

المحاضرة 13 · التمرير والنطاق

13

معاملات الدوال والمجال: مَنْ يملك القيمة؟

لماذا أعدل x داخل دالة ثم أجده لم يتغير خارجها؟

تطبيق عملي

أمثلة C++17

رسم توضيحي

فهم قبل الحفظ

مستوحاة من برمجة 1 · صفحات المصدر 142-151 · إعداد تربوي: م.براءة علي رحيمو

++C

01 · المشكلة أولاً

قبل التعريف... لماذا نحتاج هذه الفكرة؟

لماذا أعدل x داخل دالة ثم أجده لم يتغير خارجها؟

مشهد من الواقع

لديك رصيد وتريد تحديثه عبر دالة؛ هل أرسلت نسخة من الرصيد أم سمحت للدالة بالعمل على القيمة الأصلية؟

أين المشكلة في الطريقة الساذجة؟

الخلط بين التمرير بالقيمة والتمرير بالمرجع يؤدي إلى أخطاء غير ظاهرة. كما أن متغيراً محلياً قد لا يكون مرئياً خارج دالته.

كيف تساعدنا الفكرة؟

نفق بين `pass-by-value` و `pass-by-reference`. ونتبع Scope والمتغيرات المحلية، ونفضل تمرير القيم بوضوح.

شبهها بما تعرفه: أعطيت صديقك صورة مستند (قيمة) لا تغيّر الأصل؛ أما إعطاؤه الملف نفسه (مرجع) فالتعديل يصل إلى الأصل.

لا تحفظ كلمة جديدة قبل أن تعرف المشكلة التي وجدت لتساعد في حلها.

02 · النموذج العقلي

اجعل المفهوم مرئيًا

ثلاث أفكار رئيسية ثم مسار واضح يربط بينها.

Pass by value

الدالة تستقبل نسخة؛ تغيير الوسيط المحلي لا يغير متغير المستدعي.

(&) Pass by reference

الوسيط يشير إلى متغير المستدعي ويمكنه تغييره.

Scope

نطاق صلاحية رؤية الاسم واستعماله، مثل المتغير المحلي داخل الدالة.

الرسم التوضيحي: كيف تعمل الفكرة؟



السؤال الأهم: لماذا أعدل x داخل دالة ثم أجده لم يتغير خارجها؟

03 · مثال عملي أول

شغل وافهم كل سطر

نسخة لا تغيّر الأصل

C++17 / source.cpp

```
#include <iostream>
using namespace std;
void addTen(int x) { x += 10; }
int main() {
    int score = 5;
    addTen(score);
    cout << score;
}
```

الخرج المتوقع 5

لماذا كتبنا البرنامج بهذا الشكل؟

x نسخة محلية؛ رغم رفعها بمقدار 10 فإن score في main لا يتغير.

💡 قبل تجربة البرنامج: حاول توقع النتيجة على الورق، ثم قارن توقّعتك بالخرج الفعلي.

04 · مثال إضافي

نفس المبدأ، حالة استخدام مختلفة

مرجع يغيّر الأصل

C++17 / source.cpp

```
#include <iostream>
using namespace std;
void addTen(int& x) { x += 10; }
int main() {
    int score = 5;
    addTen(score);
    cout << score;
}
```

الخرج المتوقع 15

ما الذي نتعلمه من المثال الثاني؟

علامة & في وسيط الدالة هنا تعني مرجعًا يسمح بتعديل المتغير الأصلي.

جرب تغيير قيمة واحدة فقط، ثم توقّع أثرها: هل تغيّر مسار البرنامج أم الناتج فقط؟

05 · مختبر الأخطاء

الخطأ ليس للحفظ... اكتشف سببه

قارن الكود الخاطئ بالصحيح ولا تكتفِ بمعرفة الإجابة.

الكود الذي يحتاج انتباهًا

```
void setToZero(int n){ n = 0; } // أتوقع تغيير الأصل
```

تصحيح وتفسير

```
void setToZero(int& n){ n = 0; } // تعديل الأصل
```

لماذا يُعد هذا خطأ؟

التمرير بالقيمة يعطي نسخة؛ استخدام & في تعريف معامل المرجع يتطلب أن يكون التعديل مقصودًا.

جرب بنفسك

تحدي 01

جرب swap(int&,int&) لتبديل قيمتين.

تحدي 02

ما الذي يحدث لو عرّفت int temp داخل دالة ثم استخدمته في main؟

تحدي 03

ناقش متى نفضل const int & عند استقبال قيمة دون تعديل.

06 · تثبيت الفهم

ماذا يجب أن يبقى في ذاكرتك؟

لا تقرأ القواعد فقط؛ اربط كل قاعدة بخطأ تمنعه أو مشكلة تحلها.

✓ التعديل بالمرجع أثر جانبي يجب فهمه.

✓ المتغير المحلي لا يتاح بالاسم في كل البرنامج.

✓ مرر بالقيمة عندما لا تريد تعديل الأصل.

الرسم التوضيحي: كيف تعمل الفكرة؟



تحضير الكويز التفاعلي

قبل فتح اختبار هذه المحاضرة، حاول شرح المثال الأول لشخص آخر دون النظر إلى الشيفرة. إذا فهم لماذا تعمل الخطوات، فأنت جاهز.

الجسر إلى المحاضرة التالية

سنختتم السلسلة بفهم عناوين الذاكرة والمؤشرات وعلاقتها بما تعلمناه.

المرجع: «برمجة 1» لم. براءة سراقبي، الصفحات 142-151 من الملف المرفق. الأمثلة والشروح الإضافية صيغت للأغراض التعليمية وتستخدم C++17.