Deduzione automatica dei tipi

Template type deduction

La template type deduction (deduzione dei tipi per i parametri template) è il processo messo in atto dal compilatore per semplificare l'istanziazione (implicita o esplicita) e la specializzazione (esplicita) dei template di funzione. Questa forma di deduzione è utile per il programmatore in quanto consente di evitare la scrittura (noiosa, ripetitiva e soggetta a sviste) della lista degli argomenti da associare ai parametri del template di funzione.

Il processo di deduzione è intuitivo e comodo da usare, ma in alcuni casi può riservare sorprese. Per semplificare al massimo, supponiamo di avere la seguente dichiarazione di funzione templatica:

```
template
void f(PT param);
```

nella quale `TT` è il nome del parametro del template di funzione, mentre `PT` indica una espressione sintattica che denota il tipo del parametro `param` della funzione. Il caso che ci interessa, naturalmente, è quello in cui `PT` nomina il parametro templatico `TT`.

Il compilatore, di fronte alla chiamata di funzione `f(expr)`, usa il tipo di `expr(te)` per dedurre:

- un tipo specifico `tt` per `TT`
- un tipo specifico `pt` per `PT`

causando l'istanziazione del template di funzione e ottenendo la funzione

```
void f(pt param);
```

Nota: i tipi dedotti `tt` e `pt` sono correlati, ma spesso non sono identici.

Il processo di deduzione distingue tre casi:

1. `PT` è sintatticamente uguale a `TT&&` (cioè, `PT` è una *"universal reference"*, secondo la terminologia di Meyers).
2. `PT` è un tipo puntatore o riferimento (ma NON una universal reference).
3. `PT` non è né un puntatore né un riferimento.

Torna all'indice

TT&& (universal reference)

Si ha una universal reference quando abbiamo l'applicazione di `&&` al nome di un parametro typename del mio template di funzione, senza nessun altro modificatore. Per esempio, se `TT` è il parametro typename:

```
TT&&               // è una universal reference
const TT&&         // NON è una universal reference (è un rvalue reference)
std::vector&&  // NON è una universal reference (è un rvalue reference)
```

Il nome *"universal reference"* indica il fatto che, sebbene venga usata la sintassi per i riferimenti a rvalue, può essere dedotto per `PT` un riferimento a rvalue oppure a lvalue, a seconda del tipo `te` di `expr`.

Negli esempi si assume:

```
int i = 0;
const int ci = 1;
```

Esempio 1.1 (rvalue):

```
f(5);           // te = int, deduco pt = int&&, tt = int
f(std::move(i)); // te = int&&, deduco pt = int&&, tt = int
```

Esempio 1.2 (lvalue):

```
f(i);  // te = int&, deduco pt = int&,  tt = int&
f(ci); // te = const int&, deduco pt = const int&, tt = const int&
```

Torna all'indice

PT puntatore o riferimento

È diverso da `TT&&`. Fondamentalmente, si effettua un pattern matching tra il tipo `te` e `PT`, ottenendo i tipi `tt` e `pt` di conseguenza:

Esempio 2.1:

```
template
void f(TT* param);

f(&i);  // te = int*, deduco pt = int*, tt = int
f(&ci); // te = const int*, deduco pt = const int*, tt = int
```

Esempio 2.2:

```
template
void f(const TT* param);

f(&i);  // te = int*, deduco pt = const int*, tt = int
f(&ci); // te = const int*, deduco pt = const int*, tt = int
```

Il caso dei riferimenti è analogo:

Esempio 2.3:

```
template
void f(TT& param);

f(i);   // te = int&, deduco pt = int&, tt = int
f(ci);  // te = const int&, deduco pt = const int&, tt = const int
```

Esempio 2.4:

```
template
void f(const TT& param);

f(i);   // te = int&, deduco pt = const int&, tt = int
f(ci);  // te = const int&, deduco pt = const int&, tt = int
```

Torna all'indice

PT né puntatore né riferimento

```
template
void f(TT param);
```

Siccome abbiamo un passaggio per valore, argomento e parametro sono due oggetti distinti: eventuali riferimenti e qualificazioni const (a livello esterno, quelli a destra di *) dell'argomento NON si propagano al parametro.

Esempio 3.1:

```
f(i);   // te = int&, deduco pt = int, tt = int
f(ci);  // te = const int&, deduco pt = int, tt = int
```

Si noti comunque che eventuali qualificazioni const a livello interno vengono preservate:

Esempio 3.2:

```
const char* const p = "Hello";
f(p); // te = const char* const&, deduco pt = const char*, tt = const char*
```

Torna all'indice

Auto type deduction

A partire dallo standard C++11, nel linguaggio è stata introdotta la possibilità di definire le variabili usando la parola chiave `auto`, senza specificarne esplicitamente il tipo, lasciando al compilatore il compito di dedurlo a partire dall'espressione usata per inizializzare la variabile.

Esempio:

```
auto i = 5;             // `i` ha tipo int
const auto d = 5.3;     // `d` ha tipo const double
auto ii = i * 2.0;      // `ii` ha tipo double
const auto p = "Hello"; // `p` ha tipo const char* const
```

La auto type deduction segue (in larga misura) le stesse regole elencate sopra per la template type deduction.

In pratica, quando si osserva una definizione di variabile come

```
auto& ri = ci;
```

1. la parola chiave `auto` svolge il ruolo del parametro template `TT`;
2. la sintassi `auto&` corrisponde a `PT`;
3. l'inizializzatore `ci` (di tipo const `int&`) corrisponde all'espressione `expr`.

L'esempio qui mostrato corrisponde quindi al caso 2 della deduzione di parametri template (`PT` è un riferimento a lvalue, non universal). Per `auto` si deduce il tipo `const int` e quindi per `ri` si deduce il tipo `const int&` (si veda il secondo caso dell'Esempio 2.3).

La forma sintattica `auto&&` indica una universal reference, che potrebbe dedurre sia un rvalue o un lvalue reference a seconda del tipo dell'inizializzatore.

La auto template deduction differisce però dalla template type deduction quando l'inizializzatore è indicato mediante la sintassi che prevede le parentesi graffe, come nell'esempio:

```
auto i = { 1 };
```

In questo caso speciale, che non approfondiremo, l'argomento si considera di tipo `std::initializer_list<T>`.

> Alcune linee guida di programmazione suggeriscono di usare `auto` quasi sempre per inizializzare le variabili; nell'acronimo AAA (Almost Always Auto), la prima A (Almost) indica appunto la presenza di eccezioni alla linea guida (quelle viste sopra per gli inizializzatori con parentesi graffe).

Torna all'indice