Intérprete de un lenguaje para SWIFT
(Examen septiembre 2020)

Swift es un lenguaje de programación de propósito general que se puede usar en dispositivos móviles, de escritorio, servidores, etc. En su diseño se ha tenido en cuenta la seguridad, el rendimiento y la programación interactiva combinando lo mejor de los nuevos avances. Se sitúa en el entorno de Apple, pero recibe contribuciones de la comunidad de código abierto ("open source").

Se puede acceder a un completo manual del lenguaje de programación Swift en el siguiente enlace:
https://swift.org/documentation

Aunque los fundamentos que aparecen en este ejercicio se pueden encontrar en:
https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html
https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html
https://docs.swift.org/swift-book/LanguageGuide/BasicOperators.html
https://docs.swift.org/swift-book/LanguageGuide/ControlFlow.html

También hay disponible un compilador online de Swift, donde podrá probar el resultado de cualquier fichero escrito en el lenguaje de programación Swift:
http://online.swiftplayground.run

Se desea **implementar mediante Java, JFlex y Cup un intérprete del lenguaje de programación Swift** al que llamaremos **SwiftPL** pero limitando y variando ligeramente sus funciones a un núcleo básico.

Aspectos léxicos

El cuerpo de un programa en SwiftPL se compone de una o más sentencias, cada una de ellas escrita sin partirse en diferentes líneas. Cada línea puede acabar **opcionalmente** con un punto y coma ';' . En una misma línea podría existir más de una sentencia y se separarían mediante puntos y coma. El uso de espacios o tabuladores no tiene efecto (el sangrado de las líneas tampoco). No se permiten sentencias vacías terminadas en punto y coma.

Ejemplos de sentencias válidas en SwiftPL	Ejemplos de sentencias no válidas en SwiftPL

`2 + 3`
`2 + 3;`
`2 + 3; 4 + 5`
`2 + 3; 4 + 5;`
	`2 +`
	`2 + 3  4 +5`

NOTA: en Swift algunas sentencias podrían ocupar más de una línea, pero en este subconjunto del lenguaje no tendremos en cuenta esa opción.

Para mostrar cualquier resultado por pantalla existe el comando `print` que sirve para imprimir. Por ejemplo, la sentencia para imprimir por pantalla el texto "Hola Mundo" sería `print("Hello World")`. Esta sentencia incluye el salto de línea después de escribir lo que recibe como parámetro.

Cada sentencia que se ejecuta devuelve un valor. Ese valor NO se imprime automáticamente por pantalla y debe ser asignado a una variable o a una constante si se quiere utilizar posteriormente. Por ejemplo, las sentencias anteriores (2+3; 4+5) se ejecutarían, pero no tendría ninguna salida por pantalla. Tampoco tendrían ningún efecto posterior porque el resultado no es almacenado.

Si se encuentra una sentencia que contenga un error, se generará como salida la palabra `error` y se detendrá la ejecución.

Los comentarios son parecidos a los de C. Se usa la doble barra (//) para hacer un comentario en línea. También tiene comentarios de bloque, usando /* para iniciar el comentario y */ para terminarlo. La diferencia es que se usa anidamiento de bloques de comentarios, para facilitar la acción de comentar bloques de código que ya pudiesen contener comentarios.

Sentencias en SwiftPL	Salida por pantalla

`1+2;`
`print(1 + 3);`
`1 + 4`
`print(1 + 5)`
	`4`
	`6`

`1+2; print(1 + 3);` `1 + 4; print(1 + 5)`	`4` `6`

`1+2; print(1 + 3);`
 	
`1 + 4; print(1 + 5);`
	`4`
	`6`

`1+2; print(1 + 3);`
`;`
`1 + 4; print(1 + 5);`
	`4`
	`error (sobra un ;)`

`1 + 2`
`print (2+2)`
`print (3+2);`
`print (4+2;);`
`print (5+2);`
	`4`
	`5`
	`error (sobra un ;)`

`print (1+2); // comentario en linea`
`/* print (1+3); bloque comentado */`
`print (1+4); // comentario en linea`
	`3`
	`5`

`print (1+2); // comentario en linea`
`print (1+3); // comentario en linea /* sigue`
`/* print (1+4); bloque`
`comentado */`
 	
`/*`
`print (1+5);`
`/* print (1+6); bloque`
`comentado */`
`print (1+7); // comentario en linea */`
`print (1+8); // comentario en linea`
	`3`
	`4`
	`9`

`print (1+2); // comentario en linea`
`/*`
`print (1+5);`
`/* print (1+6); bloque`
`comentado */`
`print (1+7); // comentario en linea */ sigue`
`print (1+8); // comentario en linea`
	`3`
	`error`
	`(porque en la línea 5 se acaba el comentario y continua con texto no legal)`

Los nombres para las variables y constantes utilizan caracteres alfanuméricos y el guión bajo ('_'). El nombre debe empezar por una letra. Este lenguaje es sensible a mayúsculas y minúsculas, por lo que si dos nombres usan diferentes letras (aunque sean igual en mayúscula y minúscula), serán diferentes. Para definir constantes se usa la palabra reservada `let` y para las variables, la palabra reservada `var`.

Para la asignación se usa el signo '=' para hacer que el valor de la sentencia a la derecha del signo sea asignado en la variable o constante a la izquierda del signo. Tiene un comportamiento especial cuando se usa en la inicialización de variables y constantes (ver más adelante).

Sentencias en SwiftPL	Salida por pantalla
`var a = 1;` `print (a)`	`1`
`let a = 1` `print (a+2);`	`3`
`var a = 1; let b = a+1;` `print (a+b)`	`3`

`var a = 1; print(a);`
`let b = A+1;`
`print (a+b)`
	`1`
	`error (variable A no declarada)`

`var a = 2; print(a);`
`let 1_mas_a = 1 + a;`
`print (a+b)`
	`2`
	`error (var. no empieza por letra)`

`var a = 2`
`let b = a+1;`
`a=a*3`
`print (a+b)`
	`9`

`a = 1; print(a)`
`b = a+1;`
`print (a+b)`
	`error`
	`/* por a = 1; y b = a+1; porque hay que usar let o var*/`

`let b = a+1;`
`var a = 1;`
`print (a+b)`
	`error`
	`/* por let b = a+1; porque a está sin definir */`

Existen múltiples tipos de datos, pero usaremos únicamente los tipos numéricos entero y real (`Int` y `Double`) y el tipo booleano (`Bool`). Es posible definir varias variables o constantes del mismo tipo en una misma sentencia sin tener que inicializarlas. En el caso de las constantes sólo se podrán inicializar una única vez (en la declaración de la variable o después), mientras que las variables podrán cambiar su valor tantas veces como se desee. En el caso de declararlas sin inicializarlas (variables o constantes), es preciso indicar de forma explícita el tipo de datos que usará, para lo que se indica usando el signo de los dos puntos (':') seguido del nombre del tipo.

En la declaración de variables o constantes cuando se inicialicen con un valor, no es necesario definir el tipo de dato, porque se deducirá del valor asignado. En ese caso, en la declaración con inicialización (sin explicitar el tipo) debe existir, al menos, un espacio antes y después del signo '=' porque en caso contrario se trataría de un error. No se permite usar variables que únicamente hayan sido declaradas

En caso de inicializar el valor, también se podrá hacer explícito usando el signo de los dos puntos seguido del tipo después del nombre de la variable y antes de la asignación.

Sentencias en SwiftPL	Salida por pantalla

`var implicitoInt = 1`
`let implicitoDouble = 2.0`
`let explicitoDouble:Double = 3`
`print(implicitoInt)`
`print(implicitoDouble)`
`print(explicitoDouble)`
	`1`
	`2.0`
	`3.0`

`let implicitoDouble = 2.0`
`let explicitoDouble: Double = 3`
`print(implicitoDouble)`
`print(explicitoDouble / implicitoDouble)`
	`2.0`
	`1.5`

`var a= 1;`
`let b =2;`
`print (a+b)`
	`error (hay que dejar, al menos, un espacio antes y después del igual)`

`var a = 2`
`let b : Int;`
`b=a*3`
`print (a+b)`
	`8`

`var a,b: Int`
`a=1`
`b=a*3`
`a=a*2`
`print (a+b)`
	`5`

`var a,b: Int`
`let b : Int;`
`a=4`
`b=a*3`
`print (a+b)`
	`error (no se puede redeclarar una variable o constante)`

`var a = 2`
`let b = a+1;`
`b=a*3`
`print (a+b)`
	`error (por b=a*3 porque no se puede cambiar una constante)`

`var x, y, z: Double` `print (x)`	`error (x sin inicializar)`

Consideraremos los operadores aritméticos (para números) de suma, resta, multiplicación, división (+ , - , * , /). No se pueden operar variables de distinto tipo, para ello habría que hacer un casting explícito (que no se pide en este ejercicio).

Consideraremos los operadores relacionales (>, >=, <, <=, ==, !=, !, && y ||) para operaciones lógicas cuyas palabras reservadas para el valor de “verdadero” y “falso” son, respectivamente, `true` y `false`.

Se aplican los mismos criterios de prioridad y asociatividad que en C o Java, permitiendo el uso de paréntesis.

Sentencias en SwiftPL	Salida por pantalla

`var c = 1.0`
`var d = 2.3`
`var e = 3.4 + 5.6`
`var f =  c * (d + e)`
`var g = e * d + e`
`print(e)`
`print(f)`
`print(g)`
	`9.0`
	`11.3`
	`29.7`

`let implicitoInt = 1`
`let implicitoDouble = 2.0`
`print(implicitoDouble)`
`print(implicitoDouble /  implicitoInt )`
	`2.0`
	`error (no se puede operar una variable Int con una Double)`

`let x = 3 * -4`
`let y = 3 - *4`
`print(x)`
	`error (el operador * no es unario)`

`let a = false`
`print(a)`
`print(!a)`
	`false`
	`true`

`var a = true`
`print(a)`
`print(a || false)`
	`true`
	`true`

`let a = true`
`print(a)`
`var b = 3 < 2`
`print(a && b)`
	`true`
	`false`

`let a = true`
`print(a)`
`var b = 3 < 2 < 1`
`print(a && b)`
	`true`
	`error (comparación ilegal)`

`var x, y, z: Double`
`x = 2; y = 4; z = 8`
`var a: Bool ;`
`a = (z / y) == x`
`print (a)`
	`true`

Arrays

Se pueden crear arrays fácilmente usando los signos de los corchetes. Los elementos se separan por comas y después del último elemento puede quedar una coma (aunque no es necesario). Todos los datos tienen que ser el mismo tipo y, como la declaración de variables unarias, no precisa de una declaración explícita, aunque también lo admite. El acceso a un elemento del array se hace indicando la posición del elemento en el array entre corchetes, siendo 0 el número de la primera posición. En esta versión del lenguaje Swift, restringiremos la definición de arrays a aquellos que se definen en la propia declaración (es fácil crearlos desde cero o añadir o quitar elementos, pero no se tendrán en cuenta esas características en este ejercicio).

Sentencias válidas en SwiftPL	Salida por pantalla
`var numbers = [3,9,27]` `print (numbers[0])`	`3`
`var numbers = [3,9,27]` `print (numbers)`	`[3, 9, 27]`
`var numbers = [3,9,27]`	`[3, 9, 18]`

numbers[2] = 18 print (numbers)	
var numbers = [3,9,27] print (numbers) numbers[3] = 18 print (numbers)	[3, 9, 27] error (fuera de rango)
var numbers = [3,9,81] numbers[0] = numbers[2] / numbers[1] print (numbers)	[9, 9, 81]
var numbers: [Double] = [3,9,81] print (numbers)	[3.0, 9.0, 81.0]
var n = [3,9,27] let ordenado = (n[0]<n[1])&&(n[1]<n[2]) print (ordenado)	true
var reales: [Double] = [3,9,81] var enteros = [5,7,11] print (reales[0] * enteros[0])	error (no se pueden operar variables de distinto tipo)

Sentencias de control: FOR

De entre las sentencias de control disponibles se pide implementar la del bucle `for`, que tiene la siguiente sintaxis

```
for name in sequence { statement1 }
```

donde `sequence` es una secuencia de elementos (un array o un rango de números) y `name` es el nombre de la variable que se usará para recorrer los valores de la secuencia. Esa variable será declarada de forma implícita según el tipo de la secuencia y no existirá fuera del ámbito del bucle `for`. Si no se necesita una variable que vaya tomando valores, se puede usar el signo de guión bajo ('_').

Un rango se puede definir como cerrado usando la siguiente sintaxis: `valor_menor...valor_mayor` donde se pasa por todos los números enteros entre `valor_menor` y `valor_mayor` incluyendo ambos extremos.

Sentencias en SwiftPL	Salida por pantalla

var enteros = [5,7,11]
for numero in enteros
{
  print (numero)
}
	5
	7
	11

var enteros = [5,7,11]
for numero in enteros
{
  print (numero)
}
print (numero)
	5
	7
	11
	error (la variable numero no está fuera del ámbito del for)

var enteros = [5,7,11]
for index in 0...2 {
  print(enteros[index])
}
	5
	7
	11

let base, exponente: Int base = 2 exponente = 4 var resultado = 1	16

`for _ in 1...exponente {`
`  resultado = resultado * base`
`}`
`print(resultado)`	

`var a = [2,4,6]`
`var b = [2,3]`
`for i in a {`
`  for j in b{`
`  print(i+j)`
`  }`
`}`
	4
	5
	6
	7
	8
	9

Implementación

Se pide implementar un intérprete que lea un programa escrito en **SwiftPL** de un fichero de texto con extensión .SwiftPL y que dé lugar a la realización de las distintas acciones que se describen mediante ejemplos en el ANEXO y se dirigen a la salida estándar, por ejemplo:

```
> java SwiftPL a16.SwiftPL
```

Como resultado de este ejercicio se entregará un fichero comprimido SwiftPL.zip que contenga al menos los ficheros **SwiftPL.jflex**, **SwiftPL.cup** y **SwiftPL.java** más todos aquellos ficheros .java que sean necesarios para la compilación del intérprete mediante la secuencia de instrucciones:

```
> cup SwiftPL.cup
> jflex SwiftPL.jflex
> javac *.java
```

ANEXO

Construir un intérprete que permita ejecutar los siguientes tipos de sentencias:

Es necesario detectar los errores y advertirlos, pero no se exigen los detalles que se dan en el ejemplo.

Entrada	Descripción	Resultado de la ejecución	EJEMPLO
1+2; print(1 + 3); 1 + 4 print(1 + 5)	Fichero con elementos básicos, comentarios, punto y coma, espacios, sangrado, nombres de variables, asignaciones, errores, etc.	4 6	0_base_01.SwiftPL
1+2; print(1 + 3); 1 + 4; print(1 + 5)		4 6	0_base_02.SwiftPL
1+2; print(1 + 3); 1 + 4; print(1 + 5);		4 6	0_base_03.SwiftPL
1+2; print(1 + 3); ; 1 + 4; print(1 + 5);		4 error (sobra un ;)	0_base_04.SwiftPL
1 + 2 print (2+2) print (3+2); print (4+2;); print (5+2);		4 5 error (sobra un ;)	0_base_05.SwiftPL

print (1+2); // comentario en linea
/* print (1+3); bloque comentado */
print (1+4); // comentario en linea
		3
		5
			0_base_06.SwiftPL

print (1+2); // comentario en linea
print (1+3); // comentario en linea /* sigue
/* print (1+4); bloque
comentado */
 	
/*
print (1+5);
/* print (1+6); bloque
comentado */
print (1+7); // comentario en linea */
print (1+8); // comentario en linea
		3
		4
		9
			0_base_07.SwiftPL

print (1+2); // comentario en linea
/*
print (1+5);
/* print (1+6); bloque
comentado */
print (1+7); // comentario en linea */ sigue
print (1+8); // comentario en linea
		3
		error (porque en la línea 5 se acaba el comentario y continua con texto no legal)
			0_base_08.SwiftPL

var a = 1; print (a)		1	0_base_09.SwiftPL
let a = 1 print (a+2);		3	0_base_10.SwiftPL
var a = 1; let b = a+1; print (a+b)		3	0_base_11.SwiftPL

var a = 1; print(a);
let b = A+1;
print (a+b)
		1
		error (variable A no declarada)
			0_base_12.SwiftPL

var a = 2; print(a);
let 1_mas_a = 1 + a;
print (a+b)
		2
		error (var. no empieza por letra)
			0_base_13.SwiftPL

var a = 2 let b = a+1; a=a*3 print (a+b)		9	0_base_14.SwiftPL
a = 1; print(a) b = a+1; print (a+b)		error /* por a = 1; y b = a+1; porque hay que usar let o var*/	0_base_15.SwiftPL

let b = a+1;
var a = 1;
print (a+b)
		error
		/* por let b = a+1; porque a está sin definir */
			0_base_16.SwiftPL

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

`var implicitoInt = 1`
`let implicitoDouble = 2.0`
`let explicitoDouble:Double = 3`
`print(implicitoInt)`
`print(implicitoDouble)`
`print(explicitoDouble)`
	Declaración de variables y tipos de datos	1
		2.0
		3.0
			1_var_01.SwiftPL

`let implicitoDouble = 2.0`
`let explicitoDouble: Double = 3`
`print(implicitoDouble)`
`print(explicitoDouble / implicitoDouble)`
		2.0
		1.5
			1_var_02.SwiftPL

`var a= 1;`
`let b =2;`
`print (a+b)`
		error (hay que dejar, al menos, un espacio antes y después del igual)	1_var_03.SwiftPL

`var a = 2`
`let b : Int;`
`b=a*3`
`print (a+b)`
		8	1_var_04.SwiftPL

`var a,b: Int`
`a=1`
`b=a*3`
`a=a*2`
`print (a+b)`
		5	1_var_05.SwiftPL

`var a,b: Int`
`let b : Int;`
`a=4`
`b=a*3`
`print (a+b)`
		error (no se puede redeclarar una variable o constante)	1_var_06.SwiftPL

`var a = 2`
`let b = a+1;`
`b=a*3`
`print (a+b)`
		error (por b=a*3 porque no se puede cambiar una constante)	1_var_07.SwiftPL

`var x, y, z: Double` `print (x)`		error (x sin inicializar)	1_var_08.SwiftPL

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

`var c = 1.0`
`var d = 2.3`
`var e = 3.4 + 5.6`
`var f =  c * (d + e)`
`var g = e * d + e`
`print(e)`
`print(f)`
`print(g)`
	Operaciones aritméticas y booleanas	9.0
		11.3
		29.7
			2_oper_01.SwiftPL

`let implicitoInt = 1`
`let implicitoDouble = 2.0`
`print(implicitoDouble)`
`print(implicitoDouble /  implicitoInt )`
		2.0
		error (no se puede operar una variable Int con una Double)
			2_oper_02.SwiftPL

`let x = 3 * -4`
`let y = 3 - *4`
`print(x)`
		error (el operador * no es unario)	2_oper_03.SwiftPL

`let a = false`
`print(a)`
`print(!a)`
		false
		true
			2_oper_04.SwiftPL

`var a = true`
`print(a)`
`print(a || false)`
		true
		true
			2_oper_05.SwiftPL

`let a = true`
`print(a)`
`var b = 3 < 2`
`print(a && b)`
		true
		false
			2_oper_06.SwiftPL

`let a = true`
`print(a)`
`var b = 3 < 2 < 1`
`print(a && b)`
		true
		error (comparación ilegal)
			2_oper_07.SwiftPL

`var x, y, z: Double`
`x = 2; y = 4; z = 8`
`var a: Bool ;`
`a = (z / y) == x`
`print (a)`
		true	2_oper_08.SwiftPL

Entrada	Descripción	Resultado de la ejecución	EJEMPLO
var numbers = [3,9,27] print (numbers[0])	Definición y uso de arrays	3	3_vect_01.SwiftPL
var numbers = [3,9,27] print (numbers)		[3, 9, 27]	3_vect_02.SwiftPL
var numbers = [3,9,27] numbers[2] = 18 print (numbers)		[3, 9, 18]	3_vect_03.SwiftPL
var numbers = [3,9,27] print (numbers) numbers[3] = 18 print (numbers)		[3, 9, 27] error (fuera de rango)	3_vect_04.SwiftPL
var numbers = [3,9,81] numbers[0] = numbers[2] / numbers[1] print (numbers)		[9, 9, 81]	3_vect_05.SwiftPL
var numbers: [Double] = [3,9,81] print (numbers)		[3.0, 9.0, 81.0]	3_vect_06.SwiftPL
var n = [3,9,27] let ordenado = (n[0]<n[1])&&(n[1]<n[2]) print (ordenado)		true	3_vect_07.SwiftPL
var reales: [Double] = [3,9,81] var enteros = [5,7,11] print (reales[0] * enteros[0])		error (no se pueden operar variables de distinto tipo)	3_vect_08.SwiftPL

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

var enteros = [5,7,11]
for numero in enteros
{
  print (numero)
}
	Uso de operaciones con bucle for	5
		7
		11
			4_for_01.SwiftPL

var enteros = [5,7,11]
for numero in enteros
{
  print (numero)
}
print (numero)
		5
		7
		11
		error (la variable numero no está fuera del ámbito del for)
			4_for_02.SwiftPL

var enteros = [5,7,11]
for index in 0...2 {
  print(enteros[index])
}
		5
		7
		11
			4_for_03.SwiftPL

let base, exponente: Int
base = 2
exponente = 4
var resultado = 1
for _ in 1...exponente {
  resultado = resultado * base
}
print(resultado)
		16	4_for_04.SwiftPL

var a = [2,4,6]
var b = [2,3]
for i in a {
  for j in b{
  print(i+j)
  }
}
		4
		5
		6
		7
		8
		9
			4_for_05.SwiftPL