Materia  := 'Principios de Comput';
```

Las longitudes son:

Longitud Física de Texto = 100;
Longitud Lógica de Texto = 28

Longitud Física de Materia = 20;
Longitud Lógica de Materia = 20;

Muchas veces nos interesa saber cuál es la longitud lógica de una variable, por eso, Turbo Pascal posee una función llamada Length() que nos devuelve dicho valor.
Por ejemplo:

```
Length(Texto) = 28
Length(Materia) = 20
```

3.4. Operaciones entre Cadenas.

Las operaciones básicas que se pueden desarrollar entre cadenas son: asignación, comparación y concatenación. La asignación ya la analizamos en el punto 3.2. Cuando comparamos dos cadenas podemos averiguar si son iguales (=), distintas (<>) o si una es mayor o menor que otra (> ó <). En el caso de la concatenación (+) unimos dos o más cadenas para formar una única cadena.

Las comparaciones de las cadenas se hacen según el orden de los caracteres en el código ASCII y con los operadores de relación. La regla general de comparación es la siguiente: Las dos cadenas se comparan de izquierda a derecha hasta que se encuentran dos caracteres diferentes. El orden de las dos cadenas es el que corresponde al orden de los dos caracteres diferentes. Si las dos cadenas son las mismas pero una es más corta que la otra, entonces la más corta es menor que la más larga.

Ejemplo:

```
'TURBO PASCAL' < 'turbo pascal'        'tension' < 'tensión'
'TURBO PASCAL' < 'Turbo Pascal'        'tension' < 'tensiones'
'Turbo Pascal' < 'turbo pascal'        'tensiones' < 'tensión'
```

El Turbo Pascal nos permite unir dos o más cadenas en una sola cadena. Esta operación se llama concatenación y se realiza con el operador concatenación (+) o la función concat(). Ejemplos:

'Principios de ' + 'Computadoras I' → 'Principios de Computadoras I'
concat('Principios ', 'de ', 'Computadoras I') → 'Principios de Computadoras I'

'Principios' + 'de' + 'Computadoras I' → 'PrincipiosdeComputadoras I'
'Principios' + ' ' + 'de' + ' ' + 'Computadoras I' → 'Principios de Computadoras I'
'Principios' + #32 + 'de' + #32 + 'Computadoras I' → 'Principios de Computadoras I'
(Nota: ASCII #32 = Espacio)

Al resultado de la operación lo podemos almacenar en otra variable de tipo string, pero si este resultado supera la capacidad física de la cadena destino entonces se truncará el resultado y se almacenarán solamente los caracteres que entren. Ejemplo:

```
var
    Nombre,
    Apellido,
    Nom_y_Ap    : string[15];

begin
    Nombre := 'Tadeo Isidoro';      { Length( Nombre )= 13 }
    Apellido := 'Cruz';             { Length( Apellido ) = 4 }

    Nom_y_Ap := Nombre + #32 + Apellido;

    writeln(' Nombre Completo = ' , Nom_y_Ap);
end.
```

La salida por pantalla es la siguiente:

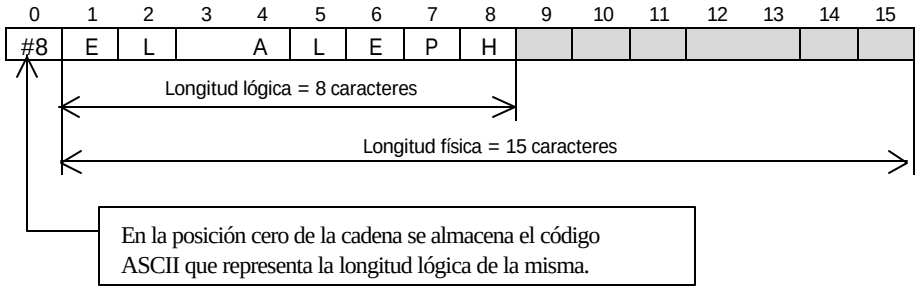
```
Nombre Completo = Tadeo Isidoro C
```

Como podemos ver el resultado está truncado ya que su longitud superaba los 15 caracteres máximos que puede almacenar la variable Nom_y_Ap. La siguiente tabla resume lo ocurrido en el programa:

Variable y / o Expresión	Contenido	Longitud Lógica
Nombre	'Tadeo Isidoro'	13 caracteres
Apellido	'Cruz'	4 caracteres
Nombre + #32 + Apellido	'Tadeo Isidoro Cruz'	18 caracteres
Nom_y_Ap	'Tadeo Isidoro C'	15 caracteres

3.5. Analogía entre Cadenas de Caracteres y Arreglos.

Analicemos la estructura interna de una variable de tipo string:



Como podemos ver la estructura es análoga a la de un arreglo de caracteres, con la única diferencia que en este caso el lugar cero de la cadena se utiliza internamente para codificar la longitud lógica de la misma.

Basándose en esta característica Turbo Pascal nos permite acceder a cada uno de los caracteres de la cadena como si ésta fuera un arreglo común. La sintaxis es la misma que la empleada para el manejo de arreglos. Por ejemplo:

```
var
      Cadena_1     : string[25];
      Cadena_2     : string[20];

begin
      Cadena_1 := 'OXIGENO';
      Cadena_2 := 'oxígeno';

      Cadena_1[1] := Cadena_2[1];
      Cadena_1[3] := Cadena_2[3];
      Cadena_1[5] := Cadena_2[5];
      Cadena_1[7] := Cadena_2[7];

      Cadena_2[1] := 'ø';
      Cadena_2[7] := 'ø';

      writeln(' Cadena_1 = ' , Cadena_1);
      writeln(' Cadena_2 = ' , Cadena_2);
end.
```

El resultado obtenido es:

```
      Cadena_1 = oXíGeNo
      Cadena_2 = øxígenø
```

Esta característica puede ser usada para convertir una palabra o frase escrita en minúsculas a mayúsculas haciendo uso de la función Upcase. Básicamente lo que se debe hacer es, por medio de un ciclo, aplicar la función Upcase a cada caracter de la cadena:

```
for i:= 1 to Length(Cadena) do
   Cadena[i] := UpCase( Cadena[i] );
```

4. Procedimientos y Funciones de Cadenas.

El Turbo Pascal cuenta con un conjunto de procedimientos y funciones que nos facilitan el manejo de las cadenas. Estos procedimientos nos permitirán copiar, insertar y/o borrar segmentos de cadenas, convertir cadenas en números y números en cadenas, etc.

4.1. Procedimiento Delete().

El procedimiento Delete() nos permite eliminar una porción definida de una cadena. La sintaxis es:

```
Delete( Cadena,  Posicion_inicial,  Cantidad_caracteres);
```

- Cadena (tipo string) es la variable que se desea modificar. Debe ser una variable.
- Posicion_inicial (tipo integer) indica la posición del primer caracter que se desea eliminar.
- Cantidad_caracteres (tipo integer) es la cantidad de caracteres que se desean eliminar a partir de Posicion_inicial.

Si Posicion_inicial es mayor que la longitud de Cadena no se eliminará ningún caracter. Si Cantidad_caracteres especifica más caracteres que los existentes desde la posición inicial hasta el final de la cadena se borrarán todos los caracteres desde Posicion_inicial hasta el final.

Ejemplos:

```
cadena := 'Principios de Computadoras I';
Delete(cadena, 9, 11);
⇒ cadena = 'Principitadoras I'

cadena := 'Principios de Computadoras I';
Delete(cadena, 1, 10);
⇒ cadena = ' de Computadoras I'

cadena := 'Principios de Computadoras I';
Delete(cadena, 29, 5);
⇒ cadena = 'Principios de Computadoras I'
```

4.2. Procedimiento Insert().

Por medio del procedimiento Insert() podemos incluir una cadena dentro de otra a partir de una posición deseada. La sintaxis es:

Insert(Cadena_Fuente, Cadena_Destino, Posición)

- Cadena_Fuente (tipo string) es la cadena que se desea insertar. Puede ser una variable o una constante.
- Cadena_Destino (tipo string) es la cadena donde se insertará la Cadena_Fuente. Debe ser una variable.
- Posición (tipo integer) indica la posición dentro de la Cadena_Destino donde se producirá la inserción.

Ejemplos:

```
cadena := 'PrincipiosComputadoras I';
Insert(' de ', cadena, 11);
⇒ cadena = 'Principios de Computadoras I'

cadena1 := 'Principios ';
cadena2 := 'de Computadoras I';
Insert(cadena1, cadena2, 1);
⇒ cadena1 = 'Principios '
  cadena2 = 'Principios de Computadoras I'
```

4.3. Función Pos().

Esta función nos permite determinar si una cadena está contenida en otra. Si la cadena está, la función nos devuelve un valor entero que indica la posición donde comienza la cadena buscada en la cadena fuente. Si la cadena no existe se devuelve el resultado 0 (cero). La sintaxis es:

Pos(Cadena_Buscada, Cadena_Fuente);

Ejemplo:

```
var Cadena          : string[50];
    a, b, c, d, e, f : integer;
```

```
begin
    Cadena := ' Una voz en la fuga cósmica ';

    a  := Pos('una',  Cadena);
    b  := Pos('voz',  Cadena);
    c  := Pos(#32,  Cadena);
    d  := Pos('cósmica',  Cadena);
    e  := Pos('cosmica',  Cadena);
    f  := Pos('Una',  Cadena);

    writeln('a=', a:3,  'b=', b:3,  'c=', c:3,  'd=', d:3,  'e=', e:3,  'f=', f:3 );
end.
```

La salida será:
a=0 b=5 c=4 d=20 e=0 f=1

Si la cadena buscada está repetida, la función Pos() sólo nos indicara la posición de la primer cadena encontrada comenzado desde la izquierda.

4.4. Función Copy().

Con esta función podemos obtener un fragmento de una cadena indicando la posición de comienzo y la cantidad de caracteres deseados. La sintaxis es:

Copy(Cadena, Posicion_inicial, Cantidad_caracteres);

- Cadena (tipo string) es la desde donde se obtendrá el segmento deseado. Puede ser una variable o una constante.
- Posicion_inicial (tipo integer) indica la posición del primer caracter del segmento.
- Cantidad_caracteres (tipo integer) es la cantidad de caracteres que tendrá el segmento extraído.

No debemos confundir esta función con el procedimiento Delete(). Copy() solamente lee (sin modificar) la cadena fuente y nos devuelve el segmento que deseamos. Delete() modifica la cadena fuente eliminando el segmento indicado.

Ejemplos:

```
cadena := 'Nucleogénesis';

segmento1 := Copy( cadena, 1, 6);   ⇒   segmento1 = 'Nucleo'
segmento2 := Copy( cadena, 7, 5);   ⇒   segmento2 = 'génes'
segmento3 := Copy( cadena, 14, 2);  ⇒   segmento3 = ''   (cadena vacía)
```

4.5. Procedimiento Str().

Este procedimiento convierte una expresión numérica en su equivalente es cadena. Recordemos que las expresiones numéricas y las cadenas pueden ser equivalentes para el ser humano pero no lo son para la computadora. Por ejemplo, el número 1974 y la cadena '1974' son equivalentes pero no son iguales ya que se codifican de distinta manera. '1974' no es interpretado como un número sino como una secuencia de 4 caracteres.

La sintaxis de Str() es:

Str(Expresión_Numérica, Cadena);

- Expresión_Numérica es la expresión que se desea convertir. Puede ser de cualquier tipo numérico conocido (integer, longint, single, real, double, etc.)

- Cadena (tipo string) es la variable donde se almacenará la cadena equivalente a la expresión numérica.

Ejemplos:

```
num := 123.456;
Str( num, cadena);                    ⇒   cadena = '1.23456000000000E+0000'
Str( num:7:3, cadena);                ⇒   cadena = '123.456'
Str( num:20:10, cadena);              ⇒   cadena = '          123.4560000000'
Str( num+7000:8:3, cadena);           ⇒   cadena = '7123.456'
Str( 3.1415926:4:2, cadena);          ⇒   cadena = '3.14'
```

Como podemos observar en los ejemplos la conversión de una expresión numérica en cadena se realiza utilizando el mismo criterio que se aplica en la instrucción write (ó writeln) cuando se desea mostrar un número en la pantalla. En estos ejemplos supusimos que el tamaño de la variable cadena es lo suficientemente grande para no tener problemas de truncamiento en el resultado.

4.6. Procedimiento Val().

Este procedimiento es el inverso al visto anteriormente, nos permite transformar una cadena de caracteres en una variable numérica. Para que esta conversión sea efectiva, el contenido de la cadena debe corresponderse a las reglas de escritura de números, es decir, la cadena sólo podrá contener dígitos (0..9), un punto, el signo menos y la letra E para simbolizar potencia 10 en notación científica. La sintaxis es:

```
Val( Cadena, Variable_Numérica, Código_Error )
```

- Cadena (tipo string) es la variable que contiene la expresión numérica entera o real que se desea convertir.
- Variable_Numérica es la variable donde se almacenará el resultado. Debe tener el mismo tipo que la expresión almacenada en la cadena.
- Codigo_Error (tipo integer) nos indicará si la conversión se pudo hacer o no. Si se pudo hacer toma el valor 0 (cero), en caso contrario contiene la posición del primer caracter de la Cadena que impide la conversión y en ese caso la Variable_Numérica no queda definida.

Ejemplos:

```
var
      cadena    : string[30];
      entero    : integer;
      e_largo   : longint;
      v_real    : real;
      error     : integer;
begin
      cadena := '123.456';
      Val( '2000', entero, error);          ⇒   entero = 2000          error = 0
      Val( cadena, v_real, error);          ⇒   v_real = 123.456       error = 0
      Val( cadena, entero, error);          ⇒   entero = ?             error = 4 (el punto)
      Val( ' 333', entero, error);          ⇒   entero = ?             error = 1 (el espacio)
      Val( '333 ', v_real, error);          ⇒   v_real = ?             error = 4 (el espacio)
      Val( '-0.3142e2', v_real, error);     ⇒   v_real = -31.42        error = 0
end.
```

Cualquier sugerencia sobre la redacción y/o contenido de este apunte será bienvenida.