数据结构程序设计说明文档

————VigorEditor

一、 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。

二、 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

三、 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

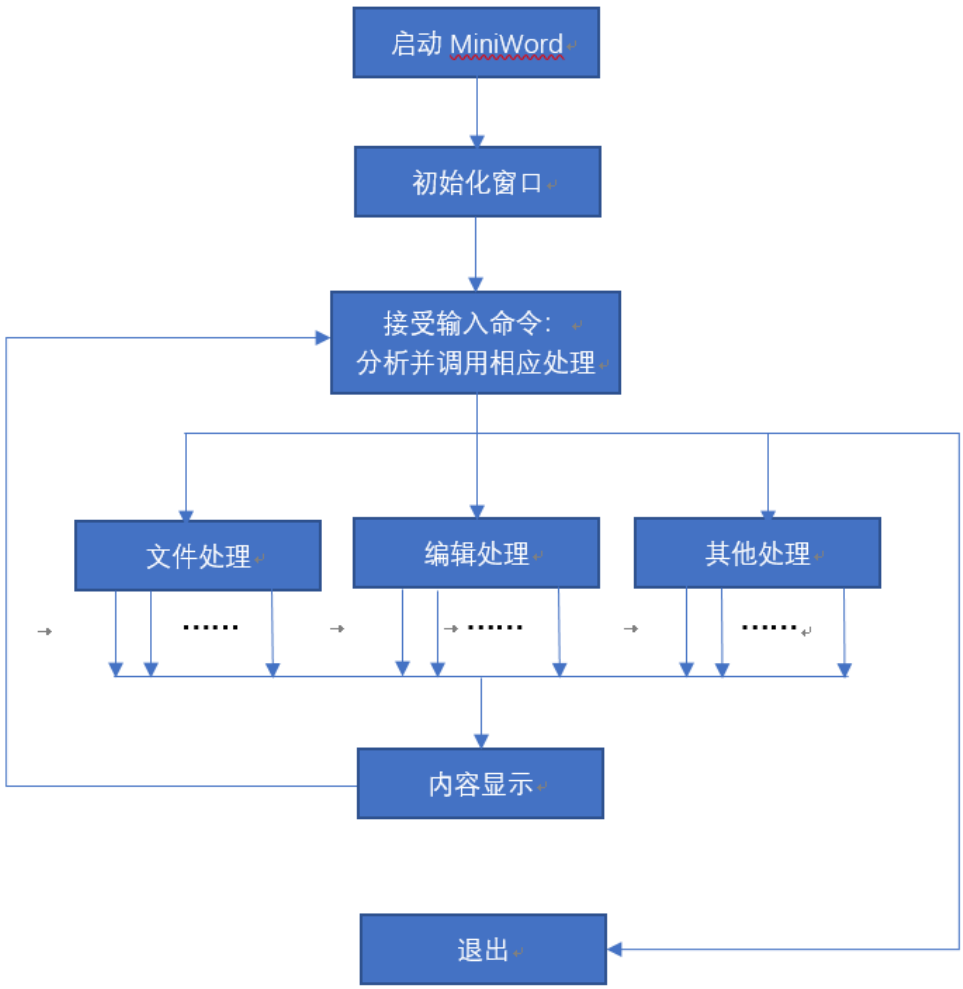


图 3.2

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；

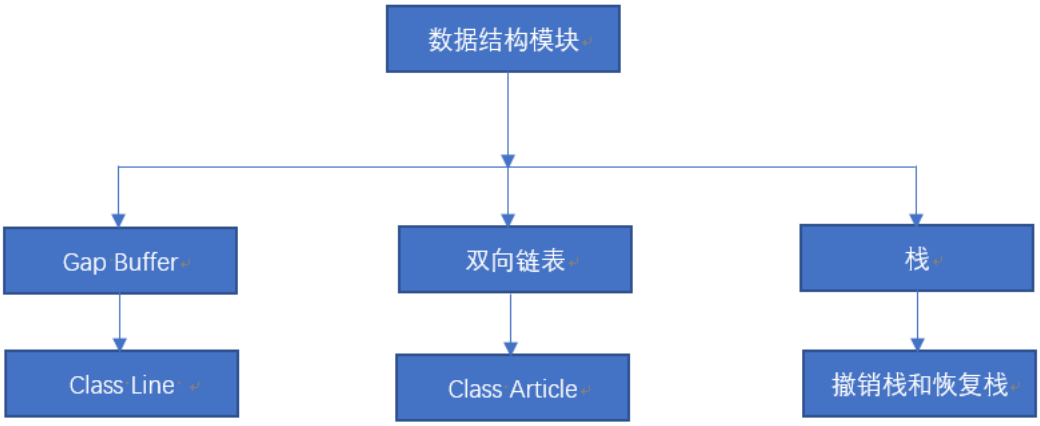


图 3.3.1

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

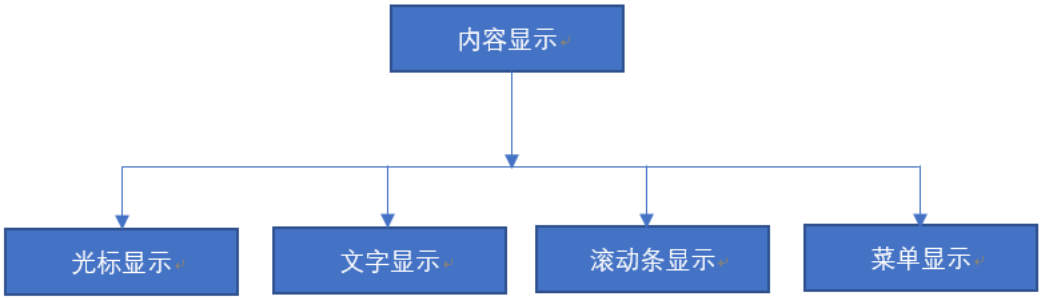


图 3.3.2

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;

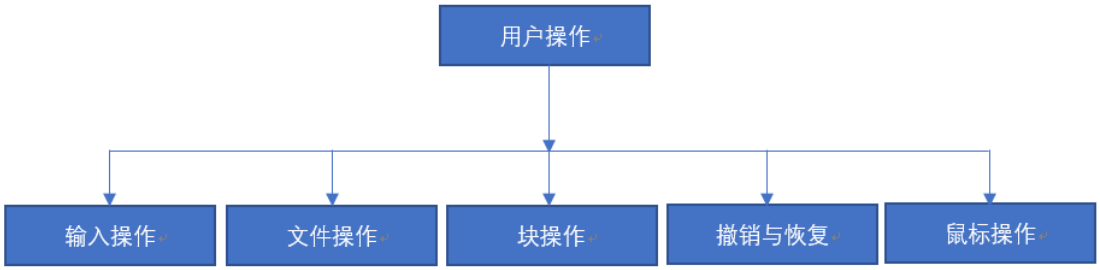


图 3.3.3

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。

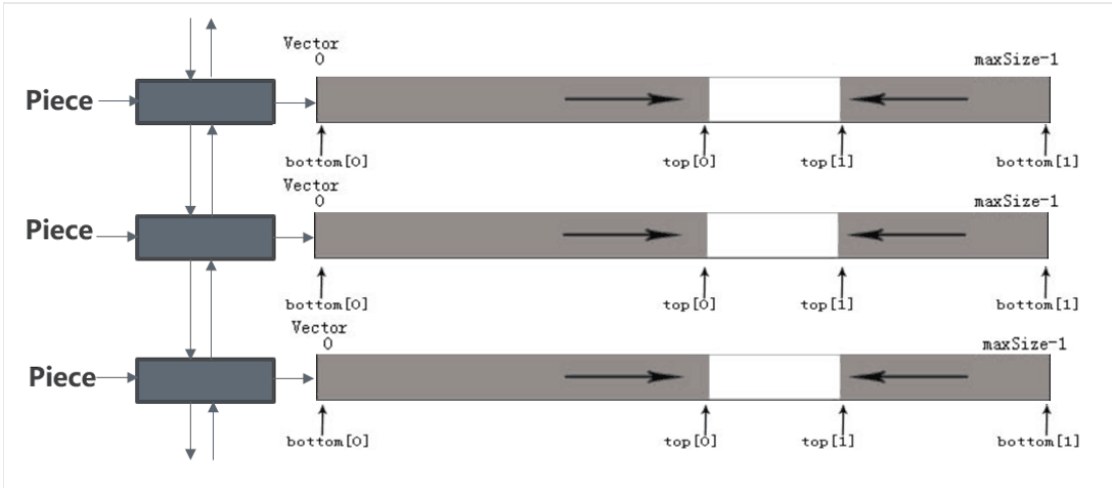
数据结构总体写在 subeditor 中。开发宗旨有二：1，确保可靠高效地操纵数据；2，尽可能使操作层与显示层方便开发。

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：

```
const int GapIncrement = 30;// gapArticle每次增加大小
const int DefaultSize = 60;//默认
typedef class Line * line;
enum stack { LF=-1, RG=1 };

enum UNRE { U, R };//区分对哪个栈进行操作

typedef class Line
{
public:
	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
	line NewLine();//在本Line后面新建一个Line

	int IsFull();//判断Line是否为满
	int RleaseProcess();//释放gapbuffer为0;
	void OverflowProcess(); //用于满了后申请数组
```

```
    int 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。
    void PointMoveto(int d);//将光标移动到第d个字符
    //PointMove(-1) →       →       int LeftMovePoint();//不改变原句，光标左移
    //PointMove(1) →        →       int RightMovePoint();//不改变原句，光标右移
    int Gapmove();//改变point后移动gap
    int UsertoGap(int);//传入面向user字符的位置，返回在arr中的真实位置
    int Gapgsize();//取gap的宽度;
    int GetPoint();//取gstart（目前光标位置）
    int Getlen();//取len有效字符长度（用户眼中字符长度）
    int Getlen(int i) const;//取len有效字符长度（用户眼中字符长度）
    wchar_t * GetPos();//返回该行字符串指针
    wchar_t * GetPos(int i);//返回左右字符串 LF, return arr , RG , return arr+gend
    wchar_t * GetStr();//返回字符串;
    int CharWidth(HDC hdc);//字符长度
    int CharWidth(int i, HDC hdc) const;//i=1 右侧，i=-1 左侧
    int IsEmpty(int i) const;       //判断Line是否为空，i可为LF，RG
    int IsEmpty();  →       →       //判断Line是否为空
    void MakeEmpty();//清空内容
    void MakeEmpty(int i);//清空内容 清空左右内容
    wchar_t Top(int i);//LG得到左侧元素 RG得到右侧元素
    void Push(const wchar_t c, int i);//插入一个字符 LF,插左，RG插右
    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。
    void Rwrite(const wchar_t &c);//替换输入下一字符
    bool IsFirstL() { return this->pre->pre == nullptr; }
    bool IsLastL() { return this->next->next == nullptr; }
```

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:

```
class Article {
private:
	line firstL;
	line lastL;
	int lineNum;

public:
	line L;//首行，无法删掉
	int totalnum;
	int totalnumwithoutspace;
	int chinesenum;

	Article();//构造 linehead的next为空
	~Article();
	bool IsEmpty() const;
	bool IsFirstL(line& L) const { return L->pre == firstL; }
	bool IsLastL(line& L) const { return L->next == lastL; }
	bool IsEnd(line& L) const { return L == lastL; }
	line GetLine(int lineNum) const;
	int GetNum(line& L) const;
	int MaxWidth(HDC) const;
	void InsertAfter(line& L);
	void Remove(line &L);
	int LineNum(void) const { return lineNum; }
	void IncLineN(void) { lineNum++; }
	void DecLineN(void) { lineNum--; }
	void clearWord(); //清空当前Article
	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
	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
	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,主要面向粘贴等
	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
	/* 查找功能 */
	line onSearch(line tmpL, const wchar_t * t);
	int KMP(const wchar_t *s, const wchar_t *t);
	int * getNextVal(const wchar_t *s);
	/*替换功能*/
	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
	/*给定坐标x（左条长+现鼠标与左端距离），行号y
	return值为：若点下后的gstart。超过行距返回glen*/
	int GetCharNum(int x, int y, HDC& hdc);
```

```
	/*撤销栈与恢复栈*/
	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();//返回总字数
	int GetTotal(selectPos Begin, selectPos End);//返回选中总字数 并返回总行数
```

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：

```
enum Operation{Ins,Del,Rep};

typedef class Undo {
public:
	selectPos Begin;
	int Op;//记录执行过的操作

	selectPos End;//插入时记入End位置，删除时删除从x,y到End
	wchar_t * str;//删除时记录进入，撤销时在x,y位置插入str

	Undo(selectPos, selectPos);//插入时记入End位置，删除时删除从x,y到End
	Undo(selectPos, wchar_t *);//删除时记录进入，撤销时在x,y位置插入str
	Undo();//替换标识符，意味着要连续三次pop
	~Undo();
}*undo;
```

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

```
Undo::Undo(selectPos B, wchar_t * wcs)
{
	Begin = B;
	Op = Del;
	str = new wchar_t[wcslen(wcs) + 1];
	memset(str, 0, sizeof(wchar_t)*(wcslen(wcs) + 1));
	wcsmove(str, wcs);
}
Undo::Undo(selectPos B, selectPos E)
{
	Begin = B;
	Op = Ins;
	End = E;
	str = NULL;
}
Undo::Undo() {
	Op = Rep;
}
Undo::~Undo()
{
	if (Op == Del && str)
		delete[] str;
```

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:

```
/*撤销*/
case ID_UNDO: {
	hdc = GetDC(hWnd);
	HideCaret(hWnd);
	if (Ar.UndoStack.empty())
		break;
	undo act = Ar.UndoStack.top();
	/*若插入则删除*/
	if (act->Op == Ins)
	{
		selectPos p = Ar.Delete(act->Begin.second, act->Begin.first, act->End.second, act->End.first, R);
		crt.CaretPosY = p.second;
		crt.CaretPosX = Ar.GetLine(crt.CaretPosY)->CharWidth(LF, hdc);
		delete act;
		Ar.UndoStack.pop();
	}
	/*若删除则插入*/
	else if (act->Op == Del) {
		Ar.Insert(act->Begin.second, act->Begin.first, act->str, R);
		crt.CaretPosY = act->Begin.second;
		crt.CaretPosX = Ar.GetLine(crt.CaretPosY)->CharWidth(LF, hdc);
		delete act;
		Ar.UndoStack.pop();
	}
	else {
		delete act;
		Ar.UndoStack.pop();
		SendMessage(hWnd, WM_COMMAND, ID_UNDO, 1L);
		SendMessage(hWnd, WM_COMMAND, ID_UNDO, 1L);
		undo a = new Undo();
		Ar.RedoStack.push(a);
	}
	tmpL = Ar.GetLine(crt.CaretPosY);
	SetCaretPos(ScrCrtPosX, ScrCrtPosY * nCharY);
	ShowCaret(hWnd);
	ReleaseDC(hWnd, hdc);
	InvalidateRect(hWnd, NULL, TRUE);

	break;
}
```

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

<table>
<tr><td rowspan="3"></td><td>光标显示模块</td><td>显示光标</td><td>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);</td></tr>
<tr><td>滚动条显示模块</td><td>实现滚动条</td><td>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</td></tr>
<tr><td>菜单显示模块</td><td>显示菜单</td><td>相关文件：MiniWord.cpp，Undo.cpp,Undo.h</td></tr>
<tr><td>相关文件</td><td colspan="3">MiniWord.cpp
MiniWord.h
Caret.h
Caret.cpp</td></tr>
<tr><td>算法</td><td colspan="3">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。</td></tr>
<tr><td>与其他模块的关系</td><td colspan="3">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。</td></tr>
</table>

5.3 用户操作模块设计说明

<table>
<tr><td>模块名称</td><td colspan="3">用户操作模块</td></tr>
<tr><td>功能</td><td colspan="3">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</td></tr>
<tr><td rowspan="2">子模块</td><td>名称</td><td>功能</td><td>相关文件及函数或代码段</td></tr>
<tr><td>输入操</td><td>根据用户的输入</td><td>相关文件：MiniWord.cpp</td></tr>
</table>

	作处理&光标处理操作	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；	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

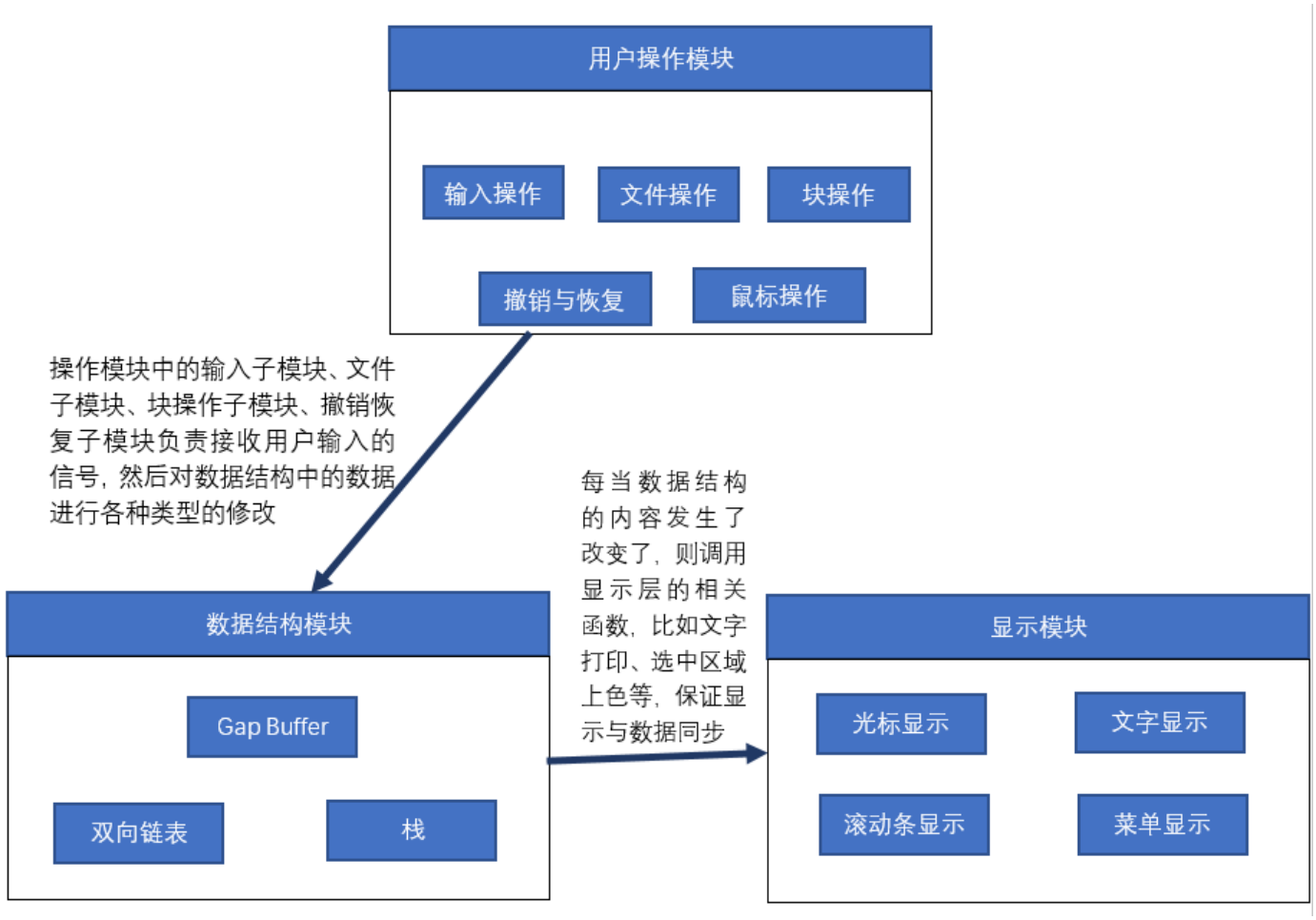
	文件操作处理	新建、打开、保存等文件操作；	相关文件：MiniWord.cpp
			函数：LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
			void openNewFile(wchar_t * Address, HDC hdc);
			void saveFile(wchar_t * Address);
			void createNewFile(HDC hdc);
			void wstrcpy(wchar_t * desStr, const wchar_t * srcStr);
			相关代码段：
			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、搜索、替换等；	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 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
	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。

模块关系：



六、 范例执行结果及测试情况说明

6.1 启动界面

Figure 6.1

6.2 普通编辑界面

Figure 6.2

6.3 大段粘贴

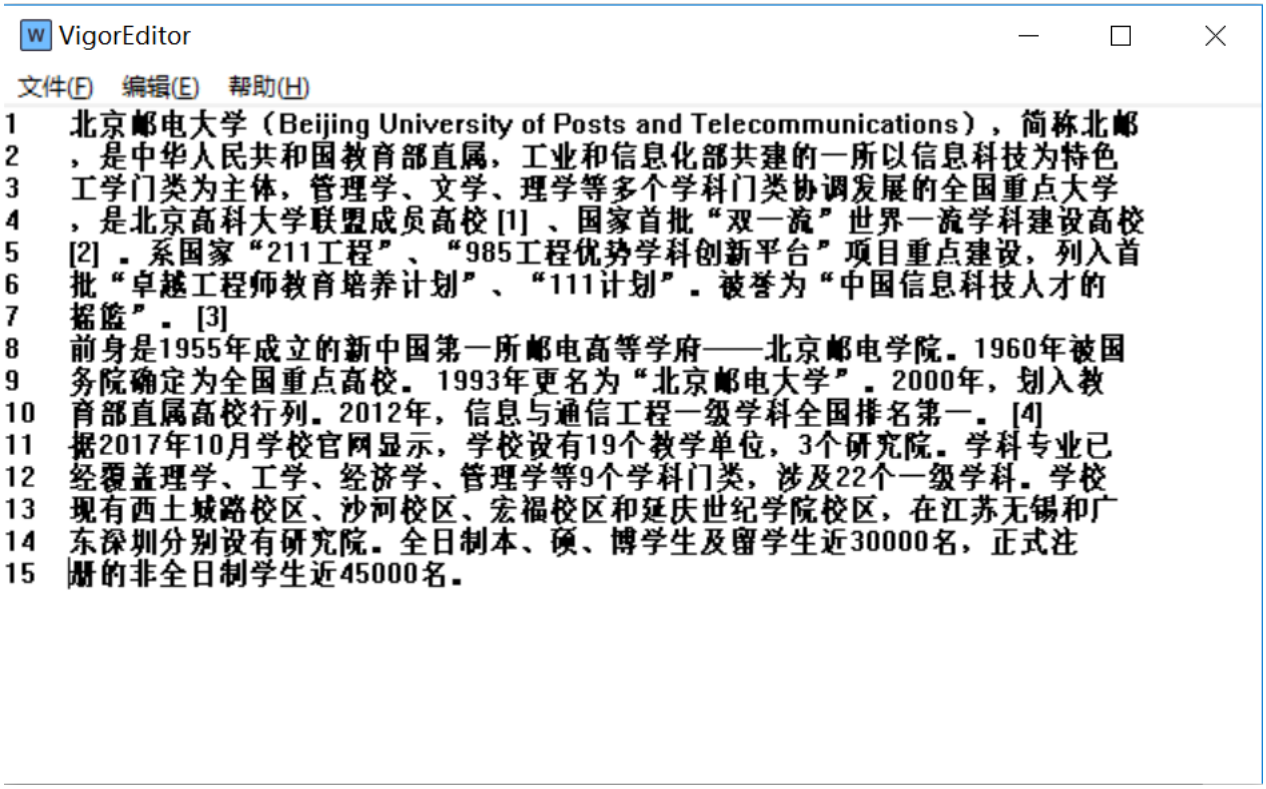


Figure 6.3

6.4 选中效果

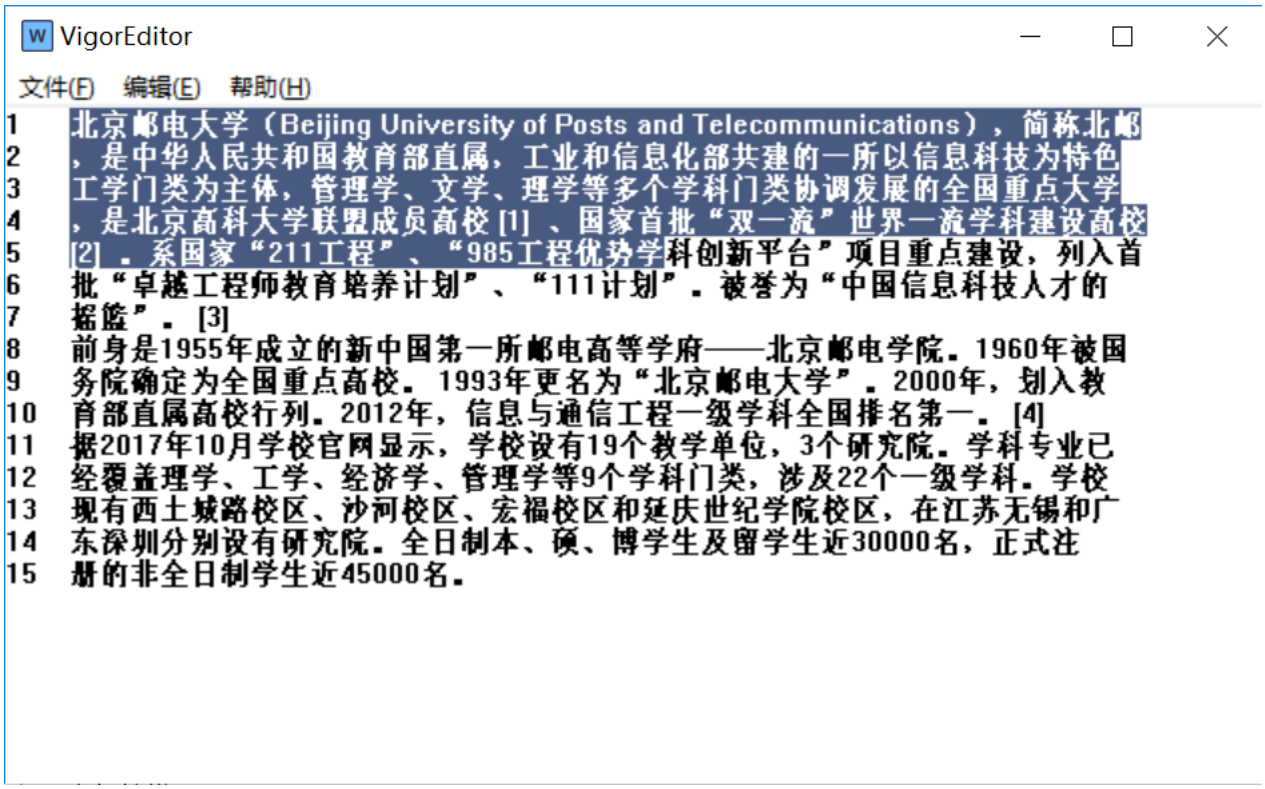


Figure 6.4

6.5 保存&另存为

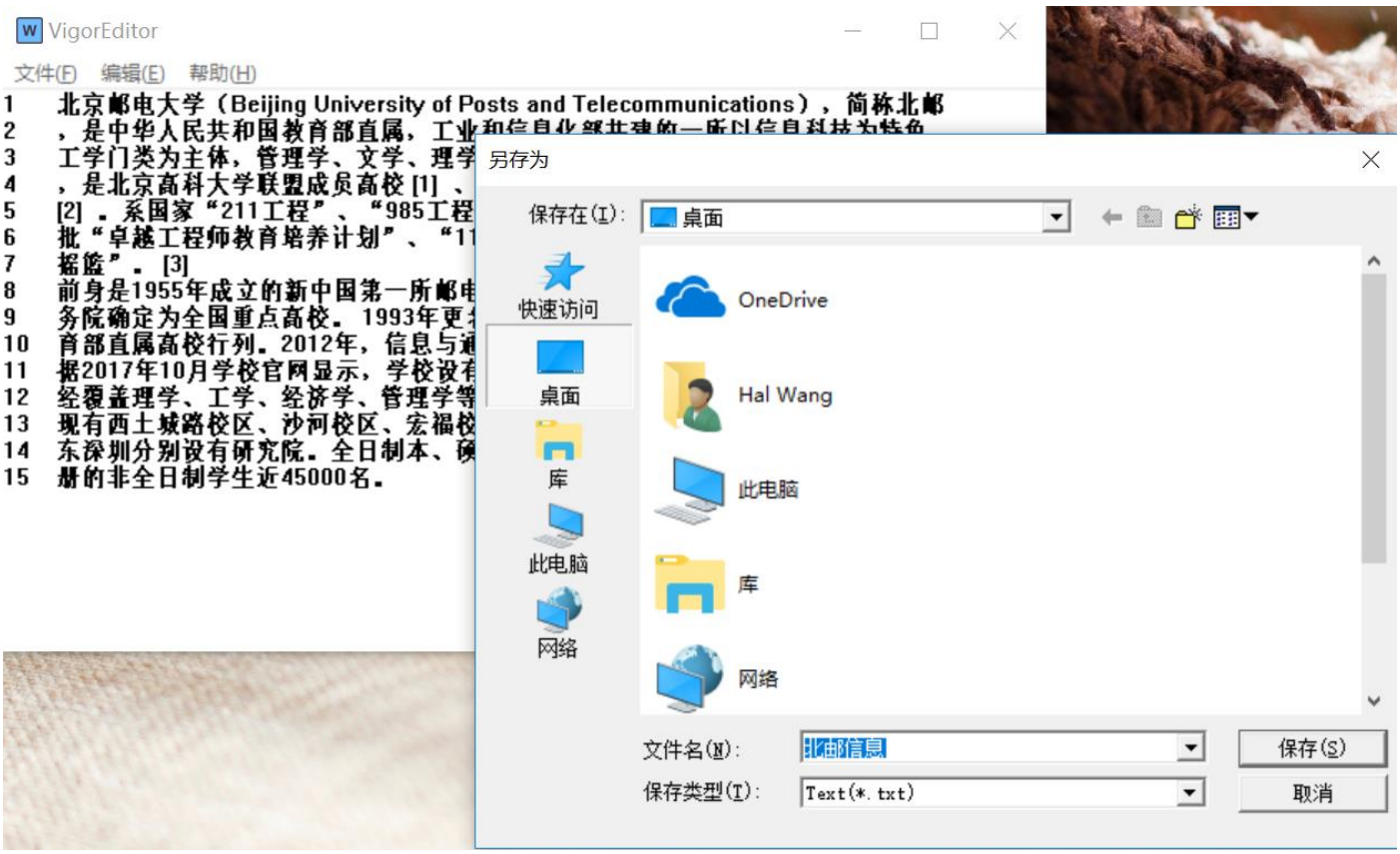


Figure 6.5.1

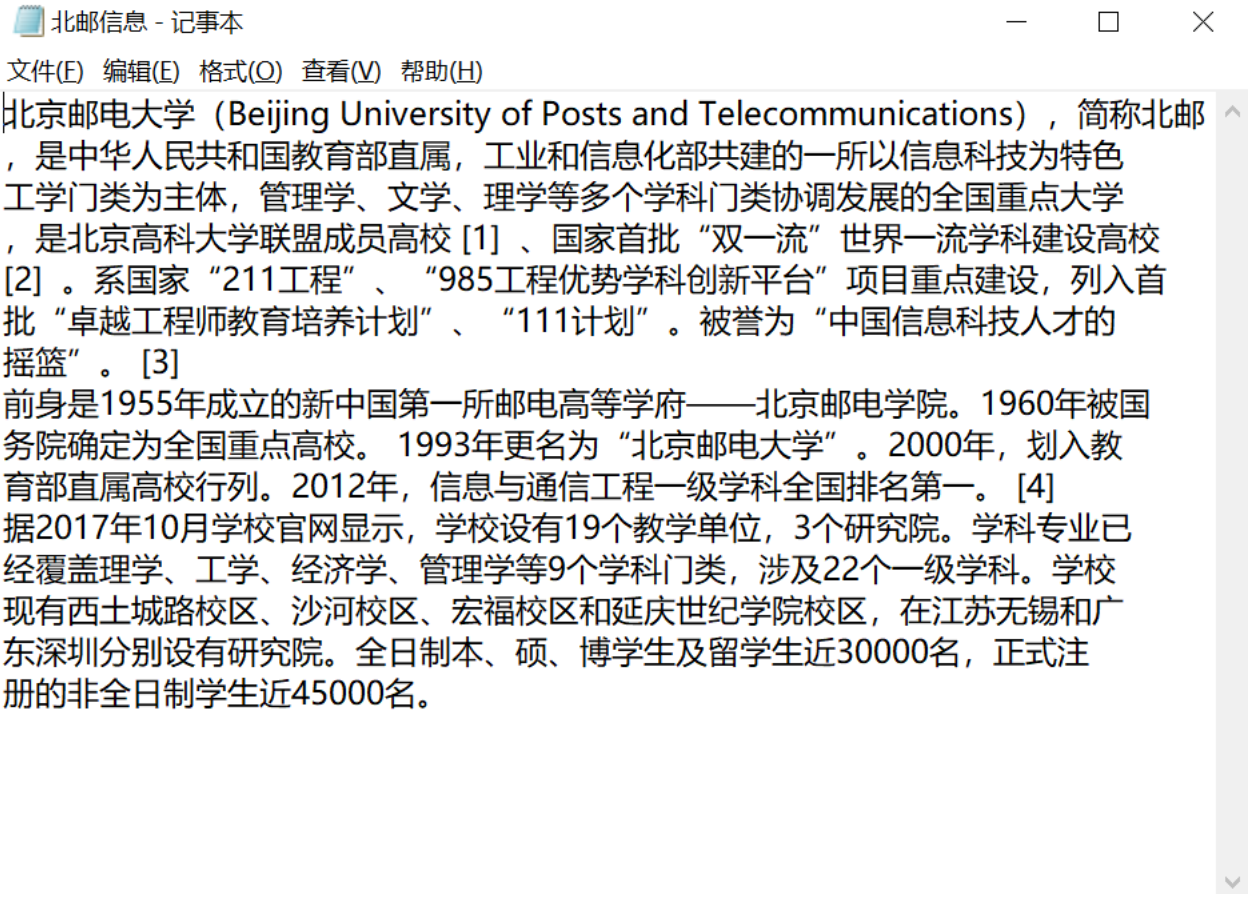


Figure 6.5.2

6.6 打开文件

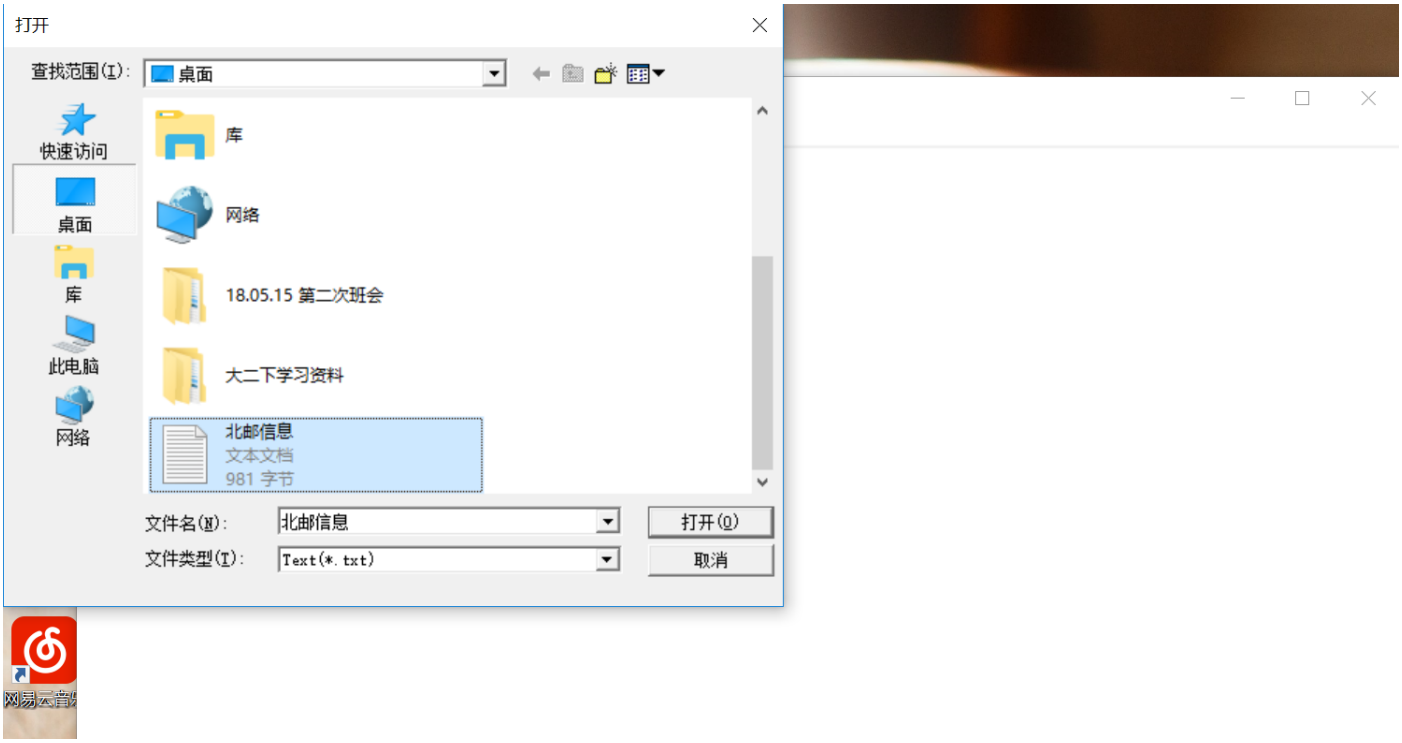


Figure 6.6

6.7 查找&替换

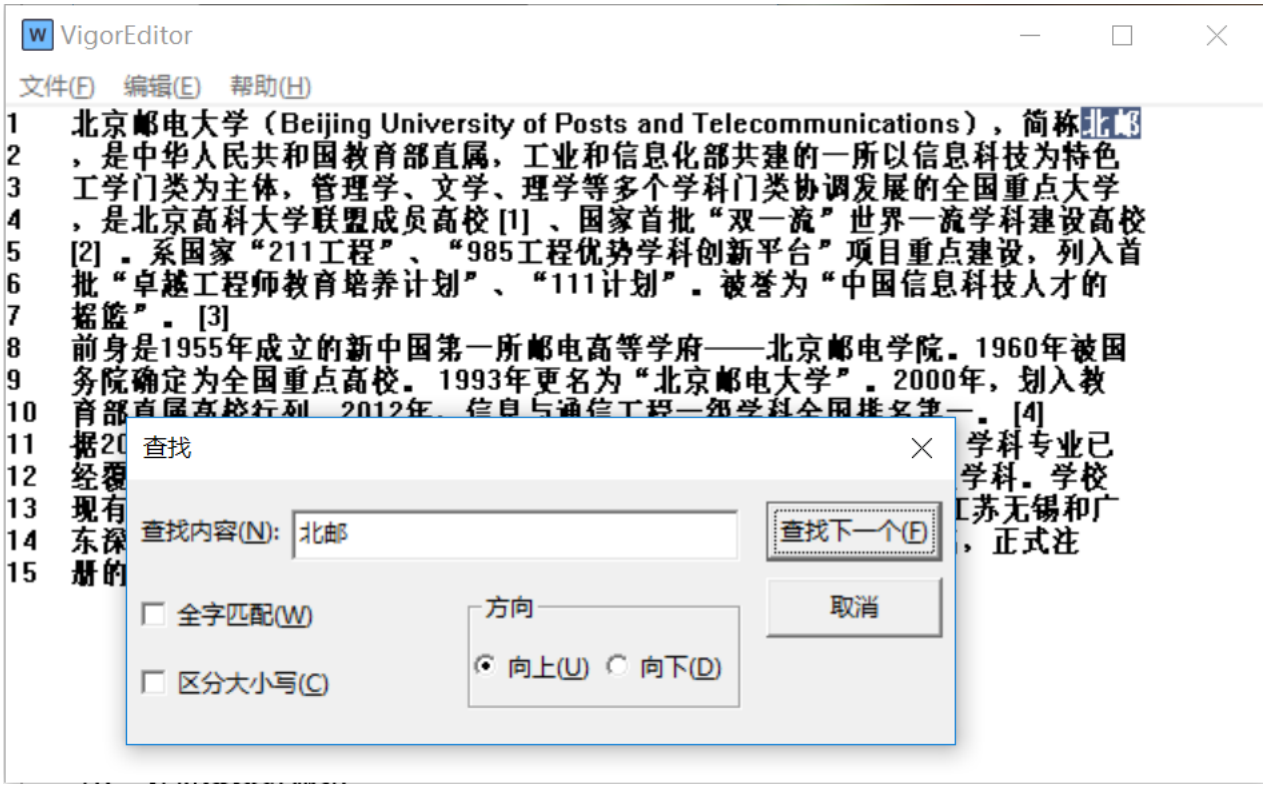


Figure 6.7.1

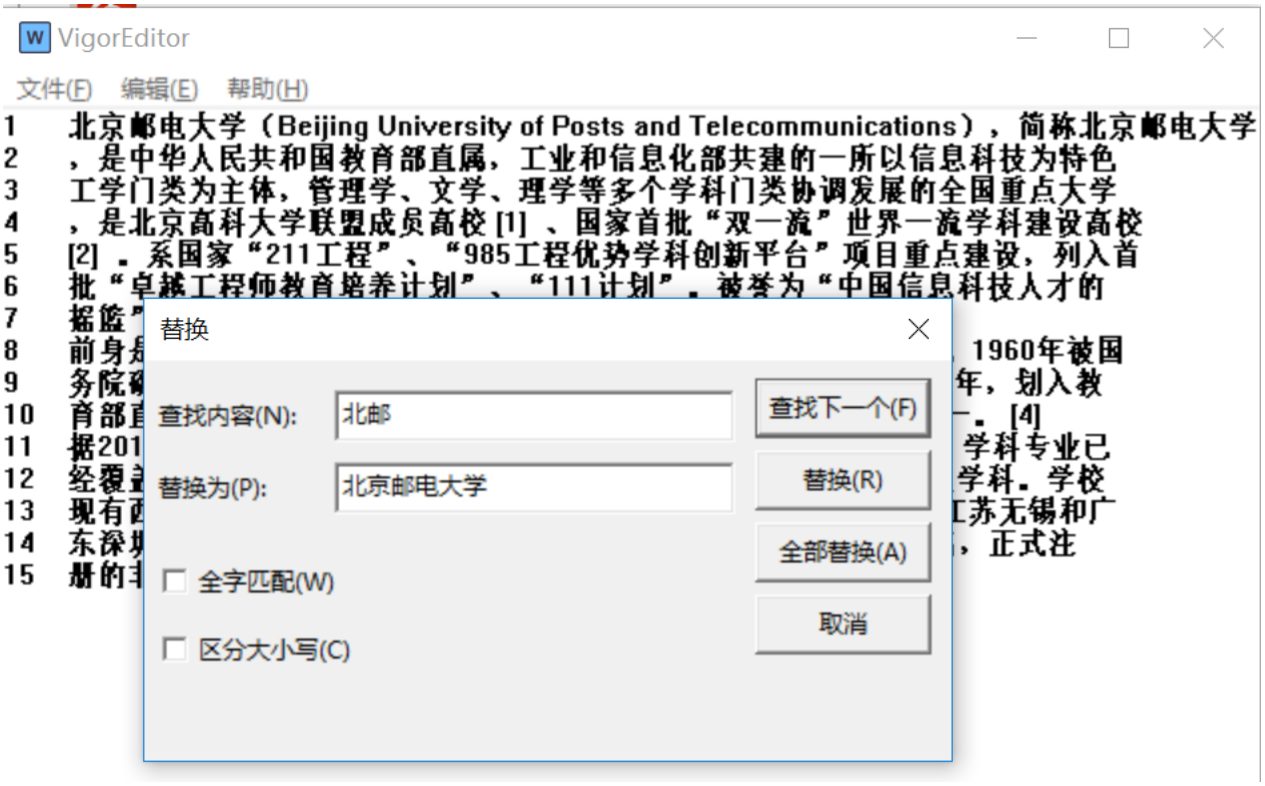


Figure 6.7.2

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。

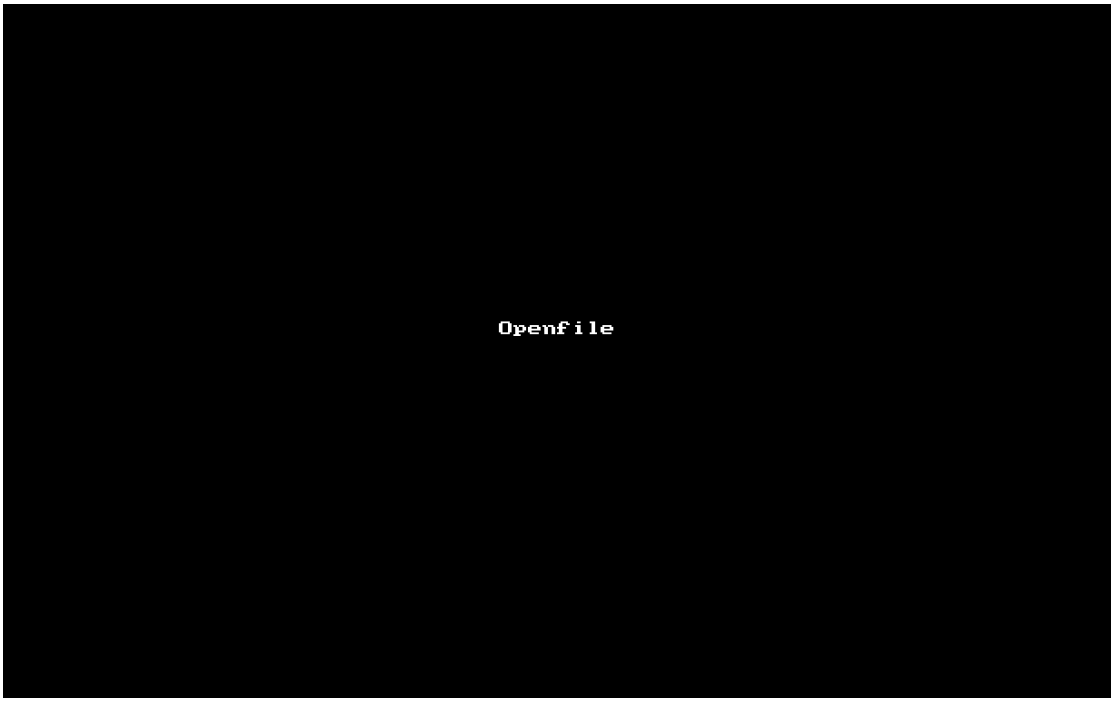
2.对 windows api 的了解不足，导致行号显示功能没能找到最佳解决方案。应该有更好的方式重绘，而现在是将行号与文本融为一体，在横向拖动滚动条时会产生 bug。

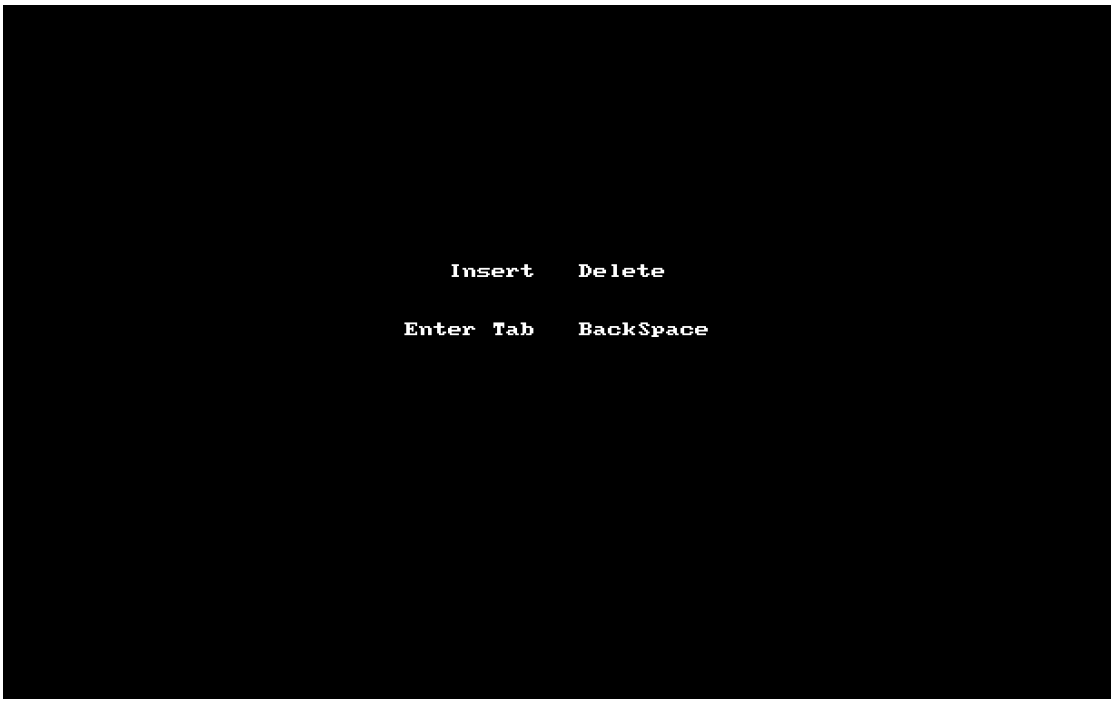
3.在项目初期对 git 了解不足，走了一些弯路。

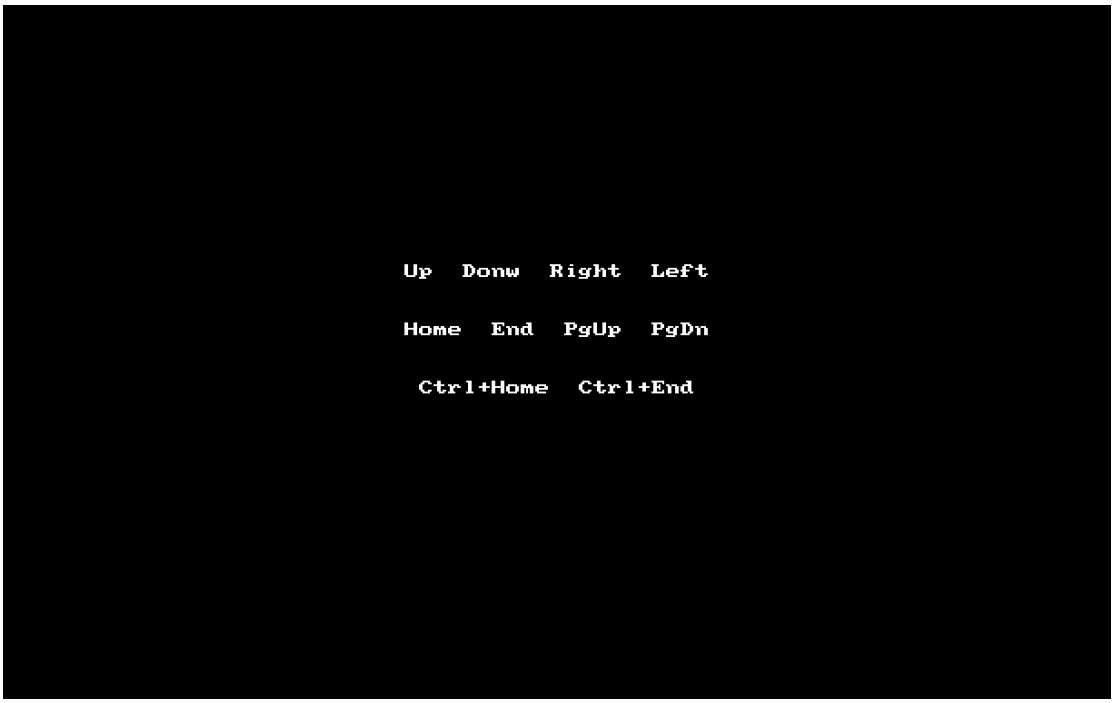
4.开发经验仍需加强，第一遍可谓是摸着石头过河。在虽然在现有的开发模式上实现了很多功能，达到了应有的效果，但是一些地方可能还可以更规范一些。(1）成员函数的设计上不得不说存在着冗余之感，同样的功能实现方式往往有多种。以下部分待优化：如可以将所有 line 成员函数放到 Article 中实现并将 line 作为参数进行操作。这样可以执行更轻松的执行多行操作，功能也更加的统一。
（2）功能扩展性还可以更强，可以给每个字单独创建一个类，字的颜色，大小，字体，宽度均可用一个位串来存储，同时改变所有字符串相关函数为对字类操作的方法。以此，进一步拓宽功能。
（3）显示渲染层与操作层写在了一起，虽然耦合度并不高，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。

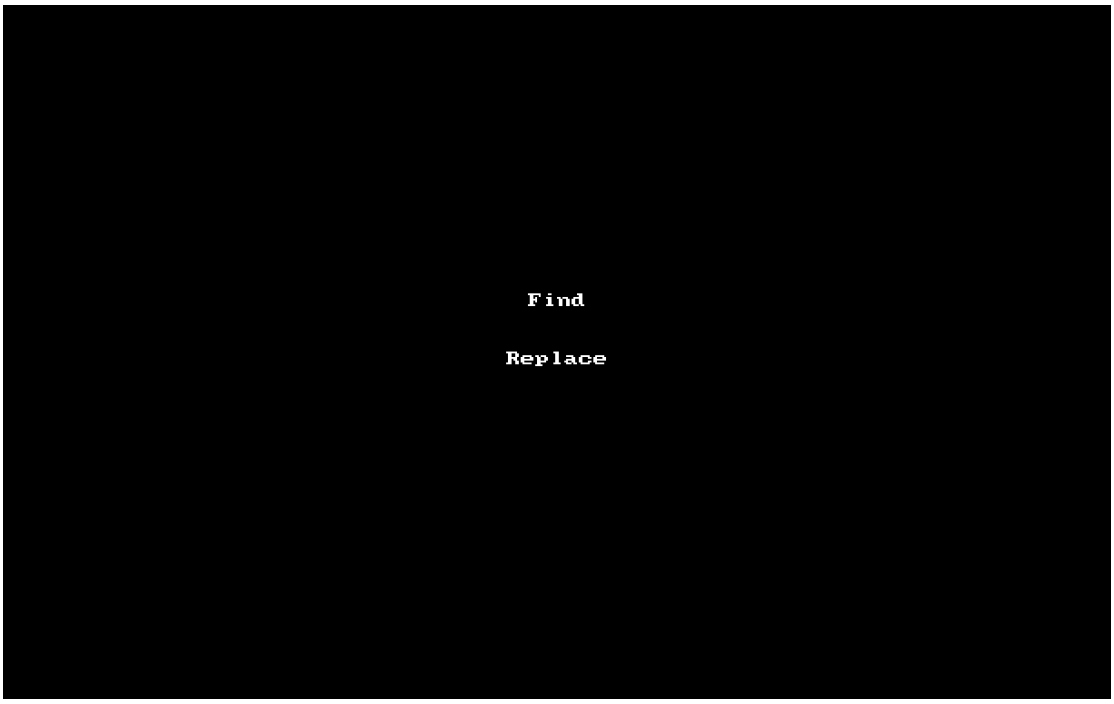
5.虽然操作丰富，但是界面略原始，如果有更长的时间，我们会愿意更好的完善界面。

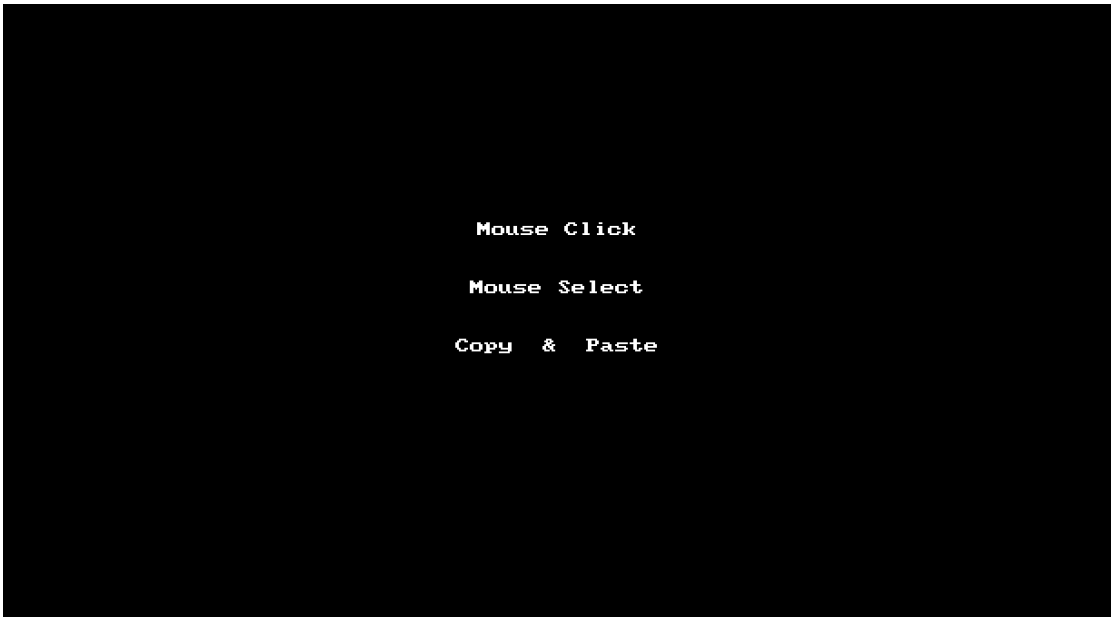
八、用户使用说明



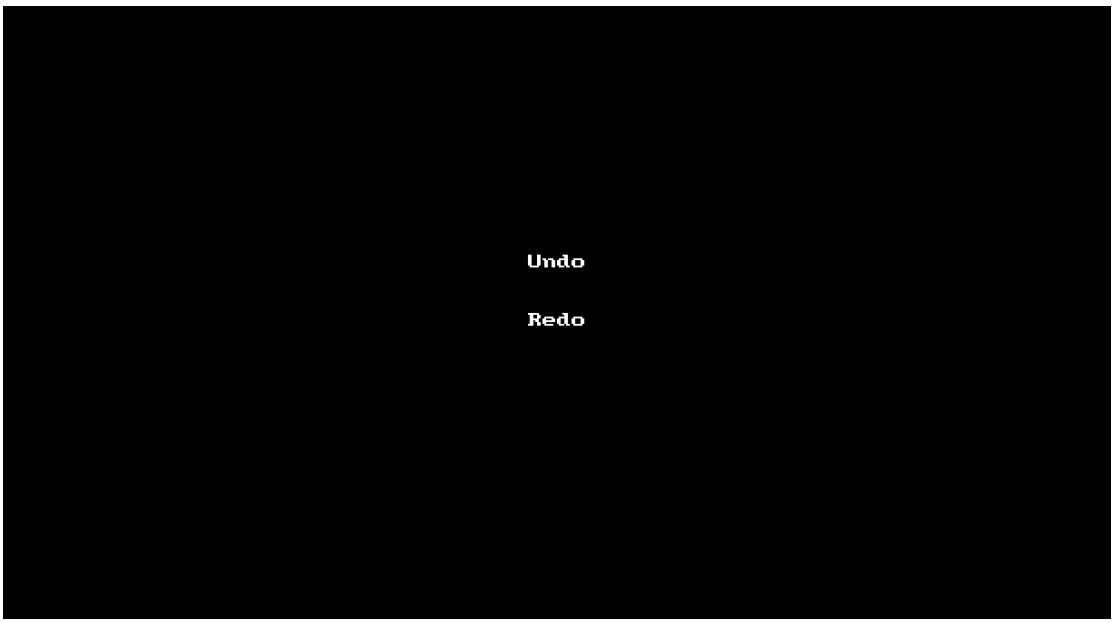

Insert Delete
Enter Tab BackSpace


Up Donw Right Left
Home End PgUp PgDn
Ctrl+Home Ctrl+End

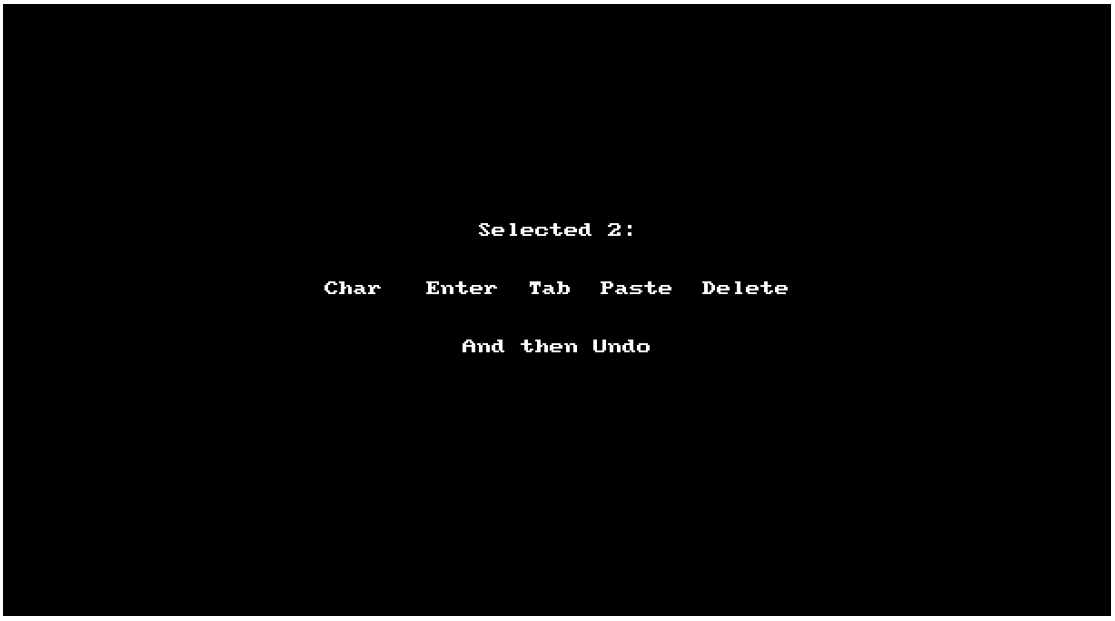

Find
Replace

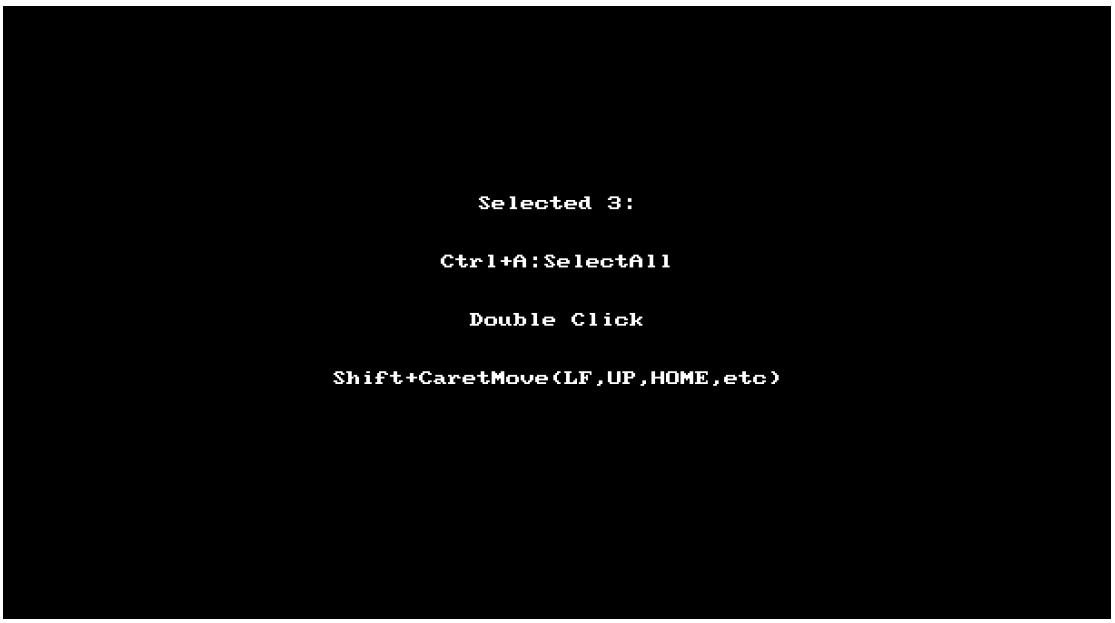

Mouse Click
Mouse Select
Copy & Paste


Scrolling

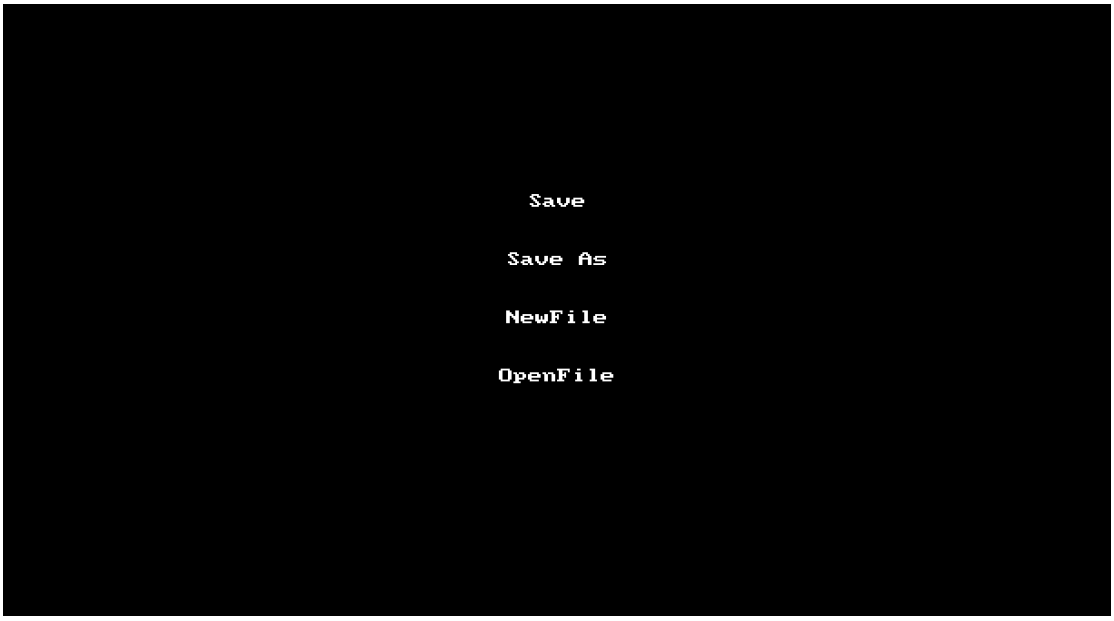

Undo
Redo

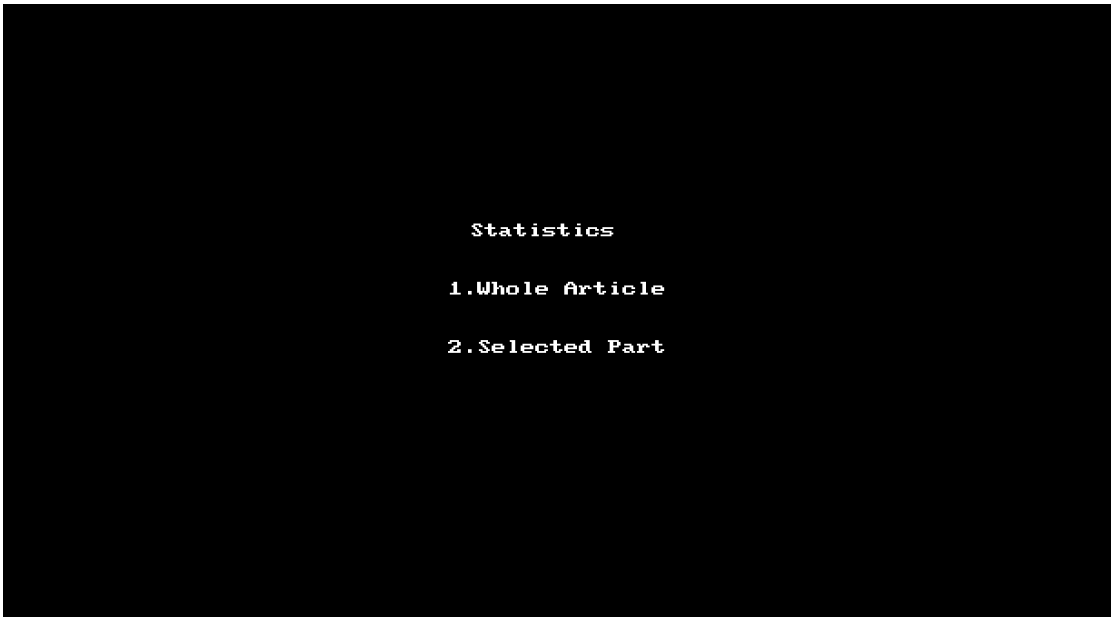

Shift select
Selected CaretMove


Selected 2:
Char Enter Tab Paste Delete
And then Undo


Selected 3:
Ctrl+A:SelectAll
Double Click
Shift+CaretMove(LF,UP,HOME,etc)


Right Botton
Menu





九、团队合作

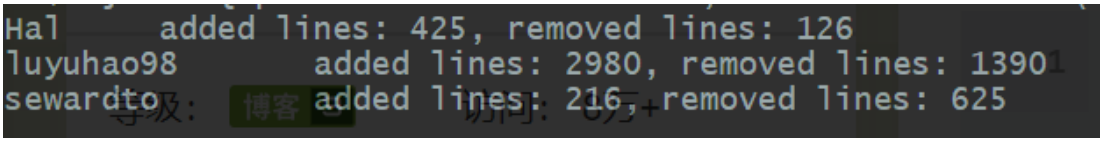
9.1 综述

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



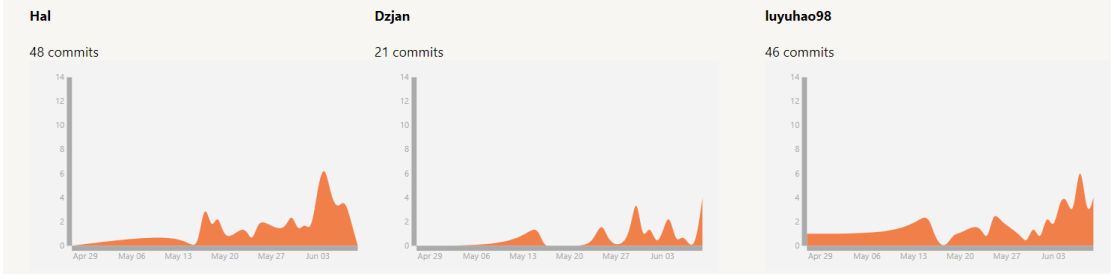


Figure 9.2.3


Miniword
任务
代码
代码浏览
分支管理
发布管理
版本对比
合并请求
项目网络
2ada201 彻底修改VM_MASK 中所有'tmpL' 为 'ttmpL'
luyuhao98
85f3eb1 Before Merge
luyuhao98
673d600 修改选中后光标移动操作的hdc问题
Hal
75bb8e5 shift+方向键 左右完成
Hal
569487a 完成shift+左右键 选中 光标显示存在bug
sewardto
ac4beca 将WM_MASK中的tmpL改为ttmpL
luyuhao98
3bdadc0 统计字数 Ctrl+M
Hal
f2dde3f 加背景测试中

Figure 9.2.4