# 学习大纲

|  |
| --- |
|  |

## MoveToEx函数的用法

|  |
| --- |
| **MoveToEx** 函数将当前位置更新为指定点，并根据需要返回上一个位置。 语法 C++复制  BOOL MoveToEx(  [in] HDC hdc,  [in] int x,  [in] int y,  [out] LPPOINT lppt  ); 参数 [in] hdc  设备上下文的句柄。  [in] x  以逻辑单元形式指定新位置的 x 坐标（以逻辑单位为单位）。  [in] y  以逻辑单元形式指定新位置的 y 坐标（以逻辑单位为单位）。  [out] lppt  指向接收上一个当前位置的 [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point) 结构的指针。 如果此参数是 **NULL** 指针，则不返回上一个位置。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 **MoveToEx** 函数影响所有绘图函数。 示例 有关示例，请参阅 [绘图标记](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/drawing-markers)。 |

## LineTo函数的用法

|  |
| --- |
| C++复制  BOOL LineTo(  [in] HDC hdc,  [in] int x,  [in] int y  ); 参数 [in] hdc  设备上下文的句柄。  [in] x  指定线条终点的 x 坐标（以逻辑单位为单位）。  [in] y  指定直线终点的 y 坐标（以逻辑单位为单位）。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 线条是使用当前笔绘制的，如果笔是几何笔，则使用当前画笔绘制。  如果 **LineTo** 成功，则当前位置设置为指定的终点。 示例 有关示例，请参阅 [绘图标记](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/drawing-markers)。 |

## Polyline函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Polyline** 函数通过连接指定数组中的点来绘制一系列线段。 语法 C++复制  BOOL Polyline(  [in] HDC hdc,  [in] const POINT \*apt,  [in] int cpt  ); 参数 [in] hdc  设备上下文的句柄。  [in] apt  指向 [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point) 结构数组（以逻辑单元为单位）的指针。  [in] cpt  数组中的点数。 此数字必须大于或等于 2。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 使用当前笔从第一个点到后续点绘制线条。 与 [LineTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-lineto) 或 [PolylineTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polylineto) 函数不同， **Polyline** 函数既不使用也不更新当前位置。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [直线和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [LineTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-lineto)  [线条和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves)  [MoveToEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-movetoex)  [点](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point)  [PolyPolyline](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polypolyline)  [PolylineTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polylineto) |

## (少用)PolylineTo函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PolylineTo** 函数绘制一条或多条直线。 语法 C++  BOOL PolylineTo(  [in] HDC hdc,  [in] const POINT \*apt,  [in] DWORD cpt  ); 参数 [in] hdc  设备上下文的句柄。  [in] apt  指向 [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point) 结构的数组的指针，该数组包含线条的顶点（以逻辑单元为单位）。  [in] cpt  数组中的点数。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 与 [Polyline](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polyline) 函数不同， **PolylineTo** 函数使用并更新当前位置。  使用当前笔从当前位置绘制到 *lppt* 参数指定的第一点的线条。 对于每个附加行，函数从上一行的终点绘制到 *lppt* 指定的下一个点。  **PolylineTo** 将当前位置移动到最后一行的终点。  如果此函数绘制的线段形成一个闭合图形，则不会填充该图形。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [直线和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [LineTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-lineto)  [线条和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves)  [MoveToEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-movetoex)  [点](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point)  [PolyPolyline](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polypolyline)  [折线](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polyline) |

## Arc函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Arc** 函数绘制椭圆弧。 语法 C++  BOOL Arc(  [in] HDC hdc,  [in] int x1,  [in] int y1,  [in] int x2,  [in] int y2,  [in] int x3,  [in] int y3,  [in] int x4,  [in] int y4  ); 参数 [in] hdc  进行绘制的设备上下文的句柄。  [in] x1  边界矩形左上角的 x 坐标（以逻辑单元为单位）。  [in] y1  边界矩形左上角的 y 坐标（以逻辑单元为单位）。  [in] x2  边界矩形右下角的 x 坐标（以逻辑单元为单位）。  [in] y2  边界矩形右下角的 y 坐标（以逻辑单元为单位）。  [in] x3  定义弧线起点的径向线终点的 x 坐标（以逻辑单位表示）。  [in] y3  定义弧线起点的径向线终点的 y 坐标（以逻辑单位表示）。  [in] x4  定义弧线终点的径向线终点的 x 坐标（以逻辑单位表示）。  [in] y4  定义弧线终点的径向线终点的 y 坐标（以逻辑单位表示）。 返回值 如果绘制了弧线，则返回值为非零值。  如果未绘制弧线，则返回值为零。 注解 (*nLeftRect*、 *nTopRect*) 和 (*nRightRect*、 *nBottomRect*) 指定边框的点。 由指定的边框形成的椭圆定义弧线的曲线。弧线从与边界矩形的中心相交的点到 (*nXStartArc*、 *nYStartArc*) 点，在当前绘图方向上延伸。 弧线从边界矩形的中心到 *(nXEndArc*、 *nYEndArc*) 点的交叉处结束。 如果起点和终点相同，则绘制一个完整的椭圆形。  使用当前笔绘制弧线;它未填充。  **Arc** 既不使用当前位置，也不更新当前位置。  使用 [GetArcDirection](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getarcdirection) 和 [SetArcDirection](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setarcdirection) 函数获取和设置设备上下文的当前绘制方向。 默认绘制方向是逆时针方向。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [AngleArc](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-anglearc)  [ArcTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-arcto)  [和弦](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-chord)  [椭圆形](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-ellipse)  [GetArcDirection](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getarcdirection)  [直线和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [线条和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves)  [Pie](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-pie)  [SetArcDirection](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setarcdirection) |

### 圆弧绘制原理

|  |
| --- |
|  |

### 默认是按照逆时针方向绘制的

## (少用)ArcTo函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ArcTo** 函数绘制椭圆弧。 语法 C++复制  BOOL ArcTo(  [in] HDC hdc,  [in] int left,  [in] int top,  [in] int right,  [in] int bottom,  [in] int xr1,  [in] int yr1,  [in] int xr2,  [in] int yr2  ); 参数 [in] hdc  进行绘制的设备上下文的句柄。  [in] left  边界矩形左上角的 x 坐标（以逻辑单元为单位）。  [in] top  边界矩形左上角的 y 坐标（以逻辑单元为单位）。  [in] right  边界矩形右下角的 x 坐标（以逻辑单元为单位）。  [in] bottom  边界矩形右下角的 y 坐标（以逻辑单元为单位）。  [in] xr1  定义弧的起点的径向端点的 x 坐标（以逻辑单位表示）。  [in] yr1  定义弧的起点的径向端点的 y 坐标（以逻辑单位表示）。  [in] xr2  定义弧的终点的径向端点的 x 坐标（以逻辑单位为单位）。  [in] yr2  定义弧的终点的径向端点的 y 坐标（以逻辑单位表示）。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 **ArcTo** 类似于 [Arc](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-arc) 函数，只不过当前位置已更新。  (*nLeftRect*、 *nTopRect*) 和 (*nRightRect*、 *nBottomRect*) 指定边框的点。 由指定的边框形成的椭圆定义弧线的曲线。弧线从与径向线相交的点逆时针方向延伸，从边框中心到 (*nXRadial1*、 *nYRadial1*) 点。 弧线从边框的中心到 *(nXRadial2*、 *nYRadial2*) 点的径向线相交处结束。 如果起点和终点相同，则绘制一个完整的椭圆形。  从当前位置到弧线的起点绘制一条线。如果未发生错误，则当前位置设置为弧线的终点。  使用当前笔绘制弧线;它未填充。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [AngleArc](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-anglearc)  [Arc](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-arc)  [直线和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [线条和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves)  [SetArcDirection](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setarcdirection) |

## SetArcDirection函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SetArcDirection** 设置要用于弧线和矩形函数的绘制方向。 语法 C++  int SetArcDirection(  [in] HDC hdc,  [in] int dir  ); 参数 [in] hdc  设备上下文的句柄。  [in] dir  新的弧线方向。 此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **AD\_COUNTERCLOCKWISE** | 逆时针方向绘制的图形。 | | **AD\_CLOCKWISE** | 顺时针方向绘制的图形。 |  返回值 如果函数成功，则返回值指定旧的弧线方向。  如果函数失败，则返回值为零。 注解 默认方向为逆时针方向。  **SetArcDirection** 函数指定以下函数的绘制方向：   * [Arc](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-arc) * [ArcTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-arcto) * [和弦](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-chord) * [椭圆形](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-ellipse) * [Pie](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-pie) * [矩形](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-rectangle) * [RoundRect](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-roundrect)  要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [线条和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [线条和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves) |

## PolyBezier函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PolyBezier** 函数绘制一个或多个贝塞尔曲线。 语法 C++  BOOL PolyBezier(  [in] HDC hdc,  [in] const POINT \*apt,  [in] DWORD cpt  ); 参数 [in] hdc  设备上下文的句柄。  [in] apt  指向 [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point) 结构的数组的指针，该结构包含逻辑单元中曲线的终结点和控制点。  [in] cpt  *lppt* 数组中的点数。 此值必须是要绘制的曲线数的三倍以上，因为每个贝塞尔曲线都需要两个控制点和一个端点，而初始曲线需要额外的起点。 返回值 如果函数成功，则返回值为非零。  如果函数失败，则返回值为零。 言论 **PolyBezier** 函数使用 *lppt* 参数指定的终结点和控制点绘制立方贝塞尔曲线。 第一条曲线使用第二和第三个点作为控制点，从第一个点绘制到第四个点。 序列中的每个后续曲线只需再三个点：上一曲线的终点用作起点，序列中的下两个点是控制点，第三个是终点。  **PolyBezier** 函数既不使用也不更新当前位置。 该图未填充。  此函数使用当前笔绘制线条。 例子 有关示例，请参阅更新区域中的 重绘。 要求  |  |  | | --- | --- | | **标头** | wingdi.h （包括 Windows.h） | | **库** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [直线和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [直线和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves)  [MoveToEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-movetoex)  [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point)  [PolyBezierTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polybezierto) |

## 绘制一段贝塞尔曲线需要4个点，第2和第3个点是控制点，1是起点，4是终点

## (少用)PolyBezierTo函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PolyBezierTo** 函数绘制一个或多个 Bézier 曲线。 语法 C++复制  BOOL PolyBezierTo(  [in] HDC hdc,  [in] const POINT \*apt,  [in] DWORD cpt  ); 参数 [in] hdc  设备上下文的句柄。  [in] apt  指向 [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point) 结构的指针，该结构包含逻辑单元中的终结点和控制点。  [in] cpt  *lppt* 数组中的点数。 此值必须是要绘制的曲线数的三倍，因为每个贝塞尔曲线都需要两个控制点和一个终点。 返回值 如果函数成功，则返回值为非零。  如果函数失败，则返回值为零。 言论 此函数使用 *lppt* 参数指定的控制点绘制立方贝塞尔曲线。 第一条曲线通过使用前两个点作为控制点从当前位置绘制到第三个点。 对于每个后续曲线，该函数只需要另外三个点，并使用上一曲线的终点作为下一个曲线的起点。  **PolyBezierTo** 将当前位置移到最后一条贝塞尔曲线的终点。 该图未填充。  此函数使用当前笔绘制线条。 要求  |  |  | | --- | --- | | **标头** | wingdi.h （包括 Windows.h） | | **库** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [直线和曲线函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/line-and-curve-functions)  [直线和曲线概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/lines-and-curves)  [MoveToEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-movetoex)  [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point)  [PolyBezier](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polybezier) |

# 演练

## 1.新建一个win32项目，取名：Lesson43-draw-lines

|  |
| --- |
|  |

## 2.找到窗口过程函数，进入WM\_PAINT消息的处理代码，然后我们绘制一条直线

|  |
| --- |
|  |

### 效果如下：

|  |
| --- |
|  |

## 3.我们我们移动起始点的位置，再绘制一条直线

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 4.如果你第一次绘制之前没有移动起点，它默认在窗口客户区的0，0位置，你可以直接画线，如我们把第一个移动代码注释了

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 5.我们把他改回来

|  |
| --- |
|  |

## 6.需要注意，并不是每一次绘制之前都需要移动起点，绘制完成后直线的终点就是新的起点，把上面的代码注释了，我们重新移动起点到100，100的地方，然后绘制4次直线

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 注意：如果你的坐标正确，绘制4条直线可以绘制一个矩形

## 7.下面我们来学习Polyline函数，它可以把多个点的坐标保存起来，然后一次绘制所有直线，在开始之前，我们需要定义一个数组来保存点的坐标

|  |
| --- |
|  |

## 8.然后我们在下面添加绘制代码

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 9.然后我们改为4个点

|  |
| --- |
|  |
|  |

### 效果

|  |
| --- |
|  |

## 10.那如果我们把起点再添加到后面

|  |
| --- |
|  |
|  |

### 效果：此时就是一个矩形

|  |
| --- |
|  |

### 注意：用Polyline绘制图形后，画笔的起始点并没有改变。如果你需要改变，就使用PolylineTo

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 11.如果此时我们再用LineTo函数绘制一条直线，代码如下

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 其实我们一般只是用Polyline函数而不用PolylineTo函数

## 12.下面，我们来学习Arc函数，代码如下，注意是按照逆时针顺序来绘制，4个点分别表示左上角，右下角，起点，终点

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 13.如果我们把起点和终点对调，就好得到另外一段圆弧

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 14. 我们可以使用SetArcDirection函数来改变圆弧的绘制方向，我们修改一下方向，然后用默认绘制短圆弧的坐标再绘制一次

|  |
| --- |
|  |

### 效果，改变了绘制方向后，原来绘制短圆弧的代码现在有可以绘制长圆弧了

|  |
| --- |
|  |

## 15，还有一个ArcTo函数，用得很少，因为它好PolylineTo一样会绘制一段多余的直线

|  |
| --- |
|  |

## 16下面我们来学习贝塞尔曲线的绘制方法，我们定义2个贝塞尔曲线端点数组，然后绘制两条贝塞尔曲线

|  |
| --- |
|  |
|  |

### 效果

|  |
| --- |
|  |

### 可见，PolyBezier函数不会改变画笔的起点

# 这一节学习到此为止，完整代码如下

## Lesson43-draw-lines.cpp

|  |
| --- |
| // Lesson43-draw-lines.cpp : 定义应用程序的入口点。  //  #include "stdafx.h"  #include "Lesson43-draw-lines.h"  #define MAX\_LOADSTRING 100  // 全局变量:  HINSTANCE hInst; // 当前实例  TCHAR szTitle[MAX\_LOADSTRING]; // 标题栏文本  TCHAR szWindowClass[MAX\_LOADSTRING]; // 主窗口类名  // 此代码模块中包含的函数的前向声明:  ATOM MyRegisterClass(HINSTANCE hInstance);  BOOL InitInstance(HINSTANCE, int);  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);  INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);  int APIENTRY \_tWinMain(HINSTANCE hInstance,  HINSTANCE hPrevInstance,  LPTSTR lpCmdLine,  int nCmdShow)  {  UNREFERENCED\_PARAMETER(hPrevInstance);  UNREFERENCED\_PARAMETER(lpCmdLine);  // TODO: 在此放置代码。  MSG msg;  HACCEL hAccelTable;  // 初始化全局字符串  LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);  LoadString(hInstance, IDC\_LESSON43DRAWLINES, szWindowClass, MAX\_LOADSTRING);  MyRegisterClass(hInstance);  // 执行应用程序初始化:  if (!InitInstance (hInstance, nCmdShow))  {  return FALSE;  }  hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LESSON43DRAWLINES));  // 主消息循环:  while (GetMessage(&msg, NULL, 0, 0))  {  if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  }  return (int) msg.wParam;  }  //  // 函数: MyRegisterClass()  //  // 目的: 注册窗口类。  //  // 注释:  //  // 仅当希望  // 此代码与添加到 Windows 95 中的“RegisterClassEx”  // 函数之前的 Win32 系统兼容时，才需要此函数及其用法。调用此函数十分重要，  // 这样应用程序就可以获得关联的  // “格式正确的”小图标。  //  ATOM MyRegisterClass(HINSTANCE hInstance)  {  WNDCLASSEX wcex;  wcex.cbSize = sizeof(WNDCLASSEX);  wcex.style = CS\_HREDRAW | CS\_VREDRAW;  wcex.lpfnWndProc = WndProc;  wcex.cbClsExtra = 0;  wcex.cbWndExtra = 0;  wcex.hInstance = hInstance;  wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_LESSON43DRAWLINES));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_LESSON43DRAWLINES);  wcex.lpszClassName = szWindowClass;  wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));  return RegisterClassEx(&wcex);  }  //  // 函数: InitInstance(HINSTANCE, int)  //  // 目的: 保存实例句柄并创建主窗口  //  // 注释:  //  // 在此函数中，我们在全局变量中保存实例句柄并  // 创建和显示主程序窗口。  //  BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)  {  HWND hWnd;  hInst = hInstance; // 将实例句柄存储在全局变量中  hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,  CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);  if (!hWnd)  {  return FALSE;  }  ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);  return TRUE;  }  //  // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)  //  // 目的: 处理主窗口的消息。  //  // WM\_COMMAND - 处理应用程序菜单  // WM\_PAINT - 绘制主窗口  // WM\_DESTROY - 发送退出消息并返回  //  //  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  {  int wmId, wmEvent;  PAINTSTRUCT ps;  HDC hdc;  //POINT ptPloy[] = {100,100,300,100,300,200};  //POINT ptPloy[] = {100,100,300,100,300,200,100,200};  POINT ptPloy[] = {100,100,300,100,300,200,100,200,100,100};  //POINT ptBezier[] = {80,50,50,80,100,100,450,150};  POINT ptBezier[] = {120,100,120,200,250,150,500,40};  POINT ptBezier2[] = {120,100,50,350,250,200,500,40};  switch (message)  {  case WM\_COMMAND:  wmId = LOWORD(wParam);  wmEvent = HIWORD(wParam);  // 分析菜单选择:  switch (wmId)  {  case IDM\_ABOUT:  DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);  break;  case IDM\_EXIT:  DestroyWindow(hWnd);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  break;  case WM\_PAINT:  hdc = BeginPaint(hWnd, &ps);  // TODO: 在此添加任意绘图代码...  //MoveToEx(hdc,100,100,NULL);  //LineTo(hdc,300,100);  //MoveToEx(hdc,100,200,NULL);//移动起点  //LineTo(hdc,300,200);  //绘制折线  //MoveToEx(hdc,100,100,NULL);  //LineTo(hdc,300,100);//此时直线的终点就是新起点  //LineTo(hdc,300,200);//以此类推  //LineTo(hdc,100,300);  //LineTo(hdc,300,400);//如果你的坐标正确，绘制4条直线可以绘制一个矩形  //Polyline，可以保存多个点然后一次绘制  //Polyline(hdc,ptPloy,3);  //Polyline(hdc,ptPloy,4);  //Polyline(hdc,ptPloy,5);  //PolylineTo  /\*PolylineTo(hdc,ptPloy,5);  LineTo(hdc,100,500);\*/  //绘制圆弧,需要指定四个点,分别表示左上角，右下角，起点，终点  //Arc(hdc,100,100,300,200,80,120,290,50);  //Arc(hdc,100,100,300,200,290,50,80,120);  //我们可以使用SetArcDirection函数来改变圆弧的绘制方向  //SetArcDirection(hdc,AD\_CLOCKWISE);//默认是逆时针，我们这里设置为顺时针  //Arc(hdc,100,100,300,200,290,50,80,120);  //贝塞尔曲线  PolyBezier(hdc,ptBezier,4);  PolyBezier(hdc,ptBezier2,4);  EndPaint(hWnd, &ps);  break;  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  }  // “关于”框的消息处理程序。  INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)  {  UNREFERENCED\_PARAMETER(lParam);  switch (message)  {  case WM\_INITDIALOG:  return (INT\_PTR)TRUE;  case WM\_COMMAND:  if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)  {  EndDialog(hDlg, LOWORD(wParam));  return (INT\_PTR)TRUE;  }  break;  }  return (INT\_PTR)FALSE;  } |