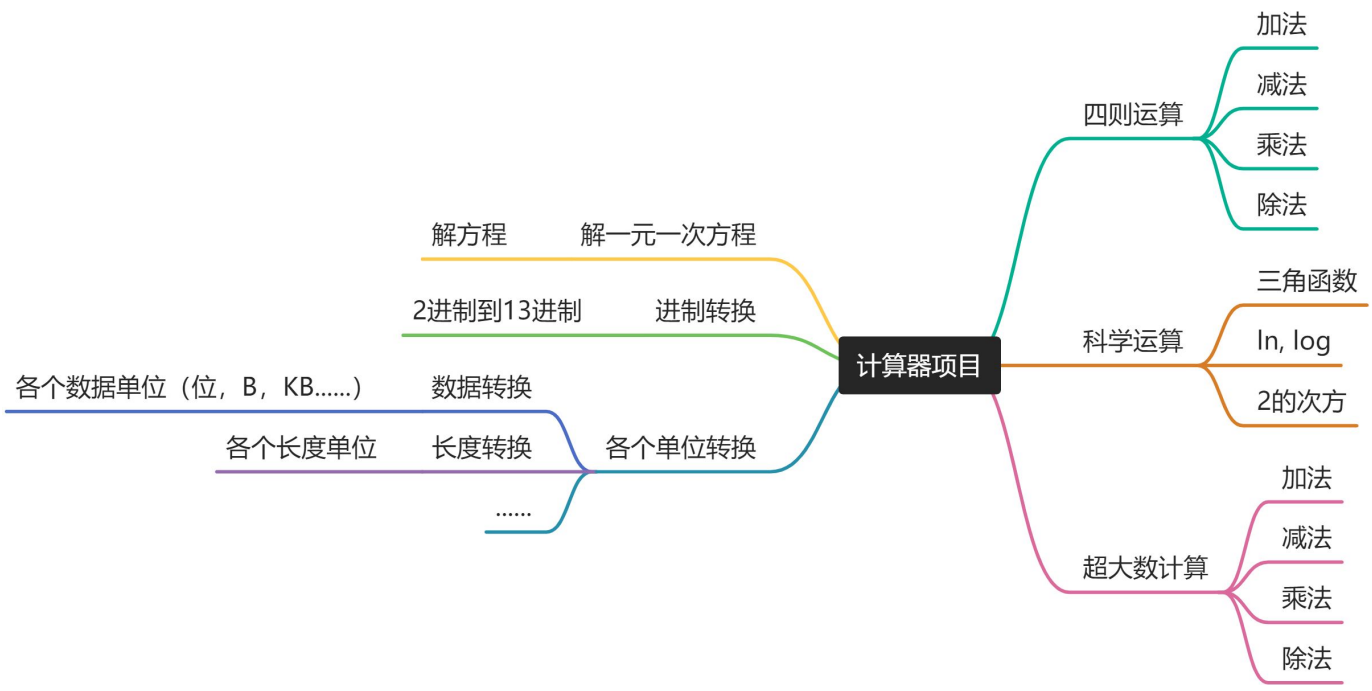
计算器代码报告 ——231275040 林方恒

1. 程序框架图



2. 核心函数

2.1. 栈的实现

`typedef TChar T;`

注：当没有定义 `_UNICODE` 宏时， `TCHAR = char, _tcslen =strlen` ,

当定义 `_UNICODE` 宏时， `TCHAR = wchar_t ， _tcslen = wcslen[1]`

`void push(T c)`

功能：入栈

形参： `c` 要入栈的数

`void pop();`

功能：出栈，移除栈顶元素

`T top();`

功能：取栈顶元素

返回值：栈顶元素

作用：返回栈顶元素，不对栈作出任何修改

`bool empty();`

功能：判断栈是否为空

返回值：栈是否为空

作用：返回栈是否为空，不对栈作出任何修改

`void clearstack();`

功能：清空栈

作用：移除栈中的所有元素

2.2. 标准 standard

// Part 1：计算函数

`void change();`

功能：中缀表达式转换为后缀表达式，利用栈实现中缀表达式到后缀表达式的转换

`void calculate();`

功能：计算后缀表达式的值，利用栈实现后缀表达式的计算

`void updatewithinput_();`

功能：普通计算，与输入有关的更新，更新输入的数据

// Part 2：图形化显示相关

`void showscreen_();`

功能：普通计算，显示函数，显示输入的数据

`void screenmovehd();`

功能：字符串滑动动画

`float getfps();`

功能：计算 fps，计算帧率

返回值：fps

2.3. 科学计算 science

`void push(int &n, int &m, int k);`

功能：根据参数n和m的值进行判断和操作，然后将n赋值为m。

形参：

n (int&): 一个整数的引用，表示当前的状态或操作数

m (int&): 一个整数的引用，表示要转移到的状态或操作数

k (int): 默认值为0，未使用(可能用来重载？)

`void updatewithinput();`

功能：处理与鼠标输入相关的更新操作，包括按键的状态切换、数字的输入、运算符的选择等。

`void showscreen();`

功能：绘制了科学计算器的界面，包括数字按钮、运算符按钮以及其他功能按钮，同时显示计算结果。

2.4. 超大数计算 bignumber

`void updatewithinput1();`

功能： 根据用户的输入更新超大数运算模块中的数据，并处理用户点击的操作

`void showscreen1();`

功能：显示超大数运算界面

`void memsenumber();`

功能：初始化超大数运算模块相关数据

```
MyChar *numbersubx(MyChar *ss, MyChar *ss1);
```

功能：超大数减法

形参：ss 被减数(int), ss1 减数(int)

返回值： 结果字符串

```
MyChar *numbersub(MyChar *ss, MyChar *ss1);
```

功能：超大数减法

形参：ss 被减数(int + fraction), ss1 减数(int + fraction)

返回值：结果字符串

```
MyChar *numberadd(MyChar *ss, MyChar *ss1);
```

功能：超大数加法

形参：ss 加数(int), ss1 被加数(int)

返回值：结果字符串

```
MyChar *numberaddx(MyChar *ss, MyChar *ss1);
```

功能：超大数加法

形参：ss 加数(int + fraction), ss1 被加数(int + fraction)

返回值：结果字符串

```
void bignumberadd();
```

功能：超大数加法

```
void bignumbersubtract();
```

功能：超大数减法

```
void bignumbermuit();
```

功能：超大数乘法

3. easyx相关函数

```
1. HWND initgraph(int width, int height, int flag = 0);
```

`width`：绘图窗口的宽度。

`height`：绘图窗口的高度。

`flag`：绘图窗口的样式，默认为 NULL。可为以下值：

值	含义
`EX_DBLCLKS`	在绘图窗口中支持鼠标双击事件。
`EX_NOCLOSE`	禁用绘图窗口的关闭按钮。
`EX_NOMINIMIZE`	禁用绘图窗口的最小化按钮。
`EX_SHOWCONSOLE`	显示控制台窗口。

返回新建绘图窗口的句柄

> **句柄**（handle）是Windows操作系统用来标识被应用程序所建立或使用的对象的整数。其本质相当于带有引用计数的智能指针。当一个应用程序要引用其他系统（如数据库、操作系统）所管理的内存块或对象时，可以使用句柄。
>
> `(2.3.10) DEFINITION. A handle of any sentential form is a leftmost simple phrase.`

2. `HWND GetHWnd();  // Get the handle of the graphics window`

在 Windows 下，句柄是一个窗口的标识，得到句柄后，可以使用 Windows API 中的函数实现对窗口的控制。

e.g.1

```
// 获得窗口句柄
HWND hWnd = GetHWnd();
// 使用 Windows API 修改窗口名称
SetWindowText(hWnd, "Hello!");
```

e.g.2

`SetWindowText(GetHWnd(), _T("计算器")); // 标题`

通过 `GetHWnd()` 函数获得当前窗口的句柄，并通过Windows本地函数将窗口命名为 `"计算器"` 。

fig.1 计算器窗口名称

3. `MOUSEMSG m;` // 定义鼠标消息

```
struct MOUSEMSG
{
    UINT uMsg;          // 当前鼠标消息
    bool mkCtrl;        // Ctrl 键是否按下
    bool mkShift;       // Shift 键是否按下
    bool mkLButton;     // 鼠标左键是否按下
    bool mkMButton;     // 鼠标中键是否按下
    bool mkRButton;     // 鼠标右键是否按下
    int x;              // 当前鼠标 x 坐标（物理坐标）
    int y;              // 当前鼠标 y 坐标（物理坐标）
    int wheel;          // 鼠标滚轮滚动值，120的倍数
};
```

其中，成员 **`uMsg`** 指定鼠标消息类型，可为以下值：

值	含义
`WM_MOUSEMOVE`	鼠标移动消息。
`WM_MOUSEWHEEL`	鼠标滚轮拨动消息。
`WM_LBUTTONDOWN`	左键按下消息。
`WM_LBUTTONUP`	左键弹起消息。
`WM_LBUTTONDBLCLK`	左键双击消息。
`WM_MBUTTONDOWN`	中键按下消息。
`WM_MBUTTONUP`	中键弹起消息。
`WM_MBUTTONDBLCLK`	中键双击消息。
`WM_RBUTTONDOWN`	右键按下消息。
`WM_RBUTTONUP`	右键弹起消息。
`WM_RBUTTONDBLCLK`	右键双击消息。

Easyx Docs里标注为：

该结构体已废弃，仅在 graphics.h 中声明。建议使用 ExMessage 替代。

```
4. bool MouseHit();
```

如果存在鼠标消息，返回 true；否则返回 false。

Easyx Docs里标注为：

该结构体已废弃，仅在 graphics.h 中声明。建议使用 peekmessage 替代。

```
5. MOUSEMSG GetMouseMsg();
```

获取鼠标信息

e.g

`m = GetMouseMsg();` 将获取到的鼠标信息赋值给m；

默认情况下，连续的鼠标单击会被识别为一系列的单击事件。如果希望两个连续的鼠标单击识别为双击事件，请在创建绘图窗口的时候指定标志位 EW_DBLCLKS。

Easyx Docs里标注为：

该结构体已废弃，仅在 graphics.h 中声明。建议使用 getmessage 替代。

```
6. void BeginBatchDraw(); // Begin batch drawing mode
```

这个函数用于开始批量绘图。执行后，任何绘图操作都将暂时不输出到绘图窗口上，直到执行 FlushBatchDraw 或 EndBatchDraw 才将之前的绘图输出。

```
7. void setbkmode(int mode);
```

这个函数用于设置当前设备图案填充和文字输出时的背景模式。

参数：`mode`

指定图案填充和文字输出时的背景模式，可以是以下值：

值	描述
OPAQUE	背景用当前背景色填充（默认）。
TRANSPARENT	背景是透明的。

8. `void setfillstyle(int style, long hatch = NULL, IMAGE* ppattern = NULL);`

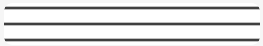
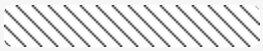
style

指定填充样式。可以是以下宏或值：

宏	值	含义
BS_SOLID	0	固实填充。
BS_NULL	1	不填充。
BS_HATCHED	2	图案填充。
BS_PATTERN	3	自定义图案填充。
BS_DIBPATTERN	5	自定义图像填充。

hatch

指定填充图案，仅当 `style` 为 `BS_HATCHED` 时有效。填充图案的颜色由函数 setfillcolor 设置，背景区域使用背景色还是保持透明由函数 setbkmode 设置。 `hatch` 参数可以是以下宏或值：

宏	值	含义
HS_HORIZONTAL	0	
HS_VERTICAL	1	
HS_FDIAGONAL	2	
HS_BDIAGONAL	3	
HS_CROSS	4	
HS_DIAGCROSS	5	

ppattern

指定自定义填充图案或图像，仅当 `style` 为 `BS_PATTERN` 或 `BS_DIBPATTERN` 时有效。

当 `style` 为 `BS_PATTERN` 时， `ppattern` 指向的 IMAGE 对象表示自定义填充图案，IMAGE 中的黑色（BLACK）对应背景区域，非黑色对应图案区域。图案区域的颜色由函数 settextcolor 设置。

当 `style` 为 `BS_DIBPATTERN` 时，ppattern 指向的 IMAGE 对象表示自定义填充图像，以该图像为填充单元实施填充。

9. `void settextstyle`

```
void settextstyle(
    int nHeight,            //指定高度（逻辑单位）。
    int nWidth,             //字符的平均宽度,如果为 0，则比例自适应。
    LPCTSTR lpszFace,       //字体名称。
    int nEscapement,        //字符串的书写角度，单位 0.1 度。
    int nOrientation,       //每个字符的书写角度，单位 0.1 度。
    int nWeight,            //字符的笔画粗细,范围 0~1000,0表示默认粗细
    bool bItalic,           //是否斜体。
    bool bUnderline,        //是否有下划线
    bool bStrikeOut,        //是否有删除线
    BYTE fbCharSet,
    BYTE fbOutPrecision,
    BYTE fbClipPrecision,
    BYTE fbQuality,
    BYTE fbPitchAndFamily
);
```

fbCharset

指定字符集。可以使用以下预定义的值：

- ANSI_CHARSET
- BALTIC_CHARSET
- CHINESEBIG5_CHARSET
- DEFAULT_CHARSET （字符集基于本地操作系统。例如，系统位置是 English (United States)，字符集将设置为 ANSI_CHARSET）
- EASTEUROPE_CHARSET
- GB2312_CHARSET
- GREEK_CHARSET
- HANGUL_CHARSET
- MAC_CHARSET
- OEM_CHARSET （字符集依赖本地操作系统）
- RUSSIAN_CHARSET
- SHIFTJIS_CHARSET
- SYMBOL_CHARSET
- TURKISH_CHARSET

fbOutPrecision

指定文字的输出精度。输出精度定义输出与所请求的字体高度、宽度、字符方向、行距、间距和字体类型相匹配必须达到的匹配程度。可以是以下值：

值	含义
OUT_DEFAULT_PRECIS	指定默认的映射行为。
OUT_DEVICE_PRECIS	当系统包含多个名称相同的字体时，指定设备字体。
OUT_OUTLINE_PRECIS	指定字体映射选择 TrueType 和其它的 outline-based 字体。
OUT_RASTER_PRECIS	当系统包含多个名称相同的字体时，指定光栅字体（即点阵字体）。
OUT_STRING_PRECIS	这个值并不能用于指定字体映射，只是指定点阵字体枚举数据。
OUT_STROKE_PRECIS	这个值并不能用于指定字体映射，只是指定 TrueType 和其他的 outline-based 字体，以及矢量字体的枚举数据。
OUT_TT_ONLY_PRECIS	指定字体映射只选择 TrueType 字体。如果系统中没有安装 TrueType 字体，将选择默认操作。
OUT_TT_PRECIS	当系统包含多个名称相同的字体时，指定 TrueType 字体。

lfClipPrecision

指定文字的剪辑精度。剪辑精度定义如何剪辑字符的一部分位于剪辑区域之外的字符。可以是以下值：

值	含义
CLIP_DEFAULT_PRECIS	指定默认的剪辑行为。
CLIP_STROKE_PRECIS	这个值并不能用于指定字体映射，只是指定光栅（即点阵）、矢量或 TrueType 字体的枚举数据。
CLIP_EMBEDDED	当使用内嵌的只读字体时，必须指定这个标志。

CLIP_LH_ANGLES	如果指定了该值，所有字体的旋转都依赖于坐标系统的方向是逆时针或顺时针。 如果没有指定该值，设备字体始终逆时针旋转，但是其它字体的旋转依赖于坐标系统的方向。 该设置影响 lfOrientation 参数的效果。

lfQuality

指定文字的输出质量。输出质量定义图形设备界面 (GDI) 必须尝试将逻辑字体属性与实际物理字体的字体属性进行匹配的仔细程度。可以是以下值：

值	含义
ANTIALIASED_QUALITY	指定输出质量是抗锯齿的（如果字体支持）。
DEFAULT_QUALITY	指定输出质量不重要。
DRAFT_QUALITY	草稿质量。字体的显示质量是不重要的。对于光栅字体（即点阵字体），缩放是有效的，这就意味着可以使用更多的尺寸，但是显示质量并不高。如果需要，粗体、斜体、下划线和删除线字体会被合成。
NONANTIALIASED_QUALITY	指定输出质量不是抗锯齿的。
PROOF_QUALITY	正稿质量。指定字体质量比匹配字体属性更重要。对于光栅字体（即点阵字体），缩放是无效的，会选用其最接近的字体大小。虽然选中 PROOF_QUALITY 时字体大小不能精确地映射，但是输出质量很高，并且不会有畸变现象。如果需要，粗体、斜体、下划线和删除线字体会被合成。

如果 ANTIALIASED_QUALITY 和 NONANTIALIASED_QUALITY 都未被选择，抗锯齿效果将依赖于控制面板中字体抗锯齿的设置。

lfPitchAndFamily

指定以常规方式描述字体的字体系列。字体系列描述大致的字体外观。字体系列用于在所需精确字体不可用时指定字体。1~2 位指定字体间距，可以是以下值：

值	含义
DEFAULT_PITCH	指定默认间距。

FIXED_PITCH	指定固定间距。
VARIABLE_PITCH	指定可变间距。

4~7 位指定字体系列，可以是以下值：

值	含义
FF_DECORATIVE	指定特殊字体。例如 Old English。
FF_DONTCARE	指定字体系列不重要。
FF_MODERN	指定具有或不具有衬线的等宽字体。例如，Pica、Elite 和 Courier New 都是等宽字体。
FF_ROMAN	指定具有衬线的等比字体。例如 MS Serif。
FF_SCRIPT	指定设计为类似手写体的字体。例如 Script 和 Cursive。
FF_SWISS	指定不具有衬线的等比字体。例如 MS Sans Serif。

字体间距和字体系列可以用布尔运算符 OR 连接（即符号 | ）。

10. `void setlinecolor(COLORREF color);`

用于设置当前设备画线颜色。

11. `void line(int x1, int y1, int x2, int y2);`

画直线，参数为起终点坐标。

12. `class IMAGE(int width = 0, int height = 0);`

公有成员

int getwidth();	返回 IMAGE 对象的宽度，以像素为单位。
int getheight();	返回 IMAGE 对象的高度，以像素为单位。
operator =	实现 IMAGE 对象的直接赋值。该操作仅拷贝源图像的内容，不拷贝源图像的绘图环境。

13. `void SetWorkingImage(IMAGE* pImg = NULL);`

`pImg` 指绘图设备指针。如果为 NULL，表示绘图设备为默认绘图窗口。

e.g.

```
IMAGE  setimage(int wigth, int height, COLORREF rgb)
{
    IMAGE blacks(wigth, height);    // 创建 img 对象

    SetWorkingImage(&blacks);       // 设置绘图目标为 img 对象
    setfillcolor(rgb);
    solidrectangle(0, 0, wigth, height);

    SetWorkingImage();              // 设置绘图目标为绘图窗口

    return blacks;
}
//其中，solidrectangle();
void solidrectangle(int left, int top, int right, int bottom);
```

14. `getimage();`

```
void getimage(
    IMAGE* pDstImg,     // 保存图像的 IMAGE 对象指针
    int srcX,           // 要获取图像区域左上角 x 坐标
    int srcY,           // 要获取图像区域的左上角 y 坐标
    int srcWidth,       // 要获取图像区域的宽度
    int srcHeight       // 要获取图像区域的高度
);
```

15. `DWORD* GetImageBuffer(IMAGE* pImg = NULL);`

获取到的显示缓冲区指针可以直接读写。

在显示缓冲区中，每个点占用 4 个字节，因此：显示缓冲区的大小 = 宽度 × 高度 × 4 (字节)。像素点在显示缓冲区中按照从左到右、从上向下的顺序依次排列。访问显示缓冲区请勿越界，否则会造成难以预料的后果。

显示缓冲区中的每个点对应 RGBTRIPLE 类型的结构体：

```
struct RGBTRIPLE {
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
};
```

RGBTRIPLE 在内存中的表示形式为：0xrrggbb (bb=蓝，gg=绿，rr=红)，而常用的 COLORREF 在内存中的表示形式为：0xbbggrr。注意，两者的红色和蓝色是相反的，请用 BGR 宏交换红色和蓝色。

如果操作绘图窗口的显示缓冲区，请在操作完毕后，执行 FlushBatchDraw() 使操作生效。

16. `putimage();`

```
void putimage(
    int dstX,               // 绘制位置的 x 坐标
    int dstY,               // 绘制位置的 y 坐标
    int dstWidth,           // 绘制的宽度
    int dstHeight,          // 绘制的高度
    IMAGE *pSrcImg,         // 要绘制的 IMAGE 对象指针
    int srcX,               // 绘制内容在 IMAGE 对象中的左上角 x 坐标
    int srcY,               // 绘制内容在 IMAGE 对象中的左上角 y 坐标
    DWORD dwRop = SRCCOPY   // 三元光栅操作码
);
```

三元光栅操作码（即位操作模式），支持全部的 256 种三元光栅操作码，常用的几种如下：

值	含义
DSTINVERT	目标图像 = NOT 目标图像
MERGECOPY	目标图像 = 源图像 AND 当前填充颜色
MERGEPAINT	目标图像 = 目标图像 OR (NOT 源图像)
NOTSRCCOPY	目标图像 = NOT 源图像
NOTSRCERASE	目标图像 = NOT (目标图像 OR 源图像)
PATCOPY	目标图像 = 当前填充颜色
PATINVERT	目标图像 = 目标图像 XOR 当前填充颜色
PATPAINT	目标图像 = 目标图像 OR ((NOT 源图像) OR 当前填充颜色)
SRCAND	目标图像 = 目标图像 AND 源图像

SRCCOPY	目标图像 = 源图像
SRCERASE	目标图像 = (NOT 目标图像) AND 源图像
SRCINVERT	目标图像 = 目标图像 XOR 源图像
SRCPAINT	目标图像 = 目标图像 OR 源图像

注：

a. AND / OR / NOT / XOR 为布尔运算。

b. "当前填充颜色"是指通过 setfillcolor 设置的用于当前填充的颜色。

```
17. void outtextxy(int x, int y, TCHAR c || LPCTSTR str);
```

LPCTSTR 是一个宏，用于定义指向以 null 结尾的字符串的指针，该字符串是一个常量字符数组。LPCTSTR 的含义是 Long Pointer to a Constant null-terminated String，其中 "LP" 表示 Long Pointer，"C" 表示 Constant，"TSTR" 表示 null 结尾的字符串（即以 null 结尾的字符串）。

在 Windows 编程中，LPCTSTR 主要用于处理以 null 结尾的字符串，例如在 API 函数参数中传递字符串，或者在程序中声明字符串常量。LPCTSTR 的实际类型取决于编译环境，通常在 ANSI 编译环境下为 const char*，在 Unicode 编译环境下为 const wchar_t*。

TCHAR是宏，用于编写可移植的代码，以便在Unicode和ANSI字符集之间进行切换。在Unicode版本的Windows上，TCHAR被定义为wchar_t类型，而在ANSI版本的Windows上，TCHAR被定义为char类型。使用TCHAR可以编写一次代码，然后根据目标平台的不同，在Unicode和ANSI之间进行切换，而无需修改源代码。

```
18. int drawtext(LPCTSTR str, RECT* pRect, UINT uFormat);
```

str

待输出的字符串。

pRect

指定的矩形区域的指针。某些 uFormat 标志会使用这个矩形区域做返回值。

uFormat

指定格式化输出文字的方法。详见后文说明。

默认情况下，输出字符串的背景会用当前背景色填充。使用函数 setbkmode 可以设置文字的背景部分保持透明或使用背景色填充。

以下是 uFormat 参数可以使用的设置项，用来设置文字输出时的格式：

标志	描述
DT_BOTTOM	调整文字位置到矩形底部，仅当和 DT_SINGLELINE 一起使用时有效。
DT_CALCRECT	检测矩形的宽高。如果有多行文字，drawtext 使用 pRect 指定的宽度，并且扩展矩形的底部以容纳每一行文字。如果只有一行文字，drawtext 修改 pRect 的右边以容纳最后一个文字。无论哪种情况，drawtext 都返回格式化后的文字高度，并且不输出文字。
DT_CENTER	文字水平居中。
DT_EDITCONTROL	以单行编辑的方式复制可见文本。具体的说，就是以字符的平均宽度为计算依据，同时用这个方式应用于编辑控制，并且这种方式不显示可见部分的最后一行。
DT_END_ELLIPSIS	对于文本显示，如果字符串的末字符不在矩形内，它会被截断并以省略号标识。 如果是一个单词而不是一个字符，其末尾超出了矩形范围，它不会被截断。 字符串不会被修改，除非指定了 DT_MODIFYSTRING 标志。
DT_EXPANDTABS	展开 TAB 符号。 默认每个 TAB 占 8 个字符位置。注意，DT_WORD_ELLIPSIS、DT_PATH_ELLIPSIS 和 DT_END_ELLIPSIS 不能和 DT_EXPANDTABS 一起用。
DT_EXTERNALLEADING	在行高里包含字体的行间距。通常情况下，行间距不被包含在正文的行高里。
DT_HIDEPREFIX	忽略文字中的前缀字符(&)，并且前缀字符后面的字符不会出现下划线。其他前缀字符仍会被处理。例如： 输入字符串: "A&bc&&d" 通常输出: "Abc&d" 设置标志 DTDT_HIDEPREFIX 后输出: "Abc&d"
DT_INTERNAL	使用系统字体计算文字的宽高等属性。
DT_LEFT	文字左对齐。
DT_MODIFYSTRING	修改指定字符串为显示出的正文。仅当和 DT_END_ELLIPSIS 或 DT_PATH_ELLIPSIS 标志同时使用时有效。
DT_NOCLIP	使输出文字不受 pRect 裁剪限制。使用 DT_NOCLIP 会使 drawtext 执行稍快一些。
DT_NOFULLWIDTHCHARBREAK	防止换行符插入到 DBCS (double-wide character string，即宽字符串)，换行规则相当于 SBCS 字符串。仅当和 DT_WORDBREAK 一起使用时有效。例如，汉字就是宽字符，设置该标志后，连续的汉字会像英文单词一样不被换行符中断。

DT_NOPREFIX	关闭前缀字符的处理。通常，DrawText 解释前缀转义符 & 为其后的字符加下划线，解释 && 为显示单个 &。指定 DT_NOPREFIX，这种处理被关闭。例如： 输入字符串: "A&bc&&d" 通常输出: "Abc&d" 设置标志 DT_NOPREFIX 后输出: "A&bc&&d"
DT_PATH_ELLIPSIS	对于显示的文字，用省略号替换字符串中间的字符以便容纳于矩形内。如果字符串包含反斜杠(),DT_PATH_ELLIPSIS 尽可能的保留最后一个反斜杠后面的文字。 字符串不会被修改，除非指定了 DT_MODIFYSTRING 标志。
DT_PREFIXONLY	仅仅在(&)前缀字符的位置下绘制一个下划线。不绘制字符串中的任何其他字符。例如： 输入字符串: "A&bc&&d" 通常输出: "Abc&d" 设置标志 DT_PREFIXONLY 后输出: " _ "
DT_RIGHT	文字右对齐。
DT_RTLREADING	设置从右向左的阅读顺序（当文字是希伯来文或阿拉伯文时）。默认的阅读顺序是从左向右。
DT_SINGLELINE	使文字显示在一行。回车和换行符都无效。
DT_TABSTOP	设置 TAB 制表位。uFormat 的 15—8 位指定 TAB 的字符宽度。默认 TAB 表示 8 个字符宽度。注意，DT_CALCRECT、DT_EXTERNALLEADING、DT_INTERNAL、DT_NOCLIP 和 DT_NOPREFIX 不能和 DT_TABSTOP 一起用。
DT_TOP	文字顶部对齐。
DT_VCENTER	文字垂直居中。仅当和 DT_SINGLELINE 一起使用时有效。
DT_WORDBREAK	自动换行。当文字超过右边界时会自动换行(不拆开单词)。回车符同样可以换行。
DT_WORD_ELLIPSIS	截去无法容纳的文字，并在末尾增加省略号。

19. `void EndBatchDraw();`

结束批量绘制，并执行未完成的绘制任务

4. 程序中的错误

4.1. 乘除法错误

乘除法会变成字符串的简单连接

fig.2 乘法前

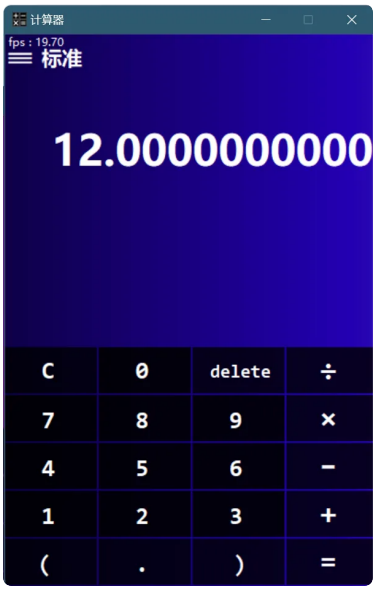


fig.3 乘法结果

4.2. 括号错误

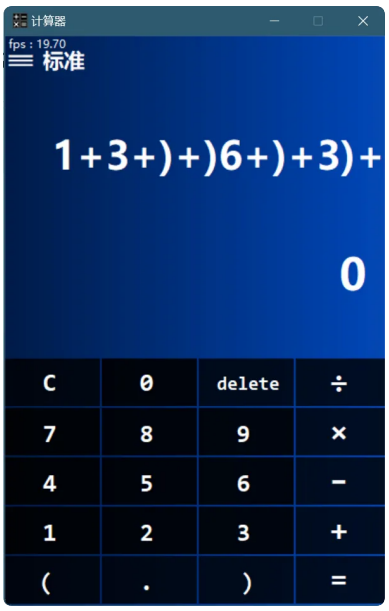


fig4. 括号不合法

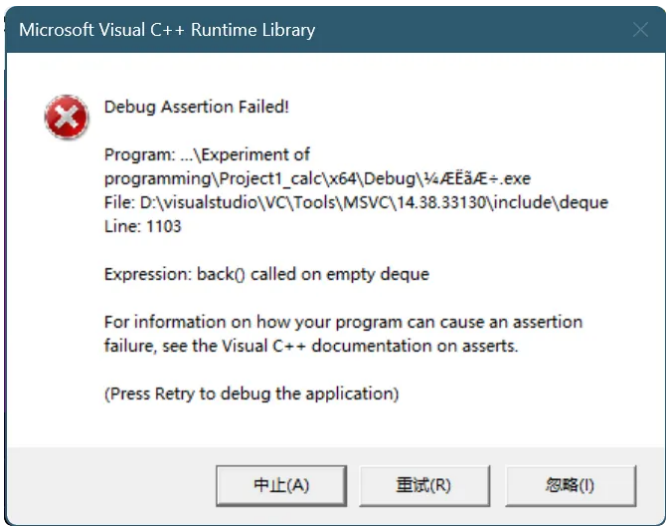


fig.5 错误弹窗

4.3. 科学运算错误

fig.5 简单的除法

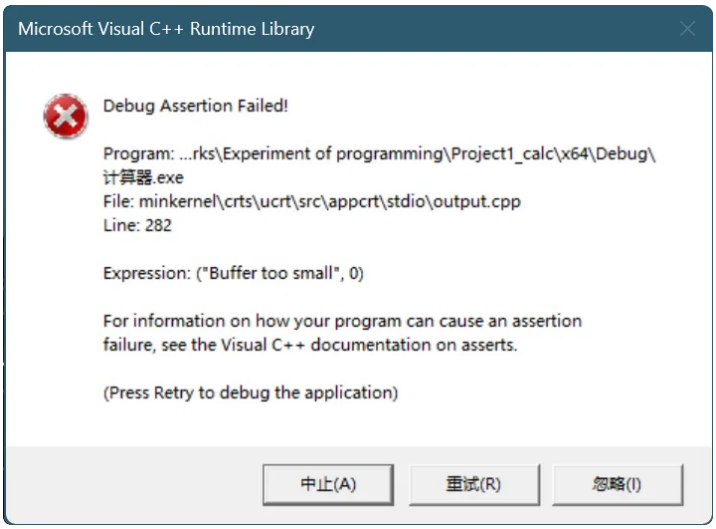


fig.6 错误弹窗

4.4. π计算有误

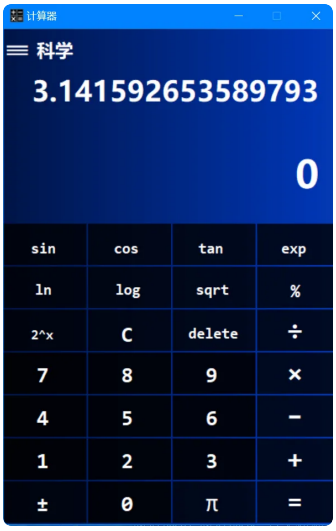


fig.7 cos π

fig.8 结果不对

π相关运算错误大多都错误了，在此只列一例