El lenguaje de programación C
- Identificadores y variables –

Identificadores y caracteres especiales

- Un **identificador** puede estar compuesto de cualquier combinación de:
  - letras (minúsculas y mayúsculas)
    - Salvo letras con tilde, ñ, ç, …
  - dígitos
  - el símbolo subrayado '_'.
- La única restricción es que el primer carácter no puede ser un dígito
- No se limita la longitud de los identificadores
  - Algunas implementaciones sólo reconocen los 8 primeros y otras (ANSI) los 31 primeros caracteres.
- *Case sensitive*: Hay diferencia entre mayúsculas y minúsculas
- Existe un conjunto de caracteres que tienen un significado especial:

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

Ejemplos de identificadores

Válidos	No válidos	Razón
`x`	`4num`	1er cararcter numérico
`y2`	`"x"`	Caracter ilegal “
`suma_1`	`orden-no`	Caracter ilegal ‘-’
`_t`	`indice lis`	Espacio ilegal
`TABLA`	`número`	No se admiten tildes

Palabras reservadas

- Existen una serie de palabras que usa el propio lenguaje
  - No pueden ser usadas en los identificadores
- Lista:

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

Variables

- Una variable:
  - Se identifica a través de su nombre (identificador) y
  - Es capaz de almacenar un valor (entero, real, texto, lógico, …)
- Manejando variables somos capaces de manejar datos
- En C cada variable tiene un tipo asociado que define:
  - El espacio que ocupa en memoria
  - Las operaciones que podemos hacer con ella

Variables de tipo entero…

Nombre	¿Qué representa?	Otro
`char`	Un **carácter**	
`short int`	Un **entero** corto	`short`
`int`	Un **entero**	
`long int`	Un **entero** con **mayor rango** de validez	`long`

`unsigned`

- `unsigned int` se puede abreviar como `unsigned`

Variables de tipo entero - Tamaños

`unsigned`

Nombre	¿Qué representa?	Otro
`char`	Un **carácter**	
`short int`	Un **entero** corto	`short`
`int`	Un **entero**	
`long int`	Un **entero** con **mayor rango** de validez	`long`

- Existe un operador, `sizeof`, que nos dice cual es el tamaño de un determinado tipo en bytes
- Lo único que el **C** garantiza es:

```
sizeof(char) = 1
sizeof(short) <= sizeof(int) <= sizeof(long)
sizeof(unsigned) = sizeof(int)
```

Variables de tipo entero

Nombre	¿Qué representa?	Otro
`char`	Un **carácter**	
`short int`	Un **entero** corto	`short`
`int`	Un **entero**	
`long int`	Un **entero** con **mayor rango** de validez	`long`

`unsigned`

- El tipo **`char`** guarda tanto un caracter como un número
  - Es el único que tiene **garantizado su tamaño: 1 byte**…
  - … es decir 8 bits...
  - … con lo cual puede almacenar 2^8=256 valores…
  - … es decir valores en el rango **-128 a 127**…
  - … o, si es `unsigned`, de **0 a 255**

Variables de tipo entero – short int

Nombre	¿Qué representa?	Otro
`char`	Un **carácter**	
`short int`	Un **entero** corto	`short`
`int`	Un **entero**	
`long int`	Un **entero** con **mayor rango** de validez	`long`

`unsigned`

- El tipo **`short int`** ocupa 2 bytes = 16 bits
  - Por tanto es capaz de almacenar 2^{16}=**65.536** valores…
  - … en el rango **-32.768 a 32.767**…
  - … salvo que pongamos delante la palabra **`unsigned`**…
  - … en cuyo caso su rango irá de **0 a 65.535**

Variables de tipo entero - int

Nombre	¿Qué representa?	Otro
`char`	Un **carácter**	
`short int`	Un **entero** corto	`short`
`int`	Un **entero**	
`long int`	Un **entero** con **mayor rango** de validez	`long`

`unsigned`

- El tipo entero principal es **`int`**
  - En gcc ocupa **4 bytes** = 32 bits (a veces ocupa 2 bytes)
  - Por tanto es capaz de almacenar 2^{32}=4.294.967.296 valores…
  - … es decir valores entre **-2,147,483.648 y 2,147,483.647** …
  - … o, si es `unsigned`, de **0 a 4.294.967.295**

Variables de tipo entero – long int

	Nombre	¿Qué representa?	Otro
unsigned	`char`	Un **carácter**	
unsigned	`short int`	Un **entero** corto	`short`
unsigned	`int`	Un **entero**	
unsigned	`long int`	Un **entero** con **mayor rango** de validez	`long`

- El tipo entero más largo es **`long int`**
  - En gcc y ubuntu ocupa **8 bytes**
  - Por tanto es capaz de almacenar 2^{64} ~$18{,}446 \times 10^{18}$ valores…

Variables de tipo entero – long long

	Nombre	¿Qué representa?	Otro
unsigned	`char`	Un **carácter**	
unsigned	`short int`	Un **entero** corto	`short`
unsigned	`int`	`Un` **entero**	
unsigned	`long int`	Un **entero** con **mayor rango** de validez	`long`
	`long long int`	Un **entero** con aun **mayor rango** de validez	

- Hay un tipo no siempre disponible llamado: **`long long int`**
  - En gcc ocupa **8 bytes** = 64 bits → igual que **`int`**
  - Por tanto es capaz de almacenar 2^{64} ~$18{,}446 \times 10^{18}$ valores

Variables de tipo entero - Reales

Nombre	¿Qué representa?	Prec.
`float`	Un **real** de precisión simple	6 dígitos
`double`	Un **real** de precisión doble	15 dígitos
`long double`	Un **real** de precisión mayor y con mayor rango	18 dígitos

- El tipo real básico es **`float`** (4 bytes)
  - Rango: $\pm 1.17549 \times 10^{-38}$ – $\pm 3.40282 \times 10^{38}$.
- El tipo real de doble precisión es **`double`** (8 bytes)
  - Rango: $\pm 2.22507 \times 10^{-308}$ – $\pm 1.79769 \times 10^{308}$.
- El tipo **`long double`** es un real de doble precisión y mayor o igual rango (16 bytes)
  - Rango: $\pm 3.3621 \times 10^{-4932}$ – $\pm 1.1897 \times 10^{4932}$.

Tipos fundamentales: Tabla resumen

	Nombre	¿Qué representa?	Tipo
`unsigned`	`char`	Un **carácter**	Entero
`unsigned`	`short int`	Un **entero** corto	Entero
`unsigned`	`int`	Un **entero**	Entero
`unsigned`	`long int`	Un **entero** con **mayor rango** de validez	Entero
	`float`	Un **real**	Real
	`double`	Un **real** de **doble precisión**	Real
	`long double`	Un **real** de **doble precisión** y **mayor**	Real

Declarando variables

- Una variable se declara indicando su tipo seguido de su identificador
  - Se puede inicializar en la misma línea ← recomendable
- El lugar donde lo declaremos define su ámbito

```
#include <stdio.h>

/* Declaracion e inicializacion de una variable global
llamada "enteroglobal". Podemos usarla en cualquier parte */
int enteroglobal = 1;

int main() {
  /*Declaracion de una variable entera llamada "numero".
    Sin inicializar. Peligroso!*/
  int numero;
  /*Un entero sin signo llamado "diez"*/
  unsigned diez = 10;
  return 0;
}
```

Mostrando variables

- Para imprimir una variable usamos una nueva forma de **`printf()`**
  - En el texto pondremos, donde queremos mostrar cada variable, el símbolo `%` seguido de una o dos letras
    - Estas letras especifican el tipo de cada variable
  - A continuación, separados por comas pondremos en el orden correcto las variables.

```
#include <stdio.h>

int main() {
  int var1 = -1;
  unsigned var2 = 2;
  printf("La variable 1 vale %i, y la 2 vale %u", var1, var2);
  return 0;
}
```

Mostrando variables - Modificadores

Tipo	Mod.	Observaciones
Entero (`int`)	%d, %i	Idénticos
`char`	%c	Si se usa %d o %i se imprime su equivalente ASCII
`unsigned`	%u	
`short ...`	%hi, %hd %hu	→ Para `short int` → Para `unsigned short int`
`long ...`	%li, %ld %lu	→ Para `long int` → Para `unsigned long int`
Real (`float` o `double`)	%e, %E %f %g, %G	→ Notación científica. Ej. 1.234567e+01 → 6 decimales tras la coma → 6 cifras significativas
`Long double`	%Le, %LE %Lf,%LF %Lg, %LG	Con las mismas características que arriba

Mostrando variables - Ejemplo

```
#include <stdio.h>

int main() {
  int aint = -130;
  printf("Int (i): %i\n", aint);
  printf("Int (d): %d\n", aint);

  short int ashortint = 10;
  printf("Short Int: %hi\n", ashortint);

  unsigned int aunsignedint = 10;
  printf("Unsigned Int: %u\n", aunsignedint);
  printf("Unsigned Int?: %u\n", aint);

  float afloat = 12.34567890;
  printf("Float (e): %e\n", afloat);
…
}
```

Leyendo variables

- Para leer una variable de tipo entero usamos la función **`scanf()`**
- Toma dos argumentos
  - Una cadena de caracteres con un **`%`** seguido del correspondiente modificador
  - La variable en la que queremos almacenar el valor leído precedida del símbolo **`&`**
- Recordad declarar la variable antes de usarla!

```
#include <stdio.h>

int main() {
  int entero; /*Declaramos la variable*/
  scanf("%d", &entero); /*Leemos un valor en la variable*/
  printf("El valor que hemos leido es %d", entero);
}
```