Índice

Introducción

¿Qué es programar?

Con el fin de exponer una noción de lo que es programar, veamos el siguiente ejemplo, disponemos de un robot que solamente atiende a las siguientes órdenes:

```
avanzar
parar
girar
esperar x segundos
```

Es posible formar una secuencia de órdenes de este tipo para que recorra un camino determinado. Si queremos indicarle al robot que se desplace desde donde está hasta un objetivo, debemos de algún modo 'decirle' lo que debe hacer.

1

ACTIVIDAD:

Usando el robot MakeBlock con el mando a distancia, llegar desde el final de la clase hasta el punto marcado en el suelo.

Cuando lo hacemos usando un mando a distancia, controlamos el comportamiento del robot pero no lo estamos programando.

Programar un robot sería diseñar un conjunto de órdenes que consigan nuestro objetivo, pero sin que sea necesaria nuestra intervención o supervisión mientras lo hace.

2

ACTIVIDAD:

Usando el programa mBlock, crear un programa que haga llegar al robot desde el final de la clase hasta el punto marcado en el suelo.

De manera informal podemos decir que programar un objeto es dejar escrito las órdenes que debe seguir para lograr un objetivo determinado.

Cuando en vez de un robot pensamos en un ordenador o un móvil, el problema inicial reside en comprender las órdenes que pueden recibir estos objetos. Las más inmediatas son las relativas a presentar algo por pantalla o solicitar un dato al usuario ya que resultan visibles, pero conforme vamos profundizando los conceptos se van haciendo cada vez más abstractos y complicados.

La clave para aprender a programar un ordenador es por tanto la capacidad de comprender y utilizar conceptos abstractos, así como de abstraer el comportamiento de una máquina.

Algoritmos

Tradicionalmente, los ordenadores se utilizaron para resolver problemas matemáticos. Por tanto, no es difícil comprender que las bases de la programación de ordenadores se asienten sobre conceptos puramente matemáticos.

El primero de ellos es el concepto de algoritmo. De manera informal, un **algoritmo** es un método para resolver un problema (definición wikipedia).

3

ACTIVIDAD:

Localizar diferentes métodos (algoritmos) de ordenación de listas y seguirlos para ordenar una serie de fichas numeradas. NOTA: Mínimo algoritmos de la burbuja (bubble sort) y ordenamiento rápido (quicksort).

Estos algoritmos de ordenación se diseñaron para que un ordenador pudiera ordenar listas de datos de forma eficaz, pero sin embargo nosotros podemos seguirlos sin problemas ya que normalmente se expresan en lenguaje natural. Para que el ordenador pueda seguir el algoritmo debemos expresarlo en un lenguaje que entienda la máquina.

El proceso de programar un algoritmo usando un lenguaje que entienda la máquina es la **implementación** de ese algoritmo.

El idioma de la máquina

Desde el algoritmo, expresado normalmente en lenguaje natural, hasta el lenguaje que comprende directamente la máquina hay mucha distancia.

El componente del ordenador que reconoce órdenes se denomina **unidad de control de proceso** o **CPU** por su acrónimo en inglés.

El lenguaje que reconoce la CPU se denomina **código máquina**. Existe un código máquina diferente por cada modelo o más bien arquitectura de CPU. Para hacernos una idea de cómo funciona vamos a optar por tomar de ejemplo una CPU bastante antigua, el famoso z80.

Para la mente humana, trabajar directamente con el código máquina es bastante complejo porque como su nombre indica, consiste en una codificación de las operaciones que puede llevar a cabo la CPU. Para subsanar en parte esta dificultad se usan una serie de códigos nemotécnicos que reciben el nombre de **lenguaje ensamblador**.

4

ACTIVIDAD:

Ejecutar el siguiente código ensamblador del z80 en el simulador de lenguajes ensambladores y descubrir qué hace:

```
ld a,0
ld b,5
bucle: adc a,b
dec b
jr nz, bucle
```

Simulador de lenguaje ensamblador: asm80.com

Instrucciones para el simulador: Copiar el código, guardar (save) el código con el nombre *ed.z80*, compile [F9], emulator [F10] y pulsar Single step (F8) varias veces observando los cambios de los dos primeros dígitos de los campos AF y BC hasta que compruebes que ha finalizado.

Como podrás adivinar, las instrucciones no se encuentran en ningún lenguaje fácil. Son contracciones de palabras en inglés (ld = Load, adc = Add, dec = Decrement y jr = Jump Relative), pero aún sabiendo su significado sigue costando bastante comprender qué hace el programa.

Estos lenguajes disponen de un número muy reducido de instrucciones derivado la arquitectura de la CPU. Esta sería la arquitectura del z80.

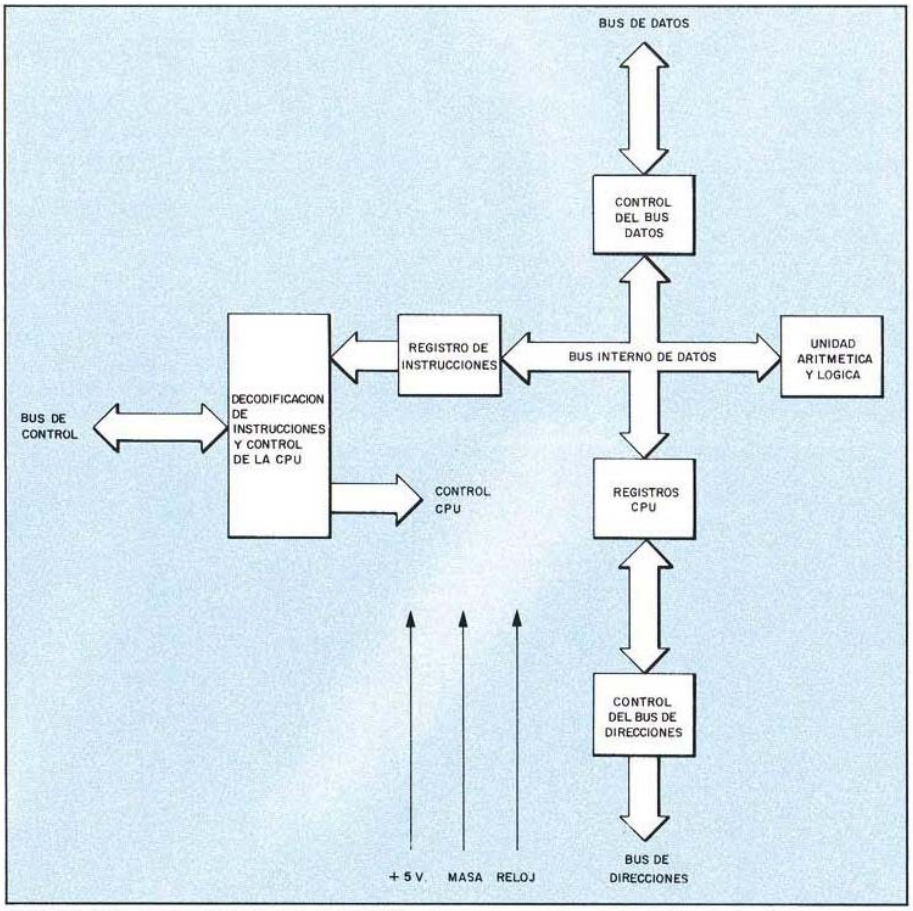


Figura 1. Diagrama de bloques del Z-8Ø.

En los microprocesadores actuales, la arquitectura de la CPU es bastante más compleja, aumentando el número y variedad de las instrucciones.

En lenguaje ensamblador no hay componentes fundamentales para hacer de la programación una tarea sencilla, por lo que suele usarse para aplicaciones donde la optimización es crítica, drivers o análisis de programas de los que no se dispone de código fuente.

Un ejemplo de optimización son los motores gráficos para juegos. En este caso no se utiliza ensamblador para programar el motor completo, sino partes críticas que permiten mejorar las imágenes por segundo (framerate) de los juegos implementados con estos motores.

En el análisis de amenazas se utiliza ensamblador sobre todo para intentar localizar trozos de código pertenecientes a programas maliciosos. Cuando nos encontramos con un programa comprometido, normalmente está en formato ejecutable y no disponemos del código fuente. Estos programas se pasan a ensamblador y se intentan localizar patrones característicos como encriptaciones, replicación de código, inoculación, etc.

En los últimos años el ensamblador ha vuelto a tomar cierta relevancia como lenguaje de desarrollo gracias a la ascensión del IoT (Internet of Things), donde tiene una aplicación directa por las características de estos dispositivos. En la actualidad casi todos los coches que se fabrican necesitan software para gestionar su parte electrónica. Este software se desarrolla en gran medida en ensamblador.

Lenguajes de alto nivel

Las dificultades para desarrollar aplicaciones complejas con ensamblador y su dependencia de la CPU, provocaron que a mediados de los años 50 comenzaran a surgir lenguajes de programación más asequibles para los programadores.

Estos lenguajes vienen a denominarse **lenguajes de alto nivel**.

Los primeros en surgir fueron lenguajes como FORTRAN y COBOL. El primero de ellos de ámbito científico y académico y el segundo para aplicaciones de gestión empresarial.

En el ámbito educativo y doméstico el lenguaje por excelencia de este periodo es BASIC. Fue diseñado a mediados de los años 60 como una herramienta de enseñanza. Su gran difusión se debe a que este lenguaje venía incorporado en los primeros ordenadores que conquistaron el ámbito doméstico a finales de los años 70 y principios de los 80. Sigue siendo popular hoy en día gracias a Visual Basic .NET, aunque en este último lenguaje quede muy poco del BASIC original.

El lenguaje C es un hito en el desarrollo de los lenguajes de programación. Nacido en los Laboratorios Bell de la mano de Dennis Ritchie a principios de los años 70, sigue siendo hoy en día uno de los lenguajes más utilizados tanto en su versión clásica como en las posteriores evoluciones C++ y C#. Podemos afirmar sin lugar a dudas que C es el lenguaje más importante de la historia de la computación.

Más adelante trataremos los lenguajes de alto nivel con más detalle.

5

ACTIVIDAD:

Ejecutar el siguiente código en lenguaje BASIC:

```
10 CLS
20 FOR X = 0 TO 6*3.1415 STEP .01
30 PLOT X*2.6,SIN(X)*18+25
40 NEXT X
```

Simulador de lenguaje BASIC: www.quitebasic.com/

El proceso de traducción

Para que un programa escrito en lenguaje de alto nivel pueda ser ejecutado por el ordenador, previamente debemos traducir las instrucciones a código máquina.

Los **traductores** son programas cuya finalidad es traducir lenguajes de alto nivel a código máquina. Existen dos tipos de traductores:

1. **Intérpretes**: Este programa se encarga de ir ejecutando las instrucciones en lenguaje de alto nivel sin traducirlas literalmente a código máquina. El propio intérprete está en código máquina.
2. **Compiladores**: Traduce las instrucciones en lenguaje de alto nivel a código máquina.

Dependiendo del lenguaje que utilicemos dispondremos de compiladores o intérpretes. La mayoría de los lenguajes de alto nivel opta por uno de los dos tipos de traducción excepto en algunos casos particulares.

Cuando se obtiene un programa ejecutable, está limitado a un sistema operativo y una arquitectura determinada.

Arquitecturas diferentes suponen código máquina incompatible, ya que este lenguaje depende del microprocesador.

Un programa ejecutable necesita un sistema operativo para ser ejecutado. En general, el formato de los programas ejecutables depende del sistema operativo y no son compatibles entre sí.

Más adelante profundizaremos un poco más en los detalles de los traductores.

6

ACTIVIDAD:

Compilar el siguiente programa usando Dev-C++:

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

int main() {
	int n;
	cout << "Escribe un numero:\n";
	cin >> n;
	if(n%2==0) {
		cout << n << " es par.";
	} else {
		cout << n << " es impar.";
	}
	return 0;
}
```

Como resultado de la compilación debemos obtener un archivo con la extensión .exe, que es el archivo ejecutable.

Usando un desensamblador (IDA Freeware) obtener el programa en lenguaje ensamblador.

7

ACTIVIDAD:

Localizar el tipo de traductor de los siguientes lenguajes de alto nivel:

- C
- PHP
- Pascal
- Python
- Visual Basic .NET
- Javascript
- Ruby
- Java

Programas informáticos y aplicaciones informáticas

Software y hardware

El software es la parte intangible de un sistema informático. Lo componen los componentes lógicos (no físicos) y, por tanto, no tangibles. Todo software está diseñado para realizar una tarea determinada en nuestro sistema.

El hardware por contra es la parte física y tangible del sistema informático. Esto incluye el ordenador, cableado, periféricos, etc. En general cualquier elemento físico involucrado en el sistema informático.

El software es el encargado de comunicarse con el hardware, es decir, se encarga de recibir órdenes por parte del usuario y transmitirlo al hardware para llevarlas a cabo.

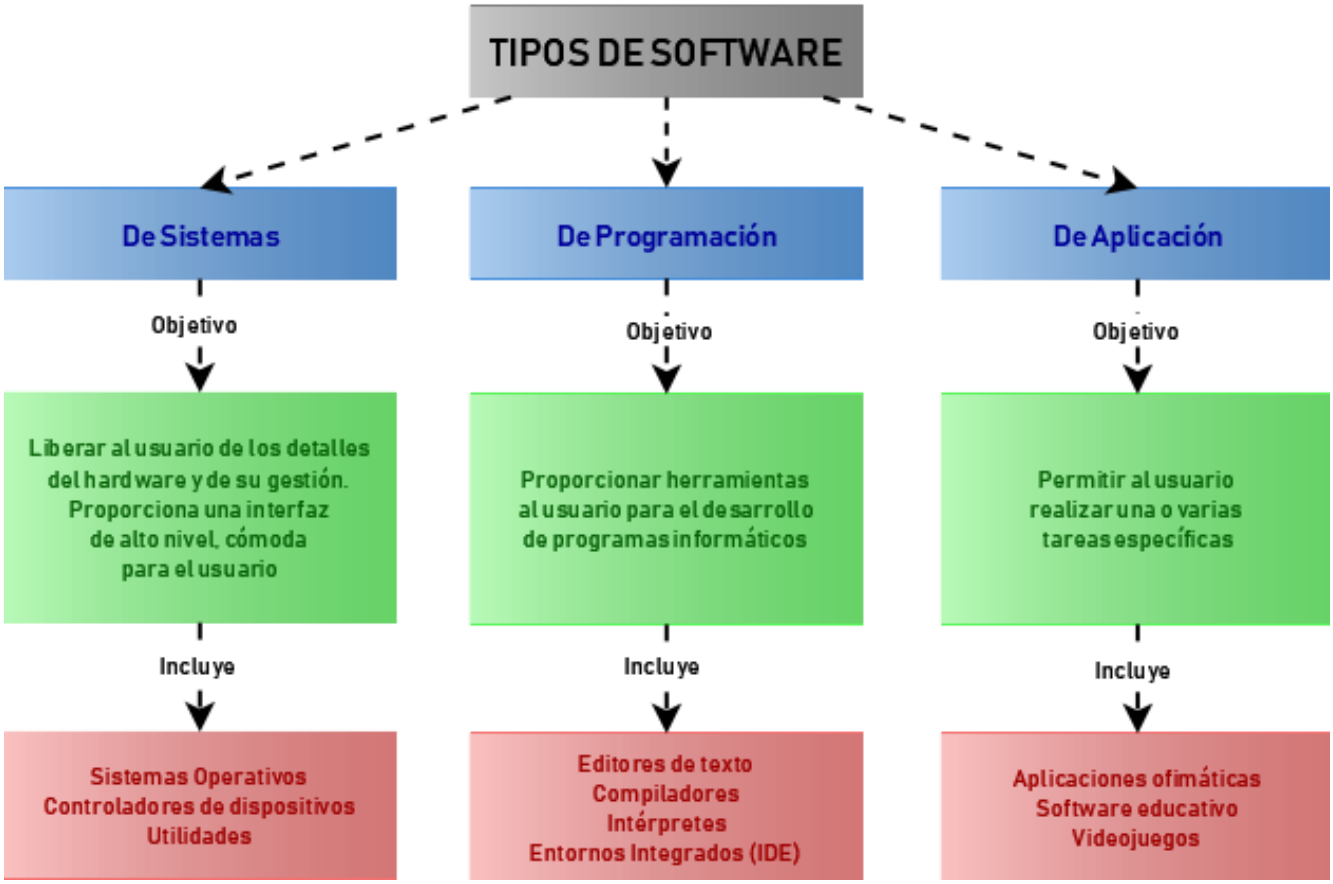
El estándar 729 de IEEE define software como "el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación".

El software tiene una serie de características muy particulares que lo definen:

a) El software es lógico, no físico. Es intangible.
b) El software se desarrolla, no se fabrica. Es más correcto decir desarrolladores de software que fabricantes de software.
c) El software no se estropea y una copia suya da lugar a un clon con las mismas características del original.
d) En ocasiones, puede construirse a medida. Existe software enlatado y a medida.

Tipos de software

Según su función se distinguen **tres tipos de software**: de sistemas (sistema operativo), de programación y de aplicación (aplicaciones o apps).



Definiciones de programa y aplicación

Un **programa** es una serie de órdenes escritas en un lenguaje de programación con una finalidad concreta y que realizan una función determinada.

Una **aplicación** es uno o varios programas que permiten al usuario llevar a cabo una tarea concreta.

Existen en el mercado muchas aplicaciones y cada una tiene su utilidad, pero a veces se agrupan en lo que se denominan suites o paquetes integrados (Microsoft Office, Corel Suite, etc).

Software a medida y software estándar

Un software a medida es una o varias aplicaciones realizadas según los requisitos e instrucciones de una empresa, particular u organismo. Estos programas se amoldan a las necesidades particulares de quien lo encarga.

Algunas características del software a medida son las siguientes:

a) Como todo software, necesita un tiempo de desarrollo.
b) Se adapta a las necesidades específicas del cliente. Eso implica que, en ocasiones, ese software no sea trasladable a otros clientes diferentes (incluso con necesidades similares).
c) Generalmente, suele contener errores y se necesita una etapa de mantenimiento en la que se subsanan y se mejora dicho software.
d) En general, es más costoso que el software estándar, debido a que el precio lo soporta un solo cliente. En el software enlatado o estándar, ese precio se comparte entre los distintos compradores.

El software estándar o enlatado es un software genérico (válido para cualquier cliente potencial), que resuelve múltiples necesidades. Normalmente, para hacerlo más adaptable, dicho software tiene herramientas de configuración que lo adaptan a las necesidades del cliente. En ocasiones, no es el software ideal, puesto que le faltan opciones, procedimientos y procesos que el cliente realiza y que, a la postre, tendrán que realizarse con otra herramienta.

Sus principales características son:

a) Se compra ya hecho. El software ya fue desarrollado en su momento y lo único que podría hacerse es adaptarlo a las necesidades del cliente.
b) Suele tener muchos menos errores que el software a medida, dado que fue probado por múltiples clientes.
c) Suele ser más barato que el software a medida, puesto que los costes de desarrollo se reparten entre las múltiples licencias que se venden.
d) Generalmente, tiene funciones que el cliente no usará y también carecerá de otras opciones. Por regla general, no se adapta completamente a las necesidades de un cliente (es más, a veces, el cliente tiene que adaptarse al software).

8

ACTIVIDAD:

Intenta localizar empresas de desarrollo de software a medida en Sevilla.

Usando linkedin.com, busca ofertas de trabajo como desarrollador Python en Sevilla y comprueba si alguna empresa coincide con la búsqueda anterior.

¿Qué titulación solicitan?
¿Qué tecnologías deberías dominar para poder trabajar en ellas?
¿Exigen experiencia?

Paradigmas de programación

Un paradigma de programación es un estilo o forma de programar. Cada lenguaje de programación se ajusta a uno o varios de estos paradigmas. Los principales paradigmas de programación son:

- **Programación imperativa**: El control del flujo de ejecución es explícito. Se basa en dar instrucciones al ordenador de cómo debe hacer las cosas en forma de algoritmos. Es la más antigua y su ejemplo principal es el código máquina.
- **Programación estructurada**: Es un tipo de programación imperativa donde el flujo del programa se controla con bucles, evitando los saltos directos a otras instrucciones (gotos). Los lenguajes más conocidos de este paradigma son Pascal y C.
- **Programación orientada a objetos**: Se basa en el envío de mensajes entre objetos. Los objetos responden a estos mensajes realizando operaciones denominadas genéricamente métodos. Realmente, los lenguajes que se declaran orientados a objetos se basan en la programación imperativa encapsulando el código en elementos denominados objetos que incluyen tanto variables como funciones. Está representado por C++, C#, Java o Python.
- **Programación declarativa**: El desarrollador expresa lo que quiere, no cómo obtenerlo. El flujo de ejecución está implícito en el lenguaje. Un ejemplo de este paradigma es el lenguaje SQL.
- **Programación funcional**: Consiste en definir funcionalmente los elementos de un programa. Es una variante del paradigma declarativo en el que las herramientas declarativas son funciones matemáticas. Lenguajes de programación funcional actuales son Haskell, F# y Rust.
- **Programación lógica**: Otra variante del paradigma declarativo que usa predicados lógicos y relaciones entre elementos para definir el programa. Un ejemplo de este paradigma es PROLOG.

Pocos lenguajes de programación implementan un paradigma al 100%. En este caso se dicen que son **puros**.

Al contrario, muchos lenguajes facilitan la programación en varios paradigmas. Los lenguajes diseñados para permitir esta flexibilidad se denominan **lenguajes multi-paradigma**.

9

ACTIVIDAD:

Localiza y ejecuta código en los siguientes lenguajes de programación:

- BASIC (Incluyendo una orden GOTO)
- Haskell
- SQL
- Python (Incluyendo el uso de objetos)
- PROLOG

Lenguajes de programación actuales

Para valorar los lenguajes de programación de mayor uso en la actualidad vamos a observar las clasificaciones de los lenguajes de programación por su popularidad publicadas en Internet.

Un resumen de las clasificaciones de 2018 se puede encontrar en el blog de la plataforma de aprendizaje de desarrollo de juegos codingame:

- Índice Tiobex: Analiza las búsquedas usando "<lenguaje> programming".
- Índice PYPL (Popularidad de lenguajes de programación): Basado en las tendencias en Google buscando "<lenguaje> tutorial" en vez de "<lenguaje> programming".
- Ranking Redmonk de lenguajes de programación: Que utiliza ocurrencias de las etiquetas de los lenguajes en StackOverflow y el número de proyectos en GitHub para comparar los lenguajes.
- GitHub octoverse: Clasifica los lenguajes por el número de peticiones de subida abiertos en GitHub.

Otras clasificaciones son:

- La encuesta de Stack Overflow: Cada año se encuesta a los desarrolladores sobre una serie de valoraciones subjetivas sobre los lenguajes de programación.
- Ranking de IEEE Spectrum: Tiene en cuenta unos diez criterios que van desde búsquedas en Google hasta enlaces en HackerNews.

Las conclusiones que extraemos de estas clasificaciones son que una serie de lenguajes de programación copan casi todas las clasificaciones: Java, Python, C++, C#, JavaScript y PHP. También podríamos ampliar la lista con Ruby y VisualBasic .NET, pero con menor incidencia.

Java

Java se consideró en su momento el lenguaje de internet. Surgió en Sun Microsystems como la evolución de C++ para su ejecución multiplataforma usando una máquina virtual.

- Orientado a objetos
- Muy portable. Ejecutable en cualquier plataforma.
- Ofrece múltiples aspectos de seguridad.
- Permite multihilos. Múltiples hilos de ejecución en un mismo programa.

Python

Creado por Guido Van Rossum, recibió su nombre por la afición de Guido a los humoristas Monty Python.
Su ventaja es la portabilidad que ofrece. Si se evitan librerías particulares de cada sistema, un programa en Python puede ejecutarse en cualquier máquina.

- Está orientado a objetos, como la mayoría de lenguajes de programación modernos.
- Permite escribir código en C y luego combinarlo con programas en Python, de esta manera esa porción de código se ejecutará más rápido.
- Puede incrustarse también en otros lenguajes de programación como C y C++.
- Es uno de los lenguajes más simples y sencillos de aprender.

C#

Desarrollado por Microsoft para su plataforma .NET. Se le denomina vulgarmente el 'Java de Microsoft'.

- Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java.
- Orientado a objetos.
- Portabilidad del código fuente.
- Eficiente en el uso de memoria y procesador.

C y C++

Son y han sido unos de los lenguajes más potentes y versátiles de la historia de la informática. Mientras que otros lenguajes se han dejado de utilizar, estos siguen utilizándose en múltiples entornos de desarrollo.

C nació a principios de los años 70 en los laboratorios Bell de la mano de Dennis Ritchie, mientras que C++ fue diseñado al final de la misma década por Bjarne Stroustrup para extender el lenguaje C a la programación orientada a objetos.

- Lenguajes estructurados y muy ligados a funciones. C++ es orientado a objetos.
- Incluyen el concepto de puntero, que es una variable que contiene la dirección de memoria de otra variable. Este aspecto ofrece mucha flexibilidad, pero, por otra parte, su utilización por parte del programador puede ser compleja.
- Combinan comandos de alto nivel, aunque pueden incluirse fragmentos de código que trabajen a más bajo nivel.
- Los programas son muy eficientes y rápidos.
- El tamaño de los programas compilados es pequeño.
- Son lenguajes relativamente portables. Puede recompilarse en cualquier tipo de máquina realizando muy pocos cambios.

JavaScript

Dialecto del estándar ECMAScript, se utiliza implementado como parte de los navegadores web.

- Tiene la ventaja de parecerse mucho a Java, C y C++. Por lo tanto, los programadores de estos lenguajes se sienten cómodos al programar con él.
- Es un lenguaje de scripting.
- Se ejecuta en el lado del cliente.
- Es un lenguaje seguro y fiable.
- Su código es visible y cualquiera puede leerlo, ya que se interpreta en el lado cliente.
- Tiene sus limitaciones como cualquier lenguaje ejecutado en el lado del cliente. Esas limitaciones tienen que ver con la seguridad.
- Actualmente, existen muchas librerías basadas en JavaScript como AngularJS, ReactJS, MeteorJS, JQuery, Foundation JS y BackBone.js, entre otras.

PHP

Lenguaje web de propósito general utilizado frecuentemente en los gestores de contenidos web como wordpress, drupal, joomla, etc. Fue creado en 1995 por Rasmus Lerdorf.

- Lenguaje multiplataforma. Puede instalarse prácticamente en cualquier sistema.
- Se orientó desde el principio al desarrollo de webs dinámicas.
- Conocidos son su buena integración con Mysql y otros servicios como ProFTPd, etc.
- Permite aplicar técnicas de orientación a objetos.
- Lenguaje interpretado.

Visual Basic .NET

Desarrollado a principios de los 90 por Microsoft y actualizado para la plataforma .NET.

- Orientado a objetos.
- Paradigma de la programación amigable.
- Basado en formularios gráficos, con gran cantidad de controles predefinidos que permiten desarrollar aplicaciones muy rápidamente.

Ruby

Creado por Yukihiro Matsumoto a mediados de los 90.

- Diseñado para la productividad y la diversión del desarrollador.
- Mutiparadigma: Orientado a objetos, estructurado y funcional.
- Altamente portable.
- Muy utilizado en el desarrollo de aplicaciones web con el entorno de trabajo Ruby on Rails.

10

ACTIVIDAD:

Localiza un entorno de desarrollo para cada uno de los lenguajes descritos:

- Java
- Python
- C#
- C y C++
- JavaScript
- PHP
- VB .NET
- Ruby

11

ACTIVIDAD:

Clasifica los lenguajes descritos por el número de ofertas de trabajo a nivel mundial en la plataforma linkedin. Realiza este estudio a nivel español y compara los resultados.

Traductores de lenguajes de alto nivel

Como ya adelantamos previamente, existen dos tipos generales de traducción: la compilación y la interpretación. Cada lenguaje de alto nivel opta por una de ellas, una combinación de ambas o deja elegir al desarrollador según sus necesidades.

Compilación

La compilación es la conversión de código en un lenguaje de programación a código en otro lenguaje como paso previo a su ejecución. Normalmente cuando pensamos en compilación hablamos de su versión más tangible, aquella que nos da un programa ejecutable como salida. Se suele denominar compilación anticipada o mediante sus siglas inglesas compilación AOT (Ahead-Of-Time).

Muchos de los lenguajes clásicos y más rápidos utilizan la compilación anticipada: desde Fortran hasta Rust, pasando por C y C++.

Este tipo de compilación permite realizar optimizaciones complejas, por muy costosas que sean, y adaptar el ejecutable final a la máquina donde se va a ejecutar.

Algunos lenguajes realizan un paso de pre-compilación intermedia que no genera código máquina sino bytecode o código portable. En ese paso se trata de producir un código de bajo nivel independiente de la plataforma de ejecución que luego pueda ser interpretado de forma eficiente por una máquina virtual.

Conocidos lenguajes que suelen utilizar compilación a bytecode son Java o la plataforma .NET.

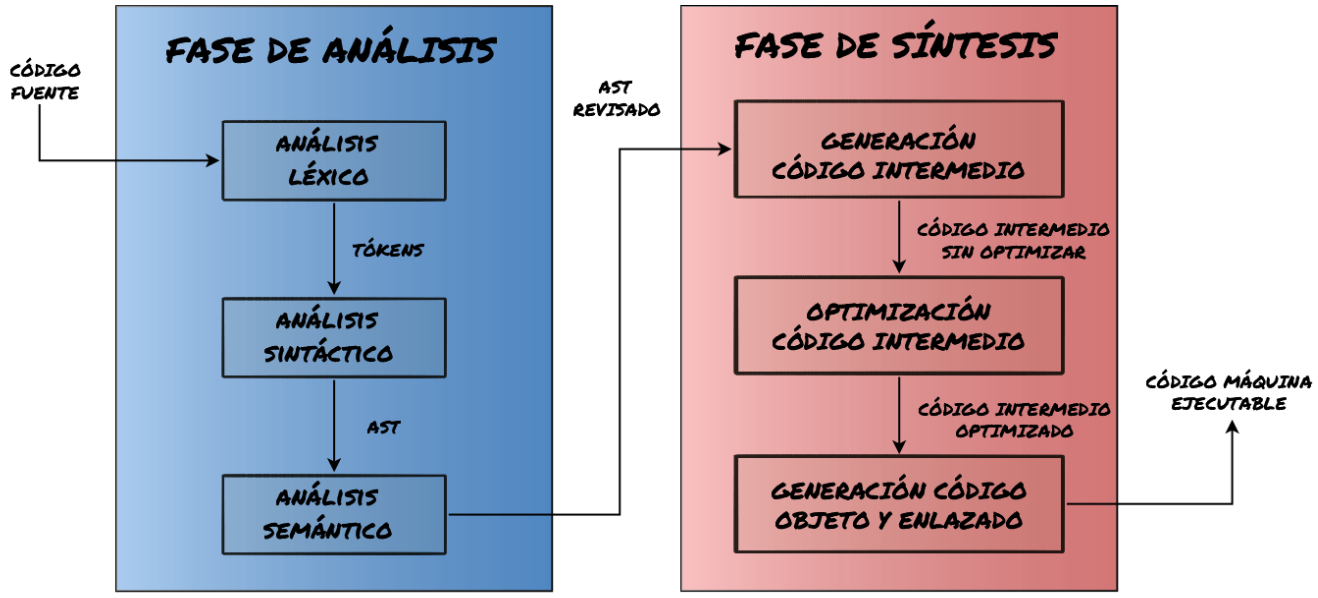
Por otro lado, cuando la compilación no genera un binario en código máquina sino un resultado en otro lenguaje de programación, se trata de una compilación fuente-a-fuente (source-to-source).

Los casos más comunes de este tipo los encontramos en tecnologías web, ya que los navegadores principalmente permiten ejecutar código JavaScript. Si queremos programar para el navegador con otro lenguaje debemos compilar a JavaScript. Algunos lenguajes que usan compilación fuente a fuente son TypeScript, PureScript o Dart.

El proceso de compilación

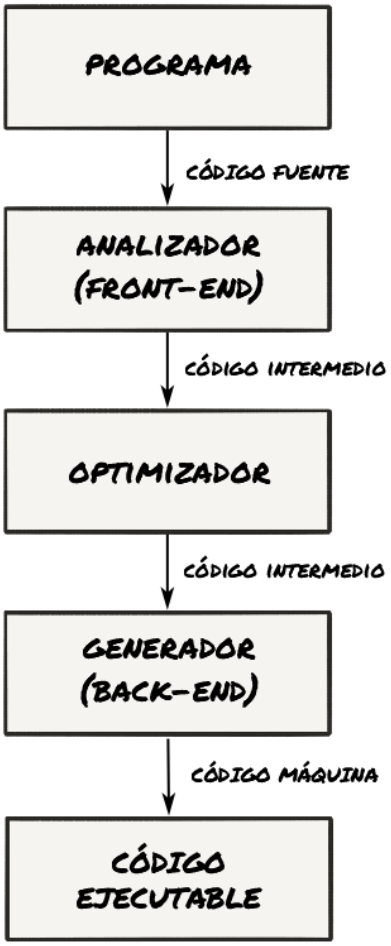
La compilación es un proceso complejo que se realiza en dos fases:

- Análisis: Se comprueba la corrección del código fuente. Esta fase incluye:
  - Análisis léxico: Se asimila cada elemento del código fuente a una serie de categorías denominadas tokens.
  - Análisis sintáctico: Se crean representaciones abstractas de la estructura del código fuente. Estas representaciones suelen ser árboles de sintaxis abstracta (AST). Si recordamos el análisis sintáctico de la asignatura de lengua nos haremos una idea.
  - Análisis semántico: Se revisan los árboles de sintaxis en busca de expresiones sin sentido, aunque correctas a nivel sintáctico.
- Síntesis: Se trata de generar el código ejecutable. Suele incluir:
  - Generación de código intermedio: A partir de los AST revisados resultantes de la fase de análisis.
  - Optimización de código: Sin modificar el sentido del código, se modifica el código intermedio buscando optimizar su ejecución.
  - Generación de código objeto y enlazado: Se genera el código máquina a partir del código intermedio y se añaden las librerías de códigos necesarias para crear un ejecutable.



Estructura de un Compilador

En la actualidad los compiladores se estructuran en tres programas que se encargan de llevar a cabo una o varias fases del proceso de compilación: Analizador (Frontend), optimizador y generador (Backend).



El analizador es genérico para todas las plataformas de destino, mientras que el optimizador puede ser común a varios lenguajes y plataformas. El generador es propio de cada plataforma, ya que genera código máquina a partir de código intermedio.

Interpretación y lenguajes interpretados

La forma alternativa de ejecutar un programa a partir del código en un lenguaje de programación es no generar una traducción a código máquina, sino analizar el código y realizar los cómputos que éste indique, bien directamente o bien a partir algún tipo de representación intermedia que no constituya un programa en código máquina. En este caso se dice que el programa es interpretado.

Mientras se produce la interpretación, debe ejecutarse en el sistema el programa que la realiza, es decir, el intérprete del lenguaje.

Las representaciones intermedias que puede generar el intérprete son generalmente de dos tipos: un código de bajo nivel o una estructura de datos.

El código de bajo nivel suele ser bytecode en la actualidad y las estructuras de datos suelen ser árboles de sintaxis abstracta (AST) que se van recorriendo para obtener los resultados de la ejecución. Estas últimas no son tan utilizadas ya que producen mayores sobrecostes que los códigos de bajo nivel.

Ejemplos de lenguajes típicamente interpretados son Python (con bytecode), Ruby (usaba ASTs hasta la versión 1.8), PHP (con bytecode) y Perl (utiliza ASTs).

Compilación en tiempo de ejecución

Conocida por sus siglas inglesas JIT (Just-In-Time), es una técnica para mejorar el rendimiento de sistemas que compilan a bytecode consistente en traducir el bytecode a código máquina en tiempo de ejecución.

Este tipo de compilación se denomina compilación dinámica, y permite analizar el código durante su ejecución y mejorar los resultados de un intérprete de bytecode convencional o hasta de una compilación anticipada tradicional (AOT).

12

ACTIVIDAD:

Describir el sistema de compilación y ejecución de la plataforma .NET.

13

ACTIVIDAD:

¿Qué es LLVM? Describirlo y ajustar sus herramientas a la estructura de un compilador.

14

ACTIVIDAD:

Localizar el uso de JIT en la actualidad. ¿Se usa en máquinas virtuales Java? ¿Y en Python?

Ingeniería del software

Con el desarrollo e implantación de las tecnologías software en todos los aspectos de nuestras vidas, se hace necesario controlar el proceso de desarrollo de software para evitar fracasos y problemas que en algunos casos extremos hasta podrían suponer la pérdida de vidas humanas.

La **ingeniería del software** es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales.

El proceso del software

Un proceso es un conjunto de actividades, acciones y tareas que se ejecutan cuando va a crearse algún producto del trabajo, en este caso software.

Este proceso se puede definir con las siguientes **actividades estructurales**:

- **Comunicación**: Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente y con otros participantes. Se busca entender los objetivos de los participantes respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software. Estos requerimientos se denominan de forma genérica **requisitos del cliente**.
- **Planificación**: Su objetivo es definir un mapa que guíe al equipo durante el desarrollo. Este mapa, denominado **plan del proyecto de software**, describe las tareas técnicas por realizar, los riesgos probables, los recursos que se necesitan, los productos que se obtendrán, un calendario de actividades y las acciones para realizar el seguimiento del mismo.
- **Modelado**: El modelado se puede asemejar a la creación del plano de un edificio. Se intenta crear un plano del producto software usando diferentes diagramas que modelan el comportamiento del sistema en todas sus facetas. Normalmente el modelado incluye dos actividades: **análisis** y **diseño**. En la fase de análisis se recopilan y formulan los requisitos del cliente, o sea, se intenta analizar qué necesita el cliente. En la fase de diseño se crea una arquitectura software que intente atender las necesidades del cliente.
- **Construcción**: Esta actividad combina la generación de código (**codificación**) de acuerdo al diseño resultante de la actividad de modelado y las **pruebas** que se requieren para descubrir los posibles errores en éste.
- **Despliegue**: El software (como un todo o como un incremento parcialmente terminado) se **entrega** al consumidor que lo evalúa y nos confirma que resuelve sus necesidades o necesita ser ajustado. Tras la entrega normalmente se produce la **asistencia** al cliente, que posiblemente propondrá cambios en el sistema como **retroalimentación**.

Modelos del proceso software

Los modelos del proceso software son formas de llevar a cabo las actividades estructurales. Cada uno pone distinto énfasis en cada una de ellas y define de forma diferente el flujo de proceso que invoca cada actividad.

Los modelos tradicionales incluyen:

- **Modelo en cascada**: Las actividades estructurales se llevan a cabo de forma secuencial hasta conseguir el producto software.
- **Modelo incremental**: Se desarrollan en paralelo incrementos en funcionalidad del sistema. El primer incremento es lo que se denomina el producto fundamental, que cubre los requisitos básicos del sistema. A partir del mismo se van desarrollando incremento para paulatinamente completar la funcionalidad del sistema.
- **Modelo evolutivo**: Se van desarrollando versiones cada vez más completas del software, de forma que pueden adaptarse a los cambios en los requisitos del cliente. Destacan los modelos basados en prototipos y el modelo espiral.

Proceso unificado

A comienzos de la década de los 90, James Rumbaugh, Grady Booch e Ivar Jacobson comenzaron a trabajar en un método unificado que combinaría lo mejor de cada uno de sus métodos individuales de análisis y diseño orientado a objetos. El resultado fue un UML, lenguaje de modelado unificado, que contiene una notación robusta para el modelado u el desarrollo de los sistemas orientados a objetos.

Usaremos el UML en este curso para construir los diagramas que expresan requisitos de usuarios, estructura de clases o comportamiento dinámico de un sistema orientado a objetos.

En concreto, el proceso unificado se define como impulsado por los casos de uso, centrado en la arquitectura, iterativo e incremental.

Metodologías ágiles

En 2001 un grupo de desarrolladores conocido como la alianza "Ágil", firmaron el "Manifiesto por el desarrollo ágil de software".

En él se establece que se van a valorar sobre todo:

- Los individuos y sus interacciones, sobre los procesos y las herramientas.
- El software que funciona, más que la documentación exhaustiva.
- La colaboración con el cliente, y no tanto la negociación del contrato.
- Responder al cambio, mejor que apegarse a un plan.

La idea que subyace es crear metodologías que se adapten a la realidad cambiante del desarrollo software.

Pongamos un ejemplo, un emprendedor nos solicita una serie de requisitos al comenzar el desarrollo del sistema, pero a los pocos meses de comenzar su competidor ofrece una nueva funcionalidad que él no nos había solicitado. ¿Qué debemos hacer? ¿Crear un sistema incompleto y dejar un cliente descontento o afrontar los cambios en los requisitos para adaptar el sistema a las funcionalidades que ofrece la competencia?

Si nuestro modelo de proceso software es estricto, es probable que generemos una cantidad de trabajo elevada para adaptarnos a los cambios, sin embargo, el uso de tecnologías ágiles evita este sobreesfuerzo porque son modelos que llevan la adaptación a los cambios como una de sus premisas.

Las metodologías ágiles más utilizadas en la actualidad son:

- **XP (Programación extrema)**: Es un modelo diseñado para mejorar la calidad y la respuesta en entornos con requisitos cambiantes. Los elementos fundamentales de este modelos son: la programación en parejas para mejorar la revisión del código, crear pruebas unitarias de todo el código, evitar la programación de funcionalidades hasta que no son necesarias, una estructura de gestión plana (sin jerarquías), simplicidad y claridad en la codificación, esperar cambios en los requisitos del cliente conforme pasa el tiempo y es más consciente de los problemas y comunicación frecuente con el cliente y entre los desarrolladores.
- **SCRUM**: Es un marco de trabajo ágil con un énfasis en el desarrollo software. Está diseñado para equipos de 3 a 9 miembros, que dividen el trabajo en acciones que se pueden completar en iteraciones denominadas 'sprints', de no más de un mes y normalmente de 15 días. Estas actividades tienen un seguimiento diario en una reunión informal de 15 minutos donde cada uno de ellos reflexiona sobre lo que va a hacer durante esa sesión de trabajo, si tiene problemas para cumplir la fecha, etc.

15

ACTIVIDAD:

Localiza y describe otras metodologías ágiles aparte de XP y SCRUM.

La documentación

Durante el proceso de desarrollo software es fundamental generar la documentación suficiente para el mantenimiento, uso e instalación del sistema.

Todas las actividades que forman el proceso de desarrollo van a generar una serie de documentos que serán la base de la **documentación técnica** del proyecto, necesaria para la incorporación al mismo de nuevo personal o para el equipo de mantenimiento de sistema.

Además es importante reunir en el **manual de instalación** todas las operaciones necesarias para poner el sistema en marcha desde cero.

Y como elemento necesario para los usuarios, un **manual de usuario** donde se expliquen todas las funcionalidades del sistema.

Roles en el equipo de desarrollo software

Los equipos de desarrollo software tradicionalmente incluían los siguientes roles:

- **Arquitecto de software**: Es la persona encargada de decidir cómo va a realizarse el proyecto. Decide la forma y los recursos con los que va a llevarse a cabo un proyecto.
- **Jefe de proyecto**: Dirige el proyecto. Tiene que saber gestionar un equipo de trabajo y los tiempos, además de tener una relación fluida con el cliente.
- **Analista de sistemas**: Es una persona con experiencia que realiza el estudio del problema y ejecuta la actividad de modelado del sistema (análisis y diseño).
- **Analista programador**: Puesto entre el analista y el programador. Suele ser un programador sénior que realiza funciones tanto de modelado como de construcción del software.
- **Programador**: Su función es conocer en profundidad el lenguaje de programación y codificar las tareas que le han sido encomendadas por el analista o el analista programador. En el caso de las aplicaciones web los desarrolladores se especializan en el desarrollo del lado del cliente (front-end) o del lado del servidor (backend).

En equipos de desarrollo ágiles la estructura jerárquica se difumina. En el caso de Scrum estos son los roles que nos podemos encontrar:

- Líder de equipo (Scrum Master)
- Desarrollador
- Diseñador de experiencia de usuario
- Gestor de calidad y pruebas

16

ACTIVIDAD:

Localiza empresas de desarrollo en linkedin y analiza su plantilla para tomar dos ejemplos: Una que utilice el modelo jerárquico tradicional y otra que haga uso de metodologías ágiles.