**数据结构程序设计说明文档**

**————VigorEditor**

1. **设计任务的描述**

1. 编写目的:

本地设计任务的目的是帮助学生巩固课堂及书本知识，提升理论联系实际的能力

，提高分析和解决实际问题的能力，同时训练软件设计、开发以及书写软件文档的能力。

2. 任务描述

本次数据结构程序设计大作业的设计任务为实现一款全屏幕编辑软件MiniWord,基本要求如下：

该编辑器需要可分为两个操作状态：分别是文件状态和编辑状态。在文件状态下，该编辑器可以对文件进行打开、保存、另存为、新建等操作；在编辑状态下，可以对正文文件(.txt)进行输入、修改、查找、替换等操作。因此，该编辑器要求具备文件处理功能、文本窗口编辑功能以及鼠标和一些热键的功能。

1. **功能需求说明及分析**

**1. 文件处理功能**

改类别的功能使得使用者可以对文件进行操作，包括新建、打开、保存、退出等等。

新建文件（New）：若编辑区有未保存的编辑内容，询问是否保存后再清空编辑区。

打开文件（Open）：要求用户输入文件名，该文件存在则打开载入编辑区，否则提示为“新文件”。

保存文件（Save）：提示用户当前文件名，用户可以重置文件名，确认后将当前编辑的文件写入磁盘。

退出系统（Quit）：退出前检查是否有未保存的编辑内容，若需要则执行Save操作后再退出。

**2. 文本窗口编辑功能**

该类别的功能主要在于对编辑器内部的文本进行编辑操作

插入字符：定位光标，在光标处之后插入字符，每插入一个字符后光标定位在新插入的字符之后。

插入行：插入字符为回车键时，光标后内容为新行

删除字符：定位光标，“Delete”键向后删除字符，“Backspace”键向前删除字符。

删除行：光标位于行首，输入“Backspace”键

查找字符/串：提示用户输入要查找的字符串，从当前光标处向后定位，找到时光标置于最后一个字符之后。

替换字符/串：提示用户输入原字符串和新字符串，从当前光标处向后定位，找到时光标置于首字符之前，由用户对是否替换进行确认。

块操作（选择）：定位块首、块尾，块拷贝、块删除。

复制：可以复制选中部分的文字/字符串

剪切：可以剪切选中部分的文字/字符串

粘贴：可以将系统剪切板中的文字/字符串粘贴到编辑器中

**3\* . 鼠标及其他编辑热键功能（选做）**

鼠标：实现鼠标点击定位光标，鼠标拖动实时选择选中区域；

Home:光标跳转到行头；

End:光标跳转到行末；

Ctrl+Home/End，光标跳转至篇首篇末

PageUp: 编辑器页面向上跳一页

PageDown:编辑器页面向下跳一页

Shift选中：按住shift执行键盘选中。

选中后操作：选中后可以执行改变内容操作替换原来选中部分，或者使用任何光标移动操作，人性化游走光标。

滚动条：当文字长度超出界面宽度或者长度的时候，提供滚动条拖动查看文件内容

全选：选中当前编辑器中所有文本内容

字数统计&行数统计：统计当前编辑器中的行数和字数

撤销：可以撤销之前的操作

恢复撤销：可以恢复之前的撤销操作

拖拽快捷打开：将待打开的文本拖拽到我们编辑器的图标上，可以实现快捷打开

打开方式：可以在打开方式中选择使用VigorWord打开

1. **总体方案设计说明**

3.1. 软件开发环境

操作系统：Windows 10操作系统

IDE: Microsoft Visual Studio 2017

代码托管： Coding

3.2. 总体结构

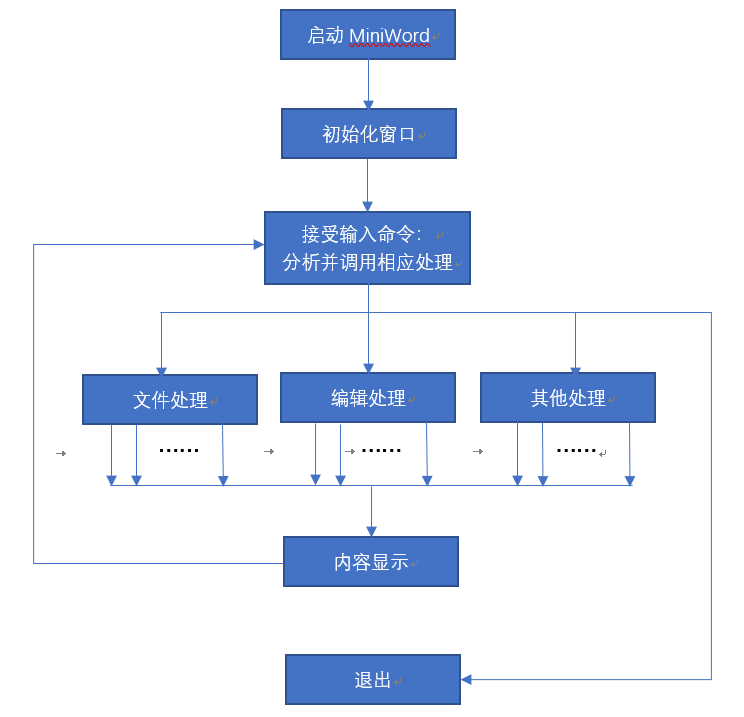


图 3.2

3.3. 模块划分

本编辑器主要分为三个模块，分别是：数据结构模块、显示模块和用户操作模块

3.3.1数据结构模块

数据结构模块主要是定义了编辑器所用到的数据结构、相关的类以及其包含的属性和方法，其代码主要实现在subeditor.cpp和subeditor.h两个文件中。我们的数据结构模块主要分为如下几个子模块：

gap buffer: 用于实现单行Line类；

双向链表: 用于实现整个文章的Article类；

栈： 撤销栈与恢复栈，用于实现撤销功能；

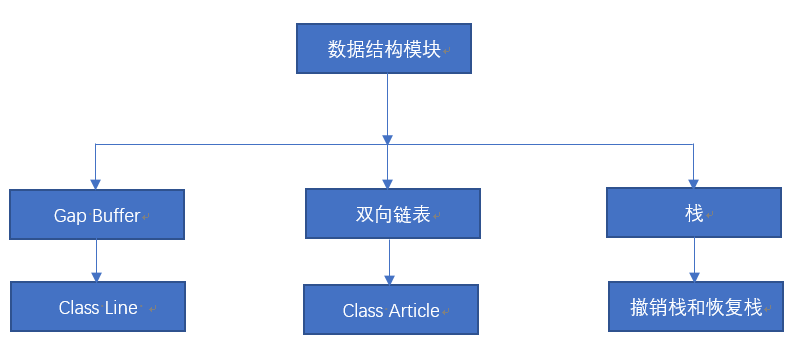


图 3.3.1

3.3.2 显示模块

显示模块主要功能是将内存中数据结构里存储的数据按照我们的期望显示到窗口上，其代码主要实现在MiniWord.cpp, MiniWord.h, Resource.h等几个文件中。我们的显示模块可以分为如下几个子模块：

光标显示：负责光标的显示功能

文字显示：负责文字的显示功能

滚动条显示：负责滚动条的显示功能

菜单显示：负责菜单的显示功能

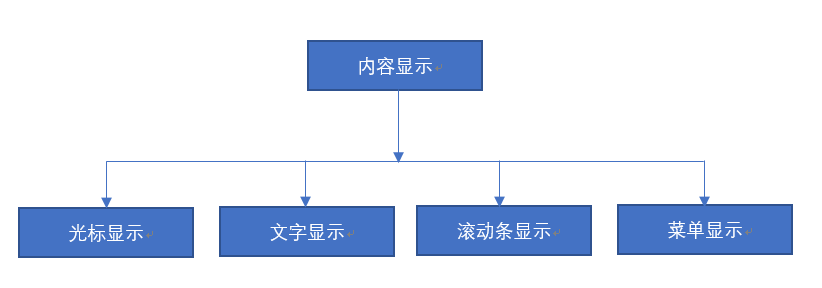


图 3.3.2

3.3.3 用户操作模块

本模块的主要功能是接受信号，根据分析信号的不同来处理用户对编辑器的各种操作，其代码主要实现在MiniWord.cpp, MiniWord.h, Caret.cpp, Undo.cpp等几个文件中。我们的用户操作模块主要分为如下几个子模块：

输入操作处理： 根据用户的输入改变内存中的数据；

文件操作处理： 新建、打开、保存等文件操作；

块操作处理：块拷贝、块粘贴、块删除等；

撤销与恢复处理： 处理撤销与恢复操作；

鼠标操作处理： 鼠标单击、双击、右键等操作；

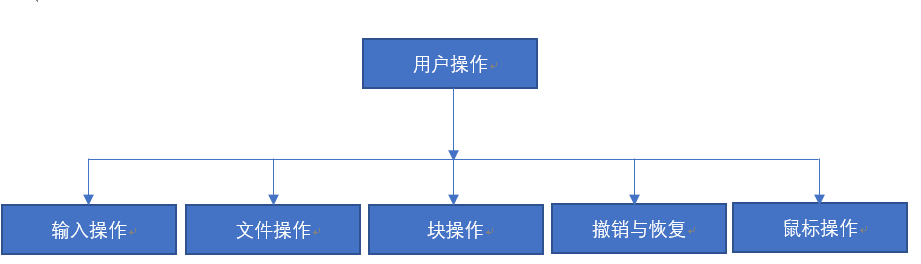


图 3.3.3

**四、 数据结构说明(附数据字典)**

4.1数据结构说明

4.1.1综述：

整个文章用一个Article的双向链表来存放。双向链表的每个结点是一个行line。行line主要采用基于顺序表的Gapbuffe的数据结构来存储，实现了O(1)实现复杂度内的增删移改的最终效率。

同时使用两个栈来存放执行过的操作，使程序能够无限次的撤销和尽可能多的恢复撤销。同时对于替换性质的操作，只需一次撤销操作即可，使记事本更加人性化。

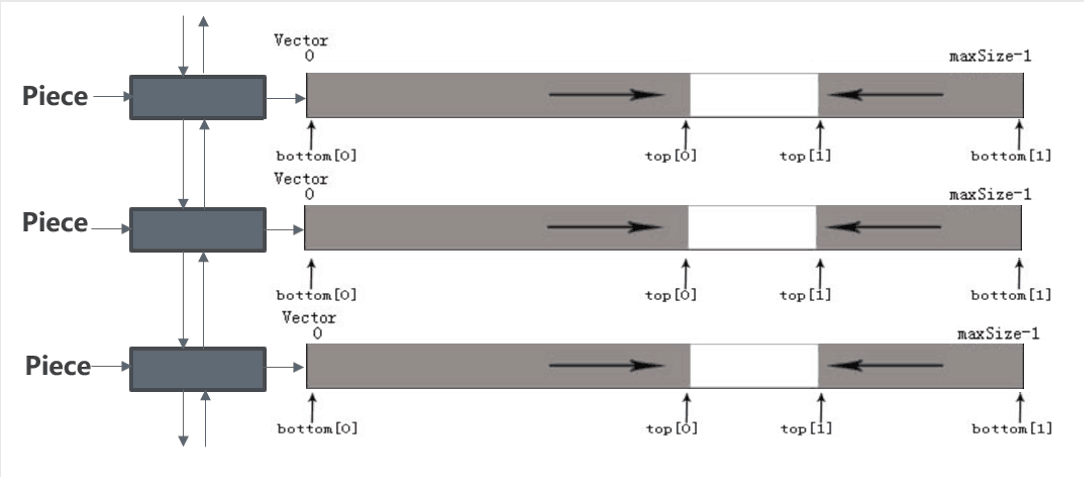
数据结构总体写在subeditor中。开发宗旨有二：1，确保可靠高效地操纵数据；2，尽可能使操作层与显示层方便开发。

4.1.2 Gap Buffer

4.1.2.1概述：

每行采用一个类似双栈的数据结构—Gap Buffer（Emacs、）来存储字符，将字符放在整个数组的两端，从而能够每次以O（1）的时间复杂度来高效插入删除更改字符。

GapBuffer:

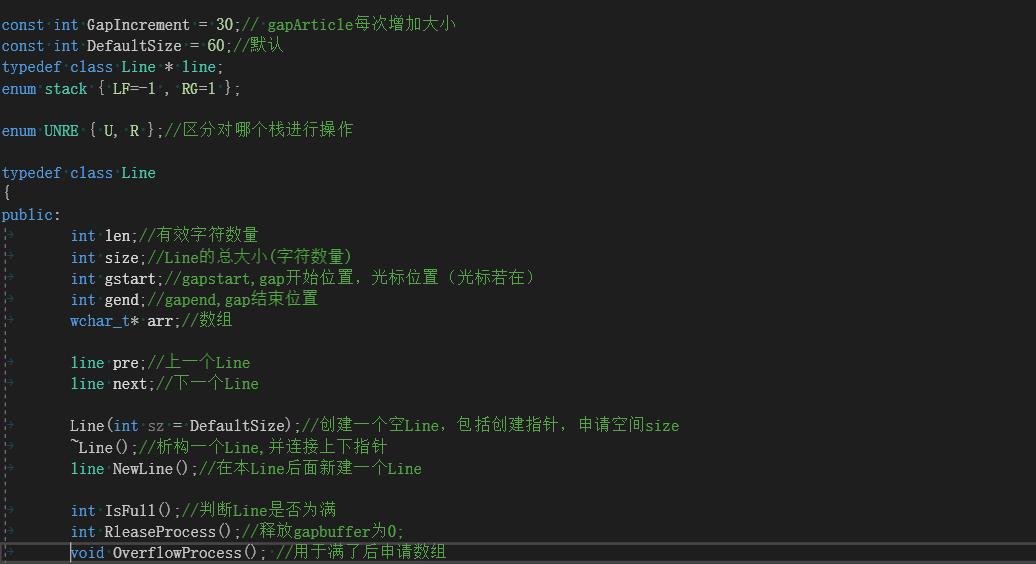


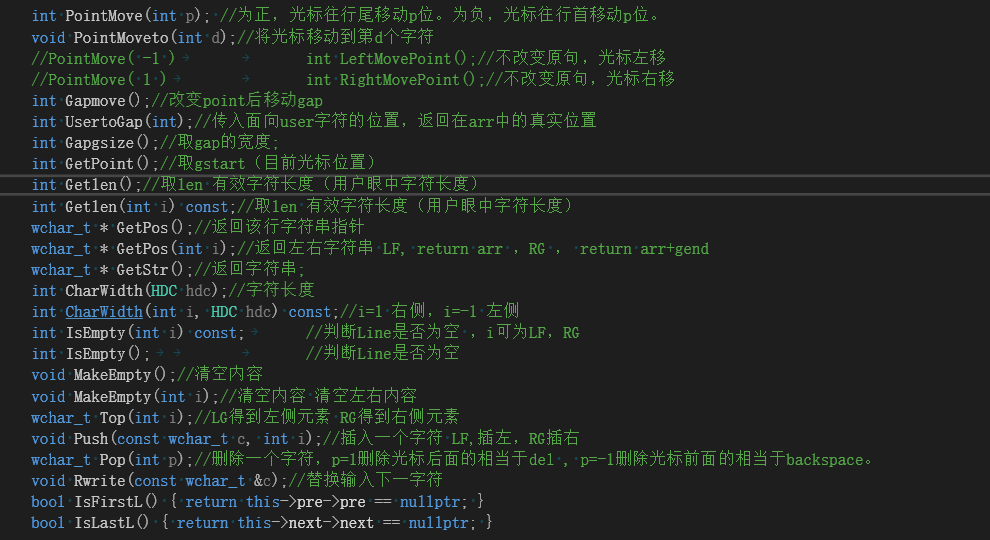
灰色部分即为我们的Gapbuffer，类似于双栈的数据结构。首先是一个数组默认大小为60，数据由光标一分为二，左侧存在数组头，右侧存在数组尾，中间为gap。并由两个索引gstart（图中的top[0]）和gend（图中top[1]）来标注左右栈顶。

Gapbuffer的巨大优势在于它的高效性。常见的修改数据操作增删移改都是O（1）的时间复杂度，并且不会频繁大量移动数据：插入时，从左侧入栈，gstart右移。删除时，左删则gstart--，右删则gend++。移动光标，例如右移则将右侧第一个拷贝至gstart位置，然后gstart++，gend++。

如果gstart=gend，则栈满，再插入则需使用OverFlowProcess()函数来申请更大空间。每次固定增大30。虽然时间复杂度为O（n），但是跟所带来的增删移改的O（1）相比，无疑是值得的。

4.1.2.2具体实现：





int len;//有效字符数量

int size;//Line的总大小(字符数量)

int gstart;//gapstart,gap开始位置，光标位置（光标若在）

int gend;//gapend,gap结束位置

wchar\_t\* arr;//数组

line pre;//上一个Line

line next;//下一个Line

Line(int sz = DefaultSize);//创建一个空Line，包括创建指针，申请空间size

~Line();//析构一个Line,并连接上下指针

wchar\_t Top(int i);//LG得到左侧元素 RG得到右侧元素 O（1）

void Push(const wchar\_t c, int i);//插入一个字符 LF,插左，RG插右 O（1）

wchar\_t Pop(int p);//删除一个字符，p=1删除光标后面的相当于del , p=-1删除光标前面的相当于backspace。O（1）

line NewLine();//在本Line后面新建一个Line，便于对行进行操作 O（1）

int IsEmpty(int i) const; //判断Line是否为空 ，i可为LF，RG O（1）

int IsEmpty(); //判断Line是否为空 O（1）

int IsFull();//判断Line是否为满 O（1）

int RleaseProcess();//释放数组为0;

void OverflowProcess(); //用于满了后申请数组，时间复杂度为O（n），插入操作O（1）所节省下的大量时间显然使得这点时间开销更值得。

void PointMove(int p); //为正，光标往行尾移动p位。为负，光标往行首移动p位。时间复杂度为O（p），取决于移动的位数。如：

//PointMove( -1 ) //不改变原句，光标左移

//PointMove( 1 ) //不改变原句，光标右移

void PointMoveto(int d);//将光标移动到第d个字符,在上一个函数基础上进行的封装,多用在光标移动操作中。

int Gapmove();//改变point后移动gap。将数组右侧数据移动回数组左侧合并，通常用于光标离开本行时操作。O（x）x为光标后字符数

int UsertoGap(int);//传入面向user字符的位置，返回在arr中的真实位置。大大简化开发过程中思考双栈数组下标的问题。

int Gapgsize();//取gap的宽度; O（1）

int GetPoint();//取gstart（目前光标位置）O（1）

int Getlen();//取len 有效字符长度（用户眼中字符长度）O（1）

int Getlen(int i) const;//取len 有效字符长度（用户眼中字符长度）O（1）

int CharWidth(HDC hdc);//字符长度 O（n）

int CharWidth(int i, HDC hdc) const;//i=1 右侧，i=-1 左侧 O（n）

wchar\_t \* GetPos();//返回该行字符串指针 O（1）；

wchar\_t \* GetPos(int i);//返回左右字符串 LF, return arr ，RG ， return arr+gend O（1）

wchar\_t \* GetStr();//返回字符串;O（n）

void MakeEmpty();//清空内容

void MakeEmpty(int i);//清空内容 清空左右内容 O（1）

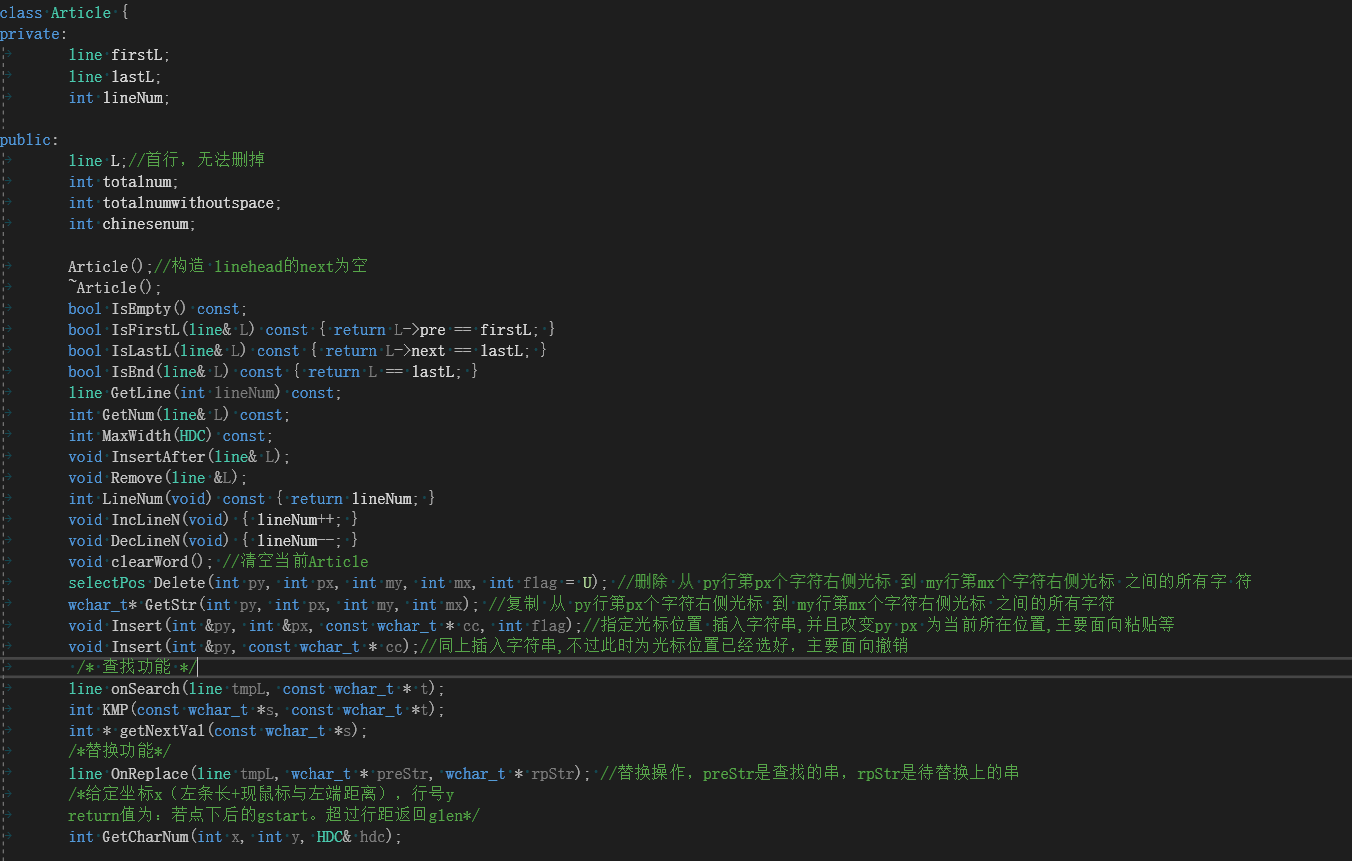
注：很多函数重载，不带参数的针对全行，带参数的针对左侧或右侧，大大方便了开发

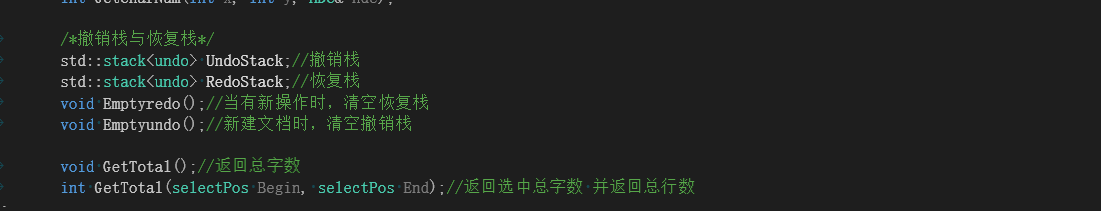
4.1.3双向链表

4.1.3.1综述：

采用规定的数据结构：双向链表。类名:Article。每个结点代表一个行。其中有两个哑行分别表示表头和表尾用来判断是否到达顶端和底端，还有一个永存的第一行L无法删掉。很多跨段操作，采用Article的成员函数。

4.1.3.2具体实现:





private:

line firstL; //链表头的哑行，用来指示表头

line lastL; //链表尾的哑行，用来指示表尾。

int lineNum; //总行数

public:

line L;//首行，无法删掉

int totalnum; //（选中）总字数（计空格）

int totalnumwithoutspace; //（选中）总字数（不计空格）

int chinesenum; //（选中）总汉字数

Article();//构造函数，申请空间，设置好基本的链表关系

~Article();//释放空间

line GetLine(int lineNum) const; //参数为行号，其中第一行行号为0。采用迭代方式，返回该行的行指针。O（n）

int GetNum(line& L) const; //参数为行指针，采用迭代方式，返回其行号。O(n)

void Remove(line &L); //移除行L，并行数-1；

void IncLineN(void) { lineNum++; }//行数+1

void DecLineN(void) { lineNum--; }//行数-1；

void clearWord(); //清空当前Article，并清空恢复栈和撤销栈。O（n）

selectPos Delete(int py, int px, int my, int mx, int flag = U); //删除 从 py行第px个字符右侧光标 到 my行第mx个字符右侧光标 之间的所有字符，并将操作信息放入撤销栈中。flag用来标注该操作放入撤销栈还是恢复栈。O（n）

wchar\_t\* GetStr(int py, int px, int my, int mx); //复制 从 py行第px个字符右侧光标 到 my行第mx个字符右侧光标 之间的所有字符。O(n)

void Insert(int &py, int &px, const wchar\_t \* cc, int flag);//指定光标位置 插入字符串,并且改变py px 为当前所在位置,，并将操作信息放入撤销栈中。Flag用来标注该操作放入撤销栈还是恢复栈。O（n）

void Insert(int &py, const wchar\_t \* cc);//同上插入字符串,不过此时为光标位置已经选好，主要面向撤销。O（n）

int KMP(const wchar\_t \*s, const wchar\_t \*t); //使用kmp算法进行匹配

line onSearch(line tmpL, const wchar\_t \* t); //在行tmpL内查找字符串t，调用KMP

line OnReplace(line tmpL, wchar\_t \* preStr, wchar\_t \* rpStr); //替换操作，preStr是查找的串，rpStr是待替换上的串

int GetCharNum(int x, int y, HDC& hdc); //给定像素横坐标x，行号y。返回值为：若在此处点击鼠标，光标前有几个字符。若超出总长度，则返回总长度值。

std::stack<undo> UndoStack;//撤销栈

std::stack<undo> RedoStack;//恢复栈

void Emptyredo();//当有新操作时，清空恢复栈

void Emptyundo();//新建文档时，清空撤销栈

void GetTotal();//全文统计功能。统计计空格字符，不计空格字符，中文字符功能。修改totalnum，totalnumwithoutspace，chinesenum等成员。O(n)

int GetTotal(selectPos Begin, selectPos End);//选中统计功能。统计选中部分总行数，计空格字符，不计空格字符，中文字符功能。修改totalnum，totalnumwithoutspace，chinesenum等成员。返回值为选中总行数。

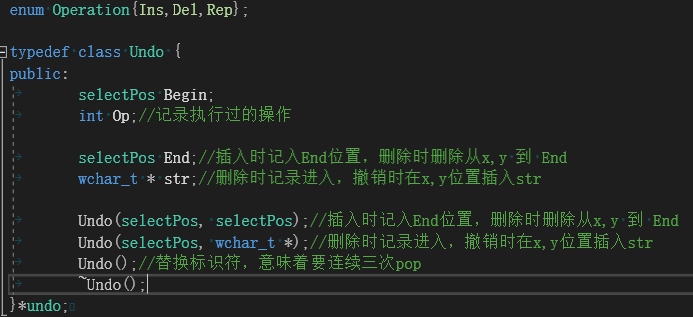
4.1.4栈

4.1.4.1概述：

采用两个栈UndoStack（撤销栈）与RedoStack（恢复撤销栈）来保存执行过的操作，以实现撤销和恢复的功能。每个栈内元素保存着操作的位置，基于插入与删除功能作为入口。并根据是插入，删除，来选择保存结束的位置还是字符串。并且使用替换机制，能够在替换后的撤销中一步恢复（而不是一步删除一步插入）。

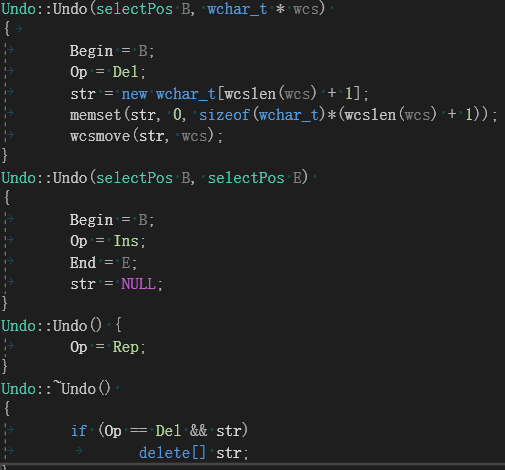
撤销栈没有限制，支持无限撤消（只要内存够）。在撤销后紧接着可以连续恢复撤销。若撤销后又开始修改当前数据，则恢复撤销栈清空。

4.1.4.2数据结构：



如上图，其中selectPos类型记录着操作的坐标值。

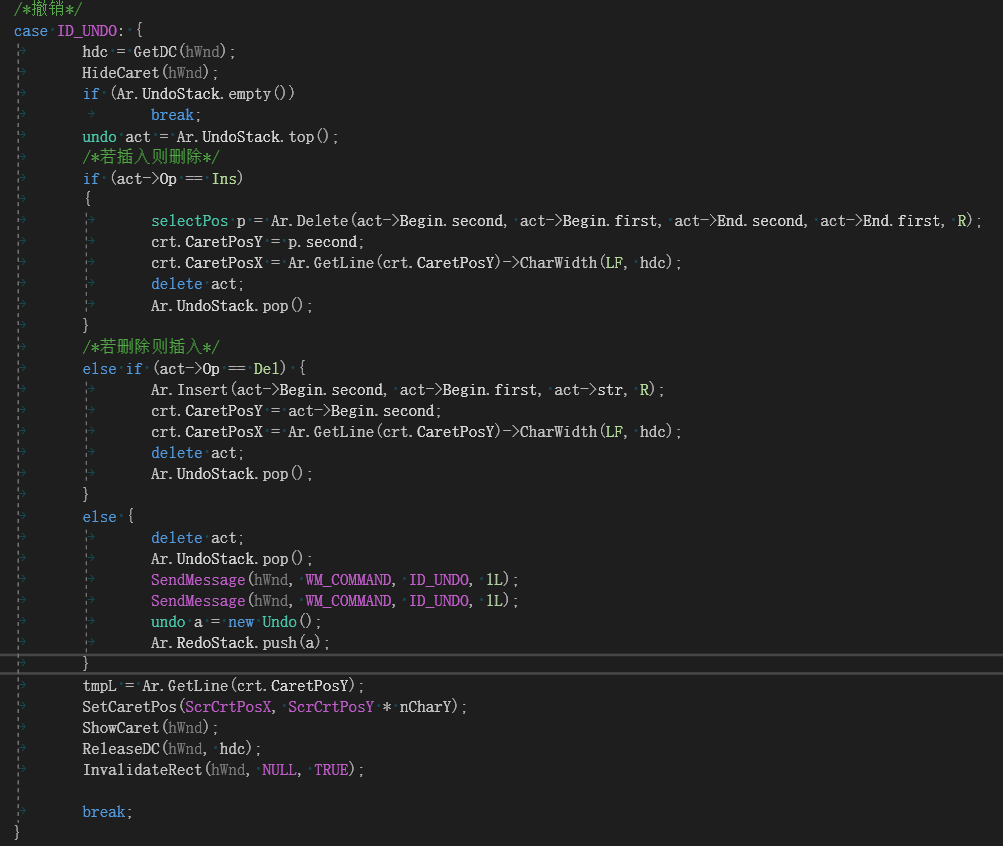
4.1.4.3函数实现



具体构造函数实现如上，设计的原则是：着重记录反操作所需要的参数。

若删除操作（图中Op等于Del那个），则重点存储删除字符串以及操作开始的位置。若插入操作（图中Op等于Ins），则重点存储删除的起始与终止位置，不记录字符串。若替换则只存Op=Rep就好。

4.1.4.4 面向用户操作：



上图为面向用户层的撤销操作。恢复撤销操作与之大体相同，只有Undo与Redo的细小差别。

若执行撤销操作，则根据栈顶元素的Op值来决定相应反操作。并且，执行反操作时，flag=R——将操作信息存入恢复栈。同理，恢复栈的反操作也会flag=U将信息存回恢复栈。

若其为替换操作，则往往伴随着删除+插入。那么这个时候发送两次UNDO操作（Redo栈将进入两个操作），然后将Rep信息存入Redo栈顶。恢复撤销时同理。

采用这种设计，不仅能节省存储空间，而且能够大大降低程序的耦合性，利于工程开发。

4.2 数据字典

4.2.1 数据结构模块数据字典

|  |  |  |
| --- | --- | --- |
| 变量名 | 变量类型 | 功能 |
| Line | 自定义 | 自定义行数据结构 |
| line | Line\* | 行指针 |
| gstart | int | Gap开始位置 |
| gend | int | Gap结束位置后一位 |
| arr | wchar\_t\* | 字符数组 |
| pre | line | 前一行 |
| next | line | 后一行 |
| len | int | 字符总数 |
| size | int | arr空间总大小 |
| GapIncrement | const int | arr满后增添的空间大小 |
| DefaultSize | const int | arr默认初始大小 |
| Article | 自定义 | 全文档的自定义数据结构 |
| firsL/lastL | line | 标志着首尾的哑行 |
| L | line | 永存的第一行 |
| LineNum | int | 总行数 |
| totalnum | Int | （选中）总字数（计空格） |
| totalnumwithoutspace | int | （选中）总字数（不计空格） |
| chinesenum | int | 中文字符数 |
| Undo | 自定义 | 自定义栈数据结构 |
| undo | Undo\* | Undo指针 |
| Begin | selectPos | 开始位置 |
| End | selectPos | 结束为止 |
| Op | int | 操作符标识 |
| str | wchar\_t\* | 删除操作存储的字符 |
| UndoStack | std::stack<undo> | 撤销栈 |
| RedoStack | std::stack<undo> | 恢复撤销栈 |

4.2.2 显示模块数据字典

|  |  |  |
| --- | --- | --- |
| 变量名 | 变量类型 | 功能 |
| crPrevBk | COLOREF | 记录编辑区背景颜色 |
| crPrevText | COLOREF | 记录编辑区文本颜色 |
| nCharX | int | 平均字符宽度 |
| nCharY | int | 平均字符高度 |
| nWindowCharX | int | 在平均宽度下编辑区一行能容纳的字符数 |
| nWindowCharY | int | 在平均高度下编辑区能容纳的字符行数 |
| nWindowX | int | 编辑区的宽度（像素值） |
| nWindowY | int | 编辑区的高度（像素值） |
| ScrCrtPosX | 宏 | 光标在屏幕中横坐标（实际位置-滚动条位置） |
| ScrCrtPosY | 宏 | 光标在屏幕中纵坐标 |
| scrFirstLine | int | 屏幕中第一行的实际行数 |
| scrLastLine | int | 屏幕中最后一行的实际行数 |
| si | SCROLLINFO | 记录滚动条信息的变量 |
| tm | TEXTMETRIC | 记录字体信息 |
| xScrPos | int | 横向滚动条的位置 |
| yScrPos | Int | 纵向滚动条的位置 |

4.2.3 用户操作模块数据字典

|  |  |  |
| --- | --- | --- |
| 变量名 | 数据类型 | 功能 |
| uFindReplaceMsg | UINT | 接受是否执行搜索或替换的信号 |
| fTextSelected | BOOL | 记录当前是否在选中状态 |
| selectBegin | selectPos | 记录选中区域的起点 |
| selectEnd | selectPos | 记录选中区域的终点 |
| curFilePath | wchar\_t \* | 记录当前文件保存的地址 |
| saveMark | BOOL | 记录当前文件是否保存 |
| isModified | BOOL | 记录当前文件是否修改过 |
| fr | FINDREPLACE | 搜索对话框对象实例 |
| szFindWhat | wchar\_t \* | 记录待搜索的字符串 |
| hdlg | HWND | 指向搜索结果的句柄 |
| MarksSearch | BOOL | 记录替换前是够已进行过搜索 |
| rp | FINDREPLACE | 替换对话框对象实例 |
| rpFindWhat | wchar\_t \* | 记录待替换的字符串 |
| rpReplaceWith | wchar\_t \* | 记录用来替换的字符串 |
| rdlg | HWND | 指向替换结果的句柄 |
| shiftPressed | BOOL | 记录shift是否按下 |

**五、 各模块设计说明**

5.1 数据结构模块设计说明：

属于最基本的模块，详见（四）。

5.2 显示模块设计说明

|  |  |  |  |
| --- | --- | --- | --- |
| 模块名称 | 显示模块 | | |
| 功能 | 将后台数据结构正确输出到屏幕上。 | | |
| 子模块 | 名称 | 功能 | 相关文件及函数或代码段 |
| 文本显示模块 | 正确显示文本以及选中时字符串变色问题。 | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  相关代码段：  编辑区重绘：主消息循环中的WM\_PAINT信号  选中变色：主消息循环中的WM\_MASK信号  选中恢复颜色：主消息循环中的WM\_DEMASK信号 |
| 光标显示模块 | 显示光标 | 相关文件：MiniWord.cpp、Caret.h、Caret.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  类：class Caret  相关代码段：  相关函数:   1. **BOOL** WINAPI SetCaretPos( 2. \_In\_ **int** X, 3. \_In\_ **int** Y 4. ); |
| 滚动条显示模块 | 实现滚动条 | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  相关代码段：  WM\_HSCROLL 横轴滚动条变化  WM\_VSCROLL 纵轴滚动条变化 |
| 菜单显示模块 | 显示菜单 | 相关文件：MiniWord.cpp，Undo.cpp,Undo.h |
| 相关文件 | MiniWord.cpp  MiniWord.h  Caret.h  Caret.cpp | | |
| 算法 | 双缓冲技术  双缓冲即在内存中创建一个与屏幕绘图区域一致的对象，先将图形绘制到内存中的这个对象上，再一次性将这个对象上的图形拷贝到屏幕上，这样能大大加快绘图的速度，避免刷新时的闪烁。 | | |
| 与其他模块的关系 | 和数据结构模块的关系：  当数据结构改变时，调用此模块进行绘图。  和操作模块的关系：  当鼠标操作模块改变了鼠标的位置或者选中的区域时，都需要显示模块来对鼠标和选中部分进行重新绘制；不仅如此，每当数据结构改变，我们都会通过显示模块刷新编辑器当前界面。 | | |

5.3 用户操作模块设计说明

|  |  |  |  |
| --- | --- | --- | --- |
| 模块名称 | 用户操作模块 | | |
| 功能 | 接收系统信号，通过分析信号来处理用户对编辑器的各种操作 | | |
| 子模块 | 名称 | 功能 | 相关文件及函数或代码段 |
| 输入操作处理&光标处理操作 | 根据用户的输入改变内存中的数据移动光标的位置；键盘选中功能；选中后替换或光标移动功能； | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  相关代码段：  按下按键：主消息循环中的WM\_KEYDOWN信号  松开按键：主消息循环中的WM\_KEYUP信号  输入字符：主消息循环中的WM\_CHAR信号 |
| 文件操作处理 | 新建、打开、保存等文件操作； | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  void openNewFile(wchar\_t \* Address, HDC hdc)；  void saveFile(wchar\_t \* Address)；  void createNewFile(HDC hdc);  void wstrcpy(wchar\_t \* desStr, const wchar\_t \* srcStr);  相关代码段：  查找功能： 主消息循环中的IDM\_SEARCH信号和if (message == uFindReplaceMsg)条件判断  替换功能： 主消息循环中的IDM\_REPLACE 信号和if (message == uFindReplaceMsg)条件判断  打开文件： 主消息循环中的IDM\_OPEN信号  保存文件： 主消息循环中的IDM\_SAVE信号  另存为文件：主消息循环中的IDM\_SAVEAS信号  新建文件：主消息循环中IDM\_NEWFILE信号 |
| 块操作处理 | 块拷贝、块粘贴、块剪切，块删除，块选中等； | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  相关代码段：  块拷贝：主消息循环中的ID\_COPY信号  块粘贴：主消息循环中的ID\_PASTE信号  块删除：主消息循环中的IDM\_DELETE信号  块剪切：主消息循环中的IDM\_CUT信号  键盘选中：主消息循环中的WM\_KEYDOWN信号  全选：主消息循环中的IDM\_SELECTALL信号 |
| 撤销与恢复处理 | 处理撤销与恢复操作； | 相关文件：MiniWord.cpp，Undo.cpp,Undo.h  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  Undo::Undo(selectPos B, wchar\_t \* wcs);  Undo::Undo(selectPos B, selectPos E);  相关代码段：  撤销：主消息循环中的ID\_UNDO信号  恢复：主消息循环中的ID\_REDO信号 |
| 鼠标操作处理 | 鼠标单击、双击、右键等操作； | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  相关代码段：  左键单击：主消息循环中的WM\_LBUTTONDOWN信号  左键双击：主消息循环中的WM\_LBUTTONDBLCLK信号  右键：主消息循环中的WM\_CONTEXTMENU信号 |
| 其他操作 | 字数统计，关于，  行号打印 | 相关文件：MiniWord.cpp  函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  Ctrl+M：主消息循环中的ID\_STATISTICS  Alt+ / ：主消息循环中的WM\_ABOUT  自动显示：主消息循环中的WM\_PAINT |
| 相关文件 | MiniWord.cpp  MiniWord.h  Undo.cpp  Undo.h  Caret.cpp  Caret.h  subeditor.cpp  subeditor.h | | |
| 效果 | 主要应用了switch case语句来分析主消息循环中收到的各种信号，然后做出相应的操作。部分采用win32的函数，比如文件处理中就运用到了win32的OPENFILENAME对象和GetOpenFileNameW(&openFile);函数，来调出一个打开文件的对话框以及接受选择的信息。  首先过滤部分输入，回车视为换行，并不保存回车符。Tab均一次性保存为4个空格（可改）。字符为变长字符（即lll与AAA宽度显然不同），可区分圆角半角。  输入子模块中，一旦接收到一个键盘信号，首先判断是字符，方向键，回车，删除，tab，shift等按键，如果是字符，删除，tab，回车，修改数据结构；如果是方向键，则根据输入信号移动光标；如果收到Home, End, PageUp, PageDown等信号，则跳转光标位置，并对界面进行相应的刷新；如果收到shift信号，则修改一个按下shift的标记，用于键盘选中  文件操作中，根据用户按下的热键或者在菜单中点击的按键，会弹出不同的会话框，接受用户选择保存或者打开文件的地址，然后将当前内存数据的内容转换为文本格式，输入到对应文件中。如果是新建文件，则在询问保存后，清空当前数据。  打开与保存过程中，支持多种格式文件的筛选。并且选择指定类型后，将默认文件后缀名为指定后缀名。  块操作模块，主要是针对信号对数据结构做出相应的处理。基于块选中，块插入，块删除，块成串等基本操作，实现了诸多功能。  ①复制，就将选中的部分内存数据转化为字符串格式，放到系统的剪贴板中  ②粘贴则将剪贴板中的内容转化为编辑器可接受的数据结构，并修改内存。  ③撤销与恢复都是基于块而进行的操作。  ④选中后修改，如：粘贴，回车，字符，tab，均为块的替换。  ⑤选中后的光标移动操作，上下左右home、end、page，均人性化移动光标位置至相邻端后操作。  撤销与恢复主要用到了栈的数据结构，每当进行一次操作，就对其进行记录。比如撤销，就对栈顶操作执行其逆操作并弹出，直到栈为空为止。同时，替换操作作了优化，替换一步到位，不需要执行两次撤销：删除+插入。  鼠标操作完全由自己实现，对鼠标进行定位后判断其在数据结构中的位置，然后进行相应操作。选中的文字渲染，光标位置判断，越位的优化，均基于最基本的x，y坐标与最基本的打印功能自己实现，效果良好，执行流畅。  支持滚动条操作，越界可自动滚动。  统计字数模块，不仅可以统计全文，而且可以统计选中部分。 | | |
| 特点 | 根据用户的输入不同，执行不同的操作。  消息处理都在主循环中，效率高。 | | |
| 与其他模块的关系 | 和数据结构模块的关系：  操作模块中的输入子模块、文件子模块、块操作子模块、撤销恢复子模块负责接收用户输入的信号，然后对数据结构中的数据进行各种类型的修改，比如块插入、块删除，打开文件(先清空，再读入目标文件的文本)、输入字符、撤销等等；  和显示模块的关系：  当鼠标操作模块改变了鼠标的位置或者选中的区域时，都需要显示模块来对鼠标和选中部分进行重新绘制；不仅如此，每当数据结构改变，我们都会通过显示模块刷新编辑器当前界面。 | | |

**六、 范例执行结果及测试情况说明**

6.1 范例1

范例1说明:……

6.2 范例2

范例2说明:……

6.3 范例3

范例3说明:……

(尽量详细，涉及到我们考虑到的很多特殊情况，需要尽量写上去)

**七、 评价及改进意见**

7.1 评价

VigorWord文本编辑器是一款功能强大，效率极致，节约空间，操作人性化的文本编辑器。在时间有限的工期以及限定的数据结构内进行了最大限度的优化与创造，倾注了三位同学很多的热情与心血，不放过任何的小细节，在撤销，字数统计等很多功能上甚至超过了win自带的记事本。总之，我们非常喜欢并愿意去使用自己创造的产品，并乐意让更多人去体验。

代码数量在2500行左右。在项目初期我们并没有忙于开发，而是做了大量资料的收集。数据结构上选择了Gapbuffer，框架选择了较为底层的Windows32 API来挑战自己的能力，并使用git进行团队协调。事实证明这是一个较好的选择。Gapbuffer使得程序高效而节省，使用windows32 api能让我们对Windows窗口程序的运行过程有较为深入的了解，同时，使用git大大降低了我们合并代码的时间成本，对我们今后的团队协作很有帮助。

VigorWord产品定位：轻盈的代码编辑器。因为采用非回车不自动换行机制，对于短字段，多回车型文字编辑友好，即比较适合编写代码。于是针对代码习惯优化了数据结构：如默认GapBuffer大小为60，在空间与时间中做出了好的权衡等。

总而言之，这次全屏幕编辑大作业完成度较高，整个协作过程契合度较好，团队协作效率高，总体上相当满意。

7.2 改进意见

虽然整个项目完成度、协作度都很棒，但还是存在一些小问题：

1. 项目初期没有明确分工。项目开始的时候没有明确的分工，只是说各自实现怎样的功能，导致在一些交叉领域分工不明确。同时没有提前写好数据字典，对一些变量的理解出现误差。

2.对windows api的了解不足，导致行号显示功能没能找到最佳解决方案。应该有更好的方式重绘，而现在是将行号与文本融为一体，在横向拖动滚动条时会产生bug。

3.在项目初期对git了解不足，走了一些弯路。

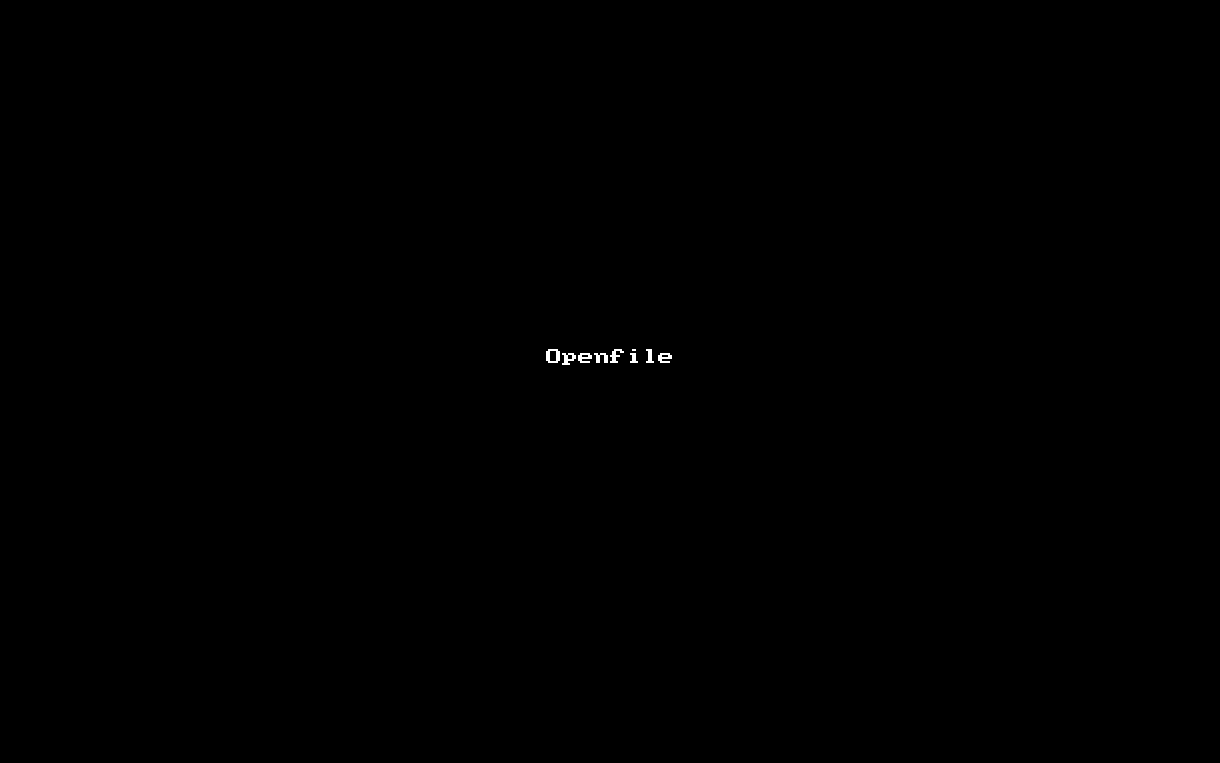
4.开发经验仍需加强，第一遍可谓是摸着石头过河。在虽然在现有的开发模式上实现了很多功能，达到了应有的效果，但是一些地方可能还可以更规范一些。（1）成员函数的设计上不得不说存在着冗余之感，同样的功能实现方式往往有多种。以下部分待优化：如可以将所有line成员函数放到Article中实现并将line作为参数进行操作。这样可以执行更轻松的执行多行操作，功能也更加的统一。

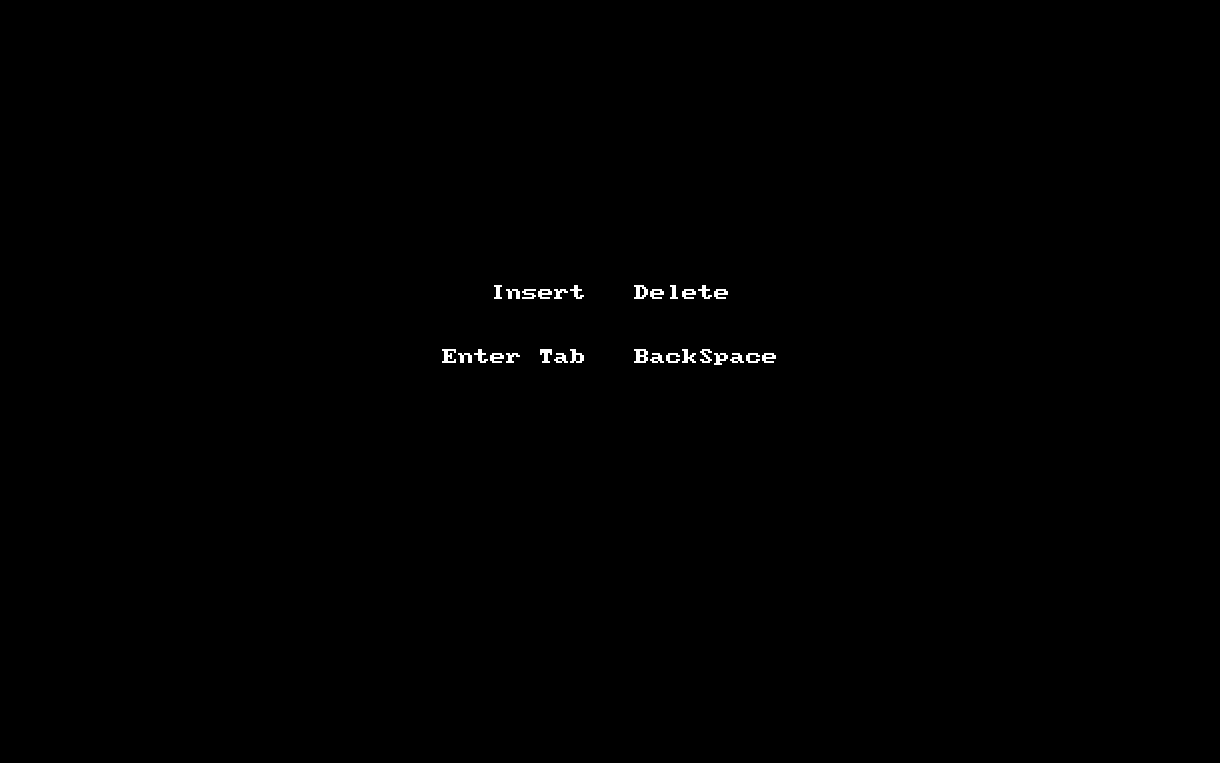
（2）功能扩展性还可以更强，可以给每个字单独创建一个类，字的颜色，大小，字体，宽度均可用一个位串来存储，同时改变所有字符串相关函数为对字类操作的方法。以此，进一步拓宽功能。

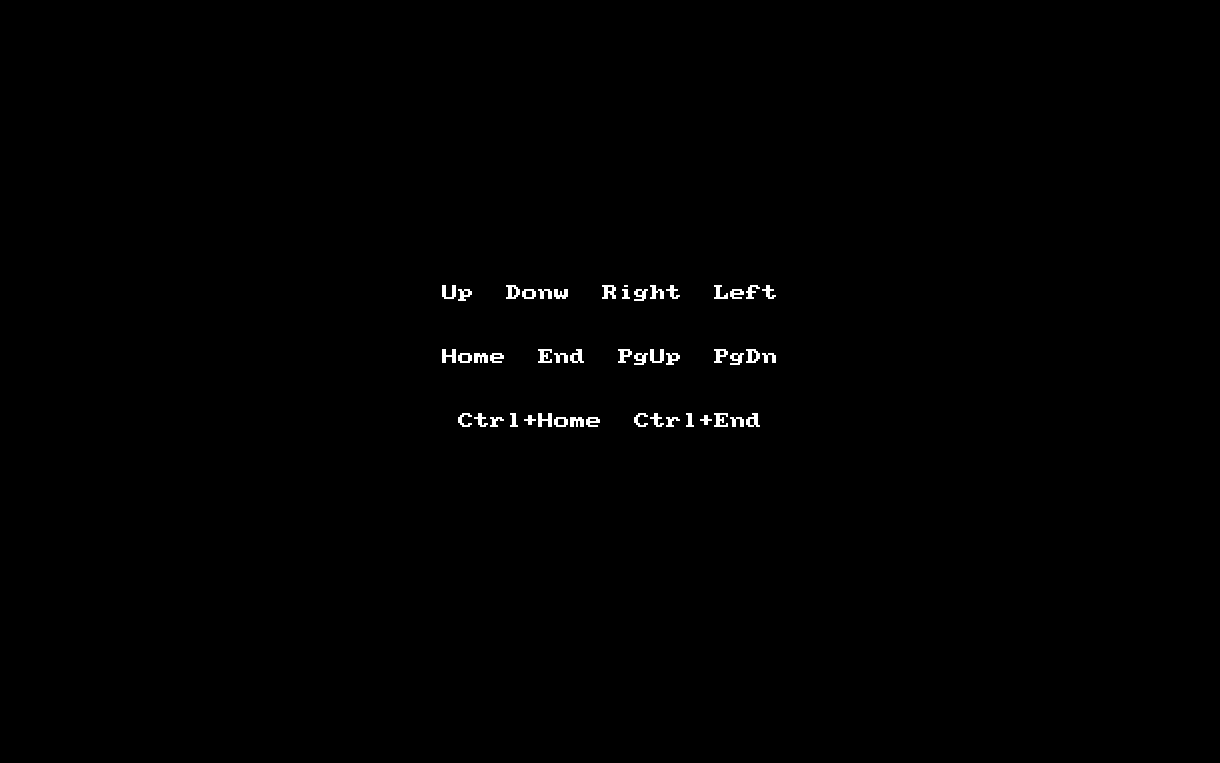
（3）显示渲染层与操作层写在了一起，虽然耦合度并不高，只差封装一下，但是还是显得主文件MiniWord.cpp过于臃肿，多文件模式可能会显得更灵活一些。

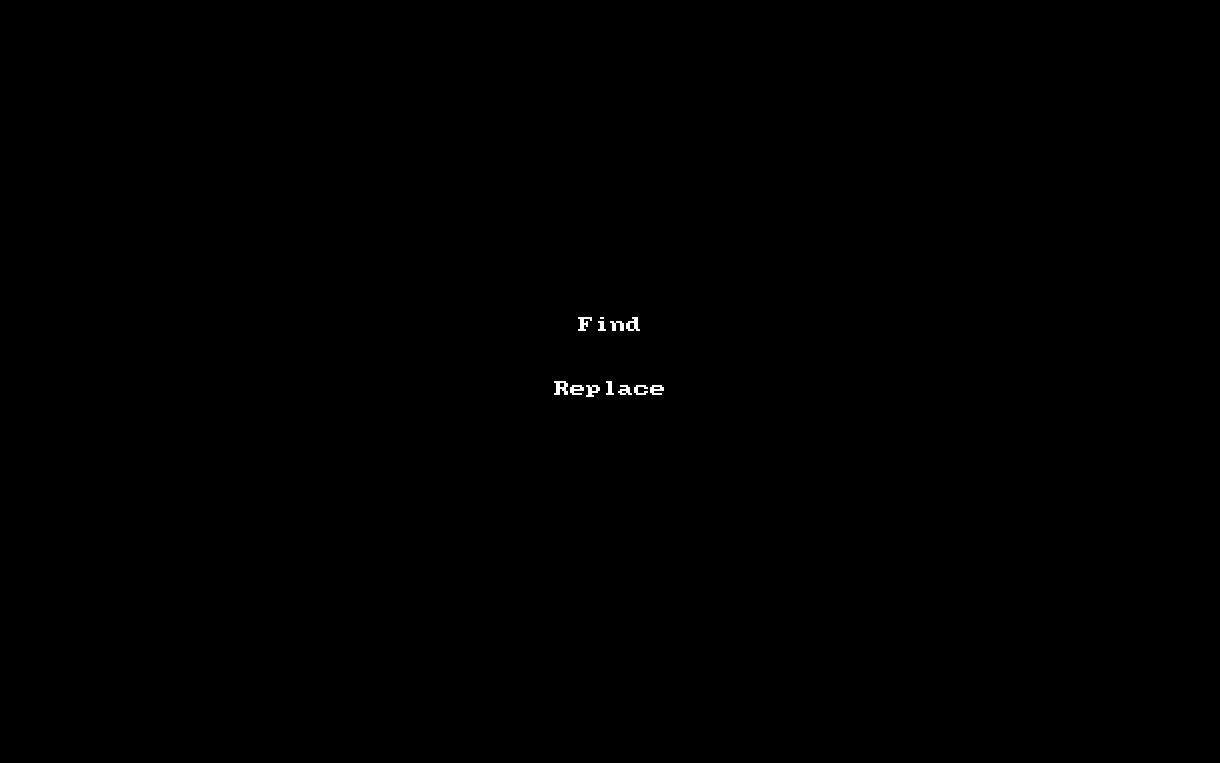
5.虽然操作丰富，但是界面略原始，如果有更长的时间，我们会愿意更好的完善界面。

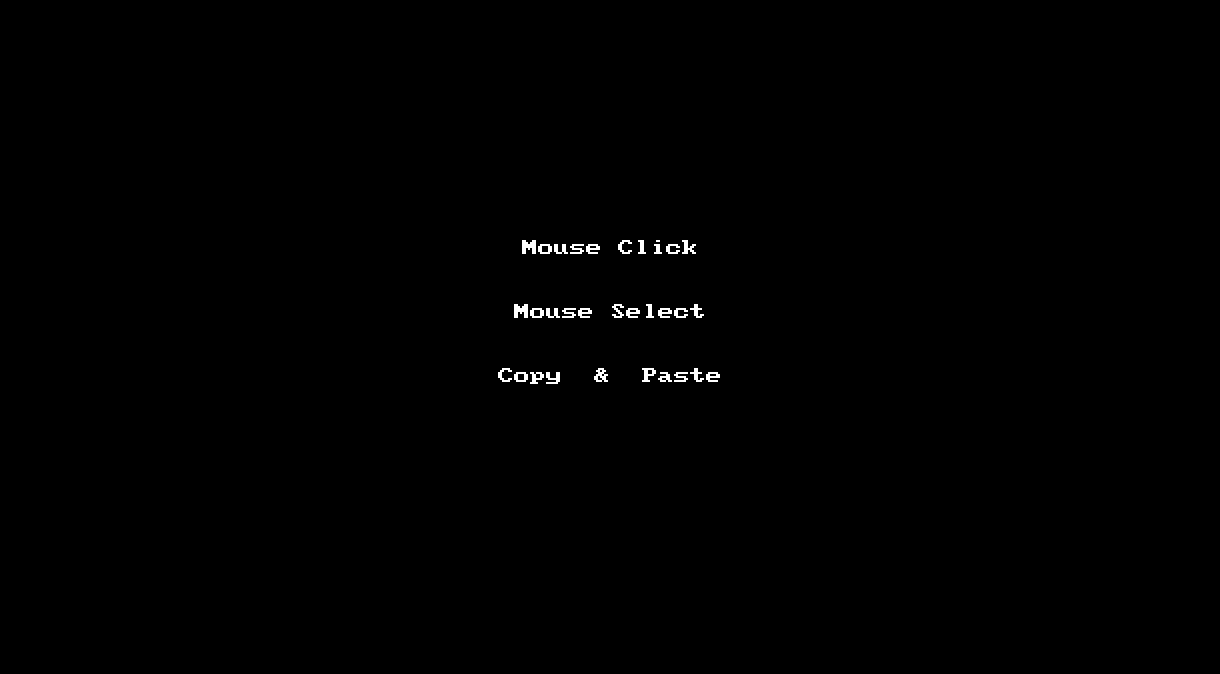
**八、用户使用说明**

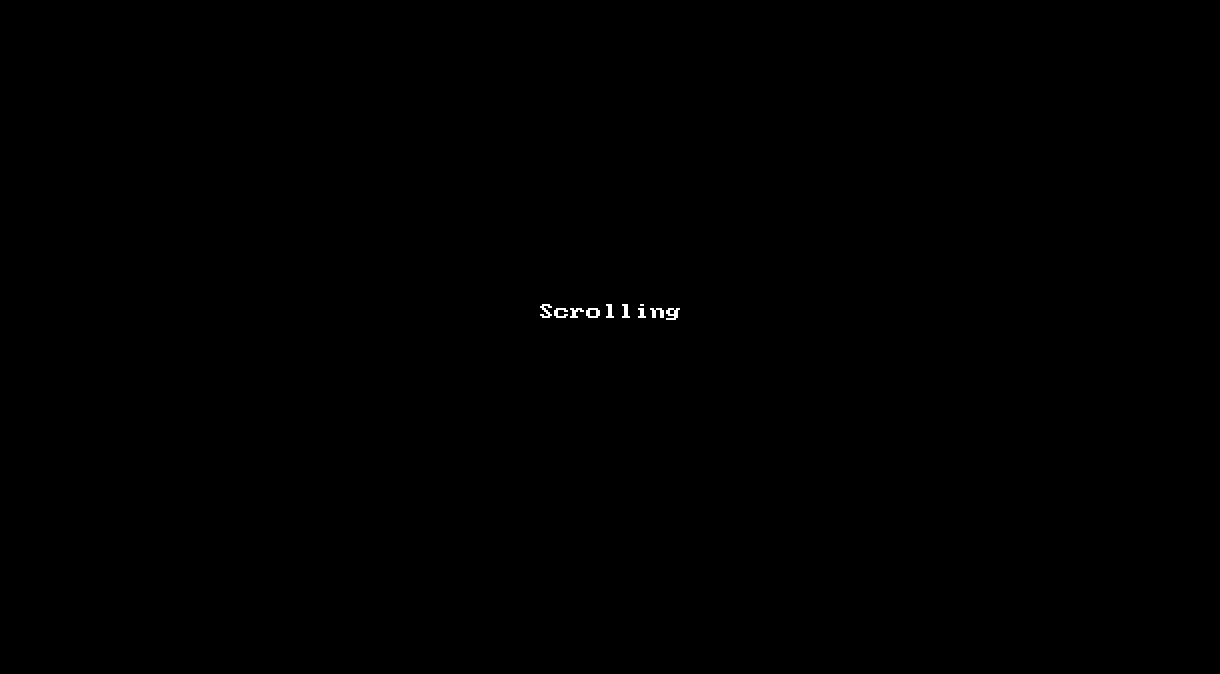
****

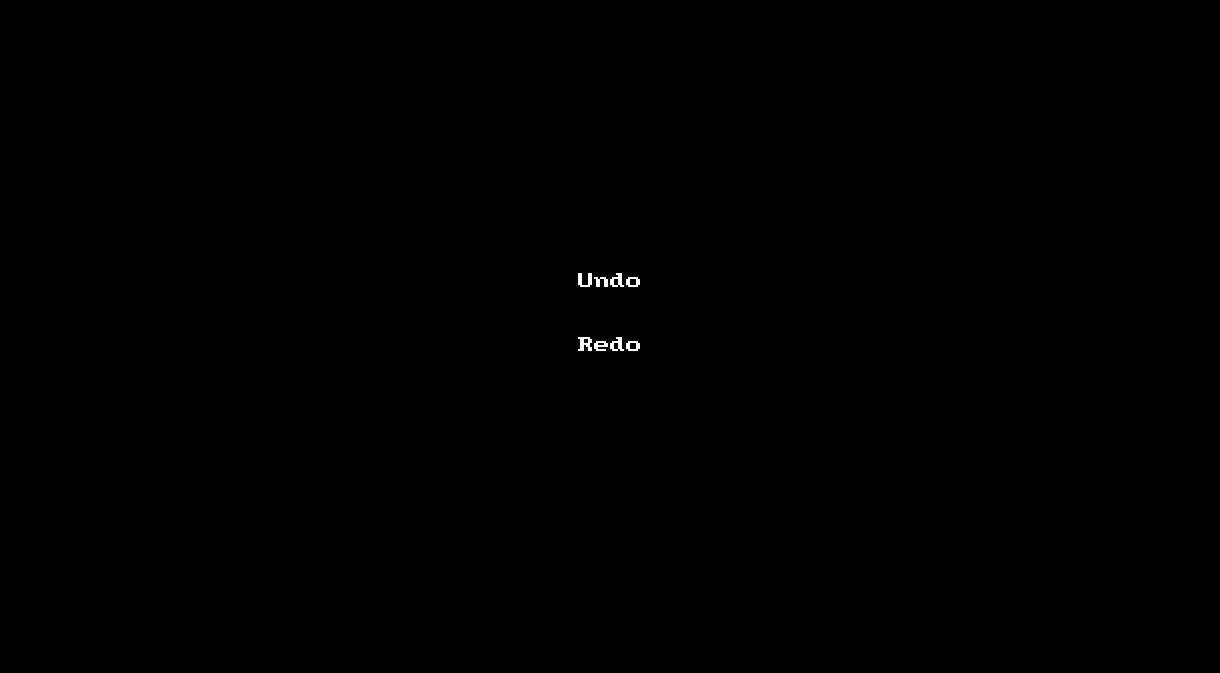
****

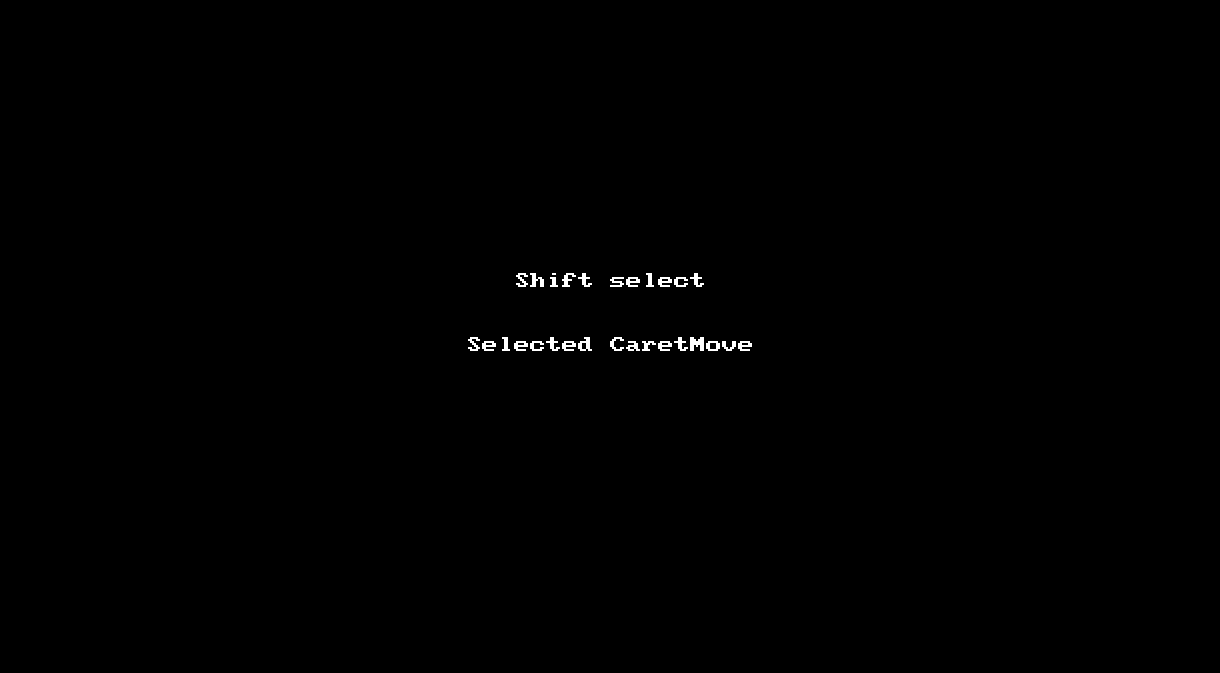
****

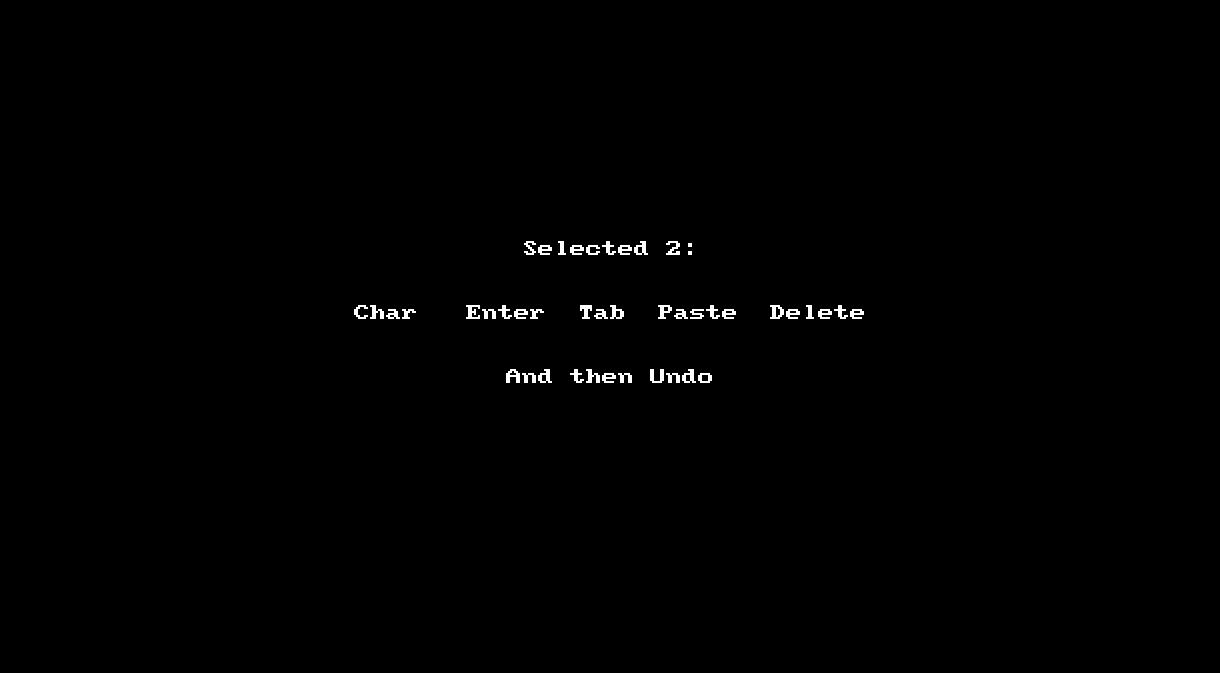
****

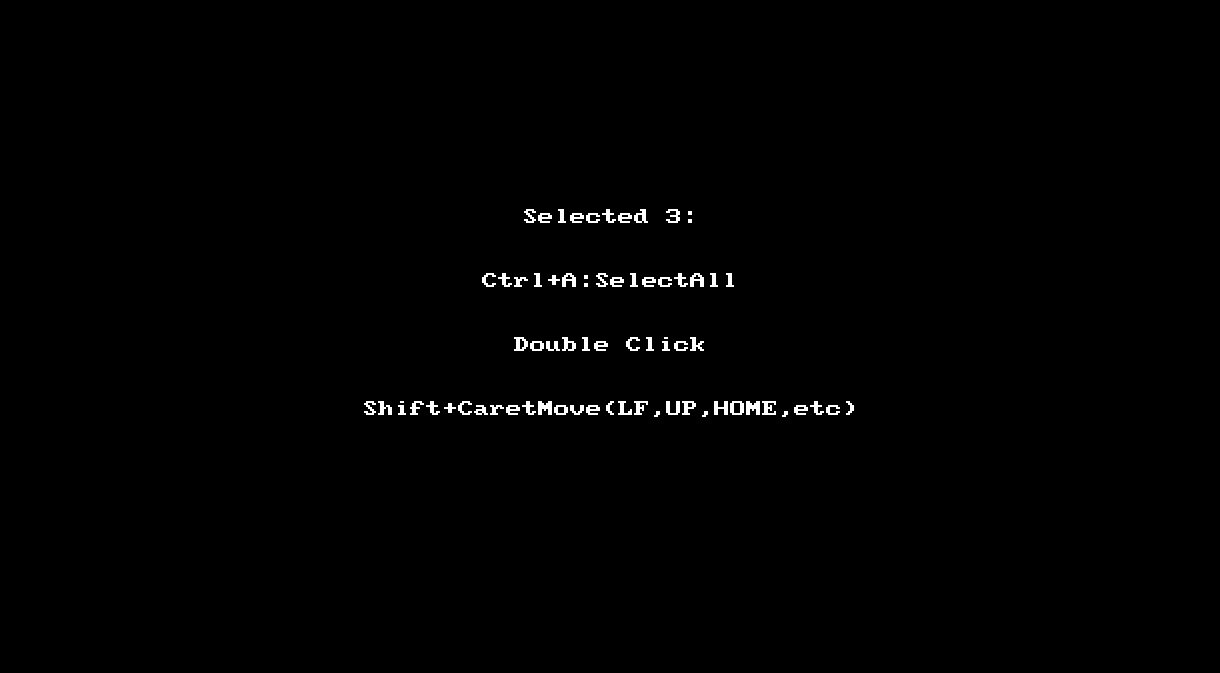
****

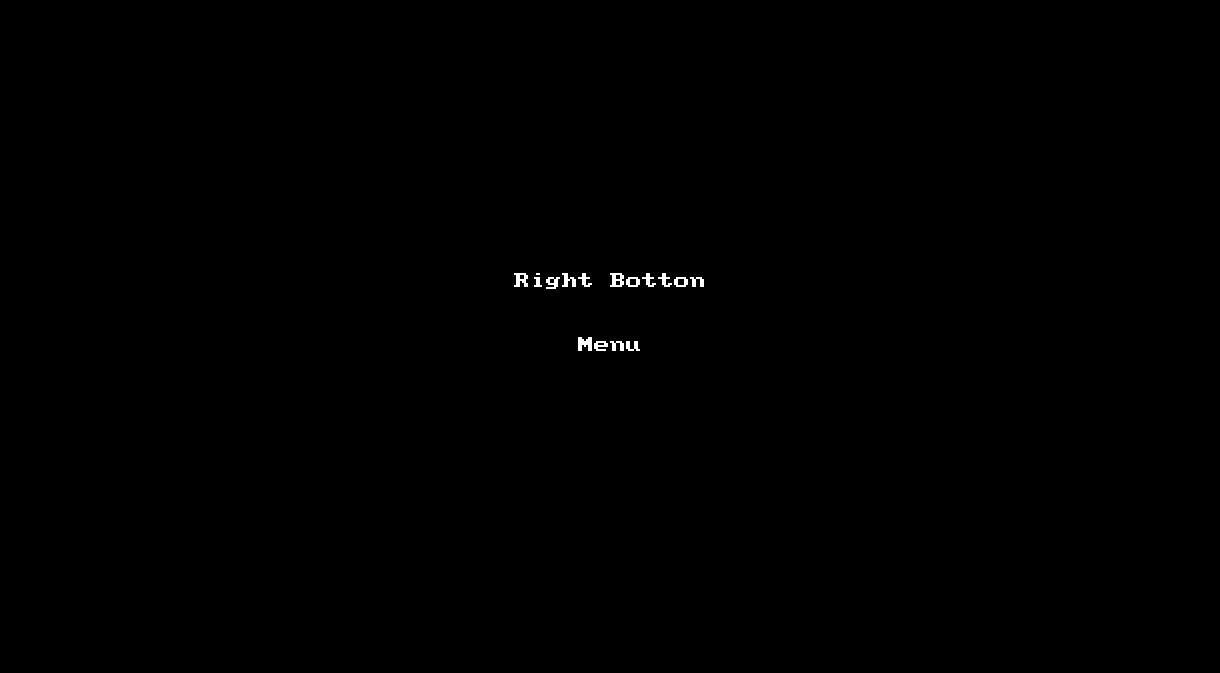
****

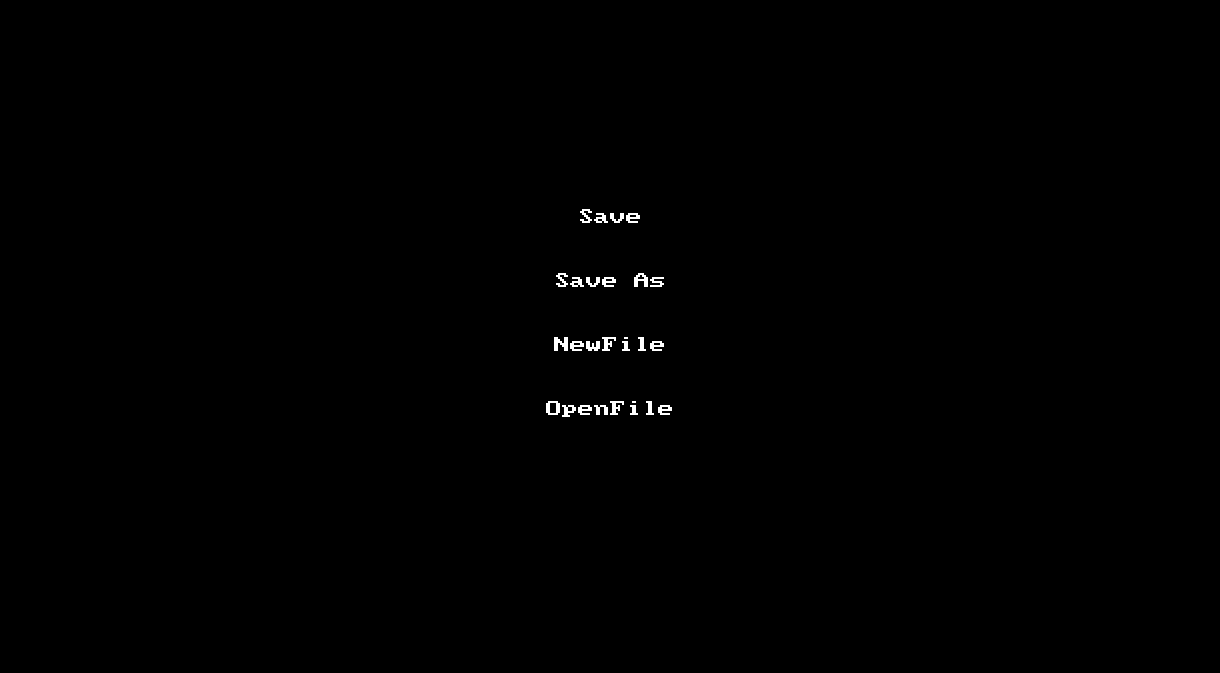
****

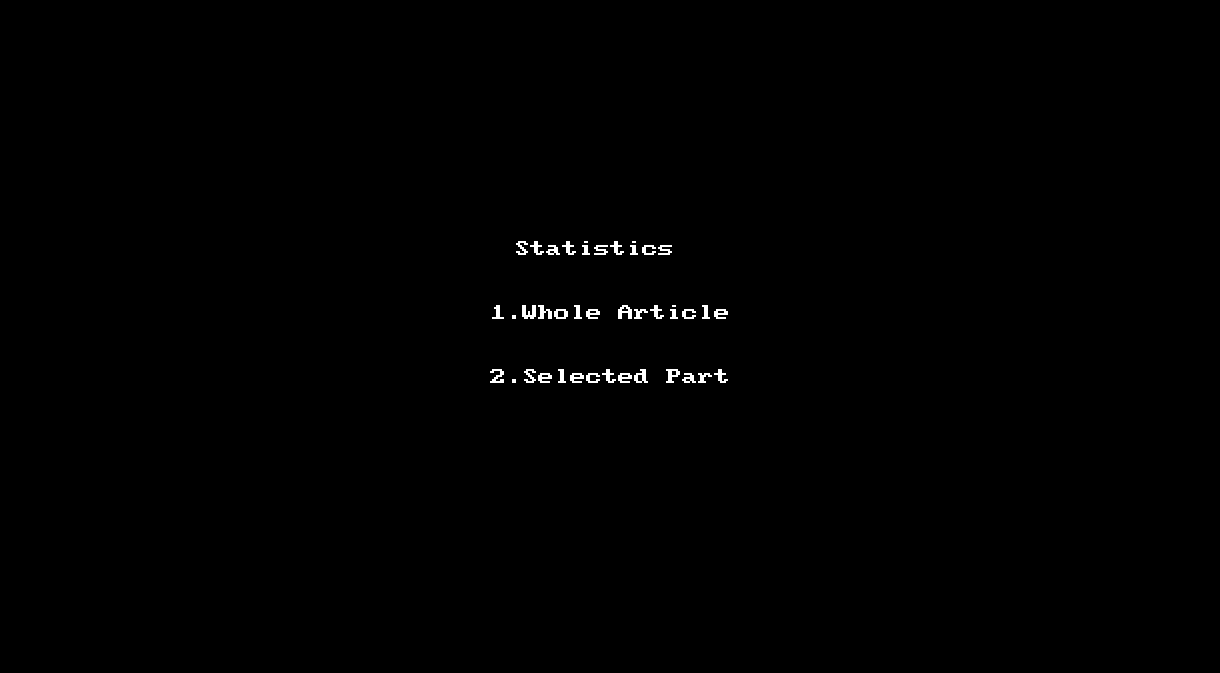
****

****

****

****

****

****

**九、团队合作**

9.1 综述

本次数据结构程序设计大作业由卢昱昊、王智韬、魏子然三名同学通力合作完成，三人分工明确，合作流畅。代码托管平台运用的是coding。通过本次大作业，我们组的同学不仅提升了自己的代码能力、解决问题的能力，同时三人均熟练掌握了如何使用git来三人同时完成一个工程。小组成员间流畅的合作大大减少了我们编写代码的时间成本，提升了编写效率，增强了代码风格的统一性和接口的契合度。

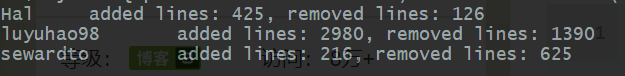
9.2 托管平台



Figure 9.2.1



Figure 9.2.2



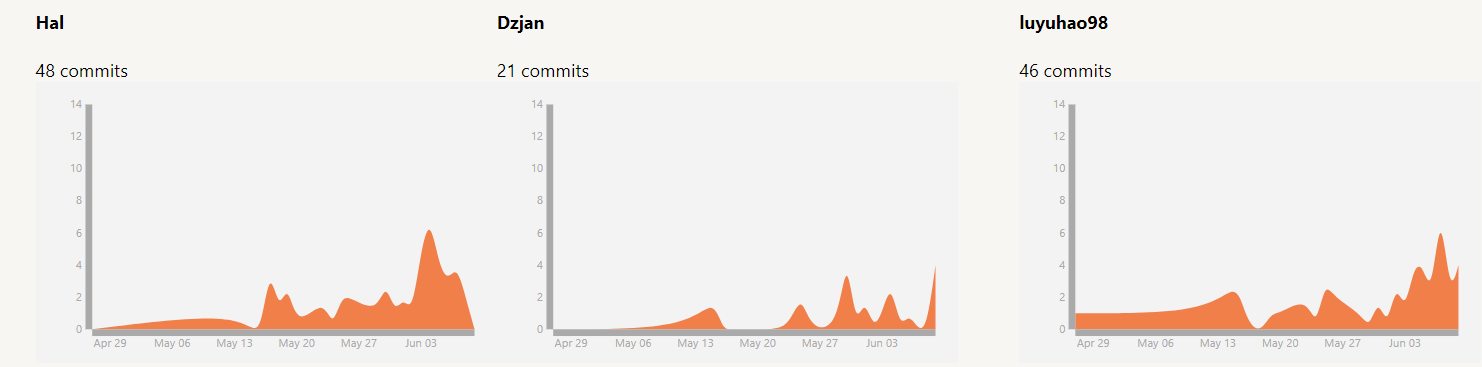


Figure 9.2.3



Figure 9.2.4