C++程式設計：結構與列舉

目錄

1. 結構基本用法

一名學生有以下資料：

- String name
- String ID
- String phone
- 各學期平均成績：float[]
- 生日：birthYear、birthMonth、birthDay

1.1 範例：使用 struct 儲存學生資料

```
#include <iostream>
#include <string>
using namespace std;

struct Student {
    string name;
    string id;
    string phone;
    float grade[4];
```

```
    int birthYear, birthMonth, birthDay;
};

int main() {
    Student john = {
        "John Smith",
        "12345",
        "1234567",
        {4.0, 3.9, 3.8, 3.6},
        2000, 1, 1
    };

    cout << "name is " << john.name << endl; // name is John Smith
    cout << "id is " << john.id << endl; // id is 12345
    cout << "phone is " << john.phone << endl; // phone is 1234567
    cout << "john.grade[0] is " << john.grade[0] << endl; // john.grade[0] is 4

    return 0;
}
```

1.2 範例：指定欄位設定（使用 string & 陣列）

在指定欄位的時候，int、float、bool 可以直接使用指定運算子(就是等號)；而字元陣列(char)要使用 strcpy()

```
#include <iostream>
#include <string>
using namespace std;

struct Student {
    string name;
    int id;
    string phone;
    float grade[4];
    int birthYear, birthMonth, birthDay;
};

int main() {
    Student john;
    john.name = "John Smith";
    john.id = 12345;
    john.phone = "1234567";
    john.grade[0] = 4.0;
```

```
    john.grade[1] = 3.9;
    john.grade[2] = 3.8;
    john.grade[3] = 3.6;
    john.birthYear = 2000;
    john.birthMonth = 1;
    john.birthDay = 1;

    cout << "name is " << john.name << endl;
    cout << "id is " << john.id << endl;
    cout << "phone is " << john.phone << endl;
    cout << "grade is ";
    for (float g : john.grade) cout << g << " ";
    cout << endl;
    cout << "birth = " << john.birthYear << "/" << john.birthMonth << "/" <<
john.birthDay << endl;

    return 0;
}
```

2. 結構指標（→ 簡化語法）

由於 struct 裡面的欄位是資料，因此如果要使用指標去操作 struct 欄位的資料，就要先取【整個欄位】，再【用星號取這個欄位的值】=> *(ptr).id。那可以簡化為 ptr->id。

2.1 範例：結構指標的使用

```
#include <iostream>
#include <string>
using namespace std;

struct Student {
    string name;
    string id;
    string phone;
    float grade[4];
    int birthYear, birthMonth, birthDay;
};

int main() {
    Student john = {
        "John",
```

```
        "12345",
        "1234567",
        {4.0, 3.9, 3.8, 3.6},
        2000, 1, 1
    };

    Student* ptr = &john;
    ptr->phone = "00000";

    cout << "name is " << ptr->name << endl;
    cout << "phone is " << ptr->phone << endl;

    return 0;
}
```

2.2 範例：使用 struct 處理複數加法（值傳遞）

```
#include <iostream>
using namespace std;

struct Complex {
    int real;
    int imag;
};

Complex addComplex(Complex a, Complex b) {
    Complex c;
    c.real = a.real + b.real;
    c.imag = a.imag + b.imag;
    return c;
}

void printComplex(Complex a) {
    cout << a.real << "+" << a.imag << "i" << endl;
}

int main() {
    Complex a = {1, 3};
    Complex b = {5, 2};
    Complex c = addComplex(a, b);
```

```
    printComplex(c);
    return 0;
}
```

2.3 範例： 使用指標參數處理複數加法

```
#include <iostream>
using namespace std;

struct Complex {
    int real;
    int imag;
};

void addComplex(const Complex* a, const Complex* b, Complex* c) {
    c->real = a->real + b->real;
    c->imag = a->imag + b->imag;
}

void printComplex(const Complex* a) {
    cout << a->real << "+" << a->imag << "i" << endl;
}

int main() {
    Complex a = {1, 3};
    Complex b = {5, 2};
    Complex c;

    addComplex(&a, &b, &c);
    printComplex(&c);

    return 0;
}
```

2.4 範例：在結構中使用「子結構」

```
#include <iostream>
#include <string>
using namespace std;

struct Date {
```

```
    int year;
    int month;
    int day;
};

struct Student {
    string name;
    string id;
    string phone;
    float grade[4];  // 4 個學期的平均成績
    Date schoolDay;
    Date birthDay;   // 使用子結構 Date
};

int main() {
    Student a = {
        "John",
        "12345",
        "1234567",
        {4.0, 3.9, 3.8, 3.6},
        {2000, 1, 1},         // SchoolDay
        {2001, 10, 29}        // BirthDay
    };

    cout << "name is " << a.name << endl;
    cout << "grade is " << a.grade[0] << endl;
    cout << "birthDay day is " << a.birthDay.day << endl;

    return 0;
}
```

3. typedef

結構在初始化時，要寫兩個英文字母，typedef 可以把它結合在一起。

3.1 範例：使用 using（typedef 替代）+ 結構指標處理複數

```
#include <iostream>
using namespace std;

// 使用 using 替代 typedef（現代 C++ 寫法）
struct ComplexStruct {
```

```
    int real;
    int imag;
};
using Complex = ComplexStruct;

// 加法
void addComplex(const Complex* a, const Complex* b, Complex* c) {
    c->real = a->real + b->real;
    c->imag = a->imag + b->imag;
}

// 輸出
void printComplex(const Complex* a) {
    cout << a->real << "+" << a->imag << "i" << endl;
}

int main() {
    Complex a = {1, 3};
    Complex b = {5, 2};
    Complex c;

    addComplex(&a, &b, &c);
    printComplex(&c);
    return 0;
}
```

4. 結構標題檔

當資料比較大的時候，struct 本身會寫在.h 檔，再去做使用。

```
.
├── complex.h
└── test.cpp (主檔案)
```

```
// complex.h
struct Complex {
    int real;
    int imag;
};
```

```
// test.cpp
```

```cpp
#include <iostream>
#include "complex.h"
using namespace std;

// 加法函式
void addComplex(const Complex* a, const Complex* b, Complex* c) {
    c->real = a->real + b->real;
    c->imag = a->imag + b->imag;
}

// 輸出函式
void printComplex(const Complex* a) {
    cout << a->real << "+" << a->imag << "i" << endl;
}

int main() {
    Complex a = {1, 3};
    Complex b = {5, 2};
    Complex c;

    addComplex(&a, &b, &c);
    printComplex(&c);

    return 0;
}
```

5. enum（列舉）

enum 是 列舉型別（enumeration），用來定義一組具名稱的整數常數，讓你的程式碼更易讀、更具意義。

5.1 範例：列出一週的列舉項目

enum class 優於傳統 enum，因為它不會把成員放進全域命名空間，更安全、更不容易衝突

```cpp
#include <iostream>
using namespace std;

// C++11：更安全的 enum 寫法（避免衝突）
enum class Week {
    SUN, MON, TUE, WED, THU, FRI, SAT
};
```

```
int main() {
    // 注意：Week 是 strongly typed，需要明確轉型才能用 int
    for (int i = static_cast<int>(Week::SUN); i < static_cast<int>(Week::SAT); ++i) {
        cout << "今天是星期：" << i << endl;
    }
    return 0;
}
```

5.2 範例：選擇你喜歡的顏色

```
#include <iostream>
using namespace std;

enum class Color {
    RED = 1,
    GREEN,
    BLUE
};

int main() {
    int input;
    cout << "請選擇你最喜歡的顏色（紅 1 綠 2 藍 3）：";
    cin >> input;

    Color myColor = static_cast<Color>(input); // 整數轉 enum

    switch (myColor) {
        case Color::RED:
            cout << "你喜歡紅色" << endl;
            break;
        case Color::GREEN:
            cout << "你喜歡綠色" << endl;
            break;
        case Color::BLUE:
            cout << "你喜歡藍色" << endl;
            break;
        default:
            cout << "輸入錯誤，請選擇 1~3" << endl;
    }
```

```
    return 0;
}
```