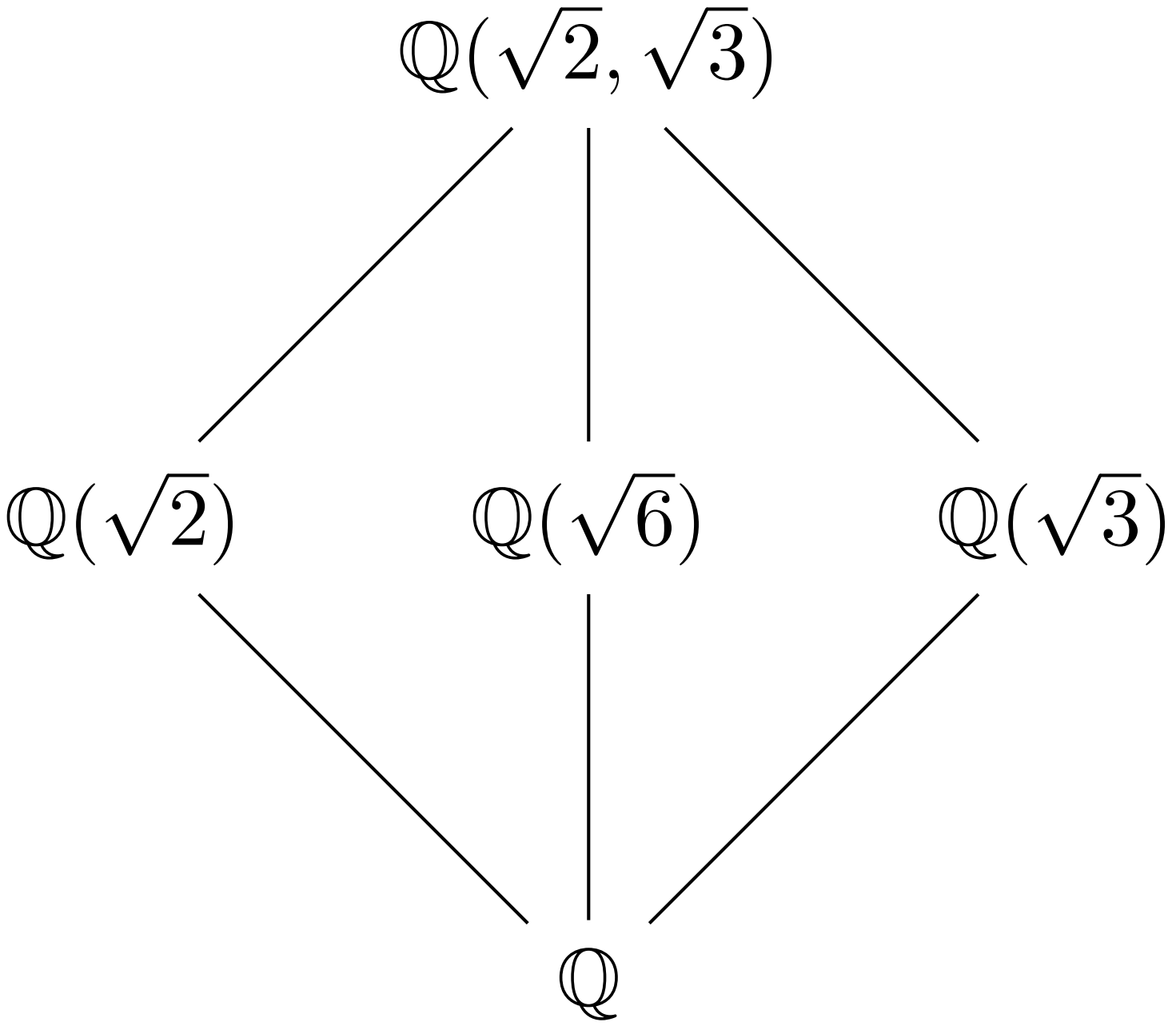

使用 C 语言构建终端文本编辑器

作者：左元

目录

第一章 设置

第 1 步: *kilo.c*

```c
int main() {
  return 0;
}
```

编写 Makefile 文件。

第 2 步: *Makefile*

```
kilo: kilo.c
	$(CC) kilo.c -o kilo -Wall -Wextra -pedantic -std=c99
```

Makefile 中一定要使用制表符. 命令中的参数:

- `$(CC)` 是一个 make 会展开的变量, 默认是 cc.
- `-Wall` 代表 “所有警告”, 并让编译器在看到程序中的代码时向你发出警告, 这些代码在技术上可能没有错误, 但被认为是 C 语言的错误或有问题的用法, 例如在初始化变量之前使用变量.
- `-Wextra` 和 `-pedantic` 会打开更多警告. 对于本教程中的每个步骤, 如果你的程序能够编译通过, 除了在某些情况下出现 “未使用的变量” 警告外, 它不应产生任何警告. 如果你收到任何其他警告, 请检查以确保你的代码与该步骤中的代码完全匹配.
- `-std=c99` 指定我们正在使用的 C 语言标准的确切版本, 即 C99. C99 允许我们在函数内的任何地方声明变量, 而 ANSI C 要求所有变量都在函数或块的顶部声明.

使用 make 命令来编译程序. 运行 `./kilo` . 然后使用命令 `echo $?` 查看程序的返回值.

第二章 进入原始模式

原始模式: raw mode

接下来读取用户的按键操作.

第 3 步: *kilo.c*

```
#include <unistd.h>

int main() {
  char c;
  while (read(STDIN_FILENO, &c, 1) == 1);

  return 0;
}
```

`read()` 和 `STDIN_FILENO` 来自 <unistd.h> . `read()` 从标准输入中读取 1 个字节到变量 c 中, 在 while 循环中一直读取, 直到没有可以读取的字节. read() 返回读取的字节数, 并在到达文件末尾时返回 0.

当运行 `./kilo` 时, 终端会连接到标准输入, 因此键盘的输入会被读取到变量 c 中. 但是, 默认情况下终端以 **规范模式**[1] 启动. 在规范模式下, 键盘输入仅在用户按下 **回车键**时发送到我们的程序. 这对许多程序都很有用: 它允许用户输入一行文本, 这样可以使用 **退格键**来修复错误, 直到文本完全按照想要的方式输入, 最后按 **回车键**将其发送到程序. 但它不适用于具有更复杂用户界面的程序, 如文本编辑器. 我们希望在每个按键输入时都对其进行处理, 以便我们可以立即做出响应.

我们想要的是 **原始模式**. 不幸的是, 没有简单的开关可以将终端设置为原始模式. 原始模式是通过关闭终端中的许多标志位来实现的, 我们将在本章的过程中逐渐做到这一点.

要退出上述程序, 请按 Ctrl-D 以告知 read() 它已到达文件末尾. 或者我们始终可以按 Ctrl-C 以发出立即终止进程的信号.

2.1 按下 q 键退出

为了演示规范模式是如何工作的, 我们将让程序在读取到用户的 q 按键操作时退出.

第 4 步: *kilo.c*

```
#include <unistd.h>

int main() {
  char c;
  while (read(STDIN_FILENO, &c, 1) == 1 && c != 'q');
```

[1] canonical mode

```
6
7   return 0;
8 }
```

要退出程序, 必须键入一行包含 q 的文本, 然后按回车键. 程序将一次一个字符地快速读取文本行, 直到读取到 q , 此时 while 循环将停止且程序将退出. q 之后的任何字符都将在输入队列中保持未读状态, 你可能会在程序退出后看到该输入被送入你的 shell .

2.2 关闭回显 (echo)

我们可以通过以下方式设置终端的属性.

1. 使用 tcgetattr() 将当前属性读入结构体 raw.
2. 手动修改结构体 raw .
3. 将修改后的结构体 raw 传递给 tcsetattr() 以写回新的终端属性. 让我们尝试以这种方式关闭 ECHO 功能.

第 5 步: *kilo.c*

```c
#include <termios.h>
#include <unistd.h>

void enableRawMode() {
  struct termios raw;

  tcgetattr(STDIN_FILENO, &raw);

  raw.c_lflag &= ~(ECHO);

  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

int main() {
  enableRawMode();

  char c;
  while (read(STDIN_FILENO, &c, 1) == 1 && c != 'q');
  return 0;
}
```

struct termios, tcgetattr(), tcsetattr(), ECHO, TCSAFLUSH 都来自 <termios.h> 头文件.

ECHO 的功能是使你键入的每个键都打印到终端, 因此你可以看到你正在键入的内容. 这在规范模式下很

有用，但当我们试图在原始模式下仔细呈现用户界面时，它确实会妨碍我们．所以我们关掉它．该程序与上一步中的程序执行相同的操作，只是不打印你输入的内容．如果你曾经不得不在终端输入密码，例如在使用 sudo 时，你可能会熟悉这种模式．

程序退出后，在 shell 里面，你可能会发现终端仍然没有回显你键入的内容．别担心，它仍然会监听你输入的内容．只需按 Ctrl-C 开始新的一行输入到你的 shell，然后输入 reset 并按回车键．在大多数情况下，这会将你的终端重置为正常．如果失败，可以随时重新启动终端模拟器．我们将在下一步中解决整个问题．

termios 可以通过 tcgetattr() 将终端属性读入结构体 raw ．修改终端的属性之后，可以使用 tcsetattr() 将修改后的属性应用到终端．TCSAFLUSH 参数指定何时应用更改：在这种情况下，它等待所有待处理的输出被写入终端，并丢弃任何尚未读取的输入．

c_lflag 字段用于“本地标志位”．其他标志位字段是 c_iflag(输入标志位)，c_oflag(输出标志位) 和 c_cflag(控制标志位)，所有这些我们都必须修改以启用原始模式．

ECHO 是一个 bitflag，二进制表示是：00000000000000000000000000001000．我们对它使用按位非运算符 ( ) 来获取 11111111111111111111111111110111．然后我们将该值与标志位字段按位与，这会强制标志位字段中的第四位变为 0，并导致每隔一位保留当前值．像这样翻转位在 C 语言中很常见．

2.3 退出时禁用原始模式

让我们善待用户，并在我们的程序退出时恢复他们终端的原始属性．我们使用变量 orig_termios 来保存终端的原始属性，并在程序退出时用于将其应用于终端．

第 6 步： *kilo.c*

```
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>

struct termios orig_termios;

void disableRawMode() {
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
}

void enableRawMode() {
  tcgetattr(STDIN_FILENO, &orig_termios);
  atexit(disableRawMode);

  struct termios raw = orig_termios;
  raw.c_lflag &= ~(ECHO);

  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

int main() { ... }
```

2.4 关闭规范模式

ICANON 标志位允许我们关闭规范模式. 这意味着我们最终将逐字节读取输入, 而不是逐行读取.

第 7 步: *kilo.c*

```c
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>

struct termios orig_termios;

void disableRawMode() { ... }

void enableRawMode() {
  tcgetattr(STDIN_FILENO, &orig_termios);
  atexit(disableRawMode);
  struct termios raw = orig_termios;
  raw.c_lflag &= ~(ECHO | ICANON);
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

int main() { ... }
```

ICANON 来自 <termios.h>. 输入标志位 (c_iflag 中的标志位) 通常以 I 开始, 例如 ICANON. 但是, ICANON 不是输入标志位, 它是 c_lflag 字段中的 “本地” 标志位. 所以这很令人困惑.

现在程序将在你按下 q 时立即退出.

2.5 显示按键

为了更好地了解原始模式下的输入是如何工作的, 让我们在 read() 时, 打印出输入的每个字节. 我们将打印每个字符的 ASCII 值, 以及它所代表的字符 (如果它是可打印字符).

第 8 步: *kilo.c*

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>

struct termios orig_termios;
```

```
8
9  void disableRawMode() { ... }
10
11 void enableRawMode() { ... }
12
13 int main() {
14   enableRawMode();
15
16   char c;
17   while (read(STDIN_FILENO, &c, 1) == 1 && c != 'q') {
18     if (iscntrl(c)) {
19       printf("%d\n", c);
20     } else {
21       printf("%d ('%c')\n", c, c);
22     }
23   }
24
25   return 0;
26 }
```

iscntrl() 来自 <ctype.h>, printf() 来自 <stdio.h>.

iscntrl() 测试一个字符是否是一个控制字符. 控制字符是我们不想打印到屏幕上的不可打印字符. ASCII 码 0-31 都是控制字符, 127 也是控制字符. ASCII 码 32-126 都是可打印的. (查看 ASCII 表以查看所有字符.)

printf() 可以打印一个字节的多种表示形式. %d 告诉它将字节格式化为十进制数 (它的 ASCII 码), %c 告诉它直接将字节作为一个字符输出.

这是一个非常有用的程序. 它向我们展示了各种按键如何转换为我们读取的字节. 大多数普通按键直接转换为它们所代表的字符. 但是试着看看当你按下 **方向键**或者 **退出键**, 或者 Page Up, 或者 Page Down, 或者 Home, 或者 End , 或者 **退格键**, **删除键**, 或者 **回车键**时会发生什么. 尝试使用带有 Ctrl 的组合键, 例如 Ctrl-A, Ctrl-B 等组合键.

你会注意到一些有趣的事情.

- 方向键, Page Up, Page Down, Home 和 End 会向终端输入 3 或 4 个字节: 27, '[', 然后是其他一两个字符. 这称为转义序列. 所有转义序列都以一个字节 27 开头. 按下 **退出键**发送一个 27 字节作为输入.
- **退格键**是字节 127. Delete 键是一个 4 字节的转义序列.
- **回车键**是字节 10, 这是一个换行符, 也称为'
  n'.
- Ctrl-A 是 1, Ctrl-B 是 2, Ctrl-C 是... 哦, 那程序就终止了, 对吧. 但 Ctrl 组合键似乎确实将字母 A-Z 映射到了代码 1-26 .

顺便说一句, 如果你碰巧按下 Ctrl-S, 你可能会发现你的程序似乎被冻结了. 你所做的是你已经要求你的程序停止向你发送输出. 按 Ctrl-Q 告诉它继续向你发送输出.

此外, 如果你按下 Ctrl-Z , 你的程序将暂停到后台. 运行 fg 命令将其带回前台.

2.6 关闭 Ctrl-C 和 Ctrl-Z 信号

默认情况下, Ctrl-C 会向当前进程发送 SIGINT 信号使其终止, 并向当前进程 Ctrl-Z 发送 SIGTSTP 信号使其挂起. 让我们关闭这两个信号的发送.

第 9 步: *kilo.c*

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>

struct termios orig_termios;

void disableRawMode() { ... }

void enableRawMode() {
  tcgetattr(STDIN_FILENO, &orig_termios);
  atexit(disableRawMode);
  struct termios raw = orig_termios;
  raw.c_lflag &= ~(ECHO | ICANON | ISIG);
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

int main() { ... }
```

ISIG 来自 <termios.h>. 就像 ICANON, 它虽然以 I 开头, 但不是输入标志位.
现在按下 Ctrl-C 可以按字节读取 3, 按下 Ctrl-Z 也可以按字节读取 26.

2.7 禁用 Ctrl-S 和 Ctrl-Q

默认情况下, Ctrl-S 和 Ctrl-Q 用来控制软件流. Ctrl-S 会停止将数据传输到终端, 直到你按下 Ctrl-Q . 这起源于很久之前, 在那时你可能想要暂停数据传输以使打印机等设备赶上来. 让我们关闭该功能.

第 10 步: *kilo.c*

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>
```

```
6
7  struct termios orig_termios;
8
9  void disableRawMode() { ... }
10
11 void enableRawMode() {
12   tcgetattr(STDIN_FILENO, &orig_termios);
13   atexit(disableRawMode);
14
15   struct termios raw = orig_termios;
16   raw.c_iflag &= ~(IXON);
17   raw.c_lflag &= ~(ECHO | ICANON | ISIG);
18
19   tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
20 }
21
22 int main() { ... }
```

IXON 来自 <termios.h>. I 这次真的代表 “输入标志位”(和我们之前看到的其他标志位不同). XON 来自两个控制字符 Ctrl-S 和 Ctrl-Q 产生的名字: XOFF 负责暂停传输, XON 负责重启传输.

现在 Ctrl-S 可以按字节读取 19, Ctrl-Q 也可以按字节读取 17.

2.8 禁用 Ctrl-V

在一些系统上, 当你键入 Ctrl-V 时, 终端会等你输入另一个字符, 然后发送字符的字面量. 例如, 在我们禁用 Ctrl-C 前, 你可能可以按下 Ctrl-V 然后再按下 Ctrl-C 来输入字节 3. 我们可以使用 IEXTEN 标志位来关闭这个功能.

第 11 步: kilo.c

```
1  #include <ctype.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <termios.h>
5  #include <unistd.h>
6
7  struct termios orig_termios;
8
9  void disableRawMode() { ... }
10
```

```
11 void enableRawMode() {
12   tcgetattr(STDIN_FILENO, &orig_termios);
13   atexit(disableRawMode);
14
15   struct termios raw = orig_termios;
16   raw.c_iflag &= ~(IXON);
17   raw.c_lflag &= ~(ECHO | ICANON | IEXTEN | ISIG);
18
19   tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
20 }
21
22 int main() { ... }
```

IEXTEN 来自 <termios.h>. 这又是一个以 I 开头但是属于 c_lflag 字段的标志位.
Ctrl-V 现在可以被读取为字节 22 了. Ctrl-O 也可以被读取为字节 15 了.