Financiado por la Unión Europea NextGenerationEU
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Plan de Recuperación, Transformación y Resiliencia
GENERALITAT VALENCIANA Conselleria d'Educació, Cultura i Esport
GVANEXT Fondos Next Generation en la Comunitat Valenciana

1DAM TEMA 1

IES ANTONIO SEQUEROS ALMORADÍ

Fundamentos de Programación

Versión: 230913.2158

IES Antonio Sequeros
Veritas Nostra Lux, Sapientia Nostrum Gaudium

ÍNDICE

1. INTRODUCCIÓN A LA PROGRAMACIÓN

1.1. ¿Por qué estudiar programación?

Las ciencias de la computación son una de las formas más increíbles de arte que hay en el mundo. No sólo son increíblemente expresivas - posibilitando un número infinito de combinaciones de palabras, imágenes e ideas - sino que cuando son bien utilizadas, dan como resultado un producto útil y funcional que puede entretener e informar a las masas.

Generalmente, cuando las personas piensan en las ciencias de la computación, suelen imaginar la programación. Es común asumir que se trata de un mundo lleno de gente que se sienta sola frente a su computadora todo el día, mirando fijamente grandes pantallas y tomando café. Sin embargo, la realidad es diferente, las ciencias de la computación son una actividad colaborativa y cautivadora, que incluye mucho más que los unos y los ceros por los que se ha vuelto famosa.

El arte de las ciencias de la computación comienza con un problema que necesita ser resuelto. Tal vez, hay demasiada información producida por un estudio para que un sólo ser humano la pueda comprender en el período de una vida.

Una vez que se ha identificado un problema, puede parecer muy complicado de resolver. Es por esto que los científicos de la computación aprenden a mirar partes individuales de un problema, en lugar pensar en todo el problema al mismo tiempo. Dividir una tarea en partes manejables es una excelente manera de hacer progresos a través de pequeñas soluciones.

Averiguar como algo debería funcionar es diferente a verlo ya funcionando. En estos casos las simulaciones por computadora entran en juego. Cuando intentas simular una experiencia del mundo real, o simular una experiencia imposible, las computadoras se vuelven una plataforma extremadamente útil.

Si tu solución corre en una computadora, es de esperar que las personas puedan interactuar con esta. Esto introduce la necesidad de tener atractivas y artísticas interfaces. Un programa debe proveer elementos gráficos, menús, botones, etc. Todos estos detalles deben ser debidamente pensados para su incorporación.

Así que, como puedes ver, los científicos de la computación no son sólo programadores. Ellos son personas creativas, que resuelven problemas de distinta índole, jugando a ser al mismo tiempo psicólogos, artistas, autores, y también, como es de esperar, científicos. Todas las industrias necesitan de científicos de la computación. Son necesitados en biología, videojuegos, escuelas, sistemas de salud, servicios públicos y, en general, en cualquier lugar en donde se necesite de innovación.

Code (www.code.org) es una organización sin ánimo de lucro fundada en el año 2013 que tiene como misión fomentar el acceso de los estudiantes a las ciencias de la computación. Su visión es que todo estudiante debe tener la oportunidad de aprender Ciencias de la Computación, y que ésta disciplina debe ser parte fundamental del currículo educativo tal como las matemáticas, la biología, la química o la física.

1.2. Programa

Secuencia de acciones (**instrucciones**) escritas en un determinado lenguaje de programación que el ordenador debe ejecutar para realizar una tarea. Una **instrucción** es un conjunto elemental (indivisible) de acciones a desarrollar por un ordenador.

Es decir, **programa**, es un conjunto de instrucciones que entiende el ordenador con el objetivo de resolver un problema

Por lo que podemos definir **programar** como la tarea de describir o indicar al ordenador lo que debe de hacer para resolver un problema

Los programas pueden ser de dos tipos:

- ✓ **Código fuente** o programa escrito en algún lenguaje de programación cercano al lenguaje humano. No es directamente ejecutable por la computadora.
- ✓ **Código objeto** o programa directamente ejecutable por un ordenador (binario).

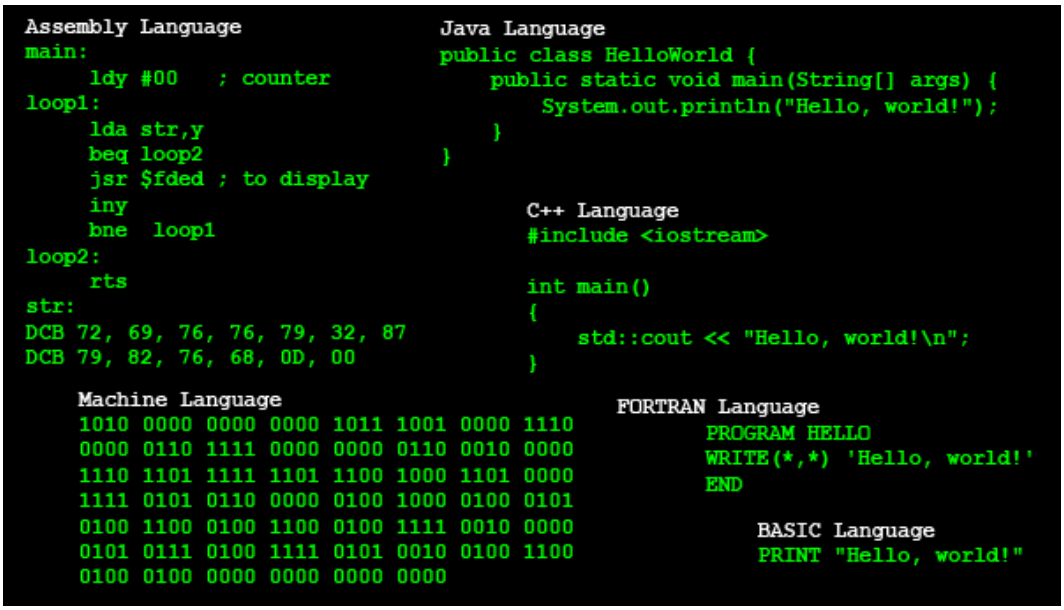
1.3. Lenguajes de programación

Un **lenguaje de programación** consta de un conjunto de símbolo y reglas sintácticas y semánticas, que nos indican como debemos escribir las instrucciones de un programa para que un dispositivo las pueda ejecutar.

Un lenguaje de programación es cualquier sistema de notación que permite expresar programas.

Existen varios niveles:

- ✓ **Lenguaje máquina**. Formado por las instrucciones escritas en código máquina (binario) que pueden ser inmediatamente obedecidas por la máquina sin necesidad de traducción, ya que es el único lenguaje que entiende.
- ✓ **Lenguaje ensamblador**. Formado por instrucciones de tipo simbólico que se aproximan al lenguaje humano. Precisan de un programa traductor denominado ensamblador. Cada instrucción equivale, al traducirse, a una sola instrucción en lenguaje máquina.
- ✓ **Lenguajes de alto nivel**. Emplean terminología fácilmente comprensible y que se aproxima más o menos al lenguaje humano. Cada instrucción se traduce a varias en lenguaje máquina. También precisan programas traductores. Ejemplos: BASIC, FORTRAN, COBOL, PASCAL, C, JAVA, PHP…



Ejemplo lenguaje máquina:

Instrucción	**Significado**
A0 2F	Acceder a la celda de memoria 2F
3E 01	Copiarlo el registro 1 de la ALU
A0 30	Acceder a la celda de memoria 30
3E 02	Copiarlo en el registro 2 de la ALU
1D	Sumar
B3 31	Guardar el resultado en la celda de memoria 31

Ejemplo lenguaje ensamblador:

Instrucción	**Significado**
READ 2F	Acceder a la celda de memoria 2F
REG 01	Copiarlo el registro 1 de la ALU
READ 30	Acceder a la celda de memoria 30
REG 02	Copiarlo en el registro 2 de la ALU
ADD	Sumar
WRITE 31	Guardar el resultado en la celda de memoria 31

1.4. Traductores

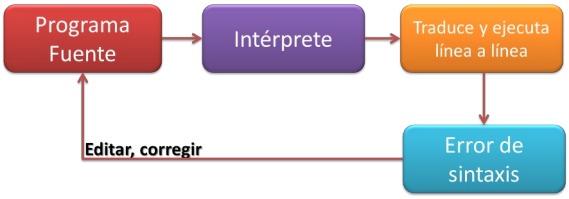
El único lenguaje que entiende directamente el ordenador es el lenguaje máquina. Para el resto se necesitan programas traductores. Para traducir los programas fuente se utilizan las siguientes herramientas:

- ✓ **Ensambladores**. Se emplean para traducir programas en ensamblador, correspondiendo una instrucción fuente a una instrucción máquina.

- ✓ **Compiladores**. Se emplean para traducir programas en lenguajes de alto nivel, correspondiendo una instrucción fuente a varias instrucciones máquina. El compilador traduce todo el programa y después se ejecuta.

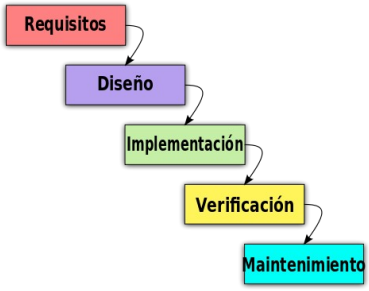
  - Ejemplos de lenguajes compilados, que necesitan un compilador, serían: C, C++, C#, Java, Pascal, ...

- ✓ **Intérpretes**. Tienen la misma utilidad que los compiladores, pero cada instrucción se ejecuta tras ser traducida, sin esperar a haber traducido todo el programa.



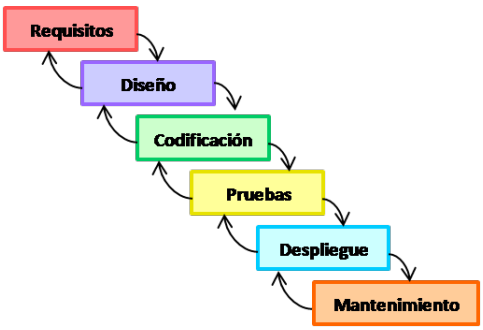
  - Ejemplo de Lenguajes interpretados, que utilizan un interprete, serían: PHP, Javascript, Python, ...
  - Por regla general, los intérpretes ejecutarán los programas más lentamente, pues al tiempo de ejecución del código de máquina se suma el que consume la traducción simultánea.

1.5. Fases del proceso de programación



- ✓ **Análisis de requisitos**. Se analizan las necesidades de los usuarios finales para determinar los objetivos a cubrir y se obtiene un documento con la especificación de requisitos de lo que debe hacer el software sin entrar en detalles internos.
- ✓ **Diseño**. Se descompone el problema en partes más pequeñas y se obtienen los algoritmos a emplear para resolver el problema.
- ✓ **Implementación**. Partiendo del algoritmo, se construye el programa correspondiente en un determinado lenguaje de programación (código fuente) y se van realizando pruebas y ensayos para corregir errores.
- ✓ **Verificación**. Comprobar el funcionamiento del programa final, resolver los posibles errores y obtener los resultados.
- ✓ **Mantenimiento**. En ocasiones es necesarios hacer algún cambio, ajuste o complementar al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

Como cada una de las fases puede tener retroalimentación, podemos representarlo con mas detalle en el siguiente gráfico:



1.6. Algoritmos

Un **algoritmo** es la secuencia precisa, finita y ordenada de pasos lógicos necesarios para resolver un problema.

Es independiente del lenguaje de programación que posteriormente se vaya a utilizar. Cada paso del algoritmo debe estar bien definido, de forma clara y precisa.

Dos programas que resuelven el mismo problema expresados en el mismo o en diferentes lenguajes de programación pero que siguen, en lo fundamental, el mismo procedimiento, son dos implementaciones del mismo algoritmo.

Una aproximación para empezar a comprender el concepto de algoritmo sería una receta de cocina:

FREÍR UN HUEVO

Encender el fuego.
Poner una sartén en el fuego.
Poner aceite en la sartén.
Esperar a que se caliente el aceite.
Cascar el huevo.
Echar la clara y la yema dentro de la sartén.
Si no hay que observar una dieta
    echar sal en el huevo.
Mientras el huevo no esté cocido
    echar aceite caliente por encima de él.
Sacar el huevo de la sartén.
Escurrir el aceite sobrante.
Poner el huevo en plato.
Servirlo.

En principio ya está: con la receta, sus ingredientes y los útiles necesarios somos capaces de cocinar un plato. Bueno, no del todo cierto, pues hay unas cuantas cuestiones que no quedan del todo claras en nuestra receta:

- ¿Cuánta sal utilizamos?: ¿una pizca?, ¿un kilo?
- ¿Cuánto aceite hemos de verter en la sartén?: ¿un centímetro cúbico?, ¿un litro?
- ¿Cuál es el resultado del proceso?, ¿la sartén con el huevo cocinado y el aceite?

Los algoritmos tienen una connotación más precisa o formal, deberían de cumplir las siguientes características:

- ✓ **Precisión**, no debe de ser ambiguo. Por eso la receta de cocina no sería un algoritmo, sino más bien un método, puesto que no especifica cuanta sal habría que ponerle, ni cuando está cocido el huevo.
- ✓ **Determinismo**, debe de responder siempre igual bajo las mismas condiciones.
- ✓ **Finitud**, su descripción debe de ser finita.

1.6.1 Ejemplo de Algoritmos

Veamos algunos algoritmos sobre la comprobación de si un número es primo:

Problema: Dado un número n, que es un entero mayor que uno, determinar si es o no primo.

Algoritmo 1:

```
Considerar todos los números comprendidos entre 2 y n (excluido).
Para cada número de dicha sucesión comprobar si dicho número divide al
número n.
Si ningún número divide a n, entonces n es primo.
```

Algoritmo 2:

```
Paso 1. Sea i un número entero de valor igual a 2.
Paso 2. Si i es mayor o igual a n parar, n es primo.
Paso 3. Comprobar si i divide a n, entonces parar, n no es primo.
Paso 4. Reemplazar el valor de i por i + 1, volver al Paso 2.
```

Algoritmo 3:

```
Paso 1. Si n vale 2 entonces parar, n es primo.
Paso 2. Si n es múltiplo de 2 acabar, n no es primo.
Paso 3. Sea i un número entero de valor igual a 3.
Paso 4. Si i es mayor que la raíz cuadrada positiva de n parar, n es primo.
Paso 5. Comprobar si i divide a n, entonces parar, n no es primo.
Paso 6. Reemplazar el valor de i por i + 2, volver al Paso 4.
```

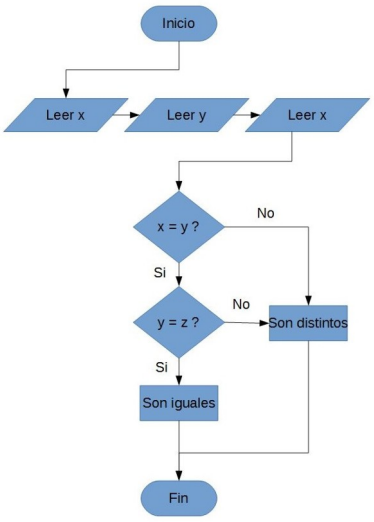
1.7. Diagramas de Flujo (Ordinogramas)

- ✓ Se trata de un diagrama que se utiliza para realizar algoritmos.
- ✓ Muestra la secuencia lógica y detallada de las operaciones que necesitamos para la realización de un programa de forma gráfica.
- ✓ Un ordinograma debe ser independiente del lenguaje de programación que se utilice.

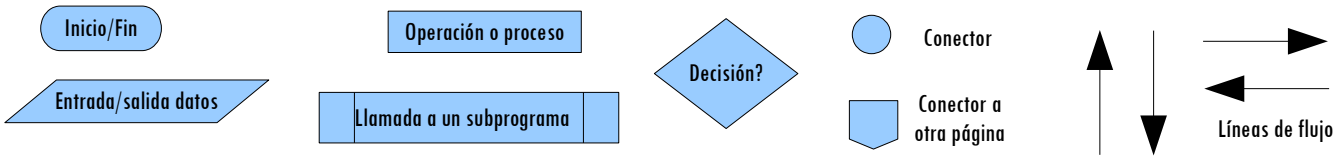
1.7.1 Estructura de un ordinograma

Todo ordinograma debe estar compuesto de:

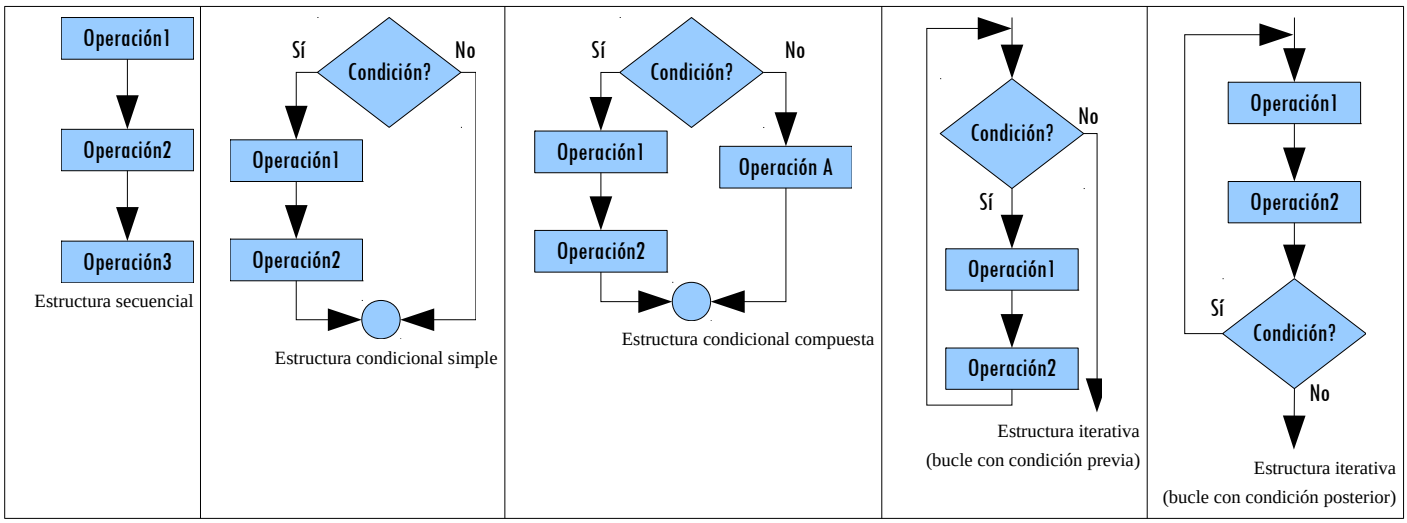
- Un símbolo de inicio de ejecución del programa
- La secuencia de operaciones necesarias para el correcto funcionamiento del programa. Las operaciones seguirán un orden (de arriba abajo y de izquierda a derecha).
- Un símbolo que indique el final del programa.



1.7.2 Símbolos básicos de un ordinograma



1.7.3 Estructuras básicas en ordinogramas



1.7.4 Reglas a la hora de hacer ordinogramas

- ✓ Todos los símbolos utilizados deben estar unidos por líneas de flujo.
- ✓ No se pueden cruzar las líneas de flujo
- ✓ A un símbolo de proceso pueden llegarle varias líneas de flujo pero solo puede salir una de él.
- ✓ Al símbolo de inicio no puede llegarle ninguna línea de flujo
- ✓ De un símbolo de fin no puede salir ninguna línea de flujo pero si le pueden llegar varias.

1.8. Pseudocódigo

El **pseudocódigo** describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de programación y palabras clave que definen las estructuras básicas. Su objetivo es permitir que el programador se centre en los aspectos lógicos de la solución a un problema.

Existen varias ventajas de utilizar un pseudocódigo a un diagrama de flujo:

- ✓ Permite representar de forma fácil operaciones repetitivas complejas.

- ✓ Es más sencilla la tarea de pasar de pseudocódigo a un lenguaje de programación formal.
- ✓ Siguiendo las reglas de indentación se puede observar claramente los niveles en la estructura del programa.

1.8.1 Estructura de un algoritmo en pseudocódigo

Cuerpo	**INICIO** **DATOS** **parámetros** **constantes** **variables...** **ALGORITMO** **acciones** **FIN**

Contiene el propio algoritmo y se divide en dos bloques

Bloque de datos: contiene la definición de los diferentes datos que utilizará el algoritmo: parámetros, variables, constantes, estructuras...

Bloque de acciones: describe de forma detallada y ordenada las operaciones a realizar en el algoritmo.

1.8.2 Estructuras básicas en pseudocódigo

Estructura secuencial	Estructura condicional simple	Estructura condicional compuesta	Estructura iterativa (bucle con condición previa)	Estructura iterativa (bucle con condición posterior)
acción 1 **accion 2** **...** **acción N**	**si** condición acción 1 acción 2 ... acción N **fin si**	**si** condición acción 1 acción 2 ... acción N **si no** acción 1 acción 2 ... acción N **fin si**	**mientras** condición **hacer** acción 1 acción 2 ... acción N **fin mientras**	**hacer** acción 1 acción 2 ... acción N **hasta** condición

Cuestiones

1. Crear los siguientes ordinogramas y su correspondiente pseudocódigo:

1.a) Dados dos números y una de las operaciones básicas, realizar dicho cálculo.

1.b) Dados dos números por teclado, realizar las operaciones básicas con ellos (suma, resta, multiplicación y división).

1.c) Calcular el área de un triángulo dadas su base y su altura.

1.d) Dado el radio calcular el perímetro del círculo ($2 \cdot \pi \cdot r$), el área de la circunferencia ($\pi \cdot r^2$) y el área ($4 \cdot \pi \cdot r^2$) y volumen ($\frac{3}{4} \cdot \pi \cdot r^3$) de la esfera que genera.

1.e) Dados dos números por teclado, mostrar el máximo.

1.f) Dado un número por teclado, indicar si es par o impar.

1.g) Dado un número por teclado, mostrar desde 0 hasta dicho número.

1.h) Dada una cantidad en euros, indicar la cantidad mínima de billetes y monedas para representarla, teniendo en cuenta que actualmente se utilizan billetes de 500€, 200€, 100€, 50€, 20€, 10€, 5€ y monedas de 2€, 1€, 0,5€, 0,2€, 0,1€, 0,05€, 0,02€ y 0,01€.

1.i) Dado un número por teclado, mostrar todos los números pares desde 0 hasta dicho número.

1.j) Dado un número por teclado, sumar desde 0 hasta dicho número.

1.k) Dado un número por teclado, sumar desde 0 hasta dicho número sólo los números pares.

1.l) Dado un número por teclado, calcular su factorial (7!=7*6*5*4*3*2*1).

1.m) Dados n números positivos, calcular el máximo y el mínimo de ellos.

1.n) Dado un número, calcular su tabla de multiplicar de cero a diez.

1.o) Crear un programa que calcule todas las tablas de multiplicar hasta un número dado.

```
*
**
***
****

**********
**********
**********
**********
```

1.p) Dado un número, crear una fila con ese número de asteriscos.
n = 8 => * * * * * * * *

1.q) Dado dos números, crear un rectángulo con asteriscos.
x = 4 , y = 8 =>

```
********
********
********
********
```

1.r) Dado un número, crear un triángulo con ese número de filas de asteriscos.
x = 4 =>

```
*
**
***
****
```

1.s) Dado un número entero por teclado, calcular sus divisores.

1.9. Paradigmas de Programación

Un paradigma de programación representa un enfoque particular o filosofía para diseñar soluciones a los problemas planteados.

Básicamente podemos dividirlos en dos Imperativa y Declarativa, la gran mayoría son variantes de ellas.

Paradigmas:

- Programación Imperativa o por procedimientos: es el más usado en general, se basa en dar instrucciones al ordenador de como hacer las cosas en forma de algoritmos.
- Programación Orientada a objetos: está basada en el imperativo, pero encapsula elementos denominados objetos que incluyen tanto variables como funciones.
- Programación Dirigida por eventos: en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.
- Programación Declarativa: está basado en describir el problema declarando propiedades y reglas que deben cumplirse, en lugar de instrucciones. Ejemplo de lenguajes sería Lisp y Prolog.

1.9.1 Técnicas Imperativa o por Procedimientos

- ✓ **Programación Estructurada**: Es una técnica que permite crear programas más sencillos, fáciles de entender, de mantener y de probar. El principal inconveniente de este método de programación, es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático su manejo. Un programa estructurado es un programa con un solo punto de entrada y un solo punto de salida que se basa en el uso de tres estructuras lógicas de control:
  - **Secuencia**: Sucesión simple de dos o más operaciones.
  - **Selección**: Bifurcación condicional de una o más operaciones.
  - **Interacción**: Repetición de una o más operaciones mientras se cumple una condición.
- ✓ **Programación modular**: Resuelve el principal problema de la programación estructurada dividiendo el programa en un conjunto de módulos, cada uno de los cuales desempeña una tarea necesaria para el correcto funcionamiento del programa global. Los módulos son interdependientes, y son codificados y compilados por separado.
- ✓ **Programación orientada a objetos**: La programación orientada a objetos expresa un programa como un conjunto de objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

1.9.2 Programación Orientada a Objetos

- ✓ **Clase:** Es una descripción del estado y comportamiento de objetos similares. Sirve como modelo para la creación de objetos. Es una entidad estática.
- ✓ **Objeto**: Entidad provista de un conjunto de propiedades y métodos, moldeada según la clase a la que pertenece. Cuando creamos un objeto tenemos que especificar la clase a partir de la cual se creará (**instanciar)**. Cada objeto tiene su propio estado, pero todos los objetos de una misma clase comparten su comportamiento. Son entidades dinámicas.
- ✓ **Propiedades**: Conjunto de atributos o datos que describen el estado de un objeto. Cuando definimos una propiedad normalmente especificamos su nombre y su tipo.
- ✓ **Métodos**: Describen el comportamiento del objeto ante determinados eventos, definiendo qué operaciones se pueden realizar con él.
- ✓ **Mensajes en objetos**: Un mensaje en un objeto es la acción de efectuar una llamada a un método.
- ✓ **Eventos**: Suceso en el sistema que provoca el envío de un mensaje al objeto pertinente, desencadenando una reacción sobre él al ejecutar el método asociado a dicho evento (en caso de existir).

1.9.3 Programación Dirigida por Eventos

La programación dirigida u orientada a eventos supone una complicación añadida debido a que el flujo de ejecución del software escapa al control del programador. En cierta manera podríamos decir que en la programación clásica el flujo estaba en poder del programador y era este quien decidía el orden de ejecución de los procesos, mientras que en programación orientada a eventos, es el usuario el que controla el flujo y decide.

Pongamos como ejemplo de la problemática existente, un menú con dos botones, botón 1 y botón 2. Cuando el usuario pulsa botón 1, el programa se encarga de recoger ciertos parámetros que están almacenados en un fichero y calcular algunas variables. Cuando el usuario pulsa el botón 2, se le muestran al usuario por pantalla dichas variables. Es sencillo darse cuenta de que la naturaleza indeterminada de las acciones del usuario y las características de este paradigma pueden fácilmente desembocar en el error fatal de que se pulse el botón 2 sin previamente haber sido pulsado el botón 1. Aunque esto no pasa si se tienen en cuenta las propiedades de dichos botones, haciendo inaccesible la pulsación sobre el botón 2 hasta que previamente se haya pulsado el botón 1.

2. ELEMENTOS DE UN PROGRAMA

Entendemos por elemento (objeto) de un programa todo aquello que pueda ser manipulado por las instrucciones. En ellos se almacenarán tanto los datos de entrada como los de salida (resultados).

Sus atributos son:

- Nombre: el identificador del objeto.
- Tipo: conjunto de valores que puede tomar.
- Valor: elemento del tipo que se le asigna.

2.1. Constantes

Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo.

Por ejemplo:

- pi = 3.14.1592
- e = 2.718281

También son utilizadas las constantes para facilitar la modificabilidad de los programas, es decir, para hacer más independientes ciertos datos del programa. Por ejemplo, supongamos un programa en el que cada vez que se calcula un importe al que se debe sumar el IVA utilizáramos siempre el valor 0.16, en caso de variar este índice tendríamos que ir buscando a lo largo del programa y modificando dicho valor, mientras que si le damos nombre y le asignamos un valor, podremos modificar dicho valor con mucha más facilidad.

2.2. Variables

Son objetos cuyo valor puede ser modificado a lo largo de la ejecución de un programa.

Por ejemplo: una variable para calcular el área de una circunferencia determinada, una variable para calcular una factura, etc.

2.3. Expresiones

Las expresiones según el resultado que produzcan se clasifican en:

- Numéricas: Son las que producen resultados de tipo numérico. Se construyen mediante los operadores aritméticos.

  Por ejemplo:

  ```
  pi * sqr(x)
  2*x)/3
  ```

- Alfanuméricas: Son las que producen resultados de tipo alfanumérico. Se construyen mediante operadores alfanuméricos.

  Por ejemplo:

  ```
  "Don " + "José"
  ```

- Booleanas o lógicas: Son las que producen resultados de tipo Verdadero o Falso. Se construyen mediante los operadores relacionales y lógicos.

  Por ejemplo:

```
a < 0
(a > 1) and (b < 5)
```

2.4. Operadores

Son símbolos que hacen de enlace entre los argumentos de una expresión.

2.4.1 Relacionales

Se usan para formar expresiones que al ser evaluadas devuelven un valor booleano: verdadero o falso.

Operador	Definición
<	Menor que
>	Mayor que
=	Igual que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto que

Ejemplos:

Expresión	Resultado
A' < 'B'	Verdadero, ya que en código ASCII la A está antes que la B
1 < 6	Verdadero
10 < 2	Falso

2.4.2 Aritméticos

Se utilizan para realizar operaciones aritméticas.

Símbolo	*Significado*	*Ejemplo*	*Resultado*
+	Suma	a = 10 + 5	a es 15
-	Resta	a = 12 - 7	a es 5
-	Negación	a = -5	a es -5
*	Multiplicación	a = 7 * 5	a es 35
**	Exponente	a = 2 ** 3	a es 8
/	División	a = 12.5 / 2	a es 6.25
//	División entera	a = 12.5 / 2	a es 6.0
%	Modulo	a = 27 % 4	a es 3

Operador	Definición
+	Suma
-	Resta
*	Multiplicación
^	Potencia
/	División
%	Resto de la división

Ejemplos:

Expresión	Resultado
3 + 5 - 2	6
24 % 3	0
8 * 3 - 7 / 2	26

2.4.3 Lógicos o booleanos

La combinación de expresiones con estos operadores producen el resultado verdadero o falso.

Operador	Definición
No	Negación
Y	Conjunción
O	Disyunción

El comportamiento de un operador lógico se define mediante su correspondiente tabla de verdad, en ella se muestra el resultado que produce la aplicación de un determinado operador a uno o dos valores lógicos. Las operaciones lógicas más usuales son:

- NO lógico (NOT) o negación:

A	NOT A
V	F
F	V

El operador NOT invierte el valor: Si es verdadero (V) devuelve falso (F), y viceversa.

- O lógica (OR) o disyunción:

A	B	A OR B
V	V	V
V	F	V
F	V	V
F	F	F

El operador OR devuelve verdadero (V) si alguno de los dos valores es verdadero.
De lo contrario, devuelve Falso (F).

- Y lógica (AND) o conjunción:

A	B	A AND B
V	V	V
V	F	F
F	V	F
F	F	F

El operador AND devuelve Verdadero (V) solo si ambos valores son verdaderos.
En cualquier otro caso devuelve Falso (F).

Ejemplos (Suponiendo que a<b):

Expresión	Resultado
9 = (3*3)	Verdadero
3 <> 2	Verdadero
9 = (3*3) Y 3 <> 2	Verdadero
3 > 2 Y b < a	Verdadero Y Falso = Falso
3 > 2 O b < a	Verdadero O Falso = Verdadero
no(a < b)	No Verdadero = Falso
5 > 1 Y NO(b < a)	Verdadero Y no Falso = Verdadero

2.4.4 Paréntesis ()

Anidan expresiones.

Ejemplos:

```
Operación (3*2) + (6/2) Resultado 9
```

2.4.5 Operador Alfanumérico (+)

Une datos de tipo alfanumérico. También llamado concatenación.

Ejemplos:

Expresión	Resultado
"Ana " + "López"	Ana López
"saca" + "puntas"	sacapuntas

2.4.6 Orden de evaluación de los operadores

A la hora de resolver una expresión, el orden a seguir es el siguiente:

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
División entera	//	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	not	Unario	—	6
Conjunción	and	Binario	Por la izquierda	7
Disyunción	or	Binario	Por la izquierda	8

1. Paréntesis.

2. Potencia ^

3. Multiplicación y división * /

4. Sumas y restas + -

5. Concatenación +

6. Relacionales < <= > >= etc.

7. Negación NOT

8. Conjunción AND

9. Disyunción OR

La evaluación de operadores de igual orden se realiza de izquierda a derecha. Este orden de evaluación tiene algunas modificaciones en determinados lenguajes de programación.