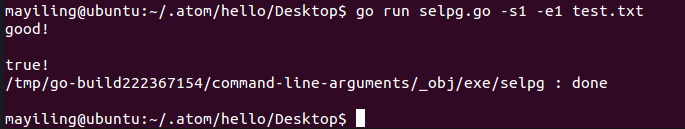
设计思想：

全代码设计思想沿袭C语言源码。

1.

$ selpg -s1 -e1 input\_file



注意：这里的test.txt里的内容为good!\n\ntrue!

结果正确，与C语言源代码一致

2.$ selpg -s1 -e1 < input\_file

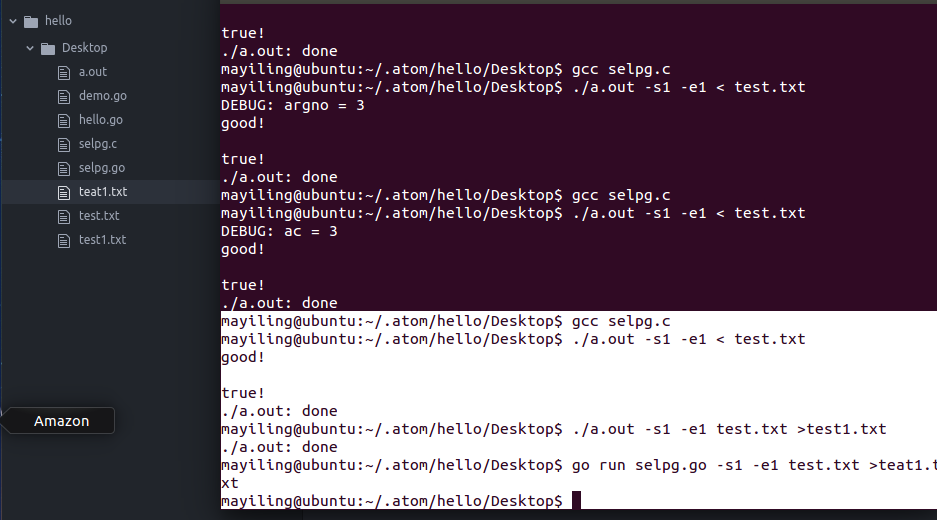
这一步出现了错误

由于我在程序中，不同的条件下赋予fin不同的值，导致后面无法用前面得到的fin来进行读取文件的操作

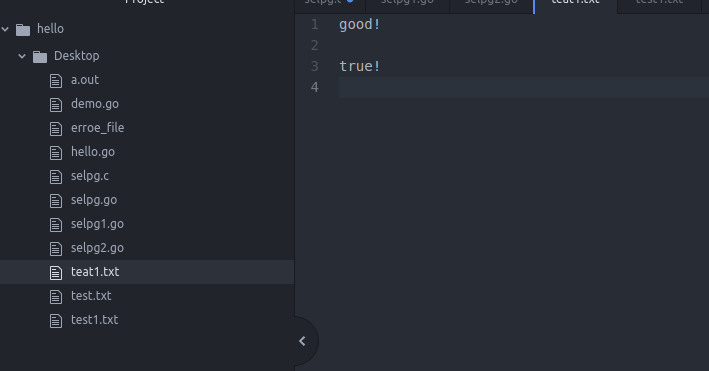
3.$ other\_command | selpg -s10 -e20

“other\_command”的标准输出被 shell／内核重定向至 selpg 的标准输入。将第 10 页到第 20 页写至 selpg 的标准输出（屏幕）。

4.$ selpg -s10 -e20 input\_file >output\_file



可以看到同C语言一样生成了txt，生成的teat.txt如图：

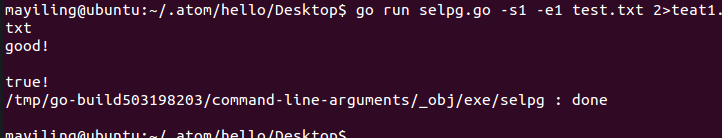


与原txt一致

5$ selpg -s10 -e20 input\_file 2>error\_file

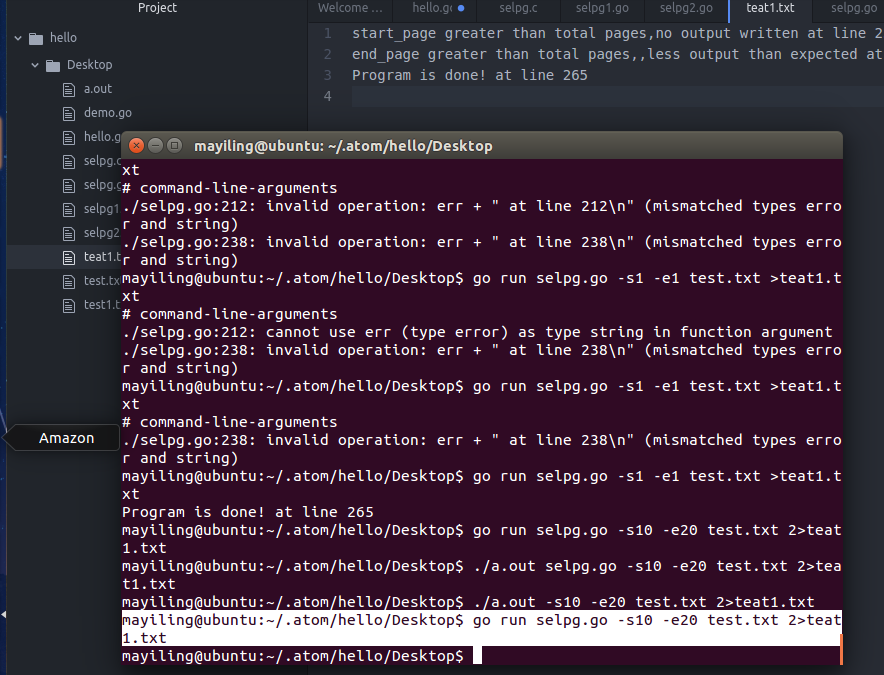
selpg 将第 10 页到第 20 页写至标准输出（屏幕）；所有的错误消息被 shell／内核重定向至“error\_file”。请注意：在“2”和“>”之间不能有空格；这是 shell 语法的一部分（请参阅“man bash”或“man sh”）。

当start\_page和end\_page为1时，如图：



此时teat.txt为空

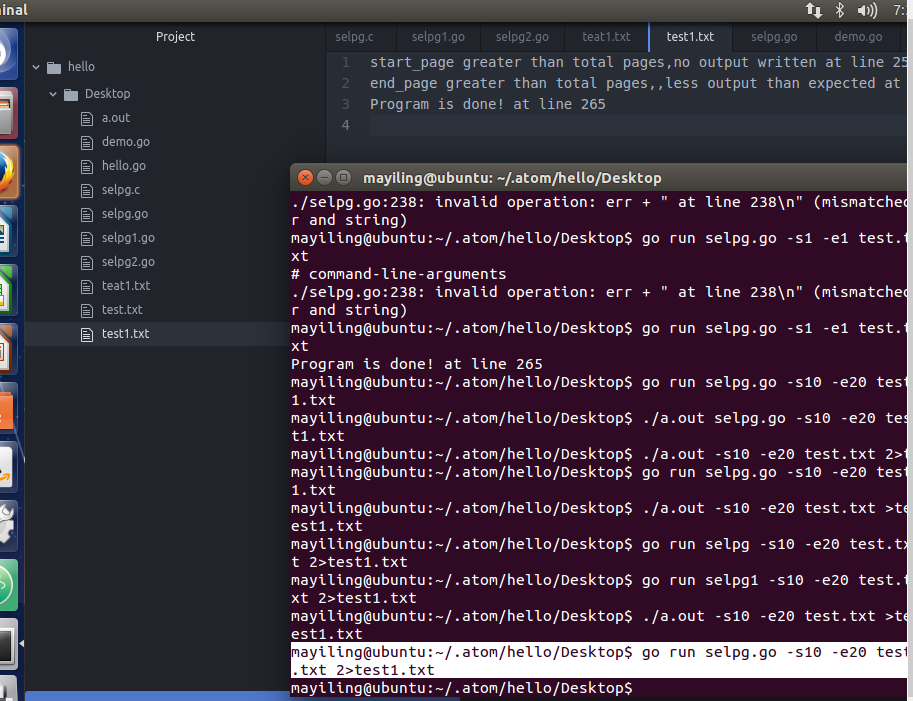
当start\_page和end\_page为10，10时，如图：



结果正确

6.$ selpg -s10 -e20 input\_file >output\_file 2>error\_file

selpg 将第 10 页到第 20 页写至标准输出，标准输出被重定向至“output\_file”；selpg 写至标准错误的所有内容都被重定向至“error\_file”。



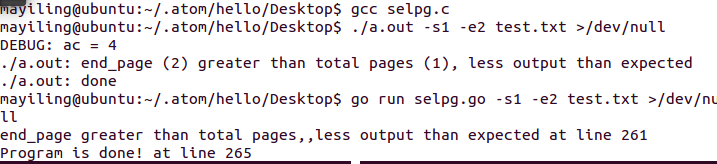
结果正确

7.$ selpg -s10 -e20 input\_file >output\_file 2>/dev/null

selpg 将第 10 页到第 20 页写至标准输出，标准输出被重定向至“output\_file”；selpg 写至标准错误的所有内容都被重定向至 /dev/null（空设备），这意味着错误消息被丢弃了。设备文件 /dev/null 废弃所有写至它的输出，当从该设备文件读取时，会立即返回 EOF。

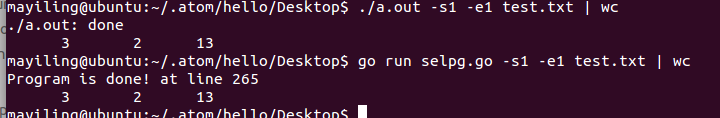
8.$ selpg -s10 -e20 input\_file >/dev/null

selpg 将第 10 页到第 20 页写至标准输出，标准输出被丢弃；错误消息在屏幕出现。这可作为测试 selpg 的用途，此时您也许只想（对一些测试情况）检查错误消息，而不想看到正常输出。



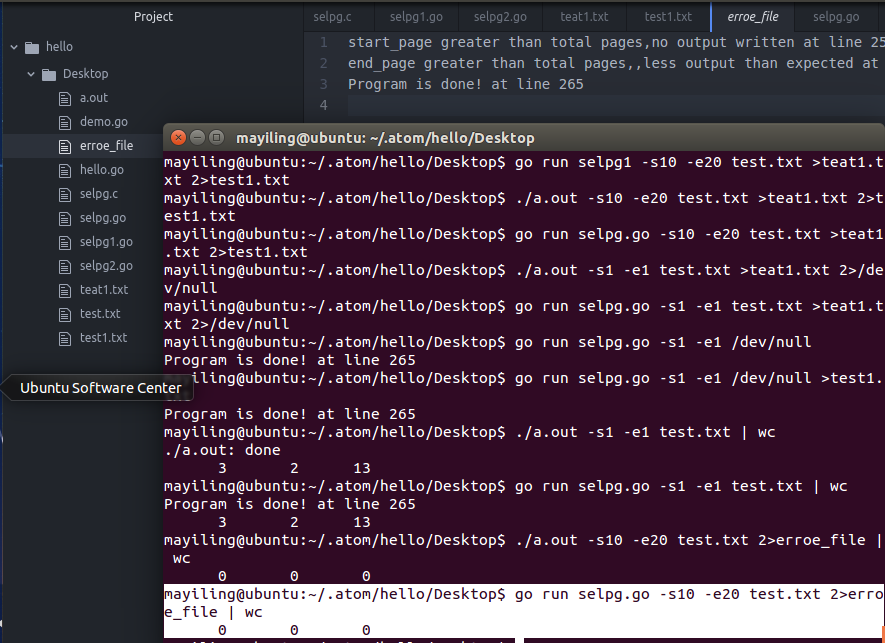
对照c语言和GO语言的代码，结果正确

9.$ selpg -s10 -e20 input\_file | other\_command



可以看到c语言和go语言结果一样

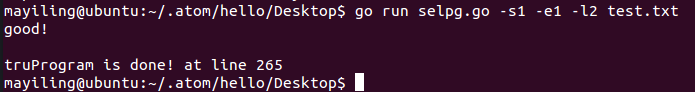
10.$ selpg -s10 -e20 input\_file 2>error\_file | other\_command



结果正确

11.$ selpg -s10 -e20 -l66 input\_file

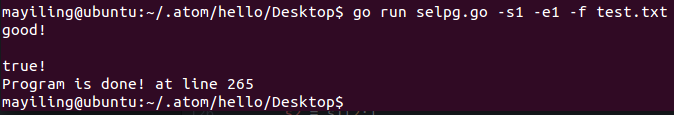
该命令将页长设置为 66 行，这样 selpg 就可以把输入当作被定界为该长度的页那样处理。第 10 页到第 20 页被写至 selpg 的标准输出（屏幕）。



结果正确

12.$ selpg -s10 -e20 -f input\_file

假定页由换页符定界。第 10 页到第 20 页被写至 selpg 的标准输出（屏幕）

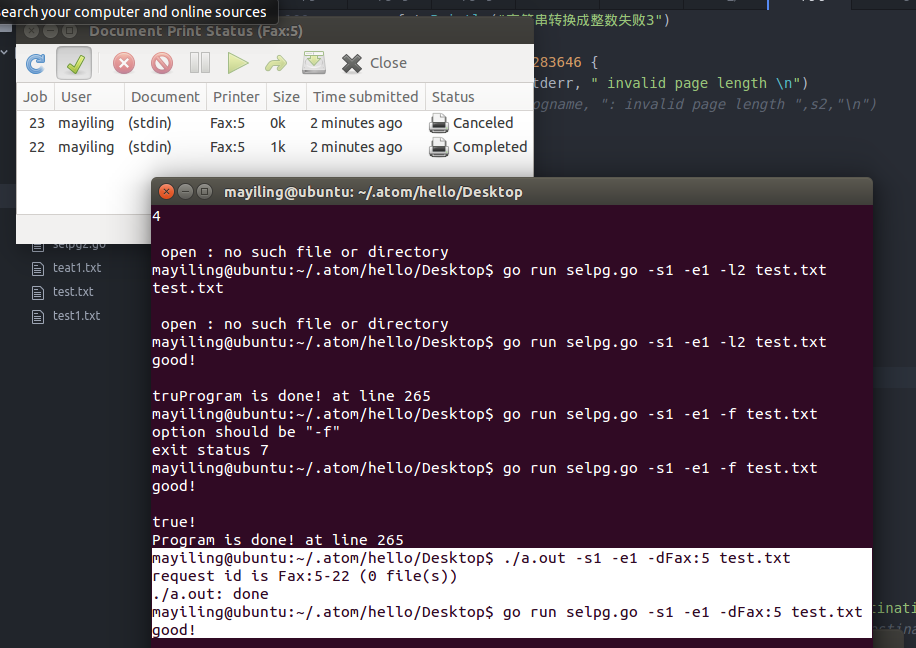


结果正确

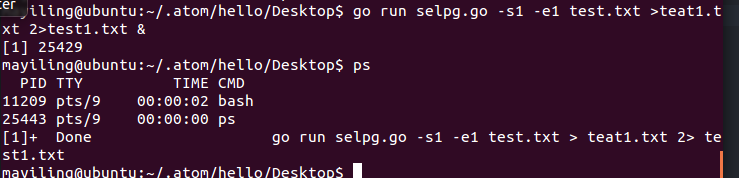
13.$ selpg -s10 -e20 -dlp1 input\_file

注意：这里用的是虚拟打印机

可以看到，打印队列中出现了要打印的东西但是打印到一半就被取消了，这与c源代码不同



14.$ selpg -s10 -e20 input\_file > output\_file 2>error\_file &



结果正确