David Bernal Navarrete – 1º DAM

1. Sí, podría aparecer. A la hora de asignar, el programa se dirige a la localización en memoria de la variable que esté a la izquierda de la asignación. No entra en detalle de lo que almacena dicha variable, solo se coloca en la dirección de memoria en la que esté. En la parte derecha de la asignación, el programa va a leer lo que esté en las direcciones de memoria de las variables que haya en ese lado, por lo que si se usa la misma variable en ambos lados, lo que está ocurriendo es que va a leer lo que hay en la dirección de memoria de dicha variable, y lo va a almacenar en la dirección de memoria de la misma variable, independientemente de lo que ya esté almacenado en ella.

Por ejemplo:

```
int var1 = 12;
int var2 = 2;
var1 = var1 + var2;
```

Lo que se está indicando al programa es lo siguiente:

-almacena el valor entero 12 en la variable entera var1.

-almacena el valor entero 2 en la variable entera var2.

-ve a la posición de memoria de var1; almacena en dicha posición la suma de los valores que estén dentro de las posiciones de memoria de var1 y de var2.

En una instrucción de asignación, en el lado izquierdo hay direcciones de memoria, y en el lado derecho hay valores.

2. Para ello usaremos la estructura if simple, y usaremos dos de ellas en sucesión (no anidadas). Cada if evaluará cada condición por separado, y la ejecución de cada tarea será independiente de la otra.

```
If(condicion1) {
        tarea1;
}
if(condicion2) {
        tarea2;
}
```

Es una solución simple, pero costosa en tiempo de ejecución, que permite evaluar dos condiciones por separado. Se puede realizar cada tarea por separado, pudiendo ser sólo una, ambas, o ninguna. Cada tarea depende sólo y únicamente de su condición.

3. La estructura switch usa resultados concretos de una expresión para cada caso. Si se usa una variable no discreta, en el caso de un ordenador que usa coma flotante, puede ser que oucrran errores en el cálculo de la expresión, que deriven en un resultado no discreto que puede diferir por muy poco del caso especificado (por ejemplo que el caso sea 1.0000001, y el resultado de la expresión, por errores de precisión sea 1.0000002), en cuyo caso, la estructura tan estricta del switch pierde precisión, y no es útil en estos casos.

4. La estructura MIENTRAS primero comprueba la expresión, y si es verdadera ejecuta el bloque de instrucciones que haya definido en dicha estructura; mientras que la estructura REPETIR ejecutará primero el bloque de instrucciones, y para volver a ejecutarlo comprobará una expresión.

Las desventajas de MIENTRAS frente a REPETIR dependen de la situación, puesto que en ambas se puede cambiar el valor de la expresión con una variable, y la única diferencia es que en REPETIR las instrucciones se ejecutan una vez antes de comprobar la expresión, aunque ésta sea falsa. Esta diferencia puede ser beneficiosa en ciertas circunstancias: por ejemplo, queremos generar un número aleatorio, y si es 10 terminamos. Para ello tendremos que calcular primero el número aleatorio, y luego lo tendremos que volver a calcular por cada vez que no sea 10. Esto resultaría en un código parecido a este (en C):

```
#include <generar_num_aleatorio>

int num;
do {
	num = generar_num_aleatorio(1, 100)
}
while (num != 10);
```
#ACLARACIONES: generar_num_aleatorio(a, b) es una función que genera un número pseudoaleatorio entre a y b, ambos incluidos.

5. Una variable interruptor, o flag, permite conducir el código por el camino que el programador escoja. Esto es, por ejemplo, si el programador quiere que se continúe la ejecución de una rama condicional o no, o cierto bucle se ejecute o no. Su utilidad es equivalente a la de un interruptor eléctrico, permitiendo el control de “apagar” o “encender” ciertas partes del código cuando uno elija. Un ejemplo sería el reto 1 de la UD02: un programa que pide ciertas condiciones para mostrar una cuenta atrás en pantalla, pero que si cualquiera de estas condiciones no se cumple, el mensaje no aparecerá en pantalla. Para controlar la aparición de la cuenta atrás en pantalla, se puede usar un interruptor o flag, inicializado a 1 (“sí se mostrará”), y que en el caso de que se cumpla cualquiera de las condiciones limitantes, se pondrá a 0 (“no se mostrará”).

```
Int flag = 1;

if(condicion1) {
	mensaje_de_error1;
	flag = 0;
} else {
	if (condicion2) {
		mensaje_de_error2;
		flag = 0;
	} else {
		if(condicion3) {
			mensaje_de_error3;
			flag = 0;
		} //Cierre del último if
	} //Cierre del segundo else
} //Cierre del primer else

if(flag == 1) {
	mostrar_cuenta_atras;
}
```

Esto resulta en que la cuenta atrás solo se mostrará si no se ha cumplido ninguna de las condiciones limitantes.

6. La instrucción continue; saltará todo el código que haya entre dicha instrucción y el final del bucle, efectivamente comenzando un nuevo ciclo del bucle. Por ejemplo, si se cumple una condición concreta, saltar el resto del código y volver al principio:

```
while(condicion1) {
	bloque_de_código1;
	if(condicion2) {
		bloque_de_código2;
		continue;
	}
	bloque_de_código3;
}
```

Si se cumple la condicion2 en el bucle while, se saltará el bloque_de_código3, y se volverá al inicio del bucle.

7. La estructura presenta un fallo en el if dentro del for. Dicho if carece de las llaves que indican su extensión ({}), por lo que daría un error de sintaxis.

Es un bucle for que por cada iteración aumentará el valor de la variable contador en 1. Cuando dicha variable llegue al valor 7, saldrá del bucle for, y mostraría en pantalla la línea “...procesando ...”.

Un bloque de código que resultaría en la misma función sería el siguiente:

```
int contador = 1;
while(contador % 7 != 0) {
        contador = contador + 1;
}
printf(“...procesando ...”);
```