Tipos de datos en C

Diego Iván Oliveros Acosta

@scalapp.co

Tipo de datos especificado

- Como se explica en la semana anterior **Variables** , una variable en C debe ser un tipo de datos especificado y se debe usar un especificador de formato dentro de la función printf() para mostrarla:

```
// Create variables
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
char myLetter = 'D';         // Character

// Print variables
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
```

Tipos de datos básicos

Data Type	Size	Description	Example
int	2 or 4 bytes	Stores whole numbers, without decimals	1
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits	1.99
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits	1.99
char	1 byte	Stores a single character/letter/number, or ASCII values	'A'

- El tipo de datos especifica el tamaño y el tipo de información que almacenará la variable. En este curso nos centraremos en los más básicos:

Especificadores de formato básicos

- Existen distintos especificadores de formato para cada tipo de datos. A continuación, se indican algunos de ellos:
- Es importante que utilice el especificador de formato correcto para el tipo de datos especificado, o el programa puede producir errores o incluso bloquearse.
- https://youtu.be/gp_D8r-2hwk

Tipos de datos de **caracteres** C

El tipo char

```
char myGrade = 'A';
printf("%c", myGrade);
```

El tipo de datos char se utiliza para almacenar un solo carácter.

El carácter debe estar rodeado por comillas simples, como 'A' o 'c', y usamos el especificador de formato %c para imprimirlo:

Como alternativa, si está familiarizado con ASCII, puede utilizar valores ASCII para mostrar determinados caracteres. Tenga en cuenta que estos valores no están entre comillas simples (' '), ya que son números:

Notas sobre los personajes

- **Consejo:** imprima una lista de todos los valores ASCII en la Referencia web de tabla ASCII.
- Si intenta almacenar más de un carácter, solo imprimirá el último carácter:
- **Nota:** No utilice el tipo char para almacenar **varios** caracteres, ya que puede producir errores.
- Por ahora, solo sepa que usamos cadenas para almacenar múltiples caracteres/texto, y el tipo char para caracteres individuales.

```
char a = 65, b = 66, c = 67;
printf("%c", a);
printf("%c", b);
printf("%c", c);

char myText[] = "Hello";
printf("%s", myText);
```

Tipos de datos numéricos de C

- Use **int** cuando necesite almacenar un número entero sin decimales, como 35 o 1000, o **float** cuando necesite un número double o de punto flotante (con decimales), como 9,99 o 3,14515.

```
int myNum = 1000;
printf("%d", myNum);
float myNum = 5.75;
printf("%f", myNum);
double myNum = 19.99;
printf("%lf", myNum);
```

Float vs.double

La precisión de un valor de punto flotante

Indica cuántos dígitos puede tener el valor después del punto decimal.

- La precisión de floates de **seis o siete dígitos decimales**, mientras que con double las variables tienen una precisión de aproximadamente **15 dígitos.**
- Por lo tanto, suele ser más seguro utilizar un double para la mayoría de los cálculos, pero tenga en cuenta que ocupa el doble de memoria
- **(float 8 bytes frente a 4 bytes).**

Números científicos

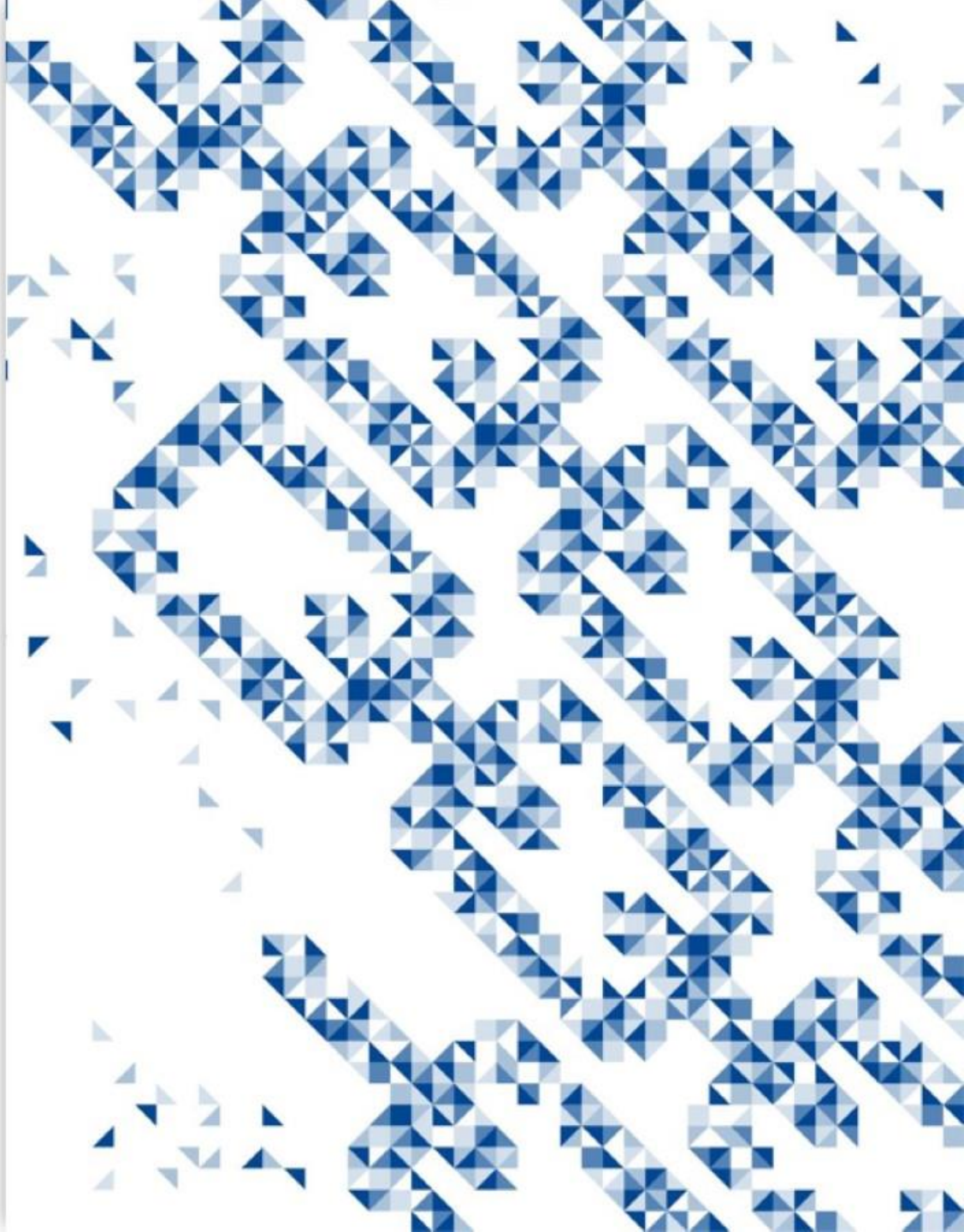
Un número de punto flotante también puede ser un número científico con una "e" para indicar la potencia de 10:

```
float f1 = 35e3;
double d1 = 12E4;

printf("%f\n", f1);
printf("%lf", d1);
```

C Precisión decimal

- **Establecer precisión decimal**
- Probablemente ya hayas notado que, si imprimes un número de punto flotante, la salida mostrará muchos dígitos después del punto decimal.
- Si desea eliminar los ceros adicionales (establecer precisión decimal), puede usar un punto (.) seguido de un número que especifique cuántos dígitos deben mostrarse después del punto decimal.



Establecer precisión decimal

```c
float myFloatNum = 3.5;
double myDoubleNum = 19.99;

printf("%f\n", myFloatNum); // Outputs 3.500000
printf("%lf", myDoubleNum); // Outputs 19.990000

float myFloatNum = 3.5;

printf("%f\n", myFloatNum);   // Default will show 6 digits after
                                 the decimal point
printf("%.1f\n", myFloatNum); // Only show 1 digit
printf("%.2f\n", myFloatNum); // Only show 2 digits
printf("%.4f", myFloatNum);   // Only show 4 digits
```

El operador sizeof en C

- En la session anterior presentamos que el tamaño de la memoria de una variable varía según el tipo.
- El tamaño de la memoria se refiere a cuánto espacio ocupa un tipo en la *memoria de la computadora* .
- Para obtener realmente el tamaño (en bytes) de un tipo de datos o variable, utilice el sizeofoperador

Data Type	Size
int	2 or 4 bytes
float	4 bytes
double	8 bytes
char	1 byte

Especificador %lu (long unsigned)

- Tenga en cuenta que usamos el de especificador formato %lu para imprimir el resultado, en lugar de %d.
- Esto se debe a que el compilador espera que el operador sizeof devuelva un long unsigned int(%lu), en lugar de int(%d).
- En algunas computadoras, podría funcionar con %d, pero es más seguro usar %lu.

```
int myInt;
float myFloat;
double myDouble;
char myChar;

printf("%lu\n", sizeof(myInt));
printf("%lu\n", sizeof(myFloat));
printf("%lu\n", sizeof(myDouble));
printf("%lu\n", sizeof(myChar));
```

¿Por qué debería saber el tamaño de los tipos de datos?

- Conocer el tamaño de los diferentes tipos de datos es importante porque dice algo sobre el uso y el rendimiento de la memoria.
- Por ejemplo, el tamaño de un tipo char es 1 byte, lo que significa que si tienes una matriz de 1000 valores char, ocupará 1000 bytes (1 KB) de memoria.
- Utilizar el tipo de datos correcto para el propósito correcto ahorrará memoria y mejorará el rendimiento de su programa.
- Aprenderá más sobre el operador **sizeof** más adelante en este curso de programación y cómo usarlo en diferentes escenarios.

Ejemplos de tipos de datos en C

Veamos un ejemplo de un comercio, un caso de la vida real, en el que se muestra un ejemplo del uso de diferentes tipos de datos para calcular y generar el costo total de varios artículos.

```c
#include <stdio.h>

int main() {
  // Create variables of different data types
  int items = 50;
  float cost_per_item = 9.99;
  float total_cost = items * cost_per_item;
  char currency = '$';

  // Print variables
  printf("Number of items: %d\n", items);
  printf("Cost per item: %.2f %c\n", cost_per_item, currency);
  printf("Total cost = %.2f %c\n", total_cost, currency);

  return 0;
}
```

Conversión de tipos en C

- Hay dos tipos de conversión en C:
    - **Conversión implícita** (automática)
    - **Conversión explícita** (manual)

A veces, es necesario convertir el valor de un tipo de datos a otro tipo:

- Por ejemplo, si intenta dividir dos números enteros 5por 2, esperaría que el resultado fuera 2.5. Pero como estamos trabajando con números enteros (y no con valores de punto flotante), el siguiente ejemplo solo mostrará 2.
- Para obtener el resultado correcto, necesitas saber cómo funciona **la conversión de tipos** .

```c
#include <stdio.h>

int main() {
  int x = 5;
  int y = 2;
  int sum = 5 / 2;

  printf("%d", sum);
  return 0;
}
```

Conversión implícita

- La conversión implícita la realiza automáticamente el compilador cuando se asigna un valor de un tipo a otro.
- Por ejemplo, si asigna un valor int a un tipo float.
- En el ejemplo, el compilador convierte automáticamente el valor int 9 en un valor float de 9.000000.
- Esto puede ser riesgoso, ya que podría perder el control sobre valores específicos en determinadas situaciones.
- Especialmente si fuera al revés.

```
// Automatic conversion: int to float
float myFloat = 9;

printf("%f", myFloat); // 9.000000
```

¿Qué pasó con .99?

```
//En el siguiente ejemplo convierte automáticamente el valor flotante 9.99 en un valor int de 9:
// Automatic conversion: float to int
int myInt = 9.99;
printf("%d", myInt); // 9
```

- ¡Quizás queramos esos datos en nuestro programa!
- Así que tenga cuidado.
- Es importante que sepa cómo funciona el compilador en estas situaciones para evitar resultados inesperados.

Ejemplo 2:

- Si divides dos números enteros: 5 por 2, sabes que la variable suma es 2.5.
- Y como sabes desde el principio de este curso, si almacenas la variable como un número entero, el resultado solo mostrará el número 2.
- Por lo tanto, sería mejor almacenar la suma como a float o a double, ¿verdad?
- ¿Por qué el resultado es 2.00000 y no 2.5?

```
/* Bueno, es porque 5 y 2
siguen siendo números
enteros en la división.
En este caso, debe
convertir manualmente los
valores enteros a valores
de punto flotante */

float sum = 5 / 2;
printf("%f", sum);
// 2.000000
```

Conversión explícita

- La conversión explícita se realiza manualmente colocando el tipo entre paréntesis () delante del valor.

- Considerando nuestro problema del ejemplo anterior, ahora podemos obtener el resultado correcto

```
// Manual conversion:
// int to float
float sum = (float) 5 / 2;

printf("%f", sum);
// 2.500000

//float sum = 5.0 / 2.0;
```

También puedes colocar el tipo delante de una variable (Casting) o "precisión decimal" eliminando los ceros adicionales “en el caso que aplique”:

```c
int num1 = 5;
int num2 = 2;
float sum = (float) num1 / num2;

printf("%f", sum); // 2.500000
printf("%.1f", sum); // 2.5
```

Ejemplo de la vida real:

```c
/* A continuación, se muestra un ejemplo real de tipos de datos y conversión de tipos en el que creamos un programa para calcular el porcentaje de la puntuación de un usuario en relación con la puntuación máxima en un juego:*/

// Set the maximum possible score in the game to 500
int maxScore = 500;

// The actual score of the user
int userScore = 423;

/* Calculate the percantage of the user's score in relation to the maximum available score.
Convert userScore to float to make sure that the division is accurate */
float percentage = (float) userScore / maxScore * 100.0;

// Print the percentage
printf("User's percentage is %.2f", percentage);
```