Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TEMA 7:
VALIDACIÓN

TÉCNICAS DE PRUEBA DEL SOFTWARE

- Introducción
- Aspectos psicológicos de las pruebas
- Flujo de información de la prueba
- Pruebas indirectas. Inspecciones, recorridos, pruebas de escritorio.
- Pruebas directas. Técnicas de diseño de casos de prueba
- Pruebas por módulos o de integración
- Depuración

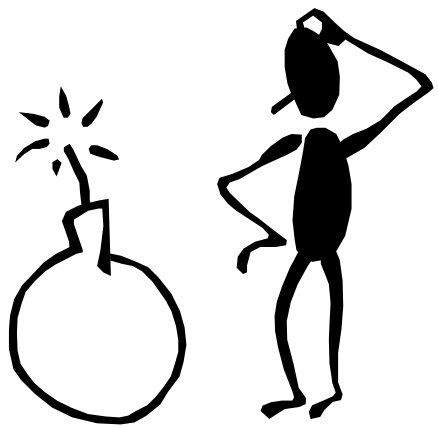
TÉCNICAS DE PRUEBAS DEL SOFTWARE. INTRODUCCIÓN

- Representan una revisión final de las especificaciones, del diseño y de la codificación.

- OBJETIVOS:
  - El objetivo de la prueba es descubrir algún error.

- Un caso de prueba es bueno cuando su ejecución conlleva una probabilidad elevada de encontrar un error y tiene éxito cuando lo detecta.

- “ LA PRUEBA ES EL PROCESO DE EJECUTAR UN PROGRAMA CON EL FIN DE ENCONTRAR ERRORES”



ASPECTOS PSICOLÓGICOS

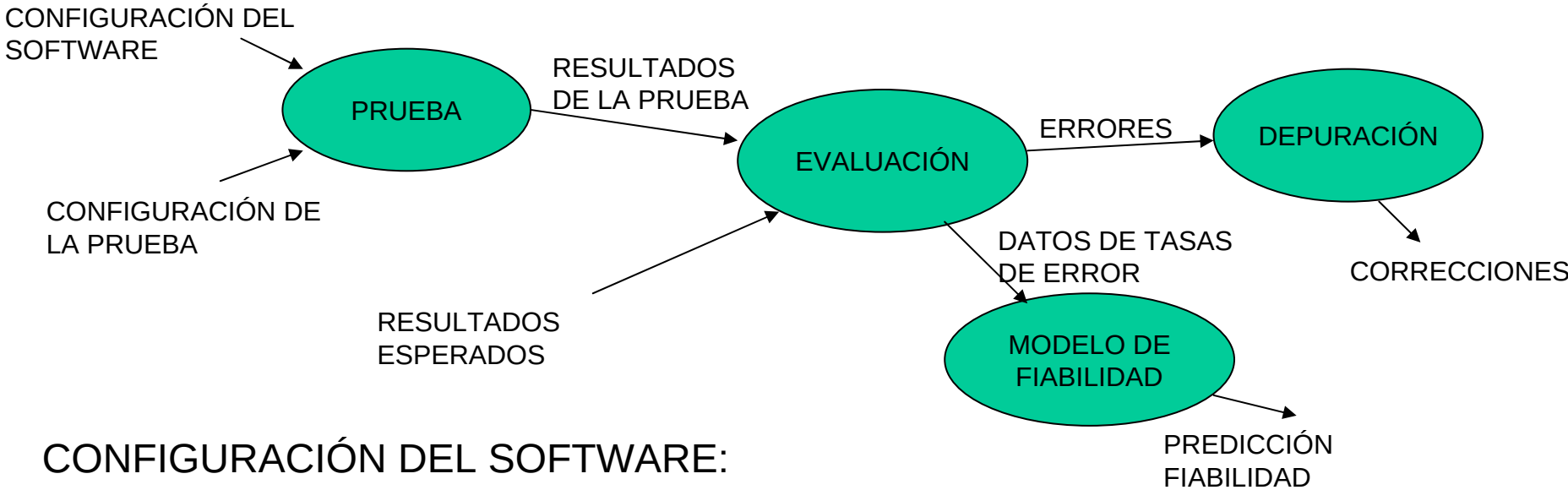
- Existe un sentimiento generalizado que considera de forma despectiva la misión de la prueba.

- Su naturaleza destructiva junto con otros factores externos tales como las limitaciones temporales, el coste, la oportunidad del producto, etc... Hacen que el ingeniero se sienta incomodo y no se involucre de forma adecuada en el proceso.

- Este hecho hace que se pueda cuestionar de una forma justificada la objetividad del creador frente a la búsqueda de fallos

- Por tanto la prueba deberá ser realizada por un equipo de personas, en su mayoría, ajenas al proyecto.

FLUJO DE INFORMACIÓN DEL SOFTWARE



CONFIGURACIÓN DEL SOFTWARE:

- Especificación de requisitos
- Especificación del diseño
- código fuente

CONFIGURACIÓN DE PRUEBA:

- Plan de procedimiento
- herramienta de prueba
- Casos de prueba
- Resultados que se espera obtener

PRUEBAS INDIRECTAS. INSPECCIONES, RECORRIDAS Y REVISIONES DEL PROGRAMA

- Estos métodos son conocidos como indirectos o procesos de prueba sin ordenador.
- Estas deben realizarse justo después de la codificación y antes de las pruebas basadas en el uso de la computadora.
- Estos métodos implican la lectura o inspección visual del programa.
- Son realizadas por un equipo de personas.
- Objetivo: Encontrar errores, no soluciones.
- Este tipo de métodos detectan “grupos de errores” lo que permite a posteriori una corrección masiva.

DISEÑO DE CASOS DE PRUEBA

- Para cualquier producto de ingeniería existen dos enfoques a la hora de probar un producto:
- **Pruebas de Caja Blanca**:
  - Se centra en el estudio minucioso de la operatividad de una parte del sistema considerando los detalles procedurales (la lógica del sistema).
- **Pruebas de Caja Negra**:
  - Analiza principalmente la compatibilidad entre sí, en cuanto a las interfaces, de cada uno de los componentes del software (no tiene en cuenta la lógica del sistema).

- Combinando ambas estrategias se obtiene una más completa validación del software

TÉCNICAS DE DISEÑO DE CASOS DE PRUEBA

- Propósito:
  - **Reducir** el número de casos de prueba manteniendo la efectividad de la prueba

- Enfoques:
  - Caja negra (que es lo que hace)
  - Caja blanca (como lo hace)

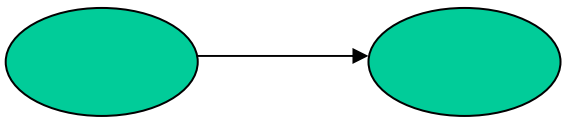
PRUEBAS DE CAJA BLANCA

- Usa la estructura de control del diseño procedimental para obtener los casos de prueba.
- Estos casos deben garantizar:
  - Que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
  - Que se ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
  - Que se ejecuten todos los bucles en sus límites operacionales.
  - Que se ejerciten las estructuras internas de datos para asegurar su validez.
- "**Los errores se esconden en los rincones y se aglomeran en los límites**" [Beizer].

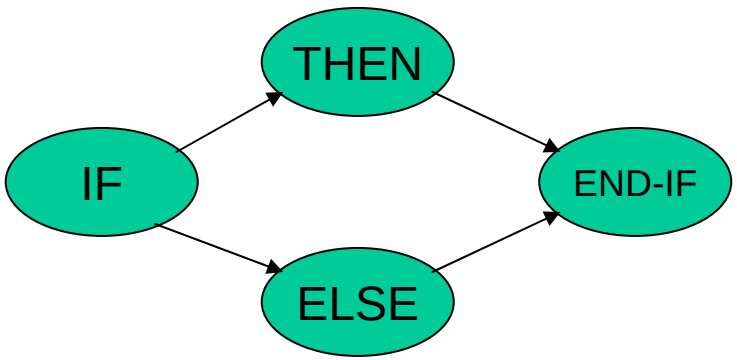
PRUEBA DEL CAMINO BÁSICO

- Es una técnica para obtener casos de prueba de caja blanca.
- Este método permite obtener una medida de la complejidad lógica de un diseño procedimental.
- Esta medida puede ser usada como guía a la hora de definir un conjunto básico de caminos de ejecución (diseño de casos de prueba).
- Para la obtención de la complejidad lógica o ciclomática emplearemos una representación del flujo de control en forma de grafo (Grafo del flujo).

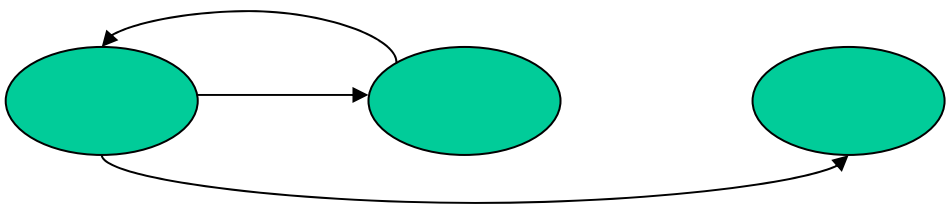
REPRESENTACIÓN DEL GRAFO DE FLUJO DE LAS ESTRUCTURAS DE CONTROL



SECUENCIA

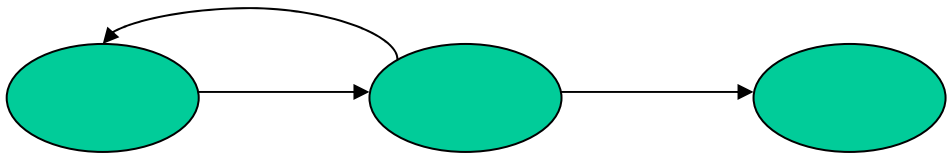


SI,..SINO..

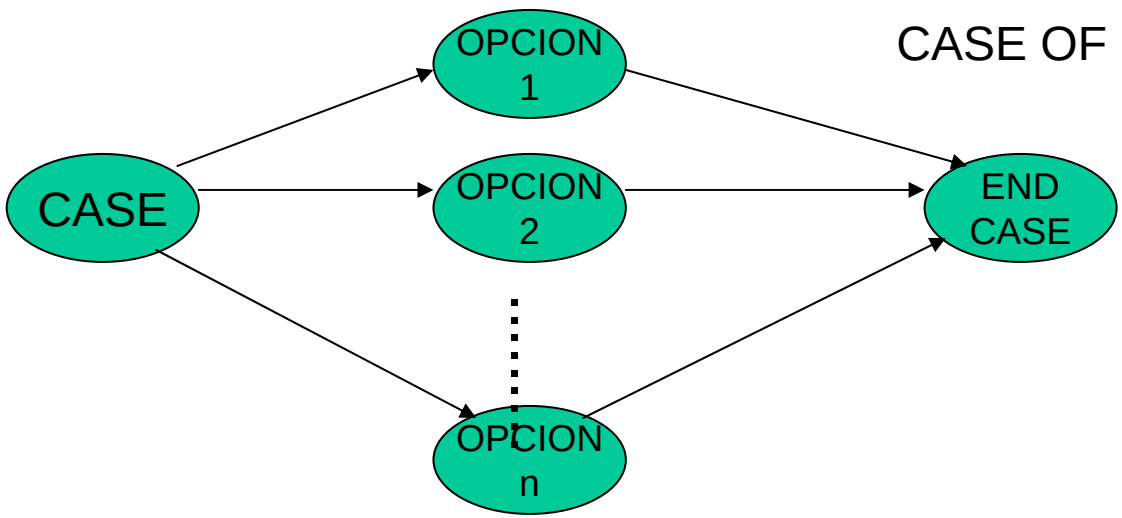


WHILE

REPRESENTACIÓN DEL GRAFO DE FLUJO DE LAS ESTRUCTURAS DE CONTROL

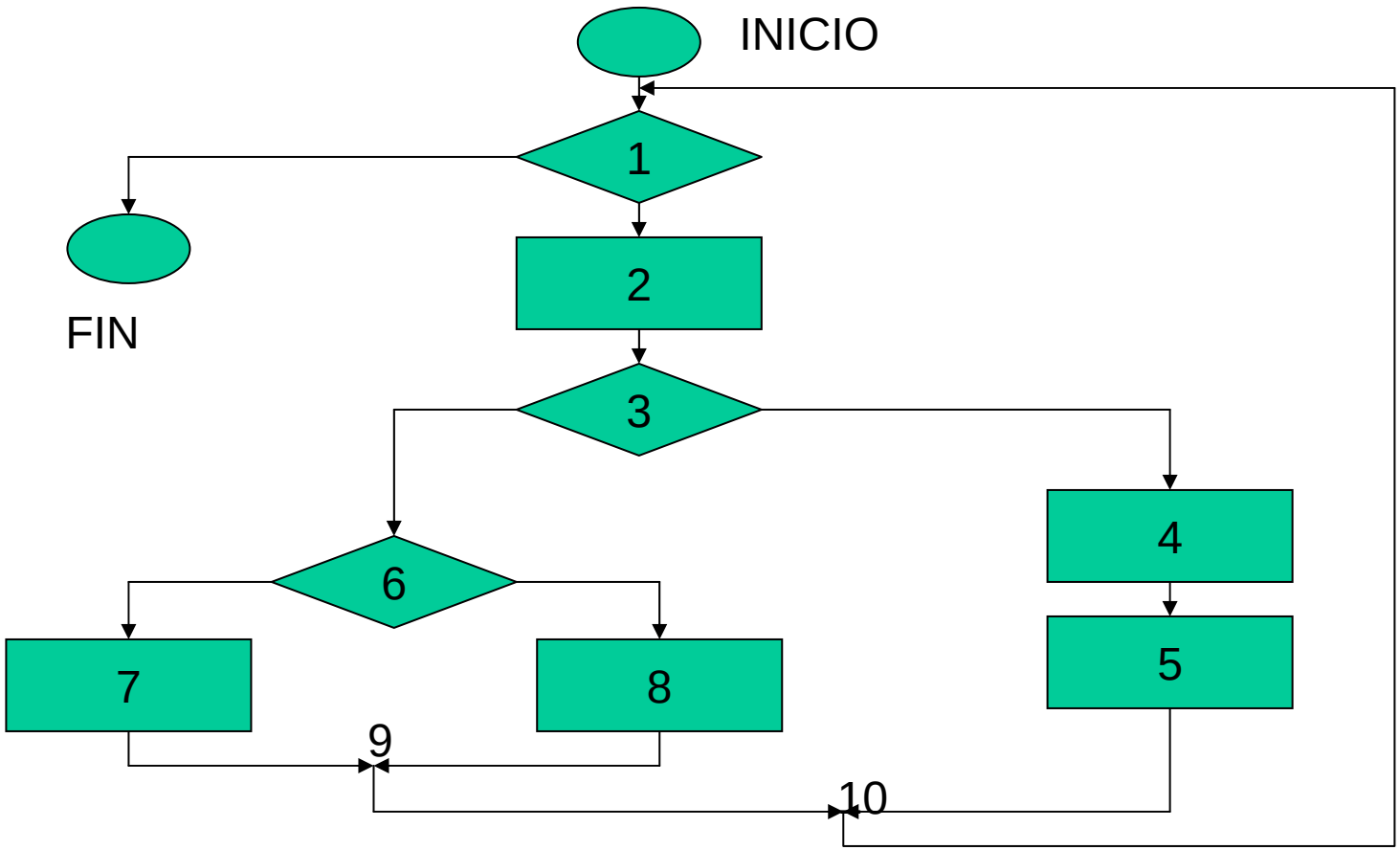


REPEAT

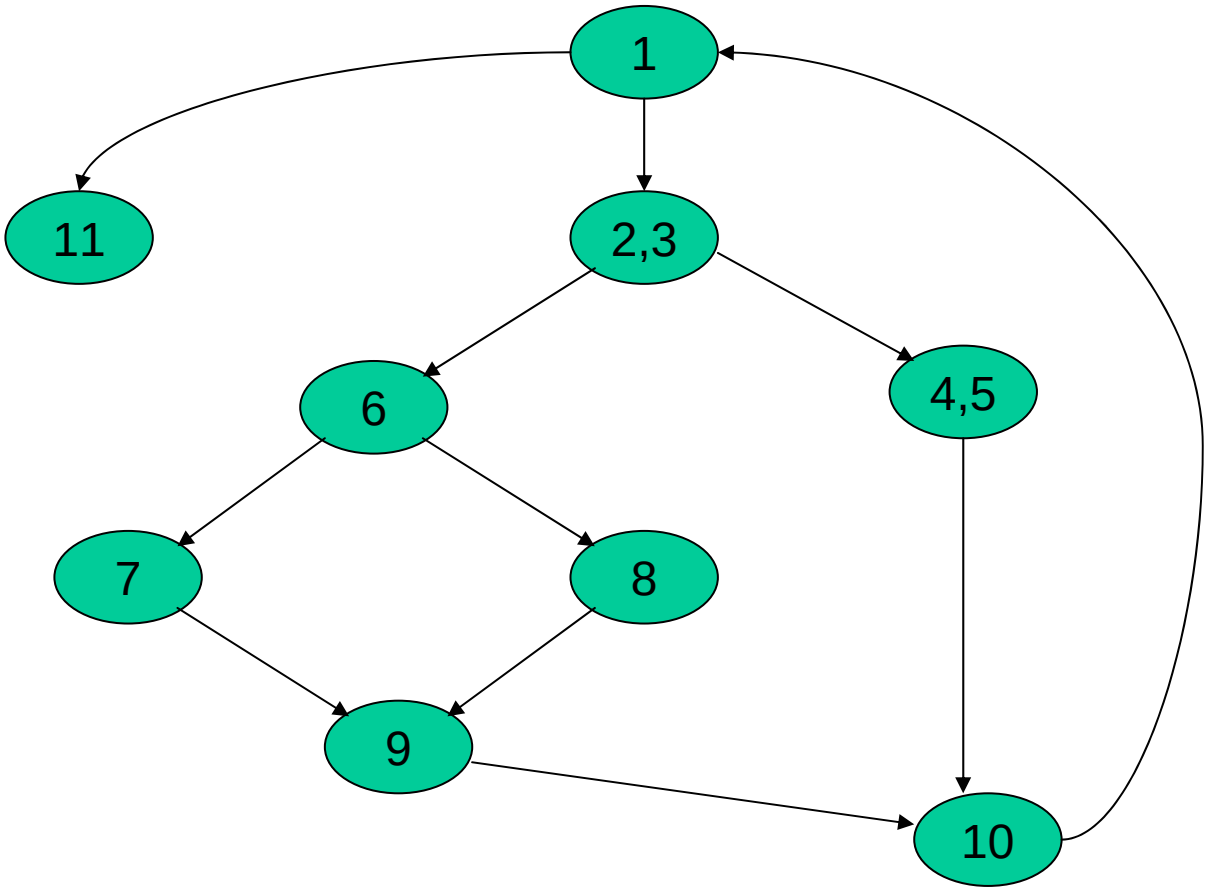


CASE OF

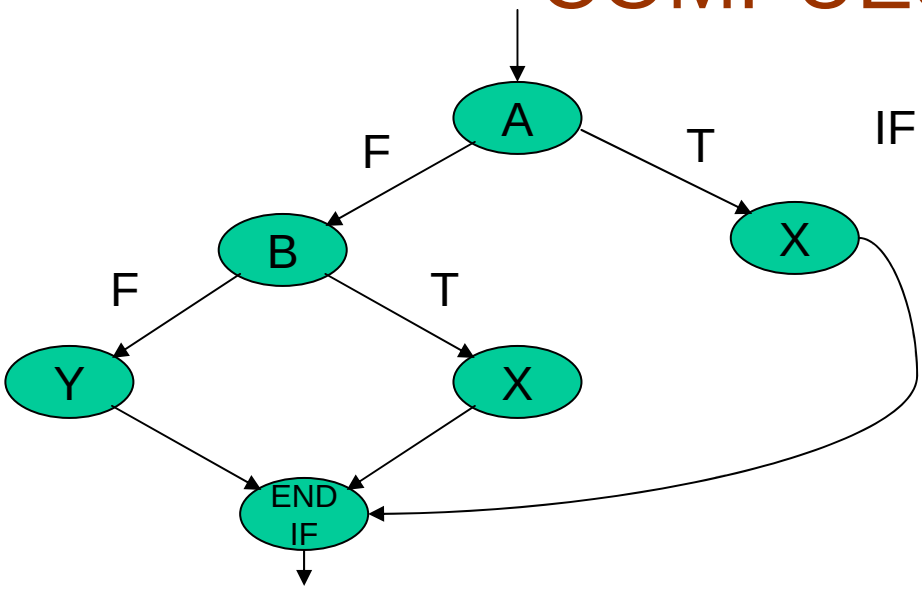
EJEMPLO. DIAGRAMA DE FLUJO



EJEMPLO. GRAFO DE FLUJO

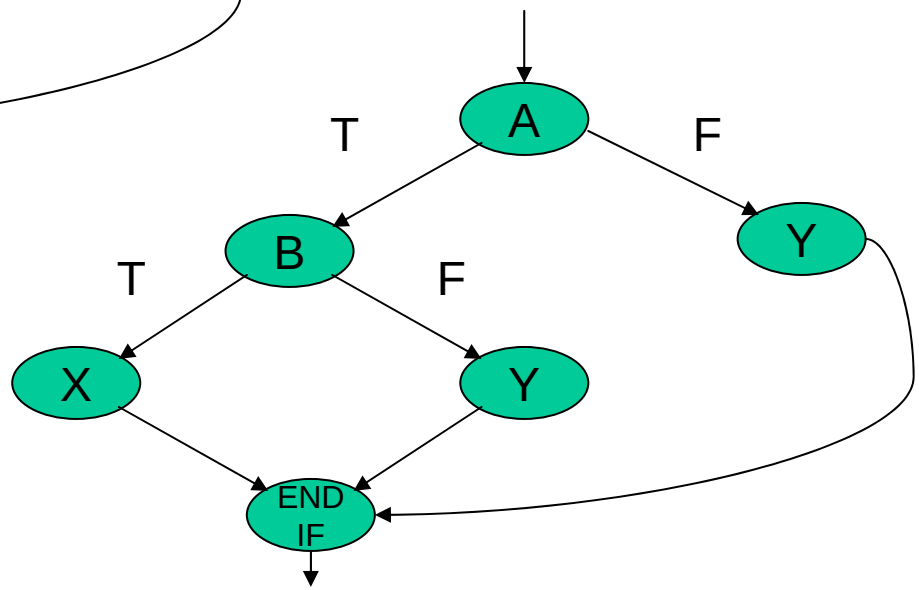


GRAFOS ASOCIADOS A CONDICIONES LÓGICAS COMPUESTAS



IF A OR B
THEN X
ELSE Y

IF A AND B
THEN X
ELSE Y



COMPLEJIDAD CICLOMÁTICA

- Es una **medida del software** que aporta una valoración cuantitativa de la complejidad lógica de un programa.
- Dentro del contexto del método de pruebas del camino básico define el **número de caminos independientes** de un programa.
- Por **camino independiente** se entiende aquel que introduce un nuevo conjunto de sentencias o una nueva condición. En términos del grafo, por una arista que no haya sido recorrida antes.
- Nos da una **cota o límite superior** para el número de casos de prueba.

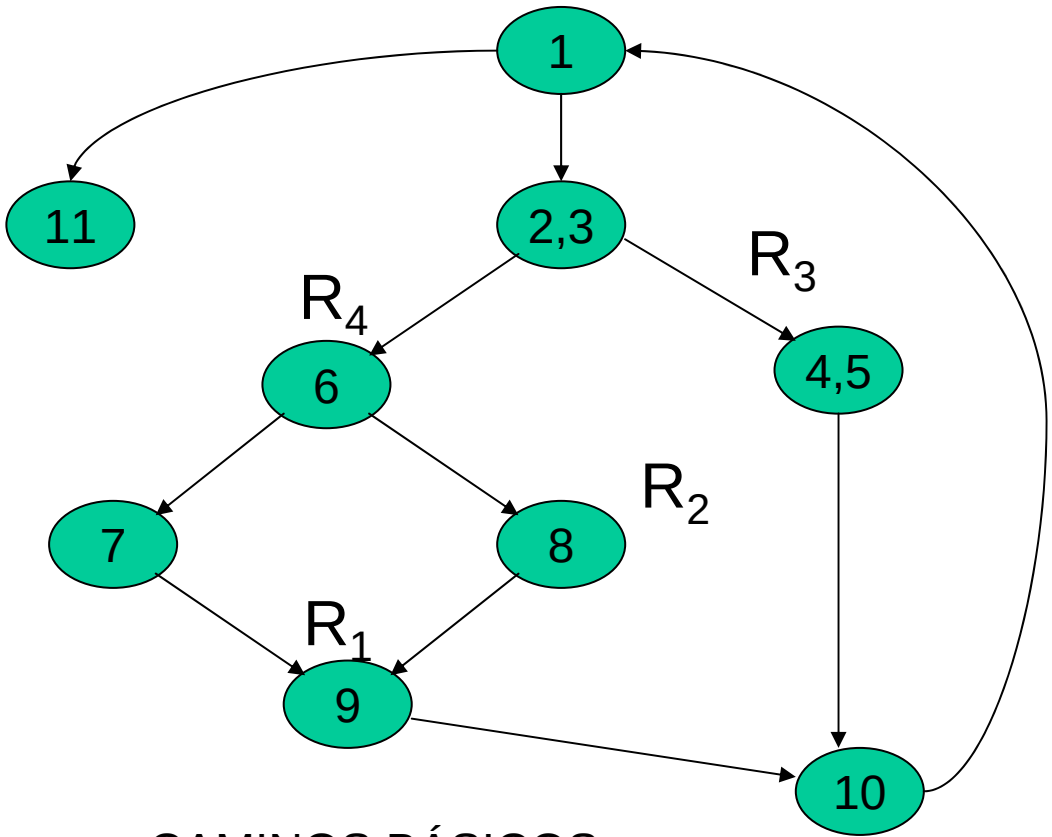
CÁLCULO DE LA COMPLEJIDAD CICLOMÁTICA

- Formas de cálculo:
  - El número de regiones del grafo es igual a la complejidad ciclomática.
  - V(G)=A-N+2
    - Donde A es el número de aristas y N es el número de nodos contenidos en el grafo.
  - V(G)=P+1
    - Donde P es el número de nodos predicados contenidos en el grafo.

DERIVACIÓN DE LOS CASOS DE PRUEBA

- Pasos a seguir:
  - Representación del grafo de flujo asociado al código fuente del programa.
  - Cálculo de la complejidad ciclomática.
  - Determinación de un conjunto de caminos básicos. Dicho conjunto básico, para un grafo dado, puede no ser único y existir distintas posibilidades.
  - Se preparan los casos que obligan a la ejecución de cada camino del conjunto básico.
- Los casos de prueba así derivados garantizan que:
  - al menos una vez se ejecute cada sentencia
  - cada condición se haya probado en sus dos vertientes (verdadera y falsa).

COMPLEJIDAD CICLOMÁTICA.CAMINOS BÁSICOS.



- V(G)=4 Regiones
- V(G)= 11A-9N+2=4
- V(G)=3NP+1=4

CAMINOS BÁSICOS:

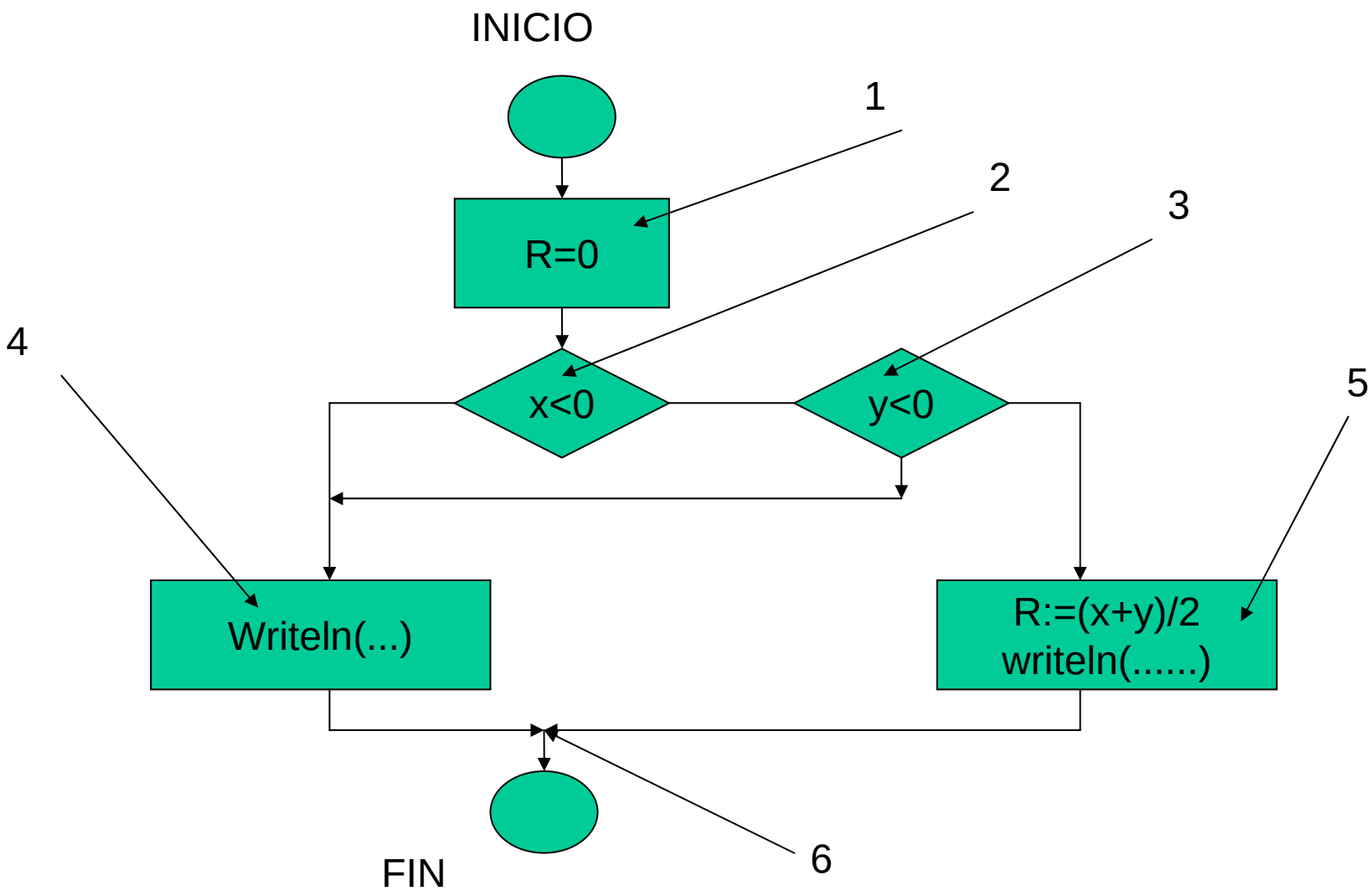
- Camino 1: 1-11
- Camino 2: 1-2-3-4-5-10-1-11
- Camino 3: 1-2-3-6-8-9-10-1-11
- Camino 4: 1-2-3-6-7-9-10-1-11

EJEMPLO

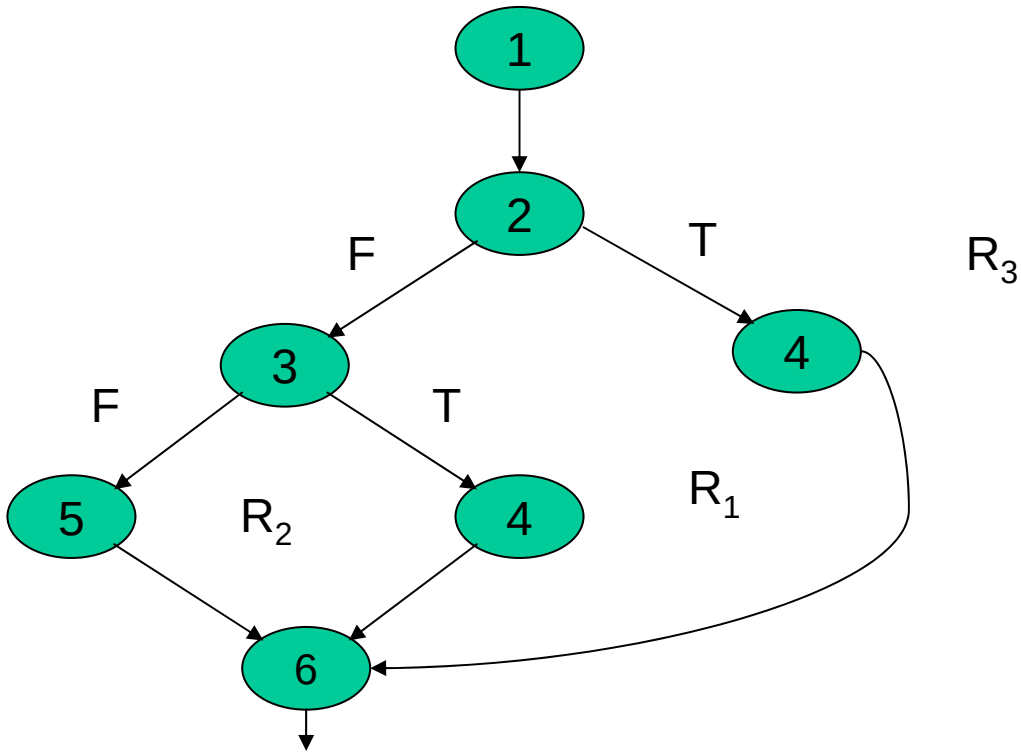
- Diseñar el conjunto de casos de prueba mediante el método de la complejidad ciclomática para el siguiente código:

```
r:=0;
if (x<0 or y<0) then
        writeln('x e y deben ser positivos')
else begin
        r:=(x+y)/2;
        writeln('la media es: ', r)
        end;
```

EJEMPLO



EJEMPLO. CONVERSIÓN AL GRAFO DE FLUJO



V(g)=3 regiones=3
V(g)=8A-7N+2=3
V(g)=2NP+1=3

EJEMPLO. CONJUNTO DE CAMINOS BÁSICOS

- Habrá tantos caminos básicos como grados de complejidad posee e código.
- En este caso tenemos tres caminos básicos:
  - C1: 1-2-4-6
  - C2: 1-2-3-4-5
  - C3: 1-2-3-5-6
- La derivación de los casos a partir de los caminos básicos es inmediato:
  - C1: $x<0$ e y indiferente
  - C2: $x\geq 0$ e $y<0$
  - C3: $x\geq 0$ e $y\geq 0$

PRUEBAS DE CAJA BLANCA. COBERTURA LÓGICA

- Cobertura de Sentencia.
- Cobertura de Decisiones.
- Cobertura de Condiciones.
- Cobertura de Condición Múltiple.

COBERTURA DE SENTENCIA

- Esta cobertura requiere que se ejecute por lo menos una vez cada sentencia del programa.
- Este criterio es necesario pero no suficiente
- Es un criterio débil
  - No se comprueban ambas vertientes de las condiciones.

COBERTURA DE DECISIÓN

- Este criterio establece que es necesario escribir un número suficiente de casos de prueba como para que cada decisión tenga por lo menos un resultado verdadero o falso.

- Este criterio es más fuerte que el de sentencia pero aún presenta debilidades.

  – En sentencias condicionales compuestas puede quedar enmascarada una de las condiciones.

COBERTURA DE CONDICIÓN

- En este criterio es necesario presentar un número suficiente de casos de prueba de modo que cada condición en una decisión tenga, al menos una vez, todos los resultados posibles.
- Este criterio es más fuerte que el anterior.
- Hay que ser cuidadoso con la elección de los casos de prueba por que aunque se garanticen la ejecución de las condiciones puede ocurrir que alguna cláusula de la decisión no sea ejecutada.

COBERTURA MÚLTIPLE

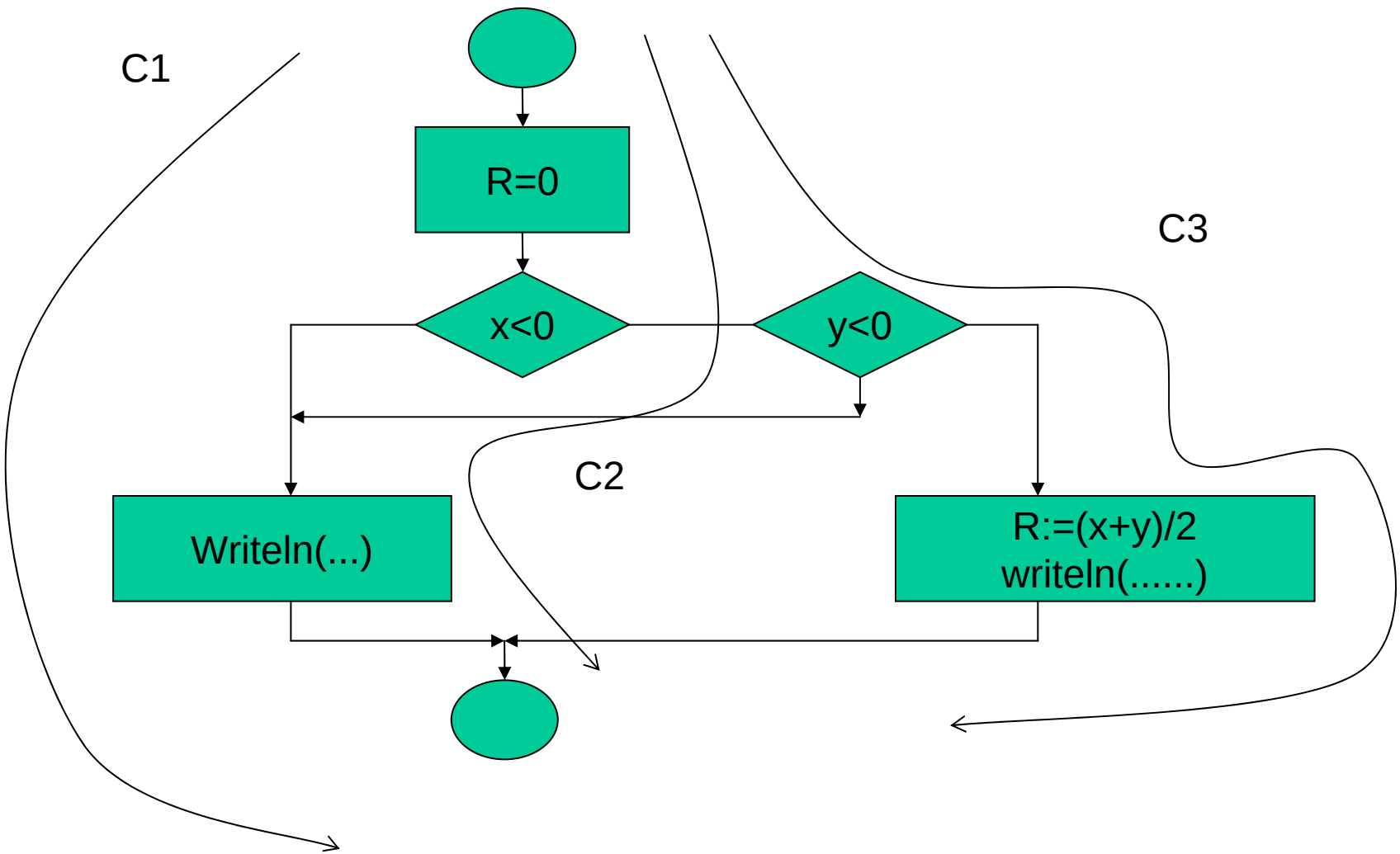
- La solución lógica a lo anterior es utilizar un criterio que requiera un número suficiente de casos de prueba tal que todas las **combinaciones posibles** de resultados de condición en cada decisión y todos los puntos de entrada se invoquen al menos una vez.

- Satisface los criterios de cobertura anteriormente citados.

EJEMPLO. POSIBLES COMBINACIONES

  - 1: $x \geq 0$ e $y \geq 0$
  - 2: $x \geq 0$ e $y < 0$
  - 3: $x < 0$ e $y \geq 0$
  - 4: $x < 0$ e $y < 0$
- Como se puede apreciar estas combinaciones son cubiertas por los siguientes caminos:
  - C1: $x < 0$ e y indiferente (3,4)
  - C2: $x \geq 0$ e $y < 0$ (2)
  - C3: $x \geq 0$ e $y \geq 0$ (1)

EJEMPLO. POSIBLES COMBINACIONES



DISEÑO DE CASOS DE PRUEBA MEDIANTE CAJA NEGRA

- Enfoque de caja negra:
  - También denominadas pruebas de comportamiento.
  - Consideran la **función específica** para la cuál fue creado el producto (lo que hace).
  - Las pruebas se llevan a cabo **sobre la interfaz** del sistema.
  - Reduce el número de casos de prueba mediante la **elección de condiciones de entrada y salida válidas y no válidas** que ejercitan toda la funcionalidad del sistema.

TIPOS DE ERRORES QUE DETECTA

- Funciones incorrectas o ausentes
- Errores de la interfaz
- Errores en estructuras de datos o accesos a bases de datos
- Errores de rendimiento
- Errores de inicialización y terminación

TÉCNICAS DE PRUEBA DE CAJA NEGRA

- Partición Equivalente
- Análisis de Valores Límites

EL MÉTODO DE PARTICIÓN EQUIVALENTE (PE)

- Se basa en la **división del campo de entrada** en un conjunto de clases de datos denominadas **clases de equivalencia.**

CONCEPTO DE CLASE EQUIVALENTE

- **Clase de equivalencia**: un conjunto de datos de entrada que definen estados válidos y no válidos del sistema.
  - Clase válida: genera un valor esperado
  - Clase no válida: genera un valor inesperado

- Se obtienen a partir de las **condiciones de entrada** descritas en las especificaciones.

CONDICIONES DE ENTRADA

- Las condiciones de entrada vienen representadas por sentencias en la especificación.

Un valor específico	"..Introducir **cinco** valores.."
Un rango de valores	"...Valores **entre 0 y 10**..."
Un conjunto de valores relacionados	"....**Palabras reservadas en un lenguaje**...."
Un condición lógica	Condición "...**debe ser**..."

EL PROCEDIMIENTO

- Identificación de las clases de equivalencia
- Obtención de los casos de prueba:
    - Se parte de la premisa de que cualquier elemento de una clase dada es representativo del resto del conjunto.

IDENTIFICACIÓN DE LAS CLASES DE EQUIVALENCIA

- Por cada condición de entrada se identifican clases de equivalencia válidas y no válidas.
- Este proceso es heurístico.
- Sin embargo existen un conjunto de criterios que ayudan a su identificación.

CRITERIOS DE IDENTIFICACIÓN DE LAS CLASES DE EQUIVALENCIA

Condiciones de Entrada	Número de clases de equivalencia válida	Número de clases de equivalencia no válida
1. Rango de valores	**1** clase que contemple los valores del rango	**2** clases fuera del rango, una por encima y otra por debajo de éste
2. Valor específico	**1** clase que contemple dicho valor	**2** clases que representen un valor por encima y otro por debajo
3. Elementos de un conjunto tratados de forma diferente por el programa	**1** clase de equivalencia por cada elemento	**1** clase que represente un elemento fuera del conjunto
4. Condición lógica	**1** clase que cumpla la condición	**1** clase que no cumpla la condición

TABLA DE CONSIGNACIÓN DE LAS CLASES DE EQUIVALENCIA

- Las clases así identificadas se consignan en esta tabla enumerándose de cara a la derivación de los casos de prueba

Condiciones de Entrada	Clases de equivalencia válida	Clases de equivalencia no válida

DERIVACIÓN DE LOS CASOS DE PRUEBA

- Clases de equivalencia :
  - Cada caso debe contemplar el máximo número de clases de equivalencia válidas.
  - Generar suficientes casos para cubrir todas las clases.

- Generar un caso de prueba por cada clase de equivalencia no válida que haya sido identificada

Ejemplo 1:

- Construcción de una batería de pruebas para detectar posibles errores en la construcción de los identificadores de un hipotético lenguaje de programación. Las reglas que determinan sus construcción sintáctica son:
  - No debe tener mas de 15 ni menos de 5 caracteres
  - El juego de caracteres utilizables es:
    - Letras (Mayúsculas y minúsculas)
    - Dígitos (0,9)
    - Guión (-)
  - Se distinguen las mayúsculas de las minúsculas
  - El guión no puede estar ni al principio ni al final, pero puede haber varios consecutivos.
  - Debe contener al menos un carácter alfabético
  - No puede ser una de las palabras reservadas del lenguaje

- Consignación de las condiciones de entrada

Condiciones de Entrada	Clases de equivalencia válida	Clases de equivalencia no válida
-Entre 5 y 15 caracteres		
-El identificador debe estar formado por letras, dígitos y guión		
-Se diferencia entre letras mayúsculas y minúsculas		
-El guión no puede estar al principio, ni al final -Puede haber varios seguidos en el medio		
-Debe contener al menos un carácter alfabético		
-No se pueden usar palabras reservadas		

- Consignación de las clases de equivalencia

Condiciones de Entrada	Clases de equivalencia válida	Clases de equivalencia no válida
-Entre 5 y 15 caracteres	**1**. 5≤nº caracteres Ident. ≤15	**2**. nº caracteres Id<5
		3. 15< nº caracteres
-El identificador debe estar formado por letras, dígitos y guión	**4.** Todos los caracteres del Ident. ∈{letras, dígitos, guión}	**5.** Alguno de los caracteres del Ident. ∉{letras, dígitos, guión}
-Se diferencia entre letras mayúsculas y minúsculas	**6.** Palabra declarada ∈{Identificadores válidos}	**7.** Utilizar la misma palabra con alguna letra conmutada para hacer referencia al mismo identificador
-El guión no puede estar al principio, ni al final -Puede haber varios seguidos en el medio	**8.** Identificador sin guiones en los extremos y con varios consecutivos en el medio	**9**. Identificador con guión al principio
		10. Identificador con guión al final
-Debe contener al menos un carácter alfabético	**11.** Al menos un carácter del Ident. ∈{letras}	**12.** Ningún carácter del Ident. ∈{letras}
-No se pueden usar palabras reservadas	**13.** El Identificador ∉{palabras reservadas}	**14, 15, 16** ....un caso por cada palabra reservada

- Derivación de los casos de prueba

Identificador	Clases de equivalencia cubiertas	Resultado
Num-1---d3 (10)	1,4,6,8,11,13 (todas las válidas)	El sistema acepta el identificador
Nd3	2	Mensaje de error
Num-1-letra3---d3 (17)	3	Mensaje de error
Nu%m-1---d3 (11)	5	Mensaje de error
NuM-1---d3 (10)	7	Mensaje de error
-um-1---d3 (10)	9	Mensaje de error
Num-1---d- (10)	10	Mensaje de error
456-1---23 (10)	12	Mensaje de error
Real	14	Mensaje de error
..(el resto de palabras reservadas)..	15,16.....	Mensaje de error

EL MÉTODO DEL ANÁLISIS DE LOS VALORES LÍMITES (AVL)

- Se basa en la **evidencia experimental** de que **los errores** suelen aparecer con **mayor probabilidad** en los **extremos de los campos de entrada**.

- Un análisis de las condiciones límites de las clases de equivalencia aumenta la eficiencia de la prueba.
  - **Condiciones límites** : valores justo por encima y por debajo de los márgenes de la clase de equivalencia.

DERIVACIÓN DE LOS CASOS DE PRUEBA

- Generar tantos casos de prueba como sean necesarios para ejercitar las condiciones límites de las clases de equivalencia.
- Proceso heurístico
- Como en el caso anterior se pueden seguir unos criterios que faciliten su obtención

Condiciones de la especificación	Obtención de los casos de prueba
1. Rango de valores como condición de entrada	**1** caso que ejercite el valor máximo del rango
	1 caso que ejercite el valor mínimo del rango
	1 caso que ejercite el valor justo por encima del máximo del rango
	1 caso que ejercite el valor justo por debajo del mínimo del rango
2. Valor numérico específico como condición de entrada	**1** caso que ejercite el valor numérico específico
	1 caso que ejercite el valor justo por encima del valor numérico específico
	1 caso que ejercite el valor justo por debajo de valor numérico específico
3. Rango de valores como condición de salida	Generar casos de prueba según el criterio 1 que ejerciten dichas condiciones de salida
4. Valor numérico específico como condición de salida	Generar casos de prueba según el criterio 2 que ejerciten dichas condiciones de salida
5. Estructura de datos como condición de salida o de entrada	**1** caso que ejercite el primer elemento de la estructura
	1 caso que ejercite el último elemento de la estructura

Ejemplo 2 (PE vs AVL):

- Programa que establece a partir de tres valores de entrada de que tipo de triángulo se trata.
- Condición:

  A+B>C **and** A+C>B **and** B+C>A

  - Según el método de partición equivalente deberíamos considerar una clase equivalente válida y otra no válida

    {A=4, B=5, C=3}, {A=1, B=2, C= 5}

  - No detectaría un error en A+B≥C

  - AVL incluiría el caso {A=1, B=2, C=3}

- AVL aplicado al ejemplo 1

Condición	Descripción de los casos de prueba
Entre 5 y 15 caracteres	**1** caso con nº de caracteres identificados=15
	1 caso con nº de caracteres identificados=5
	1 caso con nº de caracteres identificados=16
	1 caso con nº de caracteres identificados=4

Condición	Casos de prueba
Entre 5 y 15 caracteres	Num-1-let-3---d3 (15)
	Numd3 (5)
	Num-1-letr-3---d3 (16)
	Nud3 (4)

CONJETURA DE ERRORES

- Consiste en la **elaboración previa** de una lista de errores no contemplados anteriormente que sirva para generar nuevos casos de prueba.

- Ejemplo: subrutina que ejecuta la búsqueda binaria
  - Un caso que compruebe
    - La presencia de un solo dato en la lista.
    - Que las posiciones de la lista sea una potencia de dos
    - Que las posiciones de la lista sea uno más de una potencia de dos
    - Que las posiciones de la lista sea uno menos de una potencia de dos

ESTRATEGIAS PARA EL DISEÑO DE CASOS DE PRUEBAS

- Cada técnica:
  - Evalúa una fuente diferente de errores.
  - El diseño debe involucrar una combinación de estas técnicas.

- Procedimiento:
  - En todos los casos utilizar análisis de valores límites
  - Complementar con casos de prueba derivados del método de partición equivalente
  - Añadir nuevos casos mediante la conjetura de errores

SÍNTESIS FINAL

- El proceso de prueba consiste en **ejecutar el programa con el fin de localizar errores**.
- La prueba es una **actividad incompleta**.
- **Propósito de las técnicas**: reducir el número de casos de prueba sin mermar la efectividad de la prueba.
- Existen dos enfoques a la hora de abordar el diseño de los casos de prueba:
  - **Caja Negra**: evalúa la funcionalidad del sistema (lo que hace)
  - **Caja Blanca**: evalúa la lógica interna (como lo hace)

SÍSTESIS FINAL

- Técnicas de diseño:
  - **Partición equivalente**: división del campo de entrada en clases de equivalencia válidas y no válidas, a través de las condiciones que aparecen en la especificación.
  - **Análisis de valores límites**: Evaluar las condiciones límites de las clases de equivalencia.
- La prueba de programas bajo caja negra involucra una combinación de todas estas técnicas.

EJECUCIÓN DE LAS PRUEBAS: PRUEBAS POR MÓDULOS O DE INTEGRACIÓN

- Consiste en la comparación de la función del módulo con respecto de alguna especificación funcional o interfaz que lo defina.

- Motivación:
  - Es una forma de poder manejar las posibilidades combinatorias de las pruebas.
  - La prueba por módulos facilita la tarea de localización y corrección de errores.
  - Introduce un cierto paralelismo en el proceso de prueba ya que se tiene la oportunidad de probarlos simultáneamente.

PROCESO DE PRUEBA POR MÓDULOS

- Se diseñan los casos de prueba:
  - Se analiza la lógica del programa (caja blanca)
  - Se complementa con casos obtenidos aplicando métodos de caja negra.

- Se determina el orden en que deben probarse los módulos, siguiendo unas determinadas recomendaciones prácticas.

PRUEBAS INCREMENTALES
y
PRUEBAS NO INCREMENTALES

- ¿Debe probarse de forma independiente cada módulo de un programa, para luego combinarlos y obtener el programa final?.

- ¿Se debe probar cada módulo integrandolo sucesivamente con el resto de los módulos hasta conformar el programa final?.

- La primera consideración se denomina integración no incremental frente a la segunda que se conoce como incremental.

PRUEBAS INCREMENTALES, PRUEBAS NO INCREMENTALES. PROCEDIMIENTO

- Prueba no incremental:
  - Primero se efectúa la prueba de módulos. Finalmente los módulos se combinan o integran para formar el programa completo.
  - La prueba de cada módulo requiere un módulo impulsor y uno o más auxiliares.

- Prueba incremental:
  - El siguiente módulo en ser probado se combina con los módulos que ya han sido probados.
  - La prueba puedes ser ascendente o descendente.

DEPURACIÓN

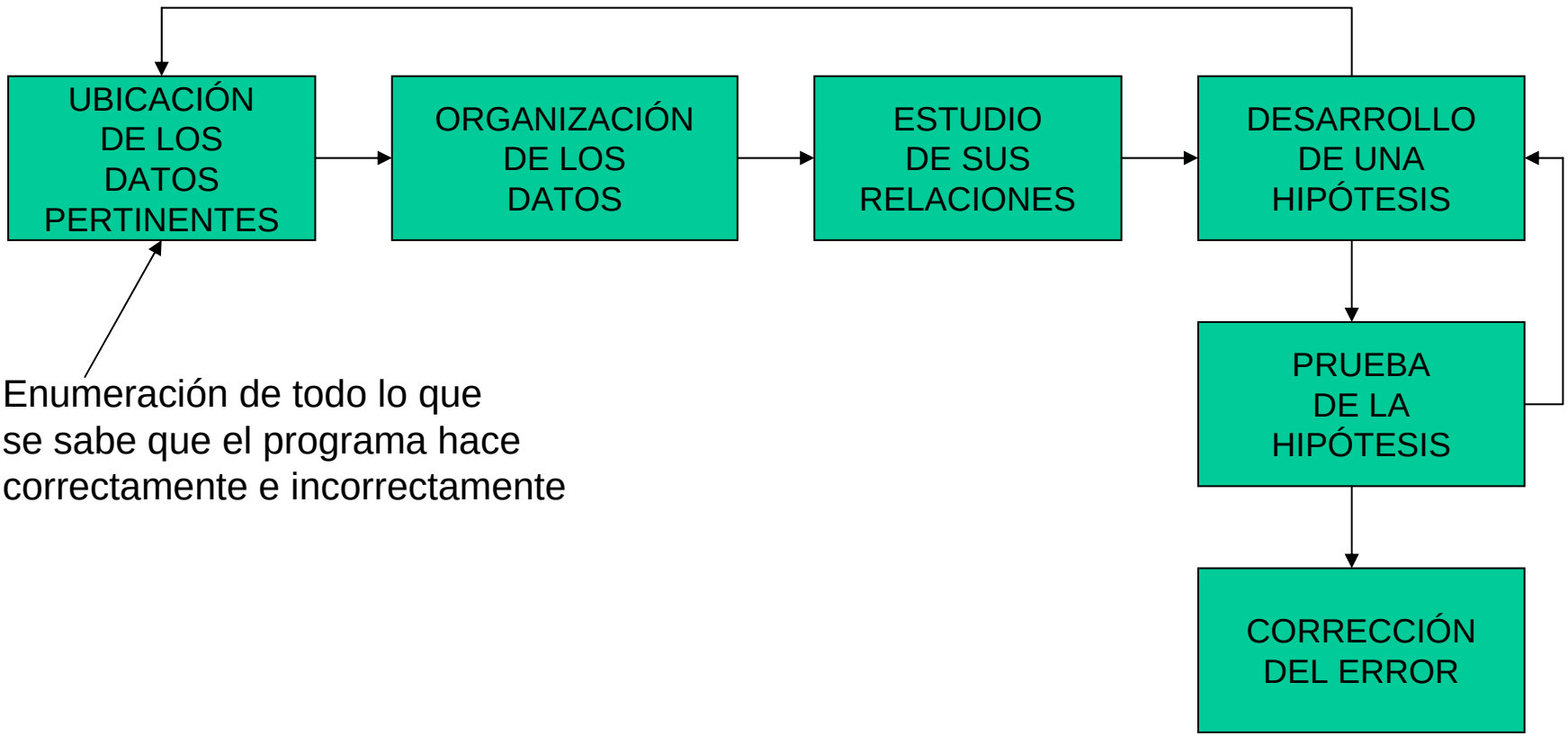
- La actividad que se desarrolla después de ejecutar una caso de prueba exitoso.
- Comprende los siguientes pasos:
  - Determinar la naturaleza exacta y la ubicación del presunto error dentro del programa.
  - Corregir el error encontrado.
- Métodos de depuración:
  - Por medio de la fuerza bruta
  - Por inducción
  - Por deducción
  - Por rastreo hacia atrás
  - Por medio de pruebas

DEPURACIÓN POR MEDIO DE LA "FUERZA BRUTA"

- Métodos que utilizan la fuerza bruta:
  - La estrategia de espacir sentencias dentro del programa de cara a producir mensajes o mostrar el valor de ciertas variables. (es un método de prueba y error).
  - Uso de herramientas que analizan la dinámica del programa empleando las características del depurador del lenguaje de programación empleado.
- Estos métodos son de prueba y error.
- Generan una cantidad excesiva de datos irrelevantes.
- No alientan a pensar en el problema que se trata resolver.

DEPURACIÓN POR INDUCCIÓN

INDUCCIÓN: Proceso que pasa de lo particular a lo general.



DEPURACIÓN POR DEDUCCIÓN

DEDUCCIÓN: Se parte de ciertas premisas o leyes generales para llegar a una conclusión, empleando para ello procesos de eliminación y clasificación.