[Γα=Ω5]

Tipos de Datos y Casting

Por Ariel Parra

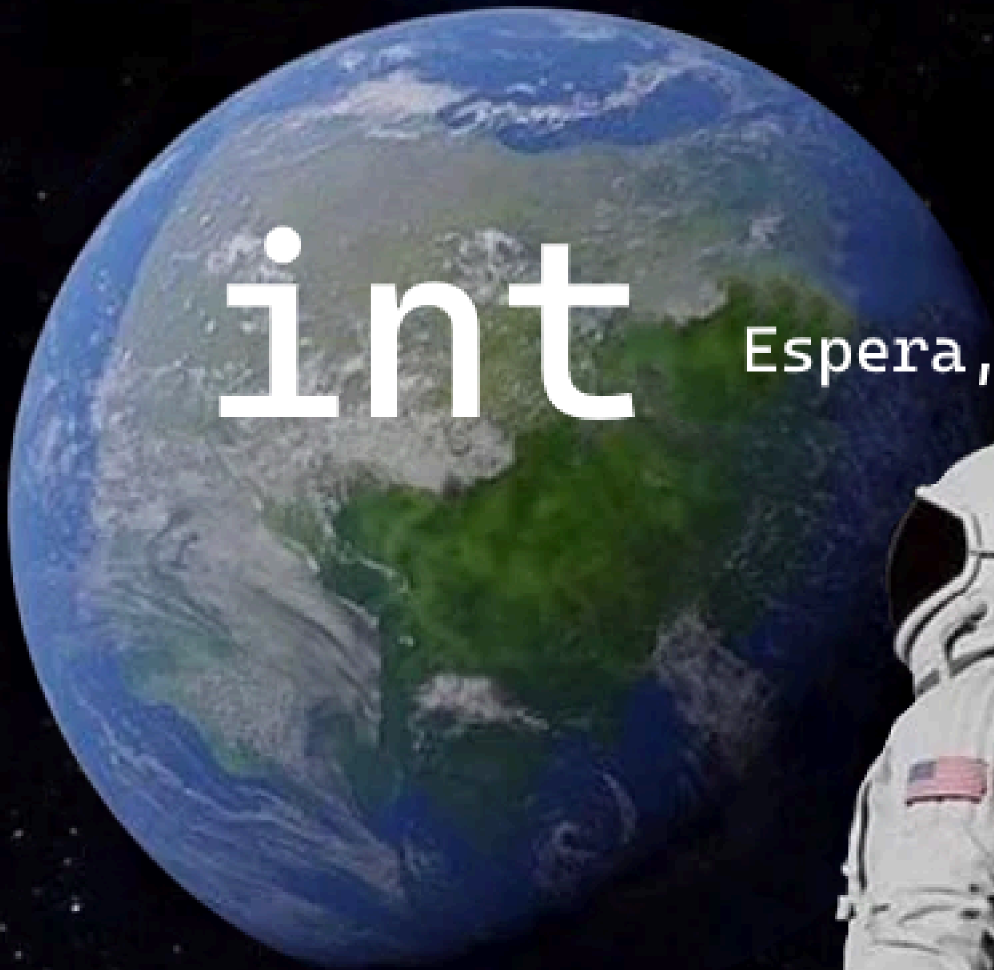
Tipos Primitivos

Tipo	librería	Tamaño	Rango (GCC x64)	Descripción
char	<cstdint>	8 bits	-128 a 127	Almacena un carácter en comillas simples ''
bool	<cstdbool>	8 bits	true (1) o false (0)	Valor booleano (verdadero o falso).
short	<cstdint>	16 bits	-32,768 a 32,767	Entero corto.
int	<cstdint>	32 bits	-2,147,483,648 a 2,147,483,647	Entero estándar.
long long	<cstdint>	64 bits	-9.22e18 a 9.22e18	Entero largo extendido.
float	<cfloat>	32 bits	Aprox ±3.4e38	Número de punto flotante de precisión simple.
double	<cfloat>	64 bits	Aprox ±1.7e308	Número de punto flotante de precisión doble.

Sinonimos / Alias en

- **1 byte**: `uint8_t`, `unsigned char`, `int8_t`, `signed char`, `int_least8_t`, `uint_least8_t`, `int_fast8_t`, `uint_fast8_t`
- **2 bytes**: `int16_t`, `short`, `short int`, `signed short int`, `uint16_t`, `unsigned short`, `unsigned short int`, `int_least16_t`, `uint_least16_t`
- **4 bytes**: `int32_t`, `int`, `signed int`, `uint32_t`, `unsigned int`, `int_least32_t`, `uint_least32_t`, `int_fast16_t`, `uint_fast16_t`, `int_fast32_t`, `uint_fast32_t`
- **8 bytes**: `int64_t`, `long long`, `signed long long`, `uint64_t`, `unsigned long long`, `int_least64_t`, `uint_least64_t`, `int_fast64_t`, `uint_fast64_t`, `intmax_t`, `uintmax_t`

> Las conversiones de tipos de datos primitivos dependen de la implementación del compilador (GCC, Clang, MSVC, etc.) como también de la arquitectura del procesador (x86, x64, ARM, etc.).



Espera,¿Todo es int?

Siempre lo ha sido

calificadores (type qualifier)

Los calificadores en C/C++ modifican el comportamiento de las variables.

- **const**: Define que una variable es constante y no puede ser modificada después de su inicialización, por lo que sera el que usaremos en las competencias. Ejemplo:

```
const int constante = 100;
```

- **static**: Tiene diferentes usos según el contexto:
    - Para **variables globales y funciones**, restringe el acceso al archivo en el que se define, limitando su alcance a ese archivo.
    - Para **variables locales**, significa que la variable conserva su valor entre llamadas a la función, y se inicializa solo una vez. Es decir, la variable tiene una duración estática. Ejemplo:

```
void contador() {
    static int cuenta = 0;
    cuenta++;
}
```

- **unsigned**: Indica que una variable solo puede almacenar valores no negativos. Ejemplo:

```
unsigned long long num = 10e9;
```

- **signed**: Indica que una variable puede almacenar valores tanto negativos como positivos (redundante al ser el calificador por defecto).
- **long**: en si long es un tipo de dato, pero al ponerse antes de otro tipo se combierte en un type qualifier se suele usar en `long long` y `long double`.
- **inline**: Solo para funciones, sugiere al compilador que expanda la función en el lugar donde se llama, en lugar de realizar una llamada normal a la función. Esto puede mejorar el rendimiento en algunos casos.
- ***** El operador estrella (*) se utiliza para declarar punteros al ponerlo posterior a un tipo primitivo. por ejemlo `int *ptr;`
- **volatile**: Indica que el valor de una variable puede cambiar en cualquier momento, sin que el programa la modifique explícitamente (no lo usaremos).

Tipos especiales

- **auto**: Introducido en C++11, `auto` permite al compilador deducir automáticamente el tipo de una variable a partir de su valor de inicialización. Esto simplifica la escritura del código y evita errores de tipo. Ejemplo:

```
auto numero = 10; // 'numero' sera int
auto pi = 3.14;   // 'pi' sera double
```

- **size_t**: es un tipo garantizado por el compilador para contener cualquier índice (en este caso: unsigned int).
- **std::byte**: Introducido en C++17 y definido en `<cstddef>` es un tipo de dato de 8 bits que **no permite** operaciones aritméticas.
- **__int128**: Es un tipo de dato especial que puede albergar 128 bits. Está disponible en compiladores GCC y Clang y puede manejar valores muy grandes, pero es más lento y requiere buffers para su uso práctico. Su rango con signo es de -1.70e38 a 1.70e38 y sin signo de 0 a aproximadamente 3.40e38.
- **void**: Cuando se usa en la declaración de una función indica que la función no devuelve ningún valor. También se utiliza para declarar punteros genéricos (`void*`), que pueden apuntar a cualquier tipo de datos.
- **NULL**: Representa una constante de puntero nulo en C.
- **nullptr**: Introducido en C++11, representa una constante de puntero nulo de forma segura en C++.

Declaración de literales

Infijo

- **Exponencial**: Notación científica, se utiliza la letra `e` después del número base y antes del exponente.

```
double valor = 1.0e7; // 1 * 10^7
```

Prefijos

- **Hexadecimal**: Prefijo `0x`.

```
int valor = 0x10; // 16 en decimal
```

- **Binario**: Prefijo `0b` (A partir de C++14).

```
int valor = 0b1011; // 11 en decimal
```

- **Octal**: Prefijo con `0`.

```
int valor = 010; // 8 en decimal
```

Postfijos

Los postfijos se utilizan para especificar el tipo del literal, permitiendo que el compilador entienda el tipo exacto. Aunque esto suele terminar siendo redundante.

- **l** o **L**: Indica que el literal es de tipo long o long long.

```
long valor = 10L; // Literal long
long long valor_ll = 10LL; // Literal long long
```

- **f** y **d**: Indica que el literal es de tipo float o double.

```
float valor = 10.5f; // Literal float
double valor = 10.5; // Literal double
```

- **u**: Indica que el literal es de tipo unsigned y se puede combinar con otros postfijos para tipos más específicos.

```
unsigned int valor = 10u; // Literal unsigned int
unsigned long long valor = 10ULL; // Literal unsigned long long
```

Casting en C

El casting o casteo es el proceso de conversión de un tipo de dato a otro y existen dos tipos:

- **Casting Implícito**
  C realiza automáticamente algunas conversiones entre tipos de datos, conocidas como casting implícito.

```c
int i = 10;
long long j = i; // i paso de int a long long
```

- **Casting Explícito**
  Para convertir un tipo de dato a otro de manera más controlada, se utiliza el casting explícito. Esto se hace mediante el uso de paréntesis con el tipo de destino antes de la expresión a convertir:

```c
int a = 15, b = 7;
float b = (float)(a / b);
```

Casting en C++

- **static_cast**: este es el que usaremos en el curso debido a que los demás son más relacionados a la programación orientada a objetos y/o manejo de memoria.

```
int a = 15, b = 7;
double b = static_cast<double>(a / b);
```

- **dynamic_cast**: Utilizado principalmente con punteros y referencias a clases polimórficas para convertir hacia abajo en la jerarquía de herencia.
- **const_cast**: Permite agregar o eliminar const de una variable.
- **reinterpret_cast**: Para conversiones de bajo nivel, como convertir entre punteros de tipos no relacionados.

Límites de cada tipo

`<climits>`

Esta librería de C incluye constantes para los límites de cada tipo:

`CHAR_BIT`, `SCHAR_MIN`, `SCHAR_MAX`, `UCHAR_MAX`, `CHAR_MIN`, `CHAR_MAX`, `MB_LEN_MAX`, `SHRT_MIN`, `SHRT_MAX`, `USHRT_MAX`, `INT_MIN`, `INT_MAX`, `UINT_MAX`, `LONG_MIN`, `LONG_MAX`, `ULONG_MAX`, `LLONG_MIN`, `LLONG_MAX`, `ULLONG_MAX`

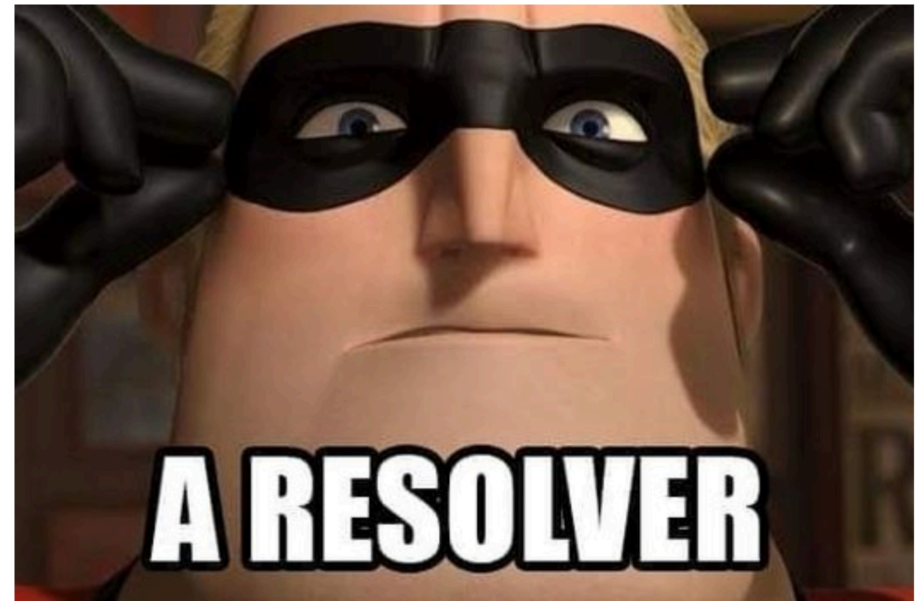
`<limits>`

A diferencia de `<climits>`, esta librería de C++ incluye funciones para manejar los límites de cada tipo:

- Valor mínimo para cualquier tipo primitivo `T`: `std::numeric_limits<T>::min()`
- Valor máximo para cualquier tipo primitivo `T`: `std::numeric_limits<T>::max()`

Problemas

- **677A** Vanya and Fence↗
- **791A** Bear and Big Brother↗



Referencias

- Agarwal, H. (2023). *C++ Data Types*. Recuperado de https://www.geeksforgeeks.org/cpp-data-types/ ⤴
- bug8wdqo. (2024). *Type Qualifiers in C++*. Recuperado de https://www.geeksforgeeks.org/type-qualifiers-in-cpp/ ⤴
- cplusplus. (2023). *< climits > (limits.h)*. Recuperado de https://cplusplus.com/reference/climits/ ⤴
- GeeksforGeeks. (2023). *Data Types in C*. Recuperado de https://www.geeksforgeeks.org/data-types-in-c/ ⤴
- GNU. (2024). *128-bit Integers*. Recuperado de https://gcc.gnu.org/onlinedocs/gcc/_005f_005fint128.html ⤴
- kushagra. (2018). *“static const” vs “#define” vs “enum”*. Recuperado de https://www.geeksforgeeks.org/static-const-vs-define-vs-enum/ ⤴

- Microsoft. (2023). *Standard types*. Recuperado de https://learn.microsoft.com/en-us/cpp/c-runtime-library/standard-types?view=msvc-170 ↩
- Microsoft. (2024). *Data types ranges*. Recuperado de https://learn.microsoft.com/en-us/cpp/cpp/data-type-ranges?view=msvc-170 ↩
- Pain--In--The--Brain. (2020). *Why use std::byte over unsigned char?*. Recuperado de https://www.reddit.com/r/cpp_questions/comments/hlyav9/why_use_stdbyte_over_unsigned_char/ ↩
- Stannum. (2016). *Unsized integer types and signedness*. Recuperado de https://stannum.io/blog/0MXgB0 ↩
- Trivedi, U. (2024). *Type Inference in C++ (auto and decltype)*. Recuperado de https://www.geeksforgeeks.org/type-inference-in-c-auto-and-decltype/ ↩
- Trivedi, U. (2024). *Understanding nullptr in C++*. Recuperado de https://www.geeksforgeeks.org/understanding-nullptr-c/ ↩