Definición del lenguaje

Carlos Brito Pérez
Christian García Viguera

Índice

Keywords **4**

Estructura **4**

Misceláneos **4**
- Instrucciones 4
- Indentado 4
- Comentarios 4

Tipos **4**
- Primitivos 4

Funciones **5**
- Return 5

Impresión por pantalla **5**

Estructuras de control **6**

Bucle **6**

Operadores **7**
- Aritméticos 7
  - Asignación 7
  - Suma 7
  - Resta 7
  - Multiplicación 7
  - División 7
- Relacionales 7
  - Igualdad 8
  - Desigualdad 8
  - Mayor que 8
  - Mayor o igual que 8
  - Menor que 9
  - Menor o igual que 9
- Lógicos 9
  - AND 9
  - OR 10
  - NOT 10
- Asignación 10
- Precedencia 11
- Código de ejemplo 11

Keywords

- *return*

Estructura

El compilador leerá el fichero línea a línea comenzando por la primera de arriba y bajando en orden. No es necesario una función principal. Se espera que los símbolos (variables y funciones) estén declarados antes de su llamada.

Las funciones, o métodos, no necesitan encabezados. Se declararán con las instrucciones que se ejecutarán al llamarse.

El programa termina cuando no tenga más instrucciones que leer en el fichero.

Código de ejemplo.

Misceláneos

Instrucciones

Cada línea del programa se corresponde con una instrucción. La instrucción acaba con la línea, por tanto no es necesario terminarla con ningún símbolo.

Indentado

No será necesario usar ninguna forma de indentado, pero soportará su uso.

Comentarios

El compilador ignorará las líneas precedidas por ~~ (doble virgulilla)

Tipos

Primitivos

- **Entero:** int (32 bits con signo)
- **Coma flotante:** float (32 bits con signo)
  Un float deberá ser declarado con un punto decimal, incluso si no posee valores decimales.
- **Booleanos:** bool (1 bit)
  Dos valores: *true* y *false*.

- **Caracteres:** char (16 bits) (unicode)
  Los chars se declararán con comillas simples (char a = ‘A’)

Funciones

Para llamar a una función se debe escribir su nombre seguido de dos paréntesis:

```
nombre_funcion()
```

Para declarar funciones se utilizará la palabra clave *meth*, seguido del nombre, paréntesis, dos puntos y el tipo de la función. Tras eso, se deben abrir llaves para contener el código de la función.

Como cualquier símbolo, dos funciones no pueden compartir nombre.

Ejemplo válido:

```
meth nombre_funcion(): t_funcion{
~~contenido sin indentar
}

meth nombre_funcion()2: t_funcion{
      ~~contenido indentado
}
```

Return

La palabra clave “return” provoca que se salga de la función en la que se encuentra, devolviendo el programa a la línea en la que se llamó a la función. No es necesaria en ningún caso, puesto que las funciones no devuelven nada.

```
meth nombre_funcion(): t_funcion{
      ~~contenido alcanzable
      return
      ~~contenido no alcanzable
}
```

Impresión por pantalla

Se utilizará la función print, que necesita una string como parámetro.

```
print(“texto”)
print(nombre_variable)
```

Estructuras de control

Las estructuras de control modifican el flujo del código, haciendo que se salten instrucciones bajo ciertas condiciones.

Para hacer que un trozo de código se ejecute bajo una condición se utilizará la estructura *if*. La condición del *if* debe ser una expresión de tipo booleano.

```
if ( … ) {
~~instrucción
}
```

Bucle

Los bucles provocan que una parte del código se repita una cantidad de veces previamente especificada.

```
loop(5){
      ~~Contenido a repetir
}
```

Operadores

Aritméticos

Todas las operaciones necesitan dos variables o valores, que se encontrarán a ambos lados del operador (+, -, *, /). Deben acompañarse de la operación de asignación (de no hacerse, se produciría un error).

```
a = 2 + 3 * 4
```

Asignación

Asigna un valor a una variable del mismo tipo, aunque existen excepciones, como los enteros y los coma flotante. Si se asigna un float a un entero, se obtiene un entero con la parte entera del float, truncándose.

Suma

```
a = 3 + 5
```

Resta

```
a = 3 - 5
```

Multiplicación

```
a = 3 * 5
```

División

```
a = 3 / 5
```

Módulo

```
a = 5 % 2 ~~ 5 mod 2
```

Relacionales

Los operadores relacionales producen resultados de tipo booleano. También necesitan de dos operandos, como los aritméticos. No deben asignarse a variables,solo utilizarse como condiciones de alguna estructura de control.

```
if (a > b) …
```

Se evaluará si la condición es verdadera o falsa, y entonces se determinará el comportamiento de la estructura de control.

Se pueden utilizar con los tipos entero, float y char. Pudiendo combinar los tipos int y float.

Igualdad

La operación “a == b” realiza la pregunta “¿es el valor de a igual al de b?”.

```
int a = 1
float b = 1.
int c = 2

(a == b) ~~ true
(a == c) ~~ false
(b == c) ~~ false
```

Desigualdad

La operación “a != b” realiza la pregunta “¿es el valor de a diferente al de b?”.

```
int a = 1
float b = 1.
int c = 2
```

```
(a != b) ~~ false
(a != c) ~~ true
(b != c) ~~ true
```

Mayor que

La operación “a > b” realiza la pregunta “¿es el valor de a mayor al de b?”.

```
int a = 1
float b = 1.
int c = 2

(a > b) ~~ false
(a > c) ~~ false
(c > b) ~~ true
```

Mayor o igual que

La operación “a >= b” realiza la pregunta “¿es el valor de a mayor al de b?”.

```
int a = 1
float b = 1.
int c = 2

(a >= b) ~~ true
(a >= c) ~~ false
(c >= b) ~~ true
```

Menor que

La operación “a < b” realiza la pregunta “¿es el valor de a menor al de b?”.

```
int a = 1
float b = 1.
int c = 2

(a < b) ~~ false
(a < c) ~~ true
(b < c) ~~ true
```

Menor o igual que

La operación “a <= b” realiza la pregunta “¿es el valor de a menor al de b?”.

```
int a = 1
float b = 1.
int c = 2
```

```
(a <= b) ~~ true
(a <= c) ~~ true
(c <= a) ~~ false
```

Lógicos

Los operadores lógicos también necesitan dos operandos, en este caso booleanos o lógicos. Estos operadores devuelven valores también booleanos. Se evalúan de izquierda a derecha.

AND

AND lógico. Devuelve true si ambos operandos son true, y falso si alguno de ellos es false.

```
bool a = true
bool b = false
bool c = true

(a and b) ~~ false
(a and c) ~~ true
```

OR

OR lógico. Devuelve true si alguno de los operandos es true, y falso si ninguno de ellos lo es.

```
bool a = true
bool b = false
bool c = false

(a or b) ~~ true
(b or c) ~~ false
```

NOT

NOT lógico. Este operador sólo necesita un operando. Invierte el valor de la variable (u expresión) lógica que lo acompaña por la derecha.

```
bool a = true
bool b = false
bool c = true

(not a) ~~ false
(a and b) ~~ false
(not (a and b)) ~~ true
```

Asignación

Estas operaciones modifican el valor al que apunta la dirección de memoria de la variable.

Si se asigna una variable a otra, la primera apuntará al valor al que apunta la segunda:

```
int a = 5
int b = 6
a = b ~~ a = 6
```

Precedencia

La asociatividad de las operaciones de cada sección es de izquierda a derecha.

Operaciones por orden de prioridad:

1. multiplicación, división
2. suma y resta
3. operadores relacionales (sin igualdad y desigualdad)
4. igualdad y desigualdad (==, !=)
5. and
6. or
7. not
8. asignaciones

Código de ejemplo

```
int a = 4
int b = 2
int c

~~ Method that computes the product of two global variables
meth multiplyInt(): int{
     c = a * b
     return
}

~~ Method that computes the subtraction of two global variables
meth subtractInt(): int{
     c = a - b
     return
}

print(c)
multiplyInt()
print(c)
subtractInt()
print(c)
```