C++ Base for Interacting with the Windows API

مقدمة عن C++

لغة **C++** هي لغة برمجة قوية وعالية الأداء، اتطورت في التمانينات على إيد **Bjarne Stroustrup** كامتداد للغة **C**.

اللي بيميز C++ هو إنها بتجمع بين:

- التحكم في **low-level memory** (يعني تقدر تشتغل على الذاكرة بشكل مباشر).
- مع **high-level abstractions** زي:
  - **classes** (يعني تقدر تكتب كود كائني التوجه OOP).
  - **templates** (إعادة استخدام الكود بذكاء).
  - **polymorphism** (نفس الـ function تشتغل بأكتر من شكل).

الخواص دي بتخلي C++ مناسبة جدًا لكتابة:

- برامج الـ **System-level**
- وبرامج الـ **Enterprise** الضخمة (زي الألعاب، الـ OS، وغيره).

ليه تتعلم C++؟

لما تتعلم C++، إنت مش بس بتتعلم syntax جديد، إنت بتبني أساس قوي في:

- إدارة الذاكرة (**Memory Management**)
- البرمجة الكائنية (**Object-Oriented Programming - OOP**)
- تحليل كفاءة الخوارزميات (**Algorithm Efficiency**)

وده كله بيساعدك في مجالات كتير زي:

- **Game engines** (زي Unreal Engine)
- تطبيقات **Desktop**
- أنظمة **Embedded systems**
- **Operating systems** زي Windows/Linux
- أدوات **Security** و **Pentesting**

ليه C++ ممتازة لبرمجة Windows API؟

الأول: يعني إيه Windows API؟

Windows API (Application Programming Interface) هو مجموعة من **C-based interfaces** مقدماها **Microsoft** علشان تتعامل مع الـ **Operating System مباشرة**.

الحاجات اللي ممكن تتحكم فيها من خلال Windows API:

- إدارة **Processes** و **Threads**
- شغل على **Files** و **Registry**
- التعامل مع **User Interface (UI)**
- إدارة **الذاكرة** و الـ **Security**
- تنفيذ **System Calls** والتعامل مع **Kernel**

طيب ليه نستخدم C++ بدل C بس؟

رغم إن Windows API مكتوبة بلغة C، لكن C++ بتقدم مزايا بتسهل الكتابة والتعامل مع الـ API:

السبب	الشرح
✅ **متوافقة تمامًا مع C**	C++ تقدر تنادي على أي Windows API function من غير أي wrapper أو overhead
🔁 **OOP support**	ممكن تغلف منطق API جوه **classes** reusable وسهلة التنظيم
🧠 **RAII & Smart Pointers**	تسهل عليك إدارة الموارد زي الـ handles والذاكرة بشكل آمن
⚙️ **Low-level Access**	تقدر تكتب **Inline Assembly** وتتحكم في الـ system من الداخل
🧰 **دعم قوي من الأدوات**	متكاملة تمامًا مع **Visual Studio** و **WinDbg** - أقوى IDEs و debuggers في بيئة Windows

طب وده يفيدك في إيه كـ Red Teamer أو Security Engineer؟

لو شغلك في الحاجات دي:

- **Red Teaming**
- **Driver Development**
- **EDR Evasion Research**

يبقى C++ هي سلاحك الأساسي علشان:

- تتحكم بالكامل في **الذاكرة** و **الـ Function Pointers**
- تكتب **System Calls** بنفسك
- تشتغل على **Low-level manipulation** بأعلى مستوى من التحكم والـ **Stealth**

ii

🧠 مدخل شامل لـ C++ بأسلوب مبسط

1. البنية الأساسية لأي برنامج C++

أي برنامج C++ بيبدأ بالـ function الأساسية `main()`، ودي تعتبر **نقطة البداية** للتنفيذ.

قبلها دايمًا بنستخدم `#include` علشان نضيف الـ **standard libraries** اللي محتاجينها زي:

```
#include <iostream>
```

الكود بيشتغل **بترتيب تسلسلي**، وغالبًا بينتهي بـ `return 0;` علشان نعرف الـ OS إن البرنامج خلص بنجاح.

2. الإدخال والإخراج (I/O)

في C++، التعامل مع الـ **input/output** بيتم عن طريق **streams**:

- `cin` ← لإدخال البيانات (عادةً من الكيبورد)
- `cout` ← لعرض البيانات (على الـ console)

الاتنين دول جايين من الـ **std namespace**، فبنكتب مثلًا:

```
std::cout << "Hello, world!";
```

3. المتغيرات وأنواع البيانات

C++ لغة **strongly typed**، يعني لازم كل متغير يتحدد نوعه من الأول:

النوع	الوصف
`int`	أعداد صحيحة
`float`	أعداد عشرية
`char`	حرف واحد
`bool`	صح أو غلط (true/false)

المتغيرات بتتخزن إما في:

- **stack** ← للحاجات المؤقتة والمحلية.
- **heap** ← للتخصيص الديناميكي باستخدام `new`.

4. التحكم في سير البرنامج (Control Flow)

C++ بتوفر أدوات تساعدك تتحكم في منطق البرنامج:

- **Conditionals:**

  `if`, `else if`, `else` ← بتنفذ أوامر حسب شرط معين.

- **Loops:**

  `for`, `while`, `do-while` ← تكرار لحد الشرط يبقى غلط.

- **Switch-case:**

  مثالي لو عندك متغير وليه كذا قيمة محتملة.

5. الدوال (Functions)

الدوال بتخليك **تعيد استخدام الكود** وتنظمه أكتر. ممكن:

- تمرر لها parameters.
- ترجع قيمة.
- تعمل Overloading (نفس اسم الدالة بس بـ parameters مختلفين).

```
int add(int a, int b) {
  return a + b;
}
```

6. المؤشرات والمراجع (Pointers & References)

- **Pointer** ← بيخزن عنوان متغير في الذاكرة.
- **Reference** ← اسم تاني لنفس المتغير (alias)، ومفيد لما تحب تمرر variable لدالة **by reference**.

الاتنين مهمين جدًا لما تشتغل على **system-level APIs**.

7. التخصيص الديناميكي للذاكرة

لو محتاج تحجز ذاكرة وقت التشغيل، استخدم:

- `new` ← تخصيص.
- `delete` ← تحرير الذاكرة.

```
int* ptr = new int;
delete ptr;
```

لو نسيت تستخدم `delete` ← هيحصل **memory leak**.

8. الـ Arrays و C-style Strings

- **Arrays:** هى sequence ثابت الحجم من العناصر.
- **C-style strings:** هى array of `char` بينتهي بـ ('\0') null character.

استخدم `strcpy_s` بدل `strcpy` علشان تتفادى **buffer overflow**.

9. Structs, Enums, و Type Aliases

النوع	الوصف
`struct`	بيجمع مجموعة متغيرات في كيان واحد
`enum`	مجموعة ثوابت لها أسماء مفهومة
`typedef` / `using`	بيعمل alias لأنواع بيانات معقدة

مفيدين جدًا لما تشتغل مع **Windows API** والـ **system structures**.

10. البرمجة الكائنية (OOP) في C++

C++ بتدعم الـ OOP بالكامل:

- **class** ← بتجمع البيانات والوظائف.
- **encapsulation** ← بتعمل Control access باستخدام `public` , `private` , `protected` .
- **constructors/destructors** ← هى Special methods to initialize and clean up objects
- **inheritance & polymorphism** ← لإعادة الاستخدام والسلوك الديناميكي.

11. الـ Namespaces والـ std::

كل حاجات المكتبة القياسية جوا **namespace** اسمه `std` .

تقدر تكتب:

```
std::cout << "Hello";
```

أو تستخدم:

```
using namespace std;
```

⚠️ لكن في المشاريع الكبيرة، يفضل تستخدم `::std` علشان تتفادى التضارب في الأسماء.

12. Smart Pointers (المؤشرات الذكية)

في C++، **Smart Pointers** هي مؤشرات ذكية بتساعدك تدير الذاكرة بشكل تلقائي وآمن، وبتوفر عليك استخدام `delete` يدويًا، وده بيقلل خطر **memory leaks** و**dangling pointers**.

الأنواع الأساسية :

🔹 `unique_ptr`

- ليه **ملكية حصرية** للذاكرة اللي بيأشر عليها.
- مينفعش يكون فيه أكتر من `unique_ptr` واحد لنفس الكائن.
- أول ما المؤشر يخرج من الـ scope → الذاكرة بتتحرر تلقائيًا.

```
#include <iostream>
#include <memory>

int main() {
    std::unique_ptr<int> ptr = std::make_unique<int>(10);
    std::cout << *ptr << std::endl;
    return 0;
}
```

🔹 `shared_ptr`

- بيستخدم **عداد مراجع (reference counting)**.
- أكتر من `shared_ptr` ممكن يأشر على نفس الكائن.
- الكائن بيتحرر لما آخر `shared_ptr` يخرج من الـ scope.

```
#include <iostream>
#include <memory>

int main() {
    std::shared_ptr<int> ptr1 = std::make_shared<int>(20);
    std::shared_ptr<int> ptr2 = ptr1; // الاتنين بيشاوروا على نفس العنوان

    std::cout << *ptr1 << " و " << *ptr2 << std::endl;
    return 0;
}
```

13. Templates

الـ **Templates** بتخليك تكتب كود generic يشتغل مع أي نوع:

```
template<typename T>
T add(T a, T b) {
  return a + b;
}
```

ودي الأساس للـ (Standard Template Library) **STL**.

14. Lambda Expressions

الـ Lambdas هي **دوال بدون اسم**:

```
auto add = [](int a, int b) { return a + b; };
```

مفيدة جدًا في:

- الفلترة
- الترتيب
- الـ callbacks
- الـ threading

15. Move Semantics (std::move)

لما تحب تنقل محتوى بدل ما تعمله copy (خصوصًا للحاجات الكبيرة):

```
std::string a = "Hello";
std::string b = std::move(a);
```

ده بيوفر وقت وكفاءة.

16. Exception Handling

C++ بتتعامل مع الأخطاء عن طريق:

```
try {
  // ممكن يحصل فيه خطأ
} catch (std::exception& e) {
  // تعامل مع الخطأ
}
```

بتخلي البرنامج يفضل شغال حتى لو حصلت مشكلة.

17. RAII (Resource Acquisition Is Initialization)

مبدأ في C++ بيقول إنك تربط الموارد بعمر الكائن:

- أول ما الكائن يتولد ← تاخد المورد.
- أول ما الكائن يخرج من الـ scope ← يتم تحرير المورد تلقائي.

مستخدم في:

- ملفات (`fstream`)
- Smart pointers
- Locks (`std::lock_guard`)

18. مكتبة STL

STL بتوفر حاويات جاهزة + خوارزميات قوية:

النوع	مثال
Containers	`vector` , `map` , `set` , `queue`
Algorithms	`sort` , `find` , `for_each`
Iterators	تتنقل داخل الحاويات بسهولة

19. التحضير لبرمجة Windows API

لو عايز تبرمج على مستوى النظام (Windows)، فـ C++ هي اللغة المثالية علشان:

- تقدر تتعامل مع الذاكرة بشكل مباشر.
- تتحكم في الـ structs والمؤشرات.
- متوافقة 100% مع C (اللي مكتوب بيها WinAPI).

مهام شائعة:

- تضمين `<Windows.h>`
- استخدام wide strings زي `wchar_t` و `"L"Hello`
- إدارة أنواع زي `LPVOID` , `DWORD` , `HANDLE`
- استدعاء دوال زي:
  - `CreateProcessW`
  - `VirtualAlloc`
  - `MessageBoxW`

ضروري جدًا تكون فاهم:

- المؤشرات
- الـ memory models
- تراكيب البيانات المستخدمة في الكيرنل

📋 ملخص المواضيع

التصنيف	المواضيع المغطاة
Fundamentals	Structure, Variables, I/O
Control Structures	if, for, while, switch
Functions	Declaration, Parameters, Overloading
Memory & Pointers	Pointers, Dynamic Allocation, nullptr
Data Aggregation	Arrays, Structs, Enums, Type Aliases
OOP	Classes, Constructors, Methods, Encapsulation
Advanced Concepts	Smart Pointers, Templates, Lambdas, Exceptions, Move
RAII & Safety	Scope-Based Resource Management
STL	Vectors, Iterators, Algorithms
Windows Integration	Windows.h, Handles, Wide Strings, Structs

تم بحمد الله

Contact

For questions, feedback, or support:

- X (Formly Twitter): @00xmora
- Linkedin: @00xmora