

Definición de la sintaxis del lenguaje C—

Procesadores del Lenguaje Entrega 1

Óscar Calvet Sisó, Noelia Barranco Godoy

1. Introducción

El objetivo de este documento es el de especificar la sintaxis de nuestro lenguaje, C—. Este va a ser un lenguaje de programación imperativo y fuertemente tipado. Nos vamos a basar en el padre de todos los lenguajes imperativos modernos: C. Nuestro objetivo con C— es el de crear una versión simplificada de C y añadirle algunas utilidades como la definición de clases (sin herencia y declaradas en el ámbito global de nuestro programa). Contaremos únicamente con un archivo de código.

El punto de entrada de todo programa válido de nuestro lenguaje será una función 'main', que no devolverá nada y recibirá un array de ints (potencialmente vacío).

Nuestro lenguaje implementará comentarios de una sola línea al estilo de C. Cabe destacar que la cadena // estará prohibida dentro de los comentarios. Por último, al final expondremos una lista de palabras reservadas. Estas son palabras que por su construcción bien podrían ser identificadores, pero nosotros las prohibiremos porque se usarán en otros contextos.

2. Identificadores y ámbitos de definición

El lenguaje permite declarar variables (o constantes) simples y arrays de cualquiera de los tipos descritos más adelante (incluidos arrays de varias dimensiones). Estas variables podrán estar alojadas en memoria estática o dinámica. Como es habitual, habrá anidamiento de bloques con sus respectivos y funciones, que podrán tener argumentos de cualquier tipo.

Las funciones solo se podrán definir en el ámbito más general del proyecto (fuera de cualquier función), así como, los tipos de usuario, tipos estructurados y clases. Las funciones podrán recibir como argumentos parámetros de cualquier tipo, incluido punteros o referencias.

En el ámbito global, también se podrán declarar variables, definir tipos nuevos .

3. Tipos

3.1. Tipos básicos y de usuario

En nuestro lenguaje existirán tres tipos básicos, el tipo entero (con identificador int), el tipo booleano (con identificador bool) y el tipo carácter (con identificador char). Además, el usuario podrá declarar pseudónimos de cualquier tipo del lenguaje así como, clases y estructurados.

3.2. Arrays y punteros

Con todos estos tipos, podremos crear arrays y punteros. La notación que identifica a los arrays será la habitual, es decir, el identificador seguido de tantos corchetes de apertura y cierre como dimensiones tenga el array. Los arrays no serán polimórficos.

La notación pertinente a los punteros serán el identificador precedido de la palabra reservada 'pointer' seguida de otro identificador de tipo. Añadiremos pues un operador de referencia que, dada una variable cualquiera devuelva un puntero que apunte a esa variable, que se expresará con el "&".

3.3. Funciones

A nivel sintáctico, todas las funciones estarán tipadas. Para esto existirá la palabra reservada "void", que indicará que una función no devuelve nada. No obstante, void no será un tipo realmente (por lo que no podrá usarse para declarar variables), void será una manera de indicar al compilador que esa función no va a devolver nada (realmente será un procedimiento).

3.4. Declaraciones

Todo tipo que no sea una función se declarará de la siguiente manera: identificador de tipo seguido de un identificador (no seguido de un paréntesis para diferenciarlo de una función). Adicionalmente, podrá ir seguido del signo igual y de una expresión del tipo a declarar.

Los tipos de array se declararán de la forma una expresión de tipo existente seguida del símbolo [, una expresión y]. Los tipos punteros se definirán con la palabra reservada 'pointer' seguida de un tipo cualquiera y terminada en un identificador seguido de un número arbitrario de corchetes cada uno conteniendo una expresión.

Las funciones se declararán de la siguiente forma: un identificador de tipo (o identificador void), seguido de un identificador y de un parentesis. Una sucesión de declaración de variables separadas por comas, terminado por un paréntesis de cierre y seguido de un bloque de código de nuestro lenguaje.

Los tipos estructurados se declararán en el ámbito general, serán inicializados por la palabra reservada "struct"seguidos de llaves que contendrán declaraciones de variables y a continuación un identificador terminado en ';'.

Las clases se declararán en el ámbito general, serán inicializadas por la palabra reservada class seguida de llaves que contendrán instrucciones, declaraciones y declaraciones de funciones, después de las llaves aparecerá un identificador terminado en ';'.

4. Conjunto de instrucciones del lenguaje

4.1. Designadores de variable

Se considerará un designador de variable cualquier cadena alfanumérica o con el símbolo "_"que no empiece por un número y también se considerará designador de variable otro designador de variable, seguido de un corchete de apertura, un número y un corchete de cierre. Esto es para considerar los accesos a arrays como designadores de variables.

4.2. Asignaciones

Toda asignación será una construcción de la forma o bien un identificador de variable, o bien una declaración de variable, seguida de un símbolo = y por último una expresión.

4.3. Llamada a función

Una instrucción de la forma identificador, paréntesis de apertura, una serie de identificadores de variable o literales separados por comas, y terminado por un cierre de paréntesis. En cualquier lugar donde pudiera ir un literal de cualquier tipo, también podrá aparecer una llamada de función en su lugar. En lugar de un identificador como nombre de función, podrá usarse la palabra reservada build, que tendrá la función de constructor en una clase (cuando le demos semántica).

4.4. Expresiones aritméticas

Siguiendo sintaxis habitual para estos, son un conjunto formado por $+$, $-$, $*$, $/$, $\%$ como operadores binarios aplicados sobre otras expresiones aritméticas. Unos identificadores constantes (números) e identificadores de variables.

También el operador prefijo $-$ seguido de otra expresión aritmética, una operación aritmética rodeada de paréntesis.

4.5. Expresiones booleanas

Otra vez siguiendo el estándar, un operador booleano se definirá con los casos base de palabras reservadas `true` y `false`, y la aplicación de los operadores binarios `&&`, `||` de manera infija sobre otras expresiones booleanas o el operador unario prefijo `!`. También se considerarán expresiones booleanas las que contienen dos identificadores de variable o expresiones de cualquier tipo y los siguientes símbolos `==`, `!=` separándolas como operador infijo, representando la igualdad y la desigualdad respectivamente.

Finalmente también existirán expresiones booleanas de la forma siguiente: dos expresiones aritméticas separadas por los operadores `<=`, `>=`, `>`, `<` de manera infija, representando las comparaciones aritméticas.

4.6. Expresiones de caracteres

Estas solamente serán de la forma comillas simples, seguida de un carácter ASCII y cerrado por otra comilla.

4.7. Expresiones de arrays

Al definir un array, siempre se hará de manera constante, o sea la apertura de un corchete seguida de una secuencia de literales separados por coma y terminado en corchete de cierre

4.8. Expresiones con punteros

Tenemos los operadores unarios `*`, `&` precede a un identificador, para acceder al valor de un puntero o a su dirección respectivamente. También tenemos el operador binario infijo `.` que se encontrará entre dos identificadores donde el de más a la izquierda hará referencia a un struct o una clase y el de la derecha a un identificador de variable o de función. La prioridad de estos operadores será la habitual de C.

4.9. Instrucción

Toda instrucción será una asignación, una de las instrucciones definidas a continuación. En el caso en el que sea una instrucción de tipo `if`, `while` o `for`, no terminará en punto y coma, pero si la instrucción es de tipo asignación o de cualquier otro tipo, terminará en punto y coma.

4.10. Instrucción while

Para representar bucles condicionales utilizaremos la siguiente sintaxis: la palabra reservada `while` seguida de paréntesis que contendrán una expresión (deberá ser booleana) y seguido de llaves que contendrán instrucciones.

4.11. Instrucción for

Para representar bucles con contadores utilizaremos la siguiente sintaxis: la palabra reservada `for` seguida de paréntesis que contendrán exactamente 3 instrucciones (declaración de variable, expresión booleana y una instrucción) seguido de llaves que contendrán instrucciones. Se separarán entre sí usando `;` y la última también habrá de terminar en este carácter.

4.12. Instrucción if simple

Para la instrucción condicional simple utilizaremos la sintaxis siguiente: la palabra reservada `if` seguida de paréntesis que contendrán una expresión (será de tipo booleana) y a continuación llaves que contendrán instrucciones.

4.13. Instrucción if compuesta

Para la instrucción condicional compuesta utilizaremos la sintaxis siguiente: la palabra reservada `if` seguida de paréntesis que contendrán una expresión (será de tipo booleana) y a continuación llaves que contendrán instrucciones. Seguidamente aparecerá la palabra reservada `else` seguida de llaves que contendrán instrucciones.

4.14. Instrucción switch

Para la instrucción `switch` utilizaremos la siguiente sintaxis: la palabra reservada `switch` seguida de paréntesis que contendrán un identificador seguido de llaves que contendrán bloques de casos consistentes del la palabra reservada `case` seguida de un identificador seguido de código entre llaves. Existe un bloque `case` especial que en lugar de empezar por la palabra `case` empieza por la palabra `default`.

4.15. Instrucción break

La instrucción `break` tendrá la sintaxis de siguiente: la palabra `'break'`. Se utilizará para salir de los bucles.

4.16. Instrucción continue

La instrucción `continue` tendrá la sintaxis de siguiente: la palabra `'continue'`. Se utilizará para saltar a la siguiente iteración en los bucles.

4.17. Instrucción return

La instrucción `return` tendrá la siguiente sintaxis: la palabra reservada `'return'` seguida de una expresión. Se utilizará argumento de retorno de una función.

4.18. Instrucción new

La instrucción `new` tendrá la siguiente sintaxis: la palabra reservada `'new'` seguida de un identificador de tipo simple o compuesto. Podrá estar seguido de una o más aperturas de corchete con un número entre ellos, esto indicará que son arrays. Se utilizará para reservar memoria dinámica.

4.19. Instrucción delete

La instrucción `delete` tendrá la siguiente sintaxis: la palabra reservada `'delete'` seguida de un identificador. Se utilizará para eliminar memoria dinámica.

4.20. Definición de tipos de usuario

La palabra reservada `'typedef'` se usará en el ámbito global para definir alias de cualquier tipo.

4.21. Identificador this

La palabra reservada `'this'` comporta como un identificador de clase y podrá aparecer allá donde aparecen estos. Luego a nivel semántico manejaremos qué pasa si la utilizamos en un contexto inadecuado.

5. Gestión de errores

La gestión de errores va a indicar el tipo de error, la ubicación en el fichero de código fuente (fila y columna) y parará la compilación. Se intentará proseguir la compilación para detectar más errores.

6. Ejemplos

```
1 void main(){
2     int a = 5;
3     int b[3] = [1, 2, 3];
4     pointer int c = new int;
5     *c = 5 + (7 % 2);
6     b = *c + a;
7     pointer int hola = new int[5];
8     for(int i = 0; i < 27 + b[2]; i = i + 1){
9         if(b[i] % 2 == 0){
10             continue;
11         }
12         else{
13             break;
14         }
15     }
16     int i = 0;
17     bool finished = false;
18     while(i <= 2 && !finished){
19         delete hola[i];
20         i = i + 1;
21     }
22 }
```

```
1 //Esto es un comentario
2 struct{
3     int a;
4     int b = 2;
5 }tStruct;
6
7 class{
8     int a;
9
10     tClase build(int a){
11         this.a = a;
12     }
13
14     void protocolo(int a){}
15     int function() {
16         this.a = this.a + 1;
17         return this.a;
18     }
19
20 }tClase;
21
22 void idonthaveatype(bool &b){
23     b = false;
24 }
25
26 int ihaveatype(pointer int k, int j){
27     *k = j + 1;
28     return *k;
29 }
30
31 void main(){
32     tClase ejemplo = tClase(5);
```

```

33     ejemplo.protocolo(2);
34     int a = ejemplo.funcion();
35     bool c = true;
36     idonthaveatype(c);
37     ihaveatype(&a, 5);
38 }
39
40 void main(){
41     tClase ejemplo = tClase(5);
42     ejemplo.protocolo(2);
43     int a = ejemplo.funcion();
44     bool c = true;
45     idonthaveatype(c);
46     ihaveatype(&a, 5);
47 }

```

```

1  typedef int aux;
2
3  void main(){
4      int a[2] = [5, 2];
5      switch(a){
6          case 1 {
7              int c = 3;
8              break;
9          }
10         case 2 {
11             if(a == 2){
12                 aux b = 3;
13             }
14         }
15         default {
16             int d = 3;
17         }
18     }
19 }

```

7. Palabras reservadas

- | | | | |
|----------|------------|-----------|-----------|
| ■ int | ■ while | ■ void | ■ false |
| ■ char | ■ if | ■ default | ■ this |
| ■ bool | ■ switch | ■ case | ■ delete |
| ■ class | ■ break | ■ new | ■ typedef |
| ■ struct | ■ continue | ■ return | ■ pointer |
| ■ for | ■ const | ■ true | |