文本编辑器大作业报告

1200012708 / 蒋捷 1200012808 / 赵万荣1200012823 / 周昊宇 1200012835 / 邢曜鹏

# 程序框架

1. 图形交互：

负责显示。给用户提供图形界面，接受并处理用户的输入，将输入转换为文本编辑器核心所接受的形式，对数据进行处理。

1. 文本编辑器核心：

负责对数据进行操作。接收图形界面发送的命令，对文本进行增删查改，写入读取等操作，并将内容输出到图形界面交互所提供的输出缓冲区中，交予其进行输出。

1. 文件交互：

将文本内容写入外存，或从中读取文本内容

1. 模式匹配：

对给定的模式串在文本内容中进行查找，并将查找结果返回图形交互界面进行输出

# 数据结构

class CEditorCore 主要成员变量及成员函数

{

private:

list<string> data; // 链表形式存储文本内容，每一行为链表的每一个元素

string clipboard; // 内置剪切板，用于在拖放操作时存储临时数据

string fileName; // 打开文件时的文件名

public:

string\* outputBuffer; // 输出缓冲区指针，由图形交互界面提供其地址

bool loadFromFile(string); // 从指定文件读取文本

bool saveToFile(string); // 写入到指定文件

bool insert(const CPosition &, const string &); // 插入一个字符串

string remove(const CPosition &, const CPosition &); // 删除段落，指定开始结束为止

string getSelection(const CPosition &, const CPosition &) const; // 获得选中的字符串

bool replace(const string &, const CPosition &, const CPosition &); // 替换指定位置的字符串

int updateBuffer(); // 更新输出缓冲区，返回行最大宽度

};

# 主要算法

## KMP算法

模板串为W，待匹配串为T。

首先要对W串进行预处理，即求出next数组（最大前缀长度）具体实现可以用递归实现，代码如下：

       while ( T[j+1] != '\0' )

       {

              if (k == -1 || T[j] == T[k])

              {

                     ++j; ++k;

                     if (T[j]!=T[k])

                            next[j] = k;

                     else

                            next[j] = next[k];

              }// if

              else

                     k = next[k];

       }// while

之后就是具体的模式匹配过程

假设在模式匹配的进程中，执行T[i]和W[j]的匹配检查。若T[i]=W[j]，则继续检查T[i+1]和W[j+1]是否匹配。若T[i]<>W[j]，则分成两种情况：若j=1，则模式串右移一位，检查T[i+1]和W[1]是否匹配；若1<j<=m，则模式串右移j-next(j）位，检查T[i]和W[next(j)]是否匹配。重复此过程直到j=m或i=n结束。

代码如下：

 int index=0,i=0,j=0;

      while(Text[i]!='\0'  && Pattern[j]!='\0' )

       {

              if(Text[i]== Pattern[j])

              {

                     ++i;// 继续比较后继字符

                     ++j;

              }

              else

              {

                     index += j-next[j];

                     if(next[j]!=-1)

                            j=next[j];// 模式串向右移动

                     else

                     {

                            j=0;

                            ++i;

                     }

              }

       }//while

       if(Pattern[j]=='\0')

              return index;

## BM算法

模板串为p，长度lp；待匹配串为t，长度lt

则算法可表示为

while (i <= lt - lp)

{

j = lp - 1;

while (j > -1 && t[i + j] == p[j]) // 从后向前匹配，直到找到匹配失败的字符

--j;

if (j == -1)

break; // 匹配成功

else

i += shift; // 移动模板串的位置

}

从后向前匹配，利用两个并行算法——坏字符与好后缀，增大每次匹配的移位量（shift），达到减少匹配次数以加快查找速度的目的。

坏字符：定义辅助数组bc[]，预处理时记录每个字符在最右边出现的位置，若在匹配过程中发现某个字符不相等，直接将模板串中最右侧的相应字符与其对齐

Shiftbc = j - bc[t[i + j]]；

好后缀：定义辅助数组gs[]，预处理时记录模板串中与某后缀完全相同子串的位置，若在匹配过程中匹配失败，则直接将模板串移动到此位置

Shiftgs = j + 1 - gs[j + 1];

最后实际的位移 i+=max(shiftbc, shiftgc)

# 实现功能

## 文件的新建，打开，保存；

## 图形界面与鼠标、键盘操作；

## 完全从底层编写的文本框、光标及其动画效果；

## 在任意位置添加、删除字符串，剪贴板操作；

## 字符串查找与替换（KMP算法、BM算法），全部替换；

## 命令行/鼠标拖放打开文件，选中文本段拖放移动/复制；

## 更改字体颜色、字号、行高等；

## 显示行号，括号匹配；

## 提供了一个方便的表达式计算工具QuickParse™

# 调试报告

调试过程是艰辛而枯燥的……由于前期一直没有结合所有模块，因而在最后结合时发现了大量的问题，特别是C#与C++之间的跨语言类型转换包装之类的奇怪问题，以及前台显示与后台的核心之间的交流。

最初的问题是对于整个文本内容，在用P/Invoke调用Dll时，C#动态数组、动态字符串与C++链表、字符串的对应问题，后来的解决方式是通过先调用C++模块获得每行宽度，在C#中分配空间，而后调用更新缓冲区函数，利用Marshal将C#中的数组指针传入并在函数中放入整个文本。后来的问题是MDA（托管调试助手）经常抛出致命异常，即程序内存被破坏，后来通过给数组分配更大的空间、改变调用逻辑来解决。这些调错经历让我深刻明白了跨语言编程的困难，在以后的编程中，除非接口很简单，否则不要轻易跨语言编程。

之后，在核心模块中遇到了大量的细节问题，比如误把string的最后一位当作-1、if表达式==写成=、循环中的迭代器变量没有自增等等。

另外，后期在支持通配符的KMP算法中发现bug，因而暂时用BM算法代替，修复bug后增加了算法选择功能，也算是因祸得福吧……

1. 经验与体会

## **邢曜鹏：**

我负责的是文件读入和文件写入的函数（ bool loadFromFile(string) bool saveToFile(); bool saveToFile(string); ) ，以及对CPosition 类的改进 ，括号进行匹配的算法。( CPosition match(CPosition); ) 以及参与KMP实现的查找替换算法的设计。

### 收获：

通过编写 文件流的读写，让我对文件流的处理又有了更深的认识，并学会利用本学期学习的数据结构知识，将文本存储在list<string>类型中便于检索以及其他处理方式。

通过编写 括号匹配函数，我进一步锻炼了算法与文件流的契合能力，将算法得以实际应用。

虽然我不是KMP实现的查找替换的程序编写者，我也参与了设计讨论，并自己编写相似的程序得以练习，将课堂学习的算法得以深刻掌握。

第一次团队合作让我自己积累了不收小组合作的经验，包括如何写好接口，如何更好的交流，统筹分工。

### 反思：

觉得自己还应该承担小组更多的责任，我将在下次大作业中主动承担更多责任，同时发现自己在界面的实现上有很多知识的欠缺，现在我正自自助学习当中。程序算法方面我也在自己思考有没有改进的方案，不会因为交完作业就抛之脑后。

## **赵万荣：**

我的工作就是写好软件的一个重要功能：查找替换中的文字匹配算法 CPosition find(const string &);

### 收获：

我充分利用了自己在数算课程上学到的KMP算法，将这种思想完全掌握在自己手里，并且有了自己的创新，即模糊查找，搜索的\*可以代表任意长度的字符。我先将\*字符串两侧分割，然后用递归算法进行匹配，输出收尾两个位置参数。经过一系列的调试我的想法终于实现了。同时我还收获了很多团队合作中交流讨论以及分工的经验。

### 反思：

算法的时间复杂度，程序的写作风格（同伴阅读是否有困难）都是我以后要改进的。我还在继续思考更有效的算法。

## **蒋捷：**

这次大作业中自己的任务主要是设计整体框架，并完成了一些对文本进行操作的函数，因为某些原因又实现了字符串匹配的BM算法。在这个过程感受到了编写代码前接口定义的重要性以及面向对象的优点。自己的工作大部分都是STL的处理，所以编码的难度并不大，但是出现了很多细节问题（如忘记了++i，做减法的时候多减了个1之类的），写代码时所有的函数都设置了一些条件检查，并返回了特定的值，所以在调试的时候省去了很多麻烦。最后将代码合并debug的时候也学会了不少调试的方法（VS确实是个很好用的东西）。另外在编写BM算法的时候，虽然编写比较麻烦（这个是在实验班的作业里面写的，当时写了一天…不过基本上是个人原因啦），但是深深的感受到了其思想的优美性。

整个过程中也有很多不足的地方。开始制定接口后来发现有很多令人不满意的地方，有一些因为其他代码与之都有关不方便更改，比如在输出缓冲区的设置上，为了封装性，core传递给图形界面的是“翻译”过的整个文本区，而由图形部分选取部分区域输出，而在后来考虑的时候想到了效率更高的方法，最后也就只能这样了。另外在时间的问题上，代码合并到一起的时间过晚，导致有不少bug不能解决。另外在分工方面，编写图形界面的同学任务有些重，开工的时候分工还需要注意。再有，在合作方面，因为bug可能会出现在不同人编写的代码中，如何修改以及之后怎么将不同人修改后的代码再次合并都是一些需要更加好好考虑的问题。下次大作业的时候会注意这些问题。

## **周昊宇：**

本次大作业我主要负责的是文本编辑器的图形界面和DLL里的导出函数。图形界面我选用C#编写，不过虽然它的开发效率很高，但是与C++编写的DLL互相操作却很是艰难。通过大量的资料查询，我终于将C#与C++的DLL之间的接口调试成功。MDA在调试这些互操作特性中起到了很重要的作用，它报出了许多关于内存分配的错误。让我能很快发现问题所在。为了深入了解Windows控件的机制，我从底层编写了一个文本框，遇到了极大的困难，不过通过MSDN等社区的帮助，我的文本框逐渐成型。虽然还有一些功能没有实现，不过已经基本实现了Windows基本文本框的功能，而且能显示行号、支持拖放。在编码过程中，我还顺手用了C#的实时编译功能做了一个表达式解析器，可以解析C#支持的表达式。

在此次工程完成之后，我想我应该继续学习C#的底层学习，同时由于本次作业所实现部分与算法关系不大，因而以后要多进行算法方面的练习。