

# C+—

Juan Diego Barrado Daganzo, Javier Saras González y Daniel González Arbelo  
4º de Carrera

24 de marzo de 2024\*

## Índice

<b>1. Especificaciones técnicas del lenguaje</b>	<b>1</b>
1.1. Identificadores y ámbitos de definición . . . . .	1
1.2. Tipos . . . . .	2
1.2.1. Enteros y booleanos . . . . .	2
1.2.2. Clases y registros . . . . .	3
1.2.3. Arrays . . . . .	3
1.2.4. Funciones . . . . .	3
1.2.5. Punteros . . . . .	3
1.2.6. Tipos definidos por el usuario y constantes . . . . .	4
1.3. Instrucciones del lenguaje . . . . .	4
<b>A. Ejemplos de programas habituales</b>	<b>6</b>

## 1. Especificaciones técnicas del lenguaje

### 1.1. Identificadores y ámbitos de definición

El lenguaje posee las siguientes características:

- **Declaración de variables:** se pueden declarar variables sencillas de los tipos definidos y variables *array* de estos tipos, de cualquier dimensión. Los nombres de los identificadores han de ser expresiones alfanuméricas que no comiencen por números y que posiblemente tengan el caracter “\_”.

---

\*Este documento se actualiza, para consultar las últimas versiones entrar en el enlace <https://github.com/JuanDiegoBarrado/PracticaPL>

- **Bloques anidados:** se permiten las anidaciones en condicionales, bucles, funciones, etc. Si dos variables tienen el mismo nombre, la más profunda (en la anidación) tapa a la más externa.
- **Funciones:** se permite la creación de funciones y la declaración implícita dentro de otras. El paso por valor y por referencia de cualquier tipo a las funciones está garantizado.
- **Punteros:** para cada tipo se puede declarar un puntero a una variable de ese tipo, mediante la asignación de su dirección de memoria a la variable puntero.
- **Registros y clases:** se incluyen dos tipos adicionales: los registros como “saco de datos” —sin métodos— y las clases, tanto con datos como con métodos de función.
- **Declaración de constantes:** se incluye la posibilidad de declarar constantes por parte del usuario.

## 1.2. Tipos

La declaración de tipos ha de hacerse de manera explícita y de forma previa al lugar donde se emplee el identificador, es decir, que para poder usar una variable tengo que haberla declarado antes.

### 1.2.1. Enteros y booleanos

Los tipos básicos del lenguaje son los enteros y los booleanos. La sintaxis de declaración de estos tipos es la siguiente:

- **Enteros:** `int var;`

Entre las operaciones habilitadas para el tipo:

- Suma: `a + b`
- Resta: `a - b`
- Multiplicación: `a * b`
- División: `a / b`
- Potencia: `a^b`
- Paréntesis: `()`
- Menor: `a < b`
- Mayor: `a > b`
- Igual: `a = b`
- Menor o igual: `a <= b`
- Mayor o igual: `a >= b`
- Distinto: `a != b`

- **Booleanos:** `bool var;`

Entre las operaciones habilitadas para el tipo:

- Y lógico: `a and b`
- O lógico: `a or b`
- No lógico: `!a`

Además, las palabras reservadas para indicar el valor verdadero o falso son `true` y `false`.

### 1.2.2. Clases y registros

Como tipos adicionales hemos incluido los registros, las clases y las funciones. La sintaxis de declaración es la siguiente:

- **Clases:** `clas var {...};`

Entre las operaciones habilitadas para el tipo:

- Acceso a campos: `var.campo`
- Acceso a métodos: `var.metodo`
- Constructor: `var(args)`

- **Registros:** `estruct var {...};`

Entre las operaciones habilitadas para el tipo:

- Acceso a campos: `var.campo`
- Constructor: `var(args)`

### 1.2.3. Arrays

Todos los tipos pueden formar un array multidimensional, la sintaxis de declaración es la siguiente:

- **Array:** `Tipo[DIMENSION] var;`

Entre las operaciones habilitadas para el tipo:

- Operador de acceso: `var[INDEX]`

### 1.2.4. Funciones

Las funciones se han declarado también como un tipo para poder hacer expresiones lambda y pasar funciones como argumento. La sintaxis de declaración de una función es la siguiente:

- **Funciones:** `func var(Tipo arg1, Tipo arg2, ...) : TipoRetorno {...};`

El paso de parámetros por defecto es por valor, pero puede cambiarse a por referencia añadiendo el carácter “&” al final del tipo del argumento.

### 1.2.5. Punteros

Pueden declararse punteros a cualquiera de los tipos definidos. La sintaxis de declaración es:

- **Puntero**<sup>1</sup>: `Tipo~ var`

Entre las operaciones habilitadas para el tipo:

- Asociación a memoria dinámica: `var := niu Tipo`
- Acceso al dato: `~var`

---

<sup>1</sup>Como nota especial, la declaración de un puntero a estructuras de tipo array, se haría como `Tipo[DIMENSION] var;`.

### 1.2.6. Tipos definidos por el usuario y constantes

Adicionalmente, permitimos la definición de tipos por parte del usuario a través de la palabra reservada:

- **Definición de tipos de usuario:** `taipdef nombre expresion.`

Por último, la declaración de constantes se ejecuta a través de la expresión:

- **Declaración de constantes:** `difain NOMBRE valor`

## 1.3. Instrucciones del lenguaje

El lenguaje tiene el siguiente repertorio de instrucciones:

- **Instrucción de asignación:** `:=`

```
1  int var := 3;
```

- **Instrucciones condicionales:** `if-els`, `suich-queis`

```
1  if (var > 3) {
2      ...
3  }
4  els {
5      ...
6  }
```

```
1  suich (var) {
2      queis(val1):
3          ...
4          breic;
5      queis(val2):
6          ...
7          breic;
8      default:
9          ...
10         breic;
11 }
```

- **Instrucción de bucle:** `while`

```
1  guail (var > 0) {
2      ...
3  }
```

Se incluyen además las instrucciones `breic` y `continiu`.

- **Instrucciones de entrada salida:** `cein()` y `ceaut()`

```
1  cein(var);
2  ceaut(var);
```

- Instrucción de retorno de función: `return`

```
1 func foo(Tipo arg) :  
    TipoRetorno {  
2     TipoRetorno var;  
3     ...  
4     return var;  
5 }
```

## A. Ejemplos de programas habituales

Listing 1: Ejemplo de programa típico en  $C + -$ .

```
1 main() : int {
2     int a;
3     int b;
4     cein(a);
5     cein(b);
6     int c := a + b;
7     ceaut(c);
8     return c;
9 }
```

Listing 2: Ejemplo de programa iterativo en  $C + -$ .

```
1 func maximo(int[] lista, int tam) : int {
2     int maximo := -1;
3     int i := 0;
4     guail(i < tam) {
5         if(lista[i] > maximo) {
6             maximo := lista[i];
7         }
8         i++;
9     }
10    return maximo;
11 }
```

Listing 3: Ejemplo de arrays multidimensionales en  $C + -$ .

```
1 func multiplicarMatrices(int[][] A, int[][] B, int tam) : int[] {
2     int C[tam][tam];
3     int i := 0;
4     int j;
5     guail(i < tam) {
6         j := 0;
7         guail(j < tam) {
8             C[i][j] := 0;
9             j++;
10        }
11        i++;
12    }
13    int k;
14    i := 0;
15    guail(i < tam) {
16        j := 0;
17        guail(j < tam) {
18            k := 0;
19            guail(k < tam) {
20                C[i][j] := C[i][j] + A[i][k] * B[k][j];
21                k++;
22            }
23            j++;
24        }
25        i++;
26    }
27    return C;
28 }
```

Listing 4: Ejemplo de programa recursivo y sentencia if-els en  $C + -$ .

```
1 func factorial(int num) : int {
2     if(num = 0){
```

```

3         return 1;
4     }
5     else {
6         return num * factorial(num - 1);
7     }
8 }

```

Listing 5: Ejemplo de uso de registros y sentencia `suich` en  $C + -$ .

```

1  estruct Alumno {
2      int DNI;
3      int nota;
4  };
5
6  func aprobado(Alumno a) : bul {
7      suich (a.nota) {
8          queis(0):
9          queis(1):
10         queis(2):
11         queis(3):
12         queis(4):
13             return fols;
14             breic;
15         queis(5):
16         queis(6):
17         queis(7):
18         queis(8):
19         queis(9):
20         queis(10):
21             return tru;
22             breic;
23     }
24 }
25
26 func nota(Alumno~ a) : int {
27     return a->nota;
28 }
29
30 func mein():int {
31     Alumno daniel := { 123456789, 6};
32     Alumno~ danielptr := &daniel;
33     ceaut( nota(danielptr) );
34     ceaut( aprobado(daniel) );
35
36     return 0;
37 }

```