U. T. 1 Introducción



Origen de la Informática

La historia del ordenador se remonta a las primeras reglas de cálculo y a las primeras máquinas diseñadas para facilitarle al ser humano la tarea de la aritmética. El ábaco, por ejemplo, fue un importante adelanto en la materia, creado alrededor de 4.000 a. C.

También hubo inventos muy posteriores, como la máquina de Blaise Pascal, conocida como Máquina de Pascal o Pascalina, creada en 1642. Consistía en una serie de engranajes que permitían realizar operaciones aritméticas. Esta máquina fue mejorada por Gottfried Leibinitz en 1671 y se dio inicio a la historia de las calculadoras.

Los intentos del ser humano por automatizar continuaron desde entonces: Joseph Marie Jacquard inventó en 1802 un sistema de tarjetas perforadas para intentar automatizar sus telares, y en 1822 el inglés Charles Babbage empleó dichas tarjetas para crear una máquina de cálculo diferencial.

Otro importante fundador en este proceso fue Alan Turing, creador de una máquina capaz de calcular cualquier cosa, y que llamó “máquina universal” o “máquina de Turing”. Las ideas que sirvieron para construirla fueron las mismas que luego dieron nacimiento al primer computador.

Otro importante avance fue el de ENIAC (Electronic Numeral Integrator and Calculator, o sea, Integrador y Calculador Electrónico Numeral), creado por dos profesores de la universidad de Pensilvania en 1943, considerado el abuelo de los computadores propiamente dicho. Consistía en 18.000 tubos de vacío que llenaban un cuarto entero.

Por último, la historia de los computadores no habría tenido el curso que tuvo sin la invención en 1947 de los transistores, fruto de los esfuerzos de los laboratorios Bell en Estados Unidos, dando origen a lo que conocemos hoy en día.

Conceptos Informáticos Fundamentales

La razón principal por la que una persona utiliza un ordenador es para resolver problemas (en el sentido más general de la palabra) o, en otras palabras, procesar una información para obtener un resultado a partir de unos datos de entrada.

Los ordenadores resuelven los problemas mediante la utilización de programas escritos por los programadores. Los programas de ordenador no son entonces más que métodos para resolver problemas. Por ello, para escribir un programa, lo primero es que el programador sepa resolver el problema que estamos tratando.

El programador debe identificar cuáles son los datos de entrada y a partir de ellos obtener los datos de salida, es decir, la solución, a la que se llegará por medio del procesamiento de la información que se realizará mediante la utilización de un método para resolver el problema denominada algoritmo.

Clasificación del Software

El software es elemento que desarrollan los programadores y se puede clasificar atendiendo a diversas características, pero en un curso de programación haremos dicha clasificación teniendo en cuenta la funcionalidad general en el sistema:

- **Software de sistema**. Elementos que permiten el mantenimiento del sistema en global: sistemas operativos, controladores de dispositivos, servidores, utilidades, herramientas de diagnóstico, de corrección y optimización, generalmente se usa el lenguaje C para dicha programación con parte en código ensamblador.
- **Software de programación**. Diferentes alternativas y lenguajes para desarrollar programas de informática: editores de texto, compiladores, intérpretes, enlazadores, depuradores, entornos de desarrollo integrados (IDE).

- **Software de aplicación**. Permite a los usuarios llevar a cabo una o varias tareas específicas en cualquier campo de actividad: aplicaciones ofimáticas, para control de sistemas y automatización industrial, software educativo, software empresarial, bases de datos, telecomunicaciones (Internet), videojuegos, software médico, software de diseño asistido (CAD), software de control numérico (CAM).

Qué es la programación

Concepto

La programación es el proceso mediante el cual se diseña y codifica un algoritmo mediante un conjunto de instrucciones siguiendo una sintaxis concreta (lenguaje). Este proceso no es sencillo ni rápido y tenemos que ser muy cuidadosos para que el resultado final sea un producto "aceptable".

La creación de un algoritmo se basa en la resolución de problemas reales por nosotros. Generalmente ante un problema, lo primero que hacemos es observarlo para determinar todos los factores que intervienen, a continuación, desarrollamos un conjunto de soluciones que nos parecen factibles para resolverlo, siendo la última etapa la implementación (llevar a cabo) una de las soluciones para resolver el problema planteado, así como realizaremos pruebas para comprobar su correcto funcionamiento.

La programación ha evolucionado a lo largo de los años, presentando diversas maneras de llevarse a cabo, creando lo que se llaman paradigmas de la programación. Estos paradigmas se siguen utilizando hoy en día y dependiendo del proyecto que estemos desarrollando se usarán unos u otros o mezcla.

Paradigmas de la programación

Como hemos mencionado anteriormente, los paradigmas son reglas aceptadas de programación que han ido evolucionando paralelas a la tecnología y que utilizaremos según el proyecto que desarrollemos.

- **Programación estructurada**. La programación estructurada se basa en utilizar solamente tres tipos de estructuras de control en nuestros programas. En concreto se usarán solo estructuras secuenciales, alternativas e iterativas. Se evitarán completamente el uso de sentencias de salto incondicionales. ¿Qué es una secuencia de control? Una regla del leguaje.
  - **Secuencias**: Ejecución de una instrucción a continuación de otra.
  - **Bucles o iterativas**: Repetición de un conjunto de secuencias un número de veces determinado o mientras que se cumpla la condición.
  - **Alternativas o condicionales**. Ejecución de unas secuencias u otras dependiendo de una condición.
- **Programación modular**. Este paradigma implica la descomposición del problema en varios subproblemas menos complejos que se puedan implementar más fácilmente. Además, se incluye la posibilidad que los subproblemas se comunique entre sí traspasando datos de unos a otros.
  La programación modular se crear desde dos perspectivas, de arriba abajo o viceversa. En la primera aproximación el problema de partida es de gran tamaño y se va dividiendo en más pequeños hasta llegar a problemas unitarios. En la segunda aproximación se crean soluciones para problemas pequeños que se van uniendo para ir creando soluciones mayores hasta dar con la solución completa.
  Los elementos constituyentes de este paradigma son las funciones y procedimientos, clases y métodos a bajo nivel; unidades, paquetes y librerías a alto nivel.
  La programación modular se basa a más bajo nivel en funciones y para crear una buena función tendremos en cuenta los siguientes puntos:
  - Un único punto de entrada y de salida.
  - La función se comportará como una caja negra.
  - El tamaño orientativo estará entre 30 y 50 líneas.
  - Tendrá máxima cohesión o relación con los elementos de la misma unidad funcional.
  - Tendrá mínimo acoplamiento o dependencia de las demás funciones.
- **Programación orientada a objetos (POO)**. Este paradigma se estudiará en una unidad de trabajo posterior. Está basada en los procesos de ingeniería tradicional.

Qué es un algoritmo

Por algoritmo entendemos un conjunto ordenado y finito de operaciones matemáticas que permiten resolver un problema con entradas y salidas definidas que además cumplen las siguientes características:

- Tiene un número finito de pasos
- Acaba en un tiempo finito. Si no acabase nunca, no se resolvería el problema.
- Todas las operaciones deben estar definidas de forma precisa y sin ambigüedad.
- Puede tener varios datos de entrada y de salida.

Un claro ejemplo de algoritmo es una receta de cocina, donde tenemos unos pasos que hay que seguir en un orden y deben de estar bien definidos, tiene un tiempo finito y tiene unos datos de entrada (ingredientes) y una salida (el plato). Por ejemplo, el algoritmo para freír un huevo podría ser el siguiente:

- Datos de entrada: Huevo, aceite, sartén, fuego.
- Datos de salida: huevo frito.

Procedimiento:

1. Poner el aceite en la sartén.
2. Poner la sartén al fuego.
3. Cuando el aceite esté caliente, cascar el huevo e introducirlo.
4. Cubrir el huevo de aceite.
5. Cuando el huevo esté hecho, retirarlo.

La codificación de un algoritmo en un ordenador se denomina programa

Un algoritmo debe cumplir una serie de requerimientos para ser funcional:

- ¿Produce los resultados esperados?
- ¿Es la solución óptima?
- ¿Cómo va a funcionar el algoritmo si incrementamos el número de datos?

Estas preguntas nos las haremos cada vez que hayamos terminado un algoritmo para asegurar la funcionalidad.

Los algoritmos que desarrollemos se engloban según las características del problema en: algoritmos que hacen uso intenso de los datos, algoritmos que hacen uso intenso del procesador y en algoritmos mezcla de los dos anteriores. Esta clasificación nos da una aproximación a los problemas que nos encontraremos a la hora de crearlo.

Representación de los algoritmos

Uno de los principales problemas que nos encontramos a lo hora de crear una aplicación, es que el programador poco experimentado cree que su herramienta principal de trabajo es el ordenador, y se lanza a crear código inmediatamente frente a la pantalla. Esta concepción nos lleva a que el aprendizaje de la programación no se hace de forma correcta y crea vicios muy difíciles de subsanar después.

Si revisamos el ciclo de vida estándar que se muestra más adelante, comprobaremos que solo una de las fases es de codificación, las otras cuatro no, por lo que hay que desbancar la idea de que la programación se aprende exclusivamente delante de la pantalla programando, la programación se aprende primero planificando y comprendiendo lo que queremos hacer y después codificando, por lo que emerge la necesidad de representar de forma gráfica nuestros algoritmos antes de la codificación.

La codificación primero se planifica en papel y después se pasa al ordenador

Elementos de los algoritmos

- Datos: expresión general que describe con los objetos con los que opera un algoritmo: números (enteros, reales, lógicos, carácter, cadena).

- Constantes: Valores que no cambian a lo largo del programa, almacena un dato.
- Variables: Un objeto que puede cambiar con el desarrollo del programa, almacena un dato.
- Expresiones: combinación de operadores y operandos. Los operadores pueden ser de diversos tipos: aritméticos, lógicos, etc. Los operandos serán datos, constantes o variables.
- **Palabras reservadas o cajas de acción**. Representan el flujo de movimiento del programa.

Diagramas de flujo (ordinogramas)

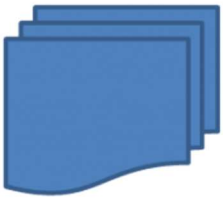
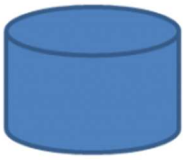
Un diagrama de flujo es una representación gráfica del algoritmo. Esta representación siempre debe seguir unas reglas básicas y usar unos símbolos predefinidos para poder entenderlos.

- El comienzo del programa figurará en la parte superior del ordinograma y la finalización abajo.
- El símbolo de comienzo y de fin deberá aparecer una sola vez en el ordinograma.
- El flujo de las operaciones será, siempre que sea posible de arriba a abajo y de izquierda a derecha.
- Se evitarán siempre los cruces de líneas utilizando conectores.
- Se podrá dividir el algoritmo entre diversas hojas usando conectores de unión.



La herramienta **Dia** es muy útil para crear este tipo de diagramas

Símbolos de los diagramas de flujo

SÍMBOLO	SIGNIFICADO	SÍMBOLO	SIGNIFICADO
	Terminal. Indica el inicio o la terminación del flujo del proceso		Actividad. Representa una actividad llevada a cabo en el proceso.
	Decisión. Indica un punto en el flujo en que se produce una bifurcación del tipo "SÍ" – "NO"		Documento. Se refiere a un documento utilizado en el proceso, se utilice, se genere o salga del proceso.
	Multidocumento. Refiere a un conjunto de documentos. Por ejemplo, un expediente que agrupa distintos documentos.		Inspección/ firma. Empleado para aquellas acciones que requieren supervisión (como una firma o "visto bueno")
	Base de datos/ aplicación. Empleado para representar la grabación de datos.		Línea de flujo. Proporciona una indicación sobre el sentido de flujo del proceso.

Símbolos básicos

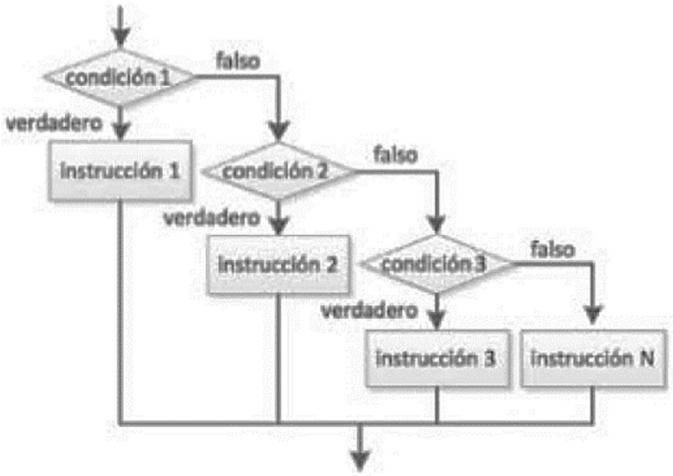


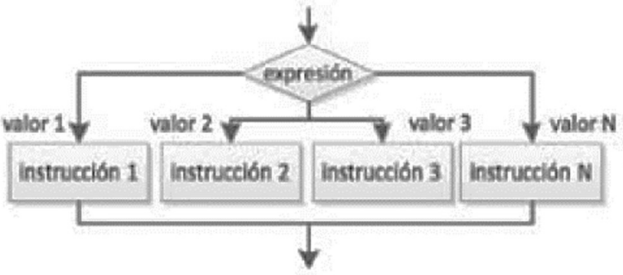
Condicionales









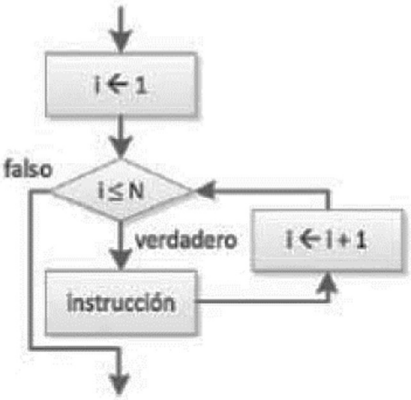


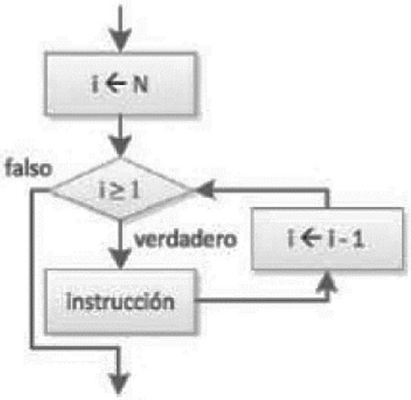
Bucles









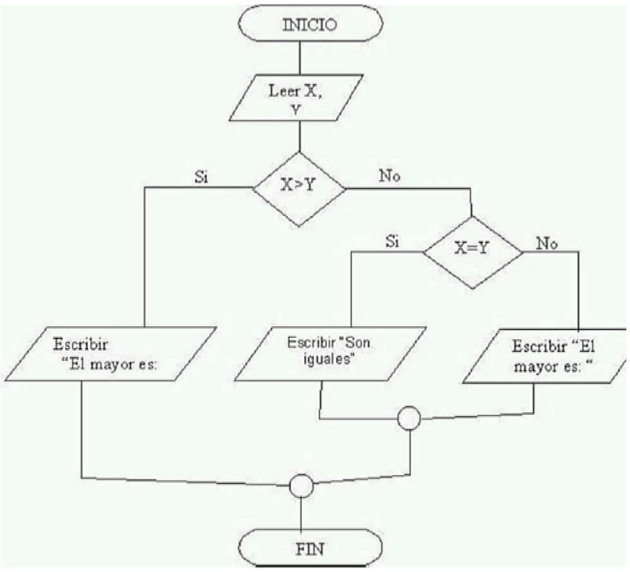


Inicio y llamadas a procedimientos





Ejemplo: Algoritmo que lee dos números "X" e "Y", determina si son iguales, y en caso de no serlo, indica cuál de ellos es el mayor.



Pseudocódigo

Esta técnica de representación utiliza un lenguaje muy cercano a las sintaxis de los lenguajes reales de programación para definir un algoritmo, siendo el paso a la codificación muy sencilla. Se basa en las siguientes estructuras.

- Operadores matemáticos: +, -, *, /
- Operadores de comparación: <, <=, >, >=, <>, =
- Operadores lógicos: and, or, not
- Operadores de asignación: ←
- Estructuras del lenguaje.
  - Inicio y fin: Inicio | Fin.
  - Repetitivas: Para| Desde |Mientras.
  - Condicionales: Si| Sino| Entonces | Según| Caso | Otro caso
  - De entrada / Salida: Leer| Imprimir.
  - Finalización: Fin otra_etiqueta.

Todo algoritmo empieza por la palabra claves Inicio y finaliza por la palabra clave Fin. El algoritmo siempre empieza por arriba y termina por abajo. En su interior utilizaremos tantas palabras claves en secuencia y anidándolas como necesitemos para crear nuestro algoritmo respetando la sintaxis. Las entradas y salidas se realizarán mediante las instrucciones correspondientes. Cada vez que iniciemos un bloque se sangrarán todas las líneas que estén dentro del mismo con cuatro espacios en blanco.

Ejemplo: Crear un algoritmo para sumar dos números pedidos por teclado.

```
Inicio
    Leer(numero_uno)
    Leer(numero_dos)
    suma ← numero_uno + numero_dos
    imprimir(suma)
Fin
```

Sintaxis del pseudocódigo

Secuencia

```
Inicio
     <instrucción_1>
     <instrucción_2>
     …
     <instrucción_n>
Fin
```

Condicional

```
Inicio
    si <condición> entonces
        <instrucción_1>
    sino
     <instrucción_2>
    Fin si
Fin
```

Condicional

```
Inicio
    según <expresión>
        caso <valor>
           <instrucción_1>
        caso <valor>
           <instrucción_2>
        …
        caso <valor>
           <instrucción_n>
        otro
           <instrucción_m>
     Fin según
Fin
```

Repetitiva

```
Inicio
    mientras <condición> hacer
        <instrucción_1>
        …
    fin mientras
Fin
```

Repetitiva

```
Inicio
    repetir
        <instrucción_1>
        …
    hasta que <condición>
    fin repetir
Fin
```

Repetitiva

```
Inicio
    para n←valor hasta valor incremento
    hacer
        <instrucción_1>
        …
    rin para
Fin
```

Ejemplo: Crear un algoritmo para determinar si un número es par o no.

```
Inicio
   imprimir(“Introduzca un número:”)
   leer(numero)
   si (numero % 2) == 0  //% indica el módulo, o resto de la división
      imprimir(“par”)
   sino
      imprimir(“impar”)
   fin si
Fin
```

Ejemplo: Crear un algoritmo para determinar la suma, resta, multiplicación y división de dos números pedidos al usuario teniendo en cuenta que no se pueden dividir números entre cero.

```
Inicio
   imprimir("Introduce el primer número:”)
   Leer(num1)
   imprimir("Introduce el segundo número”)
   leer(num2)
   Resultado←0
   Resultado←num1+num2
   imprimir(resultado)
   resultado←num1-num2
   imprimir(resultado)
   resultado←num1*num2
   imprimir(resultado)
   si num2==0 entonces
      imprimir(“La división entre num1 y num2 es infinito”)
   si no
      resultado←num1/num2
      imprimir(resultado)
   Fin si
Fin
```

Rendimiento de los algoritmos

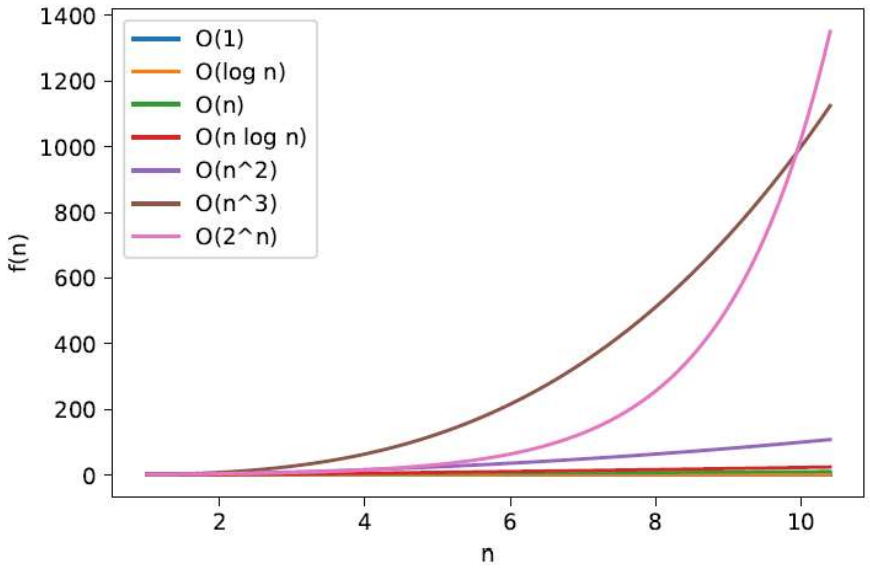
Si queremos crear algoritmo óptimos y eficientes tendremos que tener métricas (indicadores que sean medidos) que nos aporten la información suficiente para determinar el rendimiento. Pero para conseguir estos objetivos, los algoritmos deben ser correctos, deben ser entendibles y deben ser eficientes.

Para determinar la eficiencia hemos dicho que es necesario medir, y las mediciones se pueden llevar a cabo en dos ámbitos: En el ámbito del espacio y en el ámbito del tiempo. En el espacio estimaremos los recursos de almacenamiento necesarios (disco y memoria) para el funcionamiento del programa. En el ámbito del tiempo intentaremos encontrar una medida que nos indique cómo se comportaría el algoritmo en el tiempo y en función de la cantidad de datos. Para esta última medida solemos usar el orden de complejidad de un algoritmo antes de implementarlo y realizamos mediciones en ejecución (**profile**) del tiempo real que tarda en ejecutarse.

Órdenes de complejidad

El orden de complejidad es una expresión matemática que indica cómo se comportará un algoritmo si la cantidad de datos aumenta. Esta expresión es del tipo O(x), en la que x es una función que indicará dicho comportamiento. Así nos encontraremos con órdenes constantes O(1), órdenes lineales O(n), órdenes polinómicos $O(n^x)$, órdenes logarítmicos O(nlog(n)) O(log(n)), etc.

Comprenderemos mucho mejor lo que significa si vemos las gráficas correspondientes.



Si interpretamos los valores del eje x como incrementos en la cantidad de datos y el eje y el tiempo que tarda en realizar la tarea, lo ideal es que esté por debajo de la función constante (en azul en la gráfica) o se acerque lo más posible a ella. Como vemos los únicos órdenes de complejidad que se acercan son los órdenes logarítmicos.

No estudiaremos cómo determinar el orden de complejidad de un algoritmo, pero debemos de saber elegir entre varios algoritmos si nos dan un orden de complejidad para cada uno de ellos.

Codificación y ejecución

Al final, los algoritmos tienen que estar implementados en un lenguaje y la elección de este dependerá de varios factores, entre los que destacan el utilizado en el proyecto en el que estemos asignados o por una elección propia.

El proceso de codificación se realiza a través de herramientas de edición de texto simples o complejas (IDEs) con las que se facilita la tarea del programador. En estos entornos se lleva a cabo todo el proceso de creación de los algoritmos: codificación, prueba, empaquetado y distribución.

Una vez codificado el programa se tiene que compilar (pasar al lenguaje del ordenador) y ejecutar.

La compilación se realiza a través de programa especializados (compiladores o intérpretes), la ejecución es el Sistema Operativo. el encargado de realizarla.

Clasificación de los lenguajes

Los lenguajes se pueden clasificar según atendiendo a varias características:

- **Según el código**: binario, ensamblador, alto nivel. Esta clasificación a la cercanía del lenguaje usado al microprocesador. Cuanto más cerca esté el lenguaje de la máquina más difícil será su uso y la programación, pero más eficiente a su vez será. Hay ciertas partes de los sistemas operativos que deben ser programados en lenguaje ensamblador para ser eficientes.
  Del mismo modo, los lenguajes de más alto nivel son más fáciles de usar y presentan más herramientas al programador a cambio de una menor eficiencia.
  En última instancia, independientemente del lenguaje utilizado, el algoritmo debe ser "traducido" a código binario (código máquina) para que se pueda ejecutar, esa transformación se llama **compilación**.

- **Según la ejecución**: compilados, interpretados o mixtos. Una vez programado el algoritmo se tiene que ejecutar dentro de un microprocesador y este solo entiende ceros y unos (código binario o código máquina), por lo que es necesaria la traducción a ese lenguaje. Este proceso de traducción se puede hacer desde varios puntos de vista.
  Compilado: El código fuente se compila directamente en un fichero ejecutable ya en código máquina. Esta aproximación es la más rápida de las tres, pero el ejecutable es dependiente de la arquitectura y del sistema operativo para el que se haya compilado.
  Interpretado: Un programa intermedio lee el código fuente, lo traduce a código máquina y lo ejecuta.
  Intermedio o mixto: El código fuente se lee a un código intermedio llamado código objeto (**byte code**) en un lenguaje propio, posteriormente un segundo programa llamado intérprete leerá el byte code y lo traducirá a código máquina justo antes de ejecutarse. Esta aproximación pierde eficiencia, pero gana en portabilidad.



Documentación de los programas

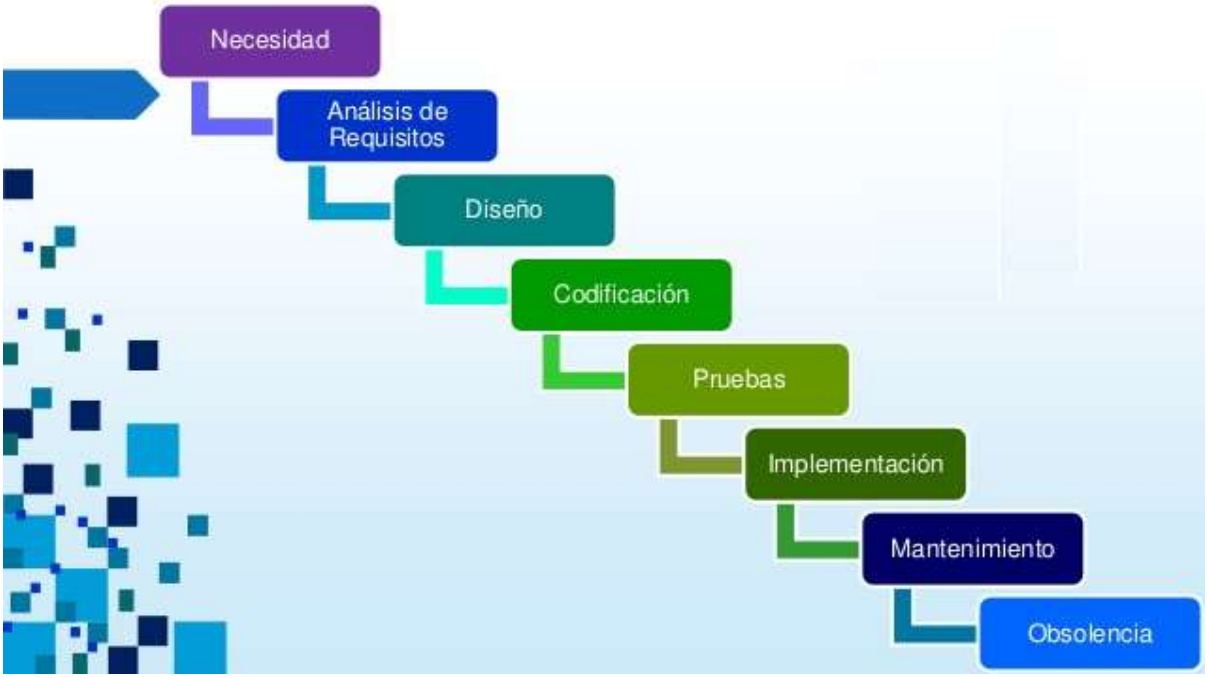
La mayor parte de los proyectos exigen la realización de una planificación previa. Esta planificación debe determinar el modelo de ciclo de vida a seguir, los plazos para completar cada fase y los recursos necesarios en cada momento. Todo esto se debe plasmar en una documentación completa y detallada de toda la aplicación.

La documentación asociada al software puede clasificarse en interna y externa. La documentación interna corresponde a la que se incluye dentro del código fuente de los programas. Nos aclaran aspectos de las propias instrucciones del programa. La documentación externa es la que corresponde a todos los documentos relativos al diseño de la aplicación, a la descripción de la misma y sus módulos correspondientes, a los manuales de usuario y los manuales de mantenimiento.

Ciclo de Vida del Software

Cuando hablamos de ciclo de vida del software nos referimos a las fases por las que pasa el desarrollo de una aplicación o programa desde su concepción hasta su sustitución. Ciclos de vida hay muchos, pero nos vamos a centrar el ciclo clásico que incluye las siguientes fases:

1º. Análisis. En la fase de análisis recogeremos información sobre el funcionamiento del programa y sus diferentes partes, veremos los actores que están implicados y las entradas salidas de datos que necesitaremos para llevar a cabo el desarrollo.
2º. Diseño. En el diseño crearemos las estructuras visuales y de datos necesarias para llevar a cabo la tarea, plasmaremos en documentos la estructura de la aplicación, las librerías y los datos que van a compartir, si es necesario se determinarán las estructuras de los flujos de comunicación, así como de todas las estructuras de datos. Definiremos normas de codificación y estándares para nombrado y almacenamiento, se crearán guían de estilo.
3º. Codificación y pruebas. En esta fase se realiza la codificación de la aplicación, se divide el trabajo en grupos y se realiza en paralelo, avanzaremos desde lo más sencillo a lo más complejo, creando pruebas para cada bloque de código que creemos. Nos aseguraremos que todas las partes funcionan y encajan perfectamente entre ellas.
4º. Implantación o implementación. Se pasa en producción la aplicación, se instala en los servidores y se hacen pruebas de integración, se forma a los usuarios y comienza el uso de la aplicación.
5º. Mantenimiento. En esta fase repararemos aquellos errores que sean notificados por los usuarios generando nuevas versiones de la aplicación, así como daremos asistencia a los usuarios sobre nuestra aplicación. El mantenimiento hará que nuestro software termine por deteriorarse y que tenga que ser sustituido por una nueva versión desarrollada por completo que nos llevará a la primera fase de nuevo.



Ejercicios

1. Escribe un algoritmo para cambiar la rueda de un coche.
2. Escribe un algoritmo para cocinar un plato de pasta.
3. 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.
4. Algoritmo que lea dos números y nos diga cuál de ellos es mayor o bien si son iguales.
5. Algoritmo que lea tres números distintos y nos diga cuál de ellos es el mayor.
6. Hacer un algoritmo que imprima los números pares entre 0 y un número solicitado por pantalla.
7. Un colegio desea saber qué porcentaje de niños y qué porcentaje de niñas hay en el curso actual. Diseñar un algoritmo para este propósito.
8. Una tienda ofrece un descuento del 15% sobre el total de la compra durante el mes de octubre. Dado un mes y un importe, calcular cuál es la cantidad que se debe cobrar al cliente.
9. Escribir un algoritmo que pida al usuario que escriba N o S y lo repita hasta que el usuario pulse otra letra. Debe funcionar tanto en minúsculas como en mayúsculas.
10. Hacer un algoritmo que pida una cadena, la imprima en mayúsculas, después en minúsculas, nos diga la longitud y escriba de una en una sus letras. Cada vez que termina debe preguntar al usuario si desea introducir otra.
11. Hacer un algoritmo que cuente las veces que aparece una determinada letra en una frase que introduciremos por teclado y nos pregunte si deseamos continuar.
12. Desarrollar un algoritmo que nos calcule el cuadrado de los 9 primeros números naturales.
13. Introducir dos números por teclado y mediante un menú, calcule su suma, su resta, su multiplicación o su división, finalizará al pulsar un valor de salida ('s').
14. Teniendo en cuenta que la clave es "eureka", escribir un algoritmo que nos pida una clave. Solo tenemos 3 intentos para acertar, si fallamos los 3 intentos nos mostrara un mensaje indicándonos que hemos agotado esos 3 intentos. Si acertamos la clave, saldremos directamente del programa.
15. Se ha establecido un programa para estimular a los alumnos, el cual consiste en lo siguiente: si el promedio global obtenido por un alumno en el último periodo es mayor o igual que 5, se le hará un descuento del 30% sobre la matrícula y no se le cobrará IVA; si el promedio obtenido es menor que 5 deberá pagar la matrícula completa, la cual debe incluir el 10% de IVA. Hacer un algoritmo que calcule el valor a pagar si se conocen las notas finales de las 6 materias que cursaron.
16. Hacer un algoritmo para ayudar a un trabajador a saber cuál será su sueldo semanal. Se sabe que, si trabaja 40 horas o menos, se le pagará 20€ por hora, pero si trabaja más de 40 horas entonces las horas extras se le pagarán a 25€ por hora.
17. Realiza un reloj digital que pida la hora, minutos y segundos en los que tiene que empezar y a partir de ahí muestre el resto de horas, minutos y segundos.
18. Realizar un juego simple que pide al usuario que adivine un número en 10 intentos.
19. La siguiente tabla muestra un algoritmo paso a paso (lista de instrucciones). Utiliza tres variables A, B y C que inicialmente valen 4, 2 y 3 respectivamente. Calcula el valor de las variables tras ejecutar cada instrucción. Las tres primeras están hechas a modo de ejemplo.

		A	**B**	**C**
	Instrucción	**4**	**2**	**3**
1	A ← B			
2	C ← A			
3	B ← (A + B + C) / 2			
4	A ← A + C			

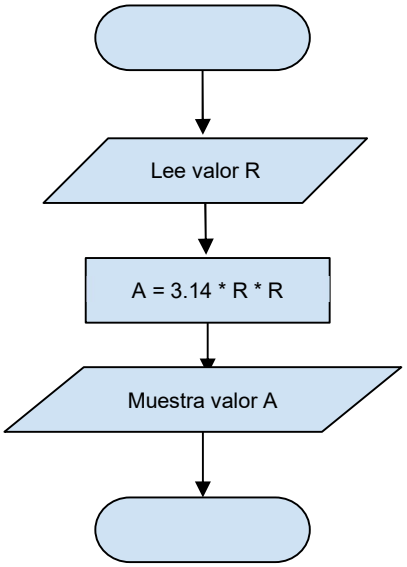
*Recuerda que **X = Y** significa que el valor de Y se copia en X.*

5	C ← B - A			
6	C ← C - A			
7	A ← A * B			
8	A ← A + 3			
9	A ← A % B			
10	C ← C + A			

20. Evalúa las siguientes expresiones:

$((3 + 2)^2 – 15) / 2 * 5$	5 – 2 > 4 AND NOT 0.5 == 1 / 2
Dado x = 1, y = 4, z = 10, pi = 3.14, e = 2.71 2 * x + 0.5 + y – 1 / 5 * z	**Dado x = 1, y = 4, z = 10, pi = 3.14, e = 2.71** (pi * x ^ 2 > y) OR (2 * pi * x <= z)

21. Evalúa las siguientes expresiones:
    1. 24 % 5
    2. 7 / 2 + 2.5
    3. 10.8 / 2 + 2
    4. (4 + 6) * 3 + 2 * (5 - 1)
    5. 5 / 2 + 17 % 3
    6. 7 >= 5 OR 27 <> 8
    7. (45 <= 7) OR NOT (5 >= 7)
    8. 27 % 4 + 15 / 4
    9. 37 / 4 * 4 – 2
    10. (25 >= 7) AND NOT (7 <=2)
    11. ('H' < 'J') AND ('9' <> '7')
    12. 25 > 20 AND 13 > 5
    13. 10 + 4 < 15 - 3 OR 2 * 5 + 1 > 14 – 2 * 2
    14. 4 * 2 <= 8 OR 2 * 2 < 5 AND 4 > 3 + 1
    15. 10 <= 2 * 5 AND 3 < 4 OR NOT (8>7) AND 3 * 2 <= 4 * 2 - 1



22. Dado el siguiente algoritmo descrito en forma de ordinograma, explica brevemente qué hace y cuál sería el resultado mostrado si el valor R leído fuera 2.
23. Dibuja un ordinograma y escribe el pseudocódigo que dé los “buenos días”.
24. Dibuja un ordinograma y escribe el pseudocódigo que calcule y muestre el área de un cuadrado de lado igual a 5.
25. Dibuja un ordinograma y escribe el pseudocódigo que calcule el área de un cuadrado cuyo lado se introduce por teclado.
26. Dibuja un ordinograma y escribe el pseudocódigo que lea dos números, calcule y muestre el valor de sus suma, resta, producto y división.
27. Dibuja un ordinograma y escribe el pseudocódigo que toma como dato de entrada un número que corresponde a la longitud de un radio y nos escribe la longitud de la circunferencia, el área del círculo y el volumen de la esfera que corresponden con dicho radio.
28. Dibuja un ordinograma y escribe el pseudocódigo que dado el precio de un artículo y el precio de venta real nos muestre el porcentaje de descuento realizado.
29. Dibuja un ordinograma y escribe el pseudocódigo que lea un valor correspondiente a una distancia en millas marinas y escriba la distancia en metros. Sabiendo que una milla marina equivale a 1.852 metros.

30. Dibuja un ordinograma y escribe el pseudocódigo de un programa que pide la edad por teclado y nos muestra el mensaje de “Eres mayor de edad” solo si lo somos.
31. Dibuja un ordinograma y escribe el pseudocódigo de un programa que pide la edad por teclado y nos muestra el mensaje de “eres mayor de edad” o el mensaje de “eres menor de edad”.
32. Dibuja un ordinograma y escribe el pseudocódigo que lee dos números, calcula y muestra el valor de su suma, resta, producto y división. (Ten en cuenta la división por cero).
33. Dibuja el ordinograma y escribe el pseudocódigo de un programa que lee 2 números y muestra el mayor.
34. Dibuja el ordinograma y escribe el pseudocódigo de un programa que lee un número y me dice si es positivo o negativo, consideraremos el cero como positivo.
35. Dibuja el ordinograma y escribe el pseudocódigo de un programa que lee dos números y los visualiza en orden ascendente.
36. Dibuja el ordinograma y escribe el pseudocódigo que lea una calificación numérica entre 0 y 10 y la transforma en calificación alfabética, escribiendo el resultado.
    - de 0 a <3 Muy Deficiente.
    - de 3 a <5 Insuficiente.
    - de 5 a <6 Bien.
    - de 6 a <9 Notable
    - de 9 a 10 Sobresaliente
37. Dibuja un ordinograma y escribe un pseudocódigo que recibe como datos de entrada una hora expresada en horas, minutos y segundos que nos calcula y escribe la hora, minutos y segundos que serán, transcurrido un segundo.
38. Dibuja un ordinograma y escribe un pseudocódigo que calcula el salario neto semanal de un trabajador en función del número de horas trabajadas y la tasa de impuestos de acuerdo a las siguientes hipótesis:
    - Las primeras 35 horas se pagan a tarifa normal.
    - Las horas que pasen de 35 se pagan a 1,5 veces la tarifa normal.
    - Las tasas de impuestos son:
    - Los primeros 500 euros son libres de impuestos.
    - Los siguientes 400 tienen un 25% de impuestos.
    - Los restantes un 45% de impuestos.

      Escribir nombre, salario bruto, tasas y salario neto.
39. Dibuja un ordinograma y escribe un pseudocódigo de un programa que muestre los números pares comprendidos entre el 1 y el 200. Utiliza un contador sumando de 1 en 1.
40. Dibuja un ordinograma y escribe un pseudocódigo de un programa que muestre los números desde el 1 hasta un número N que se introducirá por teclado.
41. Dibuja un ordinograma y escribe un pseudocódigo de un programa que lea un número positivo N y calcule y visualice su factorial N! Siendo la factorial:
    - 0! = 1
    - 1! = 1
    - 2! = 2 * 1
    - 3! = 3 * 2* 1
    - N! = N * (N-1) * (N-2) * … * 1
42. Dibuja un ordinograma y escribe un pseudocódigo de un programa que lea 100 números no nulos y luego muestre un mensaje indicando cuántos son positivos y cuantos negativos.
43. Dibuja un ordinograma y escribe un pseudocódigo de un programa que lea una secuencia de números no nulos hasta que se introduzca un 0, y luego muestre si ha leído algún número negativo, cuantos positivos y cuantos negativos.
44. Dibuja un ordinograma y escribe un pseudocódigo de un programa que calcula y escribe la suma y el producto de los 10 primeros números naturales.

Entregar

45. Dibuja un ordinograma y escribe un pseudocódigo de un programa que lee una secuencia de notas (con valores que van de 0 a 10) que termina con el valor -1 y nos dice si hubo o no alguna nota con valor 10.

46. Dibuja un ordinograma y escribe un pseudocódigo de un programa que suma independientemente los pares y los impares de los números comprendidos entre 100 y 200, y luego muestra por pantalla ambas sumas.
47. Dibuja un ordinograma y escribe un pseudocódigo de un programa que calcule el valor A elevado a B (A^B) sin hacer uso del operador de potencia (^), siendo A y B valores introducidos por teclado, y luego muestre el resultado por pantalla.
48. Dibuja un ordinograma y escribe un pseudocódigo de un programa donde el usuario "piensa" un número del 1 al 100 y el ordenador intenta adivinarlo. Es decir, el ordenador irá proponiendo números una y otra vez hasta adivinarlo (el usuario deberá indicarle al ordenador si es mayor, menor o igual al número que ha pensado).
49. Dibuja un ordinograma y escribe un pseudocódigo de un programa que dada una cantidad de euros que el usuario introduce por teclado (múltiplo de 5 €) mostrará los billetes de cada tipo que serán necesarios para alcanzar dicha cantidad (utilizando billetes de 500, 200, 100, 50, 20, 10 y 5). Hay que indicar el mínimo de billetes posible. Por ejemplo, si el usuario introduce 145 el programa indicará que será necesario 1 billete de 100 €, 2 billetes de 20 € y 1 billete de 5 € (no será válido por ejemplo 29 billetes de 5 que, aunque sume 145 € no es el mínimo número de billetes posible).

Ampliación

1. Implementar el algoritmo de Euclides para el cálculo del máximo común divisor.



2. Implementar un algoritmo que escriba en la pantalla tantos signos # como el valor del cociente de sus dos argumentos, y tantos signos $ como el resto de la división entera.