08-自訂型別 (struct)

8-1 struct

1. 自訂型別 struct

在 C++ 中，我們可以把多個彼此相關的資料包在一起，創造出一個全新的型別。

例如，一位學生可能有以下資料：

- 姓名
- 年齡
- 成績

如果分別用三個變數來表示，資料容易分散，必須想辦法維持追蹤同一個學生的姓名、年齡和成績。

`struct` 是一種**自訂型別**，可以把多個彼此相關的資料包在一起，創造出一個全新的型別。

```
struct Student {
    string name;
    int age;
    double score;
};
```

這表示我們建立了一個名為 `Student` 的型別，它裡面有三個成員：`name`、`age`、`score`。

---

2. 為什麼需要 struct

`struct` 很適合用來表示「一筆完整資料」，例如：

- 學生資料
- 座標點
- 商品資訊
- 日期時間

學會 `struct` 之後，你會開始從「很多零散變數」進步到「有結構的資料設計」，這是寫大型程式的重要基礎。

---

3. 基本語法

宣告

```
struct Person {
    std::string name;
    int age;
};
```

語法重點：

- `struct` 是關鍵字
- `Person` 是型別名稱
- 大括號裡面放的是成員變數
- 最後要有分號 `;`

---

4. 建立變數與存取成員

宣告完 `struct` 之後，就可以像一般型別一樣建立變數。

```
Person p1;
```

使用 `.` 來存取成員：

```
p1.name = "Alice";
p1.age = 18;
```

範例：

```
#include <iostream>

using namespace std;

struct Person {
    string name;
    int age;
};

int main() {
    Person p1;
    p1.name = "Alice";
    p1.age = 18;

    cout << "Name: " << p1.name << endl;
    cout << "Age: " << p1.age << endl;

    return 0;
}
```

執行結果：

```
Name: Alice
Age: 18
```

---

5. 初始化（給定初值）

除了先宣告再指定值，也可以在建立時直接初始化。

```
Person p1 = {"Bob", 20};
```

這相當於：

- `p1.name = "Bob"`
- `p1.age = 20`

```
Person p2 = {"Cindy", 22};
cout << p2.name << ", " << p2.age << endl;
```

---

6. struct 用於函數的參數

你可以把整個 `struct` 傳進函式，讓程式更清楚。

```
#include <iostream>

using namespace std;

struct Person {
    string name;
    int age;
};

void printPerson(Person p) {
    cout << "Name: " << p.name << endl;
```

```
    cout << "Age: " << p.age << endl;
}

int main() {
    Person p = {"David", 25};
    printPerson(p);

    return 0;
}
```

由於函數預設是以 **傳值呼叫(call by value)** 方式將參數傳入，如果該 struct 裡的資料很大，例如包含學生照片，通常會改用 **傳參考呼叫(call by reverence)**，避免整份資料被複製：

```
void printPerson(Person& p) {
    cout << "Name: " << p.name << endl;
    cout << "Age: " << p.age << endl;
}
```

由於傳參考會讓被傳入的外部 struct 變數，可以在函數內被修改。若要確保不被修改，可以像這樣在參數前加上 `const` 標示其為常數。

```
void printPerson(const Person& p) {
    cout << "Name: " << p.name << endl;
    cout << "Age: " << p.age << endl;
}
```

---

7. 陣列中的 `struct`

`struct` 很常和陣列搭配使用，用來儲存多筆同類型資料。

```
#include <iostream>

using namespace std;

struct Student {
    string name;
    int score;
};

int main() {
    Student students[3] = {
        {"Amy", 90},
        {"Brian", 85},
        {"Chris", 92}
    };

    for (int i = 0; i < 3; i++)
    {
        cout << students[i].name << ": " << students[i].score << endl;
    }

    return 0;
}
```

這種寫法在成績系統、通訊錄、商品清單中都很常見。

---

8. 巢狀 struct

`struct` 裡面還可以再放另一個 `struct`。

```
struct Date {
    int year;
```

```
    int month;
    int day;
};

struct Student {
    string name;
    Date birthday;
};
```

使用方式：

```
Student s;
s.name = "Helen";
s.birthday.year = 2005;
s.birthday.month = 8;
s.birthday.day = 15;
```

這讓複雜資料可以分層整理，結構更清楚。

---

9. 指標與 `struct`

如果你有一個指向 `struct` 的指標，在存取其成員時要用 `->` 而不是 `.`。

```
Person p = {"Ivy", 21};
Person* ptr = &p;

cout << ptr->name << endl;
cout << ptr->age << endl;
```

比較

```
p.name.       // 物件存取成員
(*ptr).name   // 物件存取成員
ptr->name     // 指標存取成員
```

---

10. 一個完整綜合範例

```
#include <iostream>

using namespace std;

struct Student {
    string name;
    int age;
    double score;

    void printInfo() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "Score: " << score << endl;
    }
};

int main() {
    Student s1 = {"Jack", 18, 95.5};
    s1.printInfo();

    return 0;
}
```

這個例子同時用到了：

- `struct` 宣告
- 成員變數
- 初始化
- 成員函式
- 物件呼叫函式

---

11. `struct` 的使用時機

當你遇到以下情境時，可以考慮使用 `struct`：

- 一筆資料有多個欄位
- 這些欄位彼此有關聯
- 你希望程式更容易閱讀與維護

例如：

- `Point`：座標 `(x, y)`
- `Book`：書名、作者、價格
- `Employee`：姓名、編號、薪資

---

12. 小專案練習：

專案背景

你要做一個小型的收銀系統。系統會：

- 建立一張發票（含建立時間與稅率）
- 掃描商品（品名與價格）
- 若商品重複出現，數量要累加
- 計算未稅金額與含稅總金額
- 最後列印完整發票

你會學到什麼

- 使用 struct 表示資料模型（Item、Date、Time、Invoice）
- 用函式分工（prepareInvoice、scanItem、printInvoice）
- 使用固定大小陣列儲存品項
- 在迴圈中搜尋重複資料
- 更新數量與金額
- 使用 C++ 標準函式取得目前日期時間

專案檔案

- main.cpp：主要程式
- in.txt：測試輸入資料（可用重導向測試）

功能需求

1. 初始化發票

- 建立空白發票
- 讀取目前日期與時間
- 設定稅率（例如 0.05）
- amount 初始為 0
- totalDue 初始為 amount * (1 + taxRate)

2. 掃描商品

- 每次輸入一筆：name price
- 若 name 已存在：
  - quantity + 1
  - amount 只加上本次 price
- 若 name 不存在：

    - 新增一筆 Item{name, price, 1}
    - amount 加上 price
- 每次更新 amount 後都要重算 totalDue

3. 列印發票

- 顯示日期與時間
- 顯示所有品項（品名、價格、數量）
- 顯示 amount、taxRate、totalDue

建議輸入格式

第一行輸入整數 n（要掃描幾筆） 接著輸入 n 行，每行一筆：

- name price

範例：

```
10
Apple 35.5
Banana 12
Milk 45
Bread 30
Apple 35.5
Eggs 68
Coffee 120
Bread 30
Orange 25
Milk 45
```

建議輸出格式

```
Date: 2026/4/10
Time: 10:7:56
------------------------------
Items:
- Apple: $35.5 x 2
- Banana: $12 x 1
- Milk: $45 x 2
- Bread: $30 x 2
- Eggs: $68 x 1
- Coffee: $120 x 1
- Orange: $25 x 1
------------------------------
Amount: $446
Tax Rate: 5%
Total Due: $468.3
```

如何測試

你可以用輸入重導向測試：

```
main.exe < in.txt
```

驗收標準

- 程式可成功編譯執行
- 重複品項會正確累加 quantity
- amount 不會因重複品項而重複倍算
- totalDue 計算正確（amount * (1 + taxRate)）
- 發票輸出格式清楚可讀

提示

- 先完成 prepareInvoice，再做 scanItem，最後做 printInvoice
- scanItem 先用 for 迴圈找同名品項

- 找到就更新並 return，沒找到才新增
- 注意陣列上限（itemList 最多 100 筆）
- 若輸入失敗（cin 失敗）要考慮防呆

額外挑戰

- 若同名商品出現不同價格，定義你自己的規則（拒絕、覆蓋、或視為不同品項）
- 顯示小計欄位（price * quantity）
- 將稅金獨立列印（tax = totalDue - amount）
- 改用 vector 取代固定長度陣列 - 需使用到還沒教的 STL 容器 vector
- 支援含空白的品名（例如 "Green Tea"） - 需使用到還沒教的 getline 函數

8-2 小專案參考解答

```
#include <iostream>
#include <ctime>

using namespace std;

struct Item
{
    string name;
    double price;
    int quantity;
};

struct Date
{
    int year;
    int month;
    int day;
};

struct Time
{
    int hh;
    int mm;
    int ss;
};

struct Invoice
{
    Item itemList[100];
    int countOfItemList;
    Date date;
    Time time;

    double amount;

    double taxRate;
    double totalDue;
};

Invoice prepareInvoice(double taxRate)
{
    Invoice invoice{};

    // get & set current date/time
    time_t now = time(nullptr);
    tm localTime{};
    localtime_s(&localTime, &now);

    invoice.date.year = localTime.tm_year + 1900;
    invoice.date.month = localTime.tm_mon + 1;
    invoice.date.day = localTime.tm_mday;

    invoice.time.hh = localTime.tm_hour;
    invoice.time.mm = localTime.tm_min;
    invoice.time.ss = localTime.tm_sec;

    invoice.countOfItemList = 0;
    invoice.amount = 0.0;
    invoice.taxRate = taxRate;
    invoice.totalDue = invoice.amount * (1.0 + invoice.taxRate);

    return invoice;
}
```

```
void printInvoice(const Invoice& invoice)
{
    printf("Date: %d/%d/%d\n", invoice.date.year, invoice.date.month, invoice.date.day);
    printf("Time: %d:%d:%d\n", invoice.time.hh, invoice.time.mm, invoice.time.ss);

    printf("------------------------------\n");
    printf("Items:\n");
    for (int i = 0; i < invoice.countOfItemList; ++i)
    {
        const Item& item = invoice.itemList[i];
        printf("- %s: $%.2f x %d\n", item.name.c_str(), item.price, item.quantity);
    }

    printf("------------------------------\n");
    printf("Amount: $%.2f\n", invoice.amount);
    printf("Tax Rate: %.2f%%\n", invoice.taxRate * 100);
    printf("Total Due: $%.2f\n", invoice.totalDue);
}

void scanItem(Invoice& invoice)
{
    string name;
    double price;

    cin >> name;
    cin >> price;

    for (int i = 0; i < invoice.countOfItemList; ++i)
    {
        if (invoice.itemList[i].name == name)
        {
            invoice.itemList[i].quantity += 1;
            invoice.amount += price;
            invoice.totalDue = invoice.amount * (1.0 + invoice.taxRate);
            return;
        }
    }

    Item newItem{name, price, 1};
    invoice.itemList[invoice.countOfItemList++] = newItem;
    invoice.amount += price;
    invoice.totalDue = invoice.amount * (1.0 + invoice.taxRate);
}

int main()
{
    double taxRate = 0.05;

    Invoice invoice = prepareInvoice(0.05);

    int n;
    cin >> n;
    for(int i=0; i<n; ++i)
    {
        scanItem(invoice);
    }
    printInvoice(invoice);

    return 0;
}
```