# MiniEd

> *以下我们规定一行文本内容不超过 512 个字符，文件名不超过 100 个字符。这两个常量分别对应源文件中*
> *ONE_LINE_MAX_SIZE 宏 和 MAX_FILE_NAME 宏*

## 截止日期 2021-12-29 23:55

Ed 是一个非常古老的行文本编辑器，曾经被用于文本文件的创建、显示、更改和其他一些操作。

在本次实验中，我们将实现一个 Ed 编辑器的功能子集，称为 MiniEd。

Ed 有三种模式，刚运行时 Ed 处于命令模式（command mode）。此时输入到 Ed 的文本会被当做命令进行处理。通过一些特殊命令，Ed 可以被切换到输入模式（input mode），在输入模式下输入的文本会被当做文件内容进行保存。另外一些命令可以使 Ed 进入修改模式（change mode），在修改模式下只可以输入一行文本用来替换当前行的内容，然后退出修改模式，进入命令模式

### 命令与符号解释

在以下列出的Ed命令中，每个`?`为一个数字 n，表示第 n 行。其值应该在第一行到最后一行之间（包含第一行和最后一行）。

每个`?,?`表示一个区间，其中`?`的含义同上。如`1,2`表示第一行到第二行；`5,11`表示第五行到第十一行；`6,6`表示第六行。应保证区间起点不大于终点，**数字与逗号之间紧密相间，没有其他任何字符**。

每个命令为一行输入，以下为需要实现的命令说明。**当前行号为 0 代表目前编辑器中没有行**

> - `a`

*在当前行后追加行，输入此命令后就进入了 输入模式 。退出输入模式需要输入只有一个点的行。每输入一行当前行号会被设置为刚刚输入的行。假设现在编辑器中已经有了以下语句，当前行号为 2，已经进入 输入模式*

```
ustc
yyds
```

*输入*

```
yysy
1958
.
```

*最终文本改变为*

```
ustc
yyds
yysy
1958
```

*当前行号被设置为 4。*

- `rm ?,?`

*删除位于 `?,?` 中的行（包含左右两个端点）。如果在删除区间的后面还有行，那么当前行号设置为紧随删除区间后第一个行（显然行号是删除后的行号）；如果在删除区间后没有行，那么当前行号设为紧排在删除区间前第一个行*

*假设现在已有以下文本，并且已经输入了 `rm 1,2`*

```
ustc
yyds
yysy
1958
```

*最终文本变成*

```
yysy
1958
```

*当前行号设置为 1 （yysy 的行号）*

- `i`

*在当前行前插入行，输入此命令后就进入了 输入模式 。退出输入模式需要输入只有一个点的行。每输入一行当前行号会被设置为刚刚输入的行。假设现在编辑器中已经有了以下语句，当前行号为 1，已经进入 输入模式*

```
yysy
1958
```

*输入*

```
yyds
ustc
.
```

*最后文本变成*

```
ustc
yyds
yysy
1958
```

*当前行被设置为 1 （ustc 的行号）*

- `c`

*修改当前行的内容。输入此命令后进入修改模式。先输出`Now you are changing Line $(line)`，`$(line)` 代表现在的行号，然后输出回车。接着输入一行文本后退出修改模式进入命令模式。当前行号不变。如果当前行号为 0 ，则报 范围错误 。假如现在编辑器中已经有了以下语句，当前行号为 1，已经进入修改模式*

```
ustc
yyds
yysy
1958
```

*则命令输出为*

```
Now you are changing Line 1
```

*接着你输入*

```
flxg
```

*最终文本变为*

```
flxg
yyds
yysy
1958
```

- `q`

*退出编辑器，不做任何其他处理*

- `w file`

*把所有行写入文件中 （`file` 是写入的文件名）。当前行号设置为写入的最后一行。*

- `r file`

> *清空编辑器并从文件读入新的行（`file` 是读取的文件名）。当前行号设置为读入的最后一行。*
>
> - `ls ?,?`
>
> *打印位于区间中的所有行（包含左右端点）与对应的行号，每一行先输出行号，然后输出一个 TAB （`\t`），再输出本行的内容。当前行号设置为打印输出的最后一行。如果只输入一个端点，则只打印该行。*
>
> - `la`
>
> *打印全部行，打印要求与 `ls` 指令相同。当前行号设置为打印输出的最后一行*
>
> - `?`
>
> *修改当前行号为输入的数字（`?` 代表一个整数），不做其他修改*

以下为完整的输入样例

```
cmd> a
It's input mode now. Quit with a line with a single dot(.)
ustc
yyds
.
cmd> la
1    ustc
2    yyds
cmd> i
It's input mode now. Quit with a line with a single dot(.)
yysy
1958
.
cmd> la
1    ustc
2    1958
3    yysy
4    yyds
cmd> c
```

```
Now you are changing Line 4
flxg
cmd> ls 3,4
3    yysy
4    flxg
cmd> rm 3,4
cmd> c
Now you are changing Line 2
int main(void) { printf("Hello, world\n"); return 0; }
cmd> la
1    ustc
2    int main(void) { printf("Hello, world\n"); return 0; }
cmd> 1
cmd> c
Now you are changing Line 1
#include <stdio.h>
cmd> la
1    #include <stdio.h>
2    int main(void) { printf("Hello, world\n"); return 0; }
cmd> q
GoodBye!
```

## 错误报告

Ed 包含以下几种错误报告 （一旦发现错误就不执行该指令，继续读入下一条指令）

- 区间错误

  输入命令的左端点大于右端点，报错信息为 `Wrong range: [$(left), $(right)]`

  `$(left)` 代表输入的左端点，`$(right)` 代表输入的右端点

- 范围错误

  输入区间不在合理范围（超过最大行号或者小于 1），报错信息为 `Out of range: [1, $(total)]`

  `$(total)` 代表最大行号

- 文件错误 （选做）

在打开文件或者写入文件时发生错误，报错信息为 `failed to open or write $(file)`

`$(file)` 代表文件名

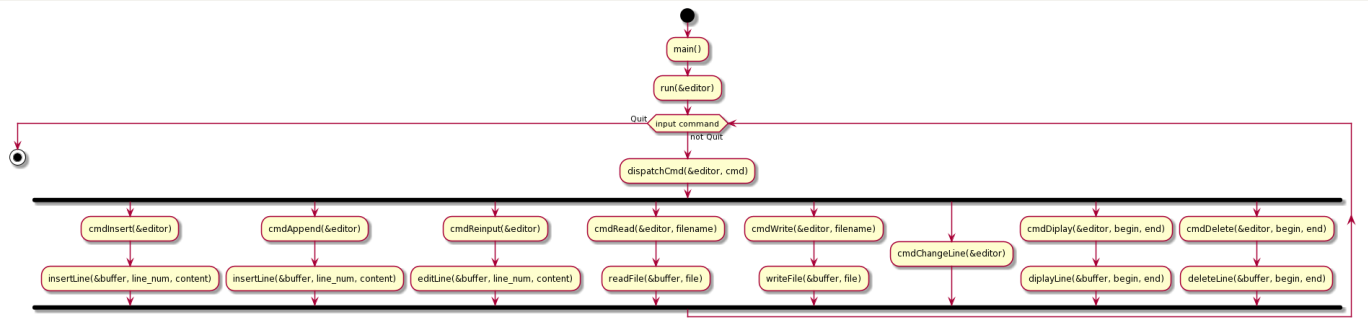
- 指令错误

  输入未知或错误的指令，报错信息为 `Bad/Unknown command $(cmd)`

  `$(cmd)` 代表指令

## 程序大体逻辑

本实验的部分代码已经给出，共六个文件：`line.c`、`line.h`、`editor.c`、`editor.h`、`info.h` 和`minied.c`。其中`line.c/h`、`editor.c/h`分别定义了两个结构体 `Line` 和 `Editor` 和其他相关函数。`minied.c`中包含了`main`函数，并在其中建造并运行了一个 `Editor` 实例。在 `run` 函数中，程序会不断读入命令，使用 `dispatchCmd` 函数对命令进行解析并调用相应的函数（`cmdXXX`）。`Line` 结构体用于保存所有输入的文本，其中每行文本保存在一个`char` 类型的数组中，所有的行保存在一个链表中。`Line` 结构体定义了各种用于操作和展示文本的函数，供 `Editor` 调用。`info.h` 提供了调试用的宏




## 编译运行与调试

首先，确保你已经安装了编译器，没安装的安装群文件软件安装包中的 tdm-gcc-10.3.0-2.exe ，安装过程不要修改任何选项。

输入以下命令

```
# Windows
cd minied文件夹的全路径 (比如 C:\Users\someone\Downloads\minied)
mingw32-make run

# macOS
cd minied文件夹的全路径 (比如 /Users/someone/Downloads/minied)
make SYSTEM=MAC run
```

如果你使用的是 VSCode，那直接在下方的终端中输入命令即可；否则使用快捷键 `Win + r`，在出现的窗口中输入 `powershell` 然后回车，在随后出现的 powershell 窗口中输入相应命令。输入命令后就可以编译运行程序。

关于调试，你可以自行配置你使用的软件，或者使用 printf 大法；助教同时也在 `info.h` 文件中提供了调试用的 info 宏

它的用法如下

```c
#include "info.h"
int foo(void) {
    return 42;
}
int main(void) {
    int i = 0;
    char str[10] = {'a', 'b'};
    info(i, str[0], foo());
}
// 会输出
// main.c:8:
//     i = 0
//     str[0] = a
//     foo() = 42
```

代表在 `main.c` 文件的第 8 行，i、str[0]、foo() 的值各为多少，这样就不用你自己去写一些复杂的 printf，节省时间

注意，info 宏不支持结构体

## 任务

代码:

1. 完成代码框架中所有注释 `TODO` 的地方 （文件读写均为选做）

   （注释 `TODO` 的地方可以添加多行代码，不允许修改除此之外的任何地方）
2. 使用提供的 `error` 函数实现对以上提到四种错误的报告 （打印错误信息，文件错误同样选做）

   `error` 函数使用方法与 `printf` 完全相同，并且会自动在输出结尾加换行符
3. 读懂整个代码框架中除了 `error` 函数和 `info.h` 的其他地方，当面检查会有提问

## 给分规则

总分 30 分，实现了文件读写和文件错误报告可获得附加分 5 分 （指令的完整实现包括了错误报告）

| 内容 | 分值 |
| --- | --- |
| 实现 `a` 和 `i` 指令 | 5 |
| 实现 `c` 指令 | 5 |
| 实现 `rm` 指令 | 5 |
| 实现 `ls` 与 `la` 指令 | 5 |
| 实现 `?` 指令 | 5 |
| 理解代码（助教提问） | 5 |
| 实现 `w` 和 `r` 指令 （选做） | 5 （附加分） |