

Chapter 15



Drawing in a Window

The Window Client Area

- ❑ A coordinate system that is local to the window.
- ❑ It always uses the **upper-left** corner of the client area as its reference point.

This is the reference point for this window's client areas.

This location of point is defined by the distances x and y .

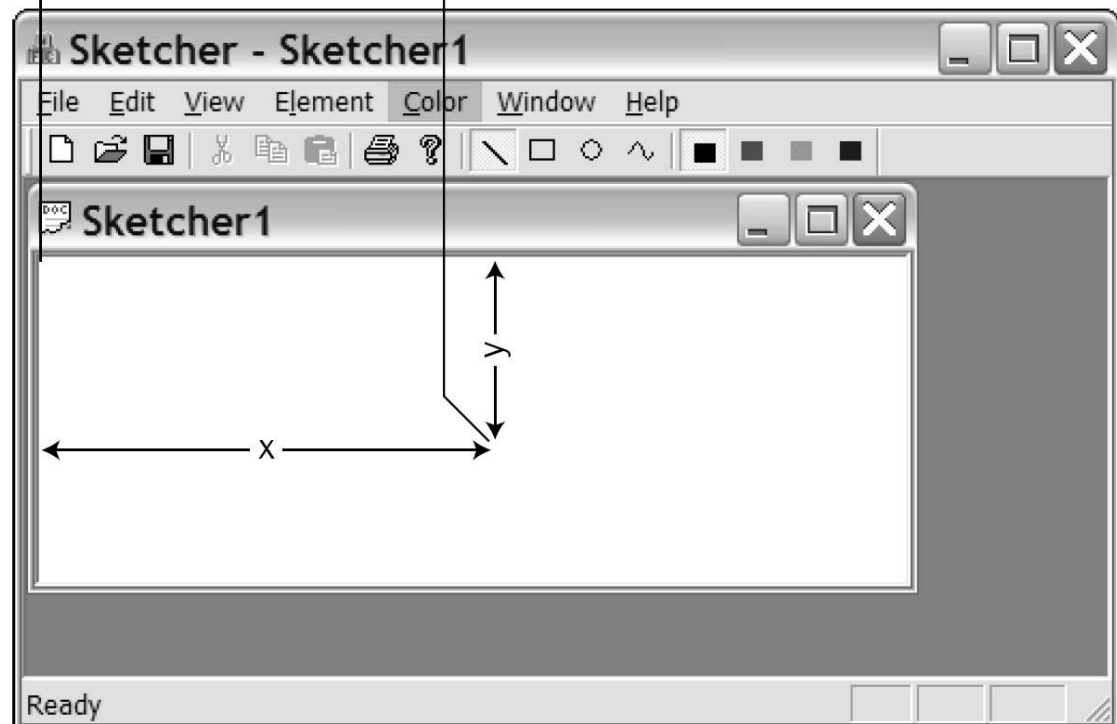


Figure 14-1

Graphical Device Interface (GDI)

- ❑ You don't draw pictures directly to the screen.
- ❑ You must define the graphical output (lines, circles, text) using the **Graphical Device Interface**.
- ❑ The GDI enables you to program graphical output independently of the hardware
 - Such as the display screen, printers, plotters

What Is a Device Context?

- ❑ You must use a **device context** to draw anything on a graphical output device.
- ❑ In a word, a device context is a data structure defined by Windows.
 - A device context contains attributes such as
 - ❑ Drawing color
 - ❑ Background color
 - ❑ Line thickness
 - ❑ Font
 - ❑ Mapping mode
- ❑ Your output requests are specified by device-independent GDI function calls.
 - A device context contains information that allows Windows to translate those requests into actions on the particular physical output device.

Mapping Modes (1)

□ MM_TEXT

- A logical unit is one device pixel with positive x from left to right, and positive y from top to bottom of the window client area.

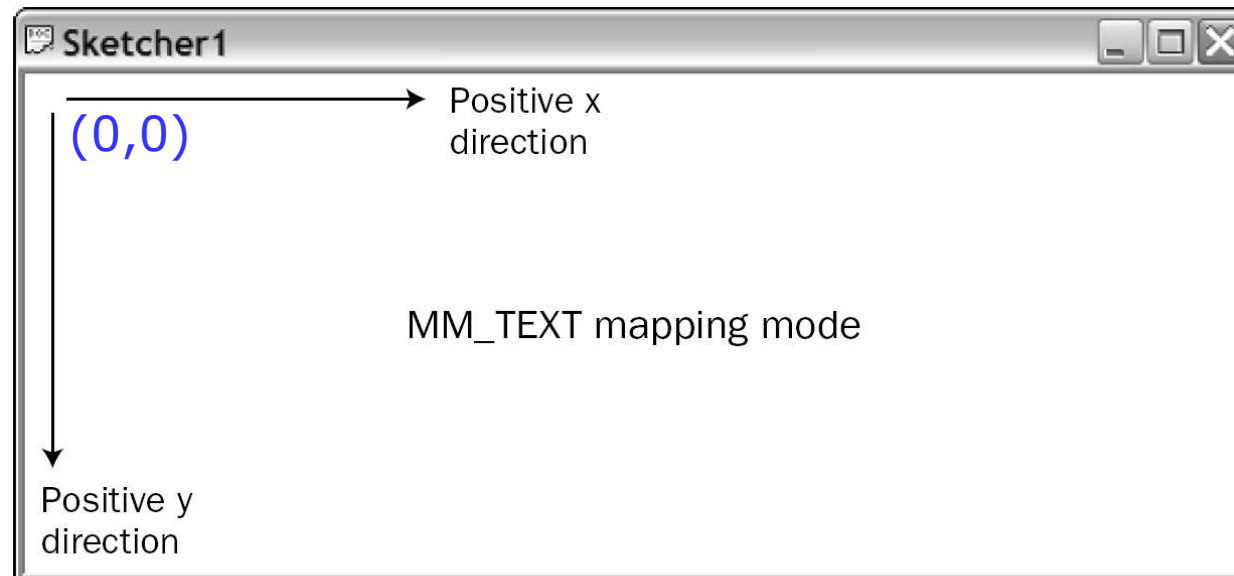


Figure 14-2

Mapping Modes (2)

□ MM_LOENGLISH

- A logical unit is 0.01 inches with positive x from left to right, and positive y from the top of the client area upwards.
 - Consistent with what we learned in high school.
- By default, the point at the upper-left corner has the coordinates (0,0) in every mapping mode.
- Coordinate are always 32-bit signed integers.

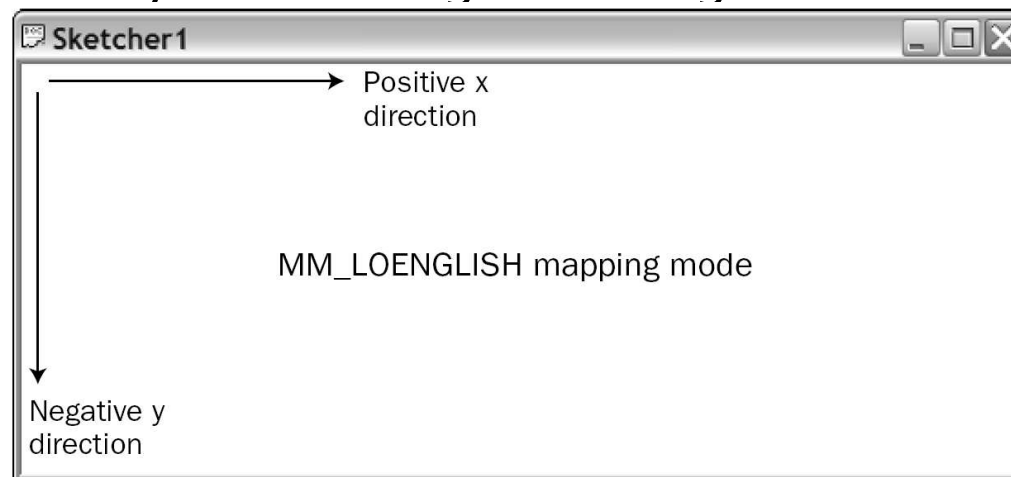


Figure 14-3

The View Class in Your Application

- In the class `CSketcherView`, the function `OnDraw()` is called when a `WM_PAINT` message is received in your program.
 - Windows sends this message to your program whenever it requires the client area to be redrawn.
 - The user resizes the window
 - Part of your window was previously “covered” by another window

The OnDraw () Member Function

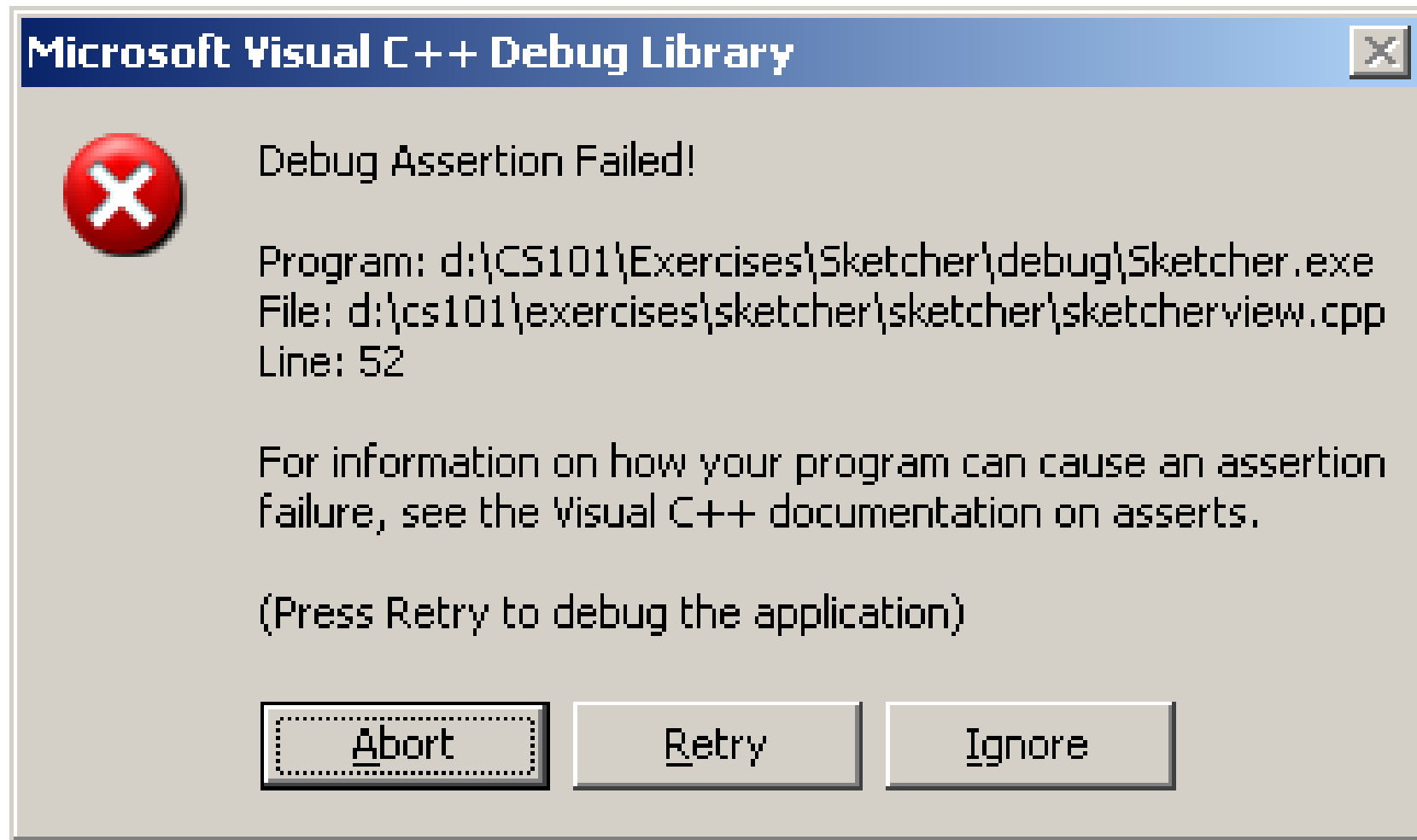
```
void CSketcherView::OnDraw(CDC* pDC)
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: add draw code for sensitive data here
}
```

Returns the address of the document object related to the current view (P.782)

Make sure the pointer pDoc contains a valid address.

Make sure the pointer pDoc is not null.

Assertion Failed



The CDC Class

- ❑ You should do all the drawing in your program using members of the CDC class.
 - C – Class
 - DC – Device Context
- ❑ There are over a hundred member functions of this class.
- ❑ Sometimes you use objects of CClientDC
 - It is derived from CDC, and thus contains all the members we will discuss.
 - Its advantage is that CClientDC always contains a device context that represents only the client area of a window.

Current Position

- ❑ In a device context, you draw entities such as lines, and text relative to a current position.
- ❑ You may set the current position by calling the MoveTo() function.

MoveTo ()

- The CDC class overloads the MoveTo() function in two versions to provide flexibility.
 - CPoint MoveTo(int x, int y);
 - CPoint MoveTo(POINT aPoint);
- POINT is a structure defined as:
 - typedef struct tagPOINT
 - {
 - LONG x;
 - LONG y;
 - } POINT;
- CPoint is a class with data members x and y of type LONG.
- The return value from the MoveTo() function is a CPoint object that specifies the position before the move.
 - This allows you to move back easily.

Drawing Lines

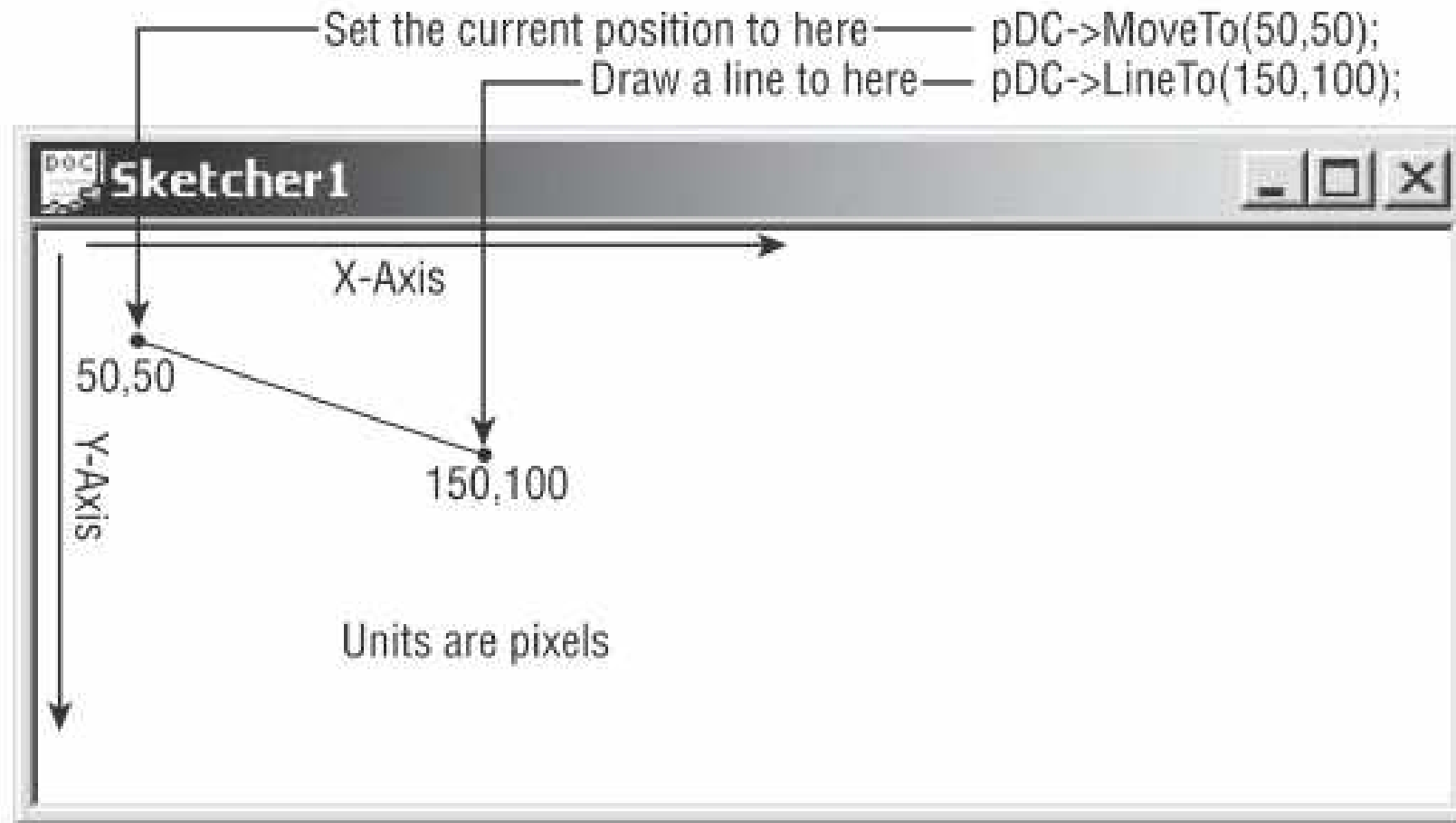


Figure 15-4

LineTo ()

- The CDC class also defines two versions of the LineTo() function
 - `BOOL LineTo(int x, int y);`
 - `BOOL LineTo(POINT aPoint);`
 - You may use either a POINT struct or a CPoint object as the argument.

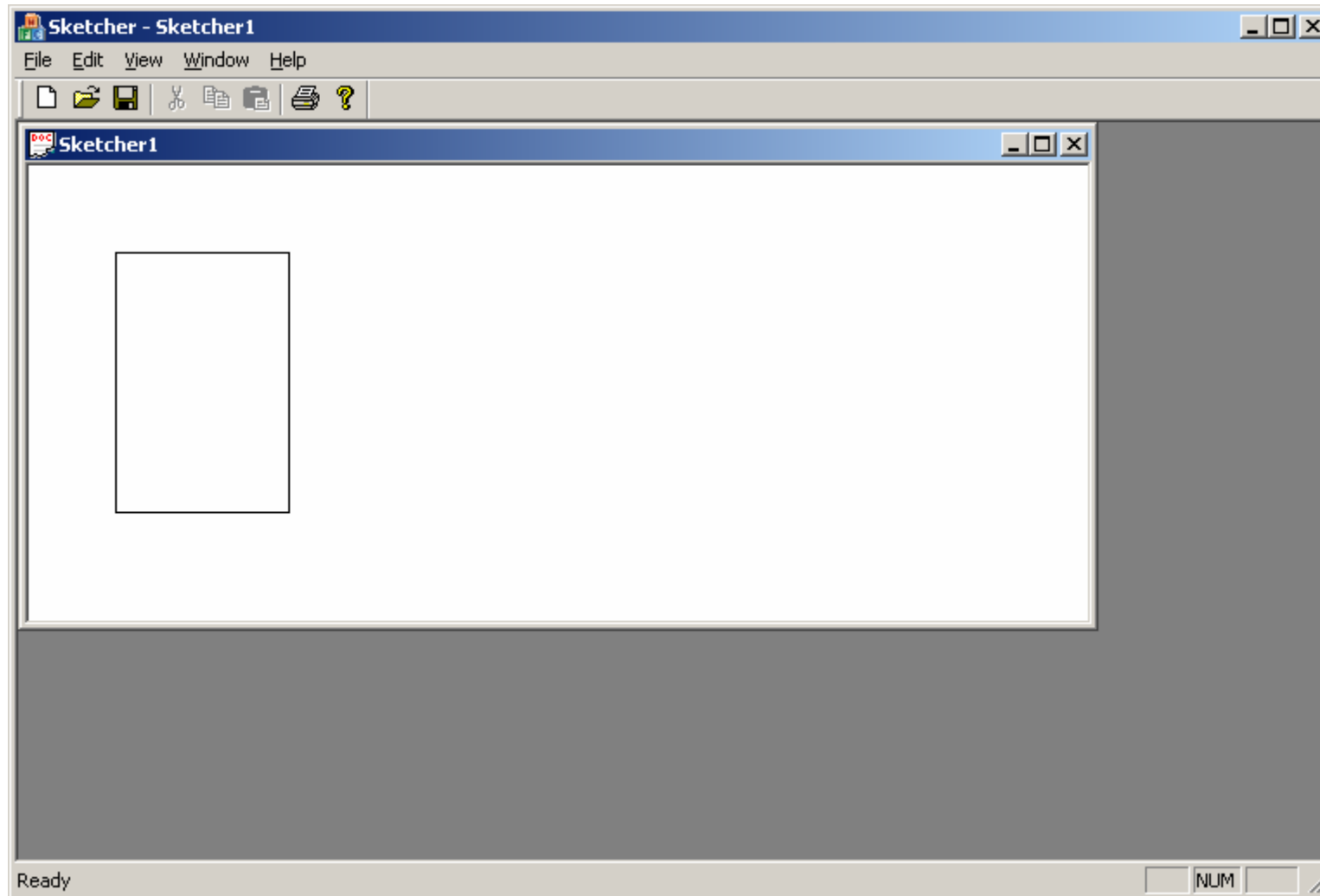
Ex15_1 (P.854)

- When the LineTo() function is executed, the current position is changed to the point specifying the end of the line.

```
void CSketcherView::OnDraw(CDC* pDC)
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    pDC->MoveTo(50,50);
    pDC->LineTo(50,200);
    pDC->LineTo(150,200);
    pDC->LineTo(150,50);
    pDC->LineTo(50,50);
}
```

Figure 15-5

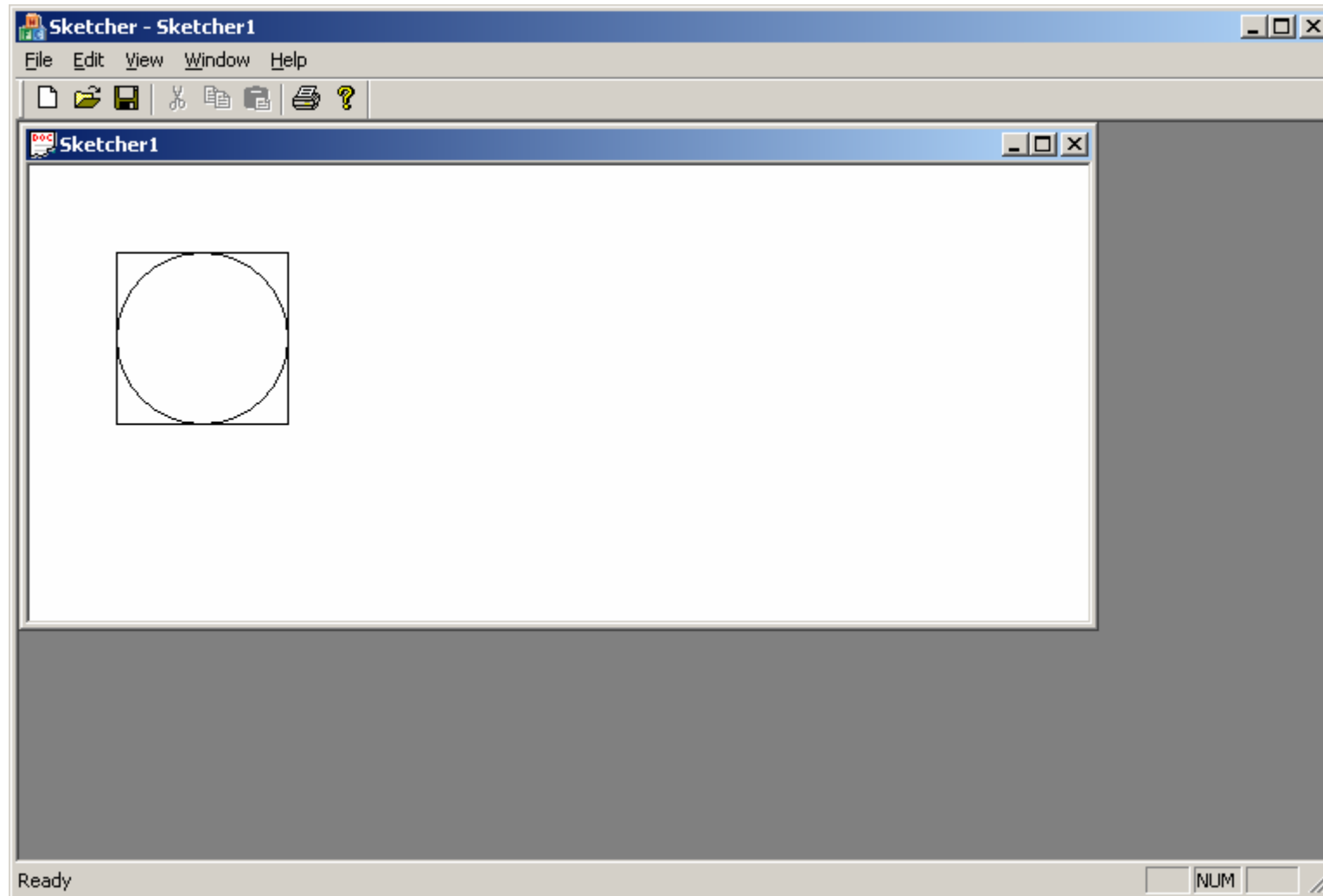


Drawing Rectangles & Circles

```
void CSketcherView::OnDraw(CDC* pDC)
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    pDC->Rectangle(50,50, 150, 150);
    pDC->Ellipse(50,50, 150,150);
}
```

A circle is a special ellipse



Arc

- ❑ Another way to draw circles is to use the Arc() function.
 - `BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);`
 - ❑ (x1, y1) and (x2, y2) define the upper-left and lower-right corners of a rectangle enclosing the circle (ellipse).
 - ❑ The points (x3, y3) and (x4, y4) define the start and end points of the arc, which is drawn counterclockwise.
 - ❑ If (x4, y4) is identical to (x3, y3), you get a circle.
 - `BOOL Arc(LPCRECT lpRect, POINT Startpt, POINT Endpt);`
 - ❑ lpRect points to an object of the class CRect, which has four public data members: left , top, right, bottom.

Drawing with the Arc () Function

```
void CSketcherView::OnDraw(CDC* pDC)
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    pDC->Arc(50,50,150,150,100,75,150,100);

    CRect* pRect = new CRect(250,50,300,100);
    CPoint Start(275,100);
    CPoint End(250,75);
    pDC->Arc(pRect, Start, End);
    delete pRect;
}
```

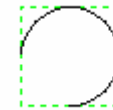
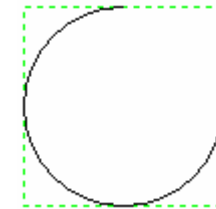
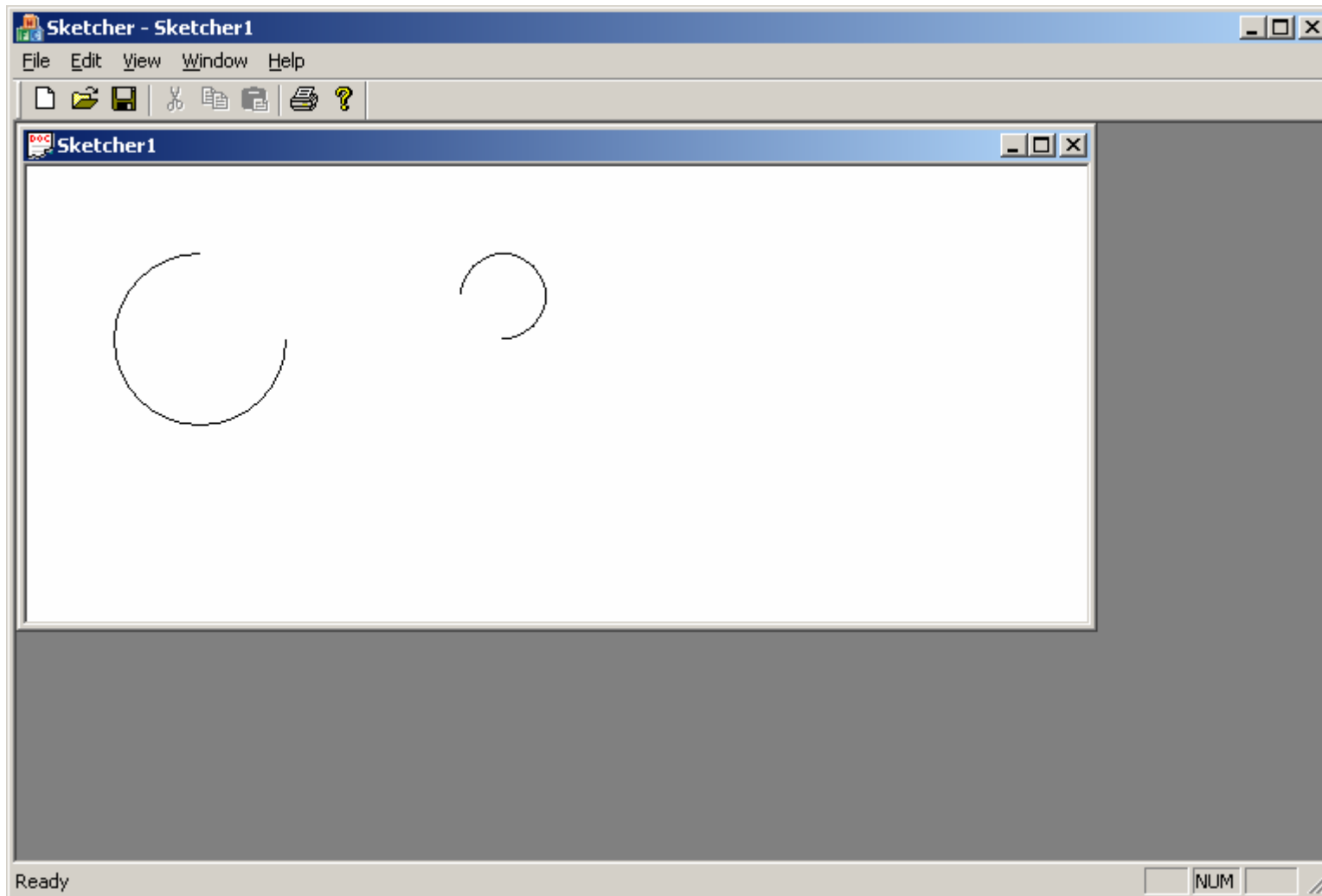
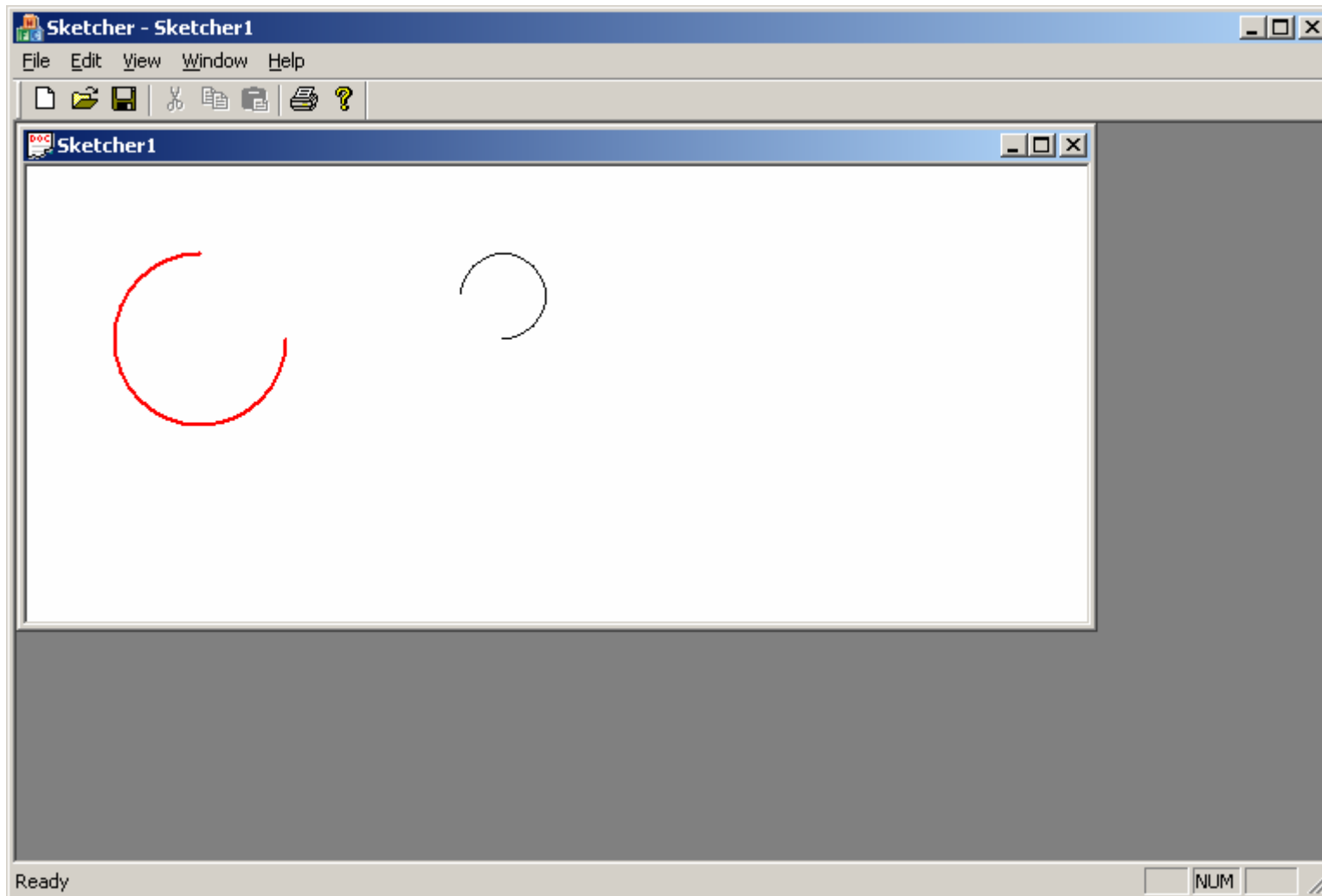


Figure 14-6



Drawing in Color



Using a Pen

- Declare a pen object and initialize it as a red solid pen drawing a line 2 pixels wide (P.858)
 - `CPen aPen;`
 - `aPen.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));`

 - `CPen* pOldPen = pDC->SelectObject(&aPen);`
 - `pDC->Arc(50,50,150,150,100,75,150,100);`

 - `pDC->SelectObject(pOldPen);`
 - `CRect* pRect = new CRect(250,50,300,100);`
 - `CPoint Start(275,100);`
 - `CPoint End(250,75);`
 - `pDC->Arc(pRect, Start, End);`
 - `delete pRect;`

Pen Style

❑ `BOOL CreatePen(int aPenStyle, int aWidth, COLORREF aColor);`

- `PS_SOLID` – solid line
- `PS_DASH` – dashed line
- `PS_DOT` – dotted line
- `PS_DASHDOT` – alternating dashes and dots
- `PS_DASHDOTDOT` – alternating dashes and double dots.
- `PS_NULL` – draw nothing

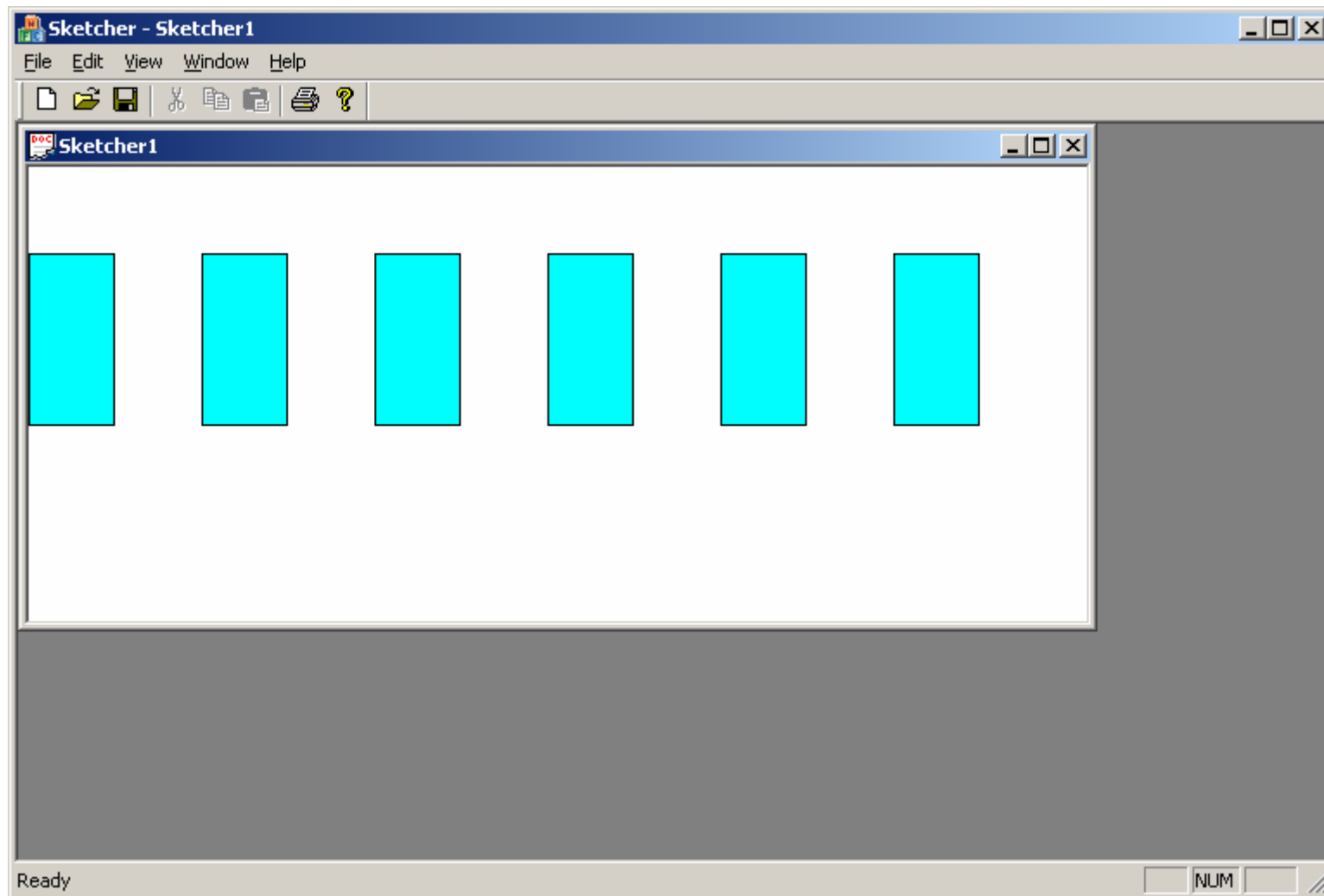


Creating a Brush

- ❑ A brush is actually an 8x8 block of patterns that's repeated over the region to be filled.
- ❑ All closed shapes in CDC will be filled with a brush (and a color).
- ❑ Select the brush into the device context by calling the `SelectObject()` member (similar to selecting a pen).

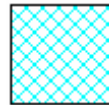
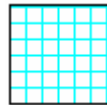
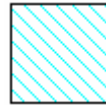
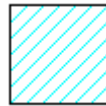
```
CBrush aBrush;  
aBrush.CreateSolidBrush(RGB(0,255,255));  
CBrush* pOldBrush =  
    static_cast<CBrush*> (pDC->SelectObject(&aBrush));  
  
const int width = 50;  
const int height = 50;  
int i;  
for (i=0; i<6; i++)  
    pDC->Rectangle(i*2*width, 50,i*2*width+50, 150);  
  
pDC->SelectObject(pOldBrush);
```

Solid Brush



Hatching Style

- HS_HORIZONTAL
- HS_VERTICAL
- HS_BDIAGONAL
- HS_FDIAGONAL
- HS_CROSS
- HS_DIAGCROSS



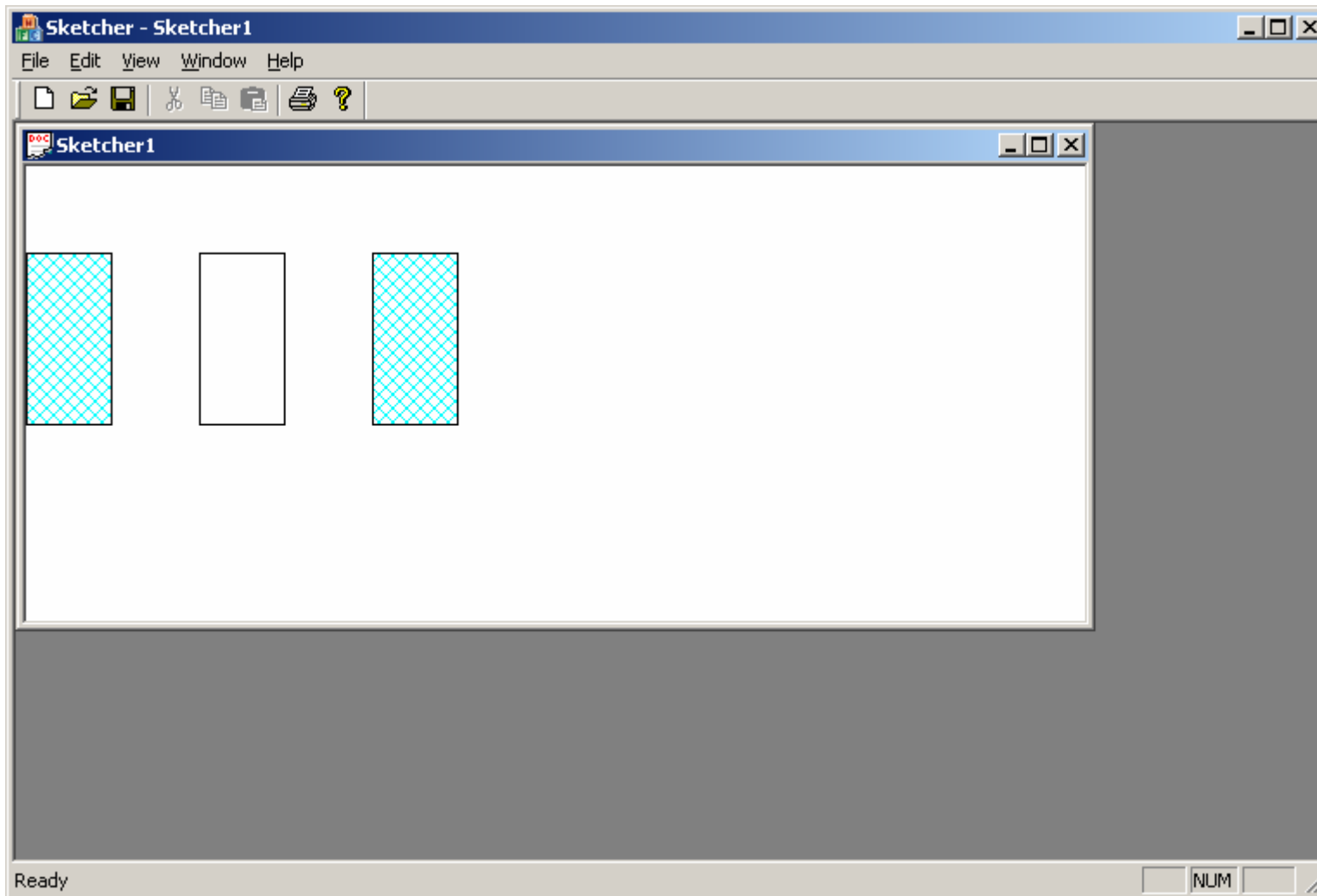
```
CBrush aBrush;
```

```
aBrush.CreateHatchBrush(HS_DIAGCROSS,  
    RGB(0,255,255));
```

```
CBrush* pOldBrush =
```

```
    static_cast<CBrush*> (pDC->SelectObject(&aBrush));
```

A Hatched Brush



Drawing Graphics in Practice

- The easiest mechanism for drawing is using just the mouse.

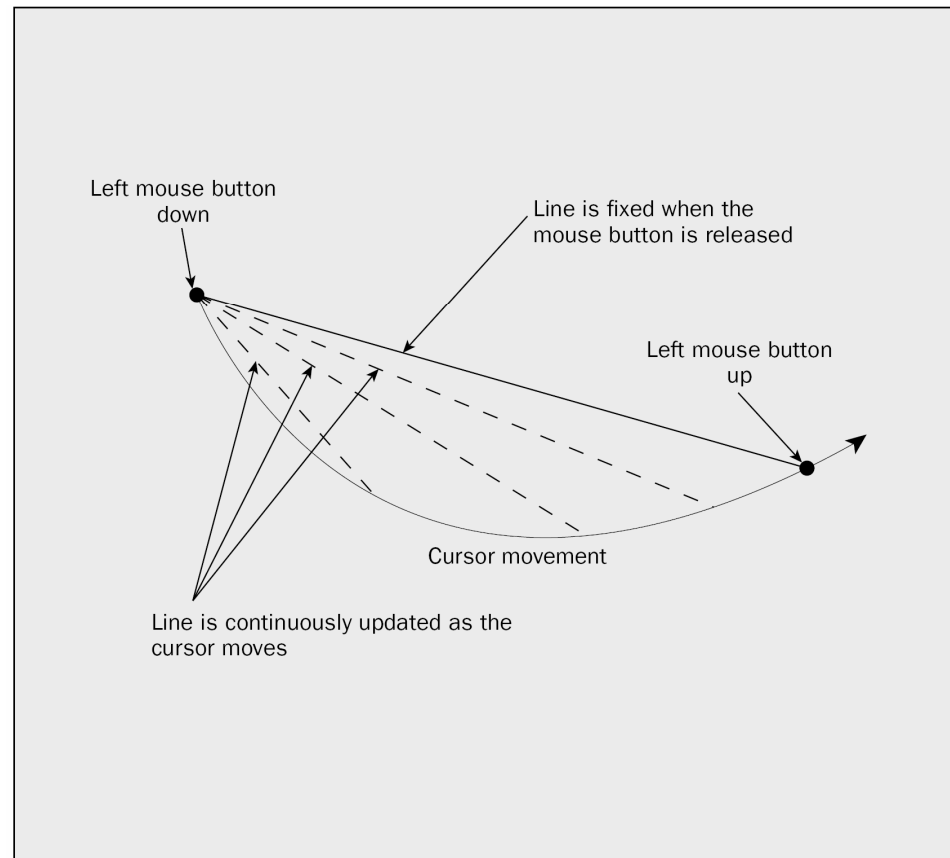


Figure 14-7

Circle

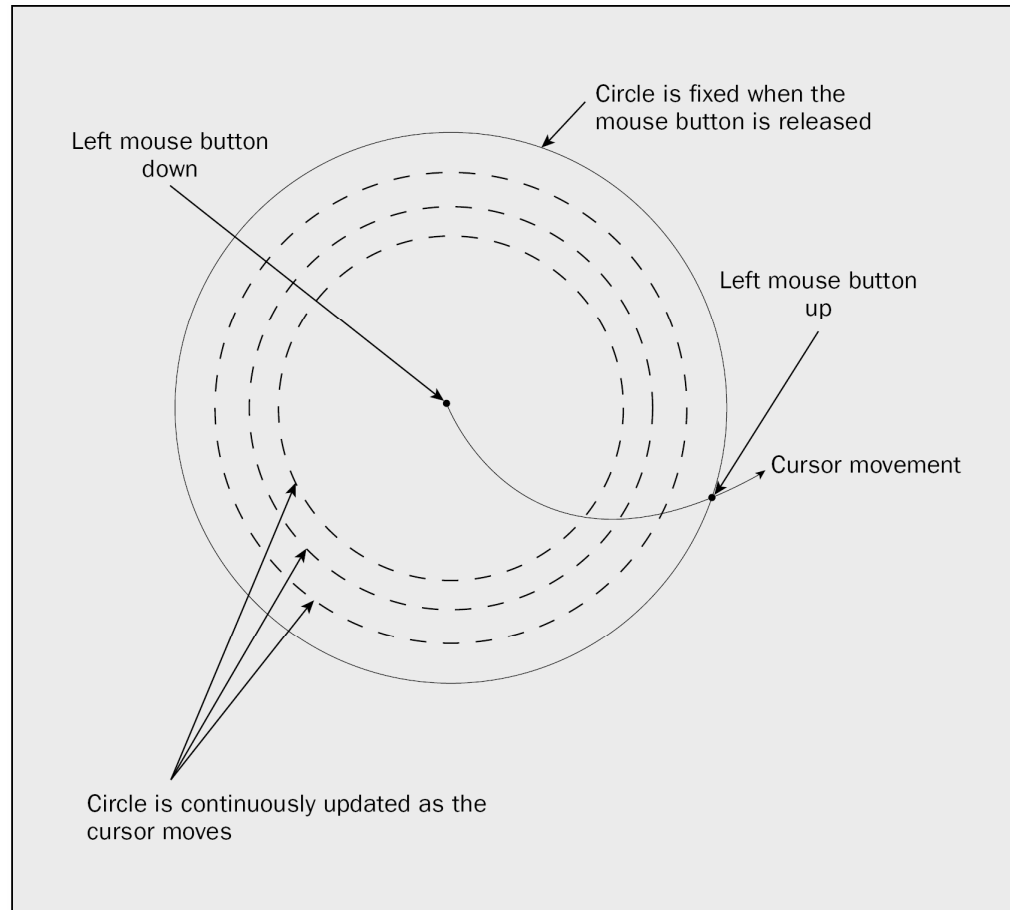


Figure 14-8

Rectangle

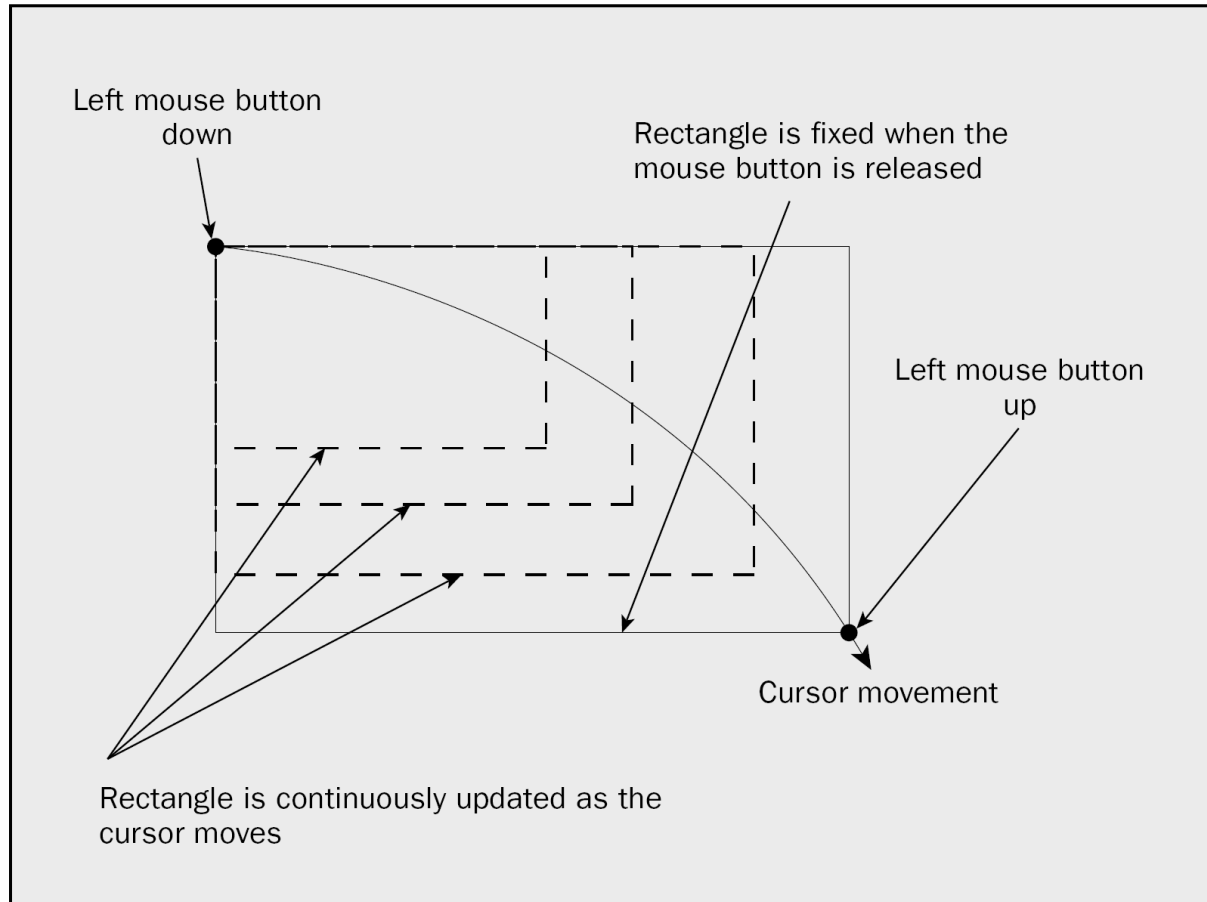


Figure 14-9

Curve

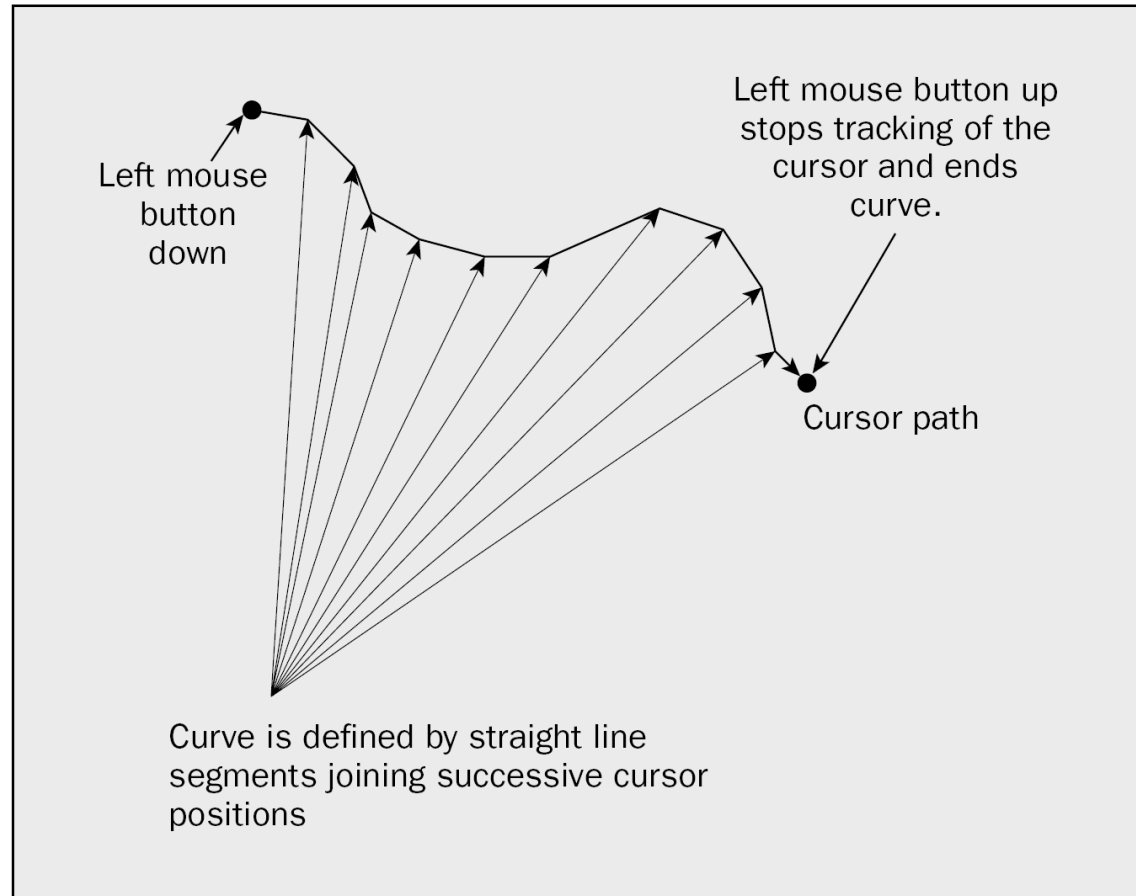


Figure 14-10

Message from the Mouse

- WM_LBUTTONDOWN
 - Left mouse button is pressed
- WM_LBUTTONUP
 - Left mouse button is released
- WM_MOUSEMOVE
 - The mouse is moved.

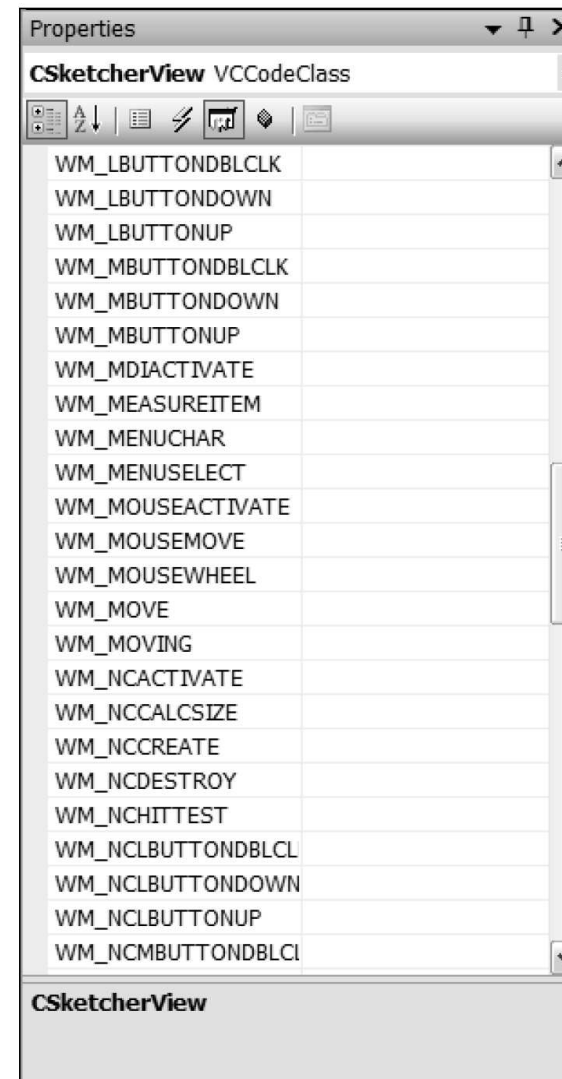


Figure 14-11

Mouse Message Handlers

- ❑ Create a handler by clicking on the ID of a mouse message.
- ❑ Then select the down arrow in its right column.
 - For example, select <add> OnLButtonUp for the WM_LBUTTONDOWN message.
 - The wizard generates the WM_LBUTTONDOWN message handler:

```
void CSketcherView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    CView::OnLButtonUp(nFlags, point);
}
```

OnMouseMove ()

```
void CSketcherView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    if (nFlags & MK_LBUTTON)
    {
        m_SecondPoint = point;
        // Test for a previous temporary element
        {
            // We get to here if there was a previous mouse move
            // so add code to delete the old element
        }

        // Add code to create new element
        // and cause it to be drawn
    }
}
```

Verify the left mouse button is down

nFlags

- MK_CONTROL
 - Ctrl key being pressed
- MK_LBUTTON
 - Left mouse button being down
- MK_MBUTTON
 - Middle mouse button being down
- MK_RBUTTON
 - Right mouse button being down
- MK_SHIFT
 - Shift key being pressed

- To test for the Ctrl key being pressed
if (nFlags & MK_CONTROL)
// Do something

bitwise AND
operator (P.80)

Storing the Position of the Cursor

- ❑ You may store the position of the cursor in the CSketcherView class.



Figure 14-12

m_FirstPoint & m_SecondPoint

- Initialization in the constructor,

```
CSketcherView::CSketcherView(): m_FirstPoint(CPoint(0,0)),  
    m_SecondPoint(CPoint(0,0))
```

- Assigning values in the message handler,

```
void CSketcherView::OnLButtonDown(UINT nFlags, CPoint point)  
{  
    // TODO: Add your message handler code here and/or call default  
    m_FirstPoint = point;        // Record the cursor position  
}
```

Defining Classes for Elements

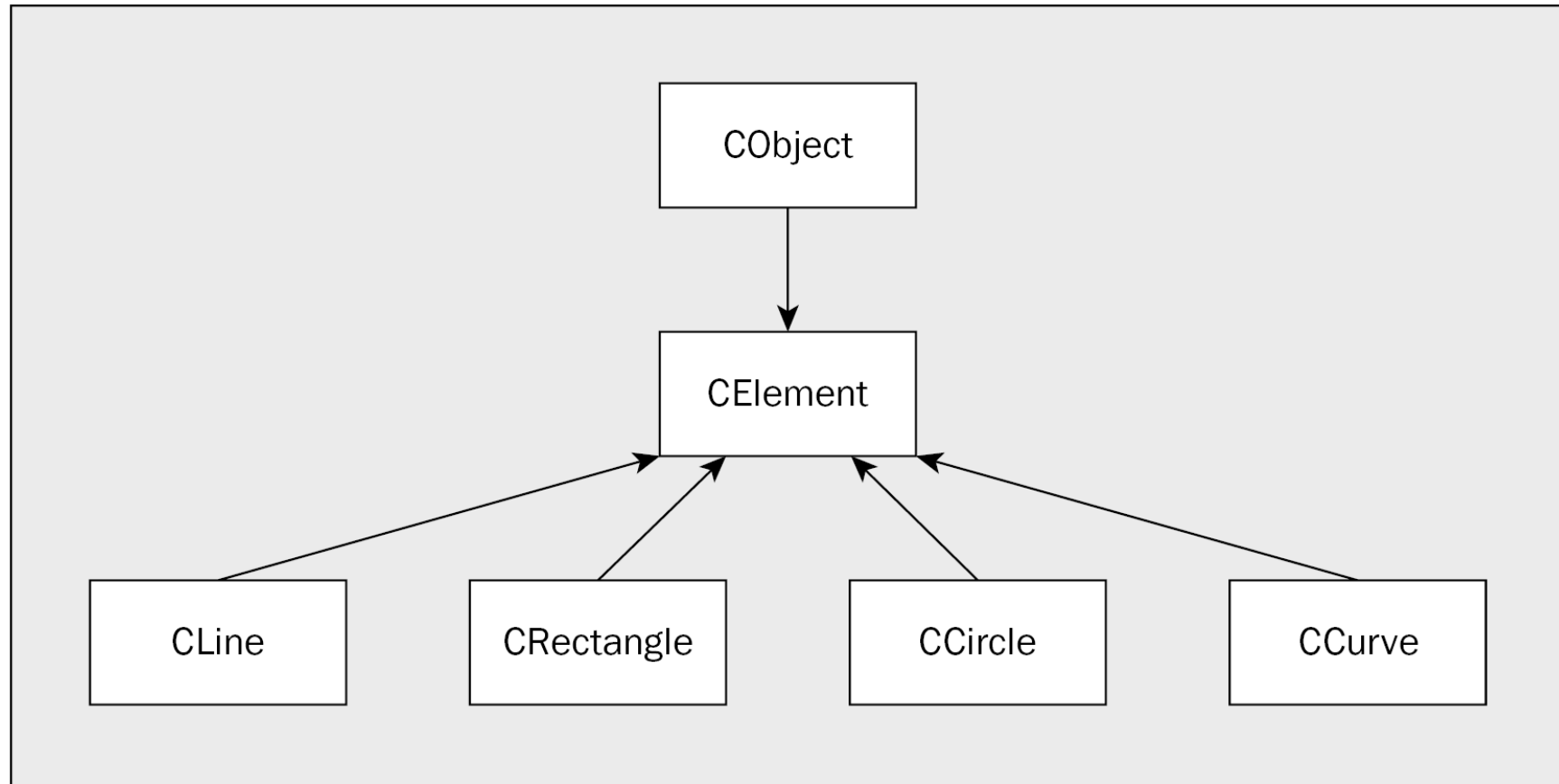


Figure 14-13

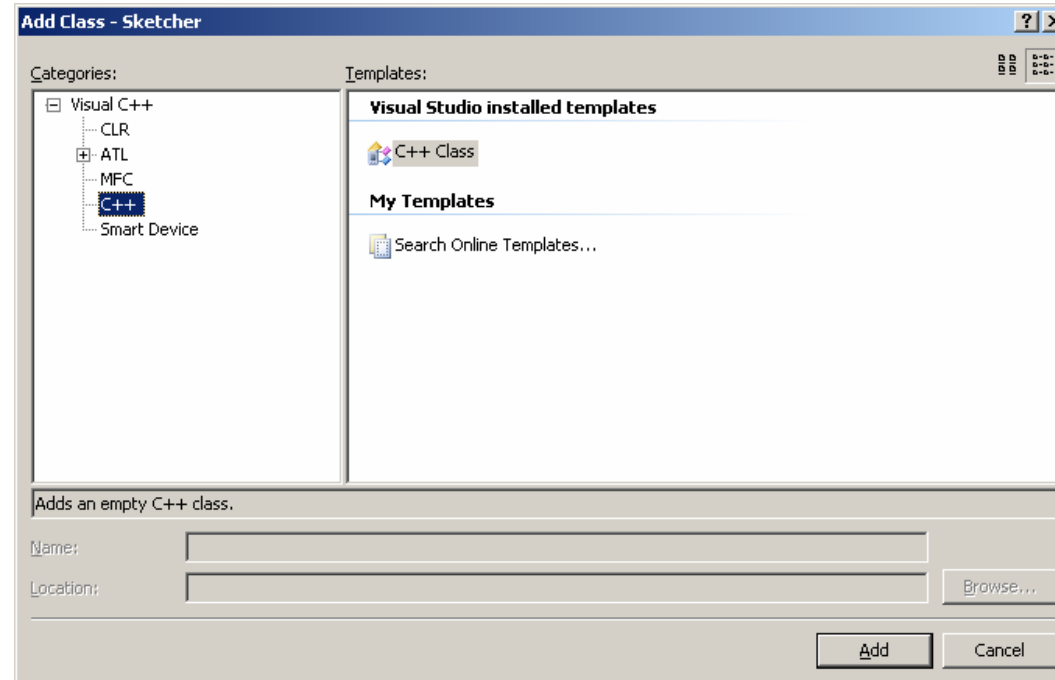
Creating the CElement Class



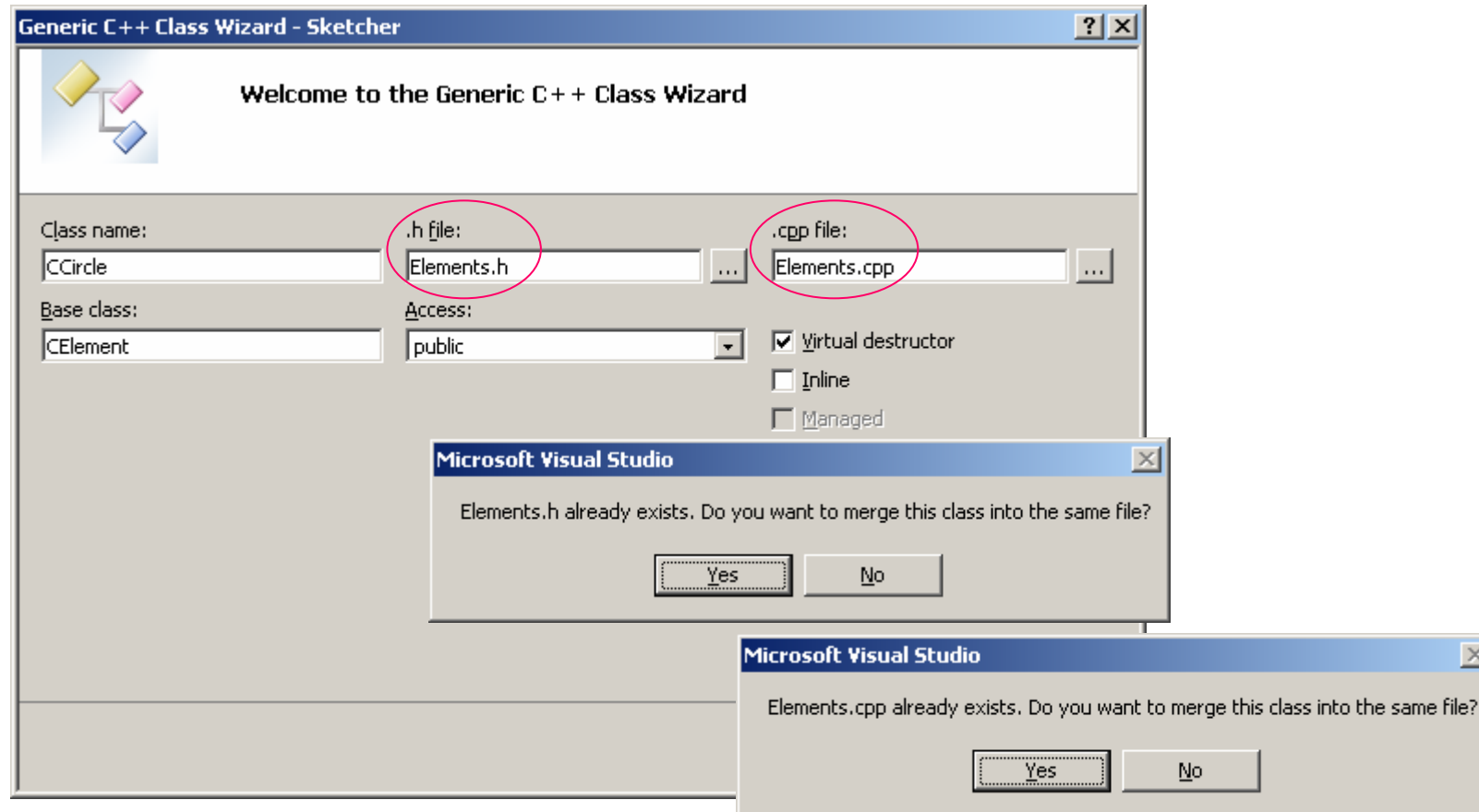
Figure 14-14

Creating the CCircle Class

- Create element classes using CElement as the base class
 - Choose the class category to be C++
 - Choose the template as C++ class



□ Create CLine, CRectangle, CCircle, CCurve



Storing a Temporary Element in the View

- In the view class, add a pointer to a CElement
 - Right-click the CSketcherView class
 - Add > Add Variable

Add Member Variable Wizard - Sketcher

Welcome to the Add Member Variable Wizard

Access: Control variable

Variable type: Control ID: Category:

Variable name: Control type: Max chars:

Min value: Max value:

.h file: .cpp file:

Comment (// notation not required):

Finish Cancel

-
- The Add Member Variable Wizard adds some code to initialize the new variable.
 - NULL fits nicely for us!

```
CSketcherView::CSketcherView()  
: m_FirstPoint(CPoint(0,0))  
 , m_SecondPoint(CPoint(0,0))  
 , m_pTempElement(NULL)  
{  
    // TODO: add construction code here  
  
}
```

Check SketcherView.h

- At the beginning, there is a line:
 - `#include "atltypes.h"`
 - The wizard assumed the CElement type is an ATL ([Active Template Library](#)) type.
- Delete this line and add the following statement:
 - `class CElement;`
 - Forward class declaration

SketcherView.cpp

- `#include "Elements.h"`
 - before `#include SketcherView.h`
- Ensure that the definition of the `CElement` class is included before the `CSketcherView` class definition.

The CElement Class

```
class CElement : public CObject
{
protected:
    COLORREF m_Color;           // Color of an element
    CElement();

public:
    virtual ~CElement();
    virtual void Draw(CDC* pDC) {}
        // Virtual draw operation

    CRect GetBoundRect();
        // Get the bounding rectangle for an element
};
```

The constructor is changed from public to protected.

The CLine Class

```
class CLine : public CElement
{
protected:
    CLine(void);          // Default constructor (should not be
                          used)
    CPoint m_StartPoint;
    CPoint m_EndPoint;

public:
    ~CLine(void);
    virtual void Draw(CDC* pDC); // Function to display a
    line

    // Constructor for a line object
    CLine(CPoint Start, CPoint End, COLORREF aColor);
};
```


Implementation of the CLine Constructor

```
CLine::CLine(CPoint Start, CPoint End,  
            COLORREF aColor)  
{  
    m_StartPoint = Start;  
    m_EndPoint = End;  
    m_Color = aColor;  
}
```

Drawing a Line

```
// Draw a CLine object
void CLine::Draw(CDC* pDC)
{
    // Create a pen for this object
    // initialize it to the object color and line width of 1 pixel
    CPen aPen;
    if(!aPen.CreatePen(PS_SOLID, m_Pen, m_Color))
    {
        // Pen creation failed. Abort the program
        AfxMessageBox(_T("Pen creation failed drawing a line"), MB_OK);
        AfxAbort();
    }

    CPen* pOldPen = pDC->SelectObject(&aPen); // Select the pen

    // Now draw the line
    pDC->MoveTo(m_StartPoint);
    pDC->LineTo(m_EndPoint);

    pDC->SelectObject(pOldPen); // Restore the old pen
}
```

Pen Width
(set in P.879)

defined in
CElement

If the pen cannot be created, display a
message box and abort the program.

Bounding Rectangles

- Not exactly coincide with the enclosing rectangles which are used to draw the elements.
- Thickness** must be taken into account.

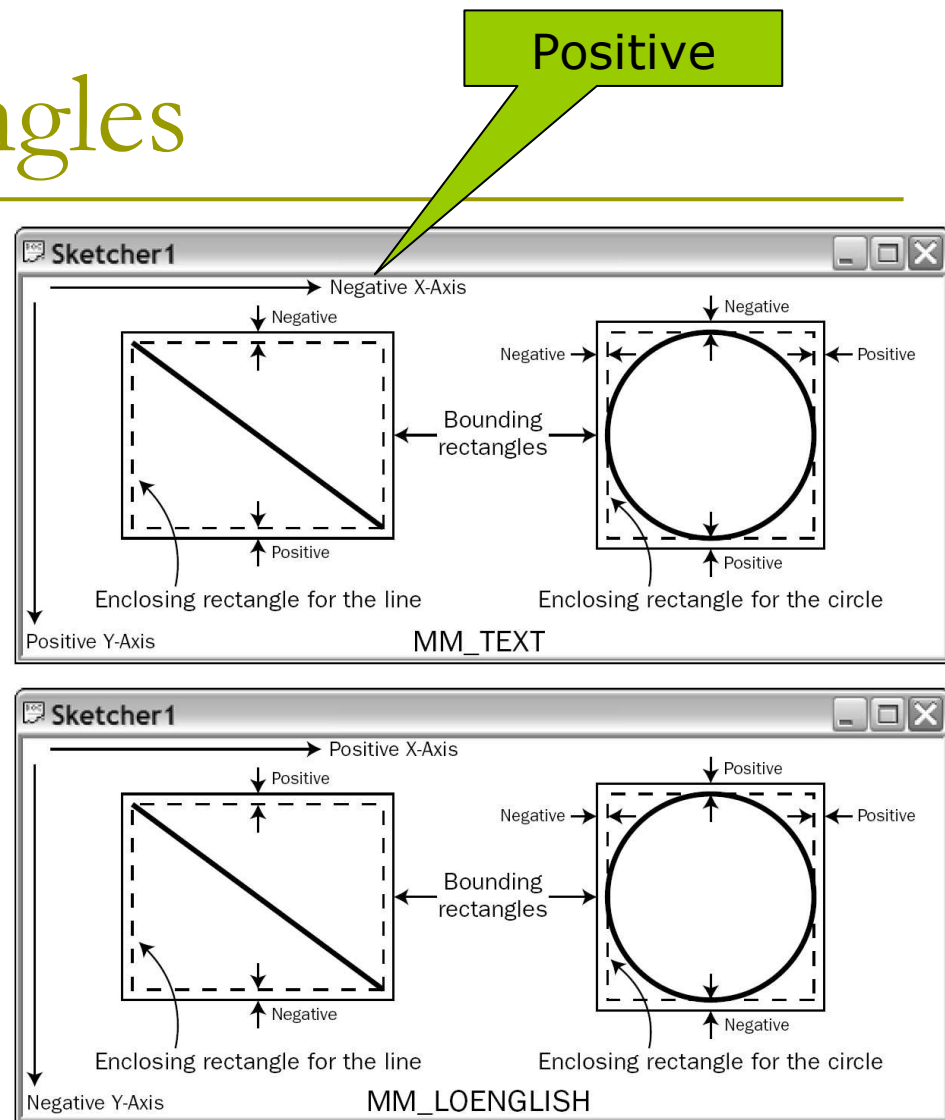


Figure 14-17

Modify the CElement Class Definition

```
class CElement : public CObject
{
protected:
    COLORREF m_Color;           // Color of an element
    CRect m_EnclosingRect;     // Rectangle enclosing an element
    int m_Pen;                 // Pen width

    CElement();

public:
    virtual ~CElement();
    virtual void Draw(CDC* pDC) {} // Virtula draw operation

    CRect GetBoundRect();
        // Get the bounding rectangle for an element
};
```

Update the CLine Constructor

```
CLine::CLine(CPoint Start, CPoint End,  
            COLORREF aColor)  
{  
    m_StartPoint = Start;  
    m_EndPoint = End;  
    m_Color = aColor;  
    m_Pen = 1;  
}
```

GetBoundRect ()

- ▣ Assuming the MM_TEXT mapping mode:

```
// Get the bounding rectangle for an element
CRect CElement::GetBoundRect()
{
    CRect BoundingRect;
        // Object to store bounding rectangle
    BoundingRect = m_EnclosingRect;
        // Store the enclosing rectangle

    // Increase the rectangle by the pen width
    BoundingRect.InflateRect(m_Pen, m_Pen);
    return BoundingRect;
        // Return the bounding rectangle
}
```

Enlarge the Enclosing Rectangle by the Pen Width

- ❑ `BoundingBox.InflateRect(m_Pen, m_Pen);`
- ❑ `BoundingBox = m_EnclosingRect +
CRect(m_Pen, m_Pen, m_Pen, m_Pen);`
- ❑ `BoundingBox = m_EnclosingRect;`
- ❑ `BoundingBox.top -= m_Pen;`
- ❑ `BoundingBox.left -= m_Pen;`
- ❑ `BoundingBox.bottom += m_Pen;`
- ❑ `BoundingBox.right += m_Pen;`

Calculating the Enclosing Rectangle for a Line

```
CLine::CLine(CPoint Start, CPoint End,
             COLORREF aColor)
{
    m_StartPoint = Start;
    m_EndPoint = End;
    m_Color = aColor;
    m_Pen = 1;

    // Define the enclosing rectangle
    m_EnclosingRect = CRect(Start, End);
}
```


The CRectangle Class

```
class CRectangle :
    public CElement
{
public:
    ~CRectangle(void);
    virtual void Draw(CDC* pDC);
        // Function to display a rectangle

        // Constructor for a rectangle object
    CRectangle(CPoint Start, CPoint End, COLORREF aColor);

protected:
    CRectangle(void);
        // Default constructor - should not be used
};
```

The CRectangle Class Constructor

- Similar to that for a CLine constructor

```
// CRectangle class constructor
CRectangle:: CRectangle(CPoint Start, CPoint End,
    COLORREF aColor)
{
    m_Color = aColor;           // Set rectangle color
    m_Pen = 1;                 // Set pen width

    // Define the enclosing rectangle
    m_EnclosingRect = CRect(Start, End);
}
```

Drawing a Rectangle

```
void CRectangle::Draw(CDC* pDC)
{
    // Create a pen for this object and
    // initialize it to the object color and line width of 1 pixel
    CPen aPen;
    if(!aPen.CreatePen(PS_SOLID, m_Pen, m_Color))
    {
        // Pen creation failed
        AfxMessageBox(_T("Pen creation failed drawing a rectangle"), MB_OK);
        AfxAbort();
    }

    // Select the pen
    CPen* pOldPen = pDC->SelectObject(&aPen);
    // Select the brush
    CBrush* pOldBrush = (CBrush*)pDC->SelectStockObject(NULL_BRUSH);

    // Now draw the rectangle
    pDC->Rectangle(m_EnclosingRect);

    pDC->SelectObject(pOldBrush);           // Restore the old brush
    pDC->SelectObject(pOldPen);           // Restore the old pen
}
```

The CCircle Class

- Similar to the CRectangle class

```
class CCircle : public CElement
{
public:
    ~CCircle(void);
    virtual void Draw(CDC* pDC);
    // Function to display a circle

    // Constructor for a circle object
    CCircle(CPoint Start, CPoint End, COLORREF aColor);

protected:
    CCircle(void);
    // Default constructor - should not be used
};
```

Implementing the CCircle Class

- ❑ The point you press the left mouse button is the center
- ❑ The point you release the left mouse button is a point on the circumference.
- ❑ We need to design the constructor to convert this to the enclosing rectangle of a circle.

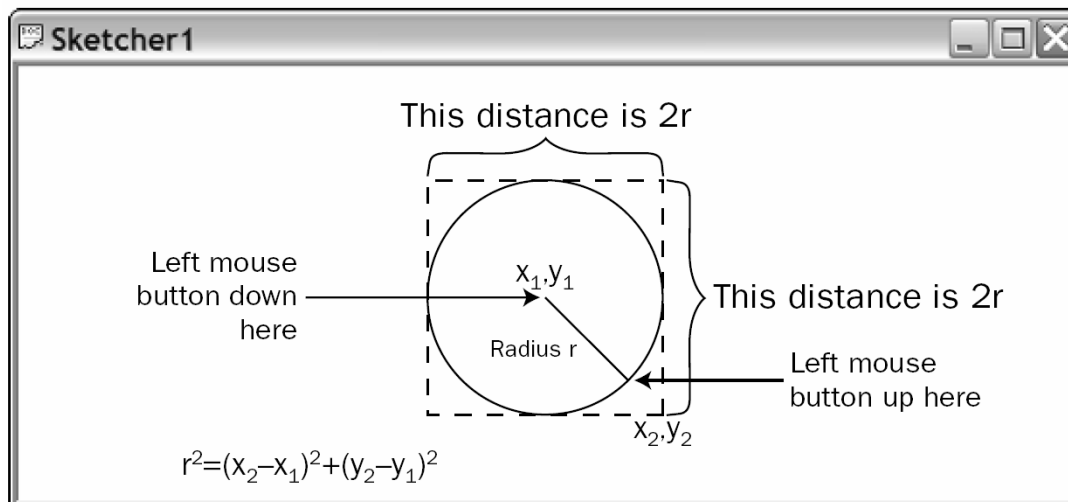


Figure 14-18

```
#include <math.h> in  
Elements.cpp
```

The CCircle Class Constructor

```
CCircle::CCircle(CPoint Start, CPoint End, COLORREF aColor)  
{  
    // First calculate the radius  
    // We use floating point because that is required by the  
    // library function (in math.h) for calculating a square root.  
    long Radius = static_cast<long> (sqrt(  
        static_cast<double>((End.x-Start.x)*(End.x-Start.x)+  
            (End.y-Start.y)*(End.y-Start.y))));  
  
    // Now calculate the rectangle enclosing  
    // the circle assuming the MM_TEXT mapping mode  
    m_EnclosingRect = CRect(Start.x-Radius, Start.y-Radius,  
        Start.x+Radius, Start.y+Radius);  
  
    m_Color = aColor;           // Set the color for the circle  
    m_Pen = 1;                 // Set pen width to 1  
}
```

Drawing a Circle

```
void CCircle::Draw(CDC* pDC)
{
    // Create a pen for this object and
    // initialize it to the object color and line width of 1 pixel
    CPen aPen;
    if(!aPen.CreatePen(PS_SOLID, m_Pen, m_Color))
    {
        // Pen creation failed
        AfxMessageBox(_T("Pen creation failed drawing a circle"), MB_OK);
        AfxAbort();
    }

    CPen* pOldPen = pDC->SelectObject(&aPen); // Select the pen

    // Select a null brush
    CBrush* pOldBrush = (CBrush*)pDC->SelectStockObject(NULL_BRUSH);

    // Now draw the circle
    pDC->Ellipse(m_EnclosingRect);

    pDC->SelectObject(pOldPen); // Restore the old pen
    pDC->SelectObject(pOldBrush); // Restore the old brush
}
```

Setting the Drawing Mode

- SetROP2()
 - **Set Raster OPeration to**
- R2_BLACK
 - All drawing is in black
- R2_WHITE
 - All drawing is in white
- R2_NOP
 - Drawing operations do nothing
- R2_COPYPEN
 - Drawing is in the pen color. This is the **default**.
- R2_XORPEN
 - Drawing is in the color produced by exclusive ORing the pen color and the background color.
- R2_NOTXORPEN
 - Drawing is in the color that is the inverse of the R2_XORPEN color.

R2_NOTXORPEN

- If you draw the same shape again, the shape disappears.

	R	G	B
Background – white	1	1	1
Pen – red	1	0	0
XORed	0	1	1
NOT XOR – produces red	1	0	0

	R	G	B
Background – red	1	0	0
Pen – red	1	0	0
XORed	0	0	0
NOT XOR – produces white	1	1	1

Coding the OnMouseMove () Handler

- Using the CClientDC class rather than CDC
 - It automatically destroys the device context when you are done.

```
void CSketcherView::OnMouseMove(UINT nFlags, CPoint point)
{ // Define a Device Context object for the view
  CClientDC aDC(this); // DC is for this view
  aDC.SetROP2(R2_NOTXORPEN); // Set the drawing mode
  if((nFlags & MK_LBUTTON) && (this == GetCapture()))
  { m_SecondPoint = point;
    if(m_pTempElement)
    { // Redraw the old element so it disappears from the view
      m_pTempElement->Draw(&aDC);
      delete m_pTempElement; // Delete the old element
      m_pTempElement = 0; // Reset the pointer to 0
    }
    // Create a temporary element of the type and color that
    // is recorded in the document object, and draw it
    m_pTempElement = CreateElement(); // Create a new element
    m_pTempElement->Draw(&aDC); // Draw the element
  }
}
```

defined in CSketcherView (P.873)

will be defined on P.890

CreateElement()

The screenshot shows the 'Add Member Function Wizard - Sketcher' dialog box. The title bar reads 'Add Member Function Wizard - Sketcher'. The main heading is 'Welcome to the Add Member Function Wizard'. The dialog contains the following fields and controls:

- Return type:** A dropdown menu set to 'CElement*'. To its right is the **Function name:** text box containing 'CreateElement'.
- Parameter type:** A dropdown menu set to 'void', which is circled in red. To its right is the **Parameter name:** text box, which is empty.
- Parameter list:** An empty text box for listing parameters. Below it are 'Add' and 'Remove' buttons.
- Access:** A dropdown menu set to 'protected'. To its right are checkboxes for **Static**, **Virtual**, **Pure**, and **Inline**, all of which are currently unchecked.
- .cpp file:** A text box containing 'sketcherview.cpp' with a browse button ('...') to its right.
- Comment (// notation not required):** A text box containing 'Create a new element on the heap'.
- Function signature:** A text box containing 'CElement* CreateElement(void)'.

At the bottom right of the dialog are 'Finish' and 'Cancel' buttons.

Implementing CreateElement()

```
CElement* CSketcherView::CreateElement(void)
{ // Get a pointer to the document for this view
  CSketcherDoc* pDoc = GetDocument();

  // Now select the element using the type stored in the document
  switch(pDoc->GetElementType())
  {
    case RECTANGLE:
      return new CRectangle(m_FirstPoint, m_SecondPoint,
        pDoc->GetElementColor());

    case CIRCLE:
      return new CCircle(m_FirstPoint, m_SecondPoint,
        pDoc->GetElementColor());

    case LINE:
      return new CLine(m_FirstPoint, m_SecondPoint,
        pDoc->GetElementColor());

    default:
      // Something's gone wrong
      AfxMessageBox(_T("Bad Element code"), MB_OK);
      AfxAbort();
      return NULL;
  }
}
```

will be defined on P.891

GetElementType ()

```
class CSketcherDoc : public CDocument
{
    // Rest of the class definition as before ...

    // Operations
public:
    // Get the element type
    unsigned int GetElementType() { return m_Element; }

    // Get the element color
    COLORREF GetElementColor() { return m_Color; }

    // Rest of the class definition as before

};
```

Dealing with WM_LBUTTONDOWN

```
void CSketcherView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // Make sure there is an element
    if (m_pTempElement)
    {
        // Call a document class function to store the element
        // pointed to by m_pTempElement in the document object

        delete m_pTempElement;
        m_pTempElement = 0;
    }

    CView::OnLButtonUp(nFlags, point);
}
```

Exercising Sketcher

- Download a partially finished project [sketcher-ch15.zip](#)
 - Uncompress the Sketcher folder to `Your Documents\Visual Studio 2008\Projects`
 - CLine is implemented. You may compile and run it.
 - Try to complete the implementation of CCircle & CCurve.