# 22 - Deduzione automatica dei tipi di dato (lez. 29-11-2024)

## Template type deduction

E' il processo attraverso cui il compilatore deduce automaticamente i tipi di dati associati ai parametri di un template. Semplifica l'uso dei template e consente di evitare la scrittura della lista di argomenti da associare ai parametri del template di funzione.

- riduce il codice scritto
- previene errori dovuti a specifiche manuali

## Regole Generali per la deduzione nei Template

```
template <typename TT>
void f(PT param);
```

- TT: parametro del template
- **PT**: è il tipo dichiarato del parametro `param` della funzione. È definito in funzione di `TT`
  Quando viene chiamata `f(arg)`, il compilatore guarda il tipo di arg e deduce:
- `tt` il tipo dedotto per il parametro template `TT`
- `pt` il tipo dedotto per `PT` in funzione di `tt`
  N.B. i tipi dedotti `tt` e `pt` sono correlati, ma spesso non sono identici
  il processo di deduzione si dividono in tre casi principali:

1. **PT** è sintatticamente uguale a `TT&&`, ovvero è una *universal reference*
2. **PT** è un tipo puntatore o riferimento (ma non una universal reference)
3. **PT** non è né un puntatore né un riferimento

### Caso 1: Universal Reference ( `PT` è `TT&&` )

- Sono riferimenti speciali che possono essere sia riferimenti a *lvalue* (oggetti già esistenti) che a *rvalue* (oggetti temporanei).
- **Differenza rispetto ad altri riferimenti**:
  - `TT&&` → *universal reference* (deduce sia rvalue che lvalue);
  - `const TT&&` → *rvalue reference* (deduce solo rvalue);
  - `std::vector<TT>&&` → rvalue reference (riferito a un tipo specifico come `std::vector`).
- la **deduzione** dipende dal tipo dell'argomento:
  - se `arg` è un *lvalue*, il tipo dedotto per **PT** sarà un riferimento a *lvalue* ( `TT&` );
  - se `arg` è un *rvalue*, il tipo dedotto per **PT** sarà un riferimento a *rvalue* ( `TT&&` )
  esempi:

1. rvalue

```
int i = 0;
f(5);   //argomento: rvalue (5) -> int
```

```
        //deduzione:
        //TT = int
        //PT = int&& (riferimento a rvalue)
```

2. lvalue

```
int i = 0;
f(i);          // argomento: lvalue (i) -> int&
               // deduzione:
               // TT = int&
               // PT = int& (riferimento a lvalue)
```

3. `const TT&&` non è universal reference, in quanto è *rvalue reference*
4. `std::vector&&` non è universal reference, in quanto è *rvalue reference*
5. std::move

```
f(std::move(i));  // argomento: rvalue (trasformazione di i con std::move) -> int&&
                  // deduzione:
                  // TT = int
                  // PT = int&& (riferimento a rvalue)
```

**Vantaggi**:

- permettono di scrivere template di funzione che funzionano sia con lvalue sia con rvalue senza duplicare il codice
- supportano perfettamente il "forwarding" (passaggio di argomenti senza perdita di informazioni sul tipo)

## Caso 2: PT puntatore

Il compilatore effettua un *pattern matching* per far coincidere il tipo dell'argomento con un puntatore o con un riferimento.

- **Regole di deduzione**:
  - il compilatore deduce il tipo `TT` che "spiega" `PT`
  - se `PT` è un riferimento (es. `TT&` o `const TT&`), il riferimento viene mantenuto;
  - se `PT` è un puntatore (es. `TT*` o `const TT*`), i qualificatori const vengono dedotti.
  Esempi puntatori:

1. puntatore semplice

```
int i = 0;
f(&i);          // argomento: puntatore a int (&i)
                // deduzione:
                // TT = int
                // PT = int* (puntatore a int)
```

2. puntatore a const

```
const int ci = 1;
f(&ci);         // argomento: puntatore a const int (&ci)
                // deduzione:
                // TT = const int
                // PT = const int* (puntatore a const int)
```

3. puntatore con const nel parametro

```
template <typename TT> void f(const TT* param);
f(&i);          // argomento: puntatore a int (&i)
                // deduzione:
                // TT = int
                // PT = const int* (puntatore costante)
```

Esempi riferimenti:

1. riferimento a lvalue

```
int i = 0;
f(i);           // argomento: lvalue
                // deduzione:
                // TT = int
                // PT = int& (riferimento a int)
```

2. riferimento a const lvalue

```
const int ci = 1;
f(ci);          // argomento: const int
                // deduzione:
                // TT = const int
                // PT = const int& (riferimento costante a int)
```

3. riferimento costante esplicito

```
template <typename TT> void f(const TT& param);
f(i);           // argomento: int
                // deduzione:
                // TT = int
                // PT = const int& (riferimento costante)
```

### Caso 3: PT né puntatore né riferimento

Quando il parametro del template viene passato per valore, il compilatore crea una copia dell'elemento passato:

- qualsiasi qualificatore const o riferimento (&, &&) dell'argomento viene rimosso nella deduzione del tipo
- il parametro della funzione è un oggetto separato
  Esempi:

1. oggetto semplice:

```
int i = 0;
f(i);          // argomento: lvalue (int)
               // deduzione:
               // TT = int
               // PT = int (nessun riferimento o costante)
```

2. oggetto costante:

```
const int ci = 1;
f(ci);         // argomento: const int
               // deduzione:
               // TT = int (senza const)
               // PT = int
```

3. qualificatori interni (const all'interno del tipo):

```
const char* const p = "Hello";
f(p);          // argomento: const char* const
               // deduzione:
               // TT = const char* (const interno conservato)
               // PT = const char*
```

## Deduzione dei tipi con `auto`

Permette di far dedurre al compilatore il tipo di una variabile basandosi sul suo inizializzatore, inoltre torna utile quando dobbiamo dare un nome a una variabile di una funzione lambda in quanto il tipo non potremmo scriverlo. Esempi:

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

Le regole per l'auto type deduction sono simili a quelle dei template. Esempio:
`auto& ri = ci; // ri = const int&`
Inoltre, caso particolare rispetto alla *template type deduction*, con l'utilizzo delle parentesi graffe possiamo ottenere:
`auto i = {1}; // Tipo dedotto: std::initializer_list<int>`
Alcune linee guida di programmazione suggeriscono di usare `auto` quasi sempre, AAA (Almost Always Auto).