



6. PRUEBAS DE CÓDIGO

La prueba del código consiste en la ejecución del programa (o parte de él) con el objetivo de encontrar errores. Se parte para su ejecución de un conjunto de entradas y una serie de condiciones de ejecución; se observan y registran los resultados y se comparan con los requisitos esperados. Se observará si el comportamiento del programa es el previsto o no y porqué.

Para las pruebas de código se van a mostrar diferentes técnicas que dependerán del tipo de enfoque utilizado: de caja blanca, se centran en la estructura del programa, o de caja negra, más centrado en las funciones, entradas y salidas del programa.

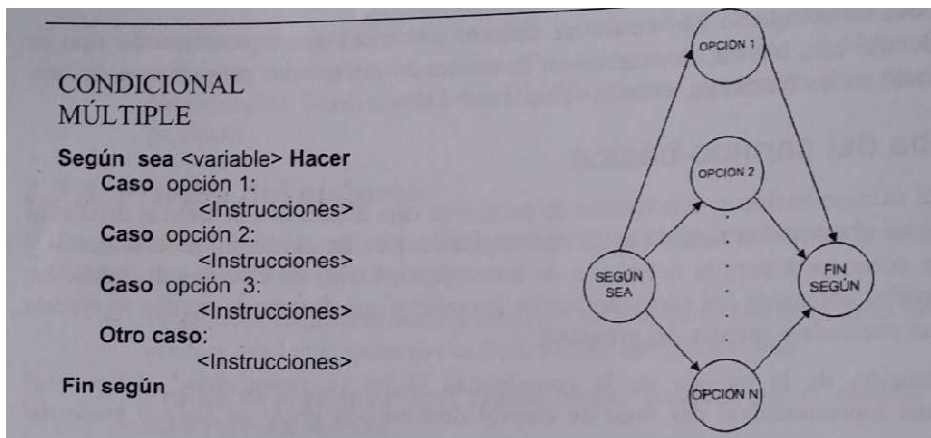
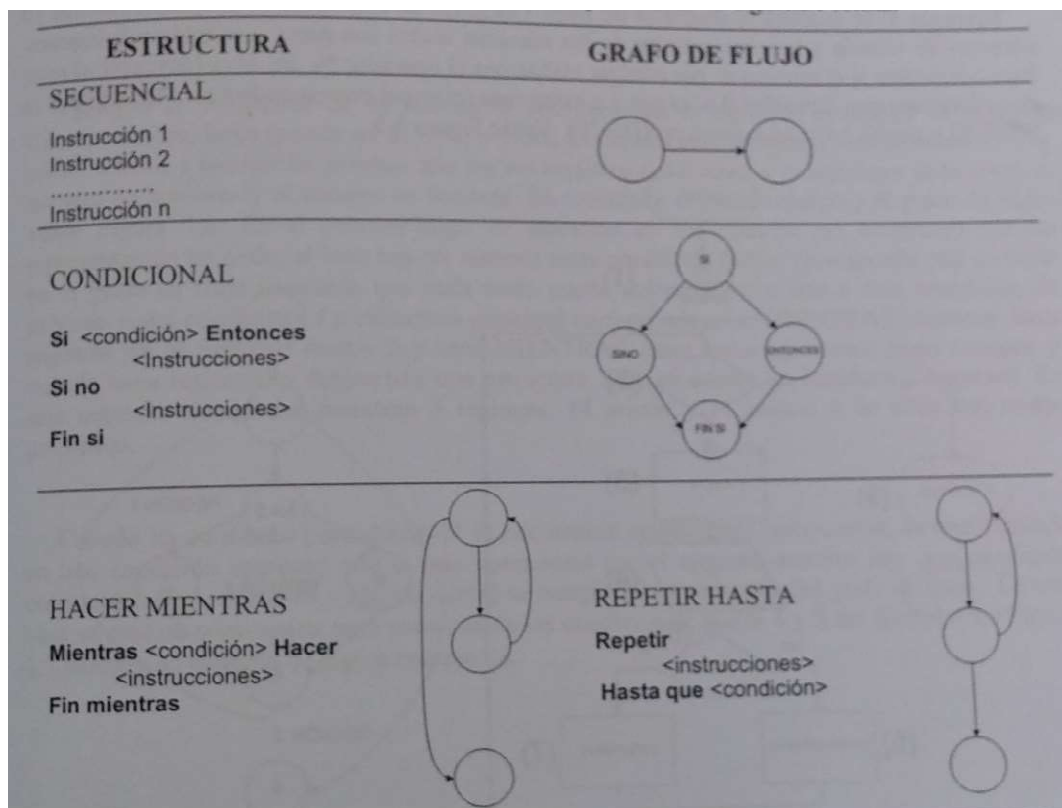
6.1 Prueba del camino básico

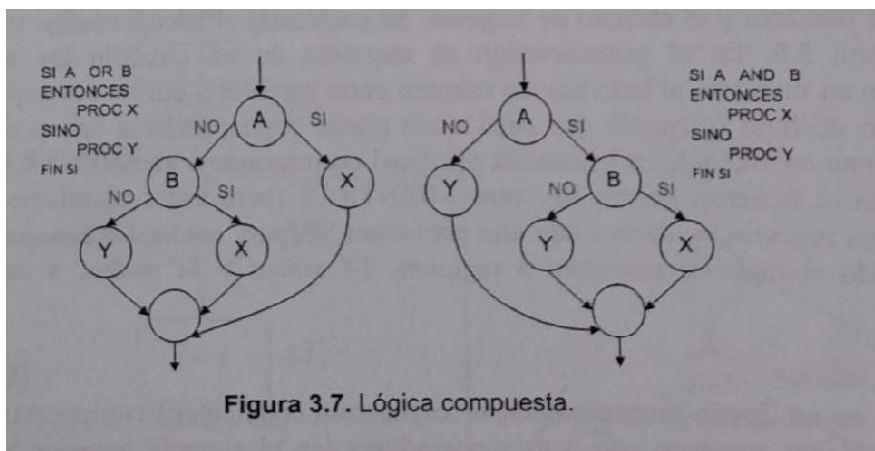
La prueba del camino básico es una técnica de caja blanca que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Para la obtención de la medida de la complejidad lógica (o *complejidad ciclomática*) emplearemos una representación del flujo de control denominada *grafo de flujo* o *grafo del programa*.

NOTACIÓN DE GRADO DE FLUJO

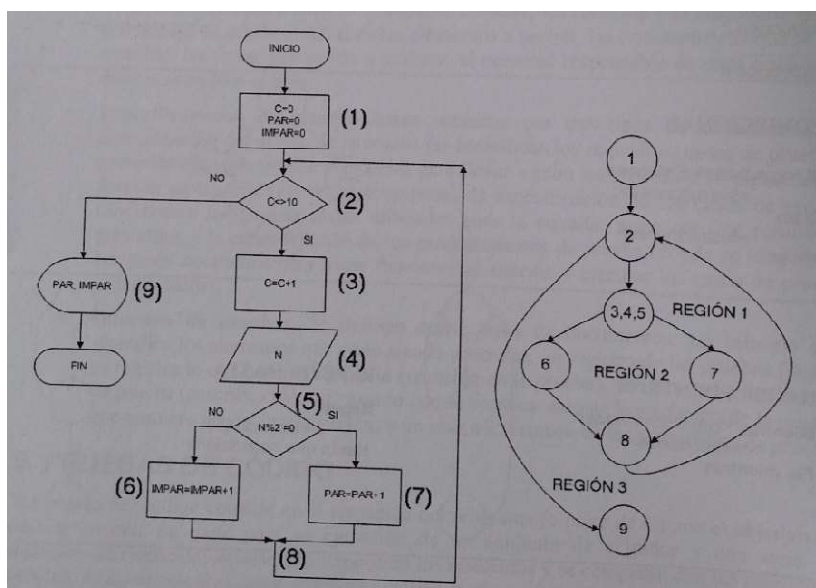
El grafo de flujo de las estructuras de control se representa de la siguiente forma:





Donde cada círculo representa una o más sentencias, sin bifurcaciones, en pseudocódigo o código fuente.

Ejemplo 1: construir un diagrama de flujo y grafo de flujo para un programa que lee 10 números de teclado y muestra cuántos de los números leídos son pares y cuántos son impares. Para comprobar si son pares o impares se utiliza el operador % (si devuelve 0 el número es par, distinto de 0 es impar).



En el diagrama de flujo se numeran cada uno de los símbolos, y los finales de las estructuras de control (el nodo 9 es el final del MIENTRAS) aunque no tengan ningún símbolo (el nodo 9 es el final de la estructura condicional).

Cada círculo del grafo de flujo se llama **nodo**. Representa una o más sentencias procedimentales. Un solo nodo se puede corresponder con una secuencia de símbolos del proceso y un rombo de decisión. Un ejemplo es el nodo numerado como 3, 4, 5.

Las flechas del grafo de flujo se denominan **aristas** (o enlaces) y representan el flujo de control, como en el diagrama de flujo. Una arista termina siempre en un nodo, aunque el nodo no tenga ninguna sentencia procedimental, en este caso el nodo 8.

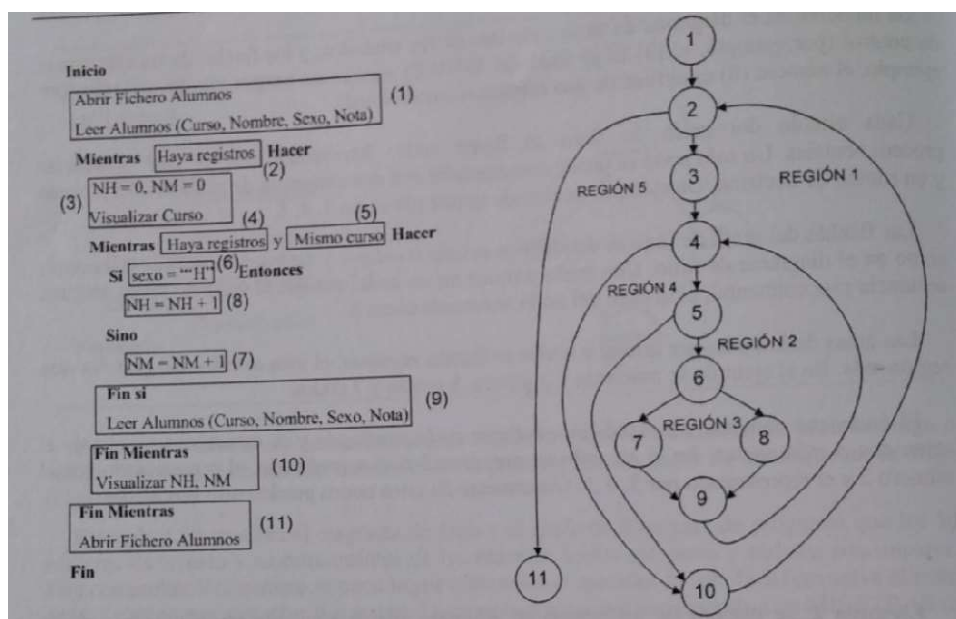
Las áreas delimitadas por aristas y nodos se llaman **regiones**, el área exterior del grafo es otra región más. En este ejemplo hay 3 regiones, 8 aristas y 7 nodos.

El nodo que contiene una condición se llama **nodo predicado** y se caracteriza porque de él salen dos o más aristas. En el ejemplo hay 2 nodos predicados, el representado por el número 2 y el representado por 3, 4, 5. Únicamente de estos dos nodos pueden salir dos aristas.

Ejemplo 2: Se dispone de un fichero de Alumnos con la siguiente estructura de registro: *Curso*, *Nombre*, *Sexo* (puede ser H o M) y *Nota*. El fichero está ordenado ascendentemente por *Curso*. Vamos a realizar un proceso que lea los registros del fichero y muestre por cada curso el número de hombres y el número de mujeres. Se construirá el pseudocódigo y el grafo de flujo. En el pseudocódigo se muestra en un recuadro las sentencias que representan cada nodo; al lado hay un número entre paréntesis que se corresponde con su nodo en el grafo de flujo. La estructura principal es un MIENTRAS (mientras haya registros en el fichero) y dentro hay un SI (para contar los hombres y las mujeres).

En este ejemplo hay 5 regiones, 14 aristas y 11 nodos (4 de ellos son nodos predicados).

Cuando en un diseño procedimental se encuentran condiciones compuestas, es decir cuando en una condición aparecen uno o más operadores (como en el ejemplo *Haya registros* y *Mismo curso*) se complica la generación del grafo del flujo. En este caso se crea un nodo aparte para cada una de las condiciones; nodos 4 y 5.



COMPLEJIDAD CICLOMÁTICA

La complejidad ciclomática es una métrica del software que proporciona una medida cuantitativa de la complejidad lógica del programa. En el método de prueba del camino básico, la complejidad ciclomática establece el número de caminos independientes del conjunto de caminos de ejecución de un programa, y por lo tanto, el número de casos de prueba que se deben ejecutar para asegurar que cada sentencia se ejecuta al menos una vez.

La complejidad ciclomática $V(G)$ se puede calcular de tres formas:

1. $V(G) = \text{Número de regiones del grafo}$
2. $V(G) = \text{Aristas} - \text{Nodos} + 2$
3. $V(G) = \text{Nodos predichados} + 1$

Para el **Ejemplo 1**, la complejidad ciclomática es **3**

1. $V(G) = \text{Número de regiones del grafo} = 3$
2. $V(G) = \text{Aristas} - \text{Nodos} + 2 = 8 - 7 + 2 = 3$
3. $V(G) = \text{Nodos predichados} + 1 = 2 + 1 = 3$

Para el **Ejemplo 2**, la complejidad ciclomática es **5**

1. $V(G) = \text{Número de regiones del grafo} = 5$
2. $V(G) = \text{Aristas} - \text{Nodos} + 2 = 14 - 11 + 2 = 5$
3. $V(G) = \text{Nodos predichados} + 1 = 4 + 1 = 5$

Se establecen los siguientes valores de referencia de la complejidad ciclomática:

| Complejidad Ciclométrica | Evaluación del riesgo |
|--------------------------|--|
| Entre 1 y 10 | Programas o métodos sencillos, sin mucho riesgo |
| Entre 11 y 20 | Programas o métodos más complejos, riesgo moderado |
| Entre 21 y 50 | Programas o métodos complejos, alto riesgo |
| Mayor que 50 | Programas o métodos no testeables, muy alto riesgo |

El valor de $V(G)$ nos da el nº de caminos independientes del conjunto básico de un programa. Un **camino independiente** es cualquier camino del programa que introduce, por lo menos un nuevo conjunto de sentencias o una condición. En el diagrama de flujo, un camino independiente está constituido por los menos por una arista que no haya sido recorrida anteriormente.

En el **Ejemplo 1**, un conjunto de caminos independientes será:

- **Camino 1:** 1 -2- 9
- **Camino 2:** 1 – 2 – 3, 4, 5 – 6 – 8 – 2 - 9
- **Camino 3:** 1 – 2 – 3, 4, 5 – 7 – 8 – 2 – 9

En el **Ejemplo 2**, un conjunto de caminos independientes será:



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

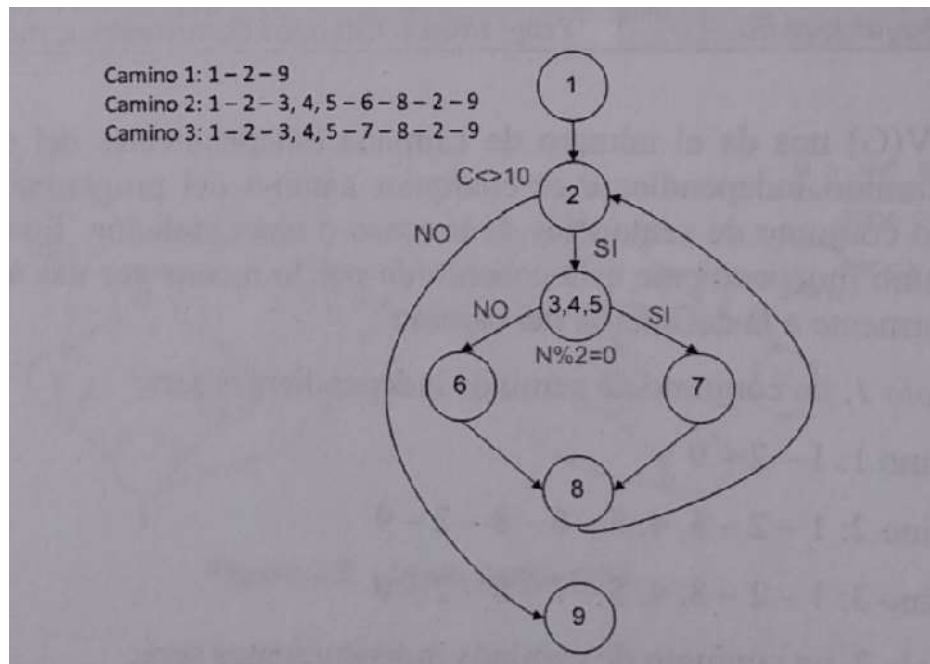
OBTENCIÓN DE LOS CASOS DE PRUEBA

El último paso de la prueba del camino básico es construir los casos de prueba que fuerzan la ejecución de cada camino. Con el fin de comprobar cada camino, debemos escoger los casos de prueba de forma que las condiciones de los nodos predicado estén adecuadamente establecidos.

Una forma de representar el conjunto de los casos de prueba es forma de tabla.

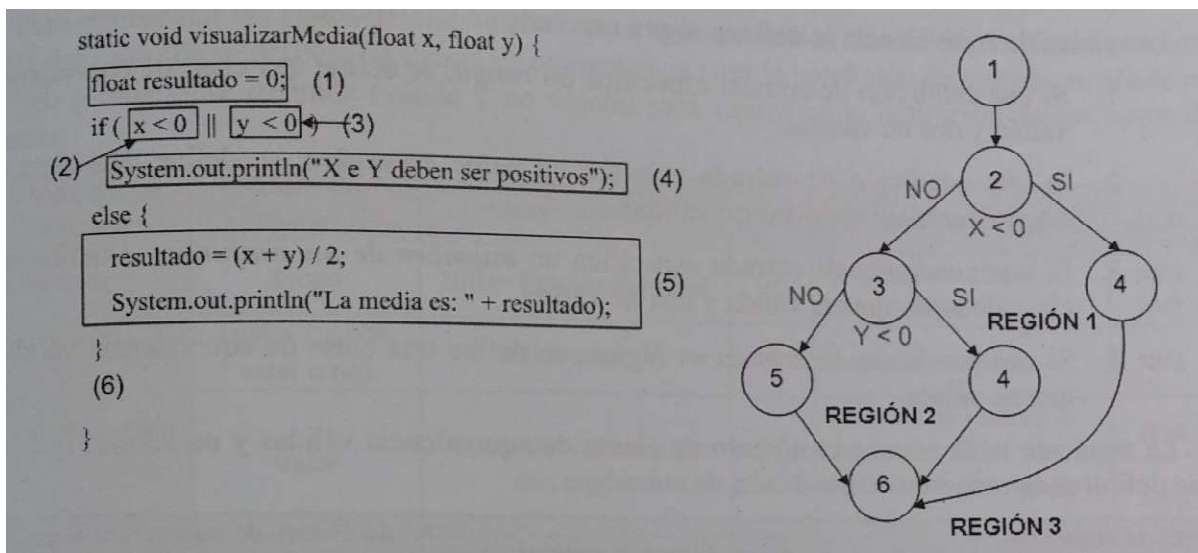
Para el ejemplo 1, podemos representar los casos de prueba en la siguiente tabla, mostrando los nodos predicado con sus condiciones para que sea más fácil obtener los casos de prueba:

| Camino | Casos de Prueba | Resultado esperado |
|--------|---|---|
| 1 | Escoger algún valor de C tal que NO se cumpla la condición $C \leq 10$ $C = 10$ | Visualizar el nº de pares y el de impares |
| 2 | Escoger algún valor de C tal que SI se cumpla la condición $C \leq 10$ Escoger algún valor de N tal que NO se cumpla la condición $N \% 2 = 0$ $C = 1, N = 5$ | Contar los nº impares |
| 3 | Escoger algún valor de C tal que SI se cumpla la condición $C \leq 10$ Escoger algún valor de N tal que SI se cumpla la condición $N \% 2 = 0$ $C = 2, N = 4$ | Contar los nº pares |



El camino 1 no puede ser probado por sí solo, debe ser probado como parte de las pruebas de los caminos 1 y 2.

Ejemplo 3: Calcular la complejidad ciclométrica, los caminos independientes y los casos de prueba para la siguiente función de Java:



Los caminos independientes y los casos de prueba para cada camino se muestran en esta tabla:

| Camino | Casos de Prueba | Resultado esperado |
|------------------------|--|--------------------------------------|
| Camino 1: 1-2-3-5-6 | Escoger algún valor de X e Y tal que NO se cumpla la condición $X < 0 \parallel Y < 0$ $X = 4, Y = 5$ VisualizarMedia(4,5) | Visualiza: La media es 4.5 |
| Camino 2: 1-2-4-6 | Escoger algún valor de X tal que SI se cumpla la condición $X < 0$ (Y puede ser cualquier valor) $X = -4, Y = 5$ VisualizarMedia(-4,5) | Visualiza: X e Y deben ser positivos |
| Camino 3: 1-2-3-4-6 | Escoger algún valor de X tal que NO se cumpla la condición $X < 0$ y escoger algún valor de Y que SI cumpla la condición $Y < 0$ $X = 4, Y = -5$ VisualizarMedia(4,-5) | Visualiza: X e Y deben ser positivos |

A) Particiones o clases de equivalencia

La partición equivalente es un método de prueba de caja negra que divide los valores de los campos de entrada de un programa en clases de equivalencia. Por ejemplo supongamos un campo de entrada llamado *número de empleado*, definido con una serie de condiciones: numérico de 3 dígitos y el primero no puede ser 0. Entonces se puede definir una clase de equivalencia no válida: *número de empleado* < 100; y otra válida: *número de empleado* comprendido entre 100 y 999.

Para identificar las clases de equivalencia se examina cada condición de entrada (son parte del dominio de valores de entrada y normalmente son una frase de la especificación) y se divide en dos o más grupos. Se definen dos tipos de clases de equivalencia:

- **Clases válidas:** son los valores de entrada válidos.
- **Clases no válidos:** son los valores de entrada no válidos.

Las clases de equivalencia se definen según una serie de directrices:

1. Si una condición de entrada especifica un **rango**, se define una clase de equivalencia válida y dos no válidas.
2. Si una condición de entrada requiere un **valor específico**, se define una clase de equivalencia válida y dos no válidas.
3. Si una condición de entrada requiere un **miembro de un conjunto**, se define una clase de equivalencia válida y una no válida.
4. Si una condición de entrada es **lógica**, se define una clase de equivalencia válida y una no válida.


Tabla resumen del número de clases de equivalencias válidas y no válidas que hay que definir para cada tipo de condición de entrada.

| Condiciones de entrada | Nº de clase de equivalencia válidas | Nº de clase de equivalencia no válidas |
|---------------------------|--|--|
| 1. Rango | 1 CLASE VÁLIDA Contempla los valores del rango | 2 CLASES NO VÁLIDAS Un valor por encima del rango Un valor por debajo del rango |
| 2. Valor específico | 1 CLASE VÁLIDA Contempla dicho valor | 2 CLASES NO VÁLIDAS Un valor por encima Un valor por debajo |
| 3. Miembro de un conjunto | 1 CLASE VÁLIDA Una clase por cada uno de los miembros del conjunto | 1 CLASE NO VÁLIDA Un valor que no pertenece al conjunto |
| 4. Lógica | 1 CLASE VÁLIDA Una clase que cumpla la condición | 1 CLASE NO VÁLIDA Una clase que no cumpla la condición |

Ejemplo 4: Se va a realizar una entrada de datos de un empleado por pantalla, se definen 3 campos de entrada y una lista para elegir el oficio. La aplicación acepta los datos de esta manera:

- *Empleado:* número de tres dígitos que no empiece por 0.
- *Departamento:* en blanco o número de dos dígitos.
- *Oficio:* Analista, Diseñador, Programador o Elige oficio.

Si la entrada es correcta el programa asigna un salario (que se muestra por pantalla) a cada empleado según estas normas:

| | | |
|---|--|---|
|  CFP VIRGEN DE GRACIA | <p align="center">UT3-DISEÑO Y REALIZACIÓN DE PRUEBAS Entornos de desarrollo (1º DAW)</p> | <p align="center">Dpto. INFORMÁTICA</p> <p align="center">Curso: 2022-23</p> |
|---|--|---|

- S1 si el *Oficio* es Analista se asigna 2500.
- S2 si el *Oficio* es Diseñador se asigna 1500.
- S3 si el *Oficio* es Programador se asigna 2000.

Si la entrada no es correcta el programa muestra un mensaje indicando la entrada incorrecta:

- ER1 si el *Empleado* no es correcto.
- ER2 si el *Departamento* no es correcto.
- ER3 si no se ha elegido *Oficio*.

Para **representar las clases de equivalencia** para cada condición de entrada se puede usar una tabla. En cada fila se definen las clases de equivalencia para la condición de entrada, se añade un código a cada clase definida (válida y no válida) para usarlo en la definición de los casos de prueba:

| Condiciones de entrada | Clases de equivalencia | Clases válidas | COD | Clases no válidas | COD |
|------------------------|---------------------------|---|----------------|--|------------|
| Empleado | Rango | 100>= Empleado<=999 | V1 | Empleado < 100 Empleado > 999 | NV1 NV2 |
| | Lógica (puede estar o no) | En blanco | V2 | No es un número | NV3 |
| Departamento | Valor | Cualquier número de dos dígitos | V3 | Nº de más de 2 dígitos Nº de menos de 2 dígitos | NV4 NV5 |
| | Miembro de un conjunto | Oficio="Programador" Oficio="Analista" Oficio="Diseñador" | V4 V5 V6 | Oficio="Elige Oficio" | NV8 |

A partir de esta tabla **se generan los casos de prueba**. Utilizamos las condiciones de entrada y las condiciones de equivalencia (a las que se les ha asignado un código en la columna COD, también se podría haber asignado un número a cada clase). Los representamos en otra tabla donde cada fila representa un caso de prueba con los códigos de las clases de equivalencia que se aplican, los valores asignados a las condiciones de entrada y el resultado esperado según el enunciado del problema:

| CASO DE PRUEBA | Clases de equivalencia | CONDICIONES DE ENTRADA | | | Resultado esperado |
|----------------|------------------------|------------------------|--------------|--------------|--------------------|
| | | Empleado | Departamento | Oficio | |
| CP1 | V1, V3, V4 | 200 | 20 | Programador | S3 |
| CP2 | V1, V2, V5 | 250 | | Analista | S1 |
| CP3 | V1, V3, V6 | 450 | 30 | Diseñador | S2 |
| CP4 | V1, V2, V4 | 220 | | Programador | S3 |
| CP5 | NV1, V3, V6 | 90 | 35 | Analista | ER1 |
| CP6 | V1, NV3, V5 | 100 | AD | Diseñador | ER2 |
| CP7 | V1, V2, NV8 | 300 | | Elige oficio | ER3 |
| CP8 | V1, NV4, V6 | 345 | 123 | Diseñador | ER2 |
| ... | | | | | |

Al rellenar la tabla de casos de prueba se han tenido en cuenta estas dos reglas: los casos de prueba válidos (CP1, CP2, CP3 y CP4) cubren tantas clases de equivalencia válidas como sea posible y los casos de prueba no válidos (CP5, CP6, CP7 y CP8) cubren una sola clase no válida (si se prueban múltiples clases de equivalencia no válidas en el mismo caso de prueba, puede ocurrir que alguna de estas pruebas nunca se ejecuten porque la primera enmascara a las otras o termina la ejecución del caso del prueba).

Los casos de prueba se van añadiendo a la tabla hasta que todas las clases de equivalencia válidas y no válidas hayan sido cubiertas. Por ejemplo a la tabla anterior le faltan clases de equivalencia válidas: (V1, V2, V6) y (V1, V3, V5); y no válidas (NV2, V2, V4) y (V1, NV5, V6).

Ejemplo 5: La siguiente función Java recibe un número entero y devuelve una cadena con el texto “par” si el número recibido es par, o “Impar” si el número es impar.

```
public string parImpar(int nume) {
    String cad="";
    if(num % 2 == 0)
        cad="Par";
    else
        cad="Impar";
    return cad;
}
```

Determinar los casos de prueba.

En este ejemplo tenemos una condición de entrada que requiere un valor específico, un número entero, entonces según la segunda directriz se define una clase de equivalencia válida y dos no válidas. Como en ese caso los números son tratados de forma diferente podemos crear una clase de equivalencia para cada entrada válida.

| Condiciones de entrada | Clases de equivalencia | Clases válidas | COD | Clases no válidas | COD |
|------------------------|------------------------|---------------------------|-----|------------------------|--------------|
| nume | Valor par | Cualquier nº entero par | V7 | Número impar Cadena | NV9 NV10 |
| | Valor impar | Cualquier nº entero impar | V8 | Número par Cadena | NV11 NV12 |

Los casos de prueba serían:

| CASO DE PRUEBA | Clases de equivalencia | CONDICIONES DE ENTRADA nume | Resultado esperado |
|----------------|------------------------|--------------------------------|----------------------|
| CP1 | V7 | 20 | Par |
| CP2 | V8 | 25 | Impar |
| CP3 | NV9 | 45 | Error, número impar |
| CP4 | NV10 | “we” | Error, es una cadena |
| CP5 | NV11 | 10 | Error, número par |
| CP6 | NV12 | “ad” | Error, es una cadena |

B) Análisis de valores límite

El análisis de valores límite se basa en que los errores tienden a producirse con más probabilidad en los límites o extremos de los campos de entrada.

Esta técnica complementa a la anterior y los casos de prueba elegidos ejercitan los valores justo por encima y por debajo de los márgenes de la clase de equivalencia. Además no sólo se centra en las condiciones de entrada, sino que también se exploran las condiciones de salida definiendo las clases de equivalencia de salida.

Las reglas son las siguientes:

1. Si una condición de entrada especifica un **rango de valores**, se deben diseñar casos de prueba para los límites del rango y para los valores justo por encima y por debajo del rango. Por ejemplo, si una entrada requiere un rango de valores enteros comprendidos entre 1 y 10, hay que escribir casos de prueba para el valor 1, 10, 0 y 11.
2. Si una condición de entrada especifica un **número de valores**, se deben diseñar casos de prueba que ejerciten los valores máximos, mínimo, un valor justo por encima del máximo y un valor justo por debajo del mínimo. Por ejemplo, si el programa requiere de dos a diez datos de entrada, hay que escribir casos de prueba para 2, 10, 1 y 11 datos de entrada.
3. Aplicar la regla 1 para la condición de salida. Por ejemplo, si se deben aplicar sobre un campo de salida un descuento de entre un 10% mínimo y un 50% máximo (dependiendo del tipo de cliente); se generarán casos de prueba para 9,99%, 10%, 50% y 50,01%.
4. Usar la regla 2 para la condición de salida. Por ejemplo, si la salida de un programa es una tabla de temperaturas de 1 a 10 elementos, se deben diseñar casos de prueba para que la salida del programa produzca 0, 1, 10 y 11 elementos. Tanto en esta regla, como en la anterior, hay que tener en cuenta que no siempre se podrán generar resultados fuera del rango de salida.
5. Si las estructuras de datos internas tienen **límites preestablecidos** (por ejemplo un array de 100 elementos), hay que asegurarse de diseñar casos de prueba que ejercite la estructura en sus límites, primer y último elemento.

Ejemplo: Determinar los casos de prueba para los siguientes elementos según las condiciones de entrada y salida:

| | Condiciones de entrada y salida | Casos de prueba |
|---------------------------|------------------------------------|--|
| Código | Entero de 1 a 100 | Valores: 0, 1, 100, 101 |
| Puesto | Alfanumérico de hasta 4 caracteres | Longitud de caracteres: 0, 1, 4, 5 |
| Antigüedad | De 0 a 25 años (real) | Valores: 0, 25, -0.1, 25.1 |
| Horas semanales | De 0 a 60 | Valores: 0, 60, -1, 61 |
| Fichero de entrada | Tiene de 1 a 100 registros | Para leer 0, 1, 100 y 101 registros |
| Fichero de salida | Podrá tener de 0 a 10 registros | Para generar 0, 10 y registros |

Array interno

De 20 cadenas de caracteres

(no se puede generar -1 registros)
Para el primer y el último elemento

Ejemplo: Partimos del Empleado (que tiene un número de tres dígitos que no empiece por 0) del ejemplo 4. Utilizando esta técnica, para la clase de equivalencia V1 que representa un rango de valores ($100 \leq \text{Empleado} \leq 999$) se deben generar dos casos de prueba con el límite inferior y superior del rango (para identificar estos casos de prueba utilizamos V1a para el límite inferior y V1b para el superior):

| CASO DE PRUEBA | Clases de equivalencia | CONDICIONES DE ENTRADA | | | Resultado esperado |
|----------------|------------------------|------------------------|--------------|---------------|--------------------|
| | | Empleado | Departamento | Oficio | |
| CP11 | V1a, V3, V4 | 100 | 20 | "Programador" | S3 |
| CP12 | V1b, V2, V5 | 999 | | "Analista" | S1 |
| CP13 | NV1, V3, V6 | 99 | 30 | "Diseñador" | ER1 |
| CP14 | NV2, V2, V4 | 1000 | | "Programador" | ER1 |

7.