计算概论

Zhen Liu

2021 年 11 月 20 日

目录

1 计算机的基本原理

1.1 从数学危机到图灵机

- 第一次数学危机是无理数的出现，毕达哥拉斯大发现边长为 1 的正方形的对角线长是多少？二百年后用过几何方法避免了这个问题，推动了几何学的发展。但是到十九世纪实数理论建立才正式解决。
- 第二次数学危机是无穷小的概念，微积分的理论建立之上，但是当时无穷小一会儿是 0，一会儿不是 0，像是幽灵。后来借助实数理论尝试解决，但是威尔斯特拉斯构建了处处连续但是不可微的函数，说明必须建立严格的数学分析理论，为不是借助几何，从而催生出集合论。
- 第三次数学危机是罗素悖论，即著名的理发师问题。后来 1931 年哥德尔证明了：任何一个数学系统，只要它是从有限的公理和基本概念推导出来，，并且能从中推出自然数系统，我们就能找到一个命题，既没有办法证明，也没有办法推翻，称为哥德尔不完备性定理。
- 可计算问题：给定函数 f 和其定义域值域，如果存在一种算法，对定义域中任意给定的 x，都能计算出 $f(x)$ 的值，就称函数是可计算的。
- 既然不可能用数学把所有的形式严格表示出来，那我们是不是就放弃研究了呢？我们应该探索的是可计算问题和不可计算问题的边界。那怎么计算边界呢？我们可以为这种计算建立一个数学模型，证明凡是能够这个计算模型能够完成的任务都是可计算的任务。图灵提出了一种图灵机模型。

1.2 图灵机的基本构成

- 图灵机包含一条存储带和一个控制器。控制器包含读写头，可以接受设定好的程序语句，存储自身当前状态，变换状态以及沿着纸带移动。
- 图灵机的工作步骤：首先进行初始化，即存储带上符号和自身状态初始化，并将控制器置放在起始位置。之后就是反复执行如下的循环：读写头读数字或者字母 → 根据自身状态和读到字符找相应的程序语句 → 根据语句执行动作（在当前方格写入/变更自身状态/移动读写头）

1.3 图灵机的运行机理

- 假设纸带上每个格子中的数字是 0 或者 1,控制器本身的状态是 $P1, P2$, 我们的控制语句是 $(P1, 1, 1, R, P1)$ 共 5 个字符，前两个字符是条件，后面三个是动作。条件分别是自身状态和当前读取到的纸带数字，动作分别对应写入，移动，改变状态。这样可以一直运行下去，直到某个状态时，对应的动作是控制器不发生移动，即可结束运行。

- 图灵机如果停机，就说明得出计算结果，说明该问题是可解的。因此给定一个输入 A 之后能否推出 B 的问题，就变成了能不能找到一台图灵机使得给定 A 之后得到 B 呢? 如果能找到这样的图灵机，就说明可以解。

- 实现上述模型的方法除了图灵机还有很多，但是为什么图灵机受到重视? 因为三个特点：简单，强大和可实现。图灵机的意义：（1）可计算性的判定；（2）引入通过读写符号和状态改变进行运算的思想；（3）引入存储区，程序，控制器等原型的概念；

1.4 数的二进制表示

- 计算机为什么能够计算?
  - “数”在计算机中是如何表示的?（必须要存储数据才能探讨计算）
  - 逻辑上” 数” 是如何计算的?（是按照四则运算法则吗还是什么?）
  - 物理上“数”是如何计算的?

- 字母表中的符号越多，虽然控制器的移动次数越少，但是程序指令的数量就越多，因为我们要针对每种当前可能的状态和读取到的符号给出下一步的指令。经过计算，得到字母表中符号的最优数量是欧拉常数 e, 取整为 3，但是与具有两个状态的电子元件比，三个状态的制造更困难，可靠性更低。

- 进制转换：N 进制转换成 10 进制数，只需要将对应位置的数字乘以其进制对应的指数幂再相加即可，注意最后一位乘以的是 0 次幂；而 10 进制转换成 N 进制，除以 N 得余数，直到除数为 0 才算停止，这时候最后得到的余数应该是最高位（因为首项系数对应的指数幂次最高）。

- 二进制转化成八进制，只需从右向左数，每三位数字对应一个八进制数；而转换成十六进制，只需从右向左，每四位数字对应一个十六进制数。

1.5 二进制的布尔运算

- 既然用二进制表示计算机中的数字，那我们如何对二进制数进行计算呢？利用布尔提出的逻辑运算方法，从基本逻辑运算（与，或，非），到复合逻辑运算（同或——两数相同为 1，异或——两数相同为 0）。
- 考虑二进制的加法运算，一种方法是直接利用四则运算进行计算，即逢 2 进 1 并且本位得 0；但是通过观察我们发现，只有当两个数都是 1 的时候，进位才为 1，即进位的计算是“与”运算；而对于本位，只要两个数相同，本位得到的数都是 0，从而本位的运算是“异或”运算，这样不考虑进位的情况下，我们就够造了两个个位数相加计算的“半加器”。当我们考虑把半加器串联起来的时候，我们就可以计算“全加器”。
- 所有的逻辑运算基本上都可以用电路来实现，因此物理上我们通过与门，或门等来实现计算机的电路设计，所以电路是能算数的！

2 计算机的历史与未来

2.1 历史上的计算设备

- 早期手工计算设备（算盘）的缺点是：无法记录计算法则，需要靠人工背口诀；无法设定计算步骤，需要人为完成计算，而不是按照固定的法则。因此工具只是来标记计算结果的。
- 第一台机械计算机：1642 年帕斯卡加法器，只能做加减运算；1673 年，莱布尼兹提出二进制的概念，并且创立了可以进行四则运算的机械计算器，计算结果可达到 16 位；1834 年，英国科学家巴贝齐制造出差分机可计算小数，并且提出分析机的概念，机器分为三部分：堆栈，运算器，控制器，并且企图用蒸汽机提供动力计算；1890 年，统计学家 Hollerith 在进行人口普查时制造了制表机（电子穿孔卡片汇总），之后成立 IBM；1935 年，IBM 制造了穿孔卡片式计算机，能够在一秒中

内计算出乘法运算；1941 年，德国工程师 Zuse 制造出第一台可编程计算机 Z3，使用了大量真空管，每秒可完成 3 到 4 次计算。

- 第一台计算机是 ENIAC（Electronic Numeric Integrator and Computer），由宾夕法尼亚大学两位老师完成，非常巨大，可以完成可编程的运算，但是它不是存储程序式的计算机，并且编程是靠手工插线的方式完成的；1942 年，Atanasoff 实现了 ABC 计算机，计算和存储分离的概念等，因此第一台计算机是有争议的，后来称他为"计算机之父"，但是 ENIAC 仍然是公认第一台计算机；1952 年冯诺依曼发表存储程序控制原理，制造完成 EDVAC，这是世界上第一台存储程序计算机。

2.2 从电子管到云计算

- 早期计算机的目标是如何实现自动的计算？而现在目标是计算如何更快速，更方便，更经济？

- 第一代计算机（1940-1950），使用真空管存储数据，真空管是一种控制真空中电子流动的电子装置，但是它体积大，耗能高还易损坏；第二代计算机（1955 左右），贝尔实验室发明了晶体管，晶体管更小，更便宜，而且在这个阶段出现了操作系统和高级语言 Fortran；第三代计算机（1965 年），Kilby 使用集成电路，把成千上万的真空管或者晶体管压在一个单独的微型芯片上，在此阶段，产生了可移植的操作系统和 C 语言；第四代计算机（1970 年代），使用超大规模集成电路，第一块微处理器是 intel 制造的。

- 计算机的分类：微型计算机；服务器（小型机，中型机）；大型计算机；巨型计算机（银河）。见算计排行榜世界 500 强，但是我们应该关注的是绿色计算，也就是花费相同的能量计算更多的东西，这对能耗有重大意义，因此我们探索新型的计算模式—云计算。

2.3 摩尔定律下的计算危机

- 摩尔定律：在单位面积上晶体管的数目将会翻一番。

- 摩尔定律坚持下去遇到的困难：（1）晶体管大小的限制；（2）电泄漏，影响芯片的计算能力；（3）最大的问题是发热，随着晶体管密度和速

度的增加，芯片会消耗更多的电力，产生更多的热能。

- 为解决发热问题，我们能不能把芯片制造的大一些呢？这样元器件的体积就可以变大，而且还能增加散热；但是这时候我们要注意，我们需要更高的电压来驱动芯片的运转，这可能会产生更多的热量。

2.4 量子计算机的基本原理

- 摩尔定理总有失效的一天，这时候我们要产生新的替代品：量子计算机，生物计算机，DNA 计算机，光子计算机等。
- 1982 年，理查德提出“利用量子体系实现通用计算”的想法，他发现分析模拟量子物理世界所需要的计算能力远远超过经典计算机所能达到的能力，因此我们为什么不能生产一台量子计算机来分析量子系统呢？
- 量子比特可以在某个时刻保持多种状态。经典计算中，2 个比特某个时刻只能存储 1 对 0 或者 1，但是量子计算中，2 个量子比特，某时刻能同时存储 4 对 0 或者 1。同理，对于 n 个量子比特，某个时刻可以同时存储 2^n 个数据。因此在量子计算中，我们可以同时接受 2^n 个输入数据，同时完成 2^n 次运算，输出 2^n 个结果。由于现实中需要对计算过程进行纠错，所以需要多个物理量子比特才能获得一个可容错的逻辑量子比特，因此大约需要 1000 个物理量子比特才能获得超越经典计算机的计算能力。
- 量子比特与外界环境隔离才能保持良好的相干性，但是对于量子计算机制造来说，只有与外界环境良好耦合，才能控制演化并读出结果。因此是实现上还有很大的困难。

3 程序运行的基本原理

3.1 问题的提出

- 由之前的知识，数字利用二进制存储，计算机利用布尔运算实现计算，而布尔运算通过电实现计算。那么我们是不是需要完成什么计算，就手动设计个电路？实际上的计算太多了，这样做太麻烦。那我们能不能设计很多原子电路，需要的时候进行临时组装呢？这就是 ENIAC 了，手动接线。

- 冯诺依曼提出：应该通过某种命令来控制计算机，让计算机按照这种命令执行，这种命令可以用电信号表示；这种命令不是临时输入到计算机，而是存放在某个地方，随时可以更改；命令改了，计算机的功能就改了。这个就是 EDVAC。

3.2 冯诺依曼式计算机

- 冯诺依曼计算机：包含输入设备，输出设备，存储器，运算器，控制器，和总线。控制器统一指挥并控制计算机各部分协调工作；运算器对数据进行逻辑或者算术运算；存储器存储待操作的信息与中间结果，包括机器指令和数据；所有设备都和总线相连。

- 工作过程：

  - 在控制器的指挥下，从存储器上取出指令（控制器中有指令记录员，它告诉控制器应该取出的是哪一条指令；指令是一串二进制数，包含一个操作和操作执行的数据对象）；
  - 控制器中的指令分析器对指令进行分析，弄明白具体完成的计算操作以及需要的操作的数据；
  - 控制器控制从存储器中取出待计算的数（可发现存储器中存储的不但有指令还有数据）放到运算器中，同时告诉运算器需要执行的运算；
  - 运算器完成运算，得到计算结果
  - 将结果输出到存储器或者是输出设备

- 上述理论上的构造在实际实现的时候，运算器，控制器和存储器的一部分都被集成在 CPU 中，存储器包含在 CPU 中的是高速缓存，存储器中还有内存和外存。一般内存是插槽的位置，而外存是硬盘和光驱的位置。键盘鼠标是输入设备，打印机是输出设备。

3.3 存储器的种类与特点

- 一个位 Bit 能够表示 0 或者 1，一个字节 Byte 包含 8 个位。字节是我们在程序中能够控制的最小内存单位。其中存储器换算是 $1TB = 1024GB, 1GB = 1024MB, 1MB = 1024KB, 1KB = 1024Byte$。

- 计算机中存储器的种类：寄存器（用于存放待操作数和结果），高速缓存（通常在 CPU 内部，用做数据缓冲区），内存（CPU 想放但是放不下的部分），外存。
- 寄存器工作速度与 CPU 运算部件节拍一致，一次存取数据大约是零点几纳秒。寄存器与运算部件直接连接，运算部件直接对寄存器进行读写操作；寄存器的制作成品较高，一般 CPU 中只配备有少数的寄存器。
- 主存储器就是我们现在通常说的内存，主要存放 CPU 中的运算数据，存放与硬盘等外部存储器交换的数据。用于临时存放，断电易丢失，并且价格低，易于更换，工作频率越来越快。
- CPU 在访问数据的时候，会优先访问寄存器，之后是 CPU 内部的高速缓存，然后是 CPU 外部的高速缓存，之后是内存，然后是外存。当在寄存器中找不到数据时，就会在 CPU 内部的高速缓存中找，找到的话就会把相邻的区域块整个加载到寄存器中，这是利用了局部性原理。
- 局部性原理：CPU 对数据的访问通常具有一定的局部性。时间局部性：一个内存地址正在被使用，那么在近期他很有可能会被再次访问；空间局部性：在最近的将来可能用到的信息很可能与当前使用的信息是相邻的。

3.4 存储器的原理与类型

- 存储器为什么能保存数据呢？利用三极管，短的一端如果是高电位，那么就是导通的，相反则是断开的。利用两个三极管可以组成一个双稳态触发器用来保存数据。将多个这样的电路连接起来构成电路板，利用行地址编译器和列地址编译器，连接读写控制电路，我们就能选择性的读取数据。
- 存储器的类型分为可读可写的 RAM(Random Access Memory) 和只能读不能写的 ROM(Read Only Memory) 两种。其中 RAM 又分为 DRAM(Dynamic RAM) 和 SRAM，两种方式都可以随机存取，其中 DRAM 虽然必须周期性的刷新以保持存储内容，但是其具有更快的存取效率。关于 ROM，随着发展，现在也能写入东西，比如（Flash EPROM）快速可擦除编程只读存储器，就是优盘。

- 内存的发展主要有如下的几个阶段：
  - EDO DRAM(Extended Data Out DRAM): 扩展数据输出动态存储器，就是把数据发送给 CPU 的同时去访问下一个页面，从而提高工作效率，与 CPU 时钟同步。
  - SDRAM(Synchronous DRAM): 同步动态存储器，工作在 CPU 外部总线的频率上。
  - DDR(Double Data Rate SDRAM): 双数据输出同步动态存储器，从理论上可以把 RAM 的速度提升一倍，它在时钟周期的上升沿和下降沿都可以读出数据。
- 32 位的计算机最多拥有 4G 的内存，因为最多有 2^{32}=4G 的内地址可以被管理。

3.5 CPU 指令的执行

- 首先不是所有的指令都是可以被 CPU 执行的。CPU 中用来计算和控制计算机系统的一套指令的集合我们称为是指令集。它是在 CPU 设计时就预先定义好的，是衡量 CPU 性能的重要标志。现在常见的指令集有 Intel X86,ARM 指令集。CPU 能接受的指令表现为二进制码，其长度随 CPU 类型不同而变化，包含指令码和操作数。其中指令码说明要做的动作，而操作数是指要操作的数或者地址。注意操作数可以没有，也可以有多个，具体需要看指令是干什么的。
- CPU 运算器包含的主要部分有：数据寄存器用来缓冲需要运算的数据（MDR），算术逻辑单元 ALU 用来专门执行算术和逻辑运算，累加寄存器（AC）用来暂存 ALU 的计算结果，还有状态条件寄存器（PSR）用来存储 ALU 运算结果和系统工作状态信息。
- 控制器中包含程序计数器 PC 用来存放下一条指令的地址，而指令寄存器（IR）用来存放当前正在执行的指令，地址寄存器（MAR）用来存放要访问的主存地址。还有指令译码器和操作控制器，时序控制器。

3.6 程序的执行

- 程序可以用高级语言书写，但是在执行的时候，首先要进行编译，变成汇编代码，之后再转化成机器码，就可以被 CPU 执行。

- 一句程序有可能转换成为多句指令；在控制器的协调下连续，依次执行相应的指令；程序执行过程是在内存中完成的，在程序执行过程中，在内存中的不同区域，存放代码和相关的数据。

4 感性认识计算机程序

4.1 说在前面的话

- 学习建议：训练得技能；须抓大放小；多练简单题；选一本薄书。第一位的永远不要考虑程序执行的效率，而是说把程序写出来能够运行即可。

- 编写程序的两件事：我们应该说些什么？以什么样的形式说？举一个例子说明：给出 10 个数求出里面最大的数字。我们把人的大脑当成计算机，那么我们完成这件事需要做什么，计算机程序也是一样的逻辑结构，只不过是换了一种语言来表达。

- 当我们考虑设计程序语言的时候，要考虑如下的问题：（1）是不是编程语言里的所有“单词”计算机都能明白？（int，for 等）（2）是不是无论我们写什么“符号”和“数”，计算机都能明白呢？（小数，等号等）（3）关于编程语言的“句式”多少才够呢？（for，if）

- 关于上述三个问题的回答：(1) 计算机能够认识的单词就三十多个，但是通常我们的程序可能包含很多字符，这是因为由这简单的三十多个就可以产生或者使用其余的；(2) 计算机只能看懂某些类型的数据，这些“数据的类型”和相应的“操作符号”是预先定义好的。数据类型包含：基本数据类型（布尔型，字符型，整型和实型），以及自定义数据类型（数组型，指针型，构造数据类型，类，空类型）。所能理解的操作符号包含：sizeof，下标运算符 []，赋值 =，算术 +，关系 <，逻辑 &，条件?:, 逗号，位运算 »，指针运算 *, 强制类型转换运算 ()，分量运算.→。

- 程序设计语言能够识别的句式只有三种，条件，顺序和循环。

4.2 快步走近 C 程序

- 相关的程序存储在文件夹 cpp 中的 chap04。这里只说明一些注意事项:
  - 所有的程序都要三行，一行把库包含进来，一行 using，一行定义返回值类型 int main。注意还需要 return 0 一般情况下。
  - 所有用到的变量都需要指定类型，尤其是循环变量，并且在指定类型的时候记得赋处置，这样可以避免出现一些未知的错误。
  - 数组的第一个地址是 0，长度为 10 的数组最多能调用 a[9]. 可以利用大括号来表示作用范围，尤其是对于 for 和 if 这种语句。另外，我们可以用 break 结束语句的执行。
  - 添加注释方便程序被阅读，合理使用缩进，简洁直观。

4.3 配置编程环境

- 对于 windows 来说一定记得添加环境变量，另外我们需要下载 mingw 的编译器。
- 几种调试方法:
  - 利用 cout 将程序的中间变量输出显示;
  - 利用断点进行调试;

5 从现实问题到计算机程序

5.1 解决方案到程序

- 计算机并不能帮助我们思考解决问题的方法，它只是一个执行者，我们必须高速计算机它要完成什么。所以当我们考虑一个问题的时候，比如切饼问题，要先想出解决办法，再进行编程。
- 从解决办法到程序还差一个描述，就是用计算机的语言把解决方法描述出来。因为我们现在考虑的是结构化编程，因为我们应该按照“先粗后细，先抽象后具体”的方法，对所要描述的解决方案进行穷尽分解，直到分解为顺序，分支，循环三种结构。

- 切饼问题（用 n 条直线最多把一个饼分成多少块），我们从大的框架下考虑无非是如下的结构：输入刀数 → 初始化第 1 刀 2 块 → 累加第 n 刀增加 n 块 → 输出总块数。所有的计算问题，我们都应该先从大的框架进行问题的考虑，再深入细节进行思考。

5.2 体验结构化的程序

- 本章节相关的程序见 chap05。
  - 鸡兔同笼问题，主要是分类讨论
  - 百元买百鸡，主要是穷举判断
  - 奇偶排序，利用数组分成奇偶两种，再分别使用选择排序
- 写程序的过程应该按照由大到小，由粗到精，由抽象到具体的方法分析，编写程序
- 程序的结构应该由若干个模块组成，模块之内应该是高内聚，模块之间是低耦合。这样当每个模块所执行的功能都只有一个的时候，我们可以很方便的进行程序的调试。这就是结构化程序设计的基本思想。

6 理性认识 C 语言程序

6.1 C 语言的由来

- 1954-1956 年提出了 FORTRAN（Formula Translation），1960 年提出了 Algol，具有里程碑式的意义，1967 年剑桥大学教授提出了 BCPL(Basic Combined Programming Language), 1970 贝尔实验室发现了 UNIX，1972-1973 年完善提出了 C 语言，并且利用 C 语言重写了 UNIX。
- 1979 年贝尔实验室开发了 C++(C with Classes);1985 年，Bjarne 博士完成巨著《The Programming Language》，1998 年 ISO 颁布设计标准，2011 年发布 C++2011.

6.2 C 语言的标准

- C 语言标准历史：

  - KRC 提出《The C programming Language》，一直被广泛作为 C 语言事实上的规范；
  - 1989 年国际标准定义了 ISO 标准
  - 2000 年三月，WG14 小组在上述基础上改进推出 C99

- C 语言规范定义的非常宽泛，比如 long 型数据长度不短于 int 型，short 型不长于 int 型。经常导致相同的程序在不同的编译器上具有不同的解释，相同的程序在不同的平台上运行结果不同，比如整型变量定义，浮点数计算精度，对 ++ 的解释等；

- 如何学习一门语言？任何一门语言无外乎以下四个部分：数据成分，运算成分，控制成分，和传输成分。下面将从如上方面进行学习和介绍。

7 C 语言中的数据成分

7.1 变量数据类型

- 把内存想象成一串格子，每行有 8 个单元，对应一个地址，每个单元可以存放一个 2 进制的数。我们在对变量进行初始化的时候，计算机完成了两个步骤：（1）开辟内存地址，一般是连续的几个格子；（2）把变量名和这串格子的首地址关联起来。当我们再次调用的时候，计算机会根据变量的名字联系其对应的首地址，从而找到内存空间。

- 整型变量分为三类：基本型 int（32bit），短整型 short（short int，16bit），长整型 long（long int，32bit）。注意上节课讲的 C 语言标准很宽泛，只要求长型变量内存不小于 int 型即可。但是在不同编辑器中可能有不同的规定，我们可以用 sizeof 来输出显示占用字节。刚才对应的几种类型，都可以再分为 signed int，unsigned int 型的两种情况，显然无符号的整型不能表示负数。

- 以 int 型变量为例，123 作为无符号整数进行存储时占用 32bit，而-123 作为有符号整数进行存储时占用 31bit，这是因为它的第一个 bit 需要用来存储正负号，0 表示正数，1 表示负数。无符号整数和有符号整数都以补码形式存储，但是正数的补码就是其原码，而 123 的原码和-123 的补码差别是取反加一，也即 123 的原码取反加一得到-123 的补码，-123 的补码取反加一得到 123 的原码。

- 无符号的最大数约为 42 亿，有符号的最大数约为 21 亿。当最高位是 1，其他位是 0 的时候，最高位既表示负号，也表示整数最高位为 1，对应的数约为 21 亿，从而无符号的最小数也是负的 21 亿。

- 浮点型变量分为三类：float（32bit，有效位 7 位），double（64bit，有效位 15 位），long double（64bit）。当我们用 cout 进行输出的时候，默认输出 6 位有效数字，我们可以用 cout«setprecision(100)«a»ednl 来进行输出，但是一旦设定不发生改变，之后的输出都将继续维持这个输出格式。

- 浮点数的范围以 float 为例，共 32bit，其中第一个 1bit 是符号位，之后的 8bit 表示的是指数位，指数位中的第一个 bit 表示的是指数的符号，从而指数的最大范围是 $log_{10}(2^{127}) = 38.23$, 而之后的 23 位表示小数部分，由于采用的是科学计数法，首项非 0，因此我们实际上约等于用 24bit 其中第一个 bit 取值为 1 来表示，从而小数部分 $log_{10}(2^{24}) = 7.225$，因此有 7 位有效数字。

- 除了常见的字符，我们还应该注意转义字符，即在字母前面加上反斜杠。常用的有 n 换行符，a 响铃，双斜杠表示斜杠。具体的可以看转义字符的表。

- 布尔型虽然只表示 0 或 1，但是其在内存中仍然占用一个字节，因为字节是计算机系统可以操作的最小单位。另外布尔型的值只能为 0 或者 1，当赋值给 0 或者假的时候，就存储 0，其他情况非零数或者真表达式等都存储 1.

7.2 输入输出

- 负数补码的求法：先确定符号位，求出绝对值的原码，对原码各位取反，再加一。注意绝对值一定是存在的，因为负数的范围相对较小。想看到一个数在计算机中的存储方式，可以直接打印输出，由于人们比较熟悉 16 进制数的 2 进制表示，因此我们直接打印 16 进制输出即可：cout«hex«a«endl。注意对于负数的 2 进制数，输出的时候只是把计算机中的表示每 4 位作为 16 进制数输出，跟原码补码无关。注意 hex 是 16 进制，oct 是 8 进制，dec 是 10 进制。赋值的时候可以用 a = 0x,a = 0 分别表示 16 进制和 8 进制的输入。

- 当我们用 float 型进行计算的时候，虽然能表示的数据范围很大，但是由于只有 7 位有效数字，因此我们应该避免将一个很大的数和很小的数直接相加或者相减，否则就会“丢失”小的数。
- 字符型变量只占用一个字节，因此我们最多表示 256 个字符，范围是 0-255。由于字符型是转换成整型变量再进行存储，因此我们可以与整数数据相互赋值，并且和整数一样进行运算。需要注意的是，当我们的赋值超过 256 的时候，系统会自动截取（低位截断）其 8 个字节的表示来进行赋值。
- 除了可以设置精度，还可以设置输出宽度，比如 setw(3), 设置字符占据的宽度是 3。

7.3 常数和变量命名

- 常量是在程序运行中值保持不变的量，一种是字面常量，就是数字；另一种是符号常量，用一个标识符代表一个常量的，称为符号常量。可以通过命令 const double PI = 3.1415926 来定义符号常量，在程序运行中 PI 的值不回改变。
- 我们在对常量进行赋值的时候是可以指定类型的。浮点型常量默认为 double 型，比如整型常量我们可以写成 n=10000L，表示长整型常量，如果是 m=-0x88abL，表示长整型十六进制常量，浮点型也可以类似指定，比如 x=3.1415F 表示单精度浮点型常量，y=3.1415L 表示长双精度浮点型常量。
- C++ 规定：标识符只能由字母，数字和下划线三种字符组成，且第一个字符必须为字母或下划线，且不可以与保留的关键字相同。
- 驼峰命名法：（1）由一个或多个单词连结在一起（2）第一个单词以小写字母开始（3）第二个或者每一个单词的首字母都采用大写字母

8 C 语言中的控制成分

8.1 分支语句

- if 语句可以多次判断，if，else if，else 构成完成的 if 语句，可以多次使用 else if。另外 if 语句可以进行嵌套，这样可以使的逻辑更清晰。if

语句中的表示式可以是任意的数值类型，可以是字符，指针等，只要是非零的，就认为是真，执行 if 语句。

- 多分支语句可以使用 switch（表达式）的形式书写，内部是 case 常量表达式: 语句; 的形式，最后一行可以设置 default 语句。注意 case 中常量表达式的值只是提供程序的入口，如果不用 break 进行终止，会依次执行之后的语句。需要注意的是 default 语句也可以不放在最后一行，而是根据实际需求换位置，但是其仍然是提供了程序的入口，只是当没有匹配的常量表达式时，就会从 default 开始执行语句。
- C++ 规定：标识符只能由字母，数字和下划线三种字符组成，且第一个字符必须为字母或下划线，且不可以与保留的关键字相同。
- 驼峰命名法：（1）由一个或多个单词连结在一起（2）第一个单词以小写字母开始（3）第二个或者每一个单词的首字母都采用大写字母。

8.2 循环语句

- 除了常见的 for 语句，for 循环语句执行的时候，先赋初值表达式 1，执行程序语句，然后执行表达式 3，之后才是执行表达式 2 进行判断，如果成立，继续执行语句，执行表达式 3，再执行表达式 2 这样循环执行。我们还有 while 语句，但是 while 语句是当条件成立的时候就执行，注意应该在执行语句中添加停止的方法，否则容易陷入死循环。while 语句还有另一种形式 do，while。dowhile 语句至少会被执行一次，而 while 语句第一次执行就需要判断。
- 循环语句之间也是可以互相嵌套的。在循环语句中也可以用 break 跳出循环体，直接结束本层的循环，如果只想跳出某一次，我们可以使用 continue 语句来实现。
- C 语言是支持 Goto 语句的，只要加上标识符，然后在编程的时候 goto 加标识符就可以使用。但是不应该使用，因为 Goto 语句难以阅读并且难以查错。Knuth《The Art of Programming》

9 C 语言中的数组

9.1 数组的介绍

- 数组的定义是：类型数组名 [常量表达式]。再次强调数组下标从 0 开始。注意数组的长度定义中必须用常量表达式来定义，不能使用变量来定义。那我们想更改数组的长度该怎么办呢？一种方法是引入常量 const int 到时候直接修改这个值即可；或者定义一个宏 #define N 4, 宏的定义是在函数定义之前的。另外，如果我们不清楚多长，那么分配尽可能多的空间即可。
- 数组的初始化时如果我们只指定部分元素，比如 a[4]={1, 2}, 那么就是前两个位置分别赋初值，其余的元素值为 0。如果初始化的时候给定元素的个数比能存储的个数多，就会报错，下标越界。
- 我们还可以定义二维数组 a[2][3], 实际存储的时候是先按照行存储，即把二维数组拉成一条直线。进行初始化的时候可以赋初值：

  $$a[2][3] = \{\{1, 2, 3\}, \{4, 5, 6\}\}$$

  由于这样进行循环的时候，要进行两层循环，我们可以这样来使用 for(int i;i<3;i++), 这样的好处是循环变量的作用范围只是第一个 for 循环，保证互相之间是不干扰的。还可以使用如下的方法赋值：

  $$a[][4] = \{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\}$$

  类似的，我们也可以定义三维数组。

9.2 数组的作用

- 用数学进行统计-统计数字出现的次数；
- 用下标做数轴-统计素数，相关代码见 chap09.

10 C 语言中的字符串

10.1 字符数组与字符串

- 当我们对字符数组进行赋初值的时候，如果赋值的字符少于分配的内存，会自动补反斜杠 0，这对应 ASC 码中的第 0 个字符，表示的是终

止字符。如果我们进行初始化如下：char a[] = {'c','h','i','n','a'}, 我们得到的 a 是 5 个元素的，但是如果我们直接用如下的赋值：char a[] = "china", 我们得到的是 6 个字符，其中第 6 个字符是反斜杠 0，也就是所有的字符串都是以反斜杠 0 作为结束的。注意字符串赋值的时候只有在初始化的时候是允许的。

- 当我们进行输入的时候，数据并不是直接就进行被保存在内存地址中去，而是会先存储到缓冲区中，然后会有指针从左向右移动，把缓冲区中的字符存储到内存地址中。另外，利用 cin 进行输入的时候，不会把空格和换行符当成字符来输入，它会把空格认为是分隔字符，把换行符认为是输入的结束处理。

10.2 字符和字符串的输入

- 可以利用 cin.get()，get 是 cin 这个对象的一个函数。利用这个函数可以读入一个字符。具体的见程序 Chap10.
- 当我们利用 cout 输出字符数组的时候，需要加上终止符号，否则就会沿着内存地址一直向下输出，我们可以对二维数组进行输出的时候，我们可以直接输出某一行，即输出 a[1] 即可。这是对于 char 型的数组，但是对于 int 型的数组，如果直接 cout«a，我们会输出其首地址。
- 当我们输入字符串的时候：
  - 直接用 cin 输入字符串，会把空格和换行当成标志符，从而需要 ctrl+z 结束输入；
  - 利用 cin.get() 输入，三个参数分别表示要赋值的字符串名字，要输入字符串的长度，终止字符。如果第三个参数没有指定，默认为反斜杠 0。
  - 利用 cin.getline(), 基本用法和 cin.get() 相同，不同的是 getline 遇到终止标志字符时结束，缓冲区的指针移到终止标志字符之后；而 get 遇到终止字符是停止读取，指针不移动。
- 字符串的练习：（1）统计单词数（2）字符串连接（3）字符串加密。需要说明的是现在只通过字符简单的一对一变换进行加密几乎是无用的，因为在长时间的英文使用中，每个字母使用的频率几乎是相对固定的。

另外我们可以用 c 语言中的 strcmp，strcat 等字符串函数进行剪切赋值等操作。

11 C 语言中的函数

11.1 函数的定义与声明

- C 语言中一些常见的函数：sqrt,pow（求指数幂）,strlen,strcmp,atoi(将字符串转换成相应的整数)。在定义函数的时候，是可以允许函数没有输入参数或者是返回值的 void。但是每个函数必须定义 main 函数，因为 main 函数是一个程序的入口。
- 函数是 C 语言程序的基本构成单位。一个 C 程序由一个或者多个源程序文件组成，而一个源程序文件可以由一个或者多个函数组成。但是当我们在编写 main 函数文件的时候，应该在开头的地方加上 include "max.h" 的命令。如果用双引号，就会优先搜索当前的目录。函数都是有类型的，函数的类型就是其返回值的类型。
- 在一个文件中，如果出了 main 函数之外有其它的函数，我们可以在 main 函数之前定义，如果想在 main 函数之后定义的话，我们应该在 main 函数之前进行声明，也就是用函数原型 = 返回值的类型 + 函数名 + 参数类型。

11.2 函数的调用

- 以函数在程序中出现位置和形式来看，函数的调用可以分为如下三种：
  - 函数调用作为独立语句，例如 stringPrint(), 调用函数完成某项功能，没有任何的返回值；
  - 函数作为表达式的一部分，例如 number = max（A，B）/2；
  - 以实参形式出现在其它函数的调用中，例如 number = min(sum(-5,10),numC);
- 函数的执行过程，当我们执行 main 函数语句遇到调用其他函数情况时，会做三件事（1）初始化 max 函数，即单独开辟一部分内存空间给 max 函数使用；（2）传递参数（给形参赋值）（3）保存当前现场；

当执行完 max 函数之后返回的时候，会做两件事：（1）接收函数的返回值（2）恢复现场，从断点处继续执行；并且调用完成之后，max () 函数将不会再存在。

- 函数参数的传递：(1) 实参和形参具有不同的存储单元，数据传递是值的传递，也就是 copy（2）实参和形参的类型必须相同或者可以兼容。

11.3 变量的作用域

- 全局变量是在所有函数外定义的变量，它的作用域是从定义变量的位置开始到本程序文件结束。而局部变量在函数内或块内定义，只在这个函数或块内起作用的变量。因此我们可以 using 之后就定义变量，比如 int a = 0；如果是在 for 循环中定义的初始变量，那么作用范围只是在 for 循环。

- 当我们想对变量进行互换值的操作等，利用形参的传递是做不到这一点的，因为它会创建额外的变量，进行的操作对原来的变量没有影响。但是我们可以利用全局变量完成上述操作。

- 当全局变量和局部变量同名的时候，局部变量将在自己作用域内是有效的，它将屏蔽同名的全局变量。在非必要的情况下，不要使用全局变量，因为全局变量破坏了函数的相对独立性，增加了函数之间的耦合性，使得函数之间的交互不够清晰。

11.4 数组和函数

- 当我们用数组元素作为参数进行传递的时候，和正常的参数传递是一样的。即用 a[] 作为实参变量进行传递，也是 copy 其中的数据。

- 当我们用数组名做参数进行传递的时候，注意，数组名 a 不是变量，而是常量，是数组在内存中的地址。虽然也是 copy，但是由于是数组，change 函数可以从其地址中修改对应的值，注意与前述区分。具体的可以见 chap11 的代码。

12 C 语言中的运算成分

12.1 赋值运算

- 赋值运算的两边类型不同的时候，要进行类型转换，不管右边是什么，都要转换成左边的类型。当长数赋值给短数的时候，直接进行截断就可以，这里的截断是二进制表示上的截断，有可能正数截断低位之后就成负数了。当短数赋给长数的时候，如果 short 型是无符号的，则高位补 0，如果是有符号的 short，则高位补 0 或 1 完全取决于该值的符号是 0 还是 1；而对于 unsigned int 和 int 之间的赋值，直接一对一赋值，不管是符号位还是数字位。
- 赋值语句本身也是表达式，比如 a=b，而表达式本身的值就是通过赋值符号所传递的值。
- 复合赋值运算，比如 a += 3 等价于 a=a+3；如果有多个运算符在一个表达式中，我们需要判断优先级。如果出现连续的赋值运算，应该是自右向左的结合顺序完成。

12.2 算术运算

- % 是模运算，求余运算必须是整数，比如 7%4 = 3。整数运算的结果仍然为整数，比如 5/3=1；实数运算的结果仍为 double 型的；四舍五入和五舍六入是看具体的编译器的。
- 模运算的优先级和乘除的运算级是一样的，在同一级别中，采取自左向右的方式进行结合。
- 复合赋值运算，比如 a += 3 等价于 a=a+3；如果有多个运算符在一个表达式中，我们需要判断优先级。如果出现连续的赋值运算，应该是自右向左的结合顺序完成。
- ++i 是指在使用 i 之前，先将 i 的值加一。但是 i++ 是在使用 i 之后，再将 i 的值进行加一。注意 ++ 号只能用于变量，不能用于表达式，也就是不能用于 (-i)++, 但是我们可以考虑如下的输出：cout«-i++«endl; 注意此时 i 后的 ++ 优先级高于-号，从而输出-3，i 的值变为 4。但是如果我们考虑 cout«-++i«endl; 此时是同级的，但是应该先运算 ++，从而输出-4，i 的值变为 4.

- 在 vscode 中，如果进行输出 cout 的时候有多个表达式，那我们应该先从最右侧的表达式开始计算，并输出结果，注意这是和我们的认知不同的。

12.3 关系运算

- 等于和不等于的优先级是相对较低的，而大于和大于等于的优先级相对较高。关系运算表达式的值只有两种情况，要不然为真，要不然为假。
- 逻辑非 > 算术运算符 > 关系运算符 > 与或者或 > 赋值运算符。
- 三个逻辑运算符之间也有优先级，非 > 与 > 或。
- 实际上，在逻辑表达式的求解中，并不总是执行所有的运算，只有在必须执行下一个逻辑运算符才能求出表达式的解的时候，才执行该运算符。逻辑运算符两侧可以是任何类型，系统只用 0 或者非 0 来判断。
- 在 vscode 中，如果进行输出 cout 的时候有多个表达式，那我们应该先从最右侧的表达式开始计算，并输出结果，注意这是和我们的认知不同的。

12.4 其他运算

- 逗号运算符，就是用逗号将两个表达式连接起来，注意，逗号运算符在所有的运算符中的优先级是最低的，甚至低于赋值运算符。如果出现多个逗号连接的表达式，会先计算左边表达式的值，但是整体表达式的值是最后一个表达式的值。
- 条件运算符，表达式 1? 表达式 2: 表达式 3。如果表达式 1 的值是真，那么就以表达式 2 的值作为条件表达式的值，否则以表达式 3 的值作为整个条件表达式的值 max=（a>b）?a:b;。
- 强制类型转换符（类型名）（表达式）。注意，强制类型转换后，被转换的量的类型并没有发生变化。
- 位运算是指进行二进制位的运算，仍然是通过字节为单位关于位进行运算的。

12.5 其他运算

- 按位与：参与运算的两个数据，各位均独立进行“与”运算。
- 按位或，按位异或（只有不同的时候才是 1），按位取反，规则与上述相同。
- 左移运算符：将一个数的各个二进制位全部左移若干位，如果溢出就舍弃，如果空出就补 0，如果溢出的不是 0，相当于乘以 2 的指数幂。
- 右移运算符：与上述不同的是，对无符号数，右移时高位补 0，但是对有符号数，符号位应该保持不变。
- 两个不同长度的数进行位运算的时候，应该按照右端对齐。补位的话如果是无符号整数型，左侧补满 0，但是如果是有符号整数型，左侧全部按照符号位补齐。
- 按位与运算可以用来取一个数的某些位，另外使用异或运算可以不需要临时变量，就可以交换两个数的值。只需使用 a =a 异或 b, b = b 异或 a , a = a 异或 b.

13 函数的递归

13.1 递归介绍

- 注意函数是不能嵌套定义的，所有的函数一律平等；但是函数是可以嵌套调用的，无论嵌套多少层，调用的原理都是一样的，开辟内存空间，执行完之后释放并返回值。但是特别的，一个函数能调用它自己，这就是递归调用，注意递归调用和普通调用没有区别!
- 深入理解递归操作：当我们的输出是在自我调用之后，那么输出的值应该等到被调用函数执行完之后返回值，再继续向下进行。

13.2 递归作用

- 完成递推：只要初始值和递推关系即可。
- 模拟连续发生的动作：首先确定连续发生的动作是什么；然后搞清楚不同次动作的关系；确定边界条件。比如汉诺塔问题，连续发生的动

作是移动 m 个盘子，不同次的关系是 m-1 个盘子的移动和第 m 次要移动的柱子不一样，边界条件就是第一次移动。类似还有进制转换。

- 进行“自动的分析”: 先假设有一个函数能给出答案，在可利用这个函数的前提下，分析如何解决问题，搞清楚最简单的情况下答案是什么，就完成了自动化分析，比如放苹果问题。

14 指针

14.1 指针介绍

- 互联网上资源的地址也称为网址就是指向资源的指针。变量的三要素: 变量的地址，变量的值，变量名。通常，把某个变量的地址称为指向该变量的指针。直接利用取址运算符作用在变量名的前面就可以得到该变量名对应的地址。有了地址之后，可以利用指针运算符来访问地址中的元素，因此我们可以 cout« *&c «endl=cout« c «endl;

- 指针变量是不同于指针的另一个概念，它是专门用于存放指针（某个变量的地址）的变量。假设 pointer 是指针变量，并且其存放的值是变量 c 的地址，我们就说 pointer 是指向变量 c 的指针变量。指针变量的定义是: int *pointer; 这里 int 是指针变量的基类型，它是指针变量指向的变量的类型；* 是指针运算符，是 pointer 的类型；而 pointer 是指针变量的名字。赋值的时候 pointer=&c。或者在初始化的时候直接指定 int *pointer = &c，这时候 pointer 所指向的存储单元的内容等于变量 c，并不是变量 c 所代表的内容，而是变量 c 本身，可以当成变量使用。

- 可以利用 NULL 空指针对指针进行初始化。即可以定义指针为 *pointer = NULL。另外，我们后置 ++ 的运算符优先级是最高的，其次是前置 ++ 运算符，逻辑非，*，& 四个同级的运算符，当同级运算符出现的时候，应该先处理离变量最近的运算符。

- 基类型 int *pointer =NULL 中基类型 int 的作用，就是当我们执行语句 pointer++ 操作的时候，虽然是对 pointer 内部的地址进行 ++ 操作，但是由于我们指定了基类型 int，因此 pointer++ 会增加 4 个字节，而不是我们所理解的一个字节，这就体现了基类型的作用。

14.2 数组与指针

- 当我们定义指向数组中的元素 a[2] 的指针时，其实和一般的指向变量的指针时完全一致的，没有什么区别。而对于整个数组来说，注意，数组名 a 相当于指向数组第一个元素的指针，但是数组名 a 不是变量，因此不能对 a 进行赋值。但是如果用于 sizeof 或者取址符，可能会有别的含义，所以用相当于这个词。

- 用指针变量访问数组，可以用 *pointer++ 来访问数组中的第二个元素，假设 pointer 是指向第一个元素的时候。当我们定义数组 int a[10] 以及 int *pointer; 有如下的结论：

  - pointer = a 等价于 pointer = &a[0] ;
  - pointer + i 等价于 a+i 等价于 &a[i] ;
  - *(pointer+i) 等价于 *(a+i) 等价于 a[i] 等价于 pointer[i]

- 利用指针做加减运算的时候一定要注意有效的范围，因为指针变量是可以指向数组最后一个元素以后的元素的。

- 再谈一维数组的几个结论，我们需要知道的，假设 a[4] = 1,2,3,4：

  - 数组名 a 相当于指向数组第一个元素的指针，即 a=&a[0];
  - &a 是指向数组的指针，&a+1 将跨越 16 个字节，所以 &a 相当于把管辖范围上升了一个等级；
  - *a 是数组的第一个元素 a[0]，即 *a=a[0], 所以 *a 相当于把管辖范围下降了一个等级；因此结合上述有 *&a=a 的结论；

- 二维数组的定义：定义一个二维数组比如 a[3][4] 就约等于定义了三个一维数组 a[0],a[1],a[2]，其中每个元素都是一个“包含四个整型元素”的数组，因此二维数组定义中的 a 是指向 a[0] 这个数组的指针，而 a[0] 是指向元素 a[0][0] 的指针；

- 我们可以通过语句 int(*p)[4] 来定义一个指向“包含 4 个 int 型”元素的一维数组的指针变量，这与二维数组中 a 的基类型是相同的。从而我们可以进行赋值操作 p=a, 然后通过 p[i][j] = *（*（p+i）+j）直接指向数组的 (i,j) 个元素。

14.3 字符串与指针

- 字符串与指针和数组与指针是很相似的，但是我们需要注意区分几点：
  - 字符串数组的结尾是反斜杠 0；指向字符串数组的指针仍然是指向字符串数组的第一个元素；
  - 当输出指向字符串数组的指针的内容时，cout«pointer<endl; 会直接把字符串输出，如果同指向数组的指针一样，只需要输出地址的话，我们需要采用命令 static_cast<void>(pointer) 的命令；
  - 可以把字符串直接赋值给指针，比如 pointer=”Hello”, 但是不能通过修改指针去修改 Hello 这个字符串的值；同时我们也可以直接改写字符串指针指向的位置 pointer = buffer, 比如其中 buffer[]=”hello”。

14.4 函数与指针

指针与函数的内容主要包含两个方面：（1）指针用做函数参数，我们考虑如何“限制”指针的功能；（2）指针用做函数返回值，我们考虑其静态局部变量的作用。

- 把指针函数用做参数的时候：
  - 可以实现交换两个元素的值等操作，当我们把指针变量的值作为实参进行传递的时候，实际上传递的是地址，因此辅助函数内可直接通过地址来修改值；
  - 注意实参和形参的基类型需要对应起来，比如传递二维数组指针 a，那要在定义形参的时候，说明指针变量是指向一个数组的指针；
  - C++ 编译器会把数组名的形参直接作为指针变量来处理，比如我们在定义一维数组 a[10] 之后，将 a 作为实参进行传递的时候，可以定义形参 int array[] 来接收，并且在辅助函数内，可把 array 作为指针变量进行计算。
  - 指针作为形参进行传递的时候传递的是地址，从而我们有可能在辅助函数中修改原来数组中元素的值，但是这可能并不是我们希望的，因此我们需要限制这种传递的功能，我们可以在定义形参的时候加上 const，说明其是指向符号常量的指针，从而里面的值就不能被修改。

  - 关于指向符号常量的指针：(1) 不能对其指向的内容进行赋值；(2) 但是可以修改符号常量本身指向的内容；（3）可以将指向符号常量的指针指向一般的整型变量，但是仍然不能通过 *pointer 的方法去修改其指向的内容。

- 当我们定义一个返回指针值的函数的时候，只需要在定义的函数名前面加上 * 号就可以了。但是需要注意的是，当返回的指针指向的是临时变量的时候，这个时候返回的指针指向的是已经被释放的内存空间的地址，因此是非常不稳定的危险的。可以用如下方法避免：

  - 通过定义全局变量，我们可以返回一个处于生命周期中的变量的地址；
  - 返回静态局部变量的地址，而非动态局部变量的地址。只需要在定义变量的时候，前面加上 static 就可以定义静态局部变量。

- 静态局部变量：函数中的局部变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次该函数调用时，仍可以继续使用该变量

15 结构体与链表

15.1 结构体变量

- 经常用一组变量来描述同一个“事物”，比如学号，姓名，性别等，我们可以构造一个新的数据类型——结构体，只需要定义 struct student; 需要特别强调的是，在后边的花括号之后要加上分号；

- 定义结构体变量，除了常见的可以利用结构体类型 student（注意当我们定义完成之后它就成了一个结构体),比如 student student1,student2; 我们还可以在声明类型的同时定义变量，也就是在后边的花括号之后分号之前加上变量的名字。

- 由于在定义结构体中的变量时已经指定了变量的类型，因此系统会为结构体变量分配一个连续的地址空间，按照预定义好的数据类型给分配内存空间，我们可以直接通过 student.name 来访问和改变该地址空间中的值。

- 两个相同类型的结构体之间是可以互相赋值的，赋值的过程实际上是把内容 copy 了一份给另一个结构体；结构体做参数与数组名做参数也是不一样的，结构体做参数相当于 copy 一份内容给对应的函数；结构体变量做返回值时相当于把变量的内容 copy 一份给调用者。
- 定义指向结构体变量的指针：student mike; student *one = &mike; 那么我们可以通过使用 (*one).id 来访问，这里我们引入另一种表示方法 one“横杠大于号”id, 这里“横杠大于号”是指向运算符。借助指向结构体的指针，把指针用作参数，我们可以借助指针来修改结构体中的值，这与普通类型的使用是一致的。另外我们还可以定义结构体数组，即 student myclass[3].

15.2 链表

- 如果我们想在结构体数组中插入一个结构体的时候非常不方便，因为内存地址是预先分配好的连续的地址，这时候我们就可以借助一种新的结构体链表，它的每个结构体之后都会有个指针，指向下一个结构体。
- 链表头：指向第一个链表结点的指针；链表结点：链表中的每一个元素，包括当前节点的数据，下一个结点的地址；链表尾：不再指向其他结点的结点，其地址部分放一个 NULL，表示到此链接结束。
- 链表可以动态的创建，首先我们动态的申请内存空间：
  - int *pint = new int(1024)；delete pint ; 申请的是一个整型内存，初始值为 1024；
  - int *pia = new int[4]; delete[] pia; 申请的是一个数组内存，数组元素个数为 4；

  知道了上述 new 和 delete 的用法，我们就可以用他们来申请指向结构体的指针，从而依次进行这个过程，我们就可以动态的创建链表。
- 链表元素的遍历, 只要最后指针不是 NULL，我们一直进行下去；链表元素删除的话，如果删除的是第一个元素，只需要借助 temp 指针，把 head 指针直接指向第二个元素即可，但是如果是中间元素，要使用 temp 找到被删除元素，follow 指向被删除元素的下一个元素；链表的插入也是类似的。

- 我们还可以定义双向链表，除了第一个链表结点之外，每个链表结点都包含指向前驱结点和后驱结点的指针。当我们进行删除和插入的时候需要更复杂的操作，但是方法是单向结点是类似的。

16 结语

16.1 面向对象编程

- 计算机系统中的程序是现实世界的解决方案在计算机系统中的映射，而映射的过程是通过使用编程语言来完成程序设计的过程，所以编程语言是工具。

- 以学生选课为例，现实生活中参与其中的有学生（数据 + 动作），教师（数据 + 动作），课程计划（数据），除了这些对象之外，还要符合一定的规章制度，这可以通过描述事物之间的关系和规则来确定。面向对象编程就是把这些现实生活中的对象作为计算机编程的对象处理，描述这种关系的语言就是面向对象语言，C++ 就是面向对象语言。