

第3課：運算符與表達式

第3課：運算符與表達式

課程目標

- 掌握C++中的各種運算符
- 理解運算符的優先級和結合性
- 學會構建複雜的表達式
- 了解不同運算符的使用場景

本課關鍵字

- 算術運算符: +, -, *, /, %
- 關係運算符: ==, !=, <, >, <=, >=
- 邏輯運算符: &&, ||, !
- 位運算符: &, |, ^, ~, <<, >>
- 賦值運算符: =, +=, -=, *=, /=, %=
- 其他運算符: ?:, ,, sizeof, ++, --

範例1：算術運算符

```
/*
 * 範例3-1：算術運算符
 * 用於基本的數學計算：加、減、乘、除、取餘數
 */
#include <iostream>
int main() {
    std::cout << "=== 算術運算符演示 ===" << std::endl;
    int a = 20;
    int b = 6;
    double c = 20.0;
    double d = 6.0;
    std::cout << "整數運算：" << std::endl;
    std::cout << "a = " << a << ", b = " << b << std::endl;
    std::cout << "a + b = " << (a + b) << std::endl; // 加法
    std::cout << "a - b = " << (a - b) << std::endl; // 減法
    std::cout << "a * b = " << (a * b) << std::endl; // 乘法
    std::cout << "a / b = " << (a / b) << std::endl; // 整數除法
```

```

std::cout << "a % b = " << (a % b) << std::endl; // 取餘數
std::cout << "\n浮點數運算：" << std::endl;
std::cout << "c = " << c << ", d = " << d << std::endl;
std::cout << "c / d = " << (c / d) << std::endl; // 浮點數除法
// 注意：整數除法和浮點數除法的區別
std::cout << "\n整數除法 vs 浮點數除法：" << std::endl;
std::cout << "10 / 4 = " << (10 / 4) << " (整數除法，結果為整數)" << std::endl;
std::cout << "10.0 / 4 = " << (10.0 / 4) << " (浮點數除法，結果為浮點數)" <<
std::endl;
// 取餘運算符 (%) 只能用於整數
// std::cout << "10.5 % 3.2 = " << (10.5 % 3.2) << std::endl; // 錯誤！
std::cout << "10 % 3 = " << (10 % 3) << std::endl;
// 運算符的優先級
std::cout << "\n運算符優先級：" << std::endl;
int result1 = 2 + 3 * 4; // 先乘除後加減
int result2 = (2 + 3) * 4; // 括號優先
std::cout << "2 + 3 * 4 = " << result1 << std::endl;
std::cout << "(2 + 3) * 4 = " << result2 << std::endl;
return 0;
}

```

範例2：關係運算符與邏輯運算符

```

/*
* 範例3-2：關係運算符與邏輯運算符
* 用於比較和邏輯判斷
*/
#include <iostream>
int main() {
std::cout << "=== 關係運算符與邏輯運算符演示 ===" << std::endl;
int x = 10;
int y = 5;
int z = 10;
// 關係運算符（比較運算符）
std::cout << "關係運算符：" << std::endl;
std::cout << "x = " << x << ", y = " << y << ", z = " << z << std::endl;
std::cout << "x > y : " << (x > y) << std::endl; // 大於
std::cout << "x < y : " << (x < y) << std::endl; // 小於

```

```

std::cout << "x >= z : " << (x >= z) << std::endl; // 大於等於
std::cout << "x <= z : " << (x <= z) << std::endl; // 小於等於
std::cout << "x == z : " << (x == z) << std::endl; // 等於
std::cout << "x != y : " << (x != y) << std::endl; // 不等於
// 邏輯運算符
std::cout << "\n邏輯運算符：" << std::endl;
bool isSunny = true;
bool isWarm = false;
bool isWeekend = true;
std::cout << std::boolalpha; // 輸出 true/false 而非 1/0
std::cout << "isSunny: " << isSunny << std::endl;
std::cout << "isWarm: " << isWarm << std::endl;
std::cout << "isWeekend: " << isWeekend << std::endl;
std::cout << "\n邏輯運算：" << std::endl;
std::cout << "isSunny && isWarm : " << (isSunny && isWarm) << std::endl; // 邏輯與 (AND)
std::cout << "isSunny || isWarm : " << (isSunny || isWarm) << std::endl; // 邏輯或 (OR)
std::cout << "!isSunny : " << (!isSunny) << std::endl; // 邏輯非 (NOT)
// 組合邏輯
bool goodWeather = isSunny && !isWarm; // 晴天但不熱
bool goodDay = isWeekend && goodWeather; // 週末且好天氣
std::cout << "\n組合邏輯：" << std::endl;
std::cout << "好天氣（晴天但不熱）：" << goodWeather << std::endl;
std::cout << "好日子（週末且好天氣）：" << goodDay << std::endl;
// 短路求值
std::cout << "\n短路求值：" << std::endl;
int a = 5, b = 0;
// 邏輯與：如果第一個為假，第二個不會被求值
if (b != 0 && a / b > 2) {
std::cout << "這行不會執行" << std::endl;
} else {
std::cout << "短路求值避免了除以零的錯誤" << std::endl;
}
// 邏輯或：如果第一個為真，第二個不會被求值
if (a > 0 || b / a > 0) {
std::cout << "短路求值：第二個表達式不會被求值" << std::endl;
}
return 0;
}

```

範例3：賦值運算符與複合賦值運算符

```
/*
* 範例3-3：賦值運算符與複合賦值運算符
* 用於賦值和簡化運算
*/
#include <iostream>
int main() {
    std::cout << "=== 賦值運算符與複合賦值運算符演示 ===" << std::endl;
    // 基本賦值運算符
    int x = 10; // 初始化賦值
    int y;
    y = 20; // 賦值
    int z = x; // 用變數初始化
    std::cout << "基本賦值：" << std::endl;
    std::cout << "x = " << x << std::endl;
    std::cout << "y = " << y << std::endl;
    std::cout << "z = " << z << std::endl;
    // 複合賦值運算符
    int a = 10;
    std::cout << "\n複合賦值運算符：" << std::endl;
    std::cout << "初始值 a = " << a << std::endl;
    a += 5; // a = a + 5
    std::cout << "a += 5 後：" << a << std::endl;
    a -= 3; // a = a - 3
    std::cout << "a -= 3 後：" << a << std::endl;
    a *= 2; // a = a * 2
    std::cout << "a *= 2 後：" << a << std::endl;
    a /= 4; // a = a / 4
    std::cout << "a /= 4 後：" << a << std::endl;
    a %= 3; // a = a % 3
    std::cout << "a %= 3 後：" << a << std::endl;
    // 多重賦值
    int p, q, r;
    p = q = r = 100; // 從右向左賦值
    std::cout << "\n多重賦值：" << std::endl;
    std::cout << "p = " << p << ", q = " << q << ", r = " << r << std::endl;
    // 賦值表達式的值
```

```

std::cout << "\n賦值表達式的值：" << std::endl;
int m, n;
n = (m = 15) + 5; // m = 15，然後 n = m + 5
std::cout << "n = (m = 15) + 5 的結果：" << std::endl;
std::cout << "m = " << m << std::endl;
std::cout << "n = " << n << std::endl;
// 複合賦值的優勢
std::cout << "\n複合賦值的優勢：" << std::endl;
// 方法1：傳統寫法
int value1 = 10;
value1 = value1 + 5;
value1 = value1 * 2;
std::cout << "方法1（傳統）：value1 = " << value1 << std::endl;
// 方法2：複合賦值
int value2 = 10;
value2 += 5; // 更簡潔
value2 *= 2;
std::cout << "方法2（複合）：value2 = " << value2 << std::endl;
// 對於複雜表達式，複合賦值更高效
int arr[5] = {1, 2, 3, 4, 5};
int sum = 0;
for (int i = 0; i < 5; i++) {
    sum += arr[i]; // 比 sum = sum + arr[i] 更好
}
std::cout << "數組總和：" << sum << std::endl;
return 0;
}

```

範例4：遞增與遞減運算符

```

/*
* 範例3-4：遞增與遞減運算符
* ++ 和 -- 運算符，分為前綴和後綴形式
*/
#include <iostream>
int main() {
    std::cout << "=== 遞增與遞減運算符演示 ===" << std::endl;
    // 前綴遞增 (++i)：先遞增，後使用

```

```

std::cout << "前綴遞增 (++i)：" << std::endl;
int a = 5;
int b = ++a; // a先增加到6，然後賦值給b
std::cout << "a = 5" << std::endl;
std::cout << "b = ++a" << std::endl;
std::cout << "現在 a = " << a << ", b = " << b << std::endl;
// 後綴遞增 (i++)：先使用，後遞增
std::cout << "\n後綴遞增 (i++)：" << std::endl;
int c = 5;
int d = c++; // 先把c的值(5)賦給d，然後c增加到6
std::cout << "c = 5" << std::endl;
std::cout << "d = c++" << std::endl;
std::cout << "現在 c = " << c << ", d = " << d << std::endl;
// 前綴遞減 (--i)
std::cout << "\n前綴遞減 (--i)：" << std::endl;
int e = 5;
int f = --e; // e先減少到4，然後賦值給f
std::cout << "e = 5" << std::endl;
std::cout << "f = --e" << std::endl;
std::cout << "現在 e = " << e << ", f = " << f << std::endl;
// 後綴遞減 (i--)
std::cout << "\n後綴遞減 (i--)：" << std::endl;
int g = 5;
int h = g--; // 先把g的值(5)賦給h，然後g減少到4
std::cout << "g = 5" << std::endl;
std::cout << "h = g--" << std::endl;
std::cout << "現在 g = " << g << ", h = " << h << std::endl;
// 在表達式中使用
std::cout << "\n在表達式中使用：" << std::endl;
int i = 5;
int j = 10;
int result1 = ++i + j; // i先增到6，然後 6 + 10 = 16
std::cout << "int result1 = ++i + j;" << std::endl;
std::cout << "result1 = " << result1 << ", i = " << i << std::endl;
int k = 5;
int l = 10;
int result2 = k++ + l; // 先 5 + 10 = 15，然後k增到6
std::cout << "int result2 = k++ + l;" << std::endl;
std::cout << "result2 = " << result2 << ", k = " << k << std::endl;
// 常見用法：循環計數器

```

```

std::cout << "使用後綴遞增：" << std::endl;
for (int count = 1; count <= 5; count++) {
std::cout << "count = " << count << std::endl;
}
// 前綴與後綴的性能差異（通常很小）
std::cout << "\n性能提示：" << std::endl;
std::cout << "前綴遞增(++i)通常比後綴遞增(i++)更高效" << std::endl;
std::cout << "因為後綴遞增需要保存臨時值" << std::endl;
std::cout << "在循環中推薦使用 ++i" << std::endl;
// 陷阱：多次修改同一變數
std::cout << "\n陷阱：未定義行為" << std::endl;
int x = 5;
// int y = x++ + ++x; // 未定義行為！避免！
std::cout << "int y = x++ + ++x; // 未定義行為，避免這樣寫！" << std::endl;
return 0;
}

```

範例5：位運算符

```

/*
* 範例3-5：位運算符
* 用於對整數的二進制位進行操作
*/
#include <iostream>
#include <bitset> // 用於顯示二進制
int main() {
std::cout << "=== 位運算符演示 ===" << std::endl;
unsigned int a = 60; // 二進制：0011 1100
unsigned int b = 13; // 二進制：0000 1101
std::cout << "a = 60 (二進制: " << std::bitset<8>(a) << ")" << std::endl;
std::cout << "b = 13 (二進制: " << std::bitset<8>(b) << ")" << std::endl;
std::cout << std::endl;
// 位與 (AND) - &
unsigned int c = a & b;
std::cout << "位與 (AND) - &" << std::endl;
std::cout << "a & b = " << c << std::endl;
std::cout << "二進制: " << std::bitset<8>(a) << " &" << std::endl;
std::cout << " " << std::bitset<8>(b) << std::endl;

```

```

std::cout << " -----" << std::endl;
std::cout << " " << std::bitset<8>(c) << std::endl;
std::cout << std::endl;
// 位或 (OR) - |
c = a | b;
std::cout << "位或 (OR) - |" << std::endl;
std::cout << "a | b = " << c << std::endl;
std::cout << "二進制: " << std::bitset<8>(a) << " |" << std::endl;
std::cout << " " << std::bitset<8>(b) << std::endl;
std::cout << " -----" << std::endl;
std::cout << " " << std::bitset<8>(c) << std::endl;
std::cout << std::endl;
// 位異或 (XOR) - ^
c = a ^ b;
std::cout << "位異或 (XOR) - ^" << std::endl;
std::cout << "a ^ b = " << c << std::endl;
std::cout << "二進制: " << std::bitset<8>(a) << " ^" << std::endl;
std::cout << " " << std::bitset<8>(b) << std::endl;
std::cout << " -----" << std::endl;
std::cout << " " << std::bitset<8>(c) << std::endl;
std::cout << std::endl;
// 位取反 (NOT) - ~
c = ~a;
std::cout << "位取反 (NOT) - ~" << std::endl;
std::cout << "~a = " << c << std::endl;
std::cout << "二進制: ~" << std::bitset<8>(a) << std::endl;
std::cout << " -----" << std::endl;
std::cout << " " << std::bitset<8>(c) << std::endl;
std::cout << std::endl;
// 左移 (Left Shift) - <<
c = a << 2;
std::cout << "左移 (Left Shift) - <<" << std::endl;
std::cout << "a << 2 = " << c << std::endl;
std::cout << "二進制: " << std::bitset<8>(a) << " << 2" << std::endl;
std::cout << " -----" << std::endl;
std::cout << " " << std::bitset<8>(c) << std::endl;
std::cout << "相當於乘以 2^2 = 4: 60 * 4 = 240" << std::endl;
std::cout << std::endl;
// 右移 (Right Shift) - >>
c = a >> 2;

```

```

std::cout << "a >> 2 = " << c << std::endl;
std::cout << "二進制: " << std::bitset<8>(a) << " >> 2" << std::endl;
std::cout << " -----" << std::endl;
std::cout << " " << std::bitset<8>(c) << std::endl;
std::cout << "相當於除以 2^2 = 4: 60 / 4 = 15" << std::endl;
std::cout << std::endl;
// 實際應用：權限控制
std::cout << "實際應用：權限控制" << std::endl;
// 定義權限標誌
const unsigned int READ = 1; // 0001
const unsigned int WRITE = 2; // 0010
const unsigned int EXECUTE = 4; // 0100
unsigned int userPermissions = 0;
// 添加權限（使用位或）
userPermissions = userPermissions | READ; // 添加讀權限
userPermissions = userPermissions | WRITE; // 添加寫權限
std::cout << "用戶權限: " << std::bitset<4>(userPermissions) << std::endl;
// 檢查權限（使用位與）
bool canRead = (userPermissions & READ) != 0;
bool canWrite = (userPermissions & WRITE) != 0;
bool canExecute = (userPermissions & EXECUTE) != 0;
std::cout << "可以讀取: " << (canRead ? "是" : "否") << std::endl;
std::cout << "可以寫入: " << (canWrite ? "是" : "否") << std::endl;
std::cout << "可以執行: " << (canExecute ? "是" : "否") << std::endl;
// 移除權限（使用位與和位取反）
userPermissions = userPermissions & ~WRITE; // 移除寫權限
std::cout << "\n移除寫權限後:" << std::endl;
std::cout << "用戶權限: " << std::bitset<4>(userPermissions) << std::endl;
canWrite = (userPermissions & WRITE) != 0;
std::cout << "可以寫入: " << (canWrite ? "是" : "否") << std::endl;
return 0;
}

```

範例6：條件運算符與其他運算符

```

/*
* 範例3-6：條件運算符與其他運算符
* 條件運算符 (?:)、逗號運算符 (,)、sizeof 等

```

```

*/
#include <iostream>
int main() {
    std::cout << "=== 條件運算符與其他運算符演示 ===" << std::endl;
    // 1. 條件運算符（三元運算符）?:
    std::cout << "1. 條件運算符（三元運算符）?:" << std::endl;
    int score = 85;
    std::string result = (score >= 60) ? "及格" : "不及格";
    std::cout << "分數: " << score << std::endl;
    std::cout << "結果: " << result << std::endl;
    // 嵌套條件運算符
    char grade = (score >= 90) ? 'A' :
        (score >= 80) ? 'B' :
        (score >= 70) ? 'C' :
        (score >= 60) ? 'D' : 'F';
    std::cout << "等級: " << grade << std::endl;
    // 條件運算符返回的值可以參與運算
    int a = 10, b = 20;
    int max = (a > b) ? a : b;
    int min = (a < b) ? a : b;
    std::cout << "a = " << a << ", b = " << b << std::endl;
    std::cout << "較大值: " << max << std::endl;
    std::cout << "較小值: " << min << std::endl;
    // 2. 逗號運算符 ,
    std::cout << "\n2. 逗號運算符 ," << std::endl;
    // 逗號運算符會依次執行表達式，返回最後一個表達式的值
    int x = 1, y = 2, z = 3;
    int commaResult = (x++, y++, z++, x + y + z);
    std::cout << "表達式: (x++, y++, z++, x + y + z)" << std::endl;
    std::cout << "結果: " << commaResult << std::endl;
    std::cout << "現在 x = " << x << ", y = " << y << ", z = " << z << std::endl;
    // 常見用法：for循環中的多個變數
    std::cout << "\n逗號運算符在for循環中的應用:" << std::endl;
    for (int i = 0, j = 10; i < j; i++, j--) {
        std::cout << "i = " << i << ", j = " << j << std::endl;
    }
    // 3. sizeof 運算符
    std::cout << "\n3. sizeof 運算符" << std::endl;
    std::cout << "基本類型的大小（位元組）:" << std::endl;
    std::cout << "sizeof(char): " << sizeof(char) << std::endl;
}

```

```

std::cout << "sizeof(double): " << sizeof(double) << std::endl;
std::cout << "sizeof(bool): " << sizeof(bool) << std::endl;
// 變數和類型都可以
int num = 100;
double price = 99.99;
std::cout << "\n變數的大小:" << std::endl;
std::cout << "sizeof(num): " << sizeof(num) << std::endl;
std::cout << "sizeof(price): " << sizeof(price) << std::endl;
// 數組的大小
int arr[10];
std::cout << "\n數組的大小:" << std::endl;
std::cout << "sizeof(arr): " << sizeof(arr) << std::endl;
std::cout << "數組元素個數: " << sizeof(arr) / sizeof(arr[0]) << std::endl;
// 4. 運算符優先級演示
std::cout << "\n4. 運算符優先級演示" << std::endl;
int m = 1, n = 2, p = 3;
// 複雜表達式
int complexResult1 = m + n * p; // 先乘後加
int complexResult2 = (m + n) * p; // 括號優先
int complexResult3 = m++ + --n * p; // 前綴遞減優先於乘法
std::cout << "m = " << m << ", n = " << n << ", p = " << p << std::endl;
std::cout << "m + n * p = " << complexResult1 << std::endl;
std::cout << "(m + n) * p = " << complexResult2 << std::endl;
std::cout << "m++ + --n * p = " << complexResult3 << std::endl;
// 使用括號明確優先級
std::cout << "\n使用括號明確優先級（推薦）:" << std::endl;
int clearResult = ((m + n) * p) / 2;
std::cout << "((m + n) * p) / 2 = " << clearResult << std::endl;
return 0;
}

```

練習題

練習3-1：計算器程序

創建一個簡單的計算器程序，可以進行加、減、乘、除四種運算。

輸入示例：

請輸入第一個數字：10

請輸入運算符 (+, -, *, /): +

請輸入第二個數字：5

結果：10 + 5 = 15

練習3-2：閏年判斷

使用關係運算符和邏輯運算符判斷一個年份是否為閏年。閏年規則：

1. 能被4整除但不能被100整除，或者
2. 能被400整除

練習3-3：位運算應用

創建一個程序，演示如何使用位運算：

1. 檢查一個數是否為2的冪
2. 交換兩個變數的值（不使用臨時變數）
3. 計算一個數的二進制表示中1的個數

練習3-4：成績轉換

使用條件運算符將百分制成績轉換為等級制：

- 90分以上: A
- 80-89分: B
- 70-79分: C
- 60-69分: D
- 60分以下: F

練習3-5：運算符優先級

計算以下表達式的值，並驗證你的計算：

```
int a = 5, b = 3, c = 2;  
int result1 = a + b * c;  
int result2 = (a + b) * c;  
int result3 = a++ + --b * c;  
int result4 = a & b | c;
```

重點摘要

1. 運算符分類

類別	運算符	描述
算術	+ - * / %	基本數學運算
關係	== != < > <= >=	比較大小
邏輯	&& !	邏輯與、或、非
位運算	& ^ ~ << >>	二進制位操作
賦值	= += -= *= /= %=	賦值和複合賦值
遞增遞減	++ --	增加或減少1
其他	?: , sizeof	條件、逗號、大小

2. 運算符優先級（從高到低）

優先級	運算符	結合性
1	() [] . ->	從左到右
2	! ~ ++ -- + - * & (type) sizeof	從右到左
3	* / %	從左到右
4	+ -	從左到右
5	<< >>	從左到右
6	< <= > >=	從左到右
7	== !=	從左到右
8	&	從左到右
9	^	從左到右
10		從左到右
11	&&	從左到右
12		從左到右
13	?:	從右到左
14	= += -= *= /= %= &= ^= = <<= >>=	從右到左
15	,	從左到右

記憶口訣：括號成員第一；全體單目第二；乘除餘三，加減四；移位五，關係六；等於不等於排第七；位與異或和位或；三分天下八九十；邏輯與，邏輯或；十二和十一；條件高於賦值；逗號運算級別低。

3. 重要概念

短路求值

- 邏輯與 &&：第一個為假時，第二個不計算
- 邏輯或 ||：第一個為真時，第二個不計算

前綴 vs 後綴

- `++i`：先遞增，後使用
- `i++`：先使用，後遞增
- **推薦使用前綴形式（性能更好）**

複合賦值的優勢

- 更簡潔
- 可能更高效（避免重複計算左值）

類型轉換規則

1. 小類型自動轉換為大類型
2. 整數與浮點數運算，整數轉為浮點數
3. 賦值時，右邊類型轉為左邊類型

常見問答

Q1: `i++` 和 `++i` 有什麼區別？

A: `i++` 是後綴遞增，返回遞增前的值；`++i` 是前綴遞增，返回遞增後的值。在單獨使用時效果相同，但在表達式中使用時效果不同。

Q2: 為什麼 `1/2` 的結果是 0 而不是 0.5？

A: 因為 1 和 2 都是整數，整數除法的結果也是整數，小數部分被捨棄。要得到 0.5，需要使用浮點數：`1.0/2` 或 `1/2.0`。

Q3: `&&` 和 `&` 有什麼區別？

A: `&&` 是邏輯與，用於布林值；`&` 是位與，用於整數的二進制位操作。例如：`(5 > 3) && (2 < 4)` 返回 `true`，而 `5 & 3` 返回 1。

Q4: 如何避免運算符優先級的混淆？

A: 使用括號明確指定優先級。即使知道優先級規則，使用括號也能使代碼更清晰易讀。

Q5: `sizeof` 是函數還是運算符？

A: `sizeof` 是運算符，不是函數。它可以用於類型和變數，編譯時計算大小。

下一步

下一課我們將學習：

- 流程控制：條件語句（`if`, `else`, `switch`）
- 條件判斷的各種形式
- 多分支選擇結構
- 嵌套條件語句



完整測試程序

```
/*
 * 第3課綜合測試
 * 練習使用各種運算符
 */
#include <iostream>
#include <string>
int main() {
    std::cout << "=== 第3課綜合測試 ===" << std::endl;
    // 1. 基本運算
    std::cout << "\n1. 基本運算：" << std::endl;
    int num1 = 15, num2 = 4;
    std::cout << num1 << " + " << num2 << " = " << (num1 + num2) << std::endl;
    std::cout << num1 << " - " << num2 << " = " << (num1 - num2) << std::endl;
    std::cout << num1 << " * " << num2 << " = " << (num1 * num2) << std::endl;
    std::cout << num1 << " / " << num2 << " = " << (num1 / num2) << std::endl;
    std::cout << num1 << " % " << num2 << " = " << (num1 % num2) << std::endl;
    // 2. 關係與邏輯運算
    std::cout << "\n2. 關係與邏輯運算：" << std::endl;
    int age = 18;
    bool hasLicense = true;
    bool canDrive = (age >= 18) && hasLicense;
    std::cout << "年齡：" << age << " 歲" << std::endl;
    std::cout << "有駕照：" << (hasLicense ? "是" : "否") << std::endl;
    std::cout << "可以開車：" << (canDrive ? "是" : "否") << std::endl;
    // 3. 複合賦值
    std::cout << "\n3. 複合賦值：" << std::endl;
    int total = 100;
    std::cout << "初始值：" << total << std::endl;
    total += 50; // 購買商品
    total -= 30; // 退款
    total *= 2; // 促銷翻倍
    total /= 4; // 分享給4人
    std::cout << "最終值：" << total << std::endl;
    // 4. 遞增遞減
    std::cout << "\n4. 遞增遞減：" << std::endl;
    int counter = 5;
    std::cout << "初始值：" << counter << std::endl;
    int afterIncrement = ++counter;
```

```

std::cout << "前綴遞增後: counter = " << counter
<< ", afterIncrement = " << afterIncrement << std::endl;
int afterDecrement = counter--;
std::cout << "後綴遞減後: counter = " << counter
<< ", afterDecrement = " << afterDecrement << std::endl;
// 5. 條件運算符
std::cout << "\n5. 條件運算符:" << std::endl;
int temperature = 28;
std::string weather = (temperature > 25) ? "炎熱" :
(temperature > 15) ? "舒適" : "寒冷";
std::cout << "溫度: " << temperature << "°C" << std::endl;
std::cout << "天氣: " << weather << std::endl;
// 6. 位運算 (權限控制)
std::cout << "\n6. 位運算 (權限控制):" << std::endl;
// 權限定義
const int PERM_READ = 1; // 0001
const int PERM_WRITE = 2; // 0010
const int PERM_EXEC = 4; // 0100
const int PERM_ADMIN = 8; // 1000
int user1 = PERM_READ | PERM_WRITE; // 普通用戶
int user2 = PERM_READ | PERM_WRITE | PERM_EXEC | PERM_ADMIN; // 管理員
std::cout << "用戶1權限: ";
if (user1 & PERM_READ) std::cout << "讀 ";
if (user1 & PERM_WRITE) std::cout << "寫 ";
if (user1 & PERM_EXEC) std::cout << "執行 ";
if (user1 & PERM_ADMIN) std::cout << "管理 ";
std::cout << std::endl;
std::cout << "用戶2權限: ";
if (user2 & PERM_READ) std::cout << "讀 ";
if (user2 & PERM_WRITE) std::cout << "寫 ";
if (user2 & PERM_EXEC) std::cout << "執行 ";
if (user2 & PERM_ADMIN) std::cout << "管理 ";
std::cout << std::endl;
// 7. sizeof 運算符
std::cout << "\n7. sizeof 運算符:" << std::endl;
std::cout << "int 大小: " << sizeof(int) << " 位元組" << std::endl;
std::cout << "double 大小: " << sizeof(double) << " 位元組" << std::endl;
std::cout << "char 大小: " << sizeof(char) << " 位元組" << std::endl;
int numbers[5] = {1, 2, 3, 4, 5};
std::cout << "數組大小: " << sizeof(numbers) << " 位元組" << std::endl;

```

```
std::cout << "數組元素個數: " << sizeof(numbers) / sizeof(numbers[0]) <<
std::endl;
return 0;
}
```

輸出結果：

=== 第3課綜合測試 ===

1. 基本運算：

15 + 4 = 19

15 - 4 = 11

15 * 4 = 60

15 / 4 = 3

15 % 4 = 3

2. 關係與邏輯運算：

年齡：18 歲

有駕照：是

可以開車：是

3. 複合賦值：

初始值：100

最終值：60

4. 遞增遞減：

初始值：5

前綴遞增後：counter = 6, afterIncrement = 6

後綴遞減後：counter = 5, afterDecrement = 6

5. 條件運算符：

溫度：28°C

天氣：炎熱

6. 位運算（權限控制）：

用戶1權限：讀 寫

用戶2權限：讀 寫 執行 管理

7. sizeof 運算符：

int 大小：4 位元組

double 大小：8 位元組

char 大小：1 位元組

數組大小：20 位元組

數組元素個數：5

本課檢查清單

- ☐ 理解各種算術運算符的使用
- ☐ 能夠使用關係和邏輯運算符進行比較
- ☐ 掌握複合賦值運算符的用法
- ☐ 理解前綴和後綴遞增/遞減的區別
- ☐ 了解位運算符的基本概念
- ☐ 能夠使用條件運算符（三元運算符）
- ☐ 理解運算符的優先級和結合性
- ☐ 能夠構建複雜的表達式

需要幫助？

如果你在練習中遇到問題：

1. 檢查運算符是否使用正確（特別是 `=` 和 `==` 的區別）
2. 注意整數除法和浮點數除法的區別
3. 理解前綴和後綴遞增/遞減的區別
4. 使用括號明確運算優先級

下一課見！ 