

Programación

Undécimo de Informática A

Introducción a la Programación y Algoritmos

1. Introducción

1.1 Cómo "piensa" una computadora

Aunque las computadoras cada vez más complejas pueden razonar mejor, una computadora solamente hace lo que se le ordena. Ahí es donde aparece el software. Un programa de software es una sucesión de instrucciones. Puede ser simple, como para que puedas calcular sumas o restas, o complicado, como para predecir la trayectoria de un cohete lanzado al espacio.

A veces, un error en el programa de la computadora (BUGS) provoca un resultado inesperado y, generalmente, desagradable. Para los ingenieros de software es extremadamente difícil diseñar un programa perfecto. Por eso la mayoría de los programas inicialmente contienen errores y la depuración se convierte en una tarea diaria para los programadores.

Depurar generalmente es una tarea difícil y agotadora. El elemento más importante para depurar un problema es la capacidad del programador para hacerlo. Sin embargo, la dificultad de la depuración del software varía considerablemente de acuerdo con el lenguaje de programación usado y a las herramientas utilizadas como depuradores.

Los depuradores son herramientas de software que permiten que el programador pueda controlar la ejecución de un programa, detenerla, reiniciarla, ejecutarla en cámara lenta, cambiar los valores de la memoria y en algunos casos, retroceder en el tiempo.

Finalmente, una computadora debe saber cómo comunicarse. Para ello es necesario conectarla a dispositivos periféricos. En el mundo digital actual, la computadora no sólo está equipada con los elementos básicos (teclado, mouse y pantalla) sino también con un módem, una cámara y una impresora.

Muchas, y en realidad probablemente la mayoría de las computadoras, no cuentan con teclado ni pantalla sino que están incorporadas en diferentes dispositivos. Así, en un automóvil hay una computadora que detecta el funcionamiento y estado del motor y controla diferentes funciones.

1.2 Definición de lenguaje de programación

El lenguaje de programación es la combinación de símbolos y reglas que permiten la elaboración de programas con los cuales la computadora puede realizar tareas o resolver problemas de manera eficiente.

2. Algoritmos y su representación

Ejemplo 1: Cambiar una llanta

INICIO

1. Aflojar los tornillos de la llanta pinchada con la llave inglesa.
2. Ubicar el gato mecánico en su sitio.
3. Levantar el gato hasta que la rueda pinchada pueda girar libremente.
4. Quitar los tornillos y la rueda pinchada.
5. Poner rueda de repuesto y los tornillos.
6. Bajar el gato hasta que se pueda liberar.
7. Sacar el gato de su sitio.
8. Apretar los tornillos con la llave inglesa.

FIN

Ejemplo 2: Ir al colegio

PROBLEMA: Un estudiante se encuentra en su casa (durmiente) y debe ir al colegio (a tomar la clase de programación), ¿qué debe hacer el estudiante?

INICIO

1. Dormir.
2. Haga 1 hasta que suene el despertador (o lo llame la mamá).
3. Mirar la hora.
4. ¿Hay tiempo suficiente?
5. Si hay, entonces:
 6. Bañarse.
 7. Vestirse.
 8. Desayunar.
9. Sino:
 10. Vestirse.
 11. Despedirse de la mamá y el papá.
12. ¿Hay tiempo suficiente?
13. Si: Caminar hasta la parada de buses.
14. Sino: Correr a la parada de buses.
15. Hasta que pase un bus para el colegio haga:
 16. Esperar el bus.
 17. Ver a las demás personas que esperan un bus.
 18. Tomar el bus.
19. Mientras no llegue al colegio haga:
 20. Seguir en el bus.
 21. Bajarse.
 22. Entrar al colegio.

FIN

3. Lenguajes de programación

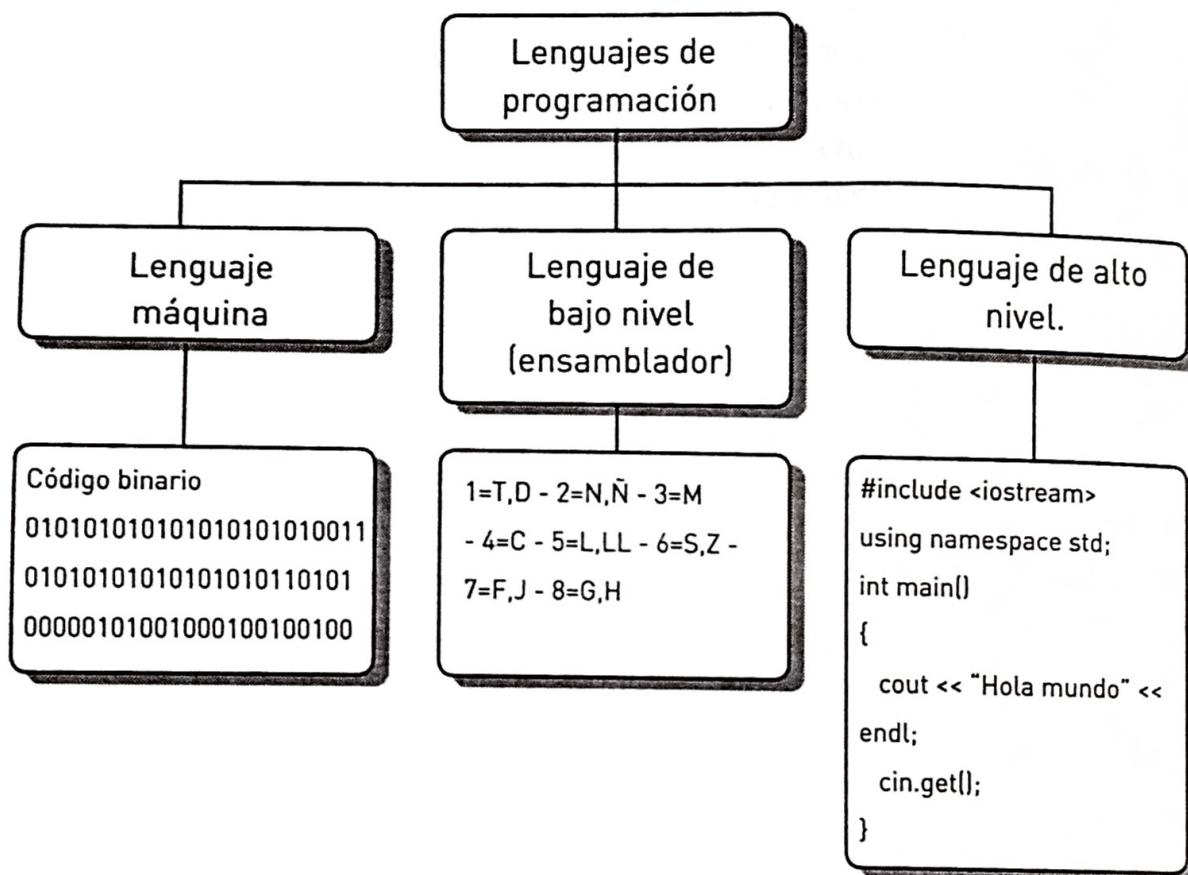
3.1 Clasificación de lenguajes de programación

1. Lenguaje máquina: Las instrucciones son directamente entendibles por la computadora y no necesitan traductor para que el CPU (unidad de procesamiento central) pueda entender y ejecutar el programa. Utiliza un código binario (0 y 1), se basa en bits (abreviatura inglesa de dígitos binarios).

2. Lenguaje de bajo nivel (ensamblador): Las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos.

3. Lenguaje de alto nivel: Es semejante al lenguaje humano (en general en inglés), lo que facilita la elaboración y comprensión del programa. Por ejemplo Basic, Pascal, Cobol, Fortran, C, C++, etc.

3.2 Diagrama de clasificación de lenguajes



4. Definición y características de algoritmos

4.1 Definición de Algoritmo

Algoritmo se define como un conjunto de instrucciones que la computadora debe seguir para resolver un problema. La palabra algoritmo se deriva de la traducción al latín del nombre Muhammad Musa Alkhawarizmi, un matemático y astrónomo árabe que en el siglo IX escribió un tratado sobre manipulación de números y ecuaciones.

Informalmente, un algoritmo es cualquier procedimiento de cálculo bien definido que toma algún valor, o conjunto de valores, como entrada y produce un cierto valor, o conjunto de valores, llamados salida. Así, un algoritmo es una secuencia de pasos de cálculo que transforman cálculos de entrada en salidas. Podemos ver un algoritmo como la herramienta para dar solución a un problema, mediante un cálculo bien especificado.

4.2 Características de los Algoritmos

- Entradas: Un algoritmo tiene cero o más entradas (cantidades que se le dan inicialmente antes de que comience su ejecución).
- Salidas: Un algoritmo tiene una o más salidas (cantidades que tienen una relación específica con las entradas).

4.3 Todo algoritmo debe ser:

1. **Finito** en tamaño o número de instrucciones (tiene un primer paso y un último paso) y tiempo de ejecución (debe terminar en algún momento). Por lo tanto, debe tener un punto particular de inicio y fin.
2. **Preciso**. Debe tener un orden entre los pasos.
3. **Definido**. No debe ser ambiguo (dobles interpretaciones); si se ejecuta el mismo algoritmo el resultado siempre será el mismo, sin importar las entradas proporcionadas.
4. **General**. Debe tolerar cambios que se puedan presentar en la definición del problema.

4.4 Ejemplo 3: Calcular área de triángulo

PROBLEMA: Escribe un algoritmo para obtener el área de un triángulo, tomando en cuenta que:

- área: (base * altura)/2.

INICIO

1. Pedir la base y la altura (B, H).
2. Multiplicar la base y la altura y dividirlas entre 2 ($A = B * H / 2$).
3. Mostrar resultados (A).

FIN

4.5 Clasificación de algoritmos

Los algoritmos se pueden clasificar en cuatro tipos:

1. Algoritmo computacional: Es un algoritmo que puede ser ejecutado en una computadora. Ejemplo: Fórmula aplicada para un cálculo de la raíz cuadrada de un valor x.
2. Algoritmo no computacional: Es un algoritmo que no requiere de una computadora para ser ejecutado. Ejemplo: Instalación de un equipo de sonido.
3. Algoritmo cualitativo: Un algoritmo es cualitativo cuando en sus pasos o instrucciones no están involucrados cálculos numéricos. Ejemplos: Las instrucciones para desarrollar una actividad física, encontrar un tesoro.
4. Algoritmo cuantitativo: Un algoritmo es cuantitativo cuando en sus pasos o instrucciones involucran cálculos numéricos. Ejemplo: Solución de una ecuación de segundo grado.

5. Programa de computadora

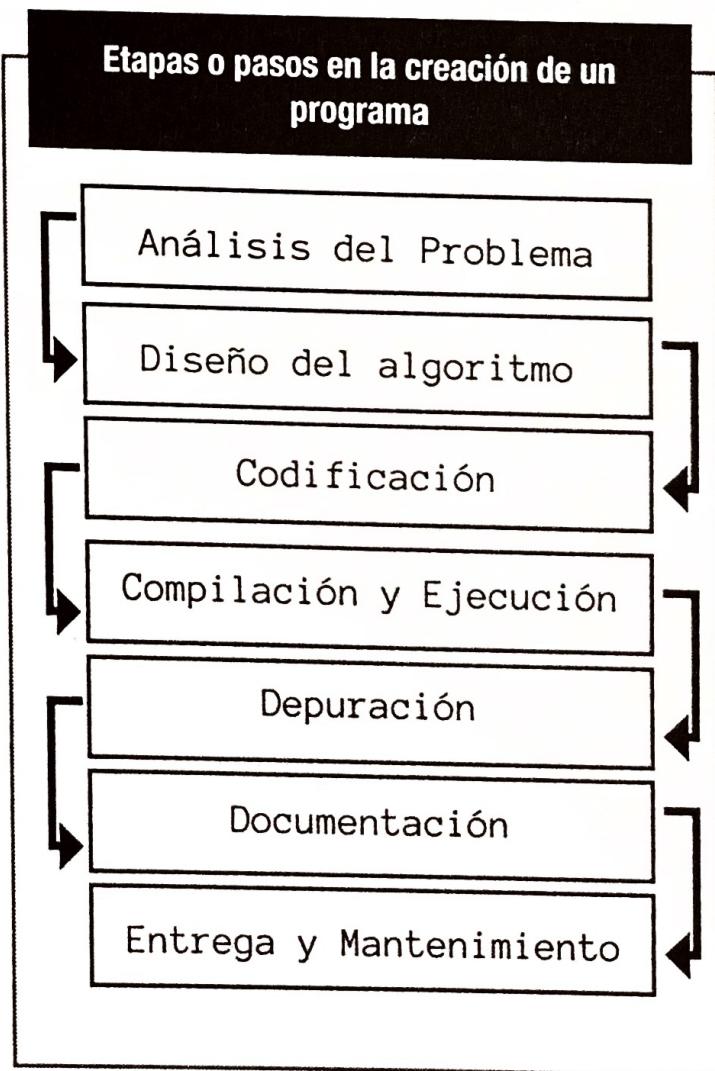
5.1 Definición de programa de computadora

Existen diferentes conceptos:

1. Es un algoritmo desarrollado en un determinado lenguaje de programación, para ser utilizado por la computadora; es decir, es una serie de pasos o instrucciones ordenadas y finitas que pueden ser procesadas por una computadora, a fin de permitirnos resolver un problema o tarea específica.

2. Secuencia de instrucciones mediante las cuales se ejecutan diferentes acciones de acuerdo con los datos que se deseé procesar en la computadora.
3. Expresión de un algoritmo en un lenguaje preciso que puede llegar a entender una computadora.

5.2 Etapas en la creación de un programa (Modelo Cascada)



5.2.1 Paso 1: Análisis del problema

Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada. Para poder definir bien un problema es conveniente responder a las preguntas:

- ¿Qué cantidad y tipo de datos de entrada se requieren?
- ¿Qué cantidad y tipo de salidas se desean?
- Los métodos y fórmulas que se necesitan para procesar los datos y producir esa salida.

5.2.2 Paso 2: Diseño del Algoritmo

En esta etapa se define cómo hace el programa la tarea solicitada, es decir, se define el algoritmo. La etapa de diseño se centra en desarrollar el algoritmo basándonos en las especificaciones de la etapa del análisis; podemos representar un algoritmo mediante el diagrama de flujo o el pseudocódigo.

5.2.3 Paso 3: Codificación

En la etapa de codificación se transcribe el algoritmo definido en la etapa de diseño en un código reconocido por la computadora; es decir, en un lenguaje de programación; a éste se le conoce como código fuente.

5.2.4 Paso 4: Compilación y Ejecución

El proceso de compilación consiste en pasar el programa de código fuente a un código ejecutable. La ejecución es "correr" el programa generado en el proceso de compilación.

5.2.5 Paso 5: Depuración

Se ejecuta el programa con datos de prueba para detectar y corregir errores en tiempo de ejecución. La prueba consiste en capturar datos hasta que el programa funcione correctamente. A la actividad de localizar errores se le llama depuración. Existen dos tipos de pruebas: de sintaxis y de lógica.

- Pruebas de sintaxis: Se ejecutan primero, son las más sencillas y las realiza el compilador del programa cada vez que se ejecuta el programa hasta que el código no presente errores, es decir, que la sintaxis que requiere el lenguaje sea la correcta, de lo contrario el propio compilador va mostrando los errores encontrados para que se modifiquen y se pueda ejecutar el código; estas pruebas pueden ser falta de paréntesis, o puntos y comas o palabras reservadas mal escritas.
- Pruebas de lógica: Son las más complicadas ya que éstas las realiza el programador, consisten en la captura de diferentes valores y revisar que el resultado sea el deseado, es decir, el programador tendría que modificar el código hasta que el programa funcione correctamente.

5.2.6 Paso 6: Documentación

- Interna: Comentarios dentro del programa.
- Externa: Manuales del programador, del usuario, de instalación, etc.

5.2.7 Paso 7: Entrega y Mantenimiento

6. Estrategias para la solución de problemas

6.1 Estrategias de solución directa

1. **Algoritmos de Solución Forzada:** Un algoritmo de este tipo resuelve el problema de la forma más simple, obvia o directa. Como resultado es posible que el algoritmo haga más trabajo que una solución más sofisticada. Por otra parte, las soluciones forzadas son más fáciles de implementar y por eso algunas veces resultan más eficientes.
2. **Algoritmos Codiciosos:** Se caracterizan porque las decisiones que toman se basan en que la búsqueda del menor costo en esa parte del problema, pero no toman en cuenta el resto de la solución y en ocasiones no generan soluciones óptimas.

6.2 Estrategias de Vuelta Atrás

Un algoritmo de vuelta atrás sistemáticamente considera todos los posibles resultados para cada decisión. En este sentido, los algoritmos vuelta atrás son como las soluciones forzadas. Sin embargo, los algoritmos vuelta atrás se distinguen por la forma en que exploran todas las posibles soluciones; en ocasiones estos algoritmos encuentran que una búsqueda exhaustiva es innecesaria y por lo tanto pueden tener una mejor ejecución.

6.3 Estrategias arriba-abajo

- Algoritmos divide y vencerás: Para resolver un problema, este se subdivide en uno o más subproblemas, cada uno de los cuales es similar al problema dado. Cada uno de los subproblemas se soluciona en forma independiente y al final las soluciones de todos los subproblemas se combinan para obtener la solución general del problema completo.

6.4 Estrategias abajo-arriba

- Programación Dinámica: Para resolver un problema se resuelven una serie de subproblemas. La serie de subproblemas es planeada cuidadosamente de tal forma que cada solución subsecuente se obtiene mediante la combinación de las soluciones de uno o más subproblemas que ya han sido resueltos. Todas las soluciones intermedias se mantienen en una tabla para evitar la duplicidad de esfuerzos.

6.5 Estrategias Probabilísticas

En los algoritmos probabilísticos existe un elemento de aleatoriedad en la forma en que el algoritmo soluciona el problema; se dice que estos métodos son el último recurso debido a que se usan cuando no hay otra técnica conocida que se pueda aplicar. Los métodos probabilísticos se usan cuando el espacio de soluciones es tan grande que una búsqueda exhaustiva no sería factible.

7. Análisis del problema

El objetivo del análisis del problema es ayudar al programador a llegar a una cierta comprensión de la naturaleza del mismo. Este análisis supone, en particular, la superación de una serie de pasos:

- Definir el problema con total precisión.
- Especificar los datos de partida necesarios para la resolución del mismo (especificaciones de entrada).
- Especificar la información que debe proporcionarse al resolver (especificaciones de salida).

Ejemplo: Elaborar el análisis para obtener el área y la longitud de una circunferencia.

1. Utilizar las fórmulas del área y la circunferencia en función del radio.
2. Las entradas de datos se reducen al dato correspondiente al radio del círculo.
3. Dada la naturaleza del mismo y el procesamiento al cual lo someteremos, su tipo de dato debe ser un número real.
4. Las salidas serán dos variables también reales: área y circunferencia.

La finalización de la fase de análisis del problema nos llevaría al siguiente resultado:

Elemento Variable Tipo de dato

Entrada RADIO Real

Salida 1 AREA Real

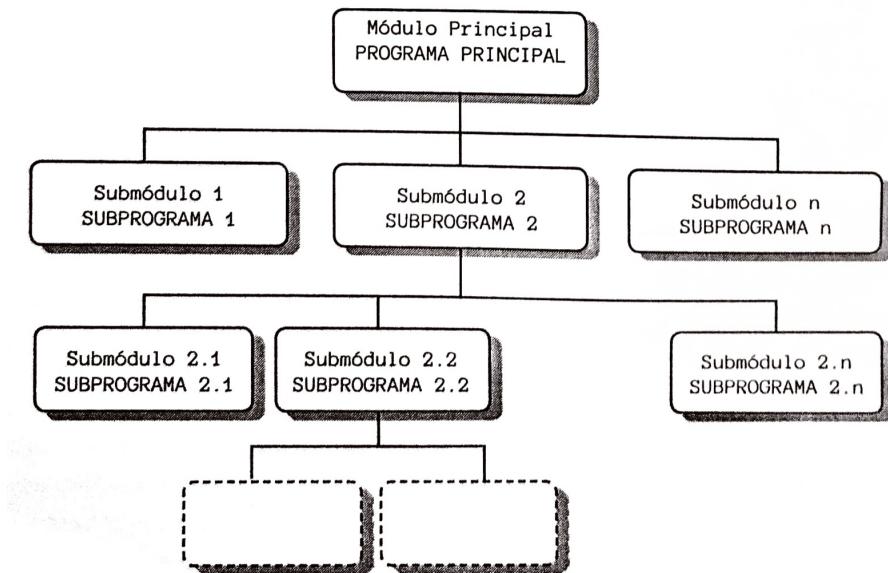
Salida 2 CIRCUNFERENCIA Real

8. Diseño del algoritmo

8.1 Diseño Descendente o Modular

Los problemas complejos se pueden resolver más eficazmente cuando se descomponen en subproblemas que sean más fáciles resolver el original. Este método se denomina divide y vencerse y consiste en convertir un problema complejo en otros más simples que, una vez resueltos, en su conjunto nos solucionen el original. Al procedimiento de descomposición de un problema en subproblemas más simples, llamados módulos para, a continuación, seguir dividiendo estos subproblemas en otros más simples, se le denomina diseño descendente. Las ventajas más importantes de este tipo de diseño son:

1. El problema se comprende más fácilmente al dividirse en módulos o partes más simples.
2. Las modificaciones en los módulos son más fáciles, pues estamos ante algoritmos más sencillos.
3. La comprobación del problema se puede realizar más fácilmente, al poder localizar los posibles fallos con mayor precisión.



8.2 Refinamiento por pasos

Durante el diseño, entenderemos por refinamiento por pasos, la metodología por la que en un primer esbozo del algoritmo nos limitamos a señalar o describir un reducido número de pasos, que deberán ser expresados con mayor detalle posteriormente. Tras esta primera descripción, éstos se especifican con mayor minuciosidad, de forma más extensa y con más pasos específicos.

En cada nivel de refinamiento hay que considerar dos fases: ¿Qué hace el módulo? para a continuación responder a ¿Cómo lo hace?.

Ejemplo: Diseñar un algoritmo que responda a la pregunta: ¿Qué debo hacer para ver la película XYZ?

```
INICIO {algoritmo para ver la película XYZ}
1. consultar la cartelera de cines
2. si proyectan "XYZ" entonces
3.   ir al cine correspondiente
4. si no proyectan "XYZ"
5.   declarar el fracaso del objetivo y terminar
6. acudir al cine correspondiente
7. si hay cola entonces ponerse en ella
8. mientras haya personas delante en la cola hacer avanzar en la cola
9. preguntar si quedan entradas
10. si hay entradas entonces
11.   comprar una entrada
12. si_no quedan entradas
13.   declarar el fracaso del objetivo, regresar a casa y terminar
14. encontrar el asiento correspondiente
15. mientras proyectan la película hacer
16.   ver la película
17. abandonar el cine
18. regresar a casa
FIN
```

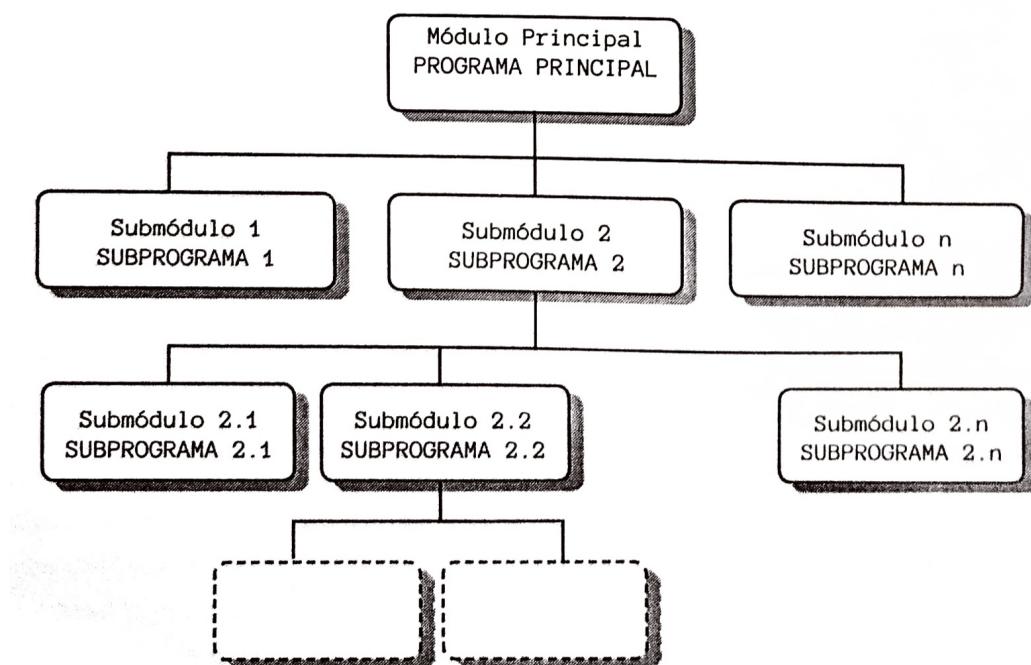
8.3 Ejemplo: Encontrar el asiento en el cine

INICIO {algoritmo para encontrar el asiento del espectador}

1. caminar hasta llegar a la primera fila de asientos
2. repetir
3. comparar número de fila con número impreso en billete
4. si no son iguales, entonces pasar a la siguiente fila
5. hasta_que se localice la fila correcta
6. mientras número de asiento no coincida con número de billete
7. hacer avanzar a través de la fila a la siguiente butaca
8. sentarse en la butaca

FIN

9. Programación del algoritmo



Esquema, del funcionamiento de un programa, desarrollado en módulos

1. Codificación del algoritmo en un programa.
2. Ejecución del programa.
3. Comprobación del programa.

10. Representación de algoritmos

10.1 Técnicas de representación

Para la representación de un algoritmo, antes de ser convertido a lenguaje de programación, se utilizan algunos métodos de representación escrita, gráfica o matemática. Los métodos más conocidos son:

- Diagramación libre (Diagramas de flujo)
- Diagramas Nassi-Shneiderman
- Pseudocódigo
- Lenguaje natural (español, inglés, etc.)
- Fórmulas matemáticas