5 - Tipi, qualificatori e costanti letterali

Tipi integrali

- tipi booleani `bool`
- tipi carattere:
  - narrow: `char`, `signed char`, `unsigned char`
  - wide: `wchar_t`, `char16_t`, `char32_t`
- tipi interi standard con segno: `signed char`, `short`, `int`, `long`, `long long`
- tipi interi senza segno: `unsigned char`, `unsigned short`, `unsigned int`...

Tipi integrali piccoli

Tipi boolenai, tipi caratteri narrow e short (con o senza segno) sono detti tipi integrali piccoli: potrebbero avere una dimensione (`sizeof`) inferiore al tipo `int` (non per forza).
I tipi integrali piccoli sono soggetti a una certa categoria di conversioni implicite: **promozioni**.

Tipi built-in non integrali

- Tipi floating point: `float`, `double`, `long double`
- Tipo void: insieme vuoto di valori
  - come tipo di ritorno di una funzione indica che non deve tornare alcun valore
  - usato nel cast esplicito, indica che il valore di una espressione deve essere scartato
    `(void) foo(3); //scarta il risultato di foo(3)`
- Tipo `std::nullptr_t` tipo puntatore convertibile implicitamente in qualunque altro tipo di puntatore; ha un solo valore possibile, la costante letterale `nullptr`, che indica il puntatore nullo (non deferenziabile).

Tipi composti

- `T&`: riferimento a lvalue T
- `T&&`: riferimento a rvalue T
- `T*`: puntatore a T
- `T[n]`: array
- `T(T1, T2, T3)`: tipo funzione
- enumerazioni, classi e struct

Tipi qualificati `const`

- **qualificatori di tipo**: `const` e `volatile`
- considerando solo il qualificatore `const`
  - essenziale per una corretta progettazione e uso delle interfacce software
  - utile come strumento di supporto
- dato un tipo T, è possibile fornire la versione qualificata `const T`
- l'accesso ad un oggetto attraverso una variabile dichiarata `const` è consentito in sola lettura

- nel caso di tipi composti è necessario distinguere tra le qualificazioni del tipo composto rispetto alla qualificazione delle sue componenti.

Costanti letterali

Vengono messe a disposizione varie sintassi per definire valori costanti; a seconda della sintassi usata, al valore viene associato un tipo specifico, che in alcuni casi dipende dall'implementazione.

tipi costanti letterali

- *booleani*: `false`, `true`;
- *char*:
  - Si rappresentano racchiusi tra apici singoli: `'a'`, `'3'`.
    - Caratteri speciali come `'\n'` (nuova linea) possono essere inclusi.
    - Esistono diverse codifiche:
      - **UTF-8**: aggiungi `u8` davanti, esempio: `u8'a'`.
      - **UTF-16**: usa `u`, esempio: `u'a'`.
      - **UTF-32**: usa `U`, esempio: `U'a'`.
      - **Wide character** (più spazio per caratteri complessi): usa `L`, esempio: `L'a'`.
- *numeri interi*
  - Esempio base: `123`.
  - Puoi specificare varianti con **suffissi**:
    - `U`: per numeri senza segno (esempio: `123U`).
    - `L`: per numeri "grandi" (esempio: `123L`).
    - `LL`: per numeri ancora più grandi (esempio: `123LL`).
    - Puoi combinarli: `123UL` (unsigned long).
- *numeri a virgola mobile (floating point)*
  - Esempio base: `123.45`.
  - Puoi scriverli anche in **notazione scientifica**: `1.23e2` (che significa 1.23·1021.23 \cdot 10^21.23·102).
  - Usa i suffissi per specificare il tipo:
    - `F`: per `float` (esempio: `123.45F`).
    - `L`: per `long double` (esempio: `123.45L`).
- *stringhe*
  - Si scrivono tra virgolette doppie: `"Hello"`.
  - Il tipo associato è `const char[]` (un array di caratteri costanti).
  - Puoi aggiungere prefissi per cambiare la codifica:
    - `u8"Hello"`: UTF-8.
    - `u"Hello"`: UTF-16.
    - `U"Hello"`: UTF-32.
    - `L"Hello"`: Wide.
- *stringhe grezze (raw string literals)*
  - Sono stringhe che possono contenere caratteri "problematici" (come virgolette o newline) senza doverli scrivere in modo complicato. Si usano con il prefisso `R`:
    `R"(Questa è una stringa "grezza" che può contenere qualsiasi cosa!)"`
- *puntatore nullo*

- Il valore `nullptr` rappresenta un puntatore nullo, cioè un riferimento "vuoto" a memoria.
- Il tipo associato è `std::nullptr_t`.