Intérprete de un lenguaje para GO
(Examen septiembre 2017)

El lenguaje de programación GO ha sido desarrollado por un grupo de trabajo en Google junto con la contribución de la comunidad open source y se distribuye bajo la licencia BSD. Según describen en su página oficial (https://golang.org) se trata de un lenguaje compilado, con un tipado estático, que permite concurrencia y otras características.

Se desea **implementar mediante Java, JFlex y Cup un intérprete del lenguaje de programación Go** al que llamaremos **GoPL** pero limitando y variando ligeramente sus funciones a un núcleo básico. Concretamente se han seleccionado características de la guía básica relativa a declaración de variables (https://tour.golang.org/basics/1) y a las sentencias de control (https://tour.golang.org/flowcontrol/1)

Aspectos léxicos

(información adicional y complementaria en https://golang.org/ref/spec#Lexical_elements)

Todo programa en GoPL se organiza en paquetes y dispone de una función main donde comienza la ejecución. Se permite la importación de paquetes donde vendrán definidas más funciones. En nuestro caso los programas escritos en GoPL siempre tendrán la siguiente forma:

```
package main

import "fmt"

func main() {
        // bloque de sentencias
        fmt.Println("Hola Mundo")
}
```

El cuerpo de una función en **GoPL** se compone de una o más **sentencias**, cada una de ellas escrita en una línea. Cada línea puede acabar opcionalmente con un punto y coma ";" . Por ejemplo, las sentencias `fmt.Println("Hola Mundo")` y `fmt.Println("Hola Mundo");` son equivalentes. El bloque de sentencias se inicia con una llave que debe estar en la misma línea que la palabra reservada `func`.

El uso de espacios o tabuladores no tiene efecto (el sangrado de las líneas tampoco).

Los comentarios se incluyen de la misma forma que en C o C++ (// comentario en línea o /* comentario general */)

El tipo de datos de las variables se declara de forma explícita. Existen múltiples tipos, pero usaremos únicamente los tipos `int`, `bool` y `string`. Los valores enteros que se guarden en las variables de tipo `int` se escribirán con su representación numérica, los valores booleanos de las variables `bool` serán `true` o `false` y las cadenas de texto que se almacenen en variables de tipo `string` vendrán limitadas por comillas dobles al principio y al final (`"hola"`).

Los nombres para las variables deben empezar con una letra y puede estar seguida por otras letras, números o el guión bajo ('_').

Existen varias formas para declarar e inicializar las variables:

- Usando la palabra reservada var para declarar una lista de variables (separadas por comas), donde el tipo se coloca al final. Supondremos que todas las variables se definen en la función (y no hay variables globales).

```
...

func main() {
        var a, b, c int;
}
```

  En el caso de que no se inicialicen explícitamente (como el caso anterior), todas las variables se inicializan con el valor "cero", que para los números es el 0, para los booleanos es el valor falso y para las cadenas de texto es la cadena vacía ("").

- Usando la palabra reservada var e indicando el tipo de las variables e inicializando las variables. Para ello, se añade, después del tipo de variable, el signo igual ('=') y se dan valores a todas las variables declaradas.

```
...

func main() {
        var a, b, c int = 1, 2, 3;
}
```

- Usando la palabra reservada var , SIN indicar el tipo de las variables e inicializando las variables. Cada variable tomará el tipo adecuado para almacenar el tipo correspondiente

```
...

func main() {
        var num_vueltas, ok, txt_output = 10, true, "Correcto";
        // serán variables de tipo int, bool y string
}
```

- Usando el operador de asignación abreviada ":=" que sustituye a la palabra reservada var , y donde el tipo se asigna, como en el caso anterior, de forma implícita según el tipo de dato que se vaya a almacenar

```
...

func main() {
        num_vueltas, ok, txt_output := 10, true, "Correcto";
        tmp := num_vueltas;
        // tmp será del mismo tipo que num_vueltas
```

```
        // y tendrá el mismo valor
}
```

Las constantes se declaran como las variables, pero usando la palabra reservada `const` en vez de `var`. No se puede usar el operador := . Por simplicidad, las constantes las declararemos fuera de la función `main`.

```
...
const Max = 314;
const TxtFin string = "Hasta luego";

func main() {
        ...
}
```

En las variables se pueden asignar valores como en otros lenguajes (C, Java) mediante el uso del operador '=' que evalúa la expresión de la derecha y la asigna a la variable que está a la izquierda. Consideraremos los operadores aritméticos (para números enteros) de suma, resta, multiplicación, división y módulo (+ , - , * , / , %), los operadores relacionales (>, >=, <, <=, ==, !=). Para las cadenas de texto no vamos a definir ningún operador. Se aplican los mismos criterios de prioridad y asociatividad, permitiendo el uso de paréntesis.

```
...
func main() {
        fmt.Println(1+16/5, 3<4);
        fmt.Println((1+16)/5, 3==4);
        // daría como salida
        // 4 true
        // 3 false
}
```

Funciones para imprimir por pantalla

La función `Print` (que está en el paquete `fmt` y por eso la llamaremos siempre `fmt.Print`) sirve para escribir por pantalla, aquello que se le pasa como parámetro (como una lista de variables, constantes o valores constantes separadas por comas). No introduce nueva línea al terminar, pero sí introduce espacios entre los elementos que debe mostrar (en realidad sólo los introduce entre números, pero por simplicidad, los pondremos entre cada elemento).

```
...
func main() {
        var i, j, k int = 1, 2, 3;
        fmt.Print(i, j);
        fmt.Print(k, i);
        // su salida sería ''1 23 1''
        // no hay espacio antes del 3 porque viene de otro Print
}
```

La función `Println` (que está en el paquete `fmt` y por eso la llamaremos siempre `fmt.Println`) sirve para escribir por pantalla, aquello que se le pasa como parámetro (como una lista de variables, constantes o valores constantes separadas por comas). Sí introduce nueva línea al terminar, y también introduce espacios entre los elementos que debe mostrar.

```
...
func main() {
        var i, j, k int = 1, 2, 3;
        c, python, java := true, false, "NO";

        fmt.Println(i, j, k);
        fmt.Println(c, python, java);
        // su salida sería
        // 1 2 3
        // true false NO
}
```

La función Printf (que está en el paquete fmt y por eso la llamaremos siempre fmt.Printf) es análoga al printf de C y permite aplicar formato especial a la salida mediante "verbos". Se propone incluir únicamente los "verbos" %v y %T que sirven para sacar el valor de una variable y su tipo, respectivamente

```
...
func main() {
        var i int = 1;
        fmt.Printf("El valor es %v y el tipo %T", i, i);
        // daría como salida "El valor es 1 y el tipo int"
}
```

Sentencias de control

De entre las sentencias de control disponibles se pide implementar la del bucle for, que es el núcleo de los bucles en GoPL. La sentencia for tiene 3 componentes separados por puntos y coma ';' antes del cuerpo del bucle. Esos 3 componentes (al igual que otros lenguajes) son para la inicialización, la condición de parada y la actualización que se ejecuta después de cada iteración y antes de volver a comprobar la condición de parada. A diferencia de C, esos 3 componentes no se encierran entre paréntesis y el bloque de sentencias a repetir debe estar encerrado entre llaves obligatoriamente. El bloque de sentencias se inicia con una llave que debe estar en la misma línea que la palabra reservada for. En el componente de inicialización sólo permite la declaración de variables con el operador :=

```
...
func main() {
        sum := 0;
        for i := 0; i < 10; i++ {
                sum = sum + i;
        }
        fmt.Print(sum);
}
```

El primer y último componente son opcionales (sería como un bucle while) e incluso podrían quitarse los puntos y coma

```
...
func main() {
```

```
        sum := 1;
        for ; sum < 10; {
                sum = sum + sum;
        }
        sum2 := 1;
        for sum2 < 100 {
                sum2 = sum2 + sum2;
        }

        fmt.Print(sum, sum2);
}
```

Implementación

Se pide implementar un intérprete que lea un programa escrito en **GOPL** de un fichero de texto con extensión .gopl 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 GoPL a16.gopl
```

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

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

ANEXO

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

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

`package main`
 	
`import "fmt"`
 	
`func main() {`
`  fmt.Print("HOLA MUNDO");`
`}`
	Fichero con elementos básicos, comentarios, punto y coma, espacios, sangrado, errores, etc.
	
	Las primeras líneas del fichero se dan por supuesto en sucesivos ejemplos
		HOLA MUNDO	0_base_01.gopl

`package main`
 	
`import "fmt"`
 	
`main() {`
`  fmt.Print("HOLA MUNDO");`
`}`
		ERROR
		(en función main)
		// no es preciso
		// identificar el tipo
			0_base_02.gopl

`package main`
 	
`import "fmt"`
 	
`func main() {`
`  fmt.Print("HOLA MUNDO")`
`}`
		HOLA MUNDO	0_base_03.gopl

`...`
`func main()`
`{`
`  fmt.Print("HOLA MUNDO");`
`}`
		ERROR
		(en función main)
		// no es preciso
		// identificar el tipo
			0_base_04.gopl

`...`
`func main() {`
`  /* FUNCION PRINCIPAL`
`  fmt.Print("HOLA "); */`
`  fmt.Print("MUNDO");`
`}`
		MUNDO	0_base_05.gopl

`...`
`func main() {`
`  /* FUNCION PRINCIPAL */`
`  fmt.Print("HOLA "); */`
`  fmt.Print("MUNDO");`
`}`
		ERROR
		(comentario mal formado)
		// no es preciso
		// identificar el tipo
			0_base_06.gopl

`...`
`func main() {`
`  // FUNCION PRINCIPAL`
`  fmt.Print("HOLA ");`
`  fmt.Print("MUNDO");`
`}`
		HOLA MUNDO	0_base_07.gopl

`...`
`func main() {`
`  // FUNCION PRINCIPAL`
`  fmt.Print("HOLA //");`
`  fmt.Print("MUNDO");`
`}`
		HOLA // MUNDO	0_base_08.gopl

`...`
`func main() {`
`  // FUNCION PRINCIPAL`
`  fmt.Print("HOLA "//);`
`  fmt.Print("MUNDO");`
`}`
		ERROR
		(comentario mal formado)
		// no es preciso
		// identificar el tipo
			0_base_09.gopl

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

`func main() {`
`  var a, b, c int;`
`  fmt.Print(a, b, c);`
`}`
	Declaración de variables en sus distintas modalidades y probando diferentes errores (sólo hay que detectar el error)	0 0 0	1_var_01.gopl

`func main() {`
`  var a int;`
`  var b bool;`
`  var c string;`
`  fmt.Print(a, b, c);`
`}`
		0 false	1_var_02.gopl

`func main() {`
`  var a, b, c;`
`  fmt.Print(a, b, c);`
`}`
		ERROR	1_var_03.gopl

`func main() {`
`  var a, b, int;`
`  fmt.Print(a, b, c);`
`}`
		ERROR	1_var_04.gopl

`func main() {`
`  var a, INT, c int;`
`  fmt.Print(a, INT, c);`
`}`
		0 0 0	1_var_05.gopl

`func main() {`
`  var 6a int;`
`  fmt.Print(6a);`
`}`
		ERROR	1_var_06.gopl

`func main() {`
`  aux int;`
`  fmt.Print(aux);`
`}`
		ERROR	1_var_07.gopl

`func main() {`
`  var a, b, c int = 1, 2, 3;`
`  fmt.Print(a, b, c);`
`}`
		1 2 3	1_var_08.gopl

`func main() {`
`  var a, b, c int = 1, 2;`
`  fmt.Print(a, b, c);`
`}`
		ERROR	1_var_09.gopl

`func main() {`
`  var a, b, c int = 1, 2, "3";`
`  fmt.Print(a, b, c);`
`}`
		ERROR	1_var_10.gopl

`func main() {`
`  var a, b, c = 1, 2, 3;`
`  fmt.Print(a, b, c);`
`}`
		1 2 3	1_var_11.gopl

`func main() {`
`  var a = 1, b = 2, c = 3;`
`  fmt.Print(a, b, c);`
`}`
		ERROR	1_var_12.gopl

`func main() {`
`  var a, txt, ok = 2, "hola", true;`
`  fmt.Print(a, txt, ok);`
`}`
		2 hola true	1_var_13.gopl

`func main() {`
`  a, txt, ok := 4, "GOPL", false;`
`  fmt.Print(a, txt, ok);`
`}`
		4 GOPL false	1_var_14.gopl

`const Max = 100;`
 	
`func main() {`
`  a, txt, ok := 4, "GOPL", false;`
`  fmt.Print(Max, a, txt, ok);`
`}`
		100 GOPL false	1_var_15.gopl

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

`func main() {`
`  var i, j, k int = 1, 2, 3;`
`  c, python, java := true, false, "NO";`
 	
`  fmt.Println(i);`
`  fmt.Println(j, k);`
`}`
	Uso de las funciones de impresión (print en sus diferentes variantes) incluyendo las operaciones aritméticas y relacionales	1
		2 3
			2_print_01.gopl

`func main() {`
`  var i, j, k int = 1, 2, 3;`
`  c, python, java := true, false, "NO";`
 	
`  fmt.Println(i, j, k);`
`  fmt.Println(c, python, java);`
`}`
		1 2 3
		true false
		NO
			2_print_02.gopl

`func main() {`
`  var i, j int = 1, 2;`
`  fmt.Println(1+16/5 , 3<4);`
`  fmt.Println((1+16)/5 , 3==4);`
`  fmt.Println((i+j*j*j*j)/5 , i+j==j*j);`
`}`
		4 true
		3 false
		3 false
			2_print_03.gopl

`func main() {`
`  var i int = 1;`
`  fmt.Printf("valor es %v y el tipo %T", i, i);`
`}`
		valor es 1 y el tipo int	2_print_04.gopl

`func main() {`
`  fmt.Printf("valor es %v y el tipo %T", 4, true);`
`}`
		valor es 4 y el tipo bool	2_print_05.gopl

`func main() {`
`  fmt.Printf("valor es %v y el tipo %T", int, true);`
`} ERROR`
		ERROR	2_print_06.gopl

Entrada	Descripción	Resultado de la ejecución	EJEMPLO

`func main() {`
`  var sum int = 0;`
`  for i := 0; i < 10; i++ {`
`  sum = sum + i;`
`  }`
`  fmt.Print(sum);`
`}`
	Uso del bucle for en distintas formas	45	3_for_01.gopl

`func main() {`
`  var sum int = 0;`
`  for var i int = 0; i < 10; i++ {`
`  sum = sum + i;`
`  }`
`  fmt.Print(sum);`
`}`
		ERROR	3_for_02.gopl

`func main() {`
`  var sum int = 1;`
`  for ; sum < 10;  {`
`  sum = sum + sum;`
`  }`
`  fmt.Print(sum);`
`}`
		16	3_for_03.gopl

`func main() {`
`  var sum int = 1;`
`  for sum < 10  {`
`  sum = sum + sum;`
`  }`
`  fmt.Print(sum);`
`}`
		16	3_for_04.gopl

`func main() {`
`  var sum int = 0;`
`  for (i := 0; i < 10; i++) {`
`  sum = sum + i;`
`  }`
`  fmt.Print(sum);`
`}`
		ERROR	3_for_05.gopl