CURSO DE INTRODUCCIÓN A LA PROGRAMACIÓN CON PSEUDOCÓDIGO

Un algoritmo es un conjunto de acciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un problema. El pseudocódigo, nos permite una aproximación del algoritmo al lenguaje natural y por tanto una redacción rápida del mismo. En este tema se presenta los fundamentos para analizar problemas y resolverlos a través de pseudocódigo.

Resolución de problemas

¿Qué es un problema?

Un problema es un asunto o un conjunto de cuestiones que se plantean para ser resueltas. La naturaleza de los problemas varía con el ámbito o el contexto: problemas matemáticos, químicos, filosóficos, etc.

Es importante que al abordar un problema se tenga una **descripción simple y precisa del mismo**, de lo contrario resultaría complejo modular, simular, o programar su solución en un ordenador.

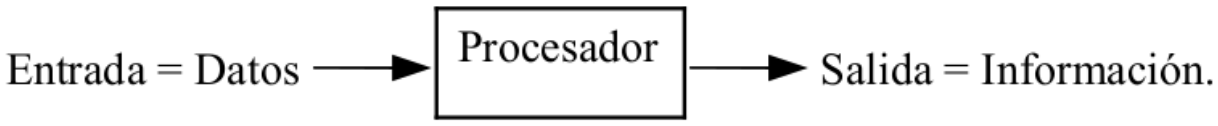
¿Cómo vamos a solucionar los problemas?

Un **programador** es una persona que resuelve problemas, y para llegar a ser un programador eficaz se **necesita aprender a resolver problemas de un modo riguroso y sistemático**:

- *Definición o análisis del problema*: consiste en el estudio detallado del problema. Se debe identificar los datos de entrada, de salida y la descripción del problema.
- *Diseño del algoritmo*: que describe la secuencia ordenada de pasos que conduce a la solución de un problema dado: **algoritmo**.
- *Transformación del algoritmo en un programa (codificación)*: Se expresa el algoritmo como un programa en un **lenguaje de programación**.
- *Ejecución y validación del programa*.

Sistemas de información

Sistema de procesamiento de información es un sistema que transforma datos brutos en información organizada, significativa y útil.



- *Datos*: se refiere a la representación de algún hecho, concepto o entidad real (palabras escritas, números, dibujos etc.)
- *Información*: Información implica datos procesados y organizados.
- *Procesador*: Proceso por el que se convierte datos de entrada en información útil.

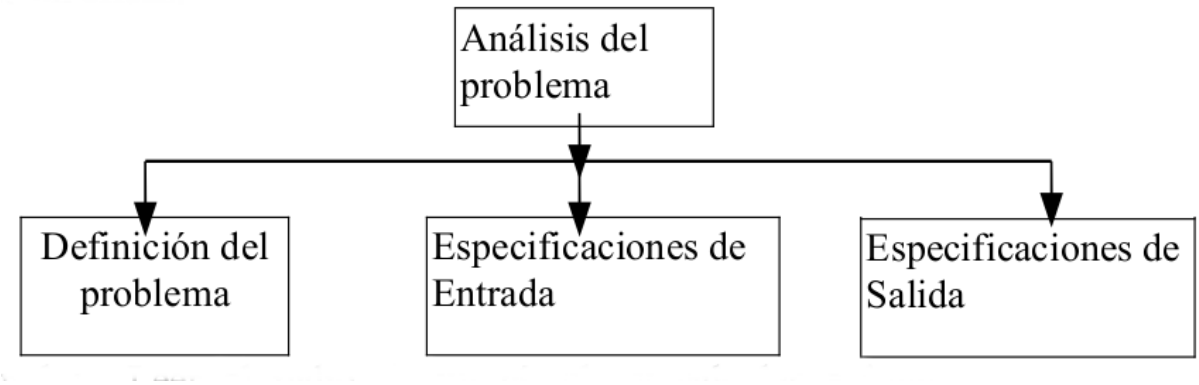
El conjunto de instrucciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un sistema específico o clase de problemas, se denomina **algoritmo**. O sea, un algoritmo es una fórmula para la resolución de un problema.

Cuando el procesador es un ordenador, el algoritmo ha de expresarse de una forma que recibe el nombre de **programa**.

Análisis del problema

El primer paso, análisis del problema, requiere un estudio a fondo del problema y de todo lo que hace falta para poder abordarlo.

El propósito del análisis de un problema es ayudar al programador (Analista) para llegar a una cierta comprensión de la naturaleza del problema. Una buena definición del problema, junto con una descripción detallada de las especificaciones de entrada/salida, son los requisitos más importantes para llegar a una solución eficaz.



Ejemplo de análisis

Nos proponen el siguiente problema:

Leer el radio de una circunferencia y calcular e imprimir su superficie y su longitud.

Análisis

Definición del problema: Tenemos que saber que es el radio de una circunferencia, y saber que es su área y su longitud. Además, tenemos que saber cómo calcular el área y la

longitud. Por lo tanto, necesitamos saber el radio y utilizar las fórmulas para calcular el área y la longitud.

	Especificaciones
Entradas:	Radio de la circunferencia (Variable RADIO).
Salidas:	Superficie de la circunferencia (Variable SUPERFICIE). Longitud de la circunferencia (Variable LONGITUD)
Variables:	RADIO, SUPERFICIE, LONGITUD de tipo REAL.

Los datos de entrada y la información de salida se van a guardar en **variables**, donde se puede guardar datos. Las variables son de distintos **tipos de datos**: entero, real, cadena, booleano,

Especificaciones del problema

El resultado final del análisis es obtener una serie de documentos (**especificación**) en los cuales quedan totalmente definido el proceso a seguir en la resolución del problema.

Diseño de algoritmos

A partir de los requerimientos, resultados del análisis, empieza la etapa de **diseño** donde tenemos que construir un **algoritmo** que resuelva el problema.

Definición de algoritmo

Un **algoritmo** es un conjunto de acciones que especifican la secuencia de operaciones realizar, en orden, para resolver un problema.

Los algoritmos son independientes tanto del lenguaje de programación como del ordenador que los ejecuta.

Las características de los algoritmos son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

Ejemplo tradicional de un algoritmo: Cambiar la rueda pinchada de un coche.

Etapa de diseño

Aunque en la solución de problemas sencillos parezca evidente la **codificación** en un lenguaje de programación concreto, es aconsejable realizar el **diseño** del algoritmo, a partir del cual se codifique el programa.

Las soluciones a problemas más complejos pueden requerir muchos más pasos. Las estrategias seguidas usualmente a la hora de encontrar algoritmos para problemas complejos son:

- **Partición o divide y vencerás**: consiste en dividir un problema grande en unidades más pequeñas que puedan ser resueltas individualmente.
    - Ejemplo: Podemos dividir el problema de limpiar una casa en labores más simple correspondientes a limpiar cada habitación.
- **Resolución por analogía**: Dado un problema, se trata de recordar algún problema similar que ya esté resuelto. Los dos problemas análogos pueden incluso pertenecer áreas de conocimiento totalmente distintas.
    - Ejemplo: El cálculo de la media de las temperaturas de las provincias andaluzas y la media de las notas de los alumnos e una clase se realiza del mismo modo.

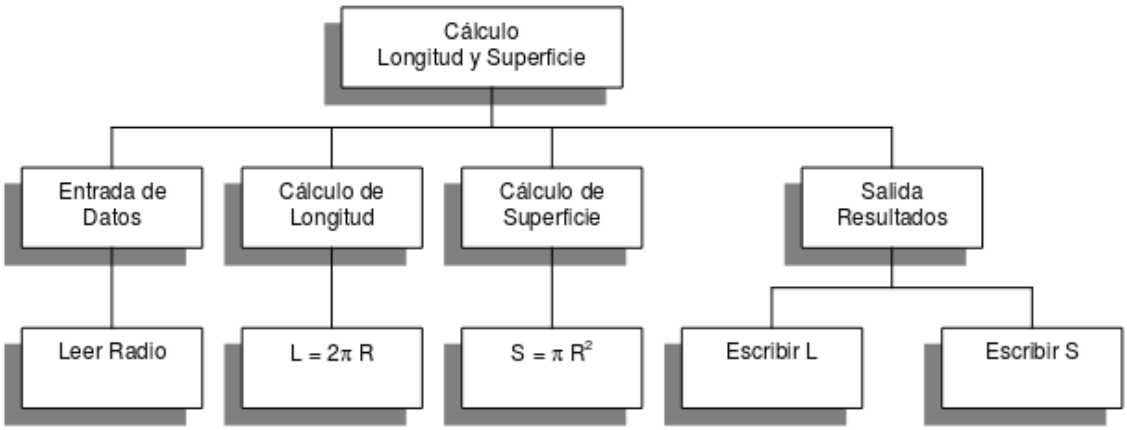
La descomposición del problema original en subproblemas más simples y a continuación dividir estos subproblemas en otros más simples se denomina **diseño descendente (top-down design).** Tras la primera descripción del problema (poco específica), se realiza una siguiente descripción más detallada con más pasos concretos. Este proceso se denomina **refinamiento del algoritmo**.

Ejemplo de diseño

Leer el radio de una circunferencia y calcular e imprimir su superficie y su circunferencia.

- Se puede dividir en tres subproblemas más sencillos:
    - Leer Radio
    - Calcular Superficie
    - Calcular Longitud
    - Escribir resultados
- Refinamiento del algoritmo:
    - `Leer Radio`
    - `Superficie <- PI * Radio ^ 2`
    - `Longitud <- 2 * PI * Radio`
    - `Escribir Radio, Longitud, Superficie`

Lo podemos ver en un **diagrama estructurado**:



Herramientas de representación de algoritmos

- Un **diagrama de flujo** es una de las técnicas de representación gráfica de algoritmos más antiguas. Ventajas: permite altos niveles de estructuración y modularización y es fácil de usar. Desventajas: son difíciles de actualizar y se complican cuando el algoritmo es grande.
- El **pseudocódigo**, nos permite una aproximación del algoritmo al lenguaje natural y por tanto una redacción rápida del mismo.

Estructura del algoritmo

Opciones del lenguaje (perfiles)

El lenguaje que utilizamos para construir el pseudocódigo no es estándar. Podemos añadir o eliminar algunas reglas de sintaxis sin ningún problema. En la opción *Configurar-Opciones del Lenguaje (perfiles)*, podemos escoger las características del pseudocódigo que vamos a utilizar. Tenemos tres alternativas:

- Escoger un perfil que define un pseudocódigo utilizado en distintos centros educativos y universidades.
- Perfil **flexible**: Está escogido por defecto, y no es muy exigente con las reglas que hay que utilizar para escribir el pseudocódigo.
- Perfil **estricto**: Establece unas reglas que hacen que el pseudocódigo se parezca más a un lenguaje de programación: se debe definir las variables y sus tipos, las instrucciones deben terminar en punto y coma, …

Estructura de un algoritmo en pseudocódigo

Todo algoritmo en pseudocódigo tiene la siguiente estructura general:

```
Proceso SinTitulo
    acción 1;
    acción 2;
    ...
    acción n;
FinProceso
```

- Comienza con la palabra clave `Proceso` (o alternativamente `Algoritmo`, son sinónimos) seguida del nombre del programa.
- Le sigue una secuencia (**Estructura de control secuencial**) de instrucciones. Una secuencia de instrucciones es una lista de una o más instrucciones y/o estructuras de control.
- Finaliza con la palabra `FinProceso` (o `FinAlgoritmo`).
- La identación no es significativo, pero se recomienda para que el código sea más legible.
- No se diferencia entre mayúsculas y minúsculas. Preferible las minúsculas, aunque a veces se añaden automáticamente los nombres con la primera letra en mayúsculas.

Comentarios

Se pueden introducir comentarios luego de una instrucción, o en líneas separadas, mediante el uso de la doble barra (`//`). Todo lo que precede a `//`, hasta el fin de la línea, no será tomado en cuenta al interpretar el algoritmo.

Nuestro primer programa

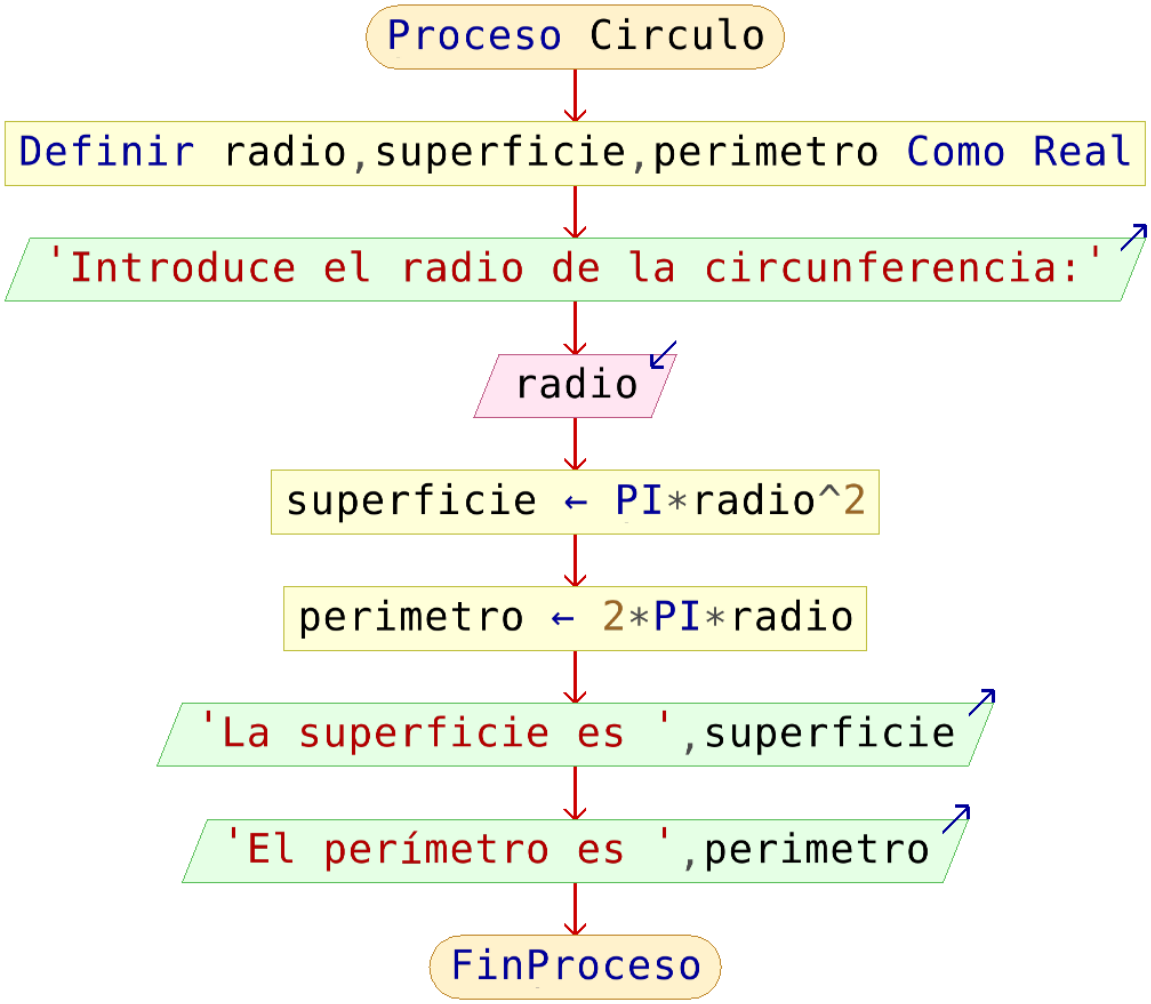
Leer el radio de un círculo y calcular e imprimir su superficie y su circunferencia.

El pseudocódigo podría ser:

```
//Leer el radio de un círculo y calcular e imprimir su superficie y su
circunferencia.
//Análisis
//Entradas: Radio del circulo (Variable RADIO).
//Salidas: Superficie del circulo (Variable SUPERFICIE) y Circunferencia
del circulo (Variable PERIMETRO)
//Variables: RADIO, SUPERFICIE, PERIMETRO de tipo REAL

Proceso Circulo
        Definir radio,superficie,perimetro como Real;
        Escribir "Introduce el radio de la circunferencia:";
        Leer radio;
        superficie <- PI * radio ^ 2;
        perimetro <- 2 * PI * radio;
        Escribir "La superficie es ",superficie;
        Escribir "El perímetro es ",perimetro;
FinProceso
```

Y el diagrama de flujo:



Tipos de datos simples

El tipo de dato representa la clase de datos con el que vamos a trabajar. Tenemos los siguientes tipos de datos simples:

- Números enteros: Nos sirve para representar números enteros.
- Números reales: Nos sirve para representar números reales.
- Cadenas de caracteres: Nos permite trabaja con cadenas de caracteres.
- Valores lógicos: Nos permite trabajar con valores lógicos.

Al escribir pseudocódigo en un programa de ordenador, tenemos que representar cada tipo de datos en binario y tenemos que decidir la memoria que va a ocupar cada uno de los tipos de datos.

En este caso PseInt representa los valores enteros con 32 bit, por lo tanto, en un tipo entero se pueden representar 2^{32} valores, es decir desde el `-2147483648` hasta el `2147483647`.

Por lo tanto, si vamos a trabajar con números más grandes o más pequeños hay que utilizar un tipo de dato Real.

Literales

Los literales nos permiten representar valores. Los literales son de distintos tipos de datos.

- **Literales enteros**: Ejemplos: 3, 0, -1, …
- **Literales reales**: Se utiliza el `.` para separar la parte entera y decimal. Por ejemplo: 3.0, -1.4, 2.345,…
- **Literales cadenas**: Se utiliza las `"` para indicar una cadena de caracteres, por ejemplo: “Hola”, “123”, “”, …
- **Literales lógicos**: sólo pueden tener dos valores: `Verdadero` y `Falso`.

Variables

Un variable nos permite almacenar información. Durante el análisis del problema, determinado las variables que vamos a necesitar en nuestro algoritmo (los datos de entrada y la información de salida).

Cada variable tiene un nombre y al crearlas hay que indicar el tipo de datos que va a almacenar.

Declaración de variables

El perfil “Estricto” nos obliga, al igual que muchos lenguajes de programación, ha indicar explícitamente las variables que vamos a utilizar y sus tipos. Aunque no es necesario es recomendable que las declaraciones se hagan al principio del algoritmo.

Para definir una variable usamos la siguiente instrucción:

```
Definir <var1>, <var2>, ..., <varN> como <Tipo de datos>;
```

Como tipo de datos podemos poner las siguientes opciones:

- Tipo entero: `Entero`
- Tipo real: `Real`, `Numérico` o `Numero`
- Tipo cadena de caracteres: `Carácter`, `Texto` o `Cadena`
- Tipo lógico: `Lógico`

Por ejemplo:

```
Definir numero1, numero2 como Entero;
Definir superficie, perimetro como Real;
Definir nombre como Carácter;
Definir mayor_edad como Lógico;
```

Operadores y expresiones

Expresiones

Una expresión es una combinación de variables, literales, operadores, funciones y expresiones, que tras su evaluación o cálculo nos devuelven un valor de un determinado tipo.

Veamos ejemplos de expresiones:

```
a + 7
(a ^ 2) + b
```

Operadores aritméticos

El valor devuelto por una operación aritmética es un número:

- `+`: Suma dos números
- `-`: Resta dos números
- `*`: Multiplica dos números
- `/`: Divide dos números.
- `%` ó `mod`: Módulo o resto de la división
- `^`: Potencia
- `+`, `-`: Operadores unarios positivo y negativo

Operadores de comparación

El valor devuelto por una operación de comparación es un valor lógico:

- `>`: Mayor que
- `<`: Menor que
- `=`: Igual que
- `<=`: Menor o igual
- `>=`: Mayor o igual

La comparación entre cadenas de caracteres se hace según el código ASCII.

Operadores lógicos

El valor devuelto por una operación lógica es un valor lógico:

- `&` ó `Y`: Conjunción, operación AND.
- `|` ó `O`: Disyunción, operación OR.
- `~` ó `NO`: Negación, operación NOT.

Tabla de verdad del operador Y

a	b	a Y b
V	V	V
V	F	F
F	V	F
F	F	F

Tabla de verdad del operador O

a	b	a O b
V	V	V
V	F	V
F	V	V
F	F	F

Tabla de verdad del operador NO

a	NO a
V	F
F	V

Ejercicios de operadores lógicos

Si tenemos 4 variables enteras a,b,c y d, expresar los siguientes predicados:

- Los valores de a y b son ambos menores que 17.
- Los valores de a,b y c son idénticos y distintos de d.
- Los valores de b y d están comprendidos estrictamente entre los valores de a y c, siendo a inferior a c.
- Entre los valores de a, b, y c hay al menos dos idénticos

Procedencia de operadores

La procedencia de operadores es la siguiente:

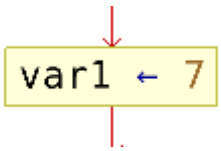
1. Los paréntesis rompen la procedencia.
2. La potencia
3. Operadores unarios
4. Multiplicar, dividir y módulo
5. Suma y resta
6. Operador lógico AND
7. Operador lógico OR
8. Operadores de comparación
9. Operadores lógicos (not, or, and)

Asignación de variables

Una vez que hemos definido una variable, podemos asignarle un valor con el operador de asignación (`<-` ó `:=`). El dato que se guarda en una variable puede estar expresado por un literal, guardado en otra variable o calculado tras operar una expresión. Por ejemplo:

```
var1 <- 7;
var2 <- var1;
var3 <- var1 + var2;
```

Como diagrama de flujo:



Tenemos que tener en cuenta lo siguiente:

- No se puede asignar un valor a una variable que no haya sido definida con anterioridad.
- No se puede utilizar una variable sin inicializar.
- Con cada asignación se pierde el valor anteriormente guardado en la variable.

Las siguientes asignaciones producen error:

- Una variable real (con parte decimal) si se asigna una variable entera.
- Una variable cadena de caracteres si se asigna a una variable numérica.
- Una variable numérica si se asigna a una variable cadena de caracteres.
- Una asignación de una variable lógica a cualquier otra variable de otro tipo, y, al contrario.

Incremento y decremento de una variable

Al incrementar o decrementar una variable numérica le modificamos su valor sumando o restando un número.

Por ejemplo, para incrementar una variable en 1:

```
numero1 <- numero1 + 1
```

Y para decrementar sería algo similar:

```
numero1 <- numero1 - 1
```

Ejemplo

```
Proceso Variables
        Definir numero1, numero2 como Entero;
        Definir superficie, perimetro como Real;
        Definir nombre como Carácter;
        Definir mayor_edad como Lógico;
        //Produce un error: no se ha definido la variable
        numero3 <- 3;
        //Produce un error: no se puede asignar un valor real a un entero
        numero1<-2.5;
        //Produce un error:no se puede utilizar una variable sin
inicializar
        numero1<-numero2*3;
        //Produce un error: Asignación de cadena a un número
        nombre<-"Pepe";
        superficie<-nombre;
        //También produce un error lo contrario
        superficie<-2.5;
        nombre<-superficie;

        //Inicializamos variables
        perimetro<-3.5;
        superficie<-4;
        numero1<-superficie;
        numero2<-5;
        nombre<-"Pepe";
        mayor_edad<-Verdadero;

        //Comprobamos como se actualiza una variable
        numero1<-7;
        numero2<-numero1*2;
        numero2<-numero2+1;
FinProceso
```

Entrada y salida de información

Entrada de datos

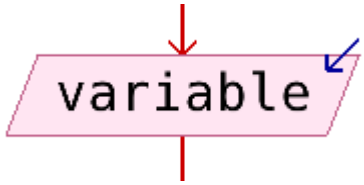
Con la instrucción `Leer` permite asignar un valor a una (o varias) variables leído por teclado. Hay que tener en cuenta las mismas consideraciones que en las asignaciones.

```
Leer <variable1>, <variable2>, ..., <variableN>;
```

Ejemplo:

```
Definir variable como entero;
Leer variable;
```

Como diagrama de flujo:



Salida de información

Para mostrar información por pantalla utilizamos la instrucción `Escribir`:

```
Escribir <dato1>, <dato2>, ..., <datoN>;
```

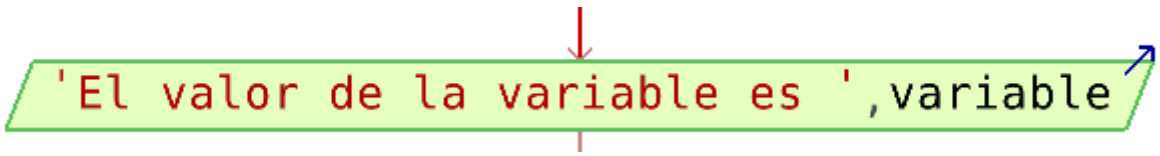
Los datos que mostramos pueden ser literales, variables o expresiones.

También podemos utilizar la instrucción `Escribir Sin Saltar`, para que no se introduzca una nueva línea para cada dato mostrado.

Ejemplo:

```
Escribir "El valor de la variable es ", variable;
```

Como diagrama de flujo:



Otras instrucciones

La instrucción `Borrar Pantalla` (o `Limpiar Pantalla`) permite, como su nombre lo indica, borrar la pantalla y colocar el cursor en la esquina superior izquierda.

```
Borrar Pantalla;
```

La instrucción `Esperar Tecla` detiene su algoritmo hasta que el usuario presione una tecla cualquiera de su teclado.

```
Esperar Tecla;
```

La instrucción `Esperar` también puede utilizarse para pausar el algoritmo durante un intervalo de tiempo predefinido, indicando a continuación de la palabra clave la longitud y unidad de dicho intervalo. Las unidades válidas son Segundos y Milisegundos.

```
Esperar 3 Segundos;
```

Funciones matemáticas

Una **función** es un subprograma que resuelve un problema determinado. Las funciones pueden tener **parámetros de entrada** y suelen devolver un valor de un tipo determinado.

En el pseudocódigo que estamos utilizando se pueden utilizar varias funciones matemáticas:

- `rc(número)` o `raiz(número)`: devuelve la raíz cuadrada del número.
- `abs(número)`: Devuelve el valor absoluto del número
- `ln(número)`: Devuelve el logaritmo natural del número
- `exp(número)`: Devuelve la función exponencial del número.
- `sen(número)`: Devuelve el seno de número.
- `cos(número)`: Devuelve el coseno de número.
- `tan(número)`: Devuelve la tangente de número.
- `asen(número)`: Devuelve el arcoseno de número.
- `acos(número)`: Devuelve el arcocoseno de número.
- `atan(número)`: Devuelve el arcotangente de número.
- `trunc(número)`: Devuelve la parte entera de número.
- `redon(número)`: Devuelve el entero más cercano a número.
- `azar(número)`: Devuelve el entero aleatorio en el rango [0;número-1].
- `aleatorio(numero1,numero2)`: Devuelve el entero aleatorio en el rango [numero1;numero2].

Las funciones trigonométricas reciben el ángulo en radianes. Para facilitar las conversiones se puede usar la constante PI (Ej: si A es un ángulo en grados, su coseno se obtiene con `cos( A * PI / 180 )`).

Ejemplo

```
Proceso Funciones_Matematicas
        Escribir "Raíz cuadrada de 9: ",rc(9);
        Escribir "Valor absoluto de -3: ",abs(-3);
        Escribir "Seno de 90 grados: ",sen(90 * PI / 180);
        Escribir "Truncamos 3.7: ",trunc(3.7);
        Escribir "Redondeamos 2.7: ",redon(2.7);
        Escribir "Un número al azar del 0 al 9: ",azar(10);
        Escribir "Un número al azar entre 10 y 20: ", aleatorio(10,20);
FinProceso
```

El resultado es:

```
Raíz cuadrada de 9: 3
Valor absoluto de -3: 3
Seno de 90 grados: 1
Truncamos 3.7: 3
Redondeamos 2.7: 3
Un número al azar del 0 al 9: 6
Un número al azar entre 10 y 20: 14
```

Funciones de cadenas de texto

El perfil del pseudocódigo con el que estamos trabajando, al igual que muchos lenguajes de programación, enumeran a partir del 0 el índice de cada carácter en la cadena. de esta manera el primer carácter de la cadena está en la posición 0.

Para trabajar con cadenas de caracteres también tenemos una serie de funciones predeterminadas:

- `longitud(cadena)`: Devuelve la cantidad de caracteres de la `cadena`.
- `mayusculas(cadena)`: Devuelve una copia de la `cadena` con todos sus caracteres en mayúsculas.
- `minusculas(cadena)`: Devuelve una copia de la `cadena` con todos sus caracteres en minúsculas.
- `subcadena(cadena,pos_ini,pos_fin)`: Devuelve una nueva cadena que consiste en la parte de la `cadena` que va desde la posición `pos_ini` hasta la posición `pos_fin`.
- `concatenar(cadena1,cadena2)`: Devuelve una nueva cadena resulta de unir las cadenas `cadena1` y `cadena2`.
- `convertirANumero(cadena)`: Recibe una cadena de caracteres que contiene un número (caracteres numéricos) y devuelve una variable numérica con el mismo.
- `convertirATexto(numero)`: Recibe un `número` y devuelve una variable cadena de caracteres de dicho real.

Ejemplo

```
Proceso Funciones_Cadena
        Definir cad1,cad2 como cadena;
        Definir num como Entero;
        cad1<-"informática";
        Escribir "La longitud de cad1 es ",longitud(cad1);
        Escribir "El primer carácter de cad1 es ", subcadena(cad1,0,0);
        Escribir    "El    último    carácter    de    cad1    es    ",
subcadena(cad1,longitud(cad1)-1,longitud(cad1)-1);
        Escribir "La cad1 en mayúsculas es ",mayusculas(cad1);
        cad2<-concatenar(cad1," es muy interesante");
        Escribir cad2;
        num<-ConvertirANumero("10");
        Escribir num;
        Escribir Concatenar("El número es ",ConvertirATexto(num));
        Escribir "El número es ",num;
FinProceso
```

El resultado es:

```
La longitud de cad1 es 11
El primer carácter de cad1 es i
El último carácter de cad1 es a
La cad1 en mayúsculas es INFORMÁTICA
informática es muy interesante
10
El número es 10
El número es 10
```

Nuestro primer pseudocódigo completo

Vamos a resolver un problema completo, siguiendo todas las etapas que hemos estudiado del ciclo de desarrollo. El enunciado del problema es el siguiente:

Queremos saber qué porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.

Análisis

- Definición del problema: Tenemos que saber cuántos hombres y mujeres hay en la clase, y calcular el porcentaje de cada uno.
- Datos de Entradas: Número de hombre y número de mujeres. Valores enteros.
- Información de salida: Porcentaje de hombres y porcentaje de mujeres. Valores reales.
- Variables: `cant_hombres`, `cant_mujeres` de tipo entero; `porcentaje_hombres`, `porcentaje_mujeres` de tipo real.

Diseño

Podemos dividir el problema en problemas más pequeños:

- Leer el número de hombres y el número de mujeres.
- Calcular el porcentaje de hombres y mujeres.
- Escribir los porcentajes

Refinamiento del algoritmo

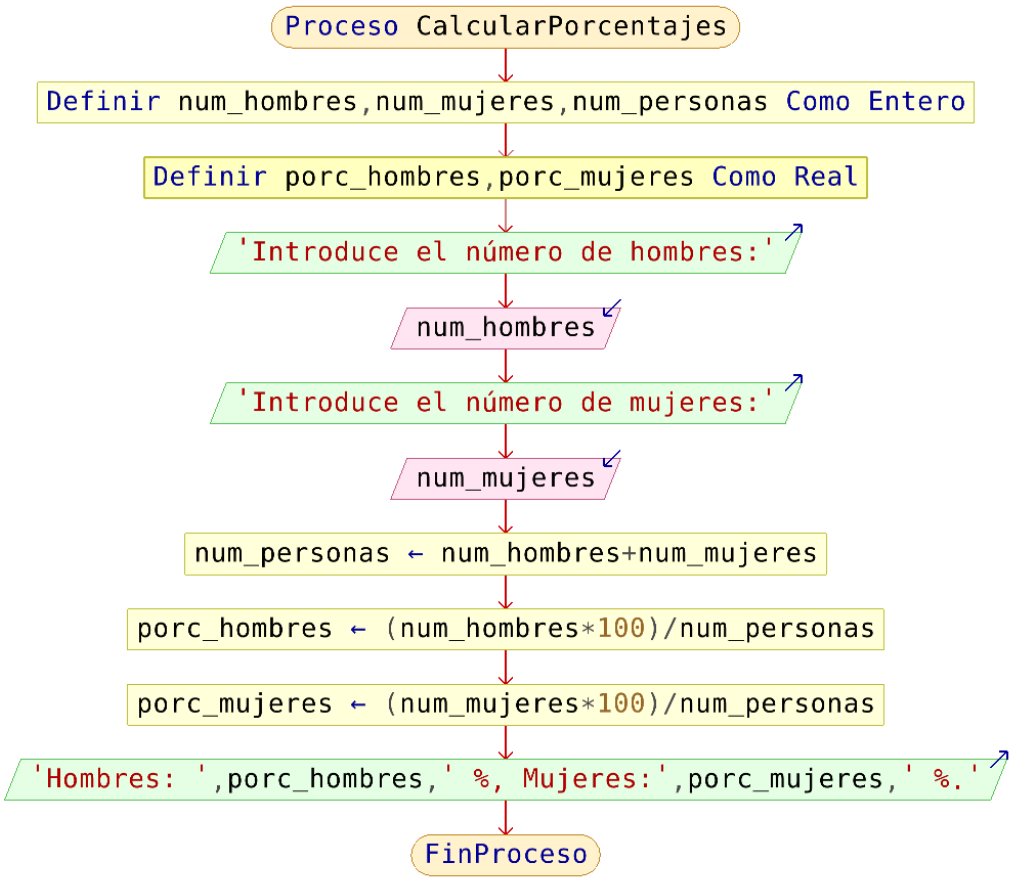
- Leer `num_hombres` y `num_mujeres`
- Calcular el número total de personas (`num_personas`)
- `porc_hombres` = `num_hombres` * 100 / `num_personas`
- `porc_mujeres` = `num_mujeres` * 100 / `num_personas`
- Escribir `porc_hombres`,`porc_mujeres`

Como vemos durante el diseño pueden aparecer nuevas posibles variables (variables intermedias), en este caso `num_personas` del tipo entero. Posteriormente cuando construimos el pseudocódigo tendremos que decidir si utilizamos todas las variables.

Pseudocódigo

```
Proceso CalcularPorcentajes
        Definir num_hombres, num_mujeres, num_personas Como Entero;
        Definir porc_hombres, porc_mujeres Como Real;
        Escribir "Introduce el número de hombres:";
        Leer num_hombres;
        Escribir "Introduce el número de mujeres:";
        Leer num_mujeres;
        num_personas <- num_hombres + num_mujeres;
        porc_hombres<- (num_hombres*100) / num_personas;
        porc_mujeres<- (num_mujeres*100) / num_personas;
        Escribir "Hombres: ",porc_hombres," %, Mujeres:",porc_mujeres,"
%.";
FinProceso
```

Diagrama de flujo



Ejecución paso a paso

En la opción del menú `Ejecutar -> Ejecutar Paso a Paso` nos permite realizar un seguimiento más detallado de la ejecución del algoritmo (proceso de **depuración** o **debugging**). Es decir, permite observar en tiempo real qué instrucciones

y en qué orden se ejecutan, como así también observar el contenido de variables o expresiones durante el proceso.

- El botón **Comenzar** del panel sirve para iniciar la ejecución automática. Cuando lo utilice, el algoritmo comenzará a ejecutarse automáticamente, avanzando de una instrucción a intervalos de tiempo regulares. Cada instrucción que se vaya ejecutando según el flujo del programa se irá seleccionando en el código de dicho algoritmo indicando además la línea con una flecha verde o amarilla sobre el margen izquierdo del pseudocódigo. La velocidad con que avanza la ejecución del algoritmo puede ajustarse desplazando el control rotulado como **Velocidad** en el panel.
- Otra forma de comenzar la ejecución paso a paso es utilizar el botón **Primer Paso** del mismo panel. Este botón iniciará la ejecución, pero a diferencia de Comenzar no avanzará de forma automática, sino que se parará sobre la primera línea del programa y esperará a que el usuario avance manualmente cada paso con el mismo botón (que pasará a denominarse **Avanzar un Paso**).
- Cuando el intérprete llega a una instrucción que involucra una llamada a un subproceso, puede avanzar directamente a la siguiente línea (ejecutando todo el subproceso como una única unidad a modo de caja negra, sin reflejar cómo es el flujo de ejecución dentro del mismo), o puede avanzar línea por línea dentro del subproceso. Esto depende del estado del cuadro **Entrar en subprocesos**.
- El botón **Pausar/Continuar** sirve para detener momentáneamente la ejecución automática del algoritmo y reanudarla nuevamente después. Detener el algoritmo puede servir para analizar el código fuente, o para verificar qué valor tiene asignado una variable o cuanto valdría una determinada expresión en ese punto.
- Para determinar el valor de una variable o expresión, una vez pausada la ejecución paso a paso, utilice el botón **Evaluar...**. Aparecerá una ventana donde podrá introducir cualquier nombre de variable o expresión arbitraria (incluyendo funciones y operadores), para luego observar su valor. Una forma rápida de observar el valor de una variable consiste en hacer click con el botón derecho del ratón sobre la misma en el panel de variables (ubicado en el margen izquierdo de la ventana).
- Si desea analizar cómo evolucionan uno o más variables a lo largo de la ejecución del algoritmo, puede activar la **Prueba de escritorio**. Esta opción genera una tabla donde las columnas representan variables o expresiones, y las filas los distintos estados ordenados por los que pasa el programa a medida que se va ejecutando. Para añadir variables, antes de empezar la ejecución paso a paso, pulsamos el botón **Agregar...**.
- Finalmente, la opción **Explicar con detalle cada paso** despliega en la parte inferior de la ventana un panel donde el intérprete comentará los pasos específicos que realiza al interpretar cada instrucción. Allí aparecen por ejemplo las expresiones que se evalúan y sus resultados, las variables que se leen o asignan, las decisiones que controlan el flujo de ejecución en las estructuras de control, etc. El panel dispone de un botón para avanzar manualmente, ya que no permite el modo de avance automático.

Ejercicios estructura secuencial

Ejercicio 1

Escribir un programa que pregunte al usuario su nombre, y luego lo salude.

Ejercicio 2

Calcular el perímetro y área de un rectángulo dada su base y su altura.

Ejercicio 3

Dados los catetos de un triángulo rectángulo, calcular su hipotenusa.

Ejercicio 4

Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.

Ejercicio 5

Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius. Recordar que la fórmula para la conversión es:

```
C = (F-32)*5/9
```

Ejercicio 6

Calcular la media de tres números pedidos por teclado.

Ejercicio 7

Realiza un programa que reciba una cantidad de minutos y muestre por pantalla a cuantas horas y minutos corresponde. Por ejemplo: 1000 minutos son 16 horas y 40 minutos.

Ejercicio 8

Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

Ejercicio 9

Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

Ejercicio 10

Un alumno desea saber cuál será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

- 55% del promedio de sus tres calificaciones parciales.
- 30% de la calificación del examen final.
- 15% de la calificación de un trabajo final.

Ejercicio 11

Pide al usuario dos números y muestra la "distancia" entre ellos (el valor absoluto de su diferencia, de modo que el resultado sea siempre positivo).

Ejercicio 12

Pide al usuario dos pares de números x1,y2 y x2,y2, que representen dos puntos en el plano. Calcula y muestra la distancia entre ellos.

Ejercicio 13

Realizar un algoritmo que lea un número y que muestre su raíz cuadrada y su raíz cúbica. PSeInt no tiene ninguna función predefinida que permita calcular la raíz cúbica, ¿Cómo se puede calcular?

Ejercicio 14

Dado un número de dos cifras, diseñe un algoritmo que permita obtener el número invertido. Ejemplo, si se introduce 23 que muestre 32.

Ejercicio 15

Dadas dos variables numéricas A y B, que el usuario debe teclear, se pide realizar un algoritmo que intercambie los valores de ambas variables y muestre cuánto valen al final las dos variables.

Ejercicio 16

Dos vehículos viajan a diferentes velocidades (v1 y v2) y están distanciados por una distancia d. El que está detrás viaja a una velocidad mayor. Se pide hacer un algoritmo para ingresar la distancia entre los dos vehículos (km) y sus respectivas velocidades (km/h) y con esto determinar y mostrar en que tiempo (minutos) alcanzará el vehículo más rápido al otro.

Ejercicio 17

Un ciclista parte de una ciudad A a las HH horas, MM minutos y SS segundos. El tiempo de viaje hasta llegar a otra ciudad B es de T segundos. Escribir un algoritmo que determine la hora de llegada a la ciudad B.

Ejercicio 18

Pedir el nombre y los dos apellidos de una persona y mostrar las iniciales.

Ejercicio 19

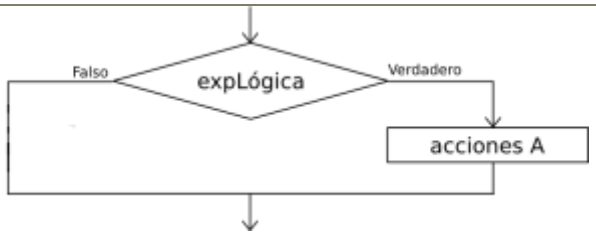
Escribir un algoritmo para calcular la nota final de un estudiante, considerando que: por cada respuesta correcta 5 puntos, por una incorrecta -1 y por respuestas en blanco 0. Imprime el resultado obtenido por el estudiante.

Ejercicio 20

Diseñar un algoritmo que nos diga el dinero que tenemos (en euros y céntimos) después de pedirnos cuantas monedas tenemos (de 2€, 1€, 50 céntimos, 20 céntimos o 10 céntimos).

Estructuras alternativas: si

Alternativa simple: si - finsi



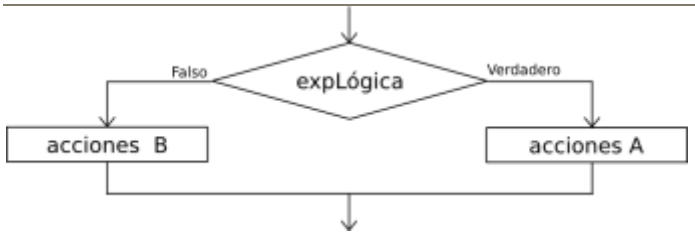
Al ejecutarse la instrucción `si` se evalúa la condición lógica. Si la condición lógica es **Verdadera** se ejecutan de manera secuencial el bloque de instrucciones *Acciones A*. Si la condición es **Falsa** no se ejecuta el bloque de instrucciones. Una vez ejecutado el `si` (opción verdadera o falsa) se continúa la ejecución de forma secuencial por la siguiente instrucción detrás del `FinSi`.

Ejemplo

Programa que pida la edad y diga si es mayor de edad.

```
Proceso mayor_edad
        Definir edad como entero;
        Escribir "Dime tu edad:";
        Leer edad;
        Si edad>=18 Entonces
                Escribir "Eres mayor de edad";
        FinSi
        Escribir "Programa terminado";
FinProceso
```

Alternativa doble: si - sino - finsi



Al ejecutarse la instrucción `si` se evalúa la condición lógica. Si la condición lógica es **Verdadera** se ejecutan de manera secuencial el bloque de instrucciones *Acciones A*. Si la condición es **Falsa** se ejecuta el bloque de instrucción *Acciones B*. Una vez ejecutado el `si` (opción verdadera o falsa) se continúa la ejecución de forma secuencial por la siguiente instrucción detrás del `FinSi`.

Ejemplo

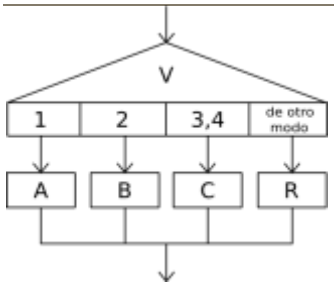
Programa que pida la edad y diga si es mayor de edad o menor de edad.

```
Proceso mayor_edad
        Definir edad como entero;
        Escribir "Dime tu edad:";
        Leer edad;
```

```
        Si edad>=18 Entonces
                Escribir "Eres mayor de edad";
        SiNo
                Escribir "Eres menor de edad";
        FinSi
        Escribir "Programa terminado";
FinProceso
```

Estructuras alternativas: Según

Alternativa múltiple: Según



La secuencia de instrucciones ejecutada por una instrucción `Según` depende del valor de una variable numérica.

```
Segun <variable> Hacer
    <número1>: <instrucciones>
    <número2>,<número3>: <instrucciones>
    <...>
    [De Otro Modo: <instrucciones>]
FinSegun
```

- Esta instrucción permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable de tipo numérico. Al ejecutarse, se evalúa el contenido de la variable y se ejecuta la secuencia de instrucciones asociada con dicho valor.
- Cada opción está formada por uno o más números separados por comas, dos puntos y una secuencia de instrucciones. Si una opción incluye varios números, la secuencia de instrucciones asociada se debe ejecutar cuando el valor de la variable es uno de esos números.
- Opcionalmente, se puede agregar una opción final, denominada `De Otro Modo`, cuya secuencia de instrucciones asociada se ejecutará sólo si el valor almacenado en la variable no coincide con ninguna de las opciones anteriores.
- Al finalizar se continúa la ejecución secuencia con la siguiente instrucción detrás del `FinSegun`.

Ejemplo

Programa que pide una nota de un examen por teclado y muestra la nota como “Sobresaliente”, “Notable”, “Bien”, “Suficiente”, “Suspendido”:

```
Proceso notas
        Definir nota como entero;
        Escribir "Dime tu nota:";
```

```
        Leer nota;
        Segun nota Hacer
                1,2,3,4: Escribir "Suspenso";
                5: Escribir "Suficiente";
                6,7: Escribir "Bien";
                8: Escribir "Notable";
                9,10: Escribir "Sobresaliente";
                De Otro Modo:
                        Escribir "Nota incorrecta";
        FinSegun
        Escribir "Programa terminado";
FinProceso
```

Ejercicios estructuras alternativas

Ejercicio 1

Algoritmo que pida dos números e indique si el primero es mayor que el segundo o no.

Ejercicio 2

Algoritmo que pida un número y diga si es positivo, negativo o 0.

Ejercicio 3

Escribe un programa que lea un número e indique si es par o impar.

Ejercicio 4

Crea un programa que pida al usuario dos números y muestre su división si el segundo no es cero, o un mensaje de aviso en caso contrario.

Ejercicio 5

Escribe un programa que pida un nombre de usuario y una contraseña y si se ha introducido “pepe” y “asdasd” se indica “Has entrado al sistema”, sino se da un error.

Ejercicio 6

Programa que lea una cadena por teclado y compruebe si es una letra mayúscula.

Ejercicio 7

Realiza un algoritmo que calcule la potencia, para ello pide por teclado la base y el exponente. Pueden ocurrir tres cosas:

- El exponente sea positivo, sólo tienes que imprimir la potencia.
- El exponente sea 0, el resultado es 1.
- El exponente sea negativo, el resultado es 1/potencia con el exponente positivo.

Ejercicio 8

Algoritmo que pida dos números ‘nota’ y ‘edad’ y un carácter ‘sexo’ y muestre el mensaje ‘ACEPTADA’ si la nota es mayor o igual a cinco, la edad es mayor o igual a dieciocho y el sexo es ‘F’. En caso de que se cumpla lo mismo, pero el sexo sea ‘M’, debe imprimir ‘POSIBLE’. Si no se cumplen dichas condiciones se debe mostrar ‘NO ACEPTADA’.

Ejercicio 9

Algoritmo que pida tres números y los muestre ordenados (de mayor a menor);

Ejercicio 10

Algoritmo que pida los puntos centrales x1,y1,x2,y2 y los radios r1,r2 de dos circunferencias y las clasifique en uno de estos estados:

- exteriores
- tangentes exteriores
- secantes
- tangentes interiores
- interiores
- concéntricas

Ejercicio 11

Programa que lea 3 datos de entrada A, B y C. Estos corresponden a las dimensiones de los lados de un triángulo. El programa debe determinar qué tipo de triangulo es, teniendo en cuenta los siguiente:

- Si se cumple Pitágoras entonces es triángulo rectángulo
- Si sólo dos lados del triángulo son iguales entonces es isósceles.
- Si los 3 lados son iguales entonces es equilátero.
- Si no se cumple ninguna de las condiciones anteriores, es escaleno.

Ejercicio 12

Escribir un programa que lea un año indicar si es bisiesto. Nota: un año es bisiesto si es un número divisible por 4, pero no si es divisible por 100, excepto que también sea divisible por 400.

Ejercicio 13

Escribe un programa que pida una fecha (día, mes y año) y diga si es correcta.

Ejercicio 14

La asociación de vinicultores tiene como política fijar un precio inicial al kilo de uva, la cual se clasifica en tipos A y B, y además en tamaños 1 y 2. Cuando se realiza la venta del producto, ésta es de un solo tipo y tamaño, se requiere determinar cuánto recibirá un productor por la uva que entrega en un embarque, considerando lo siguiente: si es de tipo A, se le cargan 20 céntimos al precio inicial cuando es de tamaño 1; y 30 céntimos si es de tamaño 2. Si es de tipo B, se rebajan 30 céntimos cuando es de tamaño 1, y 50 céntimos cuando es de tamaño 2. Realice un algoritmo para determinar la ganancia obtenida.

Ejercicio 15

El director de una escuela está organizando un viaje de estudios, y requiere determinar cuánto debe cobrar a cada alumno y cuánto debe pagar a la compañía de viajes por el servicio. La forma de cobrar es la siguiente: si son 100 alumnos o más, el costo por cada alumno es de 65 euros; de 50 a 99 alumnos, el costo es de 70 euros, de 30 a 49, de 95

euros, y si son menos de 30, el costo de la renta del autobús es de 4000 euros, sin importar el número de alumnos. Realice un algoritmo que permita determinar el pago a la compañía de autobuses y lo que debe pagar cada alumno por el viaje.

Ejercicio 16

La política de cobro de una compañía telefónica es: cuando se realiza una llamada, el cobro es por el tiempo que ésta dura, de tal forma que los primeros cinco minutos cuestan 1 euro, los siguientes tres, 80 céntimos, los siguientes dos minutos, 70 céntimos, y a partir del décimo minuto, 50 céntimos. Además, se carga un impuesto de 3 % cuando es domingo, y si es otro día, en turno de mañana, 15 %, y en turno de tarde, 10 %. Realice un algoritmo para determinar cuánto debe pagar por cada concepto una persona que realiza una llamada.

Ejercicio 17

Realiza un programa que pida por teclado el resultado (dato entero) obtenido al lanzar un dado de seis caras y muestre por pantalla el número en letras (dato cadena) de la cara opuesta al resultado obtenido.

- Nota 1: En las caras opuestas de un dado de seis caras están los números: 1-6, 2-5 y 3-4.
- Nota 2: Si el número del dado introducido es menor que 1 o mayor que 6, se mostrará el mensaje: “ERROR: número incorrecto.”.

Ejemplo:

```
Introduzca número del dado: 5
En la cara opuesta está el "dos".
```

Ejercicio 18

Realiza un programa que pida el día de la semana (del 1 al 7) y escriba el día correspondiente. Si introducimos otro número nos da un error.

Ejercicio 19

Escribe un programa que pida un número entero entre uno y doce e imprima el número de días que tiene el mes correspondiente.

Ejercicio 20

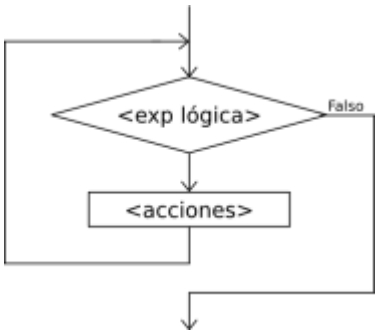
Una compañía de transporte internacional tiene servicio en algunos países de América del Norte, América Central, América del Sur, Europa y Asia. El costo por el servicio de transporte se basa en el peso del paquete y la zona a la que va dirigido. Lo anterior se muestra en la tabla:

Zona	Ubicación	Costo/gramo
1	América del Norte	24.00 euros
2	América Central	20.00 euros

Zona	Ubicación	Costo/gramo
3	América del Sur	21.00 euros
4	Europa	10.00 euros
5	Asia	18.00 euros

Parte de su política implica que los paquetes con un peso superior a 5 kg no son transportados, esto por cuestiones de logística y de seguridad. Realice un algoritmo para determinar el cobro por la entrega de un paquete o, en su caso, el rechazo de la entrega.

Estructuras repetitivas: Mientras



La instrucción `Mientras` ejecuta una secuencia de instrucciones mientras una condición sea verdadera.

```
Mientras <condición> Hacer
    <instrucciones>
FinMientras
```

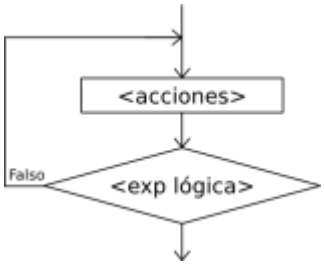
- Al ejecutarse esta instrucción, la condición es evaluada. Si la condición resulta verdadera, se ejecuta una vez la secuencia de instrucciones que forman el cuerpo del ciclo. Al finalizar la ejecución del cuerpo del ciclo se vuelve a evaluar la condición y, si es verdadera, la ejecución se repite. Estos pasos se repiten mientras la condición sea verdadera.
- Se puede dar la circunstancia que las instrucciones del bucle no se ejecuten nunca, si al evaluar por primera vez la condición resulta ser falsa.
- Si la condición siempre es verdadera, al ejecutar esta instrucción se produce un ciclo infinito. A fin de evitarlo, las instrucciones del cuerpo del ciclo deben contener alguna instrucción que modifique la o las variables involucradas en la condición, de modo que ésta sea falsificada en algún momento y así finalice la ejecución del ciclo.

Ejemplo:

Crea un programa que pida al usuario una contraseña, de forma repetitiva mientras que no introduzca “asdasd”. Cuando finalmente escriba la contraseña correcta, se le dirá “Bienvenido” y terminará el programa.

```
Proceso login
        Definir secreto, clave como cadena;
        secreto <- "asdasd";
        Escribir "Dime la clave:";
        Leer clave;
        Mientras clave<>secreto Hacer
                Escribir "Clave incorrecta!!!";
                Escribir "Dime la clave:";
                Leer clave;
        FinMientras
        Escribir "Bienvenido!!!";
        Escribir "Programa terminado";
FinProceso
```

Estructuras repetitivas: Repetir - Hasta Que



La instrucción `Repetir-Hasta Que` ejecuta una secuencia de instrucciones hasta que la condición sea verdadera.

```
Repetir
    <instrucciones>
Hasta Que <condición>
```

- Al ejecutarse esta instrucción, la secuencia de instrucciones que forma el cuerpo del ciclo se ejecuta una vez y luego se evalúa la condición. Si la condición es falsa, el cuerpo del ciclo se ejecuta nuevamente y se vuelve a evaluar la condición. Esto se repite hasta que la condición sea verdadera.
- Note que, dado que la condición se evalúa al final, las instrucciones del cuerpo del ciclo serán ejecutadas al menos una vez.
- Además, a fin de evitar ciclos infinitos, el cuerpo del ciclo debe contener alguna instrucción que modifique la o las variables involucradas en la condición de modo que en algún momento la condición sea verdadera y se finalice la ejecución del ciclo.

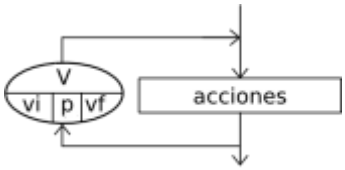
Ejemplo:

Crea un programa que pida al usuario una contraseña, de forma repetitiva mientras que no introduzca “asdasd”. Cuando finalmente escriba la contraseña correcta, se le dirá “Bienvenido” y terminará el programa.

```
Proceso login
        Definir secreto, clave como cadena;
        secreto <- "asdasd";
        Repetir
                Escribir "Dime la clave:";
                Leer clave;
```

```
                Si clave<>secreto Entonces
                        Escribir "Clave incorrecta!!!";
                FinSi
        Hasta Que clave=secreto
        Escribir "Bienvenido!!!";
        Escribir "Programa terminado";
FinProceso
```

Estructuras repetitivas: Para



La instrucción `Para` ejecuta una secuencia de instrucciones un número determinado de veces.

```
Para <variable> <- <inicial> Hasta <final> [Con Paso <paso>] Hacer
        <instrucciones>
FinPara
```

- Al ingresar al bloque, la variable `<variable>` recibe el valor `<inicial>` y se ejecuta la secuencia de instrucciones que forma el cuerpo del ciclo.
- Luego se incrementa la variable `<variable>` en `<paso>` unidades y se evalúa si el valor almacenado en `<variable>` superó al valor `<final>`.
- Si esto es falso se repite hasta que `<variable>` supere a `<final>`.
- Si se omite la cláusula `Con Paso <paso>`, la variable `<variable>` se incrementará en 1.

Ejemplo

Escribir en pantalla del 1 al 10.

```
Proceso Contar
        Definir var como Entero;
        Para var<-1 Hasta 10 Hacer
                Escribir Sin Saltar var," ";
        FinPara
FinProceso
```

Escribir en pantalla de 10 al 1.

```
Proceso ContarDescesdente
        Definir var como Entero;
        Para var<-10 Hasta 1 Con Paso -1 Hacer
                Escribir Sin Saltar var," ";
        FinPara
FinProceso
```

Escribir los números pares desde el 2 al 10.

```
Proceso ContarPares
```

```
        Definir var como Entero;
        Para var<-2 Hasta 10 Con Paso 2 Hacer
                Escribir Sin Saltar var," ";
        FinPara
FinProceso
```

Uso específico de variables: contadores, acumuladores e indicadores

Contadores

Un contador es una variable entera que la utilizamos para contar cuando ocurre un suceso. Un contador:

- Se **inicializa** a un valor inicial.

```
cont <- 0;
```

- Se **incrementa**, cuando ocurre el suceso que estamos contado se le suma 1.

```
cont <- cont + 1;
```

Ejemplo

Introducir 5 número y contar los números pares.

```
Proceso ContarPares
        Definir var,cont,num como Entero;
        cont<-0;
        Para var<-1 Hasta 5 Hacer
                Escribir Sin Saltar "Dime un número:";
                Leer num;
                Si num % 2 = 0 Entonces
                        cont<-cont+1;
                FinSi
        FinPara
        Escribir "Has introducido ",cont," números pares.";
FinProceso
```

Acumuladores

Un acumulador es una variable numérica que permite ir acumulando operaciones. Me permite ir haciendo operaciones parciales. Un acumulador:

- Se **inicializa** a un valor inicial según la operación que se va a acumular: a 0 si es una suma o a 1 si es un producto.
- Se **acumula** un valor intermedio.

```
acum <- acum + num;
```

Ejemplo

Introducir 5 número y sumar los números pares.

```
Proceso SumarPares
        Definir var,suma,num como Entero;
```

```
        suma<-0;
        Para var<-1 Hasta 5 Hacer
                Escribir Sin Saltar "Dime un número:";
                Leer num;
                Si num % 2 = 0 Entonces
                        suma<-suma+num;
                FinSi
        FinPara
        Escribir "La suma de los números pares es ",suma;
FinProceso
```

Indicadores

Un indicador es una variable lógica, que usamos para recordar o indicar algún suceso. Un indicador:

- Se **inicializa** a un valor lógico que indica que el suceso no ha ocurrido.

  indicador <- Falso

- Cuando ocurre el suceso que queremos recordar cambiamos su valor.

  indicador <- Verdadero

Ejemplo

Introducir 5 número e indicar si se ha introducido algún número par.

```
Proceso RecordarPar
        Definir var,num como Entero;
        Definir indicador como Lógico;
        indicador <- Falso;
        Para var<-1 Hasta 5 Hacer
                Escribir Sin Saltar "Dime un número:";
                Leer num;
                Si num % 2 = 0 Entonces
                        indicador <- Verdadero;
                FinSi
        FinPara
        Si indicador Entonces
                Escribir "Has introducido algún número par";
        SiNo
                Escribir "No has introducido algún número par";
        FinSi
FinProceso
```

Ejercicios estructuras repetitivas

Ejercicio 1

Crea una aplicación que pida un número y calcule su factorial (El factorial de un número es el producto de todos los enteros entre 1 y el propio número y se representa por el número seguido de un signo de exclamación. Por ejemplo 5! = 1x2x3x4x5=120),

Ejercicio 2

Crea una aplicación que permita adivinar un número. La aplicación genera un número aleatorio del 1 al 100. A continuación, va pidiendo números y va respondiendo si el número a adivinar es mayor o menor que el introducido, además de los intentos que te quedan (tienes 10 intentos para acertarlo). El programa termina cuando se acierta el número (además te dice en cuantos intentos lo has acertado), si se llega al límite de intentos te muestra el número que había generado.

Ejercicio 3

Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos.

Ejercicio 4

Realizar un algoritmo que pida números (se pedirá por teclado la cantidad de números a introducir). El programa debe informar de cuantos números introducidos son mayores que 0, menores que 0 e iguales a 0.

Ejercicio 5

Algoritmo que pida caracteres e imprima 'VOCAL' si son vocales y 'NO VOCAL' en caso contrario, el programa termina cuando se introduce un espacio.

Ejercicio 6

Escribir un programa que imprima todos los números pares entre dos números que se le pidan al usuario.

Ejercicio 7

Realizar un algoritmo que muestre la tabla de multiplicar de un número introducido por teclado.

Ejercicio 8

Escribe un programa que pida el límite inferior y superior de un intervalo. Si el límite inferior es mayor que el superior lo tiene que volver a pedir. A continuación, se van introduciendo números hasta que introduzcamos el 0. Cuando termine el programa dará las siguientes informaciones:

- La suma de los números que están dentro del intervalo (intervalo abierto).
- Cuantos números están fuera del intervalo.
- He informa si hemos introducido algún número igual a los límites del intervalo.

Ejercicio 9

Escribe un programa que, dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla el resultado de la potencia. No se puede utilizar el operador de potencia.

Ejercicio 10

Algoritmo que muestre la tabla de multiplicar de los números 1,2,3,4 y 5.

Ejercicio 11

Escribe un programa que diga si un número introducido por teclado es o no primo. Un número primo es aquel que sólo es divisible entre él mismo y la unidad. Nota: Es suficiente probar hasta la raíz cuadrada del número para ver si es divisible por algún otro número.

Ejercicio 12

Realizar un algoritmo para determinar cuánto ahorrará una persona en un año, si al final de cada mes deposita cantidades variables de dinero; además, se quiere saber cuánto lleva ahorrado cada mes.

Ejercicio 13

Una empresa tiene el registro de las horas que trabaja diariamente un empleado durante la semana (seis días) y requiere determinar el total de éstas, así como el sueldo que recibirá por las horas trabajadas.

Ejercicio 14

Una persona se encuentra en el kilómetro 70 de una carretera, otra se encuentra en el km 150, los coches tienen sentido opuesto y tienen la misma velocidad. Realizar un programa para determinar en qué kilómetro de esa carretera se encontrarán.

Ejercicio 15

Una persona adquirió un producto para pagar en 20 meses. El primer mes pagó 10 €, el segundo 20 €, el tercero 40 € y así sucesivamente. Realizar un algoritmo para determinar cuánto debe pagar mensualmente y el total de lo que pagó después de los 20 meses.

Ejercicio 16

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y, además, calcule cuánto pagó la empresa por los N empleados.

Ejercicio 17

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Para esto, se registran los días que trabajó y las horas de cada día. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y además calcule cuánto pagó la empresa por los N empleados.

Ejercicio 18

Hacer un programa que muestre un cronometro, indicando las horas, minutos y segundos.

Ejercicio 19

Realizar un ejemplo de menú, donde podemos escoger las distintas opciones hasta que seleccionamos la opción de “Salir”.

Ejercicio 20

Mostrar en pantalla los N primero número primos. Se pide por teclado la cantidad de números primos que queremos mostrar.

Ejercicios cadena de caracteres

Ejercicio 1

Escribir por pantalla cada carácter de una cadena introducida por teclado.

Ejercicio 2

Realizar un programa que comprueba si una cadena leída por teclado comienza por una subcadena introducida por teclado.

Ejercicio 3

Pide una cadena y un carácter por teclado (valida que sea un carácter) y muestra cuantas veces aparece el carácter en la cadena.

Ejercicio 4

Suponiendo que hemos introducido una cadena por teclado que representa una frase (palabras separadas por espacios), realiza un programa que cuente cuantas palabras tiene.

Ejercicio 5

Si tenemos una cadena con un nombre y apellidos, realizar un programa que muestre las iniciales en mayúsculas.

Ejercicio 6

Realizar un programa que, dada una cadena de caracteres por caracteres, genere otra cadena resultado de invertir la primera.

Ejercicio 7

Pide una cadena y dos caracteres por teclado (valida que sea un carácter), sustituye la aparición del primer carácter en la cadena por el segundo carácter.

Ejercicio 8

Realizar un programa que lea una cadena por teclado y convierta las mayúsculas a minúsculas y viceversa.

Ejercicio 9

Realizar un programa que compruebe si una cadena **contiene** una subcadena. Las dos cadenas se introducen por teclado.

Ejercicio 10

Introducir una cadena de caracteres e indicar si es un palíndromo. Una palabra palíndroma es aquella que se lee igual adelante que atrás.

Estructuras de datos: Arreglos (array)

Estructura de datos

Hasta ahora, para hacer referencia a un dato utilizábamos una variable. El problema se plantea cuando tenemos gran cantidad de datos que guardan entre sí una relación. No podemos utilizar una variable para cada dato.

Para resolver estas dificultades se agrupan los datos en un mismo conjunto, estos conjuntos reciben el nombre de **estructura de datos**.

Arreglos o arrays

Un array (o arreglo) es una estructura de datos con elementos homogéneos, del mismo tipo, numérico o alfanumérico, reconocidos por un nombre en común. Para referirnos a cada elemento del array usaremos un índice (empezamos a contar por 0).

Declaración de arrays

Para declarar un array tenemos que ejecutar dos instrucciones:

1. En primer lugar, debemos declara el tipo de datos de la variable, con `Definir`.
2. Debemos indicar el número de elementos que va a tener el array, para ello utilizamos la instrucción `Dimensión`:

3. `Dimension <identificador> [<maxl>,...,<maxN>];`
   Esta instrucción define un arreglo con el nombre indicado en y N dimensiones. Los N parámetros indican la cantidad de dimensiones y el valor máximo de cada una de ellas. La cantidad de dimensiones puede ser una o más, y la máxima cantidad de elementos debe ser una expresión numérica positiva.

Por ejemplo, definimos un array de una dimensión (también llamado **vector**) de 10 elementos enteros.

```
Definir vector como Entero;
Dimension vector[10];
```

Otro ejemplo, definir una array de dos dimensiones (también llamado **matriz** o **tabla**) de 3 filas y cuatro columnas de cadenas.

```
Definir tabla como Cadenas;
Dimension tabla[3,4];
```

Para acceder a un elemento de un array se utiliza un índice. El primer elemento está en la posición 0.

Para asignar un valor a un elemento del vector:

```
vector[0]<-10;
```

Para mostrar el primer elemento del vector:

```
Escribir vector[0];
```
Otro ejemplo asignamos y mostramos el segundo elemento de la segunda fila de la tabla:

```
tabla[1,1] <- "Hola";
Escribir tabla[1,1];
```

Arreglos unidimensionales: Vectores

Un **vector** es una array unidimensional. Para declarar un vector de 10 enteros:

```
Definir vector como Entero;
Dimension vector[10];
```
Para acceder a cada uno de los elementos del vector utilizamos un índice. el primer elemento se accede con el índice 0. Podemos trabajar individualmente con cada uno de los elementos:

```
vector[0]<-10;
Escribir vector[0];
```
El acceso a un elemento que no existe producirá un error, por ejemplo:

```
vector[10]<-10;
```
Recorrido de un vector

Vamos a inicializar todos los elementos de un vector. Para ello vamos a **recorrer** el vector e inicializar cada elemento con un valor, por ejemplo, lo vamos a inicializar a 0. Para recorrer un vector utilizamos un bucle `Para`:

```
Para i<-0 hasta 9 Hacer
        array[i]<-0;
FinPara
```
Podríamos recorrer el vector para mostrar el valor de los elementos:

```
Para i<-0 hasta 9 Hacer
        Escribir array[i];
FinPara
```
Ejemplo

Inicializar un vector de 5 cadenas a partir de los datos pedidos por teclado y posterior mostrarlos en pantalla en mayúsculas.

```
Proceso VectorCadenas
        Definir i Como Entero;
        Definir vector Como Carácter;
        Dimension vector[5];
        Para i<-0 hasta 4 Hacer
                Escribir Sin Saltar "Dime la cadena número ",i+1,":";
                Leer vector[i];
        FinPara
        Escribir "Las cadenas en mayúsculas";
        Para i<-0 hasta 4 Hacer
                Escribir Sin Saltar Mayusculas(vector[i])," ";
        FinPara
FinProceso
```

Arreglos multidimensionales: Tablas

Una **tabla** en un array bidimensional. La primera dimensión indica el número de filas y el segundo el número de columnas.

```
Definir tabla como Entero;
Dimension tabla[3,4];
```

Hemos definido una tabla de enteras con 3 filas y 4 columnas, por tanto, tenemos 12 elementos.

Para acceder a cada uno de los elementos tenemos que indicar la fila y la columna en la que se encuentra, siempre empezando por el 0. Por ejemplo, para inicializar el elemento que está en la primera fila y la segunda columna sería:

```
tabla[0,1] <- 10;
```

El acceso a un elemento que no existe producirá un error.

Recorrido de una tabla

Para recorrer todos los elementos de una tabla necesitamos utilizar dos bucles anidados. Normalmente el exterior nos va a permitir recorrer las filas y el interior las columnas. Por ejemplo, para inicializar todos los elementos a 0, quedaría:

```
Para filas<-0 hasta 2 Hacer
	Para columnas<-0 hasta 3 Hacer
		tabla[filas,columnas]<-0;
	FinPara
FinPara
```

De forma similar podríamos recorrer la tabla para mostrar los elementos:

```
Para filas<-0 hasta 2 Hacer
	Para columnas<-0 hasta 3 Hacer
		Escribir tabla[filas,columnas];
	FinPara
FinPara
```

Ejemplo

Inicializar una tabla con los números del 1 al 5, sus cuadrados y sus cubos. Por lo tanto, tenemos que definir una tabla con 5 filas y 3 columnas. Muestra los datos:

```
Proceso CuadradoCubos
	Definir tabla Como Entero;
	Definir filas,columnas Como Entero;
	Dimension tabla[5,3];
	Para filas<-0 hasta 4 Hacer
		tabla[filas,0]<-filas+1;
		tabla[filas,1]<-(filas+1)^2;
		tabla[filas,2]<-(filas+1)^3;
	FinPara

	Para filas<-0 hasta 4 Hacer
		Para columnas<-0 hasta 2 Hacer
			Escribir Sin Saltar tabla[filas,columnas]," ";
		FinPara
		Escribir "";
```

```
        FinPara
FinProceso
```

Arrays multidimensionales

Los arrays pueden ter las dimensiones que deseemos, por ejemplo, podemos tener una array de tres dimensiones:

```
Definir tabla como Entero;
Dimension tabla[4,4,4];
```

Y podríamos inicializar el primer elemento como:

```
tabla[0,0,0]<-10;
```

Necesitaríamos tres bucles para recorrer un array de tres dimensiones:

```
Para i<-0 hasta 2 Hacer
        Para j<-0 hasta 2 Hacer
                Para k<-0 hasta 2 Hacer
                        tabla[i,j,k]<-0;
                FinPara
        FinPara
FinPara
```

Ejercicios de arreglos

Ejercicio 1

Realizar un programa que defina un vector llamado “vector_numeros” de 10 enteros, a continuación, lo inicialice con valores aleatorios (del 1 al 10) y posteriormente muestre en pantalla cada elemento del vector junto con su cuadrado y su cubo.

Ejercicio 2

Crear un vector de 5 elementos de cadenas de caracteres, inicializa el vector con datos leídos por el teclado. Copia los elementos del vector en otro vector, pero en orden inverso, y muéstralo por la pantalla.

Ejercicio 3

Se quiere realizar un programa que lea por teclado las 5 notas obtenidas por un alumno (comprendidas entre 0 y 10). A continuación, debe mostrar todas las notas, la nota media, la nota más alta que ha sacado y la menor.

Ejercicio 4

Programa que declare un vector de diez elementos enteros y pida números para rellenarlo hasta que se llene el vector o se introduzca un número negativo. Entonces se debe imprimir el vector (sólo los elementos introducidos).

Ejercicio 5

Hacer un programa que inicialice un vector de números con valores aleatorios, y posterior ordene los elementos de menor a mayor.

Ejercicio 6

Crea un programa que pida un número al usuario un número de mes (por ejemplo, el 4) y diga cuántos días tiene (por ejemplo, 30) y el nombre del mes. Debes usar un vector. Para simplificarlo vamos a suponer que febrero tiene 28 días.

Ejercicio 7

Programa que declare tres vectores ‘vector1’, ‘vector2’ y ‘vector3’ de cinco enteros cada uno, pida valores para ‘vector1’ y ‘vector2’ y calcule vector3=vector1+vector2.

Ejercicio 8

Queremos guardar los nombres y las edades de los alumnos de un curso. Realiza un programa que introduzca el nombre y la edad de cada alumno. El proceso de lectura de datos terminará cuando se introduzca como nombre un asterisco (*) Al finalizar se mostrará los siguientes datos:

- Todos los alumnos mayores de edad.
- Los alumnos mayores (los que tienen más edad)

Ejercicio 9

Queremos guardar la temperatura mínima y máxima de 5 días. realiza un programa que de la siguiente información:

- La temperatura media de cada día
- Los días con menos temperatura
- Se lee una temperatura por teclado y se muestran los días cuya temperatura máxima coincide con ella. si no existe ningún día se muestra un mensaje de información.

Ejercicio 10

Diseñar el algoritmo correspondiente a un programa, que:

- Crea una tabla bidimensional de longitud 5x5 y nombre ‘matriz’.
- Carga la tabla con valores numéricos enteros.
- Suma todos los elementos de cada fila y todos los elementos de cada columna visualizando los resultados en pantalla.

Ejercicio 11

Diseñar el algoritmo correspondiente a un programa, que:

- Crea una tabla bidimensional de longitud 5x5 y nombre ‘diagonal’.
- Carga la tabla de forma que los componentes pertenecientes a la diagonal de la matriz tomen el valor 1 y el resto el valor 0.
- Muestra el contenido de la tabla en pantalla.

Ejercicio 12

Diseñar el algoritmo correspondiente a un programa, que:

- Crea una tabla bidimensional de longitud 5x15 y nombre ‘marco’.

- Carga la tabla con dos únicos valores 0 y 1, donde el valor uno ocupará las posiciones o elementos que delimitan la tabla, es decir, las más externas, mientras que el resto de los elementos contendrán el valor 0.

```
111111111111111
100000000000001
100000000000001
100000000000001
111111111111111
```

- Visualiza el contenido de la matriz en pantalla.

Ejercicio 13

De una empresa de transporte se quiere guardar el nombre de los conductores que tiene, y los kilómetros que conducen cada día de la semana.

Para guardar esta información se van a utilizar dos arreglos:

- Nombre: Vector para guardar los nombres de los conductores.
- kms: Tabla para guardar los kilómetros que realizan cada día de la semana.

Se quiere generar un nuevo vector ("total_kms") con los kilómetros totales que realza cada conductor.

Al finalizar se muestra la lista con los nombres de conductores y los kilómetros que ha realizado.

Ejercicio 14

Crear un programa que lea los precios de 5 artículos y las cantidades vendidas por una empresa en sus 4 sucursales. Informar:

- Las cantidades totales de cada artículo.
- La cantidad de artículos en la sucursal 2.
- La cantidad del articulo 3 en la sucursal 1.
- La recaudación total de cada sucursal.
- La recaudación total de la empresa.
- La sucursal de mayor recaudación.

Ejercicio 15

Crear un programa de ordenador para gestionar los resultados de la quiniela de fútbol. Para ello vamos a utilizar dos tablas:

- Equipos: Que es una tabla de cadenas donde guardamos en cada columna el nombre de los equipos de cada partido. En la quiniela se indican 15 partidos.
- Resultados: Es una tabla de enteros donde se indica el resultado. También tiene dos columnas, en la primera se guarda el número de goles del equipo que está guardado en la primera columna de la tabla anterior, y en la segunda los goles del otro equipo.

El programa ira pidiendo los nombres de los equipos de cada partido y el resultado del partido, a continuación se imprimirá la quiniela de esa jornada.

¿Qué modificación habría que hacer en las tablas para guardar todos los resultados de todas las jornadas de la temporada?

Programación estructurada

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de ordenador o algoritmo, utilizando únicamente subrutinas (funciones o procedimientos) y tres estructuras: secuencia, alternativas y repetitivas.

La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún algoritmo (divide y vencerás). Además el uso de subrutinas nos proporciona la reutilización del código y no tener repetido instrucciones que realizan la misma tarea.

Subrutinas en pseudocódigo

Tenemos dos clases de subrutinas:

- **Funciones**: Subrutina que devuelve un valor.
- **Procedimientos**: Subrutina que no devuelve ningún resultado.

Para definir una función o procedimiento:

```
Funcion [variable_de_retorno <-] nombre_de_la_funcion ( argumento_1, argumento_2, ... )
      acción 1;
      acción 1;
            .
            .
            .
      acción n;
 FinFuncion
```

- Comienza con la palabra clave `Funcion` (alternativamente puede utilizar `SubProceso` o `SubAlgoritmo`, son sinónimos) seguida de la variable de retorno, el signo de asignación, el nombre del subproceso, y finalmente, la lista de argumentos entre paréntesis.
- Si el subproceso no recibe ningún valor pueden colocarse los paréntesis vacíos u omitirse, finalizando la primera línea con el nombre del subproceso.
- Si estamos trabajando con un procedimiento se coloca directamente el nombre y los argumentos a continuación de la palabra clave `Funcion`.

Ejemplo de función

Vamos a crear una función que calcule el valor máximo de dos números:

```
Funcion max <- CalcularMaximo(num1,num2)
        Definir max Como Entero;
        Si num1>num2 Entonces
                max <- num1;
        SiNo
                max <- num2;
        FinSi
FinFuncion
```

Como vemos la variable que se devuelve `max` hay que definirla en la función.

Para utilizar dicha función en el programa principal:

```
Proceso Maximo
        Definir numero1,numero2,num_maximo Como Entero;
        Escribir "Dime el número1:";
        Leer numero1;
        Escribir "Dime el número2:";
        Leer numero2;
        num_maximo <- CalcularMaximo(numero1,numero2);
        Escribir "El máximo es ",num_maximo;
FinProceso
```

Ejemplo de procedimiento

Vamos a escribir un procedimiento que recibe una cadena de caracteres y lo muestra en pantalla subrayado. No devuelve ningún valor.

```
Funcion Subrayar(cad)
        Definir i Como Entero;
        Escribir cad;
        Para i<-1 hasta Longitud(cad) hacer
                Escribir Sin Saltar "-";
        FinPara
FinFuncion

Proceso Titulos
        Definir titulo como cadena;
        titulo <- "Ejercicio 1";
        Subrayar(titulo);
        Escribir "";
FinProceso
```

Funciones y procedimientos

Partimos del ejemplo anterior de función:

```
Funcion max <- CalcularMaximo(num1,num2)
        Definir max Como Entero;
        Si num1>num2 Entonces
                max <- num1;
        SiNo
                max <- num2;
```

```
        FinSi
FinFuncion

Proceso Maximo
        Definir numero1,numero2,num_maximo Como Entero;
        Escribir "Dime el número1:";
        Leer numero1;
        Escribir "Dime el número2:";
        Leer numero2;
        num_maximo <- CalcularMaximo(numero1,numero2);
        Escribir "El máximo es ",num_maximo;
FinProceso
```

Ámbito de variables

Las variables definidas en la función no existen en otras funciones o el programa principal. Igualmente, las variables del programa principal no existen en la función.

Por ejemplo, la variable `max` definida en la función no existe en el programa principalmente. Igualmente, la variable `numero1` definida en el programa principal no existe en la función.

Parámetros formales y reales

- **Parámetros formales**: Son las variables que recibe la función, se crean al definir la función. Su contenido lo recibe al realizar la llamada a la función de los parámetros reales. Los parámetros formales son variables locales dentro de la función.

- **Parámetros reales**: Son las expresiones que se utilizan en la llamada de la función, sus valores se copiarán en los parámetros formales.

Paso de parámetro por valor o por referencia

- **Paso por valor**: El valor de los parámetros reales se copian en los parámetros formales, por lo tanto, una modificación de algún parámetro formal no modifica el parámetro real.
- **Paso por referencia**: Cuando se pasa un parámetro por referencia implica que si modificamos el parámetro formal se modificará el parámetro real.

Pode defecto, los arreglos se pasan por referencia, las demás expresiones por valor. Si queremos indicar explícitamente como se pasan los parámetros podemos usar las palabras claves `Por Valor` o `Por Referencia`.

Ejemplos

Comprobamos que los parámetros pasados por valor no modifican los parámetros reales.

```
Funcion PasoPorValor(num)
        num <- num +1;
        Escribir num;
FinFuncion

Proceso Prueba
        Definir numero1 Como Entero;
```

```
        numero1<-5;
        PasoPorValor(numero1);
        Escribir numero1;
FinProceso
```
El resultado será 5 y 6. Hemos incrementado el valor del parámetro formal, pero no se ha modificado el real.

Veamos ahora el mismo programa, pero pasando el parámetro por referencia.

```
Funcion PasoPorReferencia(num Por Referencia)
        num <- num +1;
        Escribir num;
FinFuncion


Proceso Prueba
        Definir numero1 Como Entero;
        numero1<-5;
        PasoPorReferencia(numero1);
        Escribir numero1;
FinProceso
```
El resultado será 6 y 6. Hemos modificado el parámetro formal y se modificado el real.

Llamada a la función

Para llamar a una función se debe utilizar su nombre y entre paréntesis los parámetros reales que se mandan. La llamada a una función se puede considerar una expresión cuyo valor y tipo es el retornado por la función. Evidentemente si estamos llamando un procedimiento, la llamada no tendrá ningún tipo.

Ejemplos de llamadas:

```
num1 <- CalcularMaximo (5,6)
Escribir CalcularMaximo (1,2)
...
```

Funciones recursivas

Una función recursiva es aquella que al ejecutarse hace llamadas a ella misma. Por lo tanto, tenemos que tener "un caso base" que hace terminar el bucle de llamadas. Veamos un ejemplo:

```
Funcion fact <- CalcularFactorial(num)
        Definir fact Como Entero;
        Si num=0 O num=1 Entonces
                fact <- 1;
        SiNo
                fact <- num * CalcularFactorial(num-1);
        FinSi
FinFuncion

Proceso ProgramaPrincipal
        Escribir "El factorial de 6 es ",CalcularFactorial(6);
FinProceso
```

Ejercicios de funciones

Ejercicio 1

Crea un procedimiento EscribirCentrado, que reciba como parámetro un texto y lo escriba centrado en pantalla (suponiendo una anchura de 80 columnas; pista: deberás escribir 40 - longitud/2 espacios antes del texto). Además, subraya el mensaje utilizando el carácter =.

Ejercicio 2

Crea un programa que pida dos números enteros al usuario y diga si alguno de ellos es múltiplo del otro. Crea una función EsMultiplo que reciba los dos números, y devuelve si el primero es múltiplo del segundo.

Ejercicio 3

Crear una función que calcule la temperatura media de un día a partir de la temperatura máxima y mínima. Crear un programa principal, que, utilizando la función anterior, vaya pidiendo la temperatura máxima y mínima de cada día y vaya mostrando la media. El programa pedirá el número de días que se van a introducir.

Ejercicio 4

Crea una función "ConvertirEspaciado", que reciba como parámetro un texto y devuelve una cadena con un espacio adicional tras cada letra. Por ejemplo, "Hola, tú" devolverá "H o l a , t ú ". Crea un programa principal donde se use dicha función.

Ejercicio 5

Crea una función "calcularMaxMin" que recibe un arreglo con valores numérico y devuelve el valor máximo y el mínimo. Crea un programa que pida números por teclado y muestre el máximo y el mínimo, utilizando la función anterior.

Ejercicio 6

Diseñar una función que calcule el área y el perímetro de una circunferencia. Utiliza dicha función en un programa principal que lea el radio de una circunferencia y muestre su área y perímetro.

Ejercicio 7

Crear una subrutina llamada "Login", que recibe un nombre de usuario y una contraseña y te devuelve Verdadero si el nombre de usuario es "usuario1" y la contraseña es "asdasd". Además, recibe el número de intentos que se ha intentado hacer login y si no se ha podido hacer login incremente este valor.

Crear un programa principal donde se pida un nombre de usuario y una contraseña y se intente hacer login, solamente tenemos tres oportunidades para intentarlo.

Ejercicio 8

Crear una función recursiva que permita calcular el factorial de un número. Realiza un programa principal donde se lea un entero y se muestre el resultado del factorial.

Ejercicio 9

Crear una función que calcule el MCD de dos números por el método de Euclides. El método de Euclides es el siguiente:

- Se divide el número mayor entre el menor.
- Si la división es exacta, el divisor es el MCD.
- Si la división no es exacta, dividimos el divisor entre el resto obtenido y se continúa de esta forma hasta obtener una división exacta, siendo el último divisor el MCD.

Crea un programa principal que lea dos números enteros y muestre el MCD.

Ejercicio 10

Escribir dos funciones que permitan calcular:

- La cantidad de segundos en un tiempo dado en horas, minutos y segundos.
- La cantidad de horas, minutos y segundos de un tiempo dado en segundos.

Escribe un programa principal con un menú donde se pueda elegir la opción de convertir a segundos, convertir a horas, minutos y segundos o salir del programa.

Ejercicio 11

El día juliano correspondiente a una fecha es un número entero que indica los días que han transcurrido desde el 1 de enero del año indicado. Queremos crear un programa principal que al introducir una fecha nos diga el día juliano que corresponde. Para ello podemos hacer las siguientes subrutinas:

- `LeerFecha`: Nos permite leer por teclado una fecha (día, mes y año).
- `DiasDelMes`: Recibe un mes y un año y nos dice los días de ese mes en ese año.
- `EsBisiesto`: Recibe un año y nos dice si es bisiesto.
- `Calcular_Dia_Juliano`: recibe una fecha y nos devuelve el día juliano.

Ejercicio 12

Vamos a mejorar el ejercicio anterior haciendo una función para validar la fecha. De tal forma que al leer una fecha se asegura que es válida.

Ejercicio 13

Queremos crear un programa que trabaje con fracciones a/b. Para representar una fracción vamos a utilizar dos enteros: `numerador` y `denominador`.

Vamos a crear las siguientes funciones para trabajar con funciones:

- `Leer_fracción`: La tarea de esta función es leer por teclado el numerador y el denominador. Cuando leas una fracción debes simplificarla.

- `Escribir_fracción`: Esta función escribe en pantalla la fracción. Si el dominador es 1, se muestra sólo el numerador.
- `Calcular_mcd`: Esta función recibe dos números y devuelve el máximo común divisor.
- `Simplificar_fracción`: Esta función simplifica la fracción, para ello hay que dividir numerador y dominador por el MCD del numerador y denominador.
- `Sumar_fracciones`: Función que recibe dos funciones n1/d1 y n2/d2, y calcula la suma de las dos fracciones. La suma de dos fracciones es otra fracción cuyo `numerador=n1*d2+d1*n2` y `denominador=d1*d2`. Se debe simplificar la fracción resultado.
- `Restar_fracciones`: Función que resta dos fracciones: `numerador=n1*d2-d1*n2` y `denominador=d1*d2`. Se debe simplificar la fracción resultado.
- `Multiplicar_fracciones`: Función que recibe dos fracciones y calcula el producto, para ello `numerador=n1*n2` y `denominador=d1*d2`. Se debe simplificar la fracción resultado.
- `Dividir_fracciones`: Función que recibe dos fracciones y calcula el cociente, para ello `numerador=n1*d2` y `denominador=d1*n2`. Se debe simplificar la fracción resultado.

Crear un programa que utilizando las funciones anteriores muestre el siguiente menú:

1. Sumar dos fracciones: En esta opción se piden dos fracciones y se muestra el resultado.
2. Restar dos fracciones: En esta opción se piden dos fracciones y se muestra la resta.
3. Multiplicar dos fracciones: En esta opción se piden dos fracciones y se muestra el producto.
4. Dividir dos fracciones: En esta opción se piden dos fracciones y se muestra el cociente.
5. Salir

Ejercicio 14

Vamos a crear un programa para trabajar con una pila. Una pila es una estructura de datos que nos permite guardar un conjunto de variables. La característica fundamental es que el último elemento que se añade al conjunto es el primero que se puede sacar.

Para representar una pila vamos a utilizar un arreglo (vector) de cadena de caracteres con tamaño 10, por lo tanto, la pila no podrá tener más de 10 elementos.

Vamos a crear varias funciones para trabajar con la pila:

- `IncializarPila`: Como tenemos un arreglo de 10 elementos de cadenas tenemos que inicializarlo e introducir un carácter (por ejemplo, un `*` que indique que ese elemento del arreglo no corresponde con un dato de la pila. Esta función inicializa el vector con ese carácter.
- `LongitudPila`: Función que recibe una pila y devuelve el número de elementos que tiene.
- `EstaVaciaPila`: Función que recibe una pila y que devuelve si la pila está vacía, no tiene elementos.
- `EstaLlenaPila`: Función que recibe una pila y que devuelve si la pila está llena.

- `AddPila`: función que recibe una cadena de caracteres y una pila, y añade la cadena a la pila, si no está llena. si está llena muestra un mensaje de error.
- `SacarDeLaPila`: Función que recibe una pila y devuelve el último elemento añadido y lo borra de la pila. Si la pila está vacía muestra un mensaje de error.
- `EscribirPila`: Función que recibe una pila y muestra en pantalla los elementos de la pila.

Realiza un programa principal que nos permita usar las funciones anteriores, que nos muestre un menú, con las siguientes opciones:

1. Añadir elemento a la pila
2. Sacar elemento de la pila
3. Longitud de la pila
4. Mostrar pila
5. Salir

Ejercicio 15

Vamos a realizar un programa similar al anterior para trabajar con una cola. Una cola es una estructura de datos que nos permite guardar un conjunto de variables. La característica fundamental es que el primer elemento que se añade al conjunto es el primero que se puede sacar.

En realizada nos sirven todas las funciones del ejercicio anterior menos la función `SacarDeLaCola` que es la que tienes que modificar.

Más Ejercicios

Ejercicio 1

Realice un programa que pregunte aleatóriamente una multiplicación. El programa debe indicar si la respuesta ha sido correcta o no (en caso que la respuesta sea incorrecta el programa debe indicar cuál es la correcta). El programa preguntará 10 multiplicaciones, y al finalizar mostrará el número de aciertos.

Ejercicio 2

El DNI (Documento Nacional de Identidad) en España está formada por 8 números y una letra. La letra nos sirve para verificar que el número es correcto, por lo tanto, la letra se calcula a partir del número. Busca información de cómo se realiza el cálculo y crea una función `CalcularLetra` que recibe un número y devuelva la letra que le corresponde.

La función anterior la podemos utilizar para crear una nueva función `ValidarDNI` que recibe un DNI (cadena de caracteres con 8 números y una letra) que, valida el DNI, es decir comprueba si la letra del DNI ces igual a la letra calculada a partir del número.

Realiza un programa principal que lea un DNI y valide que es correcto (se debe comprobar también que tiene 9 caracteres).

Ejercicio 3

Realizar una aplicación que recoja por teclado la cantidad total a pagar y la cantidad que se ha entregado. La aplicación debe calcular el cambio correspondiente con el menor número de monedas y/o billetes posibles.

Ejercicio 4

Realizar un algoritmo que permita descomponer un número en sus factores primos.

Ejercicio 5

Vamos a realizar dos funciones: una que nos permita convertir un número entero a binario, y otra que nos permita convertir un numero binario a decimal.

- `ConvertirABinario`: Función que recibe un número entero y devuelve una cadena con la representación del número en binario.
- `ConvertirADecimal`: Función que recibe una cadena con la representación binaria de un número y devuelve el número en decimal.

Crea un programa principal que permita convertir de decimal a binario y de binario a decimal.

Ejercicio 6

Crear un programa que convierta un número entero (mayor que 1 y menor o igual que 1000) a número romano.

Ejercicio 7

Diseñar un programa que permita adivinar al ordenador un determinado número entero y positivo para lo cual se deben leer los límites en los que está comprendido dicho número. El programa deberá ir mostrando números que recibirán las siguientes respuestas:

1. 'S', si es correcto.
2. 'A', si es más alto que el número a adivinar.
3. 'B', si es más bajo. Al finalizar el programa, se deberá escribir el número de intentos realizados para acertar el número.

Ejercicio 8

Realizar un programa que pida un mes y un año (superior a 1900) y muestre el calendario del mes de esta manera:

```
 L   M   M   J   V   S   D
=========================
     1   2   3   4   5   6
 7   8   9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30  31
```

Para ello es necesario averiguar qué día de la semana (lunes, martes, …) corresponde con una fecha determinada. Hay muchas maneras de calcularlo: nosotros vamos a contar los días que han trascurridos desde el año 1900 (podemos hacer uso de funciones que hemos utilizado en ejercicios anteriores), y una vez calculado le hacemos el módulo 7 y el

número obtenido será el da de la semana (0: domingo, 1: lunes, …) (NOTA: ten en cuenta que queremos realizar un calendario que empiece en lunes, no en domingo).

Ejercicio 9

Vamos a programar el juego "Mastermind", para ello el programa debe "elegir" un número de cuatro cifras (sin cifras repetidas), que será el código que el jugador debe adivinar en la menor cantidad de intentos posibles. Cada intento consiste en una propuesta de un código posible que escribe el jugador, y una respuesta del programa. Las respuestas le darán pistas al jugador para que pueda deducir el código.

- Número de "MUERTOS": Es la cantidad de dígitos que están en el número secreto y en la misma posición,
- Número de "HERIDOS:" Es la cantidad de dígitos que están en el número secreto, pero no en la misma posición.

Por ejemplo, si el código que eligió el programa es el 2607, y el jugador propone el 1406, el programa le debe responder un MUERTO (el 0, que está en el código original en el mismo lugar, el tercero), y un HERIDO (el 6, que también está en el código original, pero en la segunda posición, no en el cuarto como fue propuesto).

Ejercicio 10

Escribe un programa para jugar al ahorcado.

- Un jugador introduce una palabra secreta y otro jugador tratará de adivinarla.
- Aparecerá la palabra oculta (se mostrará un conjunto de asteriscos con la longitud de la palabra que hay que adivinar).
- El programa te va pidiendo letras.
- Si la letra está en la palabra, se mostrar la palabra mostrando las letras acertadas y los asteriscos en las letras que faltan por averiguar.
- Cada vez que se introduce una letra se muestra las letras que has introducido anteriormente.
- Si la letra no se encuentra en la palabra se suma un fallo. Según el número de fallos se mostrará el dibujo del ahorcado cada vez ms completo.
- Si introduces todas las letras de la palabra has ganada y te mostrará el número de intentos que has necesitado.
- Si produces 6 fallos, habrás perdido y se mostrará el dibujo del ahorcado completo.

Introducción a los lenguajes de programación

Sistemas Informáticos

Cuando el procesamiento de los datos y la información en un **Sistema de Información** lo realiza un ordenador lo llamamos **Sistema Informático**.

En un Sistema Informático el algoritmo se describe mediante un programa.

- **Programa**: Conjunto ordenado de instrucciones que se dan al ordenador indicándole las operaciones o tareas que ha de realizar para resolver un problema.

Para escribir programas utilizamos **lenguajes de programación**.

Etapas del desarrollo de programas

- Análisis: Entender el problema.
- Diseño: Creamos el algoritmo: pseudocódigo.
- **Codificación**: Escribir el algoritmo en un lenguaje de programación (**Código fuente**).
- **Ejecución y validación**: Comprobamos que el programa resuelve el problema planteado.

Lenguajes de programación

Para que un ordenador realice un proceso, se le debe suministrar en primer lugar un algoritmo adecuado, que llamamos **programa**. El procesador debe ser capaz de interpretar el algoritmo, lo que significa:

- Comprender las instrucciones de cada paso.
- Realizar las operaciones correspondientes.

Se pueden clasificar los lenguajes (de forma tradicional) en:

Lenguaje Máquina

Son aquellos que están escritos en lenguajes que directamente entiende el ordenador, ya que sus instrucciones son cadenas binarias (secuencias de ceros y unos) que especifican una operación y las posiciones (dirección) de memoria implicadas en la operación. Se denominan instrucciones de máquina o **código máquina**. Características:

- Las instrucciones en lenguaje máquina dependen del hardware del ordenador y por tanto serán diferentes de un ordenador a otro.
- Se puede transferir un programa a memoria sin necesidad de traducción posterior, lo que supone una mayor velocidad de ejecución a cualquier otro lenguaje.
- Dificultad y lentitud en la codificación.

- Conjunto de instrucciones reducido (operaciones muy elementales)

Lenguaje de bajo nivel (Ensamblador)

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el **ensamblador**.

Las instrucciones en lenguaje ensamblador son instrucciones conocidas como mnemónicos.

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la máquina, sino que requiere una fase de traducción al lenguaje máquina. El programa original escrito en lenguaje ensamblador se denomina **programa fuente** y el programa traducido en lenguaje máquina se conoce como **programa objeto**, ya directamente entendible por el ordenador.

Lenguaje de alto nivel

Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en un lenguaje de alto nivel es independiente de la máquina, o sea, las instrucciones del programa del ordenador no dependen del diseño hardware de un ordenador en particular. Por lo tanto, los programas escritos en lenguajes de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de ordenadores.

Al igual que sucede con los lenguajes ensambladores, los programas fuente tienen que ser traducidos por **programas traductores**, llamados en este caso compiladores o intérpretes.

Ejemplos de lenguajes de programación de alto nivel:

BASIC, COBOL, PASCAL, C, VISUAL BASIC, JAVA, PYTHON, PERL, GO, PHP, RUBY,…

Programas traductores

Los traductores transforman programas escritos en un lenguaje de alto nivel en programas escritos en código máquina. Podemos indicar distintos tipos:

Compiladores

- Convierte un programa escrito en alto nivel (código fuente) a un programa máquina (código ejecutable).
- Para generar el código ejecutable el código no debe tener errores de sintaxis.

- Necesitamos un compilador para cada arquitectura y sistema operativo.
- Los programas ejecutables no son compatibles entre plataformas.
- Una vez generado el programa ejecutable, no es necesario tener el código fuente.
- Ejemplos: C, Pascal, …

Interpretes

- La traducción y ejecución de código fuente a código máquina se hace línea por línea.
- Los errores de sintaxis aparecen cuando se interpreta la instrucción con error.
- Necesitamos el código fuente para ejecutar el programa.
- Los lenguajes interpretados suelen ser más lentos en su ejecución
- Ejemplos: Python, PHP, …

Máquina virtual

- La traducción se hace en dos pasos.
- Primero se compila el código fuente a un código intermedio (bytecode).
- Segundo, este bytecode se interpreta y ejecuta por una “máquina virtual”.
- El bytecode es multiplataforma.
- Necesito una “máquina virtual” para cada plataforma.
- No necesito el código fuente.
- Ejemplo: Java, C#, …

Compilación y ejecución de un lenguaje compilado: C++

Lo primero que vamos a hacer es escribir el **código fuente**. Para ello vamos a utilizar un **IDE** (Entorno de desarrollo integrado) llamado Zinjai utilizando el sistema operativo Linux. El fichero `adivina.cpp` tendría el siguiente contenido:

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main(int argc, char *argv[]) {
	int num_secreto,numero,intentos;
	intentos=1;
	num_secreto=rand() % 100;
	cout << "Introduce un número:";
	cin >> numero;
	while (numero!=num_secreto)
	{
		if(numero>num_secreto)
		{
```

```
                        cout << "El número introducido es mayor\n";
                }
                else
                {
                        cout << "El número introducido es menor\n";
                }
                intentos++;
                cout << "Introduce un número:";
                cin >> numero;
        }
        cout << "Lo has acertado en " << intentos << " intentos.";
        return 0;
}
```

Cuando compilamos y ejecutamos nuestro código fuente se genera un programa ejecutable llamado `adivina.bin`. Por lo tanto, desde la línea de comandos lo podemos ejecutar:

```
$ ./adivina.bin
Introduce un número:
...
```

Podemos verificar que `adivina.bin` es un fichero binario para linux 64-bit:

```
$ file adivina.bin
adivina.bin: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=007d6a8507f270e6864af6243e127b4128f72277, not stripped
```

Realmente la compilación incluye varios pasos, el más importante de ellos es la creación de un fichero objeto intermedio. Que a continuación se enlaza y producir el fichero ejecutable.

¿Qué sentido tiene compilar por partes? Cuando el código es grande no hay un solo fichero fuente sino muchos, compilar individualmente estos “módulos” permite, por ejemplo, ahorrar mucho tiempo en la modificación y compilación de un solo componente.

A continuación, vamos a intentar ejecutar nuestro fichero ejecutable en Windows para comprobar que no es posible: el fichero ejecutable depende de la arquitectura (tipo de procesador, sistema operativo,) donde se ha compilado.

Podemos probar a continuación a compilar el mismo programa en Windows e intentar ejecutarlo en Linux, y volveremos a comprobar que no es posible.

Compilación e interpretación de un programa Java

A la hora de instalar Java tenemos que tener en cuenta que tenemos dos componentes diferentes:

- **Java Runtime Environment (JRE)** son el conjunto de aplicaciones que se instalan en un equipo para que puedan ejecutarse en él aplicaciones Java. Los dos componentes principales de un JRE son:

    - Java Virtual Machine: Aplicación que ejecuta el código java en bytecode y que está adaptada a la plataforma sobre la que opera.
    - Bibliotecas Java

- **Java Development Kit (JDK)** son el conjunto de programas para desarrollar aplicaciones y entre otros incluye el compilador javac que convierte un programa fuente java a bytecode. Al instalar el JDK se instala también el componente JRE.

Instalación de JDK

Existen diferentes implantaciones de JDK (la actual es la 10), siendo la versión recomendable para Linux la 1.8. Actualmente el propietario de Java es la empresa Oracle y ha modificado la antigua licencia libre de Java, por lo que ya no es posible que se distribuya legalmente en las distribuciones de software libre. Nosotros optaremos por utilizar OpenJDK, que es una implementación libre de Java. En una distribución Linux Stretch o Ubuntu 16.04, instalamos el siguiente paquete:

```
$ apt-get install openjdk-8-jdk
```

En Windows tendríamos que bajarnos el instalador desde la página de descargas de Java e instalarlo.

Compilación de un programa Java

Utilizando nuestro editor de texto favorito vamos a crear el fichero `Adivina.java` con el siguiente contenido:

```
import java.util.Scanner;
public class Adivina {
     public static void main(String[] args) {
        int num_secreto,numero,intentos;
        Scanner sc = new Scanner(System.in);
                intentos=1;
                num_secreto=(int) (Math.random()*100+1);
                System.out.println("Introduce un número:");
                numero= sc.nextInt();
                while (numero!=num_secreto)
                {
                        if(numero>num_secreto)
                        {
                                System.out.println("El            número
introducido es mayor\n");
                        }
                        else
                        {
                                System.out.println("El            número
introducido es menor\n");
                        }
                        intentos++;
                        System.out.println("Introduce un número:");
                        numero= sc.nextInt();
                }
                System.out.println("Lo  has  acertado  en  "+intentos  +"
intentos.");
     }
}
```

A continuación, compilamos la aplicación:

```
$ javac Adivina.java
```

y se creará un fichero bytecode `Adivina.class`. Este fichero es portable, por lo tanto, podemos ejecutarlo tanto en Linux como en Windows con la siguiente instrucción:

```
java Adivina
Introduce un número:
...
```

Ejecución de programas interpretados con Python

Un Lenguaje interpretado es aquel cuyo código no necesita ser preprocesado mediante un compilador, eso significa que el ordenador es capaz de ejecutar la sucesión de instrucciones dadas por el programador sin necesidad de leer y traducir exhaustivamente todo el código.

Por lo tanto, necesitamos el código fuentes para ejecutar el programa utilizando un intérprete que tendremos instalado en nuestras distintas plataformas (Linux, Windows, …)

Instalación de Python3

Python es un lenguaje interpretado, que está instalado por defecto en la mayoría de las distribuciones Linux. Para instalarlo en Windows puedes seguir las instrucciones que encontrarás en su página de descarga.

Ejecución de un programa Python3

Creamos el fichero `adivina.py` con el siguiente contenido:

```
#!/usr/bin/env python
import random

a=random.randrange(0, 100)
intentos=1
b=int(input("Introduce un número:"))
while a!=b:
        if b>a:
                print("El número introducido es mayor")
        else:
                print("El número introducido es menor")
        intentos=intentos+1
        b=int(input("Introduce un número:"))
print("Has acertado en %d intentos." % intentos)
```

Para ejecutar el código fuente, desde cualquier sistema operativo donde tengamos el intérprete de python, ejecutamos la siguiente instrucción:

```
    $ python3 adivina.py
    Introduce un número:
```