TCPL学习问题和解答

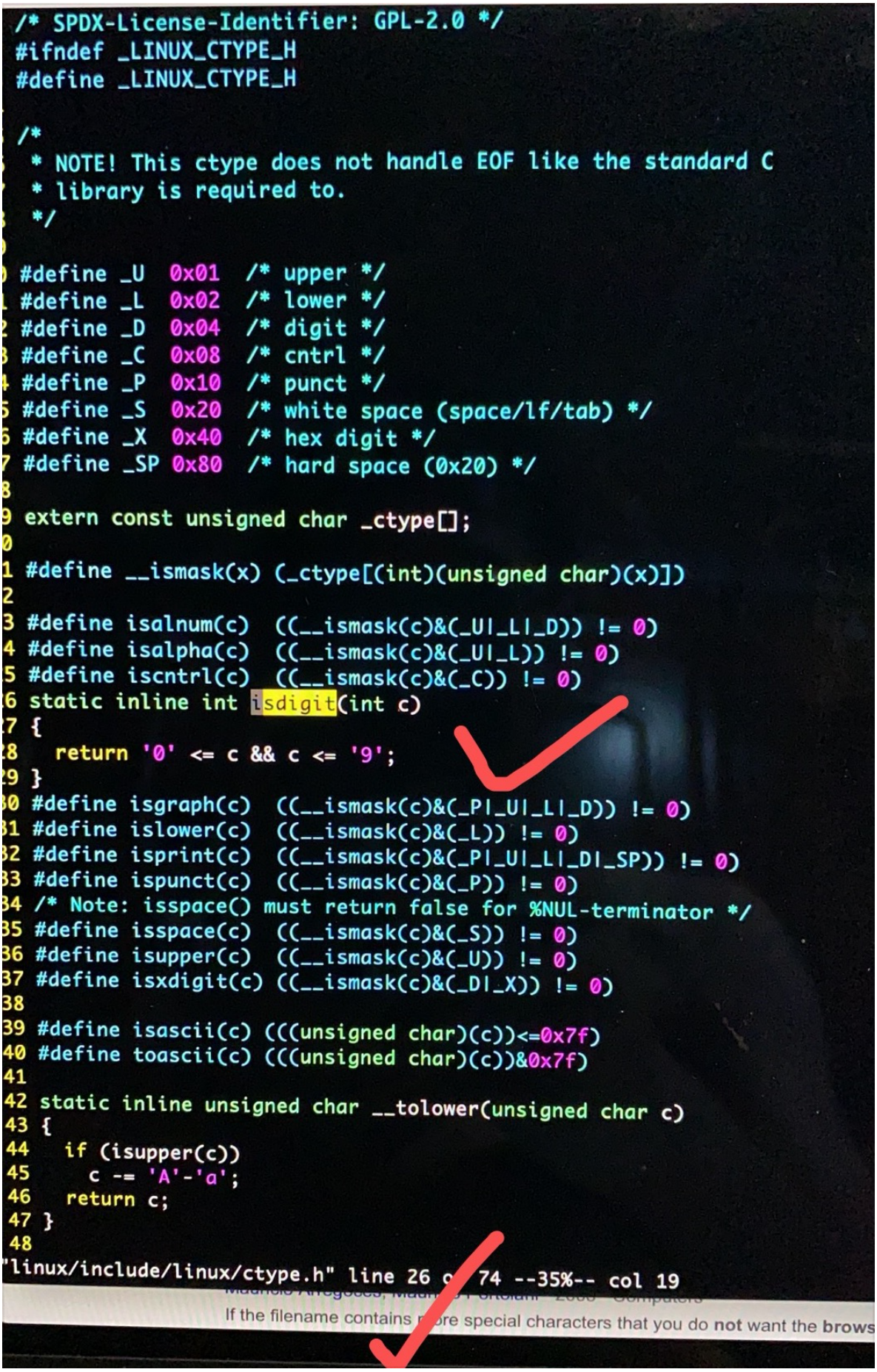
问题：既然程序检查一个逻辑表达式时是从左到右的，一旦确定了这个表达式的结果就停止后面的检查（如只要**A=0**，**“A**且**B”**就等于**0**，不须去检查**B**），那程序能不能确定某些永远为真或假的逻辑表达式，并将它们忽略?

这个问题特别好。首先，c语言的表达式计算是短路的。其次，如果写了永真或永假的表达式，编译器发现后会给出警告。实际产生的二进制文件中会去掉这些没用的语句（对应的汇编语言代码中没有对应的逻辑）。

编译器还会很多优化，gcc默认有个参数-O，后面可以写数字0到3，表示不优化和逐步递增的优化。不优化时，产生的汇编代码基本可以和原始C代码对应上，但是性能不好。优化程度越高，二者对不上的地方就会越多，性能一般会更好。默认一般是-O2。怎么看汇编以及怎么对应回C代码呢？有命令例如objdump，nm，readelf，苹果上可能是别的命令。

问题：**<ctype.h>**中的函数都是与字符集无关的，比如**isdigit()**不仅适用于**ASCII**字符，这是怎么做到的?

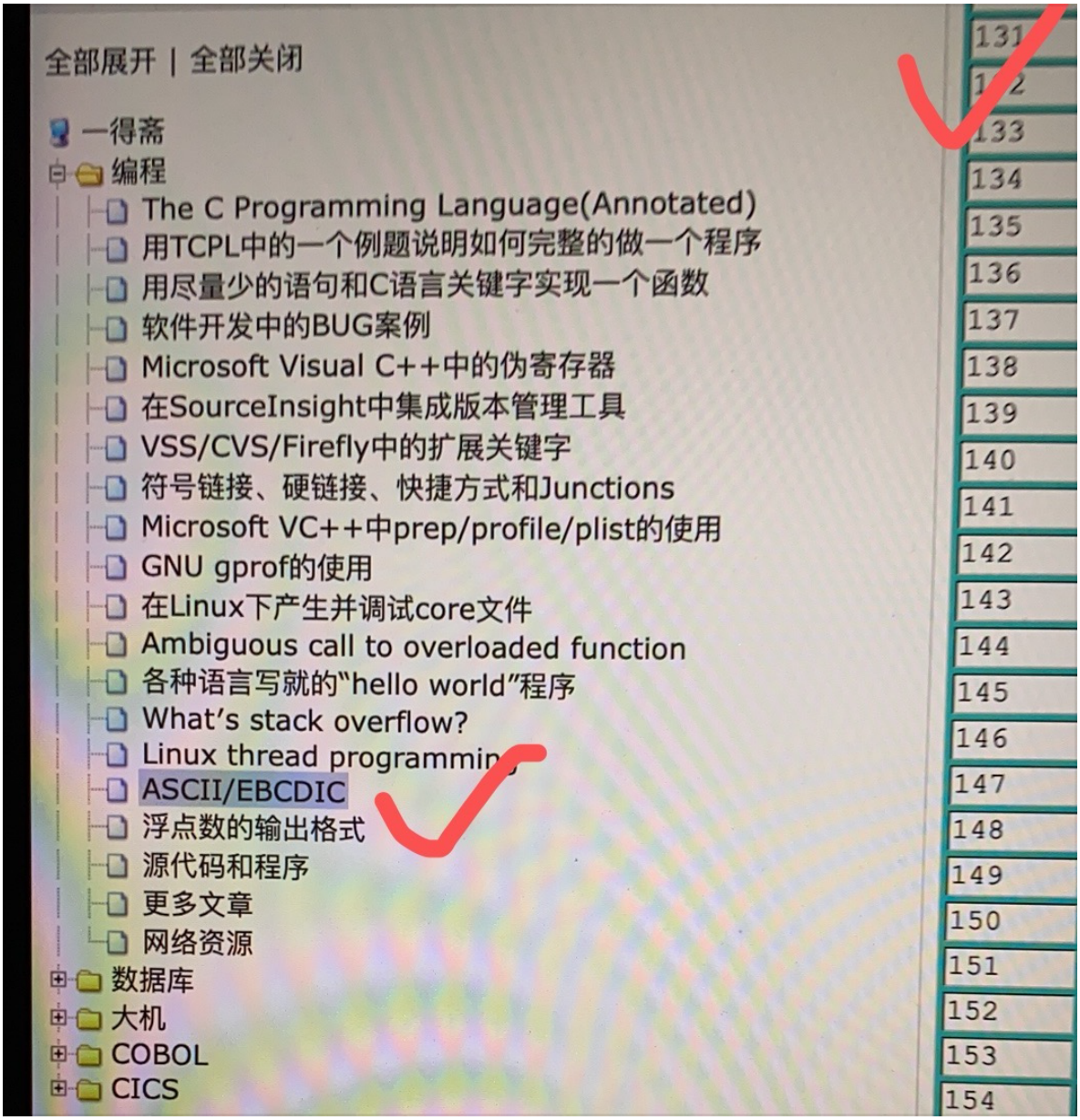
字符集这个概念有点复杂。你在浏览器Safari的View->Text Encoding下面可以看到一大堆。不过没有原始的ASCII，类似它的只有各种Latin变种。man isdigit说它跟locale无关，但是isnumber跟locale有关。locale就是Windows里头的区域设置，macOS和Linux里头有个命令locale。它跟语言和字符集有关。isdigit的代码其实类似下图，因为常见的字符集都和ascii兼容，所以它写成这样是可以的。



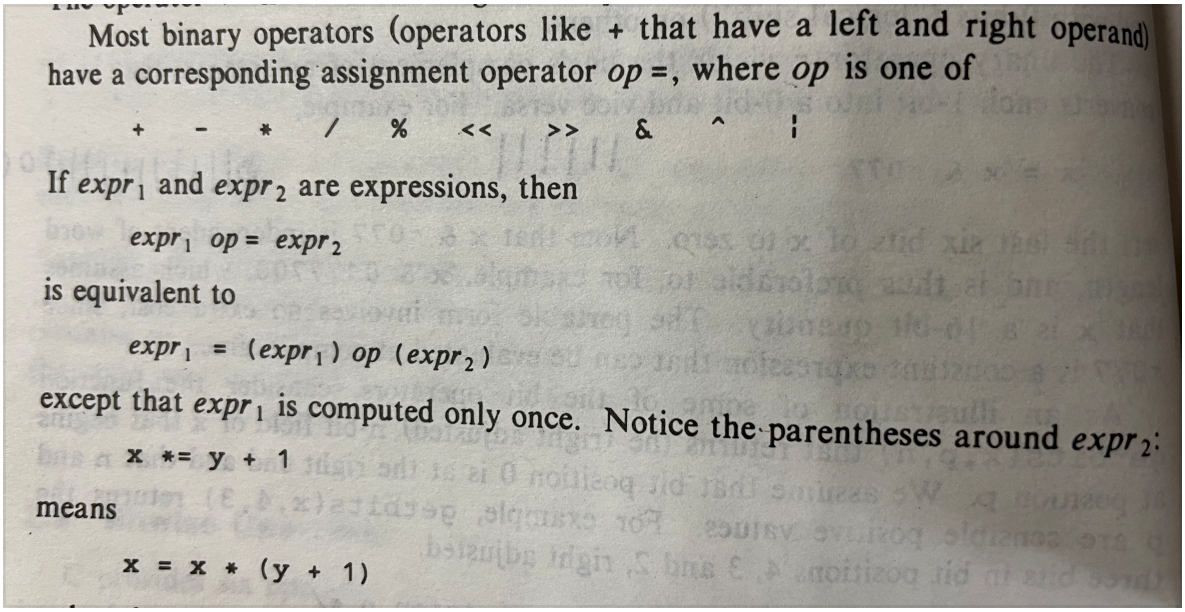
```
/* SPDX-License-Identifier: GPL-2.0 */
#ifndef _LINUX_CTYPE_H
#define _LINUX_CTYPE_H

/*
 * NOTE! This ctype does not handle EOF like the standard C
 * library is required to.
 */

#define _U  0x01  /* upper */
#define _L  0x02  /* lower */
#define _D  0x04  /* digit */
#define _C  0x08  /* cntrl */
#define _P  0x10  /* punct */
#define _S  0x20  /* white space (space/lf/tab) */
#define _X  0x40  /* hex digit */
#define _SP 0x80  /* hard space (0x20) */

extern const unsigned char _ctype[];

#define __ismask(x) (_ctype[(int)(unsigned char)(x)])

#define isalnum(c)  ((__ismask(c)&(_U|_L|_D)) != 0)
#define isalpha(c)  ((__ismask(c)&(_U|_L)) != 0)
#define iscntrl(c)  ((__ismask(c)&(_C)) != 0)
static inline int isdigit(int c)
{
   return '0' <= c && c <= '9';
}
#define isgraph(c)  ((__ismask(c)&(_P|_U|_L|_D)) != 0)
#define islower(c)  ((__ismask(c)&(_L)) != 0)
#define isprint(c)  ((__ismask(c)&(_P|_U|_L|_D|_SP)) != 0)
#define ispunct(c)  ((__ismask(c)&(_P)) != 0)
/* Note: isspace() must return false for %NUL-terminator */
#define isspace(c)  ((__ismask(c)&(_S)) != 0)
#define isupper(c)  ((__ismask(c)&(_U)) != 0)
#define isxdigit(c) ((__ismask(c)&(_D|_X)) != 0)

#define isascii(c) (((unsigned char)(c))<=0x7f)
#define toascii(c) (((unsigned char)(c))&0x7f)

static inline unsigned char __tolower(unsigned char c)
{
   if (isupper(c))
      c -= 'A'-'a';
   return c;
}

"linux/include/linux/ctype.h" line 26 c  74 --35%-- col 19
```

技巧：vim里头按下星号键会黄色高亮显示的光标处的单词。按下ctrl+g会在最底下的状态栏显示当前文件名。

大型机的字符集默认是EBCDIC,它里头的数字也是连续编码的，但是字母就不是了。我的网页有个ascii和它的对照表。



问题：不是只能给变量赋值吗，怎么给表达式赋值?



Most binary operators (operators like + that have a left and right operand) have a corresponding assignment operator *op* =, where *op* is one of

```
+   -   *   /   %   <<   >>   &   ^   |
```

If $expr_1$ and $expr_2$ are expressions, then

$expr_1$ *op* = $expr_2$

is equivalent to

$expr_1$ = ($expr_1$) *op* ($expr_2$)

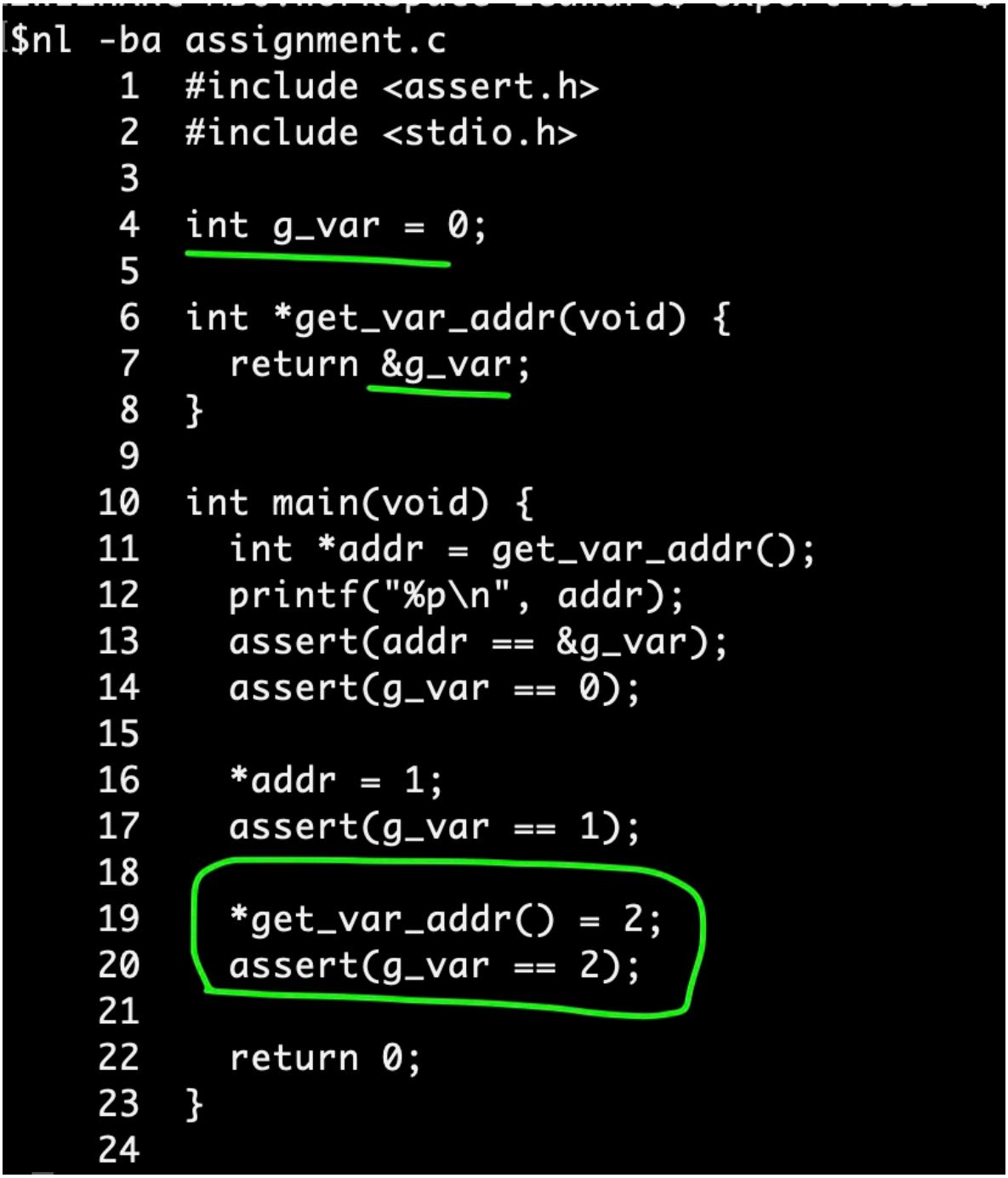
except that $expr_1$ is computed only once. Notice the parentheses around $expr_2$:

```
x *= y + 1
```

means

```
x = x * (y + 1)
```

这是个好问题！c语言比较灵活，c++后来在此基础上发展出来了更灵活的语义。通常左边是个变量，但是某些表达式可以产生类似变量的东西，最典型的就是产生一个变量的地址。例如：



```
$nl -ba assignment.c
     1  #include <assert.h>
     2  #include <stdio.h>
     3
     4  int g_var = 0;
     5
     6  int *get_var_addr(void) {
     7    return &g_var;
     8  }
     9
    10  int main(void) {
    11    int *addr = get_var_addr();
    12    printf("%p\n", addr);
    13    assert(addr == &g_var);
    14    assert(g_var == 0);
    15
    16    *addr = 1;
    17    assert(g_var == 1);
    18
    19    *get_var_addr() = 2;
    20    assert(g_var == 2);
    21
    22    return 0;
    23  }
    24
```

```
#include <assert.h>
#include <stdio.h>

int g_var = 0;

int *get_var_addr(void) {
  return &g_var;
}

int main(void) {
  int *addr = get_var_addr();
```

```
  printf("%p\n", addr);
  assert(addr == &g_var);
  assert(g_var == 0);

  *addr = 1;
  assert(g_var == 1);

  *get_var_addr() = 2;
  assert(g_var == 2);

  return 0;
}
```

问题：**if** 表达式中的 **&&** 等

If (a && b) 即可。如果a为真，继续判断b。如果a为假，表达式已经为假，b不会执行。

if (a && b) { foo(); } => if (a) { if (b) { foo(); } }

shell 里头也可以用 &&， || 。例如：

```
$ls not-exist && ls qsort.cc
ls: not-exist: No such file or directory

$ls not-exist || ls qsort.cc
ls: not-exist: No such file or directory
qsort.cc
```

问题：一个函数可不可以返回多个值?

C 的函数长成这样：void foo(void)；void bar(int x, char *y); int foobar(int x, int y)

一般只把函数名左边的这个视为返回值。void 表示没有返回值。其他类型表示只返回一个该类型的返回值。

但是函数的参数也是可以带回返回值的。只不过 C 语言默认参数是按照值传递的（pass by value），
要起到这个效果，就需要做点变动。典型的例子是交换两个整数。

```
void swap(int x, int y) { int tmp = x; x = y; y = tmp; }

int x = 0;
```

```
int y = 1;
swap(x, y);
```

这样写是不行的，函数里头看到的 x 和 y 是外面传进去的变量的一个拷贝。函数只是把自己的局部变量
给交换了（函数参数的用途跟局部变量一样）。所以，需要修改成：

```
void swap(int *x, int *y) { int tmp = *x; *x = *y; *y = tmp; }

int x = 0;
int y = 1;
swap(&x, &y);
```

这时候为什么就可以了呢？因为传递的是变量 x 和 y 的地址啦。它们指向的内存就是外面两个变量所
指向的内存。函数参数里头的 x 和 y 名字不重要，只是个标记而已。可以是 a 和 b 或者别的合法名字。

问题：如果用转义序列来表示控制符，可不可以代替此控制符的全部功能？比如**printf("%c", "\n")**来实现按下回车键登录某网站

这个不可见指的是ascii码表示的控制字符。控制字符就是0-31以及127。空格是个另类，算可见字符。

如果控制字符就是要输入并且保存到程序里头的话，就是可以的。不妨假设程序中保存输入字符的变量是个char buf[]数组，输入的字符包括控制字符会在里头占一个字节（如果是汉字等另当别论）。去查看这个字节的值就是字符对应的ascii编码，例如回车就是个数字13。

如果输入的控制字符被shell或别的程序先处理了，到我们的程序就不一定是上面说的数字了。shell或这个别的程序看到的还是13。

你的登陆网站的例子中，用户输入回车时，浏览器端执行的代码就是认出了回车，然后把它处理了一下（例如收集你在网页里头输入的东西，发送给服务器）。

总之，无论什么字符，到了程序里头都只是个二进制字节（汉字等可能是多个字节）。

问题：**goto**似乎不太好用。为什么程序中不能跳转到代码的任意一行，像汇编语言那样?

老爷爷是1968年写的文章，那时候结构化编程还没有成熟，if之类的结构好像才被发明不久。所以是一个从汇编语言过渡到高级语言的阶段。高级语言中用

goto现在很少见到了。c语言也只有我说的少数几种情况会用它。
读文章要结合当时的历史背景 为当前的学习服务。https://www.cs.utexas.edu/users/EWD/transcriptions/EWD02xx/EWD215.html 老爷爷的文章：Goto is Harmful。

问题：在**main**函数或其他函数之内可以定义并声明函数吗?

1. 函数内可以声明函数。但不是一个好用法，偶尔会用到。通常声明需要集中在单独的头文件里头，或者统一放到c文件的最前面#include之后的地方。
2. 标准的c里头函数内不可以定义函数。但是gcc里头可以，是它做的扩展特性。

c语言标准也是在进化的，不过进化得慢。目前用的多是c11，也就是2011年发布的版本。可以看看wikipedia的介绍。c++进化得很快，也更复杂。
c++等高级语言支持lambda特性，可以实现函数内定义函数的效果。例如：

```
auto f = []() { printf("haha"); };
f();
```

这几行可以放到一个函数内。打印出haha。

问题：计算器

书上计算器解析数字和运算符的过程就是编译原理中典型的词法分析和语法分析。他没有利用现成的生成代码的工具，例如yacc/lex，所以我们叫它手写版。词法分析就是把输入的字符串识别成一个个的词，token。例如1+3*4中的1、+、3、*、4就是一个一个的token。如果有联连续的空格等，它们也可以被当作一个token。

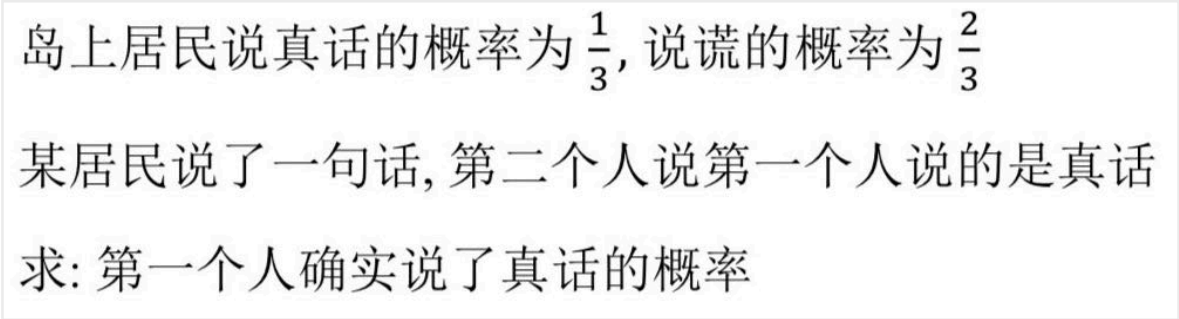
词法分析相对比较简单，毕竟一般的计算机语言或者表达式中token的种类有限。但也有很复杂的，例如正则表达式。

语法分析就比较复杂，因为语言写成的程序或表达式可以写得很复杂，有语法对的还有错的，长度也没有上限。语法分析的实质是检查程序或表达式是否符合语法，表达的是什么意思，并且用某种内部形式表示出来，以便进行后续处理。复杂的如c、go都是语言，简单点的如shell、awk、计算器、命令行参数、浏览器输入框里头的东西都可以算语言。

手写这些东西很麻烦，于是yacc/lex这样的工具很早就被发明出来了。它们产生的代码主要就是状态机的形式。为了表示待实现语言的语法，yacc支持一种叫做BNF形式的语言，可以被称为语言的语言。

https://zhuanlan.zhihu.com/p/143867739 了解就好，现在不用学它们。

问题：课上的题目。



若甲说真话，乙说的也是真话。
若甲说假话，乙说得也是假话。

所以，

第一行的概率为 1/3 * 1/3 = 1/9。
第二行的概率为 2/3 * 2/3 = 4/9。

所以，

1/9 / (1/9 + 4/9) = 1/5。

问题：逆波兰表达式怎么处理多元运算符?

普通写法：expr ? x : y，看起来有两个操作符，按照数学里头函数的写法，改成 ?:，一个操作符，或者不妨就叫f。波兰写法：f expr x y；逆波兰写法：expr x y f。

问答完毕。