Fundamentos de Informática:

Introducción a la programación en Pascal

Sergi Guillén Alonso
Técnico en Sistemas electrónicos e informáticos, y alumno de la Escuela Universitaria Salesiana de Sarriá (EUSS).

Barcelona*, abril de 2004*

Agradezco la colaboración en la elaboración de este libro a los compañeros Francesc Sort y Beatriz Álvarez por dejarme echar un vistazo a sus apuntes, a Laura Marsal por sus correcciones ortográficas y al personal del Área de Sistemas Informáticos y Comunicaciones de la EUSS por sus correcciones técnicas, y en especial a César Latorre, por su gran apoyo.

Dedicado a mis hermanos Pepe y Adelina

ÍNDICE

Introducción del autor

Era una tarde de verano y tenía 11 años. No recuerdo el título de la película, pero fue impresionante... trataba de tres niños que, con un ordenador y un programa, generaban una especie de burbuja que flotaba en el aire... se fueron a un desguace y construyeron una pequeña nave espacial. Metieron el ordenador dentro, y al iniciar el programa, se generaba la burbuja alrededor de la nave, haciendo que flotaran en el aire... y... en fin... se fueron al espacio a visitar otros planetas.

Después vinieron “juegos de guerra”, o “Tron”... ¡alucinantes!... pero...

A partir de aquel día... soñaba con tener un ordenador, aunque sabía de sobras que quedaba muy lejos ya que eran caros por aquel entonces...

Eran las navidades de 1990, y estaba toda mi familia, pero yo sólo veía aquella caja grande, ya usada, que ponía... “com”... “commm”... “commmo”... “commodore”!!!
Acababa de abrir el regalo...
Se trataba del “ordenador doméstico” Commodore VIC-20: el primero que introdujo el color en su gama... 16 fabulosos colores, con una resolución de 184x176 pixels y a 1MHz de velocidad!!! Programable en BASIC!!! Con ranura de cartuchos, cassette, y una RAM de 5Kbytes!!

Era de “segunda mano”. Unos amigos de mi hermana se lo habían cedido para darme la sorpresa... ¡cómo lo agradezco!

Estaba también mi prima, algo mayor que yo, que me ayudó a conectar el cassette, el alimentador, y el teclado ¡al televisor!

Lo sintonizamos y... allí empezó todo...
Venía un cartucho con un juego de laberintos y ratoncitos, pero... a mi no me entusiasmaba eso... yo quería construir algo grande con el ordenador!!!

Sin saberlo... yo quería programar...

Y así fue... con el ordenador venía el libro “VIC-20: PROGRAMMERS REFERENCE GUIDE”. Y aquel fue mi primer libro de programación...

¿os creeis que con 12 años, yo podía asimilar lo que era un FOR, un IF, un CASE, un NEXT, un POKE, un PEEK...

NO!!!

¿Pero qué hice...? NO tenía a NADIE que me enseñara... así que empecé a picar todos aquellos programas del libro que, por aquel entonces, los veía sin sentido...

“Calculadora... Adivina el número... Menú de comidas... Sonidos en tu televisión... Dibuja círculos... Dibuja figuras...”

Y aquí viene mi primera lección: debes picar TODOS y cada uno de los programas que te explican en las clases, y TODOS y cada uno de los programas que aquí te explico.

Aunque de entrada no los entiendas, debes empeñarte, y estudiarlos!!

Tienes a un grupo de profesores expertos en la materia, tienes muchos libros... entre ellos, éste, que es el que más te acercará al programa de la asignatura, tienes InterNET y acceso a la web de la escuela... ¡hacer consultas a los profesores es MUY recomendable y MUY importante para no perderte!

Y tienes... www.sergi2.com con la que pretendo ser compañero de realmente TODOS los alumnos de la EUSS.

Commodore VIC-20.
16 colores. 5 Kbytes RAM. 16 Kbytes ROM. Cassete, cartuchos y salida a TV. Y... SIN disco duro!!
www.vintage-computer.com

Este humilde libro de apuntes lo he escrito con mucha ilusión y entusiasmo, para acercarte a la ciencia de la programación.

Está escrito con un lenguaje “de amigos” para que lo encuentres más ameno.
No pretende ser “La Biblia del Pascal” ni mucho menos, si no una muy-fácil entrada para empezar a programar desde el 0 absoluto y... para ayudarte a aprobar la asignatura!!

Le tendrás que poner empeño si te cuesta la asignatura, pero SEGURO que lo conseguirás si le dedicas regularmente unas 5 horas semanales, a parte de las de clase.

Mi consejo, por experiencia, es que **NO FALTES A NINGUNA CLASE**, pero si lo haces, ponte al día con apuntes de tus compañeros y utiliza este arma tan poderosa que tienes en tus manos o en pantalla!

Y... ánimo, porque vale la pena no sólo estudiar informática y programación, si no, sacarse una carrera.
Te ayudará muchísimo tener una ingeniería, en este mundo de futuro cada vez más competitivo, donde nacen lenguajes de programación, tecnologías de comunicación, sistemas micro-electrónicos y sistemas mecánicos autónomos cada día...

NO te rindas, NO pierdas el tiempo, y disfruta con tus estudios!!!

De un compañero,

Sergi Guillén
www.sergi2.com
sergi2@sergi2.com

Introducción al uso de Dev-Pascal

Para introducirte a la programación en Pascal con Dev-Pascal, crea una carpeta llamada "*PASCAL*" en tu escritorio.

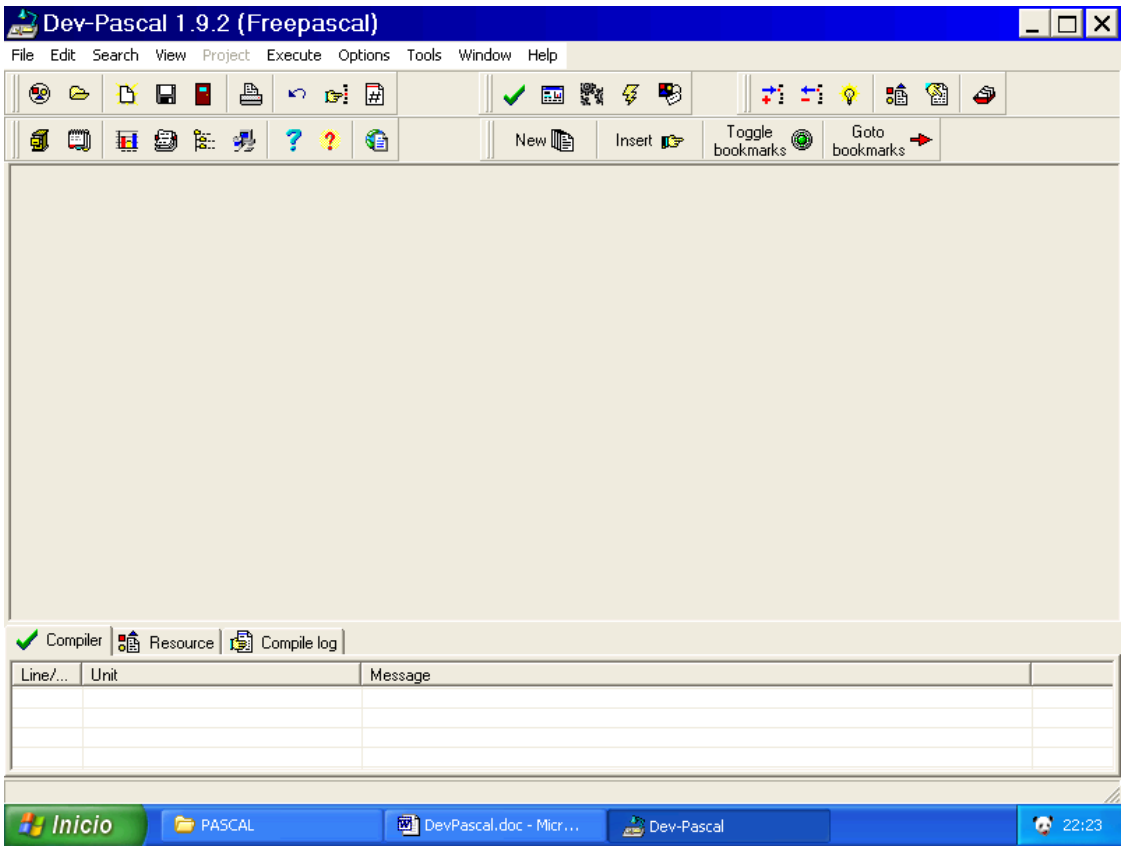
Esta carpeta, contendrá todos los programas que vayamos haciendo.

Crea otra carpeta dentro de la carpeta "*PASCAL*", que se llame "*HolaMundo1*".

En la carpeta "*HolaMundo1*", será donde crearemos nuestro primer programa.

Ahora, arranca Dev-Pascal.

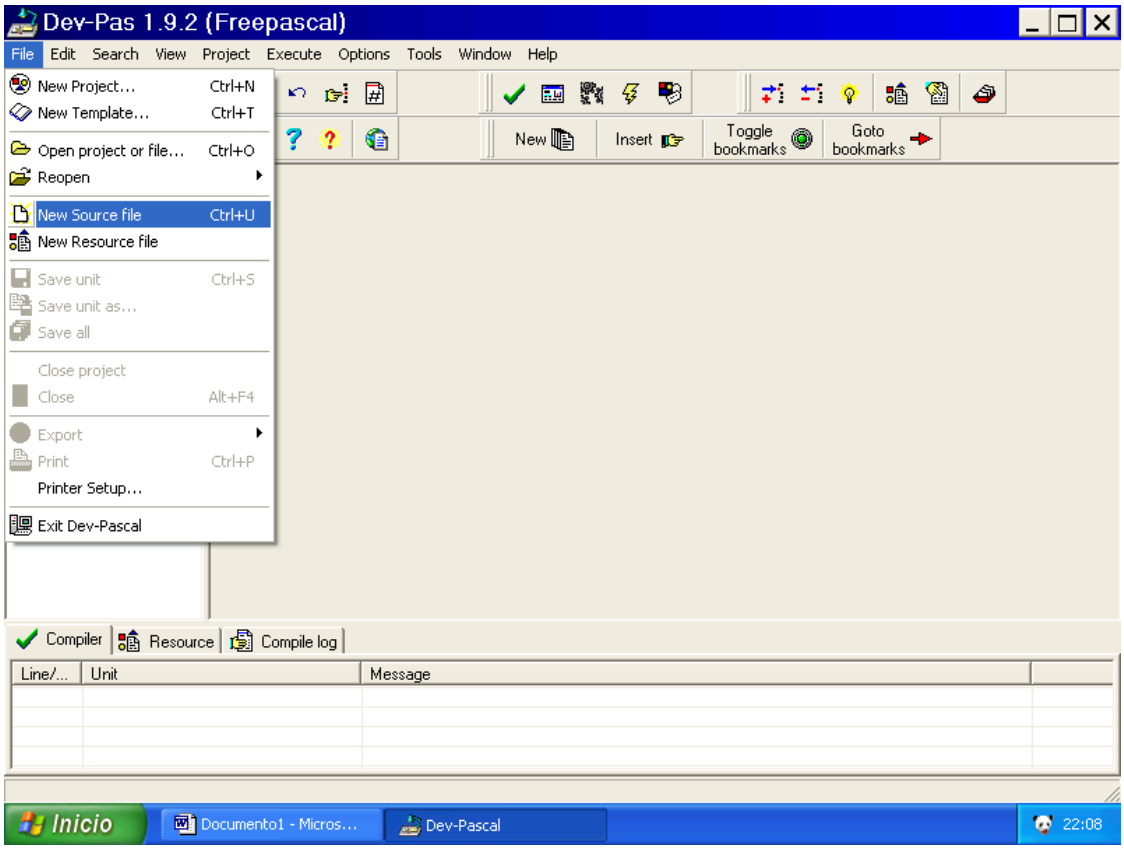
Cuando arrancamos Dev-Pascal, nos encontraremos con el siguiente entorno:



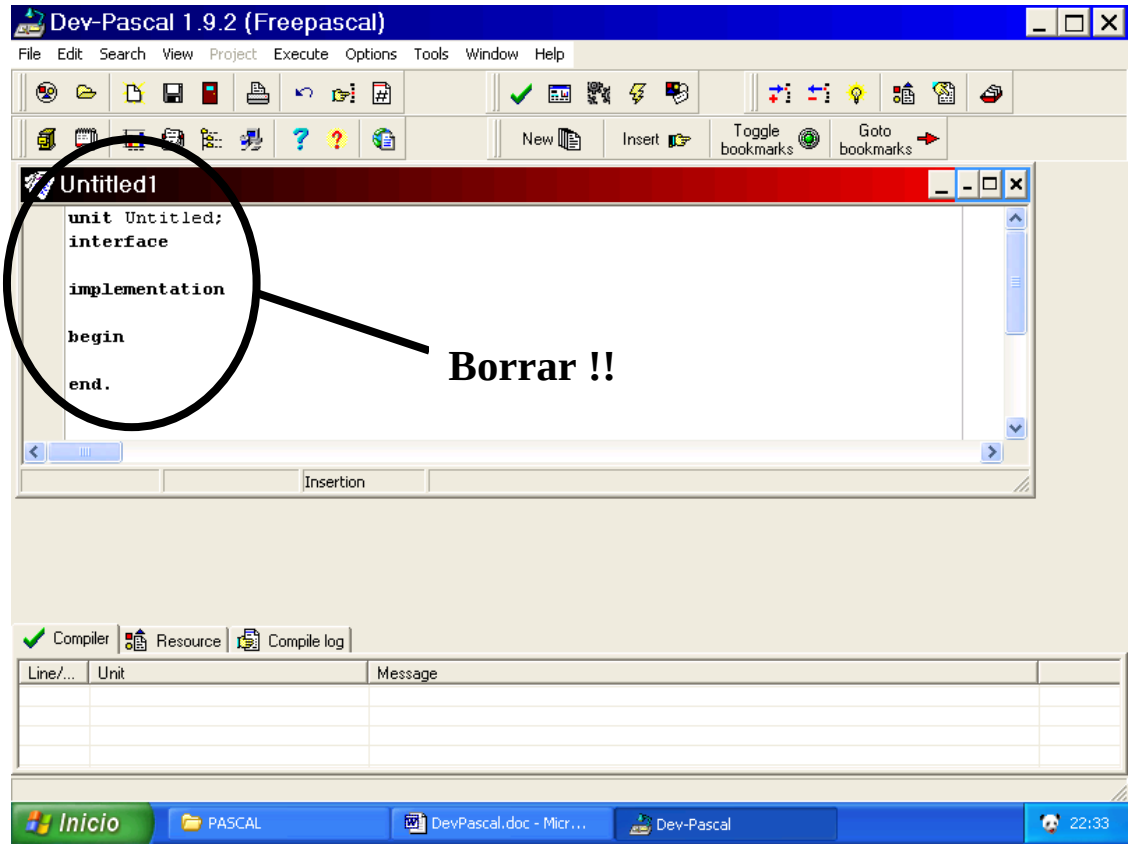
Si aquí, ves algún fichero abierto ciérralo.

Ahora crearemos nuestro primer programa con Dev-Pascal.

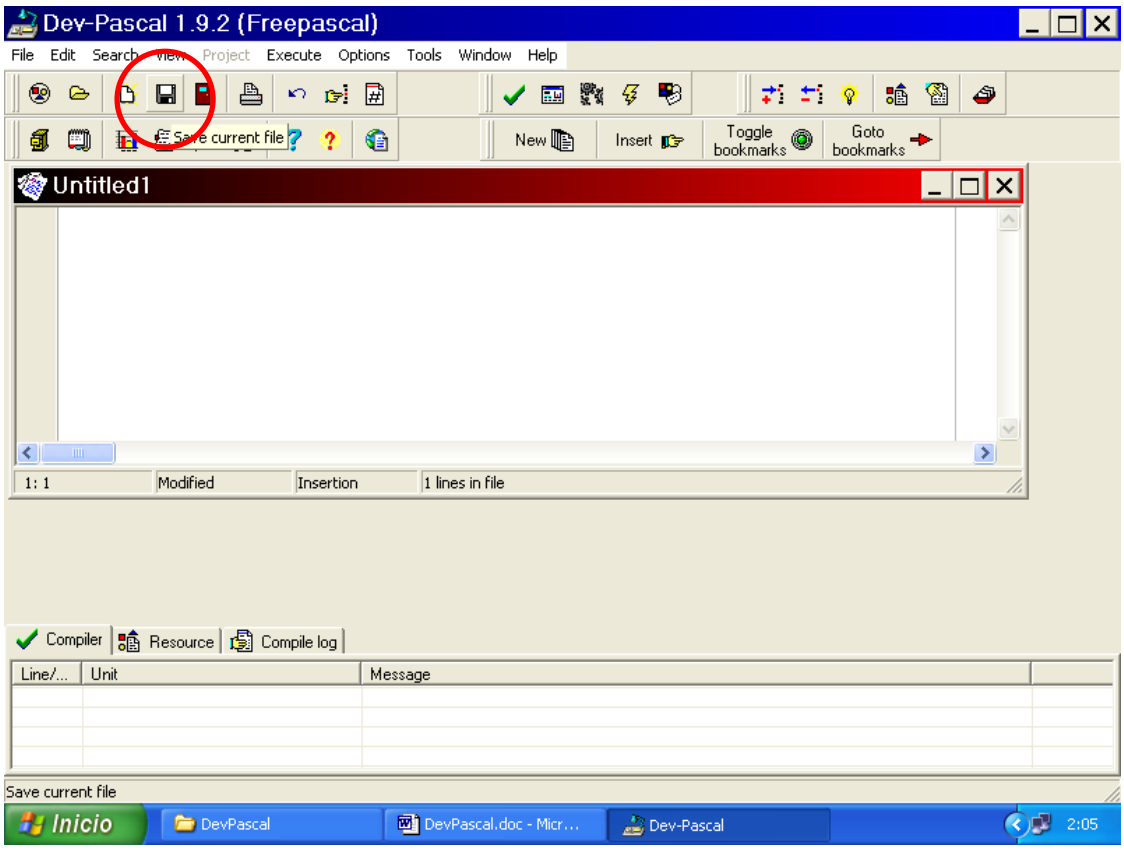
Nos vamos a **File→New Source File**:



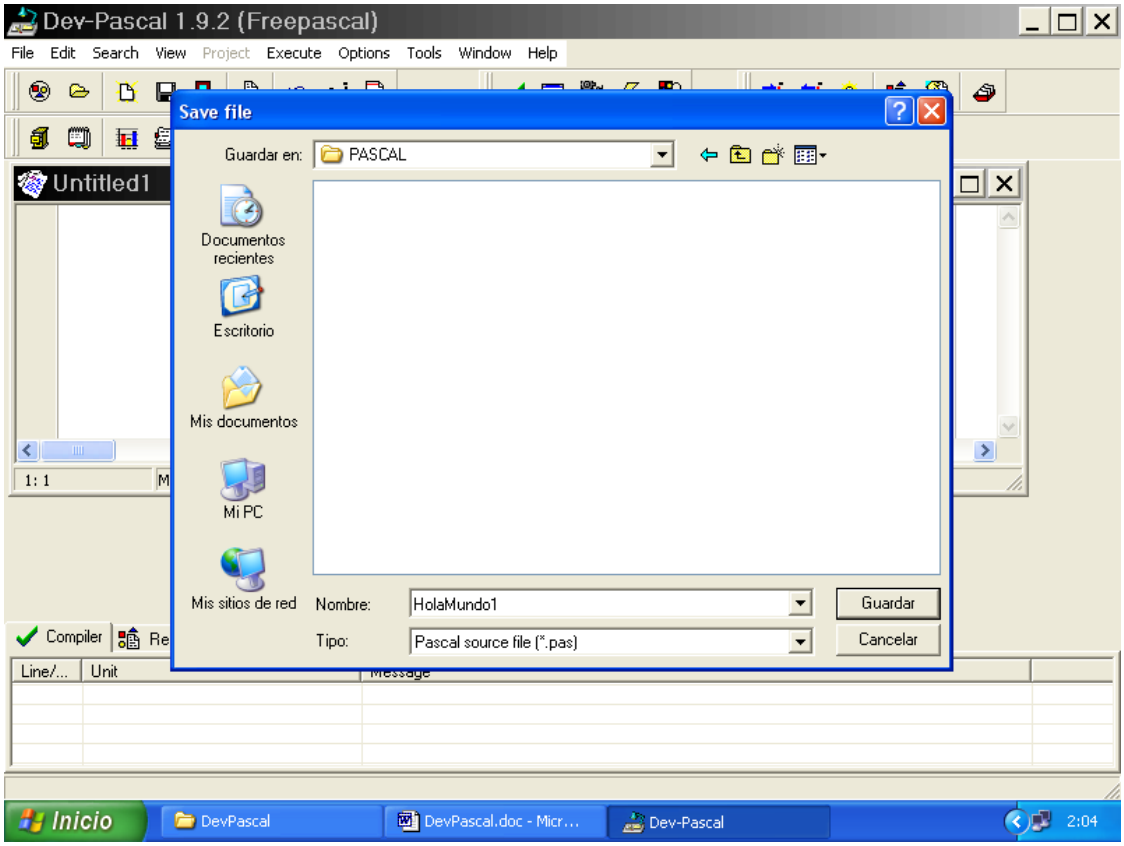
Cuando hacemos dicha operación, inmediatamente nos aparece nuestro programa “en blanco”, aunque con una base ya introducida, pero BORRALO... lo vamos a cambiar:



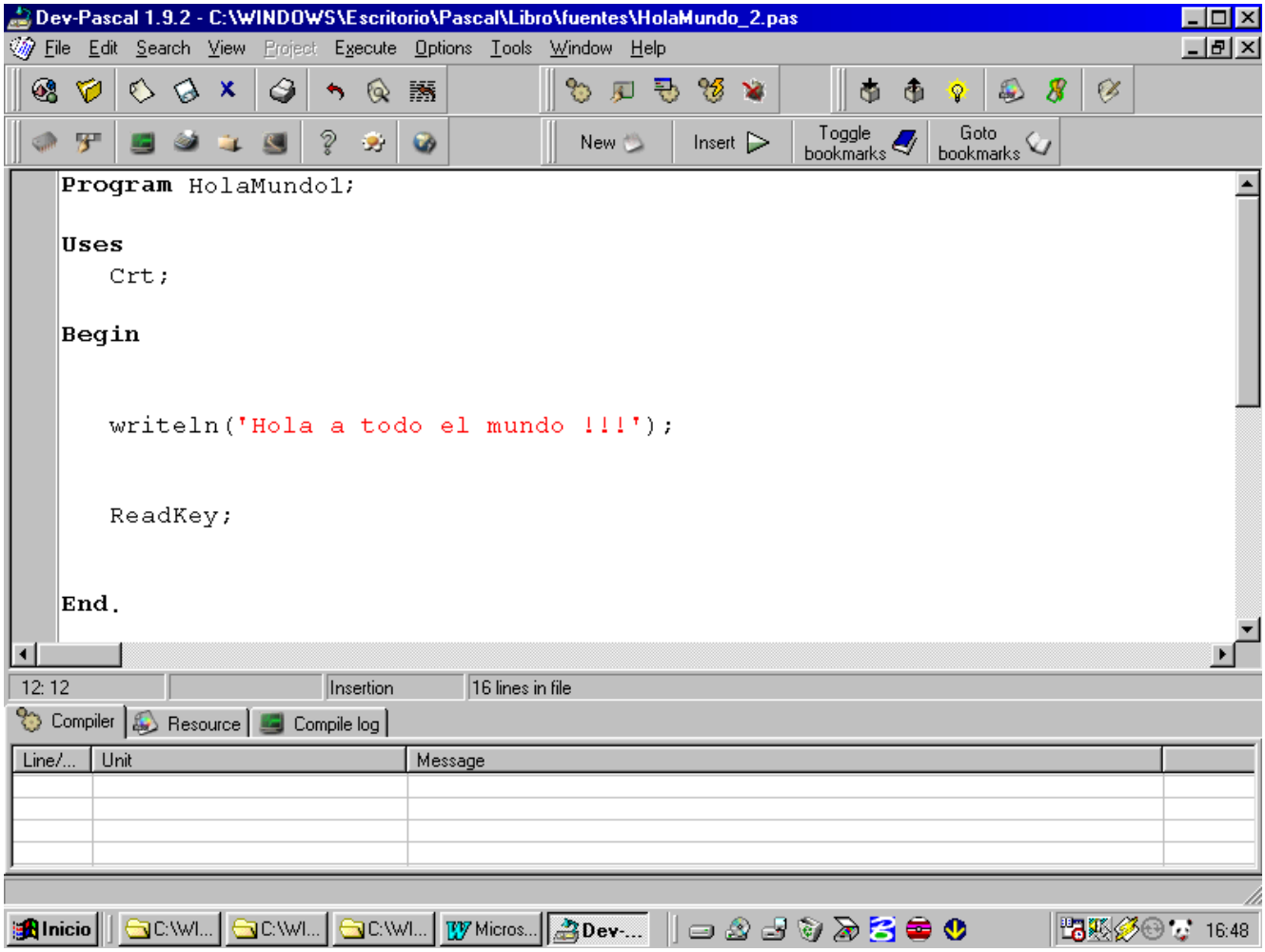
Es recomendable, grabar YA este fichero, dándole un nombre, y así las próximas veces que pulsemos el icono del “disquette”, se grabará sin preguntarnos su nombre de nuevo:



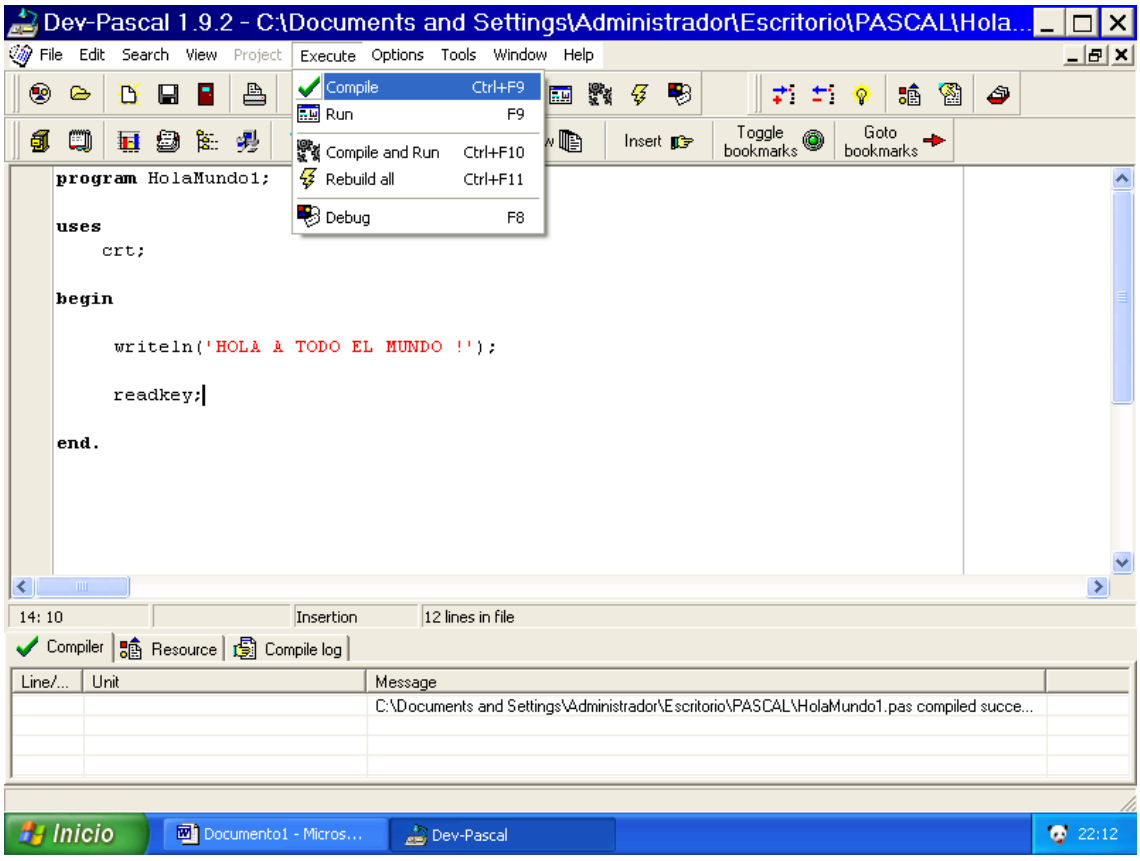
El programa se llamará “*HolaMundo1*”, y el nombre del fichero fuente, será “*HolaMundo1.pas*”.



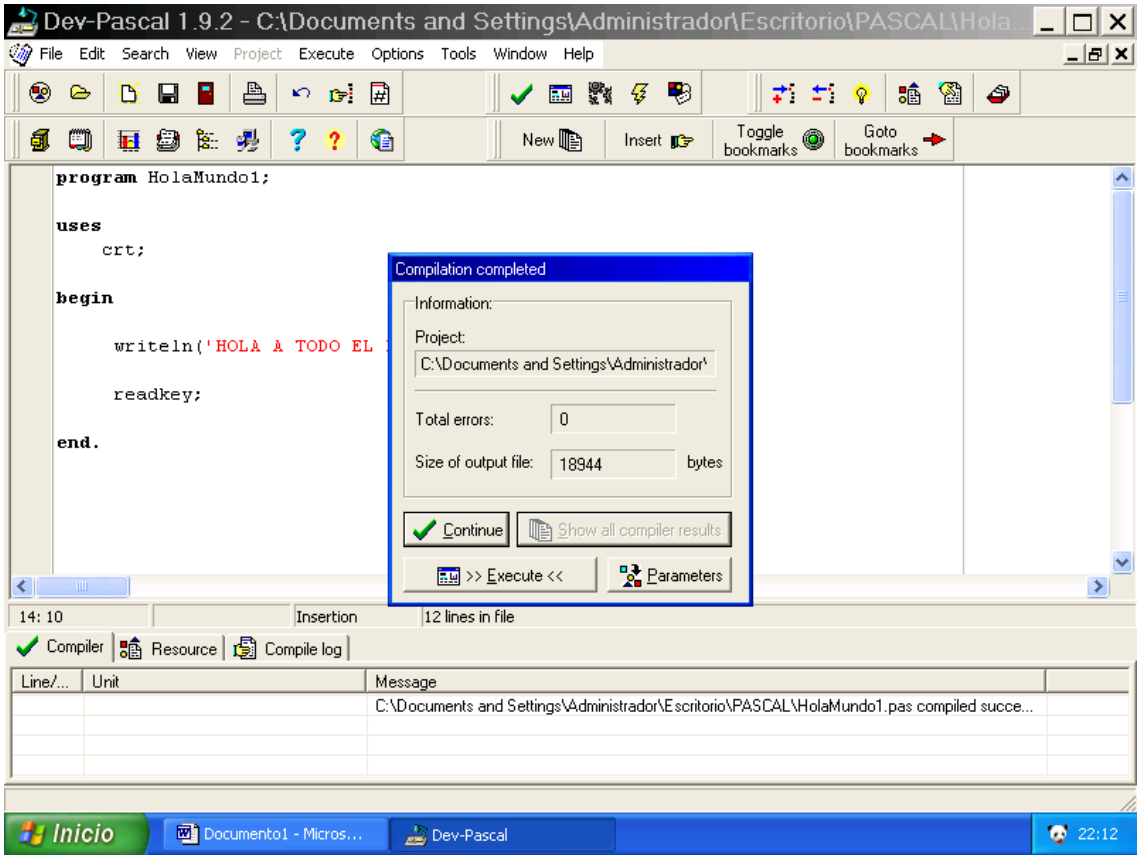
Hacemos las modificaciones que se muestran, para ver funcionar nuestro primer programa en Pascal hecho con el entorno Dev-Pascal:



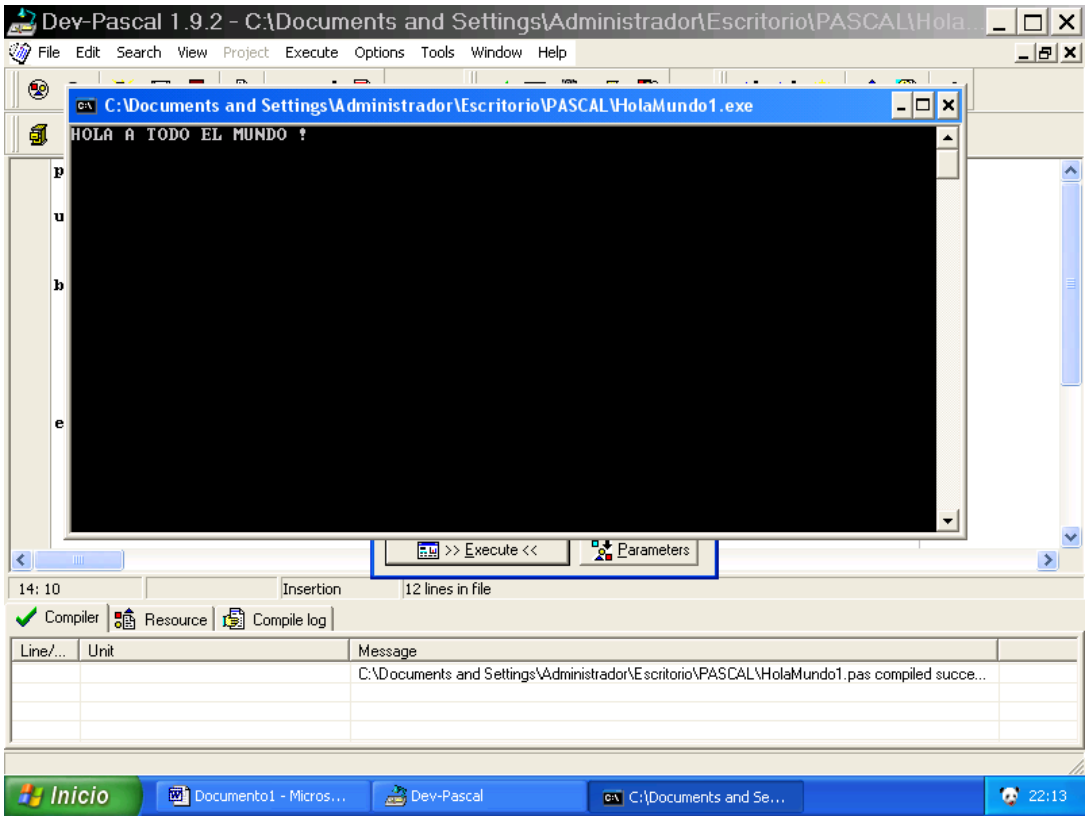
Y, cuando queremos probar el programa, primero se ha de compilar, con la opción **Execute→Compile**:



Si la compilación ha sido correcta, es decir, si el programa está escrito correctamente, nos permitirá ejecutar el programa con el botón **>> Execute <<**:

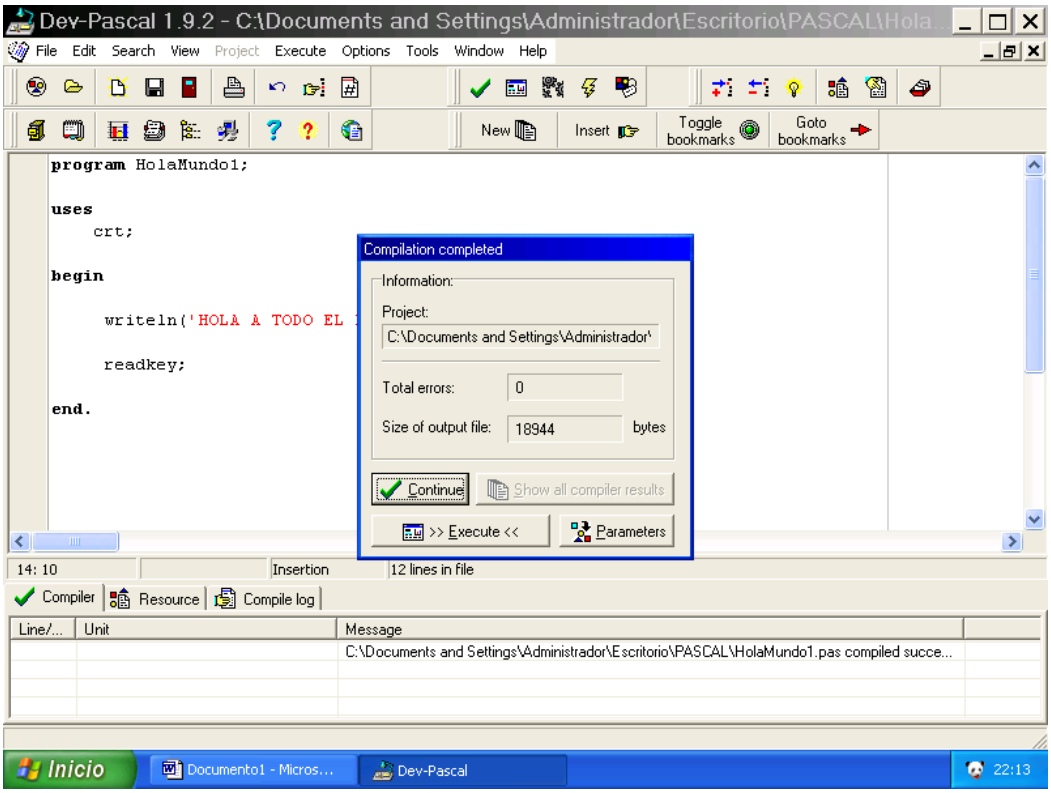


Y, pulsando **>> Execute <<**, podremos ver lo que hace nuestro primer programa... Se nos abre una ventana (llama ventana D.O.S.) con el texto “HOLA A TODO EL MUNDO !”, y esperando a que pulsemos una tecla (cualquiera) para finalizar el programa:

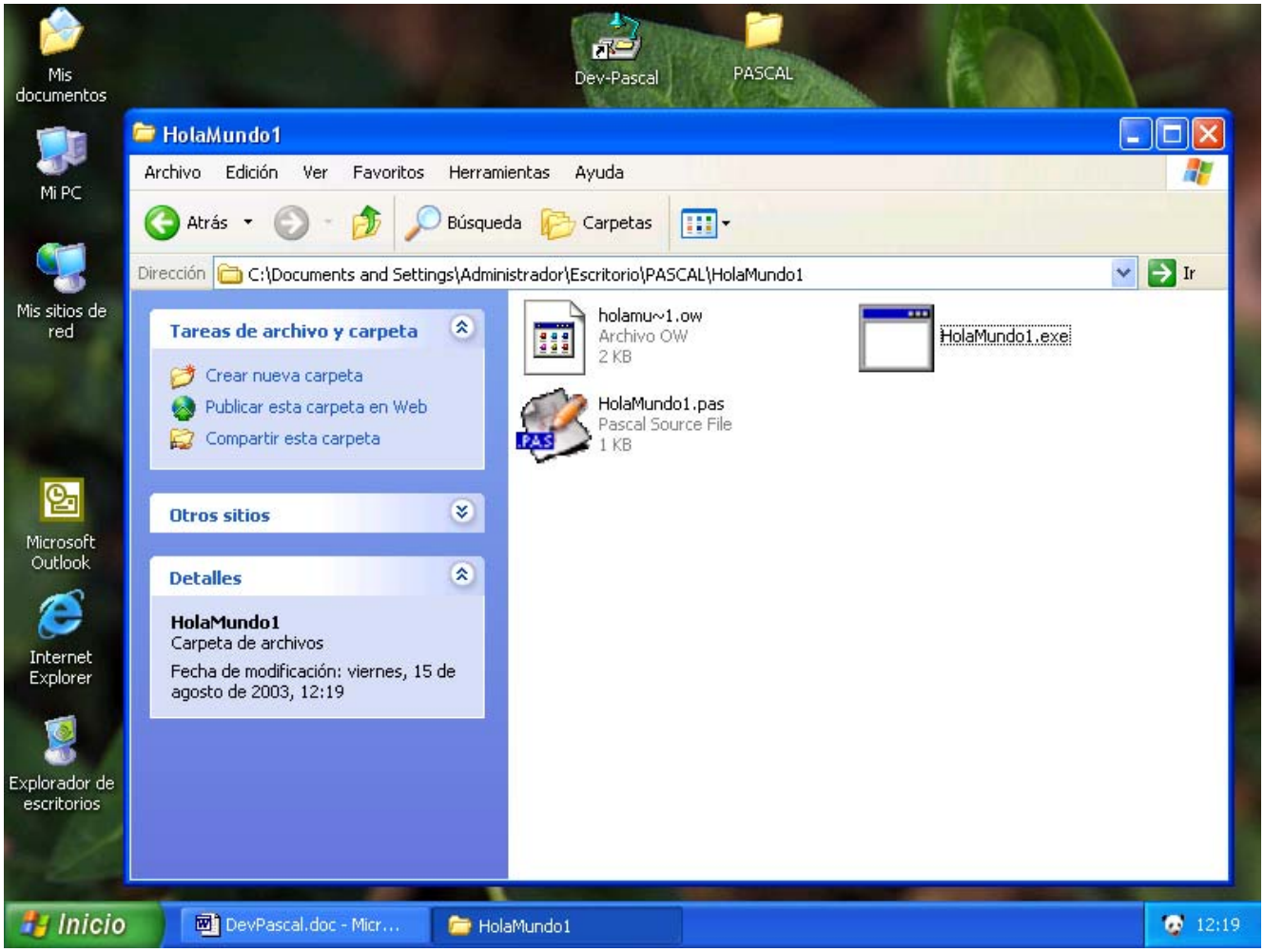


Y, tal y como te he dicho, cuando pulsemos una tecla, nuestro programa finalizará, y volveremos al entorno Dev-Pascal.

Para continuar con el desarrollo del programa, o cualquier otra cosa, pulsamos el botón de **Continue**:



Examinemos los archivos generados en la carpeta de nuestro programa:



Observamos el fichero fuente “HolaMundo1.pas”, un fichero utilizado en la compilación, llamado en este caso “Holamu~1.ow”, y el fichero “HolaMundo1.exe”.

El fichero “HolaMundo1.exe” es el fichero resultante de la compilación del programa. Es un fichero ejecutable (*EXEcutable*), y en principio, listo para distribuir a tus amigos y conocidos, sin necesidad de ningún otro fichero. Siempre y cuando, utilicen un sistema Windows.

Y con esta pequeña introdución a uno de los entornos de programación en Pascal, el Dev-Pascal, ya estás listo para adentrarte y empezar a elaborar nuestros primeros programas más complejos.

Está claro que el entorno Dev-Pascal es potente, y dispone de muchas más opciones, pero tan sólo te he explicado lo básico. El resto irá apareciendo durante los siguientes capítulos.

Nuestro primer programa en Pascal

¡¡ Página **MUY** importante !!

Ya has visto qué fácil es crear programas en Pascal, compilarlos y ejecutarlos.

Ahora te empezaré a explicar mediante ejemplos, el concepto de programar.

Observa el programa que has escrito en el primer capítulo, pero esta vez, te comento qué es cada "cosa":

```
program HolaMundo1;
```

Nombre del programa
Nos da una idea de la utilidad destinada al programa.

```
uses
     crt;
```

Librerias utilizadas
Hay funciones, como "`writeln`", que NO necesitan de librerías externas para compilarse.
Pero, muchas funciones de control del teclado, de la pantalla, funciones matemáticas, SÍ necesitan librerías.
En este caso, es la función "`readkey`" la que necesita la librería "`crt`".
Así, que si en el programa tienes un "`readkey`", y no incluyes la librería "`crt`", el programa dará un ERROR de compilación.

```
Begin
```

`Begin`
Es una palabra clave, y marca el inicio de un bloque de instrucciones. El final del bloque se indica con un "`End`".
El primer "`Begin`", que en este caso es éste, indica el inicio del bloque principal; es decir, el inicio del programa.

```
     writeln('HOLA A TODO EL MUNDO !');
```

Función "`writeln`"
Esta función, escribe por pantalla el texto indicado entre paréntesis. De las comillas simples hablaremos más adelante, pero en este caso hay que ponerlas.

```
     readkey;
```

Función "`readkey`":
Espera la pulsación de una tecla. Te explicaré que se pueden hacer más cosas.
Aquí lo ponemos para ver el resultado del programa antes de que se cierre.

```
End.
```

`End`:
Marca el final de un bloque de instrucciones. En este caso, marca el final del bloque principal, es decir, marca el final del programa, ya que le sigue un punto.
Verás que el `End` que cierra un bloque interno, acaba en punto y coma.
Es importante que sepas, que cada `Begin`, debe llevar un `End`.

¿Qué es programar?

Bien... un programa es algo parecido a un problema matemático.

Se hacen una serie de operaciones... se aplican una serie de funciones... y se obtiene un resultado.

Pero... ¿no falta algo...?

En efecto... falta una ENTRADA.

Re-defino lo que he dicho...

Un **programa**:

- obtiene unos **datos de entrada**.
- aplica sobre ellos unos cálculos y unas funciones (**procesa**).
- obtiene un resultado: unos **datos de salida**.

En el caso del programa “HolaMundo1”... NO hay datos de entrada iniciales... Aunque... sí que hay un dato de entrada que provoca el fin del programa: la pulsación de una tecla.

Debes memorizar la siguiente estructura:

```
Program nombreDelPrograma;

Uses
     Crt, etc...;

Var
     ...

Begin

-
-
-

End.
```

Es la estructura más simple de todo programa en Pascal.

Sé que ya empiezo a repetir cosas, pero es Fundamental, que empieces sólidamente a aprender a programar en Pascal.

El ”`Begin`” y el ”`End`” forman el bloque principal de todo programa. Más adelante, veremos, que podrá haber más `Begin`’s y `End`’s por el medio. Pero un “`Begin`”, y un “`End`”, como mínimo, siempre estarán presentes en todo programa.

Datos: Variables y constantes

No es exactamente el mismo concepto, pero en matemáticas, en física, etc... ya sabes que, normalmente, se definen letras, que son las variables: `x`, `y`, `z`, `v`...

Bien... en programación, pueden ser letras o palabras.

Por ejemplo, si queremos tener una variable que almacene el valor de una tecla pulsada, pues podemos llamarle "`tecla`", por ejemplo.

Si queremos almacenar un número, podemos llamarle "`num`".

Es MUY recomendable, que los nombres de las variables sean claros, y que tengan sentido y guarden relación con el dato que van a contener. Recomiendo NO dar nombres como: `x`, `z`, `p`, `q`, `r`, `t`, `b`, `a`, `x1`...

A medida que vayas leyendo el manual, lo irás entendiendo, y también verás, que a veces SÍ podemos llamarles `x`, `z`, `p`... pero SÓLO en casos MUY concretos, concisos y claros.

Bien... El símil de una variable es una "casilla" en la que guardamos información de un determinado tipo, y podemos ir cambiando su contenido durante el programa.

Hay varios tipos de información... claramente, podemos distinguir, por ejemplo: letras, números, cadenas de texto...

Una variable, en programación, es un registro que tiene un determinado tipo de dato; es decir, que hay distintos tipos de variables.

La caracterísitca importante de una variable, es que puede cambiar su valor durante la ejecución del programa.

Y esto es lo que diferencia una variable de una constante.

Verás que para declarar variables, lo harás dentro de la cláusula `Var`.

Las constantes NO pueden modificar su valor durante el programa. Se les asigna un valor inicial, y NO se pueden modificar.

Verás que para declarar constantes, lo harás dentro de la cláusula `Const`.

*En el **apéndice A**, tienes una tabla que describe todos los tipos de variables existentes. Es muy útil y de muchos tipos deberás memorizar su rango.*

Ejercicios didácticos

Es esencial que, uno a uno, piques todos los programas de los ejercicios, vayas probando su funcionamiento, y leyendo las explicaciones, para entender el “Por Qué” de cada cosa.

Poco a poco, irás asimilando una metodología de programación.

Ejercicio 1: Variables_1 - Programación

¡¡ Página **MUY** importante !!

Veamos un primer ejemplo de programa usando una variable:

```
Program Variables1;
```

Nombre del programa
Nos da una idea de la utilidad destinada al programa.

```
Uses
   crt;
```

Librerias utilizadas
Hay funciones, como "`writeln`" o "`readln`", que NO necesitan de librerías externas para compilarse bien.
Pero, muchas funciones de control del teclado, de la pantalla, funciones matemáticas, SÍ necesitan librerías.
En este caso, es la función "`readkey`" la que necesita la librería "`crt`". Así, que si en el programa tienes un "`readkey`", y no incluyes la librería "`crt`", el programa dará un ERROR de compilación.

```
Var
   num: integer;
```

Variables utilizadas en todo el programa.
En este caso, tan sólo se crea la variable "`num`", que es de tipo `integer` (números enteros).

```
Begin
```

`Begin`
Es una palabra clave, y marca el inicio de un bloque de instrucciones. El final del bloque se indica con un "`End`". El primer "`Begin`", en este caso es éste, indica el inicio del bloque principal; es decir, el inicio del programa.

```
   write('Introduce un numero: ');
```

Función "`write`"
Esta función, escribe por pantalla el texto indicado entre paréntesis.

```
   readln(num);
```

Función "`readln`"
Esta función, DETIENE el programa, y espera a que introduzcas por teclado el valor de una variable. En este caso, le has de introducir un número entero.
Si introduces cualquier otra cosa que no sea un número entero, el programa generará un error inesperado, y se terminará inmediatamente!!

```
   writeln('Has introducido el numero ', num);
```

Función "`write`"
Esta función, también permite mostrar el contenido de una variable por pantalla, y permite hacer convinaciones entre texto y datos, como es el caso. Si hemos introducido el número 56, en pantalla observaremos:

`Has introducido el numero 56`

```
   readkey;
```

Función "readkey":
Espera la pulsación de una tecla.

```
End.
```

End:
Marca el final de un bloque de instrucciones. En este caso, marca el final del bloque principal, es decir, marca el final del programa.

Ejercicio 1: Variables_1 - Explicación

Debes estar atento a esta primera explicación, ya que en las siguientes, sólo te explicaré las novedades.

La palabra clave "Program"

Establece un nombre de programa, que nos da "una idea" de la función del programa.Debe de ser un nombre corto, y que no contenga caracteres extraños (ilegales para Dev-Pascal). Caracteres ilegales en el nombre del programa son: las vocales acentuadas, el guión simple "-", el espacio, los interrogantes, las exclamaciones, etc... etc... Todos estos caracteres NO debes emplearlos.

Para el nombre del programa, utiliza letras sin acentuar (no se aceptan 'ñ', 'ç', etc...), y también puedes utilizar el guión bajo '_'.
ATENCIÓN: NO se puede separar con espacios el nombre del programa.

El nombre del programa asignado con "Program", NO tiene por qué ser el mismo que el nombre que se le da al fichero PAS.
Por ejemplo, podemos tener un Program HolaMundo; y que el fichero se llame "Programa1.pas", aunque SÍ es aconsejable que sean iguales, para evitar confusiones.

La palabra clave "Uses"

Se utiliza para invocar a las librerías externas necesarias para la correcta compilación del programa.
Depende de qué funciones utilices en el programa, tendrás que poner una o varias librerías, si no, el compilador te dará un error, porque NO conocerá esa función o funciones que utilices.
Repito, que hay funciones que pueden utilizarse sin la necesidad de ninguna librería, como són "write", "writeln", "read", "readln", etc...

*En el **apéndice B**, tienes una tabla que te relaciona las librerías existentes, con las funciones incorporadas para cada librería, y una descripción de la utilidad de cada función.*

La palabra clave "Var"

Aquí se definen las variables a utilizar durante el programa.

*En el **apéndice A**, tienes una tabla que describe todos los tipos de variables existentes. Es muy útil, y de muchos tipos deberás memorizar su rango.*

Las palabras clave "Begin" y "End"

Marcan el inicio y el final de un bloque de instrucciones.

Recuerda, que SIEMPRE hay como mínimo, un Begin y un End, que forman el bloque principal que define el programa en sí.

Más adelante veremos cómo se marcan sub-bloques dentro del programa principal.

Las funciones "`write`" y "`writeln`"

Ambas funciones escriben texto y datos por pantalla.

Diferencia entre "`write`*" y "*`writeln`*":*
La primera escribe el mensaje, y deja el cursor al final del mensaje.
La segunda escribe el mensaje, y deja el cursor en la línea inferior. Es decir: escribe el mensaje, y luego da un salto de línea.

Observa que se puede escribir sólo texto:

```
write('Introduce un numero: ');
```

Y, cuando la variable "num" ya tiene un valor, podemos escribir texto y/o datos:

```
writeln('Has introducido el numero ', num);
```

También podríamos haber hecho, directamente:

```
writeln(num);
```

Un truco... para hacer sólo un salto de línea, escribe en tu programa:

```
writeln;
```

Las funciones "`read`" y "`readln`"

Ambas funciones detienen el programa, y esperan a que el usuario introduzca el valor de una variable.

Diferencia entre "`read`*" y "*`readln`*":*
`Realnd` *descarta el resto de la entrada tras introducir un ENTER.*
`Read` *conserva en el buffer todos los datos introducidos.*
Te recomiendo utilizar sólo el "`readln`*". Te dará menos problemas!*

La función "`readkey`"

Detiene el programa, y espera la pulsación de una tecla.
Es ideal, para que, antes de que finalice el programa, puedas observar los resultados de éste, ya que si no, se cierra la ventanita negra (Denominada pantalla D.O.S.).

Dudas que se pueden haber generado

¡¡ Página **MUY** importante !!

¿Qué es una línea? ¿qué es un salto de línea?

Bien. La pantalla, en modo texto, que es como trabajaremos durante este curso, consta de 80 columnas y 25 líneas (también se le pueden llamar filas). Y en cada "casilla" cabe **un** carácter.

*Bájate el programa "**ModoTexto.pas**" que se encuentra en:*

www.sergi2.com/pascal/

Podrás observar exactamente a qué me refiero.

*En el **apéndice C**, tienes un ejemplo de uso de la pantalla en modo texto.*
Recuerda: durante TODO el curso, utilizaremos el modo texto.

¡¡ Vaya rollo !! ¡Es muy aburrido! ¿esto es programar?

Que sepas, que también existe el modo gráfico, que se divide en varios tipos, y que permite la creación, por ejemplo, de... ¡Video-juegos!

*Bájate el programa "**Arkanoid**"programado en Pascal*****, que se encuentra en:*

www.sergi2.com/pascal/

Podrás admirar el fascinante mundo de la programación, y lo que puedes llegar a hacer!!!
Pero, antes... es obligatorio tener los conocimientos básicos que quiero transmitir con este libro, y que como mínimo te ayudará a aprobar la asignatura!!

*****ATENCIÓN:

Descomprime el fichero "arkanoid.zip" en una carpeta de tu ordenador, y prueba "arkanoid.exe" para jugar. Los ficheros .PAS son los fuentes y los demás archivos son gráficos y de información.
Si quieres modificar el juego, deberás utilizar Turbo Pascal 7.
Sólo ha sido probado en Windows'98.

Ejercicio 2: Variables_2 - Programación

¡¡ Página **MUY** importante !!

Veamos un programa que te ayudará a entender y a familiarizarte con el uso del "`write`", del "`writeln`" y del "`readln`":

```
Program Variables_2;

Uses
   crt;

Var
   nombre: string;
   edad: integer;
   estatura: real;

Begin

   write('Como te llamas? ');
   readln(nombre);

   write('Cuantos anyos tienes? ');
   readln(edad);

   write('Cuanto mides? ');
   readln(estatura);



   writeln;
   writeln;


   writeln('Te llamas ', nombre, ', tienes ', edad, ' anyos, y
     mides ', estatura:1:2, ' metros.');



   writeln;
   writeln;


   write('Pulsa una tecla para salir.');
   readkey;

End.
```

Ahora tenemos 3 variables.

***nombre** es una cadena de texto.*

***edad** es un número entero.*

***estatura** es un número real, para poder poner decimales.*

Pedimos el nombre por teclado.

Pedimos la edad por teclado.

Pedimos la altura por teclado.

Dejamos 2 líneas en blanco, para distanciarnos del texto entrado por el usuario.

Mostramos todos los datos en una única frase!!!

Dejamos 2 líneas en blanco...

Ejercicio 2: Variables_2 - Explicación

¡¡ Página **MUY** importante !!

Observa que ahora utilizamos 3 variables:

nombre → es de tipo “cadena de texto”. Nos permitirá almacenar el nombre del usuario.

edad → es de tipo “entero”. Nos permitirá almacenar la edad del usuario.

estatura → es de tipo “real”. Nos permitirá almacenar la estatura del usuario, permitiendo el uso de decimales.

MUY IMPORTANTE
Para introducir un número real (con parte decimal), por teclado, la “coma” se especifica con un punto (‘.’):

`1.79` ✔ `1,79` ✘

Utilizamos dos veces seguidas la función “`writeln`”, porque, recordemos que “`writeln`” da un salto de línea al final. Si llamas a la función sin dar ningún parámetro entre paréntesis, tan sólo se escribe el salto de línea en pantalla.
Así conseguimos distanciarnos 2 líneas del texto anterior.

Y observamos la combinación de texto y variables del “`writeln`” que muestra todos los datos en una única frase:

```
writeln('Te llamas ', nombre, ', tienes ', edad, ' anyos, y
     mides ', estatura:1:2, ' metros.');
```

También hubiera sido válido, haber puesto esto:

```
writeln(nombre, ', tiene ', edad, ' anyos, y
     mide ', estatura:1:2, ' metros.');
```

O esto:

```
writeln('-', nombre, '-', edad, '-', estatura:1:2, '.');
```

Observa en la última proposición. Según los datos que hubiera entrado el usuario, saldría:

- Laura - 28 - 1.79

Fíjate en la combinación de texto y variables.

El poner “estatura:1:2”, sirve para mostrar el valor del número real forzándolo a mostrar un dígito entero y 2 decimales.

Ejercicio 3: IF 1 - Programación

¡¡ Página **MUY** importante !!

Ahora vamos a implementar un programa que nos permitirá saber si nos han pulsado la tecla ‘1’ o cualquier otra tecla.

Para ello introduzco el concepto de “sentencia de control”.

Una sentencia de control, es una expresión que nos permite controlar la dirección del programa (a la dirección que va tomando un programa, también se le llama “flujo”).

Hasta ahora hemos hecho programas con una estructura secuencial: primero una instrucción, luego otra, luego otra, ... luego otra, la última instrucción... y final de programa.

Vamos a hacer un programa que, inicialmente comienza de forma secuencial, pero que, después de pedir una tecla, dependiendo de si es ‘1’ o no, mostrará un mensaje por pantalla u otro.

Esta sentencia de control es una estructura alternativa, porque tiene una alternativa: o sí, o no. O verdadero, o falso.

```
Program IF1;

Uses
     Crt;

Var
   tecla: char;

Begin

   writeln('Pulsa una tecla, y te dire si es el 1');


   tecla:=ReadKey;


   if(tecla='1') then
      writeln('SI! HAS PULSADO EL 1 !')
   else
      writeln('NO... Has pulsado una tecla que no era el 1...');

   writeln;
   writeln;
   writeln('Pulsa una tecla para salir...');
   ReadKey;

End.
```

Variable “tecla”, que es un carácter (con las teclas escribimos carácteres)

Readkey detiene el programa y espera a que pulsemos una tecla. Cuando la pulsamos, el valor de la tecla pulsada va a parar a la variable “tecla”

Bloque IF.
Si la tecla es ‘1’ haz una cosa...
... si no, haz la otra.

Ejercicio 3: IF 1 - Explicación

¡¡ Página **MUY** importante !!

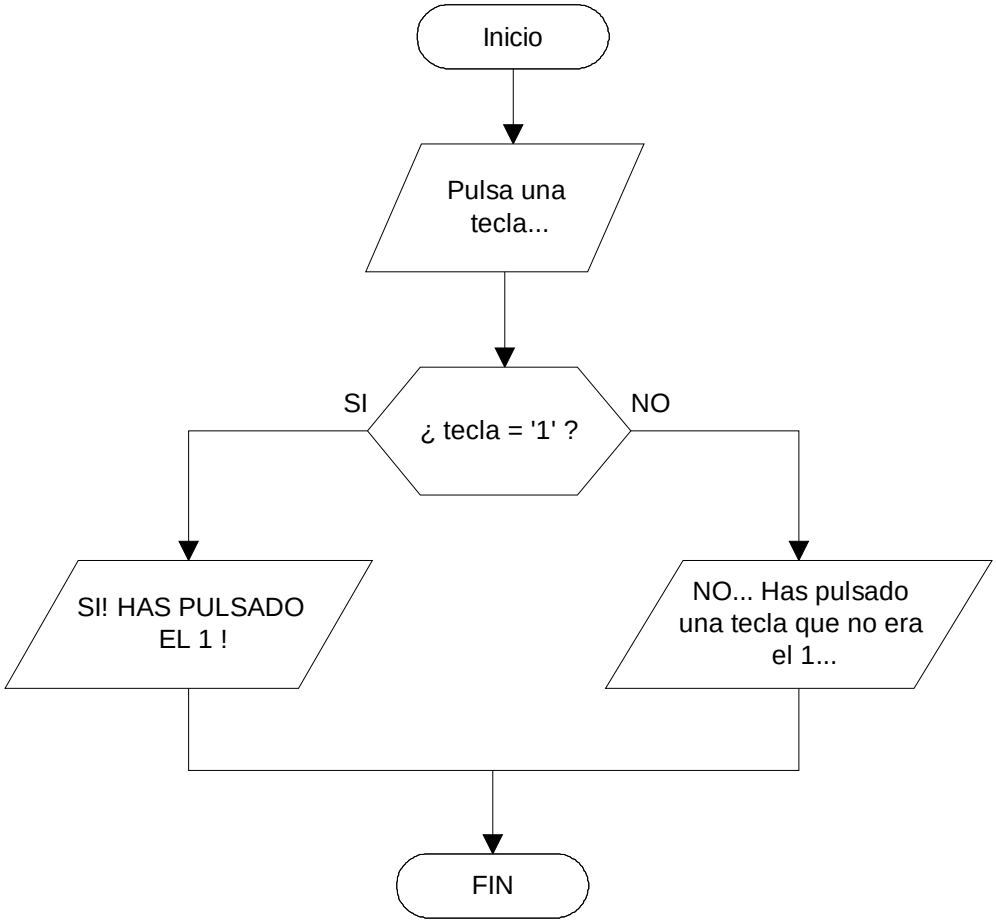
Y aquí toca, introducirte en la algorítmica...

Un algoritmo, es un esquema gráfico: organigrama, que representa el flujo de un programa.

El algoritmo nos permite ver de un vistazo las acciones del programa.

Al algoritmo, también se le conoce como "Diagrama de flujo".
*En el **apéndice D**, te explico la simbología utilizada, y ejemplos de algoritmos.*

Observa el algoritmo del programa IF1:



Observa claramente la bifurcación del programa, dependiendo de si pulsamos '1' o cualquier otra tecla...

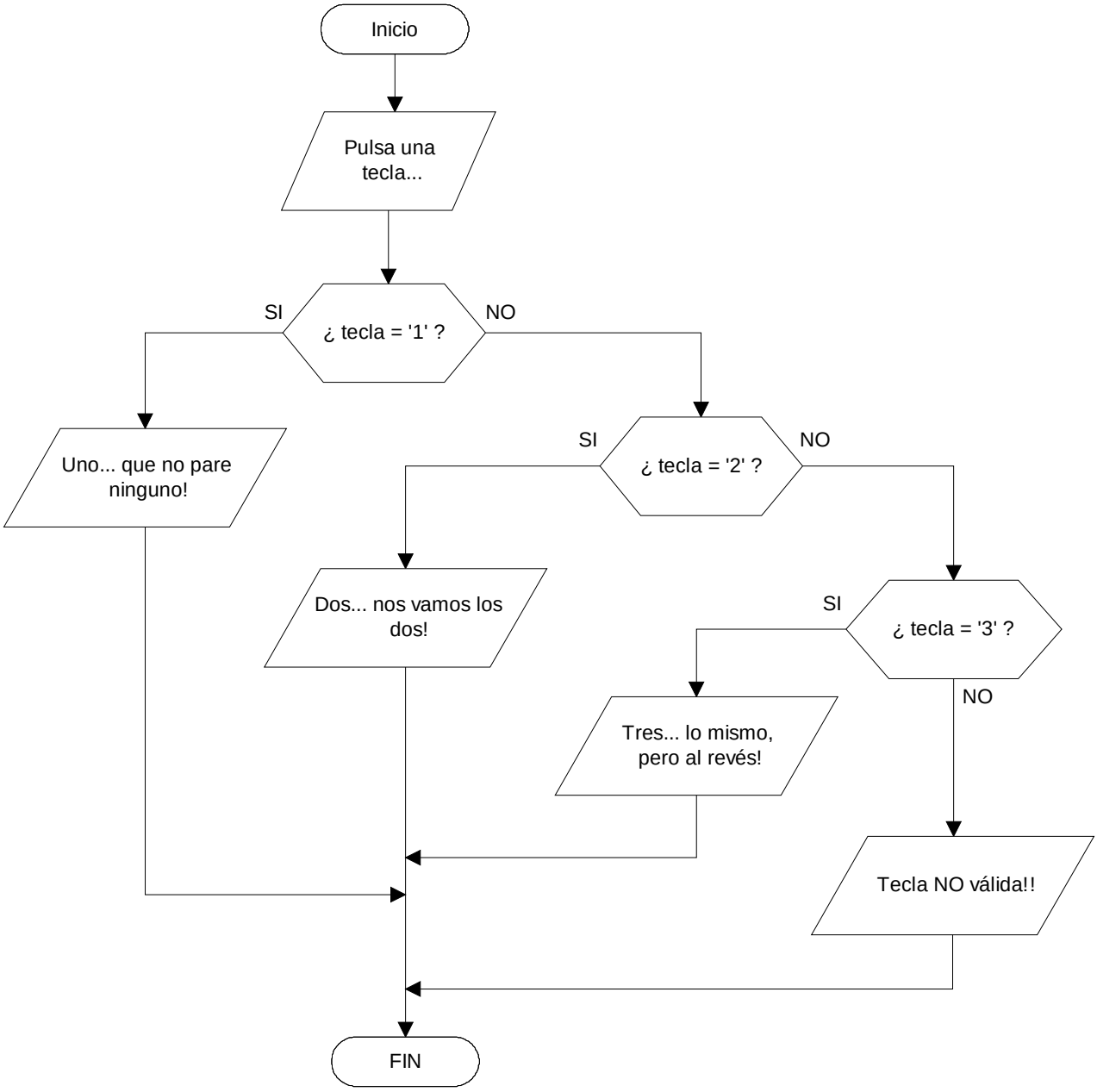
Otro tema a destacar, es el uso de "`ReadKey`".
Hasta ahora utilizábamos "`ReadKey`" simplemente para esperar una tecla, y así pausar el programa.

Pues ahora conoces otra aplicación... "`ReadKey`" es capaz de asignar el valor de la tecla pulsada a una variable de tipo **`char`**.

Ejercicio 4: IF 2 - Planteamiento y diseño

Perfeccionaremos más el anterior programa!!
Vamos a diseñar un programa que detecte si hemos pulsado '`1`', '`2`', ó '`3`'.

Te muestro el algoritmo:



Observa que tendremos que utilizar 3 estructuras alternativas del tipo `if`.

Ejercicio 4: IF 2 – Solución

Aquí tienes el listado del programa:

```
Program IF2;

Uses
     Crt;

Var
   tecla: char;

Begin

   writeln('Pulsa 1, 2 O 3...');

   tecla:=ReadKey;

   if(tecla='1') then
      writeln('Uno... que no pare ninguno!')
   else
      if(tecla='2') then
         writeln('Dos... nos vamos los dos!')
      else
         if(tecla='3') then
            writeln('Tres... lo mismo, pero al reves!')
         else
            writeln('Tecla NO valida!!');

   writeln('Pulsa una tecla para salir...');
     ReadKey;

End.
```

Observa bien la tabulación utilizada (identación).
Es importante a la hora de escribir y presentar un programa!!

Observa... hemos tenido que crear, lo que se conoce como “estructura anidada de IF’s”.

Son tres IF’s anidados:

***Si** la tecla es ‘1’ ya está.*
***Si no** es ‘1’...*
  ***Si** la tecla es ‘2’ ya está.*
  ***Si** no es ‘2’...*
    ***Si** la tecla es ‘3’ ya está.*
    ***Si no** es ‘3’... quiere decir, que no es ni ‘1’, ni ‘2’, ni ‘3’, por lo tanto... Tecla NO válida!!*

Recuerda!!!** **MUY IMPORTANTE
Es MUY importante TABULAR (identar), y hacerlo bien.
Así, vemos mejor la estructura de un programa, y lo que pertenece a cada una de las estructuras de control.

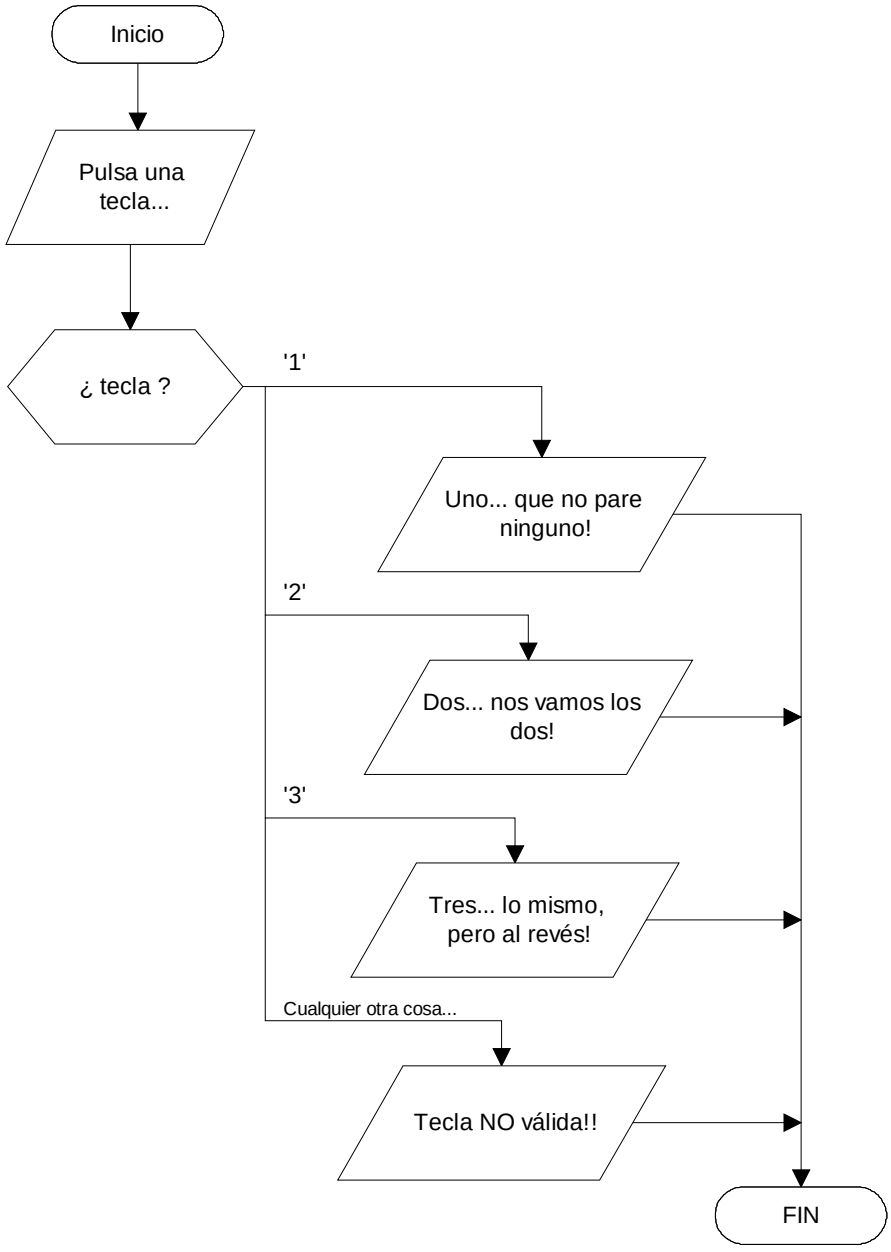
Ejercicio 5: Case Of 1 - Planteamiento y diseño

¡¡ Página **MUY** importante !!

Existe otra estructura de control alternativa, llamada "`case of`". Simplifica mucho el trabajo hecho en el anterior programa (el IF2).

Vamos a diseñar de nuevo un programa que detecte si hemos pulsado '1', '2', ó '3'.

Te muestro el "nuevo" algoritmo:



Ejercicio 5: Case Of 1 - Solución

¡¡ Página **MUY** importante !!

```
Program CaseOf1;

Uses
     Crt;

Var
   tecla: char;

Begin

   writeln('Pulsa 1, 2 O 3...');

   tecla:=ReadKey;
```

Estructura `Case Of`*.*
Evalúa una variable, y según sea su valor, tiene varios caminos para desviarse

```
   case tecla of
      '1': writeln('Uno... que no pare ninguno!');
      '2': writeln('Dos... nos vamos los dos!');
      '3': writeln('Tres... lo mismo, pero al reves!');
      else writeln('Tecla NO valida!!');
   end;



   writeln('Pulsa una tecla para salir...');
     ReadKey;

End.
```

Recuerda!!! ***MUY IMPORTANTE***
Es MUY importante TABULAR (identar), y hacerlo bien.
Así, vemos mejor la estructura de un programa, y lo que pertenece a cada una de las estructuras de control.

Ejercicio 6: Case Of 2 - Planteamiento y diseño

En este ejercicio desarrollaremos un programa que, preguntándo la edad al usuario, le diga si es:

- niño
- adolescente
- jóven
- adulto
- maduro
- anciano

Para ello, aplicaremos el siguiente criterio:

Niño: de 0 a 13 años.
Adolescente: de 14 a 18 años.
Jóven: de 19 a 29 años.
Adulto: de 30 a 49 años.
Maduro: de 50 a 69 años.
Anciano: de 70 a 100 años.

Hay que tener en cuenta, que es un criterio muy subjetivo, y que cada uno puede variar los rangos a su gusto.

Para ello, necesitaremos preguntar al usuario su edad, y, evaluándolo con el Case Of, determinar en qué parte de su vida se encuentra:

```
Program CaseOf2;

Uses
     Crt;

Var
   edad: integer;

Begin
   write('Cuantos años tienes? ');
   readln(edad);

   case edad of

     0..13: writeln('Eres un niño.');
      14..18: writeln('Eres un adolescente.');
      19..29: writeln('Eres una persona joven.');
      30..49: writeln('Eres una persona adulta.');
      50..64: writeln('Eres una persona madura.');
      65..100: writeln('Eres una persona anciana.');
      else writeln('Edad fuera de rango!');

   end;

   writeln('Pulsa una tecla para salir...');
     ReadKey;
End.
```

¡¡¡ Se pone entre 2 puntos !!!

Ejercicio 7: For 1 – Introducción a bucles

Avancemos un poco...

Es el momento de empezar a estudiar las estructuras repetitivas.

En efecto: un bucle es una estructura repetitiva, es decir, que repite una o varias instrucciones.

Se distinguen dos tipos:

Los bucles finitos, que repiten un número finito de veces la instrucción o instrucciones.
Los bucles infinitos, donde una vez se entra, se repite indefinidamente la instrucción o instrucciones.

En pascal podemos crear bucles con 3 tipos de estructuras:

- For – Do
- Repeat – Until
- While – Do

Veamos un primer ejemplo.
Vamos a diseñar un programa que imprima por pantalla los números del 1 al 10 (ambos inclusive).
Es decir, que muestre lo siguiente por pantalla:

```
1
2
3
4
5
6
7
8
10
```

Y... podrías pensar en hacer lo siguiente:

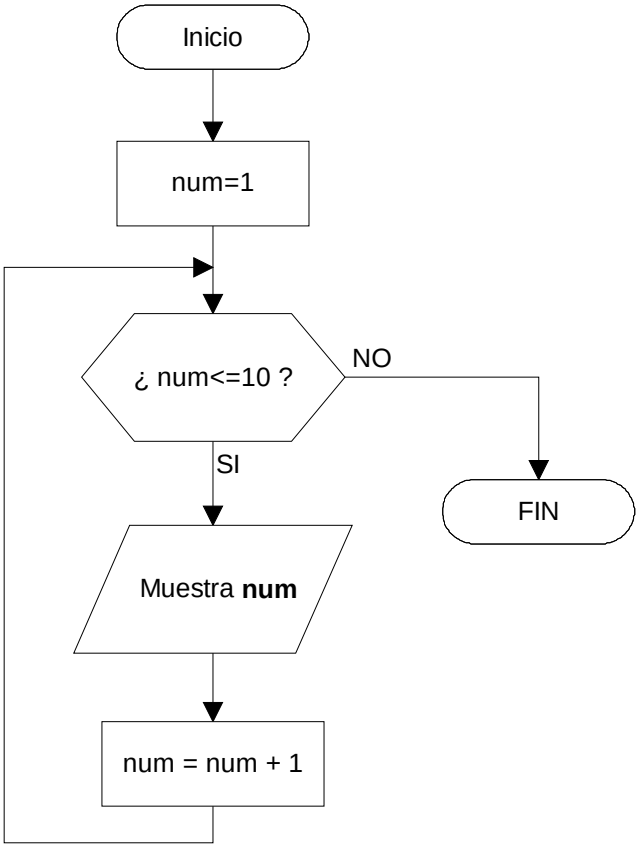
```
writeln('1');
writeln('2');
writeln('3');
writeln('4');
writeln('5');
writeln('6');
writeln('7');
writeln('8');
writeln('9');
writeln('10');
```

Y ahora te digo... ¿si tuviéras que hacerlo desde el 1 al 100000?

Pues... cuando tenemos que repetir una o varias cosas, varias veces, utilizaremos las estructuras repetitivas (bucles).

¡¡ Página **MUY** importante !!

Veamos el algoritmo del programa:



Y aquí tienes el código. Pícalo y pruébalo con Dev-Pascal:

```
Program For1;

Uses
   Crt;

Var
   num: integer;

Begin


   for num:=1 to 10 do
     writeln(num);

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

Estructura For:
Repite la instrucción, o bloque de instrucciones que le indiquemos con begin-end, en este caso, desde num=1 hasta num=10, es decir, lo repite 10 veces.
Si una estructura de control sólo tiene que ejecutar una instrucción, NO es necesario poner el begin-end *porque NO es un bloque!!... NO PONGAS* begin-end *en estos casos!*

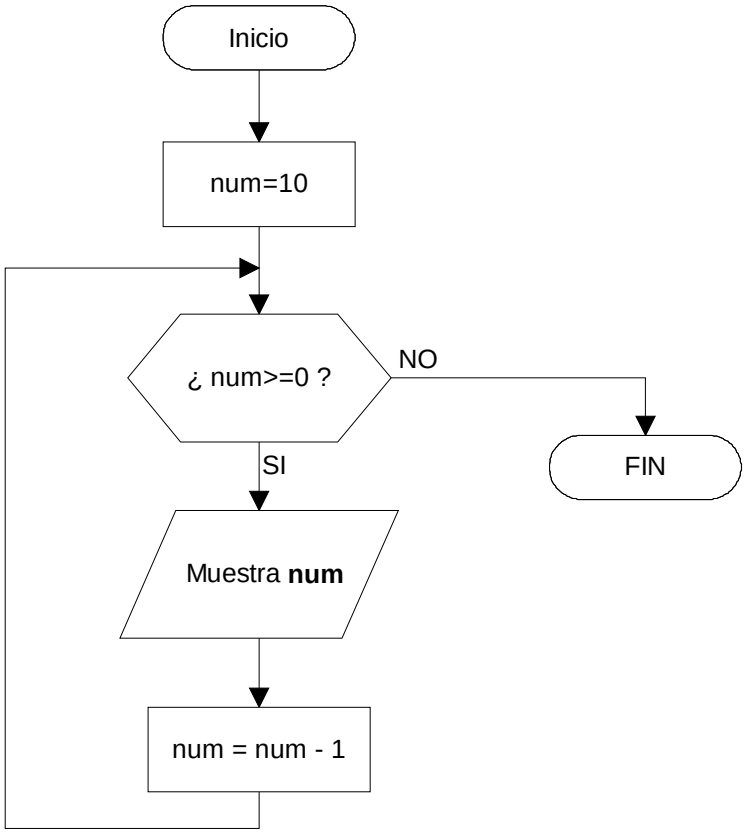
Ejercicio 8: For 1b – For en cuenta-atrás

¡¡ Página **MUY** importante !!

Vas a hacer prácticamente lo mismo que en el ejercicio anterior, pero esta vez, será una cuenta atrás, desde 10 hasta 0.

Verás que símplemente, supone cambiar una palabra del For.

Aquí tienes el algoritmo:



Y aquí tienes el código. Pícalo y pruébalo con Dev-Pascal:

```
Program For1;

Uses
   Crt;

Var
   num: integer;

Begin
   for num:=10 downto 0 do
      begin
         writeln(num);
      end;

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

Estructura For:
Repite el bloque que le indicamos con begin-end, en este caso, desde num=1 hasta num=10, es decir, lo repite 10 veces.

Downto en el For, implica una cuenta atrás.

Ejercicio 9: For 2 – Construyendo una tabla de multiplicar

Enfoquemos el `for` como algo más útil que para hacer cuentas adelante y atrás.

IMPORTANTE!! El `for` es una maravillosa herramienta para hacer contadores. Es muy intuitivo!!

Vamos a utilizar la variable num para algo más que simplemente mostrarla.

Multiplicaremos la variable num por 5, y así, a cada paso del `for`, veremos que se irá mostrando 0,5, 10, 15, 20, 25, ...

Se puede hacer de varias maneras.

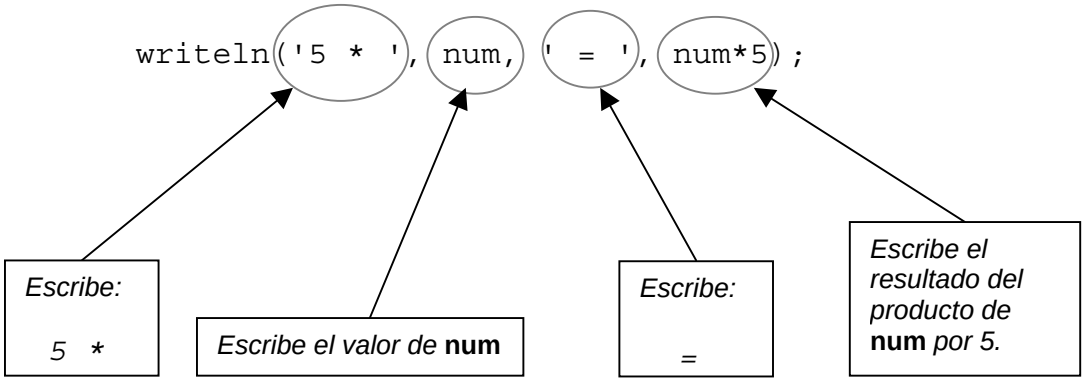
En la primera que te muestro, utilizo la potencia del `writeln`:

```
Program For2;

Uses
   Crt;

Var
   num: integer;

Begin

   writeln('Tabla de multiplicar del 5:');

   for num:=0 to 10 do
         writeln('5 x ', num, ' = ', num*5);

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

¡ESTE EJEMPLO ES MUY BUENO!
Prueba de picarlo y entenderlo!

OJO
Observa que no se pone el `begin-end` si el `for` sólo controla a una instrucción!!

Fíjate que es un `for` **idéntico** al del Ejercicio 6, pero que en lugar de hacer "`writeln(num)`", hacemos lo siguiente:



Detalle importante

Si sólo queremos ejecutar una instrucción dentro del `For`, **NO** hace falta que creemos un bloque `begin-end`.
Si queremos ejecutar varias instrucciones, **SI** debemos de recogerlas dentro de un bloque de código.

Por ejemplo, estas expresiones son equivalentes:

```
for num:=0 to 10 do
      begin
         writeln(num);
      end;
```

1

=

```
for num:=0 to 10 do
      writeln(num);
```

2

Pero estas NO son equivalentes:

```
for num:=0 to 10 do
      begin
         writeln(num);
         writeln(num+2);
end;
```

3

≠

```
for num:=0 to 10 do
      writeln(num);
      writeln(num+2);
```

4

¿Por qué?

Si debajo del `for` existe un bloque `begin-end` con varias instrucciones dentro, el bucle controlará dicho bloque.

Si debajo del `for` no hay un bloque `begin-end`, el bucle sólo controlará la instrucción inmediata a dicho `for`.

IMPORTANTE CONCLUSIÓN

Controla los bloques `begin-end`, y ponlos sólo si son necesarios, sea en un `for`, en un `if`, etc... etc...

Ejercicio 10: For 3 – Constantes... ¿para qué sirven?

Vamos a hacer un programa con exactamente la misma función que el de la página anterior, pero utilizando una constante que determinará hasta qué número se mostrará: en este caso hasta el 10.

```
Program For2;

Uses
   Crt;

Const
   MAX=10;

Var
   num: integer;

Begin

   writeln('Tabla de multiplicar del 5:');

   for num:=0 to MAX do
         writeln('5 * ', num, ' = ', num*5);

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

OJO
Observa que no se pone el `begin-end` si el `for` sólo controla a una instrucción!!

Fíjate hemos añadido lo siguiente:

```
Const
   MAX=10;
```

Fíjate que está entre el `Uses` y el `Var`, es importante que la declaración de constantes esté siempre justo debajo del `Uses`, así, en cualquier punto del resto del programa donde usemos dichas constantes, el programa las entenderá.

Las constantes sirven para estructurar mejor el programa e irás viendo, que conforme el programa se hace más grande, son útiles para los bucles y demás procesos donde se tenga que llegar hasta el determinado valor indicado por la constante, sea cual sea el número de bucles.

Entonces, cuando queramos ampliar ese límite, simplemente cambiaremos el valor de la constante, y el programa funcionará exactamente igual, pero teniendo en cuenta la ampliación en todo el programa (en todos aquellos lugares donde utilicemos la constante).

Ejercicio 11: Repeat Until – “Repite esto” hasta...

¡¡ Página **MUY** importante !!

Sabiendo un poco de inglés, se vé venir lo que voy a explicarte...

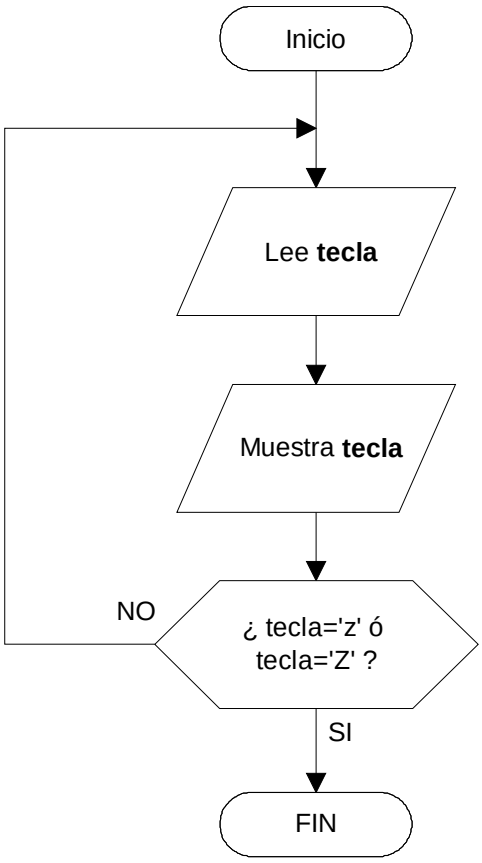
`Repeat Until`, equivale a decir “Repite hasta”.

La estructura “Repeat Until”, repite una instrucción o un bloque de instrucciones hasta que se cumpla algo...

Un ejemplo sería decir... “ Repite la lectura de una tecla hasta que pulsen la ‘**z**’ ”.

Vamos a construir un programita, que vaya pidiendo teclas, y las vaya mostrando por pantalla, hasta que pulsemos la ‘**z**’, sea minúscula ó mayúscula.

Veamos cómo sería el algoritmo:



¡ESTE EJEMPLO ES MUY BUENO!
Prueba de picarlo y entenderlo!

Ejercicio 11: Solución

¡¡ Página **MUY** importante !!

A continuación muestro el programa correspondiente:

```
Program repeatUntil1;

Uses
   Crt;

Var
   tecla: char;

Begin


   Repeat
      tecla:=ReadKey;
      writeln(tecla);
   Until (tecla='z') or (tecla='Z');


   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

Inicio del `Repeat`*. Repite...*

...Hasta que tecla valga 'z' ó 'Z'.

MUY importante el juego de paréntesis, si no te dará un error!!! MEMORÍZALO!!!

Ejercicio 12: While Do – “Mientras...” haz...

¡¡ Página **MUY** importante !!

Similar al `Repeat-Until`, pero empezamos comprobando la condición de permanencia.

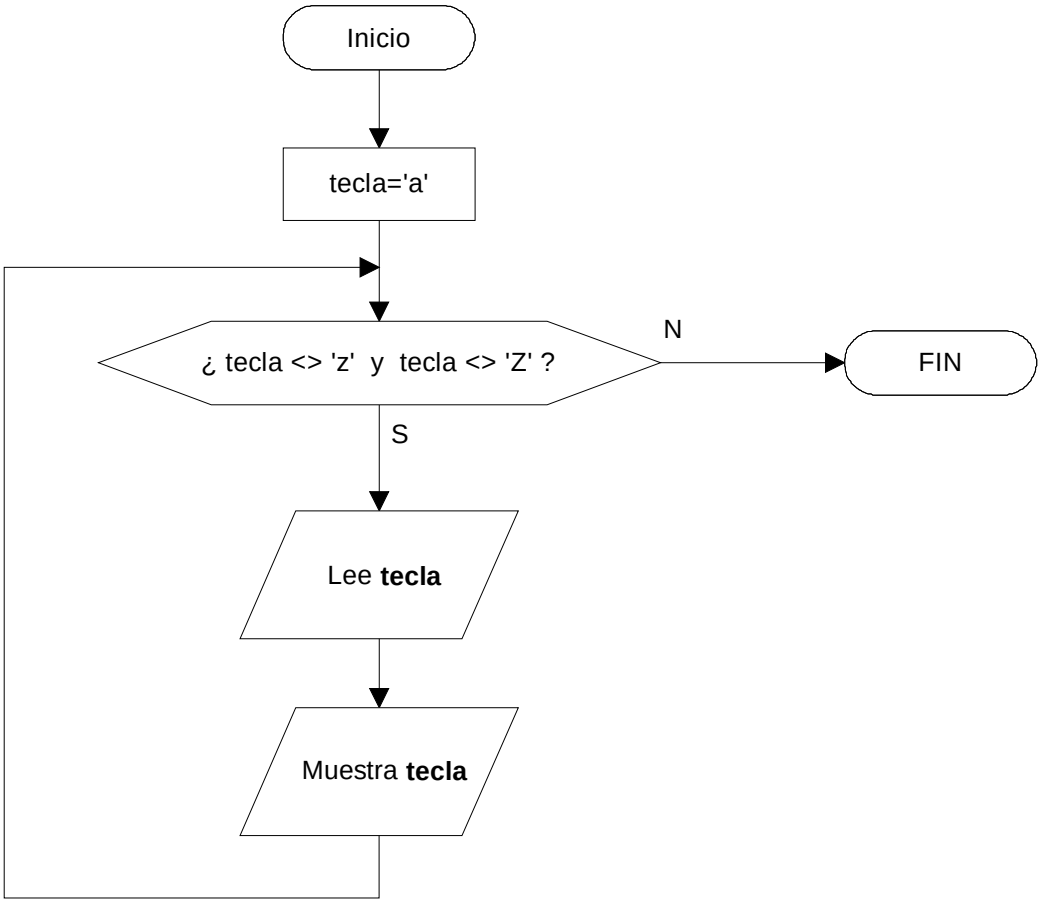
La estructura “While-Do”, mientras se cumpla algo, repite una instrucción o un bloque de instrucciones.

Es decir... sería algo así como “Mientras no pulses ‘z’ ves haciendo esto...”

Diseñemos un programa, similar al del ejercicio anterior, que vaya pidiendo teclas mientras **no** pulsemos la ‘z’ minúscula ***y*** **no** pulsemos la ‘Z’ mayúscula.

Entenderás, que inicialmente, tenemos que dar un valor a la variable tecla, que sea diferente de ‘z’ y de ‘Z’, porque si no, no entraría nunca en el bucle.

Aquí tienes el algoritmo:



El `while` empieza “preguntando” si la condición se cumple. Si no se cumple, ya NO se entra en el bucle.
Por eso inicializamos el valor de `tecla` a ‘a’, para asegurar que sea diferente de ‘z’ ó ‘Z’ y así asegurar que entra como mínimo una vez en el bucle (la primera vez).

Ejercicio 12: While Do – Programa

Aquí tienes el código.

Observa la asignación inicial `tecla:='a'` para permitir que entre en el bucle.

Observa el "juego" condicional...
El bucle se repite mientras tecla sea diferente de 'z' y tecla sea diferente de 'Z'.

Pica el código, pruébalo... estúdialo... haz pruebas... cambia cosas... ***Esta es la manera de aprender!! Ir probando cosas y entendiendo el por qué de cada cosa!!***

```
Program whileDo1;

Uses
   Crt;

Var
   tecla: char;

Begin

   writeln('Ves pulsando teclas... Z para salir...');

   tecla:='a';

   While (tecla<>'z') and (tecla<>'Z') do
      Begin
         tecla:=ReadKey;
         writeln(tecla);
      End;

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

Recuerda!!!
Es MUY importante TABULAR (identar), y hacerlo bien!!

FASE 2: Programando en serio

Si has seguido mis consejos, y has tecleado todos y cada uno de los ejercicios anteriores, habrás asimilado los conceptos básicos de la programación estructurada.

Ahora llega el paso más importante. El esfuerzo de unir todos esos conceptos y combinarlos dentro de un programa destinado a una utilidad “real” de la informática cotidiana.

Tu primer programa útil: Calculadora – Los menús

Me juego lo que quieras a que ya has navegado por InterNet... has utilizado Windows... Has jugado con una PlayStation o con una máquina recreativa... Has entrado al teletexto de un televisor, has programado un video, e incluso... ¡has sacado un café en una máquina de cafés automática! o... ¡Observa la máquina de tickets que hay en el comedor de la universidad!!!!

¿Qué quiero decir con todo esto...?

Pues... ¡que todo funciona con **menús**!

Un menú nos permite elegir entre una serie de opciones, para llevar a cabo una determinada operación.

Tu primera aplicación práctica en Pascal, va a ser una calculadora.
Sencillita, pero verás que funciona, y que ¡es fácil diseñar el programa!

El menú de la calculadora permitirá:

1) Introducir operando A.
2) Introducir operando B.
3) Calcular A + B.
4) Calcular A – B.
5) Calcular A x B.
6) Calcular A / B (División real).

0) Salir

¿qué variables necesitaremos?

`num_a` y `num_b`, como `integers`, nos servirán como operandos A y B.

`resultado` como *integer*, nos servirá para ir almacenando el resultado obtenido en las operaciones de suma, resta y producto y poder así mostrarlo al usuario.

`resultado2` como real, nos servirá para almacenar el resultado de la operación de división real, es decir, con decimales. Si fuera integer, a parte de que el compilador generaría un error, el cociente quedaría recortado de decimales!

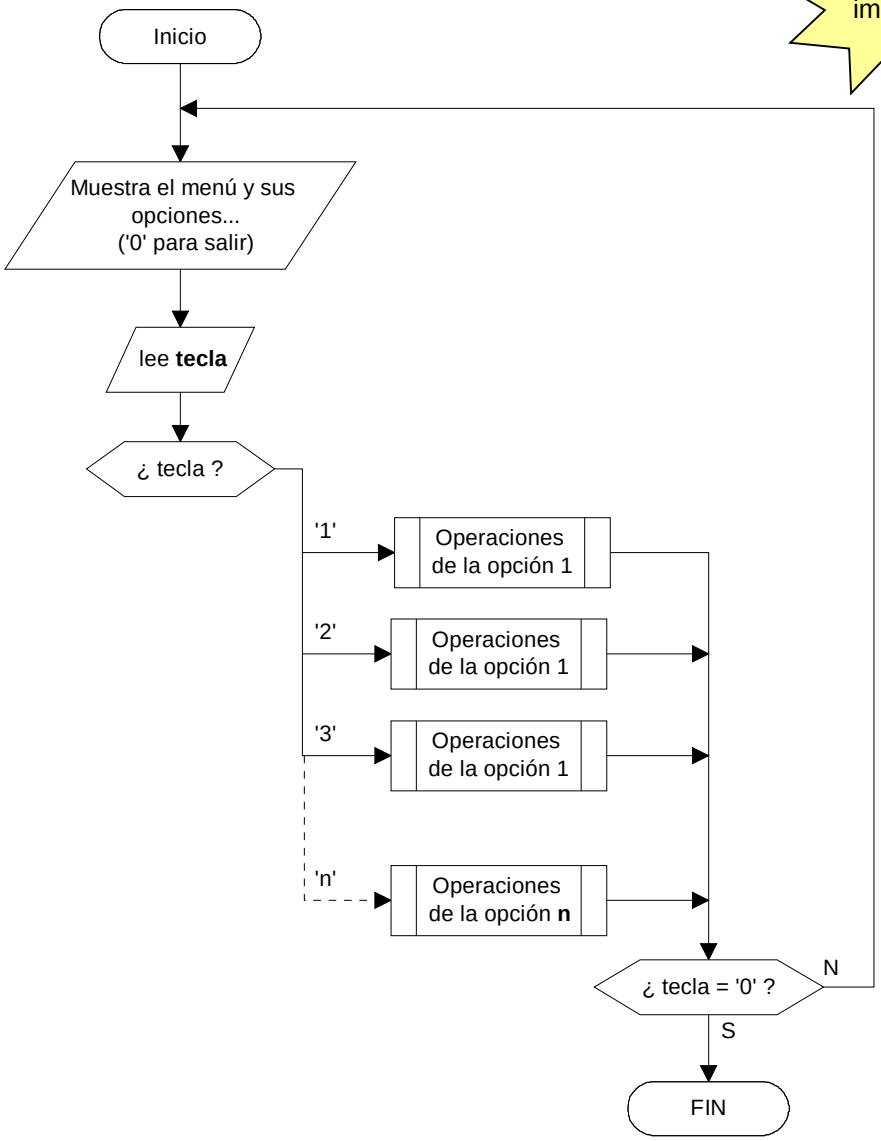
`tecla`, como `char`, nos permitirá desarrollar el menú (recuerda que el menú funcionará con las teclas '`1`', '`2`', '`3`', '`4`', '`5`', '`6`' y '`0`'.

Para hacer los cálculos... observa que son sencillos... suma, resta, multiplicación y división, es decir... utilizaremos los operadores matemáticos **+** para sumar, **-** para restar, ***** para multiplicar y **/** para dividir de forma real (es decir, que el resultado tenga decimales).

Dicho todo esto... empecemos con el diseño del programa!!

Debes memorizar que un menú en Pascal siempre se hace utilizando la siguiente estructura:

¡¡ Página **MUY** importante !!



Y en Pascal, es de la siguiente manera:

```
Repeat

  Muestra el menú y sus opciones... (‘0’ para salir)

  tecla:=ReadKey;

  Case tecla Of
    ‘1’:  Operaciones de la opción 1...
    ‘2’:  Operaciones de la opción 2...
    ‘3’:  Operaciones de la opción 3...
    ‘n’:  Operaciones de la opción n...
  End;

Until tecla=’0’;
```

Primera solución: Calculadora sencilla

```
Program calculadora1;

Uses
   Crt;

Var
   numa, numb, resultado: longint;
   resultado2: real;
   tecla: char;

Begin

   Repeat
      clrscr;
      writeln('CALCULADORA - MENU');
      writeln;
      writeln('1. Introducir operando A.');
      writeln('2. Introducir operando B.');
      writeln('3. Calcular A + B.');
      writeln('4. Calcular A - B.');
      writeln('5. Calcular A * B.');
      writeln('6. Calcular A / B.');
      writeln;
      writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of
         '1': Begin
                  write('Introduce el valor del operando A? ');
                  read(numa);
              End;
         '2': Begin
                  write('Introduce el valor del operando B? ');
                  read(numb);
              End;
         '3': Begin
                  resultado:=numa+numb;
                  writeln(numa, ' + ', numb, ' = ', resultado);
              End;
         '4': Begin
                  resultado:=numa-numb;
                  writeln(numa, ' - ', numb, ' = ', resultado);
              End;
         '5': Begin
                  resultado:=numa*numb;
                  writeln(numa, ' * ', numb, ' = ', resultado);
              End;
         '6': Begin
                  resultado2:=numa/numb;
                  writeln(numa, ' / ', numb, ' = ', resultado2);
              End;
      End;

      ReadKey;

   Until tecla='0';

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

Borramos la pantalla y mostramos el menú.

Esperamos la pulsación de la tecla que indicará la opción a escoger.

Depende del valor de la tecla pulsada, el programa seguirá uno de los caminos.

Primera solución: Calculadora sencilla → Comentarios

Teclea el programa, y pruébalo.

Imprímelo, y línea a línea, estudia con atención qué hace cada instrucción y el porqué de cada cosa.

Ahora trataremos de perfeccionarlo. Para perfeccionar un programa se ha de ser crítico, y a base de pruebas, ir arrinconando esos pequeños detalles que, si se solucionaran, haríamos el programa más rápido, operativo y fácil de usar.

Primero de todo, por estilo, vamos a "dar formato" al menú y a los resultados, gracias a la función `gotoxy`.

La función gotoxy permite mover el cursor de escritura por la pantalla. De este modo, podemos escribir textos en cualquier posición comprendida por las 25 filas y las 80 columnas de caracteres.

Modifica el programa anterior, y actualízalo a "Calculadora2.pas".

Segunda solución: Mejorando la presentación

```
Program calculadora2;

Uses
   Crt;

Var
   numa, numb, resultado: longint;
   resultado2: real;
   tecla: char;

Begin

   Repeat
      clrscr;
      gotoxy(30,10);  writeln('CALCULADORA - MENU');
      gotoxy(30,12);  writeln('1. Introducir operando A.');
      gotoxy(30,13);  writeln('2. Introducir operando B.');
      gotoxy(30,14);  writeln('3. Calcular A + B.');
      gotoxy(30,15);  writeln('4. Calcular A - B.');
      gotoxy(30,16);  writeln('5. Calcular A * B.');
      gotoxy(30,17);  writeln('6. Calcular A / B.');
      gotoxy(30,18);  writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of
         '1': Begin
                 write('Introduce el valor del operando A? ');
                 read(numa);
              End;
         '2': Begin
                 write('Introduce el valor del operando B? ');
                 read(numb);
              End;
         '3': Begin
                 resultado:=numa+numb;
                 writeln(numa, ' + ', numb, ' = ', resultado);
              End;
         '4': Begin
                 resultado:=numa-numb;
                 writeln(numa, ' - ', numb, ' = ', resultado);
              End;
         '5': Begin
                 resultado:=numa*numb;
                 writeln(numa, ' * ', numb, ' = ', resultado);
              End;
         '6': Begin
                 resultado2:=numa/numb;
                 writeln(numa, ' / ', numb, ' = ', resultado2);
              End;
      End;

      ReadKey;

   Until tecla='0';

   writeln('PULSA UNA TECLA PARA SALIR');
   ReadKey;

End.
```

Segunda solución: Mejorando la presentación → Comentarios

Observa que hemos “movido” el menú con simples llamadas a la función `gotoxy`.

Pasemos a integrar la petición por teclado de los 2 operandos. En lugar de pedirlos por separado en dos opciones diferentes, haremos que la opción 1 sea “Introducir operandos”.

Tercera solución: Optimizando

```
Program calculadora3;

Uses
   Crt;

Var
   numa, numb, resultado: longint;
   resultado2: real;
   tecla: char;

Begin

   Repeat
      clrscr;
      gotoxy(30,10);  writeln('CALCULADORA - MENU');
      gotoxy(30,12);  writeln('1. Introducir operandos.');
      gotoxy(30,13);  writeln('2. Calcular A + B.');
      gotoxy(30,14);  writeln('3. Calcular A - B.');
      gotoxy(30,15);  writeln('4. Calcular A * B.');
      gotoxy(30,16);  writeln('5. Calcular A / B.');
      gotoxy(30,17);  writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of
         '1': Begin
                  gotoxy(30, 20); write('Operando A? ');
                  gotoxy(30, 21); write('Operando B? ');
                  gotoxy(42, 20); read(numa);
                  gotoxy(42, 21); read(numb);
              End;
         '2': Begin
                  resultado:=numa+numb;
                  writeln(numa, ' + ', numb, ' = ', resultado);
              End;
         '3': Begin
                  resultado:=numa-numb;
                  writeln(numa, ' - ', numb, ' = ', resultado);
              End;
         '4': Begin
                  resultado:=numa*numb;
                  writeln(numa, ' * ', numb, ' = ', resultado);
              End;
         '5': Begin
                  resultado2:=numa/numb;
                  writeln(numa, ' / ', numb, ' = ', resultado2);
              End;
      End;

      ReadKey;

   Until tecla='0';

End.
```

OJO porque ahora la opción 2 es sumar !!!

Recuerda!!!
Es MUY importante TABULAR (identar), y hacerlo bien!!

Tercera solución: Optimizando → Comentarios

Observa que ahora en la opción 1 pedimos los 2 operandos, y entonces, la opción 2 ya NO es la de introducir el operando B, si no la de Sumar!!!

Y... ¿qué podríamos mejorar...?

Poco a poco... aunque sin dormirte!!

Vamos a posicionar también los resultados acorde con el menú. Ya sabes, con llamadas a `gotoxy`!!

Cuarta solución: Optimizando más...

```
Program calculadora4;

Uses
   Crt;

Var
   numa, numb, resultado: longint;
   resultado2: real;
   tecla: char;

Begin

   Repeat
      clrscr;
      gotoxy(30,10);  writeln('CALCULADORA - MENU');
      gotoxy(30,12);  writeln('1. Introducir operandos.');
      gotoxy(30,13);  writeln('2. Calcular A + B.');
      gotoxy(30,14);  writeln('3. Calcular A - B.');
      gotoxy(30,15);  writeln('4. Calcular A * B.');
      gotoxy(30,16);  writeln('5. Calcular A / B.');
      gotoxy(30,17);  writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of
         '1': Begin
                 gotoxy(30, 20); write('Operando A? ');
                 gotoxy(30, 21); write('Operando B? ');
                 gotoxy(42, 20); read(numa);
                 gotoxy(42, 21); read(numb);
              End;
         '2': Begin
                 resultado:=numa+numb;
                 gotoxy(30, 20);  writeln(numa, ' + ', numb, ' = ', resultado);
              End;
         '3': Begin
                 resultado:=numa-numb;
                 gotoxy(30, 20);  writeln(numa, ' - ', numb, ' = ', resultado);
              End;
         '4': Begin
                 resultado:=numa*numb;
                 gotoxy(30, 20);  writeln(numa, ' * ', numb, ' = ', resultado);
              End;
         '5': Begin
                 resultado2:=numa/numb;
                 gotoxy(30, 20);  writeln(numa, ' / ', numb, ' = ', resultado2);
              End;
      End;

      ReadKey;

   Until tecla='0';

End.
```

Cuarta solución: Optimizando más... → Comentarios

Si pruebas bien el programa, verás que tienes que pulsar demasiadas veces una tecla para continuar con otras operaciones...

Observa los cambios de la quinta solución, y comprueba las diferencias.

Quinta solución: Optimizando aún más...

```
Program calculadora5;

Uses
   Crt;

Var
   numa, numb, resultado: longint;
   resultado2: real;
   tecla: char;

Begin

   Repeat

      gotoxy(30,10);  writeln('CALCULADORA - MENU');
      gotoxy(30,12);  writeln('1. Introducir operandos.');
      gotoxy(30,13);  writeln('2. Calcular A + B.');
      gotoxy(30,14);  writeln('3. Calcular A - B.');
      gotoxy(30,15);  writeln('4. Calcular A * B.');
      gotoxy(30,16);  writeln('5. Calcular A / B.');
      gotoxy(30,17);  writeln('0. Salir.');

      tecla:=ReadKey;

      gotoxy(30, 20); write('                                        ');
      gotoxy(30, 21); write('                                        ');

      Case tecla Of
         '1': Begin
                 gotoxy(30, 20); write('Operando A? ');
                 gotoxy(30, 21); write('Operando B? ');
                 gotoxy(42, 20); read(numa);
                 gotoxy(42, 21); read(numb);
              End;
         '2': Begin
                 resultado:=numa+numb;
                 gotoxy(30, 20);  writeln(numa, ' + ', numb, ' = ', resultado);
              End;
         '3': Begin
                 resultado:=numa-numb;
                 gotoxy(30, 20);  writeln(numa, ' - ', numb, ' = ', resultado);
              End;
         '4': Begin
                 resultado:=numa*numb;
                 gotoxy(30, 20);  writeln(numa, ' * ', numb, ' = ', resultado);
              End;
         '5': Begin
                 resultado2:=numa/numb;
                 gotoxy(30, 20);  writeln(numa, ' / ', numb, ' = ', resultado2);
              End;
      End;

   Until tecla='0';

End.
```

Quitamos el ReadKey !!

Quinta solución: Optimizando aún más...→ Comentarios

Observa que ya NO borramos toda la pantalla, si no únicamente borramos las dos líneas que cambian durante el programa.

También eliminamos el último `Readkey`, porque ya que no borramos toda la pantalla, si no únicamente las líneas de resultados, y después de realizar una operación, el programa ya no tiene que esperar la pulsación de una tecla, si no, esperar a una nueva opción del menú, manteniendo por el momento el resultado de la última operación en pantalla.

Los arrays: teoría

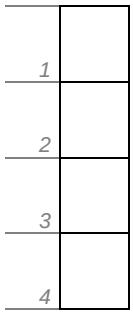
La definición técnica de un array es la de un “grupo indexado” de elementos con un mismo nombre, diferenciados por la posición que ocupan dentro del rango de posiciones existente.

Pero para que lo entiendas mejor, lo explicaré gráficamente...

Imagina un armario con 4 cajones, numerados de 1 al 4.

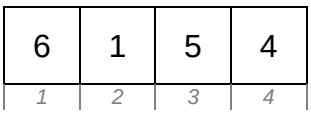
Lo podemos dibujar horizontal:

O vertical:



Más adelante, verás que depende para qué, “ayuda” más una forma de dibujarlo que otra, pero también va a gusto de cada uno.
De momento las dibujaremos horizontales para ahorrar espacio en las hojas.

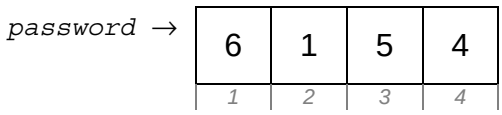
Bueno... imagina que en cada “cajón” del armario, cabe un número entero (variable de tipo `integer`):



Y ahora... piensa para qué puede servir esto...

Por ejemplo... para almacenar la contraseña de un teléfono móvil, de la libreta de ahorros o tarjeta de crédito del banco, o de una caja fuerte.

Te muestro de nuevo la última figura:



Y entro en detalles...:

La figura superior, muestra un *array* de *integers* con rango de `1` al `4`.
Fiíjate que le he puesto una etiqueta... su nombre.
Es un *array* de *integers* con rango de `1` al `4` llamado `password`.

La manera de definir este array en Pascal es así:

```
password : Array [1..4] Of Integer;
```

Por supuesto, dentro de la cláusula *Var*, ¡¡ como toda variable !!!

Atención!!!
Recuerda que utilizaremos constantes...
No te sorprenda verlas en el código del programa.

Y ahora... ¡¡ trabajemos con este array !!

Una vez definido, para acceder por ejemplo al primer cajón del armario... hablemos más técnicamente: para acceder a la primera posición del array `password`, escribiremos `password[1]`.

Para acceder a la cuarta posición del array `password` (posición 4), escribiremos `password[4]`.

Fácil, ¿no?

Practiquemos un poco!!

Vamos a diseñar un programa que pida por teclado los 4 números del password, y nos diga con palabras los números del password.
Es decir, si introducimos `4, 3, 1 y 5`, el programa mostrará:

```
cuatro, tres, uno, cinco
```

De momento, suponemos que el usuario sólo entrará números del 0 al 9. Más adelante protegeremos la entrada, obligando al usuario a reintroducir el número si no se encuentra entre 0 y 9... ¿ok? Es decir, sólo un dígito en cada posición (podrían caber números más grandes, pero no nos interesan para el ejemplo!).

Arrays 1

Te muestro el programa:

```
program Arrays1;

uses
   Crt;

Const
   MAX=4;

var
   password : Array [1..MAX] Of Integer;

begin

    write('Primer digito? ');
    readln(password[1]);
    write('Segundo digito? ');
    readln(password[2]);
    write('Tercer digito? ');
    readln(password[3]);
    write('Cuarto digito? ');
    readln(password[4]);

    writeln();
    writeln();

    case password[1] of
         0: write('cero');
         1: write('uno');
         2: write('dos');
         3: write('tres');
         4: write('cuatro');
         5: write('cinco');
         6: write('seis');
         7: write('siete');
         8: write('ocho');
         9: write('nueve');
    End;

    write(',');

    case password[2] of
         0: write('cero');
         1: write('uno');
         2: write('dos');
         3: write('tres');
         4: write('cuatro');
         5: write('cinco');
         6: write('seis');
         7: write('siete');
         8: write('ocho');
         9: write('nueve');
    End;

    write(',');
```

```
    case password[3] of
         0: write('cero');
         1: write('uno');
         2: write('dos');
         3: write('tres');
         4: write('cuatro');
         5: write('cinco');
         6: write('seis');
         7: write('siete');
         8: write('ocho');
         9: write('nueve');
    End;

    write(',');

    case password[4] of
         0: write('cero');
         1: write('uno');
         2: write('dos');
         3: write('tres');
         4: write('cuatro');
         5: write('cinco');
         6: write('seis');
         7: write('siete');
         8: write('ocho');
         9: write('nueve');
    End;


    readkey;

end.
```

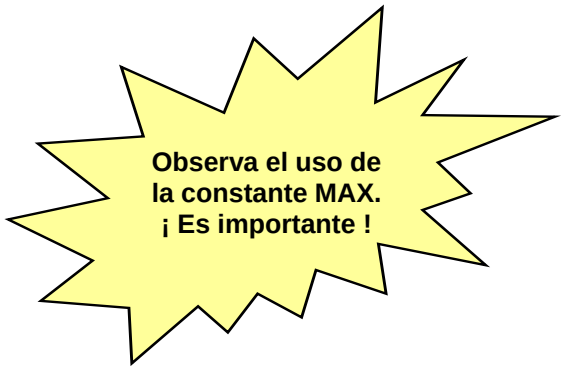
Un poco largo... ¿no?

Observa que se repite un bloque de código similar, 4 veces!

Observa que en la próxima solución utilizamos un bucle `For`, y recorremos el índice del array desde la posición 1 hasta la 4, mostrando una a una, el número con palabras que contiene.

Arrays 2

```
Program Arrays2;

uses
    Crt;

Const
   MAX=4;

var
   password : Array [1..MAX] Of Integer;
   cont: Integer;

begin

    write('Primer digito? ');
    readln(password[1]);
    write('Segundo digito? ');
    readln(password[2]);
    write('Tercer digito? ');
    readln(password[3]);
    write('Cuarto digito? ');
    readln(password[4]);

    writeln();
    writeln();

    for cont:=1 to MAX do
        begin
             case password[cont] of

                  0: write('cero');
                  1: write('uno');
                  2: write('dos');
                  3: write('tres');
                  4: write('cuatro');
                  5: write('cinco');
                  6: write('seis');
                  7: write('siete');
                  8: write('ocho');
                  9: write('nueve');
             end;

             write(',');
        end;

    readkey;

End.
```



¿Qué...? ¡¡Se ha reducido, ehh!!

Pero, si por ejemplo introducimos `3`, `5`, `1`, `2` observaremos lo siguiente por pantalla:

```
Primer digito? 3
Segundo digito? 5
Tercer digito? 1
Cuarto digito? 2


tres, cinco, uno, dos,
```

Esto sobra!!!

¡¡ La última coma sobra!!
Observa el `If` que hago en la siguiente solución para corregir el problema.

Arrays 3

```
Program Arrays3;

uses
    Crt;

Const
   MAX=4;

var
   password : Array [1..MAX] Of Integer;
   cont: Integer;

begin

    write('Primer digito? ');
    readln(password[1]);
    write('Segundo digito? ');
    readln(password[2]);
    write('Tercer digito? ');
    readln(password[3]);
    write('Cuarto digito? ');
    readln(password[4]);

    writeln;
    writeln;

    for cont:=1 to MAX do
        begin
             case password[cont] of

                  0: write('cero');
                  1: write('uno');
                  2: write('dos');
                  3: write('tres');
                  4: write('cuatro');
                  5: write('cinco');
                  6: write('seis');
                  7: write('siete');
                  8: write('ocho');
                  9: write('nueve');
             end;

             if cont<>MAX then write(',');

        end;

    readkey;

End.
```

¡Fácil...! En la última "pasada" del `For`, NO ponemos la coma, porque ya estamos en el final.

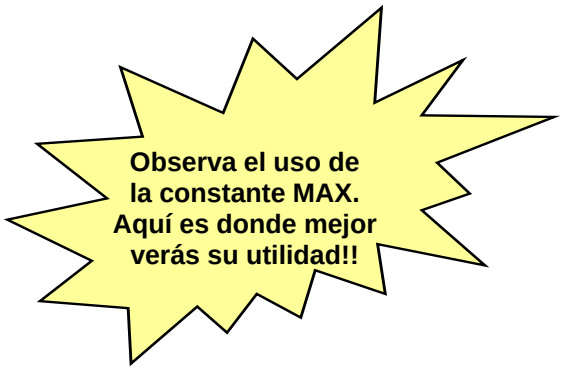
Recuerda SIEMPRE!!! <u>MUY IMPORTANTE</u>
Es MUY importante TABULAR (identar), y hacerlo bien.
Así, vemos mejor la estructura de un programa, y lo que pertenece a cada una de las estructuras de control.

Arrays 4

Vamos a rematar la solución del programa. Lo voy a mostrar aún más compactado...

Eso sí: en lugar de ir preguntando "Primer dígito?", "Segundo dígito?", etc... pondremos "Dígito 1?", "Digito 2?", etc...

Observa:




```
Program Arrays4;

uses
    Crt;

Const
   MAX=4;

var
   password : Array [1..MAX] Of Integer;
   cont: Integer;

begin

    for cont:=1 to MAX do
        begin
             write('Digito ', cont, '? ');
             readln(password[cont]);
        end;

    writeln;
    writeln;

    for cont:=1 to MAX do
        begin
             case password[cont] of

                  0: write('cero');
                  1: write('uno');
                  2: write('dos');
                  3: write('tres');
                  4: write('cuatro');
                  5: write('cinco');
                  6: write('seis');
                  7: write('siete');
                  8: write('ocho');
                  9: write('nueve');
             end;

             if cont<>MAX then write(',');
        end;

    readkey;

End.
```

*Aprovecho la variable **cont** del **For**, que va desde 1 hasta cuatro, para pedir los 4 números.*

Prueba en todas las versiones del programa, a entrar números superiores al `9`... El `12`, el `120`, etc... y prueba de introducir números negativos... el `-1`, el `-14`, etc...

¡No puede ser! ¡Única y exclusivamente debe aceptar dígitos!

Arrays 5: Versión final de Arrays: La entrada protegida

Por último, vamos a hacer aquello que te comenté en la página anterior: obligar al usuario a introducir exculsivamente números del 0 al 9... es decir, dígitos.

Para ello meteremos la entrada dentro de un bucle `Repeat...Until`.
Es decir, que el usuario entre el número, y repita la entrada hasta que sea un dígito!!

Observa el resultado:

```
Program Arrays5;

uses
    Crt;

Const
   MAX=4;

var
   password : Array [1..MAX] Of Integer;
   cont: Integer;
   digito: Integer;

begin

    for cont:=1 to MAX do
        begin

            Repeat

                write('Digito ', cont, '? ');
                readln(digito);
                if (digito<0) or (digito>9) then
                    writeln('ERROR: Escribe un digito valido [0..9]');

            Until (digito>=0) and (digito<=9);

            password[cont]:=digito;

        end;

    writeln;
    writeln;

    for cont:=1 to MAX do
        begin
            case password[cont] of

                0: write('cero');
                1: write('uno');
                2: write('dos');
                3: write('tres');
                4: write('cuatro');
                5: write('cinco');
                6: write('seis');
                7: write('siete');
                8: write('ocho');
                9: write('nueve');
            end;

            if cont<>MAX then write(',');
        end;

    readkey;
End.
```

Repeat..Until para forzar una entrada protegida.
Solo permite introducir dígitos (números del 0 al 9).

Ordenación de arrays

El concepto de ordenar un array, supongo que lo tendrás claro:

Array sin ordenar *Array ordenado*

Se trata de poner en orden los elementos de cada casilla. Si son números, del menor al mayor o del mayor al menor, según interese.

Si son letras, pues alfabéticamente, etc...

Hay 3 sistemas elementales de ordenación:

- Método de ordenación por selección
- Método de ordenación por inserción
- Método de ordenación por burbuja (también conocido como de intercambio)

Aunque en este libro de apuntes sólo te explicaré el Método por burbuja, que es el más sencillo de entender y utilizar.

Los otros dos, debes estudiarlos también con los apuntes de clase y con otros libros!

Ordenación por burbuja (explicación)

También se le denomina método de ordenación por intercambio.

Su nombre de burbuja, viene del efecto parecido que hacen las burbujas: van ascendiendo.

Su desventaja es que tarda mucho, y el tiempo de ordenación no es directamente proporcional al número de elementos a ordenar, sino que es exponencial.

Se utilizan dos variables de control: `i`, `j`

Consiste en un bucle `for` anidado dentro de otro bucle `for`.

El bucle externo, recorre desde el primer elemento hasta el último, y para cada parada, se ejcuta el bucle interno que recorre desde el último elemento hasta la posición actual del bucle externo.
Para cada parada del bucle interno, se comparan las dos posiciones que apuntan `i` y `j`, y si es menor, se intercambian los valores entre sí, si no, no.

Soy consciente que cuesta de entender a la primera el funcionamiento de este y de los otros sistemas de ordenación. Pero tú también debes ser consciente de que es dificil de explicar por escrito.
El truco consiste en picar el ejemplo, e ir analizando el programa, junto el algoritmo y con papel y lápiz al lado, simular la ordenación.

Esquema general de funcionamiento:

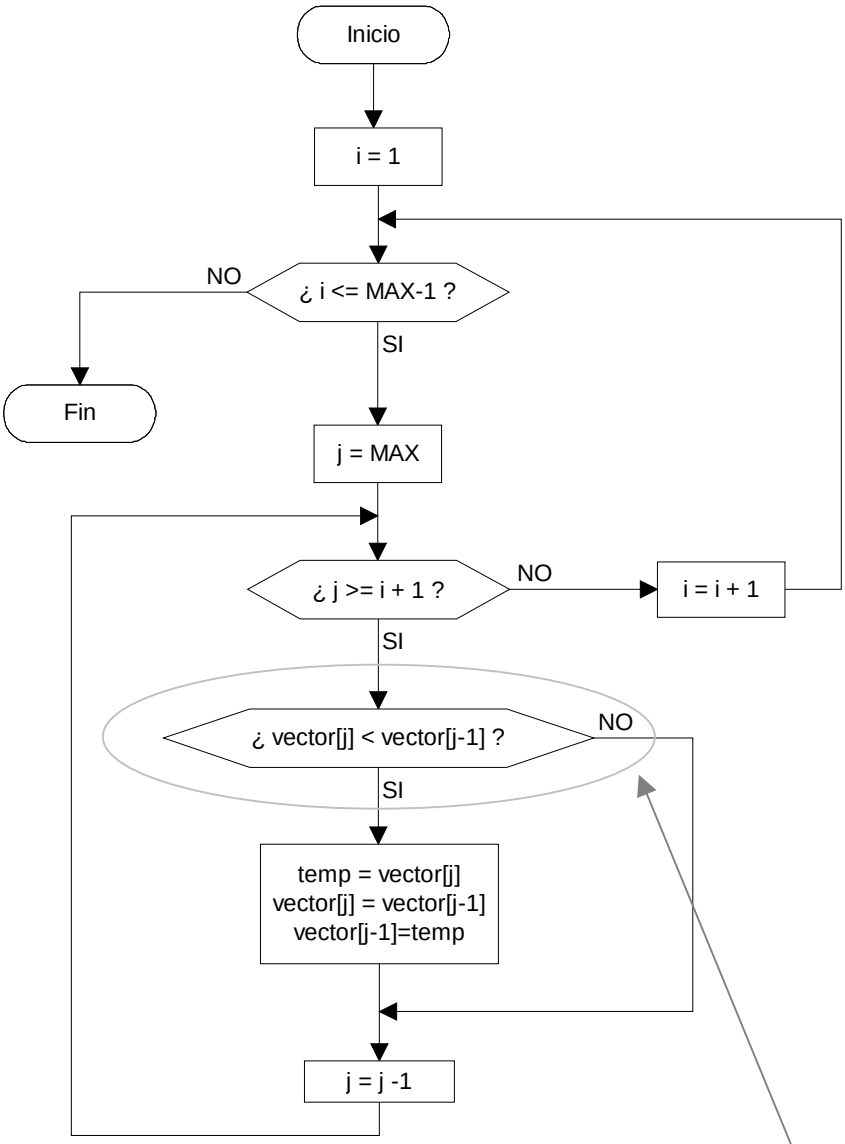
	Inicialmente desordenado	1er paso	2º paso	3º paso
vector[1]	6	1	1	1
vector[2]	4	6	4	4
vector[3]	5	4	6	5
vector[4]	1	5	5	6

Esquema de trabajo de los dos bucles anidados:

i = 1 j = ~~4~~ ~~3~~ ~~2~~ 1	i = 2 j = ~~4~~ ~~3~~ 2	i = 3 j = ~~4~~ 3
6 4 5 (1) 6 4 (1) 5 6 (1) 4 5 (1) 6 4 5	1 6 4 5 1 6 (4) 5 1 (4) 6 5	1 4 6 (5) 1 4 (5) 6

Aquí observamos el efecto "burbuja".

Ordenación por burbuja: Algoritmo



Si en la comparación utilizamos el símbolo '<', ordenará de menor a mayor.

Si en la comparación utilizamos el símbolo '>', ordenará de mayor a menor.

Ordenación por burbuja: Código fuente de ejemplo

Aquí tienes un ejemplo sencillo del uso del método de ordenación por burbuja, para ordenar un array de 4 enteros.

Pícalo, estúdialo, pruébalo, modifícalo e... intenta entenderlo!!

No te extrañes si te cuesta, y no dudes en consultar a los profesores!!

```
Program OrdenacionPorBurbuja;

Uses
   Crt;

Const
   MAX=4;

Var
   vector: array[1..MAX] of integer;
   temp, i, j: integer;


Begin

   vector[1]:=6;
   vector[2]:=1;
   vector[3]:=4;
   vector[4]:=5;

   // Si queremos probar el sistema entrando los números manualmente,
   // escribiremos, en lugar de las 4 lineas de arriba, estas:
   //
   //  for i:=1 to MAX do
   //  Begin
   //     write('Numero ,' i, '? ');
   //     readln(vector[i]);
   //  End;


   for i:=1 to MAX-1 do
      for j:=MAX downto i+1 do
         if vector[j]<vector[j-1] then
         Begin
            temp:=vector[j];
            vector[j]:=vector[j-1];
            vector[j-1]:=temp;
         End;



   for i:=1 to MAX do
      writeln(vector[i]);

   ReadKey;

End.
```

Array especial: variable de tipo `string`

Para declarar una variable de tipo `string`, haremos:

```
Var
   texto2: string;
```

Un `string` es un conjunto indexado de carácteres. El primer carácter de todo `string` ocupa la posición 1.

Si durante el programa hacemos lo siguiente:

```
   texto2:= 'Te espero';
```

Gráficamente, podemos entenderlo como una tira de casillas con una letra en cada una:

'T'	'e'	' '	'e'	's'	'p'	'e'	'r'	'o'
1	2	3	4	5	6	7	8	9

Y si ahora queremos imprimir el 2º carácter, por ejemplo, haremos:

```
   writeln('El segundo carácter es: ', texto2[2]);
```

Si queremos saber la longitud de la cadena, haremos:

```
   writeln('La cadena mide', length(texto2), 'caracteres.');
```

Recuerda SIEMPRE!!! MUY IMPORTANTE

El carácter de la primera posición será el 1 para todas las cadenas.

Fíjate que con lo que contiene ahora mismo Texto2, es decir, la cadena 'Te espero', su longitud, que la calcularemos con `length(texto2)`, es de 9 carácteres, y SIEMPRE, la posición del último carácter de toda cadena, coincide con el valor de su tamaño, en este caso 9!! (Mira el dibujo de las casillas de arriba!).

Y si queremos mostrar el texto al revés, por ejemplo hacemos un bucle `For-DownTo` que empiece desde el último carácter hasta el primero, mostrando para cada vez el caracter de la posición indicada por la variable de control del bucle `For`.

En la siguiente página veremos un ejemplo de todo lo comentado.

También se pueden hacer operaciones sobre las cadenas, como por ejemplo pasar todas sus letras a mayúsculas (Función `upcase`) o a minúsculas (Funcion `lowercase`).

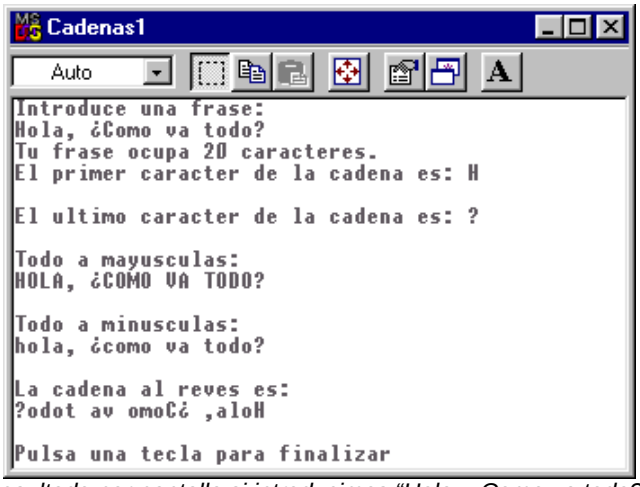
Y hay más funciones de tratamiento de las cadenas.

En el Apéndice F dispones de una gran colección de funciones de Pascal explicadas una a una, con ejemplos para cada una!

¡¡Estudia todos los ejemplos del Apéndice F!! Te serán muy útiles!!!

Aquí tienes un ejemplo de tratamiento de la cadena de carácteres que introduzcamos por teclado, y tienes un "pantallazo" del resultado:

```
Program Cadenas1;

uses Crt;

Var
   texto: string;
   z: integer;

begin

   writeln('Introduce una frase:');
   readln(texto);

   writeln('Tu frase ocupa ', length(texto), ' caracteres.');

   writeln('El primer caracter de la cadena es: ', texto[1]);
   writeln;
   writeln('El ultimo caracter de la cadena es: ', texto[length(texto)]);
   writeln;

   writeln('Todo a mayusculas:');
   writeln(upcase(texto));

   writeln;

   writeln('Todo a minusculas:');
   writeln(lowercase(texto));

   writeln;

   writeln('La cadena al reves es:');

   for z:=length(texto) downto 1 do
      write(texto[z]);

   writeln;
   writeln;

   writeln('Pulsa una tecla para finalizar');
   readkey;

End.
```



Resultado por pantalla si introducimos "Hola, ¿Como va todo?"

Típico Programa: La Agenda de teléfonos

Vamos a utilizar un ejemplo muy claro, en el que utilizaremos 3 arrays.

Diseñaremos un programa que permita almacenar* nombres de personas, incluyendo su edad y su número de teléfono.

Pero... antes de eso... veamos cómo funcionará el programa frente al usuario... es decir... ¿cómo será la interficie...? ¿cómo nos comunicamos con el usuario?

Repasa el tema de los menús...

Observa, teclea y prueba el programa en su primera fase. Aquí simplemente creamos el menú.

Primera fase

```
program Agenda1;

uses
    Crt;

var
    tecla: char;

begin

   Repeat

      // Mostramos las opciones del menú
      gotoxy(30,10);  writeln('AGENDA - MENU');
      gotoxy(30,12);  writeln('1. Agregar persona.');
      gotoxy(30,13);  writeln('2. Listar agenda.');
      gotoxy(30,14);  writeln('3. Eliminar persona.');
      gotoxy(30,16);  writeln('4. Borrar toda la agenda.');
      gotoxy(30,18);  writeln('0. Salir.');

      tecla:=ReadKey;   // Se detiene el programa en espera de una tecla.

      Case tecla Of
         '1': Begin
                   // Código para agregar una persona a la agenda.
              End;
         '2': Begin
                   // Código para listar todas las personas de la agenda.
              End;
         '3': Begin
                   // Código para eliminar una persona de la agenda.
              End;
         '4': Begin
                   // Código para borrar todas las personas de la agenda.
              End;
      End;

   Until tecla='0';

end.
```

*Almacenamos en memoria, NO en disco!!! En disco, lo estudiarás en Programación.

Segunda fase

Empecemos en serio con la agenda...

Para ello, utilizaremos 3 arrays:

- Nombres: array Strings
- Edades: array de Integers
- Teléfonos: array de Strings

Y quizás aquí te preguntes... ¿por qué un `array` de `strings` para los teléfonos, si son sólo números?
Pues... porque recuerda que un teléfono puede darse de muchas maneras...

496552432
496.55.24.32
496-55-24-32
496 55 24 32

Entonces... los puntos... los guiones... los espacios... NO pueden ponerse en un `integer`!!!

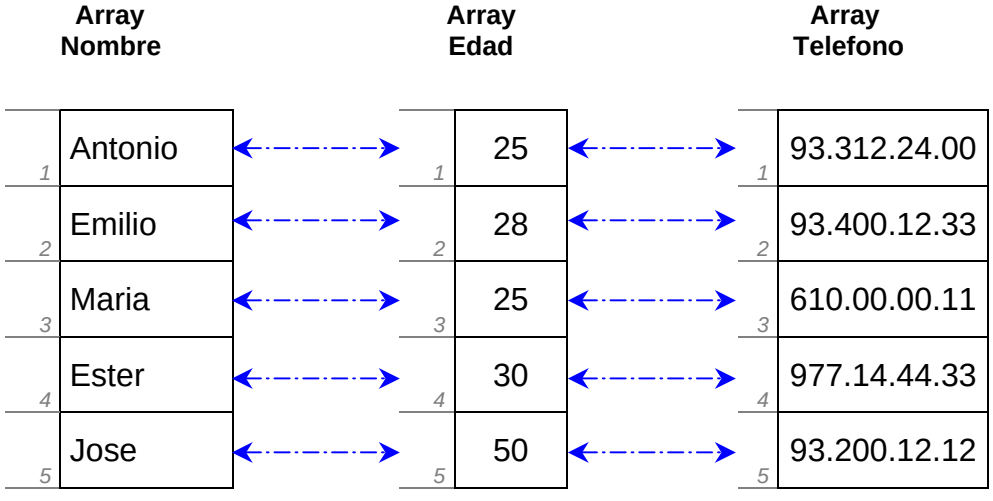
Y para no complicar la cosa con entradas protegidas que eviten poner guiones, puntos, espacios, etc... lo hacemos con cadenas y así, pongas lo que pongas, se almacenará correctamente.

Con los nombres y las edades no hay problema, porque los nombres son textos y las edades son exclusivamente números !!

Los tres `arrays`, evidentemente son independientes. Eso significa, que, para vincular el nombre, la edad y el teléfono para una misma persona, utilizaremos el índice del Array.

Es decir: existirá la persona 1, la persona 2, la persona 3, etc...

Observa el gráfico:



En la segunda fase, diseñaremos la entrada de personas y el listado, para que veas que el programa empieza a funcionar.

Observa lo que añado: la entrada de personas, y el listado:

```
Program Agenda2;

Uses
   Crt;

Const
   MAX=5;

Var
   tecla: char;
   nombres: Array[1..MAX] of String;
   edades: Array[1..MAX] of integer;
   telefonos: Array[1..MAX] of String;
   cont, z: integer;

Begin

      cont:=0;

   Repeat

      clrscr;
      gotoxy(30,10);  writeln('AGENDA - MENU');
      gotoxy(30,12);  writeln('1. Agregar persona.');
      gotoxy(30,13);  writeln('2. Listar agenda.');
      gotoxy(30,14);  writeln('3. Eliminar persona.');
      gotoxy(30,16);  writeln('4. Borrar toda la agenda.');
      gotoxy(30,18);  writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of

         // ENTRADA DE PERSONAS:
         '1': Begin

                  if cont<MAX then
                     begin
                        gotoxy(30,20); write('Nombre?');
                        gotoxy(30,21); write('Edad?');
                        gotoxy(30,22); write('Telefono?');

                        gotoxy(39,20); readln(nombres[cont+1]);
                        gotoxy(37,21); readln(edades[cont+1]);
                        gotoxy(41,22); readln(telefonos[cont+1]);

                        cont:=cont+1;
                     end
                  else
                     begin
                        gotoxy(30,20); write('La agenda esta llena!!!');
                        gotoxy(30,25); write('Pulsa una tecla para continuar...');
                        ReadKey;
                     end;
               end;
```

```
         // LISTADO DE LA AGENDA:
         '2': Begin
                 clrscr;

                 for z:=1 to cont do
                    writeln(z, ' - ',
                                    nombres[z], ' - ',
                                    edades[z], ' - ', telefonos[z]);

                 writeln('Pulsa una tecla para continuar...');
                 ReadKey;
              End;
         '3': Begin

              End;
         '4': Begin

              End;
      End;

   Until(tecla='0');

End.
```

Explicación

La entrada de personas

Nuestra agenda tiene la capacidad de 5 personas como máximo.
Inicilamente, habrán CERO personas en la agenda.
Necesitaremos una variable contador, para apuntarnos en todo momento cuántas personas hay en la agenda, y así controlar si está llena o si aún quedan espacios libres.

Observa que existe la variable `cont`, que inicialmente valdrá 0 (`cont:=0`).

Entonces, te tiene que quedar claro, que si `cont` es 5, la agenda está llena, y si es menor que 5, es que queda algún espacio libre.

Evidentemente, `cont` sólo debe tomar valores del 0 al 5. Fíjate, que queda totalmente controlado.

Cuando “intentamos” añadir una persona, primero tenemos que ver si `cont` es menor que 5.
Si es así, es que hay algún espacio libre para almacenar una persona más, con lo que iniciamos el proceso de inserción de la persona.
Si no es menor que 5, sólo puede significar que es igual a 5, con lo que... LA AGENDA ESTÁ LLENA!!

Observa la estructura básica de lo que hacemos:

```
if cont<MAX then
   begin
   // PEDIMOS LOS DATOS DE UNA PERSONA, Y LOS INSERTAMOS EN LOS ARRAYS
   end
else
   begin
   // DAMOS UN MENSAJE CONFORME LA AGENDA ESTÁ LLENA... NO CABEN MÁS PERSONAS!!
   end;
```

Si queda espacio libre en la agenda, procedemos a la inserción de una persona.
Primero mostramos los mensajes “Nombre?”, “Edad?”, “Telefono?” con el `write`, y con ayuda del `gotoxy` para conseguir una buena presentación:

```
gotoxy(30,20); write('Nombre?');
gotoxy(30,21); write('Edad?');
gotoxy(30,22); write('Telefono?');
```

Y seguidamente, con el readln, ayudándonos con el gotoxy, solicitamos los datos de la persona.
Fíjate que se almacenan en una fila de cada array de datos: `nombres`, `edades` y `telefonos`.

Pero... ¿en qué fila...?
En la fila indicada por la expresión: `cont+1`.
¿Por qué hacemos esto?
Pues... si inicialmente, cont está a CERO, quiere decir que NO hay personas en la agenda... es decir: que la agenda está bacía.
La primera fila de los arrays de datos, es la `1`. Así pues, la primera persona se almacenará en la posición `cont+1`, es decir, en la posición `0+1`... o sea... en la posición `1`.

Si, resulta que YA hay UNA persona en la agenda, `cont` estará a `1`, y la persona siguiente a almacenar, deberá guardarse en la fila `2`... y de nuevo, como se almacena en la posición `cont+1` de los arrays, se almacenará en la posición `1+1`, es decir, en la posición 2 (segunda persona!).

Observa:

```
gotoxy(39,20); readln(nombres[cont+1]);
gotoxy(37,21); readln(edades[cont+1]);
gotoxy(41,22); readln(telefonos[cont+1]);
```

Explicación

Listado de la agenda

Ya hemos solucionado la inserción de personas dentro de nuestra agenda.
Ahora te explico la técnica para listar por pantalla las personas que contiene nuestra agenda.

Hasta aquí está claro que `cont` nos dice cuántas personas hay en la agenda.
Podrás deducir, que si la agenda está vacía, `cont=0`; si la agenda contiene dos personas, `cont=2`, y si la agenda está llena, `cont=5`.

También debes saber, que si mostramos por pantalla el contenido de la primera posición de los arrays, obtendremos los datos de la primera persona.

Es decir, si la primera persona que hemos insertado es Jose Gutierrez de 21 años con el teléfono 612.45.23.11, si ejecutamos la siguiente instrucción:

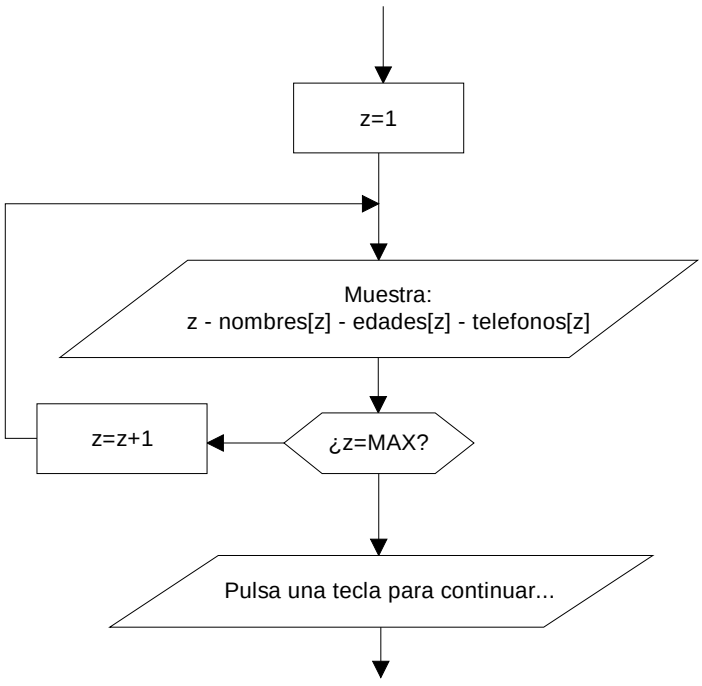
```
writeln(nombres[1], " tiene ", edades[1], "anyos con el telefono ", telefonos[1]);
```

Nos mostrará lo siguiente por pantalla:

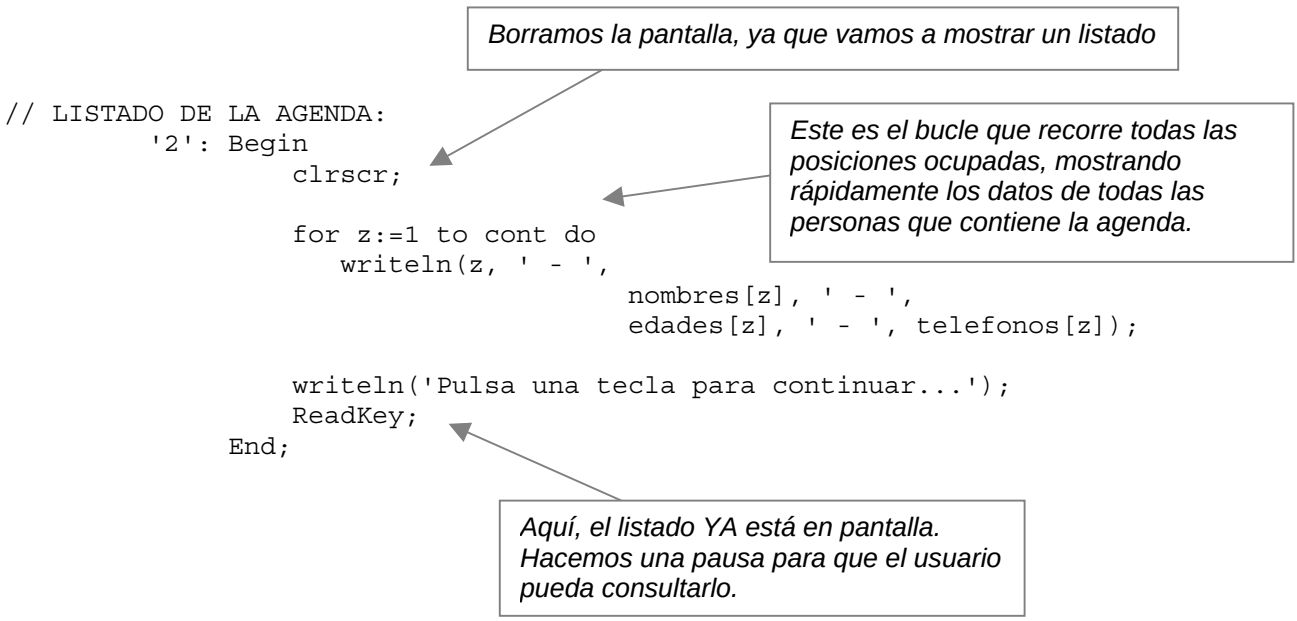
```
Jose Gutierrez tiene 21 anyos con el telefono 612.45.23.11
```

Para listar todas las personas que contiene la agenda, de lo que se trata, es de ejecutar un bucle que vaya desde la posición 1 hasta la posición que indique `cont`, y en cada uno de los pasos del bucle, imprimir los arrays apuntados, mostrando así la información de cada persona, una a una.
Como el bucle se ejecuta rápidamente, lo que observamos en pantalla es una lista de las personas que contiene nuestra agenda.

El algoritmo es el siguiente:



Examinemos el código de la opción 2, que es la de listar:



Tercera fase

Ahora ya tenemos una agenda en la que podemos insertar personas, y listar el contenido de dicha agenda.

En esta tercera fase, diseñaremos el algoritmo que nos permitirá eliminar una persona, y después, veremos lo sencillo que es borrar toda la agenda de golpe.

Os muestro el listado de todo el programa:

```
Program Agenda3;

Uses
   Crt;

Const
   MAX=5;

Var
   tecla: char;
   nombres: Array[1..MAX] of String;
   edades: Array[1..MAX] of integer;
   telefonos: Array[1..MAX] of String;
   cont, z: integer;

Begin

      cont:=0;

   Repeat

      clrscr;
      gotoxy(30,10);  writeln('AGENDA - MENU');
      gotoxy(30,12);  writeln('1. Agregar persona.');
      gotoxy(30,13);  writeln('2. Listar agenda.');
      gotoxy(30,14);  writeln('3. Eliminar persona');
      gotoxy(30,16);  writeln('4. Borrar toda la agenda');
      gotoxy(30,18);  writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of
         '1': Begin

                  if cont<MAX then
                     begin
                        gotoxy(30,20); write('Nombre?');
                        gotoxy(30,21); write('Edad?');
                        gotoxy(30,22); write('Telefono?');

                        gotoxy(39,20); readln(nombres[cont+1]);
                        gotoxy(37,21); readln(edades[cont+1]);
                        gotoxy(41,22); readln(telefonos[cont+1]);

                        cont:=cont+1;
                     end
                  else
                     begin
                        gotoxy(30,20); write('La agenda esta llena!!!');
                        gotoxy(30,25); write('Pulsa una tecla para continuar...');
                        ReadKey;
                     end;
              end;
         '2': Begin
                  clrscr;

                  for z:=1 to cont do
                     writeln(z, ' - ', nombres[z],' - ',edades[z], ' - ', telefonos[z]);

                  writeln('Pulsa una tecla para continuar...');
                  ReadKey;
              End;
```

```
         '3': Begin
                  clrscr;

                  if(cont<>0) then
                  Begin
                     Repeat
                        for z:=1 to cont do
                        Begin
                           writeln(z, ' - ', nombres[z],' - ',edades[z], ' - ', telefonos[z]);
                        End;

                        writeln('Elije el item a eliminar [1..', cont, '] 0 para salir.');
                        tecla:=Readkey;

                        if (tecla>='1') and (tecla<='4') then
                           for z:=ord(tecla)-ord('0') to cont do
                           Begin
                              nombres[z]:=nombres[z+1];
                              edades[z]:=edades[z+1];
                              telefonos[z]:=telefonos[z+1];
                           End;

                        if (tecla>='1') and (tecla<='5') then cont:=cont-1;

                        clrscr;

                     Until tecla='0';

                     tecla:='3';    // Restauramos el valor de tecla, para que no salga del programa.

                  End
                  else
                  Begin
                     writeln('LA AGENDA ESTA VACIA!! No hay nada que eliminar!!');
                     writeln('Pulsa una tecla para continuar...');
                     ReadKey;
                  End;
              End;
         '4': Begin
                  if(cont<>0) then
                  Begin
                     cont:=0;
                     clrscr;
                     writeln('AGENDA BORRADA TOTALMENTE!');
                     writeln('Pulsa una tecla para continuar...');
                     ReadKey;
                  End
                  else
                  Begin
                     clrscr;
                     writeln('LA AGENDA ESTA VACIA!! No hay nada que eliminar!!');
                     writeln('Pulsa una tecla para continuar...');
                     ReadKey;
                  End;
              End;
      End;

   Until tecla='0';
End.
```

Explicación

Examina el código de las opciones 3 y 4 mientras lees esta explicación, e irás entendiendo los procesos.

Y recuerda... si hay algo que no entiendas... ¡acude a los profesores! ¡búscate un profesor de repaso si lo ves muy negro! Si es una duda puntual... consúltamela directamente escribiendo a sergi2@sergi2.com.

Borrar un ítem (borrar una persona de la agenda)

Si picas el programa `Agenda3` de las anteriores dos páginas, verás que la agenda funciona según las especificaciones iniciales.

Ahora te explico cómo borramos una persona (opción 3).

Lo primero que haremos, será comprobar que existan personas dentro de la agenda, porque si no... ¡no tiene sentido eliminar ninguna persona!

Si no existen personas en la agenda, se mostrará un mensaje de advertencia conforme “LA AGENDA ESTÁ VACÍA!! No hay nada que eliminar!!”.

Si existen personas dentro de la agenda, lo que haremos será listar todas las personas de la agenda (exactamente como lo hacemos en la opción de listar), y como están identificadas con un número del 1 a `cont`, el usuario tendrá que especificar ese número para eliminar a la persona solicitada.

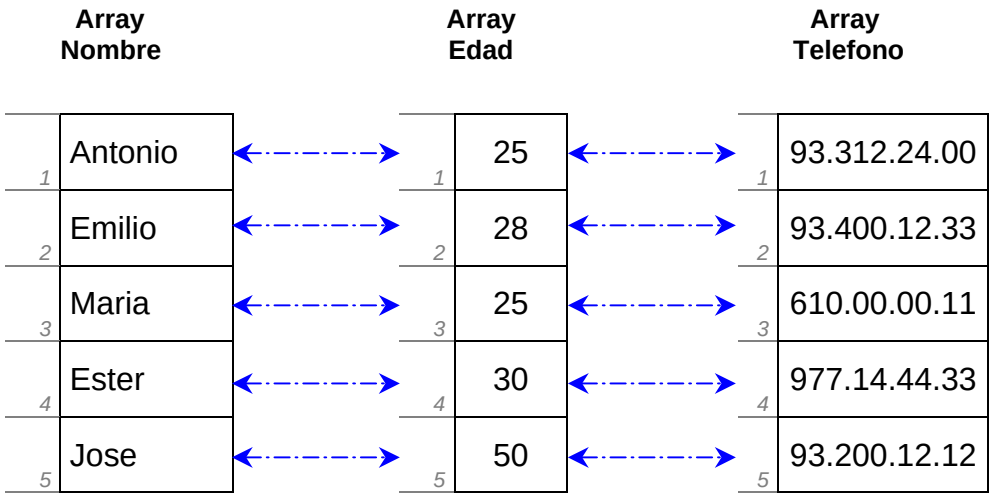
Ejemplo:

```
1 - Antonio - 25 - 93.312.24.00
2 - Emilio - 28 - 93.400.12.33
3 - Maria - 25 - 610.00.00.11
4 - Ester - 30 - 977.14.44.33
5 - Jose - 50 - 93.200.12.12
Elije el item a eliminar [1..5] 0 para salir.
```

Se ha listado el contenido de la agenda, y se muestra un mensaje al usuario, para que pulse un número del `1` al `5` (en este caso), para eliminar a la persona especificada, o `0` para salir, y cancelar la eliminación de personas.

Ahora hablemos detalladamente del concepto de 'eliminación' de personas.

Si tenemos llena la agenda, te muestro gráficamente cómo estarían los arrays de datos:

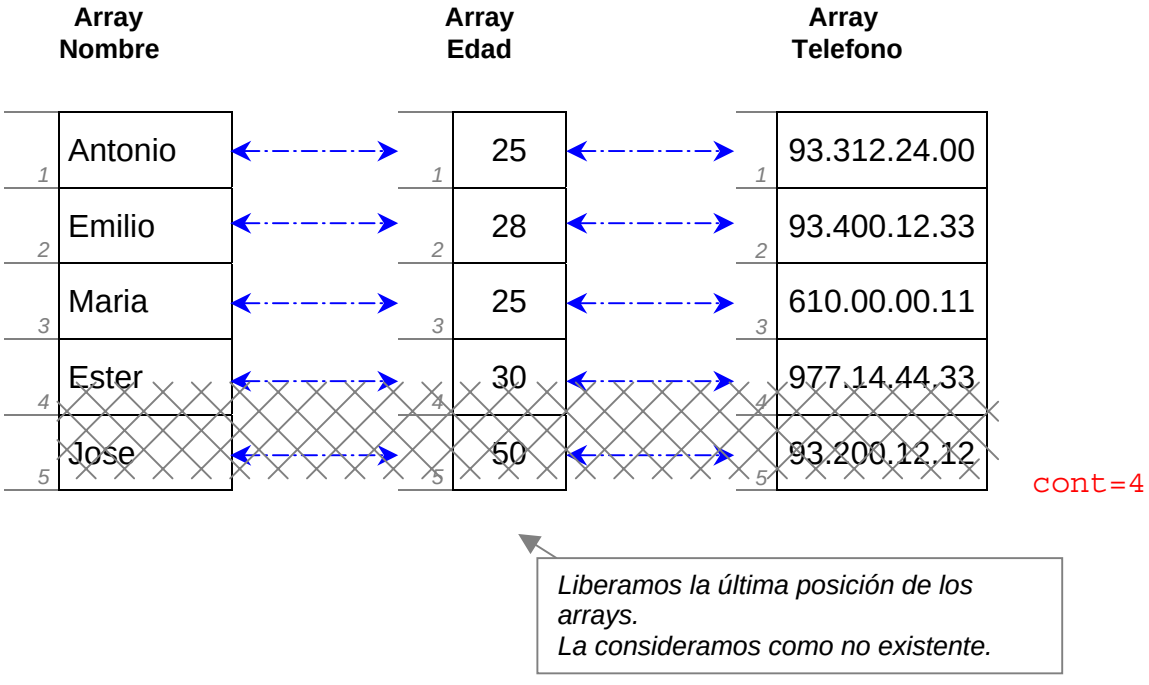


Y estarás de acuerdo en que ahora, `cont=5`.

Si por ejemplo queremos eliminar la persona `5`, el usuario pulsará `5`, y simplemente, lo que haremos será hacer `cont:=cont-1`.

Has de esforzarte en ENTENDER, que NO HACE FALTA "BORRAR" NINGÚN ARRAY NI NINGÚN CAMPO!!!

Simplemente, si ponemos `cont` a `4`, lo que hacemos, es decirle a la agenda que la próxima posición libre es la `5`.
Cuando listemos a partir de ahora, se listarán las personas de la `1` a la `4`.
Cuando insertemos una persona, el programa lo permitirá porque ya NO está llena, y... la persona que insertemos, irá a parar a la posición `5` de los arrays, "Machacando" lo que hubiera.



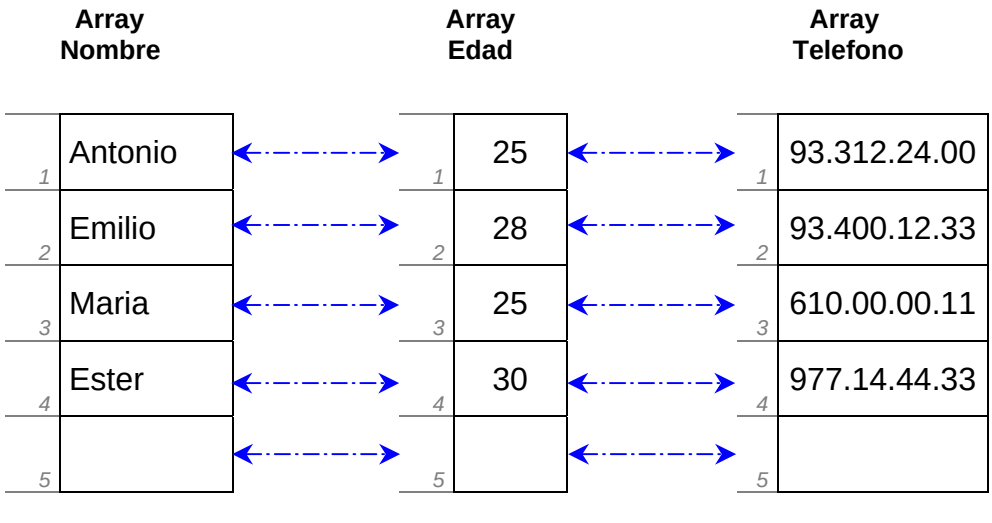
Esto quizás sea lo más difícil de entender...

Observa que si ahora sales de la opción de eliminar, pulsando el `0`, y en el menú principal elijes listar, aparece el siguiente listado:

```
1 - Antonio - 25 - 93.312.24.00
2 - Emilio - 28 - 93.400.12.33
3 - Maria - 25 - 610.00.00.11
4 - Ester - 30 - 977.14.44.33
Pulsa una tecla para continuar...
```

Y podemos decir, que en la posicion `5`, haya lo que haya, NO nos importa, ya que se considera LIBRE!

Así pues, podemos imaginar los arrays de este modo:

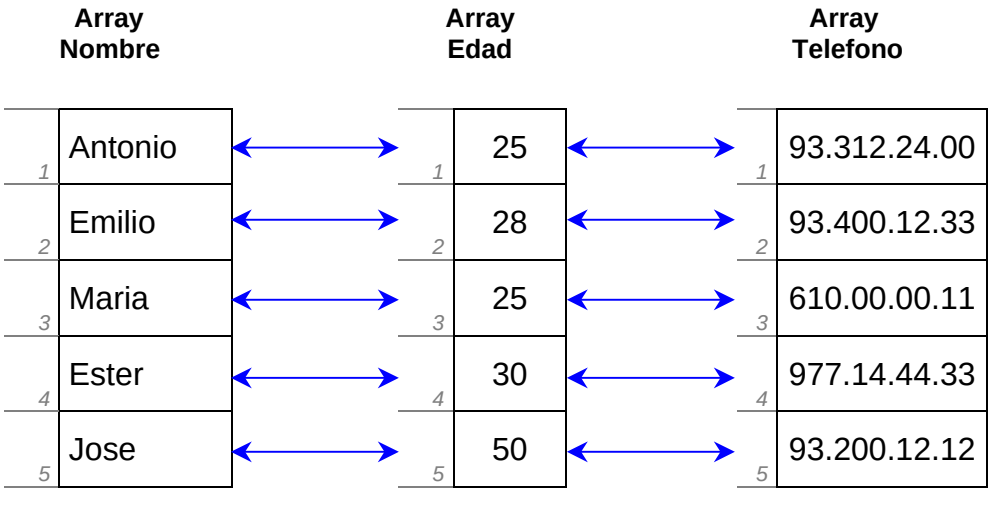


Y deducirás, que si volvemos a insertar una persona en la agenda, ocupará la posición `5`, y `cont`, pasará a valer `5`.

Ahora te explico lo que hay que hacer cuando se quiere eliminar una persona que no sea la última...

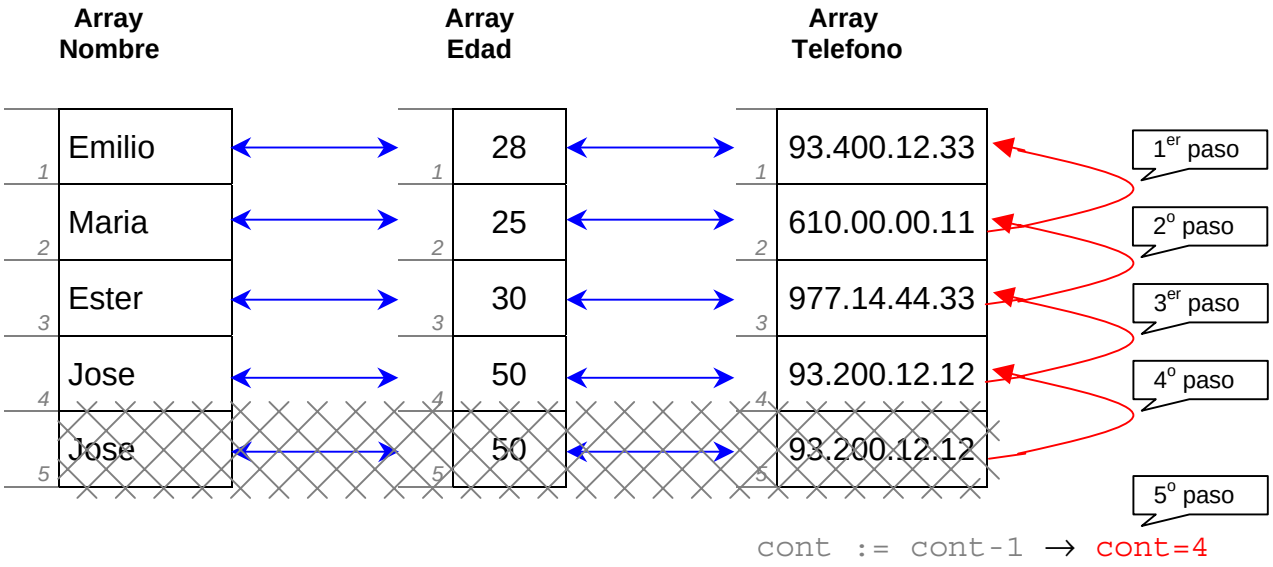
El proceso es idéntico para `cont=1` hasta `cont=4`.

Con la agenda llena (con cont=5), gráficamente, los arrays están del siguiente modo:

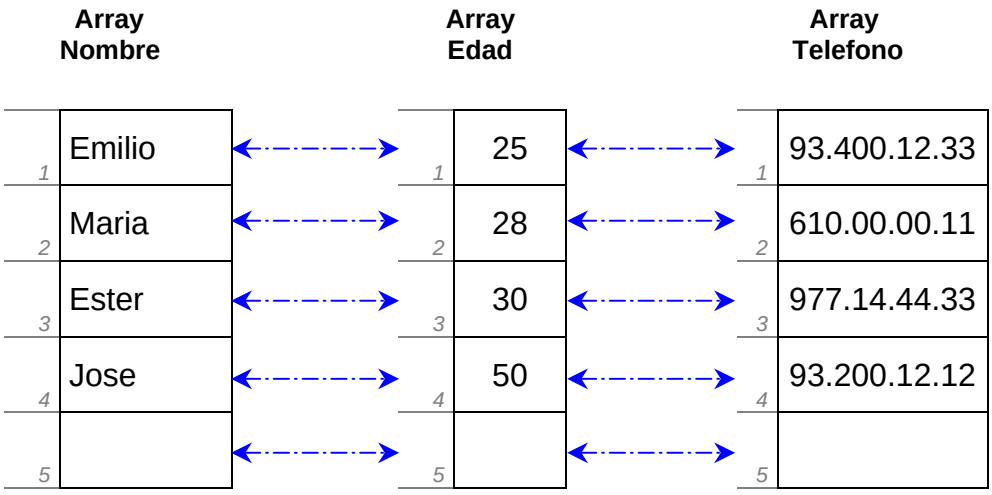


Y si queremos eliminar, por ejemplo, la persona `1` (Antonio), tendremos que copiar, los datos de Emilio a la posición `1`, "Machacando" los datos de Antonio, tendremos que copiar los datos de Maria donde están los de Emilio, los de Ester donde están los de Maria, y los de Jose donde están los Ester.
Hacemos un `cont:=cont-1`, y haya lo que haya en la posición `5`, se considerará como libre!

El siguiente esquema lo explica:



Quedando, nuestra agenda de la siguiente manera:



Resumiendo... en definitiva, lo que hacemos es desplazar todos los registros por debajo del que se va a borrar, y el último, se considera como libre, ya que sus datos habrán quedado copiados en el que era penúltimo antes de borrar, y que ahora pasa a ser el último.

Explicación

Borrar toda la agenda

Borrar toda la agenda es... realmente... LO MAS SENCILLO DE TODO!!!

Símplemente, ponermos `cont` a `0`!!!!

Previamente, nos aseguramos de que haya personas en la agenda, ya que de lo contrario, mostramos “LA AGENDA ESTÁ VACÍA!! No hay nada que eliminar!!”.

Si hay elementos en la agenda, hacemos `cont:=0`, con lo que le decimos al programa que TODOS los campos de los arrays están libres, haya lo que haya.

Haya lo que haya, la próxima persona que insertemos en la agenda, irá a parar a la posición 1 de los arrays, la siguiente a la posición 2, y así sucesivamente...

Ordenación de listas de cadenas de texto: Los arrays de string's

Supón que tienes una lista con nombres de personas y la quieres ordenar por orden alfabético.

Utilizaremos un sistema de ordenación. Podríamos utilizar cualquier método, pero utilizaremos el sistema de ordenación por burbuja para este ejemplo.

No pondremos acentos en este ejemplo porque las comparaciones no se realizan correctamente con éstos.
Se podría hacer contando con palabras acentuadas, pero se complicaría más el ejemplo, y no es el objetivo.

La lista de nombres a ordenar es la siguiente:

Angel
Maria
Jose
Aurora
Oriol
Francisco
Laura
Sergio
Beatriz
Antonio

Para declarar una variable donde almacenar esta lista de nombres, utilizaremos un array de 10 string's:

```
Const
   MAX=10;

Var
   lista: array[1..MAX] of string;
```

Y gráficamente, tendremos la siguiente lista en memoria:

Índice	*Cadena de texto (Nombre)*
1	Angel
2	Maria
3	Jose
4	Aurora
5	Oriol
6	Francisco
7	Laura
8	Sergio
9	Beatriz
10	Antonio

Observa el código y observa los comentarios insertados.

Es MUY importante poner comentarios en el programa.
Así, al cabo de un tiempo, cuando se vuelve a leer el código, se entiende más rápido lo que hace y el por qué.

```
Program OrdenacionStrings1;

uses Crt;

Const
   MAX=10;

var
   lista: array[1..MAX] of string;
   temp: string;
   i, j: integer;


Begin

   // Fijamos los nombres en la lista, con estas asignaciones:
   lista[1]:='Angel';
   lista[2]:='Maria';
   lista[3]:='Jose';
   lista[4]:='Aurora';
   lista[5]:='Oriol';
   lista[6]:='Francisco';
   lista[7]:='Laura';
   lista[8]:='Sergio';
   lista[9]:='Beatriz';
   lista[10]:='Antonio';

   // Mostramos la lista para ver que se ha ordenado.
   writeln('Lista sin ordenar:');
   for i:=1 to MAX do
      writeln(lista[i]);

   // Ordenación por burbuja de la lista:
   for i:=1 to MAX-1 do
      for j:=MAX downto i+1 do
            if lista[j]<lista[j-1] then    // Ordenamos de menor a mayor (<)
               begin
                  temp:=lista[j];
                  lista[j]:=lista[j-1];
                  lista[j-1]:=temp;
               end;


   // Mostramos la lista para ver que se ha ordenado.
   writeln;
   writeln('Lista ordenada:');
   for i:=1 to MAX do
      writeln(lista[i]);


   ReadKey;

                                       End.
```

FASE 3: Prepárate para la auténtica programación

De nuevo... si has seguido mis consejos, y has tecleado todos y cada uno de los ejercicios anteriores, habrás asimilado los conceptos básicos en programación (fundamentos de informática).

Ahora llega otro paso importante. Debes empezar a asimilar conceptos más serios en programación, y empezar a “volar por tí mismo”...

Esta última parte trata los últimos temas de Fundamentos de Informática... justo con los que se empieza el primer día de la asign atura de Programación del segundo curso.

Y... que sepas que el primer día de clase de Programación de 2º, se explica rápidamente TODO lo que te explico aquí, y el segundo día ya se empieza con temas nuevos.

Debes estar preparado!!

Programa Principal, Subprogramas, Funciones, Acciones y Procedimientos

Imagínate que tienes que construir un edificio. Vamos a compararlo con la construcción de un programa.

Olvídate de comprar un edificio pre-fabricado... eso sería como comprar un programa YA hecho.

Se trata de que tú mismo diseñes y construyas el edificio a tu gusto y necesidad, y que emplees el menor tiempo, y una buena relación calidad-coste.
Es decir... se trata de que TÚ diseñes y programes dicho programa.

Para construir un edificio, primero se hacen los bocetos, el diseño y el plano de la estructura del futuro edificio (con papel y lápiz, haces el boceto y el algoritmo del futuro programa).

Recuerda que NO siempre salen las cosas a la primera, ya que para eso se necesita muchísima experiencia, así que no te extrañes si tienes que rediseñar y modificar en ocasiones el algoritmo que inicialmente diseñaste!!

Si no... mira MicroSoft... llevan desde 1989 modificando año tras año su ""maravilloso"" sistema operativo, y con el que cada año dicen que es lo mejor que se ha hecho. Aunque ahí hay intereses económicos y financieros, de los que no voy a hablaros porque ya os habrán hablado del tema.

En fin... una vez diseñado el plano del edificio (es decir, el algoritmo del programa), tenemos que ponermos manos a la "obra".

Buscamos un solar con las dimensiones, terreno, fácil comunicación y licencia necesarios para construir nuestro edificio (es decir, buscamos un ordenador, un sistema operativo, un lenguaje de programación y un compilador compatibles con nuestro programa).

Primero construimos los cimientos del edificio: declaración de variables y tipos de variables, declaración de funciones a utilizar y declaración de la función principal del programa.

Y luego, ladrillo a ladrillo, vamos construyendo las paredes, gracias al cemento, estructuras metálicas de apoyo y demás materiales.

Y luego, instrucción a instrucción, vamos construyendo el programa principal, gracias a funciones propias del compilador, y funciones creadas por nosotros, y demás estructuras algorítimicas.

Pero recuerda: Siempre siguiendo las normativas de construcción de edificios, y también simpre siguiendo el diseño y plano del futuro edificio.

Pero recuerda: Siempre siguiendo las reglas del lenguaje de programación, y también simpre siguiendo el boceto y el algoritmo del futuro programa.

Conclusión...
Divide y vencerás!!!

Uso de `Function`: Tu primera función

Vamos a hacer un sencillo programa para calcular la media de 2 notas que nos entrarán previamente por teclado.

No tiene mucho sentido utilizar una función para hacer este programa tan pequeño, pero el objetivo es estudiar la creación de funciones.

Mas adelante ya complicaremos las cosas!

Pica el siguiente código y comprueba que funciona:

```
Program Funciones1;

uses Crt;

// Función CalculaMedia -----------------------------------------------------
// Calcula y devuelve la media de dos notas que le pasemos.

Function CalculaMedia(nota1: real; nota2: real): real;
Var
   suma: real;
   media: real;

begin
   suma:=nota1+nota2;

   media:=suma/2;

   CalculaMedia:=media;
end;
// --------------------------------------------------------------------------


// Programa Principal -------------------------------------------------------

var
   nota1, nota2: real;
   media: real;

Begin

   writeln('Nota 1? ');
   readln(nota1);
   writeln('Nota 2? ');
   readln(nota2);

   media:=CalculaMedia(nota1, nota2);

   writeln;
   writeln('Media de notas = ', media);

   ReadKey;

End.
// --------------------------------------------------------------------------
```

Insisto en que es sumamente importante que piques todos lo programas, y los estudies para comprender su funcionamiento.

De veras que es la única manera para aprender a programar bien.

En la siguiente página analizaremos el programa.

Uso de `Procedure`: Tu primer procedimiento

¿Recuerdas el programa que hicimos en la página 33? El que muestra la tabla de multiplicar del 5.

Pues a partir de ese código, vamos a hacer un programa que muestre la tabla de multiplicar de cualquier número.

Para ello crearemos un procedimiento que mostrará por pantalla la tabla de multiplicar que le pasemos en su llamada.
Dicho procedimiento, contendrá código muy similar al del programa de la página 33.

Pica el siguiente código y comprueba que funciona:

```
Program Procedimientos1;

uses Crt;


// Procedimiento Muestra Tabla ------------------------------------------------
// Muestra la tabla de multiplicar del número que le pasemos.

Procedure MuestraTabla(numero: integer);

Const
   MAX=10;

Var
   n: integer;

begin

   writeln('Tabla de multiplicar del ', numero, ':');

   for n:=0 to MAX do
      writeln(numero, ' x ', n, ' = ', numero*n);

end;
// ---------------------------------------------------------------------------


// Programa Principal ---------------------------------------------------------

Var
   num: integer;

Begin

   writeln('Introduce un numero para mostrar su tabla de multiplicar: ');
   readln(num);

   writeln;
   writeln;

   MuestraTabla(num);

   ReadKey;

End.
// ---------------------------------------------------------------------------
```

Paso por valor y por referencia

Para que en una función, un parámetro no sólo sea de entrada, si no que sea de entrada/salida, se declarará anteponiéndole la cláusula `var`. Así se denomina el paso conocido como “paso por referencia”.

Vamos a hacer una función que calcule el cuadrado de un número entero.

Dicho programa recibirá como parámetro de entrada dicho número, pero al calcular su cuadrado, el resultado irá a parar a la misma variable que contenga el número.

Las variables pasadas por referencia son parámetros de entrada/salida.

Pica el siguiente código y comprueba que funciona:

```
Program PasoPorReferencia1;

uses Crt;


// Procedimiento CalculaCuadrado ----------------------------------------
// Eleva al cuadrdado la variable que le pasen.
Procedure CalculaCuadrado(var numero: integer);
begin

   numero:=numero*numero;

end;
// ------------------------------------------------------------------------


// Programa Principal ---------------------------------------------------
Var
   num: integer;

Begin

   writeln('Introduce un numero para elevarlo al cuadrado: ');
   readln(num);

   writeln;
   writeln;

   CalculaCuadrado(num);

   writeln('Resultado = ', num);

   ReadKey;

End.
// ------------------------------------------------------------------------
```

Tipos de datos personalizados (`type`)

Un tipo de dato personalizado (usando `type`*), consiste en la definición de un tipo de dato, con un nombre que el programador decide, para dejar más claro el uso de esa variable, y poder definir más datos a partir de la definición de ese tipo.*

Los tipos se declaran inmediatamente después de la inclusión de librerías con el `Uses` *(mira los programas que iremos haciendo a partir de ahora).*

Veamos unos ejemplos:

Supongamos que queremos definir 3 nuevos tipos de datos, para dejar más claro el dato que irá dentro de cada variable:

```
type
  TNombre = string;
  TEdad = integer;
  TTelefono = string;
```

Y a partir de ahora, en las funciones, procedimientos, o en el programa principal, podemos definir datos de dichos tipos:

```
const
   MAX=5;

var
  nombres: Array[1..MAX] of TNombre;
  edades: Array[1..MAX] of TEdad;
  telefonos: Array[1..MAX] of TTelefono;
```

A los tipos, para programar "con estilo", se les pone una "T" mayúscula delante, para acordarnos siempre de que eso es un TIPO de dato, y NO un dato.

Quizás te haya quedado alguna duda con respecto a lo que te he explicado, pero si vas escribiendo los programas que haremos a partir de ahora, usando el `type`, lo llegarás a entender.

Tuplas (record)

La tupla (`record` en Pascal), es un tipo de dato que nos permite agrupar varios datos de diferente tipo en un mismo conjunto, dándole un nombre a dicho grupo.

¡¡ Página **MUY** importante !!

Un ejemplo claro es el de la agenda de personas.
En lugar de tener 3 arrays independientes entre sí, uno para cada dato de la persona, vamos a crear un tipo de "ficha" (es decir, un tipo de tupla) que agrupe los 3 datos de una persona:

```
type
  TPersona = record
      Nombre: string;
      Edad: integer;
      Telefono: string;
  End;
```

Y luego, vamos a crear un array de 5 fichas como esta, es decir, un array de 5 tuplas del tipo `TPersona`, y junto con la variable `Cont` de tipo `integer`, la pondremos en otro tipo tupla que agrupe toda la agenda:

```
const
   MAX=5;

type
  TAgenda = record
      Personas: array [1..MAX] of TPersona;
      Cont: integer;
  End;
```

Así pues, si ahora nos declaramos una variable de tipo `TAgenda`:

```
var
  agenda: TAgenda;
```

Ya podemos acceder a todos los datos de nuestra agenda.
Si por ejemplo queremos inicializar el contador de personas a 0, al iniciar el programa, haremos:

```
agenda.Cont:=0;
```

Si queremos leer por teclado el nombre de la persona 3, haremos:

```
readln(agenda.Personas[3].Nombre);
```

Y si queremos imprimir por pantalla la edad de la persona 2, haremos:

```
writeln(agenda.Personas[2].Edad);
```

Agenda 4: El uso de las tuplas

Os muestro el listado de la agenda hecha con tuplas: *Agenda4*.
Tiene exactamente la misma funcionalidad que *Agenda3*, pero utilizamos las tuplas, que en el fondo estructura mejor el programa, aunque aún no lo veas claro.

```
Program Agenda4;

Uses
   Crt;

Const
   MAX=5;

type
  TPersona = record
      Nombre: string;
      Edad: integer;
      Telefono: string;
  End;

  TAgenda = record
      Personas: array [1..MAX] of TPersona;
      Cont: integer;
  End;


Var
   agenda: TAgenda;
   tecla: char;
   z: integer;

Begin

   agenda.Cont:=0;

   Repeat

      clrscr;
      gotoxy(30,10);  writeln('AGENDA - MENU');
      gotoxy(30,12);  writeln('1. Agregar persona.');
      gotoxy(30,13);  writeln('2. Listar agenda.');
      gotoxy(30,14);  writeln('3. Eliminar persona');
      gotoxy(30,16);  writeln('4. Borrar toda la agenda');
      gotoxy(30,18);  writeln('0. Salir.');

      tecla:=ReadKey;

      Case tecla Of
         '1': Begin

                  if agenda.Cont<5 then
                     begin
                        gotoxy(30,20); write('Nombre?');
                        gotoxy(30,21); write('Edad?');
                        gotoxy(30,22); write('Telefono?');

                        gotoxy(39,20); readln(agenda.Personas[agenda.Cont+1].Nombre);
                        gotoxy(37,21); readln(agenda.Personas[agenda.Cont+1].Edad);
                        gotoxy(41,22); readln(agenda.Personas[agenda.Cont+1].Telefono);

                        agenda.Cont:=agenda.Cont+1;
                     end
                  else
                     begin
                        gotoxy(30,20); write('La agenda esta llena!!!');
                        gotoxy(30,25); write('Pulsa una tecla para continuar...');
                        ReadKey;
                     end;
               end;
```

```
         '2': Begin
                 clrscr;

                 for z:=1 to agenda.Cont do
                    writeln(z, ' - ', agenda.Personas[z].Nombre,
                               ' - ',agenda.Personas[z].Edad,
                               ' - ', agenda.Personas[z].Telefono);

                 writeln('Pulsa una tecla para continuar...');
                 ReadKey;
              End;
         '3': Begin
                 clrscr;

                 if(agenda.Cont<>0) then
                 Begin
                    Repeat
                       for z:=1 to agenda.Cont do
                          writeln(z, ' - ', agenda.Personas[z].Nombre,
                                     ' - ',agenda.Personas[z].Edad,
                                     ' - ', agenda.Personas[z].Telefono);

                       writeln('Elije el item a eliminar [1..', agenda.Cont, '] 0 para salir.');
                       tecla:=Readkey;

                       if (tecla>='1') and (tecla<='4') then
                       Begin
                          for z:=ord(tecla)-48 to agenda.Cont do   // ord convierte a decimal un char
                          Begin
                             agenda.Personas[z].Nombre:=agenda.Personas[z+1].Nombre;
                             agenda.Personas[z].Edad:=agenda.Personas[z+1].Edad;
                             agenda.Personas[z].Telefono:=agenda.Personas[z+1].Telefono;
                          End;
                       End;

                       if (tecla>='1') and (tecla<='5') then agenda.Cont:=agenda.Cont-1;

                       clrscr;

                    Until(tecla='0');

                    tecla:='3';    // Restauramos el valor de tecla, para que no salga del programa.

                 End
                 else
                 Begin
                    writeln('LA AGENDA ESTA VACIA!! No hay nada que eliminar!!');
                    writeln('Pulsa una tecla para continuar...');
                    ReadKey;
                 End;
              End;
         '4': Begin
                 if(agenda.Cont<>0) then
                 Begin
                    agenda.Cont:=0;
                    clrscr;
                    writeln('AGENDA BORRADA TOTALMENTE!');
                    writeln('Pulsa una tecla para continuar...');
                    ReadKey;
                 End
                 else
                 Begin
                    clrscr;
                    writeln('LA AGENDA ESTA VACIA!! No hay nada que eliminar!!');
                    writeln('Pulsa una tecla para continuar...');
                    ReadKey;
                 End;
              End;
      End;

   Until tecla='0';

End.
```

Explicación

Empecemos por las declaraciones.
¡¡¡ Olvida los 3 arrays independientes !!!

La declaración de tipos personalizados es la siguiente:

```
const
   MAX=5;

type
  TPersona = record
      Nombre: string;
      Edad: integer;
      Telefono: string;
  End;

  TAgenda = record
      Personas: array [1..MAX] of TPersona;
      Cont: integer;
  End;
```

Nos construimos una plantilla (tipo de dato personalizado).
Es una plantilla de la ficha que se usará para almacenar toda la información de cada persona.

Plantilla que agrupa un array con 5 fichas de tipo `TPersona` *(utilizamos la 'plantilla' declarada anteriormente, para crear el* `array`*). Incluimos también el contador que nos dirá cuántas personas hay almacenadas en la agenda.*

Las dos declaraciones anteriores <u>son tipos</u>. Son tipos de datos personalizados del programador (o sea, tú!).
<u>En los tipos de datos NO se puede escribir información.</u> Del mismo modo que no puedes declarar una variable que se llame '`integer`', tampoco podrás ahora declarar una variable que se llame '`TPersona`' ó '`TAgenda`'.

En el apartado de la declaración de variables (`Var`), nos declaramos ya nuestra "Base de Datos", es decir, nuestra `agenda`, donde SÍ podremos escribir durante el programa:

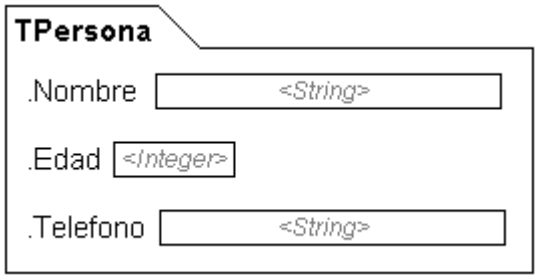
```
Var
   agenda: TAgenda;
   tecla: char;
   z: integer;
```

la variable agenda, es del tipo `TAgenda`.
Ahora tenemos los siguientes campos disponibles:

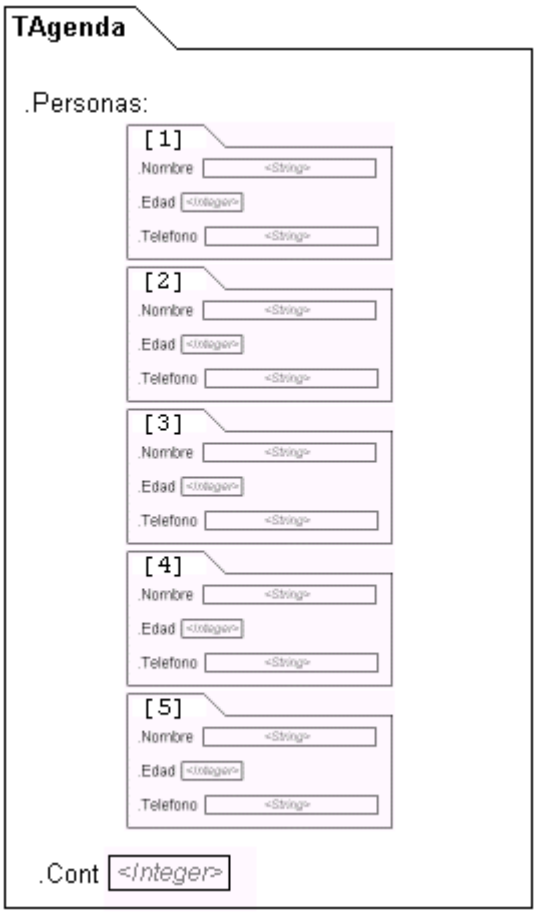
```
agenda.Cont
agenda.Personas[?].Nombre
agenda.Personas[?].Edad
agenda.Personas[?].Telefono
```

Donde el Interrogante (?) puede ir de 1 a MAX.

Gráficamente, la plantilla de una persona (tupla de tipo `TPersona`) es:



Y el array de fichas que incluye `cont` para indicar cuales son válidas (tupla de tipo `TAgenda`), es:



`agenda`, es la variable de tipo `TAgenda`, que contendrá todos los datos de todas las personas de la agenda. Es decir, será la base de datos (BDD).

Última fase: PREPÁRATE PARA EL EXÁMEN

Si has seguido todos mis consejos al pié de la letra, es decir, si...

- Has asistido a todas las clases de Fundamentos de Informática.
- Has hecho todas las prácticas de la asignatura con una actitud participativa.
- Has resuelto tus dudas acudiendo a profesores y a tus propios compañeros.
- Has leído atentamente este libro de arriba a abajo.
- Has picado todos y cada uno de los programas del libro.

Te informo de que ya estás preparado para:

- Enfrentarte con posibilidades al examen de la asignatura de Fundamentos de Informática.
- Empezar con buen pié la asignatura de Programación del 2º curso.
- Aprender a programar en otros lenguajes: C/C++, Visual Basic, PHP, etc...

¡Felicidades!

En este último apartado te explicaré la solución al exámen de Fundamentos de Informática realizado en la primera convocatoria del curso 2003-04.

Como consejos para enfrentarte al examen:

- Descansa bien la noche anterior. Es muy importante.
- Ves relajado y centrado al exámen.
- Sigue todas las normas y todos los consejos! No copies... Lleva tus propios apuntes, etc.. etc.. etc...

Y...

¡SUERTE!

	Professor: C. Latorre / J.C. Fernández / J. Yebras Data: 4 / 2 / 2004 Codi examen: CLC-6841-1-04-P Assignatura: Fonaments d'Informàtica

Duració de l'examen: 3 hores
Dies de revisió: 20 de febrer
És permès l'ús de material de consulta:
qualsevol, sense compartir amb els companys.
Notes addicionals a l'examen:
Fer cada problema en un full separat
Important:
El fet de rebre l'examen consumeix automàticament convocatòria.
No és permès sortir de l'aula d'examen fins transcorreguts 20 minuts del seu inici.

1.- Sèrie Fibonnaci

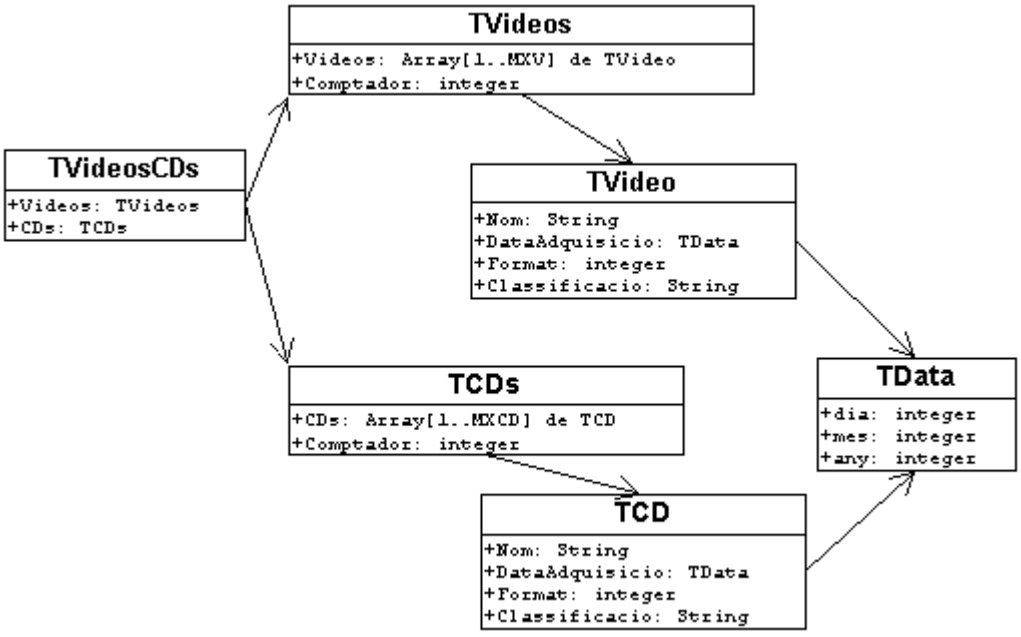
(2.5 punts) Es demana fer un procediment al que se li passen com a paràmetres: valor inicial mínim a llistar i el valor final màxim a llistar de la sèrie Fibonacci vista a teoria i mostra per pantalla els valors corresponents de la sèrie que estan entre els dos valors passats.
La capçalera d'aquesta funció serà la següent:

```
Procedure Fibo ( min,max: integer)
```

Fer també un procediment que demani el valors mínim i màxim a l'usuari i després cridi a la funció Fibonacci creada.
Nota: La sèrie Fibonacci comença amb els números 0, 1 i segueix formant el següent amb la suma dels dos anteriors. Segons això, la crida Fibo(6,20) mostrarà 8 –13 per pantalla.

2.- Definició tipus de dades

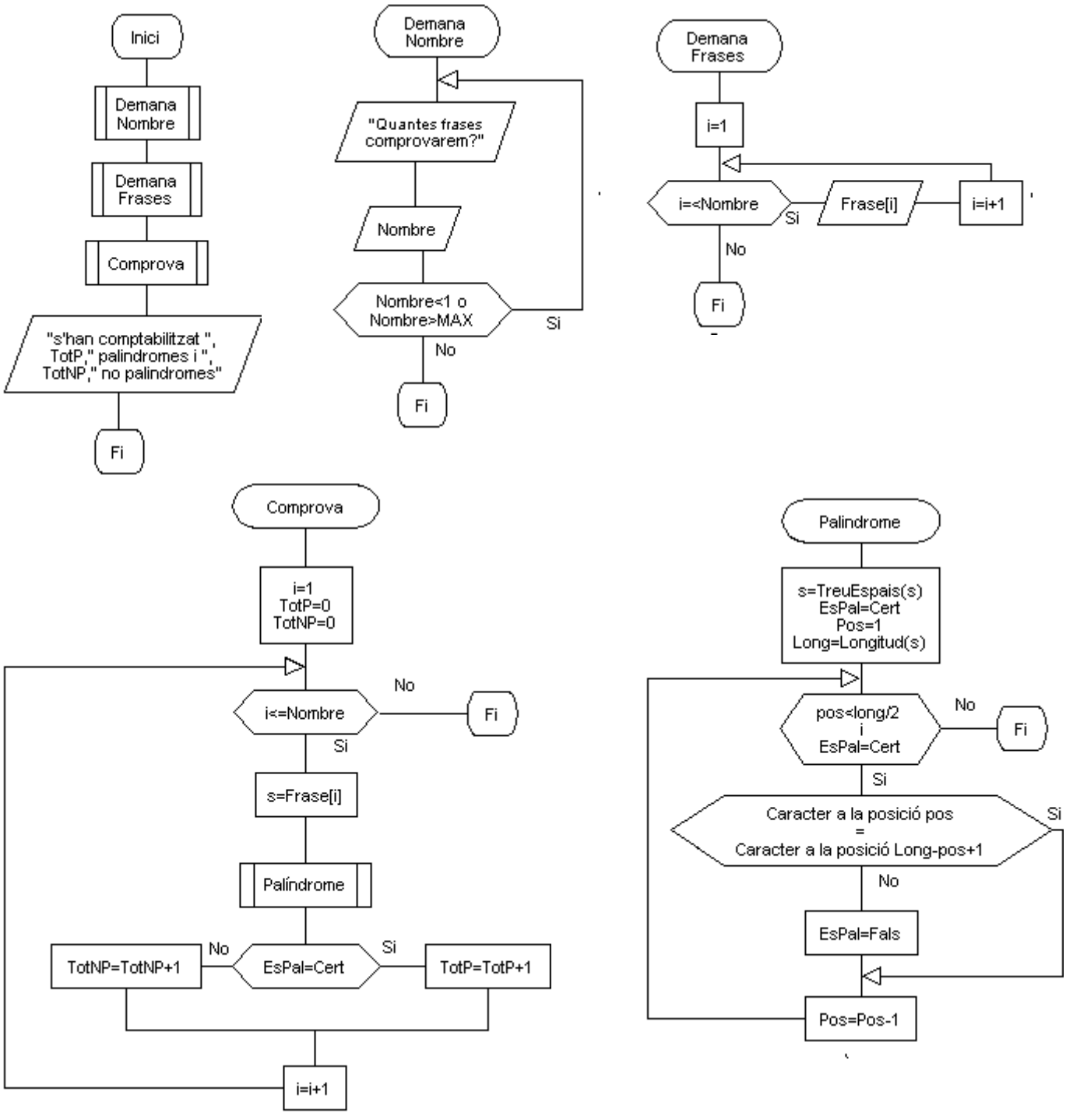
(2 punts) Declarar els tipus de dades que es presenten en el següent esquema. Serviran per emmagatzemar els videos i cds de que disposi l'usuari. Definir tots els tipus de dades que es representen en el gràfic amb els camps identificats a cada tupla.



3.- Frases palíndromes.

(2.5 punts). Es diu que una frase és palíndrome quan es llegeix igual del dret que del revés. Per exemple "Dábale arroz a la zorra el abad" , "Sé verlas al revés" o "Átale, demoniaco Cain, o me delata! " són palíndromes del castellà, mentre que "Se li veu que vil és" o "Tip el pastor ara farà rots a ple pit" ho són en català.

Tradueix el següent diagrama de flux al programa corresponent.



Nota: El diagrama està dissenyat per a ser traduït com a programa principal sense funcions. Si es tradueix algun dels subalgorismes com a funció o procediment, haureu de explicitar els paràmetres i el tipus de dades retornat d'una manera coherent.

Suposem que hi ha una funció que s'anomena TreuEspais que elimina els caràcters de puntuació que no són lletres i elimina els accents per a poder comparar les lletres, tal com s'indica a l'algorisme.

4. Fusió ordenada d'arrays ordenats.

(3 punts) Fer una funció a la que li passarem 2 arrays de strings, que suposarem ordenats, i ens generi un array de strings amb la fusió dels valors d'ambdós arrays es mantenint els valors ordenats i fent un únic recorregut dels valors de cada array que ens passen. Suposem que no ens trobarem valors repetits ni dins d'un mateix array dels 2 originals que ens passen ni entre ells.

Exemple d'entrada per Max=5 :

Anton
Emili
Joan
Josep
Lluís

i

Ester
Gemma
Maria
Paula
Sílvia

La sortida seria:

Anton
Emili
Ester
Gemma
Joan
Josep
Lluís
Maria
Paula
Sílvia

La funció demanda tindrà la següent capçalera:

```
Type
     TLlista= Array[1..max] of string;
     TFusio = Array[1..max*2] of string;

Function Fusio ( Llista1, Llista2: TLlista): TFusio;
```

RESOLUCIÓN

Program Fibonnaci;

```
uses Crt;


Procedure Fibo (min, max: integer);
var
   n1, n2: integer;
   c, r: integer;

Begin

   if (min<=0) and (max>=0) then writeln(0);
   if (min<=1) and (max>=1) then writeln(1);

   n1:=1;
   n2:=0;

   if max>0 then
      while (n1+n2)<max do
         begin
            r:=n1+n2;
            if r>=min then write(r);
            n2:=n1;
            n1:=r;
            if (r>min) and ((n1+n2)<max) then write('-');
         end;
End;


Begin

   Fibo(6, 20);

   ReadKey;
```

End.

RESOLUCIÓN

```
const
   MXV=100;
   MXCD=100;


type
   TData = record
      dia: integer;
      mes: integer;
      any: integer;
   End;


   TVidieo = record
      Nom: string;
      DataAdquisicio: TData;
      Format: ingeger;
      Classificacio: string;
   End;


   TVideos = record
      Videos: array[1..MXV] of TVideo;
      Comptador: integer;
   End;


   TCD = record
      Nom: string;
      DataAdquisicio: TData;
      Format: ingeger;
      Classificacio: string;
   End;


   TCDs = record
      CDs: array[1..MXCD] of TCD;
      Comptador: integer;
   End;


   TVideosCDs = record
      Videos: TVideos;
      CDs: TCDs;
   End;
```

RESOLUCIÓN

```
Program FrasesPalindromes;

uses Crt;

Const
   MAX=5;

var
   TotP, TotNP: integer;
   Long: integer;
   s: string;
   Nombre: integer;
   i: integer;
   Frase: Array[1..MAX] of string;
   EsPal: Boolean;
   Pos: integer;

Begin

   repeat
      write('Quantes frases comprovarem?');
      readln(Nombre);
   until not( (Nombre<1) or (Nombre>MAX) );

   for i:=1 to Nombre do
      readln(Frase[i]);

   for i:=1 to Nombre do
   begin
      s:=Frase[i];

      // s:=TreuEspais(s);

      EsPal:=TRUE;
      Pos:=1;
      Long:=length(s);

      while (pos<(Long div 2)) and (EsPal=True) do
      begin
         if not(s[pos]=s[Long-pos+1]) then EsPal:=False;
         pos:=pos-1;
      end;

      if EsPal=True then TotP:=TotP+1 else TotNP:=TotNP+1;
   end;


   writeln('S''han comptabilitzat ', TotP, ' palindromes i ', TotNP, ' no
palindromes');

   readkey;

End.
```

RESOLUCIÓN

```
Program FusioOrdenada;
uses Crt;

Const
   MAX=5;

Type
   TLlista=Array[1..MAX] of string;
   TFusio=Array[1..MAX*2] of string;


// ------ Función Fusio ---------------------------------------------------
// -- Entrada: 2 Listas de cadenas a fusionar ordenadamente.
// -- Salida: 1 Lista de cadenas.
// --------------------------------------------------------------------------
Function Fusio(Llista1, Llista2: TLlista): TFusio;
Var
   FusioTemp: TFusio;
   z1, z2: integer;
   f: integer;

Begin

   z1:=1;
   z2:=1;
   f:=1;

   repeat
      if (z1>MAX) or (z2>MAX) then
         if z1>MAX then
            begin
               FusioTemp[f]:=Llista2[z2];
               z2:=z2+1;
            end
         else
            begin
               FusioTemp[f]:=Llista1[z1];
               z1:=z1+1;
            end
      else
         if Llista1[z1]<Llista2[z2] then
            begin
               FusioTemp[f]:=Llista1[z1];
               z1:=z1+1;
            end
         else
            if Llista1[z1]>Llista2[z2] then
               begin
                  FusioTemp[f]:=Llista2[z2];
                  z2:=z2+1;
               end;

      f:=f+1;

   until f>MAX*2;

   Fusio:=FusioTemp;
End;
```

```
// ------------------------------------------------------------------------
// ------ Programa Principal ----------------------------------------------
// Probamos la función 'Fusio'.
// ------------------------------------------------------------------------
Var
   Llista1: TLlista;
   Llista2: TLlista;
   FusioFinal: TFusio;
   f: integer;

Begin

   Llista1[1]:='Anton';
   Llista1[2]:='Emili';
   Llista1[3]:='Joan';
   Llista1[4]:='Josep';
   Llista1[5]:='Lluis';

   Llista2[1]:='Ester';
   Llista2[2]:='Gemma';
   Llista2[3]:='Maria';
   Llista2[4]:='Paula';
   Llista2[5]:='Silvia';

   FusioFinal:=Fusio(Llista1, Llista2);

   for f:=1 to MAX*2 do
      writeln(FusioFinal[f]);

   ReadKey;

End.
// ------------------------------------------------------------------------
```

Apéndice A : Tipos básicos de datos

TIPOS NUMERICOS BASICOS:

BYTE	Numero entero entre 0 y 255. No apto para operaciones, salvo a nivel de bits. (8 bits sin signo)
INTEGER	Entero, entre -32768 y 32767. (16 bits con signo)
LONGINT	Entero largo, entre -2.147.483.648 y 2.147.483.647. (32 bits con signo)
SHORTINT	Entero corto, entre -128 y 127. (8 bits con signo)
WORD	Entero, entre 0 y 65.535. (16 bits sin signo)
REAL	Real, en coma fija, entre 2.910-39 y 1.71038. Precision de 11/12 digitos. (6 bytes).
SINGLE	Real entre 1.510-45 y 3.41038. Precision de 7/8 cifras. (4 bytes).
DOUBLE	Real entre 5.010-324 y 1.710308. Precision de 15/16 digitos. (8 bytes).
EXTENDED	Real entre 1.910-4931 y 1.1104932. Precision de 19/20 digitos. (10 bytes).

OTROS TIPOS:

CHAR	Caracter entre 0 y 255. (8 bits, sin signo).
BOOLEAN	Valor logico TRUE o FALSE. (1 Byte). False < True
STRING	Cadena de caracteres.

SUBRANGOS:

```
VAR
Variable : Min..Max;
Enteros : -45..120;
Caracter : 'a'..'z';
```

CARACTERES:

El dato de tipo CHAR puede almacenar cualquier caracter de la tablas ASCII extendida.

#Codigo	Equivale al caracter de codigo indicado.
^Letra	Equivale a un caracter no visualizable o codigo de control.

DATOS ENUMERADOS:

```
TYPE
Enumerada : (Dato1, Dato2, Dato3, ... , DatoN);
```

SUBRANGOS DE VARIABLES ENUMERADAS:

```
Subrango : Dato3..Dato22
```

Operaciones validas con datos enumerados:

- Comparaciones entre valores (Ej: Dato3 > Dato7)
- Asignaciones (Ej: Enumerada := ValorEnumerado)
- Pueden emplearse como indice de bucles For.
- No pueden incrementarse, sumando pero si con SUCC.

Restricciones:

- No pueden leerse o escribirse.
- Un valor enumerado solo existe para un tipo de enumeración y no puede formar parte de otro tipo enumerado.

Forma de leer o escribir:

```
CASE Enumerada OF
Dato1: WRITE('Dato1');
Dato2: ..........
...
DatoN: .........
ELSE
...........
END;
```

Apéndice B: Librerías de funciones

FUNCIONES ARITMETICAS.

`ABS (Num)`	Devuelve el valor absoluto de un entero o real. (Valor sin signo).
`FRAC (Real)`	Devuelve la parte decimal del numero real.
`INT (Real)`	Devuelve la parte entera de un numero real.
`PI`	Valor real de PI: 3.14159265358979323
`ROUND (Num)`	Devuelve el entero mas próximo a Num. Argumento real o integer.
`SQR (Num)`	Devuelve el cuadrado de real o entero.
`SQRT (Real)`	Devuelve la raíz cuadrada de real o entero.
`TRUNC (Real)`	Devuelve la parte entera del numero real.
`INC (Var, N)`	Incrementa Var en 1 o N (opcional) unidades.
`DEC (Var, N)`	Decrementa Var en 1 o N (opcional) unidades.

FUNCIONES TRIGONOMETRICAS

`ARCTAN (Radianes)`	Devuelve la arcotangente del argumento.
`COS (Radianes)`	Devuelve el coseno del ángulo indicado.
`LN (Num)`	Logaritmo natural. Argumento real o integer.
`SIN (Radianes)`	Seno del ángulo.

FUNCIONES ORDINALES.

`ORD (ordinal)`	Devuelve la posición del valor dentro del tipo de variable.
`CHR (Código)`	Devuelve el carácter ASCII del código indicado.
`PRED (Ordinal)`	Devuelve el valor anterior al indicado.
`SUCC (Ordinal)`	Devuelve el valor siguiente al indicado.

FUNCIONES VARIAS.

`CHR (Código)`	Devuelve el carácter con código indicado.
`STR (Valor, Cadena)`	Convierte un valor numérico en una cadena.
`VAL (Cad, Var, Cod)`	Convierte una cadena en un número. Da un codigo=0 si funciona o la posición del carácter erróneo.
`UPCASE (Carácter)`	Devuelve el carácter en mayúscula.
`ODD (Entero)`	True si el entero es impar y false si es par o es cero (0).
`RANDOMIZE;`	Establece el comienzo para random.
`RANDOM (Entero)`	Devuelve un numero aleatorio tal que: 0 <= Numero < 1 o 0 <= NúmEntero <= N-1
`DELAY (Miliseg)`	Produce una pausa de los milisegundos indicados.
`HI (Entero)`	Devuelve el byte de mayor peso del entero.
`LO (Entero)`	Devuelve el byte de menor peso del entero.
`SWAP (Entero)`	Devuelve el entero con los bytes cambiados.

FUNCIONES Y RUTINAS DE PANTALLA.

Estas funciones precisan la unidad (Librería) llamada CRT:

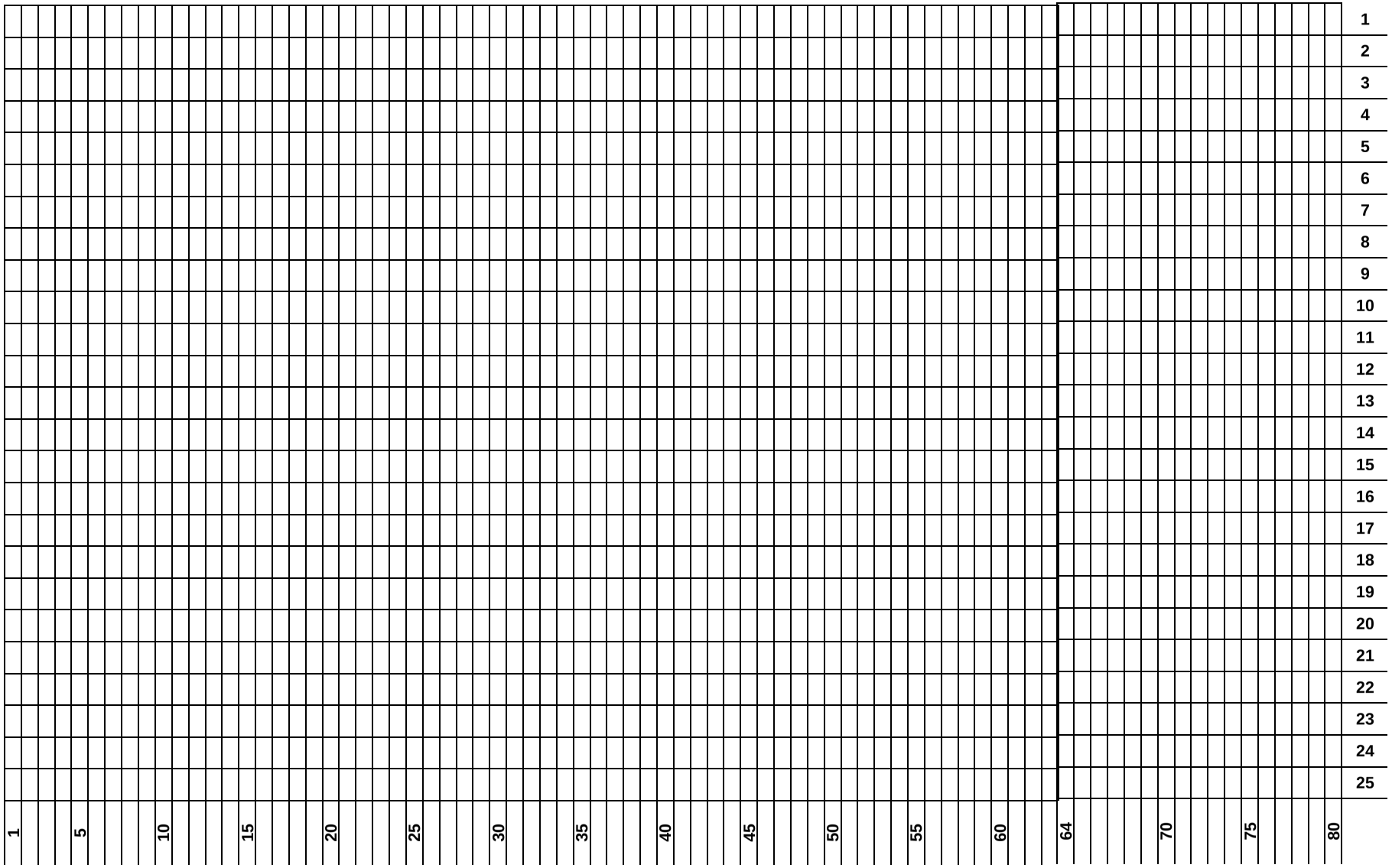
`CLRSCR;`	Borra la pantalla
`CLREOL;`	Borra hasta el final de línea
`DELLINE;`	Elimina la línea actual.
`INSLINE;`	Inserta una línea, debajo de la actual.
`GOTOXY (Col, Fil);`	Sitúa el cursor de texto en la pantalla. (1,1)
`LOWVIDEO;`	Baja la intensidad de los caracteres.
`HIGHVIDEO;`	Sube la intensidad de los caracteres.
`NORMVIDEO;`	Devuelve los atributos al texto.
`TEXTCOLOR (Color);`	Establece el color de los caracteres
`TEXTBACKGROUND (Color);`	Establece el color del fondo, para el texto.
`WHEREX`	Devuelve la posición X del cursor.
`WHEREY`	Devuelve la posición Y del cursor.

FUNCIONES DE FECHA Y HORA.

Estas funciones precisan la unidad (Librería) llamada DOS:

`GETTIME (hh, mm, ss, cc);`	Asigna a las variables la hora del sistema.
`SETTIME (hh, mm, ss, cc);`	Establece la hora del sistema.
`GETDATE (aa, mm, dd, diasemana);`	Asigna a las variables la fecha del sistema. DíaSemana es un numero entre 0 y 6.
`SETDATE (aa, mm, dd)`	Establece la fecha del sistema.

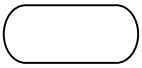
Apéndice C: La pantalla en modo texto



Apéndice D: Diagramas de flujo: algorítmica

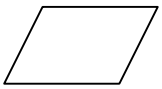
En este apéndice te explico la simbología utilizada para dibujar un digrama de flujo, y te muestro varios ejemplos sencillos de su aplicación en el lenguaje Pascal.

Aquí tienes los símbolos básicos:



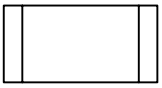
INICIO / FIN del programa
Nos indica el inicio del programa, y también el fin.
También se utiliza para determinar el inicio y el final de los subprogramas o funciones.

Proceso interno simple
Un proceso interno simple, puede ser una suma, un producto, un cálculo determinado.



Entrada por teclado/Salida por pantalla
Siempre que aparece este símbolo, el usuario y la máquina se comunican.
Si es de entrada o salida, lo determinamos por el contexto que le demos.

Condición a evaluar
Una condición que desvía el flujo del programa en 2 vías, SI/NO, para el IF-ELSE de Pascal, o varias vías para el CASE OF.
También representamos las condiciones evaluadas con los bucles con este símbolo.

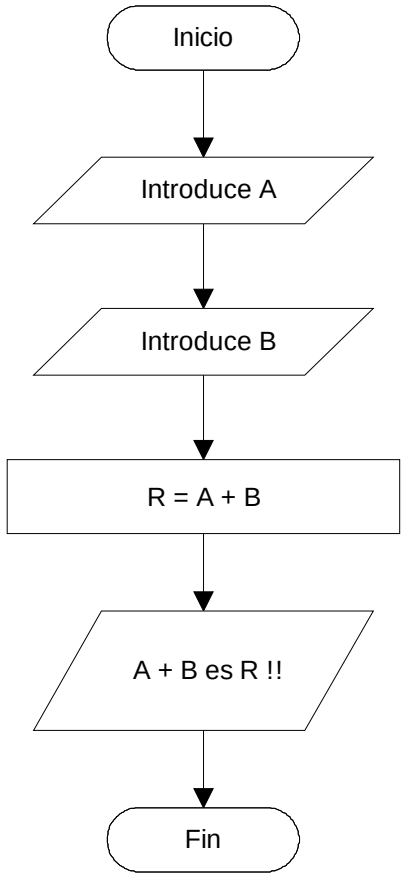


Proceso interno complejo o llamada a sub-programa
Un proceso interno complejo, que se recoge en otro sub-diagrama de flujo, ya sea un sub-programa, o varias operaciones consecutivas.

Ejemplo 1

El siguiente programa pide dos números enteros por teclado y muestra el valor de su suma.

Es un claro ejemplo de estructura secuencial (una instrucción detrás de otra).

Diagrama de flujo	*Programa en Pascal*
	``` Program calculadora1; Uses Crt; Var numa, numb : integer; result: integer; Begin write('Introduce NUMA: '); readln(numa); write('Introduce NUMB: '); readln(numb); result := numa + numb; writeln(numa, ' + ', numb, ' es ', result); readkey; End. ```

Apéndice E: Tabla ASCII

Formato de caracteres estándares:

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	⌂

Tabla extendida de códigos ASCII:

128	Ç	144	É	160	á	176	░	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	|	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	φ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Apéndice F: Funciones más utilizadas y ejemplos de uso

En este apéndice recogo una fabulosa colección de ejemplos para las funciones más utilizadas durante el curso de Fundamentos de Informática de la EUSS.

Es sumamente importante que piques todos y cada uno de estos ejemplos, esforzándote en entender todo lo que se hace en cada uno de ellos.

Sólo así conseguirás adquirir el nivel suficiente para aprobar la asignatura, para tener nociones de informática y para enfrentarte a la asignatura de Programación si la cursas en posteriores años.

¿Has leído la “introducción del autor”?
Allí te explico cómo empecé a programar. Repito: sencillamente tienes que ir picando programas y programas, intentando entender al máximo lo que hacen y el por qué, y poco a poco irás viendo que TODO tiene sentido en esta asignatura, y cada cosa nueva que aprendas, motivará tus ganas de seguir estudiándola.

¿Quieres aprender a programar? ¿Quieres aprobar?
ADELANTE!
Esfuérzate!
No faltes a clase!
Leete TODO el libro!
Busca ayuda si la necesitas!

Pero... no descanses hasta haber entendido todo lo que en este libro explico, porque tampoco es tan difícil!

Length

Prototipo:

```
function Length(s string) : Integer
```

Descripción:

La función `Length`, devuelve la longitud en carácteres de la cadena que le pasemos.

Ejemplos de uso:

```
Program cadenaTexto;

// Pide una cadena de texto por teclado.
// Muestra por pantalla cuántos carácteres ocupa, por qué letra empieza
// y por cuál acaba.

uses Crt;

var
   texto: string;
   tamanyo: integer;

Begin
   writeln('Escribe una frase ');
   readln(texto);
   tamanyo:=length(texto); // length retorna el tamaño en carácteres de
                           // la cadena que le pasamos, que siempre coincide
                           // con la posición del ultimo caracter en dicha
                           // cadena.

   writeln('Ocupa: ', tamanyo, ' caracteres');
   writeln('Empieza por ',texto[1],' y acaba por ', texto[tamanyo]);

   ReadKey;

End.
```

```
Program cuentaVocales;

// Pide una cadena de texto por teclado.
// Muestra por pantalla cuántas vocales tiene.

uses Crt;

var
   texto: string;
   z: integer;
   contador: integer;

Begin
   contador:=0;   // Inicialmente consiedaramos que hay 0 vocales

   writeln('Escribe una frase ');
   readln(texto);

   // Bucle que recorre todos los carácteres de la cadena, y uno a uno
   // comprueba si es una vocal. Si es así, se incrementa el contador
   // de vocales existentes.
   for z:=1 to length(texto) do
      if (texto[z]='a') or
         (texto[z]='A') or
         (texto[z]='e') or
         (texto[z]='E') or
         (texto[z]='i') or
         (texto[z]='I') or
         (texto[z]='o') or
         (texto[z]='O') or
         (texto[z]='u') or
         (texto[z]='U') then contador:=contador+1;

   writeln('Hay ', contador, ' vocales en el texto.');

   ReadKey;

End.
```

```
Program cuentaLetras;

// Cuenta las vocales de un texto introducido por teclado,
// contando las vocales por separado para cada una de ellas (a,e,i,o,u).
// Repite hasta que no pulses 'n' ó 'N'.

uses Crt;

var
   texto: string;
   ocupa: integer;
   i: integer;
   contA: integer;
   contE: integer;
   contI: integer;
   contO: integer;
   contU: integer;
   vocales: integer;
   tecla: char;


Begin
   repeat
      contA:=0;
      contE:=0;
      contI:=0;
      contO:=0;
      contU:=0;

      writeln('Escribe una frase o palabra: ');
      readln(texto);
      ocupa:=length(texto);

      writeln('Lo que has escrito tiene ', ocupa , ' caracteres.');

      for i:=1 to ocupa do
      begin
         if (texto[i]='a') or (texto[i]='A') then contA:=contA+1;
         if (texto[i]='e') or (texto[i]='E') then contE:=contE+1;
         if (texto[i]='i') or (texto[i]='I') then contI:=contI+1;
         if (texto[i]='o') or (texto[i]='O') then contO:=contO+1;
         if (texto[i]='u') or (texto[i]='U') then contU:=contU+1;
      end;

      vocales:=contA + contE + contI + contO + contU;

      writeln('Vocales: ', vocales);
      writeln;
      writeln('Vocal A: ', contA);
      writeln('Vocal E: ', contE);
      writeln('Vocal I: ', contI);
      writeln('Vocal O: ', contO);
      writeln('Vocal U: ', contU);

      writeln('quieres repetir (S/N)? ');
      tecla:=readkey;

   until(tecla='N') or (tecla='n');

End.
```

ReadKey

Prototipo:

```
function ReadKey : Char
```

Descripción:

La función `ReadKey`, detiene la ejecución del programa esperando a que se pulse una tecla.
El valor de la tecla pulsada se devuelve, y se puede o no utilizar.

Una llamada a `ReadKey` sin asignación, simplemente detiene el programa hasta que pulsemos una tecla, y es útil para ver los resultados del programa en cierto momento (por ejemplo antes de acabar, o para usarlo junto con el típico mensaje 'Pulse una tecla para continuar'.

Si utilizamos el valor retornado (es decir, el de la tecla pulsada), podemos hacer menús de opciones, o tomar decisiones a partir de dicho valor.

Ejemplos de uso:

```
Program continuar;

// Pide una cadena de texto por teclado.
// Muestra por pantalla cuántos carácteres ocupa, por qué letra empieza
// y por cuál acaba.

uses Crt;

var
   tecla: char;

Begin

   writeln('Pulsa una tecla...');

   tecla:=ReadKey;

   writeln('Has pulsado: ', tecla);


   writeln('Pulsa cualquier tecla para salir...');
   ReadKey;
End.
```

Chr

Prototipo:

```
function Chr(X: Byte): Char;
```

Descripción:

Le pasamos un número del 0 al 255 (es decir, de tipo `Byte`), y devuelve el carácter que corresponde a ese número en la tabla ASCII.

Ejemplos de uso:

```
Program muestraVocales;

// Muestra por pantalla las 5 vocales a través de 5 conversiones con chr de su
// código ASCII.

uses Crt;

var
   letra: char;

Begin

   letra:=chr(65);   // Ponemos la 'A' en letra
   writeln(letra);

   letra:=chr(69);   // Ponemos la 'E' en letra
   writeln(letra);

   letra:=chr(73);   // Ponemos la 'I' en letra
   writeln(letra);

   letra:=chr(79);   // Ponemos la 'O' en letra
   writeln(letra);

   letra:=chr(85);   // Ponemos la 'U' en letra
   writeln(letra);

   ReadKey;
End.
```

GotoXY

Prototipo:

```
procedure GotoXY(X, Y: Byte);
```

Descripción:

Mueve el cursor a la posición `X`, `Y` de la pantalla en modo texto.
Las operaciones que muestren datos por pantalla tras un `gotoxy`, tomarán dichas coordenadas.

Ejemplos de uso:

```
Program muevePorPantalla;

// Muestra una diagonal de asteriscos ('*') por pantalla.

uses Crt;

var
   pos: integer;

Begin

   gotoxy(20,5);
   write('ESTO ES UNA DIAGONAL!');

   for pos:=1 to 20 do
   begin
      gotoxy(pos,pos);
      write('*');
   end;

   ReadKey;

End.
```

TextColor

Prototipo:

```
procedure TextColor(Color: Byte);
```

Descripción:

Cambia el color del texto que se mostrará a partir de su llamada.
Los colores són números, aunque el compilador tiene declaradas constantes internas con el nombre de los colores (en inglés!): `Blue`, `Green`, `Red`, `Grey`, etc...

Ejemplos de uso:

```
Program textoColor1;

// Escribe texto por pantalla en diferentes colores.

uses Crt;

Begin

   textcolor(Red);
   writeln('Texto en Rojo!');
   textcolor(Green);
   writeln('Texto en Verde!');
   textcolor(Magenta);
   writeln('Texto en Magenta!');
   textcolor(Blue);
   writeln('Texto en Azul!');

   ReadKey;

End.
```

TextBackground

Prototipo:

```
procedure TextBackground(X, Y: Byte);
```

Descripción:

Cambia el color de fondo del texto que se mostrará a partir de su llamada.
Los colores són números, aunque el compilador tiene declaradas constantes internas con el nombre de los colores (en inglés!): `Blue`, `Green`, `Red`, `Grey`, etc...

Ejemplos de uso:

```
Program fondoTextoColor;

// Escribe texto por pantalla en diferentes colores de fondo.

uses Crt;

Begin

   textBackground(Red);
   writeln('Texto en fondo Rojo!');
   textBackground(Green);
   writeln('Texto en fondo Verde!');
   textBackground(Magenta);
   writeln('Texto en fondo Magenta!');
   textBackground(Blue);
   writeln('Texto en fondo Azul!');

   ReadKey;

End.
```

ClrScr

Prototipo:

```
procedure ClrScr;
```

Descripción:

Borra el contenido de la ventana.

Ejemplos de uso:

```
Program borraPantalla;
// Borra el contenido de la pantalla.
uses Crt;
Begin
   write('Pulsa una tecla para borrar la pantalla');
   readkey;
   clrscr;
   readkey;
   write('Pulsa una tecla para salir...');
   readkey;
End.
```

UpCase

Prototipo:

```
function UpCase(S: String): String;
```

Descripción:

`UpCase` pasa `S` a mayúsculas y la devuelve.
También funciona con un único carácter.

Ejemplos de uso:

```
Program cadenas;

// Pide una cadena por teclado y pasa todos su carácteres a mayúsuculas.

uses Crt;

var
   texto: string;

Begin

   writeln('Introduce un texto con mayúsculas y minúsculas');
   readln(texto);

   texto:=UpCase(texto);

   writeln('Convertido a mayúsculas:');
   writeln(texto);

   readkey;

End.
```

LowerCase

Prototipo:

```
function LowerCase(S: String): String;
```

Descripción:

`LowerCase` pasa `S` a minúsculas y la devuelve.
También funciona con un único carácter.

Ejemplos de uso:

```
Program cadenas;

// Pide una cadena por teclado y pasa todos su carácteres a minúsculas.

uses Crt;

var
   texto: string;

Begin

   writeln('Introduce un texto con mayúsculas y minúsculas');
   readln(texto);

   texto:=LowerCase(texto);

   writeln('Convertido a minúsculas:');
   writeln(texto);

   readkey;

End.
```

Insert

Prototipo:

```
procedure Insert(Fuente: String; var Destino: String; Index: Integer);
```

Descripción:

`Fuente` es una expresión de tipo `string`.
`Destino` es una variable de tipo `string` de cualquier longitud.
`Index` es una expresión entera.

Insert inserta la cadena `Fuente` en la cadena `Destino`, en la posición indicada por `Index`.

Si la cadena resultante es mayor de 255 caracteres, se trunca después del carácter número 255.

Ejemplos de uso:

```
Program muevePorPantalla;

// Inserta ' de la' en un texto que inicialmente es: 'Dia Revolucion'

uses Crt;

var
   texto: string;

Begin

   texto:='Dia Revolucion';

   writeln('Texto antes: ', texto);

   insert(' de la', texto, 4);

   writeln('Texto despues: ', texto);

   readkey;

End.
```

Apéndice G: No olvides que...

- Informática es una palabra derivada de Información y Automática.
- Es importante comentar los programas para no olvidar su funcionamiento con el paso del tiempo y para dejarlos más claros a las demás personas que lo han de examinar.
- Tabular, indentar y estructurar bien los programas para aclarar su lectura.
- Se hace **:=** para asignar y **=** para comparar:

```
num:=10;                if(num=10) then...
```

- Para dividir con decimales se hace con **/** y para la división entera se usa el **div**.
- Para poner comparaciones unidas con and o or, etc... se utilizan los paréntesis, si sólo es una comparación simple, no son necesarios:

```
if (num<1) or (num>8) do ...

while not( (num=5) and (tecla='0') ) do ...

repeat ... until tecla<>#27

if num=5 do ...
```

Apéndice H: Bibliografía

- Luís Joyanes Aguilar: Fundamentos de programación – Algoritmos y estructura de datos. McGraw-Hill, 1998.
- César Latorre: Pseudo-codis i Programació Estructurada. Edebé, 1996

Referente al autor

El autor, Sergio Guillén nació el 10 de marzo de 1978.

Técnico electrónico desde 1998 por las Escuelas Profesionales Salesianas de Sarrià (epsS), desde los 7 años no ha parado de tocar instrumentros electrónicos.

Empezó a programar con 12 años (en 1990) en lenguaje BASIC.

Sus actuales proyectos van desde la redacción de este cuaderno de apuntes, al diseño, construcción y programación de un pequeño robot de combate de sumo.

Pero lo más importante, es que pese a trabajar y tener el tiempo bastante ocupado, no deja de estudiar para de momento, acabar la Ingeniería Técnica en la Escuela Universitaria Salesiana de Sarrià (EUSS).

Referente al libro y al autor

Este libro es diferente.

Normalmente los libros, los apuntes, etc., los hacemos los profesores. El estilo que les imponemos suele ser de máximo rigor. Muy “académicos”. Y es bueno que así sea. Los alumnos de ingeniería se preparan para un mundo con mucho rigor y competitivo. Pero no podemos obviar que ese mismo rigor nos aleja, en ocasiones, de una buena comunicación con los alumnos. No entienden ni el lenguaje ni el rigor académico, por que, simplemente, no están de moda.

Este libro es diferente, insisto. Es un libro hecho por un alumno para los alumnos. Con su estilo, con su lenguaje, y fruto de la experiencia de Sergi en sus clases de repaso. Con ejemplos y modos de explicar que sus compañeros pueden entender.

El esfuerzo realizado por Sergi es grande. Sólo lo saben los que han tenido que escribir algún libro. Hay que escribir, leer varias veces lo escrito, corregir, eliminar, añadir, condensar, elegir de todo lo que quieres explicar lo más importante (los libros no pueden ser infinitos), etc. Y, lo más duro dedicar tiempo a costa de estudios, amigos, familia, etc. Pero el esfuerzo de Sergi habrá merecido la pena si uno solo de los lectores hereda su entusiasmo por la programación y si sirve de base para entender los libros “rigurosos”.

No debéis olvidar nunca que realmente sabréis programar, no cuando hagáis un programa que funcione sin errores, sino cuando entendáis esos libros llenos de rigor académico y los sepáis poner en práctica.

Quiero, por último, dar las gracias a Sergi por el esfuerzo realizado y por la ayuda que supondrá para la asignatura de Fundamentos de Informática.

César Latorre
Profesor de Fundamentos de Informática de la EUSS