Appendix D

# 運算子優先權

運算子優先權（precedence）決定運算子對數值運算時的應用順序，C++運算子擁有 18 類運算優先權，請參閱表 D.1。其中第 1 類優先權最高，第 2 類優先權次之，其餘依此類推。如果兩個運算子應用至相同的運算元，則由優先權較高的運算子先執行。如果兩個運算子有相同的運算優先權，則 C++依據結合性（associativity）決定哪一個運算子較為緊密。在同一群的所有運算子中都有相同的優先權和相同的結合性，結合性可能是由左而右（表格中的 L-R）或由右而左（表格中的 R-L）。由左而右的結合性代表最左邊的運算子先運算，而右而左的結合性代表最右邊的運算子先運算。

表 D.1 C++ 運算子的優先權與結合性

| 運算子 | 結合性 | 意義 |
|---|---|---|
| **第 1 類優先權** | | |
| `::` | | 範疇運算子 |
| **第 2 類優先權** | | |
| (*expression*) | | 分組 |
| `()` | L-R | 函數呼叫 |
| `()` | | 值建構，意即 *type* (*expr*) |
| `[]` | | 陣列足標 |
| `->` | | 間接成員運算子 |
| `.` | | 直接成員運算子 |
| `const_cast` | | 特定的型態轉換 |
| `dynamic_cast` | | 特定的型態轉換 |
| `reinterpret_cast` | | 特定的型態轉換 |
| `static_cast` | | 特定的型態轉換 |

| 運算子 | 結合性 | 意義 |
|---|---|---|
| typeid | | 型態識別 |
| ++ | | 遞增運算子，後置形式 |
| -- | | 遞減運算子，後置形式 |
| 第 3 類優先權（皆為一元） | | |
| ! | R-L | 邏輯否定 |
| ~ | | 位元否定 |
| + | | 一元加號（正號） |
| - | | 一元減號（負號） |
| ++ | | 遞增運算子，前置形式 |
| -- | | 遞減運算子，前置形式 |
| & | | 取址 |
| * | | 提領（間接值） |
| () | | 型態轉換，意即(type) expr |
| sizeof | | 以位元組為單位的大小 |
| alignof | | 對齊 |
| new | | 動態配置儲存空間 |
| new [] | | 動態配置陣列 |
| delete | | 動態釋放儲存空間 |
| delete [] | | 動態釋放陣列 |
| noexcept | | 若運算元丟出異常，則為假 |
| 第 4 類優先權 | | |
| .* | L-R | 成員提領 |
| ->* | | 間接成員提領 |
| 第 5 類優先權（皆為二元） | | |
| * | L-R | 乘法 |
| / | | 除法 |
| % | | 模數（餘數） |
| 第 6 類優先權（皆為二元） | | |
| + | L-R | 加法 |
| - | | 減法 |
| 第 7 類優先權 | | |
| << | L-R | 左移 |
| >> | | 右移 |
| 第 8 類優先權 | | |
| < | L-R | 小於 |
| <= | | 小於或等於 |

| 運算子 | 結合性 | 意義 |
|---|---|---|
| >= | | 大於或等於 |
| > | | 大於 |
| 第 9 類優先權 | | |
| = = | L-R | 等於 |
| != | | 不等於 |
| 第 10 類優先權（二元） | | |
| & | L-R | 位元 AND |
| 第 11 類優先權 | | |
| ^ | L-R | 位元 XOR（排他性 OR） |
| 第 12 類優先權 | | |
| \| | L-R | 位元 OR |
| 第 13 類優先權 | | |
| && | L-R | 邏輯 AND |
| 第 14 類優先權 | | |
| \|\| | L-R | 邏輯 OR |
| 第 15 類優先權 | | |
| :? | R-L | 條件式 |
| 第 16 類優先權 | | |
| = | R-L | 簡單的指定 |
| *= | | 乘法並指定 |
| /= | | 除法並指定 |
| %= | | 取餘數並指定 |
| += | | 加法並指定 |
| -= | | 減法並指定 |
| &= | | 位元 AND 並指定 |
| ^= | | 位元 XOR 並指定 |
| \|= | | 位元 OR 並指定 |
| <<= | | 左移並指定 |
| >>= | | 右移並指定 |
| 第 17 類優先權 | | |
| throw | L-R | 丟出異常 |
| 第 18 類優先權 | | |
| , | L-R | 結合兩個運算式為一個 |

有些符號，例如 * 和 & 運算子，不只一種功能。此時分**一元**（unary）運算子（一個運算元）和**二元**（binary）運算子（兩個運算元），編譯程式依據運算元數

目決定運算子的功能。對於可用於兩種情況的符號，表 D.1 會標示運算子類別為一元或二元。

以下是說明優先權和結合性的例子。

在下面的範例，編譯程式必須決定要作的是 3 加 5，還是 5 乘以 6：

```
3 + 5 * 6
```

* 運算子的運算優先權較 + 運算子高，所以它先用於 5，運算式變成 3 + 30，最後結果為 33。

在下面的範例，編譯程式必須決定要作的是 120 除以 6，還是 6 乘以 5：

```
120 / 6 * 5
```

/ 運算子與 * 運算子的優先權相同，但其結合性是由左而右。這意思是共用運算元 (6) 的左邊運算子會先計算，運算式變成 20 * 5，最後變成 100。

在下面的範例，編譯程式必須決定要先作的是 str 的遞增，還是對 str 作提領：

```
char * str = "Whoa";
char ch = *str++;
```

後置形式（postfix）的 ++ 運算子比一元運算子 * 有較高的優先權。這意思是遞增運算子會應用在 str 上，而不是 *str。也就是說，此運算會先遞增指標，使其指向下一個字元，而不是修改所指的字元。但是，因為 ++ 是後置形式，所以是將 *str 之原始值指定給 ch 後，然後再遞增指標。因此，將字元 W 設給 ch，然後移動 str 使其指向 h 字元。

以下是類似的範例：

```
char * str = "Whoa";
char ch = *++str;
```

前置形式（prefix）的 ++ 運算子與一元運算子 * 有相同的優先權，但是它們的結合性是由右而左。所以同樣的，會遞增 str 而不是 *str。因為 ++ 運算子是前置形式，所以先遞增 str，然後再提領結果指標。因此，移動 str 使其指向 h 字元，並將 h 字元設給 ch。

注意，表 D.1 在「優先權」欄位會標示**二元**或**一元**，以區分使用相同符號的運算子，如一元的取址運算子和二元的位元 AND 運算子。

附錄 B：「保留字」，也列出一些運算子的另一種表示方式，如表 B.2。