

# Práctica 4

- Cuestión 8.

Escribe una función que multiplique por 60. Escribe el programa principal que lea una cantidad de minutos y devuelva por consola la cantidad en segundos.

```
# Cuestión 8.
# > Escribe una función que multiplique por 60. Escribe el programa principal que
#   lea una cantidad de minutos y devuelva por consola la cantidad en segundos.

# Recordatorio
# V = retorno
# A = entrada
#####

.text

jal readInt          # Llamamos a la función de leer un entero.

move $a0, $v0
jal x60              # Llamamos a la función x60 que multiplica por 60

move $a0, $v0
jal printInt         # Llamamos a la función imprimir un entero

jal exit             # Acaba el programa

#####
# Funciones
#####

# Función: imprime un entero
printInt: li $v0, 1   # Función imprim
syscall           # Escribe el valor de $a0

jr $ra             # Vuelve al programa principal

# Función multiplica por 60
x60: move $t0, $a0

sll $t1, $t0, 6     # Multiplicamos el valor por 64 y lo guardamos en $t1
sll $t2, $t0, 2     # Multiplicamos el valor por 4 y lo guardamos en $t2
sub $v0, $t1, $t2   # Restamos $t1 - $t2 y lo devolvemos a $v0

jr $ra

# Función lee un entero.
readInt: li $a0, '>' # Ponemos un caracter para que el usuario sepa que estamos esperando un numero de entrada.
li $v0, 11
syscall

li $v0, 5           # Pedimos un entero
syscall

jr $ra             # Vuelve al programa principal

# Sale del programa
exit: li $v0, 10
syscall
```

ssages	Run I/O
>2	120
-- program is finished running --	
Reset: reset completed.	
>1	60
-- program is finished running --	
Reset: reset completed.	
>3	180
-- program is finished running --	

- Cuestión 9.

Modifica el código de tal manera que ahora lo que lea sea una cantidad de hora y muestre por consola la cantidad de segundos.

```
# Cuestión 9.
# Modifica el código de tal manera que ahora lo que lea sea una cantidad de hora y
# muestre por consola la cantidad de segundos.

# Recordatorio
# V = retorno
# A = entrada
#####

.text

# Llamamos a la función de leer un entero.
jal readInt

# Llamamos a la función x60 2 veces para pasar de horas a segundos.
move $a0, $v0
jal x60

move $a0, $v0
jal x60

# Llamamos a la función imprimir un entero
move $a0, $v0
jal printInt

# Acaba el programa
jal exit

#####
# Funciones #
#####

# Función, imprime un entero
printInt: li $v0, 1 # Función imprim
syscall # Escribe el valor de $a0
```

```
Pages  Run / O
>1
3600
-- program is finished running --

Reset: reset completed.
] >2
7200
-- program is finished running --
```

- Ejercicio 11.

Escribe el código que lee el valor  $x$  y escribe por pantalla la solución de la ecuación:  $5x^2 + 2x + 3$ .

```
# Ejercicio 11.
# Escribe el código que lee el valor x y escribe por pantalla la solución de la
# ecuación:  $5x^2 + 2x + 3$ .

# Recordatorio
# V = retorno
# A = entrada
#####

.text

# Llamamos a la función de leer un entero.
jal readInt

# Llamamos a la función que hace la ecuación con el valor pasado por $a0
move $a0, $v0
jal ecuacion

# Llamamos a la función imprimir un entero
move $a0, $v0
jal printInt

# Acaba el programa
jal exit

#####
# Funciones
#####

# Función, imprime un entero
printInt: li $v0, 1      # Función imprim
syscall      # Escribe el valor de $a0

jr $ra      # Vuelve al programa principal

# Función multiplica por 60

ecuacion: move $t0, $a0
# ecuación:  $5x^2 + 2x + 3$ .

mult $t0, $t0      #  $x^2$ 
mflo $t0      #  $t0 = x^2$ 

li $t1, 5      #  $t1 = 5$ 
mult $t1, $t0      #  $5x^2$ 
mflo $t0      #  $t0 = 5x^2$ 

li $t1, 2      #  $t1 = 2$ 
mult $t1, $a0      #  $2x$ 
mflo $t1      #  $t1 = 2x$ 

addu $t0, $t0, $t1      #  $t0 = 5x^2 + 2x$ 
addiu $t0, $t0, 3      #  $t0 = 5x^2 + 2x + 3$ 

move $v0, $t0

jr $ra

# Función lee un entero.
readInt: li $a0, '>'      # Ponemos un caracter para que el usuario sepa que estamos esperando un numero de entrada.
li $v0, 11
syscall

li $v0, 5      # Pedimos un entero
syscall

jr $ra      # Vuelve al programa principal

# Sale del programa
exit: li $v0, 10
syscall
```

```
>1
10
-- program is finished running --

Reset: reset completed.

>2
27
-- program is finished running --

Reset: reset completed.

>5
138
-- program is finished running --
```

## Práctica 4+1

- Cuestión 9.

➤ Haz el código que lea dos enteros de la consola y escriba la suma y vuelva a comenzar si el resultado es distinto de 0.

Es pseudocódigo sería:  
(bucle do-while)

seguir: Leer el primer valor (A)  
Leer el segundo valor (B)  
Imprimir A+B  
Si (A+B) == 0 ir a acabar  
ir a seguir

acabar:

```
# CuestiÃ³n 9.
# Haz el cÃ³digo que lea dos enteros de la consola y escriba la suma y vuelva a
# comenzar si el resultado es distinto de 0. Es pseudocÃ³digo serÃ¡-a:

# (Bucle do-while)
#     seguir: Leer el primer valor (A)
#     Leer el segundo valor (B)
#     Imprimir A+B
#
#     Si (A+B) == 0
#         ir a acabar
#
#     ir a seguir
# acabar

.text
BucleWhile: jal readInt      # Lee un entero
move $s0, $v0               # nos los guardamos en $s0

jal readInt                  # Lee el segundo entero
move $s1, $v0               # nos los guardamos en $s1

move $a0, $s0                # Lo movemos para imprimir
jal printInt                 # Imprimimos

jal printPlus                # Iprimimos un '+' par que quede: A+B

move $a0, $s1                # Lo movemos para imprimir
jal printInt                 # Imprimimos

jal returncarriage           # Imprimimos un enter.

add $t0, $s0, $s1            # Sumamos ambos valores y guardamos el resultado en $t0

seq $t1, $t0, $zero          # Comprobamos si son iguales, devuelve 1 en $1 si se cumple

bne $t1, 1, BucleWhile      # Vuelve a iterar si $t1 != 1

j exit

#####
#     Funciones     #
#####

# Funcion, imprime un entero
printInt: li $v0, 1          # FunciÃ³n imprimir
syscall                      # Escribe el valor de $a0

jr $ra                      # Vuelve al programa principal

# Imprimimos un caracter +
printPlus: li $a0, '+'
li $v0, 11
syscall

jr $ra

# Funcion imprime un caracter de retron de carro
returncarriage: li $a0, '\n'
li $v0, 11
syscall
jr $ra

# Funcion lee un entero.
readInt: li $a0, '>'         # Ponemos un caracter para que el usuario sepa que estamos esperando un numero de entrada.
li $v0, 11
syscall

li $v0, 5                    # Pedimos un entero
syscall

jr $ra                      # Vuelve al programa principal

# Sale del programa
exit: li $v0, 10
syscall
```

```
>1
>5
1+5
>2
>3
2+3
>6
>-6
6+-6
-- program is finished running --
```

- Cuestión 10.

➤ Haz el código que lee de teclado dos valores positivos A y B en los que  $A < B$ . El programa tiene que escribir por consola los valores comprendidos entre ambos, incluyéndose a ellos mismos. Es decir, si  $A=3$  y  $B=6$ , escribe en la consola 3 4 5 6 (puedes escribir, por ejemplo, un salto de línea después de cada uno de los valores a mostrar).

```
# Cuestión 10.
# Haz el código que lee de teclado dos valores positivos A y B en los que  $A < B$ .
# El programa tiene que escribir por consola los valores comprendidos entre ambos,
# incluyéndose a ellos mismos.
# Es decir, si  $A=3$  y  $B=6$ , escribe en la consola: 3 4 5 6.
# (puedes escribir, por ejemplo, un salto de línea después de cada uno de los valores a mostrar).

jal readInt    # Lee un entero
move $s0, $v0  # nos los guardamos en $s1, inicio.

jal readInt    # Lee el segundo entero
move $s1, $v0  # nos los guardamos en $s2, condicion final del bucle
move $s2, $s0  # Contador

inicio_for: move $a0, $s2  # Mostramos el contador
jal printInt

jal printSpace  # Imprimimos un espacio entre los numeros

seq $t1, $s2, $s1 # Si $s2 es menor que $s1, $t1 = 1, en caso contrario, $t1 = 0

addi $s2, $s2, 1 # incremento del contador

beqz $t1, inicio_for # Si es igual a 0, reitera.

final_for:
j exit

#####
#      Funciones      #
#####

# Funcion que imprime un espacio
printSpace: li $a0, ' '
li $v0, 11
syscall

jr $ra

# Funcion, imprime un entero
printInt: li $v0, 1    # Función imprimir
syscall              # Escribe el valor de $a0

jr $ra              # Vuelve al programa principal
```

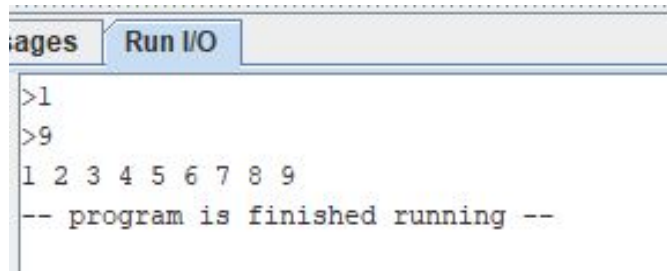


```
# Funcion lee un entero.
readInt: li $a0, '>'    # Ponemos un caracter para que el usuario sepa que estamos esperando un numero de entrada.
li $v0, 11
syscall

li $v0, 5    # Pedimos un entero
syscall

jr $ra      # Vuelve al programa principal

# Sale del programa
exit: li $v0, 10
syscall
```



```
>1
>9
1 2 3 4 5 6 7 8 9
-- program is finished running --
```



# Práctica 6

- Cuestión 7 (con 4 números).

➤ Escribe el código que lee cuatro enteros del teclado. Para ello deberás mostrar dos mensajes por la consola: uno que pida al usuario que introduzca el primer valor y tras haberlo leído que muestre otro solicitando el segundo valor. Los datos se almacenarán en posiciones consecutivas de la memoria, para lo cual debes haber reservado previamente espacio en el segmento de datos con la directiva `.space`. A continuación el programa leerá los valores guardados en la memoria y los mostrará en la pantalla.

```
# Question 7.

# Escribe el código que lee cuatro enteros del teclado. Para ello deberás mostrar dos
# mensajes por la consola: uno que pida al usuario que introduzca el primer valor
# y tras haberlo leído que muestre otro solicitando el segundo valor. Los datos se
# almacenarán en posiciones consecutivas de la memoria, para lo cual debes haber
# reservado previamente espacio en el segmento de datos con la directiva .space.
# A continuación el programa leerá los valores guardados en la memoria y los
# mostrará en la pantalla.

#####
#      DATOS      #
#####

.data          # Especificamos que queremos empezar a manipular datos para empezar desde el segmento de datos.
A: .space 4 # Numero 1.
B: .space 4 # Numero 2.
C: .space 4 # Numero 3.
D: .space 4 # Numero 4.

GREETER: .ascii "Dime un numero\n" # String para pedir un numero

#####
#      INSTRUCCIONES      #
#####

.text          # Especificamos que queremos empezar a manipular instrucciones para empezar desde el segmento de instrucciones.

# Pedimos los numeros y los guardamos en memoria.
jal setLoopStarters # Setea el contador y el puntero a su valor original para volver a reutilizarlo.

BucleWhilePedir: jal readInt # Pedimos el entero
move $t2, $v0           # Movemos el entero a nuestro registro
sw $t2, 0($t1)          # Guardamos el valor del registro a la memoria

jal continueLoops      # Funcion que se encarga de continuar el bucle y la iteracion sobre el puntero.

bne $t7, 1, BuCLEWhilePedir # Vuelve a iterar si $t7 != 1

# Mostramos los numeros desde la memoria
jal setLoopStarters    # Setea el contador y el puntero a su valor original para volver a reutilizarlo.

BucleWhileMostrar: lw $a0, 0($t1) # Cargamos el valor del byte a $a para mostrarlo.

jal printInt           # Imprimimos el numero
jal printSpace         # Imprimimos un espacio
```

```
jal continueLoops    # Funcion que se encarga de continuar el bucle y la iteracion sobre el puntero.

bne $t7, 1, BucleWhileMostrar    # Vuelve a iterar si $t7 != 1

# Salimos el programa
jal exit

#####
#      Funciones      #
#####

# Esta función contiene las partes compartidas de ambos bucles, la logica para decrementar el contador,
# avanzar a la siguiente posición de memoria (siguiente numero)
continueLoops: addi $t1, $t1, 4    # Cambiamos el valor del puntero para acceder al siguiente byte en la siguiente iteración

seq $t7, $t0, $zero    # Si el contador es 0, guarda 1 en $t7, 0 si no es igual a 0.

subi $t0, $t0, 1    # Decrementamos el contador

jr $ra    # Vuelve al programa principal

# Setteamos los contadores y la primera posición de memoria
setLoopStarters: li $t0, 3    # Cantidad de veces que queremos pedir/mostrar un número contado desde 0
la $t1, A    # $t0 = Dir Mem A

jr $ra    # Vuelve al programa principal

# Funcion lee un entero mostrando la petición.
readInt: la $a0, GREETER
li $v0, 4
syscall

li $v0, 5    # Pedimos un entero
syscall

jr $ra    # Vuelve al programa principal

# Funcion que imprime un espacio
printSpace: li $a0, ' '
li $v0, 11
syscall
```

```
jr $ra

# Funcion, imprime un entero
printInt: li $v0, 1    # Función imprimir
syscall    # Escribe el valor de $a0

jr $ra    # Vuelve al programa principal

# Sale del programa
exit: li $v0, 10
syscall
```

```
Dime un numero
5
Dime un numero
89
Dime un numero
723
Dime un numero
5214321
5 89 723 5214321
-- program is finished running --
```

- Cuestión 8.

➤ Modifica el programa de la cuestión 7 para que muestre en la pantalla los datos guardados en la memoria ordenados de menor a mayor valor.

```
# Question 8.

# Modifica el programa de la cuestión 7 para que muestre en la pantalla los datos
# guardados en la memoria ordenados de menor a mayor valor

#####
#      DATOS      #
#####

.data          # Especificamos que queremos empezar a manipular datos para empezar desde el segmento de datos.
A: .space 4 # Numero 1.
B: .space 4 # Numero 2.
C: .space 4 # Numero 3.
D: .space 4 # Numero 4.

GREETER: .asciiz "Dime un numero\n" # String para pedir un numero

#####
#      INSTRUCCIONES      #
#####

.text          # Especificamos que queremos empezar a manipular instrucciones para empezar desde el segmento de instrucciones.
# Pedimos los numeros y los guardamos en memoria.
jal setLoopStarters # Setea el contador y el puntero a su valor original para volver a reutilizarlo.

BucleWhilePedir: jal readInt # Pedimos el entero
move $t2, $v0          # Movemos el entero a nuestro registro
sw $t2, 0($t1)         # Guardamos el valor del registro a la memoria

jal continueLoops      # Funcion que se encarga de continuar el bucle y la iteracion sobre el puntero.

bne $t7, 1, BuclWhilePedir # Vuelve a iterar si $t7 != 1

# Ordena los numeros de mayor a menor en memoria.
la $s1, A # $s1 = Dir Mem A
la $s2, B # $s2 = Dir Mem B
la $s3, C # $s3 = Dir Mem C
la $s4, D # $s4 = Dir Mem D

lw $t1, 0($s1) # $t1 = Val Mem A
lw $t2, 0($s2) # $t2 = Val Mem B
lw $t4, 0($s4) # $t3 = Val Mem C
lw $t3, 0($s3) # $t4 = Val Mem D

# Dios, dime que hay una manera mejor de hacer esto
# Comparamos A y B, si A > B, intercambiamos, si no, pasamos a la siguiente comprobación
cmpAB:
blt $t1, $t2, cmpCD
xor $t1, $t1, $t2 # ($t1) = xor $t1 $t2
xor $t2, $t1, $t2 # $t2 = xor ($t1) $t2
xor $t1, $t1, $t2 # ($t1) = xor ($t1) ($t2)

# Comparamos C y D, si C > D, intercambiamos, si no, pasamos a la siguiente comprobación
cmpCD:
blt $t3, $t4, cmpAC
xor $t3, $t3, $t4 # ($t3) = xor $t3 $t4
xor $t4, $t3, $t4 # $t4 = xor ($t3) $t4
xor $t3, $t3, $t4 # ($t3) = xor ($t3) ($t4)

# Comparamos A y C, si A > C, intercambiamos, si no, pasamos a la siguiente comprobación
cmpAC:
blt $t1, $t3, cmpBD
xor $t1, $t1, $t3 # ($t1) = xor $t1 $t3
xor $t3, $t1, $t3 # $t3 = xor ($t1) $t3
xor $t1, $t1, $t3 # ($t1) = xor ($t1) ($t3)

# Comparamos B y D, si B > D, intercambiamos, si no, pasamos a la siguiente comprobación
cmpBD:
blt $t2, $t4, cmpBC
xor $t4, $t2, $t4 # ($t4) = xor $t2 $t4
xor $t2, $t2, $t4 # $t2 = xor ($t2) $t4
xor $t4, $t2, $t4 # ($t4) = xor ($t2) ($t4)
```

```
# Comparamos B y C, si B > C, intercambiamos, si no, pasamos a la siguiente comprobación
cmpBc:
blt $t2, $t3, finishSorting
xor $t3, $t3, $t2 # ($t3) = xor $t3 $t2
xor $t2, $t3, $t2 # $t2 = xor ($t3) $t2
xor $t3, $t3, $t2 # ($t3) = xor ($t3) ($t2)

# Acabamos guardando en memoria los valores ordenados.
finishSorting:
sw $t1, 0($s1) # Val Mem A = $t1
sw $t2, 0($s2) # Val Mem B = $t2
sw $t4, 0($s4) # Val Mem C = $t3
sw $t3, 0($s3) # Val Mem D = $t4

# Mostramos los numeros desde la memoria
jal setLoopStarters # Setea el contador y el puntero a su valor original para volver a reutilizarlo.

BucleWhileMostrar: lw $a0, 0($t1) # Cargamos el valor del byte a $a para mostrarlo.

jal printInt # Imprimimos el numero
jal printSpace # Imprimimos un espacio

jal continueLoops # Funcion que se encarga de continuar el bucle y la iteracion sobre el puntero.

bne $t7, 1, BucleWhileMostrar # Vuelve a iterar si $t7 != 1

# Salimos el programa
jal exit

#####
# Funciones #
#####

# Esta función contiene las partes compartidas de ambos bucles, la logica para decrementar el contador,
# avanzar a la siguiente posición de memoria (siguiente numero)
continueLoops: addi $t1, $t1, 4 # Cambiamos el valor del puntero para acceder al siguiente byte en la siguiente iteración

seq $t7, $t0, $zero # Si el contador es 0, guarda 1 en $t7, 0 si no es igual a 0.

subi $t0, $t0, 1 # Decrementamos el contador

jr $ra # Vuelve al programa principal

# Setteamos los contadores y la primera posición de memoria
setLoopStarters: li $t0, 3 # Cantidad de veces que queremos pedir/mostrar un número contado desde 0
la $t1, A # $t0 = Dir Mem A

jr $ra # Vuelve al programa principal

# Funcion lee un entero mostrando la petición.
readInt: la $a0, GREETER
li $v0, 4
syscall

li $v0, 5 # Pedimos un entero
syscall

jr $ra # Vuelve al programa principal
```



```
# Funcion que imprime un espacio
printSpace: li $a0, ' '
li $v0, 11
syscall

jr $ra

# Funcion, imprime un entero
printInt: li $v0, 1      # Función imprimir
syscall                # Escribe el valor de $a0

jr $ra                # Vuelve al programa principal

# Sale del programa
exit: li $v0, 10
syscall
```

```
Dime un numero
5
Dime un numero
4
Dime un numero
3
Dime un numero
2
2 3 4 5
-- program is finished running --
```

```
Dime un numero
1
Dime un numero
2
Dime un numero
3
Dime un numero
4
1 2 3 4
-- program is finished running --
```

```
Dime un numero
1
Dime un numero
7
Dime un numero
2
Dime un numero
4
1 2 4 7
-- program is finished running --
```

```
Dime un numero
7
Dime un numero
6
Dime un numero
6
Dime un numero
7
6 6 7 7
-- program is finished running --
```