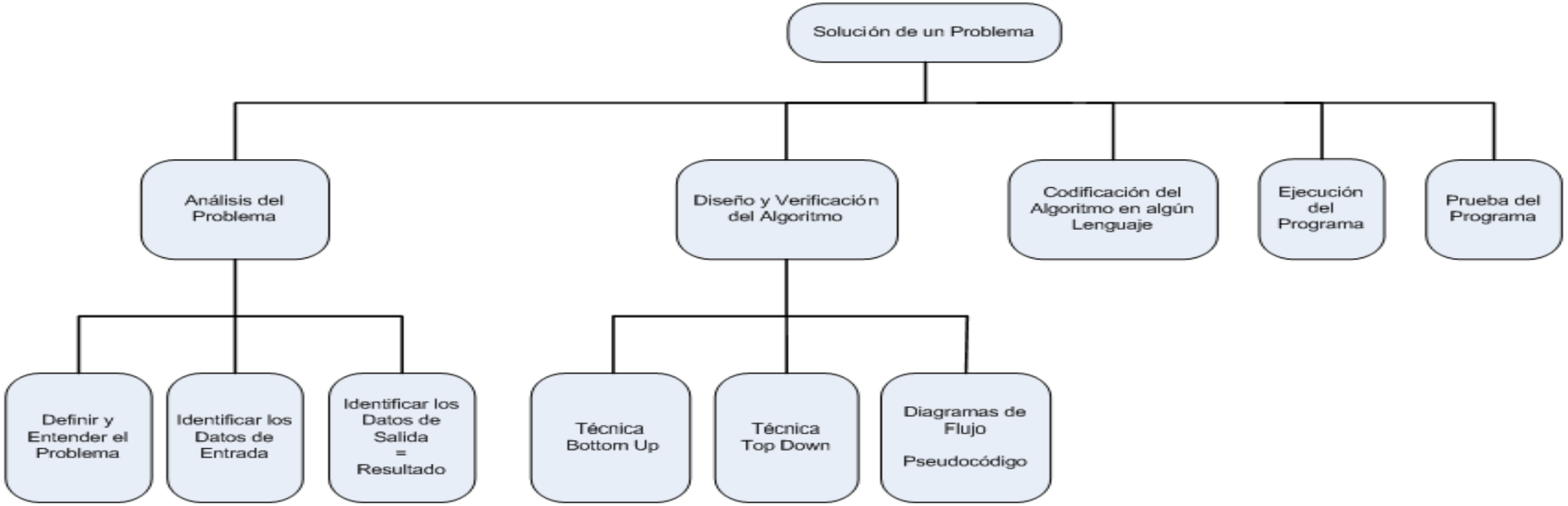
Metodología de resolución de problemas

ALGORITMOS Y ESTRUCTURAS DE DATOS I

Fases en la resolución de un problema



Fase 1: Análisis del problema

- ETAPA INICIAL PARA LA RESOLUCIÓN DEL PROBLEMA
- REQUIERE DE IMAGINACIÓN Y CREATIVIDAD
- IDENTIFICACIÓN DE DATOS DE ENTRADA Y TIPO DE INFORMACIÓN DE SALIDA
- TIPOS DE SALIDA Y RESULTADOS ESPERADOS:

  - Impresión por pantalla
  - Retorno del método
  - Modificación de los datos de entrada

Pre y Post condiciones

Luego de identificar las entradas y salidas Se debe especificar :

- **Las Pre-condiciones:**
  - Condiciones que el algoritmo asume que deben cumplir los datos de entrada incluyendo las relaciones entre ellos.
- **Las Post-condiciones:**
  - Condiciones que cumplen los datos de salida, las relaciones entre los datos de salida y las relaciones entre los datos de entrada y los datos de salida.

Pre y Post condiciones (cont.)

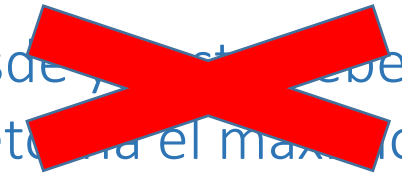
Ejemplo:

Desarrollar una función

int maximoDelVector (int v[ ], int desde, int hasta)

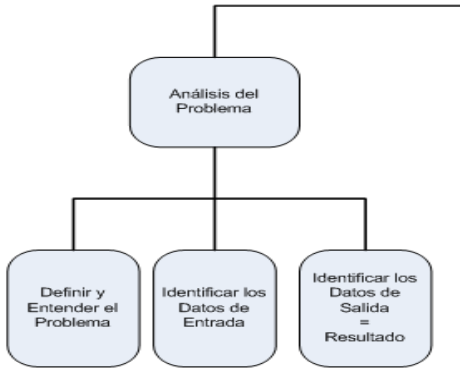
que devuelve el máximo valor encontrado en el vector v considerando únicamente las posiciones entre desde y hasta.

Sería correcto lo siguiente?



//Pre: desde y hasta deben ser válidos.
//Post: Retorna el máximo.

1. Entender el problema



Desarrollar una función int maximoDelVector (int v[ ], int desde, int hasta) que devuelve el máximo valor encontrado en el vector v considerando únicamente las posiciones entre desde y hasta.

20	3	9	4	23	54	-4
0	1	2	3	4	5	6

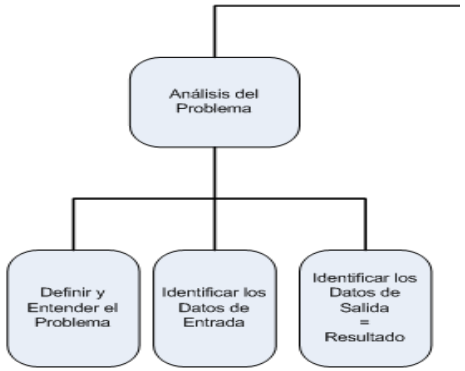
¿Que retorna con: desde 1 y hasta 4?

¿Que retorna con: desde 3 y hasta 8?

¿Que retorna con: desde -2 y hasta 2?

¿Que retorna con: desde 5 y hasta 1?

2. Identificar entradas y salidas



int maximoDelVector (int v[ ], int desde, int hasta)

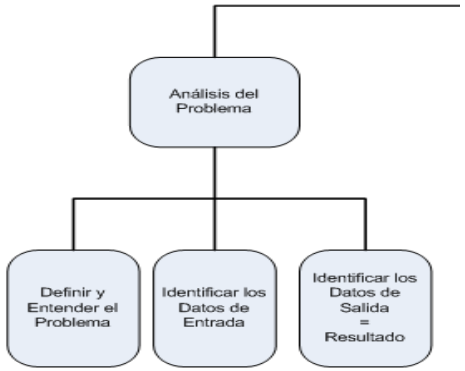
Entradas

- Identifico en la firma:
  - Vector
  - 2 enteros que serán utilizados como índices

Salidas

- 1 entero que representa el valor máximo del arrray entre desde y hasta inclusive

Identificar entradas y salidas



- Pre condiciones
  - “2 enteros que serán utilizados como índices
    - vector desordenado de enteros
    - desde >= 0
    - hasta >= 0
    - desde < largo del array
    - hasta < largo del array
    - desde <= hasta
- Post condiciones
  - 1 entero que representa el valor máximo del arrray entre desde y hasta inclusive
    - Retorna el máximo valor en el array entre desde y hasta inclusive

Pre y Post condiciones (cont.)

Ejemplo:

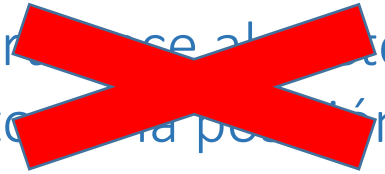
Desarrollar una función

int posiciónDeK (int v[ ], int k, int tamaño)

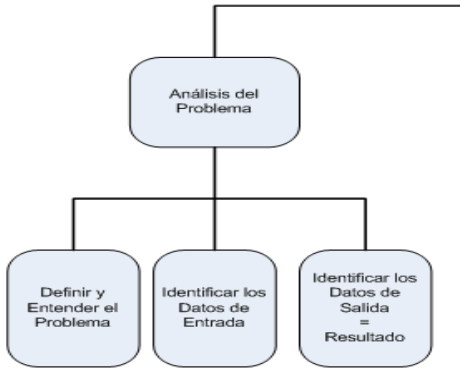
que retorne la posición del valor k en el vector. Si no se encuentra k retornar -1.

Sería correcto lo siguiente?

//Pre: k pertenece al vector.
//Post: Retorna la posición de k en el vector.



1. Entender el problema



Desarrollar una función int posiciónDeK (int v[ ], int k, int tamaño) que retorne la posición del valor k en el vector. Si no se encuentra k retornar -1.

20	3	9	4	23	54	-4
0	1	2	3	4	5	6

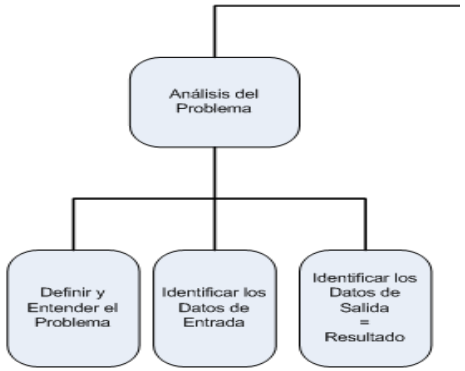
¿Que retorna con: k = 3 y tamaño = 1?

¿Que retorna con: k = 8 y tamaño = 16?

¿Que retorna con: k = -2 y tamaño = -3?

¿Que retorna con: k = 9 y tamaño = 7?

2. Identificar entradas y salidas



int posiciónDeK (int v[ ], int k, int tamaño)

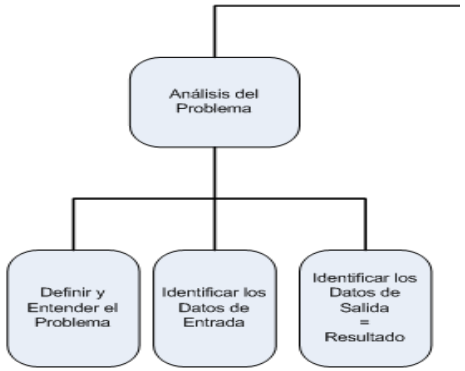
Entradas

- Identifico en la firma:
  - Vector
  - 1 enteros que será buscado en el array
  - 1 entero que representa hasta donde buscar en el array

Salidas

- 1 entero que representa el índice donde se encuentra k en el vector o -1 si no se encuentra.

Identificar entradas y salidas



- Pre condiciones
  - 1 entero que representa hasta donde buscar en el array
    - vector desordenado de enteros
    - tamaño = largo del array
    - K es un valor entero

- Post condiciones
  - 1 entero que representa el índice donde se encuentra k en el vector o -1 si no se encuentra.
    - Retorna el índice de la primer aparición de k o -1 si no se encuentra en el array

Pre y Post condiciones

Las pre y post condiciones deben cumplir:

- Ser claras
- No ambiguas
- Contemplar todos los escenarios
- No modificar el problema

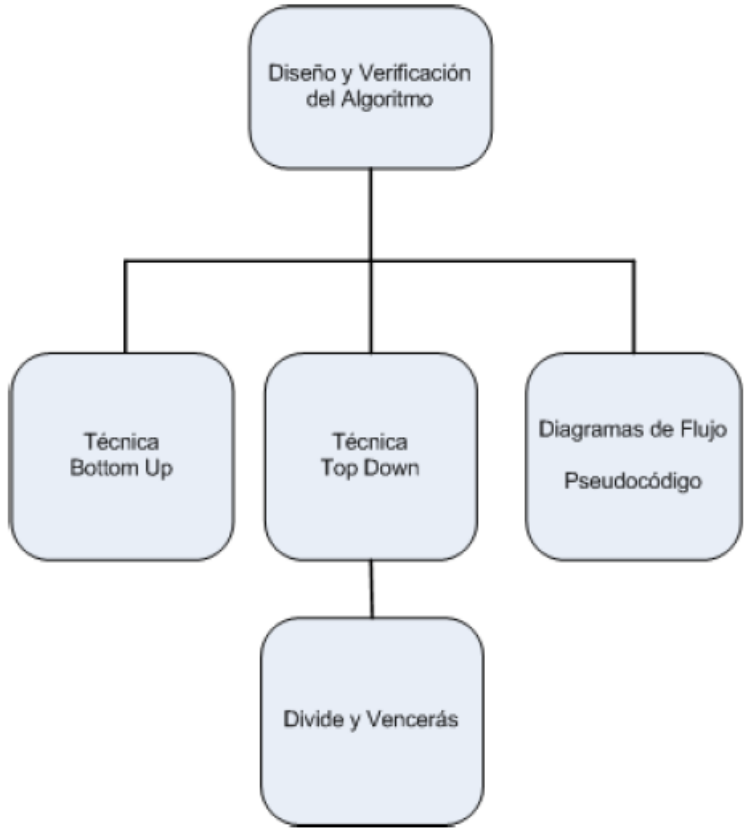
Las especificaremos como comentarios antes de la firma de la función.

Fase 2: Diseño y verificación del algoritmo

- **Técnica Bottom- Up o Diseño Ascendente:** Se comienza diseñando las partes más atómicas y luego se las une para ir formando componentes más grandes.

- **Técnica Top-Down o Diseño Descendente:** Se comienza desde un punto de vista macro del sistema y luego cada componente se va detallando sucesivamente.

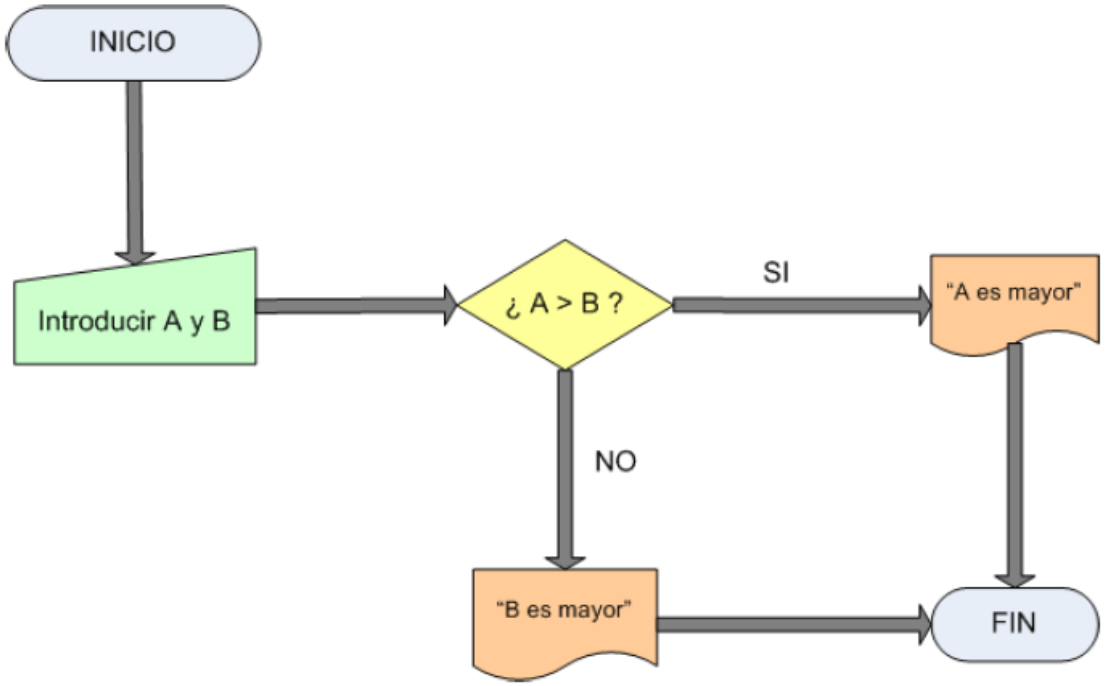
- **Diagramas de Flujo y/o Pseudocódigo.**



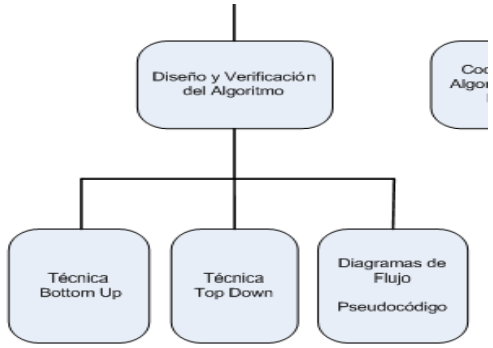
Fase 2: Diseño y verificación del algoritmo

- Diagramas de Flujo y/o Pseudocódigo.

Ej: Mayor entre dos números



Diseño del algorítmo

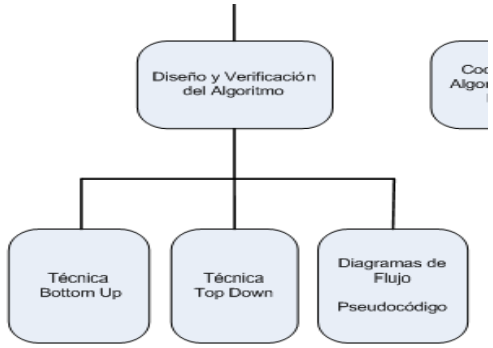


¿Qué algoritmos elegir para resolver un problema?

- Que sean fáciles de entender, codificar y depurar
- Que usen eficientemente los recursos del sistema: que usen poca memoria y que se ejecuten con la mayor rapidez posible

Ambos factores en general se contraponen

RAPIDEZ DE EJECUCIÓN

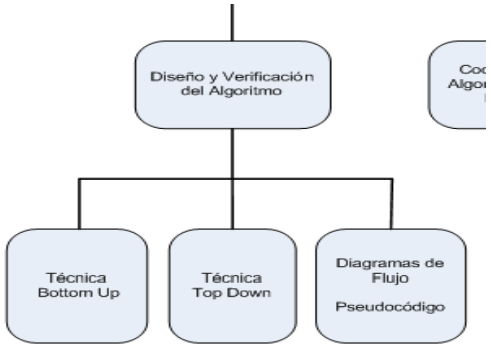


Factores que intervienen:

- Los datos de entrada al programa
- La calidad del código generado por el compilador
- La naturaleza y rapidez de las instrucciones de máquina
- La complejidad del algoritmo

Nos basaremos en desarrollar algoritmos eficientes según el volumen de datos de entrada.

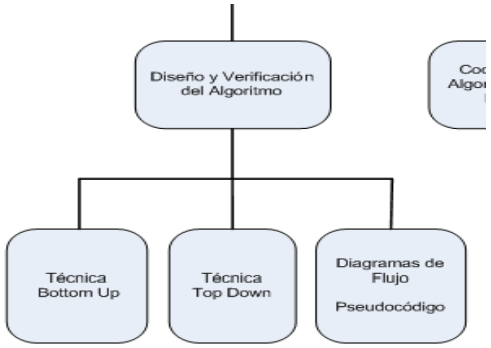
ABSTRACCIÓN



Los programas deben resolver problemas.

Construir programas para resolver problemas dados involucra representar conceptos abstractos en términos del lenguaje de programación.

PROGRAMACIÓN ESTRUCTURADA

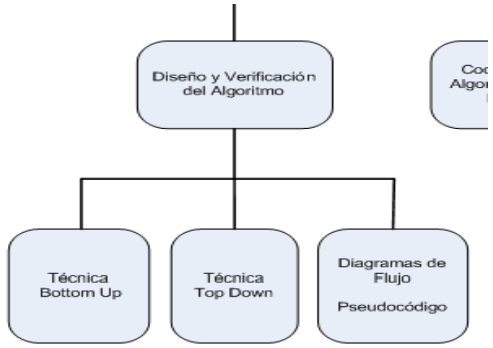


La Programación Estructurada es una forma de escribir programas de forma clara, comprensible y de fácilmente mantenimiento

Todo programa puede escribirse utilizando únicamente las estructuras:

- Secuencia
- Condicional (if, if-else)
- Iteración

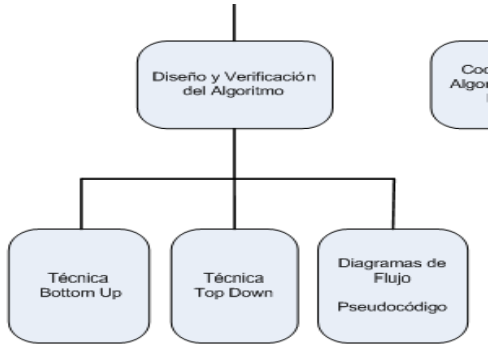
PROGRAMACIÓN ESTRUCTURADA



Ventajas:

- Los programas son más fáciles de entender, ya que pueden ser leídos de forma secuencial.
- La estructura del programa es clara puesto que las instrucciones están relacionadas entre sí.
- Reducción de esfuerzo en las pruebas (debugging).
- Reducción de los costos de mantenimiento de los programas.
- Los bloques de código son auto-explicativos, facilita la documentación

ABSTRACCIÓN DE DATOS



¿Qué es un tipo abstracto de datos?

Es un tipo de datos, es decir, puede ser especificado como tal con precisión, pero no está dado como un tipo de datos concreto del lenguaje.

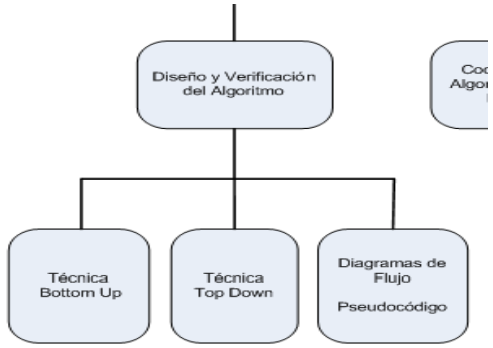
¿Cómo se introduce un Tipo Abstracto de Datos?

Básicamente: dándole un nombre y asociando a él un número de operaciones aplicables a los elementos del tipo.

Usualmente se habla de:

- ESPECIFICACION: Definición, que es lo que debe hacer
- IMPLEMENTACION: Es la codificación de la especificación para una estructura que satisface la especificación

ABSTRACCIÓN DE DATOS



Ventajas:

- Modularidad
- Adecuados para sistemas no triviales
- Separación entre especificación e implementación.
- Esto hace al sistema:
  - más legible
  - más fácil de mantener
  - más fácil de verificar y probar que es correcto. Robustez.
  - más fácil de rehusar
  - más extensible
  - lo independiza en cierta manera de las distintas implementaciones (complejidad tiempo-espacio).

Fase 5: Prueba del Programa

¿Cómo planificar el testing?

- ❑ Identificar las entradas y el posible rango de valores.
- ❑ Planificar casos de prueba de forma de buscar testear todos los posibles flujos del programa.
- ❑ Probar casos de borde en valores de variables.
- ❑ Probar casos de borde y criterios de parada de los bucles.
- ❑ Documentar pruebas.