

20241210 计算概论B笔试资料

Updated 2101 GMT+8 Jan 6, 2025

2024 fall, Complied by Hongfei Yan

说明：

从网上公开的4年笔试题目看：1) 里面涉及到的编程，相对简单。例如：筛法、哥德巴赫猜想、回文、Dijkstra。2) 有的题目背景知识非cs，可能是生物、物理、数学，乍一看有点懵，多读两遍，就会发现其中cs原理。3) 计算题、编码应用题，需要加强训练。

2022年计概课程基本大纲，提到“数据结构初步：结构，链表及其应用”。笔试中有stack题目，没看到链表、queue、heap的题目。没看到补码、浮点数题目。

我们总共讲了三个计算机原理：1) ASCII编码，2) 虚拟地址空间（对于理解操作系统和内存管理至关重要），3) 图灵机（对深入理解算法和可计算性有着不可替代的作用）。后两个没有看到考题。第一个在考试中占据了相当大的比重。应熟悉大写字母、小写字母及数字所对应的十进制和十六进制编号，并掌握Python中的ord() 和 chr() 等函数，以及位运算的相关知识。

计算机原理

硬件和软件的组合形成可用的计算系统。硬件通常由软件指示以执行任何命令或指令。计算机硬件包括计算机的物理，有形部件或组件，例如主板（Main Board）、中央处理器（Central Processing Unit, CPU）、显示器、键盘和鼠标。计算机软件包括系统软件和应用软件。系统软件负责管理计算机系统中各种独立的硬件，使得它们可以协调工作。系统软件中常用的操作系统有 Linux、macOS、Unix、Windows等。应用软件是为了某种特定的用途而被开发的软件。常见的应用软件有文字处理、程序设计、网页浏览器、输入法和媒体播放器等。

对计算机设计具有重要意义的三个原理：图灵机、进程的虚拟地址空间和 ASCII 表。

1.ASCII 编码

在《计算机科学导论》[6]第三章中讲到，计算机外部的不同数据类型（Text, Number, Image, Audio, Video）的数据都采用统一的数据表示法转换后存入计算机中，输出时再还原回来，这种通用的格式称为位模式（bit pattern）。不同数据类型的存储如图1-16所示。

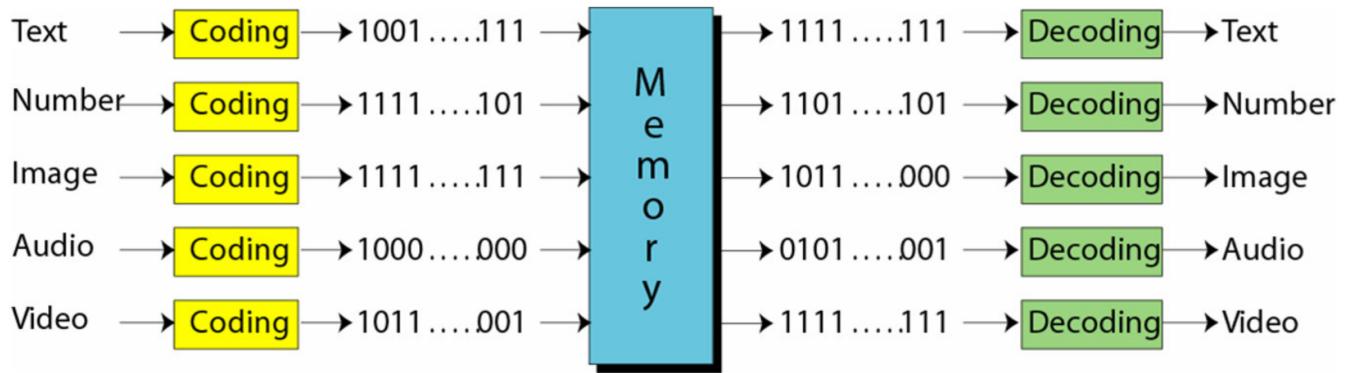


图1-16 不同数据类型的存储 (注: 图片来源为 Behrouz Forouzan[6], 2008年12月)

位 (bit) 是存储在计算机中的最小数据单位, 它是0或1。位代表设备的某一状态, 例如: 用1表示开关合上, 0表示断开。为了表示数据的不同类型, 使用位模式, 它是一个序列, 是0和1组和。通常长度位8的位模式称为1个字节 (byte)。

接下来我们来看文本 (Text) 是如何存储的。在任何语言中, 文本都是由一些符号组成。位模式可以表示任何一个符号。需要多少位来表示一个符号取决于该语言使用的符号的数量, 如表1-2所示。

表1-2 符号数量和位模式长度

符号数目	位模式的长度
2	1
4	2
8	3
16	4
256	8
65536	16

不同的位模式集合被设计用于表示文本符号, 每个集合被称为代码表, 表示符号的过程称为编码。美国国家标准协会 (American National Standards Institute, 记为 ANSI) 发布了美国信息交换标准码 (American Standard Code for Information Interchange, 记为 ASCII) 的代码表。该代码使用一串 7位二进制数表示每个符号, 可以定义 $2^7=128$ 种不同的符号, 包括英语中常用的 26 个大写字母, 26 个小写字母, 9 个字符, 以及标点符号等。如图1-17所示, 展示了四个大写字母对应的二进制位模式表示。

B **Y** **T** **E**
 | | | |
1000010 **1011001** **1010100** **1000101**

图1-17 大写字母的二进制位模式表示

编程语言，如果注释也写成英文，使用ASCII中包含的字符就够了。ASCII为控制字符保留了前32个代码，这些代码最初的目的不是为了携带可打印信息，而是为了控制使用ASCII的设备（如打印机）。例如，十进制字符10代表“换行”功能（使打印机推进纸张），字符27代表“转义”键，经常出现在普通[键盘]的左上角。代码127（全部七位开启），另一个特殊字符，相当于“删除”或“擦除”。需要掌握的是图1-18中划线标识的代码，10换行，13回车，48-57表示数字0-9，65-90表示大写字母，97-122表示小写字母。如果记不住确切的对应十进制字符，记住大写字母在小写字母之前也可以。

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0 000	000	NUL (null)	32	20 040	 	Space	64	40 100	@	Ø	96	60 140	`	`			
1	1 001	001	SOH (start of heading)	33	21 041	!	!	65	41 101	A	A	97	61 141	a	a			
2	2 002	002	STX (start of text)	34	22 042	"	"	66	42 102	B	B	98	62 142	b	b			
3	3 003	003	ETX (end of text)	35	23 043	#	#	67	43 103	C	C	99	63 143	c	c			
4	4 004	004	EOT (end of transmission)	36	24 044	$	\$	68	44 104	D	D	100	64 144	d	d			
5	5 005	005	ENQ (enquiry)	37	25 045	%	%	69	45 105	E	E	101	65 145	e	e			
6	6 006	006	ACK (acknowledge)	38	26 046	&	&	70	46 106	F	F	102	66 146	f	f			
7	7 007	007	BEL (bell)	39	27 047	'	'	71	47 107	G	G	103	67 147	g	g			
8	8 010	010	BS (backspace)	40	28 050	((72	48 110	H	H	104	68 150	h	h			
9	9 011	011	TAB (horizontal tab)	41	29 051))	73	49 111	I	I	105	69 151	i	i			
10	A 012	012	LF (NL line feed, new line)	42	2A 052	*	*	74	4A 112	J	J	106	6A 152	j	j			
11	B 013	013	VT (vertical tab)	43	2B 053	+	+	75	4B 113	K	K	107	6B 153	k	k			
12	C 014	014	FF (NP form feed, new page)	44	2C 054	,	,	76	4C 114	L	L	108	6C 154	l	l			
13	D 015	015	CR (carriage return)	45	2D 055	-	-	77	4D 115	M	M	109	6D 155	m	m			
14	E 016	016	SO (shift out)	46	2E 056	.	.	78	4E 116	N	N	110	6E 156	n	n			
15	F 017	017	SI (shift in)	47	2F 057	/	/	79	4F 117	O	O	111	6F 157	o	o			
16	10 020	020	DLE (data link escape)	48	30 060	0	0	80	50 120	P	P	112	70 160	p	p			
17	11 021	021	DC1 (device control 1)	49	31 061	1	1	81	51 121	Q	Q	113	71 161	q	q			
18	12 022	022	DC2 (device control 2)	50	32 062	2	2	82	52 122	R	R	114	72 162	r	r			
19	13 023	023	DC3 (device control 3)	51	33 063	3	3	83	53 123	S	S	115	73 163	s	s			
20	14 024	024	DC4 (device control 4)	52	34 064	4	4	84	54 124	T	T	116	74 164	t	t			
21	15 025	025	NAK (negative acknowledge)	53	35 065	5	5	85	55 125	U	U	117	75 165	u	u			
22	16 026	026	SYN (synchronous idle)	54	36 066	6	6	86	56 126	V	V	118	76 166	v	v			
23	17 027	027	ETB (end of trans. block)	55	37 067	7	7	87	57 127	W	W	119	77 167	w	w			
24	18 030	030	CAN (cancel)	56	38 070	8	8	88	58 130	X	X	120	78 170	x	x			
25	19 031	031	EM (end of medium)	57	39 071	9	9	89	59 131	Y	Y	121	79 171	y	y			
26	1A 032	032	SUB (substitute)	58	3A 072	:	:	90	5A 132	Z	Z	122	7A 172	z	z			
27	1B 033	033	ESC (escape)	59	3B 073	;	;	91	5B 133	[[123	7B 173	{	{			
28	1C 034	034	FS (file separator)	60	3C 074	<	<	92	5C 134	\	\	124	7C 174	|				
29	1D 035	035	GS (group separator)	61	3D 075	=	=	93	5D 135]]	125	7D 175	}	}			
30	1E 036	036	RS (record separator)	62	3E 076	>	>	94	5E 136	^	^	126	7E 176	~	~			
31	1F 037	037	US (unit separator)	63	3F 077	?	?	95	5F 137	_	_	127	7F 177		DEL			

Source: www.LookupTables.com

图1-18 ASCII代码表（注：图片来源为lookuptables.com。增加了两条横线和三个矩形框，用来突出重点部分）

如果有Python环境，可以在命令行中，输出ASCII表内容。

```

1 In[1]: import string
2 In[2]: string.printable
3
4 Out[2]:
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\''()*+,.-./:;
<=>?@[\`]^_`{|}~\t\n\r\x0b\x0c'

```

第 2 行的 string.printable 是调用了 string 类中的 printable 函数。如果想知道 string 中包含哪些函数，可以 dir。

```
1 In[3]: dir(string)
2 Out[3]:
3 ['Formatter',
4  'Template',
5  '_ChainMap',
6  '_TemplateMetaclass',
7  '__all__',
8  '__builtins__',
9  '__cached__',
10  '__doc__',
11  '__file__',
12  '__loader__',
13  '__name__',
14  '__package__',
15  '__spec__',
16  '_re',
17  '_sentinel_dict',
18  '_string',
19  'ascii_letters',
20  'ascii_lowercase',
21  'ascii_uppercase',
22  'capwords',
23  'digits',
24  'hexdigits',
25  'octdigits',
26  'printable',
27  'punctuation',
28  'whitespace']
```

ASCII 有结构特点。数字 0-9 以二进制的方式表示，其数值前缀为 0011。小写字母和大写字母在位模式上只有一位的差别，这就把大小写转换简化为一个范围测试（以避免转换不是字母的字符）和一个单一的比特操作。快速的大小写转换很重要，因为它经常被用于大小写搜索算法中。

例子：将任何 ASCII 字母变成小写字母。

在 ASCII 中，大写字母和小写字母的区别在于位 00100000（十六进制表示是 20h）的值，该位在小写字母中被打开。如果 "打开" 这个位，大写字母就会变成小写字母。（如果该字母已经是小写字母，20h 位已经打开；打开它没有任何区别）。“打开” 20h 位，被称为 ORing-in 位，因为使用的是位布尔 OR 操作符（bitwise Boolean OR operator，http://teaching.idallen.com/cst8214/08w/notes/bit_operations.txt）。

```
1 ...
2      01000001 = 41h = ASCII upper-case letter 'A'
3
4 OR      00100000 = 20h <-- this is the bit we want turned on
5
```

```

6      -----
7
8 EQUALS 01100001 = 61h = ASCII lower-case letter 'a'
9
10
11 uppera = ord('A')
12 lowera = uppera | 0x20    # bitwise OR with 20h
13 print(chr(lowera))
14
15 lowera = uppera | (1<<5)
16 print(chr(lowera))
17
18
19 lowera = ord('a')
20 uppera = lowera & ~0x20    # bitwise AND with 10111111
21 print(chr(uppera))
22
23 uppera = lowera & ~(1<<5)
24 print(chr(uppera))
25
26 #a
27 #a
28 #A
29 #A

```

2. 虚拟地址空间

三大计算机原理之一，@Book_my_flight_v0.3.md

计算机的基础架构自从 20 世纪 40 年代起就已经形成规范，包括处理器、存储指令和数据的内存、输入和输出设备。它通常叫作冯·诺依曼架构，以约翰·冯·诺依曼（德語：John Von Neumann，1903 年 12 月 28 日—1957 年 2 月 8 日）的名字来命名，他在 1946 年发表的论文里描述了这一架构。论文的开头句，用现在的专业术语来说就是，CPU 提供算法和控制，而 RAM 和磁盘则是记忆存储，键盘、鼠标和显示器与操作人员交互。其中需要重点理解的是与存储相关的进程的虚拟地址空间。

在《深入理解计算机系统》[8]第一章中讲到了进程的虚拟地址空间。虚拟存储器是一个抽象概念，它为每个进程提供了一个假象，好像每个进程都在独占地使用主存。每个进程看到的存储器都是一致的，称之为虚拟地址空间。如图1-15所示的是 Linux 进程的虚拟地址空间（其他 Unix 系统的设计与此类似）。在 Linux 中，最上面的四分之一的地址空间是预留给操作系统中的代码和数据的，这对所有进程都一样。底部的四分之三的地址空间用来存放用户进程定义的代码和数据。请注意，图中的地址是从下往上增大的。

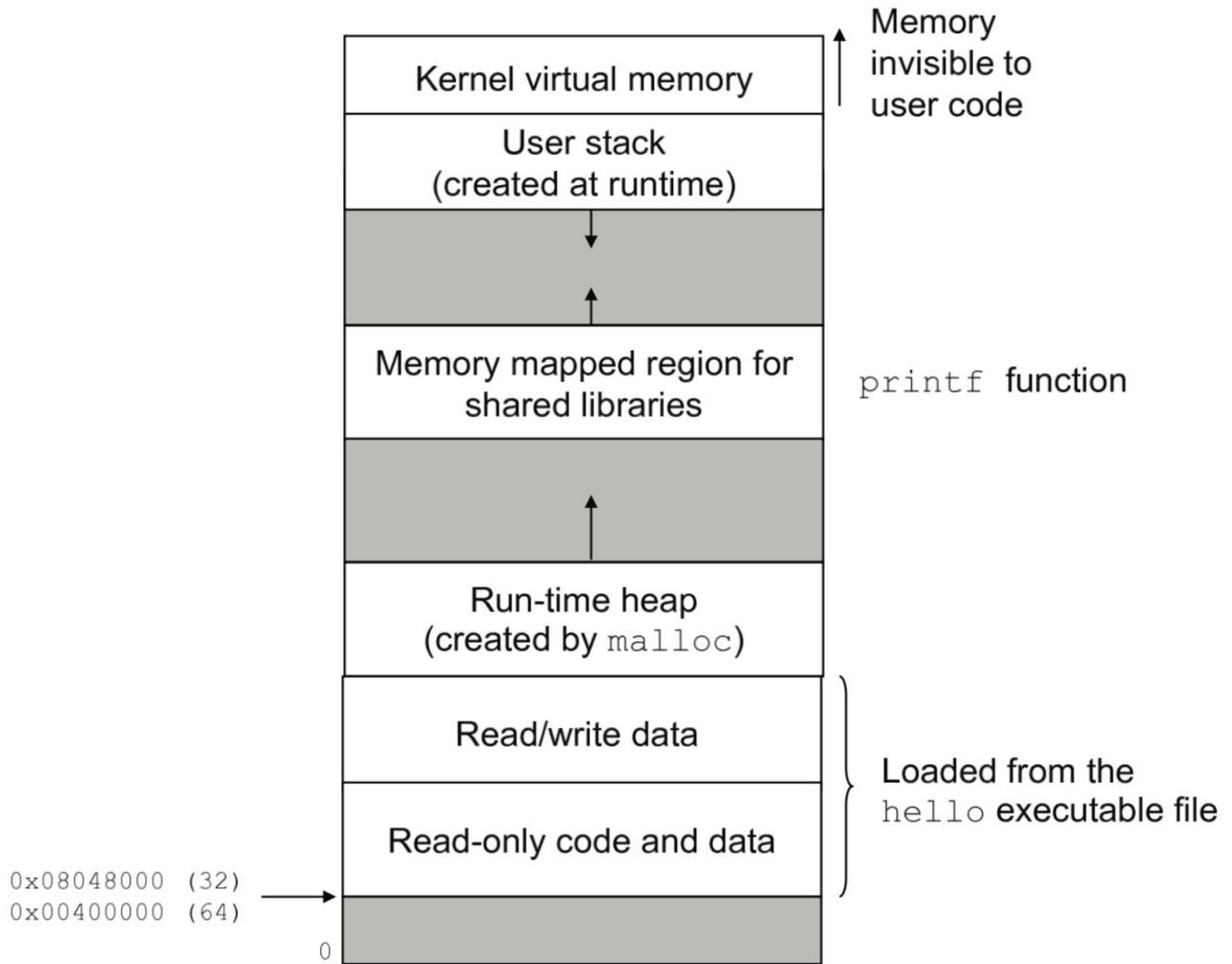


图1-15 进程的虚拟地址空间 (Process virtual address space) (注: 图片来源为 Randal Bryant[8], 2015年3月)

每个进程看到的虚拟地址空间由准确定义的区 (area) 构成，每个区都有专门的功能。简单看下每一个区，从最低的地址开始，逐步向上研究。

- 程序代码和数据 (code and data)。代码是从同一固定地址开始，紧接着的是和全局变量相对应的数据区。代码和数据区是由可执行目标文件直接初始化的，示例中就是可执行文件hello。
- 堆 (heap)。紧随代码和数据区之后的是运行时堆 (Run-time heap)。代码和数据区是在进程一旦开始运行时就被指定了大小的，与此不同，作为调用像 malloc 和 free 这样的 C 标准库函数的结果，堆可以在运行时动态地扩展和收缩。
- 共享库 (shared libraries)。在地址空间的中间附近是一块用来存放像标准库和数学库这样共享库的代码和数据的区域。共享库的概念非常强大。
- 栈 (stack)。位于用户虚拟地址空间顶部的是用户栈，编译器用它来实现函数调用。和堆一样，用户栈 (User stack) 在程序执行期间可以动态地扩展和收缩。特别地，每次我们调用一个函数时，栈就会增长。每次我们从函数返回时，栈就会收缩。
- **内核虚拟存储器 (kernel virtual memory)**。内核是操作系统总是驻留在存储器中的部分。地址空间顶部是为内核预留的。应用程序不允许读写这个区域的内容或者直接调用内核代码定义的函数。

虚拟存储器的运作需要硬件和操作系统软件间的精密复杂的互相合作，包括对处理器生成的每个地址的硬件翻译。基本思想是把一个进程虚拟存储器的内容存储在磁盘上，然后用主存作为磁盘的高速缓存。

对写程序重要，能感觉到程序在计算机中哪个位置，为什么出错。

1.7.3 虚拟内存

虚拟内存是一个抽象概念，它为每个进程提供了一个假象，即每个进程都在独占地使用主存。每个进程看到的内存都是一致的，称为虚拟地址空间。图 1-13 所示的是 Linux 进程的

虚拟地址空间（其他 Unix 系统的设计也与此类似）。在 Linux 中，地址空间最上面的区域是保留给操作系统中的代码和数据的，这对所有进程来说都是一样的。地址空间的底部区域存放用户进程定义的代码和数据。请注意，图中的地址是从下往上增大的。

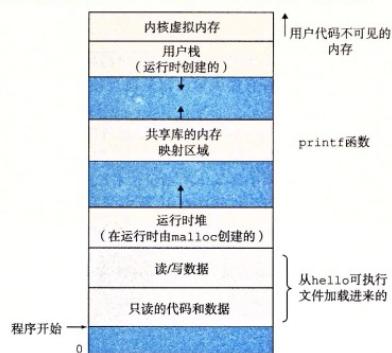


图 1-13 进程的虚拟地址空间

每个进程看到的虚拟地址空间由大量准确定义的区构成，每个区都有专门的功能。在本书的后续章节你将学到更多有关这些区的知识，但是先简单了解每一个区是非常有益的。我们从最低的地址开始，逐步向上介绍。

- 程序代码和数据。对所有的进程来说，代码是从同一固定地址开始，紧接着的是和 C 全局变量相对应的数据位置。代码和数据区是直接按照可执行目标文件的内容初始化的，在示例中就是可执行文件 hello。在第 7 章我们研究链接和加载时，你会学习更多有关地址空间的内容。
- 堆。代码和数据区后紧随着的是运行时堆。代码和数据区在进程一开始运行时就被指定了大小，与此不同，当调用像 malloc 和 free 这样的 C 标准库函数时，堆可以在运行时动态地扩展和收缩。在第 9 章学习管理虚拟内存时，我们将更详细地研究堆。
- 共享库。大约在地址空间的中间部分是一块用来存放像 C 标准库和数学库这样的共享库的代码和数据的区域。共享库的概念非常强大，也相当难懂。在第 7 章介绍动态链接时，将学习共享库是如何工作的。
- 栈。位于用户虚拟地址空间顶部的是用户栈，编译器用它来实现函数调用。和堆一样，用户栈在程序执行期间可以动态地扩展和收缩。特别地，每次我们调用一个函数时，栈就会增长；从一个函数返回时，栈就会收缩。在第 3 章中将学习编译器是如何使用栈的。
- 内核虚拟内存。地址空间顶部的区域是为内核保留的。不允许应用程序读写这个区域的内容或者直接调用内核代码定义的函数。相反，它们必须调用内核来执行这些操作。

虚拟内存的运作需要硬件和操作系统软件之间精密复杂的交互，包括对处理器生成的每个地址的硬件翻译。基本思想是把一个进程虚拟内存的内容存储在磁盘上，然后用主存作为磁盘的高速缓存。第 9 章将解释它如何工作，以及为什么对现代系统的运行如此重要。

1.7.4 文件

文件就是字节序列，仅此而已。每个 I/O 设备，包括磁盘、键盘、显示器，甚至网络，都可以看成是文件。系统中的所有输入输出都是通过使用一小组称为 Unix I/O 的系统函数调用读写文件来实现的。

文件这个简单而精致的概念是非常强大的，因为它向应用程序提供了一个统一的视图，来看待系统中可能含有的所有各式各样的 I/O 设备。例如，处理磁盘文件内容的应用程序员可以非常幸福，因为他们无须了解具体的磁盘技术。进一步说，同一个程序可以在使用不同磁盘技术的不同系统上运行。你将在第 10 章中学习 Unix I/O。

3. 图灵机

艾伦·麦席森·图灵 (Alan Mathison Turing, 又译阿兰·图灵, 1912 年 6 月 23 日 – 1954 年 6 月 7 日)。阿兰·图灵在 1937 年首次提出一个通用计算设备的设想。设想所有的计算都可能在一种特殊的机器上执行, 这就图灵机 (Turning Machine)。他将模型建立在人们进行计算过程的行为上, 并将这些行为抽象到用于计算的机器的模型中。图灵机由两部分构成, 如图1-13所示。

- 一条**存储带** (tape) : 双向无限延长, 上有一个个方格 (field), 每个方格可以包含一个有限字母的字符。在一个真正的机器中, 磁带必须足够大, 以包含算法的所有数据。
- 一个**控制器**: 包含一个可以双向移动的**读写头** (head), 可以在所处方格中读写一个字符; 图灵机每时每刻都处于某种**状态** (current state), 是有限数量的状态中的一种; 可以接受设定好的**图灵程序** (program), 该程序是一个转换列表, 它决定了一个给定的 State 和 head 下字符的新状态, 一个必须写入 head 下方格的字符和 head 的运动方向, 即左、右或静止不动。

看见 状态, 想起dp

对于一个子串而言, 如果它是回文串 (Palindrome string), 并且长度大于 2, 那么将它首尾的两个字母去除之后, 它仍然是个回文串。

状态: `dp[i][j]` 表示子串 `s[i:j+1]` 是否为回文子串

状态转移方程: `dp[i][j] = dp[i+1][j-1] ^ (s[i] == s[j])`

动态规划中的边界条件, 即子串的长度为 1 或 2。对于长度为 1 的子串, 它显然是个回文串; 对于长度为 2 的子串, 只要它的两个字母相同, 它就是一个回文串。

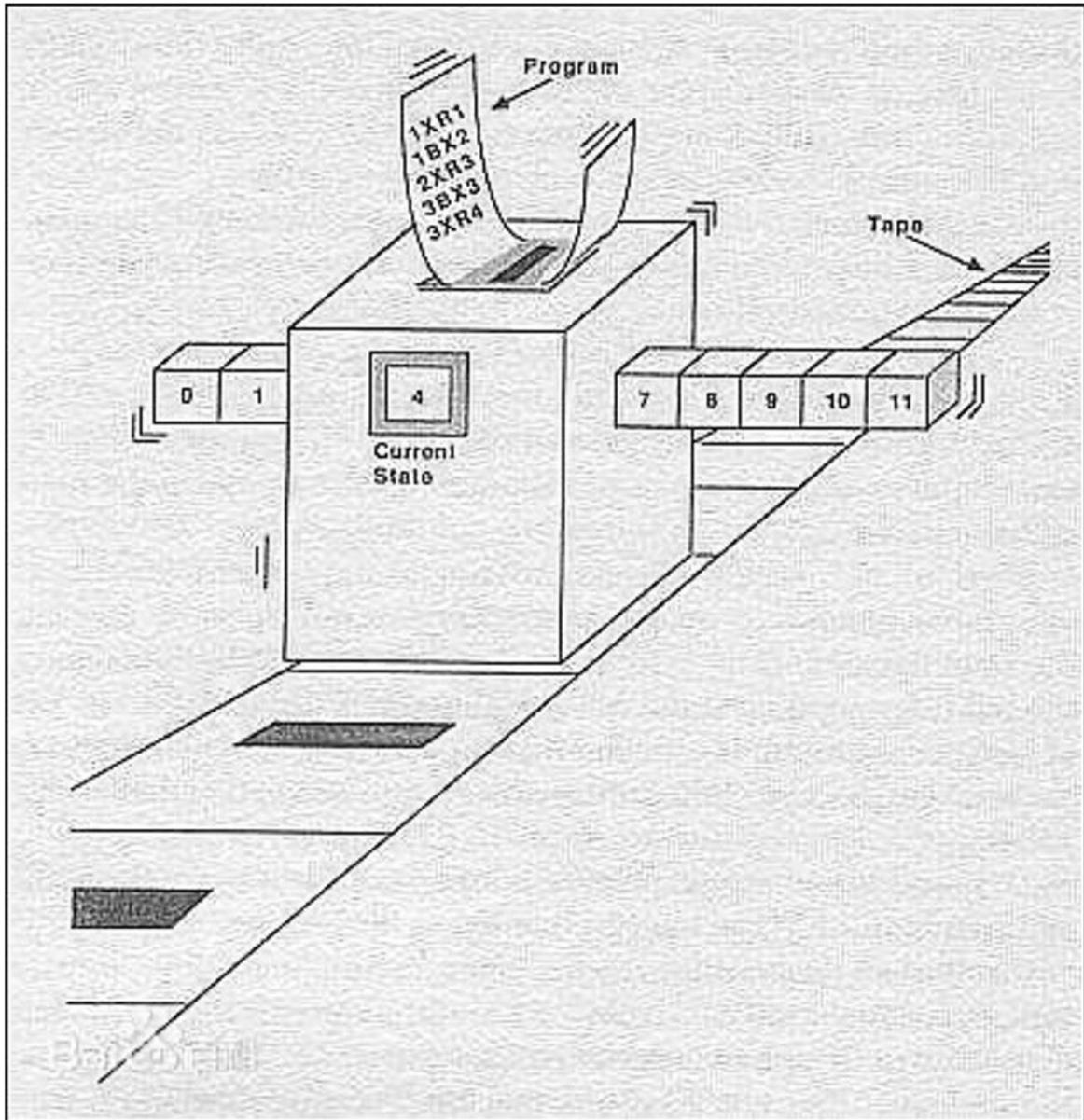


图1-13 由一条存储带和一个控制器构成的图灵机（注：图片来源为 baike.sogou.com, 2023年1月）

aturingmachine.com 上面展示了 Mike Davey 建造一台机器，如图1-14所示，体现图灵论文中提出的机器的经典外观和感觉。这台图灵机是由微控制器控制的，但它在运行时的操作只基于从 SD 卡加载的一组状态转换，以及从磁带上写入和读取的内容。虽然看起来磁带只是机器的输入和输出，但事实并非如此！磁带也不仅仅是机器的存储器。在某种程度上，磁带就是计算机。当磁带上的符号被简单的规则所操纵时，计算就发生了。图灵机的核心是读写头，传送磁带，并将磁带上的单元格适当地定位。它可以读取一个单元，确定那里写了什么，如果有的话，是什么符号。这台机器一次只工作在一个单元上，并且知道一个单元的情况。机器中的磁带是一卷1000英尺长的白色35毫米胶片带。字符1和0，是由机器用黑色笔写的。

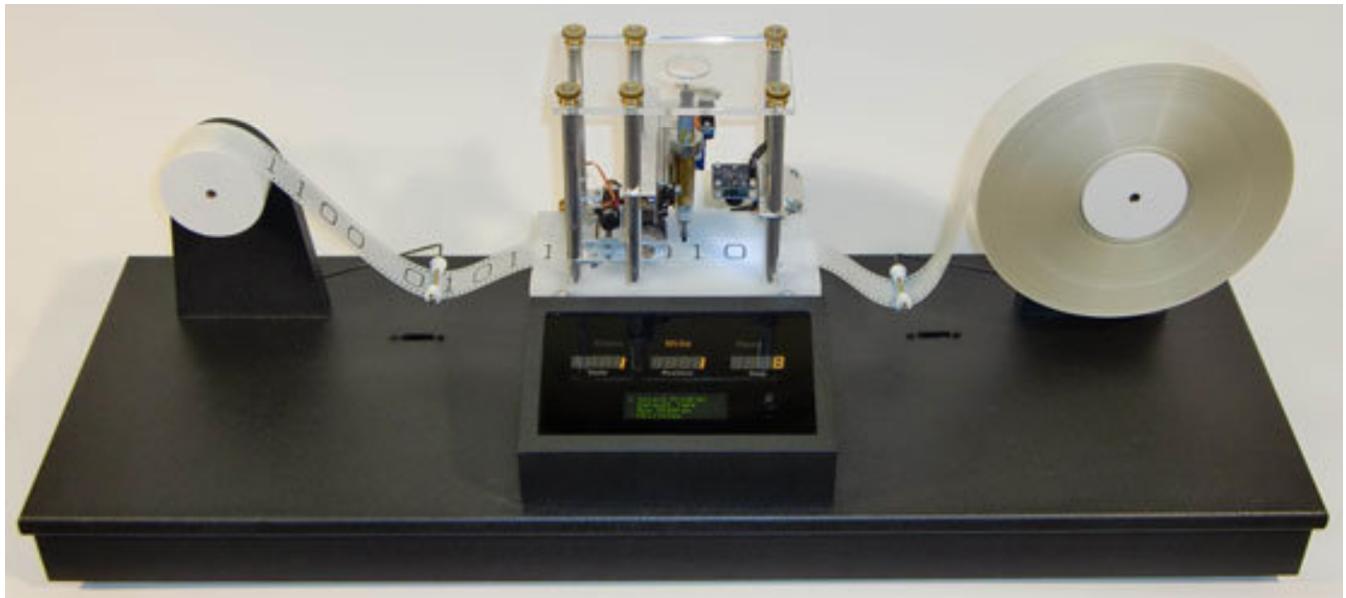


图1-14 体现图灵机模型的机器（注：图片来源为aturingmachine.com，2022年5月）

附录1A 是软件实现图灵机，<https://github.com/GMyhf/2019fall-cs101/tree/master/TuringMachine>。
参考：https://www.python-course.eu/turing_machine.php，增加了加法操作，实现下面14分钟视频中的加法。软件实现是用 Python 语言面向对象的编程（Object Orientation Programming, OOP）方式实现的。
Python的 OOP 语法，可以参考 <https://www.runoob.com/python3/python3-class.html>。

图灵机的构成，6分钟视频讲解在 https://www.bilibili.com/video/BV12B4y1X7QV/?spm_id_from=333.788

图灵机运作原理及示例，14分钟视频讲解在 https://www.bilibili.com/video/BV13v4y1w7yM/?spm_id_from=333.788

4.其他知识点

《计算机科学导论》，佛罗赞，2018年1月第4版。

4.1 进制转换

位置化数字系统中，在数字中符号所占据的位置决定了其表示的值。

表2-1是四种位置化系统（十进制、二进制、八进制和十六进制）的小结。

表2-2 显示了数字15在十进制中使用2个数码，在二进制中使用4个数码，在八进制中使用2个数码，在十六进制中仅仅使用1个数码。十六进制表示法显然是最短的。

表2-1 4种位置化系统的小结

系统	底	符号	例子
十进制	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	2345.56
二进制	2	0, 1	$(1001.11)_2$
八进制	8	0, 1, 2, 3, 4, 5, 6, 7	$(156.23)_8$
十六进制	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	$(A2C.A1)_{16}$

表2-2 4种位置化系统中的数字比较

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0	0	0	8	1000	10	8
1	1	1	1	9	1001	11	9
2	10	2	2	10	1010	12	A
3	11	3	3	11	1011	13	B
4	100	4	4	12	1100	14	C
5	101	5	5	13	1101	15	D
6	110	6	6	14	1110	16	E
7	111	7	7	15	1111	17	F

在十进制系统中，底 $b=10$ 并且用10个符号来表示一个数。该系统中的符号常被称为十进制数码或仅称为数码。

在二进制系统中，底 $b=2$ 并且用2个符号来表示一个数。该系统中的符号常被称为二进制数码或位。

在十六进制系统中，底 $b=16$ 并且用16个符号来表示一个数。该系统中的符号常被称为十六进制数码。

在八进制系统中，底 $b=8$ 并且用8个符号来表示一个数。该系统中的符号常被称为八进制数码。

可以从任意底转换到十进制。将数码乘以其在源系统中的位置量并求和便得到在十进制中的数。

能够将十进制数转换到与其等值的任意底。需要两个过程：一是用于整数部分，另一个是用于小数部分。**整数部分需要连除，而小数部分需要连乘。**

将数字从二进制转换到十六进制很容易，反之亦然。这是因为**二进制中的4位恰好是十六进制中的1位。**

将数字从二进制转换到八进制很容易，反之亦然。这是因为**二进制中的3位恰好是八进制中的1位。**

4.1.1 十进制-二进制

例2.14 将十进制数0.625转换为二进制数。

解 因为0.625没有整数部分，该例子显示小数部分如何计算。这里是以2为底。在左边一角写上这个十进制数。连续乘2，并记录结果的整数和小数部分。小数部分移到右边，整数部分写在每次运算的下面。当小数部分为0，或达到足够的位数时结束。结果是 $0.625 = (0.101)_2$ 。

1	十进制	0.625 → 0.25 → 0.50 → 0.00
2		↓ ↓ ↓
3	二进制	. 1 0 1

例2.15 下面演示如何将0.634转换为八进制数且精确到小数4位。结果是 $0.634 = (0.5044)_8$ 。

1	十进制	0.634 → 0.072 → 0.576 → 0.608 → 0.864
2		↓ ↓ ↓ ↓
3	八进制	. 5 0 4 4

例2.16 下面演示如何将十进制数178.6转换为十六进制数，且精确到1位小数。结果是 $178.6 = (B2.9)_{16}$ 。注意，以16为底时除或乘以16。

1	十进制	0 ← 11 ← 178 0.6 → 0.6
2		↓ ↓ ↓
3	十六进制	B 2 . 9

例2.17 把小的十进制数（通常小于256）转换为二进制数有一个变通的方法，即把这个数分解为下列二进制位置量对应数的和。

1	位置量	$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$
2	十进制对等量	128 64 32 16 8 4 2 1

使用该表可以转换165为二进制数 $(10100101)_2$ ，如下所示：

1	十进制	$165 = 128 + 0 + 32 + 0 + 0 + 4 + 0 + 1$
2	二进制	1 0 1 0 0 1 0 1

例2.18 当分母是2的幂次时，用类似的方法可以把十进制数小数转换为二进制。

1	位置量	$2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \quad 2^{-5} \quad 2^{-6} \quad 2^{-7}$
2	十进制对等量	$1/2 \quad 1/4 \quad 1/8 \quad 1/16 \quad 1/32 \quad 1/64 \quad 1/128$

使用该表可以转换 $27/64$ 为二进制数 $(0.011011)_2$ ，如下所示：

1	十进制 $27/64 = 16/64 + 8/64 + 2/64 + 1/64$
2	$1/4 + 1/8 + 1/32 + 1/64$

根据十进制对等的值排列这些分数:

1	十进制 $27/64 = 0 + 1/4 + 1/8 + 0 + 1/32 + 1/64$
2	二进制 0 1 1 0 1 1

4.1.2 二进制-十六进制

我们能轻松将数字从二进制转换到十六进制，反之亦然。这是因为在这两个进制之间存在一种关系：二进制中的4位恰好是十六进制中的1位。

例2.19 下面演示如何将二进制数 $(10011100010)_2$ 转换为十六进制数。

解 我们先将二进制数排为4位一组的形式: 100 1110 0010。注意最左边一组可能是1到4位不等。我们根据表2-2所示的值对照每组等量转换得到十六进制数 $(4E2)_{16}$ 。

例2.20 与十六进制数 $(24C)_{16}$ 相等的二进制数是多少？

解 将每个十六进制数码转换成4位一组的二进制数: 2→0010, 4→0100, 以及C→1100。该结果是 $(001001001100)_2$ 。

在二进制表示中，前面的零（即前导零）是可以去掉的，因为它们并不改变数值的大小。如果题目明确问内存中的表示形式，不能去掉前导零。

4.1.3 二进制-八进制

例2.21 下面演示如何将二进制数 $(101110010)_2$ 转换为八进制数。

解 每3位一组转换为1位八进制数码。根据表2-2所示的值对照每3位一组等量转换得到八进制数结果是 $(562)_8$ 。

例2.22 与 $(24)_8$ 相等的二进制数是多少？

解 将每个八进制数码写成对等的二进制位组，得到 $(010100)_2$ 。

4.1.4 八进制-十六进制

将数字从八进制转换到十六进制并不难，反之亦然。可以使用二进制系统作为中介系统。

该步骤如下：

- 从八进制转到十六进制，先将八进制转到二进制。将位数重排成4位一组，找到十六进制的对等值。
- 从十六进制转到八进制，先将十六进制转到二进制。将位数重排成3位一组，找到八进制的对等值。

4.2 数据存储

4.2.1 计算机内部的数据

所有计算机外部的数据类型的数据都采用统一的数据表示法转换后存入计算机中，当数据从计算机输出时再还原回来。这种通用的格式称为位模式。

1. 位

位 (bit, binary digit的缩写) 是存储在计算机中的最小单位，它是0或1。位代表设备的某一状态，这些设备只能处于两种状态之一。例如，开关要么合上要么断开。用1表示合上状态，0表示断开状态。电子开关就表示一个位。换句话说，一个开关能存储一个位的信息。今天，计算机使用各种各样的双态设备来存储数据。

2. 位模式

为了表示数据的不同类型，应该使用位模式，它是一个序列，有时也称为位流。图3-2展示了由16个位组成的位模式。它是0和1的组合。这就意味着，如果我们需要存储一个由16个位组成的位模式，那么需要16个电子开关。如果我们要存储1000个位模式，每个16位，那么需要16 000个开关。通常长度为8的位模式被称为1个字节 (byte)。有时用字 (word) 这个术语指代更长的位模式。

存储数字

在存储到计算机内存中之前，数字被转换为二进制系统。但是，这里还有两个问题需要解决：

1. 如何存储数字的符号。
2. 如何显示十进制小数点。

有多种方法可处理符号问题，本章后面陆续讨论。对于小数点，计算机使用两种不同的表示方法：**定点**和**浮点**。第一种用于把数字作为整数存储——没有小数部分，第二种把数字作为实数存储——带有小数部分。

4.2.2 存储整数

整数是完整的数字（即没有小数部分）。例如，134和-125是整数而134.23和-0.235则不是。整数可以被当作小数点位置固定的数字：小数点固定在最右边。因此，定点表示法用于存储整数。在这种表示法中，小数点是假定的，但并不存储。

但是，用户（或程序）可能将整数作为小数部分为0的实数存储。这是可能发生的，例如，整数太大以至于无法定义为整数来存储。为了更有效地利用计算机内存，无符号和有符号的整数在计算机中存储方式是不同的。

整数通常使用定点表示法存储在内存中。

1. 无符号表示法

无符号整数就是没有符号的整数。它的范围介于0到无穷大之间。然而，由于计算机不可能表示这个范围的所有整数，通常，计算机都定义了一个最大无符号整数的常量，称为最大无符号整数，它的值是 ($(2^n - 1)$)。这里 (n) 就是计算机中分配用于表示无符号整数的二进制位数。

(1) 存储无符号整数

使用以下步骤输入存储无符号整数：

- 首先将整数变成二进制数。
- 如果二进制位数不足 (n) 位，则在二进制整数的左边补0，使它的总位数为 (n) 位。如果位数大于 (n)，该整数无法存储。导致溢出的情况发生，我们后面要讨论这方面内容。

例3.1 将7存储在8位存储单元中。

解 首先将整数转换为二进制数 $(111)_2$ 。加5个0使总的位数为8位，即 $(00000111)_2$ 。再将该整数保存在存储单元中。注意，右下角的2用于强调该整数是二进制的，并不存储在计算机中。

把7变为二进制 → 1 1 1

在左边加5位 → 0 0 0 0 0 1 1 1

例3.2 将258存储在16位存储单元中。

解 首先把整数转换为二进制 $(100000010)_2$ ，加7个0使总的位数满足16位的要求，即得到 $(0000000100000010)_2$ 。再将该整数被存储在存储单元中。

把258变为二进制 → 1 0 0 0 0 0 0 0 1 0

在左边加7位 → 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0

(2) 译解无符号整数

输出设备译解内存中位模式的位串并转换为一个十进制的无符号整数。

例3.3 当译解作为无符号整数保存在内存中的位串00101011时，从输出设备返回什么？

解 使用第2章的解题过程，二进制整数转换为十进制无符号整数43。

(3) 溢出

因为大小（即存储单元的位的数量）的限制，可以表达的整数范围是有限的。在n位存储单元中，我们可以存储的无符号整数仅为0到 $2^n - 1$ 之间。

2. 二进制补码表示法

几乎所有的计算机都使用二进制补码表示法来存储位于n位存储单元中的有符号整数。这一方法中，无符号整数的有效范围（0到 $2^n - 1$ ）被分为2个相等的子范围。第一个子范围用来表示非负整数，第2个子范围用于表示负整数。例如，如果n是4，该范围是0000到1111。这个范围分为两半：0000到0111以及1000到1111。

尽管整数的符号影响二进制整数存储时的每一位，但是首位（最左位）决定符号。如果最左位是0，该整数非负；如果最左位是1，该整数是负数。

在二进制补码表示法中，最左位决定符号。如果它是0，该整数为正，如果是1，该整数为负。

(1) 两种运算

在深入讨论这种表示法之前，我们需要介绍两种运算。第一种称为反码或取一个整数的反码。该运算可以应用到任何整数，无论是正的还是负的。该运算简单反转各个位，即把0位变为1位，把1位变为0位。

正数的补码和其原码相同；负数的补码是它的原码除符号位外逐位取反(即0变1，1变0)，最后在末位加1。

我看国外的教材都没有强调原码、反码，通常直接讲计算中数据表示的补码，在补码的操作中用到了反码？

你提到的这个问题其实反映了不同教学方法和教材侧重点的差异。在一些现代计算机体系结构教材中，原码、反码、补码的概念可能没有被单独强调，尤其是补码已经成为大多数计算机系统中实际使用的标准表示方法。这是因为补码具有一些非常重要的优点：

1. 统一的加减法：补码的最大优势是它使得加法和减法操作可以使用相同的硬件电路进行处理，不需要专门的减法器。反码和原码则需要额外的处理，特别是减法操作会变得更复杂。
2. 简化零的表示：在原码和反码中，零有两个表示（+0 和 -0），而在补码中，零只有一个表示，简化了计算机系统的设计。
3. 符号扩展：补码表示法在进行符号扩展时非常方便，尤其是进行位数增加时，符号位的扩展非常自然。

由于这些原因，现代计算机体系结构和操作系统等课程中，通常直接使用补码作为数据表示的基础，而不会过多涉及原码和反码。补码不仅在整数的表示中使用，而且在浮点数的表示中也起着关键作用。

在补码的操作中，确实会涉及到反码的概念。比如，减法可以通过将被减数的反码加一，得到补码，进而进行加法运算。因此，虽然教材中没有单独强调反码的操作，但实际上补码的运算机制中是隐含着反码的概念的。

总结来说，补码作为一种有效的数字表示方式，已经成为现代计算机系统的标准，它简化了很多计算，反码的操作其实是补码实现的一个重要部分。

我看国外的教材都没有强调原码、反码，通常直接讲计算机中数据表示的补码，在补码的操作中用到了反码？



在国外的计算机科学教材中，确实更倾向于直接教授补码（Two's Complement），这是因为补码是现代计算机系统中最常用的表示有符号整数的方法。补码不仅简化了硬件设计，而且使得加法和减法运算可以使用相同的电路实现，从而提高了效率。

原码、反码与补码的关系

1. 原码 (Sign-Magnitude Representation) :

- 最高位作为符号位，0表示正数，1表示负数；其余位表示数值的绝对值。
- 优点：直观易懂。
- 缺点：存在两个零（+0和-0），并且加法器需要处理符号位，增加了复杂性。

2. 反码 (One's Complement) :

- 正数的反码与其原码相同。
- 负数的反码是将其原码除符号位外的所有位取反（即0变1，1变0）。
- 优点：简化了加法操作，但仍存在两个零的问题。
- 缺点：仍然需要特殊处理符号位，并且在实际应用中不如补码普遍。

3. 补码 (Two's Complement) :

- 正数的补码与其原码相同。
- 负数的补码是先求其反码，然后对结果加1。
- 优点：解决了双零问题，简化了加法器设计，能够用统一的方式处理加法和减法运算。
- 缺点：对于初学者来说可能不太直观，但一旦理解其原理，就非常实用。

补码中的反码

虽然国外教材可能不特别强调反码，但在学习补码的过程中，反码的概念实际上是一个重要的中间步骤。具体来说，在计算一个数的补码时，我们首先会得到该数的反码，然后再加1。因此，尽管反码不是最终的数据表示形式，它仍然是理解和掌握补码的一个关键部分。

教学重点

由于补码是最常用的数据表示方法，教学上通常会集中讲解补码及其相关运算规则。而原码和反码更多地是为了帮助学生理解补码的概念和发展历史，以及了解不同编码方式之间的转换过程。

实际应用

在实际编程或硬件设计中，程序员和工程师几乎总是使用补码来表示有符号整数，因为这是大多数现代计算机体系结构所采用的标准。因此，从实用角度来看，直接学习补码是更加高效的选择。

总之，虽然国外教材可能不会专门讨论原码和反码，但这些概念仍然是理解补码的基础。如果你有兴趣深入了解这些编码方式，研究它们的转换过程将有助于你更好地掌握补码的工作原理。

例3.8 下面显示如何取整数00110110的反码。

原来的模式： 0 0 1 1 0 1 1 0

进行反码运算： 1 1 0 0 1 0 0 1

例3.9 下面显示如果进行2次反码运算，就可以得到原先的整数。

原来的模式： 0 0 1 1 0 1 1 0

进行1次反码运算： 1 1 0 0 1 0 0 1

进行2次反码运算： 0 0 1 1 0 1 1 0

第2种运算称为二进制中的**补码**或取一个整数的补码。该运算分为两步：首先，从右边复制位，直到有1被复制；接着，反转其余的位。

例3.10 下面显示如何取整数00110100的补码。

原来的模式： 0 0 1 1 0 1 0 0

进行1次补码运算： 1 1 0 0 1 1 0 0

例3.11 下面显示我们如果进行2次补码运算，就可以得到原先的整数。

原来的模式： 0 0 1 1 0 1 0 0

进行1次补码运算： 1 1 0 0 1 1 0 0

进行2次补码运算： 0 0 1 1 0 1 0 0

另一种将一个整数进行补码运算的方法是先对它进行1次反码运算再加上1得到结果

(2) 以二进制补码格式存储整数

以二进制补码格式存储整数，计算机遵循以下步骤：

- 将整数变成n位的二进制数。
- 如果整数是正数或零，以其原样存储；**如果是负数，计算机取其补码存储。**

(3) 从二进制补码格式还原整数

从二进制补码格式还原整数，计算机遵循以下步骤：

- 如果最左位是1，计算机取其补码。如果最左位是0，计算机不进行操作。
- 计算机将该整数转换为十进制。

例3.12 用二进制补码表示法将整数28存储在8位存储单元中。

解 该整数是正数（无符号意思是正的），因此在把该整数从十进制转换成二进制后不再需要其他操作。注意，5个多余的零加到该整数的左边使其成为8位。

把28变为8位的二进制： 0 0 0 1 1 1 0 0

例3.13 用二进制补码表示法将整数-28存储在8位存储单元中。

解 该整数是负数，因此在转换成二进制后计算机对其进行二进制补码运算。

把28变为8位的二进制： 0 0 0 1 1 1 0 0

进行补码运算： 1 1 1 0 0 1 0 0

例3.14 用二进制补码表示法将存储在8位存储单元中的00001101还原成整数。

解 最左位是0，因此符号为正。该整数需要转换为十进制并加上符号即可。

4.2.3 存储实数

实数是带有整数部分和小数部分的数字。例如：23.7是一个实数（整数部分是23而小数部分是7/10）。尽管固定小数点的表示法可用于表示实数，但结果不一定精确或达不到需要的精度。以下两个例子说明了原因。

例3.16 在十进制系统中，假定用一种小数点右边2个数码，左边14个数码，总共16个数码的定点表示法。那么如果试图表示十进制数1.002 34，该系统的实数精度就会受损。该系统把这个数字存储为1.00。

例3.17 在十进制系统中，假定用一种小数点右边6个数码，左边10个数码，总共16个数码的定点表示法。那么如果试图表示十进制数236 154 302 345.00，该系统的实数精度就会受损。该系统把这个数字存储为6 154 302 345.00。整数部分比实际小了很多。

带有很大的整数部分或很小的小数部分的实数不应该用定点表示法存储。

1. 浮点表示法

用于维持正确度或精度的解决方法是使用浮点表示法。该表示法允许小数点浮动：我们可以在小数点的左右有不同数量的数码。使用这种方法极大地增加了可存储的实数范围：带有很大的整数部分或很小的小数部分的实数可以存储在内存中。在浮点表示法中，无论十进制还是二进制，一个数字都由3部分组成。

符号 位移量 定点数

第一部分是符号，可正可负。第二部分显示小数点应该左右移动构成实际数字的位移量。第三部分是小数点位置固定的定点表示法。

一个数字的浮点表示法由3部分组成：符号、位移量和定点数。

浮点表示法在科学中用于表示很小或很大的十进制数。在称作科学记数法的表示法中，定点部分在小数点左边只有1个数码而且位移量是10的幂次。

例3.18 下面演示用科学记数法(浮点表示法)表示的十进制数7 425 000 000 000 000 000.00。

解

实际数字 → + 7 425 000 000 000 000 000.00

科学记数法 → + 7.425 × 10²¹

这三部分为符号 (+)、位移量 (21) 以及定点部分 (7.425)。注意那个位移量就是指数。这种表示法的好处显而易见。即使在一张纸上写数字，科学记数法也是更短并更省空间。这种记数法使用了浮点表示法的概念，因为那个靠近例题右下方的小数点位置已经向左移了21位形成该数字的定点部分。一些程序设计语言和计算器按照 +7.425E21 来显示该数字，因为以10为底是不言而喻的。

例3.19 用科学记数法表示数字-0.000 000 000 000 023 2。

解 使用前例同样的方法，将小数点移到数码2之后，如下所示：

$$\text{实际数字} \rightarrow -0.000\ 000\ 000\ 000\ 023\ 2$$

$$\text{科学记数法} \rightarrow -2.32 \times 10^{-14}$$

注意这里指数是负的，因为小数点需要左移（14位）来构成原数字。我们可再次说该记数法中的数字由3部分组成：符号 (-)、实数 (2.32) 以及负整数 (-14)。一些程序设计语言和计算器按照-2.32E-14来显示该数字。

类似方法可用于表示很小或很大的存储于计算机中的二进制数字（整数和实数皆可）。

例3.20 用浮点格式表示数字(10100100000000000000000000000.00)₂。

解 使用前例同样的方法，小数点前只保留一位数字，如下所示：

$$\text{实际数字} \rightarrow +(10100100000000000000000000000.00)_2$$

$$\text{科学记数法} \rightarrow +1.01001 \times 2^{32}$$

注意我们不必担心从最右边的1开始的右侧的那些0，这是因为当我们使用实数(1.01001)₂时它们并不重要。指数显示为32，但它实际上以二进制存储在计算机中。我们也已经显示符号为正，但它可能作为1位存储。

例3.21 用浮点格式表示数字-(0.000000000000000000000101)₂。

解 使用前例同样的方法，小数点左边只留一个非零数码：

$$\text{实际数字} \rightarrow -(0.000000000000000000000101)_2$$

$$\text{科学记数法} \rightarrow -1.01 \times 2^{-24}$$

注意指数作为负的二进制数存储在计算机中。

2. 规范化

为了使表示法的固定部分统一，科学记数法（用于十进制）和浮点表示法（用于二进制）都在小数点左边使用了唯一的非零数码，这称为规范化。十进制系统中的数码可能是1到9，而二进制系统中该数码是1。在下面，d是非零数码，x是一个数码，y是0或1。

十进制 $\rightarrow \pm d.xxxxxxxxxxxxxxx$ 注意：d是1到9，每个x是0到9

二进制 $\rightarrow \pm 1.yyyyyyyyyyyy$ 注意：每个y是0或1

3. 符号、指数和尾数

在一个二进制数规范化之后，我们只存储了该数的3部分信息：符号、指数和尾数（小数点右边的位）。

例如，+1000111.0101规范化后变成为：

+	2^6	\times	1.0001110101
+	6		0001110101
↑ ↑ ↑			
符号	指数	尾数	

注意小数点和定点部分左边的位1并没有存储，它们是隐含的。

(1) 符号

一个数的符号可以用一个二进制位来存储（0或1）。

(2) 指数

指数（2的幂）定义为小数点移动的位数。注意幂可以为正也可以为负。余码表示法（后面讨论）是用来存储指数位的方法。

(3) 尾数

尾数是指小数点右边的二进制数。它定义了该数的精度。尾数是作为无符号整数存储的。如果我们把尾数和符号一起考虑，则可以说这个组合是作为符号加绝对值格式的整数存储的。但是，我们需要记住它不是整数，而是像整数那样存储的小数部分。我们强调这一点是因为在尾数中如果在数字的右边插入多余的零，这个值将会改变，而在一个真正的整数中，如果在数字的左边插入多余的零，这个值是不会改变的。

尾数是带符号的小数部分，像以符号加绝对值表示法存储的整数那样对待。

电气和电子工程师协会(IEEE)已定义了几种存储浮点数的标准。这里我们讨论其中两种最常用的--单精度和双精度。

4.3 计算机组成

计算机的组成部分成三大类(或子系统)：**CPU**、**主存和输入/输出子系统**。

中央处理单元(CPU)执行数据上的操作，它有三部分：**算术逻辑单元(ALU)**，**控制单元**和**一系列寄存器**。算术逻辑单元(ALU)负责算术、移位和逻辑运算。寄存器是快速独立的存储设备它可暂时地保留数据。控制单元控制CPU中每个部分的操作。

主存是**存储单元的集合**。每一个单元有一个称为**地址的标识符**。数据被传输到内存或从内存取出是以称为字的二进制位组的方式。内存中唯一可标识的单元总数称为**地址空间**。有两种内存可用：**随机存取存储器(RAM)**和**只读存储器(ROM)**。

输入/输出(I/O)子系统的设备集合允许计算机与外界交流，存储程序和数据，即使在计算机已关机时也可以。输入/输出设备分成两大类；非存储设备和存储设备。非存储设备允许CPU/内存与外界通信；存储设备可以存储以后被检索的大量信息。存储设备被分成磁的和光的。

1.6 存储设备形成层次结构

在处理器和一个较大较慢的设备(例如主存)之间插入一个更小更快的存储设备(例如高速缓存)的想法已经成为一个普遍的观念。实际上，每个计算机系统中的存储设备都被

组织成了一个存储器层次结构，如图 1-9 所示。在这个层次结构中，从上至下，设备的访问速度越来越慢、容量越来越大，并且每字节的造价也越来越便宜。寄存器文件在层次结构中位于最顶部，也就是第 0 级或记为 L0。这里我们展示的是三层高速缓存 L1 到 L3，占据存储器层次结构的第 1 层到第 3 层。主存在第 4 层，以此类推。



图 1-9 一个存储器层次结构的示例

存储器层次结构的主要思想是上一层的存储器作为低一层存储器的高速缓存。因此，寄存器文件就是 L1 的高速缓存，L1 是 L2 的高速缓存，L2 是 L3 的高速缓存，L3 是主存的高速缓存，而主存又是磁盘的高速缓存。在某些具有分布式文件系统的网络系统中，本地磁盘就是存储在其他系统中磁盘上的数据的高速缓存。

正如可以运用不同的高速缓存的知识来提高程序性能一样，程序员同样可以利用对整个存储器层次结构的理解来提高程序性能。第 6 章将更详细地讨论这个问题。

计算机中三个子系统的连接起重要的作用，因为在这些子系统间需要进行信息的通信。CPU 和内存通常被三个连接连在一起(每个称为总线)：**数据总线、地址总线和控制总线**。输入/输出设备通过**输入/输出控制器或接口**与总线相连，使用的控制器有多种，如今最常见的有：**SCSI、火线和USB**。

有两种方法输入/输出设备的寻址：**I/O独立寻址**和**I/O存储器映射寻址**。在I/O独立寻址方法中用来从内存读/写的指令完全不同于用来从输入/输出设备的读1写指令。在I/O存储器映射寻址方法中，CPU把I/O控制器中的每个寄存器看成是内存中的字。

如今，通用计算机使用称为程序的一组指令来处理数据。计算机执行程序，从输入数据创建输出数据。程序和数据都存储在内存中。CPU使用重复的机器周期一条接一条，从头到尾执行程序中的指令。简化的周期由三阶段组成：**取指令、译码和执行**。

有三种使CPU和输入/输出设备同步的方法：程序控制输入/输出、中断控制输入/输出和直接存储器存取(DMA)。

在最近的几十年中，计算机的体系结构和组织经历了许多变化。计算机体系结构分成两大类：CISC（复杂指令集计算机）和RISC（精简指令集计算机）。

现代计算机使用流水线技术来提高吞吐量。这个理念允许控制单元同时执行两个或三个阶段，这意味着下一条指令的处理可以在前一条结束前开始。例如：**28701:炸鸡排**

计算机传统上有一个控制单元、一个算术逻辑单元和一个内存单元。并行处理通过使用多指令流处理多数据流来改善吞吐量。

在计算机中存储不同类型的数据，在这些数据上的运算（数据运算）可以分为三大类：算术运算、逻辑运算和移位运算。重点讲解逻辑运算和移位运算，算术运算直接看考试题目。

4.4 逻辑运算

计算机中的数据是以位模式（bit pattern）存储的事实。逻辑运算是指那些应用于模式中的一个二进制位，或在两个模式中相应的两个二进制位的相同基本运算。这意味着我们可以在位层次上和模式层次上定义逻辑运算。模式层次上的逻辑运算是具有相同类型的位层次上的 n 个逻辑运算，这里的 n 就是模式中的位的数目。

4.4.1 位层次上的逻辑运算

一个位可能是0或1，可以假设“0”代表逻辑“假”，而“1”代表逻辑“真”。我们可以应用布尔代数中定义的运算去操纵二进制位。为纪念乔治·布尔（George Boole）而命名的布尔代数属于逻辑的特殊数学领域。我们将简单介绍4种被用来操纵二进制位的位层次上的运算：非（NOT）、与（AND）、或（OR）和异或（XOR）。

非（NOT）

x	NOT x
0	1
1	0

与（AND）

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

或（OR）

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

异或 (XOR)

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

显示了这4种位层次上运算的符号及其真值表。真值表定义了对于每一种可能的输入或输出值。注意每个运算符的输出总是一位，但输入可以是一位或两位。

1. 非 (NOT)

NOT运算符是一元操作符：它只有一个输入。输出位是输入位的相反，如果输入是0，则输出为1；如果输入为1，则输出为0。换言之，NOT运算符是输入的反转。NOT运算符的真值表只有两行，因为单个输入只有两种可能：0或1。

2. 与(AND)

AND运算符是二元运算符：它有两个输入。如果输入都是1，则输出为1，而在其他三种情况下，输出都是0。AND运算符的真值表有4行，因为两个输入有4种可能的输入组合。AND运算符有趣的一点是：如果输入中有一位是0，则不需要检查其他输入的相应的位便可迅速得到结果为0。

对于 $x = 0$ 或 1 , $x \text{ AND } 0 \rightarrow 0$ 和 $0 \text{ AND } x \rightarrow 0$ 。

3. 或 (OR)

OR运算符也是二元运算符：它有两个输入。如果输入都是0，则输出为0，而在其他三种情况下，输出都是1。OR运算符的真值表也有4行。OR运算符有时被称为包含或运算符，因为输出值为1时，不但有其中一个输入为1，而且还有当两个输入都是1。这与我们下面要介绍的运算符相对照。

OR运算符有趣的一点是：如果输入中有一位是1，则不需要检查其他输入的相应的位，便可迅速得到结果为1。当我们讨论此运算符在模式层次上的应用时，我们将用到这个特性。

对于 $x = 0$ 或 1 , $x \text{ OR } 1 \rightarrow 1$ 和 $1 \text{ OR } x \rightarrow 1$ 。

4. 异或 (XOR)

像或运算符一样，XOR运算符（发音为“exclusive-or”）也是二元运算符，只是有一点不同：如果输入都是1，则输出为0。我们能用另一种方式来看这个运算符：当输入相同时，则输出为0；当输入不同时，则输出为1。

例4.1

在英语中，我们使用连词“或”时，有时意思是包含或，有时意思却是异或。

- a. 句子“*I wish to have a car or a house*”使用的“or”就是包含或的意思，我希望有一辆车，一栋房子，或二者兼而有之。
- b. 句子“*Today is either Monday or Tuesday*”使用的“or”就是异或的意思，今天不是星期一就是星期二，但不能两个都是。

例4.2

XOR运算符其实不是新的运算符，我们能用其他三个运算符来模拟它。下面两个表达式是等价的。

$$x \text{ XOR } y \leftrightarrow [x \text{ AND } (\text{NOT } y)] \text{ OR } [(\text{NOT } x) \text{ AND } y]$$

如果我们给出两个表达式的真值表，等价就可以得到证明。

XOR的特性是：如果输入中的一位是1，那结果就是与其他输入中相应位相反。当我们讨论此运算符在模式层次上的应用时，我们将用到这个特性。

对于 $x = 0$ 或 1 , $x \text{ XOR } 1 \rightarrow \text{NOT } x$ 和 $x \text{ XOR } 1 \rightarrow \text{NOT } x$ 。

4.4.2 模式层次上的逻辑运算

相同的4个运算符（NOT、AND、OR和XOR）可以被应用到n位模式。效果就是对NOT运算来说，把每个运算符应用于每个位；对于其他3个运算符就是把每个运算符应用于相应的位对。

例4.3 用NOT（非）运算符来计算位模式10011000

解 结果显示如下，注意NOT运算符把每个0变成1，把每个1变成0。

1	NOT	1 0 0 1 1 0 0 0	输入
2		0 1 1 0 0 1 1 1	输出

例4.4 用AND（与）运算符来计算位模式10011000和00101010

解 结果显示如下，注意只有输入中相应的位都为1，输出中的位才为1。

1	AND	1 0 0 1 1 0 0 0	输入1
2		0 0 1 0 1 0 1 0	输入2
3		0 0 0 0 1 0 0 0	输出

例4.5 对位模式数值10011001和00101110应用或（OR）运算

解 结果显示如下，注意只有输入中相应的位都为0，输出中的位才为0。

1	OR	1 0 0 1 1 0 0 1	输入1
2		0 0 1 0 1 1 1 0	输入2
3		1 0 1 1 1 1 1 1	输出

例4.6 使用XOR（异或）运算符对10011001和00101110位运算

解 结果显示如下，注意该例中只有输入中相应的位都为0，输出中的位才为0。

1	XOR	1 0 0 1 1 0 0 1	输入1
2		0 0 1 0 1 1 1 0	输入2
3		1 0 1 1 0 1 1 1	输出

4种逻辑运算可用于修改位模式

(1) 求反

NOT运算符的唯一应用就是对整个模式求反。对模式应用此运算符把每个0变成1，把每个1变成0。例4.3显示了求反的效果。

(2) 使指定的位复位

与运算的一个应用就是把一个位模式的指定位复位（置0）。这种情况下，第二个输入称为掩码。掩码中的0位对第一个输入中相应的位进行复位。掩码中的1位使得第一个输入中相应的位保持不变。这是由AND运算符的特性决定的：如果输入中有一个是0，不管其他输入是什么，输出都是0。模式中的复位有许多应用，例如，如果一个图像使用的是每像素只有一位（黑白图像），那么我们能使用掩码和AND运算符使指定像素变黑。

例4.7 使用掩码复位模式的最左5位。用模式10100110测试掩码。

解 掩码是00000111，应用掩码的结果是：

1		1 0 1 0 0 1 1 0	输入
2	AND	0 0 0 0 0 1 1 1	掩码
3		0 0 0 0 0 1 1 0	输出

注意最右边的3位保持不变，而最左边5位，不管它们先前的值是什么，都被复位（变为0）。

(3) 对指定的位置位

或运算的一个应用是把一个位模式的指定位置位（置1）。我们再次使用掩码，但是一个不同的掩码。掩码中的1位对第一个输入中相应的位进行置位，而掩码中的0位使第一个输入中相应的位保持不变。这是由OR操作符的特性决定的：如果一个输入为1，不管其他输入是什么，输出都将是1。模式中的位置位有许多应用，例如，如果一个图像使用的是每像素只有一位（黑白图像），那么我们能使用掩码和OR运算符使指定像素变白。

例4.8 使用掩码把一个位模式的最左5位置位。使用10100110测试这个掩码。

解 此掩码为11111000。应用此掩码的结果为：

1	1 0 1 0 0 1 1 0	输入
2	OR 1 1 1 1 1 0 0 0	掩码
3	1 1 1 1 1 1 1 0	输出

(4) 使指定的位反转

异或运算的一个应用是使指定的位反转，我们再次使用掩码，但是一个不同的掩码。掩码中的1位对第一个输入中相应的位进行反转，而掩码中的0位使第一个输入中相应的位保持不变。这是由XOR操作符的特性决定的：如果一个输入为1，输出与相应的位相反。注意NOT运算符和XOR运算符间的区别。NOT运算符是将输入中所有的位求反，而XOR运算符只是对第一个输入中指定的位求反，正如掩码中所定义的。

例4.9 用掩码来反转一个模式的最左边5位。用模式10100110检验掩码。

解 掩码为11111000。运用掩码后的结果为：

1	1 0 1 0 0 1 1 0	输入
2	XOR 1 1 1 1 1 0 0 0	掩码
3	0 1 0 1 1 1 1 0	输出

4.5 移位运算

移位运算移动模式中的位，改变位的位置。它们能向左或向右移动位。我们可以把移位运算分成两大类：逻辑移位运算和算术移位运算。

4.5.1 逻辑移位运算

逻辑移位运算应用于不带符号位的数的模式。原因是这些移位运算可能会改变数的符号，此符号是由模式中最左位定义的。

1. 逻辑移位

逻辑右移运算把每一位向右移动一个位置。在n位模式中，最右位被丢弃，最左位填0。逻辑左移运算把每一位向左移动一个位置。在n位模式中，最左位被丢弃，最右位填0。

例4.10 对位模式10011000使用逻辑左移运算。

解 如下所示，最左位被丢弃，0作为最右位被插入。

1	1 0 0 1 1 0 0 0	原始的
2	0 0 1 1 0 0 0 0	移位后

2. 循环移位

循环移位运算（旋转运算）对位进行移位，但没有位被丢弃或增加。循环右移（或右旋转）把每一位向右移动一个位置，最右位被回环，成为最左位。循环左移（或左旋转）把每一位向左移动一个位置，最左位被回环，成为最右位。

例4.11 对位模式10011000使用循环左移运算。

解 解如下所示，最左位被回环，成为最右位。

1	1 0 0 1 1 0 0 0	原始的
2	0 0 1 1 0 0 0 1	移位后

4.5.2 算术移位运算

算术移位运算假定位模式是用二进制补码格式表示的带符号位的整数。算术右移被用来对整数除以2；而算术左移被用来对整数乘以2。这些运算不应该改变符号位（最左）。算术右移保留符号位，但同时也把它复制，放入相邻的右边的位中，因此符号被保存。算术左移丢弃符号位，接受它的右边的位作为符号位。如果新的符号位与原先的相同，那么运算成功，否则发生上溢或下溢，结果是非法的。

4.6 网络

计算机网络通常被划分为多个层次，每个层次负责不同的功能，并且有其特定的协议来实现这些功能。最常用的模型是 OSI（开放系统互连）七层模型和 TCP/IP 模型。下面是这两个模型中各层的主要协议：

4.6.1 OSI 七层模型

网络协议是指在计算机网络中，为了实现不同设备之间的通信而定义的一系列规则和约定。网络协议通常按照 OSI（开放系统互联）参考模型或者TCP/IP模型的各个层次进行划分。以下是网络中各层常见的协议：

1. 物理层（Physical Layer）

物理层负责通过物理媒介传输原始的比特流。它的主要任务是定义电气信号、传输介质等。

- 协议：

物理层通常没有具体的“协议”定义，但涉及的标准有：

- Ethernet (以太网)
- DSL (数字用户线路)
- 光纤传输协议（如Fiber Channel）
- USB
- Wi-Fi (也涉及到数据链路层，但物理层部分是无线通信标准)

2. 数据链路层（Data Link Layer）

数据链路层负责在物理链路上进行数据帧的传输与接收，保证数据的可靠性。

- 协议：
 - **Ethernet** (以太网)：用于局域网中的数据传输。
 - **PPP (Point-to-Point Protocol, 点对点协议)**：用于点对点链路，常用于拨号上网。
 - **Wi-Fi (无线局域网)**：用于无线通信。
 - **Frame Relay**：一种用于连接多个局域网的协议，较早期常见。
 - **HDLC (High-Level Data Link Control)**：一种面向比特的协议，用于帧的交换。

3. 网络层 (Network Layer)

网络层负责数据包从源主机到目的主机的传输，主要处理路由、寻址等问题。

- 协议：
 - **IP (Internet Protocol, 互联网协议)**：主要用于在不同网络之间传输数据包，常见的版本有IPv4和IPv6。
 - **ICMP (Internet Control Message Protocol, 互联网控制报文协议)**：用于发送控制消息，如ping命令。
 - **ARP (Address Resolution Protocol, 地址解析协议)**：将IP地址解析为MAC地址。
 - **RARP (Reverse ARP)**：将MAC地址解析为IP地址（较少使用）。
 - **OSPF (Open Shortest Path First)**：一种动态路由协议，常用于大型网络中的路由选择。
 - **BGP (Border Gateway Protocol)**：用于互联网中的路由选择，支持跨自治系统的路由。

4. 传输层 (Transport Layer)

传输层负责端到端的数据传输，确保数据的可靠性和顺序。

- 协议：
 - **TCP (Transmission Control Protocol, 传输控制协议)**：面向连接、可靠的传输协议，确保数据的顺序性和完整性。
 - **UDP (User Datagram Protocol, 用户数据报协议)**：无连接、低延迟的传输协议，不保证可靠性。
 - **SCTP (Stream Control Transmission Protocol)**：面向消息的协议，结合了TCP和UDP的一些特点。

5. 会话层 (Session Layer)

会话层负责建立、管理和终止会话。它确保不同计算机间的通信保持正确的顺序。

- 协议：
 - **NetBIOS**：提供网络会话管理服务。
 - **RPC (Remote Procedure Call, 远程过程调用)**：允许不同主机上的程序进行通信。
 - **SMB (Server Message Block)**：用于共享文件和打印机资源。
 - **TLS/SSL (Transport Layer Security / Secure Sockets Layer)**：用于在会话层加密数据传输。

6. 表示层 (Presentation Layer)

表示层负责数据的格式化、加密和压缩，以便于应用层能够正确理解数据。

- 协议：

- **JPEG、GIF、PNG**: 常见的图像文件格式。
- **MPEG、MP3**: 常见的音频和视频压缩格式。
- **ASCII、EBCDIC**: 字符编码格式。
- **TLS/SSL**: 可以在表示层加密数据。

7. 应用层 (Application Layer)

应用层是与用户直接交互的层，负责提供应用程序需要的通信服务。

- 协议：

- **HTTP (Hypertext Transfer Protocol, 超文本传输协议)** : 用于网页浏览。
- **HTTPS (Hypertext Transfer Protocol Secure)** : HTTP的安全版本，使用SSL/TLS进行加密。
- **FTP (File Transfer Protocol, 文件传输协议)** : 用于文件传输。
- **SMTP (Simple Mail Transfer Protocol, 简单邮件传输协议)** : 用于发送电子邮件。
- **POP3 (Post Office Protocol 3, 邮局协议3)** : 用于接收电子邮件。
- **IMAP (Internet Message Access Protocol, 互联网邮件访问协议)** : 比POP3更强大的电子邮件接收协议。
- **DNS (Domain Name System, 域名系统)** : 用于将域名解析为IP地址。
- **Telnet**: 用于远程登录服务器。
- **SSH (Secure Shell)** : 用于远程登录，具备加密功能。
- **DHCP (Dynamic Host Configuration Protocol, 动态主机配置协议)** : 用于动态分配IP地址。

每一层都有其对应的协议和作用，它们共同合作使得网络通信得以顺畅进行。

4.6.2 TCP/IP 四层模型

1. 链路层 (Link Layer)

- 协议：与 OSI 数据链路层类似，包括 Ethernet, Wi-Fi 等。

2. 网络层 (Internet Layer)

- 协议：**IP, ICMP, ARP (地址解析协议), RARP (逆向地址解析协议)**

3. 传输层 (Transport Layer)

- 协议：TCP, UDP

4. 应用层 (Application Layer)

- 协议：包含了 OSI 模型中的会话层、表示层和应用层的功能，例如 HTTP, FTP, SMTP, DNS 等。

关键差异

- **OSI 模型** 更加理论化，将网络通信过程细分为七个层次。
- **TCP/IP 模型** 是实际应用更广泛的模型，它简化为四个层次，更加注重实用性。

在实践中，大多数网络协议都是基于 TCP/IP 模型设计的，而 OSI 模型更多地用于教学目的和概念上的理解。然而，了解两者可以帮助更好地理解网络通信的工作原理。

以下列举相关的应用，便于记忆。如给出了DNS应用，SSH协议，python的urlopen抓取网页，C的socket编程等。

示例DNS应用

GitHub的 `raw.githubusercontent.com` 域名解析失败了 通过 `IPAddress.com` 首页,输入 `raw.githubusercontent.com` 查询到真实IP地址、



The screenshot shows the results of a DNS lookup for the domain `raw.githubusercontent.com`. The top section displays the domain name in large bold letters. Below it, a summary table is shown:

Hostname Summary	
Domain	<code>githubusercontent.com</code>
IP Address	199.232.68.133
Web Server Location	 United States

At the bottom of the page, there is a note indicating the data was updated on June 10, 2020, at 18:30 GMT, and reviewed on June 11, 2020, at 14:12 GMT. A URL `https://blog.csdn.net/51CTO博客` is also present.

修改hosts

Linux系统 在终端输入

```
1. sudo vi /etc/hosts
```

添加以下内容保存即可

```
1. 199.232.4.133 raw.githubusercontent.com
```

从上图来看，`raw.githubusercontent.com` 的域名解析出现了问题。这可能是由于 DNS 污染(DNS hijacking)导致的。DNS 污染是指某些网络服务提供商或恶意软件篡改了 DNS 解析结果，将用户引导到错误的 IP 地址。

DNS 污染通常发生在以下几种情况下：

1. ISP 篡改:一些互联网服务提供商(ISP)可能会故意篡改 DNS 解析结果，将用户重定向到广告页面或其他网站。
2. 恶意软件:某些恶意软件会修改本地的 hosts 文件或系统设置，以达到类似的目的。
3. 政府控制:在某些国家和地区，政府可能会实施 DNS 污染来阻止访问特定网站。

如何解决 DNS 污染

使用公共 DNS 服务器

你可以尝试使用公共 DNS 服务器，如 Google Public DNS (8.8.8.8和 8.8.4.4)或 Cloudflare DNS (1.1.1.1 和 1.0.0.1)，这些服务器通常不会被篡改。

更改 DNS 设置：可以通过系统偏好设置中的网络设置来更改 DNS 服务器地址。

修改本地 hosts 文件

如果你确定某个域名被污染，可以手动修改本地的 hosts 文件，将正确的IP 地址与域名绑定。

示例SSH协议

SSH (Secure Shell) 协议位于OSI七层模型的应用层。它主要用于在不安全的网络中为网络连接提供安全保障，特别是用于远程登录和执行命令。SSH不仅提供了强大的加密机制来保护数据的安全性和完整性，还支持身份验证，以确保通信双方的真实性。

尽管SSH主要用于终端仿真和命令行界面的交互，它同样可以用来传输文件（通过SFTP或SCP）或者隧道其他应用程序。由于它的功能直接服务于应用程序之间的交互，并且是用户可以直接调用的服务，因此被归类到应用层。

在TCP/IP模型中，SSH也是位于应用层。TCP/IP模型不像OSI模型那样严格区分表示层和会话层的功能，而是将这些功能都包含在了应用层中。所以，在这两个网络模型中，SSH都被认为是在应用层工作。

NIFE:

校内疑似中毒机器IP， 2024-10-15 17:12

<https://bbs.pku.edu.cn/v2/post-read.php?bid=668&threadid=18859011>

今天实验室服务器被115.27.246.28和115.27.246.31尝试爆破ssh，望排查

PKUCC (北京大学计算中心) :

您好，这是官方扫描器。会定期检测弱密码。

NIFE:

每次被官方扫描器扫的时候我们的服务器的ssh都会拒绝所有新建立的连接，导致所有人都登录不到服务器，这个策略应该在ubuntu系统的哪个地方设置呢？

JimmyChen:

直接在ufw里把这两ip屏蔽掉

wwwxxx:

防火墙加rule永封ip就可以了

centos: firewall-cmd --permanent --zone=public --add-rich-rule 'rule family="ipv4" source address="x.x.x.x" drop'

此外防爆破用fail2ban比较好

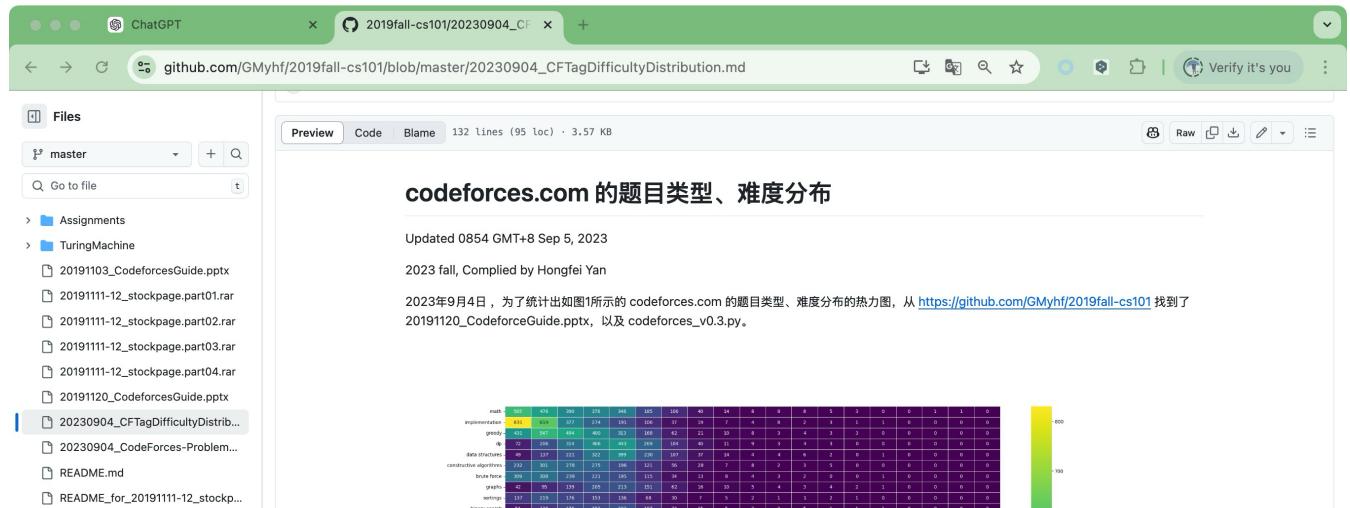
个人建议

TallestDwarf :

我用fail2ban

Python的urlopen

抓取Codeforces.com题目列表，进行简单的分类统计。



https://github.com/GMyhf/2019fall-cs101/blob/master/codeforces_v0.3.py

```

2019fall-cs101 / codeforces_v0.3.py
Code Blame 171 lines (152 loc) · 5.84 KB
11
12 import re
13 import urllib.request
14 from urllib.request import Request
15 from bs4 import BeautifulSoup
16 import os
17 import csv
18 import time
19
20 #%% retrieve the problem set
21 def spider(url):
22     req = Request(
23         url=url,
24         headers={'User-Agent': 'Mozilla/5.0'}
25     )
26     try:
27         response = urllib.request.urlopen(req)
28     except urllib.error.HTTPError as e:
29         print("Error:", e.code)
30     import sys;
31     sys.exit(0)
32
33 response = urllib.request.urlopen(url)
34 soup = BeautifulSoup(response.read(), 'html.parser')
35 pattern = {'name': 'tr'}
36 content = soup.findAll(**pattern)
37 #content = soup.findAll('tr')
38 for row in content:
39     item = row.findAll('td')
40     try:
41         # get the problem id
42         id = item[0].find('a').string.strip()
43         #print('====item[0]===%n %n', item[0])
44         col2 = item[1].findAll('a')
45         #print('====item[1]===%n %n', item[1])
46         # get the problem title
47         title = col2[0].string.strip()
48         # get the problem tags
49         tags = [foo.string.strip() for foo in col2[1:]]
50         # get the number of AC submissions
51         solved = re.findall(r'(\d+)', str(item[4].find('a')))[0]
52         #print('====item[4]===%n %n', item[4])
53         # update the problem info
54         codeforces[id] = {'title':title, 'tags':tags, 'solved':solved, 'accepted':0}
55         #print("id = (%s",format(id))
56         #print(codeforces[id])
57     except:
58         continue
59     return soup
60
61 codeforces = {}
62 wait = 15 # wait time to avoid the blocking of spider
63 last_page = 89 # the total page number of problem set page
64 url = ['http://codeforces.com/problemset/page/%d' % page for page in range(1,last_page+1)]
65 for foo in url:
66     print('Processing URL %s' % foo)

```

`urllib.request.urlopen` 是 Python 标准库 `urllib` 中的一个函数，用于打开 URL。它能够处理多种协议，具体取决于所提供的 URL 方案 (scheme)。以下是 `urllib.request.urlopen` 可能用到的主要网络协议：

- 1. HTTP (HyperText Transfer Protocol):** 这是最常用的协议之一，用于在 Web 浏览器和服务器之间传输网页。
 - `urlopen` 使用 HTTP 来获取文本、图片等资源。
- 2. HTTPS (HTTP Secure):** 这是 HTTP 的安全版本，通过 SSL/TLS 加密通道进行数据传输，提供更高的安全性。当需要安全地传输敏感信息时，如登录凭据或信用卡信息，通常会使用 HTTPS。
- 3. FTP (File Transfer Protocol):** 尽管不如以前那么常见，但 `urlopen` 仍然支持 FTP 协议，用于从 FTP 服务器下载文件。请注意，对于某些类型的 FTP 操作，你可能需要使用更专门的库，如 `ftplib`。
- 4. FILE:** 虽然不是真正的网络协议，但 `urlopen` 支持 file:// URLs 来访问本地文件系统中的文件。
- 5. DATA:** 允许直接嵌入小量的数据到 URL 中，例如 data: URI 方案可以用来内联表示图像或其他内容。
- 6. 其他:** 理论上，如果安装了适当的处理器 (handler)，`urllib.request` 可以扩展以支持更多的协议。然而，在标准库中，上述提到的是最常使用的几种。

当调用 `urllib.request.urlopen` 函数时，它会根据提供的 URL 自动选择合适的协议，并执行相应的操作来获取资源。例如，如果 URL 开头是 "http://" 或 "https://"，它将使用 HTTP 或 HTTPS 协议；如果是 "ftp://"，则会使用 FTP 协议。对于每种协议，`urllib.request` 内部都有对应的协议处理器来处理细节。

C的socket编程

tcp_echo_server.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 int main() {
11     int server_fd, new_socket;
12     struct sockaddr_in address;
13     int addrlen = sizeof(address);
14     char buffer[BUFFER_SIZE] = {0};
15     int opt = 1;
16
17     // 创建套接字
18     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
19         perror("socket failed");
20         exit(EXIT_FAILURE);
21     }
22
23     // 设置套接字选项
24     if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
25         perror("setsockopt");
26         close(server_fd);
27         exit(EXIT_FAILURE);
28     }
29
30     address.sin_family = AF_INET;
31     address.sin_addr.s_addr = INADDR_ANY;
32     address.sin_port = htons(PORT);
33
34     // 绑定套接字到指定的IP和端口
35     if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
36         perror("bind failed");
37         close(server_fd);
38         exit(EXIT_FAILURE);
39     }
```

```

40
41     // 开始监听
42     if (listen(server_fd, 3) < 0) {
43         perror("listen");
44         close(server_fd);
45         exit(EXIT_FAILURE);
46     }
47
48     printf("Server listening on port %d\n", PORT);
49
50     // 接受连接
51     if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
52     (socklen_t*)&addrlen)) < 0) {
53         perror("accept");
54         close(server_fd);
55         exit(EXIT_FAILURE);
56     }
57
58     // 读取客户端消息并回显
59     while (read(new_socket, buffer, BUFFER_SIZE) > 0) {
60         printf("Received: %s", buffer);
61         send(new_socket, buffer, strlen(buffer), 0);
62         memset(buffer, 0, BUFFER_SIZE);
63     }
64
65     // 关闭套接字
66     close(new_socket);
67     close(server_fd);
68
69     return 0;
70 }
```

tcp_echo_client.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <string.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 int main() {
11     struct sockaddr_in servaddr;
12     int sock = 0;
13     char buffer[BUFFER_SIZE] = {0};
14     const char *hello = "Hello from client";
15 }
```

```

16 // 创建套接字
17 if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
18     printf("\n Socket creation error \n");
19     return -1;
20 }
21
22 servaddr.sin_family = AF_INET;
23 servaddr.sin_port = htons(PORT);
24
25 // 将 IP 地址从字符串转换为二进制形式
26 if (inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr) <= 0) {
27     printf("\nInvalid address/ Address not supported \n");
28     return -1;
29 }
30
31 // 连接到服务器
32 if (connect(sock, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
33     printf("\nConnection Failed \n");
34     return -1;
35 }
36
37 // 发送消息给服务器
38 send(sock, hello, strlen(hello), 0);
39 printf("Hello message sent\n");
40
41 // 接收服务器回显的消息
42 read(sock, buffer, BUFFER_SIZE);
43 printf("Server echoed: %s\n", buffer);
44
45 // 关闭套接字
46 close(sock);
47
48 return 0;
49 }
50

```

这两个程序（TCP Echo Server 和 TCP Echo Client）主要用到了以下网络协议：

传输层协议

1. TCP (Transmission Control Protocol):

- 这两个程序使用了面向连接的、可靠的传输层协议 TCP。在代码中，当创建套接字时指定了 SOCK_STREAM 类型，这表明将使用 TCP 协议来确保数据包按序到达，并提供错误检查和流量控制机制。

```

1 if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
2     perror("socket failed");
3     exit(EXIT_FAILURE);
4 }

```

网络层协议

2. IP (Internet Protocol):

- IP 是互联网的核心协议，负责在网络间路由数据包。无论是在服务器端还是客户端，当你指定地址族为 AF_INET 时，你就是在使用 IPv4 地址格式进行通信。

```
1 | address.sin_family = AF_INET;
2 | address.sin_addr.s_addr = INADDR_ANY;
3 | address.sin_port = htons(PORT);
```

应用层协议

3. 自定义应用层协议:

- 在这个例子中，实际上并没有使用标准的应用层协议（如 HTTP、FTP 等）。相反，这是一个简单的回显（Echo）服务，它接收来自客户端的数据并将其原样返回给客户端。这种类型的交互可以被视为一种非常基础的应用层协议。

其他涉及的概念

- 端口：

- 端口号用于标识计算机上的具体服务或应用程序。在这个例子中，服务器监听的是 8080 端口，而客户端连接到同一端口。

- 套接字编程：

- 套接字是应用程序与网络协议之间的接口。通过调用操作系统提供的套接字 API，程序可以发送和接收数据。

综上所述，这两个程序主要使用了 TCP 和 IP 协议，在此基础上实现了简单的回显服务。它们并不依赖于特定的标准应用层协议，而是实现了一个自定义的应用层逻辑来处理消息的接收和发送。

这两个C程序（tcp_echo_server.c, tcp_echo_client.c）编译后，运行，可以互通信息。如下图示

The screenshot shows two terminal windows side-by-side. The left window, titled '(base)', shows the server's perspective. It outputs: 'Server listening on port 8080', 'Received: Hello from client', and then a prompt '(base) hfyang@Mac-mini network %'. The right window, also titled '(base)', shows the client's perspective. It outputs: 'Hello message sent', 'Server echoed: Hello from client', and then a prompt '(base) hfyang@Mac-mini network %'. This demonstrates a simple echo service where the client sends a message and the server returns it.

示例Telnet 和 FTP退出了

Telnet 和 FTP（文件传输协议）的传统实现确实是以明文形式传输用户名和密码，这意味着在网络上传输的数据没有加密。这种做法存在严重的安全风险，因为任何能够截获网络流量的人都可以轻易读取到这些敏感信息。

随着网络安全意识的提高和技术的发展，这两种协议已经被认为不再安全，并逐渐被淘汰或被更安全的替代品所取代：

1. **Telnet**: 已被SSH（Secure Shell）取代。SSH提供了加密的数据传输通道，不仅用于远程登录，还可以用于执行命令、文件传输等其他操作，同时保证了通信的安全性。

2. **FTP**: 虽然传统FTP仍然存在，但推荐使用更安全的替代方案，比如：
 - **SFTP (SSH File Transfer Protocol)**: 基于SSH协议，提供安全的文件传输。
 - **FTPS (FTP Secure)**: 这是在FTP基础上添加了SSL/TLS加密层的版本，可以在数据传输过程中保护信息不被窃听或篡改。
 - **SCP (Secure Copy Protocol)**: 也基于SSH，主要用于安全地复制文件。

尽管如此，在一些受限环境中，如某些内网环境或者老旧系统中，Telnet 和 FTP 可能仍然在使用。然而，在公共互联网上或对安全性有要求的场景下，它们几乎已经完全被更安全的协议所替代。对于新的项目或应用，强烈建议采用支持加密传输的现代协议以确保通信安全。

4.6.3 网络类别

是的，A类、B类、C类网络的地址范围可以用 **二进制** 或 **十六进制** 表示，这有助于更清晰地理解它们的结构和主机地址的分配情况。

1. A类网络 (Class A)

- 地址范围: 1.0.0.0 到 126.255.255.255
 - 二进制表示: 00000001.00000000.00000000.00000000 到 01111110.11111111.11111111.11111111
 - 十六进制表示: 01.00.00.00 到 7E.FF.FF.FF
- 默认子网掩码: 255.0.0.0 或 /8
 - 二进制掩码: 11111111.00000000.00000000.00000000
 - 十六进制掩码: FF.00.00.00
- 主机部分大小: A类地址的主机部分占 **24位** (32位减去8位的网络部分)。
 - 可分配的主机地址数:

$$1 \mid 2^{24} - 2 = 16,777,214$$

■ 去掉网络地址和广播地址，剩余可分配给主机的地址数为 **16,777,214** 个。

2. B类网络 (Class B)

- 地址范围: 128.0.0.0 到 191.255.255.255
 - 二进制表示: 10000000.00000000.00000000.00000000 到 10111111.11111111.11111111.11111111
 - 十六进制表示: 80.00.00.00 到 BF.FF.FF.FF
- 默认子网掩码: 255.255.0.0 或 /16
 - 二进制掩码: 11111111.11111111.00000000.00000000
 - 十六进制掩码: FF.FF.00.00
- 主机部分大小: B类地址的主机部分占 **16位** (32位减去16位的网络部分)。
 - 可分配的主机地址数:

$$1 | 2^{16} - 2 = 65,534$$

- 去掉网络地址和广播地址，剩余可分配给主机的地址数为 **65,534** 个。

3. C 类网络 (Class C)

- 地址范围: 192.0.0.0 到 223.255.255.255
 - 二进制表示: 11000000.00000000.00000000.00000000 到 11011111.11111111.11111111.11111111
 - 十六进制表示: C0.00.00.00 到 FF.FF.FF.FF
- 默认子网掩码: 255.255.255.0 或 /24
 - 二进制掩码: 11111111.11111111.11111111.00000000
 - 十六进制掩码: FF.FF.FF.00
- 主机部分大小: C 类地址的主机部分占 **8 位** (32 位减去 24 位的网络部分)。
 - 可分配的主机地址数:

$$1 | 2^8 - 2 = 254$$

- 去掉网络地址和广播地址，剩余可分配给主机的地址数为 **254** 个。

总结：

网络类别	地址范围 (十进制)	地址范围 (二进制)	默认子网掩码	可分配的主机数
A 类	1.0.0.0 到 126.255.255.255	00000001.00000000.00000000.00000000 到 01111110.11111111.11111111.11111111	255.0.0.0 或 /8	16,777,214
B 类	128.0.0.0 到 191.255.255.255	10000000.00000000.00000000.00000000 到 10111111.11111111.11111111.11111111	255.255.0.0 或 /16	65,534
C 类	192.0.0.0 到 223.255.255.255	11000000.00000000.00000000.00000000 到 11011111.11111111.11111111.11111111	255.255.255.0 或 /24	254

注意：

- 每个网络类别的地址范围是基于 IP 地址的 **首字节** 来区分的。
- 主机可分配的地址数总是减去 2 个，因为网络地址和广播地址不能分配给主机。

网络中 01111111.11111111.11111111.11111111 不分配？

在网络中，二进制表示的 IP 地址 01111111.11111111.11111111.11111111 转换成十进制是 127.255.255.255。这个地址属于 127.0.0.0/8 地址段，这是专门保留用于环回 (loopback) 功能的地

址段。

环回地址主要用于网络软件测试以及本地机进程间通信，并不分配给任何物理网络接口，也不会路由到远程网络。所有发往环回地址的数据包都会被操作系统截获，并在内部循环回来，如同发送到了“自己”一样。

因此，`127.255.255.255` 不会分配给任何实际的网络设备或在网络上进行传输。实际上，整个`127.0.0.0/8`网段都是如此，不过最常用的环回地址是`127.0.0.1`。

每类网络去掉网络地址和广播地址？

在IP地址分类中，每类网络（A、B、C、D、E）都有特定的范围，并且每个网络内部有两个特殊的地址不分配给主机：一个是网络地址（用于标识该网络本身），另一个是广播地址（用于向该网络中的所有主机发送信息）。下面是对于IPv4地址中A、B、C三类网络去掉网络地址和广播地址后可用主机数量的简要说明：

- **A类网络：**

- 地址范围：0.0.0.0 到 127.255.255.255
- 网络地址：最左边的一位是0，因此网络地址是该范围内最小的地址。
- 广播地址：该网段内的最大地址。
- 可用主机数： $2^{24} - 2 = 16,777,214$ 个主机

- **B类网络：**

- 地址范围：128.0.0.0 到 191.255.255.255
- 网络地址：前两位是10，网络地址是该范围内最小的地址。
- 广播地址：该网段内的最大地址。
- 可用主机数： $2^{16} - 2 = 65,534$ 个主机

- **C类网络：**

- 地址范围：192.0.0.0 到 223.255.255.255
- 网络地址：前三位是110，网络地址是该范围内最小的地址。
- 广播地址：该网段内的最大地址。
- 可用主机数： $2^8 - 2 = 254$ 个主机

请注意，上述计算基于传统的分类网络。然而，在现代网络中，可变长子网掩码（VLSM）和无类别域间路由（CIDR）被广泛使用，这使得网络划分更加灵活，不再严格遵循传统分类。因此，实际可用的主机数量可能会因具体的子网掩码而有所不同。

另外，D类和E类地址不用于常规的主机通信，D类地址用于多播，E类地址保留供将来或实验使用。

A类首位为0，对应1到126；

B类前两位10，对应128到191；

C类前三位110，对应192到223；

D类前四位1110，对应224到239；

E类前5位为11110，对应240到255

“可分配的地址”，指的是同一个网段下的可分配主机数量？

比如A类掩码`255.0.0.0`，那某个网段下就可以有 $2^{24}-2$ 个主机，这是因为后24位全0和全1都不可分配，全1用来广播，全0应该直接对应这个网段

B类可用网络数是 $2^{14}=16384$ ， 可用主机数是 $2^{16}-2=65534$

128.0到191.255共16384个B类网段

4.7 只读方式打开一个utf-8 编码文件

这个程序涉及到了按照绝对路径，或者相对路径打开一个utf-8编码的文件。

我是mac系统，斜杠与window下面是相反的。双斜杠是因为有一个是转义作用。

我的程序，在下面三种路径中，都可以正常显示。

在program/data目录中

```
1 | echo "北京大学PKU" > math.csv
```

在program目录

```
1 import csv
2
3 #file_path = "data/math.csv"
4 file_path = "data//math.csv"
5 #file_path = '/Users/hfyan/Code/5.1/program/data/math.csv'
6 try:
7     fo = open(file_path, 'r', encoding='utf-8')
8     reader = csv.reader(fo)
9     for row in reader:
10         print(row)
11
12 except FileNotFoundError:
13     print(f"Error: The file at {file_path} was not found.")
14 except Exception as e:
15     print(f"An error occurred: {e}")
```

运行python read_file.py

```
1 | ['北京大学PKU']
```

<https://www.runoob.com/python3/python3-file-methods.html>

20221227笔试（Python）

一、填空题（每空1分，共15分）

- 以下设备中，属于输入设备的有 (A、B、D、F) ，属于输出设备的有 (C、D、E) 。
 - A、键盘
 - B、鼠标
 - C、打印机
 - D、智能手机触控显示屏
 - E、喇叭
 - F、摄像头
- 计算机软件系统中最重要的软件是 (操作系统或 OS) ，主要负责资源管理、任务调度等。
- 主存 (内存) 管理是以8个二进制位 (一个Byte) 作为一个管理单元的，并且每个单元都编有一个唯一的地址，内存的访问是通过其地址进行的。我们称主存(内存)是随机存储器，这是因为： (CPU可以直接通过地址访问任意单元，无需按顺序扫描。)

主存 (内存) 被称为随机存储器 (Random Access Memory, RAM) 的原因是内存中的每个存储单元都可以通过其地址直接访问，访问时间与存储单元的位置无关。这意味着，无论访问的是哪一个存储单元，访问速度基本上是相同的，而无需像顺序存储设备 (如磁带) 那样必须按顺序查找。

以下是主要原因的总结：

- 随机访问：**

内存中的每个存储单元都有唯一的地址，CPU可以直接通过地址访问任意单元，无需按顺序扫描。
- 访问时间一致：**

无论访问的是首地址还是尾地址，内存的访问时间在理论上是固定的，不依赖于物理位置。
- 对比顺序存储设备：**

顺序存储设备 (如硬盘、磁带) 在访问数据时需要按顺序查找，而内存无需这样，因此被称为“随机存储器”。

这种随机访问特性是内存速度快的关键，也是它区别于其他存储介质的重要特征。

原答案是这句“每个地址的访问时间与地址无关（或访问每个地址的时间相同等意思）”，感觉有点别扭。

- 如果一个数据项正在被访问，那么在近期它很可能还会被再次访问，这说的是程序执行时，CPU访问数据的 (时间局部性 (或局部性)) 。
- 要反转二进制数01001001的高4位，保留低4位不变，应将它与11110000进行 (异或) 运算。
- 在ASCII编码表中，数字字符和大小写字母都是按顺序编码的，大写字母A的二进制ASCII码为01000001，大写字母D的二进制ASCII码为 (01000100) ；对于一个保存了小写字母的字符串变量x，请给出其对应大写字母的表达式 (x.upper()或者 char(ord(x)+ord('A')-ord('a'))) 。
- 在Python语言中，表达式7/2.0的类型和值分别是 (float) 和 (3.5) 。
- 已知x=200，那么表达式1<x<100的值是 (False) 。
- 在Python语言中，字符串 s="你好，@_pku" 的长度是 (8) ，其中"_"表示一个英文空格；对s的切片操作 s[1:-1]结果是 ("好你") 。

在Python中，字符串切片的基本语法是 `string[start:stop:step]`，其中：

- `start` 表示开始索引，默认为0 (即从字符串的第一个字符开始) 。
- `stop` 表示结束索引 (不包含)，默认为空 (即直到字符串的末尾) 。
- `step` 表示步长，默认为1 (即逐个字符取) 。如果步长为负数，则表示逆序取字符。

对于给定的操作 `s[1:-1]`：

- `start` 是 1，意味着我们从索引1处开始，也就是从第二个字符开始。
- `stop` 是空，意味着我们要一直走到字符串的开头，但因为步长是负数，所以实际上是在向字符串的前面走，直到索引0处。
- `step` 是 -1，这意味着以逆序的方式取字符。

请注意，中文字符和英文字符一样，在Python中每个都被视为单个字符，所以在进行切片操作时，它们的处理方式是一致的。

2022年计概（B）C语言版，填空第9题

在C语言中，字符串 `s="你好,@_pku"` 的长度是 11，其中"_"表示一个英文空格；在 32 位机器上，一个整型指针变量(如 `int *p`)占用 4个字节的内存空间，一个双精度浮点型指针变量(如 `double *g`)占用 4 个字节的内存空间。

这个题目有bug，没有说明汉字编码，如果是GBK编码，可以如下：

在Mac上复现

```
1 | echo "你好,@_pku" > input.txt
```

使用 `iconv` 将文件的编码从 UTF-8 转换为 GBK

```
1 | iconv -f UTF-8 -t GBK input.txt > input_gbk.txt
```

编译C程序 `gcc str_len.c`

：不要忘记末尾'\0'。

```
1 | #include <stdio.h>
2 | #include <string.h>
3 |
4 | char str[100];
5 | int main() {
6 |     //memset(str, 0, 100);
7 |     scanf("%s", str);
8 |     int len = strlen(str);
9 |     printf("%d\n", len + 1);
10 |    return 0;
11 | }
```

```
1 | ./a.out < input_gbk.txt
2 | 11
```

```
1 i, j = 0, 1
2 while i < 100:
3     print(i)
4     i += 1
5     j += 1
6     i += j
```

输出的最后一个数字是 (88) , 其中输出语句执行了 (12) 次。

由于 `i` 在循环体内两次被增加 (一次是 `i += 1`, 另一次是 `i += j`) , 这意味着 `i` 每次循环实际上增加了 `1 + j`。随着 `j` 的递增, `i` 的增长速度会越来越快, 最终超过或等于100, 从而终止循环。

```
1 i, j = 0, 1
2 count = 0 # 记录print语句执行次数
3
4 while i < 100:
5     count += 1
6     print(i) # 假设这里只用于计数, 实际运行时可注释掉以避免大量输出
7     i += 1
8     j += 1
9     i += j
10
11 print("最后一次打印的i值为:", i - j - 1) # 因为最后一次增加使i>=100, 所以我们需要减去这次增加
12 print("print语句执行次数:", count)
```

```
0
3
7
12
18
25
33
42
52
63
75
88
最后一次打印的i值为: 88
print语句执行次数: 12
```

二、单项选择题 (每题1分, 共15分)

根据图片中的内容, 以下是提取的文字:

1. 一个 CPU 能够在 8 个时钟周期/时钟节拍内完成一个指令周期。如果这个 CPU 的时钟频率/主频是 4GHz, 那么这个 CPU 的运算速度是 _ MIPS。

- A) 5 B) 50 C) 500 D) 5000

为了计算CPU的运算速度（以每秒百万条指令，即MIPS为单位），我们可以使用以下公式：

$$\text{MIPS} = \frac{\text{时钟频率}}{\text{每个指令周期的时钟周期数}}$$

给定条件是：

- 时钟频率（主频）是4GHz，也就是4,000,000,000 Hz。
- 每个指令周期需要8个时钟周期。

将这些值代入上述公式中：

$$\text{MIPS} = \frac{4,000,000,000}{8} = 500,000,000$$

然后，我们需要将结果转换成MIPS（每秒百万条指令）。由于1 MIPS等于1,000,000条指令/秒，因此：

$$\text{MIPS} = \frac{500,000,000}{1,000,000} = 500$$

2. 下面哪个人物不是计算机发展史上的重要人物：

- A) 阿兰·图灵 A.M Turing B) 冯·诺依曼 John Von Neumann
C) 史提芬·库克 Stephen A.Cook D) 冯·布劳恩 Wernher Von Braun

3. 请按照访问一次所需时间由短到长（即速度由快到慢）对下列存储硬件进行排序：

- ①高速缓存 ②主存储器 ③寄存器 ④外部存储设备
A) ①②③④ B) ①③②④ C) ③①②④ D) ②①③④

4. 下面哪个选项不是 Python 语言中合法的标识符：

- A) 123abc B) abc123 C) abc D) abc_123

5. 使用 2 个字节表示整数，则表示的范围可能是：

- A) -2 ~ 1 B) -128 ~ 127 C) -32768 ~ 32767 D) -65536 ~ 65537

6. Python 语言的整数类型中，逻辑“真”等价于：

- A) 大于零的整数 B) 小于零的整数 C) 非零的整数 D) 等于 0 的整数

7. 上网时，经常需要使用验证码，下面有关其功能的描述，正确的是：

- A) 验证码与用户名和密码密切相关，因此需要牢记，并保证每次都输入相同的验证码。
B) 验证码通常都是英文字母和数字的变形，其目的是让旁边其他人不容易看清楚。
C) 验证码主要是让计算机程序（机器人）难以自动识别，防止自动登录或破解账户。
D) 验证码主要是减少用户登录次数，防止用户沉迷网络。

8. 以下和收发邮件最相关的计算机网络协议为：

- A) POP/SMTP B) UDP C) VoIP D) TCP/IP

9. 在 CPU 的内部结构中，负责处理紧急情况的部件是：

- A) 算术逻辑运算器 B) 寄存器 C) 中断处理器 D) 程序控制器

10. 以下说法中，不正确的是：

- A) 采用位图图像表示方法在计算机中表示图像，图像放大时，通常会产生锯齿、颗粒状等失真现象。

B) 计算机中存储的电影资料是以二进制方式存储的。

C) 计算机内用二进制来表示数据，任何一个十进数，都可以转化为完全相等的对应的二进制数。

D) 在计算机中表示图像的矢量图像方法中，图像分解为几何图形的组合。

11. 如果计算机断电，那么下列设备中，哪里的数据将会丢失：

A) 硬盘 B) 内存 C) 光盘 D) 磁带

12. 在 Python 语言中，已知字符“2”的 ASCII 编码为 50，执行以下语句后，输出是：

```
1 | c = "0"
2 | c = c + 10
3 | print(c)
```

A) 010 B) 58 C) 10 D) 类型错误

13. 关于循环结构，以下选项中描述错误的是：

A) 每个 `continue` 语句有能力跳出当前层次的循环。

B) `break` 可以用来跳出当前层次 `for` 或者 `while` 循环，脱离该循环后程序从循环代码后继续执行。

C) 通过 `for`、`while` 等关键字描述循环结构。

D) `while` 循环和 `for` 循环可以互相转换。

14. 以下关于计算机系统的说法正确的是：

A) 现代计算机通常为冯·诺伊曼结构，即由中央处理器、控制器、存储器、输入设备和输出设备五大功能模块构成。

B) 主板是计算机主机箱中的主要部件，计算机的其他硬件设备通过各种接口与主板相连并发挥作用。

C) 现代计算机中的许多存储器都是易失的，即断电后数据会丢失，只有磁性介质构成的存储器才能保证断电后数据依然存在。

D) 不同存储器的速度存在较大不同，其中内存与外存（硬盘）的速度常有几个数量级的差距。为了提高运行速度，在内存和外存之间通常设置一个缓冲性的部件，被称为高速缓存。

在这四个选项中，最准确的描述是：

A) 的描述不完全正确，控制器实际上是中央处理器（CPU）的一部分，不应该被单独列出作为与CPU并列的功能模块。

C) 虽然许多现代计算机中的主存储器（例如RAM）确实是易失性的，即断电后数据会丢失，但并不是只有磁性介质构成的存储器才能保证断电后数据依然存在。实际上，现在有许多非易失性存储技术，比如固态硬盘（SSD），它使用闪存技术，即使在断电情况下也能保存数据。此外，还有ROM（只读存储器）、EPROM、EEPROM等类型的存储器，在断电后同样可以保持数据。

D) 的描述基本正确，不同存储器的速度确实存在较大差异，并且为了提高运行速度，在内存和外存之间设置高速缓存（Cache）是很常见的做法。然而，高速缓存通常是指位于CPU内部或紧邻CPU的一种快速存储器，用来临时存储频繁访问的数据副本，以减少对较慢的主存访问次数。因此，尽管D选项描述的现象是真实的，但“缓冲性的部件”这一表述可能不够精确，容易让人误解为一种更通用的缓冲机制而非特指的高速缓存。

A) 的描述不够准确，**中央处理器（CPU）**：这是执行程序指令的核心部件。它包含了算术逻辑单元（ALU），用于进行算术和逻辑运算；以及控制单元（CU），负责从内存中取出指令、解码并执行它们。

“中央处理器”一词已经涵盖了执行单元（即算术逻辑单元）和控制单元，所以再次提及“控制器”会导致重复且混淆概念。

冯·诺伊曼结构计算机的5个主要部件分别是：

1. **运算器（Arithmetic and Logic Unit, ALU）**

- 负责执行计算和逻辑运算，比如加法、减法、逻辑与、或等操作。

2. **控制器（Control Unit, CU）**

- 负责解释和执行指令，协调计算机其他部件的工作。它从内存中读取指令并将其转换为具体操作。

3. **存储器（Memory）**

- 用于存储程序、数据以及中间结果，通常分为主存储器（如RAM）和辅助存储器（如硬盘）。

4. **输入设备（Input Devices）**

- 用于向计算机输入数据和指令，例如键盘、鼠标、扫描仪等。

5. **输出设备（Output Devices）**

- 用于将计算结果和信息输出给用户，例如显示器、打印机、扬声器等。

这五大部件通过**总线（Bus）**相互连接，构成一个完整的计算机系统。冯·诺伊曼结构的核心思想是存储程序，即将指令和数据以相同的形式存储在内存中，并依次执行。

15. 以下关于 IPv6 的说法不正确的是：

A) IPv6 是下一代的 IP 协议，用于代替 IPv4。

B) IPv6 地址由 64 位二进制数表示，通常每 8 位一段写成 16 进制数表示以便于使用。

C) 域名服务器（缩写为 DNS）的可以将一个合法域名（如 www.pku.edu.cn）转换成对应主机的 IPv6 地址。

D) IPv6 的地址资源非常充足，可以为现在地球上的所有计算机、手机甚至家用电器分配独立的地址。

IPv6 地址实际上是由 128 位二进制数构成的，而不是 64 位。这大大增加了可用地址的数量，解决了 IPv4 地址枯竭的问题。

三、计算题（共20分）

1. 数制转换运算 (4分, 前2空1分, 第3空2分)

$$(2022)_{10} = (11111100110)_2 = (3746)_8$$

$$(42.625)_{10} = (101010.101)_2$$

2.二进制算术运算 (4分)

$$10001 * 1010 = (10101010)$$

$$10100101 / 1111 = (1011)$$

3.二进制逻辑运算 (4分)

与: &, 或: |, 非: ~, 异或: ^; >>为二进制数右移运算符, 右侧整数为左侧二进制数要右移的位数, 移出去的位丢弃, 左侧则补0)

$$10110110 \& (00110001 >> 1) = (00010000 或 10000)$$

$$11101101 \wedge 00110101 = (11011000)$$

4.ASCII编码

又称美国信息交换标准码, 是国际上使用最广泛的字符编码。在计算机中, 每个字符的ASCII码用一个字节存储。已知ASCII码中A的编码为01000001, A - Z在码表中是连续的。现在计算机里有三段编码10101111, 10110100, 10101010, 将它们取反求对应的英文字母串 (2分)

10101111 取反 -> 01010000 -> 十进制80, 对应P

10110100 取反 -> 01001011 -> 十进制75, 对应K

10101010 取反 -> 01010101 -> 十进制85, 对应U

5.图像编码

采用不同分辨率、不同颜色编码的图像, 其图像质量差别非常大。对于同样一幅原始图像, 分辨率越高, 则图像越精细, 质量越好, 当然所需要的存储空间也是非常大的。一幅分辨率为 4096×2048 的真彩色(24位, 3Bytes)图像, 如果不做压缩, 其所需的存储空间为? (结果单位采用MB) (2分)

$$4096 \times 2048 \times 24 / (8 \times 1024 \times 1024) = 24\text{MB}$$

6.编码计算

假设视频的帧率为30FPS (即每秒包含30帧图像)。对于8K的电影而言, 每帧图像包含 7680×4320 个像素, 其中每个像素包含RGB三种颜色, 每种颜色采用1个字节进行记录。 (计算中 7680×4320 可近似为33,000,000, 以下计算中 $1\text{GB} = 1000\text{MB}$, 其余进制同理。请写出计算过程) (4分, 每小题2分)

(1) 一块1000GB的移动硬盘能存储多少秒这样的8K视频? (保留整数即可)

$$33,000,000 \times 3 \times 1 \text{ Byte} \times 30 = 2,970,000,000 \text{ 字节/秒}$$

$$1000 \times 1000 \times 1000 / 2,970,000,000 \approx 337 \text{ 秒}$$

(2) 在实际应用中，视频在存储与传输中会经过编码压缩。如果经过编码后，上述1000GB的移动硬盘能够存储56小时的8K视频，请问所采用的视频编码方法的压缩率为多少？(压缩率为x:1表示每x个字节的信息被压缩为1个字节，x保留到整十即可)

计算压缩后的数据量：

- 压缩后能存储56小时的视频
- $56\text{小时} = 56 \times 3600 = 201,600\text{秒}$
- 压缩后的每秒数据量： $1,000,000,000,000 / 201,600 \approx 4,960,000\text{字节}$

计算压缩率：

- 原始每秒数据量： $2,970,000,000\text{字节}$
- 压缩后每秒数据量： $4,960,000\text{字节}$
- 压缩率： $2,970,000,000 / 4,960,000 \approx 600$

四、编码应用题（共12分）

1. (6分) X教授开设的《计算概论》共有183名同学选修。X教授认为用学校分配的10位十进制学号来标识班上的同学过于冗余。他决定用8位二进制数给同学们重新分配编号，并保证每位同学被分到的编号都是唯一的。

(1) (2分) 某位同学的编号为10011101，对应的十六进制表示为0x9D。被分配的编号是从00000000开始的连续二进制数，那么该班级同学中最大的编号为10110110（用二进制表示）。

(2) (2分) 由于X教授开的课越来越受欢迎，选课人数越来越多，X教授不得不扩展他的二进制数表示方法来容纳更多同学选修。原有的8位二进制数最多能表示256位同学的编号，为了表示666位同学的编号，至少需要把二进制数扩展到10位。

(3) (2分) 期末考试时，X教授希望把同学们分到4个考场进行考试，他希望根据编号的二进制表示的第3位和第2位来给同学们区分考场，如果第183位同学需要在3号考场考试，第3位同学的考场需要在考场1考试。那么第64位同学需要在3号考场考试，第100位同学需要在1号考场考试。

$$128 + 32 + 16 + 4 + 2 + 1$$

因为被分配的编号是从00000000开始，最大编号是182，对应二进制是10110110

3 二进制是00000010

根据答案判断，题面的第3位和第2位是从右数过来。

已知

$$11 \rightarrow 3$$

$$01 \rightarrow 1$$

所以

64 二进制是00111111 -> 3

100 二进制是01100011 -> 1

2. (6分) 在互联网中，主机都配有IP地址；主机之间通过发送数据分组来传输数据。

(1) (1分) IP地址为32位二进制无符号整数，可以用“点分十进制”的字符串来表示，例如IP地址00000000 00000000 00000000 00000010（二进制表示中，前缀0要保留；为了表示清晰，我们将每8个位一组用空格分隔开）的点分十进制为字符串“0.0.0.2”。IP地址11000000 00110101 00111000 00000111的点分十进制为192.53.56.7。

(2) (2分) 通常，一个互联网IP地址的前半段为网络地址，后半段为主机地址。现有一类特殊的IP地址，称为T类地址，其前20位为网络地址，后12位为主机地址。一个T类地址段（指网络地址相同的所有IP地址）中有一个特殊IP地址，叫做“广播地址”（通过它可以发送消息给所有主机），其编址方法是网络地址不变，其余表示主机地址的位全部设置为1。请给出T类地址220.139.160.0所在的T类地址段的广播地址 220.139.175.255。

要找到T类地址220.139.160.0所在的地址段的广播地址，我们需要先理解给定的信息。根据题目描述，T类地址的前20位是网络地址，后12位是主机地址。对于广播地址来说，其主机部分的所有位都设置为1。

首先，将给定的IP地址220.139.160.0转换为二进制形式：

$$220 = 11011100$$

$$139 = 10001011$$

$$160 = 10100000$$

$$0 = 00000000$$

因此，该IP地址的二进制表示为：11011100.10001011.10100000.00000000

根据题目，前20位是网络地址，所以我们将保留这20位不变，然后将剩下的12位全部置为1来得到广播地址。该IP地址的前20位是“11011100.10001011.1010”，所以我们需要保持这部分不变，然后将后面的12位置为1：

11011100.10001011.10101111.11111111

现在我们将其转换回十进制：

$$11011100 = 220$$

$$10001011 = 139$$

$$10101111 = 175$$

$$11111111 = 255$$

所以，T类地址220.139.160.0所在的地址段的广播地址是220.139.175.255。

(3) (2分) 网络中的数据分组使用一个三元组<源地址，目的地址，数据>来表示。路由器在网络中负责转发数据分组，为此，每个路由器都配有一个转发规则表，表中有多条规则，每条规则也是一个三元组<地址，掩码，下一跳>。路由器的转发行为如下：1、接收数据分组，2、在规则表中逐条匹配，发现第一条成功匹配的规则时，转发到该规则中描述的下一跳。

数据分组和一条规则的匹配运算如下：如果 数据分组的目的地址 & 规则中的掩码 == 规则中的地址，则匹配成功；否则，匹配失败。（& 是二进制数按位与运算符）

假设某台路由器的转发规则表如下，数据分组A <135.46.57.14, 135.46.63.10, data>的下一跳是接口2；数据分组B <192.53.40.7, 135.46.52.2, dataB> 的下一跳是路由器2。

规则序号	地址	掩码	下一跳
1	135.46.56.0	255.255.252.0	接口1
2	135.46.60.0	255.255.252.0	接口2
3	192.53.40.0	255.255.254.0	路由器1
4	0.0.0.0	0.0.0.0	路由器2

为了确定数据分组A和B的下一跳，需要根据给定的转发规则表，使用目的地址与每条规则中的掩码进行按位与运算（&），然后比较结果是否等于规则中的地址。如果相等，则匹配成功，且该规则定义了数据分组的下一跳。

数据分组A

数据分组A的目的地址是135.46.63.10，我们用它来尝试匹配规则：

规则1

- 目的地址 & 掩码 = $135.46.63.10 \& 255.255.252.0 = 135.46.60.0$
- 地址 = 135.46.56.0 不匹配，因为 $135.46.60.0 \neq 135.46.56.0$

规则2

- 目的地址 & 掩码 = $135.46.63.10 \& 255.255.252.0 = 135.46.60.0$
- 地址 = 135.46.60.0
- 匹配成功，因此**数据分组A的下一跳是接口2**。

既然已经找到了第一条匹配的规则，我们可以停止搜索。

数据分组B

数据分组B的目的地址是135.46.52.2，我们用它来尝试匹配规则：

规则1

- 目的地址 & 掩码 = $135.46.52.2 \& 255.255.252.0 = 135.46.52.0$
- 地址 = 135.46.56.0 不匹配，因为 $135.46.52.0 \neq 135.46.56.0$

规则2

- 目的地址 & 掩码 = $135.46.52.2 \& 255.255.252.0 = 135.46.52.0$
- 地址 = 135.46.60.0 不匹配，因为 $135.46.52.0 \neq 135.46.60.0$

规则3

- 目的地址 & 掩码 = $135.46.52.2 \& 255.255.254.0 = 135.46.52.0$
- 地址 = 192.53.40.0 不匹配，因为 $135.46.52.0 \neq 192.53.40.0$

规则4

- 这是一个默认路由，因为它使用的是0.0.0.0/0的掩码，它可以匹配任何地址。

- 因此，数据分组B的下一跳是路由器2。

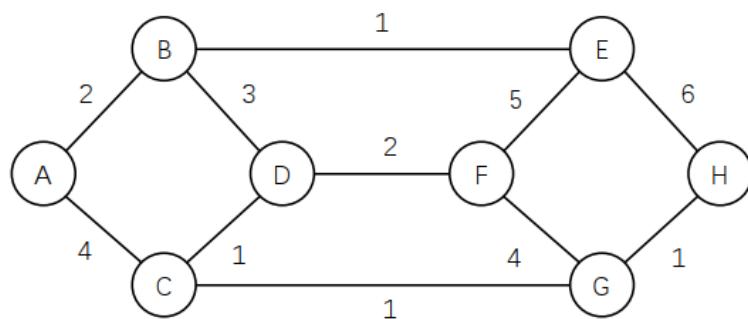
综上所述：

- 数据分组A <135.46.57.14, 135.46.63.10, data> 的下一跳是接口2。
- 数据分组B <192.53.40.7, 135.46.52.2, dataB> 的下一跳是路由器2。

(4) (1分) 最短路径算法是计算机网络里一个常用的路由算法，该算法可以找出网络中从一个节点到另一个节点的最短路径。假设有一个网络，其拓扑如下距离描述，图中一共有8个节点，为节点A到节点H，相邻节点间的距离标注在边上，如节点A到节点B的距离为2。现在，假如从节点A出发，要到达节点H，请给出一条最短路径。

图中的节点和边的距离如下：

A-B: 2, A-C: 4, B-D: 3, B-E: 1, C-D: 1, C-G: 1, D-F: 2, E-F: 5, E-H: 6, G-H: 1



A-C-G-H

为了找到从节点A到节点H的最短路径，我们可以使用Dijkstra算法，这是一个经典的最短路径算法。它适用于边权重非负的情况，能够有效地计算出单源最短路径问题，即从一个给定的起点到所有其他节点的最短路径。

根据你提供的网络拓扑信息，我们来构建这个图并找出最短路径：

1. A-B: 2
2. A-C: 4
3. B-D: 3
4. B-E: 1
5. C-D: 1
6. C-G: 1
7. D-F: 2
8. E-F: 5
9. E-H: 6
10. G-H: 1

现在我们将使用Dijkstra算法来确定从A到H的最短路径。不过，这里我们可以直观地分析一下可能的路径和它们的距离：

- 通过B可以到达E然后到H: A->B->E->H = 2 + 1 + 6 = 9
- 通过C可以到达G然后到H: A->C->G->H = 4 + 1 + 1 = 6
- 通过C到达D再到F这条路不会是最短路，因为没有直接从F到H的路径，并且绕行会增加距离。
- 通过B到达D这条路也不会比直接从C到G更短，所以也不考虑。

从上述直观分析中，我们可以看出最短路径可能是A->C->G->H，总长度为6。这是目前看来从A到H的最短路径。

如果我们严格按照Dijkstra算法进行计算，我们也会得到相同的结果。因此，从节点A出发到达节点H的一条最短路径是 A -> C -> G -> H，路径的总长度是6。

五、程序阅读题（共12分）

阅读下面的程序，说明该程序实现的功能是什么；并对给定的输入，写出程序执行后的输出。

第1题

```
1 # s 大于等于 100，小于 1000。e 大于等于 100，小于 1000，e 大于 s。
2 s, e = [int(x) for x in input().split()]
3 sum = 0
4 for i in range(s, e + 1):
5     a = 0
6     b = i
7     while b:
8         a = a * 10 + b % 10
9         b //= 10
10    if a == i:
11        sum += 1
12 print(sum)
```

程序功能(2分)：

输出大于等于 s，小于等于 e 的回文数的个数。回文数指倒着读和正着读都一样的数。

当输入为：100 130 时，程序输出为(2分)：

3. 在这个范围内，回文数有101、111、121。

第2题

```
1 # 读入两个整数k，正整数n，n是下面待输入数据的数量（小于 100）
```

```

2 k, n = map(int, input().split())
3
4 # 读入n个整数，放在数组nums之中
5 nums = [int(i) for i in input().split()]
6
7 # 初始化三个整数变量left, sum和minLength
8 left = 0
9 sum = 0
10 minLength = n + 1
11
12 # 算法主体部分
13 for right in range(n):
14     sum += nums[right]
15     while left <= right and sum >= k:
16         if minLength > right - left + 1:
17             minLength = right - left + 1
18         sum -= nums[left]
19         left += 1
20
21 # 根据minLength值的大小情况分别进行输出
22 if minLength == n + 1:
23     print(0)
24 else:
25     print(minLength)

```

程序功能（2分）：

求给定数组中和大于等于 k 的子数组的最短长度(如果没有则输出 0)。子数组指的是数组中占据连续位置的一个或多个整数组成的数组。

当输入的数为：

1	7 4
2	5 1 4 3

程序的输出为（2分）：2

输入 7 4 表示 $k = 7$ 和数组长度 $n = 4$ 。

输入 5 1 4 3 是数组的具体值。

程序会找到和大于等于 7 的最短子数组，这里是 [4, 3]，长度为 2。

第3题

```

1 top = -1
2
3 def push(a, elem):
4     global top
5     top += 1
6     a[top] = elem

```

```

7
8 def pop(a):
9     global top
10    if top == -1:
11        return
12    top -= 1
13
14 def visit(a):
15    if top != -1:
16        return a[top]
17    else:
18        return ' '
19
20 if __name__ == '__main__':
21     a = [0]*100
22     s = input()
23     length = len(s)
24
25     for i in range(length):
26         if s[i] == '(':
27             push(a, s[i])
28         else:
29             if s[i] == ')':
30                 if visit(a) == '(':
31                     pop(a)
32                 else:
33                     print("False at %d" % i)
34                     exit(0)
35
36     if top == -1:
37         print("True")
38     else:
39         print("False at %d" % i)

```

程序功能（2分）：

检查一个字符串内的小括号是否匹配，如果不匹配，输出不匹配的位置

定义栈的相关操作：

- `push(a, elem)`：将元素 `elem` 压入栈 `a`。
- `pop(a)`：从栈 `a` 中弹出一个元素。
- `visit(a)`：访问栈顶元素，如果栈为空则返回一个空格。

当输入为：(1+2)*(3/(5+6) 时，输出为（2 分）：

False at 5。

六、程序填空（2题，每空1分，共10分）

1、如果一个正整数从高位开始，奇数位为奇数，偶数位为偶数，则称该数为一个特殊的数，如 5, 14, 121, 1234 等。请完成下列函数，其功能为判断 n 是否是特殊的数，是：返回 1，否：返回 0。

```
1 def isSpecial(n):
2     flag = cnt = length = 0
3     b = n
4     while n > 0: # [1]
5         n //= 10 # [2]
6         length += 1
7
8     if length % 2 == 0:
9         flag = 1
10
11    for i in range(length):
12        temp = b % 10
13        if flag and temp % 2 == 1:
14            return 0
15        if not flag and temp % 2 == 0: # [3]
16            return 0
17        b //= 10 # [4]
18        flag = not flag # [5]
19
20    return 1
```

17,18 行次序可换

2、下面程序功能：在主串 s 中检索子串 p，打印输出各子串的第一个字符在主串中的下标。例如，主串 s="abc123abc123"，子串 p="123"，输出：3 9。

需要说明的是：主串中每个子串的字符不重叠。例如，s = "abc111111"，p = "111"，则输出：3 6，而不是 3 4 5 6。主串 s 的长度不超 1024，子串长度不超 32。

```
1 # 函数 match 的功能：返回子串 p 的第一个字符在主串 s 中的下标，如果 p 不是 s 的子串，则返回 -1
2 def match(s, p):
3     n1 = len(s)
4     n2 = len(p)
5     i = 0
6     while i <= n1 - n2:
7         j = 0
8         while j < n2 and s[i+j] == p[j]: # [1]
9             j += 1
10        if j == n2: # [2]
11            return i
12        i += 1
```

```

13     return -1
14
15 s = input()
16 p = input()
17 lengp = len(p)
18 i = 0
19 j = match(s, p)    # 【3】
20 while j >= 0:
21     print(i + j, end=" ") # 【4】
22     i += j + lengp      # 【5】
23     j = match(s[i:], p)

```

这段代码的目的是在主串 `s` 中查找所有子串 `p` 的出现位置，并打印出每个匹配的子串在主串中的起始下标。下面是对代码的解读：

函数 `match(s, p)` 解读

2. 参数：

- `s`：主串，字符串类型。
- `p`：子串，字符串类型。

3. 逻辑：

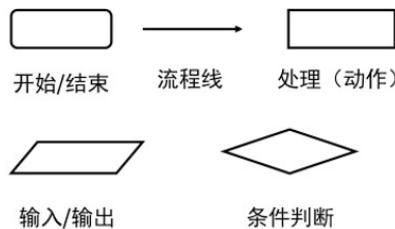
- 获取主串 `s` 和子串 `p` 的长度 `n1` 和 `n2`。
- 使用变量 `i` 作为主串 `s` 的索引，从0开始遍历到 `n1 - n2`。这是因为如果剩余的字符数量少于子串 `p` 的长度，那么就不可能再有匹配了。
- 内层循环使用变量 `j` 来逐个比较子串 `p` 和主串 `s` 中从索引 `i` 开始的连续部分是否相等【1】。
 - 如果对应字符相同，则增加 `j` 继续比较下一个字符。
 - 如果 `j` 达到了 `n2` 【2】，说明找到了一个完整的匹配，于是返回当前的索引 `i`。
- 如果内层循环因为不匹配而中断，外层循环会将 `i` 增加1，继续检查下一个可能的位置。
- 如果遍历完所有可能的位置后仍未找到匹配，函数返回 `-1`。

主程序部分解读

- 从标准输入读取主串 `s` 和子串 `p`。
- 调用 `match(s, p)` 函数来寻找第一个匹配的子串 `p` 的位置【3】。
- 进入 `while` 循环，只要 `j` 不为 `-1`（即只要找到了匹配），就执行循环体内的操作。
 - 打印出匹配子串在原字符串 `s` 中的起始位置【4】。这里 `i + j` 是因为在后续步骤中我们对 `s` 切片并调整了 `i` 的值，所以需要加上原始的偏移量 `i`。
 - 更新 `i` 的值为当前匹配位置之后的位置，加上子串 `p` 的长度 `lengp`【5】，以确保不会重复匹配同一个子串。
 - 再次调用 `match` 函数，但这次是在更新后的主串切片 `s[i:]` 上查找下一个匹配。
- 当 `match` 函数找不到更多的匹配时，它会返回 `-1`，此时 `while` 循环结束。

七、流程图与编程题 (2题, 共12分)

流程图例：



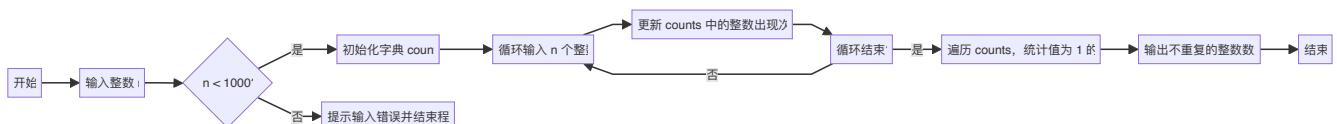
1. 给定一个由若干整数形成的序列，求这个序列中不重复的整数的数量。

例如：对于整数序列 [5 3 1 2 4 6 3 2 4 4 7 2]，其中存在的不重复的整数是 [5 1 6 7]，因此，不重复的整数的数量是 4。

请你编写一个程序，实现上述功能。具体而言，该程序：首先，接收用户从控制台输入的一个整数 n ($n < 1000$)，表示即将输入的整数序列中包含的整数的数量；然后，依次接收用户从控制台输入的 n 个整数（每行一个整数）；最后，在控制台输出计算结果（一个整数）。

1) 画出算法流程图 (3分)

(对照其程序符合流程，以及流程图规范即可)



2) 写出程序的代码 (3分)

[说明：无需考虑程序的执行效率，程序中不可 import 任何模块]

```
1 # 开始
2 n = int(input("请输入整数 n (n < 1000) : "))
3
4 # 判断 n 的范围
5 if n >= 1000:
6     print("输入错误, n 必须小于 1000")
7 else:
8     # 初始化一个字典用于记录每个整数的出现次数
9     counts = {}
10    print(f"请依次输入 {n} 个整数: ")
11    for _ in range(n):
12        num = int(input())
13        if num in counts:
14            counts[num] += 1
15        else:
16            counts[num] = 1
17
```

```

18 # 统计不重复的整数数量
19 unique_count = 0
20 for key in counts:
21     if counts[key] == 1:
22         unique_count += 1
23
24 # 输出不重复的整数数量
25 print("不重复的整数数量是: ", unique_count)
26

```

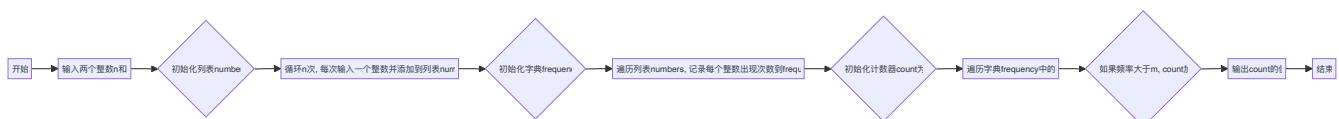
2. 给定一个由若干整数形成的序列，求这个序列中出现次数大于 m 次的整数的数量。

例如：令 $m = 2$ ，对于整数序列 $[5 \ 3 \ 1 \ 2 \ 4 \ 6 \ 3 \ 2 \ 4 \ 4 \ 7 \ 2]$ ，其中出现次数大于 2 的整数是 $[2 \ 4]$ ，因此，输出的结果是 2 。

请你编写一个程序，实现上述功能。具体而言，该程序：首先，接收用户从控制台输入的空格分隔的两个整数 n ($n < 1000$) 和 m (前者表示即将输入的整数序列中包含的整数的数量；后者即前文中给出的 m 的含义)；然后，依次接收用户从控制台输入的 n 个整数 (每行一个整数)；最后，在控制台输出计算结果 (一个整数)。

1) 画出算法流程图 (3分)

(略，对照其程序符合流程，以及流程图规范即可)



2) 写出程序的代码 (3分)

[说明：无需考虑程序的执行效率，程序中不可 import 任何模块]

###

```

1 # 出现次数大于 m 的整数的数量
2 def count_numbers_greater_than_m():
3     n, m = map(int, input().split())
4     numbers = []
5     for _ in range(n):
6         num = int(input())
7         numbers.append(num)
8
9     frequency = {}
10    for num in numbers:
11        if num in frequency:
12            frequency[num] += 1

```

```
13     else:
14         frequency[num] = 1
15
16     count = 0
17     for freq in frequency.values():
18         if freq > m:
19             count += 1
20
21     print(count)
22
23 count_numbers_greater_than_m()
```

八、问答题（4分）

请结合自身情况，简要阐述计算机技术如何能在未来帮助你更好地完成专业课的学习和胜任未来工作？

只要正常写了就可以给满分

可运用相关统计软件或编写代码(如R studio等)处理心理学相关实验数据，通过将数据可视化得到更为直观的结论，方便后读研究。

20211121笔试（Python）

一、填空题（每空1分，共15分）

- 根据`(空间)`局部性原理，在程序执行时，如果一个信息项正在被访问，那么近期它很可能也会被再次访问，存储在它附近的信息也很可能被访问到。
- 表达式`((ord('A')+1)<50) or (5*(ord('a')>90))+1`的值是`6`，类型是`int`整数。（字符A的ASCII码数值是65）

字符'A'的ASCII码值是65。因此，`ord('A')`返回65。

- `(ord('A')+1):`
 - 这里我们将字符'A'的ASCII码值（65）加1，得到66。
- `(ord('A')+1)<50:`
 - 检查66是否小于50，显然不是，所以这部分表达式的值为False。
- `ord('a'):`
 - 字符'a'的ASCII码值是97。

4. `(ord('a') > 90)`:

- 检查97是否大于90，显然是的，所以这部分表达式的值为True。

5. `5 * (ord('a') > 90)`:

- 由于`(ord('a') > 90)`的结果是True，在Python中True等价于1，所以我们有`5 * 1`，结果为5。

6. `(False or 5)`:

- 在Python中，`or`操作符会在第一个真值表达式处短路并返回它。因为5是一个非零数字，它被视为真值，所以整个表达式的值是5。

7. `((False or 5)) + 1`:

- 最后我们将5加上1，得到最终结果6。

综上所述，表达式的值是6，类型是int（整数）。

```
1 print(((ord('A') + 1) < 50) or (5 * (ord('a') > 90))) + 1    # 6
2 print((ord('A') + 1) < 50)  # False
3 print((ord('a') > 90))     # True
4 print((5 * (ord('a') > 90)))  # 5
5 print(False or 5)    # 5
6 print(1 or 5)      # 1 or先判断前面的，短路原因，前面为真，后面不再判断
7 print(5 or 1)      # 5
```

- 在登录12306网站购票时，除了要求输入用户名、密码之外，还需要购票者按要求点击网页上的图片，这实际上就是要用户输入验证码，它的作用是比较好的解决了对用户密码的暴力破解问题。
- 一幅1024×1024的图像，其颜色为24位真彩色，如果不压缩，该图像至少需要3MB存储空间；假设该图像的颜色数为200种，为了压缩存储空间，对每种颜色编号，则200种颜色至少需要1个字节表示；这时所需存储空间降低至1MB；32位的CPU通过32位地址总线所能访问的内存空间能存储如此压缩后的图片4096张。

- 图像的原始大小：

一幅1024×1024像素的图像，每个像素使用24位（即3字节）来表示颜色，那么整个图像的大小为：

$$1024 \times 1024 \times 3 = 3,145,728 \text{字节}，\text{也就是大约}3\text{MB} \text{（更准确地说是}3.145728\text{MB})。$$

- 压缩后的图像大小：

如果该图像的颜色数减少到200种，并且每种颜色用1个字节（即8位）表示，那么每个像素只需要1个字节。所以，压缩后的图像大小为：

$$1024 \times 1024 \times 1 = 1,048,576 \text{字节}，\text{也就是大约}1\text{MB} \text{（更准确地说是}1.048576\text{MB})。$$

- 32位CPU访问的内存空间：

32位地址总线可以访问的最大内存空间是 2^{32} 字节，也即是4GB（Gigabytes）。这是因为32位二进制数能表示的最大值是 $2^{32} - 1$ ，但是为了简化计算我们通常说它能够访问4GB的地址空间。

现在要计算4GB的内存空间能存储多少张这样的压缩后图片：

- 一张压缩后的图片需要1MB的空间。

- 因此，4GB的内存空间可以存储 $(4\text{GB} / 1\text{MB} = 4096)$ 张这样的图片。

综上所述，32位的CPU通过32位地址总线所能访问的内存空间能存储如此压缩后的图片共4096张。

两种不同的度量单位系统，它们用来表示计算机存储容量或信息量。这两个术语经常被混淆，但实际上它们代表不同的值：

- Kilobyte (KB):** 在国际单位制 (SI) 中，前缀 "kilo-" 表示1000。因此，在这种情况下，1 Kilobyte 等于 1000 字节。这个定义通常用于硬盘制造商和某些网络传输速度的描述中。
- Kibibyte (KiB):** 这是一个二进制前缀，由国际电工委员会 (IEC) 提出，用来更准确地表示基于2的幂次方的数值。1 Kibibyte 等于 1024 字节，即 (2^{10}) 字节。这个定义在计算机科学中更为常见，因为它反映了计算机内存地址空间是以2的幂来组织的事实。

因此，当我们谈论计算机存储时，使用 KiB 更为精确，尤其是在涉及操作系统报告文件大小或RAM 容量的时候。而 KB 则更多地出现在需要与十进制标准兼容的情况下，比如硬盘驱动器的标注。

总结：

- 1 KB (kilobyte) = 1000 bytes
- 1 KiB (kibibyte) = 1024 bytes

- 已知二进制数 a 是 00101101，如果想通过整型变量 b 与 a 做异或运算，使变量 a 的高 4 位取反，低 4 位不变，则 b 的二进制数值应是 **11110000**。

异或运算的规则是：

- 如果两个相对的位相同，则结果为0；
- 如果两个相对的位不同，则结果为1。

所以，要让 a 的高 4 位取反，b 的高 4 位应该全部为 1，因为 1 和 a 中的任何一位进行异或都会导致该位被取反。同时，为了让 a 的低 4 位保持不变，b 的低 4 位应该全部为 0，因为 0 和 a 中的任何一位进行异或都不会改变该位的值。

因此，对于一个8位的二进制数 a = 00101101，你想要的 b 的二进制数值应该是：

```
1 | a = 00101101
2 | b = 11110000
```

这样，当你对 a 和 b 进行异或运算时，你会得到：

```
1 | a ^ b = 00101101 ^ 11110000 = 11011101
```

- 微型计算机的 CPU 由寄存器，中断处理器，算术逻辑运算器和**程序控制器**四个部件组成。根据冯·诺伊曼结构，计算机由运算器，**控制器**，存储器，输入设备和输出设备五个部分相互连接组成。
- CPU 通过数据总线，**控制总线**以及地址总线和其它部件进行各种信息的传递。为了保证性能，数据总线的宽度应该与 CPU 字长一致。

数据总线的宽度通常与CPU的字长（Word Size）一致是为了保证性能和效率。这里有几个关键点：

1. **CPU 字长**：指的是CPU一次能够处理的数据量大小，它决定了CPU内部寄存器、算术逻辑单元（ALU）等组件的操作数宽度。比如32位CPU的一次操作可以处理32位的数据，而64位CPU则可以处理64位的数据。
2. **数据总线宽度**：是指数据总线一次能传输的数据量，它直接影响到CPU与内存或其他外部设备之间数据交换的速度。如果数据总线的宽度与CPU字长相同，那么在进行数据读写时，就不需要分多次传输，从而提高了数据传输的效率。
3. **地址总线宽度**：决定了CPU可以直接寻址的地址空间大小。例如，32位地址总线可以访问 (2^{32}) 个不同的地址，即4GB的物理地址空间；而64位地址总线理论上可以访问 (2^{64}) 个地址，这远远超过了目前大多数系统的实际需求。
4. **控制总线**：它不直接与性能或数据量相关，而是用于传递控制信号，如读/写命令、中断信号等，以协调CPU与其他系统组件之间的操作。

为了最大化性能，理想情况下，数据总线的宽度应该匹配CPU的字长，这样每次内存访问都可以最有效地利用CPU的能力。不过，在实际设计中，也会考虑到成本、功耗等因素来决定总线宽度。有时，为了降低成本或出于其他考虑，可能会选择较窄的数据总线，但这通常会导致性能上的妥协。

8. 如果 $38+1=40$ ，这说明使用的是**9进制数**。
9. 在计算机分层存储体系的设计中，对于寄存器、外存、内存、Cache 高速缓存，一般情况下价格最低的是外存，存储容量最小的是**寄存器**。
10. 对于按如下定义的数组 `a = [-1, 2, 5, 1, 8]+[0]*5; a[7]-a[4]` 的值是 **-8**。

二、单选题（每小题1分，共15分）

1. 下列关于存储设备说法中正确的是：

- A) CPU 寄存器的数据存取速度高于高速缓存
- B) 主存储器属于 ROM（只读存储器）
- C) 外存储器的容量一定高于主存储器
- D) 外存储器是具有易失性的存储设备

2. 计算机界的最高荣誉是：

- A) 诺贝尔奖
- B) 图灵奖
- C) 冯·诺依曼奖
- D) 斯隆奖

3. 下列关于信息表示错误的说法是：

- A) N 位二进制原码序列能表示最大有符号整数是 $2^N - 1$
- B) 英文字母的编码表示只需占用一个字节
- C) 在 ASCII 字符集中，所有数字字符 0~9 连续编码
- D) 对于声音的编码，采样频率会直接影响声音的质量

4. 判断字符变量 c 的值不是数字也不是字母时，应采用下述哪个表达式。注：关系运算的优先级高于逻辑运算。
- A) `(c<=0 or c>=9) and (c<='A' or c>='Z') and (c<='a' or c>='z')`
- B) `not((c<='0' or c>='9') and (c<='A' or c>='Z') and (c<='a' or c>='z'))`
- C) `not((c>='0' and c<='9') or (c>='A' and c<='Z') or (c>='a' and c<='z'))`
- D) `(c>='0' and c<='9') or (c>='A' and c<='Z') or (c>='a' and c<='z')`
5. 以下有关摩尔定律的描述中，错误的是：
- A) 每 18 个月，集成电路芯片上集成的晶体管数将翻一番
- B) 每 18 个月，集成电路芯片的速度将提高一倍
- C) 每 18 个月，集成电路芯片的价格将降低一半
- D) 集成电路技术一直会遵循摩尔定律发展
6. 一台笔记本电脑的配置写着“8G”，“2T”，“8 核”。这三个数字分别对应电脑的什么部件？
- A) CPU，硬盘，内存 B) 硬盘，内存，CPU
- C) CPU，内存，硬盘 D) 内存，硬盘，CPU
7. 以下哪一项决定 CPU 能直接访问的主存地址空间大小？
- A) CPU 数据总线的宽度 B) CPU 的主频
- C) CPU 地址总线的宽度 D) CPU 的指令周期
8. 操作系统的主要功能是：
- A) 控制和管理计算机系统软硬件资源和信息资源
- B) 对汇编语言程序和高级语言程序进行翻译
- C) 管理用各种语言编写的源程序
- D) 管理数据库文件
9. 关于信息安全，下面说法正确的是：
- A) 验证码与用户名和密码密切相关，因此需要牢记，并保证每次都输入相同验证码
- B) 防火墙是一种内外网的隔离技术
- C) 有些厂商宣称的启发式杀毒可以对付未知病毒是肯定可靠的
- D) 加密和解密所用的密码必须是一样的
10. 以下关于互联网的说法中不正确的是：
- A) 互联网是计算机科学与通信科学紧密结合的产物，其结构是一幅图。
- B) IPv4 协议用 32 位 2 进制数表示一个 IP 地址，通常每 8 位一段写成 10 进制数表示，如 `20.168.302.12` 是一个合法的 IPv4 地址
- C) 域名服务器（缩写 DNS）的功能是将一个合法域名（如 `pku.edu.cn`）转换成对应主机的 IP 地址。
- D) 通信协议栈保证了互联网上的电脑间能够正确的交流，它通常由定义在应用层、传输层、网络层、数据链路层以及物理层上的一系列协议构成。
11. 下列选项中，不是 C 语言中合法的变量名称的是：

A) iphone12 B) _9mate C) 8oppo D) xiaomi_11

12. 下面几个选项中不是操作系统的是：

A) Linux B) Windows 10 C) IOS D) Chrome

13. 下面有关程序变量说法错误的是：

A) 变量是程序运行过程中可以变化的量 B) 变量的值存储于内存之中

C) 变量的值存储于如硬盘、U 盘等外存之中 D) 变量可以存储值的范围大小与类型相关

14. 在 ASCII 编码表中，数字和大小写字母都是按顺序编码的，大写字母 A 的二进制 ASCII 码为 01000001，大写字母 F 的二进制 ASCII 码为：

A) 01000010 B) 01000100 C) 01000110 D) 01000011

15. 下面四个无符号整数中，（）超出了一个字节的表数范围。

A) $(231)_{10}$ B) $(257)_8$ C) $(102)_{16}$ D) $(111)_2$

三、计算题（共20分）

1. 数制转换运算（4分）

$$(15068)_{10} = (3ADC)_{16} = (35334)_8$$

- 从十进制到十六进制： $(15068)_{10}$ 转换为十六进制。

要将十进制数转换为十六进制，我们可以不断地用该数除以16并记录余数。计算过程如下：

$$(15068 \div 16 = 941) \text{ 余 } (12(C))$$

$$(941 \div 16 = 58) \text{ 余 } (13(D))$$

$$(58 \div 16 = 3) \text{ 余 } (10(A))$$

$$(3 \div 16 = 0) \text{ 余 } (3)$$

所以， $(15068)_{10} = (3ADC)_{16}$ 是正确的。

- 从十六进制到八进制： $(3ADC)_{16}$ 转换为八进制。

首先将十六进制转换为二进制，然后将二进制转换为八进制。每个十六进制位可以表示为四个二进制位，而每个八进制位则由三个二进制位表示。

$$3_{16} = 0011_2, A_{16} = 1010_2, D_{16} = 1101_2, C_{16} = 1100_2$$

连接这些二进制值，我们得到 0011101011011100_2 。现在我们将这个二进制字符串分组为每三个二进制位一组（从右向左）：

$$(000\ 011\ 101\ 011\ 011\ 100_2)$$

转换为八进制： $0_8, 3_8, 5_8, 3_8, 3_8, 4_8$

因此， $(3ADC)_{16} = (35334)_8$ 也是正确的。

$$(365.625)_{10} = (101101101.101)_2$$

- 从十进制到二进制：整数部分通过不断除以2并记录余数来转换，小数部分通过不断乘以2并记录整数部分来转换。

对于整数部分 (365)：

$$(365 \div 2 = 182) \text{ 余 } (1)$$

$$(182 \div 2 = 91) \text{ 余 } (0)$$

$$(91 \div 2 = 45) \text{ 余 } (1)$$

$$(45 \div 2 = 22) \text{ 余 } (1)$$

$$(22 \div 2 = 11) \text{ 余 } (0)$$

$$(11 \div 2 = 5) \text{ 余 } (1)$$

$$(5 \div 2 = 2) \text{ 余 } (1)$$

$$(2 \div 2 = 1) \text{ 余 } (0)$$

$$(1 \div 2 = 0) \text{ 余 } (1)$$

所以整数部分的二进制表示是 101101101_2 。

对于小数部分 (0.625)：

$$(0.625 \times 2 = 1.25) \text{ 整数部分 } (1)$$

$$(0.25 \times 2 = 0.5) \text{ 整数部分 } (0)$$

$$(0.5 \times 2 = 1.0) \text{ 整数部分 } (1)$$

所以小数部分的二进制表示是 $.101_2$ 。

1. 二进制算术运算 (4分)

$$110100 * 1100 = (1001110000)$$

$$10101011 / 1001 = (10011)$$

运算 1: 110100×1100

转换为十进制：

- $110100_2 = 52_{10}$

- $1100_2 = 12_{10}$

计算乘积：

- $52 \times 12 = 624_{10}$

转换回二进制：

- $624_{10} = 1001110000_2$

运算 2: $10101011 \div 1001$

转换为十进制：

- $10101011_2 = 171_{10}$
- $1001_2 = 9_{10}$

计算商：

- $171 \div 9 = 19_{10}$

转换回二进制：

- $19_{10} = 10011_2$

1. 二进制逻辑运算 (4分, 与: &, 或: |, 非: ~, 异或: ^)

- $1010\ 1001 \& (\sim 0101\ 0011) = (1010\ 1000)$
- $1100\ 0110 \wedge 1001\ 0101 = (0101\ 0011)$

2. 数据的加密处理在日常生活中处处可见，收发双方通过约定好的方式对信息进行一定的加密处理。 (4分, 每小题各2分)

(1) 现在已知用于加密的密钥为 0000 1001，一种简单的加密方式是使用密钥对每个需要加密的 ASCII 字符进行异或操作。假设一封秘文原始内容为 HELLO，请计算加密后的秘文内容。 (A 的 ASCII 码为 0100 0001，其余字母顺序递增) (2分)

ALEEF

ASCII 字符 'H' 到 'L' 的二进制表示如下：

- H: 0100 1000
- E: 0100 0101
- L: 0100 1100
- L: 0100 1100
- O: 0100 1111

计算每个字母的加密结果：

1. H (0100 1000) XOR 0000 1001 = 0100 0001 (A)
2. E (0100 0101) XOR 0000 1001 = 0100 1100 (L)
3. L (0100 1100) XOR 0000 1001 = 0100 0101 (E)
4. L (0100 1100) XOR 0000 1001 = 0100 0101 (E)
5. O (0100 1111) XOR 0000 1001 = 0100 0110 (F)

因此，原始文本 "HELLO" 加密后的秘文内容为 "ALEEF"。请注意，这种简单的加密方法并不安全，因为它很容易被破解，特别是当密钥较短或者重复使用时。

(2) 对于纯字母的信息，另一种常见的方法是分组处理。比如将原本使用 8 位存储的 ASCII 码两两分为一组。首先将第一个待加密字符的最高位强制置为 1，得到第一个加密字符，再用这个加密字符减去第二个待加密字符作差得到第二个加密字符。假设需要加密的 ASCII 字符串是 DK，请写出使用该方法加密后对应的十六进制数据（2分）

C479

字符 D 的 ASCII 值： 68 (十进制)

字符 K 的 ASCII 值： 75 (十进制)

设置第一个字符的最高位为 1

将字符 D 的 ASCII 值 68 转换为二进制表示，结果是 01000100。

将最高位设置为 1：

- 修改后为： 11000100 (二进制)
- 转换回十进制为： 196。此值是第一个加密字符。

计算第二个加密字符

根据题目要求，用第一个加密字符减去第二个待加密字符，得到第二个加密字符：

- 计算： 196 - 75 = 121。此值是第二个加密字符。

将结果转换为十六进制

1. 第一个加密字符： 196 转换为十六进制为 C4

2. 第二个加密字符： 121 转换为十六进制为 79

3. 视频编码中“帧率”是指单位时间内包含的图像帧数，一般使用 FPS（帧/秒）为单位。50 FPS 即每秒钟播放 50 帧图像。常见的 1080P 原始视频每帧图像包含 1920×1080 个像素点，每个像素点使用 8 位进行存储。（计算时 1920×1080 可以算作 2,000,000；1 GB = 1000 MB，其余进制同理）

(1) 一段 5 分钟的 1080P 50FPS 的原始视频约占多少 GB 的存储空间？（结果仅保留整数）（3分）

$$2\text{MB} \times 5 \times 60 \times 50 = 30000\text{MB} = 30\text{GB}$$

(2) 视频压缩技术可以通过压缩算法用减少视频占用的存储空间。我国自主研发的 AVS3 视频编码标准实现的视频压缩比可达 600:1，即视频存储时仅需要原始视频大小的 $1/600$ 。假设手机的存储空间大小为 128GB，不考虑其他因素，这部手机最多可以存储多长小时的 1080P 50FPS 的视频。

（结果仅保留整数）（1分）

$$128\text{GB} \times 600 \times 1000 / (2\text{MB} \times 3600 \times 50) = 213 \text{ 小时}$$

四、编码应用题（共12分）

1. (6分) 当今世界，计算机在医学、生物学方面有着很广阔的应用。在外访问的郭教授观测到了一段DNA序列，他希望通过互联网将观测到的这段序列传回自己的电脑上。DNA序列是由四种不同的脱氧核糖核酸（A, G, C, T）排列构成的，郭教授使用两位二进制数对它们编码，得到的编码表如下：

A	00
G	01
C	10
T	11

(1) (2分) 根据郭教授的编码规则，对核酸序列“ACCA”的二进制编码为**00101000**。一段编码的16进制表示为ACCA，则这段编码表示的核酸序列为**CCTATACC**。

16进制的ACCA转换成二进制是：1010110011001010。

由于郭教授使用两位二进制数对A, G, C, T进行编码，我们将上述二进制串每两位分为一组：

10 -> C
 10 -> C
 11 -> T
 00 -> A
 11 -> T
 00 -> A
 10 -> C
 10 -> C

因此，16进制ACCA表示的核酸序列为“CCTATACC”。

(2) (2分) 郭教授使用的传输协议是将传输的数据打成数据包来传输的，每个数据包的大小为128Byte，那么，每个数据包最多可以传输长度为**512**的核酸序列。郭教授发现的核酸序列的长度为10086，那么郭教授最少需要**20**个数据包来传输他发现的核酸序列。

为了计算每个数据包最多可以传输的核酸序列长度，首先需要知道用郭教授的编码规则，每个核酸（A, G, C, T）占用多少位。根据编码规则，每个核酸使用2位二进制数表示。

1 Byte = 8 bits。因此，一个128 Byte的数据包共有：

$$128 \text{ Bytes} * 8 \text{ bits/Byte} = 1024 \text{ bits}$$

既然每个核酸占用2位，那么一个数据包可以传输的核苷酸数量为：

$$1024 \text{ bits} / 2 \text{ bits/nucleotide} = 512 \text{ nucleotides}$$

所以，每个数据包最多可以传输长度为512的核酸序列。

计算传输长度为10086的核酸序列所需要的最少数据包数量。可以将总长度除以每个数据包可以传输的最大序列长度，然后向上取整，因为即使最后一个数据包没有被完全填满，也需要一个完整的数据包来传输剩余的数据。

$$\text{所需数据包数量} = \text{ceil}(10086 \text{ nucleotides} / 512 \text{ nucleotides/packet})$$

计算结果：

$$10086 / 512 \approx 19.697$$

由于我们需要的是最小的整数个数据包，所以我们需要向上取整到20。

(3) (2分) 在实际传输中，经常会因为各种原因导致传输的数据包出错，具体表现为某一比特由1变为0或由0变为1。在本题中，假设数据包中最多只会有一位发生0/1翻转。为了检查传输到的数据包是否有问题，常常采用奇偶校验的方式。即在数据包末尾添加一位，表示数据包中之前的位中1的个数为奇数或偶数。若为奇数，则该位为1，否则为0。假设郭教授的一个数据包中编码的核酸序列内容为“AGCGCGGGT”，则校验位应为1。郭教授收到的一个数据包编码的核酸序列为“AACGTTG”，收到的校验位为0，则郭教授收到的这段序列有(有/无)问题。

首先，根据郭教授的编码规则将核酸序列转换为二进制形式：

A: 00 G: 01 C: 10 T: 11

对于核酸序列“AGCGCGGGT”，我们将其转换为二进制：

A -> 00 G -> 01 C -> 10 G -> 01

C -> 10 G -> 01 G -> 01 G -> 01

T -> 11

连起来就是：00011001 10010101 11

由于1的数量是奇数（9个），为了使整个数据包（包括校验位）中的1的总数保持偶数，校验位应该设置为1。

现在我们来检查郭教授收到的数据包“AACGTTG”和其校验位0是否有问题。首先转换为二进制：

A -> 00 A -> 00 C -> 10 G -> 01

T -> 11 T -> 11 G -> 01

连起来就是：00001001 111101

计算1的数量有7。因为7是奇数，并且收到的校验位是0，说明有问题。

奇偶校验只能检测到奇数个错误，如果发生了偶数个错误，它们会互相抵消，导致奇偶校验无法检测到错误。奇偶校验不是完全可靠的错误检测方法，因为它不能检测所有类型的错误。

2. IP地址是连接在互联网上的主机或路由器的唯一身份标识。在目前最广泛使用的IPv4协议中，IP地址是一个32位无符号整数。例如，北京大学主页的IP地址为1010010011010011000011101010000。显然，这样的表示是不便于记忆和使用的，程序员通常使用点分制，即将IP以8位一组分成四组，将其转化为十进制，以点分隔。北京大学主页IP地址的点分制表示为162.105.131.160。

(1) (1分) 百度的IP地址为1011011000111101100100000000111，其点分制表示为182.61.200.7。

(2) 为加强校园网络安全防护，为广大学生营造健康文明的网络环境，北京大学计算中心拟屏蔽部分存在违法内容的网站，部分代码逻辑如下：

```
1 # fetchIPAddress() 是一个自定义函数，用来输入一个
2 # 点分制表示的 IP 地址，并将其转换成二进制 IP 地址
3 def fetchIPAddress():
4
5     a = b = c = d = 0
```

```

6
7     【1】读入点分制 IP 地址
8     return (a << 24) | (b << 16) | (c << 8) | d;
9     # "<<" 是二进制数左移操作, ">>" 是二进制数右移操作
10    # 其后的整数表示该二进制数向左或向右移动多少位。
11    # 如 10010111<<2, 表示二进制数左移 2 位得到 01011100
12    # 如 10010111>>2, 表示二进制数右移 2 位得到 00100101。
13    # " | " 是二进制数据按位操作
14
15
16 def main():
17     # 读入 blacklistsiz 个黑名单地址, 将其存储在 blacklist 数组中
18     blacklist_size = int(input())
19     blacklist = [0] * 1000
20     # 读入黑名单中的IP地址
21     for i in range(blacklist_size):
22         blacklist[i] = fetchIPAddress()
23
24     # 读入发送地址和接收地址
25     sender_ip = fetchIPAddress()
26     receiver_ip = fetchIPAddress()
27
28     # 检查数据包的接收地址是否为北京大学局域网地址, 不是则丢弃
29     【2】提取局域网编号
30     if subnetID != 0x2269:
31         ...丢弃
32         return 0
33
34     # 检查数据包的发送地址是否在黑名单中, 是则屏蔽, 否则转发
35     for i in range(blacklist_size):      # 【3】
36         if sender_ip == blacklist[i]:
37             ...屏蔽
38             return 0
39         else:
40             ...转发
41
42     return 0
43
44
45 main()
46

```

(i) (1 分) 在 【1】 处需要读入点分制的 IP 地址, 并依次存入整型变量 a, b, c, d 中, 下列读入方法正确的是 A 一。

- A. a, b, c, d = map(int, input().split("."))
- B. a, b, c, d = map(int, input().split())
- C. a, b, c, d = map(int, input().split(" "))
- D. 以上方法均不对

(ii) (2分) 北京大学网络的32位IP地址的编码规则如下：



最高的两位10表示北京大学网络属于B类地址，接下来的14位表示局域网编号（北大的局域网编号是100010011010001，十六进制表示为0x2269），最后16位表示局域网内的主机编号。在【2】处，提取接收地址的局域网编号代码为（“>>”是二进制数右移操作）：

```
subnetID = (receiverIP & 0xffff0000 或二进制表示) >> 16
```

(iii) (2分) 在实际运行过程中，发现很多黑名单网站没有被成功屏蔽，原因是【3】处的循环内存在错误，请指出并给出改正方案：

错误：必须将发送地址与完整的黑名单比较，与任何一个黑名单地址都不同才能被转发。原代码中只要与某一个黑名单地址不同即被转发。（意思对即可）

修改：使用一个标记变量，初始时为0，当发送地址与黑名单地址相同时标记变量修改为1，在循环结束后判断标记变量的值决定是屏蔽还是转发。或将i设为全局变量，通过比较i是否等于blacklistsize进行判断（意思对即可）。

else缩进往前提，与for对齐

```
1 for i in range(blacklist_size):      # 【3】
2     if sender_ip == blacklist[i]:
3         ...屏蔽
4     return 0
5 else:
6     ...转发
```

五、程序阅读题（共12分）

阅读下面的程序，说明该程序实现的功能是什么；并对给定的输入，写出程序执行后的输出。

第1题

```
1 m = [float(x) for x in input().split()]
2
3 a = b = c = d = 0
4 n = int(m[0])
5 for i in range(1, n + 1):
6     c += m[i]
```

```

7     if i == 1:
8         a = b = m[i]
9     else:
10        if a < m[i]:
11            a = m[i]
12        if b > m[i]:
13            b = m[i]
14    c /= n
15    for i in range(1, n + 1):
16        d += (m[i] - c) * (m[i] - c)
17    d /= n
18
19 print(f"{a:.2f} {b:.2f} {c:.2f} {d:.2f}")

```

程序功能（2分）：

计算输入数组的最大值、最小值、均值和方差，并依次输出，保留两位小数。

输入为： 5 5.0 3.0 1.0 4.0 2.0

程序输出为（2分）： 5.00 1.00 3.00 2.00

第2题

```

1 n = int(input())      # n在1-30之间
2 alist = [True for i in range(n + 1)]
3 k = 0
4 for i in range(2, n + 1):
5     if alist[i]:
6         j = 2
7         while i * j <= n:
8             alist[i * j] = False
9             j += 1
10        k += 1
11 print(k)

```

程序功能（2分）：

输出 n 以内的素数的个数

当输入的数为： 30， 输出为（2分）： 10

第3题

```

1 def issth(x):
2     d = 3
3     while d * d <= x:
4         if x % d == 0:
5             return False
6         d += 2

```

```

7     return True
8
9 n = int(input())
10 if (n > 4) and (n % 2 == 0):
11     i = 3
12     while i < n:
13         if issth(i) and issth(n - i):
14             print(f"{n}={i}+{n - i}")
15             break
16         i += 2
17     else:
18         print("WOO")
19 else:
20     print("BYE")

```

程序功能（2分）：

对大于4的偶数验证哥德巴赫猜想，输出两个素数的和。其它整数则输出BYE

当输入的数分别为：4, 5, 6, 7, 8时，输出分别为（2分）：

输入为4时：BYE

输入为5时：BYE

输入为6时：6=3+3

输入为7时：BYE

输入为8时：8=3+5

六、程序填空（每空1分，共10分）

1、下面的程序是一个简易计算器：

如果输入是“* 2 3”则输出：2 * 3 = 6

如果输入“/ 2 0”则输出为：The variable b can NOT be zero!

如果输入“/ 4 5”则输出：4 / 5 = 0

如果输入“+ 4 5”则输出：4 + 5 = 9

如果输入“% 4 5”则输出：Invalid operator!

请完成程序中空白的部分

```

1 op, a, b = input().split() # 【1】
2 a, b = int(a), int(b) # 【2】
3
4 if op == "+":
5     print(f"{a} + {b} = {a + b}")
6 elif op == "-":
7     print(f"{a} - {b} = {a - b}")
8 elif op == "*":
9     print(f"{a} * {b} = {a * b}")
10 elif op == "/":
11     if b == 0: # 【3】

```

```

12     print("The variable b can NOT be zero!")
13 else:
14     print(f"{a} / {b} = {a // b}") # 【4】 使用浮点除法
15 else: # 【5】
16     print("Invalid operator!")

```

2、长整数整除问题：

以字符串形式输入一个不超过 10000 位的正整数（超出 int 表示范围），判断其是否能被 22 整除，能则输出“YES”，否则输出“NO”。如果字符串中出现非数字字符，输出“Invalid input!”。请补充完善程序。

提示：

- 1: 语句 `s = input()` 表示读入字符串 `s`
- 2: 函数 `len(s)` 返回值为字符串 `s` 的长度
- 3: 字符 '0'~'9' 的 ASCII 码分别为 48 ~ 57
- 4: 整数被 11 整除当且仅当奇数位之和与偶数位之和的差被 11 整除。例如：

$13915 = 11 * 1265 \rightarrow (1+9+5)-(3+1)=11 \rightarrow$ 能被 11 整除

$465362 = 11 * 42305 + 7 \rightarrow (4+5+6)-(6+3+2)=4 \rightarrow$ 不能被 11 整除

- 5: 整数被 22 整除当且仅当同时被 2 和 11 整除

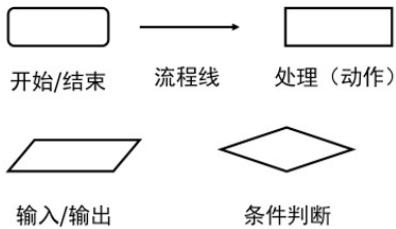
```

1 def main():
2     s = input()
3     n = len(s)
4     a = []
5     for i in range(n):
6         if not ('0' <= s[i] <= '9'): # 【1】
7             print("Invalid input!")
8             return 0
9         else:
10            a.append(ord(s[i]) - 48)
11
12    feature_11 = 0
13    for i in range(n):
14        if i % 2 == 0: # 【2】
15            feature_11 += a[i]
16        else:
17            feature_11 -= a[i] # 【3】
18
19    if (a[n - 1] % 2 == 0) and (feature_11 % 11 == 0): # 【4】 【5】
20        print("YES")
21    else:
22        print("NO")
23
24
25 main()

```

七、流程图与编程题 (共12分)

流程图例：

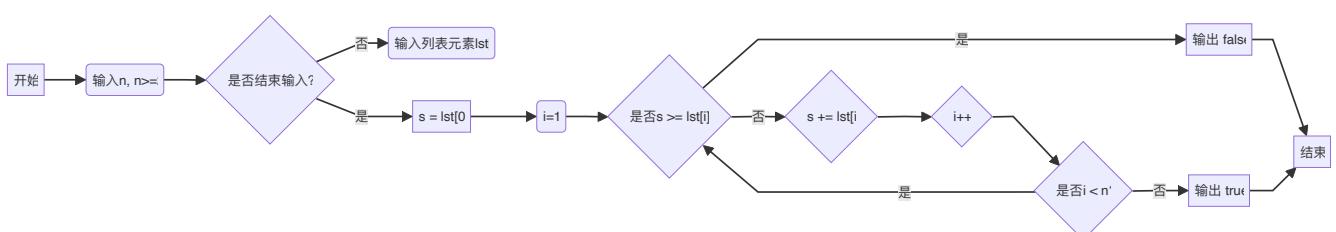


1、给定至少由两个整数形成的一个序列，称这个整数序列是一个陡峭的序列，当且仅当如下条件成立：对于这个序列中的每一个元素 a 而言，如果 a 的前面存在其他元素，那么， a 前面所有元素的和小于 a 。例如：[1,2]是一个陡峭的整数序列，[1,2,3]不是一个陡峭的整数序列，[1,2,4]是一个陡峭的整数序列，[1,2,4,8]也是一个陡峭的整数序列。

请你编写一个程序，判断用户从控制台输入的一个整数序列是否是陡峭的。

具体而言，该程序首先接收用户从控制台输入的一个整数 n （假设 $n \geq 2$ ），表示即将输入的整数序列中包含的整数的数量；然后，程序依次接收用户从控制台输入的 n 个整数；最后，程序在控制台输出计算结果（如果输入的整数序列是陡峭的，则输出字符串 true；否则，输出字符串 false）。

1) 画出算法流程图 (3 分)



2) 写出程序的代码 (3 分)

```
1 n = int(input()) # n>=2
2 lst = []
3 for i in range(n):
4     lst.append(int(input()))
5
6 s = lst[0] # s是累计和
7 for i in range(1, n):
8     if s >= lst[i]: # 如果不陡峭
9         print("false")
10        break
11     s += lst[i] # 继续累计
12 else:
13     print("true")
```

2、对于一个整数序列，其中连续的 0 到多个元素形成该序列的一个子段。一个子段中所有元素的和称为“子段和”。一个包含 0 个元素的子段，其“子段和”定义为 0。给定一个整数序列，求这个整数序列的最大子段和（所谓最大子段和，就是一个整数序列的所有子段和中，最大的那个）。

例如：[-1,-2]的最大子段和是 0；[1,2]的最大子段和是 3；[1,2,-2]的最大子段和是 3；[1,2,-2,3]的最大子段和是 4；[1,2,-2,3,-4,5,6,-11]的最大子段和是 11；[3,1,-4,-1,1,2,-1]的最大子段和是 4。

请你编写一个程序，计算用户从控制台输入的一个整数序列的最大子段和。

具体而言，该程序首先接收用户从控制台输入的一个非负整数 n ，表示即将输入的整数序列中包含的整数的数量；然后，程序依次接收用户从控制台输入的 n 个整数；最后，程序在控制台输出这个整数序列的最大子段和。

1) 画出算法流程图 (3 分)

2) 写出程序的代码 (3 分)

[提示：该题是上一题“陡峭序列判断”的扩展版本。想一想，需要在哪里扩展呢？回顾一下，在上一题中你是不是用了一个变量累加到目前为止遍历到的数值的和。这个累加变量在这个题目中也是需要用到的。同时，想一想，为什么允许“一个子段可以包含 0 个元素”且规定“一个包含 0 个元素的子段，其子段和定义为 0”，这给我们解题带来了什么便利？】

```
1 n = int(input()) # n>=2
2 lst = []
3 for i in range(n):
4     lst.append(input())
# 方法1：直接求子段和
5 smax = 0
6 for i in range(n):
7     for j in range(i, n):
8         s = sum(lst[i : j + 1]) # 子段和[i, j]
9         if smax <= s:
10            smax = s
11
12 print(smax)
13
14 # 方法2：累计子段和
15 smax = 0
16 for i in range(n):
17     s = 0
18     for j in range(i, n):
19         s += lst[j] # 累计子段和[i, j]
20         if smax <= s:
21             smax = s
22
23 print(smax)
24
25 # 方法3：动态规划
26 smax = 0
27 s = 0
28 for i in range(n):
29     s = max(lst[i], s + lst[i])
30     if smax <= s:
31         smax = s
32
33 print(smax)
```

八、问答题 (4分)

(以下三个题目，选做一题即可)

- (1) 从你的角度出发，阐述计算机是如何影响人类的职业分工？
- (2) 现代计算机开始大量使用非冯架构芯片，例如：专门用于矩阵运算的芯片。这种芯片功能单一，只能进行矩阵乘法计算，但是计算速度是普通 CPU 的 1000 多倍。专用芯片不能单独使用，必须和通用 CPU 联合使用，由 CPU 来驱动专用芯片的运行。请论述专用芯片的发展会给计算机各个方面带来哪些变化？
- (3) 假设未来人类发明了一种 3-bit，可以用一个 bit 来表示一位 3 进制数，并提出了电路构造方法，实现了 3 进制计算机，你认为这种 3 进制计算机可能在哪些方面优于现有 2 进制计算机？

只要正常写了就可以给满分

20111104笔试 (C)

一、填空题 (每空1分，共10分)

1. CPU内部结构由哪四部分组成：**寄存器、算术逻辑运算器(ALU)、程序控制器和中断处理器。**
2. 根据计算机信息的**分层存储原理**，在存储器硬件的金字塔结构中，从上到下，容量越来越大，速度越来越慢。
3. 在计算机系统中，是通过**文件**和**文件系统**来组织和管理存储在外存储设备(硬件)上的信息的。
4. 现已声明浮点类型的变量x，从控制台接收用户输入的浮点数值、对x进行赋值的程序语句为**x = float(input())**
5. 现有整型变量x，判断"x大于1，且小于100这个条件是否成立的条件表达式为 **x > 1 and x < 100**
6. for i in range(6)这个语句中，控制的循环次数为 **6**

二、单项选择题 (每题2分，共20分)

- 1.下列模块哪一个不属于“冯·诺依曼”结构?
A) 存储器 B) 运算器 C) **连接器** D) 控制器
- 2.第一代电子计算机是基于什么电子技术制成的?
A) **电子管** B) 晶体管 C) 二极管 D) 硅片
- 3.著名的“摩尔定律”是指，芯片的零件密度每**18**个月增加**1**倍?
- 4.当你在浏览器里面打开www.pku.edu.cn主页时，是什么服务器把"www.pku.edu.cn"这个URL解析为具体的IP地址--162.105.203.104?
A) WWW B) WEB C) **DNS** D) HTTP

5.下面说法正确的是:

- A) 一个Python程序是由若干个名称为**main**的主函数和若干个自定义函数(可缺省)组成。
- B) Python语言对字母的大小写不敏感。
- C) Python 3.x 中的字符串类型**str**是基于Unicode的, 这意味着每个字符 (包括汉字) 都被看作是一个单独的字符, 而不管它实际上由多少字节组成。
- D) 计算机病毒是指通过计算机传播的危害人体健康的一种病毒。

原题是针对C语言描述。在UTF-8编码下, 一个汉字可能占用3个或更多字节; 而在GBK等编码下, 一个汉字占用2个字节。

在2012年11月23日计概试卷中, 有下面一个选择, 目前看来是没有正确的答案。

下面说法正确的是

- A) 一个C程序是由若干个名称为**main**的主函数和若干个自定义函数 (可缺省) 组成。
- B) C语言对字母的大小写不敏感。
- C) 对于汉字字符, C语言中需要将2个**char**数据当作一个整体。
- D) 计算机病毒是指通过计算机传播的危害人体健康的一种病毒

对于汉字字符, C语言中需要将2个**char**数据当作一个整体。python也是这样吗?

在C语言和Python中处理汉字字符的方式有所不同。

C语言

在C语言中, **char** 类型通常用于表示单个字节 (8位), 因此它适合存储ASCII字符集中的字符。然而, 对于多字节字符如汉字, 一个汉字通常需要多个字节来表示 (例如, 在UTF-8编码下, 一个汉字可能占用3个或更多字节; 而在GBK等编码下, 一个汉字占用2个字节)。为了正确地处理汉字, 你不能简单地将两个**char** 当作一个整体, 而是应该使用能够容纳多字节字符的数据类型或者结构体, 并且按照特定的字符编码规则进行操作。

Python

Python 3.x 对字符串的处理与C语言有很大不同。Python 3.x 中的字符串类型 **str** 是基于Unicode的, 这意味着每个字符 (包括汉字) 都被看作是一个单独的字符, 而不管它实际上由多少字节组成。Python会自动处理底层的字节表示, 使得开发者无需关心单个字符的具体字节数。例如:

```
1 # 定义一个包含汉字的字符串
2 s = "你好, 世界"
3
4 # 遍历字符串中的每个字符
5 for char in s:
6     print(char) # 每次迭代都会输出一个完整的汉字或符号, 而不是字节的一部分
7
8 # 获取字符串长度, 这里返回的是字符数量, 而不是字节数
9 length = len(s)
```

如果你确实需要访问字符串的字节表示 (例如为了网络传输或文件保存), 你可以使用 **encode()** 方法将 **str** 对象转换为字节类型的对象 (**bytes**), 然后根据需要处理这些字节数据。同样地, 可以使用 **decode()** 方法从字节数据还原成 **str** 对象。

总结来说，在Python中你不需要特别将两个char数据当作一个整体来处理汉字字符，因为Python已经提供了更高层次的抽象来简化这种操作。

6.下列关于信息存储的说法不正确的是?

- A) 一般而言，硬盘存储的数据量比内存大。 B) 硬盘的读取速度比内存要慢。
C) 存有数据的硬盘比空硬盘重。 D) 内存里的数据断电之后丢失。

7.下列函数定义哪一个语法完全正确?

- A) `int a; scanf("Please input A:%d", a); printf("A is %d", a);`
B) `int a; scanf("Please input A: %d", &a); printf("A is %d", a);`
C) `int a; scanf("Please input A:%d", a); printf("A is %d",&a);`
D) `int a; scanf("Please input A:%d", &a); printf("A is %d", &a);`

8.请问程序

```
1 #include <stdio.h>
2
3 int main() {
4     int x = 2011;
5     if (x++ == 2012) {
6         printf("THE END,%d", x);
7     } else {
8         printf("NOT YET,%d", x);
9     }
10    return 0;
11 }
```

的输出是什么?

- A) THE END, 2011 B) NOT YET, 2012
C) THE END, 2012 D) NOT YET, 2011

9.请问程序

```
1 #include <stdio.h>
2
3 int main() {
4     int x = 1;
5     if( x = 0 )
6         x = x + 2;
7
8     printf("%d", x);
9     return 0;
10 }
```

执行之后 x 的值是多少? 0

10.下列程序运行后，输出为 12

```

1 #include <stdio.h>
2
3 int main() {
4     int k=5;
5     if(k > 5)
6         k = k * 2;
7     k = k + 1;
8     if(k > 3)
9         k = k + k,
10    printf("%d", k);
11    return 0;
12 }
```

三、计算题（共15分）

1.数制转换运算（4分）

$(58.493)_{10} = (2011.1111)_3$ 要求精确到小数点后4位

整数部分: 58_{10} 转三进制。通过不断除以 3 取余数的逆序得到。

1	$58 \div 3 = 19 \dots 1$
2	$19 \div 3 = 6 \dots 1$
3	$6 \div 3 = 2 \dots 0$
4	$2 \div 3 = 0 \dots 2$

小数部分: 0.493_{10} 转三进制。通过不断乘以 3 取整数部分（直到结果为整数或达到所需精度为止）。

$(110111100010101001)_2 = (674251)_8$

将二进制数分为每 3 位一组，从最低位开始补零： 110 111 100 010 101 001

每组按八进制规则计算。

- $110 = 6, 111 = 7, 100 = 4, 010 = 2, 101 = 5, 001 = 1$

2.二进制算术运算(4分)

$11100.0101 + 1010.01111 = 100110.11001$

将两个二进制数对齐小数点，逐位相加，进位处理

$1001110 * 11.010 = 11111101.10$

二进制乘法遵循与十进制乘法相同的规则，只是更简单，因为每个位上只能是0或1。

直接用二进制进行乘法运算。下面是如何一步一步完成这个计算的：

```

1      1001110
2  ×     11.010
3
4      0000000      (1001110 * 0)
5      1001110      (1001110 * 1, 向左移一位)
6      0000000      (1001110 * 0, 向左移两位)
7      1001110      (1001110 * 1, 向左移三位)
8  + 1001110      (1001110 * 1, 向左移四位)
9
10     11111101.100

```

所以, $1001110_2 \times 11.010_2 = 11111101.100_2$ 。最后一位是0, 通常我们会省略它

这里的关键点在于理解如何处理小数点的位置。在二进制乘法中, 你只需要像平时一样做乘法, 然后根据两个因数中小数点后的总位数来确定最终结果中的小数点位置。在这个例子中, 11.010_2 小数点后有3位, 因此结果也必须有3位小数 (最后一位是0, 通常我们会省略它)。

3.二进制按位逻辑运算(3分)

按位逻辑或: $1100101 \mid 1010101 = 1110101$

规则: 对应位中, 只要有一个为1, 则结果为1。

按位逻辑异或: $1010111 \wedge 1001110 = 0011001$

规则: 对应位中, 两位不同则结果为1, 相同则为0。

4.表达式(4分)

计算表达式的值:

```
((4/2==2 &&5/3==1) & (2011%2==0)) ^ (1111<=10001)
```

1

给出一个条件表达式, 表示某个小于1000的整数(x)不能被5整除, 且各位数字都不等于5时为真。

```
(x%5!=0) && (x/100!=5) && (x%100/10!=5) && (x%10!=5)
```

四、程序阅读题 (3题, 共15分)

1.阅读下面的程序, 说明这个程序实现的功能是什么; 并对给定的输入, 写出程序执行后的输出。(4分)

```

1 #include <stdio.h>
2
3 int main() {
4     int x, y, z;
5     int tmp;
6     scanf("%d %d %d", &x, &y, &z);

```

```

7   if (x >y){
8       tmp = x;
9       x = y;
10      y = tmp;
11  }
12  if (y > z){
13      tmp = y;
14      y = z;
15      z = tmp;
16  }
17  if (x > y){
18      tmp = x;
19      x = y;
20      y = tmp;
21  }
22  printf("%d %d %d", x, y, z);
23  return 0;
24 }
25 // 输入: 12 45 31

```

程序功能：接收从控制台输入的三个整数，然后按照从小到大的顺序输出这三个数。

对于“12 45 31”这个输入，程序的输出是“12 31 45”。

2.阅读下面的程序，计算这个程序的输出值是多少。 (5分)

```

1 #include <stdio.h>
2
3 int main() {
4     int i, t;
5     int a = 2, b = 1;
6     double s = 0;
7
8     for(i=1; i<=10; i++){
9         s += a/b;
10        t = a;
11        a = a + b;
12        b = t;
13    }
14
15    printf ("%lf\n", s);
16    return 0;
17 }

```

11.000000

如果学生给出的是11，或者带小数点的11，无论小数点后跟了几个零，仍然给满分5分

程序运行过程

在每次循环中，程序执行以下操作：

1. 初始化：

- o `a = 2`
- o `b = 1`
- o `s = 0`

2. 循环10次，每次执行：

- o 计算 `a / b` 并将结果累加到 `s` 中（注意这里是整数除法）
- o 更新 `a` 和 `b` 的值，类似于斐波那契数列的生成方式

3. 具体步骤：

循环次数	a (初始)	b (初始)	a/b (整数除法)	s += a/b (累计)	更新后的a	更新后的b
1	2	1	2	$0 + 2 = 2$	3	2
2	3	2	1	$2 + 1 = 3$	5	3
3	5	3	1	$3 + 1 = 4$	8	5
4	8	5	1	$4 + 1 = 5$	13	8
5	13	8	1	$5 + 1 = 6$	21	13
6	21	13	1	$6 + 1 = 7$	34	21
7	34	21	1	$7 + 1 = 8$	55	34
8	55	34	1	$8 + 1 = 9$	89	55
9	89	55	1	$9 + 1 = 10$	144	89
10	144	89	1	$10 + 1 = 11$	233	144

结果解释

从表格可以看出，除了第一次循环外，所有其他的 `a / b` 都是 `1`（因为 `a` 和 `b` 是连续的斐波那契数，而较大的斐波那契数除以较小的斐波那契数总是接近于1）。因此，`s` 的值在每次循环中几乎都增加了1（除了第一次增加了2）。

最终，经过10次循环后，`s` 的总和为 `11`，这就是为什么输出结果是 `11.000000`。

使用 `%lf` 来正确打印双精度浮点数：

3.阅读下面的程序，说明这个程序实现的功能是什么；并对给定的输入，写出程序执行后的输出。（6分）

```

1 #include <stdio.h>
2
3 int main() {
4     int num, s1 = 0, s2 = 0;
5     scanf("%d", &num);
6     if (num < 0) {
7         num = - num;
8     }
9     while (num != 0) {
10         s1 += ((num%10)%2 == 0) ? 0 : (num%10);
11         s2 += ((num%10)%2 != 0) ? 0 : (num%10);
12         num /= 10;
13     }
14     printf("%lf\n", 1.0*s1/s2);
15     return 0;
16 }
```

参考答案：

- 程序功能：接收用户从控制台输入的一个整数，计算并输出这个数的各个数位中奇数数字之和与偶数数字之和的比值。
- 对于输入“1234”，程序的输出是“0.666667”；对于输入“-233”，程序的输出是“3.000000”

{如果学生给出的输出数字小数点后的位数不为6，或者小数点后的第6位数字没有四舍五入，则应该扣除1分}

五、程序填空题（每空1分，共10分）

1. 对某些带电感L和电阻R的电路，其自然衰减频率由公式：频率 $f = \sqrt{\frac{1}{LC} - \frac{R^2}{4C^2}}$ 给定。希望研究频率随电容C的波动情况。下面的程序，用于计算从0.01到0.1、步长为0.01的不同C值时的频率（保留3位小数），请在空白处填写缺失的程序代码。

```

1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     float i, C, L, R, f;
6     scanf("%f %f", &L, &R);
7
8     for (i = 1; i <= 10; i++) {
9         C = 0.01 * i;
10        f = sqrt(1 / (L * C) - R * R / (4 * C * C));
11        printf("%.3f\n", f);
12    }
13
14    return 0;
15 }
```

需要填空的地方是：

第2行 <math.h>

第10行 $R * R / (4 * C * C)$

第11行 %3.f

2.下面的程序给出了2006年指定月份天数，请在空白处填写缺失的程序代码。

```
1 #include <stdio.h>
2
3 int main() {
4     int m, d; /* 定义月份及天数变量 */
5     scanf("%d", &m); /* 输入月份 */
6     if (m == 4 || m == 6 || m == 9 || m == 11) { /* 判断是否4、6、9或11月 */
7         d = 30; /* 是则当月30天 */
8     } else if (m == 2) { /* 否则，是否为2月 */
9         d = 28; /* 是则当月28天 */
10    } else { /* 其余月份31天 */
11        d = 31;
12    }
13
14    printf("%d", d); /* 输出月份天数 */
15    return 0;
16 }
```

需要填空的地方是：

第3行 &m

第4行 $m == 4 || m == 6 || m == 9 || m == 11$

第8行 else if ($m == 2$)

第10行 else

3.下面的程序，对于给定一个正整数 k ($1 < k < 10$)，求1到 k 的立方和 m 。即 $m = 1 + 2^3 + 3^3 + \dots + k^3$ 。

```
1 #include <stdio.h>
2
3 int main() {
4     int i, k; /* 循环相关的变量 */
```

```

5     int m = 0; /* 累加变量，初值为0 */
6     scanf("%d", &k); /* 读入循环次数k */
7
8     for (i = 1; i <= k; i++) { /* 循环k次，递推计算m(i) */
9         m += i * i * i; /* m(i) = m(i-1) + i*i*i */
10    }
11
12    printf("%d", m); /* 输出最终结果 */
13
14    return 0;
15 }
```

需要填空的地方是：

第5行 `m = 0`

第8行 `i = 1; i <= k; i++`

第9行 `+=`

六、程序设计题（4题，共25分）

1. 求加速度。（5分）

描述

在物理学中，我们知道速度和加速度之间的关系为： $V_t = V_0 + a \cdot t$ ，其中 V_0 是起始速度（米/秒）， a 是加速度（米/秒²）， t 是时间（秒）， V_t 是经过 t 秒后的速度（米/秒）。已知 V_0 ， V_t 和 t ，求加速度 a 。

关于输入

一行，有3个数，均为浮点数（float），分别是 V_0 ， V_t 和 t ，用空格隔开。

关于输出

一行，输出加速度 a 的值。

```

1 #include <stdio.h>
2
3 int main() {
4     float vt, v0, a, t;
5     scanf("%f %f %f", &v0, &vt, &t);
6
7     a = (vt - v0) / t;
8
9     printf("%.2f", a);
10
11    return 0;
12 }
```

以上是参考，任何正确的程序，均可给满分。其他情况，酌情给分。

2.计算上机成绩。 (5分)

描述

编程网络上布置给大家的上机题目既有练习，也有作业。计算上机成绩时，练习和作业的完成情况都会计入总成绩中。作业占总成绩的80%，练习占总成绩的20%。现请根据学生A的做题情况给出他的成绩。若练习总题数为m、作业总题数为n，学生A完成的练习题数为a、作业题数为b，则学生A的上机成绩（百分制）g为：
$$(100*a/m)*0.2 + (100*b/n)*0.8$$
。若 $g \geq 85$ ，则学生A的上机成绩为“优”，若 $85 > g \geq 75$ ，则学生A的上机成绩为“良”，若 $75 > g \geq 60$ ，则学生A的上机成绩为“中”，若 $g < 60$ ，则学生A的上机成绩为“差”。

关于输入

4个整数，分别是练习总题数m、作业总题数n、学生A完成的练习题数a、作业题数b

关于输出

学生A的上机成绩：优、良、中、差

```
1 #include <stdio.h>
2
3 int main() {
4     int m, n, a, b;
5     double g;
6     scanf("%d%d%d%d", &m, &n, &a, &b);
7
8     g = (100.0 * a / m) * 0.2 + (100.0 * b / n) * 0.8;
9
10    if (g >= 85) {
11        printf("优");
12    } else if (g >= 75) {
13        printf("良");
14    } else if (g >= 60) {
15        printf("中");
16    } else {
17        printf("差");
18    }
19
20    return 0;
21 }
```

以上是参考，任何正确的程序，均可给满分。其他情况，酌情给分。

3.甲流病人初筛。 (7分)

描述

在甲流盛行时期，为了更好地进行分流治疗，医院在挂号时要求对病人的体温和咳嗽情况进行检查，对于体温超过37.5度（含等于37.5度）并且咳嗽的病人初步判定为甲流病人（初筛）。现需要统计某天前来挂号就诊的病人中有多少人被初筛为甲流病人。

关于输入

第一行是某天前来挂号就诊的病人数n，

其后有n行，每行是病人的信息，包括2个信息：1<体温（float）<100、是否咳嗽（整数，1表示咳嗽，0表示不咳嗽）

关于输出

一行，是一个整数m，表示被初筛为甲流的病人数

例子输入

```
1 | 5
2 | 38.3 0
3 | 37.5 1
4 | 37.1 1
5 | 39.0 1
6 | 38.2 1
```

例子输出

```
1 | 3
```

```
1 | #include <stdio.h>
2 |
3 | int main() {
4 |     int n, m = 0, i, isCough;
5 |     float t;
6 |
7 |     scanf("%d", &n);
8 |
9 |     for (i = 0; i < n; i++) {
10 |         scanf("%f%d", &t, &isCough);
11 |         if (t >= 37.5 && isCough) {
12 |             m++;
13 |         }
14 |     }
15 |
16 |     printf("%d", m);
17 |
18 |     return 0;
19 | }
```

以上是参考，任何正确的程序，均可给满分。其他情况，酌情给分。

4.最大值和最小值的差。 (8分)

描述

输出一个整数序列中最大的数和最小的数的差。

关于输入

输入分为两行：

第一行为n， 表示整数个数， $1 < n < 100$

第二行为n个整数， 以空格隔开， 每个整数不会大于1000

关于输出

输出n个数中最大值和最小值的差

例子输入

5

2 5 7 4 2

例子输出

5

```
1 #include <stdio.h>
2
3 int main() {
4     int n, t, i, max, min;
5
6     // 读取输入的整数个数
7     scanf("%d", &n);
8
9     // 循环读取每个整数，并确定最大值和最小值
10    for (i = 0; i < n; i++) {
11        scanf("%d", &t);
12
13        if (i == 0) {
14            // 初始化 max 和 min 为第一个输入的值
15            max = t;
16            min = t;
17        } else {
18            // 更新 max 和 min
19            if (t > max) {
20                max = t;
21            }
22            if (t < min) {
23                min = t;
24            }
25        }
26    }
27
28    // 输出最大值与最小值的差
29    printf("%d\n", max - min);
30
31    return 0;
```

以上是参考，任何正确的程序，均可给满分。其他情况，酌情给分。

七、问答题（5分）

谈谈你对计算机的理解及对本课程的建议。

回答，就满分。

20081116笔试 (C)

一、填空题（共 10 分，每空0.5分）

1. 为现代电子计算机的出现作出了重要贡献的两位科学家分别是（）和（）。
○ 阿兰·图灵（Alan Turing）和冯·诺伊曼（John von Neumann）是计算机科学领域的两位重要人物。
2. 冯·诺伊曼结构计算机的5个主要部件分别是：运算器、控制器、存储器、输入设备和输出设备。

冯·诺伊曼结构计算机的5个主要部件分别是：

运算器（Arithmetic and Logic Unit, ALU）

- 负责执行计算和逻辑运算，比如加法、减法、逻辑与、或等操作。

控制器（Control Unit, CU）

- 负责解释和执行指令，协调计算机其他部件的工作。它从内存中读取指令并将其转换为具体操作。

存储器（Memory）

- 用于存储程序、数据以及中间结果，通常分为主存储器（如RAM）和辅助存储器（如硬盘）。

输入设备（Input Devices）

- 用于向计算机输入数据和指令，例如键盘、鼠标、扫描仪等。

输出设备（Output Devices）

- 用于将计算结果和信息输出给用户，例如显示器、打印机、扬声器等。

这五大部件通过**总线（Bus）**相互连接，构成一个完整的计算机系统。冯·诺伊曼结构的核心思想是存储程序，即将指令和数据以相同的形式存储在内存中，并依次执行。

3. 请列举出你所知道的三种互联网通讯协议的名称：**TCP, UDP和HTTP**。

TCP (Transmission Control Protocol) : 传输控制协议

UDP (User Datagram Protocol) : 用户数据报协议

HTTP (Hypertext Transfer Protocol) : 超文本传输协议

4. 计算机程序中的3种基本控制结构是：顺序结构、分支结构和循环结构。
5. CPU内部包含的四个主要部件是：算术逻辑运算器（ALU）、寄存器组、中断处理器、和程序控制器。

算术逻辑运算器（ALU）：执行算术和逻辑运算

寄存器组：临时存储数据

中断处理器：处理中断请求

程序控制器：控制程序执行

6. 在C语言中，使用标准输入函数scanf接收用户输入的int、char和double型变量时，所用到的输入修饰符分别是：%d, %c和%lf。

%d：用于整数（int）

%c：用于字符（char）

%lf：用于双精度浮点数（double）

二、单选题（每题 2 分，共 20 分）

1. 在一个局域网内，把各台计算机连接在一起的设备是

- A) 集线器 B) 防火墙 C) 调制解调器 D) 以上均不是

如果答案写交换机就没有歧义了。集线器确实能连，但是淘汰买不到了。

Which device is used to connect multiple computers in a LAN? 2 POINTS

A Firewall B Switch C Modem D Router

<https://www.gauthmath.com/solution/1803567712329734/Which-device-is-used-to-connect-multiple-computers-in-a-LAN-2-POINTS-A-Firewall->

Answer B) Switch.

Explanation

The device used to connect multiple computers in a LAN is a Switch.

What is a device that connects multiple devices on a LAN?

<https://www.quora.com/What-is-a-device-that-connects-multiple-devices-on-a-LAN>

There are several choices, but the one most commonly used that is both practical and cheap for the home is a small router. It is more capable than the other options, and theoretically more expensive, but because they are so common and widely manufactured, small routers for home use are actually pretty cheap. In addition to store-and-forwarding packets, it is common for a router to include security functions and often a WiFi access point.

The nominally cheaper option is a switch. A switch has less capability but does the basic job of storing-and-forwarding packets on each of the connected ports according to the destination address. Over time, “switches” have added functionality, and the line between a router and a switch is somewhat blurred. Since it takes so little additional effort to make a switch into a router, most consumer device manufacturers don’t bother making switches. In high-volume commercial environments, a switch can be faster.

An older device which is rarely used anymore is the hub. A hub is little more than a connection sharing device. A bit like a power strip for electric power. The problem is that each device connected to a hub *shares* the same link. This allows collisions, which is how Ethernet was originally designed to work. But using switches and routers keeps all the traffic separate and enables each device to get the full bandwidth the media allows - so nearly everyone changed to switches and routers.

2. 在计算机软件系统中，最基础、最核心的软件是

- A) 数据库 B) 办公软件 C) 操作系统 D) 设备驱动程序

3. 在计算机硬件系统中，连接主板和外部设备之间的硬件设备是

- A) 总线 B) 芯片组 C) 内存 D) 适配器

适配器（如网卡、声卡等）用于连接主板和外部设备。

4. 下列选项中，哪一个是有效的IPv4地址

- A) 8.200.256.3 B) 12.34.56.78 C) 162.-105.3.1 D) 123.458.789.0

5. 下列存储器中，存取速度最快的是

- A) 硬盘 B) 内存 C) U盘 D) 高速缓存

6. 下列存储器中，切断电源后，其中存储的信息会丢失的是

- A) 硬盘 B) 内存 C) 光盘 D) 软盘

7. 下列选项中，不是C语言中合法的变量名称的是

- A) 3com B) _3com C) com3 D) com_3

8. 假设x, y是两个整型变量，则执行完语句“y = x++;”后，

- A) x与y的值相等 B) x的值大于y的值
C) x的值小于y的值 D) 以上三种情况皆有可能

x++是后置递增运算符，先赋值后递增。

9. 1GB代表的字节数量是

- A) 2^{10} B) 2^{20} C) 2^{30} D) 2^{40}

10. 表达式5/2的类型是

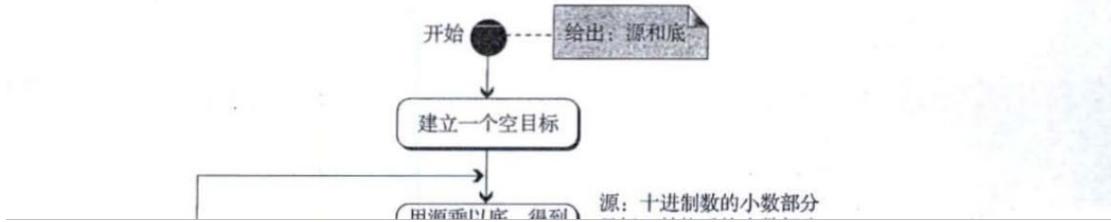
- A) int B) float C) double D) char

三、计算题（共 20 分）

1. 数制转换 (6分) (每小题2分)

$(12.321)_{10} = (20.153)_6$ 要求精确到小数点后3位。

当创建了足够的位数后停止。



分别处理整数部分和小数部分。

整数部分转换

对于整数部分12，用短除法来转换：

$$\begin{array}{r} 1 \quad 12 \div 6 = 2 \text{ 余 } 0 \\ 2 \quad 2 \div 6 = 0 \text{ 余 } 2 \end{array}$$

所以， $12_{10} = 20_6$ 。

小数部分转换

对于小数部分0.321，采用乘以基数（6）的方法，每次取整数部分作为结果的一部分，直到达到所需的精度或循环为止。我们将进行四次转换以确保能够舍入到第三位小数。

1. $0.321 \times 6 = 1.926$, 取整数部分1;
2. $0.926 \times 6 = 5.556$, 取整数部分5;
3. $0.556 \times 6 = 3.336$, 取整数部分3;

因此, 0.321_{10} 约等于 0.153_6 (精确到小数点后三位)。

结合整数部分和小数部分

结合整数部分和小数部分的结果, 得到:

$$(12.321)_{10} = (20.153)_6$$

$$(32)_5 = (122)_3$$

将2进制数 (符号: 0、1) 转换为32进制数 (符号: 0、1、...、9、A、B、...、U、V)

$$(1001101110001101101110110100)_2 = (4RHMTK)_{32}$$

2. 二进制数算术运算 (2.5分)

$(1001001 / 101 = 1110.10)$ 或者 (1110.1001) (循环)

3. 二进制数逻辑运算 (4.5分) (每小题1.5分)

1. 逻辑与:

$$11\ 1011\ 1001\ 0111 \& 01\ 0110\ 1110\ 1011 = 01\ 0010\ 1000\ 0011$$

2. 逻辑或:

$$11\ 1011\ 1001\ 0111 | 01\ 0110\ 1110\ 1011 = 11\ 1111\ 1111\ 1111$$

3. 逻辑异或:

$$11\ 1011\ 1001\ 0111 ^ 01\ 0110\ 1110\ 1011 = 10\ 1101\ 0111\ 1100$$

4. 小明购买了一台新的计算机, 该计算机的CPU和内存之间的地址总线宽度是42位 (bit), 请问, 小明的这台计算机最多可以使用多大的内存? (2分)

$$2^{42}\text{Byte} = 2^{12}\text{GB} = 4\text{TB}$$

地址总线宽度确实直接决定了能够寻址的内存空间大小, 并且通常是以字节为单位进行寻址的。

对于一个具有42位地址总线的系统来说, 它理论上可以寻址 2^{42} 个不同的地址, 每个地址对应一个字节 (Byte)。

为了转换成更常见的单位, 需要知道:

- 1 KB (千字节) = 2^{10} Bytes
- 1 MB (兆字节) = 2^{20} Bytes
- 1 GB (吉字节) = 2^{30} Bytes
- 1 TB (太字节) = 2^{40} Bytes

5. 有一种影像，人们对它的要求很高：影像的播放速度是每秒32帧图像，每帧图像的分辨率为 2048×1536 像素，其颜色系统是512色；而声音的采样频率则要达到 65536Hz ，采用双声道，每声道用4字节（Byte）存储采样值。请问，要保存10分钟这种原始影像，需要多大的存储空间？（5分）

每秒钟影像：

- 颜色编码：

$$2^5 * 2^{11} * 3 * 2^9 * 9 / 8B = 4 * 3 * 9 \text{ MB} = 108 \text{ MB}$$

- 每秒钟声音：

$$2^{16} * 2 * 4B = 0.5 \text{ MB}$$

10分钟：

$$10 * 60 * (108 + 0.5) \text{ MB} = 65100 \text{ MB} \approx 63.6 \text{ GB}$$

列出正确的计算式子，即可给满分，其余视具体情况酌情给分，不超过3分。

如果考虑对512种颜色进行编码需要2进制的9位，8位一个字节，9位必须用到第二个字节，所以每个像素用两字节存储。颜色编码

$$2^5 * 2^{11} * 3 * 2^9 * 2B = 4 * 3 * 9 \text{ MB} = 108 \text{ MB}$$

四、问答题（共30分）

1. 列举出至少3种计算机在医学上的应用。（6分）

- 医疗信息管理：计算机用于管理患者的医疗记录、病历、处方等信息。
- 医学专家系统：计算机系统用于辅助医生进行诊断和治疗决策。
- 医学信息处理：计算机用于处理医学图像、信号等数据，辅助医生进行分析和诊断。

随便列举3个都可以，不限于课程上讲授的那些应用。（答对1个给2分）

2. 计算机刚刚发明的时候程序员直接使用机器语言编写程序，后来人们发明了汇编语言，再后来随着技术的进步，又发明了高级语言（如：C、C++、Java等）。请思考并解释，到底是什么动力推动人们不断的改进编程语言？换言之，编程语言的不断进化带来了什么好处？（6分）

编程语言是程序员与计算机之间交流的语言，既要为程序员掌握，以描述需要计算机处理的问题，同时又要能够被计算机识别（或者能够转换成计算机能够识别的形式）从而被计算机执行。早期的编程语言（如：机器语言、汇编语言）更贴近计算机的认知方式，更容易被计算机处理，但不易于被人所掌握；随着技术的发展，软件的规模和复杂度日益增长，对程序开发维护的效率要求更高，因此出现了一些更加适应人类思维方式的编程语言。

编程语言的不断进化使得程序员能够以人类更加熟悉的思维方式编写程序，从而提高程序开发的效率和质量。

（此题两个要点：1，从机器语言到汇编再到高级语言，语言越来越接近人类思维方式；2，这种演变是为了提高软件开发的效率和质量。建议这两个要点答对一个就给4~5分，两个都答对给满分。）

3. 简要描述 CPU 的中断处理过程，并且解释为什么 CPU 需要设置中断处理的功能？（6 分）

CPU 的中断处理部件用于处理临时出现的紧急事情，如鼠标移动。

中断处理过程：发现中断信号，程序控制部件暂停正在运行的程序，保存该程序的运行现场（当前所有执行状态信息），以便其恢复执行；根据中断信号从特定位置启动中断处理程序（操作系统提供）处理中断。中断处理完毕后，回到原来的程序工作轨道继续工作。

（此题两个要点：1，中断处理是用来做什么的；2，中断处理的过程。答对要点 1 给 2 分，要点 2 给 4 分）

4. 大多数计算机中都装备有硬盘、内存和寄存器三种数据存储设备。（12 分）

1) 按存储容量从大到小的顺序排列这三种存储设备；（3 分）

硬盘、内存、寄存器

2) 按数据访问速度从快到慢的顺序排列这三种存储设备；（3 分）

寄存器、内存、硬盘

3) 既然三种数据存储设备的功能相同——都是存储数据，那么为什么大多数计算机需要同时装备这三种设备，而不是干脆使用一种设备呢？给出你的解释。（6 分）

三种不同的存储设备，有的容量大，却速度慢，有的速度快，却容量小。计算机将那些常用的、或者当前需要使用的数据、程序放到速度较快的存储设备中，以提高计算机整体的计算速度；而那些不常用的或者当前不需要立刻使用的数据、程序则可以存储在速度较慢的大容量存储设备中；另外，操作系统制定了专门的策略实现不同存储设备中数据的互换。计算机采用分级的存储结构，使得不同类型的存储设备可以实现性能上的互补。

（此题前两问是客观题，每个 3 分；第三问 6 分，主要是想考核学生是否理解“分级存储”的含义，请酌情给分）

五、程序题（共 20 分）

1. 求 1 到 100 的平方和

请补充下面的程序，完成计算 1 到 100 的平方和。（4 分）

```
1 #include <stdio.h>
2
3 int main() {
4     int i, sum = 0; // 1分【1】
5     for (i = 1; i <= 100; i++) { // 3分【2】
6         sum += i * i;
7     }
8     printf("%d", sum);
9     return 0;
10 }
```

其他正确形式也可以。“;”写成“,”（扣 1 分）。边界条件不完全正确（扣 1 分）

2.交换两个整数变量的值

下面的程序把输入的两个整数交换顺序后输出（例如，输入的两个整数为 23 43，则输出的两个整数为 43 23），请补充缺失的完成变量 a 和变量 b 交换的 3 行程序。（3 分）

```
1 #include <stdio.h>
2
3 int main() {
4     int a, b, t;
5     scanf("%d%d", &a, &b);
6     t = a; // 1分【1】
7     a = b; // 1分【2】
8     b = t; // 1分【3】
9     printf("%d %d", a, b);
10    return 0;
11 }
```

3.说明程序的含义

请阅读下面的程序，简要说明程序的含义，并对下面两组给定的输入，分别写出程序执行后的输出。（4 分）

```
1 #include <stdio.h>
2
3 int main() {
4     double input, sum = 0;
5     for (int i = 0; i < 5; i++) {
6         scanf("%lf", &input);
7         sum += input;
8     }
9     printf("%.2lf", sum / 5);
10    return 0;
11 }
```

输入一： 7 8 9 10 11

输入二： 3.1 5.3 2.1 2.8 9.12

含义：求五个数的均值（2 分）

输入一的输出： 9.00（1 分）（小数点位数不正确，扣 0.5 分）

输入一的输出： 4.48（1 分）（小数点位数不正确，扣 0.5 分）

4.说明程序的含义

请阅读下面的程序，简单说明这个程序的含义，并对下面两组给定的输入，分别写出程序执行后的输出。（4分）

```
1 #include <stdio.h>
2
3 int main() {
4     int s, x, i, n;
5     scanf("%d", &n);
6     s = 0;
7     for (i = 0; i < n; i++) {
8         scanf("%d", &x);
9         if (x > s) {
10             s = x;
11         } else if (-x > s) {
12             s = -x;
13         }
14     }
15     printf("%d", s);
16     return 0;
17 }
```

输入一： 5 76 2345 -223 24 -87

输入二： 10 6 2 -2 4 -7 5 -3 -2 8 -9

含义：读入 n 个整数，输出绝对值最大的数的绝对值。（2分）

输入一的输出： 2345 （1分）

输入一的输出： 9 （1分）

5.编程题

请按题面要求写一段简单的完整的 Python 程序代码（5分）

编写一个完整的 Python 程序，求三个数中的最大值（仅使用比较和分支判断）。

输入：3个整数，整数之间以一个空格分隔；

输出：一个整数，即3个整数中数值最大的整数值。

例子输入：

23 25 17

例子输出：

25

```
1 def find_max_of_three():
2     # 定义三个变量存储输入的整数（1分）
```

```
3     num1, num2, num3 = map(int, input().split())
4
5     # 使用了正确的输入输出形式 (1 分)
6
7     # 比较并找出最大值 (使用了 if-elif-else) (1 分)
8     if num1 >= num2 and num1 >= num3:
9         max_num = num1
10    elif num2 >= num1 and num2 >= num3:
11        max_num = num2
12    else:
13        max_num = num3
14
15    # 输出最大值 (结果正确) (2 分)
16    print(max_num)
17
18    # 调用函数执行代码
19    find_max_of_three()
```

定义了三个或四个变量 (1 分)

使用了正确的输入输出形式 (1 分)

使用了 if-else if-else (1 分)

结果正确 (2 分)， 其中结果不完全正确 (扣 1 分)