Nombre y apellidos:

En la sesión anterior repasamos las estructuras de control IF, IF – ELSE. El objetivo de esta última sesión de repaso es abordar las estructuras de control repetitivas **DO…WHILE, WHILE** y **FOR**.

ESTRUCTURAS DE CONTROL REPETITIVAS:

WHILE: esta estructura permite la ejecución reiterada de una serie de instrucciones mientras se cumpla una determinada condición.

Estructura:

```
while (condición) {
        cuerpo-bucle
}
```

Traducción 1: dejar condición intacta y saltar condicionalmente sobre el salto incondicional al final del bucle.

```
etiqueta-bucle:
        evaluación condición utilizando después branch a etiqueta-cuerpo
        j etiqueta-fin
etiqueta-cuerpo:
        cuerpo-bucle
        j etiqueta-bucle
etiqueta-fin:
```

Traducción 2: complementar condición y saltar condicionalmente al final del bucle.

```
etiqueta-bucle:
        evaluación condición complementada utilizando después branch a etiqueta-fin
        cuerpo-bucle
        j etiqueta-bucle
etiqueta-fin:
```

DO...WHILE: casi igual que la estructura anterior, con la diferencia de que el cuerpo del bucle se ejecuta al menos una vez.

Estructura:

```
do {
        cuerpo-bucle
    } while (condición);
```

Traducción:

```
etiqueta-bucle:
        cuerpo-bucle
        evaluación condición saltando a etiqueta bucle
```

FOR: la estructura de control **for** se diferencia de la **while** en que el número de iteraciones es conocido de antemano. Así pues, es necesaria una variable que almacene el número de iteraciones que se van realizando para compararlo con el número de iteraciones deseado.

Estructura:

```
for (inicialización;condición;actualización) {
        cuerpo-bucle
}
```

Traducción: traducirlo previamente a un bucle while en pseudocódigo y después traducir dicho bucle while a ensamblador.

Paso 1.

```
inicialización
while (condición) {
        cuerpo-bucle
        actualización
}
```

Nombre y apellidos:

Ejercicio 1. Realice la traducción del siguiente programa, escrito en pseudocódigo, a ensamblador de MIPS. Cuando haya finalizado, intente minimizar el número de instrucciones ejecutadas sin alterar el sentido del programa.

```
do {
a = a + 3
} while (a < b*2);
```

Mapeo de registros:
a: $t0, b: $t1

Ejercicio 2.- Realice las traducciones del siguiente programa, escrito en pseudocódigo, a ensamblador de MIPS.

```
while (a <= c + 4) { a = a + 3 }
b = b + a
```

Mapeo de registros:
a: $t0, b: $t1, c: $t2

Ejercicio 3.- Realice la traducción del siguiente programa, escrito en pseudocódigo, a ensamblador de MIPS.

```
sum = 0
for (i = 0; i < n; i++) {
        sum = sum + i
}
```

Mapeo de registros:
n: $t0, i: $t1, sum: $t2

Debe entregar el código generado para los ejercicios anteriores en 3 ficheros diferentes, comprimidos en un único fichero .zip