0. Conceptos básicos sobre programación

Un **programa** es un conjunto de órdenes para un ordenador.

Estas órdenes se le deben dar en un cierto **lenguaje,** que el ordenador sea capaz de comprender.

El problema es que los lenguajes que realmente entienden los ordenadores resultan difíciles para nosotros, porque son muy distintos de los que nosotros empleamos habitualmente para hablar. Escribir programas en el lenguaje que utiliza internamente el ordenador (llamado "**lenguaje máquina**" o "código máquina") es un trabajo duro, tanto a la hora de crear el programa como (especialmente) en el momento de corregir algún fallo o mejorar lo que se hizo.

Por eso, en la práctica se emplean lenguajes más parecidos al lenguaje humano, llamados "lenguajes de **alto nivel**". Normalmente, estos son muy parecidos al idioma inglés, aunque siguen unas reglas mucho más estrictas.

0.1. Lenguajes de alto nivel y de bajo nivel.

Vamos a ver en primer lugar algún ejemplo de lenguaje de alto nivel, para después comparar con lenguajes de bajo nivel, que son los más cercanos al ordenador.

Uno de los lenguajes de alto nivel más sencillos es el lenguaje BASIC. En este lenguaje, escribir el texto Hola en pantalla, sería tan sencillo como usar la orden

```
PRINT "Hola"
```

Otros lenguajes, como Pascal, nos obligan a ser algo más estrictos, pero a cambio hacen más fácil descubrir errores:

```
program Saludo;

begin
  write('Hola');
end.
```

El equivalente en lenguaje C resulta algo más difícil de leer, por motivos que iremos comentando un poco más adelante:

```
#include <stdio.h>

int main()
{
  printf("Hola");
}
```

Los lenguajes de bajo nivel son más cercanos al ordenador que a los lenguajes humanos. Eso hace que sean más difíciles de aprender y también que los fallos sean más difíciles de descubrir y corregir, a cambio de que podemos optimizar al máximo la velocidad (si sabemos cómo), e

incluso llegar a un nivel de control del ordenador que a veces no se puede alcanzar con otros lenguajes. Por ejemplo, escribir Hola en lenguaje ensamblador de un ordenador equipado con el sistema operativo MsDos y con un procesador de la familia Intel x86 sería algo como

```
dosseg
.model small
.stack 100h

.data
hello_message db 'Hola',0dh,0ah,'$'

.code
main  proc
      mov    ax,@data
      mov    ds,ax

      mov    ah,9
      mov    dx,offset hello_message
      int    21h

      mov    ax,4C00h
      int    21h
main  endp
end   main
```

Resulta bastante más difícil de seguir. Pero eso todavía no es lo que el ordenador entiende, aunque tiene una equivalencia casi directa. Lo que el ordenador realmente es capaz de comprender son secuencias de ceros y unos. Por ejemplo, las órdenes "mov ds, ax" y "mov ah, 9" (en cuyo significado no vamos a entrar) se convertirían a lo siguiente:

```
1000 0011 1101 1000 1011 0100 0000 1001
```

(Nota: los colores de los ejemplos anteriores son una ayuda que nos dan algunos entornos de programación, para que nos sea más fácil descubrir errores).

0.2. Ensambladores, compiladores e intérpretes

Está claro entonces que las órdenes que nosotros hemos escrito (lo que se conoce como "programa **fuente**") deben convertirse a lo que el ordenador comprende (obteniendo el "programa **ejecutable**").

Si elegimos un lenguaje de bajo nivel, como el ensamblador (en inglés *Assembly*, abreviado como *Asm*), la traducción es sencilla, y de hacer esa traducción se encargan unas herramientas llamadas **ensambladores** (en inglés *Assembler*).

Cuando el lenguaje que hemos empleado es de alto nivel, la traducción es más complicada, y a veces implicará también recopilar varios fuentes distintos o incluir posibilidades que se encuentran en bibliotecas que no hemos preparado nosotros. Las herramientas encargadas de todo esto son los **compiladores**.

El programa ejecutable obtenido con el compilador o el ensamblador se podría hacer funcionar en otro ordenador similar al que habíamos utilizado para crearlo, sin necesidad de que ese otro ordenador tenga instalado el compilador o el ensamblador.

Por ejemplo, en el caso de Windows (y de MsDos), y del programa que nos saluda en lenguaje Pascal, tendríamos un fichero fuente llamado SALUDO.PAS. Este fichero no serviría de nada en un ordenador que no tuviera un compilador de Pascal. En cambio, después de compilarlo obtendríamos un fichero SALUDO.EXE, capaz de funcionar en cualquier otro ordenador que tuviera el mismo sistema operativo, aunque no tenga un compilador de Pascal instalado.

Un **intérprete** es una herramienta parecida a un compilador, con la diferencia de que en los intérpretes no se crea ningún "programa ejecutable" capaz de funcionar "por sí solo", de modo que si queremos distribuir nuestro programa a alguien, deberemos entregarle el programa fuente y también el intérprete que es capaz de entenderlo, o no le servirá de nada. Cuando ponemos el programa en funcionamiento, el intérprete de encarga de convertir el programa en lenguaje de alto nivel a código máquina, orden por orden, justo en el momento en que hay que procesar cada una de las órdenes.

Para algunos lenguajes, es frecuente encontrar compiladores pero no suele existir intérpretes. Es el caso del lenguaje C, de Pascal y de C++, por ejemplo. En cambio, para otros lenguajes, lo habitual es trabajar con intérpretes y no con compiladores, como ocurre con Python, Ruby y PHP.

Además, hoy en día existe algo que parece intermedio entre un compilador y un intérprete: Existen lenguajes que no se compilan para obtener un ejecutable para un ordenador concreto, sino un ejecutable "genérico", que es capaz de funcionar en distintos tipos de ordenadores, a condición de que en ese ordenador exista una "**máquina virtual**" capaz de entender esos ejecutables genéricos. Esta es la idea que se aplica en Java: los fuentes son ficheros de texto, con extensión ".java", que se compilan a ficheros ".class=". Estos ficheros ".class=" se podrían llevar a cualquier ordenador que tenga instalada una "máquina virtual Java" (las hay para la mayoría de sistemas operativos). Esta misma idea se sigue en el lenguaje C#, que se apoya en una máquina virtual llamada "Dot Net Framework" (algo así como "armazón punto net").

0.3. Pseudocódigo

A pesar de que los lenguajes de alto nivel se acercan al lenguaje natural, que nosotros empleamos, es habitual no usar ningún lenguaje de programación concreto cuando queremos plantear los pasos necesarios para resolver un problema, sino emplear un lenguaje de programación ficticio, no tan estricto, muchas veces escrito incluso en español. Este lenguaje recibe el nombre de **pseudocódigo.**

Esa secuencia de pasos para resolver un problema es lo que se conoce como **algoritmo** (realmente hay alguna condición más, por ejemplo, debe ser un número finito de pasos). Por tanto, un programa de ordenador es un algoritmo expresado en un lenguaje de programación.

Por ejemplo, un algoritmo que controlase los pagos que se realizan en una tienda con tarjeta de crédito, escrito en pseudocódigo, podría ser:

```
Leer banda magnética de la tarjeta
Conectar con central de cobros
Si hay conexión y la tarjeta es correcta:
   Pedir código PIN
  Si el PIN es correcto
    Comprobar saldo_existente
    Si saldo_existente > importe_compra
      Aceptar la venta
      Descontar importe del saldo.
    Fin Si
  Fin Si
Fin Si
```

Ejercicios propuestos

1. Localizar en Internet el intérprete de Basic llamado Bywater Basic, en su versión para el sistema operativo que se esté utilizando y probar el primer programa de ejemplo que se ha visto en el apartado 0.1.
2. Localizar en Internet el compilador de Pascal llamado Free Pascal, en su versión para el sistema operativo que se esté utilizando, instalarlo y probar el segundo programa de ejemplo que se ha visto en el apartado 0.1.
3. Localizar un compilador de C para el sistema operativo que se esté utilizando (si es Linux o alguna otra versión de Unix, es fácil que se encuentre ya instalado) y probar el tercer programa de ejemplo que se ha visto en el apartado 0.1.