uc3m

Universidad **Carlos III** de Madrid

Programación - Curso 2020-2021

Introducción a los Diagramas de Flujo

Grado en Ingeniería Informática

Doble Grado en Ingeniería Informática y Administración de Empresas

Índice

1 Instrucciones generales

Durante este curso se deberán realizar prácticas **semanales**.

La puntuación máxima total obtenible por la entrega de estas prácticas es de 0,5 puntos. Para obtener puntuación en este apartado es necesario haber entregado al menos el 80% de las prácticas.

En la clase de prácticas posterior a la entrega de cada práctica semanal, se solicitará a alguno de los alumnos que explique **oralmente** la solución a uno de los ejercicios entregados. La exposición oral tendrá una puntuación total máxima de 0,5 puntos.

Las prácticas deberán realizarse de forma individual.

Se puede entregar cualquier práctica semanal aunque no se haya entregado la anterior

Aunque el peso específico de las prácticas semanales en la nota global de la asignatura puede parecer pequeño, la realización de estos ejercicios semanales es fundamental para la comprensión de la asignatura y para obtener el conocimiento necesario para aprobar el examen final. Estos ejercicios semanales también constituyen la base para la realización del proyecto final, por lo que es muy recomendable seguir la asignatura desde el principio realizando todas las prácticas asignadas

2 Segunda práctica

Esta práctica tiene como objetivo familiarizar al alumno con la metodología de la programación y en particular con el diseño y desarrollo de algoritmos que se representarán a muy alto nivel utilizando diagramas de flujo, mediante una serie de ejercicios básicos.

3 Introducción general

Programar implica ser capaces de analizar un problema, determinar cuáles son las acciones que necesitamos realizar para resolverlo, dividir estas acciones en los pasos básicos que es capaz de realizar un ordenador y posteriormente codificarlas en el lenguaje de programación elegido.

A menudo se limita la importancia de dividir el problema en sus pasos fundamentales antes de implementarlo en el lenguaje correspondiente, lo que conlleva programas que, en caso de no funcionar, son tremendamente difíciles de poder arreglar (o, como veremos en prácticas posteriores, de “depurar”). El resultado suele ser lo que se conoce como “código espagueti”.

Este segundo ejercicio semanal tratará de centrarse en la importancia de estos planteamientos previos, a través de la herramienta fundamental de los diagramas de flujo. Esta herramienta intenta recurrir a un lenguaje que sea fácilmente entendible por los seres humanos, representando un algoritmo mediante una serie de gráficos predefinidos. Esta manera de representar los programas facilita el primer acercamiento al mundo de la programación, pero, al mismo tiempo, sigue siendo una herramienta imprescindible para programadores ya experimentados.

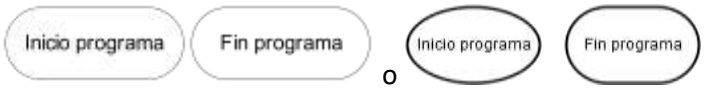
4 Diagramas de flujo

Los diagramas de flujo representan los pasos de un proceso cualquiera (en nuestro caso un programa) a través de una serie de flechas, que marcan la dirección en la que se mueve la ejecución del proceso. La

utilidad de estos diagramas está relacionada con nuestra capacidad como seres humanos para procesar símbolos, lo que hace que se facilite el proceso de detección de fallos y planteamiento de los problemas.

Inicio y final del diagrama

Un diagrama de flujo debe tener un **punto de inicio** y un **punto de final**, y el final siempre debe ser alcanzable desde el principio. Los puntos de inicio y de final se representan con los siguientes símbolos:



Definición de variables

Un programa tiene que resolver una necesidad de un usuario (que puede ser un ser humano u otro programa). Por lo tanto debemos ser capaces de manejar información, que almacenaremos en la memoria del ordenador mediante **variables**[1]**.** Estas variables, como su propio nombre indica, pueden cambiar de valor a lo largo de la ejecución del programa. Para dar el valor a una variable (almacenar un determinado dato en la memoria) utilizaremos el símbolo ←[2], que pondrá el valor de lo que haya a la derecha en la variable de la izquierda.

Ejemplo: asignación de valor a una variable

Variable ← *1*
Variable ← *Variable+1*

En numerosos lenguajes, entre ellos Java, las variables tienen un tipo fijo explícito, que determina qué tipo o clase de valores pueden guardar (valores enteros, reales, cadenas de texto, matrices…). Sin embargo, en otros lenguajes, como Python, el tipo de las variables se determina por su contenido (tipo inferido). Los tipos de las variables no se suelen representar en los diagramas de flujo.

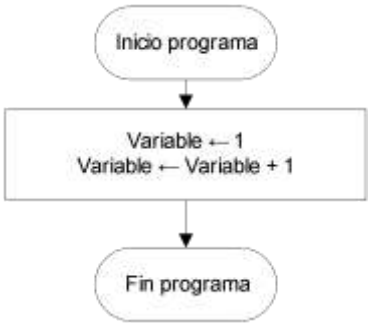
Instrucciones

Básicamente lo que hace un programa es transformar variables (transformar datos) hasta alcanzar el resultado deseado. Se puede pensar en cualquier calculadora de mano, que va tomando números y haciendo operaciones con ellos hasta llegar al valor final. En nuestro caso esas transformaciones serán más complicadas que simples operaciones matemáticas, pero el proceso es el mismo. Todas esas transformaciones son lo que llamamos instrucciones, que se representan con el siguiente símbolo:

Las asignaciones a variables que veíamos antes no dejan de ser instrucciones, por lo tanto deben estar dentro de su rectángulo correspondiente, como vemos en el siguiente ejemplo de diagrama:

[1] Una variable no es más que una posición de la memoria a la que nos referimos mediante un nombre. Así no hay que recordar la posición exacta de la memoria en la que hemos guardado un dato, sino solamente el nombre que le dimos.
[2] En muchos lenguajes de programación se usa el signo "=" en lugar de "←" para asignar un valor a una variable. No se debe confundir este símbolo con la igualdad matemática: la forma de interpretarlo es "asigna el valor que hay a la derecha a la variable de la izquierda".

Ejemplo: diagrama de flujo para la asignación de valor a una variable y luego incrementar ese valor en una unidad[3].



Operaciones de entrada / salida

Como hemos comentado anteriormente, un programa existe para satisfacer la necesidad de un usuario, por lo que necesitamos poder comunicarnos con él. Aunque este usuario puede no ser humano, para facilitar las explicaciones supondremos que en nuestros programas siempre lo será, al menos de momento. ¿Cómo podemos comunicarnos con el usuario? A través de las operaciones de **entrada / salida**.

Una operación de **salida** representa una información que deseemos mostrarle al usuario por pantalla. Para indicarlo en nuestros diagramas de flujo se utiliza el símbolo siguiente:

Uno de los programas más famosos en cualquier lenguaje de programación es el llamado “Hola mundo”, que muestra ese mensaje por pantalla. Veamos con un ejemplo cómo podríamos hacerlo en nuestros diagramas de flujo:

Ejemplo: Hola mundo como diagrama de flujo.

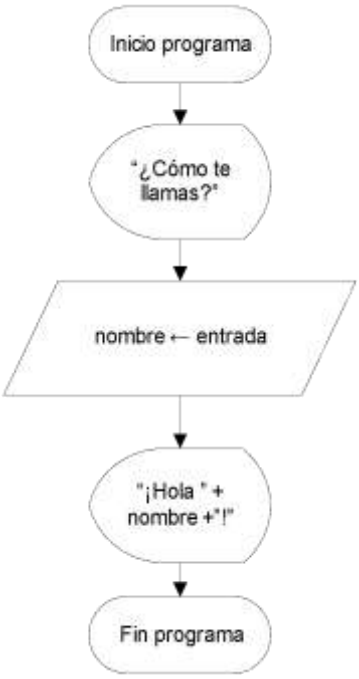
Es importante que os fijéis en que la cadena de texto va entre comillas, lo que indica que se reproduce literalmente. Si fueran variables que contuvieran texto, no se utilizarían las comillas (ver el ejemplo de saludo personalizado posterior).

[3] Nótese que cuando tenemos varias instrucciones seguidas podemos poner cada una en su rectángulo o agruparlas todas en el mismo rectángulo.

Una operación de **entrada**, por el contrario, permite que el usuario introduzca información que se utilizará como parte de nuestro programa. Se representa mediante el siguiente símbolo:

Pensemos en el ejemplo anterior. Decir "Hola mundo" está bien, pero queda muy impersonal. Sería más agradable poder preguntarle el nombre al usuario y saludarle directamente, ¿no? Veamos cómo podríamos hacerlo en nuestros diagramas de flujo:

Ejemplo: Saludo personalizado



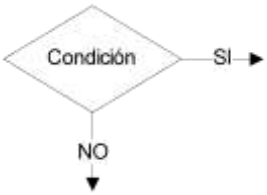
Una vez más, es interesante que os fijéis en la diferencia entre mostrar una cadena literalmente (entre comillas) o una variable con texto (sin comillas). Además, veis que el símbolo "+", además de para la suma aritmética, sirve para indicar la unión de cadenas.

¿Qué se mostraría por pantalla? Algo así:

```
¿Cómo te llamas?
José Luis     (esto lo escribe el usuario, por el teclado)
¡Hola José Luis!
```

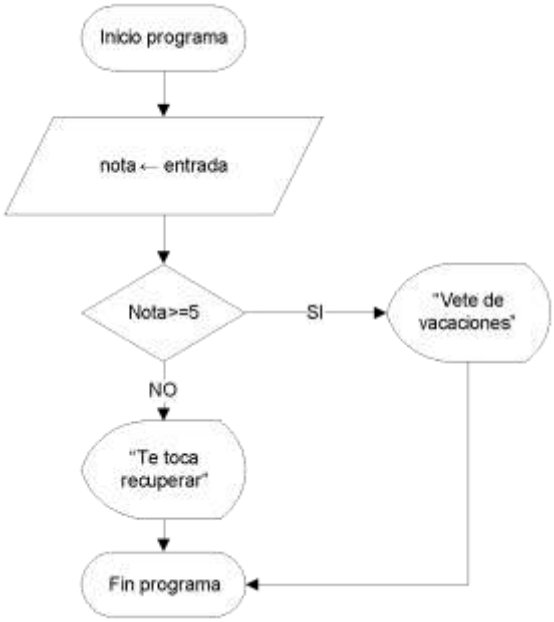
Control de flujo: operaciones condicionales

Sin duda, una de las cuestiones fundamentales de un diagrama de flujo es cómo se controla el flujo del programa. En muchos problemas es necesario realizar una u otra operación según se cumpla o no una condición. El control de flujo equivale a una instrucción condicional, en la que hacemos una u otra cosa en función de una condición. Su representación en los diagramas de flujo es la siguiente:



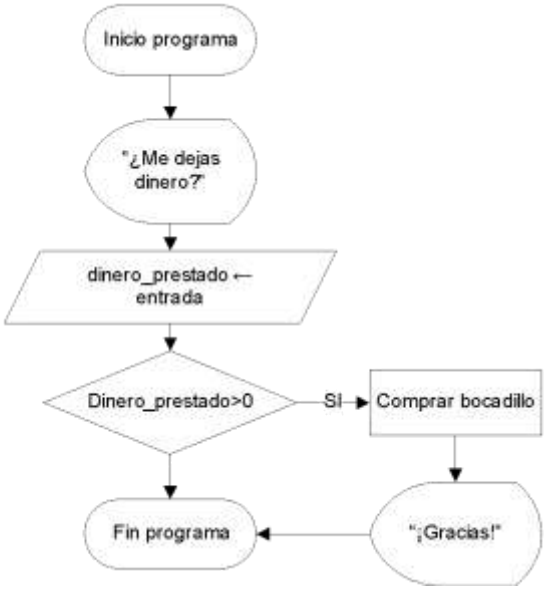
Por ejemplo, podemos presentarnos a la recuperación de una asignatura si hemos suspendido, pero no tendría sentido hacerlo si ya hemos aprobado. Veamos cómo se representaría esto en un diagrama de flujo:

Ejemplo: recuperación de una asignatura



Otra alternativa es que sólo hagamos algo si se da una condición determinada, y nada en el caso contrario. Por ejemplo, puedes pedirle dinero a un amigo y comprar un bocadillo si te deja el dinero. Se representaría de la siguiente manera:

Ejemplo: dinero para un refresco



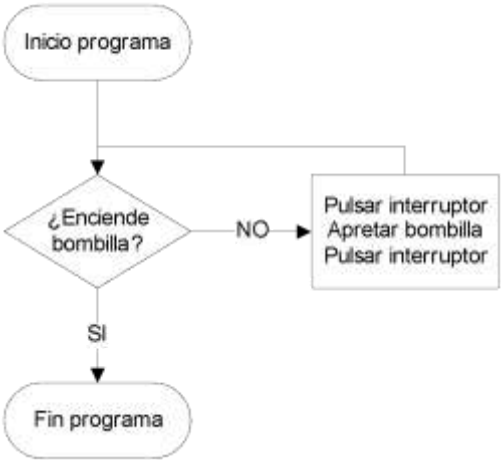
A continuación presentamos los operadores de comparación y lógicos más comunes que se usan en la evaluación de las condiciones en el control de flujo. Usaremos los operadores de Python:

Operador	Significado	Operador	Significado
<	menor que	<=	menor o igual que
>	mayor que	>=	mayor o igual que
==	igual que	!=	distinto de
not	negación lógica	and	Y
or	O		

Control de flujo: bucles de repetición

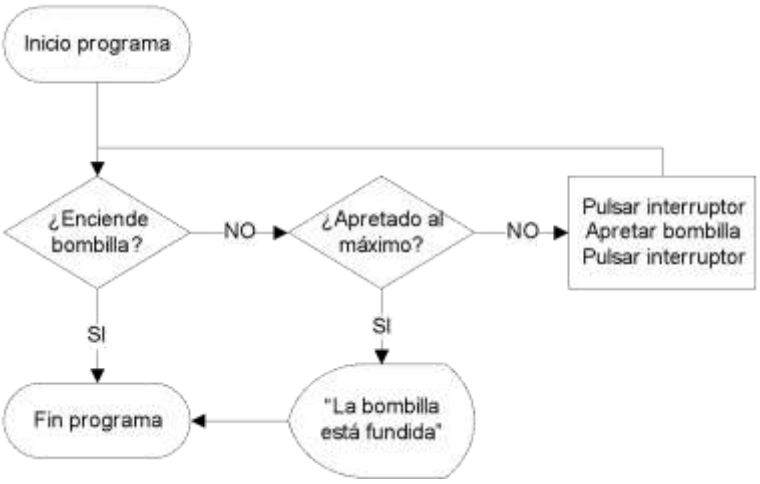
Frecuentemente, la operación de control de flujo puede devolver el flujo al punto de partida (tras haber realizado alguna operación), creando **bucles de repetición**. Un ejemplo de estos bucles puede ser colocar una bombilla, a la que damos vueltas hasta comprobar que se enciende con el interruptor, que se representaría como en el siguiente ejemplo:

Ejemplo: encendido de bombilla suelta



¿Qué pasaría si hemos tenido mala suerte y la bombilla está fundida en su caja? Nos pasaríamos la eternidad enroscando la bombilla, sin poder dejar nunca de hacerlo. Eso es lo que se conoce como un **bucle infinito,** porque las acciones que hacemos nunca nos permiten salir de él. Para solucionarlo en el ejemplo anterior, vamos a añadir un segundo elemento de control de flujo, que nos permita determinar si seguir apretando la bombilla servirá para algo.

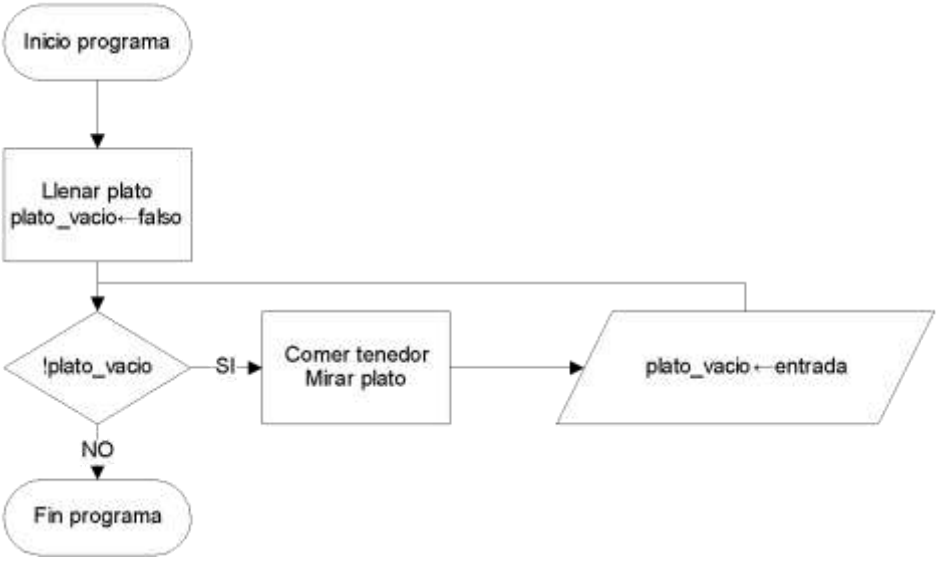
Ejemplo: encendido de bombilla sin posibilidad de bucle infinito



Normalmente se distinguen tres tipos de bucles de repetición: **mientras (*while*), hacer… hasta (*do…while*) y para (*for*)**.

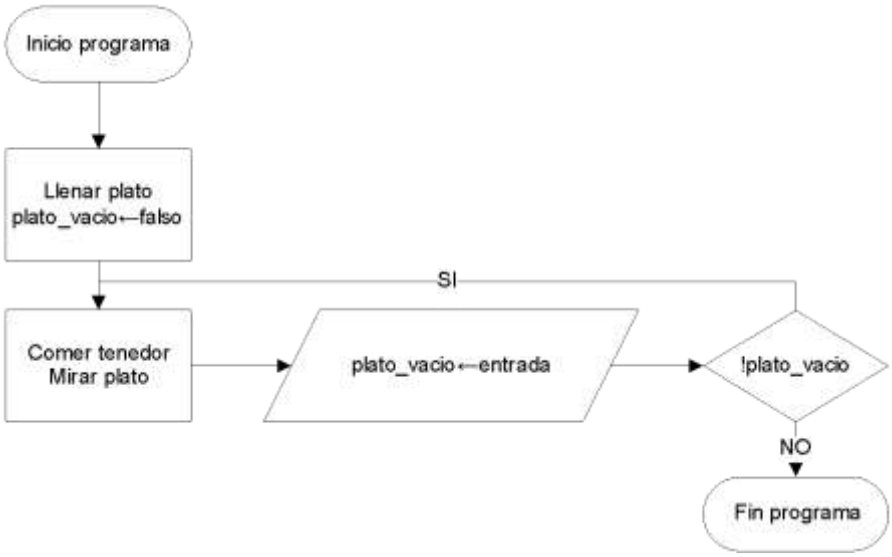
El bucle **while** se basa en comprobar una condición, hacer una serie de acciones si se cumple y devolver el flujo para comprobar otra vez la condición.

Ejemplo: bucle *while* para comer hasta que el plato está vacío



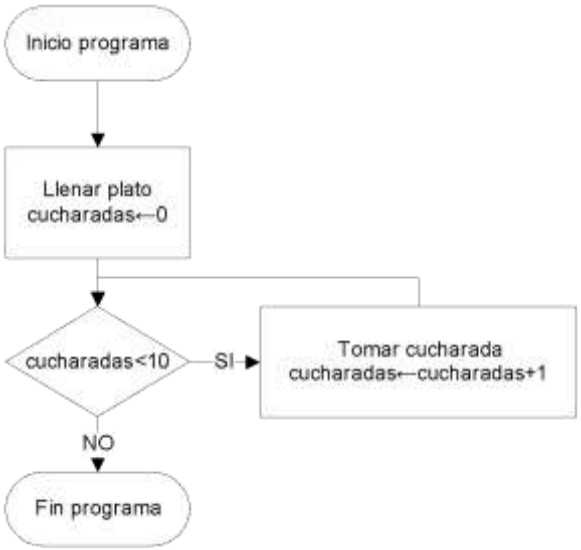
Una alternativa es empezar a comer (sin mirar el plato) y no parar mientras el plato no se vacía. La diferencia es que, en este caso, **siempre** comeremos del plato **al menos una** vez. Esto es lo que se conoce como el bucle **do… while**. En el diagrama es importante darse cuenta de que quien envía el flujo un número de instrucciones hacia atrás no es una instrucción o una operación de entrada, sino el operador de control de flujo (es decir, la flecha que retrocede instrucciones en este caso saldrá desde el rombo).

Ejemplo: bucle *do… while* para comer de un plato hasta que se vacía



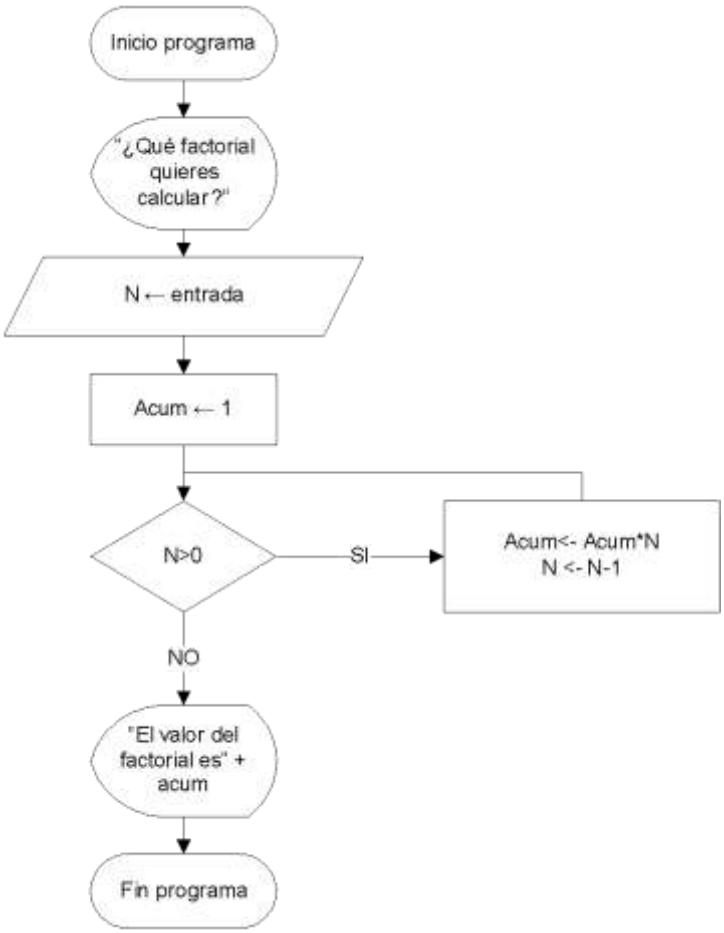
Planteemos ahora que sabemos cuántas cucharadas de sopa caben en el plato y que se va a tomar exactamente ese número de cucharadas. Eso se representa mediante un bucle **for**. Este tipo de bucle sirve para representar repeticiones donde **sabemos cuántas veces** se va a repetir la operación, y lo controlamos mediante un contador (variable contador), que se va modificando para hacer que se den ese número de repeticiones. Es importante darse cuenta de que el iterador suele necesitar empezar en un valor determinado, que se asigna de manera previa al control de flujo.

Ejemplo: bucle *for*



5 Ejemplo final: factorial de un número

Antes de pasar a proponer los ejercicios que debéis entregar, queremos practicar con un último ejemplo: calcular el factorial de un número que se pide al usuario por pantalla. El ejemplo puede resolverse de la siguiente manera:



6 Ejercicios a realizar

Ejercicio 1. Hacer un diagrama de flujo que describa los pasos para hacer una tortilla de patatas tal y como aparece en el Ejercicio 3 de la semana pasada.

Ejercicio 2. Hacer un diagrama de flujo que describa el algoritmo que compruebe si un número A es múltiplo de otro número B, usando solamente sumas, condicionales (comparaciones) y bucles. (Según el ejercicio de la semana pasada)

Ejercicio 3. Hacer un diagrama de flujo que permita escribir los 100 primeros pares.

Ejercicio 4. Hacer un diagrama de flujo para sumar los N primeros impares.

Ejercicio 5. Escribir el diagrama de flujo de un programa que pida ingresar la coordenada de un punto en el plano, es decir dos valores enteros x e y (deberá comprobar que ambos son distintos a cero). Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto. (1º Cuadrante si $x > 0$ Y $y > 0$, 2º Cuadrante: si $x < 0$ Y $y > 0$, etc.)

Ejercicio 6. De un operario se conoce su sueldo y los años de antigüedad. Se pide confeccionar el algoritmo de flujo de un programa que lea los datos de entrada y visualice:

> Si el sueldo es inferior a 1000 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20 %, mostrar el sueldo a pagar.
> Si el sueldo es inferior a 1000 pero su antigüedad es menor a 10 años, otorgarle un aumento de 5 %.
> Si el sueldo es mayor o igual a 1000 mostrar el sueldo en pantalla sin cambios.

Ejercicio 7. Dados dos números enteros positivos N y D, se dice que D es un divisor de N si el resto de dividir N entre D es 0. Se dice que un número N es perfecto si la suma de sus divisores (excluido el propio N) es N. Por ejemplo 28 es perfecto, pues sus divisores (excluido el 28) son: 1, 2, 4, 7 y 14 y su suma es 1+2+4+7+14=28. Hacer un diagrama de flujo que dado un número N nos diga si es o no perfecto.

Ejercicio 8. Un año es bisiesto si es múltiplo de 4, exceptuando los múltiplos de 100, que sólo son bisiestos cuando son múltiplos además de 400, por ejemplo el año 1900 no fue bisiesto, pero el año 2000 si lo será. Hacer un diagrama de flujo que dado un año nos diga si es o no bisiesto.

Ejercicio 9. Una línea de autobuses cobra un mínimo de 20 euros por persona y trayecto. Si el trayecto es mayor de 200 kilómetros, el billete tiene un recargo de 3 céntimos por kilómetro. Sin embargo, para trayectos de más de 400 kilómetros el billete tiene un descuento del 15 %. Por otro lado, para grupos de 3 o más personas el billete tiene un descuento del 10%. Con las consideraciones anteriores se pide confeccionar el algoritmo del flujo del programa.

7 Normas de entrega

Los ejercicios se deben **subir a Aula Global** hasta las 7:00 del lunes 28 de Septiembre de 2020. Se deberá subir un fichero comprimido **zip** que contenga los 9 archivos en formato **PDF**. El nombre del fichero será "s2-iniciales-del-alumno.zip" (por ejemplo Lucía Pérez Gómez subirá un archivo llamado p2-lpg.zip).

Nota: se recomienda usar www.draw.io para crear los diagramas, pero se pueden crear con cualquier herramienta.