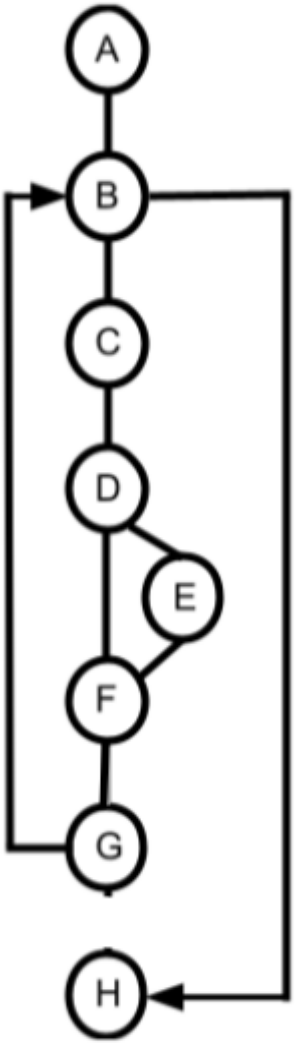
Ejercicio 1

```
A | Begin
      read (Registro);
  B | while not eof() do
        |   IRPF = CalculaIRPF (Registro.Sueldo, Registro.IRPF);
      C |   SS = CalculaSS (Registro.Sueldo);
        |   Salario = CalculaSalario (Registro.Sueldo, IRPF, SS);
        |   PrintNomina (Registro.Persona, Salario);
      D |   if MesActual = 'Diciembre' and Registro.MesesTrabajados >= 6 then
            E |   PagaExtra = CalculaPagaExtra (Registro.Extra, IRPF, SS);
              |   PrintNomina (Registro.Persona, PagaExtra);
      F |   endif
      G |   read (Registro);
        | enddo
H | End
```



4

a)
v(G) = #aristas - #nodos + 2 = 9 - 8 + 2 = 3
v(G) = #areasCerradas + 1 = 2 + 1 = 3

Complejidad ciclomática de 3

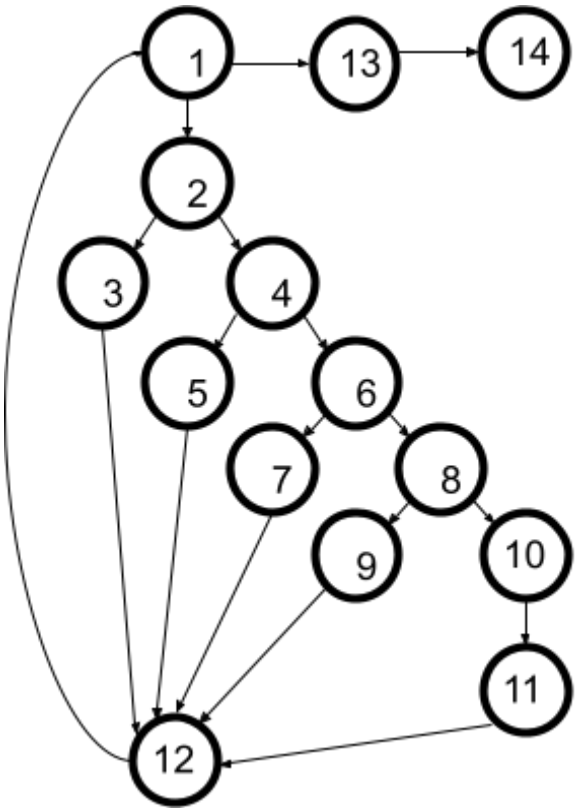
b)
D1: not eof()
D2: MesActual = ‘Diciembre’ & Registro.MesesTrabajados >= 6
C21: MesActual = ‘Diciembre’
C22: Registro.MesesTrabajados >= 6

D1	C21	C22	D2	eof()	MesActual	Registro.MesesTrabajados
T	T	T	T	T	‘Diciembre’	6
T	F	F	F	T	‘Enero’	5
F	-	-	-	F	-	-

Ejercicio 2

	`public int comprobacion (int a, int b, int c) {`
	`int valor=0;`
1	`while (valor<=0) {`
2	`if (a<0 & b<0 & c<0)`
3	`valor=a+b;`
4	`else if (a<0)`
5	`valor=a;`
6	`else if (b<0)`
7	`valor=b;`
8	`else if (c<0)`
9	`valor=c;`
10	`else`
11	`valor=a+b+c;`
12	`}`
13	`return valor;`
14	`}`

a) Este es el grafo de flujo G del código anterior (los números de los nodos son los números de cada línea del programa):



Y para calcular la complejidad ciclomática, V(G), lo hacemos por los dos métodos:

- V(G) = nº aristas - nº nodos + 2 = 18 - 13 + 2 = **6**
- V(G) = nº de zonas cerradas + 1 = 6 + 1 = **6**

b) En el código anterior hay cuatro decisiones:

D1: “a<0 & b<0 & c<0”, que a su vez está compuesta por tres condiciones:

C11: a<0

C12: b<0

C13: c<0

D2: C11: a<0

D3: C12: b<0

D4: C13: c<0

Hay que garantizar que cada condición tome al menos una vez el valor *True* y el *False*. Los datos concretos para los casos de prueba podrían ser los siguientes:

		Valor verdadero	**Valor falso**
D1	**C11**	a = -1	a = 1
	C12	b = -1	b = 1
	C13	c = -1	c = 1
D2		a = -1	a = 1
D3		b = -1	b = 1
D4		c = -1	c = 1

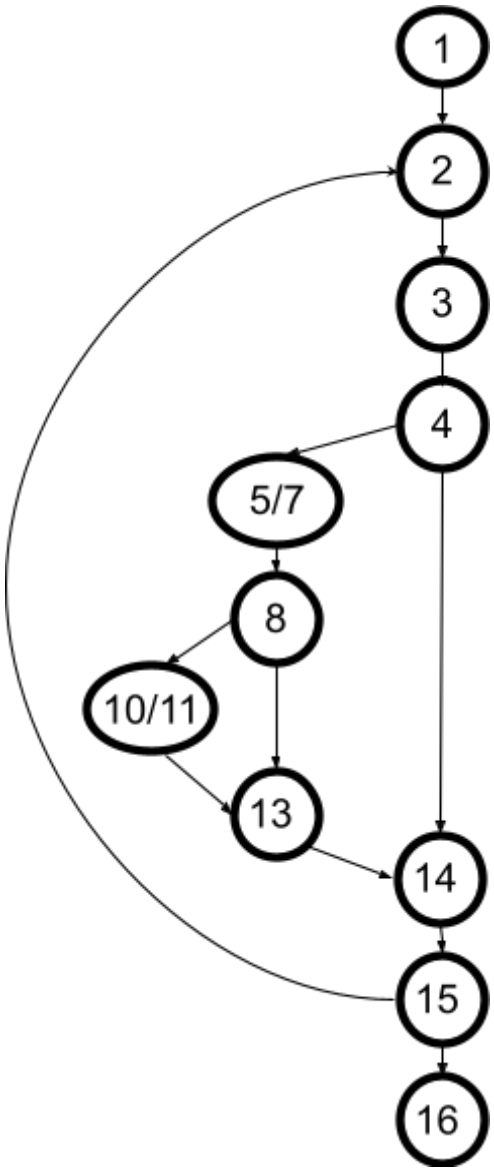
Y la tabla de casos de prueba quedaría así:

Caso de prueba	**D1**	**D2**	**D3**	**D4**	**Entradas**	**Salida**
1	V	V	V	V	a =b = c = 1	valor = 2
2	F	V	F	F	a = 1, b = c = -1	valor = 1
3	F	F	V	F	b = 1, a = c = -1	valor = 1
4	F	F	F	V	c = 1, a = b = -1	valor = 1
5	F	F	F	F	a = b = c = -1	valor = 3

c) Aunque otras combinaciones de valores serían posibles (por ejemplo, D2=D3=Verdadero, y D4=Falso), estas no producen una cobertura diferente de las líneas de código, así como un cambio en el valor de la salida (en el ejemplo anterior, D1 sigue siendo Falso, y la línea de D3 no se ejecuta porque antes se ejecuta la de D2, que después salta directamente al final de la cadena de *if-else*). Por tanto, aunque por *combinatoria* de valores V y F se podrían diseñar más de 5 casos de prueba, esto sería redundante, ya que la tabla anterior ya recoge todos los escenarios *diferentes* que se pueden producir con el código dado.

Complejidad ciclomática líneas 2-11: 5 corresponde al número mínimo de casos de prueba para la cobertura de decisión, que es igual al nº de casos de prueba diseñados para la cobertura de decisión/condición. Por tanto no faltan casos de prueba.

Ejercicio 3




```
1   Begin
2     repeat
3         LeerDatosJuegoUsuario(Datos);
4         if not eof() then
5             Puntuacion=CalculaPuntuacion(Datos.JuegosJugados, Datos.Puntos);
6             RegistrarPuntuacion(Datos.Usuario, Puntuacion);
7             Bonificacion=0;
8             if  Datos.NivelJuego>4  and  Datos.JuegosJugados>=3
9             then
10                Bonificacion=CalculaBonus(Datos.JuegosJugados, Datos.Bonus);
11                RegistrarBonificacion(Datos.Usuario, Datos.NivelJuego);
12            endif
13            ImprimePuntuacion(Datos.Usuario, Puntuacion, Bonificacion);
14        endif
15    until eof();
16  End
```

Y para calcular la complejidad ciclomática, V(G), lo hacemos por los dos métodos:

- V(G) = nº aristas - nº nodos + 2 = 13 - 11 + 2 = **4**
- V(G) = nº de zonas cerradas + 1 = 3 + 1 = **4**

FUNCTION ValidaUsuario (Usuario: String): Integer;

Atributo	Válida	Inválida
Usuario	([a-z|A-Z][a-z |A-Z|" "]^{8}#) 10 caracteres 1º caracter [a-z] 8 caracteres siguientes [a-z|A-Z] Último caracter = #	(!([a-z|A-Z][a-z |A-Z|" "]^{8}#)) < 10 caracteres > 10 caracteres 1º caracter != [a-z|A-Z] 8 caracteres siguientes != [a-z|A-Z|" "] Último caracter != #
CODIGO_ERROR	{0, -1, -2, -3}	!= {0, -1, -2, -3}

Caso de prueba	Entradas	Salida
1	Usuario=aaaaaaaaa#	valor = 0
2	Usuario=1aaaaaaaa#	valor = -3
3	Usuario=Za bDfaaa#	valor = 0
4	Usuario=a,DEkgLnu#	valor = -2
5	Usuario=aaaaaaaaaa#	valor = -1

Ejercicio 4

Suponiendo que el hospital se fundó en 1990 y a fecha de hoy estamos en 2022.

Atributo	**Válida**	**Inválida**
F_Ingreso	DD-MM-AAAA, donde **1.** DD = [1, 31] si MM es impar de [1, 7] o par de [8, 12]. **2.** DD = [1, 30] si MM es par de [2, 6] o impar con valor {9, 11}. **3.** DD = [1, 28] si MM es 2. **4.** MM = [1, 12]. **5.** AAAA = [1990, 2022].	**6.** Si formato diferente a DD-MM-AAA. **7.** Si DD o MM o AAAA diferente a número. **8.** DD < 1 o DD > 28 si MM es 2. **9.** DD < 1 o DD > 30 si MM es par y diferente de 10. **10.** DD < 1 o DD > 31 si MM es impar o 10. **11.** MM < 1 o MM > 12. **12.** AAAA < 1990 o AAAA > 2022.
F_Alta	DD-MM-AAAA, donde **13.** DD = [1, 31] si MM es impar de [1, 7] o par de [8, 12]. **14.** DD = [1, 30] si MM es par de [2, 6] o impar con valor {9, 11}. **15.** DD = [1, 28] si MM es 2. **16.** MM = [1, 12]. **17.** AAAA = [1990, 2022]. **43.** > F_Ingreso	**18.** Si formato diferente a DD-MM-AAA. **19.** Si DD o MM o AAAA diferente a número. **20.** DD < 1 o DD > 28 si MM es 2. **21.** DD < 1 o DD > 30 si MM es par y diferente de 10. **22.** DD < 1 o DD > 31 si MM es impar o 10. **23.** MM < 1 o MM > 12. **24.** AAAA < 1990 o AAAA > 2022.
Fallecido	**25.** [0, 1].	**26.** < 0 o > 1.
Residencia	**27.**$[A\text{-}Z]^N$, N=3.	**28.** $[A\text{-}Z]^N$, N != 3. **29.** Alguno de los caracteres no son letras o no son letras mayúsculas.
Edad	**30.** $[0\text{-}9]^2$.	**31.** < 00 o > 99. **32.** Alguno de los dos caracteres es no numérico.
Género	**33.** {M, F}.	**34.** != {M, F}.
Código_UCI	**35.** [A-D]$[0\text{-}9]^2$.	**36.** Primer caracter != [A-D]. **37.** Segundo o tercer caracter no numérico. **38.** Longitud de la cadena != 3.
Salida	**39.** {ContagioTipo$[0\text{-}9]^2$,	**40.** El tipo de contagio no

	[001, 999]}.	comienza por la cadena ‘ContagioTipo’ seguida de 2 dígitos entre 0 y 9. **41.** La duración del enfermo es < 001 o > 999. **42.** La duración del enfermo contiene algún caracter no numérico.