10_Lvalue_Rvalue

Table of contents

- Lvalue vs Rvalue
 - 1. prvalue
 - 2. xvalue
 - 3. Ivalue
- Funzioni speciali

Lvalue vs Rvalue

Nel C++ possiamo categorizzare le espressioni.

Un'espressione e' caratterizzata da due proprieta' indipendenti: il *tipo* e il *valore categorico* o valore. Ogni espressione appartiene a una delle 3 categorie principali:

- prvalue (pure rvalue)
 un espressione che computa il valore di un operando di un operatore built-in o che inizializza un oggetto (il numero 5 è un prvalue)
- xvalue (eXpiring value)
 che denota un oggetto le quali risorse possono essere riutilizzate (la risorsa sta per morire)
- Ivalue espressione determinante un'identita' di oggetto o funzione (sinistra di un'operatore d'assegnamento)

L'esistenza di una categorizzazione simile è dovuta dall'esistenza di una classe di priorità, dove nella quale viene dettato chi ha il permesso di "rubare" o meno una risorsa: non sempre è necessario copiare e incollare, può essere utile passare il riferimento.

prvalue

Il risultato della computazione puo' essere una variabile, un oggetto creato dell'espressione new, un temporaneo derivante dalla materializzazione temporanea o un suo membro. Sono pure values, i seguenti esempi.

```
42, true, nullptr // letterali (tranne che di stringa)
a++, a--; //pre e post incremento
a + b, a && b // tutti gli operatori aritmetici e logici
a, b; // la virgola, con 'b' un rvalue
```

xvalue

Sono expiring values, i seguenti esempi.

```
a.m // il membro di un oggetto
a[n] // dove un operando e' un array rvalue
```

Ivalue

Il termine Ivalue e' storico: deriva dal fatto che i left value potevano apparire dal lato sinistro dell'assegnamento. Sono left value, i seguenti esempi.

```
i = 7; // il nome di una qualsiasi variabile o funzione
std::cout << 1; // function calls o overloading di operandi</pre>
```

```
a = b && a+=b; // tutti gli assegnamenti e operatori composti
"Hello, world!" // letterali di stringa
```

Parliamo di rvalue, per indicare quando un'espressione puo' essere, o prvalue, o xvalue.

Un prvalue non identifica un oggetto in memoria e quindi non sarebbe lecito assegnarvici un valore o prenderne l'indirizzo.

```
i = 4 + 1; // '4+1' e' prvalue, quindi rvalue
i = i + 1; // 'i+1' e' prvalue, quindi rvalue
```

Funzioni speciali

Vedere la categorizzazione delle espressioni, ci aiuta a capire che, prima dello standard C++ 2011, alcuni problemi erano rilevanti: certi costrutti non erano implementati efficientemente. Questo ci diventa chiaro, quando discutiamo di riferimenti a Ivalue T& e riferimenti a rvalue T&. Ogni classe, nello standard 2003, era fornita di 4 funzioni speciali:

costruttore di default

```
Matrix();
```

costruttore di copia

```
Matrix (const Matrix&);
```

· assegnamento per copia

```
Matrix& operator=(const Matrix&);
```

distruttore

```
~Matrix();
```

Il fatto che non ci fosse modo intuitivo di prendere un valore in input e modificarlo restituendolo come variante modificata, portava a sprechi di memoria:

- non esisteva un modo di dire alla funzione, che l'oggetto preso in input non era piu' d'interesse, e che quindi potesse essere modificato sul posto;
- non esisteva modo di ritornare l'oggetto, senza prima farne una seconda copia.

```
Matrix bar(const Matrix& arg) {
  Matrix res = arg; // copia (1)
  // modifica di res
  return res; // ritorna una copia (2)
}
```

Proprio per questa ragione, furono introdotti 2 nuovi costrutti.

Usando un riferimento a rvalue, in entrambi i casi le risorse contenute nell'oggetto non possono essere utilizzate da altri e quindi possono essere spostate.

costruttore per spostamento

```
Matrix(Matrix&&);
```

· assegnamento per spostamento

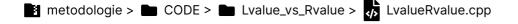
```
Matrix& operator=(Matrix&&);
```

L'esempio visto sopra ora invochera' automaticamente il costruttore di spostamento, in quanto il compilatore si accorgera' che return res e' un xvalue, che dovra' restituirlo al chiamante (stiamo evitando la copia (2)). Riscriviamo l'esempio andando in overloading, saltando tutte le copie non necessarie:

```
Matrix bar(Matrix&& arg) {
    // modifica di arg
    return arg; // spostamento
}
```

Usando lo standard C++11, possiamo comunque riscrivere e fare un passaggio per argomento, siccome starà al compilatore capire se abbiamo fornito come argomento un Ivalue o rvalue:

```
Matrix bar(Matrix arg){
    // modifica di arg
    return arg; // spostamento
}
```



07/03/2023