Tipi, qualificatori e costanti letterali

Enea Zaffanella

Metodologie di programmazione
Laurea triennale in Informatica
Università di Parma
enea.zaffanella@unipr.it

I 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 standard **senza segno**:
  `unsigned char`, `unsigned short`, `unsigned int`, ...

I tipi integrali piccoli

- i tipi booleani, i tipi carattere narrow e short (con o senza segno) sono detti tipi integrali **piccoli**: potrebbero (non è detto) avere una dimensione (`sizeof`) inferiore al tipo `int`
- la cosa è rilevante perché i tipi integrali piccoli sono soggetti a una particolare categoria di conversioni implicite: le **promozioni**
- distinguere le promozioni da altre conversioni implicite è importante (ad esempio, per la risoluzione dell'overloading di funzione)

Tipi built-in non integrali

- tipi floating point:
  `float`, `double`, `long double`
- tipo `void`: insieme vuoto di valori
  - come tipo di ritorno, indica che una funzione non ritorna alcun valore
  - usato in un cast esplicito, indica che il valore di una espressione deve essere scartato (i.e., non interessa)

```
(void) foo(3); // scarta il risultato di foo(3)
```

- tipo `std::nullptr_t` (dal c++11)
  tipo puntatore convertibile implicitamente in qualunque altro tipo puntatore; ha un solo valore possibile, la costante letterale `nullptr`, che indica il puntatore nullo (non dereferenziabile)

I tipi composti

- `T&`: riferimento a lvalue T
- `T&&`: riferimento a rvalue T (dal c++11)
- `T*`: puntatore a T
- `T[n]`: array
- `T(T1, T2, T3)`: tipo funzione
- enumerazioni, classi e struct

Tipi qualificati: il qualificatore const

- **qualificatori di tipo** `const` e `volatile`
- consideriamo solo il qualificatore `const`
  - essenziale per una corretta progettazione e uso delle interfacce software
  - utile come strumento di supporto alla manutenzione del software
- dato un tipo `T`, è possibile fornirne la versione qualificata `const T`
- l'accesso ad un oggetto (o una parte di un oggetto) attraverso una variabile il cui tipo è dotato del qualificatore const è consentito in sola lettura: non sono consentite le modifiche
- nel caso di tipi composti è necessario distinguere tra la qualificazione del tipo composto rispetto alla qualificazione delle sue componenti

Esempio qualificazione const

```
struct S {
  int v;
  const int c;
  S(int cc) : c(cc) { v = 10; }
};

int main() {
  const S sc(5)
  sc.v = 20; // errore: sc è const e anche le sue componenti

  S s(5);
  s.v = 20; // legittimo: s non è const e S::v non è const
  s.c = 20; // errore: s non è const, ma S::c è const
}
```

Altro esempio

- lo stesso oggetto può essere modificabile o meno a seconda del percorso usato per accedervi

```
struct S { int v; };
void foo() {
  S s;
  s.v = 10; // legittimo
  const S& sr = s; // riferimento a s, qualificato const
  sr.v = 10; // errore: non posso modificare s passando da sr.
}
```

Le costanti letterali

- il linguaggio mette 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
- nota bene: nel seguito vedremo un certo numero di casi, ma **non tutti**; per esempio, per gli interi vedremo solo la sintassi **decimale**, ma esistono anche:
  - la sintassi binaria: `0b1100`
  - la sintassi ottale: `014`
  - la sintassi esadecimale: `0xC`

Esempi di costanti letterali

- `bool`: `false`, `true`
- `char`:
  `'a'`, `'3'`, `'\n'` (caratteri ordinari)
  `u8'a'`, `u8'3'` (codifica UTF-8)
- `signed char`, `unsigned char`: nessuna costante
- `char16_t`: `u'a'`, `u'3'`
- `char32_t`: `U'a'`, `U'3'`
- `wchar_t`: `L'a'`, `L'3'`
- `short`, `unsigned short`: nessuna costante

Esempi di costanti letterali (cont.)

- `int`: 12345
- nota: in assenza di **suffissi** (`U`, `L`, `LL`) ad una costante **decimale** intera viene attribuito il primo tipo tra `int`, `long` e `long long` che sia in grado di rappresentarne il valore; il tipo dipende quindi dalla particolare implementazione utilizzata
- uso dei suffissi:
  - suffisso `U`: si sceglie la variante unsigned
  - suffisso `L`: ampiezza scelta tra `long` e `long long`
  - suffisso `LL`: ampiezza `long long`
  - `U` può comparire anche insieme a `L` o `LL`
- nota: le regole per le altre sintassi (booleana, ottale, esadecimale) prendono in considerazione anche i tipi senza segno

Esempi di costanti letterali (cont.)

```
// long
12345L
// long long
12345LL
// unsigned int:
12345U
// unsigned long
12345UL
// unsigned long long:
12345ULL
```

Esempi di costanti letterali (cont.)

- per i floating point si può scegliere tra notazione decimale e "scientifica" (nota: esiste anche una notazione esadecimale):

```
// float
123.45F, 1.2345e2F
// double
123.45, 1.2345e2
// long double
123.45L, 1.2345e2L
```

- `void`: nessuna costante
- `std::nullptr_t`: solo `nullptr`

Esempi di costanti letterali (cont.)

- letterali stringa (C-string):

```
"Hello"
```

  il tipo associato è `const char[6]` (array di 6 caratteri costanti, con terminatore `'\0'`)
- possiamo specificare un prefisso di encoding (u8, u, U, L) come nel caso delle costanti carattere, che modifica in modo analogo il tipo degli elementi dell'array
- letterali stringa "grezza" (raw string literal): sono costruiti usando lo schema

```
R"DELIMITATORE(stringa)DELIMITATORE"
```

  usando il prefisso `R` e un delimitatore a piacere; e.g., usando `STRING` come delimitatore:

```
R"STRING(la␣mia␣stringa,␣lunga␣lunga,␣contiene␣anche
caratteri␣newline,␣senza␣doverli␣codificare␣in␣modo␣strano
e␣può␣anche␣contenere␣" "␣""␣doppi␣apici,␣e␣altri␣caratteri
strani,␣eccetera,␣eccetera)STRING"
```

Letterali definiti dall'utente

- c++11 ha reso possibile la definizione dei cosiddetti **user defined literal**: notazione per aggiungere un **suffisso** che causa l'invocazione di una **conversione definita dall'utente**
- ad esempio, dal c++14, il tipo `std::string` consente l'uso del suffisso `s` (e dell'operatore di conversione `operator""s` definito nel namespace `std::literals`)

```
#include <iostream>
#include <string>

int main() {
  // stampa la stringa C   (tipo const char[6])
  std::cout << "Hello";
  // stampa la stringa C++ (tipo std::string)
  using namespace std::literals;
  std::cout << "Hello"s;
}
```