Chuleta de lenguaje C

{MTI. Luis Alberto Mendoza San Juan}

Sintaxis "básica" de un programa en C

```
[ directivas_del_preprocesador> ]

int main()
{
   <bloque_de_instrucciones>
}
```

```
/* Programa: Hola mundo */
#include <conio.h>
#include <stdio.h>
int main()
{
   printf( "Hola mundo." );
   getch(); /* Pausa */
   return 0;
}
```

Tipos de instrucciones

Instrucción simple o elemental (de expresión):

```
<instrucción_de_expresión>;
```

Instrucción compuesta (alberga a un bloque de instrucciones):

```
{
   <bloque_de_instrucciones>
}
```

Comentarios en el código fuente de un programa

Se pueden escribir entre los caracteres *barra-asterisco* (`/*`) y *asterisco-barra* (`*/`).

Declaración de constantes simbólicas

```
#define <nombre> <secuencia>
```

- Ejemplo: **`#define PI 3.141592`**

Declaración de tipos enumerados

```
enum <tipo_de_dato> { <constante_1> [ = <valor_1> ], ..., <constante_n> [ = <valor_n> ] };
```

- Ejemplo: **`enum direcciones { NORTE = -2, SUR, ESTE, OESTE };`**

Declaración de variables

```
<tipo_de_dato> <nombre_de_variable> [ = <expresión> ];
```

- Ejemplo: **`int n = 35;`**

Directivas del preprocesador

`#define` Permite representar constantes simbólicas. Ejemplo: **`#define PI 3.141592`**
`#include` Permite incluir en el código fuente el contenido de archivos de cabecera. Ejemplo: **`#include <stdio.h>`**

Especificadores de formato

`%c` Carácter.
`%d` Número entero decimal con signo.
`%e` En **`printf`** muestra un número en notación científica con **`e`** minúscula. En **`scanf`** lee un número en coma flotante.
`%E` En **`printf`** muestra un número en notación científica con **`E`** mayúscula. En **`scanf`** lee un número en coma flotante.
`%f` Número real en coma flotante.
`%g` En **`printf`** muestra un número real en coma flotante **`f`** o en notación científica con **`e`** minúscula, en función de la magnitud del valor. En **`scanf`** lee un número en coma flotante.
`%G` En **`printf`** muestra un número real en coma flotante **`f`** o en notación científica con **`E`** mayúscula, en función de la magnitud del valor. En **`scanf`** lee un número en coma flotante.
`%i` En **`printf`** muestra un número entero decimal con signo. En **`scanf`** lee un número entero decimal, octal o hexadecimal con signo.
`%n` En **`printf`** puntero a entero donde se almacena el número de caracteres escritos hasta ese momento con **`printf`**. En **`scanf`** almacena el número de caracteres ya leídos.
`%o` Número entero octal sin signo.
`%p` Puntero (dirección de memoria).
`%u` Número entero decimal sin signo.
`%s` Cadena de caracteres.
`%x` Número entero hexadecimal sin signo con letras minúsculas.
`%X` Número entero hexadecimal sin signo con letras mayúsculas.
`%%` En **`printf`** muestra el carácter porcentaje (`%`). En **`scanf`** lee un carácter porcentaje.
`%[]` En **`scanf`** permite establecer un conjunto de exploración de caracteres que restringirán los caracteres que se podrán leer.

Funciones

fflush Permite vaciar (limpiar) el *buffer* del teclado: **fflush(stdin);**

getch Permite leer un carácter por teclado, sin eco por pantalla: **getch();**

main Contiene al bloque de instrucciones principal de un programa.

pow Permite realizar operaciones con potencias, devolviendo un valor de tipo **double** con independencia de que los operandos sean reales o enteros. Sintaxis: **pow(*<operando_número_base>*, *<operando_exponente>*)**

printf Permite llevar hacia la salida estándar (la pantalla) los valores (datos) obtenidos de la evaluación de una lista de argumentos. Sintaxis: **printf(*<cadena_de_control> [, <lista_de_argumentos>]*)**

scanf Permite asignar a una o más variables, uno o más valores (datos) recibidos desde la entrada estándar (el teclado). Sintaxis: **scanf(*<cadena_de_control>*, *<lista_de_argumentos>*)**

strcat Permite concatenar cadenas. Sintaxis: **strcat(*<cadena_destino>*, *<cadena_fuente>*)**

strcpy Permite asignar una expresión de cadena a un array de caracteres. Sintaxis: **strcpy(*<variable_destino>*, *<cadena_fuente>*)**

Identificadores (reglas de sintaxis)

1. Consta de uno o más caracteres.
2. El primer carácter debe ser una letra o el carácter *subrayado* "_", mientras que, todos los demás pueden ser letras, dígitos o el carácter *subrayado* "_". Las letras pueden ser minúsculas o mayúsculas del alfabeto inglés. Así pues, no está permitido el uso de las letras 'ñ' y 'Ñ'.
3. No pueden existir dos identificadores iguales, es decir, dos elementos de un programa no pueden nombrarse de la misma forma. Lo cual no quiere decir que un identificador no pueda aparecer más de una vez en un programa.

- Ejemplos de identificadores válidos en C: **Numero**, **dia_del_mes**, **T4**, **_ciudad**, **FJD**
- Ejemplos de identificadores no válidos en C: **3527**, **_ESTACIÓN_DE_TREN**, **informe***, **box de urgencias**, **año**

Instrucciones de control alternativas y repetitivas

Alternativa *múltiple* (**switch**):

```
switch ( <expresión> )
{
  case <expresión_1> : [ <bloque_de_instrucciones_1> ]
                       [ break; ]
  case <expresión_2> : [ <bloque_de_instrucciones_2> ]
                       [ break; ]
  ...
  case <expresión_n> : [ <bloque_de_instrucciones_n> ]
                       [ break; ]
          [ default : <bloque_de_instrucciones_n+1> ]
}
```

Alternativa *doble* (**if else**):

```
if ( <expresión_lógica> )
{
  <bloque_de_instrucciones_1>
}
else
{
  <bloque_de_instrucciones_2>
}
```

Alternativa *simple* (**if**):

```
if ( <expresión_lógica> )
{
  <bloque_de_instrucciones_1>
}
```

Repetitiva *mientras* (**while**):

```
while ( <expresión_lógica> )
{
  <bloque_de_instrucciones>
}
```

Repetitiva *hacer mientras* (**do while**):

```
do
{
  <bloque_de_instrucciones>
} while ( <expresión_lógica> );
```

Repetitiva *para* (**for**):

```
for ( <expresión_1> ; <expresión_2> ; <expresión_3> )
{
  <bloque_de_instrucciones>
}
```

Cómo elegir qué instrucción repetitiva utilizar:

- ***¿Se conoce, de antemano, el número de veces (iteraciones) que tiene que ejecutarse un determinado bloque de instrucciones?***

Si la respuesta es afirmativa, habitualmente se usa un bucle *para* (**for**). En caso contrario, se puede plantear la siguiente pregunta:

- ***¿El bloque de instrucciones debe ejecutarse al menos una vez?***

En este caso, si la respuesta es afirmativa, generalmente se hará uso de un bucle *hacer mientras* (**do while**), y si la respuesta es negativa, usaremos un bucle *mientras* (**while**).

Operadores (prioridad de operadores aritméticos, relacionales, lógicos, de asignación y otros)

`( ) [ ]`	Paréntesis y corchetes
`+ - ++ -- ! (<tipo>)`	Signo más, signo menos, incremento, decremento, negación (no) y conversión de tipo
`* / %`	Multiplicación, división y módulo
`+ -`	Suma y resta
`< <= > >=`	Menor que, menor o igual que, mayor que, mayor o igual que
`== !=`	Igual que y distinto que
`&&`	Conjunción (y)
`||`	Disyunción (o)
`= += -= *= /= %=`	Operadores de asignación

Palabras reservadas

`break` Instrucción de salto que interrumpe (rompe) la ejecución de un bucle o de una instrucción de control alternativa múltiple (**`switch`**).
`case` Caso de una instrucción de control alternativa múltiple (**`switch`**).
`char` Tipo de dato carácter.
`const` Cualificador que sirve para declarar una variable indicando que su valor es inalterable.
`continue` Instrucción de salto que interrumpe (rompe) la ejecución de un bucle.
`default` Caso por defecto en una instrucción de control alternativa múltiple (**`switch`**).
`double` Tipo de dato real.
`else` Si no, en una instrucción de control alternativa doble (**`if else`**).
`enum` Se utiliza para declarar tipos enumerados.
`float` Tipo de dato real.
`for` Instrucción de control repetitiva para.
`goto` Instrucción de salto que transfiere el control de un programa a la primera instrucción después de una etiqueta.
`if` Se emplea para escribir instrucciones de control alternativas simples (**`if`**) o dobles (**`if else`**).
`int` Tipo de dato entero (*integer*).
`long` Modificador de los tipos de datos **`int`** y **`double`**.
`return` Se usa para indicar el valor de retorno de una función.
`short` Modificador del tipo de dato **`int`**.
`signed` Modificador de los tipos de datos **`int`** y **`char`**.
`switch` Instrucción de control alternativa múltiple (según sea).
`unsigned` Modificador de los tipos de datos **`int`** y **`char`**.
`void` Tipo de dato sin valor (vacío).
`while` Se usa para escribir bucles mientras (**`while`**) y bucles hacer mientras (**`do while`**).

Secuencias de escape

`\a` (*Alerta*) Genera una alerta (campana).
`\b` (*Retroceso*) Mueve el cursor una posición hacia atrás.
`\f` (*Salto de página*) Mueve el cursor al principio de la página siguiente.
`\n` (*Nueva línea*) Mueve el cursor al principio de la línea siguiente.
`\r` (*Retorno de carro*) Mueve el cursor al principio de la línea actual.
`\t` (*Tabulador horizontal*) Mueve el cursor a la posición siguiente del tabulador horizontal.
`\v` (*Tabulador vertical*) Mueve el cursor a la posición siguiente del tabulador vertical.
`\"` (*Comilla doble*) Muestra el carácter comilla doble.
`\'` (*Comilla simple*) Muestra el carácter comilla simple.
`\?` (*Interrogación*) Muestra el carácter de interrogación.
`\\` (*Barra invertida*) Muestra el carácter barra invertida.
`\ooo` (*Constante octal*) Representa al carácter ASCII correspondiente a la constante octal (**`ooo`**) que se indique.
`\xhh` (*Constante hexadecimal*) Representa el carácter ASCII correspondiente a la constante hexadecimal (**`hh`**) que se indique.

Tipos de datos básicos y modificadores (combinaciones, tamaños más típicos en bits y rangos mínimos)

Tipos de datos básicos:

- Número entero (`int`).
- Número real (`float` y `double`).
- Carácter (`char`).
- Sin valor (`void`).

Modificadores de los tipos de datos básicos:

- `signed` (aplicable a `int` y `char`).
- `unsigned` (aplicable a `int` y `char`).
- `long` (aplicable a `int` y `double`).
- `short` (aplicable a `int`).

`char` *(8 bits) -127 a 127*
`unsigned char` *(8 bits) 0 a 255*
`signed char` (*8 bits*) -127 a 127
`int` (*16 o 32 bits*) -32.767 a 32.767
`unsigned int` (*16 o 32 bits*) 0 a 65.535
`signed int` (*16 o 32 bits*) -32.767 a 32.767
`short int` (*16 bits*) -32.767 a 32.767
`unsigned short int` (*16 bits*) 0 a 65.535
`signed short int` (*16 bits*) -32.767 a 32.767
`long int` (*32 bits*) -2.147.483.647 a 2.147.483.647
`unsigned long int` (*32 bits*) 0 a 4.294.967.295
`signed long int` (*32 bits*) -2.147.483.647 a 2.147.483.647
`float` (*32 bits*) 1E-37 a 1E+37 con seis dígitos de precisión
`double` (*64 bits*) 1E-37 a 1E+37 con diez dígitos de precisión
`long double` (*80 bits*) 1E-37 a 1E+37 con diez dígitos de precisión