实习2 图形基本属性与区域填充

**1、实验目的**

**1）编程实现图形基本属性控制，线宽、线色、线型，字大、字小、字色、字体等；**

**2）给定多边形，实现区域填充算法模拟。**

**2、实验内容**

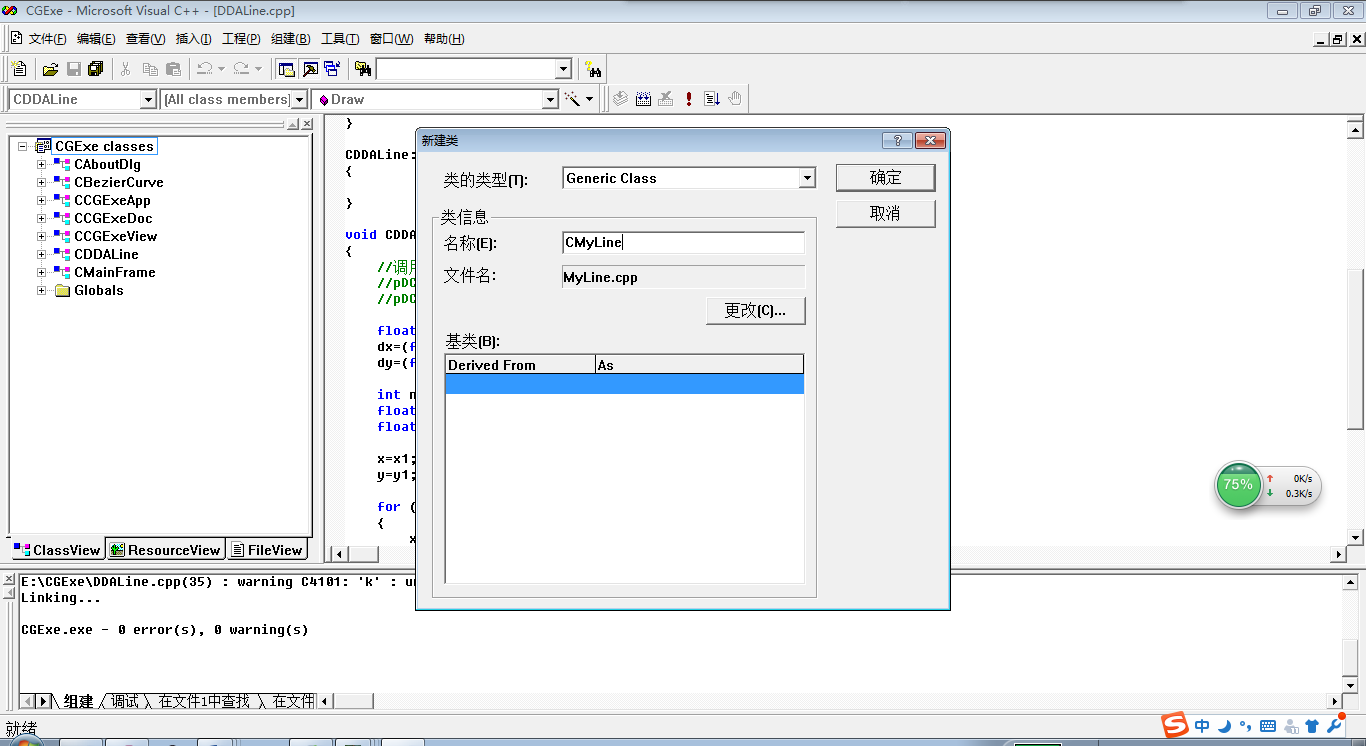
1）对给定的线段，实现线宽、线色、线型等显示；

2）对给定的字符，实现字体、字大字小、字色显示；

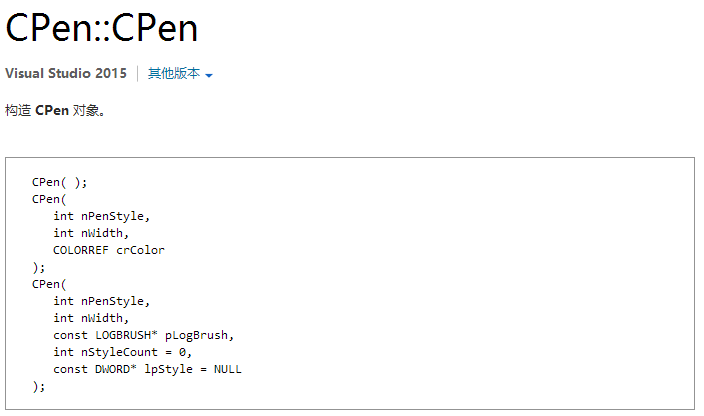
3）给定多边形边界及颜色，实现区域填充算法。

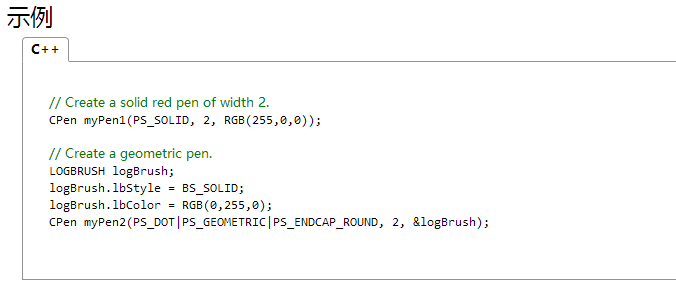
具体步骤

1、在实习1工程CGExe中，添加CMyLine类，**用以实现线状要素属性测试**（VC中，通过CPen类实现线宽、线色、线型）；



以下是CPen构造函数，参数取值或更多细节，自己Baidu。





2、添加涉及线状绘制的成员变量：( 参照第一次实习添加成员变量方法 )

CPoint pt1，pt2; //起点、终点坐标（VC中定义的点类对象）

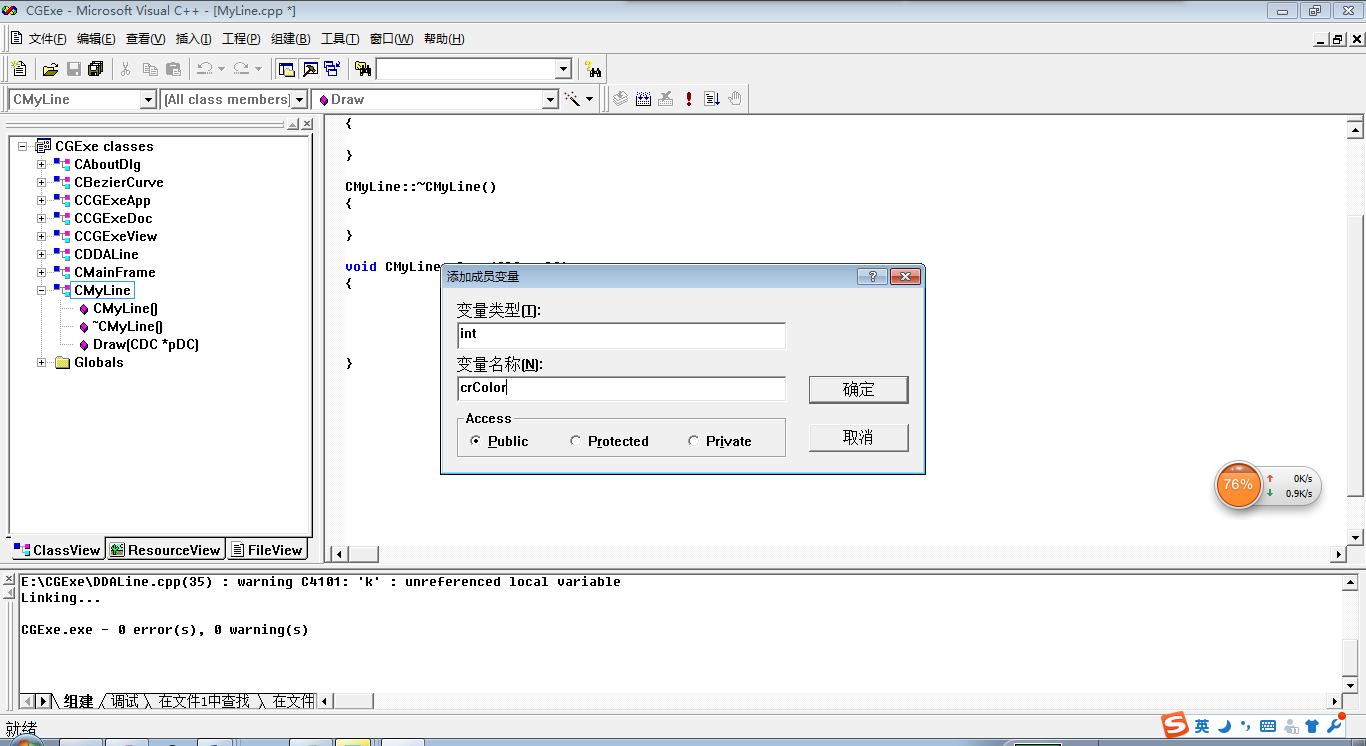
或者 int x1,y1,x2,y2; //起点、终点坐标

int crColor；//线的颜色 COLORREF内部也是通过一个int定义的，所以通用

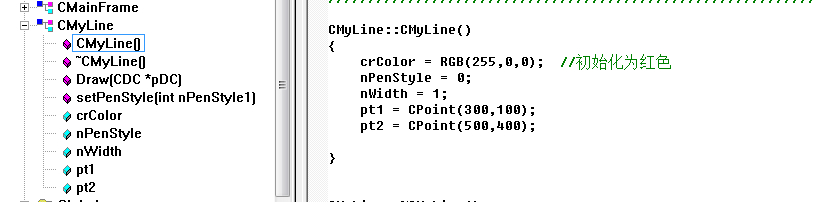
或者 COLORREF color;

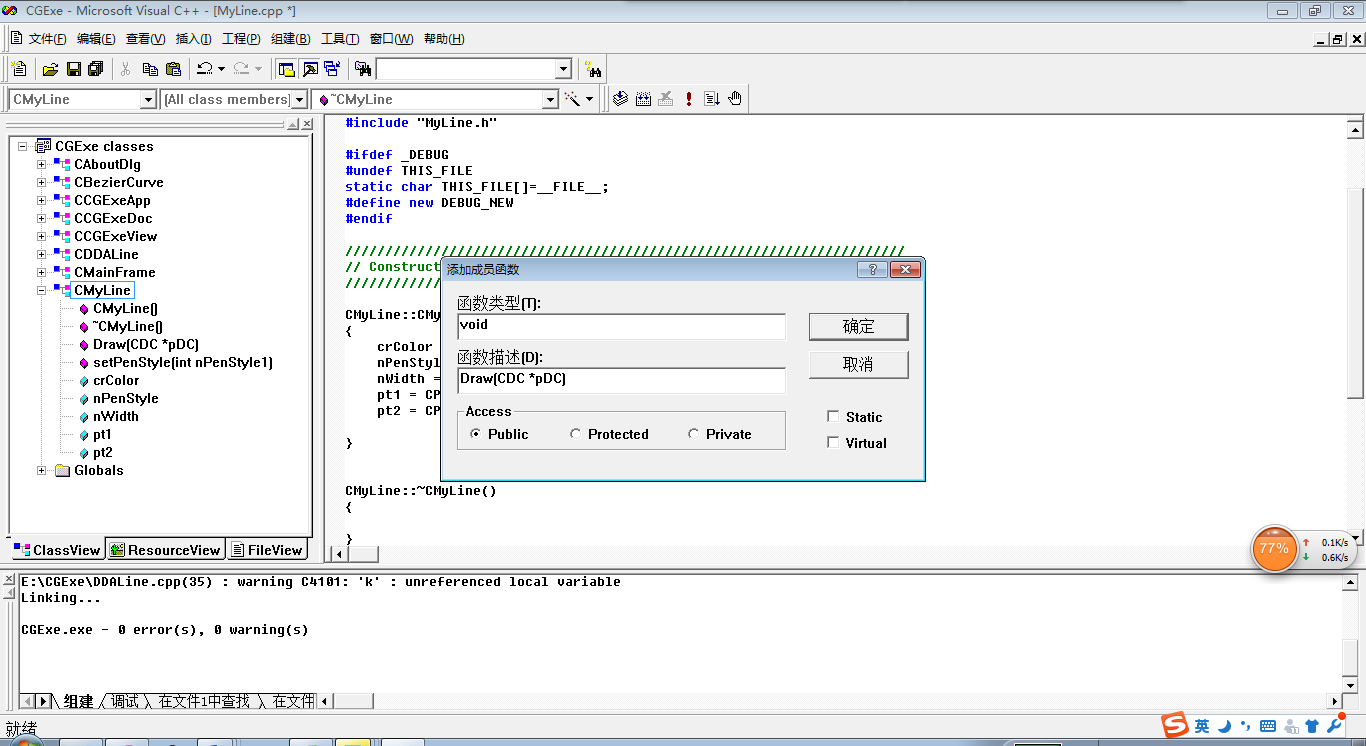
int nPenStyle; //线型

int nWidth; //线宽

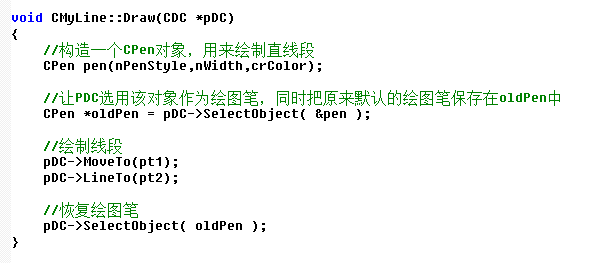


注意在构造函数中对其初始化；

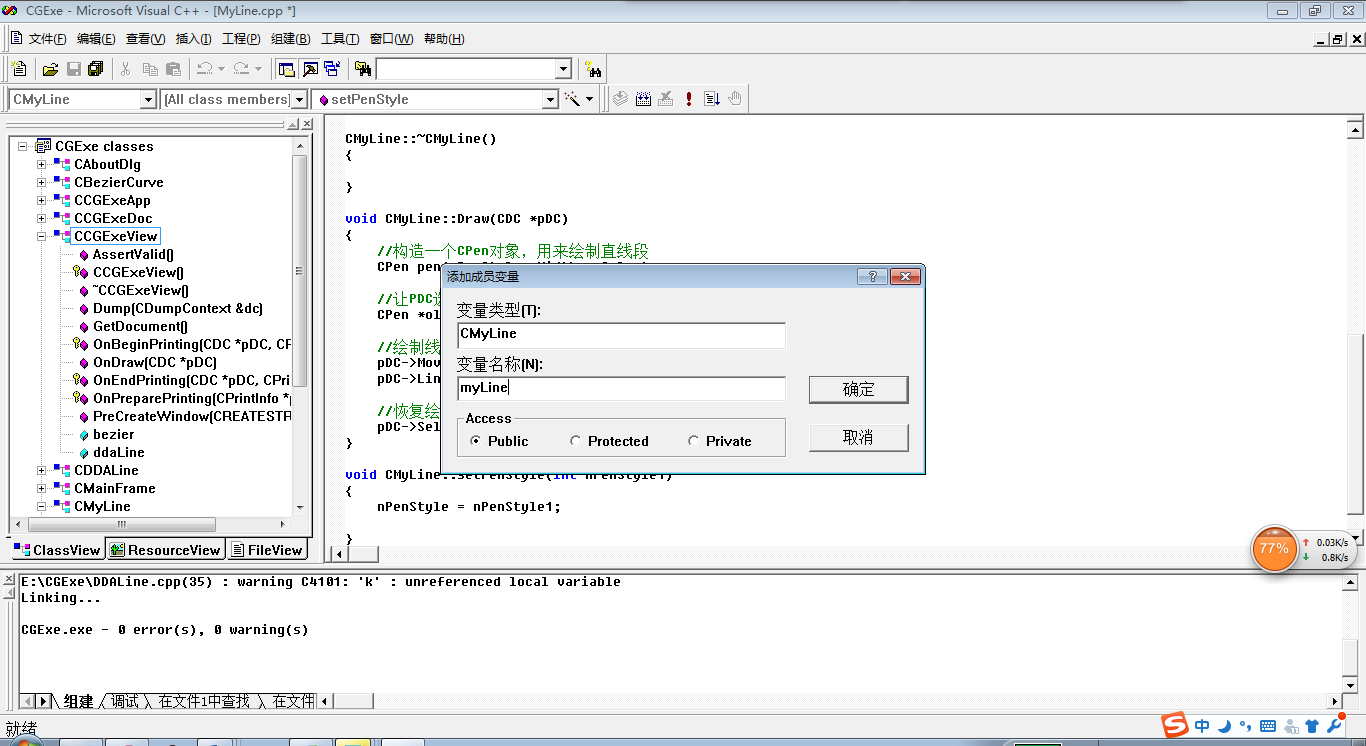
3、添加绘制数据成员函数Draw(CDC \*pDC) ( 参照第一次实习添加成员方法 )



并完成绘制方法：



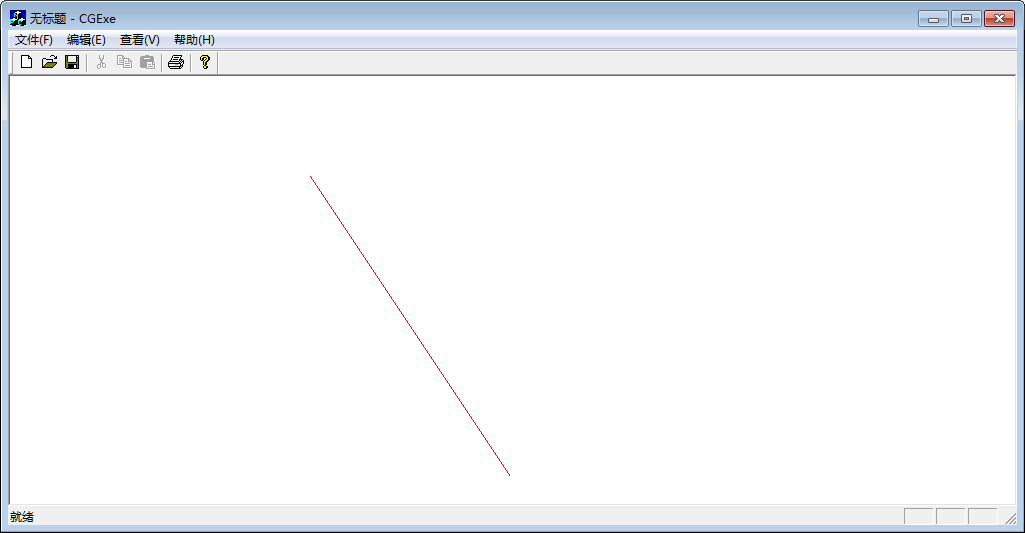
4、在CEXECGView类中添加CMyLine对象myLine；



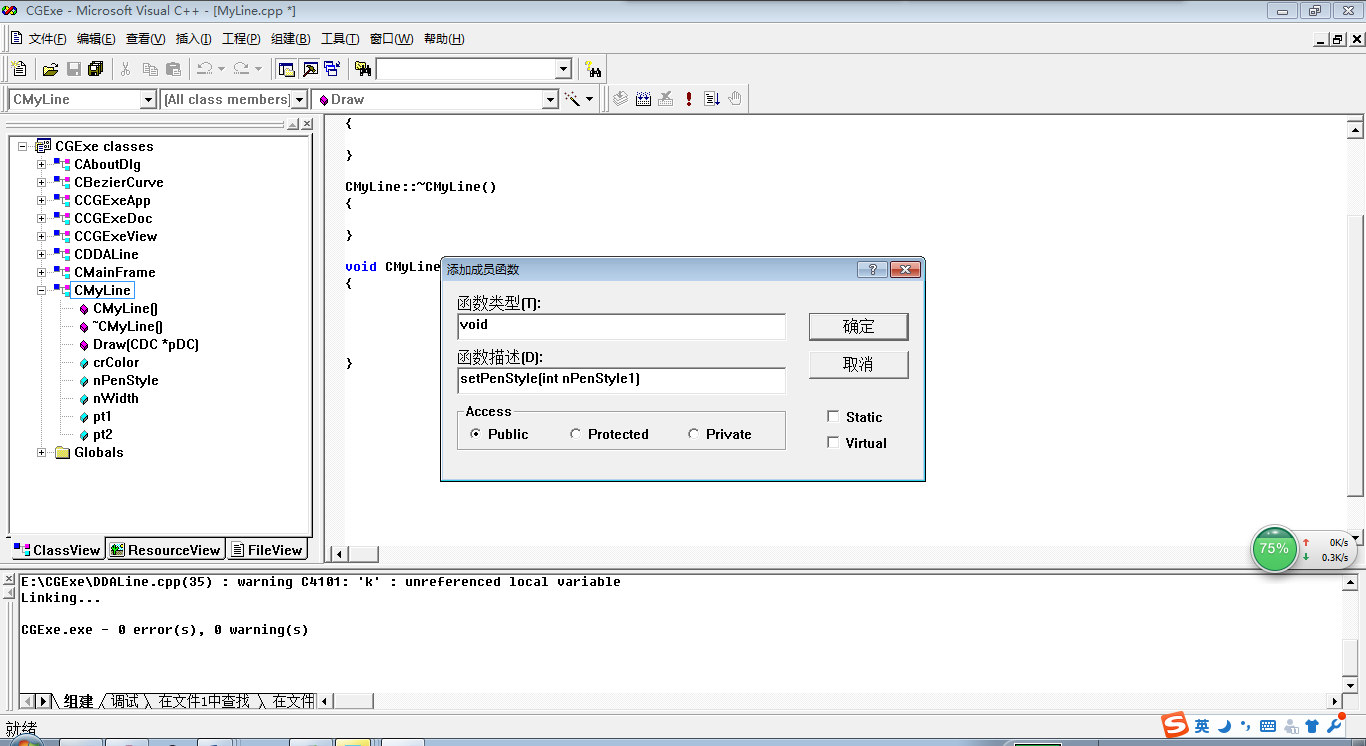
并在CEXECGView的onDraw中调用myLine的绘制：

myLine.Draw(pDC);

编译运行，可以看到：



5、添加对成员变量复制的成员函数：( 参照第一次实习添加成员方法 )



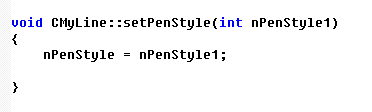
void setPenStyle(int nPenStyle1);

void setWidth(int nWidth);

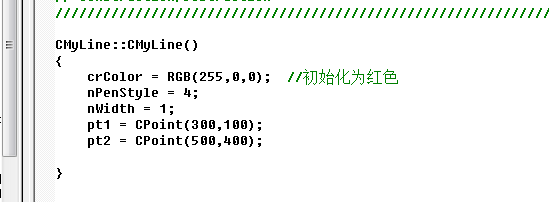
void setPoints(CPoint pta,CPoint ptb); 或者 void setPoints(int x1,int y1,int x2,int y2);

void setColor(int nColor);

**并完成对应的赋值**，如：



5、修改绘制线的色彩、线型、线宽即可得到不同显示结果；



（注意不同线型、线宽配合使用）

6、可以参考VC编程，进一步在CEXECGView中添加消息机制，实现从键盘输入数据，对线型、线宽、颜色等调整，并实现图形绘制；

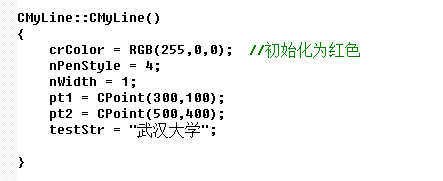
**文字部分：**

7、在已经添加的CMyLine类中，添加文字绘制部分对应成员变量和成员函数；

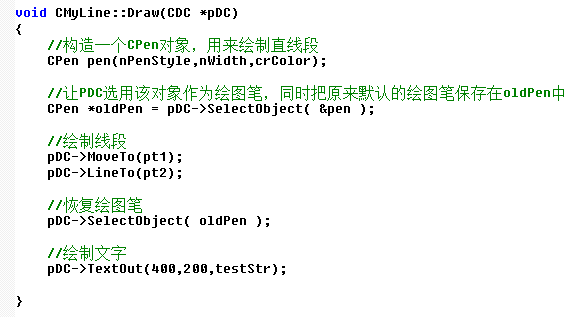
CString testStr;

在CMyLine构造函数中，初始化：

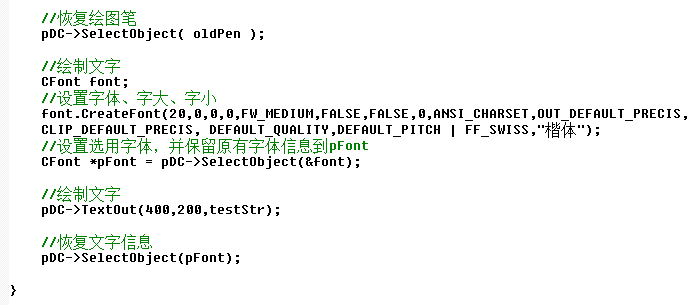
testStr = “武汉大学”;

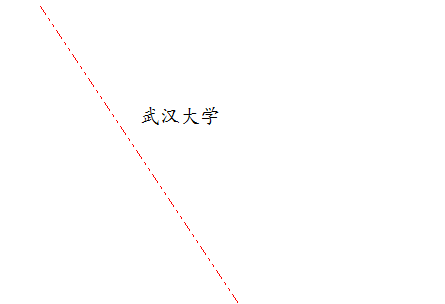
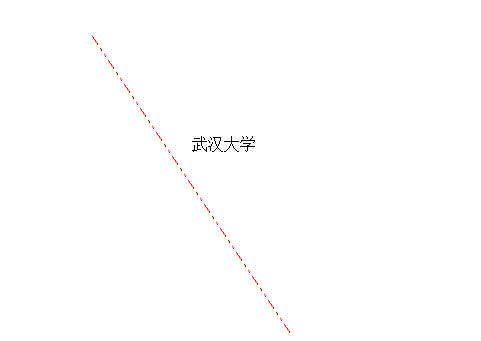


8、在CMyLine的Draw函数中，添加绘制文本信息部分；



9、扩展字体部分，使其可以考虑字体及字大小等；





10、自己调整字体信息，以显示不同样式；

VC字体由CFont类进行管理，创建CFont类必须使用CFont类的成员函数，这点与CPen和CBrush有所不同。

VC常用两种方法创建字体，其一用CFont类的成员函数CreateFont直接创建逻辑字体，其二利用LOGFONT结构，将该结构作为参数调用CFont类的成员函数CreateFontIndirect创建逻辑字体。

**第一种：CreateFont（实例如上），具体内容及参数，请Baidu**

BOOL CreateFont(

int nHeight, //字体的高度

int nWidth, //字体的宽度

int nEscapement, //字体显示的角度

int nOrientation, //字体的角度

int nWeight, //字体的磅数

BYTE bItalic, //斜体字体

BYTE bUnderline, //带下划线的字体

BYTE cStrikeOut, //带删除线的字体

BYTE nCharSet, //所需的字符集

BYTE nOutPrecision, //输出的精度

BYTE nClipPrecision, //裁减的精度

BYTE nQuality, //逻辑字体与输出设备的实际

//字体之间的精度

BYTE nPitchAndFamily, //字体间距和字体集

LPCTSTR lpszFacename //字体名称

);

**第二种：利用**LOGFONT**结构体，具体内容及参数，请Baidu**

typedef struct tagLOGFONT { // lf

LONG lfHeight; //字体的高度

LONG lfWidth; //字体的宽度

LONG lfEscapement; //字体显示的角度

LONG lfOrientation; //字体的角度

LONG lfWeight; //字体的磅数

BYTE lfItalic; //斜体字体

BYTE lfUnderline; //带下划线的字体

BYTE lfStrikeOut; //带删除线的字体

BYTE lfCharSet; //所需的字符集

BYTE lfOutPrecision; //输出的精度

BYTE lfClipPrecision; //裁减的精度

BYTE lfQuality; //逻辑字体与输出设备的

//实际字体之间的精度

BYTE lfPitchAndFamily; //字体间距和字体集

TCHAR lfFaceName[LF\_FACESIZE]; //字体名称

} LOGFONT;

具体实例：

LOGFONT lf; //定义字体结构

lf.lfWeight=10; //字体磅数=10

lf.lfHeight=56; //字体高度(旋转后的字体宽度)=56

lf.lfWidth=20; //字体宽度(旋转后的字体高度)=20

lf.lfUnderline=FALSE; //无下划线

lf.lfStrikeOut=FALSE; //无删除线

lf.lfItalic=FALSE; //非斜体

lf.lfEscapement=0; //字体显示角度=0°

lf.lfCharSet=DEFAULT\_CHARSET; //使用缺省字符集

strcpy(lf.lfFaceName,"楷体"); //字体名=楷体

CFont font; //定义字体对象

font.CreateFontIndirect(&lf); //创建逻辑字体

CFont \*pFont=pDC->SelectObject(&font);//选入设备描述表

**实现区域填充算法**

主要思路，给定多边形坐标信息，要求按照某种填充算法，实现多边形的填充。

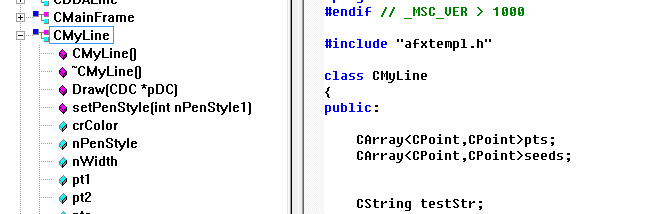
11、在CMyLine中添加用来存放坐标数据的动态数组及实现栈操作的动态数组：

**注意：这是VC模板类，所以只能手动添加对应的对象，同时需要添加头文件**

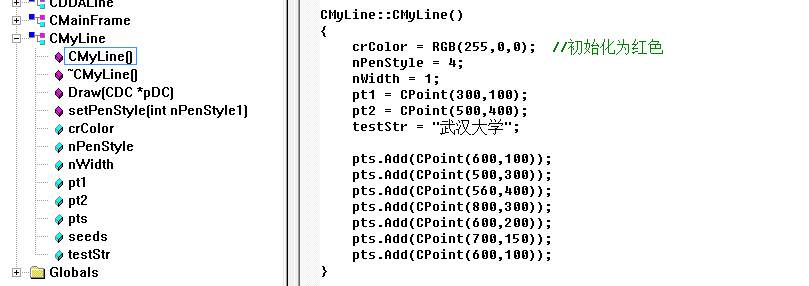
头文件： #include “afxtempl.h”

CArray<CPoint,CPoint>pts; //多边形坐标序列

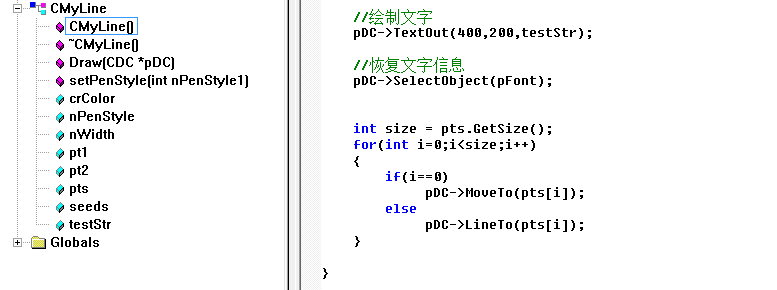
CArray<CPoint,CPoint>seeds; //所有需要填充的坐标点

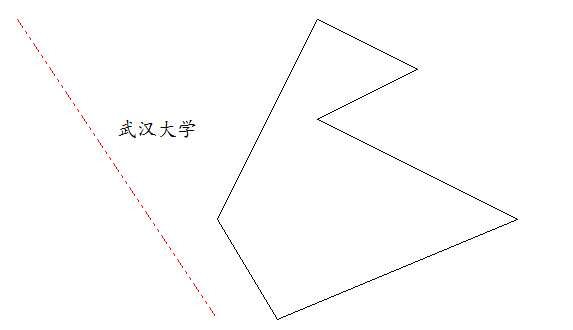


12、在CMyLine构造函数中，对pts进行添加数据点操作，注意保障数据为封闭多边形，且不要出现自相交；



13、在CMyLine中添加绘制多边形方法；





**14、实现区域填充算法**

添加一个函数FillPolygon（CDC \*pDC,CPoint pt,int Color）,以给定点为种子，实现连续区域（有可能多边形内部，也有可能外部）用给定色彩(color)填充；

注意：该类中绘制，只能包含一个函数，即pDC->SetPixel;

//确定种子点pt周围8方向判断相邻点是否落在多边形内；

//如果落在多边形内，添加到种子点队列，否则跳过，直到一个点周围所有点判断完成；

//种子队列取出最后一个点，先填充，再判断周围8方向是否落在多边形内...

//循环到最后，种子队列没有种子点，即结束！

15、调用该函数，实现调用。

