27 - Cast (lez. 04-12-2024)

Il C++ fornisce varie sintassi per effettuare il cast (:= conversione esplicita di tipo) di una espressione, allo scopo di ottenere un valore di un tipo (potenzialmente) diverso:

1. `static_cast`
2. `dynamic_cast`
3. `const_cast`
4. `reinterpret_cast`
5. cast "funzionale"
6. cast "stile C"
   I cast devono essere utilizzati solo quando necessario. È possibile classificare gli usi dei cast in base alla motivazione:

Caso 1: Implementazione di conversioni non consentite implicitamente

Il cast implementa una conversione di tipo non consentita dal linguaggio come conversione implicita, poiché potrebbe essere una frequente fonte di errori. Utilizzando un cast esplicito, il programmatore si assume la responsabilità della correttezza della conversione.

```
struct B { /* ... */ };
struct D : public B { /* ... */ };
D d;
B* b_ptr = &d;
// b_ptr è un puntatore a B, ma dinamicamente punta a un oggetto di tipo D.
D* d_ptr = static_cast<D*>(b_ptr); // Il programmatore forza il down-cast.
```

N.B. se qualcuno nel frattempo avesse modificato `b_ptr` e questo non puntasse più ad un oggetto di tipo D, si ottiene un *undefined behavior*.

Caso 2: Uso di `dynamic_cast` per verificare la conversione a runtime

Il programmatore utilizza un cast dinamico per verificare a runtime se la conversione è consentita.

```
struct B { virtual ~B() {} }; // Classe base dinamica
struct D : public B { /* ... */ };

void foo(B* b_ptr) {
    if (D* d_ptr = dynamic_cast<D*>(b_ptr)) {
        // Uso d_ptr perché il cast è andato a buon fine.
    } else {
        // b_ptr non punta a un oggetto di tipo D.
    }
}
```

Caso 3: Conversioni consentite implicitamente, ma fatte per documentazione

Il cast non è strettamente necessario (la conversione implicita è permessa), ma il programmatore preferisce renderlo esplicito per migliorare la leggibilità e documentare l'intenzione.

```
double d = 3.14;
int approx = static_cast<int>(d); // Evidenzia che si perde informazione.
```

Caso 4: Conversione a `void` per ignorare un valore

In alcuni casi, si usa un cast esplicito a `void` per ignorare il valore di un'espressione e silenziare warning del compilatore.

```
void foo(int pos, int size) {
    assert(0 <= pos && pos < size);
    static_cast<void>(size); // Silenzia il warning sul mancato uso di size.
}
/// in casi come questo si tollera il cast in stile C:
(void)size;
```

Tipologie di cast

1. `static_cast`

È il cast più comune. Si usa per conversioni ben definite e controllate. La sintassi è:

```
static_cast<T>(expr)
```

Esempi:

- conversioni implicite:

  ```
  double d = 3.14;
  int i = static_cast<int>(d);
  ```

- downcast nelle gerarchie di classi :

  ```
  B* b_ptr = new D();
  D* d_ptr = static_cast<D*>(b_ptr);
  ```

- conversioni a `void` :

  ```
  static_cast<void>(expr);
  ```

2. `dynamic_cast`

Si usa per verificare la validità di conversioni in gerarchie di classi a runtime. Funziona solo con classi dinamiche (contenenti almeno un metodo virtuale).

```cpp
struct B { virtual ~B() {} };
struct D : public B { /* ... */ };

B* b_ptr = new D();
D* d_ptr = dynamic_cast<D*>(b_ptr); // Controlla se b_ptr è effettivamente un D.
if (d_ptr) {
    // Il cast è valido.
} else {
    // Il cast è fallito.
}
```

3. `const_cast`

Serve per rimuovere (o aggiungere) la qualificazione `const`.

```cpp
void modifica(const int& ci) {
    int& i = const_cast<int&>(ci);
    i = 42; // Modifica il valore ignorando la promessa di non modificarlo.
}
```

Nota: L'abuso di `const_cast` è considerato cattivo stile. In casi legittimi, può essere usato per aggiornare dati interni dichiarati `mutable`.

4. `reinterpret_cast`

Il cast più pericoloso. Si usa per reinterpretare un bit pattern senza alcuna garanzia di validità.

```cpp
int x = 42;
void* v_ptr = &x;
int* i_ptr = reinterpret_cast<int*>(v_ptr);
```

Nota: è responsabilità del programmatore assicurarsi che il risultato sia valido.

5. Cast Funzionale

Ha la sintassi:

```cpp
T(expr)
```

Si usa spesso per tipi primitivi o per costruire oggetti.

```cpp
int i = int(3.14); //cast funzionale
```

6. Cast stile C

Ha la sintassi:

```cpp
(T)expr
```

È considerato cattivo stile, eccetto quando viene usato per conversioni a `void`.