

المحاضرة 04 · اختيار الأداة

04

# لماذا C++؟ من لغة الآلة إلى برنامج قابل للتنفيذ

لماذا لا يكتب المبرمجون الأوامر على شكل أصفار وآحاد؟

تطبيق عملي

أمثلة C++17

رسم توضيحي

فهم قبل الحفظ

مستوحاة من برمجة 1 · صفحات المصدر 31-39 · إعداد تربوي: م.براءة علي رحيمو

C++

## 01 · المشكلة أولاً

# قبل التعريف... لماذا نحتاج هذه الفكرة؟

لماذا لا يكتب المبرمجون الأوامر على شكل أصفار وآحاد؟

## مشهد من الواقع

عندما تريد أن تطلب من الحاسوب حساب مساحة دائرة، أنت تفكر بالصيغة  $\pi r^2$  لا بسلسلة طويلة من الأرقام الثنائية.

## أين المشكلة في الطريقة الساذجة؟

الآلات تفهم تعليمات منخفضة المستوى، بينما الإنسان يحتاج أسماء ومعاني وتنظيمًا ليقبل الخطأ.

## كيف تساعدنا الفكرة؟

تعطيك ++C تعليمات أقرب لفهمنا، ثم يقوم Compiler ومراحل البناء بتحويل المصدر إلى برنامج قابل للتنفيذ.

**شبهها بما تعرفه:** المترجم البرمجي مثل وسيط يترجم الوصفة إلى خطوات تفهمها آلة المطبخ، لكن لا يصلح منطق الوصفة الخاطئ.

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.

## Source Code

ملف نصي مثل main.cpp يكتب فيه المبرمج.

## Compiler

يترجم البرنامج ويتحقق من قواعد اللغة وأنواع كثيرة من الأخطاء.

## Linker

يجمع الوحدات المترجمة والمكتبات لإنتاج الملف التنفيذي.

## خريطة المفهوم: من الفكرة إلى التنفيذ



**السؤال الأهم:** لماذا لا يكتب المبرمجون الأوامر على شكل أصفار وآحاد؟

## 03 · مثال عملي أول

## شغل وافهم كل سطر

برنامج صغير بلغة بشرية القراءة

C++17 / source.cpp

```
#include <iostream>
using namespace std;
int main() {
    double radius = 3;
    double area = 3.14159 * radius * radius;
    cout << "Area = " << area;
}
```

الخرج المتوقع تقريباً Area = 28.2743

## لماذا كتبنا البرنامج بهذا الشكل؟

اللغة عالية المستوى تعبّر مباشرة عن المعادلة، بينما تفاصيل تنفيذ العمليات تُترك لسلسلة البناء.

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

## 04 · مثال إضافي

# نفس المبدأ، حالة استخدام مختلفة

ما الذي يفعله المترجم عند وجود خطأ؟

• • • C++17 / source.cpp

```
#include <iostream>
using namespace std;
int main() {
    int tickets = 5;
    tickets = tickets - 2;
    cout << "Remaining: " << tickets;
}
```

الخروج المتوقع Remaining: 3

## ما الذي نتعلمه من المثال الثاني؟

اختبر تغيير اسم tickets في سطر واحد؛ ستظهر مشكلة تعريف اسم عند الترجمة.

جرب تغيير قيمة واحدة فقط، ثم توقع 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 انتباهاً

```
int main() { cout << unknown; }
```

تصحيح وتفسير

```
int main() { int unknown=7; cout << unknown; }
```

## لماذا يُعد هذا خطأ؟

لا ينقذنا مترجم ++C من اسم غير معرّف؛ ووجود البرنامج التنفيذي لا يثبت صحة الحل.

## جرب بنفسك

### تحدي 01

قارن متى تستخدم لغة عالية المستوى ولغة تجميع.

### تحدي 02

جرب تغيير اسم متغير في المثال وحدد مرحلة اكتشاف الخطأ.

### تحدي 03

ارسم خطأ يبدأ Source وينتهي Executable.

## 06 · تثبيت الفهم

## ماذا يجب أن يبقى في ذاكرتك؟

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

✓ كل البرامج يجب أن تُنفَّذ بالنهاية على عتاد الحاسوب.

✓ ++C اختيار مفيد للسيطرة على الأداء والموارد، وليس الحل الوحيد لكل مشكلة.

✓ Compiler ليس مصحّحًا للمنطق.

### خريطة المفهوم: من الفكرة إلى التنفيذ



### 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

في المحاضرة القادمة سنفهم كيف تمثّل ++C الأرقام والحروف والحقائق داخل الذاكرة.

المرجع: «برمجة 1» لم. براءة سراقبي، الصفحات 31-39 من الملف المرفق، الأمثلة والشروح الإضافية صيغت للأغراض التعليمية وتستخدم ++C17.