

浙江大学实验报告

专业: 计算机科学与技术

姓名: 耿晨

学号: 3190104958

日期: 2020 年 4 月 13 日

课程名称: C 程序设计专题

指导老师: 翁恺

实验名称: 行编辑器

1 实验题目要求

本题要求我们实现一个 GUI 窗口内的单行的文本输入编辑器。

基础需求有: 支持左右方向键、回退和删除键、ins 切换插入和覆盖状态, 编辑过程中有光标闪烁, 插入和覆盖状态的光标形状不同。回车后, 结束输入, 将输入的内容在标准输出输出。

由于学有余力, 在完成了基础需求的基础上, 我又给自己添加了以下几个需求:

- (a) 实现多行编辑 (为了不和基础需求中的要求相冲突, 我将 Shift+Enter 组合键设置为了换行键, 而单独的 Enter 作为基础需求中标准输出输出的指令)。
- (b) 实现使用 Ctrl+S 来进行文件的保存, 并且可以在文件编辑过程中多次进行保存。
- (c) 支持键盘所有字符, 以及支持 Shift 切换大小写及不同符号。
- (d) 支持上下方向键和 Home 键, End 键
- (e) 添加欢迎界面。

最终, 该程序完美完成了以上所有需求。具体展示在本报告的第五节。

2 实验思路 and 过程描述

由于本题中, 我们需要实现的是一个中小型的 Win32 图形界面程序, 所以, 为了我们编写代码以及维护方便, 我们可以采用 MVC 架构进行开发, 同时, 应尽量使用面向对象程序设计的思想来编写代码。不过, 由于 C 语言的语言限制, 所以并无法使用过多的面向对象特性。

MVC 架构分为三个部分, 三个部分各自独立又互相联系, 通过分别开发好这三个模块并将它们耦合起来, 我们就可以完成应用程序的开发。第一部分是 M, 也就是 Model, 在我的理解中, 就是数据存储的位置, 是各种操作的对象, Model 只能由 Control 部分操作, 由

View 部分读取。第二部分是 C，也就是 Control，负责接收外界的操作，并调用 Callback Function 来对 Model 中的数据进行操作。第三部分是 V，也就是 View，通过读取 Model 中的数据绘制图形。

MVC 架构的一个关键就是：Control 部分和 View 部分**不能**直接进行耦合，中间必须要经过 Model 层，否则会带来代码编写中的极大不便。

由于本次开发属于敏捷开发，我使用了 TDD (Test-Driven Development, 测试驱动开发) 的方式进行开发工作，通过分别进行一个个模块和 feature 的测试来驱动每一部分的开发。

在本次实验中，V 层和 M 层都是好写的，简要思路如下：

对于 View 层，在每次刷新时，我们用一个矩形清空窗口，再，我们直接输出文本即可，关于光标，需要注意以下两个问题。首先，我们应当注意使用等宽字体，这样每个字符的宽度相同，我们才有可能确定光标的位置，这里，我们选用的是 Consolas 字体，其次，对于光标的绘制，我们可以使用画一个黑色矩形的方式来代表光标，按照编辑器的惯例，我们用较窄的光标表示插入状态，用较宽的光标代表覆盖状态。

View 层还有一个工作是显示欢迎界面和保存界面，我们在此不展开，将在第四章详细阐述。

对于 Model 层，存储了各个属性和元素，直接实现即可，具体 Model 层的结构，将会在第三章和第四章进行详细解释。

由于 View 层和 Model 层代码量不大，在本项目中，我们并没有采取额外的文件存储这两个模块。后续如果要重构，我们或许会考虑将其分成多个文件。

关键是 Control 层的开发。Control 层需要处理两个回调。

首先是计时器回调，用于实现光标闪烁。为了实现这个功能，我们可以直接设置一个 500 毫秒的计时器，然后每次计时器 callback 时，将 model 中 cursorStat 这个属性反转，这样就可以实现光标闪烁。

其次是键盘的响应。键盘按键大致可以分为两种，一种是值键，代表了一个字符，比如字母、数字、符号等，都属于此类。另外一种是功能键，如方向键，ins 键等等。

对于值键的实现，比较简单，具体会在第三章第四章详细讲解，在此不再赘述。

对于功能键，有两类，一类是状态类功能键，比如 shift, ctrl, CapsLock 等，他们会和其他的键构成组合键，因此，我们在 model 中开始一个域来存储这个键是否被按下。比如 isCtrl 表示 Ctrl 键是否被按下。另一类是操作类功能键，比如方向键等等，这类键需要我们具体地处理 model 中相关的量，由于如果在此处阐述，那么会和后续章节有重合，所以在此不再赘述。

3 基础部分实验代码解释

我们仍然按照 model, view, control 的顺序, 讲解基础部分代码。

3.1 Model

首先是 model。原型如下。

```
1 typedef struct b{
2     int width;
3     int height;
4     char str[MAXN][MAXN];
5     int cursorStat;
6     int cursorPos;
7     int isInsert;
8     int fontSize;
9     int isCap;
10    int endWelcome;
11    int line;
12    int totLine;
13    int isCtrl;
14    int isSaving;
15    char saveStr[MAXN][MAXN];
16    int saveLines;
17    int saveLine;
18    int saveCursorPos;
19 }Model;
20 Model model;
```

其中有一部分是扩展部分用到的属性, 我们留待那一部分讲解。此处只讲解基础部分用到的属性。

- (a) width, height: 分别是窗口的宽度和高度
- (b) str[]: str[i][j] 代表第 i 行的第 j 个字符, i 从 1 开始, j 从 0 开始。
- (c) cursorStat: 光标状态, 用于实现光标的闪烁。如果为 0, 代表光标不显示, 如果为 1, 代表光标显示。
- (d) cursorPos: 代表本行中光标的位置。设其值为 i, 那么光标在第 i 个字符之前。
- (e) isInsert: 代表是否处于覆盖状态。1 代表处于覆盖状态, 0 代表处于插入状态。
- (f) fontSize: 代表字体大小。

(g) isCap: 代表是否大写。

M 层还包括字符集代码以及初始化代码，由于比较简单在此不赘述。

3.2 View

```
1 void refresh() {
2     if(!model.endWelcome) return;
3     beginPaint();
4     setBrushColor(WHITE);
5     rectangle(0, 0, model.width, model.height);
6     int currentX = 10;
7     int currentY = 10;
8     for(int i = 1; i <= model.totLine; i++){
9         setTextSize(model.fontSize);
10        setTextFont("Consolas");
11        paintText(currentX, currentY, model.str[i]);
12        currentY += 100;
13    }
14    currentY = 10;
15    currentX += (model.cursorPos - 1) * (model.fontSize - 10) * 0.55 - 5;
16    currentY += 100 * (model.line - 1);
17    if(model.cursorPos > 17) currentX -= 10;
18    if(model.cursorStat == 1) {
19        setBrushColor(BLACK);
20        if(model.isInsert) {
21            rectangle(currentX, currentY, currentX + (model.fontSize - 10) * 0.55 +
22                        5, currentY + 80);
23        }
24        else {
25            rectangle(currentX, currentY, currentX + 5, currentY + 80);
26        }
27    }
28    endPaint();
29 }
```

本函数为刷新屏幕，根据 model 数据重新绘制屏幕的函数。

略去为了实现进阶特性而写的代码，只解释基础部分代码。

首先使用一个白色的大矩形覆盖屏幕来将屏幕清空，然后使用 paintText 函数绘制文字。接下来绘制光标，currentX 和 currentY 代表光标的坐标，通过查阅资料可得对于 Consolas 字体，宽度大致是字体大小的 0.55 倍。但是，实际上这个比例不太准，为了适应实际

的比例，我调整了一下具体的参数，从而产生了几 Magic Number，这些数字怎么出来的我也无法解释。总之，经过微调，以上的代码即可完美绘制出光标。

3.3 Control

Control 分为计时器和键盘回调。下分别阐述。键盘回调的函数过长（165 行），所以分成几个小节。

3.3.1 计时器回调

```
1 void timer(int id) {  
2     model.cursorStat ^= 1;  
3     refresh();  
4 }
```

每次计时器回调时，直接将 cursorStat 取反，再刷新屏幕。

3.3.2 插入字符

插入字符的核心是 addChar 函数，即将字符 ch 插入光标所在的位置。

```
1 void addChar(char ch) {  
2     char tempstr[MAXN];  
3     strcpy(tempstr, model.str[model.line]);  
4     for (int i = 0; i < model.cursorPos - 1; i++) {  
5         model.str[model.line][i] = tempstr[i];  
6     }  
7     model.str[model.line][model.cursorPos - 1] = ch;  
8     for (int i = model.cursorPos - 1; i < strlen(tempstr); i++) {  
9         model.str[model.line][i + 1] = tempstr[i];  
10    }  
11    model.str[model.line][strlen(tempstr) + 1] = 0;  
12    model.cursorPos++;  
13 }
```

由于本题数据量比较小，所以我用了比较 dirty 的做法。建立一个临时的字符串，然后把当前行拷贝过去，然后先拷贝光标前面的那一部分，然后把字符插进去，然后把光标后面的部分搞过来。最后要注意把光标加一。

覆盖函数的写法类似，请读者自行查看源代码。

3.3.3 删除字符

```
1 char tempstr[MAXN];
2 strcpy(tempstr, model.str[model.line]);
3 for (int i = 0; i < model.cursorPos - 2; i++) {
4     model.str[model.line][i] = tempstr[i];
5 }
6 for (int i = model.cursorPos - 1; i < strlen(tempstr); i++) {
7     model.str[model.line][i - 1] = tempstr[i];
8 }
9 model.str[model.line][strlen(tempstr) - 1] = 0;
10 model.cursorPos--;
```

上面所展示的是 backspace 部分的代码，与插入部分写法类似，直接建立一个临时的字符串，把当前行拷贝过去，先拷贝光标前面的部分去掉最后一个字符，再把光标后面的拷贝回来即可。注意要把光标值减一。

Delete 函数写法类似，请读者自行查阅源代码。

3.3.4 基础功能键

基础功能键包括左右方向键和回车键以及 Insert 键。

对于左右方向键：

```
1 else if(key == 0x25) { //Left arrow
2     if(model.cursorPos > 1) {
3         model.cursorPos--;
4     } else {
5         if(model.line > 1) {
6             model.line--;
7             model.cursorPos = strlen(model.str[model.line]) + 1;
8         }
9     }
10 }
```

直接移动光标即可。注意边界条件。

对于回车键：

```
1 for(int i = 1; i <= model.totLine; i++) {
2     printf("%s", model.str[i]);
3     puts("");
4 }
5 exit(0);
```

直接输出字符串，并退出程序即可。

对于 insert 键：

```
1 else if(key == 0x2D) {  
2     model.isInsert ^= 1;  
3 }
```

直接改变 model 中的值就可以了。
至此，我们完成了所有基础部分的功能。

4 拓展部分实验代码解释

同样采用 model, view, control 的顺序进行阐述。对于已经在第三章阐述过的内容，在此章将不会提及。

4.1 Model

```
1 typedef struct b{  
2     int width;  
3     int height;  
4     char str[MAXN][MAXN];  
5     int cursorStat;  
6     int cursorPos;  
7     int isInsert;  
8     int fontSize;  
9     int isCap;  
10    int endWelcome;  
11    int line;  
12    int totLine;  
13    int isCtrl;  
14    int isSaving;  
15    char saveStr[MAXN][MAXN];  
16    int saveLines;  
17    int saveLine;  
18    int saveCursorPos;  
19 }Model;
```

我们解释一下之前没有解释的几个属性。

- (a) endWelcome: 代表是否结束欢迎界面，如果值为 1 表示结束了欢迎，反之没有。
- (b) line: 代表光标所在的行数。
- (c) totLine: 代表文件总共有多少行。

- (d) isCtrl: 代表是否按下 Ctrl 键。
- (e) isSaving: 代表是否处于保存状态。
- (f) saveStr: 用来临时保存需要储存的字符串。
- (g) saveLines: 代表保存操作之前，文件总共有多少行。
- (h) saveLine: 代表保存操作之前，光标所在的行数。
- (i) saveCursorPos: 代表保存操作之前，光标所在的位置。

4.2 View

4.2.1 Refresh

```
1 void refresh() {
2     if(!model.endWelcome) return;
3     beginPaint();
4     setBrushColor(WHITE);
5     rectangle(0, 0, model.width, model.height);
6     int currentX = 10;
7     int currentY = 10;
8     for(int i = 1; i <= model.totLine; i++){
9         setTextSize(model.fontSize);
10        setFont("Consolas");
11        paintText(currentX, currentY, model.str[i]);
12        currentY += 100;
13    }
14    currentY = 10;
15    currentX += (model.cursorPos - 1) * (model.fontSize - 10) * 0.55 - 5;
16    currentY += 100 * (model.line - 1);
17    if(model.cursorPos > 17) currentX -= 10;
18    if(model.cursorStat == 1) {
19        setBrushColor(BLACK);
20        if(model.isInsert) {
21            rectangle(currentX, currentY, currentX + (model.fontSize - 10) * 0.55 +
22                        5, currentY + 80);
23        }
24        else {
25            rectangle(currentX, currentY, currentX + 5, currentY + 80);
26        }
27    }
28    endPaint();
```



```
28 }
```

为了实现多行的 refresh，在渲染光标时，我们还需要去调整 currentY，这里，我们使用 100 作为每行的高度，进行渲染。

4.2.2 欢迎界面

```
1 void showWelcome() {
2     beginPaint();
3     setBrushColor(WHITE);
4     rectangle(0, 0, model.width, model.height);
5     setTextSize(40);
6     setFont("Consolas");
7     paintText(260, 200, "ACEIM - AClib based Editor IMproved");
8     paintText(20, 300, "Welcome! Before use, please switch your input mode to
9         ENGLISH!");
10    paintText(300, 400, "Press SPACE to enter the software");
11    endPaint();
12 }
```

以上是欢迎界面的代码，直接先清空，然后输出三行文字即可。

4.2.3 保存界面

为了方便实现保存界面，我直接采用更改 model 中的 str 的方式，显示提示词，这样可以减少代码编写量，所以保存这一部分大部分代码应当属于 control。

4.3 Control

4.3.1 大小写切换以及键盘字符全覆盖

为了实现大小写切换的功能，我们写了以下代码：

```
1 //code omitted
2 else if(key == 0xA0 || key == 0xA1 || key == 0x10) {
3     model.isCap ^= 1;
4 }
5 //code omitted
6 else if(event == KEY_UP) {
7     if(key == 0xA0 || key == 0xA1 || key == 0x10) {
8         model.isCap ^= 1;
9     }
10 }
```

通过这两段代码，就可以实现，shift 按下和释放的时候，isCap 的值都会取反。

```
1  if(key >= 0x41 && key <= 0x5A) {
2      if (model.isCap == 0) {
3          ch = key - 0x41 + 'a';
4      } else {
5          ch = key - 0x41 + 'A';
6      }
7      if((ch == 's' || ch == 'S') && model.isCtrl) {
8          showSaveScreen();
9          return;
10     }
11     processChar(ch);
12 }
```

通过在回调的时候，判断此时 isCap 的值，就可以方便地得到，是小写还是大写。
为了实现键盘字符的全覆盖，我们写了以下代码：

```
1  char numberMapSymbol[20] = {')', '!', '@', '#', '$', '%', '^', '&', '*', '('};
2  char keyMap1[20] = {';', '=', ',', '-', '.', '/', '`'};
3  char keyMapCap1[20] = {':', '+', '<', '_', '>', '?', '~'};
4  char keyMap2[20] = {'[', '\\', ']', '\\'};
5  char keyMapCap2[20] = {'{', '|', '}', '\\'};
```

调用时，

```
1  else if(key >= 0xBA && key <= 0xC0) {
2      if(model.isCap == 0) ch = keyMap1[key - 0xBA];
3      else ch = keyMapCap1[key - 0xBA];
4      processChar(ch);
5  }
```

通过这样一个 map（数组实现），就可以方便地实现键盘字符的全覆盖。

4.3.2 多行意义下的功能键处理

对于单行编辑器，功能键的处理是简单的，但是如果是多行，就要多考虑很多问题。

4.3.3 左右方向键

以左方向键为例：

```
1  else if(key == 0x25) { //Left arrow
2      if(model.cursorPos > 1) {
3          model.cursorPos--;
4      } else {
```

```
5     if(model.line > 1) {
6         model.line--;
7         model.cursorPos = strlen(model.str[model.line]) + 1;
8     }
9 }
10 }
```

如果当前并不是行的开头，那么直接移动就好了，但是，如果已经处于某一行的开头，那么我们需要回到上一行。注意判断，如果已经在第一行，那么是不能移动的。

4.3.4 上下方向键

以上方向键为例：

```
1 else if(key == 0x26) { //Up arrow
2     if(model.line > 1) {
3         model.line--;
4         if(model.cursorPos > strlen(model.str[model.line]) + 1) {
5             model.cursorPos = strlen(model.str[model.line]) + 1;
6         }
7     }
8 }
```

首先判断当前是否可以往上移动，如果可以，那么光标上移。接下来，判断光标的水平位置。显然，如果上一行比当前光标位置短，那么我们需要移动光标到行末尾。

4.3.5 Backspace, 删除键

```
1 else if(key == 0x08) { //backspace button
2     if(model.cursorPos > 1) {
3         char tempstr[MAXN];
4         strcpy(tempstr, model.str[model.line]);
5         for (int i = 0; i < model.cursorPos - 2; i++) {
6             model.str[model.line][i] = tempstr[i];
7         }
8         for (int i = model.cursorPos - 1; i < strlen(tempstr); i++) {
9             model.str[model.line][i - 1] = tempstr[i];
10        }
11        model.str[model.line][strlen(tempstr) - 1] = 0;
12        model.cursorPos--;
13    } else {
14        if(model.line > 1) {
15            model.line--;
```

```
16         model.cursorPos = strlen(model.str[model.line]) + 1;
17         strcat(model.str[model.line], model.str[model.line+1]);
18         for(int i = model.line+1; i < model.totLine; i++) {
19             strcpy(model.str[i], model.str[i+1]);
20         }
21         model.totLine--;
22     }
23 }
24 }
```

如果当前光标不在最开始,那么直接按照第三章那样操作既可。如果光标在最开始,就要把当前行与上一行进行合并,然后把之后的行都向前移动一行,最后减少 totLine。

delete 是类似地,请读者自行查阅。

4.3.6 Enter 键

```
1  else if(key == 0x0D) { //Enter Button
2      if(model.isSaving == 1) {
3          //detail omitted
4      }
5      if(model.isCap == 0){
6          for(int i = 1; i <= model.totLine; i++) {
7              printf("%s", model.str[i]);
8              puts("");
9          }
10         exit(0);
11     } else {
12         model.totLine++;
13         for(int i = model.totLine; i >= model.line + 2; i--) {
14             strcpy(model.str[i], model.str[i-1]);
15         }
16         strcpy(model.str[model.line+1], model.str[model.line] + model.cursorPos - 1)
17             ;
18         model.str[model.line][model.cursorPos - 1] = 0;
19         model.line++;
20         model.cursorPos = 1;
21     }
```

第一部分是保存部分的代码,在此不赘述。

第二部分,如果是直接按下 Enter,那么就直接在标准输出输出文本,然后退出程序。如果是 Shift+Enter,那么就是正常换行,那么,光标后面的部分需要另起一行。这里

的实现，我们直接截断上一行，然后把光标后面的另成一行，然后后面每一行都拷贝到后面一行。

4.3.7 保存文件

当 ctrl+S 被按下，调用以下函数：

```
1 void showSaveScreen() {
2     model.isSaving = 1;
3     for(int i = 1; i <= model.totLine; i++) {
4         strcpy(model.saveStr[i], model.str[i]);
5     }
6     model.saveLine = model.line;
7     model.saveCursorPos = model.cursorPos;
8     model.saveLines = model.totLine;
9     model.totLine = 2;
10    strcpy(model.str[1], "Save the file in:");
11    model.str[2][0] = 0;
12    model.line = 2;
13    model.cursorPos = 1;
14 }
```

先将 isSaving 置成 1，表示此时正在保存。然后将缓冲区的数据全部保存到 saveStr, saveLine 等中，之后把 str 改成提示语以及输入框。

当用户输入文件名并按下回车，调用：

```
1 if(model.isSaving == 1) {
2     FILE* fp = fopen(model.str[2], "w");
3     for(int i = 1; i <= model.saveLines; i++) {
4         fprintf(fp, "%s", model.saveStr[i]);
5         fprintf(fp, "\n");
6     }
7     fclose(fp);
8     model.isSaving = 0;
9     for(int i = 1; i <= model.saveLines; i++) {
10        strcpy(model.str[i], model.saveStr[i]);
11    }
12    model.totLine = model.saveLines;
13    model.line = model.saveLine;
14    model.cursorPos = model.saveCursorPos;
15    refresh();
16    return;
17 }
```

新建一下这个文件，然后把缓冲区内的数据全部写入文件，最后把保存的临时数据全部写回缓冲区。

4.3.8 Misc

Home 键和 End 键的实现是简单的，直接移动光标：

```
1 else if(key == 0x24) {  
2     model.cursorPos = 1;  
3 } else if(key == 0x23) {  
4     model.cursorPos = strlen(model.str[model.line])+1;  
5 }
```

可以使用一个循环实现 TAB 键（对，我是四空格党 O(^_^)O）。

```
1 else if(key == VK_TAB) {  
2     for(int i = 1; i <= 4; i++) processChar(' ');  
3 }
```

5 实验成果展示

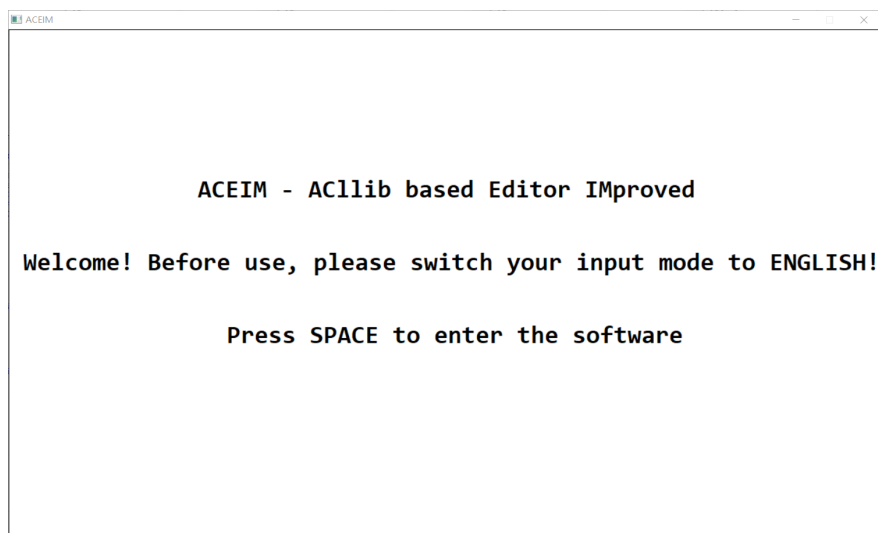
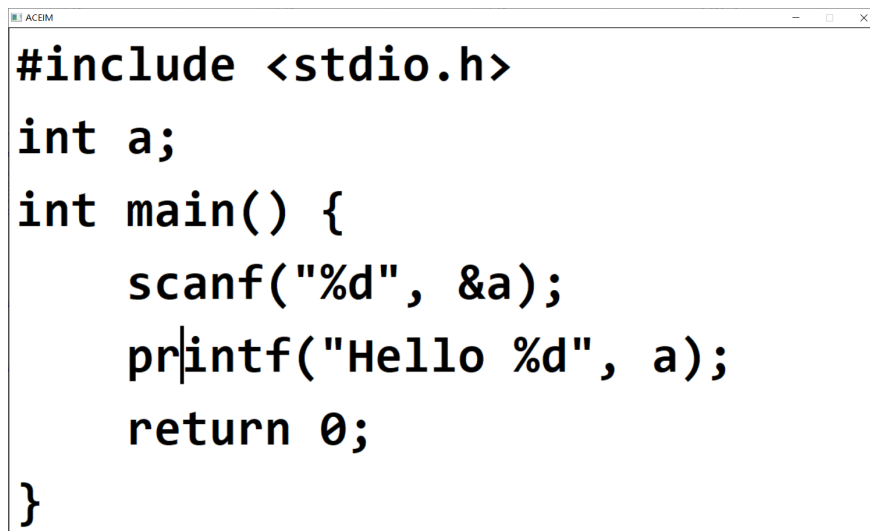


图 1: 欢迎界面

如图 1 所示，为程序进入后的欢迎界面。

这个界面是仿照 VIM 制作的。（事实上，本来想开发一个仿制的 vim，不过因为工作量太大只能放弃）

在这个界面输入任何按键都可以直接进入编辑。

A screenshot of a window titled 'ACEIM'. The window contains the following C code:

```
#include <stdio.h>
int a;
int main() {
    scanf("%d", &a);
    printf("Hello %d", a);
    return 0;
}
```

图 2: 多行文本编辑

注意一定要切换到英文输入法！不然会出现严重的 bug。

图 2 所示界面就是正常的编辑界面。在这个界面可以支持市面上常见的编辑器的绝大多数特性，包括但不限于：

- (a) 正常的方向键移动。
- (b) 键盘上所有英文字母和字符的支持。
- (c) Home, end, insert, delete, backspace 等功能键
- (d) 直接按 Enter 键可以将缓冲区内容输出到 stdout，同时程序会退出，按 Shift+Enter 可以换行（类似 QQ 等 IM 软件的设计）（众所周知，VIM 是新手最难退出软件的编辑器，而 ACEIM 则是新手最难不退出软件的编辑器（笑））
- (e) 按 ctrl+S 键可以进行保存。

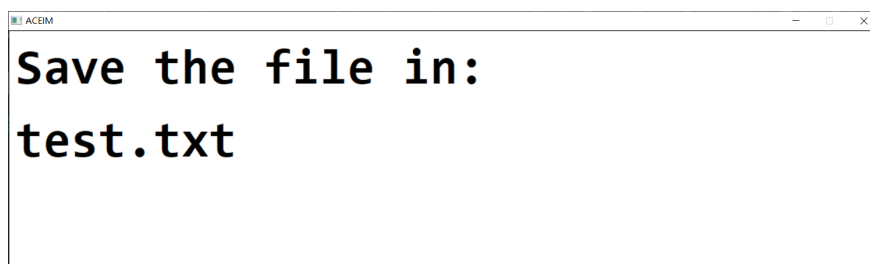


图 3: 保存文件

图 3 所示为保存界面。在这个界面，请输入你想保存到的文件名，系统会自动新建一个文件保存缓冲区内的所有内容。

输入文件名后回车，文件将会自动保存。保存后会自动回到之前的编辑界面。

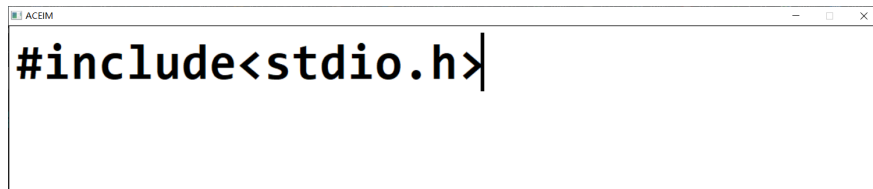


图 4: 单行编辑并标准输出

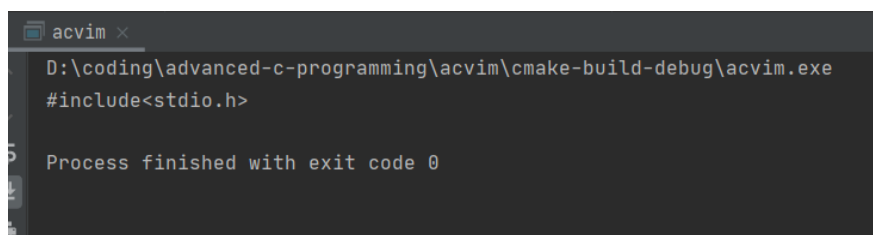


图 5: 单行编辑并标准输出

图四图五所示为基础要求中的回车将缓冲区内容输出到 stdout 的功能。

6 未来开发计划

未来本项目计划添加以下几个特性。

- (a) 支持中文。
- (b) 支持文本的选定、复制与粘贴。
- (c) 支持代码高亮与补全。
- (d) 支持鼠标操作。
- (e) 增加页面的滚动。
- (f) 实现 VIM 的部分功能。

同时，由于 C 语言本身的限制，未来如果可能，本项目计划使用 C++ 重构。

7 实验体会和心得

这个项目是我第一次使用 C 语言开发图形界面的程序。之前其实已经开发过不少 GUI 程序，比如使用 electron 等前端框架，C++ 的 qt 框架，python 的 pygame 库等等。C 语

言和他们相比，更为基础和底层，由于少了面向对象的特性，以及少了许多好用的轮子，在编写和维护代码的时候往往会更加辛苦些。

尽管如此，C 语言也有其优势，最突出的优势就是 C 语言的运算速度无疑是我上述提到的语言中最快的。在本次程序编写过程中，我全程使用数组这种比较 dirty 的方法实现，并没有使用链表。并不是因为我不知道链表会带来效率的提升，而是我认为根本没有必要，原因有三，第一，因为我对于 C 语言的执行效率是有自信的，本次编辑器的数据量也不大，dirty 的方法易于编写，易于维护。第二，链表的写法不一定就快。虽然，链表在很多操作复杂度都是 $\Theta(1)$ ，数组的操作复杂度大多是 $\Theta(n)$ ，但是，我们分析复杂度还要考虑一个东西：常数。链表是使用指针进行操作，其常数是比较大的。在 n 比较小的情况下，链表是不一定有优势的。第三，链表使用指针编写，编写不慎就会产生野指针，对系统有潜在的危害性。

在本次开发过程中，我再一次认识到了 TDD（测试驱动开发）和 OOP 等先进的软件工程思想和 MVC 框架这样的设计模式的好处。采用了这些思想之后，我可以更加高效的设计和维持代码。

在 ACLLIB 的帮助下，本次实验我可以专心于核心代码的编写。总的来说，在这次实验中，我有很大的收获。