项目说明文档

数据结构课程设计

——关键词检索系统

作 者 姓 名　林继申

学　　号　2250758

指 导 教 师　张　颖

学 院 专 业　软件学院　软件工程

同济大学
TONGJI UNIVERSITY

二〇二三 年 十二 月 十三 日

目录

1 项目分析

1.1 项目背景分析

关键词检索系统是一种常见的应用程序，旨在帮助用户在大量文本数据中快速查找和定位特定关键词或短语。这种系统在各种领域都有广泛的应用，包括信息检索、文档管理、搜索引擎、学术研究等。关键词检索系统可以大大提高用户的工作效率，尤其是在处理大规模文本数据时。

本项目是一个关键词检索系统的实现，旨在为用户提供一种简单而有效的方式来检索文本文件中的特定关键词。用户可以输入关键词和选择不同的字符串模式匹配算法来执行搜索操作。该系统支持两种主要的字符串匹配算法：BF（Brute-Force）算法和 KMP（Knuth-Morris-Pratt）算法。

1.2 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)文件操作：用户能够创建新的文本文件，并向其输入文本内容。用户还能够打开新创建的文本文件以进行关键词检索；

(2)关键词输入：用户能够输入待搜索的关键词。系统需要验证输入的关键词是否符合指定的格式要求；

(3)字符串模式匹配算法：系统支持两种不同的字符串匹配算法，包括 BF（Brute-Force）算法和 KMP（Knuth-Morris-Pratt）算法。用户可以选择其中一种算法来执行关键词检索；

(4)关键词检索：系统会根据用户输入的关键词和选择的算法，在文本文件中执行关键词检索，并计算关键词的出现次数；

(5)性能统计：系统会记录关键词检索所用的时间，以便用户了解检索的效率；

(6)用户界面：系统提供简单的命令行界面，让用户能够轻松地进行操作和查看检索结果。

1.3 项目功能分析

本项目旨在通过文件操作、关键词输入、关键词检索和异常处理等功能，以实现关键词检索系统的功能。下面对项目的功能进行详细分析。

1.3.1 文件操作功能

1.3.1.1 创建文件功能

通过本项目，用户可以创建一个新的文本文件，用于存储文本内容和执行关键词检索操作。程序提供了创建文件的功能，用户可以指定文件名，然后将其创建。

1.3.1.2 写入文件功能

用户可以在创建的文本文件中输入文本内容。这个功能允许用户逐字符输入文本，直到按下回车键为止。程序将用户输入的文本内容写入文件中。

1.3.1.3 读取文件功能

项目还提供了读取文件的功能，用户可以查看文本文件的内容。这有助于用户核对他们输入的文本内容，并在需要时执行关键词检索操作。

1.3.2 关键词输入功能

用户可以输入一个关键词，该关键词将被用于关键词检索操作。程序会要求用户输入一个不包含空格的字符串，然后将其用于后续的检索操作。

1.3.3 关键词检索功能

项目提供了两种关键词检索算法，分别是 BF（Brute-Force）算法和 KMP（Knuth-Morris-Pratt）算法，用户可以选择其中一种算法进行关键词检索。

1.3.3.1 BF(Brute-Force)算法

BF（Brute-Force）算法是一种朴素的字符串匹配算法，用于检测文本中是否包含用户输入的关键词。算法逐字符比较关键词和文本内容，以查找匹配。算法的执行效率相对较低，但对于小型文本文件仍然是有效的。

1.3.3.2 KMP(Knuth-Morris-Pratt)算法

KMP（Knuth-Morris-Pratt）算法是一种高效的字符串匹配算法，用于检测文本中是否包含用户输入的关键词。算法利用预处理的 next 数组来跳过部分匹配，从而提高匹配速度。算法的执行效率比 BF（Brute-Force）算法更高，特别适用于大型文本文件的关键词检索。

1.3.4 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

2 项目设计

2.1 数据结构设计

基于项目分析，在本项目中，待检索文本存储在文本文件中，并且检索时直接从文本文件中读取数据，这种方法具有以下好处：

(1)节省内存资源：将待检索文本存储在外部文本文件中，而不是将其加载到内存中，可以大大减少内存消耗。这对于处理大型文本文件尤其重要，因为加载整个文本文件可能会导致内存不足或性能下降；

(2)支持大型文本文件：这种方法允许处理非常大的文本文件，而无需担心内存限制。即使文本文件非常庞大，也可以有效地进行检索；

(3)持久性存储：将文本数据存储在文件中使得数据具有持久性，即使应用程序关闭，数据仍然保存在磁盘上。这对于长期存储和文档管理非常有用；

(4)减少初始化时间：在启动应用程序时，不需要将整个文本文件加载到内存中，因此可以更快速地初始化应用程序。这对于提高用户体验和响应时间很有帮助；

(5)避免数据丢失：将文本数据存储在文件中可以避免因应用程序崩溃或意外关闭而导致数据丢失的风险。文本文件数据会持久保存，可以在需要时重新打开和使用。

总之，将待检索文本存储在外部文本文件中并在检索时直接从文件中读取数据有助于减少内存消耗，支持大型文本文件，提高应用程序性能，以及确保数据的持久性和可靠性。

2.2 KeywordSearch 类的设计

2.2.1 概述

KeywordSearch 类是关键词检索系统的核心部分，它包含了文件操作、关键词输入、关键词检索的功能。这个类允许用户初始化文件、输入文本内容和关键词，执行 BF（Brute-Force）和 KMP（Knuth-Morris-Pratt）两种字符串模式匹配算法进行关键词检索，以及输出检索结果。

2.2.2 类定义

```
class KeywordSearch {
private:
    long long fileLen;
```

```
        int keywordLen;
        char filename[MAX_LENGTH + 1];
        char keyword[MAX_LENGTH + 1];
        char getCharFromFile(std::fstream& file, int index);
        void getNext(int next[]);
    public:
        KeywordSearch(const char* _filename) :fileLen(0),
keywordLen(0), filename{ '\0' }, keyword{ '\0' }
{ strcpy(filename, _filename); }
        void initializeFile(void);
        void inputTextAndKeyword(std::fstream& file);
        void outputText(std::fstream& file);
        int BF_Search(std::fstream& file);
        int KMP_Search(std::fstream& file);
        void search(std::fstream& file, int optn);
    };
```

2.2.3 私有数据成员

long long fileLen：文本文件长度（文本文件中字符的数量）

int keywordLen：关键词长度（关键词中字符的数量）

char filename[MAX_LENGTH + 1]：文本文件名（使用 char 类型数组保存数据，故长度为 MAX_LENGTH + 1，以存储尾零’\0’）

char keyword[MAX_LENGTH + 1]：关键词（使用 char 类型数组保存数据，故长度为 MAX_LENGTH + 1，以存储尾零’\0’）

2.2.4 构造函数

```
KeywordSearch(const char* _filename) :fileLen(0),
keywordLen(0), filename{ '\0' }, keyword{ '\0' }
{ strcpy_s(filename, _filename); }
```

构造函数用于初始化 KeywordSearch 对象，接受一个文件名作为参数。它初始化了文本文件长度、关键词长度以及文本文件名和关键词的字符数组。

2.2.5 私有成员函数

```
char getCharFromFile(std::fstream& file, int index);
```

从文件中读取指定位置的字符。它允许程序从文件中获取一个字符，然后将其用于关键词检索操作。

`void getNext(int next[]);`

为 KMP（Knuth-Morris-Pratt）算法中的匹配操作计算所需的 next 数组，该数组包含了用于跳过部分匹配的信息，以提高匹配效率。

2.2.6 公有成员函数

`void initializeFile(void);`

该函数用于初始化文本文件，创建文件对象。

`void inputTextAndKeyword(std::fstream& file);`

该函数用于输入文本内容和关键词，用户可以逐字符输入文本内容，然后输入一个关键词。它还记录关键词的长度和文本文件的长度，以便后续的检索。

`void outputText(std::fstream& file);`

该函数用于输出文本文件的内容。它会从文件中逐字符读取文本内容，并将其输出到控制台。

`int BF_Search(std::fstream& file);`

该函数实现 BF（Brute-Force）算法的关键词检索，通过逐字符比较文本内容和关键词，以查找匹配，并返回关键词在文本中出现的次数。

`int KMP_Search(std::fstream& file);`

该函数实现 KMP（Knuth-Morris-Pratt）算法的关键词检索，通过预处理的 next 数组来提高匹配速度，并返回关键词在文本中出现的次数。

`void search(std::fstream& file, int optn);`

该函数用于执行关键词检索，接受一个整数参数 optn，表示选择的算法类型。在检索过程中，它会记录执行时间，并输出检索结果。

2.3 项目主体架构设计

项目主体架构设计为：

(1)系统进入提示：程序开始时，输出系统信息并且对用户的操作给出提示；

(2)初始化文件操作：用户被要求输入一个文本文件的名称。程序首先检查文件是否已存在，如果存在则报错并退出（FILE_EXIST_ERROR）。如果文件不存在，程序会尝试创建该文件，如果文件创建失败则报错并退出（FILE_CREATE_ERROR）。成功创建文件后，打印出提示信息；

(3)文本文件操作：创建或打开用户指定的文本文件，以便后续的文本输入和输出操作。如果文件打开失败，则报错并退出（FILE_OPEN_ERROR）；

(4)文本内容输入：用户被要求输入文本内容，直到用户按下回车键为止。输入的文本会被写入到打开的文本文件中；

(5)关键词输入与验证：用户被要求输入一个关键词，该关键词将被用于关

键词检索操作，程序会验证输入的关键词字符串是否为空或是否有空格；

(6)文本内容输出：打印出文本文件的内容，以供用户查看；

(7)选择算法：用户被要求选择字符串模式匹配算法，可以选择 BF（Brute-Force）算法或 KMP（Knuth-Morris-Pratt）算法；

(8)关键词检索

①根据用户选择的算法，创建一个 keywordSearch 对象，并将文本文件传递给该对象；

②执行关键词检索操作，计算检索所用的时间；

③输出关键词在文本中的出现次数和检索时间；

(9)等待程序退出：程序等待用户按下回车键以退出程序。

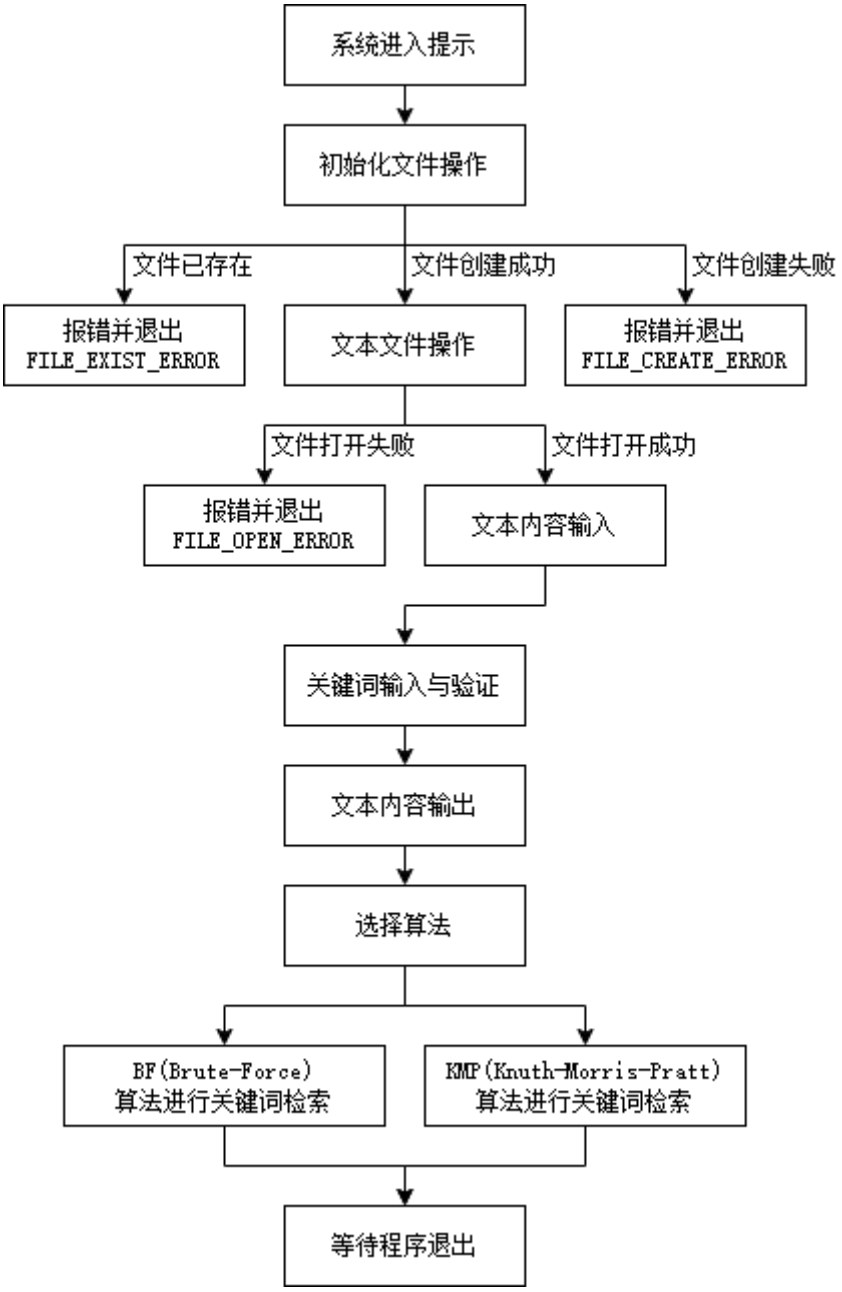

图 2.3.1 项目主体架构设计流程图

3 项目功能实现

3.1 项目主体架构的实现

3.1.1 项目主体架构实现思路

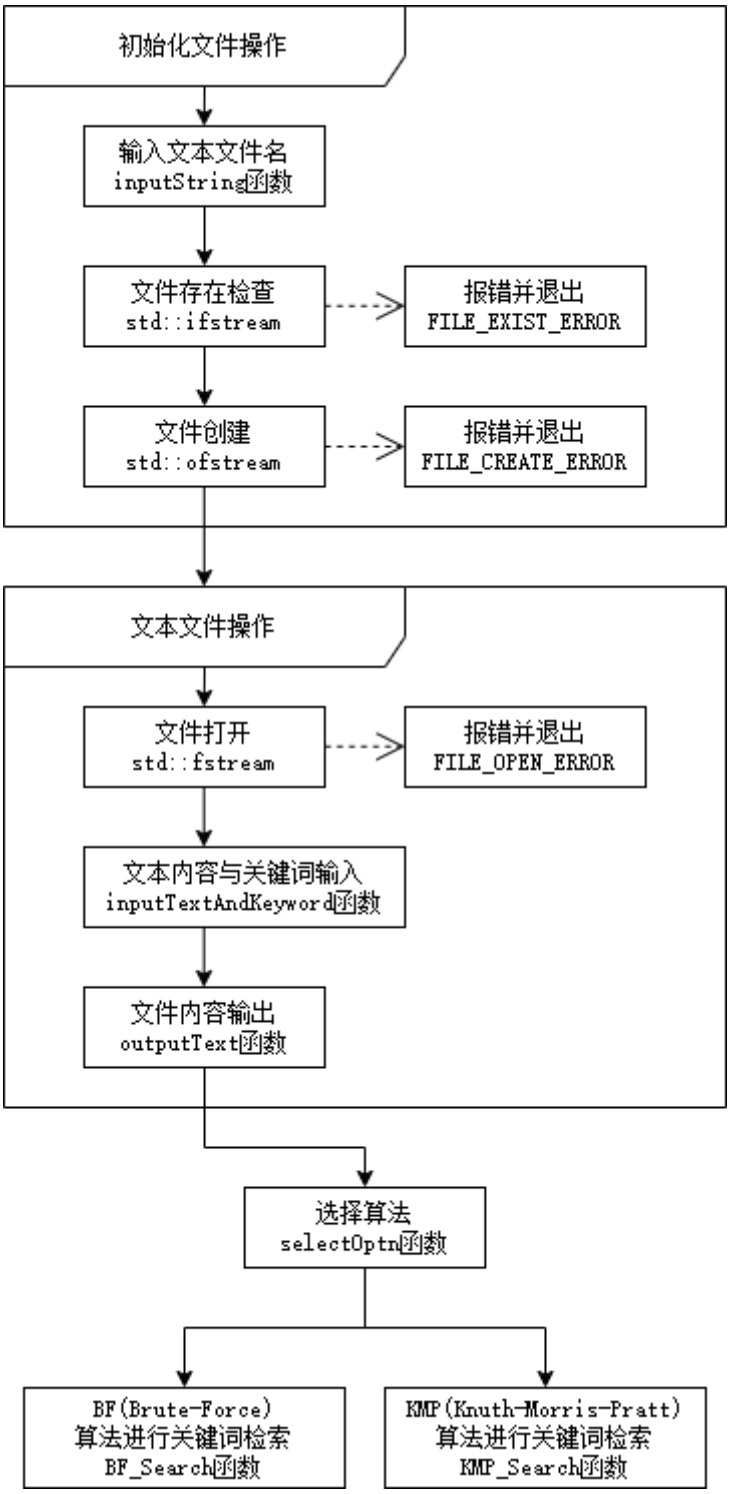

图 3.1.1.1 项目主体架构实现流程图

项目主体架构实现思路为：

⑴系统进入提示：在 main 函数中，首先输出系统信息和操作提示；

⑵初始化文件操作

①用户输入文本文件名，调用 inputString 函数来获取不包含空格的文件名；

②调用 std::ifstream 进行文件存在检查，如果文件存在则报错（FILE_EXIST_ERROR）并退出；

③如果文件不存在，调用 std::ofstream 来尝试创建该文件。如果文件创建失败，报错（FILE_CREATE_ERROR）并退出。如果文件创建成功，打印提示信息；

⑶文本文件操作

①在 main 函数中，通过 std::fstream 打开用户指定的文本文件，以备后续文本输入和输出。如果文件打开失败，则报错（FILE_OPEN_ERROR）并退出；

②文本内容与关键词输入：用户被要求输入文本内容，调用 KeywordSearch 类的成员函数 inputTextAndKeyword，通过 std::cin.get 来获取文本内容的输入，输入的文本被写入已打开的文本文件中。并通过调用 inputString 函数获取不包含空格的关键词。在 inputString 函数可以对字符串为空和字符串中是否含有空格的情况做检查；

③文本内容输出：通过调用 outputText 函数，打印文本文件的内容，供用户查看；

⑷选择算法：用户被要求选择字符串模式匹配算法，通过调用 selectOptn 函数获取用户选择；

⑸关键词检索：根据用户选择的算法，调用 KeywordSearch 类的成员函数 search 执行关键词检索。search 函数内部会根据选择调用 BF_Search 或 KMP_Search 函数，执行关键词检索操作，并使用 Windows API 中的性能计数器来测量执行时间；

⑹等待程序退出：程序在 main 函数中等待用户按下回车键以退出程序。

3.1.2 项目主体架构核心代码

```
int main()
{
    /* System entry prompt */
    std::cout << "+-------------------------+" << std::endl;
    std::cout << "|     关键词检索系统      |" << std::endl;
    std::cout << "|  Keyword Search System  |" << std::endl;
    std::cout  <<  "+-------------------------+"  <<  std::endl  <<
std::endl;
```

```
        /* Initialize file operations */
        char filename[MAX_LENGTH + 1] = { '\0' };
        std::cout << ">>> 请创建文本文件" << std::endl;
        inputString(filename, "文件名");
        std::ifstream fileCheck(filename);
        if (fileCheck) {
            fileCheck.close();
            std::cerr << "Error: File " << filename << " already exists."
<< std::endl;
            exit(FILE_EXIST_ERROR);
        }
        std::ofstream emptyFile(filename);
        if (!emptyFile.is_open()) {
            std::cerr << "Error: File " << filename << " creation failed."
<< std::endl;
            exit(FILE_CREATE_ERROR);
        }
        std::cout << std::endl << ">>> 文本文件 " << filename << " 创建成
功" << std::endl;
        emptyFile.close();
        std::fstream textFile(filename);
        if (!textFile.is_open()) {
            std::cerr << "Error: File " << filename << " open failed." <<
std::endl;
            exit(FILE_OPEN_ERROR);
        }

        /* Keyword search system */
        KeywordSearch keywordSearch(filename);
        keywordSearch.inputTextAndKeyword(textFile);
        keywordSearch.outputText(textFile);
        keywordSearch.search(textFile, selectOptn());
        textFile.close();

        /* Wait for enter to quit */
        std::cout << "Press Enter to Quit" << std::endl;
        while (_getch() != '\r')
            continue;

        /* Program ends */
        return 0;
    }
```

3.1.3 项目主体架构示例

图 3.1.3.1 项目主体架构示例（BF 算法）

图 3.1.3.2 项目主体架构示例（KMP 算法）

3.2 文件操作功能的实现

3.2.1 创建文件功能的实现

3.2.1.1 创建文件功能实现思路

创建文件功能的实现思路如下：

⑴定义一个字符数组 filename 用于存储用户输入的文件名，并初始化为全零（'\0'）；

⑵提示用户输入文件名，使用 inputString 函数，该函数会验证文件名是否包含空格，并将文件名存储在 filename 中；

⑶使用 std::ifstream 来尝试打开用户指定的文件，以进行检查文件是否已经存在。如果文件存在，关闭文件并报告错误；

⑷如果文件不存在，继续执行下一步。创建一个 std::ofstream 对象 emptyFile，并使用用户提供的文件名尝试创建一个空的文本文件。检查文件是否成功打开；

⑸如果文件创建失败，报告文件创建错误并退出程序；

⑹如果文件创建成功，输出创建成功的消息，并关闭 emptyFile 对象；

⑺使用 std::fstream 打开用户指定的文件，以便后续的文本内容输入和搜索操作。

3.2.1.2 创建文件功能核心代码

```
    /* Initialize file operations */
    char filename[MAX_LENGTH + 1] = { '\0' };
    std::cout << ">>> 请创建文本文件" << std::endl;
    inputString(filename, "文件名");
    std::ifstream fileCheck(filename);
    if (fileCheck) {
        fileCheck.close();
        std::cerr << "Error: File " << filename << " already exists." <<
std::endl;
        exit(FILE_EXIST_ERROR);
    }
    std::ofstream emptyFile(filename);
    if (!emptyFile.is_open()) {
        std::cerr << "Error: File " << filename << " creation failed." <<
std::endl;
        exit(FILE_CREATE_ERROR);
    }
    std::cout << std::endl << ">>> 文本文件 " << filename << " 创建成功"
<< std::endl;
    emptyFile.close();
    std::fstream textFile(filename);
    if (!textFile.is_open()) {
        std::cerr << "Error: File " << filename << " open failed." <<
std::endl;
        exit(FILE_OPEN_ERROR);
    }
```

3.2.2 写入文件功能的实现

3.2.2.1 写入文件功能实现思路

写入文件功能的实现思路如下：

⑴定义一个字符变量 ch，用于存储用户输入的字符；

⑵输出提示信息，告诉用户输入文本内容，并换行；

⑶进入一个循环，使用 std::cin.get()逐字符读取用户输入的内容，直到用户按下回车键（'\n'）结束输入；

⑷在循环中，将读取到的字符 ch 逐个写入到文件 file 中，使用 file.put(ch)实现写入；

⑸输出保存成功的消息，告知用户文本文件的文件名；

⑹清除文件的状态标志（使用 file.clear()），以确保后续操作不受之前的读取操作的影响；

⑺使用 file.seekg(0,std::ios::end)定位文件指针到文件的末尾，然后使用 file.tellg()获取文件的长度（文件尾的位置），将这个长度存储在 fileLen 中；

⑻使用 file.seekg(0,std::ios::beg)将文件指针重新定位到文件的开头，以便后续的文本搜索操作。

3.2.2.2 写入文件功能核心代码

```
    char ch;
    std::cout << std::endl << "请输入文本内容（按回车键结束输入）:" << std::endl << std::endl;
    while ((ch = std::cin.get()) != '\n')
        file.put(ch);
    std::cout << std::endl << ">>> 文本文件 " << filename << " 保存成功" << std::endl;
    file.clear();
    file.seekg(0, std::ios::end);
    fileLen = file.tellg();
    file.seekg(0, std::ios::beg);
```

3.2.3 读取文件功能的实现

3.2.3.1 读取文件功能实现思路

读取文件功能的实现思路如下：

KeywordSearch::getCharFromFile 函数接受一个 std::fstream 文件对象 file 和一个整数 index 作为输入参数，用于读取文件中指定位置的字符。

⑴使用 file.seekg(index, std::ios::beg)将文件指针定位到指定位

置 index 处，其中 std::ios::beg 表示从文件的开头开始定位；

⑵使用 file.get(ch)从文件中读取一个字符，并将其存储在字符变量 ch 中；

⑶使用 if(file.get(ch).good())来检查文件读取是否成功。如果成功，函数返回读取的字符 ch；

⑷如果文件读取失败，表示指定的索引超出文件范围，函数会报告错误信息并调用 exit 函数退出程序，错误码为 INVALID_INDEX_ERROR。

KeywordSearch::outputText 函数接受一个 std::fstream 文件对象 file 作为输入参数，用于输出文件的内容到控制台。

⑴定义一个字符变量 ch，用于存储从文件中读取的字符；

⑵输出一个提示信息，告诉用户要显示的文本文件的内容，并添加换行；

⑶进入一个循环，使用 file.get(ch)来逐个读取文件中的字符。在循环中，使用 std::cout.put(ch)将读取的字符 ch 逐个输出到控制台，从而显示文件的内容。循环会一直执行，直到文件结束；

⑷使用 file.clear()清除文件的状态标志，以确保后续的文件操作不受之前的读取操作的影响；

⑸使用 file.seekg(0,std::ios::beg)将文件指针重新定位到文件的开头，以便后续的文件操作。

3.2.3.2 读取文件功能核心代码

```
char KeywordSearch::getCharFromFile(std::fstream& file, int index)
{
    file.seekg(index, std::ios::beg);
    char ch;
    if (file.get(ch).good())
        return ch;
    else {
        std::cerr << "Error: Index out of range." << std::endl;
        exit(INVALID_INDEX_ERROR);
    }
}

void KeywordSearch::outputText(std::fstream& file)
{
    char ch;
    std::cout << std::endl << ">>> 文本文件 " << filename << " 内容"
<< std::endl << std::endl;
    while (file.get(ch))
        std::cout.put(ch);
    std::cout << std::endl;
```

```
        file.clear();
        file.seekg(0, std::ios::beg);
    }
```

3.3 关键词输入功能的实现

3.3.1 关键词输入功能实现思路

关键词输入功能的实现思路如下：

(1)使用 inputString 函数，该函数用于输入字符串，并会验证输入是否包含空格，并将用户输入的字符串存储在字符数组 keyword 中；

(2)使用 strlen(keyword)函数获取用户输入的关键词的实际长度，并将其存储在整数变量 keywordLen 中。

3.3.2 关键词输入功能核心代码

```
inputString(keyword, "关键词");
keywordLen = static_cast<int>(strlen(keyword));
```

3.4 关键词检索功能的实现

3.4.1 BF(Brute-Force)算法的实现

3.4.1.1 BF(Brute-Force)算法实现思路

BF（Brute-Force）算法的实现思路如下：

(1)初始化一个计数变量 count 为 0，用于记录关键词在文本中出现的次数；

(2)遍历文本中的每一个可能的起始位置 i，从 0 到(fileLen-keywordLen+1)，其中 fileLen 是文本文件的长度，keywordLen 是关键词的长度。这是因为如果剩余的文本长度不足以容纳关键词，就没有必要再搜索；

(3)在每个起始位置 i，初始化一个循环变量 j 为 0，用于迭代关键词的每个字符。进入一个内层循环，从 0 到关键词的长度 keywordLen，比较关键词中的字符和文本中当前位置 i+j 处的字符。这是一个逐字符的比较。如果在内层循环中，存在字符不匹配（即 getCharFromFile(file,i+j)!=keyword[j]），则退出内层循环，表示当前位置不是匹配的起始位置。如果内层循环完成后，j 等于 keywordLen，表示在当前位置 i 开始的子串与关键词匹配，增加 count 计数器；

(4)继续循环，遍历文本中的所有可能起始位置；

(5)循环结束后，返回计数器 count，表示关键词在文本中出现的总次数。

3.4.1.2 BF(Brute-Force)算法核心代码

```
int KeywordSearch::BF_Search(std::fstream& file)
{
    int count = 0;
    for (int i = 0; i < fileLen - keywordLen + 1; i++) {
        int j = 0;
        for (; j < keywordLen; j++)
            if (getCharFromFile(file, i + j) != keyword[j])
                break;
        if (j == keywordLen)
            count++;
    }
    return count;
}
```

3.4.2 KMP(Knuth-Morris-Pratt)算法的实现

3.4.2.1 KMP(Knuth-Morris-Pratt)算法实现思路

KMP（Knuth-Morris-Pratt）算法的实现思路如下：

(1)构建匹配表 next 数组。getNext 函数负责构建部分匹配表 next 数组，函数的实现思路为：

①初始化一个整数 k 为-1，next[0]初始化为-1。使用一个循环遍历关键词中的每个位置 i，从 1 到 keywordLen-1；

②在每个位置 i，进入一个内层循环，不断向前回溯，将 k 的值减小，直到找到满足 keyword[i]==keyword[k+1]或 k 变为-1。如果找到满足条件的 k，则将 k 增加 1，然后将 next[i]设置为 k。否则，next[i]仍然为-1，表示没有部分匹配；

(2)搜索过程：KMP_Search 函数执行关键词搜索，函数的实现思路为：

①初始化计数变量 count 为 0，用于记录关键词在文本中出现的次数；

②动态分配一个整数数组 next 用于存储部分匹配表，以处理 KMP 算法的搜索；

③调用 getNext(next)函数，构建部分匹配表；

④使用两个变量 i 和 k 分别指向文本和关键词的当前位置，初始值都为-1；

⑤遍历文本中的每个字符，从 0 到 fileLen-1，其中 fileLen 是文本文件的长度；

⑥在每个位置 i，使用 while(k >= 0 && getCharFromFile(file,i) != keyword[k+1]) 循环，处理 KMP 算法的匹配。如果条件成立，表示当前字符不匹配，根据部分匹配表 next 回溯，将 k 更新为 next[k]。如果条件不成立，表示当前字符匹配，将 k 增加 1。如果 k 达到 keywordLen-1，表示找到了一个完

整匹配，将计数器 count 增加 1；

(3)返回计数器 count，表示关键词在文本中出现的总次数。

3.4.2.2 KMP(Knuth-Morris-Pratt)算法核心代码

```
void KeywordSearch::getNext(int next[])
{
    int k = -1;
    next[0] = -1;
    for (int i = 1; i < keywordLen; i++) {
        while (k >= 0 && keyword[i] != keyword[k + 1])
            k = next[k];
        if (keyword[i] == keyword[k + 1])
            k++;
        next[i] = k;
    }
}

int KeywordSearch::KMP_Search(std::fstream& file)
{
    int count = 0, k = -1;
    int* next = new(std::nothrow) int[keywordLen];
    if (next == NULL) {
        std::cerr << "Error: Memory allocation failed." << std::endl;
        exit(MEMORY_ALLOCATION_ERROR);
    }
    getNext(next);
    for (int i = 0; i < fileLen; i++) {
        while (k >= 0 && getCharFromFile(file, i) != keyword[k + 1])
            k = next[k];
        if (getCharFromFile(file, i) == keyword[k + 1])
            k++;
        if (k == keywordLen - 1) {
            count++;
            k = next[k];
        }
    }
    delete[] next;
    return count;
}
```

3.5 异常处理功能的实现

3.5.1 动态内存申请失败的异常处理

在进行 KMP 算法中的 next 数组等的动态内存申请时，程序使用 new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL 或 nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1) 向标准错误流 std::cerr 输出一条错误消息"Error: Memory allocation failed."，指出内存分配失败；

(2)调用 exit 函数，返回错误码 MEMORY_ALLOCATION_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

下面是动态内存申请的异常处理的一个代码示例：

```
int* next = new(std::nothrow) int[keywordLen];
if (next == NULL) {
    std::cerr << "Error: Memory allocation failed." << std::endl;
    exit(MEMORY_ALLOCATION_ERROR);
}
```

3.5.2 输入非法的异常处理

3.5.2.1 字符串（文本文件名与关键词）输入非法的异常处理

程序通过调用 inputString 函数输入文本文件名与关键词的字符串。inputString 函数用于获取用户输入的字符串，同时检查输入的字符串是否为空和是否含有空格，函数的代码如下：

```
void inputString(char* str, const char* strName)
{
    while (true) {
        std::cout << std::endl << "请输入" << strName << "（不含空格的不超过 " << MAX_LENGTH << " 个英文字符或 " << MAX_LENGTH / 2 << " 个汉字字符组成的字符串，超出部分将被截断，如果含有空格）: ";
        std::cin.get(str, MAX_LENGTH + 1, '\n');
        std::cin.clear();
        std::cin.ignore(INT_MAX, '\n');
        str[MAX_LENGTH] = '\0';
        if (*str == '\0')
            std::cout << std::endl << ">>> " << strName << "为空，请重新输入！" << std::endl;
        else if (hasSpace(str))
            std::cout << std::endl << ">>> " << strName << "含有空格，请重新输入！" << std::endl;
        else
```

```
            break;
        }
    }
```

inputString 函数对输入非法的情况进行了处理，代码具体执行逻辑如下：

⑴获取用户输入：函数进入一个无限循环 while(true)，使用 cin.get 获取用户的输入，最多获取 MAX_LENGTH+1 个字符，直到用户按下回车键（'\n'）为止；

⑵清理输入缓冲：函数使用 cin.clear()清除输入流的错误标志，以便后续的输入操作；

⑶字符串截断：如果用户输入的字符串超过了 MAX_LENGTH，函数会将超出部分截断；

⑷检查输入是否为空：函数检查用户输入是否为空，即是否只包含回车符。如果用户只输入回车符，表示输入为空，将输出错误消息并要求用户重新输入；

⑸检查是否包含空格：函数使用 hasSpace 函数检查输入的字符串是否包含空格。如果包含空格，将输出错误消息并要求用户重新输入；

⑹合法输入退出循环：如果用户输入的字符串满足所有要求，函数会退出循环，否则将一直要求用户重新输入，直到满足条件为止。

3.5.2.2 字符串模式匹配算法选择输入非法的异常处理

字符串模式匹配算法选择输入非法的异常处理通过如下代码实现：

```
int selectOptn(void)
{
    std::cout << std::endl << ">>> 字符串模式匹配算法: [1]BF(Brute-Force)算法 [2]KMP(Knuth-Morris-Pratt)算法" << std::endl;
    std::cout << std::endl << "请选择字符串模式匹配算法: ";
    char optn;
    while (true) {
        optn = _getch();
        if (optn == 0 || optn == -32)
            optn = _getch();
        else if (optn >= '1' && optn <= '2') {
            std::cout << "[" << optn << "]" << std::endl << std::endl;
            return optn - '0';
        }
    }
}
```

这段代码会一直等待用户输入，只有当用户输入有效的数字字符（'1'或'2'）时，才会返回该数字的整数值，表示用户选择的操作选项。如果用户输入无效字符，循环会继续等待用户提供有效的输入。这段代码的具体执行逻辑如下：

(1)显示提示信息，其中包含选项[1]与[2]；

(2)进入一个无限循环，以等待用户输入；

(3)用户输入的字符会被_getch()函数获取。这个函数通常用于在控制台应用程序中获取单个字符而不显示在屏幕上。用户的输入存储在变量 optn 中；

(4)代码检查用户输入是否是数字字符（'1'或'2'）或特殊的控制字符。如果用户按下非数字字符或特殊控制字符，它将不执行下面的操作；

(5)如果用户输入是数字字符（'1'或'2'），它会在屏幕上显示用户输入的数字字符，并返回对应的整数值；

(6)如果用户输入的不是数字字符（'1'或'2'），代码将保持在循环中，继续等待有效的输入，这确保了只有在用户输入正确的选项时才会退出循环。

4 项目测试

4.1 文件操作功能测试

4.1.1 创建文件功能测试

在创建文件过程中，要求输入文件名，要求为：不含空格的不超过 64 个英文字符或 32 个汉字字符组成的字符串，超出部分将被截断。程序会对输入非法（字符串为空和字符串中含有空格）的情况做出提示并重新输入。

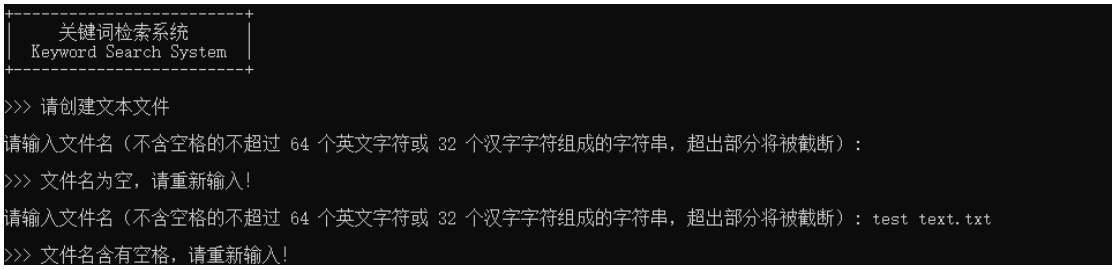

图 4.1.1.1 创建文件功能测试（文件名输入非法）

当输入合法时，程序会进行文件创建操作。当文件已存在时，则输出文件已存在的提示信息，报错并退出。

图 4.1.1.2 创建文件功能测试（文件已存在）

当文件不存在时，则创建新文件并输出文件创建成功的提示信息。

图 4.1.1.3 创建文件功能测试（文件不存在）

4.1.2 写入文件功能测试

输入文本内容并按回车结束输入，则输出文件保存成功的提示信息。

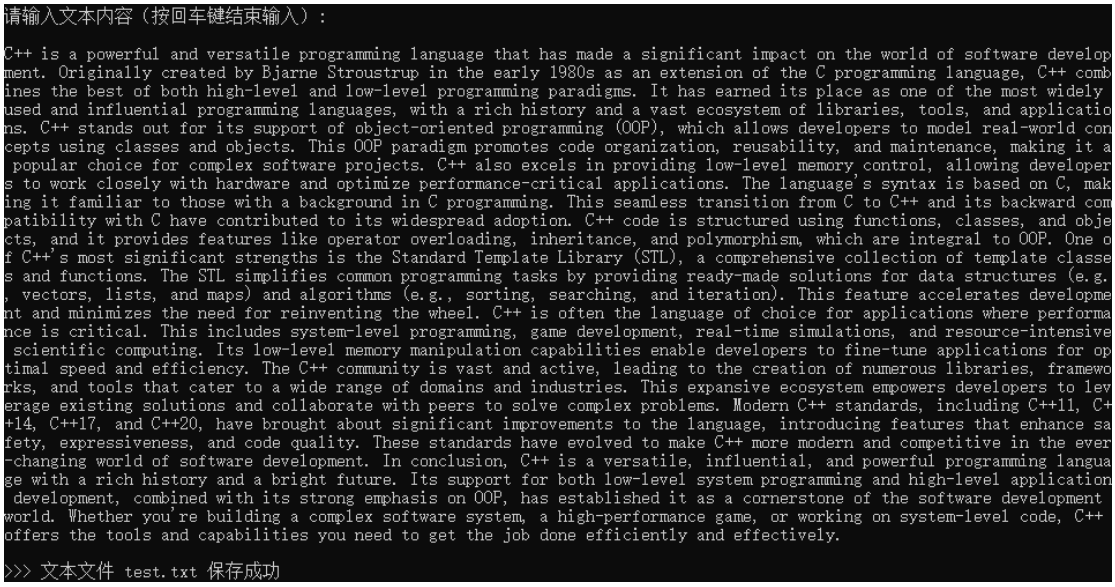

图 4.1.2.1 写入文件功能测试（控制台输出）

这时在可执行文件的目录下打开新创建的文本文件（在本测试中文本文件名为 test.txt），可以看到在控制台中输入的文本内容已经成功保存至文本文件中。

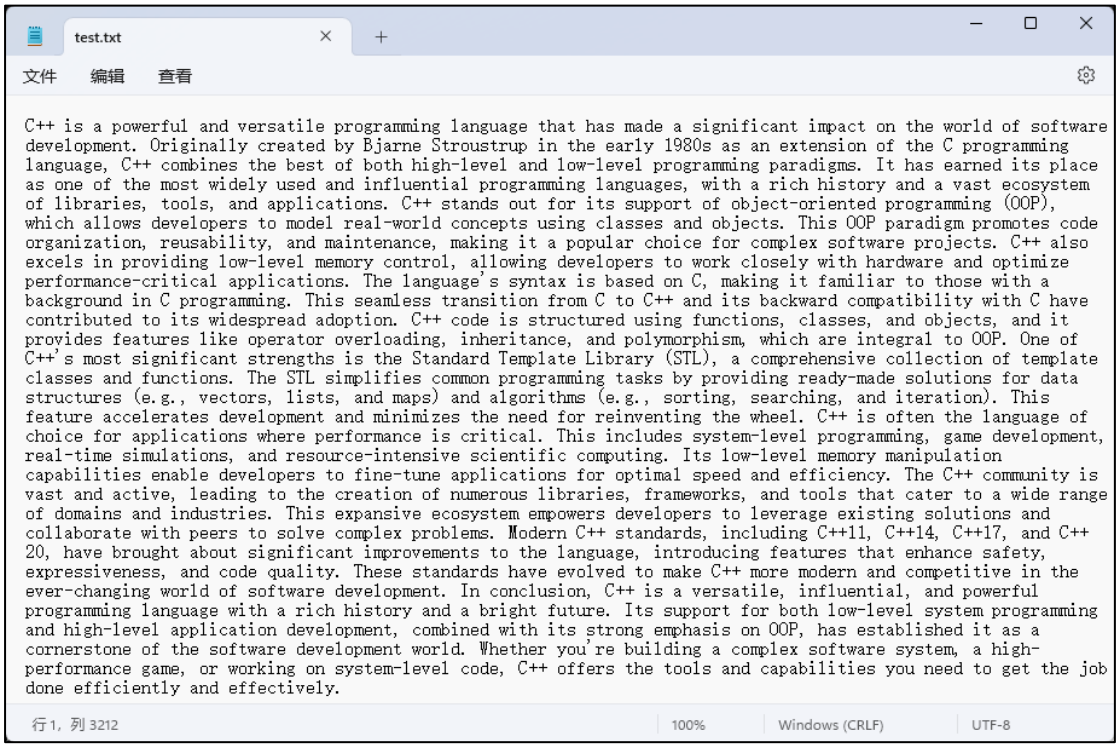

图 4.1.2.2 写入文件功能测试（文本文件）

4.1.3 读取文件功能测试

在关键词输入完成之后，程序会打开新创建的文本文件，并在控制台中输出文本文件中的所有内容。

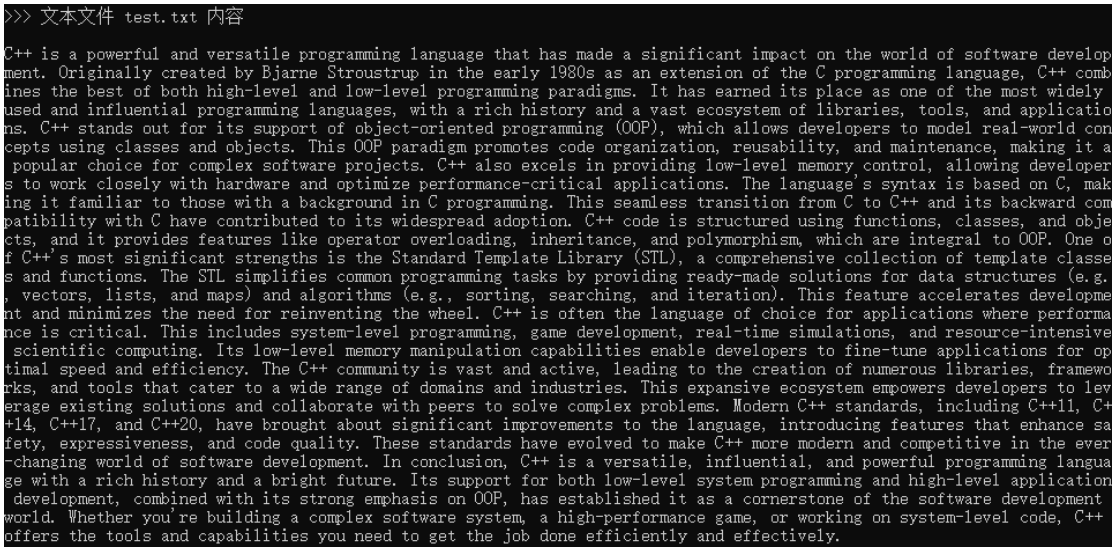

图 4.1.3.1 读取文件功能测试

4.2 关键词输入功能测试

在关键词检索过程中，要求输入关键词，要求为：不含空格的不超过 64 个英文字符或 32 个汉字字符组成的字符串，超出部分将被截断。程序会对输入非法（字符串为空和字符串中含有空格）的情况做出提示并重新输入。

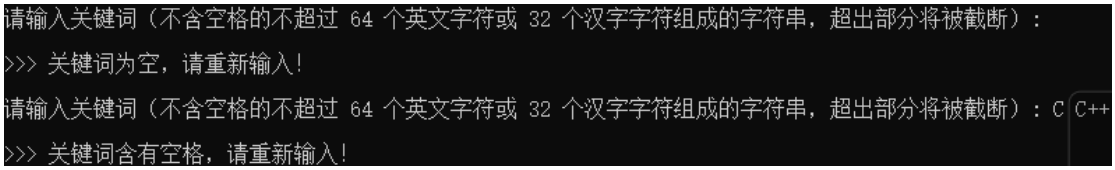

图 4.2.1 关键词输入功能测试

4.3 关键词检索功能测试

关键词检索功能测试在写入文件功能测试的基础上进行，测试关键词为“code”，使用 Microsoft Word 2021 的检索功能对文本内容进行测试，可以得知关键词“code”在测试文本内容中出现了 4 次。

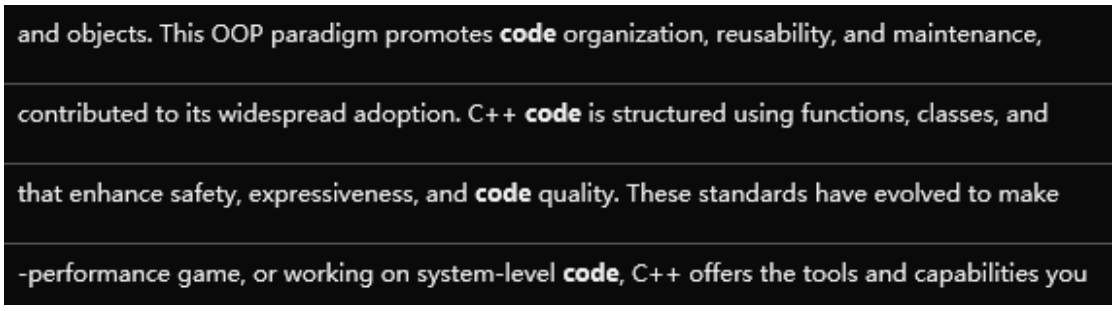

图 4.3.1 关键词检索功能测试

4.3.1 BF(Brute-Force)算法测试

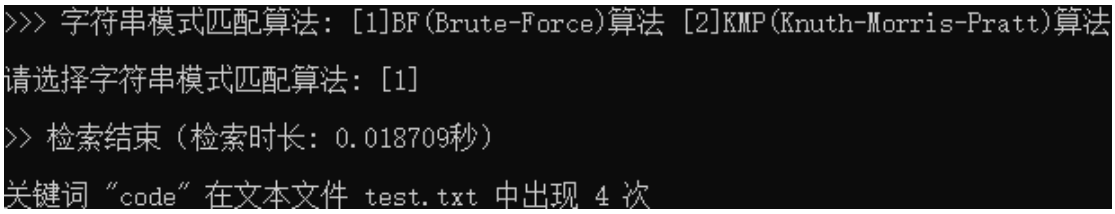

图 4.3.1.1 BF(Brute-Force)算法测试

4.3.2 KMP(Knuth-Morris-Pratt)算法测试

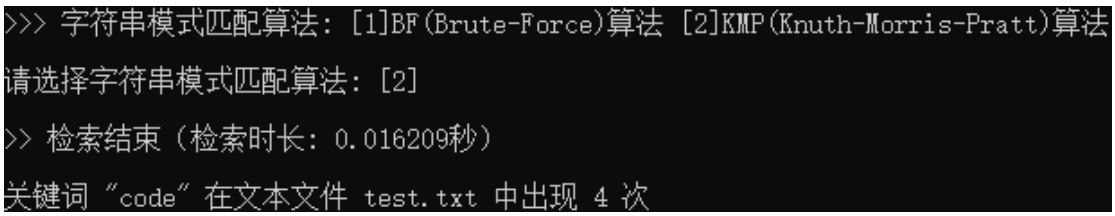

图 4.3.2.1 KMP(Knuth-Morris-Pratt)算法测试

4.4 退出程序功能测试

为避免直接运行可执行文件在输入完成后会发生闪退的情况，本程序使用如下代码，创建了一个无限循环，等待用户按下回车键，以便退出程序。避免结果输出结束后程序迅速退出的情况。

```
/* Wait for enter to quit */
std::cout << "Press Enter to Quit" << std::endl;
while (_getch() != '\r')
    continue;
```

5 集成开发环境与编译运行环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：Microsoft Visual Studio 2022 (Release 模式)

Windows 编译运行环境：本项目适用于 x86 架构和 x64 架构

Linux 系统：CentOS 7 x64

Linux 编译命令：

g++ '/root/桌面/Share_Folder/keyword_search_system.cpp' -std=c++11 -o '/root/桌面/Share_Folder/keyword_search_system' -lncurses

Linux 运行命令：

'/root/桌面/Share_Folder/keyword_search_system'

图 5.1 Linux 环境程序运行示例

本项目使用条件编译解决 Windows 系统和 Linux 系统编译环境的差异，示例代码如下。

```
#ifdef _WIN32
#include <conio.h>
#elif __linux__
#include <ncurses.h>
#endif
```