浙江大学实验报告

姓名：赵泓珏　　**学号**：3190104515　　**日期**：2020.4.14　　**成绩**：____

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

一、　实验题目要求

本题要求实现一个 GUI 窗口内的单行的文本输入编辑器。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中输出。

二、　实验思路和过程解释

在实现行编辑器的过程中，最核心的问题有三个：传入信息的接收和转换，字符的存储以及人与电脑的交互。解决了这三个核心问题，行编辑器的框架就会基本实现，此时只需对细节稍加完善即可。

1. 传入信息的接收和转化

在这里采用 acllib 库中的有关回调函数，获取键盘和鼠标传入的信息，检测到对应内容后进行转换，通过其他函数实现功能。

2. 字符的存储

考虑到未知用户输入字符个数，在这里采用链表对有关信息进行存储。为了尽可能的方便，我们采用第一次大程作业中的 linkedlist 库，并将其进行适当修改，使之更适合行编辑器。

3. 人与电脑的交互

在这里需要及时把用户的有关输入以及光标的位置、状态告诉用户便于进行调整。同样采用 acllib 中有关函数实现字符的打印以及设置光标的位置和状态。

三、　实验代码解释

此次大程作业中共涉及一个 main.c 文件和三个非标准库（acllib 库，row 库和 edit 库）。在解释之前需要说明的是为了使解释更加连贯流畅很多部分与具体代码的编写顺序不同，同样为了便于批阅理解在此次实验报告中也酌情复制粘贴了部分代码，还请见谅。接下来先对 main.c 进行解释。

1. main.c

1) 定义的有关宏

```
#define WinWidth 1200
#define WinHeight 100
#define charsize 100
#define delta 51
```

WinWidth：窗口宽度；

WinHeight：窗口高度；

charsize：在 GUI 窗口中字体的大小；

delta：经过实测测得当字体高度大小为 100（charsize）时字体大概的宽度。

2) 定义的全局变量

```
int i;//used to know where the caret is
List list={NULL,NULL};
int caretwid=3;
```

i：用于对光标进行定位以及定位目前光标所在位置对应的结点便于插入删除；

list：用于存储用户输入的字符；

caretwid：通过变量名实现注释，即光标宽度，主要涉及到 insert 键对光标状态的转化，详见 INSERT()函数。

在这里定义全局变量的目的是用于传值。虽然采用全局变量进行传值是一

件十分危险的事情，但是这也是不得已而为之。因为在回调函数中函数的参数类型被严格定义，无法更改（也主要是我的水平有限不能直接更改回调函数），但是在回调函数中有些函数的功能实现需要用到这些参数，因此只得在 main.c 中采用全局变量实现部分函数的传值。同时遵循尽量不使用全局函数的原则，在这里只在 main.c 中使用全局变量而不在头文件中进行声明，降低风险。

3) 直接定义在 main.c 中的函数

```
void keyboard_listener(int key,int event);
void char_listener(char key);
void mouse_listener(int x,int y,int button,int event);
void time_listener(int timeID);
```

这四个函数主要用于实现第一个核心问题：传入信息的接收和转换。采用回调函数读入用户在键盘、鼠标上输入的信息，并将其进行转换，对应到有关函数。

```
void keyboard_listener(int key,int event){
    if(event==0){
        switch(key){
            case 37: LEFT(&i);break;//left
            case 39: RIGHT(&list,&i);break;//right
            case 13: ENTER(&list,&i);break;//enter
            case 45: INSERT(&caretwid,i);break;//insert
            case 46: deletekey(&list,&i);break;//delete
            case 8: BACKSPACE(&list,&i);break;//backspace;
            case 35: END(&list,&i);break;//end
            case 36: HOME(&i);break;//home
        }
    }
    putCaret(i);
}
```

void keyboard_listener(int key,int event)主要用于检测用户按下的有关功能键。首先确保键盘上的键处于按下状态（即 event==0）。利用 switch-case 语句，通过检测按下键的虚拟键码（存在 key 中），对应到具体函数。这些函数的有关内容详见之后 edit.c 的代码解释中。除题目要求外在这里增加了 home 和

end 的功能。每当按下一个键都调用 putCaret()函数确保光标的位置得以更新。

```
void char_listener(char key){
    if(key!=13&&key!=8){//is not backspace or enter
        if(caretwid==3)list_getin(&list,key,i);
        else{
            if(i==list_size(&list))list_getin(&list,key,i);
            else list_set(&list,i,key);
        }
        print(&list);
        i++;
    }
    putCaret(i);
}
```

void char_listener(char key)用于获取用户输入的字符信息。在正式代码编写前的测试注意到，enter 键和 backspace 键会对输入进行干扰，所以一开始先排除按下的键 enter 键和 backspace 键的情况。之后对光标状态进行检测（即光标宽度），如果为插入状态，则调用 list_getin()函数在光标所指位置插入结点；如果为覆盖状态，则检测光标所处位置，如果覆盖了某个字符（即光标位置 i 小于链表长度则调用 list_set()函数实现对应结点字符内容的修改；如果未覆盖字符（即光标位置 i 等于链表长度）则继续插入字符，从而实现光标状态不同时输入状态的不同。每次发生字符变动都调用 print()函数在 GUI 窗口上重新打印链表，同时令 i 加 1 实现光标的重新定位（因为每次输入一个字符光标都会向后移动一个字母的宽度）。同样每次用户按下键时都调用 putCaret()函数更新光标位置。

```
void mouse_listener(int x,int y,int button,int event){
    if(button==1&&event==0){
        int cnt=list_size(&list), j;
        for(j=0;j<cnt;j++){
            if(x>=j*delta&&x<(j+1)*delta){
                i=j;
                break;
            }
```

```
        }
        if(x>=cnt*delta){
            i=cnt;
        }
    }
    putCaret(i);
}
```

void mouse_listener(int x,int y,int button,int event)用于获取用户鼠标输入的信息。在用户单击左键（即 button==1&&event==0）时用 for 循环判断鼠标点击的位置，鼠标点击的位置大于当前输入字符的长度在之后的 if 语句中实现。检测到鼠标点击的位置后更新 i 的值即光标应该所在的位置，通过调用 putCaret()函数实现光标位置的更新。在这里通过 cnt 来表示链表长度，同时当鼠标点击一个字符时默认光标移到这个字符之前的位置。此函数为为了使得行编辑器功能更加完善的附加功能，题目中并无要求。

```
void time_listener(int timeID){
    static flag=1;
    setCaretSize(caretwid,100);

    if(flag==1){
        showCaret();
        flag=0;
    }
    else{
        hideCaret();
        flag=1;
    }
}
```

void time_listener(int timeID)主要用于控制光标的闪烁。虽然 acllib 中使光标出现时光标会自己闪烁六次，但是在这里为了实现光标的持续闪烁定义了这样的函数。利用静态局部变量生存周期为整个程序的性质，在一开始定义静态局部变量 flag 来表示光标的亮灭状态。之后设定光标的宽度和高度，符合要求。然后采用 if 语句检测，flag 为 1 时光标出现，同时使 flag 变为 0；flag 为 0

时隐藏光标，同时使 flag 变成 1。以此实现光标的定时闪烁。

4) main 函数中的有关代码

main 函数里主要是有关大小定义以及有关回调函数的内容，基本上没有需要解释的部分，在此不再赘述。实验代码如下。

```
    initConsole();

    initWindow("row_edit",DEFAULT,DEFAULT,WinWidth,WinHeight);

    beginPaint();
    setTextSize(charsize);

    startTimer(0,500);
    registerKeyboardEvent(keyboard_listener);
    registerCharEvent(char_listener);
    registerMouseEvent(mouse_listener);
    registerTimerEvent(time_listener);
    endPaint();
    return 0;
```

2. row.h 和 row.c

该库改编自上次大程作业中的 linkedlist。为了贴合行编辑器（Row edit）的主题将其重新命名为 row.h 和 row.c，同时对其中的部分函数进行了修改。鉴于和 linkedlist 有很大重复部分在这里只解释发生更改的部分。

与之前不同的地方首先在于在 row.h 中没有采用前向声明而是直接在头文件对用到的两个结构体 Node 和 List 进行了声明。原因在于在之后的 edit.c 中有对 Node 的一些操作。为了通过函数功能以及核心要解决的问题来对函数进行更好的归类并没有把 edit.h 中涉及到 Node 的有关函数移到 row.c 中。同时把结点存储的数据类型改为 char，原来 linkedlist.c 中的有关内容也进行相应的修改。

此外在 linkedlist.c 的基础上增加了几个函数，接下来对这几个函数的有关

代码进行解释。

a) Node* list_find(List *list, int index)

```
Node* list_find(List *list, int index){
    int i=1;
    Node* p=list->head;
    while(i<index){
        i++;
        p=p->next;
    }
    return p;
}
```

该函数用于找到第 i 个结点。需要说明的时虽然在大部分情况下链表中结点的编号从 0 开始，但是这里的结点编号从 1 开始。这么做并非是因为日常生活中更习惯从 1 开始，而是因为一般光标都位于刚输入字符的后面，当输入一个字符后光标应该处在 1*delta 的位置，这样的话采用从 1 开始的编号系统更加合理。Index 为 0 时在说明光标处在一行的最左侧。

之后定义计数变量 i 来寻找第 index 个结点所在的位置，定义 Node 的指针 p 用于实现链表的遍历。利用 while 循环遍历，找到后直接返回 Node 的指针 p。list_find 的全部代码编写结束。

b) void list_getin(List* list,char v,int i);

```
void list_getin(List* list,char v,int i){
    if(i==0)list_insert(list,v);
    else if(i==list_size(list))list_append(list,v);
    else{
        Node* r=list_find(list,i);
        Node* p=(Node*)malloc(sizeof(Node));
        p->value=v;
        p->next=r->next;
        r->next=p;
    }
}
```

该函数用于向指定位置插入结点。首先确定需要插入结点的位置（光标所

指的位置 i)，如果 i==0 则说明要向头之前插入结点，调用之前所写的 list_insert()函数；如果 i 为链表长度则说明要向尾插入结点，则调用之前写的 list_append()函数；除此之外则说明字符插在中间的位置，首先利用 list_find()函数找到第 i 个结点，之后创建新的结点，把需要插入的字符存入新结点中，更新对应结点之间的指向关系即可。void list_getin(List* list,char v,int i)的代码编写结束。

c) void list_print(List* list);

```
void list_print(List* list){
    Node* p=list->head;
    for(;p;p=p->next){
        printf("%c",p->value);
    }
    printf("\n");
}
```

这个函数用于在标准输出中顺序打印链表所以结点存储的字符值。利用 for 循环遍历同时调用 printf()即可。在打印完一个链表中所有结点的值后进行换行便于下一次的打印。void list_print(List* list)的全部代码编写结束。

d) 除上述函数外 list_remove()的有关内容发生了更改，主要是利用 list_find()函数找到对应结点再去删除。由于原理和之前所写的 list_remove()基本一致并且在上次的分享课上很多同学已经做了很好的阐述，此处不再赘述。

3. edit.h 和 edit.c

在这个库里主要实现的是按下特定键时触发的效果。其中大部分函数的代码都十分简单，最短的甚至只有一行，单独编写函数的主要原因是实现函数功能的封装，减少用户不需要知道的信息，同时也使代码更加具有逻辑性。与此同时这部分函数的编写也需要十分小心。不同于以前的程序，这个程序大部分是通过按键来触发函数功能，有可能发生误按的情况。因此需要设置合理的边

界条件，在边界条件之外的范围内调用函数不应该对程序有任何影响，否则则会导致程序的崩溃。有关标准头文件结构等的内容多说无益，接下来直接对核心部分进行解释。

1) void LEFT(int *pi)和 void RIGHT(List *list,int *pi);

两个函数分别对应键盘上的左右键。由于具有很大的相似性放到一起来说。

```
void LEFT(int *pi){
    if(*pi>0)(*pi)--;
}

void RIGHT(List *list,int *pi){
    if(*pi<list_size(list))(*pi)++;
}
```

当按下左右键时主要对应的是光标位置的移动。因此传入 main.c 中用于对光标进行定位的 i 的指针，在 i>0&&i<list_size(list)的范围内定位光标位置，如果按下左键就令 i 减 1，按下右键就令 i 加 1。在之前的 keyboard_listener 函数中已经说明每次按下一个键都会实现光标的更新，因此只需在这两个函数中对光标的定位即可。

2) void ENTER(List* list,int *pi);

此函数主要对应 enter 键的功能。

```
void ENTER(List* list,int *pi){
    list_print(list);
    list_free(list);
    *pi=0;
    print(list);
}
```

按照要求，每当按下 enter 键就在标准输出中打印整个链表，因此调用 list_print()函数。此外我还实现了一些额外的功能。每次按下 enter 键就调用 list_free()函数清空整个链表，同时使 i 的值归 0，为了覆盖之前的痕迹在 GUI 窗

口上重新打印链表内容（调用 print()函数）。由于此时是空链表因此此时 GUI 窗口恢复空白状态。这么做的目的是使其更像对话框，同时也方便多次重复测试。

3) void BACKSPACE(List *list,int *pi)和 void deletekey(List* list,int *pi)

两个函数分别对应键盘上 Backspace 键和 Delte 键的功能，具有很大的相似性，因此放到一起来说。

```
void BACKSPACE(List *list,int *pi){
    if(*pi>0){
        list_remove(list,*pi);
        LEFT(pi);
    }
    if(list->head==NULL)list->tail=NULL;
    print(list);
}

void deletekey(List* list,int *pi){
    if(*pi<list_size(list))list_remove(list,*pi+1);
    if(list->head==NULL)list->tail=NULL;
    print(list);
}
```

Backspace 是删除光标前一个结点的值，Delete 是删除光标后一个结点的值，因此分别对对应结点调用 list_remove()删除即可。不同的地方在于按下 backspace 后光标会左移，因此调用 LEFT()函数，按下 delete 光标位置不动，不做其他处理。为了保证程序稳定运行在用 backspace 或者 delete 删除掉所有结点（即 list->head==NULL）时令 list->tail 也变成 NULL，由于此时链表内容发生变化重新在 GUI 窗口打印整个链表，两个函数的代码编写结束。

4) void HOME(int *pi)和 void END(List* list,int *pi)

两个函数分别对应键盘上的 home 键和 end 键，需要实现的功能是实现光标的跳转。光标的重新定位会在 keyboard_listener 中实现，因此在这两个函数

中只需要对光标进行定位即可。

```
void HOME(int *pi){
    *pi=0;
}

void END(List* list,int *pi){
    *pi=list_size(list);
}
```

根据尝试，按下 home 键光标会跳转到最开始，因此令 i=0；按下 end 键，光标会跳转到最后，因此令 i 为链表长度。两个函数的代码编写结束。

5) void putCaret(int i);

该函数主要用于更新光标在 GUI 界面上显示的位置。只需要根据光标应在的位置 i 利用 setCaretPos()对光标重新定位即可。void putCaret(int i)的代码编写结束。

```
void putCaret(int i){
    setCaretPos(i*delta,0);
}
```

6) void print(List* list);

该函数主要用于在 GUI 窗口打印链表所有的值，具体代码解释如下。

```
    setPenColor(EMPTY);
    setBrushColor(WHITE);
    rectangle(0,0,WinWidth,WinHeight);
```

首先将之前的打印内容覆盖。这一块的思路来源于老师的 mooc。当时在游戏设计部分提到每当发生一些事件时就将整个画面重画，在这里思路是差不多的，每次我们需要打印链表时，我们就将之前打印的内容全部覆盖，之后再在上面重新打印链表。因此一开始先定义画笔颜色为 EMPTY，brush 的颜色为 WHITE，画一个和窗口大小一样的白色长方形实现覆盖。

```
    Node* p=list->head;
    char str[2];
    int i=0;
```

```
for(;p;p=p->next){
    sprintf(str,"%c",p->value);
    paintText(i*delta,0,str);
    i++;
}
```

定义遍历用的 Node 结点 p，对文字进行定位的 int 型变量 i，以及用于存储字符信息的 str 字符串。之后利用 for 循环进行遍历，首先用 sprintf 函数把结点中存储的字符存入 str 中，之后再用 paintText()函数打印字符，同时为了下一个字符能正常显示令 i 加 1。之上涉及到的所有代码都应该处在 beginPaint()和 endPaint()之间，确保可以正常显示。至此全部代码编写结束。

四、 实验体会和心得

1. 实验总结

此次实验的核心正如一开始所说的三个问题，在解决了这三个问题后基本大框架也就基本实现。这个实验里最重要的也就在于人机交互方面，这也是我第一次做人机交互性比较强的程序。在我完成代码的过程中感觉一个很需要重视的问题就是如何维持程序运行的稳定性。不同于之前的程序，很大程度上都是按照既定流程向后推进，但是在目前的情况下可能由于用户的误操作导致实际函数调用条件不符，因此在这时需要合理设置函数能够运行的范围。此外为了使程序的编写尽可能简单，要灵活运用之前已经实现的一些代码，比如在这次就利用了上次写的 linkedlist，这次只需要适当修改便可实现所需要的功能。

2. 不足之处

在这次行编辑器的编写中，基本上我能想到的功能都写了进去，唯一比较遗憾的是没有实现当字符的总宽度大于窗口宽度时的解决方案，时间有限精力有限，只能做到这种程度，还希望之后能有机会进行完善。

此外在本次实验里实际上我原本是用带头结点的双向链表写的，并且我一直感觉实际上双向链表更适合行编辑器的编写。这样的话很多设计到定位的部分可以通过指针实现，但是在我第一版写的程序里双向链表部分总是运行出错，不知为何。出于效率考虑才想起上次写的 linkedlist，在之后我也希望能进一步完善我自己有关双向链表的知识。

3. 心得体会

实际上我在完成行编辑器的过程中一度十分奔溃……因为双向链表一直写不出，后来采用单向链表后也一直有很多 bug，调试了好久……想起上学期的老师和我们说的话：不管什么程序都会有 bug，能发现 bug 是好事，发现不了 bug 不代表程序编写成功，只是说明程序的 bug 你没发现，并且很有可能在之后的关键时刻掉链子（好吧和我上次大程作业的情况简直一模一样）。还希望之后自己 debug 的能力能够提高，因为一直找不出 bug 的感觉实在太让人难受了……不过一个好的程序也是慢慢调试出来的（深有体会，因为在这次大程作业前前后后也改了好多次代码），希望之后的自己也更有耐心吧。虽然现在水平还是不高，但是终有一天我能达到我想达到的水平的。