

**课程设计**

**课 程： 算法与数据结构**

**题 目： 简单行编辑器**

**专 业： 软件工程**

**班 级： 2201班**

**学 号： 3221306105**

**姓 名： 张宏志**

**2024年 1月7日**

**答辩记录**

|  |
| --- |
| **题目** |
| **答辩问题：（至少3个问题）**  **1、**  **2、**  **3、** |
| **问题回答：**  **1、**  **2、**  **3、** |

# 题目：简单行编辑程序

## 一、要解决的问题

文本编辑程序是利用计算机进行文字加工的基本软件工具，实现对文本文件的插入、删除等修改操作。限制这些操作以行为单位进行的编辑程序称为行编辑程序。被编辑的文本文件可能很大，全部读入编辑程序的数据空间（内存）的做>法既不经济，也不总能实现。一种解决方法是逐段地编辑。任何时刻只把待编辑文件的一段放在内存，称为活区。试按照这种方法实现一个简单的行编辑程序。设文件每行不超过320个字符，很少超过80字符。

## 二、算法基本思想描述

字符串匹配算法：BF、KMP、Tire以及字符串哈希、AC自动机等。

1. **BF算法**：暴力枚举匹配起点和模式串的每一个字符，进行逐个匹配。**时间复杂度：O(NM)**

2. **KMP算法**：该算法的核心是利用匹配失败后的信息，尽量减少模式串与母串的匹配次数以达到快速匹配的目的。具体实现就是通过一个next[]函数实现，函数本身包含了模式串的局部匹配信息。**时间复杂度：O(N+M)**

3. **Trie算法**：Trie树(字典树)是一个可以高效地存储和查找字符串集合的一种多叉树的数据结构。每个节点都拥有若干个字符指针，若在插入或检索字符串时扫描到一个字符c，就沿着当前节点的c字符指针，走向该指针指向的节点>。**时间复杂度：O(M)，空间复杂度：O(NM)。**（其中N为节点个数，M为字符集大小）

## 三、程序设计思路

### 1. 功能描述

程序开始运行时需要键入输入和输出文件。其中输入文件可以为空，表示不从外部文件读取文本。之后程序会在屏幕上显示当前活区所展示的文本内容（最多100行），如果你是第一次使用该编辑器，可以键入help获取帮助菜单。

#### help menu

1. 行插入。格式：**i<行号><回车><文本><回车>** 将<文本>插入活区中第<行号>行之后。

**- Insert Line: Format - i<line\_number><Enter><text><Enter>.**

**Inserts <text> after line <line\_number> in the Active Area.**

2. 行删除。格式：**d<行号1>[ <行号2>]<回车>** 删除活区中第<行号1>行（到第<行号2>行）。

**- Delete Line: Format - d<line\_number1>[ <line\_number2>]<Enter>.**

**Deletes line <line\_number1> (to line <line\_number2>) in the Active Area.**

3. 活区切换。格式：**n<回车>** 将活区写入输出文件，并从输入文件中读入下一段，作为新的活区。

**- Switch Active Area: Format - n<Enter>**

**Write the Active Area to the output file, and read the next segment from the input file as the new Active Area.**

4. 活区显示。格式：**p<回车>** 逐页地（每页20行）显示活区内容。

**- Display Active Area: Format - p<Enter>**

**Display the Active Area content page by page (20 lines per page).**

**5**. 串替换。格式：**S<行号1>[<,行号2>]@<串1>@<串2><回车>** 将第<行号>行中的<串1>替换成<串2>。

**- Replace String:**

**Format - S<line\_number1><,line\_number2>@<string1>@<string2><Enter>**

**Replace <string1> with <string2> in line <line\_number1>( to line <line\_number2>) of the Active Area.**

6. 串匹配。格式：**m <串>[ <行号1>[ <行号2>]]<回车>** 匹配（第<行号1>行（到第<行号2>行））所有<串>，并打印匹配成功的位置。

**- Match String: Format - m <string>[ <line\_number1>[ <line\_number2>]]<Enter>**

**Matched <string>( in line <line\_number1>( to line <line\_number2>)) in the Active Area and print the position where the match is successful.**

7. 读取外部文件。格式：**r <文件名><回车>** 若当前非活区为空，则可以从外部文件中读入新文本到活区中作为待操作内容。

**- Read From File: Format - r <filename><Enter>**

**If Other Area is empty, new text can be read from <filename> to Active Area as the content to be operated.**

8. 写入外部文件。格式：**w <文件名>[ <行号1>[ <行号2>]]** 将活区中（第<行号1>行（到第<行号2>行））的内容写入外部文件中。

**- Write To File:**

**Format - w <filename>[ <line\_number1>[ <line\_number2>]]<Enter>**

**Write the contents of the Active Area (line <line number 1> (to line <line number 2>)) into <filename>.**

9. 退出编辑器。格式：**exit<回车>**

**- Exit the editor: Format - exit**

### 2. 活区与非活区

活区指的是用户当前可修改的文本区，活区最多可以存100行文本，每行文本的最大长度为320个字符。用户可以从输入文件中读取文本到活区中，若输入文件文本内容多于80行，则只读取前80行文本，留下20行给用户操作的空间，而输入文件中的其余文本会暂时存入非活区中。

用户可对文本进行自定义操作，操作结束后按下`n`写到输出文件中。此时若非活区中还存有文本，则将非活区中的文本继续按照上述方式继续读入活区中。

考虑到文本文件行长通常为正态分布，且峰值在60 ~ 70之间，用320×100大小的字符数组实现存储将造成大量浪费。因此本编辑器采用以标准行块为单位为各行分配存储，每个标准行块含80个字符，并且用动态链表将每一个行块连接起来，从而不会导致内存的浪费。

### 3. 活区的表示

每一行采用行块链表存储，每一块最多存储80个字符，若超出限制，则新增一个行块，并且将其于前面的行块连接起来，构成动态链表的形式。如下图所示：

图示

描述已自动生成

### 4. 字符串匹配算法的设计

在字符串匹配方面：用户可以自行选择采用哪种匹配算法对字符串进行匹配，目前仅提供4种算法：KMP、Trie、字符串哈希、BF。默认采用KMP算法进行字符串匹配。

1. **KMP算法**

该算法的核心为next数组，next数组的含义：next[i]表示以i为结尾的后缀最多有多少个字符与前缀相等。在匹配过程中，若未匹配成功，则跳到next所指向的地方继续匹配，而不是每次都跳到起点重新匹配。更详细的算法思路可以参考我之前写过的一篇博客：

<https://blog.csdn.net/m0_75186429/article/details/134505128?spm=1001.2014.3001.5501>

1. **Trie树**

Trie树又称字典树或前缀树，是一种用于实现字符串快速检索的多叉树结构。匹配效率和KMP算法不相上下。

1. **字符串哈希**

字符串哈希思路：1. 先预处理所有前缀的哈希，把字符串看成是一个p进制的数，每一位上的字符表示它p进制的数（当p取131或13331，Q取2^64时，99.99%情况下可以假定不会发生冲突）。由于最后要对Q取模，因此可以直接用`unsigned long long`来存储哈希值，溢出的时候会自动对2^64取模。

s = "ABCD"

-> 'A' 'B' 'C' 'D'

   (1   2   3   4)p // p进制为：1234

 = 1 \* p^3 + 2 \* p^2 + 3 \* p^1 + 4 \* p^0 // 十进制

// 这样就可以把一个字符串转换成一个数字

// 这个数字可能会非常大，因此要在最后模上一个非常小的数Q

// 这样就可以把很大的数映射到0 ~ Q - 1

1. **BF算法**

BF(Brute Force)算法。顾名思义，是一种最朴素也是最易理解的字符串匹配算法。在匹配过程中，若未匹配成功，则直接跳到起点进行下一次匹配，时间复杂度较高，效率不如KMP算法。

int n = strlen(str), m = strlen(match\_str);

// 暴力枚举每次匹配的起点

    for (int i = 0; i <= n - m; i ++ )

    {

        int j;

        for (j = 0; j < m; j ++ )  // 枚举匹配长度

            if (str[i + j] != match\_str[j])

                break;

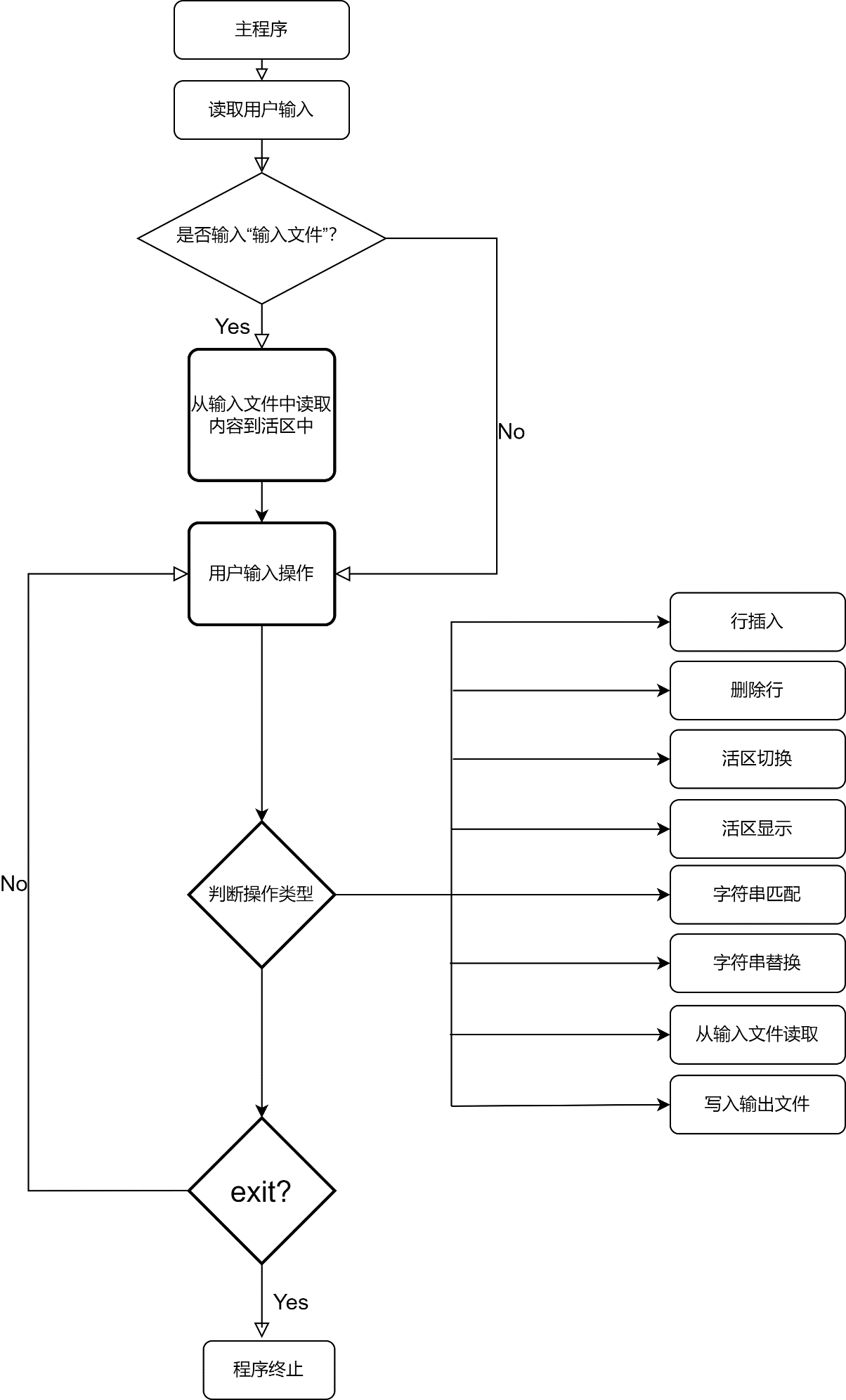
        // 若匹配成功，保存位置

        if (j == m) pos[cnt ++ ] = i;

    }

## 四、程序模块结构

### 1. 流程图

****

### 2. 结构体设计

// 存储每一行的文本，链式存储

typedef struct LineBlock {

    char data[MAXBLOCKSIZE];  // 行中每一块的字符内容

    LineBlock\* next;  // 指向下一个行块

    // 默认构造

    LineBlock(const char\* text, LineBlock\* ne = nullptr)

    {

        int length = strlen(text);

        memcpy(data, text, min(length, MAXLINESIZE - 1));  // 把text的内容拷贝给data

        data[length] = '\0';

        next = ne;

    }

} LineBlock;

// 存储活区中的每一行，链式存储

typedef struct Line {

    int line\_no;  // 行号

    LineBlock\* content;  // 指向行内容

    Line\* next;  // 指向下一行

    Line(int number, LineBlock\* line\_block, Line\* ne = nullptr) : line\_no(number), content(line\_block), next(ne) {}

} Line, \*ActiveArea;

// 存储活区中匹配到的字符串所处的位置(row,col)

typedef struct Position {

    int row, col;

} Pos;

// Trie节点

struct TrieNode {

    bool isEnd;  // 标记是否是单词的结尾

    std::unordered\_map<char, TrieNode\*> son;  // 子节点集合

    TrieNode() : isEnd(false) {}

};

// Trie树

class Trie {

public:

    // 默认构造

    Trie() : root(new TrieNode()) {}

    // 插入字符串

    void insert(const char\* word)

    {

        TrieNode\* node = root;

        while (\*word)

        {

            if (node->son.find(\*word) == node->son.end())

                node->son[\*word] = new TrieNode();

            node = node->son[\*word];

            word ++ ;

        }

        node->isEnd = true;

    }

    // 搜索字符串

    bool search(const char\* src, const char\* match\_str, int pos[], int& cnt)

    {

        cnt = 0;

        insert(match\_str);

        int len = strlen(src);

        for (int i = 0; i < len; i ++ )

        {

            TrieNode\* node = this->root;

            for (int j = i; j < len; j ++ )

            {

                if (node->son.find(src[j]) == node->son.end()) break;

                node = node->son[src[j]];

                // 匹配成功，保存位置

                if (node->isEnd) pos[cnt ++ ] = i;

            }

        }

        return cnt > 0;

    }

private:

    TrieNode\* root;

};

### 3. 主要函数设计

/\* 显示函数 \*/

void showMainMenu();

void showHelpMenu();

void showActiveArea(const ActiveArea&, int, int, int);

void printLine(const LineBlock\*);

void printPositions(Pos[], int);

/\* 辅助函数 \*/

void readFile(char\*, char\*);

int getValidInput();

void getNumber(const char\*, int&, int, int);

void getString(const char\*, char\*, int, int);

bool CHECK\_AREA(const ActiveArea&);

bool CHECK\_BLOCK(const LineBlock\*);

bool CHECK\_BOUND(int, int, int, int low = 0);

/\* 对Line操作的一些函数 \*/

bool initArea(ActiveArea&);

int countLine(const ActiveArea&);

bool emptyArea(const ActiveArea&);

bool insertLine(ActiveArea&, char\*, int, ActiveArea&, char\* output\_file = nullptr);

void changeActiveArea(ActiveArea&, ActiveArea&, char\*);

bool deleteLine(ActiveArea&, ActiveArea&, int, int);

bool matchString(ActiveArea&, char\*, int, int, Pos[], int&, int);

bool replaceString(ActiveArea&, int, char\*, char\*, int&, int);

/\* 对LineBlock操作的一些函数 \*/

bool insertLineBlock(LineBlock\*&, char\*);

bool emptyLineBlock(const LineBlock\*);

bool clearLineBlock(LineBlock\*&);

void blocks\_to\_str(LineBlock\*, char\*);

void str\_to\_blocks(char\*, LineBlock\*&);

/\* 对文件操作的一些函数 \*/

bool readFromInputFile(char\*, ActiveArea&, ActiveArea&, char\* ouput\_file = nullptr);

bool writeToOutputFile(char\*, ActiveArea&, ActiveArea&, int, int);

bool readFromOtherArea(ActiveArea&, ActiveArea&, int);

/\* 对字符串操作的一些函数 \*/

bool insertString(char\*, int, const char\*);

bool eraseString(char\*, int, int);

bool kmp(char\*, char\*, int[], int&);

bool trie(char\*, char\*, int[], int&);

unsigned long long get(int, int, unsigned long long[], unsigned long long[]);

bool string\_hash(char\*, char\*, int[], int&);

bool bf(char\*, char\*, int[], int&);

## 五、程序源码

1. #include <iostream>
2. #include <algorithm>
3. #include <cstring>
4. #include <fstream>
5. #include <iomanip>
6. #include <unordered\_map>

9. /\* 宏定义，用于打印各种不同等级的信息，便于调试 \*/
10. #define INFO(message) std::cout << Hazel::GREEN << message << Hazel::RESET << std::endl;
11. #define WARNINGS(message) std::cout << Hazel::YELLOW << message << Hazel::RESET << std::endl;
12. #define TRACE(message) std::cout << Hazel::MAGENTA << message << Hazel::RESET << std::endl;
13. #define ERROR(message)\
14. std::cerr << Hazel::RED << "Error: " << Hazel::YELLOW << \_\_FILE\_\_ << Hazel::RESET\
15. << " : in function " << Hazel::CYAN <<  \_\_func\_\_ << Hazel::RESET\
16. << " at line " << Hazel::MAGENTA << \_\_LINE\_\_ << Hazel::RESET << std::endl\
17. << "        Compiled on " << \_\_DATE\_\_\
18. << " at " << \_\_TIME\_\_ << std::endl\
19. << "        " << Hazel::RED << message << Hazel::RESET << std::endl;

22. /\* 定义一些要用到的常量 \*/
23. **namespace** Hazel
24. {
25. // 颜色变量
26. **const** **char**\* RESET = "\033[0m";
27. **const** **char**\* BLACK = "\033[30m";
28. **const** **char**\* RED = "\033[31m";
29. **const** **char**\* GREEN = "\033[32m";
30. **const** **char**\* YELLOW = "\033[33m";
31. **const** **char**\* BLUE = "\033[34m";
32. **const** **char**\* MAGENTA = "\033[35m";
33. **const** **char**\* CYAN = "\033[36m";
34. // 一些大小常量
35. **const** **int** MAXACTIVELEN = 100;  // 活区的最大长度
36. **const** **int** MAXLINELEN = 321;  // 每一行的最大长度
37. **const** **int** MAXBLOCKLEN = 81;  // 每一个行块的最大长度
38. **const** **int** MAXFILENAMELEN = 100;  // 文件名的最大长度
39. // 字符串匹配算法的选择
40. **const** **int** KMP = 1;
41. **const** **int** TRIE = 2;
42. **const** **int** STRINGHASH = 3;
43. **const** **int** BF = 4;
44. };

47. /\* 结构体的定义 \*/
49. // 存储每一行的文本，链式存储
50. **typedef** **struct** LineBlock {
51. **char** data[Hazel::MAXBLOCKLEN];  // 行中每一块的字符内容
52. LineBlock\* next;  // 指向下一个行块
53. // 默认构造
54. LineBlock(**const** **char**\* text, LineBlock\* ne = nullptr)
55. {
56. **int** length = strlen(text);
57. memcpy(data, text, std::min(length, Hazel::MAXLINELEN - 1));  // 把text的内容拷贝给data
58. data[length] = '\0';
59. next = ne;
60. }
61. } LineBlock;
62. // 存储活区中的每一行，链式存储
63. **typedef** **struct** Line {
64. **int** line\_no;  // 行号
65. LineBlock\* content;  // 指向行内容
66. Line\* next;  // 指向下一行
67. Line(**int** number, LineBlock\* line\_block, Line\* ne = nullptr) : line\_no(number), content(line\_block), next(ne) {}
68. } Line, \*ActiveArea;
69. // 存储活区中匹配到的字符串所处的位置(row,col)
70. **typedef** **struct** Position {
71. **int** row, col;
72. } Pos;
73. // Trie节点
74. **struct** TrieNode {
75. **bool** isEnd;  // 标记是否是单词的结尾
76. std::unordered\_map<**char**, TrieNode\*> son;  // 子节点集合
77. TrieNode() : isEnd(**false**) {}
78. };
79. // Trie树
80. **class** Trie {
81. **public**:
82. // 默认构造
83. Trie() : root(**new** TrieNode()) {}
84. // 插入字符串
85. **void** insert(**const** **char**\* word)
86. {
87. TrieNode\* node = root;
88. **while** (\*word)
89. {
90. **if** (node->son.find(\*word) == node->son.end())
91. node->son[\*word] = **new** TrieNode();
92. node = node->son[\*word];
93. word ++ ;
94. }
95. node->isEnd = **true**;
96. }
97. // 搜索字符串
98. **bool** search(**const** **char**\* src, **const** **char**\* match\_str, **int** pos[], **int**& cnt)
99. {
100. cnt = 0;
101. insert(match\_str);
102. **int** len = strlen(src);
103. **for** (**int** i = 0; i < len; i ++ )
104. {
105. TrieNode\* node = **this**->root;
106. **for** (**int** j = i; j < len; j ++ )
107. {
108. **if** (node->son.find(src[j]) == node->son.end()) **break**;
109. node = node->son[src[j]];
110. // 匹配成功，保存位置
111. **if** (node->isEnd) pos[cnt ++ ] = i;
112. }
113. }
114. **return** cnt > 0;
115. }
117. **private**:
118. TrieNode\* root;
119. };

122. /\* 函数的声明 \*/
124. /\* 显示函数 \*/
125. **void** showMainMenu();
126. **void** showHelpMenu();
127. **void** showActiveArea(**const** ActiveArea&, **int**, **int**, **int**);
128. **void** printLine(**const** LineBlock\*);
129. **void** printPositions(Pos[], **int**);
130. /\* 辅助函数 \*/
131. **void** readFile(**char**\*, **char**\*);
132. **int** getValidInput();
133. **void** getNumber(**const** **char**\*, **int**&, **int**, **int**);
134. **void** getString(**const** **char**\*, **char**\*, **int**, **int**);
135. **bool** CHECK\_AREA(**const** ActiveArea&);
136. **bool** CHECK\_BLOCK(**const** LineBlock\*);
137. **bool** CHECK\_BOUND(**int**, **int**, **int**, **int** low = 0);
138. /\* 对Line操作的一些函数 \*/
139. **bool** initArea(ActiveArea&);
140. **int** countLine(**const** ActiveArea&);
141. **bool** emptyArea(**const** ActiveArea&);
142. **bool** insertLine(ActiveArea&, **char**\*, **int**, ActiveArea&, **char**\* output\_file = nullptr);
143. **void** changeActiveArea(ActiveArea&, ActiveArea&, **char**\*);
144. **bool** deleteLine(ActiveArea&, ActiveArea&, **int**, **int**);
145. **bool** matchString(ActiveArea&, **char**\*, **int**, **int**, Pos[], **int**&, **int**);
146. **bool** replaceString(ActiveArea&, **int**, **char**\*, **char**\*, **int**&, **int** type = Hazel::KMP);
147. **bool** replaceString(ActiveArea&, **char**\*, **char**\*, **int**, **int**, **int**&, **int** type = Hazel::KMP);
148. /\* 对LineBlock操作的一些函数 \*/
149. **bool** insertLineBlock(LineBlock\*&, **char**\*);
150. **bool** emptyLineBlock(**const** LineBlock\*);
151. **bool** clearLineBlock(LineBlock\*&);
152. **void** blocks\_to\_str(LineBlock\*, **char**\*);
153. **void** str\_to\_blocks(**char**\*, LineBlock\*&);
154. /\* 对文件操作的一些函数 \*/
155. **bool** readFromInputFile(**char**\*, ActiveArea&, ActiveArea&, **char**\* ouput\_file = nullptr);
156. **bool** writeToOutputFile(**char**\*, ActiveArea&, ActiveArea&, **int**, **int**);
157. **bool** readFromOtherArea(ActiveArea&, ActiveArea&, **int**);
158. /\* 对字符串操作的一些函数 \*/
159. **bool** insertString(**char**\*, **int**, **const** **char**\*);
160. **bool** eraseString(**char**\*, **int**, **int**);
161. **bool** kmp(**char**\*, **char**\*, **int**[], **int**&);
162. **bool** trie(**char**\*, **char**\*, **int**[], **int**&);
163. unsigned **long** **long** get(**int**, **int**, unsigned **long** **long**[], unsigned **long** **long**[]);
164. **bool** string\_hash(**char**\*, **char**\*, **int**[], **int**&);
165. **bool** bf(**char**\*, **char**\*, **int**[], **int**&);

168. /\* 函数的具体实现 \*/

171. /\* 1. 显示函数的实现 \*/
173. // 显示主操作界面
174. **void** showMainMenu()
175. {
176. INFO("\*\*\*\*\*\*\*\*\*\*\*\*  Welcome to Hazel Editor  \*\*\*\*\*\*\*\*\*\*\*\*")
177. std::cout << Hazel::CYAN << "Please input your operation: (If you are new, you can type help to get help menu)" << Hazel::RESET << std::endl;
178. }
179. // 显示帮助菜单
180. **void** showHelpMenu()
181. {
182. INFO("help menu")
183. std::cout << Hazel::CYAN << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << Hazel::RESET << std::endl;
184. std::cout << Hazel::CYAN << "\*" << Hazel::RESET << " 1. Insert Line: Format - i<line\_number><Enter><text><Enter>.\n"
185. << Hazel::CYAN << "\*" << Hazel::RESET << " 2. Delete Line: Format - d<line\_number1>[ <line\_number2>].\n"
186. << Hazel::CYAN << "\*" << Hazel::RESET << " 3. Switch Active Area: Format - n\n"
187. << Hazel::CYAN << "\*" << Hazel::RESET << " 4. Display Active Area: Format - p\n"
188. << Hazel::CYAN << "\*" << Hazel::RESET << " 5. Replace String: Format - S<line\_number1>( <line\_number2>)@<string1>@<string2>.\n"
189. << Hazel::CYAN << "\*" << Hazel::RESET << " 6.  Match String: Format - m <string>[ <line\_number1>[ <line\_number2>]].\n"
190. << Hazel::CYAN << "\*" << Hazel::RESET << " 7. Read From File: Format - r <file\_name>.\n"
191. << Hazel::CYAN << "\*" << Hazel::RESET << " 8. Write To File: Format - w <file\_name>[ <line\_number1>[ <line\_number2>]].\n"
192. << Hazel::CYAN << "\*" << Hazel::RESET << " 9. Exit Editor: Format - exit" << std::endl;
193. std::cout << Hazel::CYAN << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << Hazel::RESET << std::endl;
194. }
195. // 显示活区，从第begin行开始显示，一次显示cnt行
196. **void** showActiveArea(**const** ActiveArea& activeArea, **int** begin, **int** cnt, **int** page\_num)
197. {
198. INFO("---------------  Current Active Area (Page " << page\_num << ")  ---------------")
199. **if** (emptyArea(activeArea))
200. {
201. WARNINGS("    Current Active Area is None!\n")
202. **return**;
203. }
204. // 输出当前活区
205. Line\* cur = activeArea;
206. **while** (cur && begin -- ) cur = cur->next;  // 找出要显示的活区起始行第begin行
207. **if** (!cur)  // 若cur为空，说明活区遍历到尾了
208. {
209. WARNINGS("    Current Active Area is None!\n")
210. **return**;
211. }
212. // 此时cur指向第begin行
213. **while** (cur && cnt -- )
214. {
215. std::cout << Hazel::BLUE << std::setw(4) << cur->line\_no << Hazel::RESET << ' ';  // 显示行号（蓝色显示）
216. printLine(cur->content);
217. cur = cur->next;
218. }
219. std::cout << Hazel::YELLOW << std::setw(60) << "total line: " << activeArea->line\_no << Hazel::RESET << std::endl;  // 显示总行数（黄色显示）
220. std::cout << std::endl;
221. }
222. // 打印一行的文本内容
223. **void** printLine(**const** LineBlock\* line\_content)
224. {
225. **while** (line\_content)
226. {
227. std::cout << line\_content->data;
228. line\_content = line\_content->next;
229. }
230. std::cout << std::endl;
231. }
232. // 打印位置(row,col)函数
233. **void** printPositions(Pos positions[], **int** n)
234. {
235. **for** (**int** i = 0; i < n; i ++ )
236. std::cout << i + 1 << "-th match position: " << Hazel::CYAN <<positions[i].row << "," << positions[i].col << Hazel::RESET << std::endl;
237. }

240. /\* 2. 辅助函数的实现 \*/
242. // 读取输入输出文件名
243. **void** readFile(**char**\* input\_file, **char**\* output\_file)
244. {
245. std::cout << "Please enter Input File: ";
246. std::cin.getline(input\_file, Hazel::MAXFILENAMELEN);
247. std::cout << "Please enter Output File: ";
248. std::cin.getline(output\_file, Hazel::MAXFILENAMELEN);
249. }
250. // 获取有效输入
251. **int** getValidInput()
252. {
253. **int** input;
254. **while** (**true**)
255. {
256. std::cin >> input;
257. **if** (std::cin.fail())  // 检查输入有效性
258. {
259. WARNINGS("Invalid Input! Please input a number")
260. std::cin.clear();  // 清除错误状态
261. std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  // 丢弃无效的输入
262. }
263. **else** **break**;
264. }
265. **return** input;
266. }
267. // 读取串src中从下标为[begin,end]的区间内的数字给res （若end超出范围表示读到字符串结束）
268. **void** getNumber(**const** **char**\* src, **int**& res, **int** begin, **int** end)
269. {
270. **if** (!CHECK\_BOUND(begin, end, strlen(src), 0)) **return**;
271. res = 0;
272. **for** (**int** i = begin; src[i] && i <= end; i ++ ) res = res \* 10 + (src[i] - '0');
273. }
274. // 读取src的字串src[begin,end]给dist
275. **void** getString(**const** **char**\* src, **char**\* dist, **int** begin, **int** end)
276. {
277. **if** (!CHECK\_BOUND(begin, end, strlen(src), 0)) **return**;
278. **int** cnt = 0;
279. **for** (**int** i = begin; i <= end; i ++ ) dist[cnt ++ ] = src[i];
280. dist[cnt] = '\0';
281. }
282. // 检查ActiveArea是否合法
283. **bool** CHECK\_AREA(**const** ActiveArea& area)
284. {
285. **if** (!area)
286. {
287. ERROR("The area is not exist")
288. **return** **false**;
289. }
290. **return** **true**;
291. }
292. // 检查行块是否合法
293. **bool** CHECK\_BLOCK(**const** LineBlock\* block)
294. {
295. **if** (!block)
296. {
297. ERROR("The block is not exist")
298. **return** **false**;
299. }
300. **return** **true**;
301. }
302. // 检查区间[begin,end]是否合法，high为区间的上界，low为下界（默认为0）
303. **bool** CHECK\_BOUND(**int** begin, **int** end, **int** high, **int** low)
304. {  // 也可以改成抛出异常
305. **if** (begin < low || begin > high)
306. {
307. ERROR("The parameter 'begin' is invalid")
308. // throw std::runtime\_error("index out of bounds");
309. **return** **false**;
310. }
311. **if** (end < low || end > high)
312. {
313. ERROR("The parameter 'end' is invalid")
314. // throw std::runtime\_error("index out of bounds");
315. **return** **false**;
316. }
317. **if** (begin > end)
318. {
319. ERROR("Invalid interval")
320. // throw std::runtime\_error("invalid interval");
321. **return** **false**;
322. }
323. **return** **true**;
324. }
325. // 清屏函数
326. **void** clearScreen()
327. {
328. #ifdef \_WIN32
329. system("cls");
330. #elif \_\_linux\_\_
331. system("clear");
332. #endif
333. }

336. /\* 3. 对Area操作的相关函数的实现 \*/
338. // 初始化ActiveArea
339. **bool** initArea(ActiveArea& area)
340. {
341. **try**
342. {
343. area = **new** Line(0, nullptr, nullptr);
344. }
345. **catch** (std::bad\_alloc& e)
346. {
347. ERROR(e.what())
348. **return** **false**;
349. }
350. **return** **true**;
351. }
352. // 返回活区的长度（总行数）
353. **int** countLine(**const** ActiveArea& area)
354. {
355. **if** (!CHECK\_AREA(area)) **return** -1;
356. **return** area->line\_no;
357. }
358. // 判断ActiveArea是否为空
359. **bool** emptyArea(**const** ActiveArea& area)
360. {
361. **if** (!CHECK\_AREA(area)) **return** **false**;
362. **return** area->line\_no == 0/\* && area->next == nullptr\*/;
363. }
364. // 将行插入到第i行后面，当output\_file为空时说明正在往非活区中插入新行
365. **bool** insertLine(ActiveArea& activeArea, **char**\* text, **int** i, ActiveArea& otherArea, **char**\* output\_file)
366. {
367. **if** (i < 0 || i > countLine(activeArea))  // 插入位置不合法
368. {
369. ERROR("The position of the inserted row is invalid")
370. **return** **false**;
371. }
372. // 1. 处理输入文本
373. LineBlock\* line\_content = nullptr;  // 新行的头指针
374. str\_to\_blocks(text, line\_content);  // 将字符串转换成块的形式存储在line\_content中
375. // 2. 将文本插入活区中
376. Line\* new\_line = **new** Line(i + 1, line\_content);  // 新行，作为第i+1行存在与活区中
377. Line\* cur = activeArea;  // cur指向插入节点的前趋节点，初始化为头节点
378. **while** (cur->next && i -- ) cur = cur->next;  // 寻找插入位置
379. **if** (i > 0)
380. {
381. ERROR("The position of the inserted row is invalid")
382. **return** **false**;
383. }
384. new\_line->next = cur->next;
385. cur->next = new\_line;
386. // 修改new\_line后面所有行的行号
387. cur = new\_line->next;
388. **while** (cur)
389. {
390. cur->line\_no ++ ;
391. cur = cur->next;
392. }
393. activeArea->line\_no ++ ;  // 更新活区（或非活区）长度
394. // 只有写入活区时才需要检查：若插入新行后活区长度超出限制，则将第一行输出（只有output\_file有值时，才是插入到活区中）
395. **if** (activeArea->line\_no > Hazel::MAXACTIVELEN && output\_file)
396. writeToOutputFile(output\_file, activeArea, otherArea, 1, 1);  // 将活区中的第1行输出
397. **return** **true**;
398. }
399. // 活区切换，将活区写入输出文件，并从非活区中读取下一段，作为新的活区。
400. **void** changeActiveArea(ActiveArea& activeArea, ActiveArea& otherArea, **char**\* output\_file)
401. {
402. **if** (!CHECK\_AREA(activeArea)) **return**;
403. **int** len = countLine(activeArea);  // len存储要把活区中多少行的内容写入输出文件，默认为全部
404. // 若输入文件尚未读完，活区切换命令可将原活区中最后10行留在活区顶部，以保持阅读连续性；否则，它意味着结束编辑或开始编辑另一个文件
405. **if** (countLine(otherArea)) len -= 10;
406. writeToOutputFile(output\_file, activeArea, otherArea, 1, len);
407. }
408. // 删除活区中[begin,end]区间的所有行，若非活区中还存在内容，则将非活区中的内容接到活区的最后一行后面
409. **bool** deleteLine(ActiveArea& activeArea, ActiveArea& otherArea, **int** begin, **int** end)
410. {
411. **if** (!CHECK\_AREA(activeArea)) **return** **false**;
412. **if** (!CHECK\_BOUND(begin, end, countLine(activeArea), 1)) **return** **false**;
413. // 删除[begin,end]内的所有行
414. **int** len = end - begin + 1;  // len存储删除的总行数
415. Line\* cur = activeArea;
416. **while** (cur->next && -- begin) cur = cur->next;  // 让cur找到第begin行的前一行
417. Line\* left = cur;  // left指向第begin行的前一行
418. // 删除[begin,end]内的所有行，并让cur指向第end行的后一行
419. **int** tmp\_length = len;  // 临时变量存储len
420. cur = cur->next;  // 此时cur指向第beg行
421. **while** (cur && tmp\_length -- )  // 依次删除len行
422. {
423. Line\* tmp = cur;
424. cur = cur->next;
425. **delete** tmp;
426. } // 此时cur指向第end行的后一行
427. left->next = cur;
428. // 修正end行后面所有行的行号
429. **while** (cur)
430. {
431. cur->line\_no -= len;
432. cur = cur->next;
433. }
434. // 更新行数
435. activeArea->line\_no -= len;
436. // 若非活区中还有内容，则读入到活区中，读入的行数与删除的行数相同
437. **if** (!emptyArea(otherArea)) readFromOtherArea(activeArea, otherArea, len);
438. **return** **true**;
439. }
440. // 在活区activeArea中的[begin,end]行区间内匹配字符串match\_str，并把结果放入Pos数组中，cnt为Pos数组长度，type指定匹配算法，默认使用KMP算法进行匹配。若匹配成功，返回true
441. **bool** matchString(ActiveArea& activeArea, **char**\* match\_str, **int** begin, **int** end, Pos positions[], **int**& cnt, **int** type = Hazel::KMP)
442. {
443. **if** (!CHECK\_AREA(activeArea) || !CHECK\_BOUND(begin, end, countLine(activeArea), 1)) **return** **false**;
444. cnt = 0;  // cnt初始化成0
445. **int** len = end - begin + 1;  // len存储总共有多少行需要匹配
446. Line\* cur = activeArea;
447. **while** (cur->next && begin -- ) cur = cur->next;  // 找到第begin行，并让cnt指向第begin行
448. **while** (cur && len -- )
449. {
450. **char** line\_content[Hazel::MAXLINELEN];  // 存储当前行的文本内容
451. blocks\_to\_str(cur->content, line\_content);  // 将当前行块转成一个大的字符数组
452. // 在当前行内进行匹配（kmp）
453. **int** pos[Hazel::MAXLINELEN] = {0}, matched\_cnt = 0;  // pos中位置下标从0开始，因此下边需要+1
454. // 指定匹配算法
455. **if** (type == Hazel::KMP) kmp(line\_content, match\_str, pos, matched\_cnt);
456. **else** **if** (type == Hazel::TRIE) trie(line\_content, match\_str, pos, matched\_cnt);
457. **else** **if** (type == Hazel::STRINGHASH) string\_hash(line\_content, match\_str, pos, matched\_cnt);
458. **else** **if** (type == Hazel::BF) bf(line\_content, match\_str, pos, matched\_cnt);
459. **else** kmp(line\_content, match\_str, pos, matched\_cnt);
460. // 将匹配成功的字符串位置存入positions中
461. **for** (**int** i = 0; i < matched\_cnt; i ++ ) positions[cnt ++ ] = {cur->line\_no, pos[i] + 1};
462. // 继续进行下一行匹配
463. cur = cur->next;
464. }
465. INFO("Match Ends")
466. **return** cnt > 0;  // 若cnt大于0，说明匹配到了，否则，没有匹配到
467. }
468. // 将活区中第line\_no行的字符串original替换成字符串replaced，cnt记录替换成功次数，type指定匹配算法，默认使用KMP算法
469. **bool** replaceString(ActiveArea& activeArea, **int** line\_no, **char**\* original, **char**\* replaced, **int**& cnt, **int** type)
470. {
471. **if** (!CHECK\_AREA(activeArea)) **return** **false**;
472. **if** (line\_no < 1 || line\_no > countLine(activeArea))
473. {
474. ERROR("The parameter 'line\_no' is invalid")
475. **return** **false**;
476. }
477. **int** tmp = line\_no;  // 临时变量记录目标行号
478. **char** line\_content[Hazel::MAXLINELEN];  // 存储当前行所有文本内容
479. Line\* cur = activeArea;
480. **while** (cur->next && tmp -- ) cur = cur->next;  // 找到第line\_no行
481. blocks\_to\_str(cur->content, line\_content);  // 将第line\_no行的每一块合并成一个完整的字符串赋值给line\_content
482. // 字符串匹配
483. **int** pos[Hazel::MAXLINELEN] = {0};  // pos存储匹配成功的下标
484. // 指定匹配算法
485. **if** (type == Hazel::KMP) kmp(line\_content, original, pos, cnt);
486. **else** **if** (type == Hazel::TRIE) trie(line\_content, original, pos, cnt);
487. **else** **if** (type == Hazel::STRINGHASH) string\_hash(line\_content, original, pos, cnt);
488. **else** **if** (type == Hazel::BF) bf(line\_content, original, pos, cnt);
489. **else** kmp(line\_content, original, pos, cnt);
490. // 枚举每个替换点
491. **int** original\_length = strlen(original);
492. **int** replaced\_length = strlen(replaced);
493. **int** last = 0;  // last存储替换成新串后整个字符串的偏移量
494. **for** (**int** i = 0; i < cnt; i ++ )
495. {
496. **int** idx = pos[i] + last;  // 获取匹配到的每个字符在母串line\_content中的起点下标，需要加上偏移量
497. // idx始终指向匹配到的字符串在line\_content中的下标
498. **if** (idx > Hazel::MAXLINELEN) **break**;  // 若超出范围，直接退出，替换结束
499. eraseString(line\_content, idx, idx + original\_length - 1);  // 删除字符串line\_content中的original字符串
500. insertString(line\_content, idx, replaced);  // 在第idx个字符后面插入字符串replaced
501. last += replaced\_length - original\_length;  // 更新偏移量last
502. }
503. str\_to\_blocks(line\_content, cur->content);  // 将修改后的line\_content“还给”cur->content
504. // !注意：若用临时变量LineBlock\* content记录cur->content不会把操作完的结果映射到activeArea上
505. **return** cnt > 0;
506. }
507. // 将活区中第[begin,end]区间内所有行的字符串original替换成字符串replaced，cnt记录替换成功次数，type指定匹配算法，默认使用KMP算法
508. **bool** replaceString(ActiveArea& activeArea, **char**\* original, **char**\* replaced, **int** begin, **int** end, **int**& cnt, **int** type)
509. {
510. **if** (!CHECK\_AREA(activeArea) || !CHECK\_BOUND(begin, end, countLine(activeArea), 1)) **return** **false**;
511. **for** (**int** i = begin; i <= end; i ++ )
512. {
513. **int** line\_match\_cnt = 0;  // 存储每一行成功匹配的次数
514. replaceString(activeArea, i, original, replaced, line\_match\_cnt, type);
515. cnt += line\_match\_cnt;
516. }
517. INFO("Replace Ends")
518. **return** cnt > 0;
519. }
521. /\* 4. LineBlock相关函数的实现 \*/
523. // 将行块插入到line\_block后面，尾插法
524. **bool** insertLineBlock(LineBlock\*& line\_block, **char**\* text)
525. {
526. **if** (emptyLineBlock(line\_block))  // 若该行为空，则直接插入
527. {
528. line\_block = **new** LineBlock(text);
529. **return** **true**;
530. }
531. // 插入到行的尾部。因为这只是一个行编辑器。
532. LineBlock\* cur = line\_block;
533. **while** (cur->next) cur = cur->next;
534. LineBlock\* new\_block = **new** LineBlock(text);
535. cur->next = new\_block;
536. **return** **true**;
537. }
538. // 判断LineBlock是否为空
539. **bool** emptyLineBlock(**const** LineBlock\* block)
540. {
541. **return** block == nullptr;
542. }
543. // 清空LineBlock
544. **bool** clearLineBlock(LineBlock\*& blocks)
545. {
546. **if** (!CHECK\_BLOCK(blocks)) **return** **false**;
547. // 清空每一块字符数组
548. **while** (blocks)
549. {
550. LineBlock\* tmp = blocks;  // 临时指针，指向要删除的那个节点
551. blocks = blocks->next;  // 头指针后移
552. **delete** tmp;  // 删除节点
553. }
554. blocks = nullptr;  // 置空
555. **return** **true**;
556. }
557. // 将src中的每一块拷贝给字符数组dist
558. **void** blocks\_to\_str(LineBlock\* src, **char**\* dist)
559. {
560. LineBlock\* cur = src;
561. // 将每一块拷贝给dist
562. **int** len = 0;  // len存储dist字符数组长度
563. **while** (cur)
564. {
565. **for** (**int** i = 0; cur->data[i]; i ++ ) dist[len ++ ] = cur->data[i];  // 按字符拷贝
566. cur = cur->next;
567. }
568. dist[len] = '\0';  // 标记字符结束符
569. }
570. // 将字符数组src按照80个字符一块拷贝给dist
571. **void** str\_to\_blocks(**char**\* src, LineBlock\*& dist)
572. {
573. **if** (!emptyLineBlock(dist)) clearLineBlock(dist);  // 先清空dist
574. // 再将src分块
575. **int** len = strlen(src);  // len存储当前还剩下多少字符没处理
576. **char** block\_content[Hazel::MAXBLOCKLEN];  // 存储每一块的文本
577. **int** idx = 0;  // 记录src的下标索引
578. **while** (len > 0)  // 依次处理每一块
579. {
580. **int** block\_len = 0;  // 记录当前块的大小
581. **for** (**int** i = idx; src[i]; i ++ )  // 将src中的内容拷贝到每一块中
582. {
583. **if** (block\_len == Hazel::MAXBLOCKLEN - 1) **break**;
584. block\_content[block\_len ++ ] = src[i];
585. }
586. block\_content[block\_len] = '\0';  // 将最后一个字符置为'\0'，表示块结束
587. insertLineBlock(dist, block\_content);  // 将该块内容插入到dist中
588. len -= Hazel::MAXBLOCKLEN - 1;  // 更新len
589. }
590. }

593. /\* 5. 对文件操作的一些函数 \*/
595. // 将输入文件中的内容读入活区中，读取失败返回false，若output\_file为空，则表示读入非活区中
596. **bool** readFromInputFile(**char**\* input\_file, ActiveArea& activeArea, ActiveArea& otherArea, **char**\* output\_file)
597. {
598. std::ifstream ifs(input\_file);
599. **if** (!ifs.is\_open())
600. {
601. ERROR("Input File: '" << input\_file << "' opening failed")
602. **return** **false**;
603. }
604. std::cout << "Reading from the file: '" << Hazel::MAGENTA << input\_file << Hazel::RESET <<  "' ...\n";
605. **char** text[Hazel::MAXLINELEN];
606. **int** cnt = countLine(activeArea);  // cnt记录当前活区行数
607. // 说明正在往活区中读入内容（此时非活区必须为空）
608. **if** (output\_file != nullptr)
609. {
610. **while** (ifs.getline(text, Hazel::MAXLINELEN) && cnt < Hazel::MAXACTIVELEN - 20)  // 最多读取到ACTIVEMAXLEN - 20行
611. insertLine(activeArea, text, cnt ++ , otherArea, output\_file);  // 将text插入到第cnt行后面
612. activeArea->line\_no = cnt;  // 更新活区的长度（在insertLine中其实会自动更新当前活区长度）
613. // 将剩余部分读入非活区，非活区无长度限制
614. **if** (strlen(text) != 0) insertLine(otherArea, text, 0, activeArea, nullptr);  // 读取第81行到非活区中
615. cnt = 1;  // 此时cnt记录非活区长度，接下来从第82行开始读，并且从otherArea的第1行开始写
616. **while** (ifs.getline(text, Hazel::MAXLINELEN)) insertLine(otherArea, text, cnt ++ , activeArea, nullptr);  // 剩余部分读入非活区
617. }
618. // 说明正在往非活区中读入内容
619. **else**
620. {
621. **while** (ifs.getline(text, Hazel::MAXLINELEN)) insertLine(otherArea, text, cnt ++ , activeArea, nullptr);
622. }
623. INFO("End of Reading")
624. ifs.close();  // 关闭文件输入流
625. **return** **true**;
626. }
627. // 将活区中区间[begin, end]行的内容写入输出文件
628. **bool** writeToOutputFile(**char**\* output\_file, ActiveArea& activeArea, ActiveArea& otherArea, **int** begin, **int** end)
629. {
630. std::ofstream ofs(output\_file, std::ios::app);  // 以追加的方式打开文件
631. **if** (!ofs.is\_open())
632. {
633. ERROR("Output file opening failed")
634. **return** **false**;
635. }
636. **if** (!CHECK\_BOUND(begin, end, countLine(activeArea))) **return** **false**;
637. // 将区间内的所有行写入输出文件中
638. std::cout << "Writing to the file: '" << Hazel::MAGENTA << output\_file << Hazel::RESET <<"' ...\n";
639. **int** len = end - begin + 1;  // len存储要写入输出文件的总行数
640. **char** text[Hazel::MAXBLOCKLEN];
641. Line\* cur = activeArea;
642. **while** (cur->next && -- begin) cur = cur->next;  // 找到第begin行的前趋节点
643. Line\* left = cur;  // left指向第begin行的前趋
644. cur = cur->next;  // 让cur指向第begin行
645. **for** (**int** i = 0; i < len && cur; i ++ )  // 依次写入第begin行到第end行
646. {
647. Line\* tmp = cur;
648. **char** text[Hazel::MAXLINELEN];
649. blocks\_to\_str(cur->content, text);  // 将行块转成char字符数组
650. ofs << text << std::endl;  // 依次写入输出文件中
651. cur = cur->next;  // 继续下一行
652. clearLineBlock(tmp->content);  // 清空当前行
653. **delete** tmp;
654. }
655. left->next = cur;  // 将第left行和end后一行连接起来
656. // 修正第left行后面所有行的行号
657. **while** (cur)
658. {
659. cur->line\_no -= end;  // 后面的行号减去end即可
660. cur = cur->next;
661. }
662. activeArea->line\_no -= len;  // 更新活区当前总行数
663. // 若非活区中还有内容，则从非活区中读入len行到活区中
664. **if** (!emptyArea(otherArea))
665. {
666. readFromOtherArea(activeArea, otherArea, len);  // 需要读取len行非活区的内容
667. }
668. INFO("End of Writing")
669. ofs.close();  // 关闭输出文件流
670. **return** **true**;
671. }
672. // 将非活区中的len行写入活区中 （若不足len行，则全部写入活区）
673. **bool** readFromOtherArea(ActiveArea& activeArea, ActiveArea& otherArea, **int** len)
674. {
675. **if** (!CHECK\_AREA(activeArea) || !CHECK\_AREA(otherArea)) **return** **false**;
676. **int** actualLength = 0;  // 存储实际写入的行数
677. Line\* tail = activeArea;  // tail指向活区的尾节点
678. **while** (tail->next) tail = tail->next;
679. // 将otherArea的前len行读入activeArea
680. Line\* head = otherArea->next;  // head指向非活区的首元节点
681. **while** (head && len -- )
682. {
683. tail->next = head;
684. otherArea->next = head->next;  // otherArea的首元节点后移
685. head = otherArea->next;  // 更新head始终指向非活区的首元节点
686. // tail->next->line\_no += countLine(activeArea);  // 非活区读到活区的每一行的line\_no都要加上偏移量
687. tail = tail->next;  // tail后移
688. tail->next = nullptr;  // tail的next指针置空
689. actualLength ++ ;  // 实际写入行数+1
690. }
691. otherArea->next = head;  // 更新otherArea的next指针指向非活区的首元节点head
692. // Update: 暴力修正活区中的所有行号-len
693. Line\* cur = activeArea->next;
694. **int** no = 1;
695. **while** (cur)
696. {
697. cur->line\_no = no ++ ;
698. cur = cur->next;
699. }
700. // 更新活区和非活区的长度
701. otherArea->line\_no -= actualLength;
702. activeArea->line\_no += actualLength;
703. INFO("Read from Other Area successfully")
704. **return** **true**;
705. }

708. /\* 6. 字符串相关函数的实现 \*/
710. // 在str中第pos个字符的后面插入字符串value
711. **bool** insertString(**char**\* str, **int** pos, **const** **char**\* value)
712. {
713. **int** n = strlen(str);
714. **if** (pos < 0 || pos > n)  // 插入位置不合法
715. {
716. ERROR("The parameter 'pos' is invalid")
717. **return** **false**;
718. }
719. **int** len = strlen(value);  // len表示要插入的字符串的长度
720. // 先将原串插入位置后面的所有字符向后偏移len位，为value的插入腾出空间来
721. **for** (**int** i = n + len; i > pos + len - 1; i -- )
722. {
723. **if** (i >= Hazel::MAXLINELEN - 1) **continue**;  // 如果超出范围，则丢弃
724. str[i] = str[i - len];
725. }
726. // 再把要添加的字符串value拷贝到腾出的那块空间中
727. **for** (**int** i = pos; i < pos + len; i ++ ) str[i] = value[i - pos];
728. // 最后标记上字符结束符即可
729. str[n + len] = '\0';
730. **return** **true**;
731. }
732. // 删除字符串str中[begin,end]区间内的字符
733. **bool** eraseString(**char**\* str, **int** begin, **int** end)
734. {
735. **int** n = strlen(str);
736. **if** (!CHECK\_BOUND(begin, end, n)) **return** **false**;
737. **int** len = end - begin + 1;  // 存储要删除的字符长度
738. **for** (**int** i = end + 1; str[i]; i ++ ) str[i - len] = str[i];  // 将后面的字符往前移len个单位
739. str[n - len] = '\0';  // 标记字符结束符
740. **return** **true**;
741. }
742. // kmp，O(n+m)，匹配母串str中的所有模式串match\_str，并将其位置放入pos数组中，长度为cnt。若匹配成功，返回true
743. **bool** kmp(**char**\* str, **char**\* match\_str, **int** pos[], **int**& cnt)
744. {
745. cnt = 0;  // 初始化cnt为0
746. **int** n = strlen(match\_str), m = strlen(str);  // n为模式串长度，m为母串长度
747. **if** (n > m) **return** **false**;
748. **char** p[Hazel::MAXLINELEN + 1], s[Hazel::MAXLINELEN + 1];  // p：模式串，s：母串。偏移量+1，更好计算
749. memcpy(p + 1, match\_str, std::min(n, Hazel::MAXLINELEN - 1));
750. memcpy(s + 1, str, std::min(m, Hazel::MAXLINELEN - 1));
751. p[n + 1] = s[m + 1] = '\0';  // 标记结束符
752. // 求next数组
753. **int** ne[Hazel::MAXLINELEN + 1] = {0};
754. **for** (**int** i = 2, j = 0; i <= n; i ++ )
755. { // ne[1] = 0，因为如果第一个匹配不上，就只能从0开始重新匹配了
756. **while** (j && p[i] != p[j + 1]) j = ne[j]; // 与KMP匹配过程类似
757. **if** (p[i] == p[j + 1]) j ++ ;
758. ne[i] = j; // 记录ne[i]的值为j
759. }
760. // KMP匹配过程
761. **for** (**int** i = 1, j = 0; i <= m; i ++ )
762. {
763. **while** (j && s[i] != p[j + 1]) j = ne[j]; // 如果s[i]与p[j + 1]不能匹配，将j移动到ne[j]的位置继续与s[i]匹配，直到j为0
764. **if** (s[i] == p[j + 1]) j ++ ; // 匹配成功，j向后移动
765. **if** (j == n) // 此时，整个字符串都匹配成功
766. {
767. pos[cnt ++ ] = i - n;  // 将匹配成功的位置存储pos数组中，下标从0开始
768. j = ne[j]; // 从ne[j]开始，继续下一次匹配
769. }
770. }
771. **return** cnt > 0;
772. }
773. // trie，O(n)，匹配母串str中的所有模式串match\_str，并将其位置放入pos数组中，长度为cnt。若匹配成功，返回true
774. **bool** trie(**char**\* str, **char**\* match\_str, **int** pos[], **int**& cnt)
775. {
776. Trie tr;
777. **if** (strlen(match\_str) > strlen(str)) **return** **false**;
778. **return** tr.search(str, match\_str, pos, cnt);
779. }
780. // 字符串哈希，O(n)，匹配母串str中的所有模式串match\_str，并将其位置放入pos数组中，长度为cnt。若匹配成功，返回true
781. unsigned **long** **long** get(**int** l, **int** r, unsigned **long** **long** h[], unsigned **long** **long** p[])
782. {
783. **return** h[r] - h[l - 1] \* p[r - l + 1];
784. }
785. **bool** string\_hash(**char**\* str, **char**\* match\_str, **int** pos[], **int**& cnt)
786. {
787. unsigned **long** **long** h[Hazel::MAXLINELEN], p[Hazel::MAXLINELEN];  // h:存储母串str中每个子串的哈希值, p[i]:存储p的i次方
788. **const** **int** P = 131;  // 哈希基数，取131或13331
789. **int** n = strlen(match\_str), m = strlen(str);  // n为模式串长度，m为母串长度
790. **if** (n > m) **return** **false**;
791. **char** pt[Hazel::MAXLINELEN + 1], s[Hazel::MAXLINELEN + 1];  // pt：模式串，s：母串。偏移量+1，更好计算
792. memcpy(pt + 1, match\_str, std::min(n, Hazel::MAXLINELEN - 1));
793. memcpy(s + 1, str, std::min(m, Hazel::MAXLINELEN - 1));
794. // 1. 初始化p数组和h数组
795. p[0] = 1;
796. **for** (**int** i = 1; i <= m; i ++ )
797. {
798. p[i] = p[i - 1] \* P;
799. h[i] = h[i - 1] \* P + s[i];
800. }
801. // 2. 求出模式串的哈希值
802. unsigned **long** **long** target\_hash\_value = 0;
803. **for** (**int** i = 1; i <= n; i ++ ) target\_hash\_value = target\_hash\_value \* P + pt[i];
804. // 3. 在母串中匹配模式串，寻找与模式串哈希值相同的子串
805. **for** (**int** l = 1; l + n - 1 <= m; l ++ )  // 枚举左端点
806. {
807. **int** r = l + n - 1;  // 右端点
808. **if** (get(l, r, h, p) == target\_hash\_value) pos[cnt ++ ] = l;
809. }
810. **return** cnt > 0;
811. }
812. // bf，O(nm)，匹配母串str中的所有模式串match\_str，并将其位置放入pos数组中，长度为cnt。若匹配成功，返回true
813. **bool** bf(**char**\* str, **char**\* match\_str, **int** pos[], **int**& cnt)
814. {
815. cnt = 0;
816. **int** n = strlen(str), m = strlen(match\_str);
817. **if** (m > n) **return** **false**;
818. // 暴力枚举每次匹配的起点
819. **for** (**int** i = 0; i <= n - m; i ++ )
820. {
821. **int** j;
822. **for** (j = 0; j < m; j ++ )  // 枚举匹配长度
823. **if** (str[i + j] != match\_str[j])
824. **break**;
825. // 若匹配成功，保存位置
826. **if** (j == m) pos[cnt ++ ] = i;
827. }
828. **return** cnt > 0;
829. }

832. /\* 主函数 \*/
833. **int** main(**int** argc, **char**\* argv[])
834. {
835. ActiveArea activeArea = nullptr, otherArea = nullptr;
836. initArea(activeArea);
837. initArea(otherArea);
839. **char** input\_file[Hazel::MAXFILENAMELEN + 1], output\_file[Hazel::MAXFILENAMELEN + 1];
841. **if** (argc >= 3)
842. {
843. strcpy(input\_file, argv[1]);
844. strcpy(output\_file, argv[2]);
845. }
846. **else**
847. {
848. ERROR("Usage: " << argv[0] << " <input\_file> <output\_file>")
849. readFile(input\_file, output\_file);  // 若未从参数传入文件信息，则自行输入文件信息
850. std::cout << input\_file << ", " << output\_file << std::endl;
851. }
853. // 若有输入文件，则先让程序从输入文件中读取文本
854. **if** (strlen(input\_file) == 0)
855. {
856. readFromInputFile(input\_file, activeArea, otherArea, output\_file);
857. }
858. // 开始读入操作
859. **char** op[Hazel::MAXLINELEN] = {0};
860. **int** count = 0;  // count记录当前已经进行多少次操作
861. **while** (**true**)
862. {
863. // 打印上次操作
864. TRACE("last operation: " + (strlen(op) == 0 ? std::string("none") : std::string(op)))
865. // 显示活区
866. showActiveArea(activeArea, 1, countLine(activeArea), 1);
867. // 展示主操作界面
868. showMainMenu();
869. // 读入本次操作
870. std::cin.getline(op, **sizeof** op);
871. // 操作次数+1
872. count ++ ;
874. /\* 判断操作类型 \*/
875. // 1. 退出系统
876. **if** (strcmp(op, "exit") == 0)
877. {
878. **break**;
879. }
880. // 2. 显示帮助菜单
881. **else** **if** (strcmp(op, "help") == 0)
882. {
883. showHelpMenu();
884. **continue**;
885. }
886. // 3. 行插入
887. **else** **if** (op[0] == 'i')
888. {
889. **char** str[Hazel::MAXLINELEN];
890. std::cin.getline(str, **sizeof** str);
891. **int** idx = 0;
892. getNumber(op, idx, 1, strlen(op) - 1);  // 插入到第idx行
893. insertLine(activeArea, str, idx, otherArea, output\_file);  // 调用插入函数
894. }
895. // 4. 删除行
896. **else** **if** (op[0] == 'd')
897. {
898. **int** i;
899. **for** (i = 1; op[i] && op[i] != ' '; i ++ );  // 让i指向空格位置
900. **int** begin = 0, end = 0;  // 删除行的上下界
901. getNumber(op, begin, 1, i - 1);
902. **if** (op[i] && op[i] == ' ') getNumber(op, end, i + 1, strlen(op) - 1);
903. **else** end = begin;
904. // 若可删除，则调用删除函数
905. **if** (!emptyArea(activeArea)) deleteLine(activeArea, otherArea, begin, end);
906. }
907. // 5. 活区切换
908. **else** **if** (op[0] == 'n' && strlen(op) == 1)
909. {
910. changeActiveArea(activeArea, otherArea, output\_file);  // 调用活区切换函数
911. }
912. // 6. 活区显示
913. **else** **if** (op[0] == 'p' && strlen(op) == 1/\* || count == 3\*/)
914. {
915. **int** cur\_line = 1, show\_lines = 20;  // 起始显示行数和一次展示多少行
916. **int** page\_num = 1;  // 存储当前页数
917. **while** (**true**)
918. {
919. showActiveArea(activeArea, cur\_line, show\_lines, page\_num ++ );  // 显示一次后页数+1
920. cur\_line += show\_lines;
921. **if** (cur\_line > countLine(activeArea))  // 显示完毕，退出
922. {
923. TRACE("End of Active Area")
924. std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  // 过滤最后一个'Y'的回车
925. **break**;
926. }
927. std::cout << "Do you want to continue to the next page? (Y/N): ";
928. **char** choice; std::cin >> choice;  // 输入选择
929. **if** (choice != 'Y') **break**;  // 若不继续显示或者显示完毕，退出
930. }
931. }
932. // 7. 字符串替换
933. **else** **if** (op[0] == 'S')
934. {
935. **int** begin = 0, end = 0;
936. **int** i, j;
937. **for** (i = 1; op[i] != '@'; i ++ );  // 让i指向第一个'@'处
938. **for** (j = i + 1; op[j] != '@'; j ++ );  // 让j指向第二个'@'处
939. **char** original[Hazel::MAXLINELEN], replaced[Hazel::MAXLINELEN];
940. **int** k = 1;
941. **while** (op[k] && op[k] != ',') k ++ ;  // 让k指向逗号处（或字符串结束符'\0'处）
942. **if** (op[k] == ',')  // 说明存在<行号2>
943. {
944. getNumber(op, begin, 1, k - 1);  // 获取<行号1>
945. getNumber(op, end, k + 1, i - 1);  // 获取<行号2>
946. }
947. **else**
948. {
949. getNumber(op, begin, 1, i - 1);  // 仅获取<行号1>
950. end = begin;
951. }
952. getString(op, original, i + 1, j - 1);  // 获取原串
953. getString(op, replaced, j + 1, strlen(op) - 1); // 获取替换串
954. **int** replaced\_cnt = 0;  // 存储替换次数
955. replaceString(activeArea, original, replaced, begin, end, replaced\_cnt);  // 调用替换字符串函数（默认使用KMP）
956. std::cout << Hazel::CYAN << "total replaced counts: " << replaced\_cnt << std::endl;
957. }
958. // 8. 字符串匹配
959. **else** **if** (op[0] == 'm')
960. {
961. **char** match\_str[Hazel::MAXLINELEN];
962. **int** i = 1;
963. **while** (op[i] && op[i] == ' ') i ++ ;  // 过滤空格，让i指向待匹配串的起始下标
964. **int** j = i;
965. **while** (op[j] && op[j] != ' ') j ++ ;  // 让j指向下一个空格处（或字符串结束符'\0'处）
966. getString(op, match\_str, i, j - 1);  // 获取待匹配字符串
967. i = j;
968. **while** (op[i] && op[i] == ' ') i ++ ;  // 过滤空格，让i指向<行号1>起始下标处
969. j = i;
970. **while** (op[j] && op[j] != ' ') j ++ ;  // 过滤<行号1>，让j指向下一个空格处（或'\0'处）
971. **int** begin = 0, end = 0;
972. **if** (i == j)  // 说明只输入的匹配串
973. {
974. begin = 1, end = countLine(activeArea);
975. }
976. **else**
977. {
978. getNumber(op, begin, i, j - 1);  // 获取行号1
979. i = j;
980. **while** (op[i] && op[i] == ' ') i ++ ;  // 过滤空格，让i指向<行号2>起始下标处
981. j = i;
982. **while** (op[j] && op[j] != ' ') j ++ ;  // 过滤<行号2>，让j指向最终结束符'\0'处
983. **if** (i == j)  // 说明只输入了<行号1>
984. end = begin;
985. **else**  // 否则，获取<行号2>
986. getNumber(op, end, i, j - 1);
987. }
988. Pos positions[Hazel::MAXLINELEN \* Hazel::MAXACTIVELEN] = {0};  // 存储匹配成功的位置
989. **int** matched\_cnt = 0;  // 记录匹配成功的个数
990. // 匹配活区内所有match
991. **bool** is\_matched = matchString(activeArea, match\_str, begin, end, positions, matched\_cnt);
992. std::cout << Hazel::CYAN << "total matches: " << matched\_cnt << Hazel::RESET << std::endl;
993. **if** (is\_matched) printPositions(positions, matched\_cnt);  // 若匹配成功，则打印所有匹配到的位置
994. }
995. // 9. 读取输入文件到活区中（前提是非活区中无内容）
996. **else** **if** (op[0] == 'r')
997. {
998. **int** i = 1;
999. **while** (op[i] && op[i] == ' ') i ++ ;  // 过滤空格，i指向<文件名>起始下标
1000. getString(op, input\_file, i, strlen(op) - 1);
1001. **if** (!emptyArea(otherArea))  // 若非活区中还存在内容，则无法读取
1002. {
1003. WARNINGS("Other Area is not empty, can't read from Input File: '" <<  input\_file << "\'")
1004. **continue**;
1005. }
1006. readFromInputFile(input\_file, activeArea, otherArea, output\_file);
1007. }
1008. // 10. 写入输出文件
1009. **else** **if** (op[0] == 'w')
1010. {
1011. **int** i = 1;
1012. **while** (op[i] && op[i] == ' ') i ++ ;  // 过滤空格，i指向<文件名>起始下标
1013. **int** j = i;
1014. **while** (op[j] && op[j] != ' ') j ++ ;  // 过滤文件名，j指向下一个空格处（或'\0'处）
1015. getString(op, output\_file, i, j - 1);  // 获取<文件名>
1016. **while** (op[j] && op[j] == ' ') j ++ ;  // 过滤空格，j指向<行号1>处
1017. i = j;
1018. **while** (op[i] && op[i] != ' ') i ++ ;  // 过滤第一个数字，i指向下一个空格处（或'\0'处）
1019. **int** begin = 0, end = 0;
1020. **if** (i == j) begin = 1, end = countLine(activeArea);  // 说明只输入了<文件名>
1021. **else**
1022. {
1023. getNumber(op, begin, j, i - 1);  // 获取<行号1>
1024. **while** (op[i] && op[i] == ' ') i ++ ;  // 过滤空格，i指向<行号2>处
1025. j = i;
1026. **while** (op[j]) j ++ ;  // 过滤第二个数字，j指向最后字符串结束符'\0'处
1027. **if** (i == j) end = begin;  // 说明只输入了<行号1>
1028. **else** getNumber(op, end, i, j - 1);  // 获取<行号2>
1029. }
1030. // 写入获取中区间[begin,end]行的内容到输出文件中
1031. writeToOutputFile(output\_file, activeArea, otherArea, begin, end);
1032. }
1033. // 9. 非法输入
1034. **else**
1035. {
1036. **if** (count == 1)  // 第一次不是非法操作
1037. {
1038. clearScreen();
1039. **continue**;
1040. }
1041. WARNINGS("Invalid Input!")
1042. }
1043. // 按下回车以继续
1044. std::cout << Hazel::GREEN << "Press enter to continue..." << Hazel::RESET;
1045. std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  // 判断只接收回车
1046. clearScreen();  // 每操作一次后都清屏
1047. }
1049. **return** 0;
1050. }

## 六、测试数据及测试结果

#### 测试数据

输入：

input.txt output.txt

help

i1<回车>hello world

d4 6

m hello

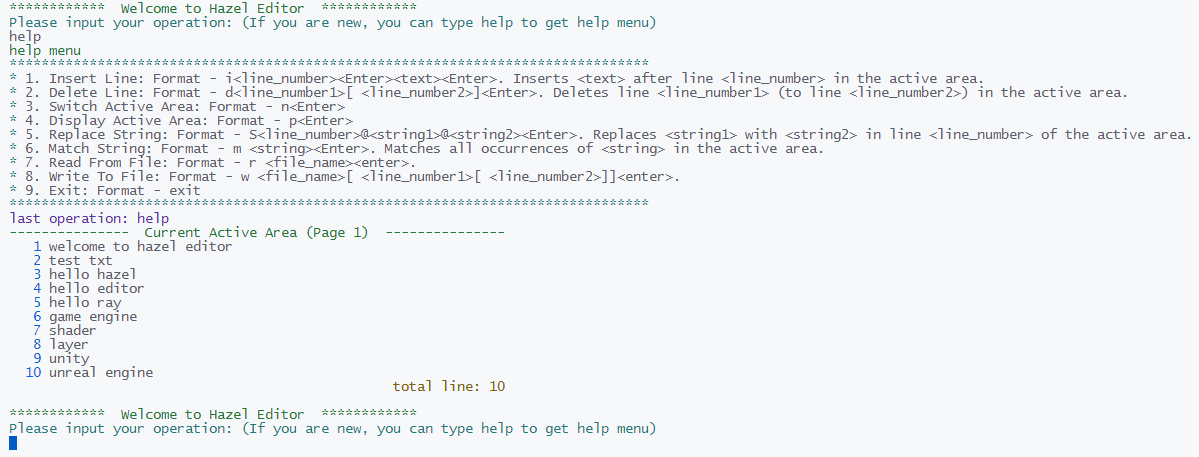
S2@world@hazel

p

exit

#### 测试结果





图片包含 文本

描述已自动生成

文本

描述已自动生成

图片包含 文本

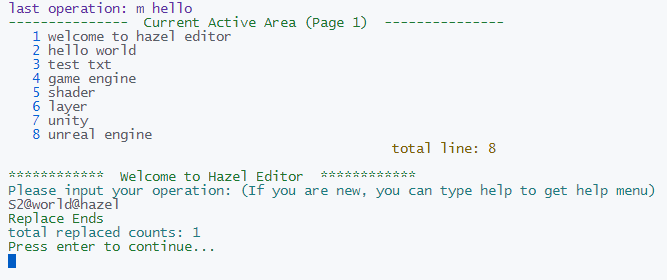
描述已自动生成

文本

描述已自动生成

图片包含 文本

描述已自动生成



文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

## 六、课程设计总结及心得体会

### 1. 心得体会

本次课设使我对链表的使用更加的娴熟，在看完这道题后的思路应该是这道题如果能用STL库，那么我会选择vector<string>来存储活区的文本内容，基于我之前自己实现的Vector和String，也就是使用动态数组来存储内容，就可以很轻易的写出这道题，但是我最终采用的是嵌套链表的形式来实现该编辑器。一来可以尝试新方式来实现这种vector<string>的结构，二来可以很好的复习链表的知识。

### 2. 遇到的问题

1. 在链表初始化过程中栽了不少跟头，于是写了一个CHECK\_AREA()函数用来判断是否已经初始化（然而用C++的类就不用担心这些小问题）。
2. 当输入文件中文本内容多于80行后，会把80行往后的文本读入非活区，在按下n将活区内容写入输出文件后，需要修正活区每一行的行号。在修正行号过程中，经过多次模拟尝试终于把修正错误的情况全部规避了。
3. 在将行块与字符数组的相互转换过程中，会用到清空行块的操作，而clearLineBlock函数由于传入参数没带上引用而导致只清空了函数内部的行块，而不是清空调用处的行块，从而导致没有正常删除该行，在调试的过程中发现该bug并及时修复。
4. 将活区多次写入输出文件中，并且从非活区（或者外部文件）中读入新文本时，修复了该过程中行号修正bug。

### 3. 总结

经过这几天的不懈努力和调试，最终成果：该编辑器不仅能够支持需求中所有的功能，还额外新增了一些其他功能：

1. 新增帮助文档。键入help即可获取帮助文档。
2. 新增输入文件的切换。当一个输入文件中的内容全部处理完毕后，可以选择继续从其他文件中读入内容，而不用退出程序。命令：r <filename>。
3. 新增输出文件的切换。当需要把文本写入输出文件时，提供了更换输出文件的功能，用户可以随时更改输出文件；并且还增加了写入的行区间，用户可以自定义想要写到输出文件中的内容。命令：w <filename> <line\_number1> <line\_number2>。
4. 新增字符串匹配区间。用户可以自定义字符串匹配功能的匹配区间，而不是每次都匹配活区中的全部内容，命令：m <string> <line\_number1> <line\_number2>。
5. 新增字符串替换区间。用户可以自定义字符串替换功能的替换区间，而不是只能匹配某一行。命令：S<line\_number1>,<line\_number2>@<string1>@<string2>。
6. 新增退出编辑器功能。命令：exit。