C 语言学习小册

——轻量级 C 语言入门教程

站长微信

「关于教程」

这本「C 语言学习小册」由站长亲自创作，并长期保持更新。

这不是传统的教科书，而是你枕边的故事书，所以它阅读起来非常轻松，一点也不沉重。这也不是百科全书，而是简明教程，所以它懂得取舍，只对重要知识进行讲解，避免又臭又长。

读书，还是小册好；小就是轻，轻就是爽。

相信我，这本小册将帮你踹开编程的大门，让你对 C 语言本身，以及 C 语言的周边都有一个整体上的认知。

「目录」

1、C 语言到底是什么玩意？

C 语言发布于 1972 年，今年是 2023 年，它已经 50 多岁了，是一个不折不扣的“大叔”或者“大爷”。

但是呢，现在的 C 语言依然非常流行，每一个科班出身的程序员都必学 C 语言。在 2023 年 10 月份发布的世界编程语言排行榜中，C 语言拿了一个第二名的好成绩，占据了 12.08% 的市场份额。

第一名是 Python，占据了 14.82% 的市场份额，比 C 语言多一丢丢。

1972 年，一个叫丹尼斯·里奇（Dennis Ritchie）的程序员大佬，在美国著名的贝尔实验室中开发了 C 语言。C 语言的初衷很简单，就是被设计成一种系统编程语言，帮助团队开发 UNIX 操作系统。

UNIX 可以说是现代操作系统的开山鼻祖，你所听说过的大部分操作系统都受到 UNIX 的影响，比如：

- Linux 是模仿 UNIX 进行开发的，可以说是 UNIX 的山寨版；
- macOS、iOS、iPadOS、watchOS 是在 UNIX 的基础上衍生出来的，可以说是基于 UNIX 进行的二次开发；
- Android、HarmonyOS 都使用了 Linux 内核；
- 深度（Deepin）、优麒麟（Kylin）、统信 UOS 等国产操作系统的内核都是 Linux。

为了让初学者更加精准地理解 C 语言，这里我们不妨给 C 语言贴几个标签。

标签 1：中级语言

很多教材都把 C 语言定位成高级语言，但是从现在的编程环境来看，这种说不再准确了。C 语言最好的定位应该是中级语言。

第一，C 语言比较简单，它的语法特性并不丰富，不支持面向对象，不支持泛型编程，不支持函数编程，而 Java、Python、C#、JavaScript 等高级语言，至少支持其中的两种。

语法特性多了，开发起来就比较方便，维护大型项目也更加得心应手。

第二，C 语言语法严格，编写 C 语言代码需要非常小心，稍微不注意就会出错，比如指针。而 Python、JavaScript、PHP 等高级语言，代码书写非常灵活，基本不太关注细枝末节，死扣那些和项目功能无关的边边角角。

语法要求严格的话，就会耗费过多精力，写起代码来效率就低。

第三，C 语言不支持垃圾内存回收，需要自己管理内存，实在是太麻烦了，一不小心就会出现内存泄露（Memory leak）、段错误（Segmentation fault）等运行时错误。

虽然有一些工具（Valgrind、ASan 等）可以帮助检测类似的问题，但是这些工具使用起来也非常麻烦，还需要了解很多底层知识。更要命的是，工具也不是万能的，总有一些不常见的内存问题是无法检测的。

内存问题，是 C/C++ 程序员最头疼的问题。

第四，使用 C 语言开发对程序员的要求很高，程序员需要对计算机底层的执行细节有更深入的了解，比如内存布局、编译原理等，所以基础不扎实的程序员压根就玩不了 C 语言。

总起来说，C 语言在使用的灵活性和强大型方便，比不过 Java、Python、C#、JavaScript、PHP 等语言，所以它不高级；但是，C 语言又比汇编好用太多了，简直不是一个时代的产物，所以它也不低级。

既不高级也不低级，那就算是中级语言吧。

标签 2：通用语言

根据世界编程语言排行榜（TIOBE）的统计，能叫得上名字来的编程语言大概有 300 多种，其中常用的有 50 种，也就是榜单里面的前 50 名。

从应用范围上来说，这些编程语言大致可以分为两种：
- 一种是专用型语言，也就是针对某个特定领域而设计出来的语言；
- 另一种是通用型语言，它们可以开发多种类型的应用程序，而不是局限在某个特定的领域。

专用型编程语言举例：

编程语言	说明
PHP	专门用来开发网站后台。
JavaScript	最初只能运行在浏览器里面，用于开发网站前端，实现一些网页特效；后来有“好事的”程序员把它从浏览器里面提取出来，拿到操作系统上直接运行，这个时候 JavaScript 又变成了一种通用型的语言，可以干很多其它事情了，比如开发网站后来、GUI 程序、APP 或者一些小工具。
SQL	专门用于操作数据库的语言，可以实现数据的增删改查。
MATLAB	主要用于数值计算、科学计算和数据可视化等领域。它提供了许多工具箱，可以进行信号处理、图像处理、控制系统设计等任务。
R 语言	用于统计分析和数据科学，是一种开源的语言和环境，提供了许多数据分析和可视化的函数和工具。
VHDL/Verilog	用于数字电路设计，是一种硬件描述语言。它们允许开发人员描述数字电路的行为和结构，并用于电路仿真和综合等任务。

通用型编程语言举例：

编程语言	说明
Python	能做的事情很多，可以用于科学计算、数据分析、Web 开发、网络爬虫、人工智能、自动化运维等领域。
Java	可以用于网站开发、Android APP 开发、GUI 程序开发、云计算等领域。
Objective-C Swift	开发苹果系统上的应用程序，具体包括 iOS、macOS、watchOS、iPadOS 等。
Go 语言	Google 公司开发，可以用于网络编程、云计算、分布式系统开发、Web 后台开发、游戏后台开发、GUI 程序开发等领域。
C/C++	用于底层开发、单片机/嵌入式开发、游戏开发、GUI 程序开发等。

标签 3：高效语言

C 语言先被“翻译”成汇编代码，然后再由汇编器编译成机器代码，也就是 0 和 1 组成的那一堆东西。

C 语言语法特性并不丰富，不支持那些高级的玩法，没有乱七八糟的语法糖，什么东西都得由你自己实现，所以 C 语言生成的汇编代码比较简洁，比直接手写汇编代码也复杂不了多少。

因为简单，因为原始，所以运行效率高，而且是极高，比汇编也差不了多少，所以在很多场景下都可以代替汇编的工作。

C 语言的运行效率，不是 Python、Java、JavaScript、C#、PHP 这些高级语言所能比肩的，它们之间甚至会存在数量级的差距。

总结

C 语言是一种高效的、通用的中级语言。

C 语言是一名合格的程序员必须掌握的编程语言，很少有不了解 C 语言的大佬。

C 语言除了能让你学习编程相关的概念，带你走进编程的大门，还能让你明白程序的运行原理，比如，计算机的各个部件是如何交互的，程序在内存中是一种怎样的状态，操作系统和用户程序之间有着怎样的“爱恨情仇”，这些底层知识决定了你的发展高度，也决定了你的职业生涯。

如果你希望成为出类拔萃的人才，而不仅仅是码农，这么这些知识就是不可逾越的。也只有学习 C 语言，才能更好地了解它们。

2、C 语言到底能干什么？我列举了 8 种经典案例

虽然 C 语言执行速度极快，占用资源极少，但是它使用起来非常麻烦，完全没有 Java、Python、Go、JavaScript、C# 等方便和灵活，会严重拖慢项目的开发进度，所以，通常只有在「不得不」的情况下才会使用 C 语言。

再说得直白点，就是没得选了，才会使用 C 语言。

下面，我给大家列举了 8 种 C 语言的实际用途。

1. 单片机/嵌入式开发

这是目前使用 C 语言最广泛的一个领域，尤其是单片机开发，基本被 C 语言霸占了。

嵌入式开发是一个比较广泛的概念，通常来说包含两个方向：

- 单片机开发：由于硬件资源有限，通常不使用操作系统，让代码跑在裸机上；或者，仅仅安装一个简单的实时操作系统（RTOS），比如 FreeRTOS、µC/OS、RT-Thread 等。
- 一个是基于通用操作系统之上的开发，比如 Linux、Windows、Android 等。当然，这些操作系统都要经过裁剪和优化，以减少对资源的占用。这个才是通常所讲的嵌入式开发。

也就是说，单片机开发是嵌入式开发的一个分支。

几个例子：

- 家电（空调、自动洗衣机、遥控器）
- 汽车（ECU 发动机控制单元、BCU 车身控制单元、车辆信息检测）
- 医疗（心电图仪、血压仪、血糖仪、监护设备）
- 智能卡（门禁卡、公交卡、银行卡）
- 玩具（遥控汽车、摇摇椅）

2. 桌面软件开发

也叫 GUI 开发或者 PC 软件开发。

在实际应用中，C 语言通常用来开发某些关键模块或者效率模块，而不是开发一个完整的软件。这些模块一般对性能有着很高的要求，同时也关注资源消耗情况，除了 C 语言也没有其它更好的选择了。

几个例子：
- Office 早期版本使用 C 实现核心功能
- PS 早期版本使用 C 实现核心功能
- CAD 和 3D 建模软件使用 C 实现渲染和计算功能
- Notepad++、Sublime Text 使用 C 实现核心的编辑功能
- MySQL Workbench 和 SQL Server Management Studio 数据库客户端软件，使用 C 语言来实现数据库连接、查询、管理和界面

3. 底层开发

基础组件、核心算法、硬件驱动、通信协议的实现，都离不开 C 语言。

C 语言底层开发举例：
- 基础组件：文件系统、进程管理、用户界面（CLI+GUI）
- 核心算法：加密/安全算法（MD5、SHA、AES、RSA、SSL）、调度算法（线程/进程调度、内存页面置换）、LZ 压缩算法、CRC 和海明码等数据校验算法、随机数生成算法（又细分为多种方式）
- 硬件驱动：声卡驱动、显卡驱动、网卡驱动、蓝牙驱动、键鼠驱动、扫描仪/打印机驱动、USB 驱动
- 通信协议：TCP/IP 协议族（UDP、DNS、路由选择）、HTTP/HTTPS、SMTP/POP3/IMAP、FTP、SNMP、Bluetooth、NFC

4. 开发操作系统

这是 C 语言的初衷，它就是为开发操作系统而生的，UNIX、Linux、Windows 的内核就大量使用 C 语言。

5. 开发其它编程语言

有些编程语言的编译器（解释器）和标准库就使用 C 语言开发，比如 Python、PHP、Rust、Perl。

有些编程语言是在 C 语言的基础上进行的扩展，比如 C++、Objective-C、Swift、D 语言。

由于 C 语言具有可移植性，适应性强，有时也被用作不同编程语言的中间语言，这样不同编程语言之间就可以共享组件/模块。把 C 语言作为中间件的编译器有：
- Gambit（Scheme 语言的编译器和开发环境）
- BitC（系统级编程语言）

- GHC（Haskell 语言的编译器）
- Vala（基于 C 的语言，创建 GNOME 桌面程序）
- Squeak（基于 Smalltalk 的面向对象编程语言）

6. 信号处理

C 语言在电气工程领域也有很多用途，它可以使用信号处理算法来管理微处理器、微控制器等集成电路。

几个例子：
- 无线通信中的调制和解调，包括数字调制解调、射频调制解调。
- 数字信号处理（DSP），包括滤波、频谱分析、时频分析、数字滤波器设计、谱估计。

7. 音视频处理

C 语言的速度非常快，能够快速地对音频和视频数据进行处理。音频和视频数据通常比较大，需要高效的算法和数据结构来处理，而 C 语言运行速度非常快，能够及时处理这些数据。

C 语言提供了丰富的底层库和工具，如 FFmpeg、OpenCV 等，这些库和工具可以方便地对音频和视频数据进行编码、解码、剪辑、处理和转换等操作。

C 语言支持指针和位运算等底层操作，这些操作可以对音频和视频数据进行高效的操作，如数据拷贝、移位、变换等。这些操作对于实现一些高级算法和数据结构非常有帮助。

几个例子：
- 音频编解码：MP3、AAC、WAV
- 视频编解码：H.264、H.265、VP9
- 音频处理：音频滤波、均衡器、混响、降噪、语音识别、音频合成
- 视频处理：图像滤波、色彩空间转换、运动估计、视频编辑
- 媒体库：FFmpeg 和 OpenCV 可以用于编解码、格式转换、滤波、图像处理等

8. 数据库开发

数据库是软件领域的基础设施，它的性能直接影响整个应用程序的运行效率，所以必须使用一种高效的语言进行开发。

使用 C 语言开发的数据库有：MySQL、SQLite、PostgreSQL、Oracle Database、Microsoft SQL Server。

总结

站在现实的角度说，C 语言主要用作底层开发，或者关键组件的开发，或者贴近硬件的开发。这些开发场景都比较关注运行效率和硬件资源。

3、学编程难吗？多久能学会？

这篇文章主要是解答初学者的疑惑，没有信心的读者看了会吃一颗定心丸，浮躁的读者看了会被泼一盆冷水。

学编程难吗?

编程是一门技术，我也不知道它难不难，我只知道，只要你想学，肯定能学会。每个人的逻辑思维能力不同，兴趣点不同，总有一部分人觉得容易，一部分人觉得吃力。

在我看来，技术就是一层窗户纸，是有道理可以遵循的，最起码要比搞抽象的艺术容易很多。

但是，隔行如隔山，学好编程也不是一朝一夕的事，想“吃快餐”的读者可以退出编程界了，浮躁的人搞不了技术。

在技术领域，编程的入门门槛很低，互联网的资料很多，只要你有一台计算机，一根网线，具备初中学历，就可以学习，投资在 5000RMB 左右。

不管是技术还是非技术，要想有所造诣，都必须潜心钻研，没有几年功夫不会鹤立鸡群。所以请先问问你自己，你想学编程吗，你喜欢吗，如果你觉得自己对编程很感兴趣，想了解 APP 或网站是怎么做的，那么就不要再问这个问题了，尽管去学就好了。

多久能学会编程?

这是一个没有答案的问题。每个人投入的时间、学习效率和基础都不一样。如果你每天都拿出大把的时间来学习，那么两三个月就可以学会 C/C++，不到半年时间就可以编写出一些软件。

但是有一点可以肯定，几个月从小白成长为大神是绝对不可能的。要想出类拔萃，没有几年功夫是不行的。学习编程不是看几本书就能搞定的，需要你不断的练习，编写代码，积累零散的知识点，代码量跟你的编程水平直接相关，没有几万行代码，没有拿得出手的作品，怎能称得上“大神”。

每个人程序员都是这样过来的，开始都是一头雾水，连输出九九乘法表都很吃力，只有通过不断练习才能熟悉，这是一个强化思维方式的过程。

知识点可以在短时间内了解，但是思维方式和编程经验需要不断实践才能强化，这就是为什么很多初学者已经了解了 C 语言的基本概念，但是仍然不会编写代码的原因。

程序员被戏称为”码农“，意思是写代码的农民，要想成为一个合格的农民，必须要脚踏实地辛苦耕耘。

也不要压力太大，一切编程语言都是纸老虎，一层窗户纸，只要开窍了，就容易了。

浸泡理论

这是我自己独创的一个理论，意思是说：一个人要想在某一方面有所成就，就必须有半年以上的时间，每天花 10 个小时“浸泡”在这件事情上，最终一定会有所收获。

很多领域都是「一年打基础，两年见成效，三年有突破」，但是很多人在不到一年的时间里就放弃了，总觉得这个行业太难，不太适合自己。

轻言放弃是很可怕的，你要知道，第一次放弃只是浪费了时间，第二次放弃会打击你的信心，第三次放弃会摧毁你的意志，你就再也没有尝试的勇气了，“蹉跎人生”就是这么来的。

你也不要羡慕那些富二代官二代，你以为人生就是一次百米短跑，你赢了就是赢了，其实人生是一场接力赛，你的父辈祖辈都得赢，那些富二代官二代从好几十年以前就开始积累了。

所以，沉下一颗心来，从现在开始积累吧，有执念的人最可怕。

4、程序员必须要学习 C 语言吗？

C 语言是一个老古董了，初学者即使学了，一时半会也用不上，看起来有点鸡肋。然而，几乎所有大学的计算机/软件专业都将 C 语言作为必修课，这又是为什么呢？难道真的是因为大学教育落后吗？

当然不是！

C 语言是一门基础语言，很多其它的课程都依赖 C 语言；如果你不了解 C 语言，那么这些课程是学习不了的。

不妨举几个例子：

- 编译原理课程通常以 C 语言为例讲解，因为 C 语言的编译过程相对简单、规范和透明，适合教学。
- 数据结构课程通常使用 C 语言编程，因为 C 语言比较底层，能够让大家看到数据结构的各种细节。另外，数据结构是一种被频繁调用的组件，必须要追求效率，C 语言再合适不过了。
- 学习操作系统原理（内存、进程、线程、通信等）也要具备 C 语言基础，否则是学不明白的。

C 语言是一门面向计算机的语言，它能帮助我们快速了解底层。而其它的高级语言（Python、Java、C# 等）是面向用户的，它能让我们快速上手，搞出点实用的工具来，比如 PC 软件、网站、APP 等。

借助 C 语言学习原理，相当于修炼内功；使用其它语言开发程序，相当于精通招式。既有内功也有招式，才是一名合格的程序员。

从整体上讲，计算机软件大概可以分为两种：

- 一种是基础设施，比如操作系统、数据库、浏览器、云计算系统、大数据系统、编译器、编程语言、通信协议、区块链、服务器等。
- 一种是应用软件，比如桌面软件、APP、网站、小程序等。

内功不扎实的话，开发一般的应用软件是没问题的，这也是各家互联网公司正在做的事情。但是，要想开发高性能的软件，或者开发基础设施，那绝对是不行的，门都没有。

互联网已经不再是浪潮之巅，只搞简单的应用软件越来越没有前途，所以各家公司也在慢慢下沉，越来越重视根基了。

这意味着，编程不再是一种小把戏，而是逐渐演变成了一种技术，或者一种科研，你需要系统性地学习理论和基础，那种「上几个月培训班就能找份月入过万的工作」的时代已经越来越远了。

文章最后，我再给大家罗列一下软件相关的理论和基础：

- 数据结构
- 常见算法
- 简单的编译原理
- 程序运行流程
- 程序内存分布
- 进程/线程
- 网络通信（Socket）
- 动态/静态链接库
- 操作系统

不管将来从事开发还是研发，这些都是一个合格程序员应该具备的内功，只有掌握了这些，才算摸清了软件产业的脉络。

这些东西都是以 C 语言为基础的，不学 C 语言很难把它们搞清楚。

虽然工资高低不仅跟个人能力挂钩，还跟情商、机遇、市场挂钩，但是和半吊子程序员相比，功底扎实的程序员更容易找到高薪工作，更容易走得长远。

5、C 语言和 C++到底有什么关系？

C++ 读作“C 加加”，是“C Plus Plus”的简称。

顾名思义，C++ 就是在 C 语言的基础上增加了新特性，玩出了新花样，所以才说“Plus”，就像 Win11 和 Win10、iPhone 15 和 iPhone 15 Pro 的关系。

C 语言是 1972 年由美国贝尔实验室开发成功的，在当时算是高级语言，它的很多新特性都让汇编程序员羡慕不已，就像今天的 Go 语言，刚出生就受到追捧。C 语言也是“时髦”的语言，后来的很多软件都用 C 语言开发，包括 Windows、

Linux、Mac OS、MySQL、Python 等。

但是随着计算机性能的飞速提高，硬件配置与几十年前已有天壤之别，软件规模也不断增大，很多软件的体积都超过 1G，例如 PhotoShop、Visual Studio 等，用 C 语言开发这些软件就显得非常吃力了，这时候 C++ 就应运而生了。

C++ 主要在 C 语言的基础上增加了面向对象和泛型的机制，提高了开发效率，以适用于大中型软件的编写。

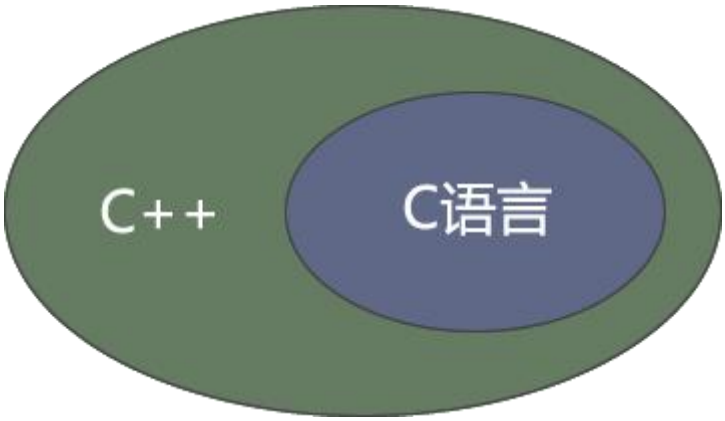
C 和 C++的血缘关系

早期并没有“C++”这个名字，而是叫做“带类的 C”。“带类的 C”是作为 C 语言的一个扩展和补充出现的，目的是提高开发效率，如果你有 Java Web 开发经验，那么你可以将它们的关系与 Servlet 和 JSP 的关系类比。

这个时期的 C++ 非常粗糙，仅支持简单的面向对象编程，也没有自己的编译器，而是通过一个预处理程序（名字叫 cfront），先将 C++ 代码“翻译”为 C 语言代码，再通过 C 语言编译器合成最终的程序。

随着 C++的流行，它的语法也越来越强大，已经能够很完善的支持面向对象编程和泛型编程。但是一直也没有诞生出新的 C++ 编译器，而是对原来 C 编译器不断扩展，让它支持 C++ 的新特性，所以我们通常称为 C/C++ 编译器，因为它同时支持 C 和 C++，例如 Windows 下的微软编译器(cl.exe)，Linux 下的 GCC 编译器。

也就是说，你写的 C、C++ 代码都会通过一个编译器来编译，很难说 C++ 是一门独立的语言，还是对 C 的扩展。



图：C 语言和 C++的关系

学习顺序

C++ 语法繁杂，是最恶心的编程语言，没有之一。如果针对没有任何编程经验的读者写一本 C++ 的书，那将是一项不小的任务，写出来的书也会非常厚。即使这样，也仅仅是在讲语法。

更重要的是，这些知识你很难全部吸收，会严重打击你的信心，失去学习的兴趣。

没有任何编程基础的读者，我建议先从 C 语言学起，不要贪多嚼不烂。有编程基础的读者，相信你自己能做出正确的判断。

学习编程是一个循序渐进的过程，不要期望一口吃个胖子。学习 C 语言，一来是学习它的语法，为 C++ 打基础，同时培养编程兴趣，二来是学习内存、编译和链接，弄清编程语言的内在机理。

每个初学者都经历过这样的窘境：已经学习了语法，明白了编程语言都有什么，也按照教程敲了不少代码，但是遇到实际问题就挂了，没有思路，不知道从何下手。说白了就是只会学不会用。

究其原因，就是实践少，没有培养起编程思维！学习知识容易，运用知识难！

等你熟悉了 C 语言，能编写出上百行的代码，就对编程有些概念了。这个时候再去了解 C++ 究竟在 C 语言基础上增加了什么，你就站在了一定的高度。

从“学院派”的角度来说，C++ 支持面向过程编程、面向对象编程和泛型编程，而 C 语言仅支持面向过程编程。就面向过程编程而言，C++ 和 C 几乎是一样的，所以学习了 C 语言，也就学习了 C++ 的一半，不需要从头再来。

咱们教程也是这样安排的：先讲解 C 语言，再讲解 C++，主要包括 C++ 和 C 语言的一些差别，以及面向对象编程和泛型编程。

结论

C++ 是在 C 语言的基础上扩展而来的，可以把 C 语言当成 C++ 的一个部分。

我建议先从 C 语言学起，打好基础了，再继续学习 C++。反正学习 C 语言就是在学习 C++，怕什么。

6、不要这样学习 C 语言，这是一个坑！

对于大部分初学者来说，学习 C 语言的目的是希望做一名合格的程序员，开发出靠谱的软件来。但是大家学了 C 语言的基本语法后，发现只能开发“黑底白字”的 DOS 程序，完全没有漂亮的界面和生动的交互。于是学数据结构，学算法，学操作系统，越陷越深，越来越难，最后迷茫了，不知道学 C 语言能做什么，认为学习编程很难，开始怀疑自己，甚至想放弃。

其实，这是很多初学者都会踩到的一个坑！

C 语言本身是一门很简单的语言，提供的实用功能不多，大部分要借助操作系统、第三方库、单片机来完成。也就是说，只学 C 语言基本什么也做不了，也基本找不到工作。

前面我们提到过，C 语言是一门通用性的语言，并没有针对某个领域进行优化，在实际项目中，C 语言主要用于较底层的开发，例如：

- 单片机和嵌入式属于软硬件的结合，有很多使用 C 语言的地方；
- Windows、Linux、Unix 等操作系统的内核 90% 以上都使用 C 语言开发；
- 开发硬件驱动，让硬件和操作系统连接起来，这样用户才能使用硬件、程序员才能控制硬件；
- 开发系统组件或服务，用于支撑上层应用；
- 编写 PHP 扩展，增强 PHP 的功能；
- 如果对软件某个模块（例如算法和搜索部分）的效率要求较高，也可以使用 C 语言来开发。

既然 C 语言的应用这么多，为什么很多读者觉得它什么也做不了呢？

我们先说一个概念，就是库（Library）。库就是编程专家写好的代码，我们可以拿来直接使用，这样能够节省开发成本，提高开发效率，并且库代码的执行效率、严谨性、安全性和规范性要明显优于我们自己编写的代码，市场上有很多优秀的库，有的收费，有的免费，我们要善于利用这些库，尽量不要重复造轮子。

库一般分为两种：

- 编程语言的开发者在开发编程语言的时候，一般都要预先写好常用的代码，或者说常用的功能，例如输入输出、数学计算、文件操作、网络操作、日期时间、错误处理、字符串处理等，这些由官方编写的库称为标准库（Standard Library），它们随编程语言一起发布，可以认识是编程语言的一部分。
- 有一些组织机构或者个人也会开发一些库，有的是为了盈利，有的是业余爱好，有的是本公司正在使用的代码，开源出来造福人类，这些库称为第三方库（Third-party Library）。

第三方库不是由官方开发，没有质量把控，良莠不齐，但是有相当一部分也非常优秀，已经得到了大家的认可，已经应用在大公司的项目开发中，这些库能够和标准库媲美。

标准库是我们在学习编程语言时就要一起学习的，例如 C 语言的输入输出、文件操作、日期时间、字符串处理、内存管理等都是标准库提供的功能，它们并不是 C 语言语法的内容。

如果一门编程语言的标准库强大，初学者经过简单的学习后就很容易开发出实用的项目。例如 Java，它的标准库包含了 GUI（图形界面）、图形处理、网络通信、网络服务器、HTML 解析、HTTP 协议、多线程、多进程、正则表达式、压缩文件、加密解密、数据校验、音频视频处理、数据库操作、XML 操作等常用功能，初学者学了以后立马就能够开发网站、开发 PC 软件，感觉很实用，也感觉学到了东西。

Python、C#、PHP、JavaScript、Ruby 等都是非常实用的语言，学了就能做出东西来。

反观 C 语言，它的标准库只有输入输出、文件操作、日期时间、数学计算等基本功能，都是在黑黑的控制台下进行的，跟网站、PC 软件、APP 等八竿子打不着，所以初学者觉得 C 语言没有用。

那么，C 语言到底能不能开发网站、PC 软件或者 APP 呢？

C 语言的标准库肯定不能干这些事情啦，就得依靠第三方库了，遗憾的是，C 语言的第三方库大都也是底层库，支持应用开发的库寥寥无几，只有一个 GTK 库能够开发出 PC 软件来，而没有与网站开发和 APP 开发相关的库。

GTK 库在 PC 软件开发中也很少用了，PC 软件开发已经是 C++、C#、Java、Python 等其它语言的天下了。换句话说，开发 PC 软件基本不使用 C 语言，而是使用 C++、C#、Java、Python 等其它语言。

记住，C 语言几乎不用来做软件、网站、APP 等这些应用层开发，其它的编程语言能够更好地完成任务，没必要非得使用 C 语言，C 语言基本都是用来做底层开发，也就是看不见摸不着的、在后台默默提供服务的那些项目，而这样的项目对初学者来说基本没有实用价值，初学者也不知道它们该怎么使用。

初学者想要的 C 语言没有，C 语言能做的初学者用不到，就是这种矛盾导致初学者非常迷茫。

有人可能会问，C 语言不是还可以用来开发单片机或者嵌入式吗？是的没错，但是这个方向是软硬件结合的，不是在我们的电脑上进行开发，而是在特殊的开发板上进行开发，并且还需要学习数字电路、模拟电路、汇编、ARM、Linux 等方面的知识，只学 C 语言也没有用武之地。

如果你觉得学了 C 语言没用，那么恭喜你，你是对的，应用层的开发一般真的用不上它。

但是，没用也要学，学习 C 语言并不一定是要应用它，C 语言可以夯实你的编程基础，尤其是数据结构、算法、内存、线程、进程、通信、操作系统等底层的计算机知识，没有 C 语言基础是学不好的。

这些底层知识并不一定能够直接应用在实际开发中，但是它们会让你有底气，会让你透彻地理解编程概念，会让你站的“低”看得远，会让你避免很多低级错误，会让你心中有“架构师”的思维。不学 C 语言是码农，学了 C 语言是程序员。

初学者必须 C 语言开始吗？

建议从 C 语言开始，然后学习数据结构、算法、内存、线程、进程、通信、操作系统等基本的概念，它们是学习编程的基础，不管是应用层开发还是底层开发，这些知识都是必须的。

如果你非要跳过 C 语言，从其他语言开始，比如 Java、Python、JavaScript、C#、Golang 等，也不是不可以；但是，在学习的过程中你会有一种雾里看花、空中楼阁的感觉，很多东西只会用，却理解不了，深入不了，原因就是没有计算机基础，没学会走就想跑了，这个时候，还得老老实实回来学习 C 语言。

如何学习底层知识

关于数据结构、算法、内存、线程、进程、通信、操作系统等这些基本的知识，重要的是理解概念，知道计算机是怎么回事，千万不要深入细节，把自己绕进去，耽误一两年的功夫，要尽早跳出来去做应用开发，找到兴趣点，获得成就感。

这个时候，C 语言主要的作用是让你入门，了解编程语言的基本语法，强化编程思维，学习计算机底层知识，为以后的职业生涯打下坚实的基础，而不是用它来做实际开发。

在实际开发中，遇到问题，或者哪里理解不透了，可以再来回顾这些底层知识，这个时候就可以深入细节了。因为有了实际开发经验，再学习底层知识就知道哪里是重点了，不会像无头的苍蝇一样乱飞，什么都学。

【拓展】C 语言为什么没有应用层开发的库

C 语言是一门“古老”的语言了，它只支持面向过程编程，不支持面向对象编程和泛型编程，在中大型的应用层项目开发中，C 语言已经显得捉襟见肘了，C++、Java、Python、C#、JavaScript 等其他编程语言能够更好地胜任，为 C 语言开发应用层的库简直是费力不讨好，所以几乎没人这么做。

GTK 算是一个应用层的库，但是它也比较老了，新版的 GTK+ 已经支持 C++ 了，不再仅仅支持 C 语言了。

我们先不管面向过程、面向对象、泛型这些晦涩的编程概念，简单地理解就是，C 语言支持的特性少，用起来费劲，开发效率低，而 C++、Java、Python、C#、JavaScript 等支持的特性多，用起来方便，开发效率高。

C 语言的优势是运行效率极高，这正是底层开发所看重的。底层开发有时候就是一个模块，或者是一个服务，规模不算大，但是对效率有严格的要求，此时用 C 语言就非常合适，所以针对底层开发的 C 语言库较多，因为它们有非常大的实用价值。

7、程序运行在内存中，而不是硬盘中

如果你的电脑上安装了 QQ，你希望和好友聊天，会双击 QQ 图标，打开 QQ 软件，输入账号和密码，然后登录就可以了。

那么，QQ 是怎么运行起来的呢？

首先，有一点你要明确，你安装的 QQ 软件是保存在硬盘中的。

现在的硬盘有两种形式，一种是固态硬盘（SSD），一种是机械硬盘（HDD）。

双击 QQ 图标，操作系统就会知道你要运行这个软件，它会在硬盘中找到你安装的 QQ 软件，将数据（安装的软件本质上就是很多数据的集合）复制到内存。对！就是复制到内存！QQ 不是在硬盘中运行的，而是在内存中运行的。

所谓内存，就是我们常说的内存条，就是下图这个玩意，相信你肯定见过。

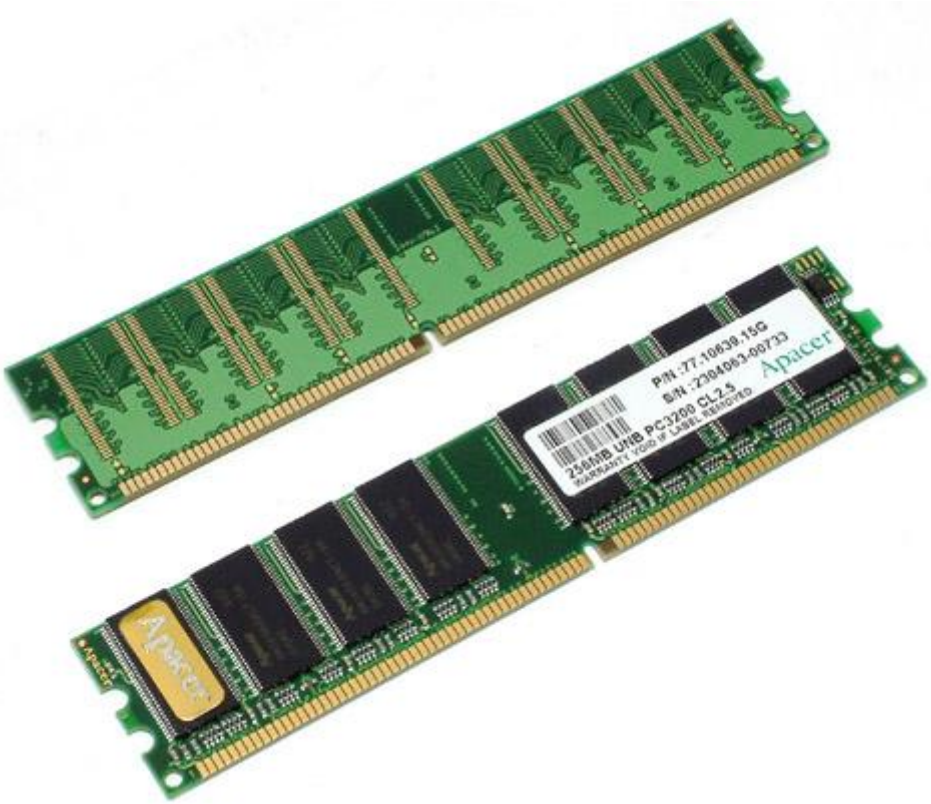


图 1：内存条示意图

程序之所以要在内存中运行，因为内存的读写速度比硬盘快很多（通常超过一个数量级）。

对于读写速度，内存 > 固态硬盘 > 机械硬盘。机械硬盘是靠电机带动盘片转动来读写数据的，而内存条通过电路来读写数据，电机的转速肯定没有电的传输速度（几乎是光速）快。虽然固态硬盘也是通过电路来读写数据，但是因为与内存的控制方式不一样，速度也不及内存。

所以，不管是运行 QQ 还是编辑 Word 文档，都是先将硬盘上的数据复制到内存，才能让 CPU 来处理，这个过程就叫作载入内存（Load into Memory）。完成这个过程需要一个特殊的程序（软件），这个程序就叫做加载器（Loader）。

CPU 直接与内存打交道，它会读取内存中的数据进行处理，并将结果保存到内存。如果需要保存到硬盘，才会将内存中的数据复制到硬盘。

例如，打开 Word 文档，输入一些文字，虽然我们看到的不一样了，但是硬盘中的文档没有改变，新增的文字暂时保存到了内存，Ctrl+S 才会保存到硬盘。因为内存断电后会丢失数据，所以如果你编辑完 Word 文档忘记保存

就关机了，那么你将永远无法找回这些内容。

> 较新版本的 Word 为了避免内容丢失的问题，采用了一种「自动保存」机制，所以忘记 Ctrl+S 一般也不会有太大关系。

虚拟内存

如果我们运行的程序较多，占用的空间就会超过内存（内存条）容量。例如计算机的内存容量为 2G，却运行着 10 个程序，这 10 个程序共占用 3G 的空间，也就意味着需要从硬盘复制 3G 的数据到内存，这显然是不可能的。

操作系统（Operating System，简称 OS）为我们解决了这个问题：当程序运行需要的空间大于内存容量时，会将内存中暂时不用的数据再写回硬盘；需要这些数据时再从硬盘中读取，并将另外一部分不用的数据写入硬盘。这样，硬盘中就会有一部分空间用来存放内存中暂时不用的数据。这一部分空间就叫做虚拟内存（Virtual Memory）。

3G - 2G = 1G，上面的情况需要在硬盘上分配 1G 的虚拟内存。

硬盘的读写速度比内存慢很多，反复交换数据会消耗很多时间，所以如果你的内存太小，会严重影响计算机的运行速度，甚至会出现”卡死“现象，即使 CPU 强劲，也不会有大的改观。如果经济条件允许，建议将内存升级为 8G，这样在 Windows 7/8/10/11 下运行软件就会比较流畅了。

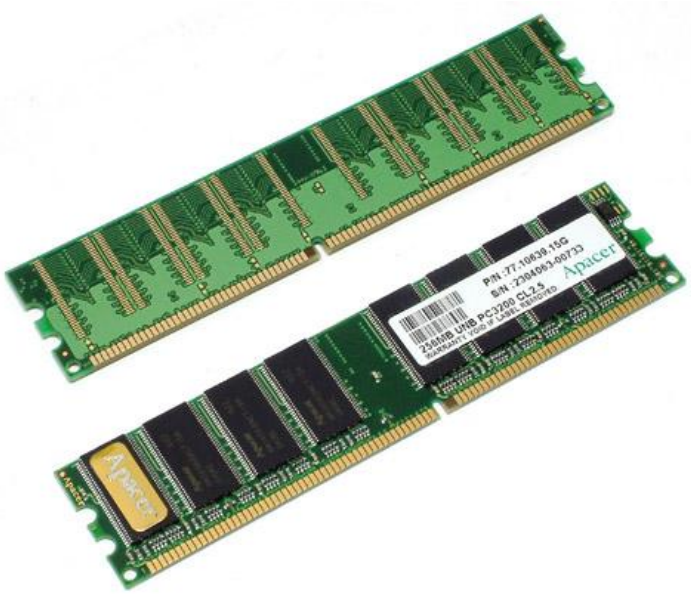
总结：CPU 直接从内存中读取数据，处理完成后将结果再写入内存。

图 2：CPU、内存、硬盘和主板的关系

8、学习 C 语言，一定要学习内存

程序是在内存中运行的，一名合格的程序员必须了解内存，学习 C 语言是了解内存布局的最简单、最直接、最有效的途径，C 语言简直是为内存而生的，它比任何一门编程语言都贴近内存。

所谓内存，就是我们常说的内存条，就是下图这个玩意，上节已经提到过了。



图：内存条

所有的程序都在拼尽全力节省内存，都在不遗余力提高内存使用效率，计算机的整个发展过程都在围绕内存打转，不断地优化内存布局，以保证可以同时运行多个程序。

不了解内存，就学不会进程和线程，就没有资格玩中大型项目，没有资格开发底层组件，没有资格架构一个系统，命中注定你就是一个菜鸟，成不了什么气候。

以上这点我有深刻的体会！

工作期间我曾专注于网站开发，虽然能够设计出界面漂亮、体验良好的网页，但是对内存泄漏、多线程、共享内存等底层概念一窍不通，感觉和周围同事的差距很大，这让我非常郁闷，不知道如何突破。我曾多次尝试学习内存和线程，也找了很多资料，但是无论如何都啃不懂，到头来还是一头雾水。

离职后我全职运营 C 语言中文网，于是决定再次系统、深入、全面地学习 C 语言，并结合 C 语言去了解一些内存知识，这个时候我才发现，原来 C 语言就是为内存而生的，C 语言的设计和内存的布局是严密贴合的，我因为学习 C 语言而吃透了内存，了解了计算机内存是如何分布和组织的。

> C 语言无时无刻不在谈内存，内存简直就是如影随形，你不得不去研究它。

至关重要的一点是，我能够把内存和具体的编程知识以及程序的运行过程结合起来，真正做到了学以致用，让概念落地，而不是空谈，这才是最难得的。

另外一个惊喜是，攻克内存后我竟然也能够理解进程和线程了，原来进程和线程也是围绕内存打转的，从一定程度上讲，它们的存在也是为了更加高效地利用内存。

从 C 语言到内存，从内存到进程和线程，环环相扣：不学 C 语言就吃不透内存，不学内存就吃不透进程和线程。

我感觉自己瞬间升华了，达到了一个新的高度，之前的很多谜团都解开了，和大神交流也没有障碍了。

「内存 + 进程 + 线程」这几个最基本的计算机概念是菜鸟和大神的分水岭，也只有学习 C 语言才能透彻地理解它们。Java、C#、PHP、Python、JavaScript 程序员工作几年后会遇到瓶颈，有很多人会回来学习 C 语言，重拾底层概念，让自己再次突破。

9、主流 C 语言编译器汇总（包含所有平台）

C 语言的历史比较久，而且早期没有规范，整个计算机产业也都处于拓荒的年代，所以就涌现了很多款 C 语言编译器，它们各有特点，适用于不同的平台，本文就来给大家科普一下。

我们分两部分介绍 C 语言的编译器，分别是桌面操作系统和嵌入式操作系统。

桌面操作系统

对于当前主流桌面操作系统而言，可使用 Visual C++、GCC 以及 LLVM Clang 这三大编译器。

Visual C++（简称 MSVC）是由微软开发的，只能用于 Windows 操作系统；GCC 和 LLVM Clang 除了可用于 Windows 操作系统之外，主要用于 Unix/Linux 操作系统。

像现在很多版本的 Linux 都默认使用 GCC 作为 C 语言编译器，而像 FreeBSD、macOS 等系统默认使用 LLVM Clang 编译器。由于当前 LLVM 项目主要在 Apple 的主推下发展的，所以在 macOS 中，Clang 编译器又被称为 Apple LLVM 编译器。

MSVC 编译器主要用于 Windows 操作系统平台下的应用程序开发，它不开源。用户可以使用 Visual Studio Community 版本来免费使用它，但是如果要把通过 Visual Studio Community 工具生成出来的应用进行商用，那么就得好好阅读一下微软的许可证和说明书了。

而使用 GCC 与 Clang 编译器构建出来的应用一般没有任何限制，程序员可以将应用程序随意发布和进行商用。

低版本的 MSVC 编译器对 C99 标准的支持十分有限，直到发布 Visual Studio Community 2019，也才对 C11 和 C17 标准做了部分支持。 所幸的是，Visual Studio Community 2017 加入了对 Clang 编译器的支持，官方称之为——Clang with Microsoft CodeGen，当前版本基于的是 Clang 3.8。

C 语言从诞生到现在，发布了很多重要的版本（称为 C 语言标准），比如 C89/ANSI C、C99、C11、C17 等。

也就是说，应用于 Visual Studio 集成开发环境中的 Clang 编译器前端可支持 Clang 编译器的所有语法特性，而后端生成的代码则与 MSVC 效果一样，包括像 long 整数类型在 64 位编译模式下长度仍然为 4 个字节，所以各位使用的时候也需要注意。

为了方便描述，本教程后面涉及 Visual Studio 集成开发环境下的 Clang 编译器简称为 VS-Clang 编译器。

嵌入式系统

而在嵌入式系统方面，可用的 C 语言编译器就非常丰富了，比如：

- 用于 Keil 公司 51 系列单片机的 Keil C51 编译器；
- 当前大红大紫的 Arduino 板搭载的开发套件，可用针对 AVR 微控制器的 AVR GCC 编译器；
- ARM 自己出的 ADS（ARM Development Suite）、RVDS（RealView Development Suite）和当前最新的 DS-5 Studio；
- DSP 设计商 TI（Texas Instruments）的 CCS（Code Composer Studio）；
- DSP 设计商 ADI（Analog Devices，Inc.）的 Visual DSP++ 编译器，等等。

通常，用于嵌入式系统开发的编译工具链都没有免费版本，而且一般需要通过国内代理进行购买。所以，这对于个

人开发者或者嵌入式系统爱好者而言是一道不低的门槛。

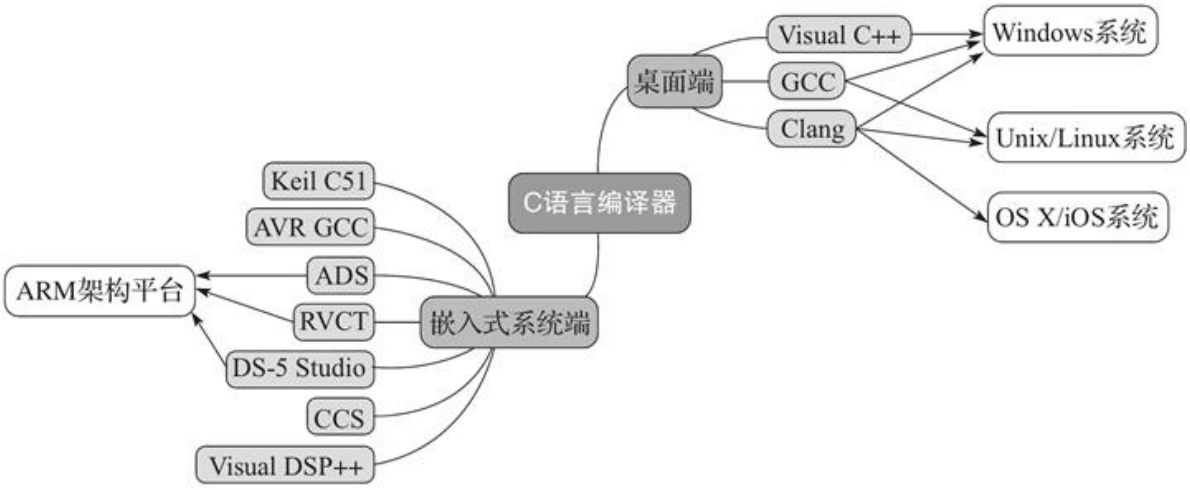
不过 Arduino 的开发套件是可免费下载使用的，并且用它做开发板连接调试也十分简单。Arduino 所采用的 C 编译器是基于 GCC 的。

还有像树莓派（Raspberry Pi）这种迷你电脑可以直接使用 GCC 和 Clang 编译器。此外，还有像 nVidia 公司推出的 Jetson TK 系列开发板也可直接使用 GCC 和 Clang 编译器。树莓派与 Jetson TK 都默认安装了 Linux 操作系统。

在嵌入式领域，一般比较低端的单片机，比如 8 位的 MCU 所对应的 C 编译器可能只支持 C90 标准，有些甚至连 C90 标准的很多特性都不支持。因为它们一方面内存小，ROM 的容量也小；另一方面，本身处理器机能就十分有限，有些甚至无法支持函数指针，因为处理器本身不包含通过寄存器做间接过程调用的指令。

而像 32 位处理器或 DSP，一般都至少能支持 C99 标准，它们本身的性能也十分强大。而像 ARM 出的 RVDS 编译器甚至可用 GNU 语法扩展。

下图展示了上述 C 语言编译器的分类。



10、主流 C 语言开发环境汇总（15 款 IDE 推荐）

实际开发中，除了编译器是必须的工具，我们往往还需要很多其他辅助软件，例如：

- 编辑器：用来编写代码，并且给代码着色，以方便阅读；
- 代码提示器：输入部分代码，即可提示全部代码，加速代码的编写过程；
- 调试器：观察程序的每一个运行步骤，发现程序的逻辑错误；
- 项目管理工具：对程序涉及到的所有资源进行管理，包括源文件、图片、视频、第三方库等；
- 漂亮的界面：各种按钮、面板、菜单、窗口等控件整齐排布，操作更方便。

这些工具通常被打包在一起，统一发布和安装，例如 Visual Studio、Dev C++、Xcode、CLion、Code::Blocks 等，它们统称为集成开发环境（IDE，Integrated Development Environment）。

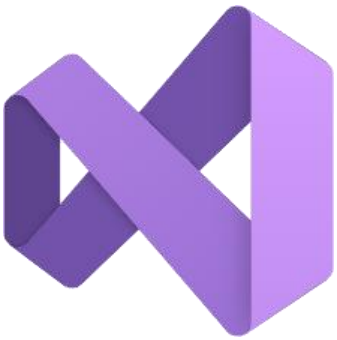
集成开发环境就是一系列开发工具的组合套装。这就好比台式机，一个台式机的核心部件是主机，有了主机就能独立工作了，但是我们在购买台式机时，往往还要附带上显示器、键盘、鼠标、U 盘、摄像头等外围设备，因为只有主机太不方便了，必须有外设才能玩的爽。

集成开发环境也是这个道理，只有编译器不方便，所以还要增加其他的辅助工具。在实际开发中，我一般也是使用集成开发环境，而不是单独地使用编译器。

> 有时候为了称呼方便，或者初学者没有严格区分概念，也会将 C 语言集成开发环境称作“C 语言编译器”或者“C 语言编程软件”。这里大家不要认为是一种错误，就把它当做“乡间俗语”吧。

这里我给大家汇总了 15 款 C 语言 IDE，它们有的比较主流，有的比较小众，所以最后我还给出了使用建议。

1. Visual Studio



Visual Studio，简称 VS，最新版是 VS2022，默认使用 Visual C++ 编译器（微软开发的编译器，所以简称 MSVC）。

为了适应最新的 Windows 操作系统，微软每隔一段时间（一般是一两年）就会对 VS 进行升级。VS 的不同版本以发布年份命名，例如 VS2017 是微软于 2017 年发布的，VS2019 是微软于 2019 年发布的。

不过 VS 有点庞大，安装包有 2~3G，下载不方便，而且会安装很多暂时用不到的工具，安装时间可能长达 30 分钟。

适用平台：Windows

费用：有免费版和收费版，初学者使用免费版足以。

推荐指数：★★★★★

2. CLion

由捷克 JetBrains 公司开发的一款 C/C++ IDE，同时支持 GCC、Clang、MSVC 三种编译器，在编程过程中可以随意切换。

JetBrains 出品，必属精品，这是一款非常性感的编译器。

适用平台：跨平台，同时支持 Windows、Mac OS 和 Linux。

费用：收费，没有免费版本。

推荐指数：★★★★★

3. Dev C++

一款 Windows 平台下的轻量级 C/C++ IDE，免费开源，适合初学者，默认使用 MinGW/GCC 编译器（GCC 编译器的 Windows 移植版）。

优点是体积小（只有 100MB 左右）、安装卸载方便、学习成本低，缺点是调试功能弱。如果你讨厌 VS 的复杂性，那么可以使用 Dev C++。

NOI、NOIP 等比赛的指定工具。

适用平台：Windows

费用：免费

推荐指数：★★★★☆

4. Netbeans

甲骨文开发的一款跨平台的 IDE，支持 C/C++、Java、PHP 等多种语言，一般在 Linux 下才考虑使用。

适用平台：跨平台，支持 Solaris、Windows、Linux 和 macOS。

费用：开源免费

推荐指数：★★★☆☆

5. Eclipse CDT

Eclipse 是一款著名的、开源的、跨平台的 IDE，Eclipse CDT 是一个基于 Eclipse 主平台的项目，它提供了一个完整功能的 C/C++ IDE，一般在 Linux 平台下才考虑使用。

适用平台：跨平台，支持 Windows、Linux 和 macOS。

费用：开源免费

推荐指数：★★★☆☆

6. CodeLite

一款为 C/C++、JavaScript（Node.js）和 PHP 编程专门设计打造的自由而开源的、跨平台的 IDE。

适用平台：跨平台，支持 Windows、Linux 和 macOS。

费用：开源免费

推荐指数：★★★☆☆

7. Code::Blocks

一款免费开源的 C/C++ IDE，支持 GCC、MSVC 等编译器，优点是跨平台，体积小，安装和卸载方便。不过 Code::Blocks 的界面要比 Dev C++ 复杂一些，不如 Dev C++ 来得清爽。

适用平台：跨平台，同时支持 Windows、Mac OS 和 Linux。

费用：免费

推荐指数：★★★☆☆

8. C-Free

国产的轻量级的 C/C++ IDE，最新版本是 C-Free 5.0，整个软件才 14M，非常轻巧，安装简单，适合初学者玩玩。

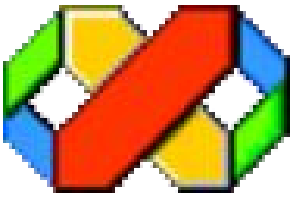
官方基本停止开发了，在 XP、Win7 下能够运行，但是在 Win8、Win10、Win11 下可能会存在兼容性问题。

适用平台：Windows

费用：收费（不贵）

推荐指数：★☆☆☆☆

9. Visual C++ 6.0



简称 VC6.0 或者 VC++6.0，是微软 1998 年推出的 C/C++ IDE，现在已经被 Visual Studio 替代了。

老古董了，要不是落后的学校课程强制使用，千万不要触碰了。

适用平台：Windows

费用：收费（但是网上下载的都是破解的了）

推荐指数：☆☆☆☆☆

10. Turbo C

美国 Borland 公司开发的一款应用于 DOS 平台上的 IDE，只能使用键盘来操作，不能使用鼠标，所以非常不方便。

没事别碰了，没有意义。

适用平台：Windows/DOS

费用：免费

推荐指数：★☆☆☆☆

11. GCC

Linux 下使用最广泛的 C/C++ 编译器，不管是开发人员还是初学者，一般都将 GCC 作为 Linux 下首选的编译工具。

仅仅是一款命令行编译器，没有界面，实际开发中往往需要和 GDB、Make 等工具搭配使用，或者配置 VS Code、Sublime Text、Gedit 等编辑器。

如果你希望使用 IDE，那么可以选择 CLion、Netbeans、Eclipse CDT、CodeLite、Code::Blocks 等。

适用平台：Linux

费用：免费开源

推荐指数：★★★★★

12. Xcode

macOS 平台下的一款 IDE，由 Apple 官方开发，默认使用 LLVM/Clang。

适用平台：macOS

费用：使用免费，但是发布应用收费

推荐指数：★★★★★

13. Sublime Text

Sublime Text 是一个非常流行的、跨平台的文本编辑器，界面简介，插件众多，配置好编译器（一般是 GCC）就能编译代码。

适用平台：跨平台，同时支持 Windows、Mac OS 和 Linux。

费用：开源免费

推荐指数：★★★★☆

14. Visual Studio Code

简称 VS Code，由微软开发，当前热门的跨平台的文本编辑器，插件众多，配置好编译器（一般是 GCC）就能编译代码。

适用平台：跨平台，同时支持 Windows、Mac OS 和 Linux。

费用：开源免费

推荐指数：★★★★☆

15. KDevelop

KDevelop 是一款跨平台的开源 IDE，它基于 KDevPlatform、KDE 和 Qt 库。

适用平台：跨平台，同时支持 Windows、Mac OS、Linux、Solaris 和 FreeBSD 等。

费用：开源免费

推荐指数：★★☆☆☆

我的建议

如果不差钱，可以考虑 CLion，好用，跨平台，无需适应不同的 IDE。

如果没有特殊需求，我的建议如下：

- Windows 下推荐使用 VS、CLion、Dev C++；
- Linux 下推荐使用 GCC、CLion、Netbeans、Eclipse CDT；
- macOS 下推荐使用 Xcode、CLion；
- 如果愿意折腾，可以使用 VS Code、Sublime Text 编辑器来配置开发环境，但是不建议初学者尝试。

下载、安装和使用各种开发环境，请猛击《C 语言编译器（C 语言编程软件）完全攻略》一章。