19 - Template

Template di funzione

Un template di funzione consente di scrivere un modello parametrico per una funzione, rendendola generica e adatta a lavorare con diversi tipi di dato.

Dichiarazione e Definizione

Un template di funzione si dichiara e definisce con la parola chiave `template`, seguita da una lista di parametri di template racchiusa tra parentesi angolate `<>`. Questi parametri rappresentano tipi di dato generici.

```
//dichiarazione
template <typename T>
T max(T a, T b);

//definizione
template <typename T>
T max(T a, T b) {
    return (a > b)? a : b;
}
```

Parametri del template

I parametri di un template possono rappresentare:

1. **tipi generici** (ad esempio, `T` in `typename T`);
2. **valori** (ad esempio, interi o puntatori);
3. **template di altri template** (template nidificati);
   Sintassi:

- la parola chiave `typename` o `class` può essere usata per indicare un tipo generico. Le due parole chiave sono equivalenti, ma `typename` è preferibile per coerenza semantica.
- i nomi dei parametri di tipo sono convenzionalmente maiuscoli.

```
template <typename T, int N>
T arrayMax(T (&arr)[N]) {
  T maxVal = arr[0];
  for (int i = 1; i < N; ++i) {
    if (arr[i] > maxVal) {
      maxVal = arr[i];
    }
  }
  return maxVal;
}
```

Istanziazione dei template di funzione

Quando si utilizza un template, il compilatore crea automaticamente una istanza del template con i tipi specificati.

Istanziazione implicita

Il compilatore deduce automaticamente i tipi degli argomenti passati alla funzione.

```
int maxInt = max(10, 20);     // T dedotto come int
double maxDouble = max(3.5, 2.1); // T dedotto come double
```

Istanziazione esplicita

È possibile specificare esplicitamente i tipi nella chiamata alla funzione.

```
int maxInt = max<int>(10, 20);
```

Problemi di deduzione

La deduzione dei tipi fallisce se gli argomenti non corrispondono univocamente a un tipo.

```
int result = max(10.5, 20); // Errore: T non può essere dedotto
int result = max<int>(10.5, 20); // Corretto: forzo T = int
```

Specializzazione esplicita

La specializzazione esplicita consente di fornire una definizione specifica per determinati tipi.

```
template <typename T>
T max(T a, T b) {
  return (a > b) ? a : b;
}

// Specializzazione per const char*
template <>
const char* max<const char*>(const char* a, const char* b) {
  return strcmp(a, b) > 0 ? a : b;
}
```

N.B. la lista vuota di parametri indica una specializzazione totale per uno specifico tipo.

Istanziazione Esplicita

È possibile richiedere al compilatore di istanziare un template senza usarlo direttamente nel codice.

```
// Dichiarazione di istanziazione
extern template int max<int>(int, int);

// Definizione di istanziazione
template int max<int>(int, int);
```

N.B.

Esiste una differenza sostanziale tra un template di funzione e le sue possibili istanziazioni

- un template di funzione non è una funzione (è un "generatore" di funzioni);
- una istanza di un template di funzione è una funzione.

Template di classe

Un template di classe consente di scrivere un modello parametrico per una classe, rendendo possibile generare classi per tipi diversi.

Dichiarazione e Definizione

Un template di classe viene definito con la stessa sintassi dei template di funzione, ma con una classe al posto di una funzione.

```
template <typename T>
class Stack {
private:
  std::vector<T> elements;

public:
  void push(const T& element) {
    elements.push_back(element);
  }
  void pop() {
    elements.pop_back();
  }
};
```

Istanziazione di template di classe

Per i template di classe, i parametri non vengono dedotti automaticamente; devono essere specificati esplicitamente.

```
Stack<int> intStack;       // Istanza per int
Stack<std::string> strStack; // Istanza per std::string
```

Deduzione con `auto`:

```
auto copyStack = intStack; // Deduzione del tipo
```

Specializzazione dei template di classe

Come per i template di funzione, è possibile creare specializzazioni **totali** e **parziali**.

Specializzazione totale

Una definizione completa per un tipo specifico.

```
template <>
class Stack<bool> {
```

```
private:
  std::vector<unsigned char> bits; // Ottimizzazione per bool

public:
  void push(bool value) { /* ... */ }
  void pop() { /* ... */ }
};
```

Specializzazione parziale

Una definizione per un sottoinsieme di tipi.

```
template <typename T>
class Stack<T*> { // Specializzazione per puntatori
private:
  std::vector<T*> elements;

public:
  void push(T* element) { elements.push_back(element); }
  void pop() { elements.pop_back(); }
};
```

Altri tipi di template

Template di Alias

Un template di alias consente di creare alias per tipi complessi.

```
template <typename T>
using Vec = std::vector<T, std::allocator<T>>;
```

Template di variabile

Un template di variabile consente di creare costanti parametrizzate.

```
template <typename T>
constexpr T pi = T(3.1415926535897932385);
```