

数据结构大作业报告

——VIM编辑器青春版

1. 需求分析

将的需要分为以下几个模块，分而治之：

主模块：用于读入和分析指令并选择子模块进行功能跳转。

- 读取文本模块：读取指定文本文件并显示其内容，指令为:open filename，其中filename为打开的文件名。
- 保存文本模块：保存文本到文件，指令为:w filename，其中filename为保存的文件名，如文件已存在则直接覆盖。
- 退出模块：指令为:q，退出编辑器。
- 撤销操作模块：指令为u，表示进行一步撤销操作，这里的撤销是指撤销上一步的操作。比如：1. 上一步操作插入了多个字符，那么撤销是指将该次操作添加的所有字符都撤销，也就是删除。2. 上一步的操作是删除操作，那么撤销是指撤销该步"删除操作"，将删除的字符重新插入。
- 反撤销操作模块：指令为v，表示进行一步撤销撤销操作，回到撤销前的状态。
- 删除字符模块：指令为x，删除当前光标处的一个字符。
- 搜索模块：输入/pattern进行全文搜索，其中pattern为搜索的关键字，需要从光标处向后搜索直到找到第一处符合的地方，光标跳转至该位置。
- 光标移动模块：使用h j k l分别表示光标向左下上右移动。
- 文本键入模块：指令为i，主要功能为键入字符，所有输入字符都应被如实显示到光标处，使用方向键进行光标移动，按esc退出该模块。
- 光标置底模块：用于显示指令及状态。
- 光标归位模块：用于字符的输出，区别于指令及状态的显示。

2. 概要设计(部分主要功能实现思路)

1. 读入字符：选用_getch()函数，该函数为一个一个字符如实读入，不会回显，符合题目中部分回显，部分不回显的要求，要求回显的再用cout输出即可。
2. 记录文本的数据结构：一维的行文本采用string，再将string装入vector中，构成当前行列文本。再用一个vector来存此文本，表示文本的不同状态用于撤销与反撤销，前者状态-1，后者状态+1即可。文本的搜索、删除和插入则调用STL库的find、erase和insert函数，注意函数的参数。
3. 光标移动与模式输出：使用SetConsoleCursorPosition()函数，移动只需相应横纵坐标+-1即可。而模式要在底端输出，故要先把光标移至底端，输出模式后再将光标移回原位。
4. 控制台文本的更新：当每次文本被改变时，控制需要进行一次清屏操作system("cls")，再将改变后的文本输出到控制台屏幕上。要注意光标此时在文档末尾，需要把光标移动至原位。
5. 文本的读入与保存：分别使用ifstream()和ofstream()函数。
6. open模式的开启:其他模式都只需读入一个字符即可进入函数，而open模式需要键入多个字符，故需要对键入的字符进行保存以备下一次判断。只有当此次读入的字符为' '，且上一次为'n'，再上一次为'e'，再上一次为'p'，再上一次为'o'时，才会进入文件打开函数。每次只有上一次字符正确才会进行下一步，保证了open指令的唯一性。

3. 详细设计

- 主模块：

```
int main() {  
  
    int exit_ret = 0; //flag用于标记退出函数是否操作
```

```

char back_space[30];
bf.push_back(buf);
co++; //把文本初始状态作为初值赋给bf
memset(back_space, ' ', 29);
back_space[29] = '\0'; //空格文本用于行清屏
char pre_c = '!'; //pre_c赋一不用的指令初值

//读入指令
while (1) {
    char fs = _getch();
    //光标移动
    if (fs == 'h' || fs == 'j' || fs == 'k' || fs == 'l')
        move_cursor(fs);
    //删除操作
    else if (fs == 'x') {
        delete_cursor();
    }
    //撤销操作
    else if (fs == 'u') {
        paste();
    }
    //反撤销操作
    else if (fs == 'v') {
        antipaste();
    }
    else {
        string ch;
        //判断o指令后是否是open指令
        if (fs == 'o')
            pre_c = fs;
        if (fs == 'p' && pre_c == 'o')
            pre_c = fs;
        if (fs == 'e' && pre_c == 'p')
            pre_c = fs;
        if (fs == 'n' && pre_c == 'e')
            pre_c = fs;
        //文本键入模式
        if (fs == 'i') {
            bottom_cursor();
            cout << back_space;
            bottom_cursor();
            cout << fs;
            text_editor_mode();
        }
        //文本搜索模式
        else if (fs == 47) {
            bottom_cursor();
            cout << back_space;
            search_x();
        }
        //文本退出模式
        else if (fs == 'q') {
            bottom_cursor();
            cout << back_space;
            bottom_cursor();
            cout << fs;
            exit_ret = exit_editor();
        }
        //文本保存模式
        else if (fs == 'w') {
            bottom_cursor();

```

```

        cout << back_space;
        bottom_cursor();
        cout << fs;
        save_text_mode();
    }
    //o指令单独判断
    else if (fs == 'o') {
        bottom_cursor();
        cout << back_space;
        bottom_cursor();
        cout << fs;
    }
    //文本打开模式
    else if (fs == 32 && pre_c == 'n') {
        cout << fs;
        read_file();
    }
    //无法读入指令，不回显
    else if (fs >= 20 && fs <= 126) {
        if (pre_c != '!' && pre_c != 'n') cout << fs;
    }
}
//结束程序
if (exit_ret == -1)
    break;
}
return 0;
}

```

■ 光标置底模块：

```

void bottom_cursor() {
    HANDLE hOut;
    CONSOLE_SCREEN_BUFFER_INFO scr;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
    GetConsoleScreenBufferInfo(hOut, &scr);
    if (scr.dwCursorPosition.Y != scr.dwMaximumWindowSize.Y - 20) {
        pre_x = scr.dwCursorPosition.X;
        pre_y = scr.dwCursorPosition.Y;
    }
    COORD pos = { 0, scr.dwMaximumWindowSize.Y - 20 };
    SetConsoleCursorPosition(hOut, pos);
}

```

■ 光标归位模块：

```

void back_cursor() {
    HANDLE hOut;
    CONSOLE_SCREEN_BUFFER_INFO scr;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
    GetConsoleScreenBufferInfo(hOut, &scr);
    COORD pos = { pre_x, pre_y };
    SetConsoleCursorPosition(hOut, pos);
}

```

■ 读取文本模块：

```

void read_file() {

```

```

system("cls");
buf.clear();
bottom_cursor();
cout << "open ";
string file;
while (1) {
    char as = _getch();
    if (as == 13)
        break;
    cout << as;
    file.push_back(as);
}
file += ".txt";
ifstream infile;
infile.open(file.data()); //将文件流对象与文件连接起来
if (!infile.is_open()) {
    cout << "无法找到";
    return;
}
back_cursor();

string s;
int count = 0;
while (getline(infile, s)) {
    if (count > 0)
        cout << endl;
    count++;
    buf.push_back(s);
    cout << s;
}
infile.close(); //关闭文件输入流
}

```

■ 保存文本模块：

```

void save_text_mode() {
    string name;
    if (_getch() == 32) {
        cout << ' ';
        while (1) {
            char ch = _getch();
            if (ch == 13 || ch == 32) break;
            name.push_back(ch);
            cout << ch;
        }
    }
    ofstream out(name + ".txt");
    if (out.is_open()) {
        ofstream fileout(name + ".txt", ios::trunc);
        for (int i = 0; i < buf.size(); i++) {
            for (int j = 0; j < buf[i].size(); j++) out << buf[i][j];
            if (i + 1 < buf.size())
                out << endl;
        }
        out.close();
        cout << "    " << name + ".txt" + "保存完成";
        back_cursor();
    }
}
}

```

■ 退出模块:

```
int exit_editor() {
    exit(0);
    return -1;
}
```

■ 撤销操作模块:

```
void paste() {
    HANDLE hOut;
    CONSOLE_SCREEN_BUFFER_INFO scr;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
    GetConsoleScreenBufferInfo(hOut, &scr);
    int x = scr.dwCursorPosition.X;
    int y = scr.dwCursorPosition.Y;
    if (co - 1 >= 0) {
        buf = bf[co - 1];
        co--;
    }
    system("cls");
    for (int i = 0; i < buf.size(); i++) {
        for (int j = 0; j < buf[i].size(); j++) cout << buf[i][j];
        if (i + 1 < buf.size())
            cout << endl;
    }
    COORD pos = { x, y };
    if (co > 1)
        SetConsoleCursorPosition(hOut, pos);
}
```

■ 反撤销操作模块:

```
void antipaste() {
    HANDLE hOut;
    CONSOLE_SCREEN_BUFFER_INFO scr;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
    GetConsoleScreenBufferInfo(hOut, &scr);
    int x = scr.dwCursorPosition.X;
    int y = scr.dwCursorPosition.Y;
    if (co + 1 < bf.size()) {
        buf = bf[co + 1];
        co++;
    }
    system("cls");
    for (int i = 0; i < buf.size(); i++) {
        for (int j = 0; j < buf[i].size(); j++) cout << buf[i][j];
        if (i + 1 < buf.size())
            cout << endl;
    }
    COORD pos = { x, y };
    if (co > 1)
        SetConsoleCursorPosition(hOut, pos);
}
```

■ 删除字符模块:

```
void delete_cursor() {
```

```

HANDLE hOut;
CONSOLE_SCREEN_BUFFER_INFO scr;
hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
GetConsoleScreenBufferInfo(hOut, &scr);
int x = scr.dwCursorPosition.X;
int y = scr.dwCursorPosition.Y;
bf.push_back(buf);
co++;
buf[scr.dwCursorPosition.Y].erase(scr.dwCursorPosition.X, 1);
system("cls");
for (int i = 0; i < buf.size(); i++) {
    for (int j = 0; j < buf[i].size(); j++) cout << buf[i][j];
    if (i + 1 < buf.size())
        cout << endl;
}
COORD pos = { x, y };
SetConsoleCursorPosition(hOut, pos);
}

```

■ 搜索模块:

```

void search_x() {
    HANDLE hOut;
    CONSOLE_SCREEN_BUFFER_INFO scr;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
    GetConsoleScreenBufferInfo(hOut, &scr);
    bottom_cursor();
    cout << '/';
    string pattern;
    while (1) {
        char sc = _getch();
        if (sc == 13)
            break;
        cout << sc;
        pattern.push_back(sc);
    }
    int reco = 0;
    for (int i = 0; i < buf.size(); i++) {
        if (buf[i].find(pattern) != string::npos) {
            COORD pos = { buf[i].find(pattern), i };
            reco = 1;
            SetConsoleCursorPosition(hOut, pos);
            break;
        }
    }
    if (reco == 0) cout << "未找到";
}

```

■ 光标移动模块:

```

void move_cursor(char ch) {
    HANDLE hOut;
    CONSOLE_SCREEN_BUFFER_INFO scr;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
    GetConsoleScreenBufferInfo(hOut, &scr);
    COORD pos = { 0, 0 };
    short int x1 = scr.dwCursorPosition.X - 1;
    short int x2 = scr.dwCursorPosition.X + 1;
    short int y1 = scr.dwCursorPosition.Y - 1;

```

```

short int y2 = scr.dwCursorPosition.Y + 1;
if (ch == 'h')
    pos = { x1, scr.dwCursorPosition.Y };
if (ch == 'j')
    pos = { scr.dwCursorPosition.X, y2 };
if (ch == 'k')
    pos = { scr.dwCursorPosition.X, y1 };
if (ch == 'l')
    pos = { x2, scr.dwCursorPosition.Y };
short int a = max(buf.size() - 1, 0);
if (pos.Y > 0)
    pos.Y = min(pos.Y, a);
else
    pos.Y = 0;
short int b = 0;
int p = buf[pos.Y].size();
if (a > 0)
    b = max(p - 1, 0);
if (pos.X > 0)
    pos.X = min(pos.X, b);
else
    pos.X = 0;
SetConsoleCursorPosition(hOut, pos);
}

```

■ 文本键入模块：

```

void text_editor_mode() {
    back_cursor();
    //进入编辑文本模式
    while (1) {
        int input_ch = _getch();
        //insert模式下的光标移动
        if (input_ch == 224) {
            int i_ch = _getch();
            HANDLE hOut;
            CONSOLE_SCREEN_BUFFER_INFO scr;
            hOut = GetStdHandle(STD_OUTPUT_HANDLE);
            GetConsoleScreenBufferInfo(hOut, &scr);
            COORD pos = { 0, 0 };
            short int x1 = scr.dwCursorPosition.X - 1;
            short int x2 = scr.dwCursorPosition.X + 1;
            short int y1 = scr.dwCursorPosition.Y - 1;
            short int y2 = scr.dwCursorPosition.Y + 1;
            if (i_ch == 75)
                pos = { x1, scr.dwCursorPosition.Y };
            if (i_ch == 80)
                pos = { scr.dwCursorPosition.X, y2 };
            if (i_ch == 72)
                pos = { scr.dwCursorPosition.X, y1 };
            if (i_ch == 77)
                pos = { x2, scr.dwCursorPosition.Y };
            short int a = max(buf.size() - 1, 0);
            if (pos.Y > 0)
                pos.Y = min(pos.Y, a);
            else
                pos.Y = 0;
            short int b = 0;
            int p = buf[pos.Y].size();
            if (a > 0)

```

```

        b = max(p - 1, 0);
    if (pos.X > 0)
        pos.X = min(pos.X, b);
    else
        pos.X = 0;
    SetConsoleCursorPosition(hOut, pos);
}
else
{
    if (input_ch == 27) {
        bf.push_back(buf);
        co++;
        bottom_cursor();
        cout << '\n';
        back_cursor();
        return;
    }
    else {
        HANDLE hOut;
        CONSOLE_SCREEN_BUFFER_INFO scr;
        hOut = GetStdHandle(STD_OUTPUT_HANDLE); /* 获取标准输出句柄 */
        GetConsoleScreenBufferInfo(hOut, &scr);
        int x = scr.dwCursorPosition.X;
        int y = scr.dwCursorPosition.Y;

        if (input_ch == 13) {
            string sc = buf[y].substr(x);
            buf[y] = buf[y].substr(0, x);
            if (y < buf.size())buf.insert(buf.begin() + y + 1, 1, sc);
            else buf.push_back(sc);
            x = 0;
            y++;
        }
        else {
            if (y < buf.size()) {
                if (x < buf[y].size())buf[y].insert(x, 1, input_ch);
                else buf[y].push_back(input_ch);
            }
            else {
                string tc;
                tc.push_back(input_ch);
                buf.push_back(tc);
            }
            x++;
        }
    }

    system("cls");
    bottom_cursor();
    cout << "i";
    back_cursor();
    for (int i = 0; i < buf.size(); i++) {
        for (int j = 0; j < buf[i].size(); j++) cout << buf[i][j];
        if (i + 1 < buf.size())
            cout << endl;
    }
    COORD pos = { x, y };
    SetConsoleCursorPosition(hOut, pos);
}
}
}
}

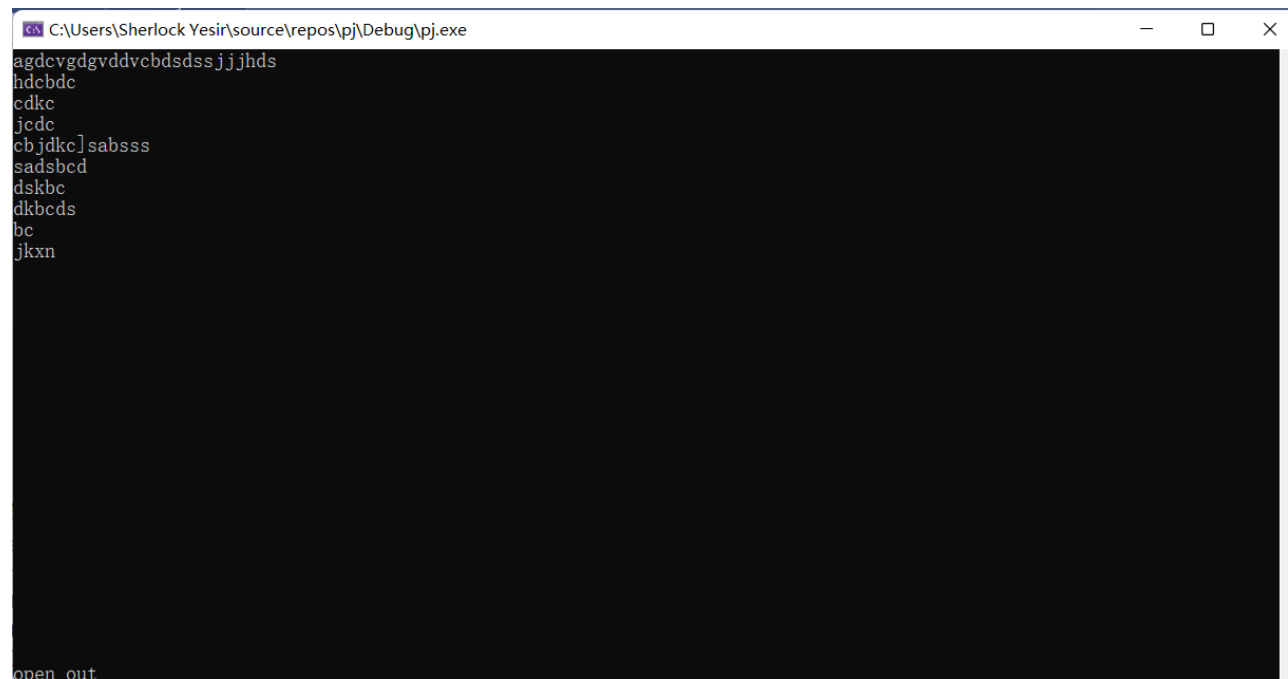
```


4. 调试分析

test1. 读取文本操作

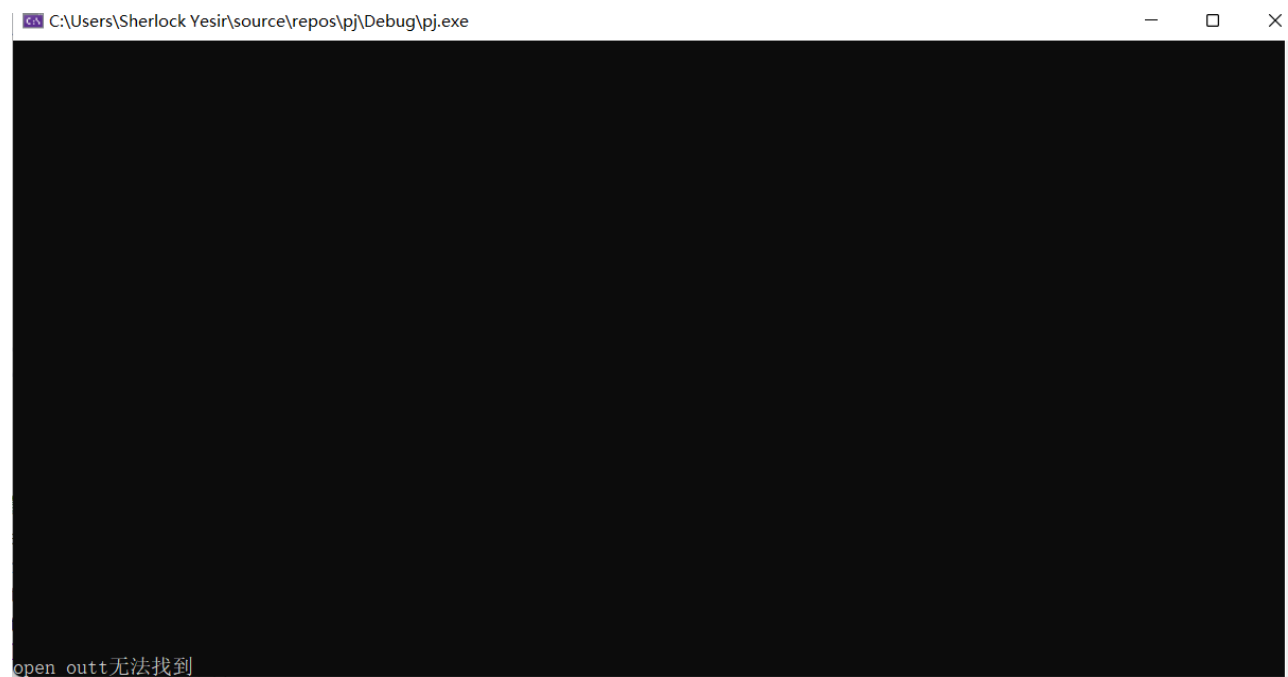
具体操作是normal模式下键入open name（name为文件名，且与cpp源文件在同一目录下，打开成功则输出文件内容，打开失败则输出错误文字

成功样例输出：



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvvgdgvddvcbsdssjjjhds
hdcdbc
cdkc
jcdc
cbjdkc]sabsss
sadsbcd
dskbc
dkbceds
bc
jkxn
open_out
```

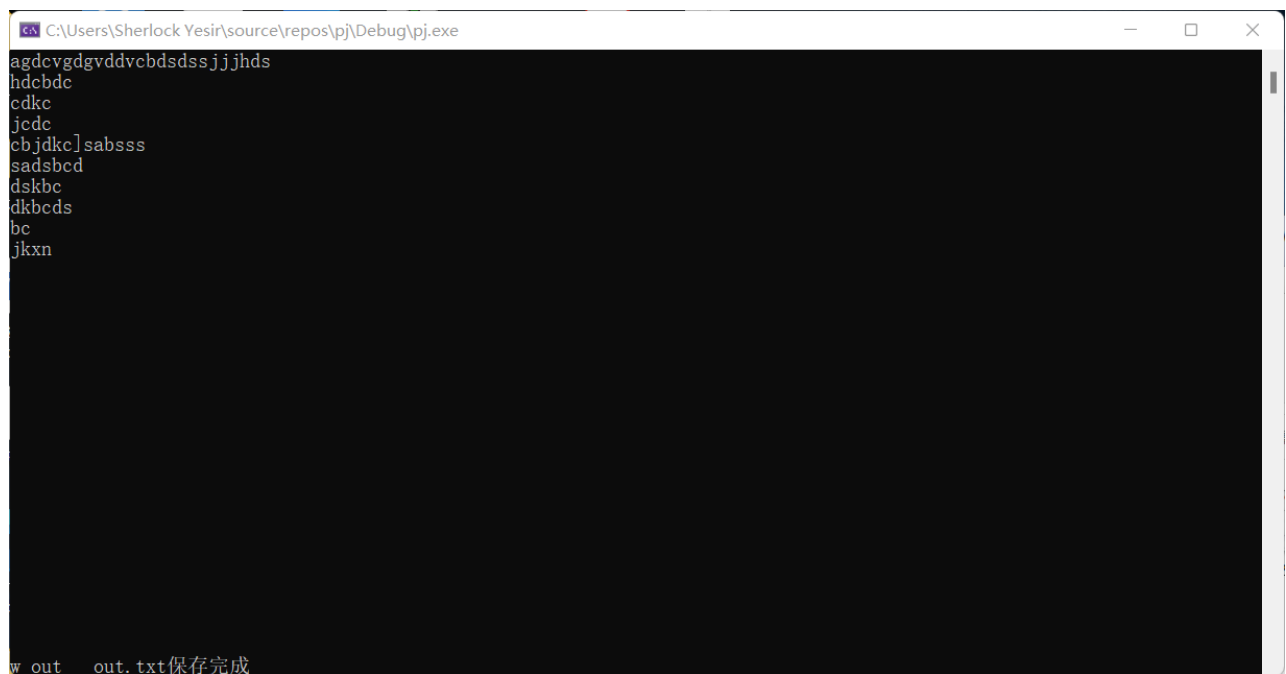
失败样例输出：



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
open_outt无法找到
```

test2. 保存文本操作

具体操作是normal模式下键入w name（name为文件名，且与cpp源文件在同一目录下，保存成功则输出成功，保存失败则输出失败



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvvgdgvddvcbsdssjjjhds
hdcbbc
cdkc
jcdc
cbjdkc]sabsss
sadsbcd
dskbc
dkbods
bc
jkxn

w out out.txt保存完成
```

test3. 退出编辑器

具体操作是normal模式下键入q，退出该程序



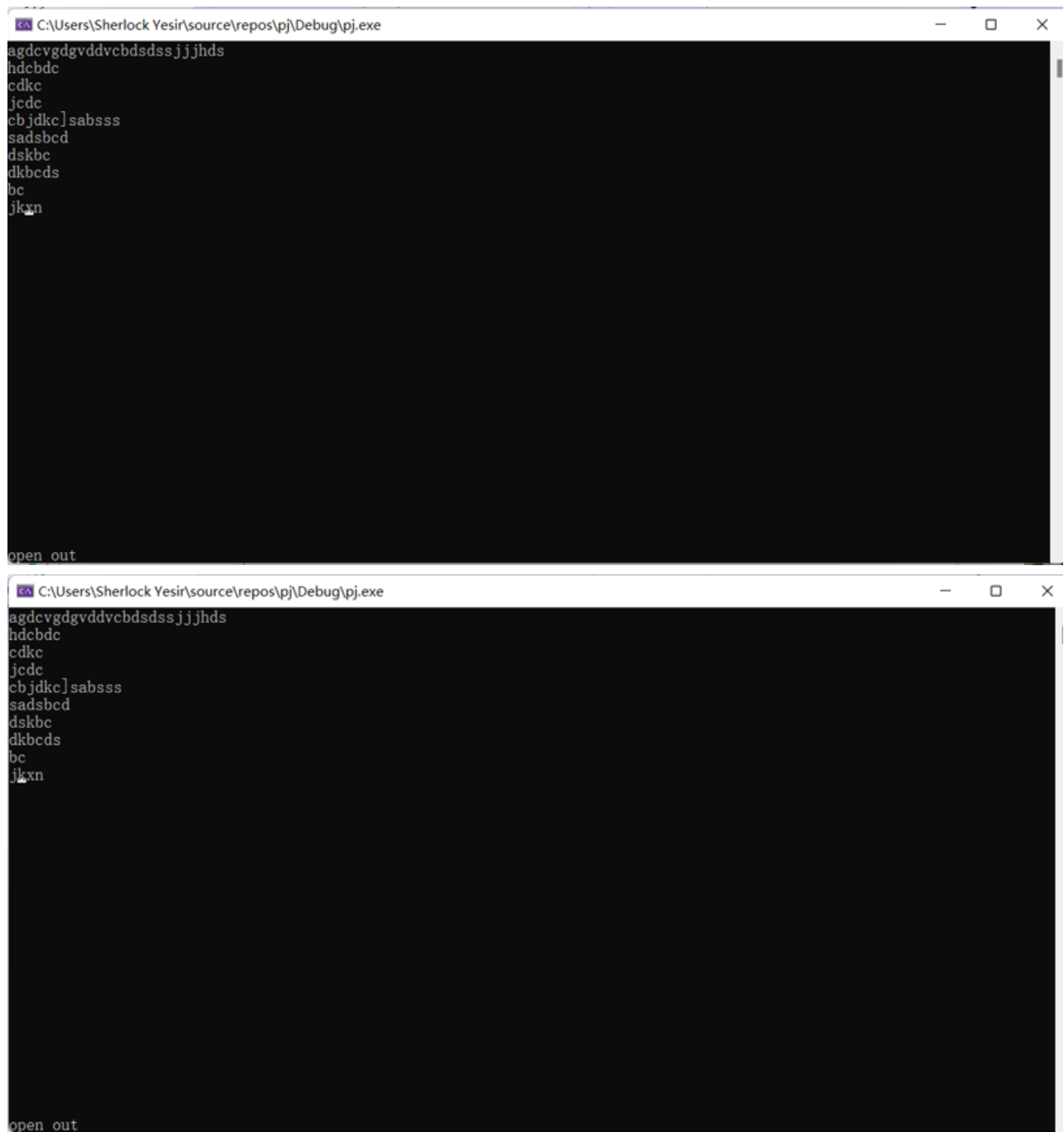
```
Microsoft Visual Studio 调试控制台

q
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe (进程 7916) 已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。...
```

test4. 光标移动操作（normal模式下）

具体操作是在normal模式下键入 h j k l， 分别表示光标向左下上右移动

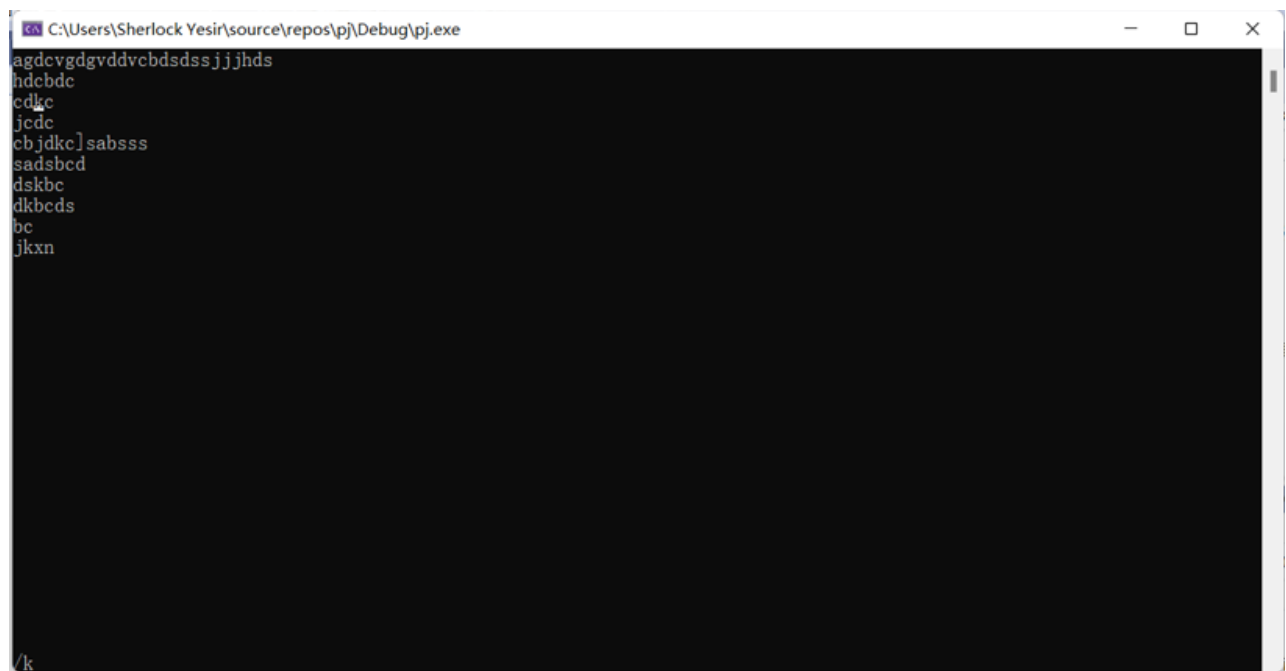
下面演示左移，即键入h



test5. 文本搜索操作

具体操作是在normal模式下，键入/pattern进行全文搜索，其中pattern为搜索的关键字（可以替换为任意字符串），需要从光标处向后搜索直到找到第一处符合的地方，光标跳转至该位置

下面以搜索out.txt中的"k"为例:

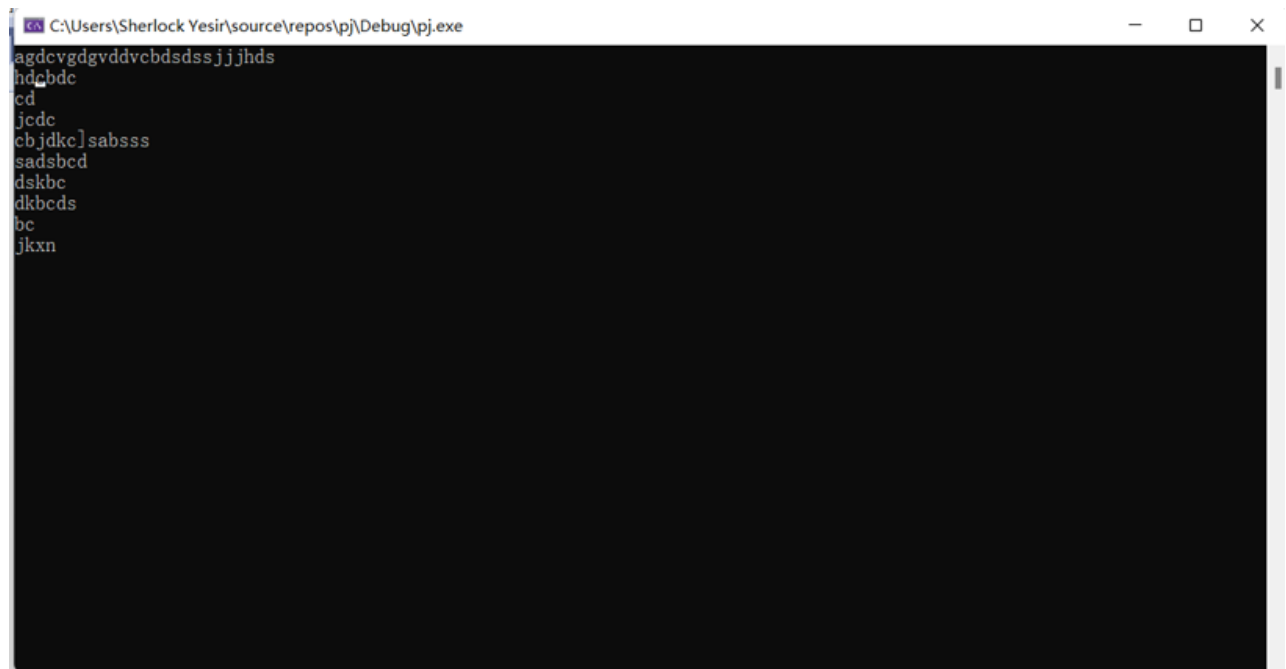


```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvgdgvddvcbsdssjjjhds
hdcbbc
cdkc
jcdc
cbjdkc]sabsss
sadsbcd
dskbc
dkbeds
bc
jkxn
/k
```

test6. 文本删除操作

具体操作是在normal模式下，键入x，删除当前光标处的字符

下面以删除out.txt第二排的c为例：



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvgdgvddvcbsdssjjjhds
hdcbbc
cdkc
jcdc
cbjdkc]sabsss
sadsbcd
dskbc
dkbeds
bc
jkxn
```

```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvgdgvddvcbsdssjjjhds
hdhdc
cd
jcde
cbjdkc]sabsss
sadsbcd
dskbc
dkbcbds
bc
jkxn
```

test7. 撤销操作

具体操作是在normal模式下，键入u，撤销之前的操作

撤销test6所做的删除：

```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvgdgvddvcbsdssjjjhds
hdhdc
cd
jcde
cbjdkc]sabsss
sadsbcd
dskbc
dkbcbds
bc
jkxn
```

```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdevgdgvddvcbdsdssjjjhds
hdabdc
cd
jcdc
cbjdkc]sabsss
sadsbcd
dskbc
dkbcdc
bc
jkxn
```

撤销insert模式键入的内容:

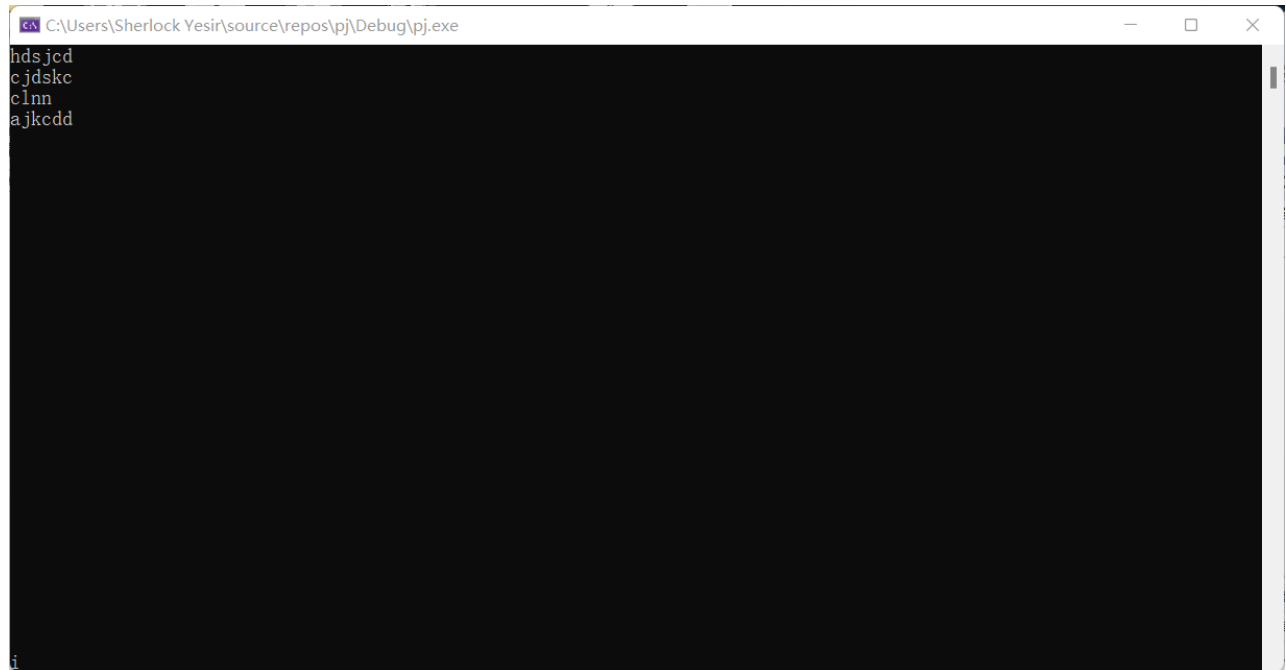
```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
hdsjcd
cjdskc
clnn
ajkcd
```

```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
```

test8. 反撤销操作

具体操作是在normal模式下，键入v，撤销之前的撤销

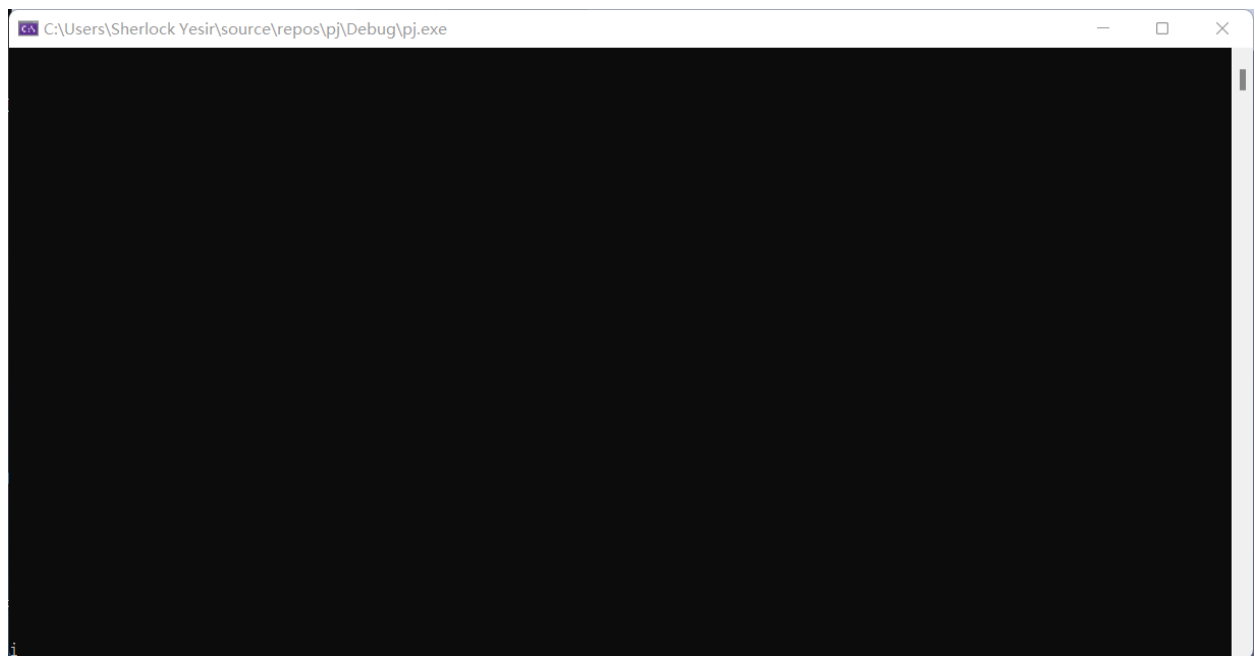
撤销test7（2）所做的撤销：



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
hds jcd
cjdskc
clnn
ajkcdd
i
```

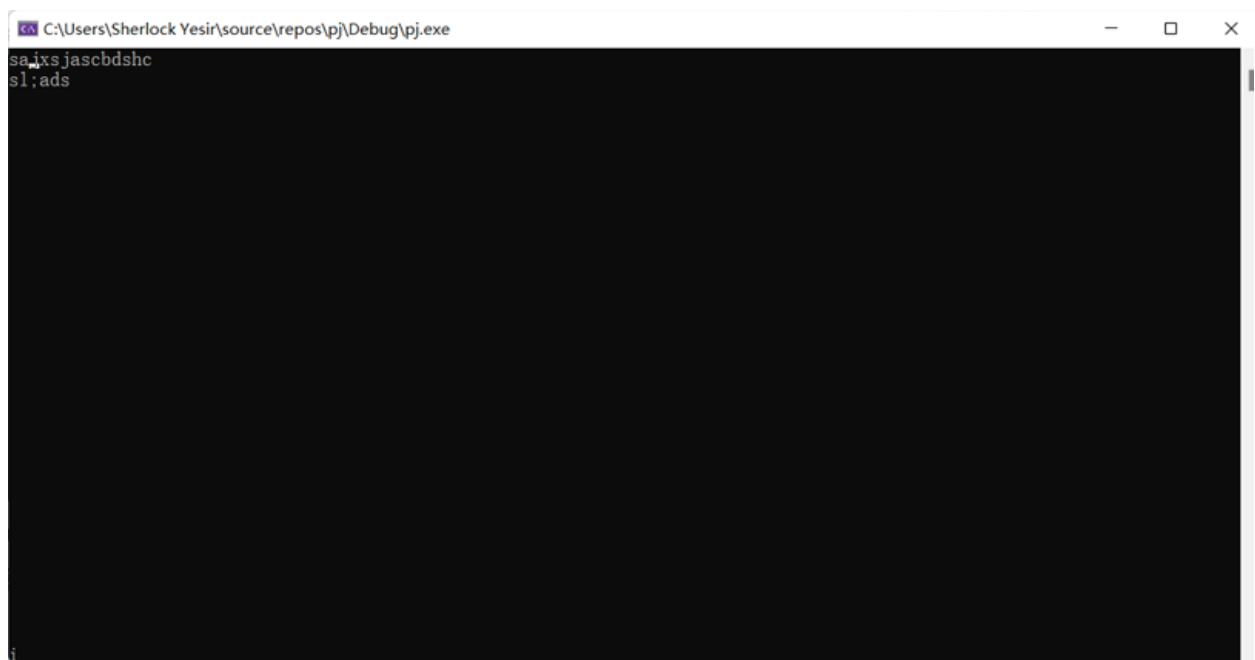
test10. 插入模式

- 进入模式操作：具体操作是在normal模式下键入i，进入插入模式，底端显示i

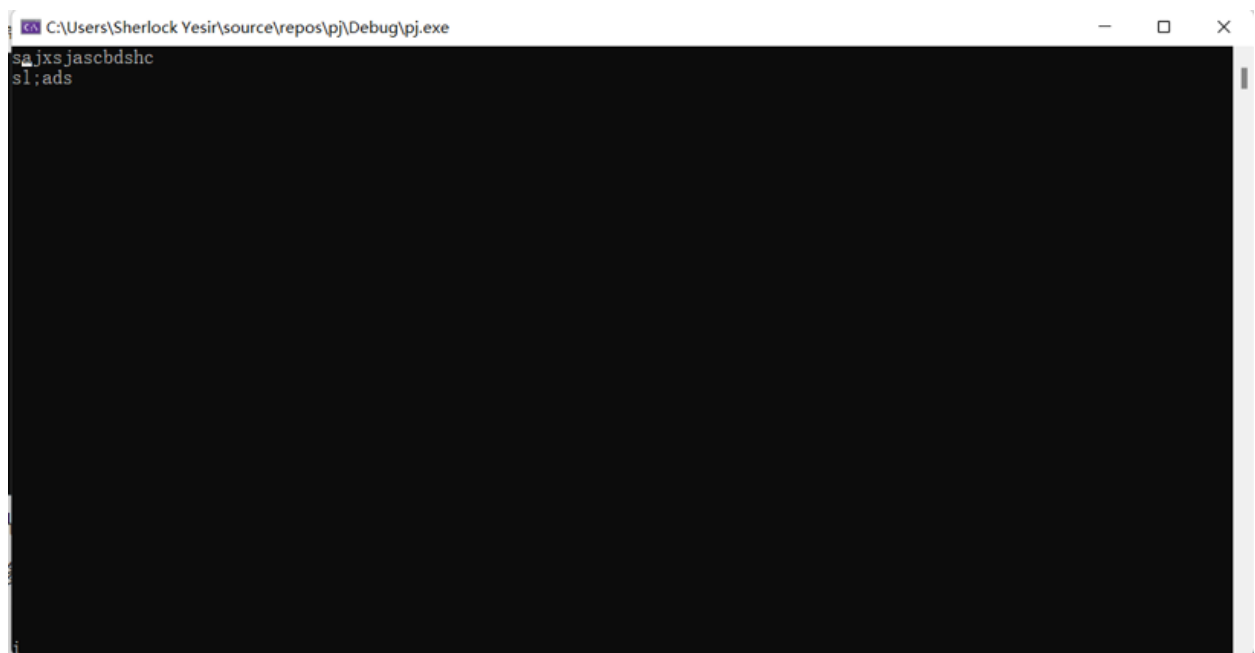


```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
i
```

- 移动光标操作：插入模式下使用方向键进行光标移动
以左移为例：

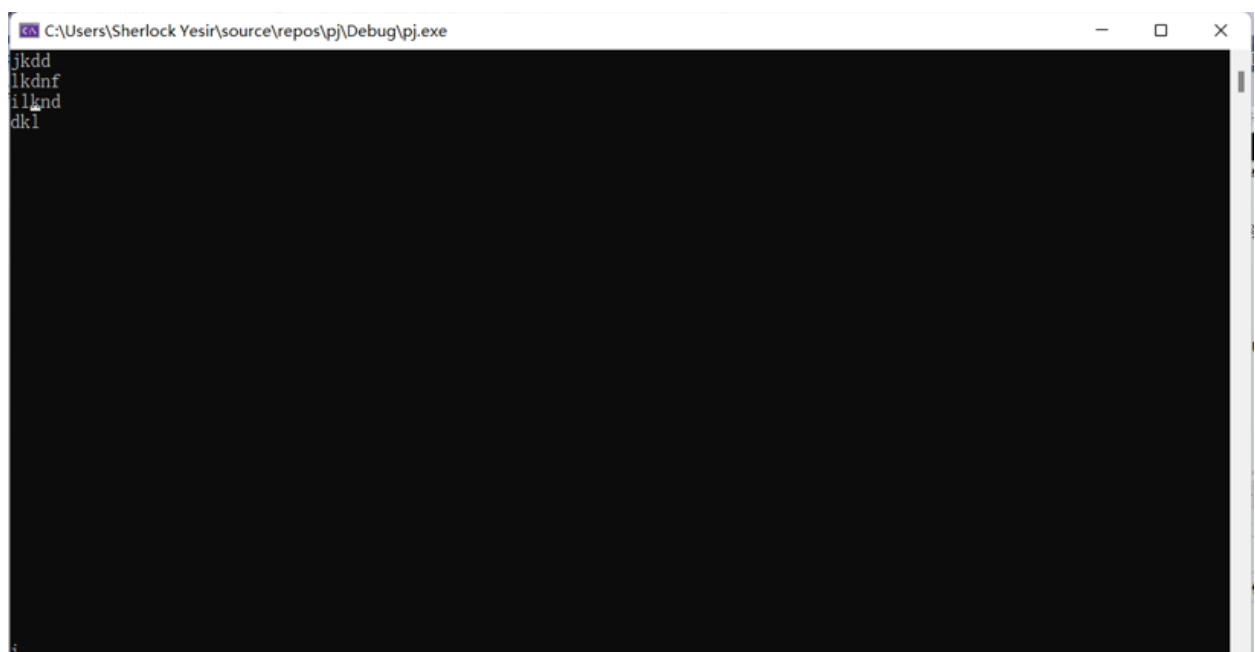


```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
sa:xs jascbdshc
sl:ads
```



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
sa:xs jascbdshc
sl:ads
```

- 文本插入操作：具体操作是在已有文本（空文本也可）基础上在光标前进行插入
插入文本：



```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
jkdd
lkdnf
ilknd
dkl
```



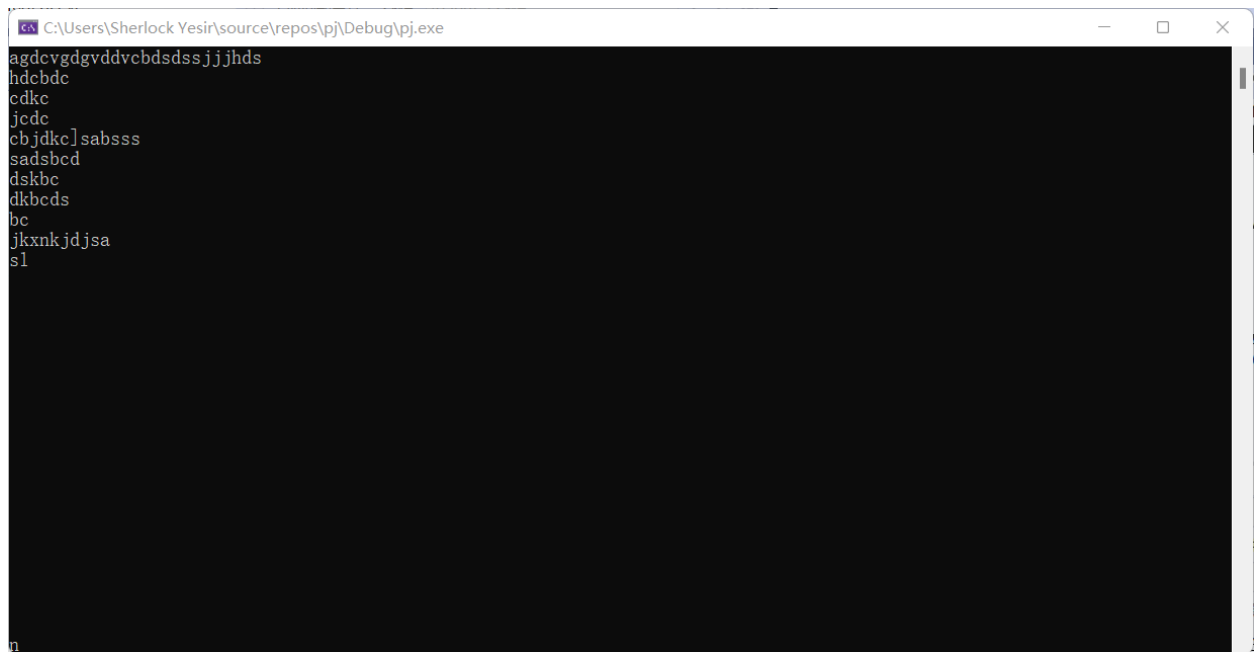
```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
jkdd
lkdnf
iliknd
dkl
```

插入回车：

```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvgdgvddvcbsdssjjjhds
hdcbbc
cdkc
jcde
cbjdkc]sabsss
sadsbcd
dskbc
dkcds
bc
jkxn
```

```
C:\Users\Sherlock Yesir\source\repos\pj\Debug\pj.exe
agdcvgdgvddvcbsdssjjjhds
hdcbbc
cdkc
jcde
cbjdkc]sabsss
sadsbcd
dskbc
dk
cds
bc
jkxn
```

- 退出操作：具体操作是键入esc，退出insert模块，回到normal模式，底端变为n



5. 设计中的不足

1. 因为使用的是控制台屏幕，要实现文本的更新显示，就需要清屏后再输出，造成闪屏，影响使用体验。
2. 实际的文本光标则是在文本之前，而insert模式下光标只能在字符上移动，故会造成最后一个字符后面无法添加文字。
3. 当一行文本过长时，文本会显示到下一行，但光标获取的位置还是以实际位置为准，故长文本的光标移动会出现问题。例如，溢出到下一行的文本，当光标左移是无法回到上一行的。

6. 课程设计总结

- 这次课程设计，使我对《数据结构》这门课程有了更加深入的理解。《数据结构》不只是一门理论学习的课程，还是一门必须要上机实践的课程。因此，为了学好这门课程，甚至是学好整个计算机专业，必须在掌握理论知识的同时，加强上机实践。
- 我的课程设计是做vim青春版，因为c语言文件处理那一章学得比较水，刚开始做这个程序的时候感到完全无从下手。但计算机人是不会轻言放弃的，于是我将各个模块拆开，分别学习和实现。写程序时，运行时有很多问题，时常会出现想的和跑出来的完全不一样的情况，此时就需要对局部产生的变量进行打印测试。虽然调试过程很痛苦，但却对函数的底层操作有了更深入的了解。例如STL库的find函数，vector和string返回的分别是迭代器和下标数字。
- 在本课程设计中，我明白了理论与实际应用相结合的必要性，并提高了自己组织数据及编写大型程序的能力。培养了基本的程序设计技能和良好的程序设计习惯，比如从不写注释的我这次发现注释真的对理解自己之前的编写的代码逻辑有很大帮助，尤其是像这次的较大工程的项目。这次课程设计同样提高了我的综合运用所学知识的能力，并对面向对象程序设计的封装特性更深入的了解。
- 另一方面，上机实践是对学生软件设计的综合能力的训练，包括问题分析，总体结构设计，程序设计基本技能和技巧的训练。此外，还有更重要的一点是：机器是比任何教师更严厉的检查者。通过这段时间的课程设计，我认识到数据结构的上机练习不只是在力扣这些oj网站上去刷算法题，还要能运用这些算法到项目实践中。这次的程序训练培养了我实际分析问题、编程和动手能力，使我掌握了程序设计的基本技能，提高了我适应实际，实践编程的能力。

7. 参考资料

[控制台函数 - Windows Console | Microsoft Docs](#)

[C++判断字符与数字，按下esc键退出 \(csdn博客\)](#)

[SetConsoleCursorPosition光标的位置控制 \(csdn博客\)](#)

[C++中的清屏函数【博客园 \(cnblogs.com\)】](#)

[_getch\(\) 函数，应用于输入密码敲入回车前修改【博客园 \(cnblogs.com\)】](#)

[c++输出文件流ofstream用法详解 \(csdn博客\)](#)

[c++输入文件流ifstream用法详解 \(csdn博客\)](#)

C语言使用getch()读取方向键 (csdn博客)

string插入和删除 (csdn博客)

vector插入和删除 (csdn博客)