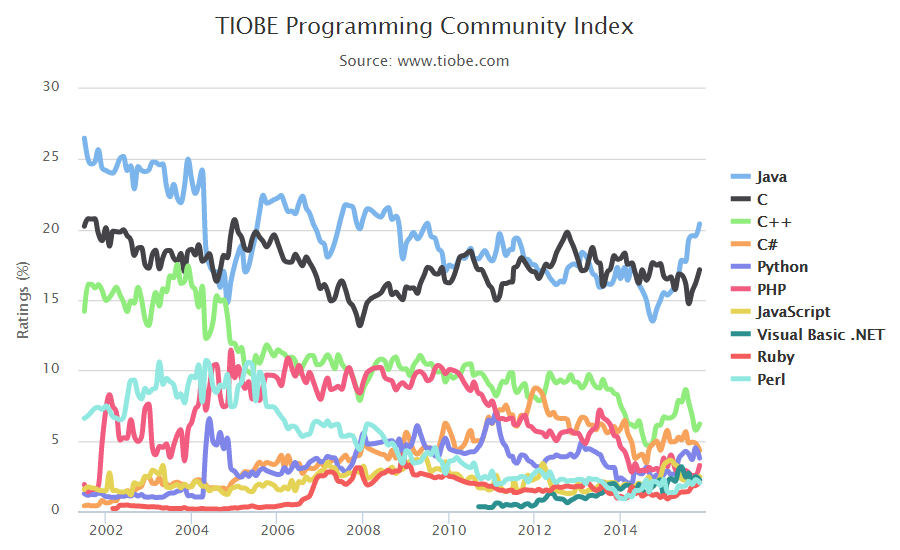
# 第一个程序

## 1. C 语言被淘汰了吗？

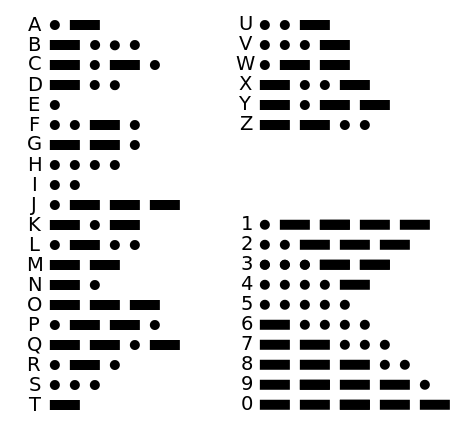
对于其他编程语言来说，C 语言确实是一个老古董了。但却正如古董一样，C 语言是越老越值钱。



从最新的 TIOBE 世界编程语言排行榜，我们可以看到虽然这么多年过去了，但 C 语言一直都是在第一名和 第二名之间徘徊，并且目前仍然保持着上升的趋势。

## 2. 莫斯密码

莫斯密码的原理非常简单，就是查表，这儿是一张莫斯编码表，大家看：



将明文对着编码表翻译为“点横”组合的过程，我们称之为编码，反过来，将“点横”组合解密回原文的过程，叫做解码。

## 3. 机器语言

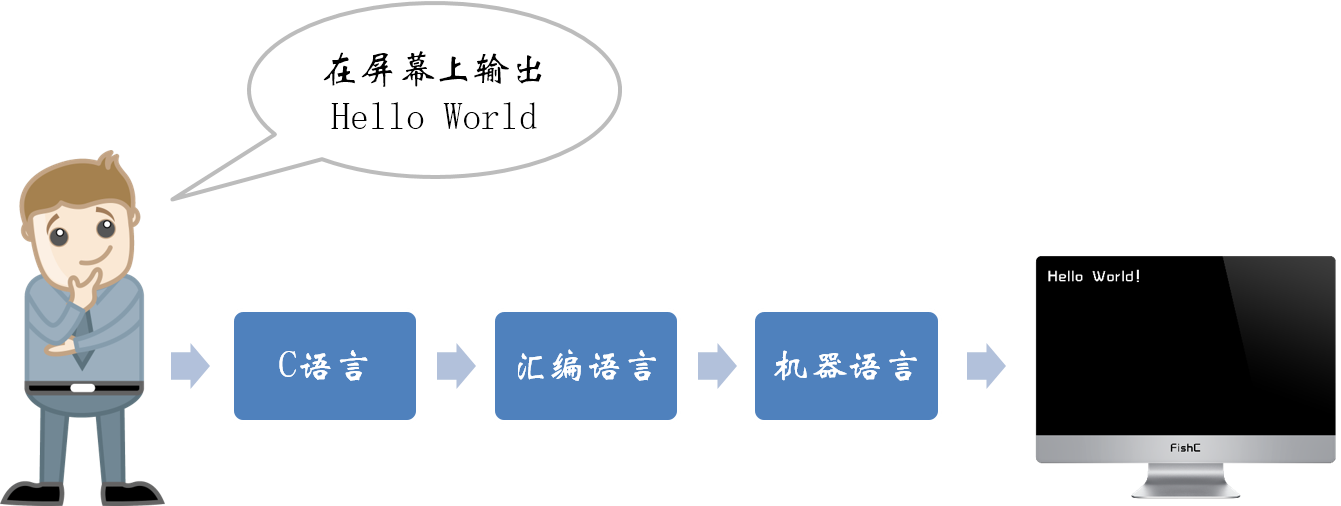
通过查找指令表，CPU 能将 0 和 1 的组合跟具体的指令挂钩，那么这些 0 和 1 的组合我们称之为机器码，也叫机器语言，属于第一代编程语言，也是 CPU 唯一可以直接读得懂的语言。

## 4. 汇编语言

在汇编语言中，引入了大量的助记符来帮助人们编程，然后由汇编编译器将这些助记符转换为机器码，这个转化的过程我们称之为编译。

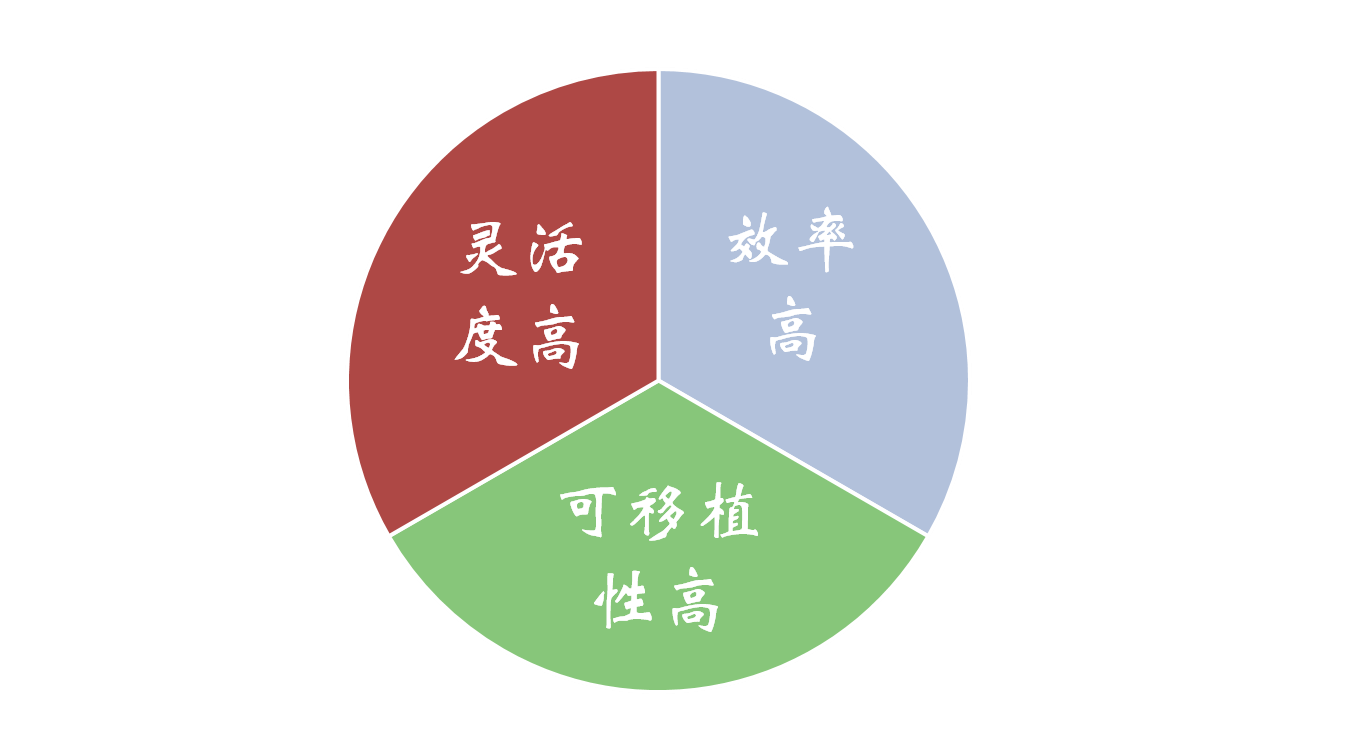
## 5. C 语言

C 语言属于第三代编程语言，第三代编程语言我们称之为高级语言。正如你所听说的 C++、C#、JAVA、Delphi、Python、Object-C、Swift 这些都属于第三代编程语言。

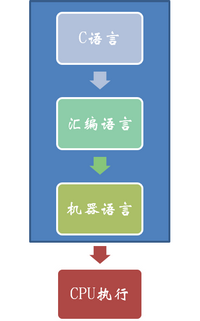


事实上呢，你用 C 语言进行编程，编译器会将你的 C 语言代码编译成汇编语言，再由汇编语言的编译器编译为机器语言，通常我们看到的可执行文件事实上就是机器语言的形式，进而让 CPU 理解和执行。

## 6. C 语言的优势

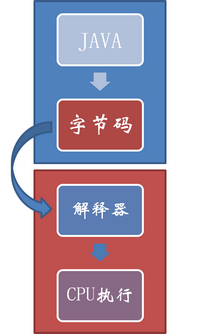


- 效率高



我们说 C 语言效率高是针对其他第三代编程语言来讲的，C 语言是编译型语言，源代码最终编译成机器语言，也就是我们所说的可执行文件，从此 CPU 就可以直接执行。

除了编译型语言，时下很流行的还有解释型语言，像 JAVA，Python，Ruby 这类都是解释型语言。解释型语言不直接编译成机器码，而是将源码转换成中间代码，然后发送给解释器，由解释器逐句翻译给 CPU 来执行。这样做的一个好处就是可以实现跨平台的特性，而缺点就是效率相对要低一些，因为每执行一次都要翻译一次。



打个吃货比较容易懂的比喻，编译型语言就是做好一桌子菜再开吃；而解释型语言就是打火锅，想吃什么，就下什么料。很明显，打火锅总是比较费时间的。

- 灵活度高

C 语言不仅提供多种运算符，还可以完成类似于计算机底层操作的位运算；语法简单，约束少；丰富多变的结构和数据类型；还拥有可以直接操作计算机硬件能力。这一点大家在学习到指针这一章节的时候将深有体会。指针可以说是 C 语言的灵魂，C 语言有多灵活和强大，完全取决于你对指针这一知识点的掌握程度。所以不同的人在学 C 语言，有些成了大神，有些仍然还是初学者。

- 可移植性高

可移植性高是指源代码在不需要做改动或只需稍加修改，就能够在其他机器上编译后正确运行。

统计资料表明，对于单片机来说，不同机器上的 C 语言编译程序 80% 的代码是公共的，因此使用 C 语言的编译程序更便于移植。另外，无论是 Windows 系统、Linux 系统还是苹果的 MacOS 系统，抛开现象看本质，它们都与 C 语言有不可分割的联系。

## 7.彩蛋——Dennis Ritchie



没错，上边这个屌屌的大叔跟乔帮主同一年离我们而去（事实上他们还是同一个月份走的 ）

与世界上大多数名人一样，Dennis Ritchie（下边简称 Ritchie）并不是一个人在奋斗、工作……他也有一个很要好的基友——Ken Thompson（下边简称 Thompson）。



嗯，挺有那个\*\*相的，对吧？

他们两不仅是 C 语言的发明者，UNIX 系统之父，在 1983 年的那个冬天，他们还一起获得了图灵奖（计算机界的诺贝尔奖）……剩下的你们自己脑补

噢，我们提到了 UNIX 系统，或许刚接触编程的鱼油可能不大了解 UNIX 系统……那么苹果帝国的个人电脑、iPhone、iPad、iPod、Apple Watch 的操作系统全是基于 UNIX 的！

起初 UNIX 是用汇编语言写的，显得很笨拙，在编制数据结构时浪费了大量的时间，而且系统难以调试，理解起来也很困难。所以哥俩做了个疯狂的决定——用 C 语言重写 UNIX 系统。

下边说说生平：

1941年9月9日，Ritchie 出生于纽约州的勃浪克斯山庄（Bronxville），但在 9 岁时移居新泽西州的塞米特。Ritchie 的父亲是一个电气工程师，在贝尔实验室的交换系统工程实验室当主任，因此，Ritchie 一家可谓“贝尔世家”。

Ritchie 中学毕业后进哈佛大学学物理，并于 1963 年获得学士学位。其间，哈佛大学有了一台 UNIVAC I，并给学生开设有关计算机系统的课程，Ritchie 听了以后产生了很大的兴趣。毕业以后他在应用数学系攻读博士学位，完成了一个有关递归函数论方面的课题，写出了论文，但不知什么原因没有答辩，没有取得博士学位，他就离开了哈佛，他说：“我厌倦了，怎么也不能深入。这是一种计算数学，是机器都可以做的理论。”

最终在父亲的介绍下，26 岁的丹尼斯·里奇进入贝尔实验室，从事计算机研究，一个在当时非常前沿的研究工作。同时与比他早一年到贝尔的 Thompson 会合，从此开始了他们长达数十年的基友生活。

听上去有些荒谬，C 语言的产生竟然是源于一个失败的项目。

1969 年，通用电气、麻省理工学院和贝尔实验室联合创立了一个庞大的项目——Multics 工程（这是 Ritchie 到贝尔实验室做的第一个项目）。该项目的目的是创建一个操作系统，但显然遇到了麻烦：它不但无法交付原先所承诺的快速而便捷的在线系统，甚至连一点有用的东西都没有弄出来。

虽然开发小组最终勉强让 Multics 开动起来，但他们还是陷入了泥淖，就像 IBM 在 OS/360 上面一样。他们试图建立一个非常巨大的操作系统，能够应用于规模很小的硬件系统中。Multics 成了总结工程教训的宝库，但它同时也为 C 语言体现“小即是美”铺平了道路。

当心灰意冷的贝尔实验室的专家们撤离 Multics 工程后，他们又去寻找其他任务。好基友 Thompson 此时对另一个操作系统很感兴趣，他为此好几次向贝尔管理层提议，但均遭否决。

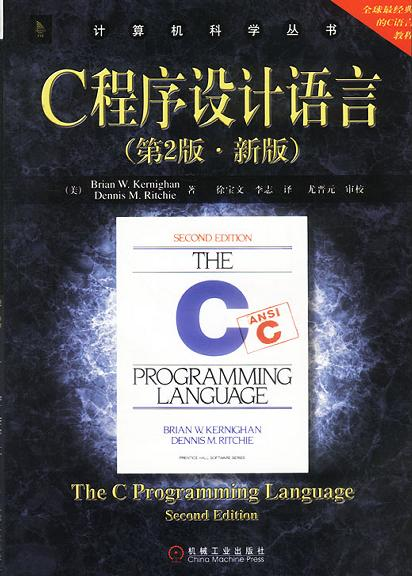


在等待官方批准的时候，Thompson 和 Ritchie 就自己搞，用业余的时间把 Thompson 的“太空旅行”软件移植到不太常用的 PDP-7 系统上。太空旅行软件模拟太阳系的主要星体，把它们显示在图形屏幕上，并创建了一架航天飞机，它能够飞行并降落到各个行星上。与此同时，Thompson 加紧工作，为 PDP-7 编写了一个简易的新型操作系统。它比 Multics 简单得多，也轻便得多。整个系统都是用汇编语言编写的。Brian Kernighan 在 1970 年给它取名为 UNIX，自嘲地总结了从 Multics 中获得的那些不应该做的教训。

UNIX 由许多小程序构成，每个小程序只能完成一个功能，任何复杂的操作都必须分解成一些基本步骤，再由小程序逐一完成。虽然看起来很低效，实际上这些小程序可以像七巧板一样自由组合，轻易就能完成意想不到的大任务。另一方面，UNIX 一改 Multics 操作系统的大型程序的复杂性，每个单一执行目的的小程序都是简洁高效，开发时间短，稳定性却更好。

在 1973 年的时候，Ritchie 感到用汇编语言做移植太过于头痛，他们想用高级语言来完成第三版，对于当时完全以汇编语言来开发程序的年代，他们的想法算是相当的疯狂。一开始他们想尝试用 Fortran，可是失败了。后来他们用一个叫 BCPL（Basic Combined Programming Language）的语言开发，他们整合了 BCPL 形成 B 语言，再后来 Ritchie 觉得 B 语言还是不能满足要求，就是就改良了 B 语言，称之为”New B“，也就是今天的大名鼎鼎的 C 语言。

C 语言也贯彻了大道至简的那个原则，语法简洁，结构清晰。他还自己写了本教材，就是今天大名鼎鼎的 The C Programming Language，小甲鱼推荐鱼油买来放床头，就供奉着：



虽然 UNIX 最初的开发是以 Thompson 为主的，但文献资料中一提到 UNIX，都一致地说是 Ritchie 和 Thompson 共同开发的，而且在“排名”上往往是 Ritchie 在前，Thompson 在后。包括 1973 年首次向社会推介 UNIX 的论文的署名，Ritchie 也是第一作者，Thompson 则为第二作者。

其实这里有两个很重要的因素。首先，UNIX 的成功应归功于它的创新。UNIX 吸取与借鉴了 Multics 的经验，但是 UNIX 在继承中又有创新，它采用了一些新方法，比 Multics 更高明一些。正是在这些方面，Ritchie 发挥了很重要的作用，使 UNIX 独具特色。其次，UNIX 成功的一个重要因素是它的可移植性。正是 Ritchie 竭尽全力开发了 C 语言，并把 UNIX 用 C 重写了一遍，这才使它具有了这一特性。（所以啊， Ritchie 才是最佳助攻手  ）

由于 C 和 UNIX 系统的优秀特性，很快就吸引了一大波黑客的加入（当时黑客这个词指的是极致钻研技术的宅男，相当于今年的极客吧），然后他哥俩就出名了：



Ritchie 和 Thompson 在成名以后，都没有走办公司、挣大钱的路，他们仍在贝尔做他们喜爱做的事，而且还一直保持着他们历来的生活习惯和作风，常常工作到深夜，在贝尔是出名的“夜猫子”。Ritchie 在接受记者采访时，就自称自己是“definitely a night person”（绝对是个也猫子）。

最后，Ritchie 1983 年接受图灵奖时已经 42 岁，但仍然单身。看吧，在座还单着的鱼油们，你们可是拥有随时撼动世界的能力！

# 打印

## 1. 在编程中，“打印”一般就是“输出”的意思。

下边的小程序就是往屏幕上打印“I love FishC.com!”这行文本：

#include <stdio.h>

int main()

{

printf("I love FishC.com!\n");

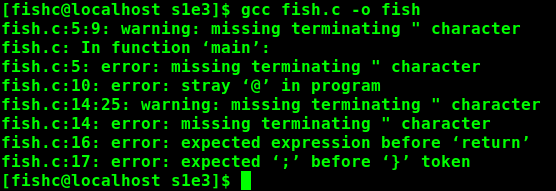
return 0;

}

## 2. 使用 GCC 编译程序：gcc 源代码 -o 可执行文件。

成功编译后是不显示任何东西的。

如果哗啦啦地显示一丢丢文字，那肯定是你的代码哪里出了问题：



## 3. printf 是格式化输出函数，点击这里可以查看中文文档。

C 语言为我们提供了很多基本函数，它们用于实现不同的功能，比如 printf 函数，就是实现格式化输出的功能。没有它，我们压根儿不可能仅用六行代码，就将文本打印到屏幕上。

所以说，函数是一个好东西！

## 4. 转义字符

在 C 语言中，用双引号括起来的内容我们称之为字符串，也就是我们平时所说的文本。

字符串可以由可见字符和转义字符组成，像课堂上演示那条鱼的主要组成部分——星号（\*），就是可见字符。可见字符就是你输入什么，显示出来就是什么。

而你如果想将一个字符串分为两行来显示，那么你就需要使用到转义字符。

转义字符一般是表示特殊含义的非可见字符，以反斜杠开头：

## 5. 反斜杠的奥义

在字符串中反斜杠 + 字符是转义字符，表示特殊含义。

但反斜杠如果后边不带任何字符（直接换行），表示我们希望 C 语言将该行以及下一行看做是一个整体。

比如：

#include <stdio.h>

int main()

{

printf("Hello World!\n");

return 0;

}

可以写成：

#in\

clude \

<stdio.h>

int ma\

in()

{

print\

f("Hello World!\n");

return \

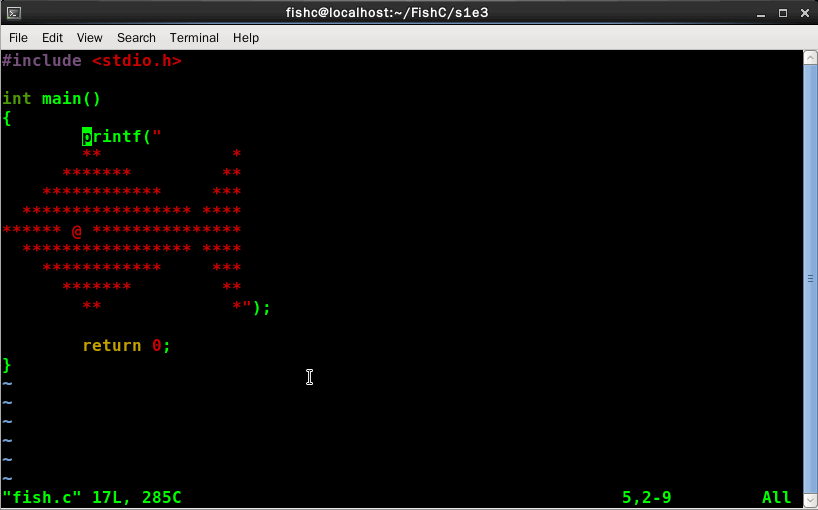
0;

}

当然，你这样写毫无意义！！！

这个方法主要用在当你的字符串或语句太长（一行不足以存放，或严重影响阅读），那么你可以通过反斜杠（\）进行“断行”。

## 6. 彩蛋

你知道在视频中，小甲鱼一气呵成的这个动作是咋做到的吗？  
  


在 VIM 的普通模式下，点（.）操作表示重复上一次的修改。

比如在视频中：

先输入大写字母 A 在行尾进行插入模式

然后输入 \n\，点击 Esc 回到普通模式

点击字母 j 将光标移动到下一行

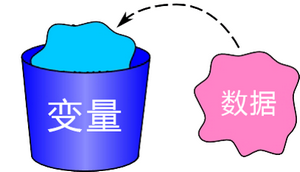
按一下点（.），VIM 将自动实现上方 1 和 2 的操作（即在行尾插入 \n\）

接下来我只需要重复点击字母 j + 点（.）即可

# 变量

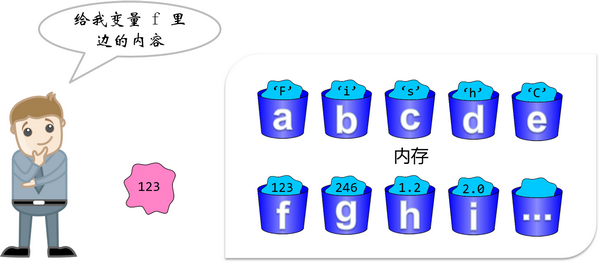
## 1. 变量和常量是程序处理的两种基本数据对象。

我们把要让 CPU 处理的数据都放在内存中，但如果你没有给他安排一个位置，而是随意存放，那么你在后边需要再次用到这个数据的时候，就再也找不到它了。所以变量的意义就是确定目标并提供存放的空间。



## 2. 变量名

为了确定目标，我们需要给变量命名，一旦变量有了名字，我们就可以通过直呼其名的方式来获取它里边存放的数据



## 3. C 语言变量命名需要遵守的规范

C语言变量名只能是英文字母（A-Z，a-z）和数字（0-9）或者下划线（\_）组成，其他特殊字母不行。下横线通常用于连接一个比较长的变量名，比如i\_love\_fishC

第一个字母必须是由英文字母或者下划线开头，也就是不能用数字开头。

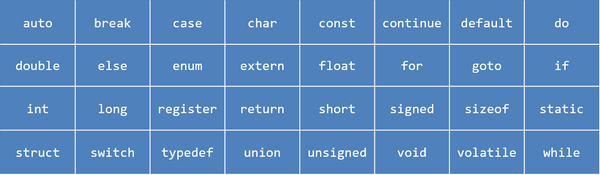
变量名区分大小写。因为C语言是大小写敏感的编程语言，也就是大写的FISHC跟小写的fishc会被认为是不同的两个名字。在传统的命名习惯中，我们用小写字母来命名变量，用大写字母来表示符号常量名。

不能使用关键字来命名变量。

## 3. 关键字

关键字就是 C 语言内部使用的名字，这些名字都具有特殊的含义。如果你把变量命名为这些名字，那么 C 语言君搞不懂你到底想干嘛了。

传统的 C 语言（ANSI C）有 32 个关键字：



1999年，ISO 发布 C99，添加了 5 个关键字：



2011年，ISO 发布 C11，添加了 7 个关键字：



## 4. C 语言常用的基本数据类型

char —— 字符型，占用一个字节

int —— 整型，通常反映了所用机器中整数的最自然长度

float —— 单精度浮点型

double —— 双精度浮点型

## 5. C 语言声明变量的语法

int a;  // 在内存中找到一个整型大小的位置，然后霸占起来，并给它命名叫 a

char b;  //在内存中找到一个字节大小的位置，然后霸占起来，并给它命名叫 b

float c;  //在内存中找到一个单精度浮点型数据大小的位置，然后霸占起来，并给它命名叫 c

double d;  //在内存中找到一个双精度浮点型数据大小的位置，然后霸占起来，并给它命名叫 d

## 6. printf 函数的具体用法

来来来，看这里 -> http://bbs.fishc.com/thread-66471-1-1.html

## 7.彩蛋

编译并运行程序我们这么写：

gcc test.c -o test && ./test

想知道 && 是什么意思，以及具体用法吗？

&& 是 Linux 操作符，用它前后连接两个命令。表示如果第一个命令执行成功，才会执行第二个命令。换句话说，如果 test.c 源文件编译不通过，便不会执行（即使已经存在 test 可执行文件）。

与之区别的有 & 操作符，它则使你可以在后台执行多个命令（不管前边的命令成功与否）。

# 常量和宏定义

## 1. C 语言中常见的常量

整型常量：520, 1314, 123

实型常量：3.14, 5.12, 8.97

字符常量

普通字符：'L', 'o', 'v', 'e'

转义字符：'\n', '\t', '\b'

字符串常量："FishC"

符号常量：使用之前必须先定义

## 2. 定义符号常量（宏定义）

符号常量的定义格式是：

#define 标识符 常量

其中这个 #define 是一条预处理命令（预处理命令都以"#"开头），我们也称为宏定义命令。它的功能就是把程序中所有出现的标识符都替换为随后的常量。

演示案例：

#include <stdio.h>

#define URL "http://www.fishc.com"

#define NAME "鱼C工作室"

#define BOSS "小甲鱼"

#define YEAR 2010

#define MONTH 5

#define DAY 20

int main()

{

printf("%s成立于%d年%d月%d日\n", NAME, YEAR, MONTH, DAY);

printf("%s是%s创立的……\n", NAME, BOSS);

printf("%s的域名是%s\n", NAME, URL);

return 0;

}

上边的大写字母 URL、NAME、BOSS、YEAR、MONTH、DAY 这些都是符号常量，正如大家看到的，为了将符号常量和普通的变量名区分开，我们习惯使用全部大写字母来命名符号常量，使用小写字母来命名变量。

## 3. 标识符

在 C 语言中，标识符指的就是一切的名字。比如刚刚的符号常量名是标识符，变量名也是一个标识符。以及我们即将学到的函数、数组、自定义类型这些的名字都称之为标识符。

## 4. 字符串常量

C 语言用一个特殊的转义字符来表示字符串的结束位置。这样当操作系统读取到这个转移字符的时候，就知道该字符串到此为止了。

这个转义字符就是空字符：'\0'

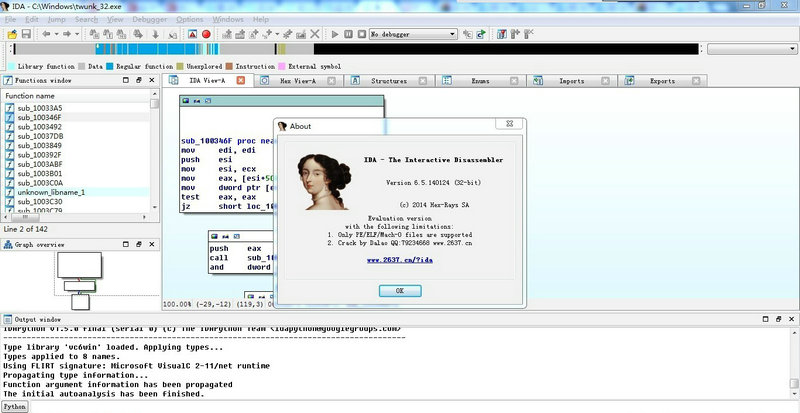
## 5. 彩蛋

你知道被公认为世界上第一位程序员是谁吗？

就是她：



大家在学习逆向破解的时候肯定听说过 IDA 这款反编译工具（人家是屠龙刀，是利器好不好？没听说过还好意思说学破解嘛~~~）



作为世界顶级交互式反汇编静态调试工具，人家直接就拿这妹子做头像，你说屌不屌

还没完，美国国防部为克服软件开发危机，耗费巨资，历时近 20 年开发了一门编程语言，被誉为第四代计算机语言的成功代表（目前是美国军方的专用计算机语言！）

这门语言就是 Ada，就是咱今天要谈的主人公——阿达！

她，奥古斯塔·阿达·拜伦。

没错，她的父亲就是那个著名的诗人——拜伦（不是干爹，是亲爹！）

阿达的麻麻是数学家，粑粑是诗人！

正如浪漫与理性本是毫无瓜葛的两极……

粑粑麻麻在阿达出生不久就离婚了（似乎八卦这个不好 ）

然后阿达就随着母亲同住，她的母亲鼓励她从事数学研究，以避免像她的父亲那样出现“危险的诗人倾向”（拜伦不仅是一位伟大的诗人，还是一个为理想战斗一生的勇士，积极而勇敢地投身革命——参加了希腊民族解放运动，并成为领导人之一。 ）

为啥这妹子这么牛呢？

为啥呢？！

这是因为她遇到了她老公——查尔斯·巴贝奇，英国文学家、数学家、哲学家、机械工程师，计算机概念的第一奠基人，他发明的分析机是现代电子计算机地雏形。（呃……为啥古时候一个人可以当这么多“家”，小甲鱼想了下，主要有两点：1. 木有微信、微博、非死不可；2. 木有那么多概念要学，我们在课堂上苦逼背的概念都是他们创造的好不！）

艾达在 8 岁时就展现了极佳的数学天赋（文献也木有说具体表现在哪里，应该是可以背诵圆周率小数点后 100 位吧？！），是当时极少数能够理解巴贝奇高瞻远瞩思想的人之一（都睡同一张床了，不理解？你系不系不想活了……）。

于是乎二人展开了维多利亚式的终身合作关系，这也让阿达成为世界上当之无愧的第一位计算机程序员。（嗯~~~她老公创造了计算机，她驾驭着计算机，很好理解嘛~~~）

值得一提的是循环和子程序的概念也是阿达创造滴

不过，比较凄凉的是尽管阿达有着辉煌的一生，但最终却以悲剧结束：她变成了一个赌徒、酒鬼、瘾君子，最后死于癌症，享年 36 岁……

尼玛，36 岁，多少人还在为房款首付而奔波……

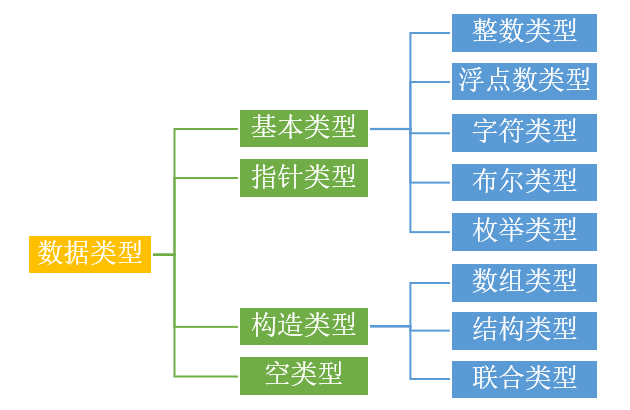
再看一眼，她，美吗？



# 数据类型

## 1. 数据类型

在 C 语言里，所谓的数据类型就是坑的大小。我们说变量就是在内存里边挖一个坑，然后给这个坑命名。那么数据类型指的就是这个坑的尺寸。C 语言允许使用的类型如下：

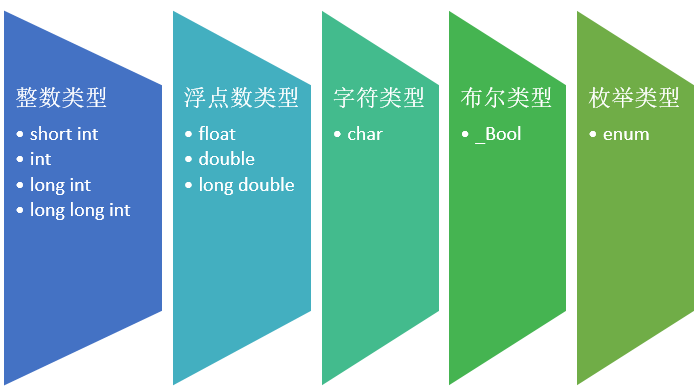


## 2. 【扩展阅读】为什么 C99 新增加的布尔类型叫 \_Bool，而不是 bool 或 BOOL？

[**为什么 C99 新增加的布尔类型叫 \_Bool，而不是 bool 或 BOOL？**](http://bbs.fishc.com/thread-67033-1-1.html)

## 3. short 和 long

我们可以为这些基本数据类型加上一些限定符，比如表示长度的 short 和 long。比如 int 经过限定符修饰之后，可以是 short int，long int，还可以是 long long int（这个是 C99 新增加的）。

 这里需要说的是，这里 C 语言并没有限制 int 的坑具体要挖多大，short int 或 long int 的坑又要挖多大。标准只是要求：short int <= int <= long int <= long long int。

## 4. sizeof 运算符

sizeof 用于获得数据类型或表达式的长度，它有三种使用方式：

sizeof(type\_name);  //sizeof(类型);

sizeof(object);  //sizeof(对象);

sizeof object;  //sizeof 对象;

## 5. signed 和 unsigned

还有一对类型限定符是 signed 和 unsigned，它们用于限定 char 类型和任何整型变量的取值范围。

signed 表示该变量是带符号位的，而 unsigned 表示该变量是不带符号位的。带符号位的变量可以表示负数，而不带符号位的变量只能表示正数，它的存储空间也就相应扩大一倍。默认所有的整型变量都是 signed 的，也就是带符号位的。

因此加上 signed 和 unsigned 限定符，四种整型就变成了八种：

[signed] short [int]

unsigned short [int]

[signed] int

unsigned int

[signed] long [int]

unsigned long [int]

[signed] long long [int]

unsigned long long [int]

# 取值范围

## 1. 比特位和字节

CPU 能读懂的最小单位（只能存放 0 和 1）—— 比特位，bit，b

内存机构的最小寻址单位 —— 字节，Byte，B

关系：1Byte == 8bit

因此，一个字节可以表示最大的数是：11111111

## 2. 二进制、十进制和十六进制



【扩展阅读】[进制转换](http://bbs.fishc.com/thread-67123-1-1.html)

## 3. 符号位

存放 signed 类型的存储单元中，左边第一位表示符号位。如果该位为 0，表示该整数是一个正数；如果该位为 1，表示该整数是一个负数。

一个 32 位的整型变量，除去左边第一位符号位，剩下表示值的只有 31 个比特位。

## 4. 补码

计算机是用补码的形式来存放整数的值。

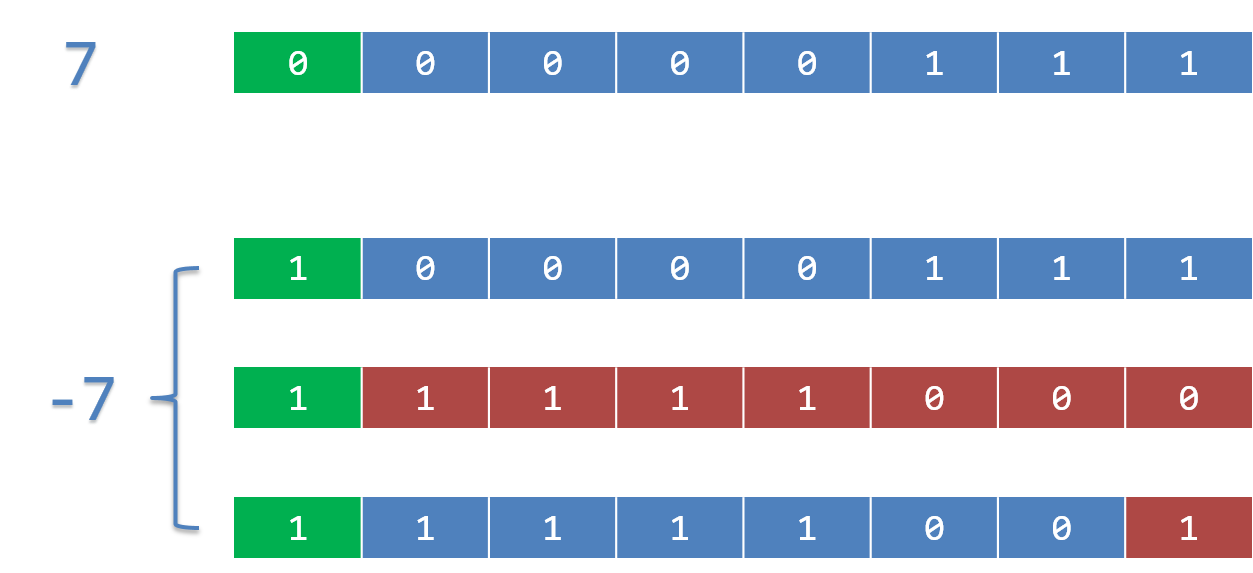
正数的补码是该数的二进制形式。

负数的补码需要通过以下几步获得：

1.先取得该数的绝对值的二进制形式

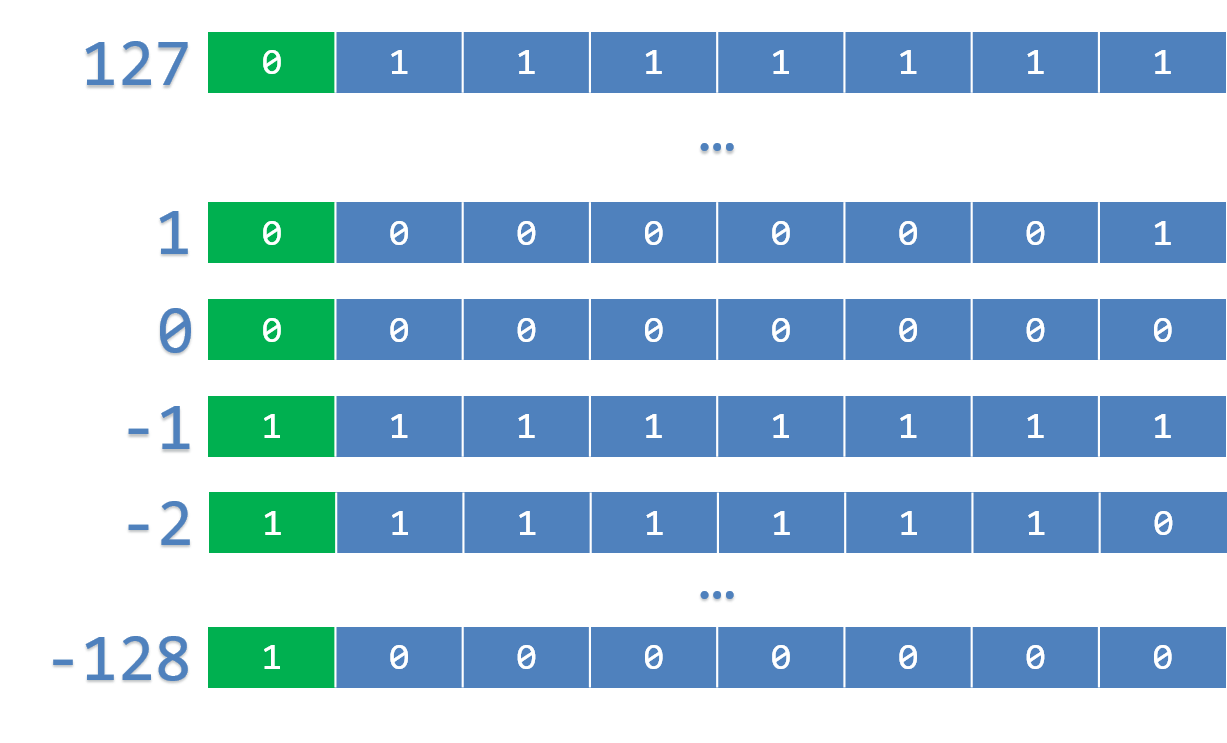
2.再将第1步的值按位取反

3.最后将第2步的值加1



【扩展阅读】[使用补码的好处](http://bbs.fishc.com/thread-67124-1-1.html)

## 5. 二进制表示最大值和最小值



## 6. 基本数据类型的取值范围





关于浮点数的存放形式和取值范围就没那么好理解了，对于初学者来说可以说是生涩难懂……

所以我们这里就不要求掌握了，不过我知道有些鱼油是拦也拦不住，所以我这里准备了两篇图文并茂的扩展阅读，有兴趣的朋友可以自己学习提升一下：

【扩展阅读】[定点数：用二进制表示小数（#）](http://bbs.fishc.com/thread-67211-1-1.html)  
  
【扩展阅读】[浮点数：表示更大范围的小数（#）](http://bbs.fishc.com/thread-67214-1-1.html)

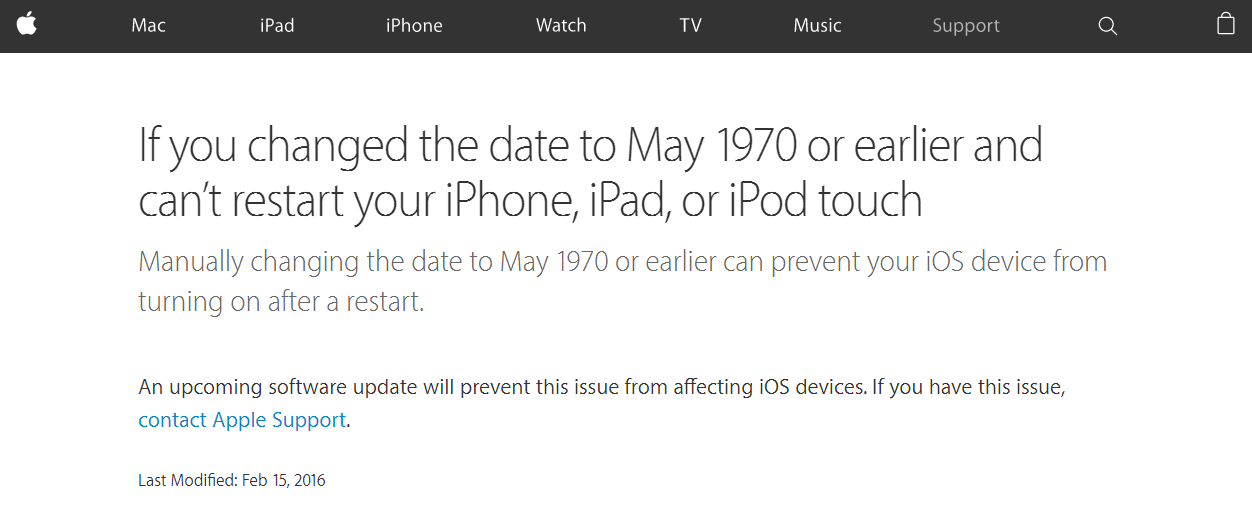
## 7. 彩蛋

最近苹果的 1970 事件可是炒的风风火火啊，终于让小甲鱼赶上一回热乎的了



日前有消息称，把 iOS 设备日期设置为 1970 年 1 月 1 日，重启后就会变砖。很多好奇网友尝试后中招。

苹果也在官网正式承认这一问题，并表示即将发布一个系统更新，以防该问题在未来影响 iOS 设备。

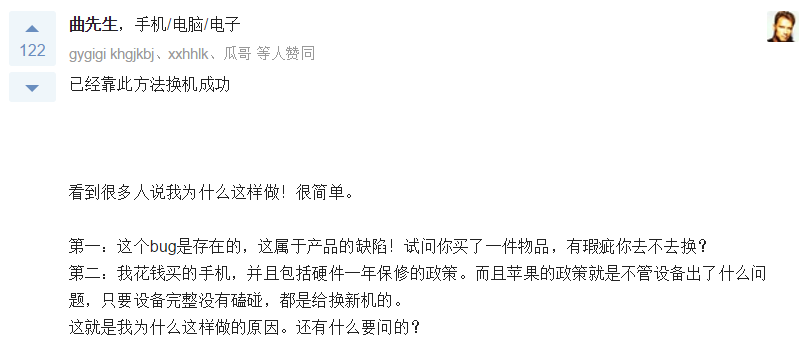


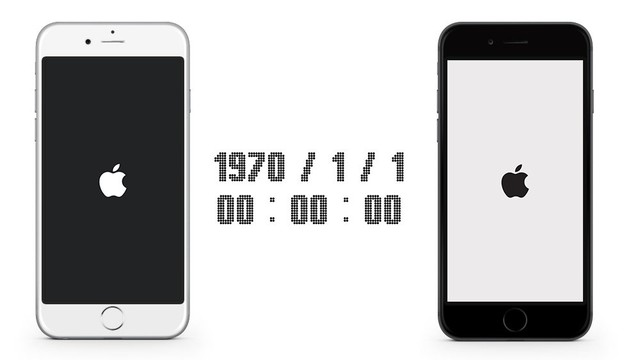
小甲鱼劝大家不要轻易尝试，至少不要拿自己的机子开玩笑，因为已经有很多烈士身先士卒了……



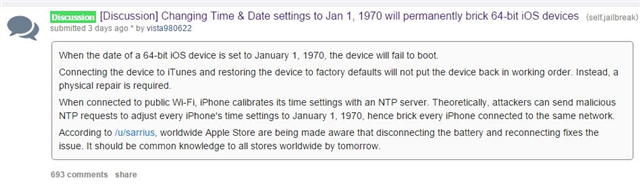








这个 bug 最初的源头是在 reddit（类似国内的知乎），一个叫 vista980622 的网友发了一个帖子：



大致说了一下自己无意间发现了这个 bug，变砖。最后是以拆机的方式解决的。并且指出，黑客可以建立一个公共的 WIFI，在连上后，利用这个漏洞让他人的手机变砖。

然后，讨论区下面一位貌似苹果天才吧的工作人员在昨天给出了一个解决方案：

“Can confirm the solution. Our Genius team worked through a few steps and came to the conclusion that disconnecting the battery and reconnecting fixes the issue and allows the phone to boot to the lock screen.

At least we know how to fix it now and avoid those phone swap outs.

We've passed this info on globally to the relevant Apple Support teams.

It should be common knowledge to all stores worldwide by tomorrow.”

“可以确认这个解决方案。我们天才吧的团队试了几下，发现卸掉电池，再连接上，就能恢复手机。

至少我们现在知道如何解决这个问题，以及避免它发生了。

我们已经把这个消息传递给了全球的苹果售后支持团队。

明天全世界的Apple Store都会知道这个“常识性”的解决方案。”

没错，通过拆电池，让电池静置 10 分钟再装回去，系统会在开机时自动重设时间，从而让机子恢复正常

但是这么蛋疼的 bug，究竟是怎么来的呢？

这要从 Unix 系统说起……

苹果 iOS 系统是来源于 Unix 系统，自然就继承了 Unix 使用时间戳（Unix epoch）表示系统时间的规定

Unix 时间戳使用有符号整型来存储，规定按 UTC 时区的 1970 年 1 月 1 日 00:00 这个时间开始计时，即在此时间后每经过 1 秒，时间戳 +1。

这样一来，2016 年 2 月 16 日 17:00 就能够表示为 1455613200 秒，换算成二进制就是“……（64 位，前面还有 33 个 0）1010110110000101110010100010000”。

iOS 中时间的设定最多也只能回溯到 1970 年 1 月 1 日 00:00 也是这个原因，因为对于 iOS 来说， 1970 年这一天，就是时间的起点。

那终点呢？

对于32位的苹果系统来说，是 2038 年 1 月 19 日 3:14:07；

对于64位的苹果系统来说，是 292,277,026,596 年 12 月 4 日15:30:08（约 2900 亿年后的事儿了）。

说了这么多，那导致这个 bug 的原因是什么？

虽然苹果在设置项目中限定了日期不能设置在时间戳零点之前，但可能局部关键功能具有查询过往信息的规则，并且在开机时自动触发。

当局部时间设置比时间戳的起点更早之后，终极问题出现了：我们应该怎么表示比时间戳起点更早的时间？

No pic you say a J8，直接上图（这是 32 位的情况，大家仔细观察数字的变化）：

  
  
同样的道理，对于 64 位系统来说，起始时间之前的“负值”就相当于一个无穷大的时间值（也就动辄一两千亿年以后……）。

同样的道理，对于 64 位系统来说，起始时间之前的“负值”就相当于一个无穷大的时间值（也就动辄一两千亿年以后……）。

对于局部关键功能的时间值是无穷大，而系统的时间却是零点，打起架来的结果就是整个系统直接罢工（具体是哪部分功能罢工我们暂不得知，但是肯定是来自于像电池检查、系统内部的关键性功能。

这就是 1970 bug 的根本原因……

# 字符和字符串