Yogur

Jorge Osés Grijalba y Álvaro Rodríguez García

Este documento detalla la sintaxis del lenguaje de programación **Yogur**.

Estructura de archivo

Los archivos de Yogur pueden contener código fuera de ámbito. Este es, de hecho, casi imprescindible, porque la ejecución comenzará por la primera línea de código del archivo. Por tanto, la ejecución de un archivo que contenga tan solo funciones o clases no va a hacer nada.

Identificadores y ámbitos de definición

- Los delimitadores serán los saltos de línea
- Declaración de variables simples:

  ```
  var nombre : Tipo
  ```

- Declaración de arrays: igual pero [Tipo]

  ```
  var nombre : Tipo [longitud]
  ```

  Donde longitud es un entero

- La delimitación de los bloques se hará con {}
- La Tabla de Símbolos se representará mediante un hashmap, ya que nos permite tiempos constantes.
- Las funciones no contarán con una cláusula return como tal sino que si devuelven una variable. Esta se especificará en la cabecera de la función (argR en el siguiente ejemplo):

  ```
  def nombre (arg1 : Tipo, arg2 : Tipo) -> argR : Tipo {
      ...
  }
  ```

- Para los procedimientos, la sintaxis será la misma pero sin valor de retorno. La sintaxis de clases será la siguiente:

```
class Clase {
    var a : Tipo
    ...
    def fun(...) -> argR : Tipo {
        ...
    }
}
```

- Se accederá a los miembros de una clase y a sus funciones mediante el operador ".", de la forma típica.

Tipos

- Los tipos predefinidos serán **int** para los enteros (de 32 bits) y **bool** para los booleanos.
- Los operadores infijos serán +, *, -, /, and, not, or , ==, >=, >, <=, <, y contarán con la asociatividad y prioridad clásica de los lenguajes de programación (como puede ser la de C).
- También usaremos **false** y **true** como palabras reservadas.

Instrucciones ejecutables

- El acceso a los elementos de array se hará como es habitual con la notación `a[i]` para acceder al elemento `i` del array `a` , indexados desde 0.
- Cada array guardará de forma especial su longitud para poder realizar operaciones como obtener el último elemento `a[-1]` o acceder a subarrays dentro del array con una notación de rangos, de forma que `a[:5]` devolvería los elementos hasta el 5 y `a[5:]` devolvería los elementos a partir del 5.
- Las instrucciones condicionales se realizarán de la manera usual:

```
if condition {
    // Instrucciones
} else if condition {
    // Instrucciones
} else {
    // Instrucciones
}
```

- Los bucles while también serán los habituales:

```
while condition {
    // Instrucciones
}
```

- La sintaxis de los for será, para un for desde `a` hasta `b-1` :

```
for i in a to b {
    // Instrucciones
}
```

- Las llamadas a funciones se realizarán de la siguiente manera:

```
var a : Tipo = fun(arg1, arg2, ...)
```

- Y las llamadas a procedimientos:

```
proc(arg1, arg2, ...)
```

Comentarios

Los comentarios serán estilo c, con `//` para un comentario de línea y `/* ... */` para un comentario de bloque.

Ejemplos

Ejemplo general

```
class Container {
    var one : int
    var two : int
}

def sum(one : int, two : int) -> sum : int {
    sum = one + two
}

var cont : Container
cont.two = 2
cont.one = 1

var estaeluno : bool = false
var estaeldos : bool = false

var numbers : int[4]
numbers[0] = 1
numbers[1] = 2
numbers[2] = 3
numbers[3] = 4

for i in 0 to 4 {
    if numbers[i] == cont.two {
        estaeldos = true
    } else if numbers[i] == cont.one {
        estaeluno = true
    }
    numbers[i] = sum(cont.two,cont.one) // set numbers[i] to 3
}
```

Ejemplo bucle while: cálculo del factorial

```
var fact : int = 1
var n : int = 5     // Para calcular 5!

while n > 1 {
    fact = fact * n
    n = n - 1
}
```