

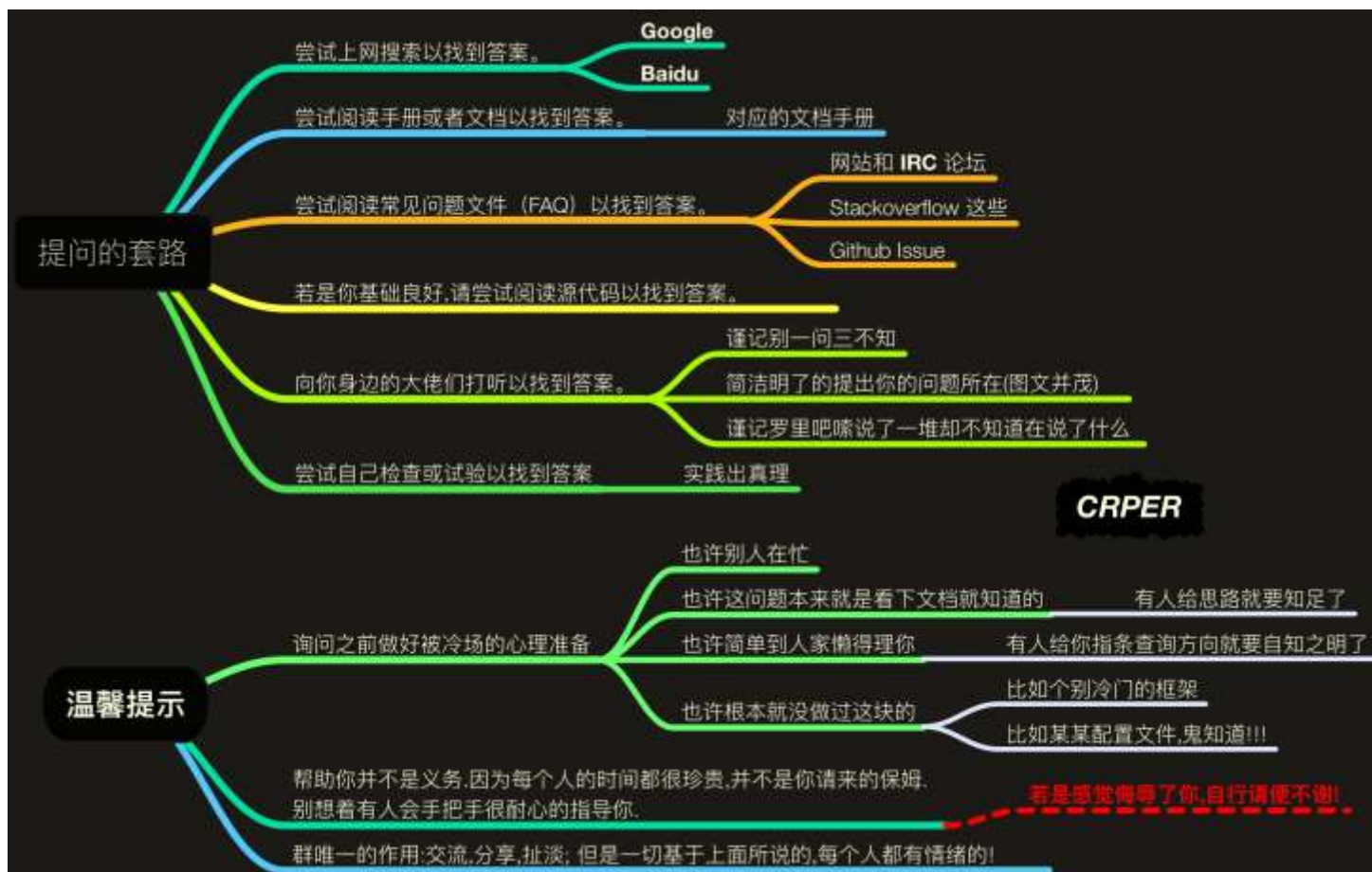
## 如何科学地提问

# 事先声明



# 如何科学地提问

你真的准备好了吗？



# 提问也有教程？

- [Stop-Ask-Questions-The-Stupid-Ways]\_Stop-Ask-Questions-The-Stupid-Ways/README.md at master · tangx/Stop-Ask-Questions-The-Stupid-Ways (github.com)
- [How To Ask Questions The Smart Way]\_How-To-Ask-Questions-The-Smart-Way/README-zh\_CN.md at main · ryanhanwu/How-To-Ask-Questions-The-Smart-Way (github.com)
- [可供参考的提问模板] <https://ysyx.oscc.cc/docs/2205/misc/ask.html>

# Agenda

- 如何科学地提问
- **为什么学习C语言**
- 怎么学习C语言
- 用什么写C语言
- 数制与码制
- C语言基础语法
- 命令行工具
- VsCode和vim

# C语言介绍

- “C 不是一种“非常高级”的语言，也不是一种“体量大”的语言，并且不专门针对任何特定的应用领域。但它没有限制且具有通用性，这使得它比所谓更强大的语言更方便、更有效地完成许多任务。”——K&R
- 实现了第一个非汇编语言编写的操作系统
- 为什么要学习C?
  - 我们可以编写程序来利用计算机体系结构的底层功能
    - 内存管理!
  - 然而，这也意味着在C语言中事情更容易出错

# C的优点

NekoBytes-TheMissing 2024

- 贴近硬件
- 功能强大
- 语法简介
- 学习轻松
- 应用广泛
- 关注底层
- 以及...

• 极致的速度与节能！



Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

NekoBytes




# Linux系统内核

- bpf\_local\_storage.c
- bpf\_lru\_list.c
- bpf\_lru\_list.h
- bpf\_lsm.c
- bpf\_struct\_ops.c
- bpf\_struct\_ops\_types.h
- bpf\_task\_storage.c
- btf.c
- cgroup.c
- core.c
- cpumap.c
- devmap.c
- disasm.c
- disasm.h
- dispatcher.c
- hashtab.c
- helpers.c
- inode.c
- local\_storage.c

## The Linux Kernel Archives

[About](#)[Contact us](#)[FAQ](#)[Releases](#)[Signatures](#)[Site news](#)



### Languages

C 98.4%

Assembly 0.9%

Shell 0.3%

Makefile 0.2%

Python 0.1%

Perl 0.1%

	Protocol	Location
	HTTP	<a href="https://www.kernel.org/pub/">https://www.kernel.org/pub/</a>
	GIT	<a href="https://git.kernel.org/">https://git.kernel.org/</a>
	RSYNC	<a href="rsync://rsync.kernel.org/pub/">rsync://rsync.kernel.org/pub/</a>

La 5.14

mainline:	5.15-rc4	2021-10-03	<a href="#">[tarball]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>		
stable:	5.14.10	2021-10-07	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
stable:	5.13.19 [EOL]	2021-09-18	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
longterm:	5.10.71	2021-10-06	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
longterm:	5.4.151	2021-10-06	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
longterm:	4.19.209	2021-10-06	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
longterm:	4.14.249	2021-10-06	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
longterm:	4.9.285	2021-10-06	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
longterm:	4.4.287	2021-10-07	<a href="#">[tarball]</a>	<a href="#">[pgp]</a>	<a href="#">[patch]</a>	<a href="#">[inc. patch]</a>	<a href="#">[view diff]</a>	<a href="#">[browse]</a>	<a href="#">[changelog]</a>
linux-next:	next-20211007	2021-10-07						<a href="#">[browse]</a>	



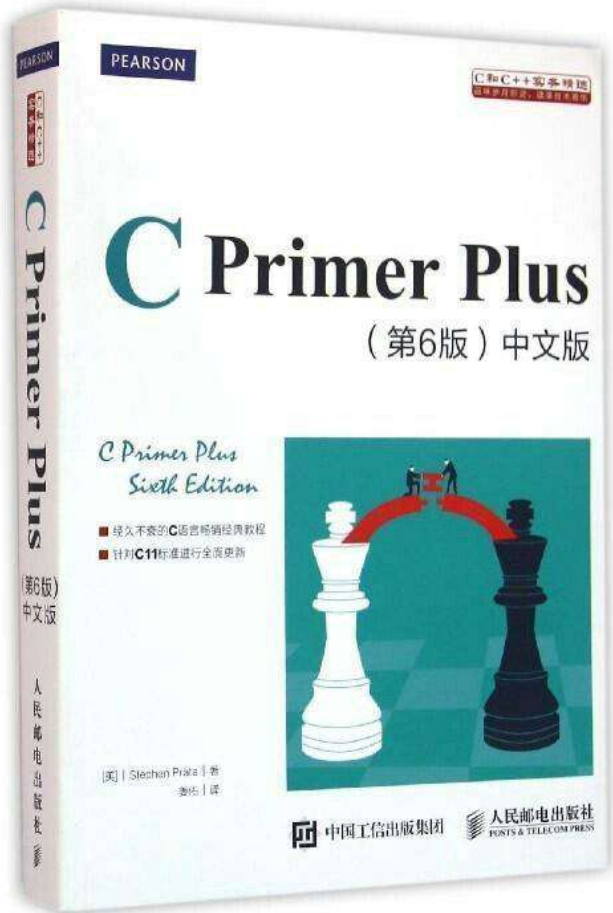
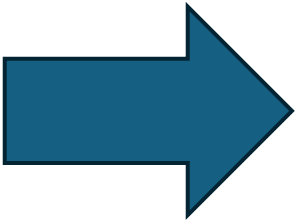
# Agenda

- 如何科学地提问
- 为什么学习C语言
- **怎么学习C语言**
- 用什么写C语言
- 数制与码制
- C语言基础语法
- 命令行工具
- VsCode和vim

# 学习C语言看什么书？



浅薄

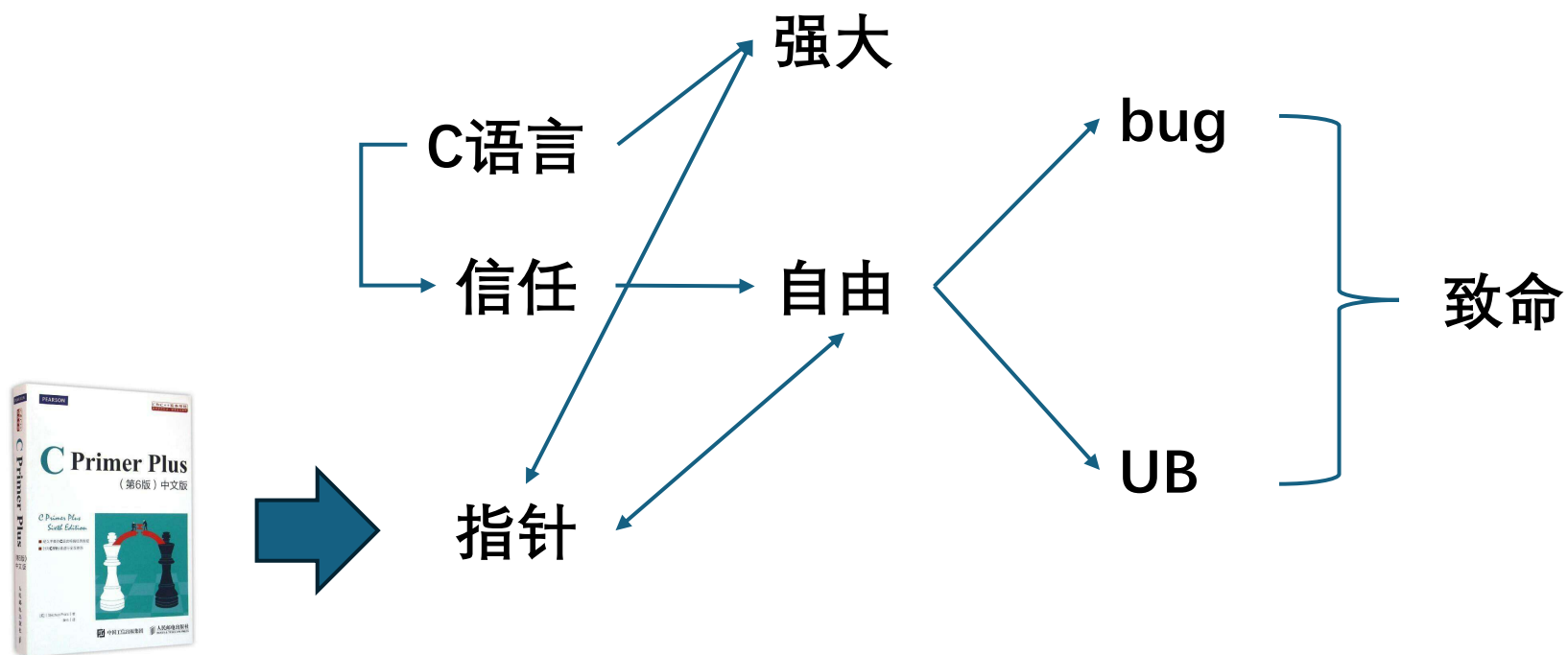


# C语言的推荐资源

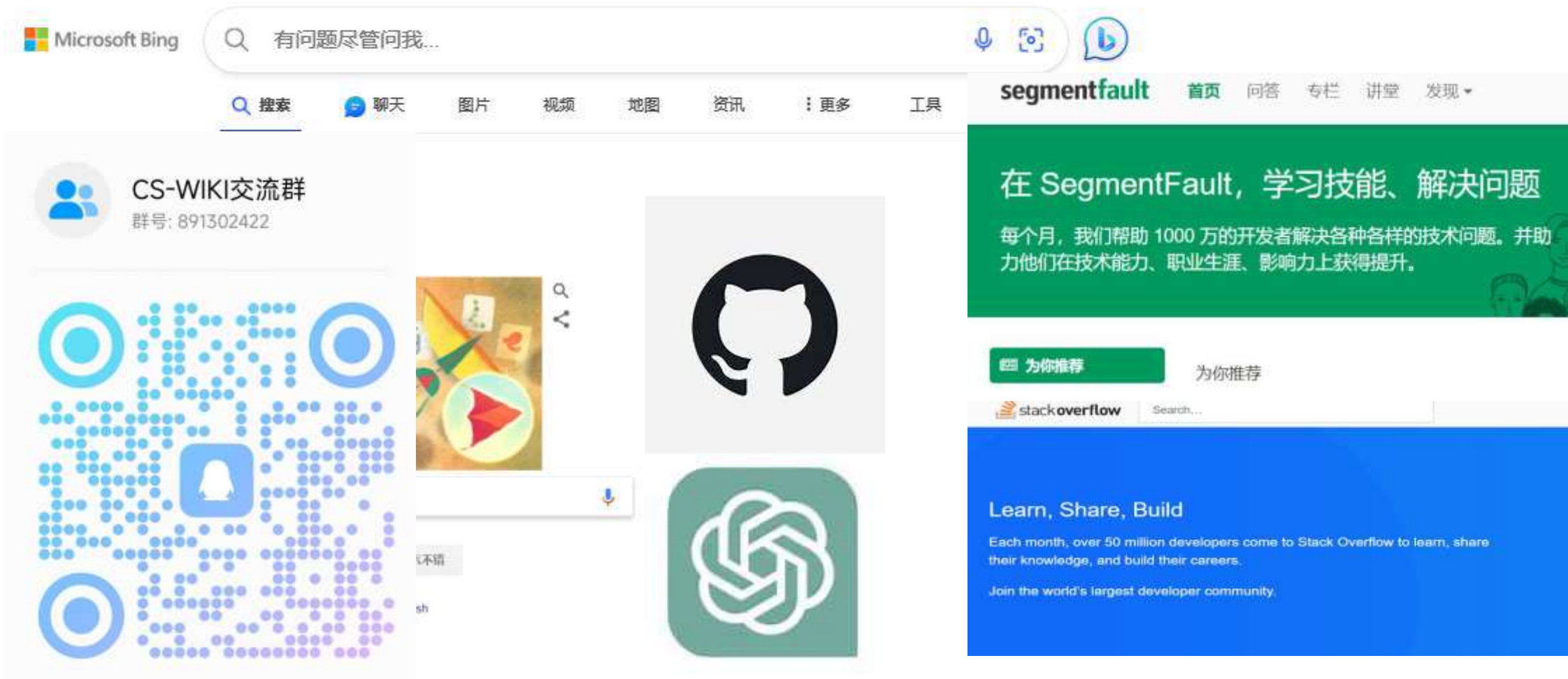
- [Introductory C Programming 专项课程]  
<https://www.coursera.org/specializations/c-programming?>
- [翁恺C语言基础入门]@bilibili:  
<https://www.bilibili.com/video/BV1dr4y1n7vA> 一节课入门C语言
- [C参考手册] <https://zh.cppreference.com/w/c>



# 快乐C之旅



# 遇到问题怎么解决？



# Agenda

- 如何科学地提问
- 为什么学习C语言
- 怎么学习C语言
- **用什么写C语言**
- 数制与码制
- C语言基础语法
- 命令行工具
- VsCode和vim



# IDE与编辑器 (1/2)

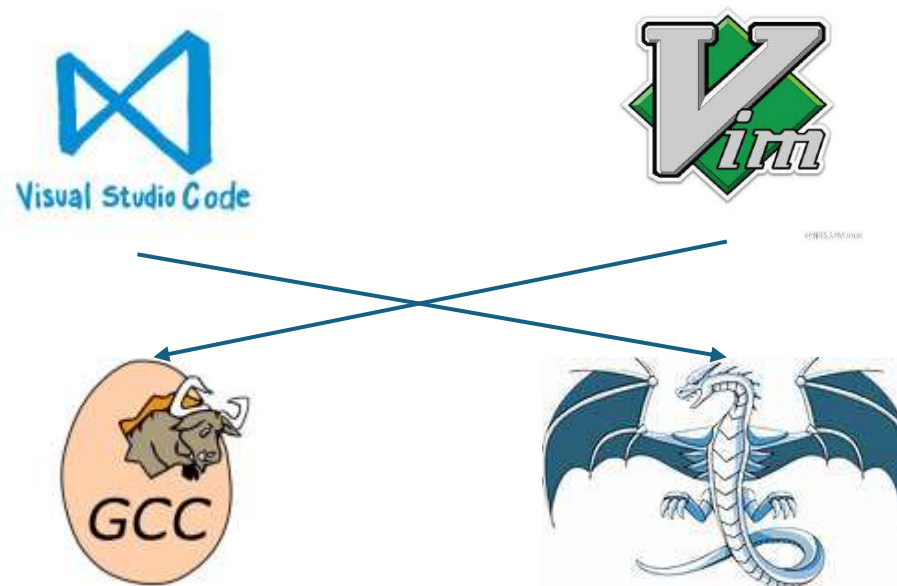
- IDE（集成开发环境）和编辑器是软件开发中常用的工具。它们都用于编写、编辑和管理代码，但在功能和用途上有一些区别。
- IDE是一种集成了多个工具和功能的软件，旨在提供全面的开发环境。它通常包括代码编辑器、调试器、编译器、构建工具、版本控制系统等。IDE提供了一个统一的界面，使开发人员可以在一个应用程序中完成多个开发任务。
  - Visual Studio
  - Clion
- 编辑器则更加轻量级，专注于提供代码编辑功能。它通常具有语法高亮、自动完成、代码折叠等基本功能，但不包含其他开发工具。编辑器通常更加灵活和可定制，适合开发人员根据自己的需求选择插件和扩展。
  - Visual Studio Code

# IDE与编辑器 (2/2)

## IDE



## 编辑器



# Agenda

- 如何科学地提问
- 为什么学习C语言
- 怎么学习C语言
- 用什么写C语言
- **数制与码制**
- C语言基础语法
- 命令行工具
- VsCode和vim

# 二进制

- 使用0和1字符串表示数字的方法
- 为什么计算机使用二进制？
  - 计算机的基本构建模块是只能表示两个值的晶体管。我们决定将这两个值标记为 0 和 1
- 通过在字符串前面添加0b或添加下标2来表示

# 术语

- 比特 (Bit)
  - 1位二进制数
- 半字节 (Nibble)
  - 4 Bits
- 字节 (Byte)
  - 8 Bits
- 基数
  - 一个进制系统中所使用的数字的个数

- 最高有效位 (MSB)
  - 最高位置的位

1	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---

- 最低有效位 (LSB)
  - 最低位置的位

1	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---

# 十进制表示法

The diagram illustrates the expansion of the decimal number 5072 into its place value components. At the top, the number 5072 is shown with a subscript 10. Four arrows point from each digit to its corresponding place value expression below: 5 to  $(5 \times 10^3)$ , 0 to  $(0 \times 10^2)$ , 7 to  $(7 \times 10^1)$ , and 2 to  $(2 \times 10^0)$ . These expressions are then summed:  $(5 \times 10^3) + (0 \times 10^2) + (7 \times 10^1) + (2 \times 10^0)$ . Below this, the expanded form is shown as 5000 + 0 + 70 + 2. Finally, the original number 5072 with a subscript 10 is shown at the bottom, indicating the sum of these components equals the original number.

$$5072_{10}$$
$$\swarrow \quad \searrow \quad \swarrow \quad \searrow$$
$$(5 \times 10^3) + (0 \times 10^2) + (7 \times 10^1) + (2 \times 10^0)$$
$$5000 + 0 + 70 + 2$$
$$5072_{10}$$



# 二进制表示法

The diagram illustrates the conversion of the binary number  $1011_2$  to its decimal value. It shows the expansion of the binary number into its constituent powers of 2, followed by the decimal values of those powers, and finally the sum of those values.

$$\begin{array}{c} 1011_2 \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ 8 \quad + \quad 0 \quad + \quad 2 \quad + \quad 1 \\ 11_{10} \end{array}$$

# 二进制加法

$$\begin{array}{r} 11 \\ + 3 \\ \hline 14 \end{array} \quad \begin{array}{r} 11 \\ 1011 \\ + 0011 \\ \hline 1110 \end{array}$$

# 二进制加法 – 溢出

- 当无法用给定位数表示运算结果时，就会发生溢出

- 当发生溢出时，最终会得到不正确的结果

$$\begin{array}{r} 11 \\ + 5 \\ \hline 16 \end{array} \quad \begin{array}{r} 1111 \\ + 1011 \\ + 0101 \\ \hline 0000 \end{array}$$

# 十六进制

- 更易于人类阅读的二进制表示方法
- 基数：16
- 一位十六进制数字可以表示16个数字
- 一位十六进制数字 = 1个半字节
- 通过前置“0x”或附加下标16来表示

Decimal	Binary(0b)	Hex(0x)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

# 二进制转十六进制

1. 将数字从右到左分成 4 组
2. 预先添加所需的前导零，使最左边的组具有四个二进制数字
3. 将每个组转换为相应的十六进制字符

0b111010  
0011      1010  
3            A  
0x3A

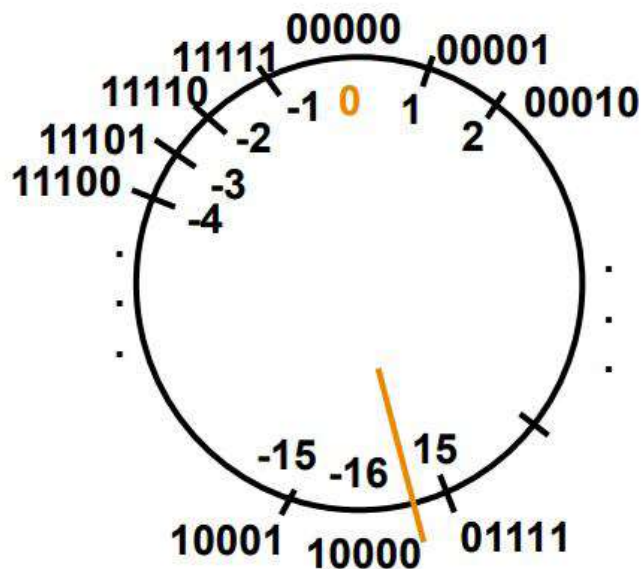
# 二进制如何表示负数？





# 补码

- 如何形成负数?
  - 翻转位并加一
- 最高位 代表值的符号
  - 0 表示其正数
  - 1 表示其负数



- 例 1: 将下列二进制补码转换为十进制: 0b11010
  - 最高位 为1, 因此为负数
  - 翻转位: 0b00101
  - 加一: 0b00110
  - 答案: -6
- 例 4: 将下列二进制补码转换为十进制: 0b01111
  - 最高位 为0, 因此为正数
  - 无需翻转位
  - 答案: 15

# 补码的加法

- 二进制补码算术有效!

$$\begin{array}{r} + \quad -2 \\ + \quad -4 \\ \hline -6 \end{array}$$

$$\begin{array}{r} 11 \\ + 1110 \\ + 1100 \\ \hline 1010 \end{array}$$

0101  
0110

# 补码的计算溢出

$\begin{array}{r} 111 \\ 0111 \\ + 0101 \\ \hline 1100 \end{array}$	Overflow	$\begin{array}{r} 111 \\ 1111 \\ + 0110 \\ \hline 0101 \end{array}$	No Overflow
$\begin{array}{r} 1 \\ 1000 \\ + 1001 \\ \hline 0001 \end{array}$	Overflow	$\begin{array}{r} 1111 \\ 1101 \\ + 1011 \\ \hline 1000 \end{array}$	No Overflow

# 补码的计算溢出

- 溢出：当运算结果无法用给定位数表示时
- 两个正数相加时，结果为负时发生溢出
- 两个负数相加时，结果为正时发生溢出
- 永远不会发生溢出 当两个符号相反的数相加时
- <https://godbolt.org/z/oYG95TPjn>

# 实数呢？

- 定点数表示法
- 浮点数表示法
  - IEEE 754 浮点数标准
  - IEEE 754 模拟器：  
<https://www.h-schmidt.net/FloatConverter/IEEE754.html>
- <https://godbolt.org/z/Yf4Edbjee>

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

31	30	23	22	0
1	1000 0001	111 0000 0000 0000 0000 0000		
s	exponent = 1 + 128 = 129		significand	

$(-1)^s \times (1.\text{significand}) \times 2^{(\text{exponent}-127)}$

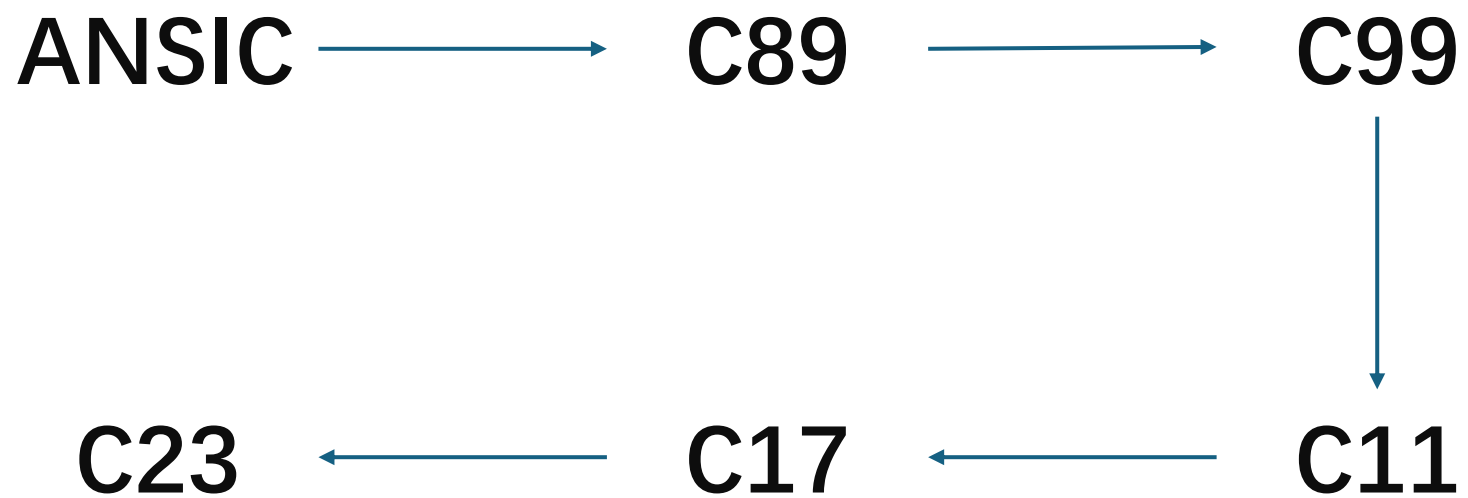
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
s	exponent												fraction																			
1 bit													11 bits											20 bits								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
fraction																																
32 bits																																

# Agenda

- 如何科学的提问
- 为什么学习C语言
- 怎么学习C语言
- 用什么写C语言
- 数制与码制
- **C语言基础语法**
- 命令行工具
- VsCode和vim



# 语法标准



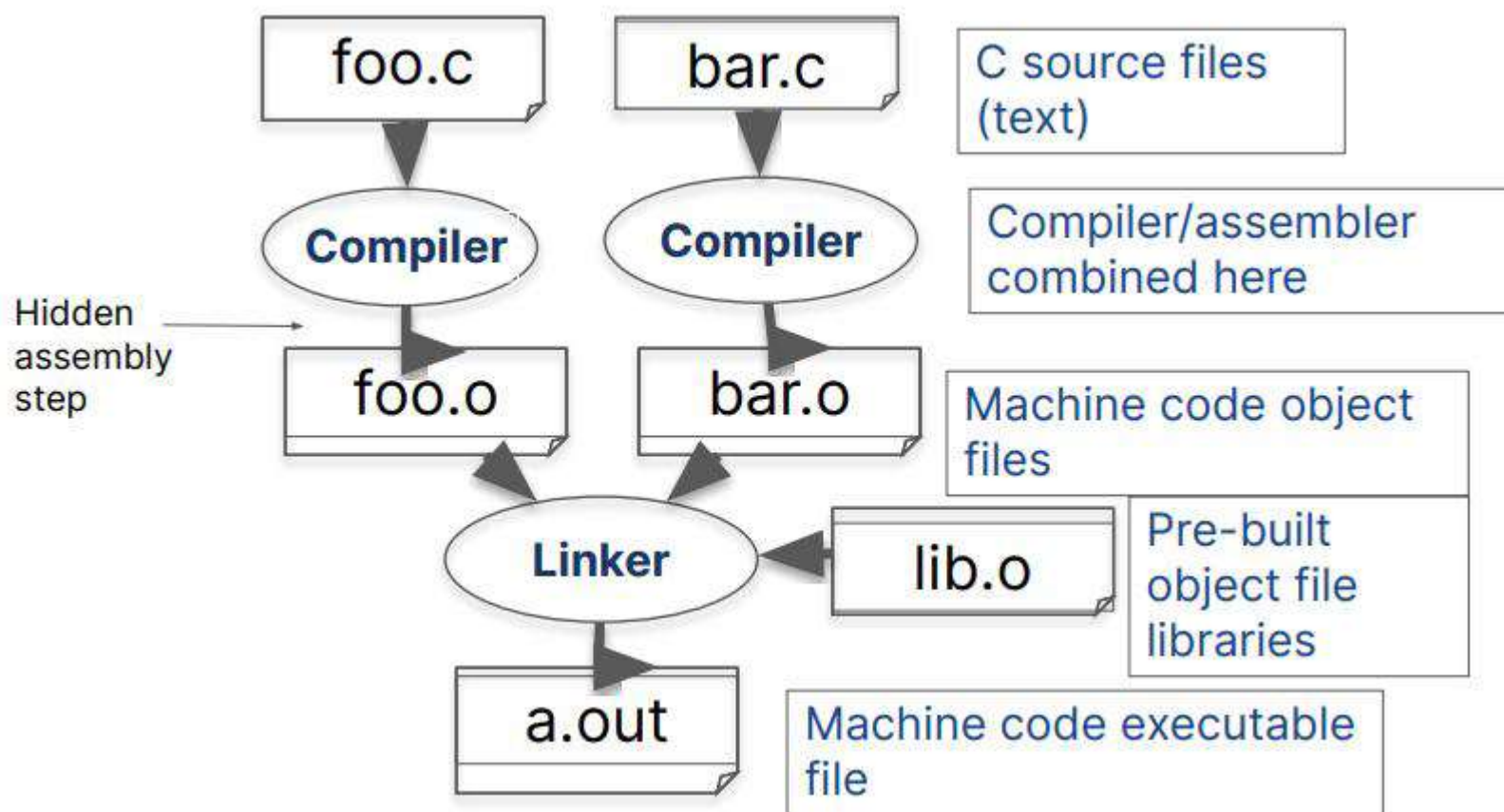
# 编译 vs 解释

- 我们在某种程度上用英语编写代码，但系统实际上只看到0和1。  
我们该如何解决这个问题？
  - 如果有人不懂英语，但懂法语，我们会翻译文字！
  - 系统中的处理类似，但我们翻译成非人类可读的语言
- 翻译以两种方式进行
  - 编译（事前翻译）
  - 解释（在线翻译）
  - 有些语言同时使用这两种方式！

# 编译

- C 编译器将 C 程序直接映射为特定于体系结构的机器代码（1 和 0 的数值串）。
  - Java 转换为独立于体系结构的字节码，然后由即时 (JIT) 编译器进行编译。
  - Python 在运行时而不是编译时转换为 Python 字节码。
    - 运行时编译与 JIT 编译的不同之处在于程序转换为低级汇编语言并最终转换为机器代码的时间。
- 对于 C，处理 .c 文件通常分为 3 个部分
  - .c 文件被编译为 .s 文件 ⇒ 由编译器编译
  - .s 文件被汇编为 .o 文件 ⇒ 由汇编器汇编（此步骤一般是隐藏的，所以大多数时候我们直接将.c文件转换为.o文件）
  - .o文件链接在一起创建可执行文件⇒由链接器链接

# C 编译概述



# 起点：Hello World

- <https://godbolt.org/z/4esj9rvar>
- 程序的组成要件
  - 预编译指令
  - main函数
  - 注释
  - 限定符
  - 花括号、函数体和块
  - 声明
  - 赋值
  - 函数调用
  - return语句——返回值

# C Pre-Processor (CPP) ——C 预处理器

- C 源文件在编译器看到代码之前首先经过预处理器 CPP
- CPP 用单个空格替换注释
- CPP 命令以“#”开头
  - `#include "file.h" /* 将 file.h 插入到文件 */`
  - `#include <stdio.h> /* 在标准位置查找文件，但没有实际区别 */`
  - `#define PI (3.14159) /* 定义常量 */`
  - `#if/#endif /* 有条件地包含文本 */`
- 使用 gcc 的 `-save-temps` 选项查看预处理结果
  - 完整文档位于: [http:// gcc.gnu.org/onlinedocs/cpp/](http://gcc.gnu.org/onlinedocs/cpp/)

# 标准库——官方DLC

- 标准库（Standard Library）是一组在编程语言中提供常用功能和工具的软件库。在C语言中，标准库是由C标准委员会定义的，它包含了一系列的头文件和函数，提供了许多常见的操作和功能，如输入输出、字符串处理、内存管理、数学运算等。
- 官方DLC

此游戏的内容		浏览所有(15)
Fjordur - ARK Expansion Map	免费	
Lost Island - ARK Expansion Map	免费	
ARK: Genesis Season Pass	¥ 58.00	
Vaiguero - ARK Expansion Map	免费	
Ragnarok - ARK Expansion Map	免费	
ARK: Extinction - Expansion Pack	N/A	
ARK: Aberration - Expansion Pack	N/A	
ARK: Scorched Earth - Expansion Pack	N/A	
Primitive+ ARK Total Conversion	免费	
The Center - ARK Expansion Map	免费	
Crystal Isles - ARK Expansion Map	免费	
ARK: Survival Evolved Original Soundtrack	N/A	
ARK: Expansion Packs Original Soundtrack	N/A	
ARK: Genesis Part 1 Original Soundtrack	N/A	
ARK: Genesis Part 2 Original Soundtrack	N/A	
¥ 58.00		将所有 DLC 添加至购物车

## C 标准库头文件

C 标准库的接口由下列头文件的汇集定义。

<assert.h>	条件编译宏，将参数与零比较
<complex.h> (C99)	复数运算
<ctype.h>	用来确定包含于字符数据中的类型的函数
<errno.h>	报告错误条件的宏
<fenv.h> (C99)	浮点环境
<float.h>	浮点类型的极限
<inttypes.h> (C99)	整数类型的格式转换
<iso646.h> (C95)	运算符的替代写法
<limits.h>	整数类型的范围
<locale.h>	本地化工具
<math.h>	常用数学函数
<setjmp.h>	非局部跳转
<signal.h>	信号处理
<stdalign.h> (C11)	alignas 与 alignof 便利宏
<stdarg.h>	可变参数
<stdatomic.h> (C11)	原子操作
<stdbool.h> (C99)	布尔类型的宏
<stddef.h>	常用宏定义
<stdint.h> (C99)	定宽整数类型
<stdio.h>	输入/输出
<stdlib.h>	基础工具：内存管理、程序工具、字符串转换、随机数、算法
<stdnoreturn.h> (C11)	noreturn 便利宏
<string.h>	字符串处理
<tgmath.h> (C99)	泛型数学（包装 math.h 和 complex.h 的宏）
<threads.h> (C11)	线程库
<time.h>	时间/日期工具
<uchar.h> (C11)	UTF-16 和 UTF-32 字符工具
<wchar.h> (C95)	扩展多字节和宽字符工具
<wctype.h> (C95)	用来确定包含于宽字符数据中的类型的函数

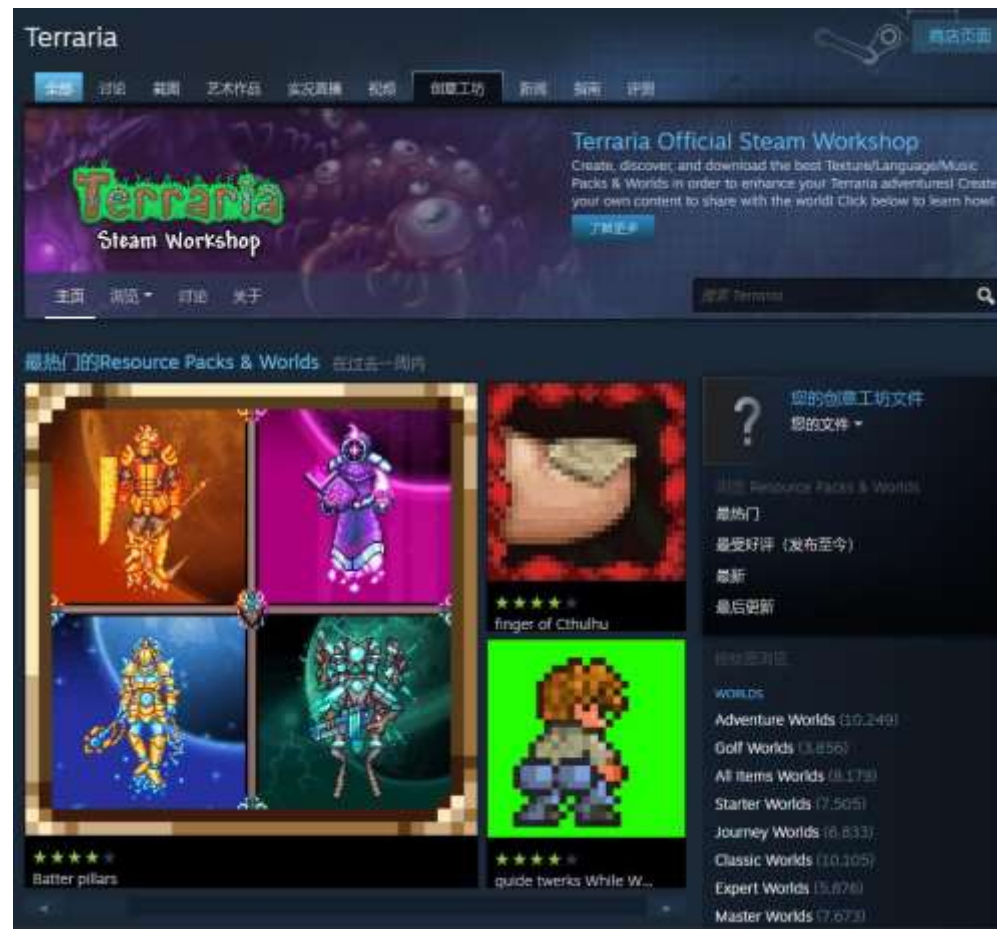


# 第三方库——社区模组

NekoBytes-TheMissing 2024

NekoBytes

- 第三方库（Third-party library）是由独立的开发者或组织创建和维护的软件库，它们不是编程语言的标准库的一部分。第三方库通常提供了额外的功能和工具，以扩展编程语言的能力，使开发者能够更快速、更方便地开发应用程序。
- 社区模组





# 主函数：main

- 简单的main函数形式：
  - `int main(void)`
- 要让 main 函数接受参数，请使用以下语句：
  - `int main (int argc, char *argv[])`
- 这是什么意思？
  - `argc` 将包含命令行上的字符串数量（可执行文件计为 1，每个参数加 1）。这里 `argc` 是 2
    - `$ touch a.txt`
  - `argv` 是一个指向数组的指针，该数组包含字符串形式的参数。

# C 基本变量类型

- 必须声明变量的类型
  - 强变量类型——类型不能更改。 例如. `int var = 2;`

类型	描述	例子
• char	• 8位, ASCII	• 'a', 'A', '\n', 12
• int	• 整数值 (正、负、0) , $\geq 16$ 位, 一般为32位	• 0, 78, -217, 0x2E
• unsigned int	• 整数值 (正、0)	• 0, 6, 35102
• short	• 整数值 (正、负、0) , $\geq 16$ 位, 一般为16位	• 0, -8, 32767
• long	• 整数值 (正、负、0) , $\geq 32$ 位, 一般为32位	• 0, 78, -217, 301713123194
• long long	• 整数值 (正、负、0) , $\geq 32$ 位, 一般为64位	• 31705192721092512
• float	• 单精度浮点数, 32位, IEEE 754	• 0.0, 3.14, 6.02e3
• double	• 双精度浮点数, 64位, IEEE 754	• ^
• etc.		

# 类型的本质

- 类型的本质是对二进制串的不同理解方式
  - 计算机中本只有0和1，理解的方式多了，便成了数据类型

- <https://godbolt.org/z/bP1rbWTae>



4nsw3r/vidar/CTF/网络安全/不解答vscode问题

今日C语言问题：给定一个类型为 int 或 float 类型的变量，如何使用程序判断这个变量的类型是 int 还是 float?

```
int num1=0;
int num2=1;
int num3=-1;
int num4=1.5;
float num5=0;
float num6=1;
float num7=-1;
float num8=1.5;
```

# C 中的常量

- 常量在声明中被分配一次类型值； 在程序的整个执行过程中， 值不能改变
  - `const float GOLDEN_RATIO = 1.618;`
  - `const int DAYS_IN_WEEK = 7;`
  - `const double THE_LAW = 2.99792458e8;`
  - 您可以拥有任何基本C变量类型的常量版本。
- 宏定义
  - 本质是预处理器的文本替换
  - `#define MAXLENGTH 1000`

# C 运算符

- 算术： +、 -、 \*、 /、 %
- 赋值： =
  - type var\_name;  $\Rightarrow$  变量声明
  - type var\_name = var\_value;  $\Rightarrow$  初始化
- 扩充赋值： +=、 -=、 \*=、 /=、 %=、 &=、 |=、 ^=、 <<=、 >>=
- 按位逻辑： ~、 &、 |、 ^
- 按位移位： <<、 >>
- 布尔逻辑： !、 &&、 ||
- 相等性测试： ==、 !=
- 子表达式分组： (、 )
- 顺序关系： <、 <=、 >、 >=
- 递增/递减： ++、 --
- 成员访问： .、  $\rightarrow$ 、 [、 ]
- 三元运算符
  - expr ? true\_ret : false\_ret
- 取地址运算符： &
- 解引用运算符： \*
- 类型转换运算符： (type)
- sizeof

# 布尔逻辑

- C 中什么计算结果为 FALSE?
  - 0
  - NULL
  - 假值表达式 (例如 `1 == 2`)
- C 中什么计算结果为 TRUE?
  - ...其他所有内容
  - 非零数字
  - 非 NULL 的指针
  - 真值语句 (例如 `1 == 1`)
- `true` 和 `false` 仅当包含 `stdbool.h` (标准布尔) 标头时才能使用
  - `#include <stdbool.h>`

# C 中的控制流

- 语句可以是代码块 `{ }` 或只是一个独立的语句
- if-else
  - `if (expr) statement`
  - `if (x == 0) y++;`
  - `if (x == 0) {y++;}`
  - `if (x == 0) {y++; j = j + y;}`
  - `if (expr) statement1 else statement2`
- switch case
- while
  - `while (expr)`
- for
  - `for (initialize; check; update) statement`

# 函数调用 (1/2)

- 在编程中，函数是一段可重复使用的代码块，用于执行特定的任务或完成特定的操作。函数可以接受输入参数，并且可以返回一个值或执行一些操作
- 代码重用
- 提高代码可读性
- 提高代码可测试性（单元测试）
- **抽象**



# 函数调用 (2/2)

```
1  #include<stdio.h>
2
3  int max(int a, int b); //函数声明, 注意; 不要忘记
4
5  int main(void)
6  {
7      int num1 = 1;
8      int num2 = 2;
9      //函数名(参数1, 参数2.....)
10     int num3 = max(num1,num2); //调用函数, num1和num2分别作为参数a和参数b, 将函数的返回值赋给num3
11
12     printf("the max number is %d\n", num3);
13     return 0;
14 }
15 //返回类型 函数名 (参数1, 参数2.....) {具体实现}
16 int max(int a, int b)
17 {
18     if (a > b)
19     {
20         return a;
21     }
22     return b;
23 }
```

# 数组

- C语言数组是通过下标索引的一组数据。其中每个元素都有一个唯一的索引，可以通过索引来访问和操作数组中的元素。
  - `int array[20];`
  - 从0开始计数  $\text{index} \in [0, 19]$

# 字符串

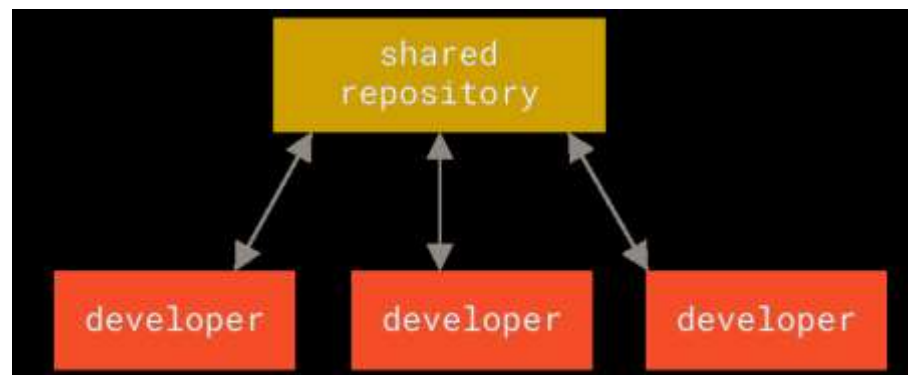
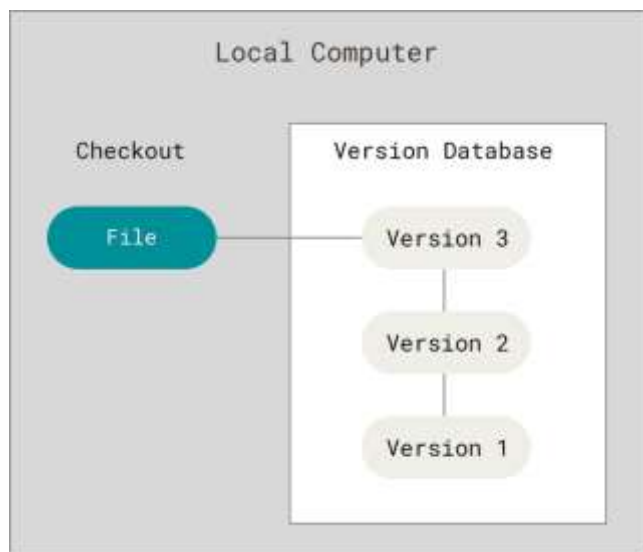
- 以字符'\0'为结束的一组字符

# Agenda

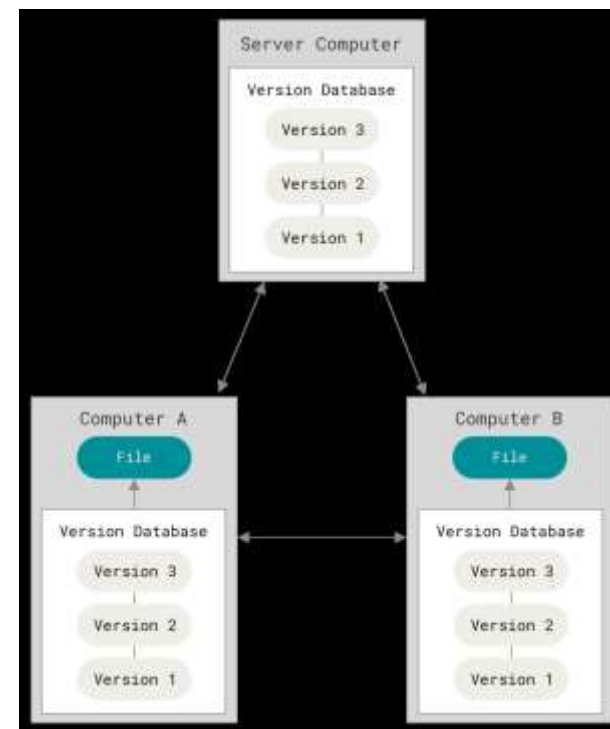
- 如何科学地提问
- 为什么学习C语言
- 怎么学习C语言
- 用什么写C语言
- 数制与码制
- C语言基础语法
- **命令行工具**
- VsCode和vim

# Git

- Git是分布式版本控制系统的一大代表
- 本地版本控制系统
- 集中化的版本控制系统

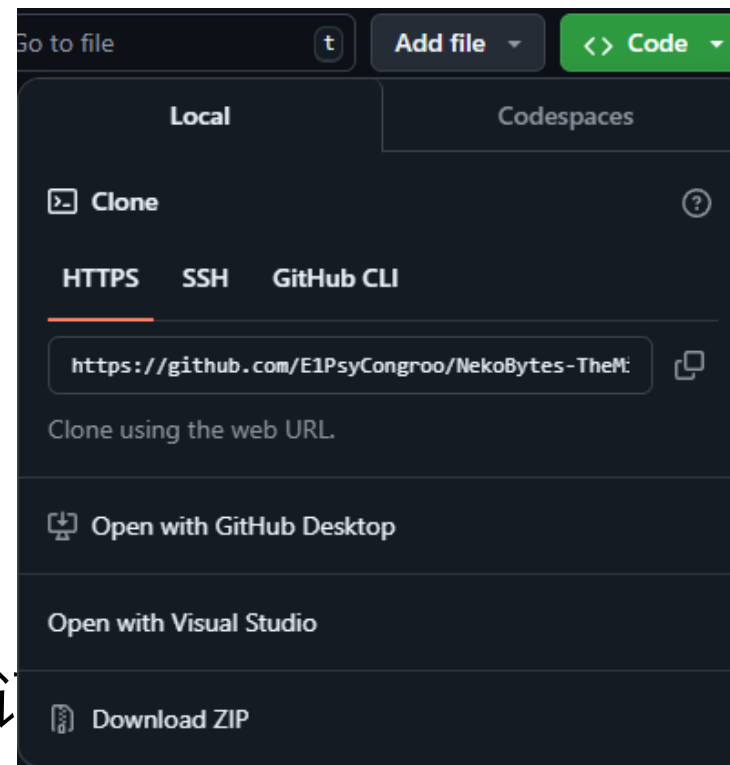


- 分布式版本控制系统



# Git的命令行接口

- `sudo apt install git`
- `git help <command>`
- `git init`
- `git status`
- `git add <filename>`
- `git commit`
- `git log`
- `git log --all --graph --decorate`: 可视化历史记录
- **`git clone <url>`**



# 怎么学git

- 自己动手实操
  - Learn Git Branching <https://learngitbranching.js.org/> （没有项目也能实操）
- 推荐书籍：
  - Pro Git <https://git-scm.com/book/en/v2>

# tar

## 文件后缀

- .tar
- .tar.gz
- .tar.bz2
- .tar.xz

## 解压命令

- `tar -xvf <filename>`
- `tar -xzvf <filename>`
- `tar -xjvf <filename>`
- `tar -xJvf <filename>`



# GCC

- gcc 全称GNU project C and C++ compiler
- gcc是C语言的一种编译器
- 易混淆的三样东西
  - Visual Studio——IDE
  - Visual Studio Code——文本编辑器
  - GCC——编译器

# GCC的命令行接口

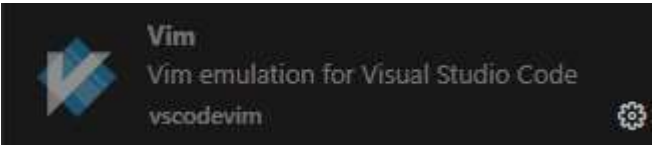
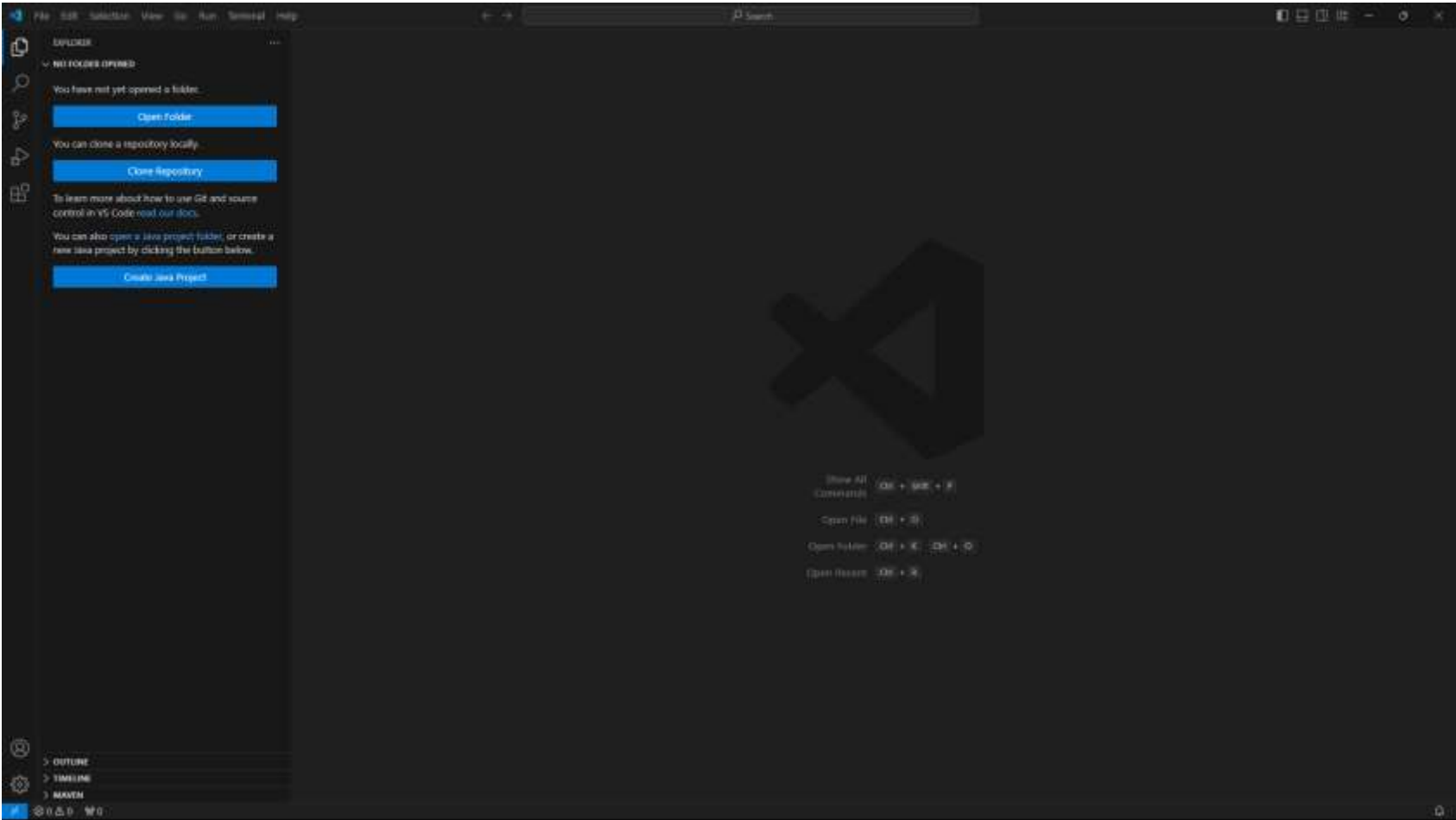
- 功能参数
  - -o 指定输出文件名
  - -Wall 开启警告
  - -O(2/3, ...) 优化
- 完整命令
  - gcc -o xxx xxx.c
- 运行
  - ./xxx
- 进阶用法
  - Make与Makefile

# Agenda

- 如何科学地提问
- 为什么学习C语言
- 怎么学习C语言
- 用什么写C语言
- 数制与码制
- C语言基础语法
- 命令行工具
- VsCode和vim

# VsCode环境配置

# VsCode中的Vim插件



# Vim的哲学

- 在编程的时候，你会把大量时间花在阅读/编辑而不是在写代码上。所以，Vim 是一个\_多模态\_编辑器：它对于插入文字和操纵文字有不同的模式。Vim 是可编程的（可以使用 Vimscript 或者像 Python 一样的其他程序语言），Vim 的接口本身也是一个程序语言：键入操作（以及其助记名）是命令，这些命令也是可组合的。Vim 避免了使用鼠标，因为那样太慢了；Vim 甚至避免用 上下左右键因为那样需要太多的手指移动。
- 这样的设计哲学使得 Vim 成为了一个能跟上你思维速度的编辑器。

