# Lenguaje de Programación C. Sintaxis básica.

**Índice**:

1. Introducción. Estructura de un programa en C.
2. Elementos de datos en la memoria.
   2.1. Tipos de datos.
   2.2. Constantes.
   2.3. Variables.
   2.4. Conversiones de tipo.
3. Operadores.
   - Aritméticos.
   - Relacionales y lógicos.
   - Incremento y decremento.
   - Lógicos para bits
   - Operadores y expresiones de asignación.
   - Operador condicional.
   - Precedencia y orden de evaluación.
4. Instrucciones.
   - Bloques de instrucciones.
   - Declaraciones.
   - Comentarios.
   - Asignación.
   - Instrucciones de control.
     - Selección simple (if-else).
     - Selección múltiple (switch).
     - Repetición: while, for, do-while.
     - Instrucciones break, continue, goto.
   - printf y scanf.
5. Estilo.

## 1.- Introducción. Estructura de un programa en C.

Características del lenguaje de programación C:

- Lenguaje desarrollado en 1970 por Dennis Ritchie. En 1983 se estableció el estándar ANSI del lenguaje que sería aprobado en 1989.
- Se trata de un lenguaje muy utilizado. Dentro de los lenguajes imperativos clásicos es tal vez el más utilizado, siendo a su vez base de diversos lenguajes orientados a objetos, como es el caso del lenguaje C++.
- Compilado. El código fuente se ha de traducir completamente a lenguaje máquina antes de su ejecución.
- Modular. Un programa grande se puede dividir en partes (funciones) con variables locales válidas exclusivamente para esa parte del programa.
- Medianamente estructurado. Permite desarrollar programas según el método de la programación estructurada con algunas restricciones, relativas a la estructuración de módulos y funciones, que se tratarán más adelante. Incluye instrucciones de control: secuencia, selección y repetición.

- De relativo bajo nivel. Ofrece una funcionalidad mixta entre los lenguajes de alto nivel (con un alto nivel de abstracción) y los de bajo nivel (que permiten manejar directamente los recursos físicos de las máquinas).
- Es obligatorio declarar todas las variables usadas de forma explícita.
- Lenguaje muy conciso y estandarizado. Permite incorporar nuevas funcionalidades a través de bibliotecas de funciones externas.
- Lenguaje con definición de tipos relativamente débil; no siempre encontrará discordancias entre los tipos de datos, lo que podrá ser causa de problemas en tiempo de ejecución.
- Permite definir nuevos tipos de datos con gran flexibilidad.
- Diferencia entre mayúsculas y minúsculas (***main*** y ***MAIN*** son distintas).

Para formar los elementos básicos del programa, C utiliza como bloques de construcción:

- Letras mayúsculas de la ***A*** a la ***Z***.
- Letras minúsculas de la a ***a*** la ***z***.
- Los dígitos del ***0*** al ***9***.
- Los siguientes caracteres especiales:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **+** | **-** | ***** | **/** | **=** | **%** | **&** | **#** |
| **!** | **?** | **^** | **"** | **'** | **~** | **** | **\|** |
| **<** | **>** | **(** | **)** | **[** | **]** | **{** | **}** |
| **:** | **;** | **.** | **_** | **(espacio en blanco)** | | | |

Hay ciertas palabras reservadas del estándar ANSI C que tienen un significado predefinido y solo se pueden utilizar para su propósito ya establecido, y son las siguientes:

| | | | | | |
|---|---|---|---|---|---|
| **auto** | **break** | **case** | **char** | **const** | **continue** |
| **default** | **do** | **double** | **else** | **enum** | **extern** |
| **float** | **for** | **goto** | **if** | **int** | **long** |
| **register** | **return** | **short** | **signed** | **sizeof** | **static** |
| **struct** | **switch** | **typedef** | **union** | **unsigned** | **void** |
| **volatile** | **while** | | | | |

Las palabras reservadas no se pueden utilizar como identificador de los distintos elementos que el programador define al construir el programa. Las palabras reservadas están escritas con minúsculas.

**- Estructura de un programa C:**

```
#include <stdio.h>                                   (1)
...
main(){                                              (2)
      declaraciones;
      instrucciones;
}
función1(){                                          (3)
      declaraciones;
      instrucciones;
}

función2(){
      declaraciones;
      instrucciones;
}
...
```

Un programa C consta de un conjunto de definiciones de funciones (2)+(3). Cada función contiene una serie de instrucciones y realiza una tarea específica. Un programa en C debe contener por lo menos la función ***main()*** que es la que comienza la ejecución del programa (2); esta función ***main*** se corresponderá con el módulo o programa principal.

La funcionalidad directa del lenguaje C es muy reducida. Ciertas operaciones, como son las entradas por teclado y las salidas por pantalla, se incorporan mediante funciones externas que se suministran mediante bibliotecas ajenas al compilador. Éste es el caso de las operaciones de entrada y salida que se encuentran en una biblioteca estándar de nombre ***stdio.h.*** La directiva ***#include***, le indica al entorno de trabajo que ciertas operaciones se encuentran en las bibliotecas externas de funciones (1).

Ejemplo:
```
#include <stdio.h>
int main(){
      printf("Hola, soy un programa C\n");
      return 0;
}
```

Los primeros programas que vamos a implementar son sencillos y contienen exclusivamente la función ***main()***, señalando la necesidad de incluir otras funciones de biblioteca cuando se requieran (***stdio.h, stdlib.h, conio.h, math.h,...***).

Podemos observar como la zona de instrucciones, o ejecutiva, está encerrada entre llaves ***{ }***, correspondiendo en este caso a las palabras reservadas ***Inicio... Fin_algoritmo*** del pseudo-código que estamos utilizando. Los distintos elementos, funciones, declaraciones o instrucciones, utilizan como separador el carácter punto y como (***;***).

## 2.- Elementos de datos en memoria.

### 2.1. Tipos de datos.

| | | |
|---|---|---|
| Tipos de datos básicos | ***int*** | Cantidad entera. Representación exacta dentro de la computadora. |
| | ***char*** | Carácter. Representa un carácter del conjunto ***ASCII***. Se puede utilizar para almacenar valores enteros de hasta un byte. |
| | ***float*** | Número en coma flotante (incluye punto decimal y/o exponente). Representación aproximada dentro de la computadora (problemas de pérdida de precisión en las operaciones aritméticas con reales). |
| | ***double*** | Número en coma flotante de doble precisión (más cifras significativas y mayor valor posible del exponente). |
| Calificadores de tipos de datos | ***short*** | Corto. |
| | ***long*** | Largo. |
| | ***signed*** | Con signo. |
| | ***unsigned*** | Sin signo. |

Nota: el tipo de dato booleano o lógico no existe realmente en C; en su lugar se utilizan bien valores enteros o bien caracteres como bytes con un valor numérico.

El rango de los tipos elementales que se ofrece puede ser distinto según la máquina y/o compilador con que se esté trabajando; a modo de ejemplo, se detallan a continuación los propios del entorno Bloodshed Dev-C++, palabras de 32 bits; si se cambia de entorno de trabajo, es necesario comprobar los tipos y rangos válidos.

| Tipo de datos | Identificador de tipo | Rango | Tamaño |
|---|---|---|---|
| Enteros | ***short int***<br>***signed short int*** | -32768..32767 | 2 bytes |
| | ***int***<br>***signed int*** | -2147483648..2147483647 | 4 bytes |
| | ***long int***<br>***signed long int*** | -2147483648..2147483647 | 4 bytes |
| | ***unsigned int*** | 0..4294967295 | 4 bytes |
| | ***unsigned long int*** | 0..4294967295 | 4 bytes |
| Reales | ***float*** | $1.4*10^{-45}..3.4*10^{38}$ | 4 byes |
| | ***double*** | $5.0*10^{-324}..1.8*10^{308}$ | 8 bytes |
| Booleanos o lógicos | ***int***<br>***char*** | 0 (falso)<br>≠0 (verdadero) | 1 byte |
| Carácter | ***char*** | (tabla ASCII) | 1 byte |
| | ***signed char*** | -128..127 | 1 byte |
| | ***unsigned char*** | 0..255 | 1 byte |
| Cadenas de caracteres | ***char [n]*** | | n bytes |

**Nota**: existe un operador que calcula el tamaño de los objetos en bytes (incluye el \0 de las cadenas): `sizeof(tipo_dato)` `sizeof nombre_identificador`

**2.2 Constantes.**

- Enteras: `10, -3987`
  Las constantes enteras que no caben en un ***int*** se almacenan en forma ***long***.
  También se pueden definir constantes ***long*** directamente: `123L, 123l`
  Definición de constantes sin signo: `45U, 45u`
  Definición de constantes ***long*** sin signo: `234UL` (U antes de L)
- Enteros no decimales: octales: `0123` (0 delante)
  hexadecimales: `0X478, 0x478` (0X ó 0x delante).
- Reales: punto fijo: `146.09, -234.3`
  Notación científica: `2.23E-15`
  Las constantes reales se almacenan en forma ***double***.
- Caracteres: entre apóstrofes: `'A'`
  en octal ASCII: `'\101'`
  Para ciertos caracteres no imprimibles (secuencias de escape) se usa la barra invertida (****):

| | |
|---|---|
| `'\n'` | nueva línea |
| `'\t'` | tabulador horizontal |
| `'\v'` | tabulador vertical |
| `'\b'` | retroceso |
| `'\f'` | avance de página |
| `'\r'` | retorno de carro |
| `'\a'` | sonido |
| `'\0'` | nulo (marca de fin de cadena) |

| | |
|---|---|
| `'\?'` | signo de interrogación |
| `'\\'` | \ (barra invertida) |
| `'\''` | ' (apóstrofe) |
| `'\"'` | " (comillas) |

- Cadenas de caracteres ("string"): `"esto es una cadena"`
  Las constantes de cadena deben ir encerradas entre comillas.
  El compilador coloca la marca ***\0*** al final de la cadena.

**2.3 Variables.**

Nombres de variables (e identificadores):

- Se admiten letras, dígitos y subrayado (_).
- Las mayúsculas son diferentes de las minúsculas. Como norma de estilo usar minúsculas para los nombres de variables y reservar las mayúsculas para las constantes simbólicas del pre-procesador (elemento de trabajo de los compiladores de C que se verá más adelante).
- El primer carácter debe ser una letra ó **_** .
- Son significativos los 8 primeros caracteres.
- Deben ser distintos de las palabras reservadas.

**2.4 Conversiones de tipo.**

C no hace una comprobación fuerte del tipo de dato, y permite que tipos de datos diferentes se puedan mezclar en una misma expresión. Los ***char*** y los ***int*** se mezclan libremente (nótese que la marca fin de archivo ***EOF*** tiene asociado un código -1, y para que se pueda leer mediante ***getchar***, ha de leerse como un entero).

Como norma general, cuando en una expresión aparecen tipos de datos diferentes, los tipos inferiores se convierten al tipo superior de dicha expresión y ese será el tipo del resultado. Los ***char*** y ***short*** se convierten automáticamente en ***int*** y los ***float*** en ***double***. A continuación, si en la expresión hay un ***double***, todos los datos se convierten a ***double***. Si no hay ningún ***double*** y hay un dato ***long***, todos los datos se convierten a ***long***. Si ese no es el caso y hay un dato ***unsigned***, todos los datos se convierten a ***unsigned***. En caso contrario, el tipo de dato de la expresión debe ser ***int***.

En las asignaciones también se produce una conversión del tipo de dato de la expresión de la derecha al tipo de dato de la variable de la izquierda:

- de ***float*** a ***int*** por truncamiento.
- de ***double*** a ***float*** por redondeo.
- de ***long*** a ***int*** ó ***short*** por eliminación de bits de orden superior.
- de ***int*** a ***char*** por eliminación de los bits de orden superior.

Existe también la posibilidad de asignar un tipo a una expresión mediante un operador "cast" (conversión de tipo), de la siguiente manera:

```
(nombre del tipo)expresión
```

Por ejemplo:

```
(float)(5+19)
```

El resultado que sería un valor entero, 24, se convierte en un real, 24., mediante el operador de conversión de tipo ***(float)***.

## 3.- Operadores.

**- Aritméticos:** binarios: + − * / % (resto)
Unarios: − (cambio de signo)

"underflow"=0
"overflow"=mayor ó menor valor
división por cero=0 (depende del compilador)

**- Relacionales y lógicos.** Relacionales: > >= < <=
De igualdad: == (igual a) != (distinto de)
Conectivas lógicas: && ("and" lógico) || ("or" lógico)
Negación lógica: ! ("not" lógico)

En C no hay tipos lógicos. En su lugar se utilizan valores numéricos:

`verdadero≠0` `falso=0`

**- Incremento y decremento.** ++ incrementa en uno una variable.
−− decrementa en uno una variable.

Si se utilizan como prefijo (delante de la variable) producen el efecto antes de utilizar la variable en la expresión en que esté. Si se utilizan como sufijo, producen su efecto después de utilizar la variable en la expresión en que esté. Ejemplo:

| Ej: | a=3; | | a=3; | |
|---|---|---|---|---|
| | b=++a; → | a=4<br>b=4 | b=a++; → | a=4<br>b=3 |

Su utilización genera un código más eficiente.
Nota: no usarlas en variables que se empleen más de una vez en una expresión ó como argumentos de una función.

**- Lógicos para bits.**

| | | |
|---|---|---|
| Binarios: | & | Y lógico a nivel de bits. |
| | \| | O lógico a nivel de bits. |
| | ^ | O lógico exclusivo de bits. |
| | << | Rotación a la izquierda. ($x<<y \rightarrow x*2^y$) |
| | >> | Rotación a la derecha. ($x<<y \rightarrow x/2^y$) |
| Unario: | ~ | Complemento a 1. |

**- Operadores y expresiones de asignación.**

- Operador de asignación: `=`

Se utiliza en instrucciones de asignación simple:

```
variable=expresión;
```

y también en asignaciones en tubería:

```
variable1=variable2=...=variableN=expresión;
```

El operador de asignación es asociativo de derecha a izquierda.

- Operadores de acción sobre las variables:

```
+=   -=   *=   /=   %=   <<=   >>=   &=   ^=   |=
```

éstos últimos se utilizan en asignaciones de la forma:

```
variable+=expresión;
```

que equivale a:

```
variable=variable+expresión;
```

Nótese que como resultado de una expresión de asignación aparece un valor que es manejado como tal. Ejemplos:

1) `while((c=getchar())!=EOF)...`
2) `e=(a=f(...))+(b=g(...));`

**- Operador condicional.**

Sintaxis: `expresión1 ? expresión2 : expresión3`

Este operador ***?:*** evalúa ***expresión1***, si esta es verdadera, da el resultado de ***expresión2***, y en caso contrario da el resultado de ***expresión3***. Ejemplos:

1) `c=(a>b)?a:b;`
2)
```
for(i=0;i<N;i++)
      printf("%6d%c",a[i],(i%10==9||i==N-1)?'\n':' ');
```

**- Precedencia y orden de evaluación.**

| Operador | Asociatividad |
|---|---|
| `() []  ->  .` | I-D (Izquierda-Derecha) |
| `!  ~  ++  --  - (tipo) *  &  sizeof` | D-I |
| `*  /  %` | I-D |
| `+  -` | I-D |
| `<<  >>` | I-D |
| `<  <=  >  >=` | I-D |
| `==  !=` | I-D |
| `&` | I-D |
| `^` | I-D |
| `\|` | I-D |
| `&&` | I-D |
| `\|\|` | I-D |
| `?:` | D-I |
| `=  +=  -=  ...` | D-I |
| `,` | I-D |

Nota: hay que tener cuidado con la utilización de operandos dependientes del resultado de otros operandos. Ejemplo: `x[i]=i++;` da problemas.

# 4.- Instrucciones.

**- Bloques de instrucciones.**

Instrucción simple:
```
instrucción;
```
Bloque de instrucciones:
```
{     instrucción1;
      instrucción2;
      . . .
      instrucciónN;
}
```

**- Declaraciones.**

Declaración de las variables:
```
tipo var1,var2,...;
```
Ejemplos:
```
int a,b;
float y,x;
char c;
```

Declaración con inicialización: ejemplo:
```
int x=53, y=23;
```

Declaración de "arrays" unidimensionales (vectores): se encierra entre corchetes la dimensión del "array". El acceso a cada miembro del "array" se realiza mediante un índice que empieza en 0 (ese es el primer elemento del "array"). Ejemplos:

```
int vector1[12], vector2[15];
float precios[6];
char tira[20];
```

Una cadena de caracteres es en realidad un "array" unidimensional de caracteres con una señal fin de cadena \0. Para el ejemplo anterior, esa marca deja libres solo 19 caracteres y permite un tratamiento especial de la cadena por un grupo de funciones estándar (si el vector no lleva esa marca fin de cadena, hay que manipularlo carácter a carácter).

**- Comentarios.**

Van encerrados entre los símbolos /* y */, y pueden encontrarse en cualquier lugar del programa. Son ignorados por el compilador.

Ejemplo:
```
float a,b;  /* base y altura de un rectángulo */
```

**- Asignación.**

Asignación simple:
```
variable=expresión;
```
Asignación en tubería:
```
var1=var2=...=varN=expresión;
```

En esta última, la asignación se realiza de derecha a izquierda.

**- Instrucciones de control.**

**- Selección simple (if-else).**

```
if(expresión)
      instrucción1;
else
      instrucción2;
```

La parte ***else*** es opcional. Nótese que `if(expresión)` es equivalente a `if(expresión!=0)`. `instrucción1/instrucción2` puede ser una instrucción simple ó un bloque de instrucciones; a su vez, una de las instrucciones puede ser otra selección simple. En el caso de tener varias instrucciones ***if-else*** anidadas, hay que tener cuidado con las posibles ambigüedades sintácticas en el caso de que alguna instrucción ***if*** no lleve asociada la parte ***else*** correspondiente (utilizar llaves para evitar errores ya que el compilador asocia el ***else*** al ***if*** mas próximo).

| Ejemplo de ambigüedad sintáctica en las construcciones ***if-else*** anidadas | | |
|---|---|---|
| Pseudo-código | Implementación incorrecta | Implementación correcta |
| Si (c1) entonces<br>  Si (c2) entonces<br>  Si (c3) entonces<br>  instrucción3<br>  Fin_si<br>  Fin_si<br>Sino instrucción1<br>Fin_si | if (c1)<br>  if (c2)<br>  if (c3)<br>  instrucción3;<br>else instruccion1;<br>/* Si c1 es falso no se ejecuta nada; instrucción1 se ejecuta cuando c1 y c2 son verdaderas y c3 falsa*/ | if (c1){<br>  if (c2)<br>  if (c3)<br>  instrucción3;<br>}else instruccion1; |

**- Selección múltiple (switch).**

```
switch(expresión){            /* expresión entera     */
      case cte1:              /* se admiten caracteres*/
      case cte2:
      ...
      case cteN:  instrucción;
                  . . .
                  instrucción;
                  break;      /* para salir del switch */
      case cteN+1:
      ...
      case cteM:  instrucción;
                  . . .
                  instrucción;
                  break;
      . . .       . . .
      default:    instrucción;/* casos para los que no */
                  ...         /* hay una opción dada   */
                  instrucción;
                  break;
}
```

**- Repetición: while, for, do-while.**

```
while(expresión)                    do  instrucción;
  instrucción;                      while(expresión);
```

***while*** comprueba la expresión, y si el resultado es verdadero ejecuta instrucción; se sale de la repetición cuando expresión sea falsa (igual a 0).
***do-while*** ejecuta primero instrucción y luego comprueba la expresión; se sale de la repetición cuando expresión sea falsa.

```
for(exp1;exp2;exp3)
      instrucción;
```

`exp1`: inicialización. Puede haber más de una asignación separadas por comas.
`exp2`: condición de comprobación para volver a ejecutar otra iteración del bucle.
`exp3`: actualización. Puede haber más de una asignación separadas por comas.

Puede faltar cualquier parte del bucle, pero los separadores "**;**" no pueden faltar. El operador "**,**" se usa para separar las distintas asignaciones en `exp1` y `exp3`; es el operador de más baja prioridad y es asociativo de izquierda a derecha (no confundirlo con la coma de las funciones ó de las declaraciones).

**- Instrucciones break, continue, goto.**

***break***: obliga a salir de un bloque controlado por ***for***, ***while***, ***do-while*** ó ***switch***.
***continue***: pasa a la siguiente iteración (a la comparación en los bucles ***while*** y ***do-while***, y a la re-actualización en los bucles ***for***).
***goto***: transferencia incondicional del control. Sintaxis:

```
goto etiqueta;
...
etiqueta: instrucción;
```

**- printf y scanf.**

Son dos funciones para la entrada y salida por terminal. Forman parte de la biblioteca estándar y no forman parte de la definición del C (C no tiene funciones predefinidas). Su formato es:

```
printf("cadena de control",argumento1,argumento2,...);
scanf("cadena de control",argumento1,argumento2,...);
```

***printf*** se utiliza para la salida por pantalla y ***scanf*** para la entrada por teclado. La cadena de control va entre comillas y puede contener:

- Caracteres normales ASCII.
- Caracteres especiales (de control de salida ó de representación confundible):
  - `\n` nueva línea
  - `\t` tabulador
  - `\b` espacio atrás
  - `\r` retorno de carro
  - `\f` nueva página

| | |
|---|---|
| \0 | nulo, fin de cadena ("string") |
| \" | comillas |
| \' | apóstrofe |
| \\ | barra invertida |
| *** | carácter (valor ASCII en octal) |

- Secuencias de salida/entrada: indican un cierto argumento a escribir ó leer (correspondencia con los argumentos):

| | | |
|---|---|---|
| ***printf***: | %d | el dato es visualizado como un entero decimal con signo |
| | %i | entero decimal con signo |
| | %x | entero hexadecimal sin signo (sin el prefijo ***0x***) |
| | %o | entero octal sin signo (sin el cero inicial) |
| | %u | entero decimal sin signo |
| | %c | un solo carácter |
| | %s | una cadena de caracteres (\0 es el último carácter) |
| | %e | real en notación exponencial |
| | %f | real en notación decimal |
| | %g | %e ó %f, según mas corto |
| | %% | para escribir % |
| ***scanf***: | %d | el dato a leer es un entero decimal |
| | %u | entero decimal sin signo |
| | %o | entero octal sin signo |
| | %x | entero hexadecimal sin signo |
| | %i | entero decimal, octal o hexadecimal |
| | %c | un solo carácter |
| | %s | una cadena (***scanf*** lee palabras) |
| | %f | real |
| | %e | real |
| | %h | entero short |

Los argumentos de ***printf*** pueden ser constantes, variables y expresiones, y los de ***scanf*** han de ser variables. Al encontrar una secuencia de salida/entrada, se busca el siguiente argumento y se imprime/lee según se indica.

Ejemplos:
```
printf("v1=%d\nv2=%d\n",5+4,3*5);
```
resultado:
```
v1=9
v2=15
```
```
scanf(" %d",&ve);   /* leer por teclado un entero    */
scanf(" %c",&vc);   /* leer por teclado un carácter  */
scanf(" %f",&vf);   /* leer por teclado un real s.p. */
scanf(" %lf",&vd);  /* leer por teclado un real d.p  */
```

Observe que los argumentos de ***scanf*** han de ser variables. Cualquier variable de los tipos básicos ha de estar precedida por ***&*** (las cadenas de caracteres no lo necesitan).

Modificadores de expresiones de conversión:

Formateadores de ***printf***: %-n1.n2f, %-n1.n2s

| | |
|---|---|
| n1 | anchura mínima del campo |
| - | ajustar a la izquierda |
| n2 | reales: número de cifras decimales a la derecha del punto (6 por defecto)<br>cadenas de caracteres: número de caracteres a imprimir |

Formateadores de ***scanf***: `%*d, %nf`

- `*` no trata el correspondiente valor
- `n` longitud máxima que puede tener la lectura

Ejemplos de utilización de ***scanf*** para leer cadenas de caracteres con espacios en blanco:

`scanf(" %[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]s", linea);`

Asigna a la variable de cadena ***linea*** los caracteres introducidos por el dispositivo estándar de entrada, y finalizará la lectura cuando aparezca un carácter diferente de los encerrados entre corchetes.

`scanf(" %[^\n]s", linea);`

Los caracteres de dentro de los corchetes se interpretan como los caracteres de finalización de la asignación por teclado a la variable ***linea***. Nótese el espacio en blanco que precede a ***%*** (ignora los caracteres no deseados introducidos con anterioridad).

## 5.- Estilo.

C tiene formato de escritura libre, sin embargo es conveniente para facilitar la legibilidad del programa seguir las siguientes recomendaciones:

- Escribir una instrucción por línea.
- Separar bloques de declaración de bloques de acción mediante una línea en blanco.
- Las llaves de función van alineadas verticalmente, y las instrucciones internas tabuladas.
- En los bloques de las instrucciones de control, la llave de apertura se sitúa con la instrucción, el cuerpo del bloque se tabula (se sangra) con respecto a la instrucción y la llave de cierre se alinea con la instrucción.
- Escribir las palabras reservadas en minúsculas, así como los nombres de las variables (las mayúsculas quedan reservadas).
- Llenar el programa de comentarios.

# Anexo. Tabla de correspondencia entre pseudo-código y sintaxis de C.

**- Estructura del programa:**

| | **Pseudo-código** | Sintaxis de C |
|---|---|---|
| Cabecera | ALGORITMO nombre algoritmo | |
| Declaraciones de constantes | CONST n=5<br>Max=100<br>C=3.45 | #define n 5<br>#define Max 100<br>#define C 3.45 |
| Declaraciones de variables | VAR var1, var2:tipo1<br>var3, var4: tipo2 | int main(){<br>tipo1 var1, var2;<br>tipo2 var3, var4; |
| Instrucciones del algoritmo | INICIO instrucción 1<br>instrucción 2<br>. . .<br>instrucción N<br>FIN_ALGORITMO_PRINCIPAL | instrucción 1;<br>. . .<br>instrucción N;<br>return 0;<br>} |

**- Clases de instrucciones:**

| | **Pseudo-código** | Sintaxis de C |
|---|---|---|
| Comentarios | { comentario } | /* comentario */ |
| E/S por terminal | Leer(lista de variables)<br>Escribir(lista de expresiones) | #include <stdio.h><br>scanf("cadena control",lista de variables);<br>printf("cadena control",lista expresiones); |
| Secuencia de instrucciones | instrucción 1<br>instrucción 2<br>. . .<br>instrucción N | instrucción 1;<br>instrucción 2;<br>. . .<br>instrucción N; |
| Selectiva simple | SI condición<br>ENTONCES instrucción 1<br>instrucción 2<br>. . .<br>instrucción N<br>SINO instrucción N+1<br>instrucción N+2<br>. . .<br>instrucción P<br>FIN_SI | if (condición){<br>instrucción 1;<br>instrucción 2;<br>. . .<br>instrucción N;<br>}else { instrucción N+1;<br>instrucción N+2;<br>. . .<br>instrucción P;<br>} |
| Selectiva múltiple | SEGÚN SEA expresión_ordinal<br>HACER<br>$v_{11}$,$v_{12}$,..,$v_{1n}$: instrucciones 1<br>$v_{21}$,$v_{22}$,..,$v_{2n}$: instrucciones 2<br>. . .<br>SINO instrucciones N+1<br>FIN_SEGÚN_SEA | switch (expresión_ordinal){<br>case $v_{11}$:<br>case $v_{12}$:<br>. . .<br>case$v_{1n}$: instrucciones 1;<br>break;<br>case $v_{21}$:<br>case $v_{22}$:<br>. . .<br>case $v_{2n}$: instrucciones 2;<br>break;<br>. . .<br>default: instrucciones N+1;<br>break;<br>} |

| | | |
|---|---|---|
| Repetición DESDE | DESDE i←$v_{inicial}$ HASTA $v_{final}$ HACER<br>instrucción 1<br>instrucción 2<br>. . .<br>instrucción N<br>FIN_DESDE | for(i=$v_{inicial}$; i<=$v_{final}$; i++){<br>instrucción 1;<br>instrucción 2;<br>. . .<br>instrucción N;<br>} |
| Repetición MIENTRAS | MIENTRAS condición HACER<br>instrucción 1<br>instrucción 2<br>. . .<br>instrucción N<br>FIN_MIENTRAS | while (condición) {<br>instrucción 1;<br>instrucción 2;<br>. . .<br>instrucción N;<br>} |
| Repetición REPETIR HASTA QUE | REPETIR instrucción 1<br>instrucción 2<br>. . .<br>instrucción N<br>HASTA QUE condición | do{ instrucción 1;<br>instrucción 2;<br>. . .<br>instrucción N;<br>}while (!condición); |

**- Estructuras de datos:**

| | **Pseudo-código** | Sintaxis de C |
|---|---|---|
| Datos simples | entero | int<br>short int<br>unsigned int<br>long int |
| | real | float<br>long float<br>double |
| | caracter | char |
| | lógico o booleano | /* no existe: usar cualquier valor numérico (0: falso, distinto de cero: verdadero) */ |

**- Operadores:**

| | **Pseudo-código** | Sintaxis de C |
|---|---|---|
| Aritméticos | + - * /<br>mod (resto división entera) | + - * /<br>% |
| Asignación | ← | = |
| Relacionales | < ≤ > ≥<br>= (igual a) ≠ (distinto de) | < <= > >=<br>= = != |
| Conectivas lógicas | Y (“and”) O (“or”) | && \|\| |
| Negación lógica | NO (“not”) | ! |

**- La biblioteca estándar.**

| **Funciones de E/S** | | **#include <stdlib.h>** |
|---|---|---|
| system("cls"); | void | Borra la pantalla (ventana de texto de salida) y posiciona el cursor a la esquina superior izquierda |
| **Funciones de E/S** | | **#include <stdio.h>** |
| printf(...) | int | Escribe en pantalla. |
| scanf(...) | int | Lee datos del teclado. Devuelve el número de conceptos leídos (EOF si falla). |

| **Funciones de E/S** | | **#include <conio.h> (Biblioteca no estándar)** |
|---|---|---|
| getch() | int | Lectura de 1 carácter del teclado sin buffer.<br>Ejemplo: c=getch(); |
| **Funciones que manejan caracteres** | | **#include <ctype.h>** |
| toupper(c) | int | Convierte un carácter a mayúsculas.<br>Ejemplo: sal=toupper(ent); |
| **Funciones matemáticas** | | **#include <math.h>** |
| acos(d) | double | Arco coseno (0-pi) del argumento (-1,+1). |
| asin(d) | double | Arco seno (-pi/2,+pi/2) del argumento (-1,+1). |
| atan(d) | double | Arco tangente (-pi/2,+pi/2) del argumento. |
| atan2(d1,d2) | double | Devuelve el arco tangente de d1/d2. |
| ceil(d) | double | Devuelve el entero más pequeño mayor ó igual al argumento (redondeo hacia arriba). |
| cos(d) | double | Coseno del argumento expresado en radianes. |
| cosh(d) | double | Coseno hiperbólico del argumento. |
| exp(d) | double | Exponencial del argumento. |
| fabs(d) | double | Valor absoluto de un número real. |
| floor(d) | double | Redondeo hacia abajo (mayor entero menor ó igual al argumento). |
| fmod(d1,d2) | double | Devuelve el resto de d1/d2 con el mismo signo que d1. |
| labs(l) | long int | Valor absoluto de un entero largo. |
| log(d) | double | Logaritmo natural del argumento. |
| log10(d) | double | Logaritmo decimal del argumento. |
| pow(d1,d2) | double | Calcula d1 elevado a d2. |
| sin(d) | double | Seno del argumento expresado en radianes. |
| sinh(d) | double | Seno hiperbólico del argumento. |
| sqrt(d) | double | Raíz cuadrada positiva del argumento. |
| tan(d) | double | Tangente del argumento expresado en radianes. |
| tanh(d) | double | Tangente hiperbólica del argumento. |

Nota: la segunda columna indica el tipo de dato devuelto por la función. En la primera columna, los argumentos que aparecen son: c: carácter, d: doble precisión, s: cadena.

| | | | |
|---|---|---|---|
| **Asignatura** | Programación | | |
| **Plan de Estudios** | Grado en Ingeniero Mecánico, Eléctrico, Electrónico Industrial y Químico Industrial | | |
| **Actividad** | Trabajo individual | **Sesión** | 3 |
| **Tiempo empleado** | | | |

| Apellidos, nombre | DNI | Firma |
|---|---|---|
| | | |

**Resultados de la auto-evaluación**: anote los resultados de las diferentes ejecuciones de la prueba de auto-evaluación de la sintaxis de C. Si tras la primera ejecución no obtiene el resultado adecuado, se le recomienda vuelva a ejecutarla una segunda vez (tras volver a estudiar la sintaxis de C). Si tras esta segunda ejecución no alcanza una puntuación mínima del 75% (ó 25/33 respuestas correctas al primer intento), vuelva a repetir el proceso tras estudiar exhaustivamente la sintaxis del lenguaje de programación C.

| **Auto-evaluación** | | | |
|---|---|---|---|
| **Nº de sesión** | **Tiempo (mn)** | **Puntuación** | **Preguntas correctas primer intento** |
| 1 | | | |
| 2 | | | |
| 3 | | | |