Chapter 14

Drawing in a Window

The Window Client Area (P.664)

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

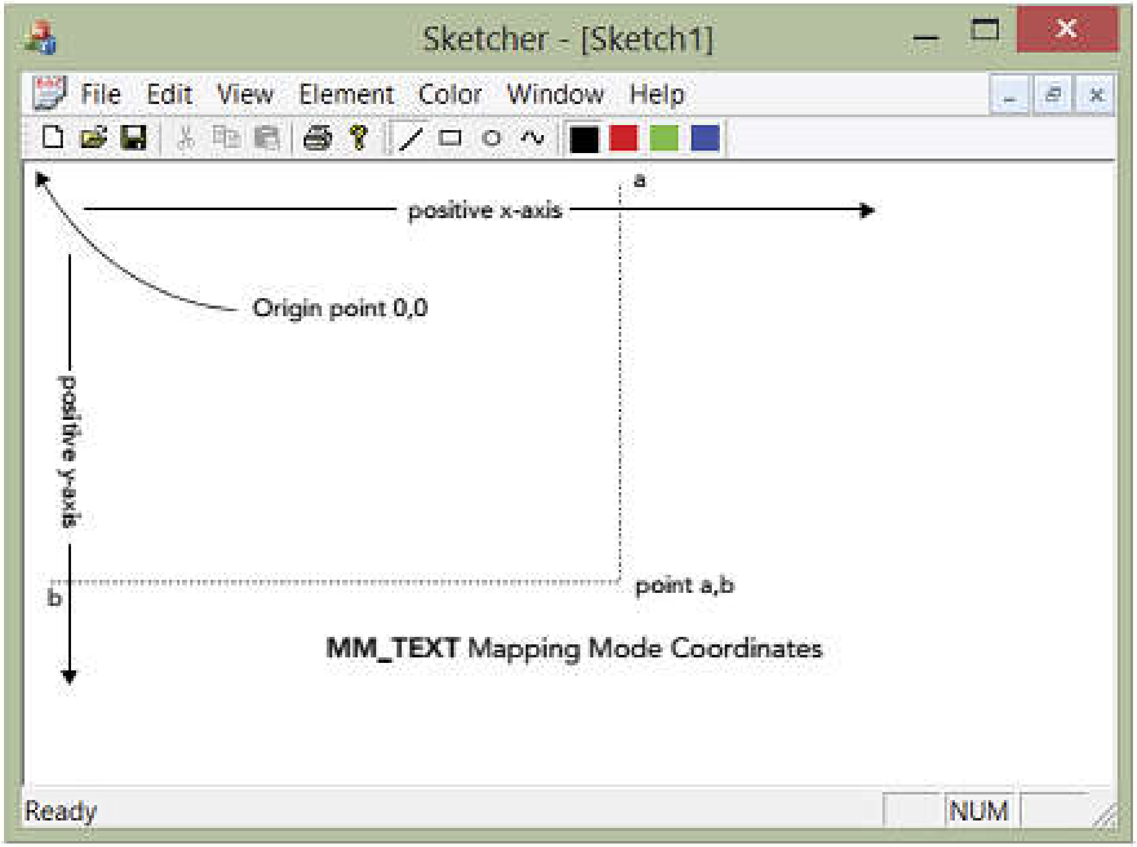


FIGURE 14-1

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) P.665

- 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.

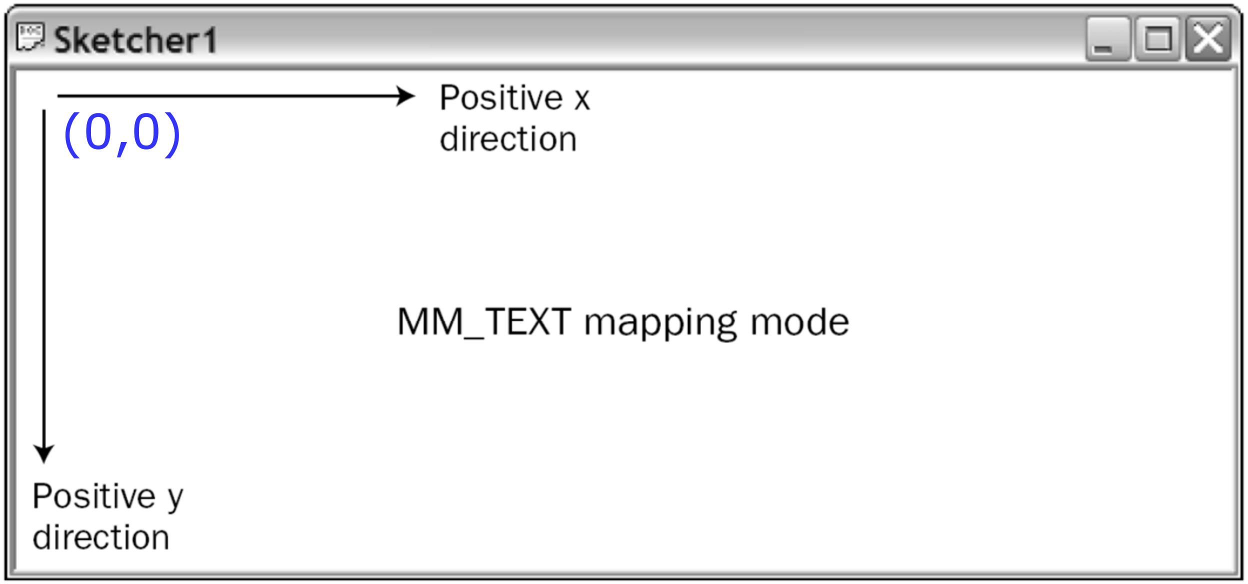


Figure 14-1

Mapping Modes (2)

- MM_LOENGLISH (P.667)
  - 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.

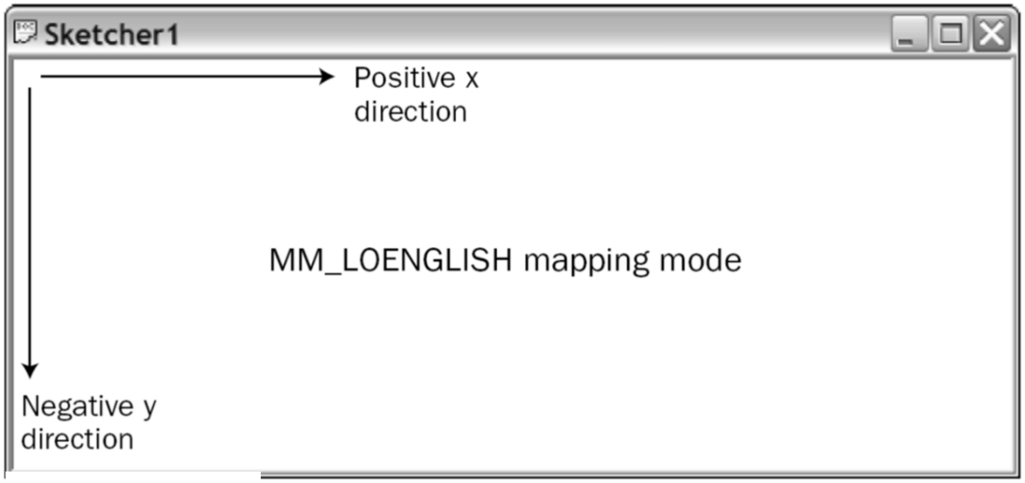


Figure 14-2

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

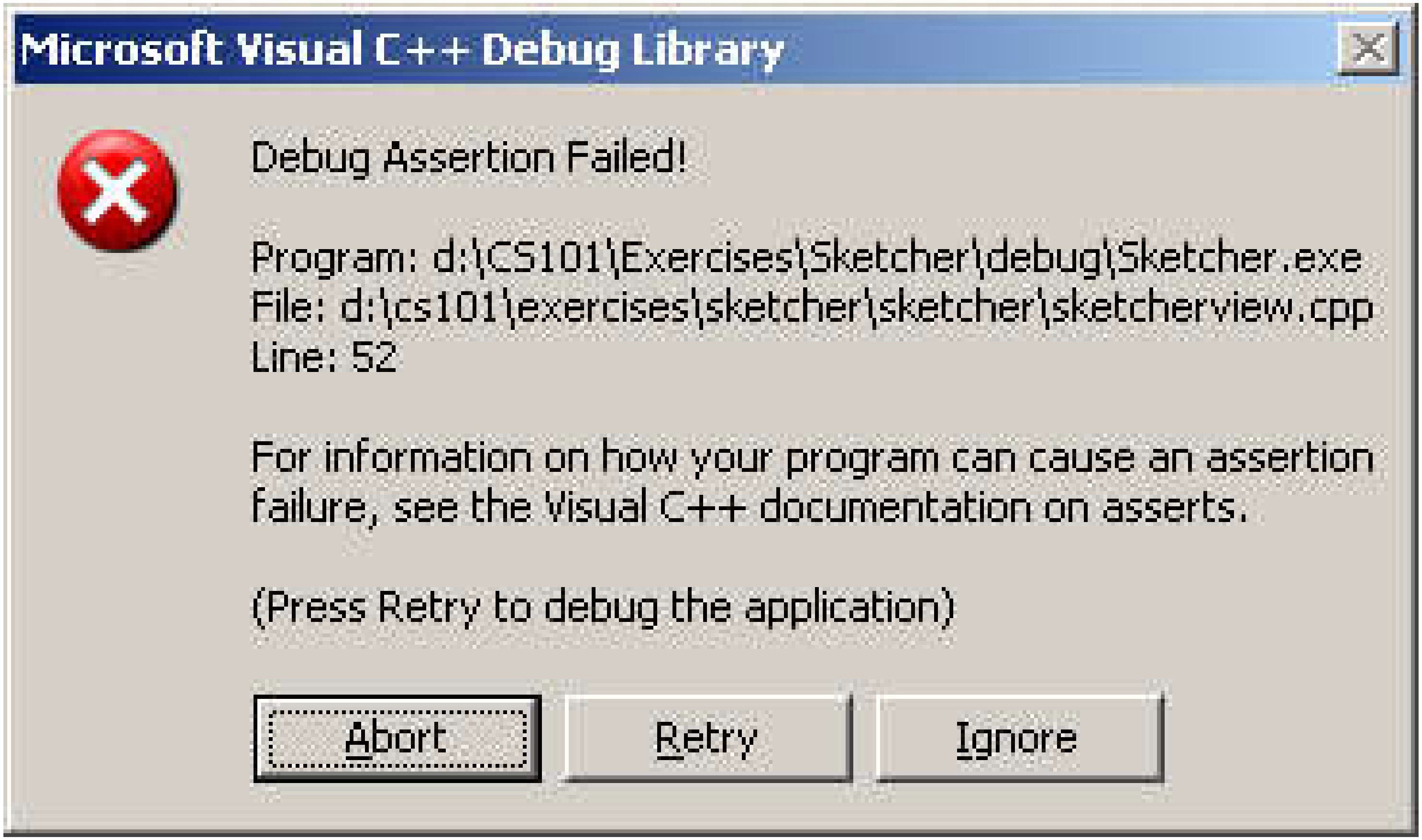
```
void CSketcherView::OnDraw(CDC* pDC)// P.668
{
  CSketcherDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
     return;
// TODO: add draw code for native data here
}
```

Returns the address of the document object related to the current view (P.616)

Make sure the pointer pDoc contains a valid address.

Make sure the pointer pDoc is not null.

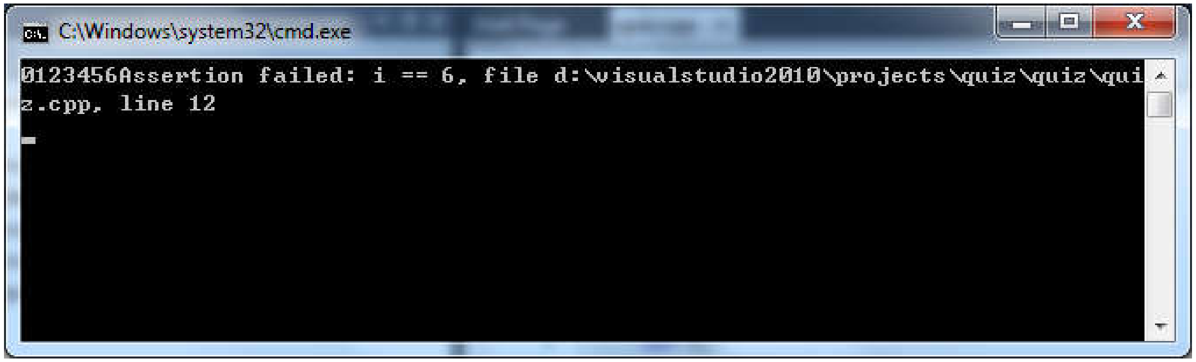
Assertion Failed



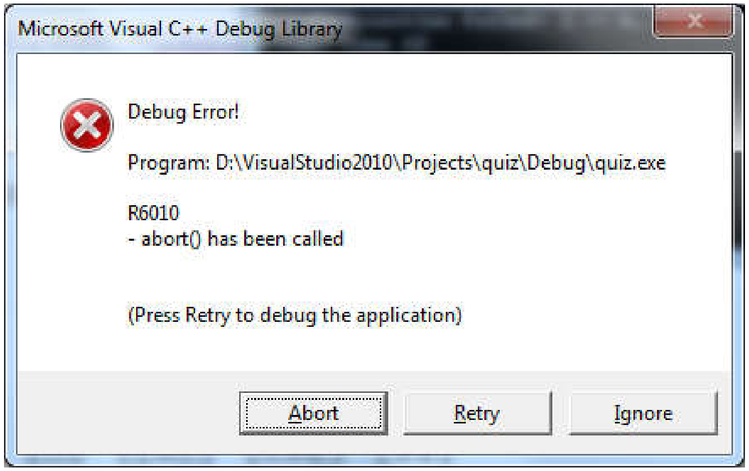
```
Debug Assertion Failed!

Program: d:\CS101\Exercises\Sketcher\debug\Sketcher.exe
File: d:\cs101\exercises\sketcher\sketcher\sketcherview.cpp
Line: 52

For information on how your program can cause an assertion
failure, see the Visual C++ documentation on asserts.

(Press Retry to debug the application)
```

Using Assertions (P.708)

```
#include <iostream>
#include <cassert>

using std::cout;
using std::endl;

int main()
{
    int i;
    for (i=0; i<=6; i++)
        cout << i;
    assert(i == 6);
    return 0;
}
```



```
0123456Assertion failed: i == 6, file d:\visualstudio2010\projects\quiz\quiz\qui
z.cpp, line 12
```



Debug Error!

Program: D:\VisualStudio2010\Projects\quiz\Debug\quiz.exe

R6010
- abort() has been called

(Press Retry to debug the application)

The CDC Class (P.669)

- 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.

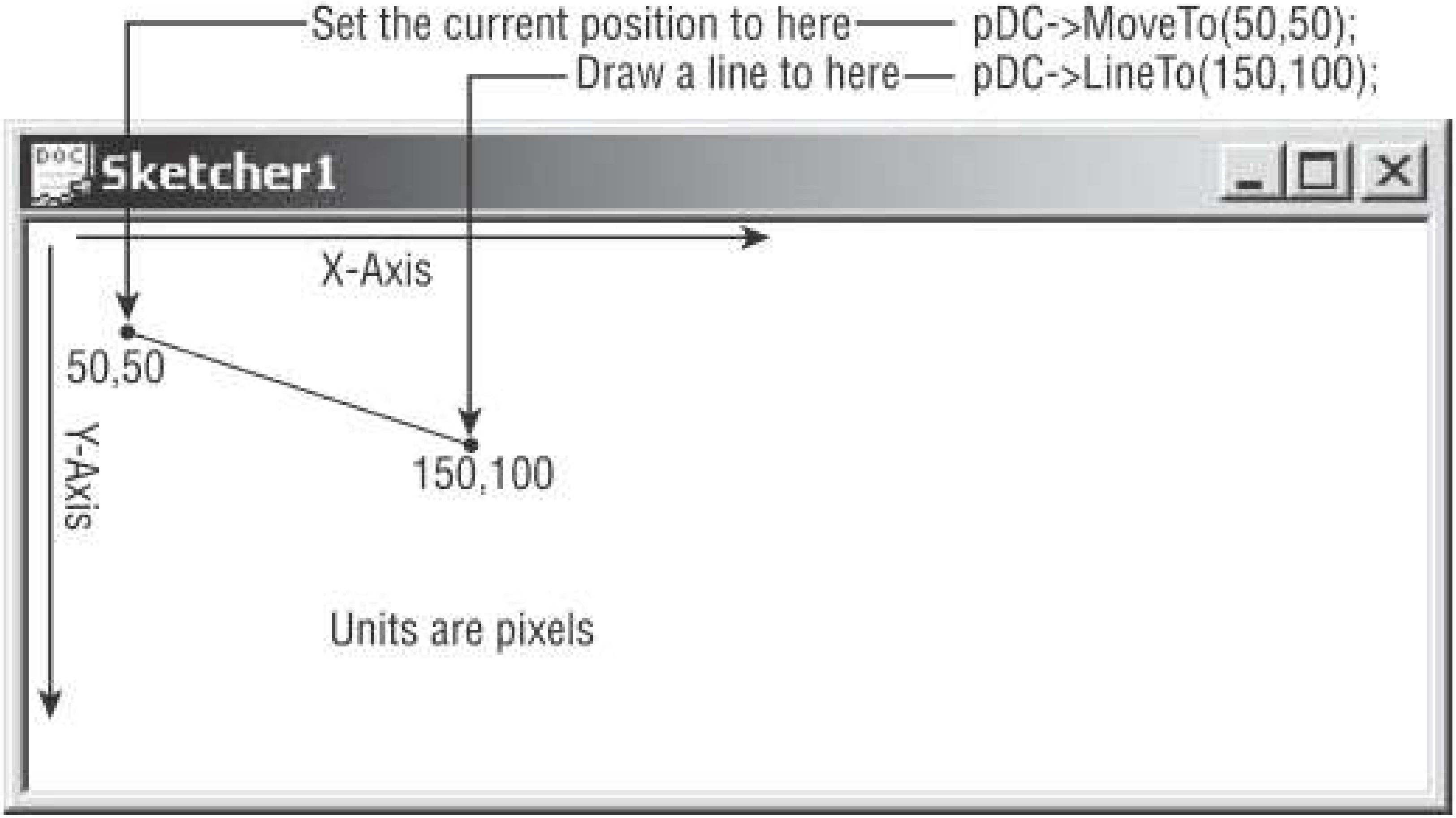
MoveTo()

- 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 (P.670)



```
pDC->MoveTo(50,50);
pDC->LineTo(150,100);
```

Figure 14-3

LineTo()

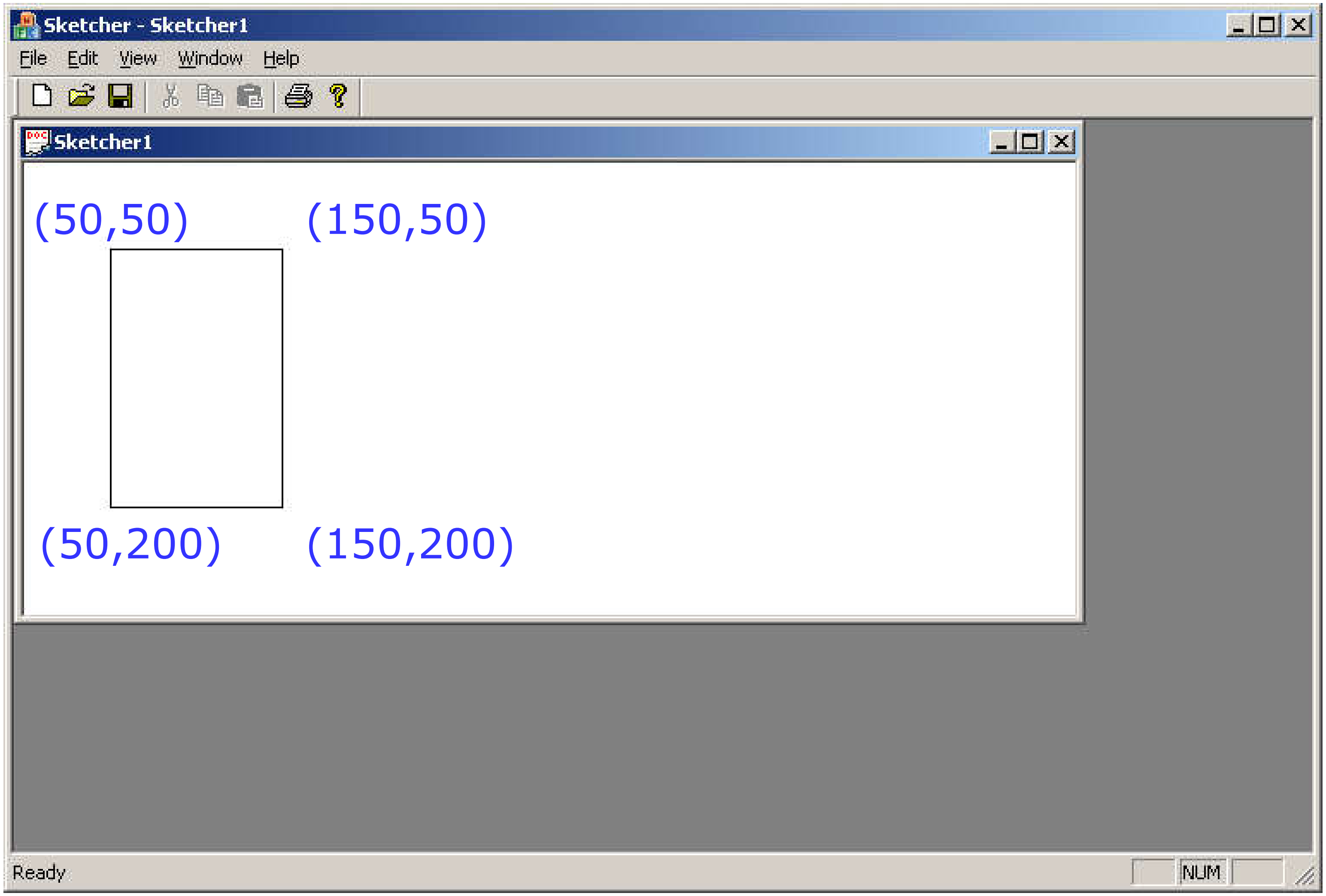
- The CDC class also defines two versions of the LineTo() function
  - BOOL LineTo(int x, int y);
  - BOOL LineTo(POINT aPoint);
    - You may use either a POINT struct or a CPoint object as the argument.

Ex16_1 (P.671)

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

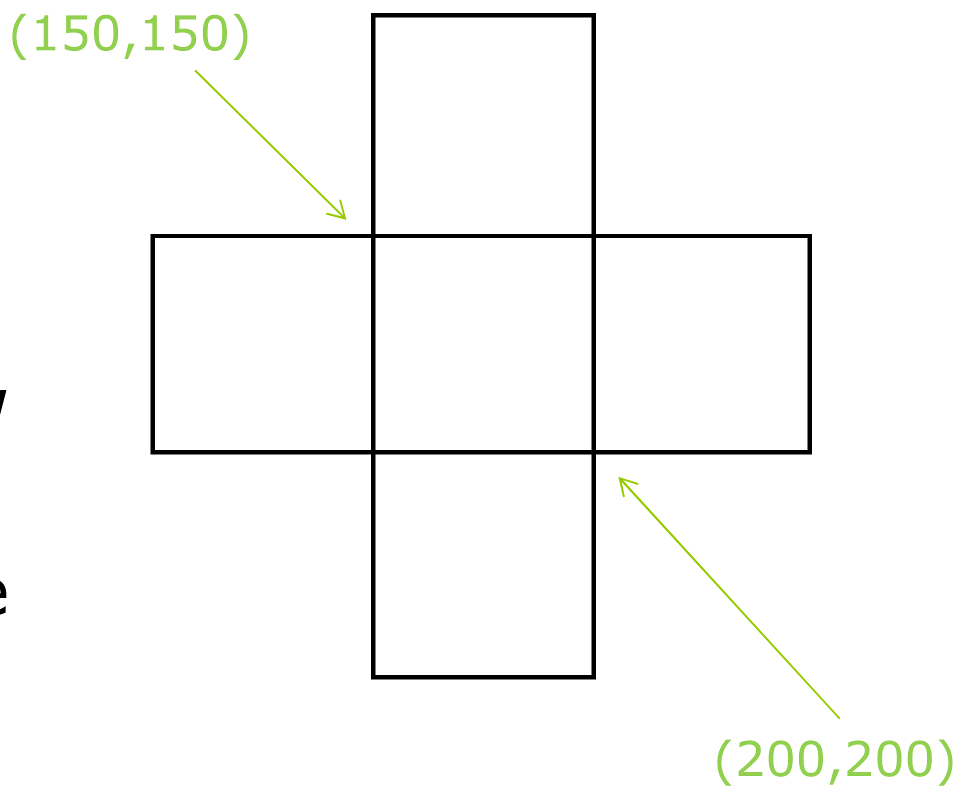
```
void CSketcherView::OnDraw(CDC* pDC)
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
      return;

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

Figure 14-4 (P.671)



Exercise: Lines and Rectangles

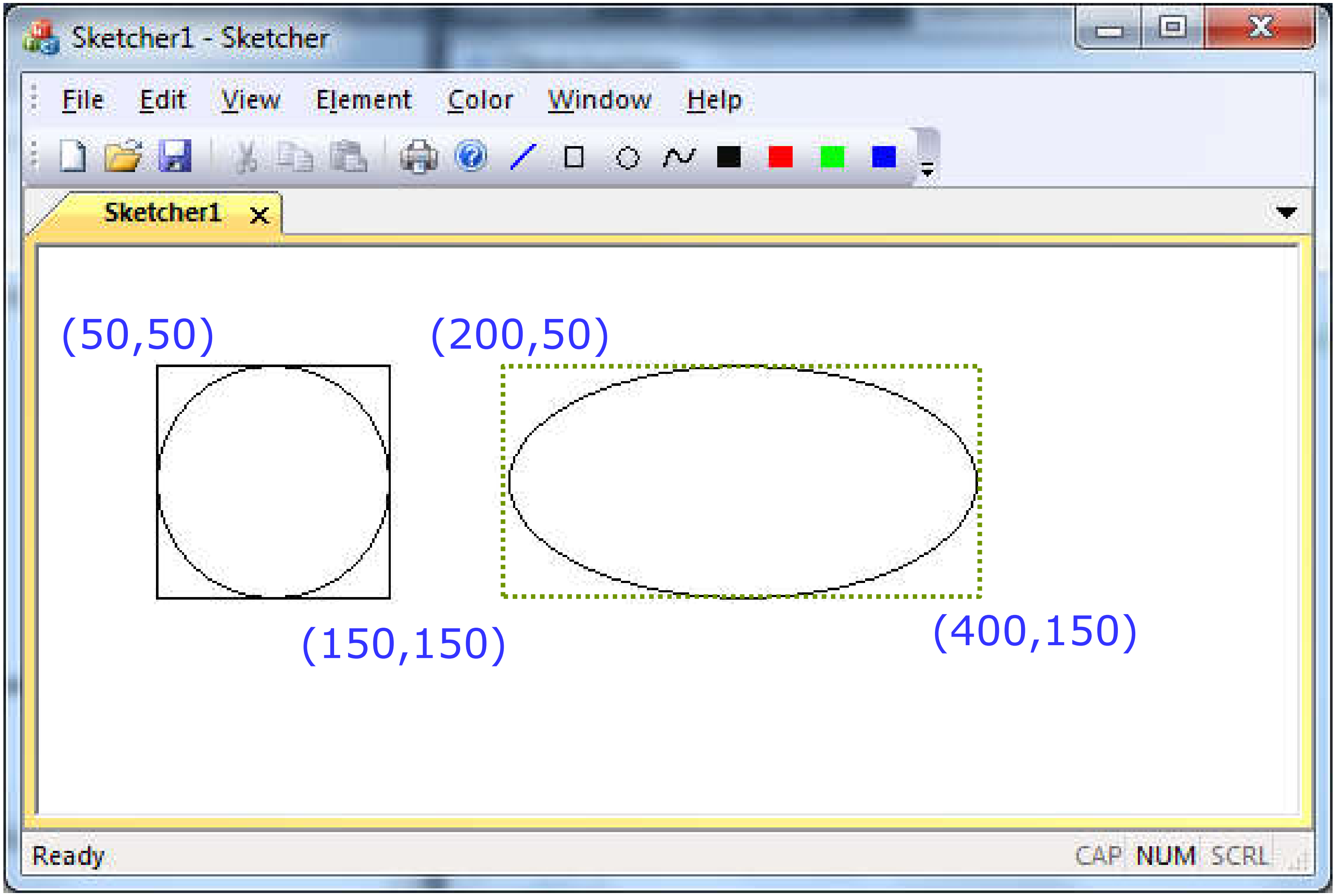
- Create an MFC application.
- Modify the OnDraw() member function of your View class, to draw a figure like this.
  - The coordinates are for your reference. You don't need to show them.



Drawing Rectangles & Circles

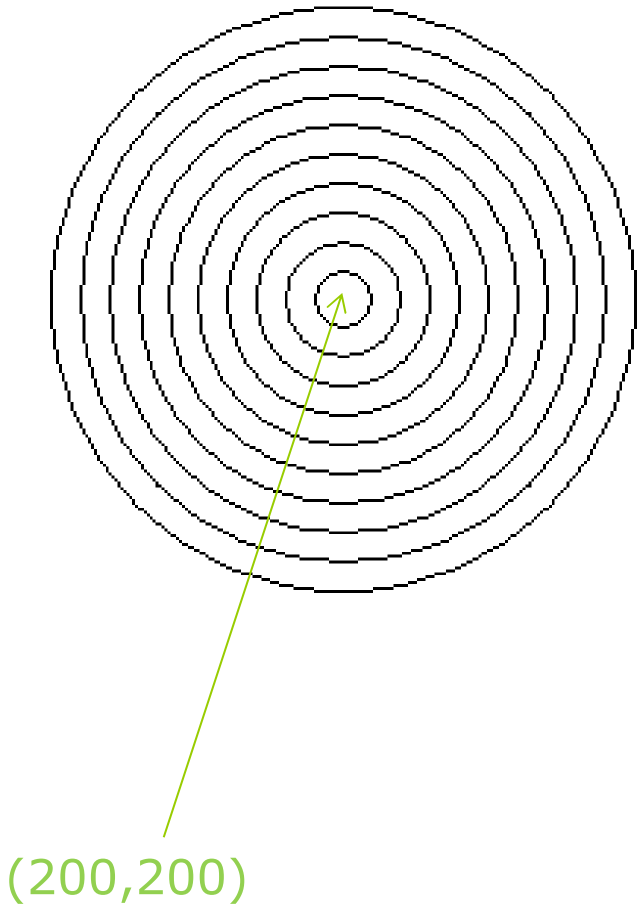
```
void CSketcherView::OnDraw(CDC* pDC)
{
  CSketcherDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
      return;

  pDC->Rectangle(50,50, 150, 150);
  pDC->Ellipse(50,50, 150,150);
  pDC->Ellipse(200,50, 400,150);

}
```

A circle is a special ellipse



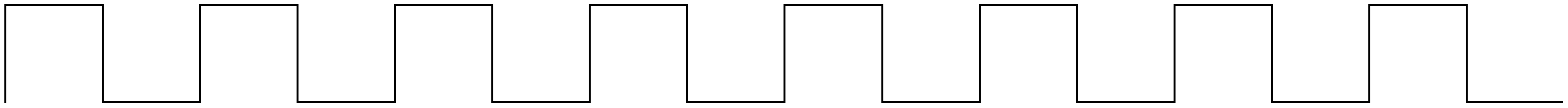
Exercise: Circles

- Use a for-loop in OnDraw() to draw a figure like this.
- Note that a rectangle or an ellipse has a solid background color (default to be white). Therefore, if you plot the smaller circles first, they will be covered by larger ones.



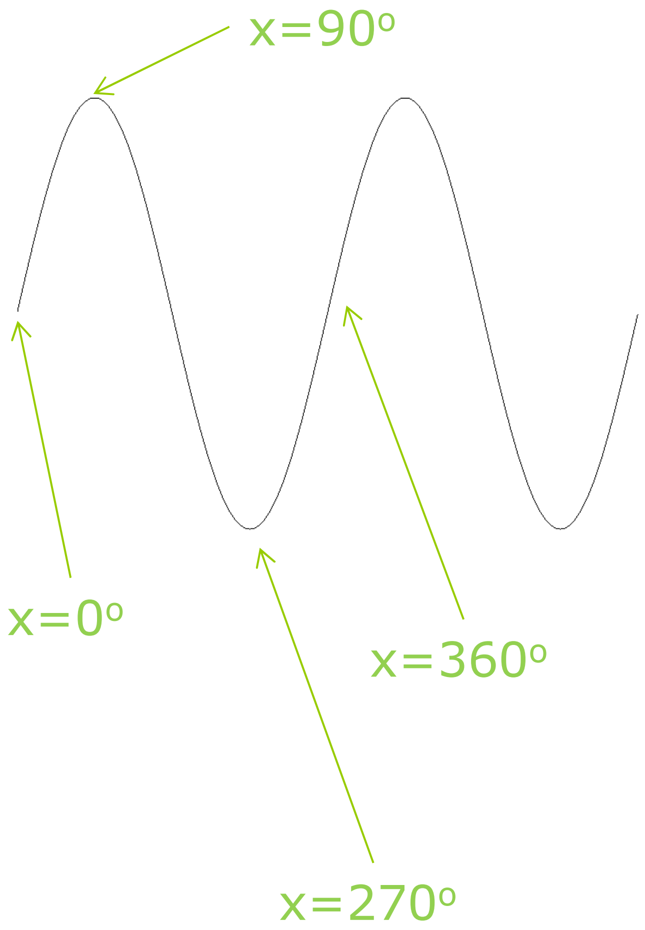
Exercise: Square Wave

- Write a program to draw the square wave below.
- Observe the pattern. You can see it is a repetition of 8 periods, so you can use a for-loop to easy repeat the same pattern.



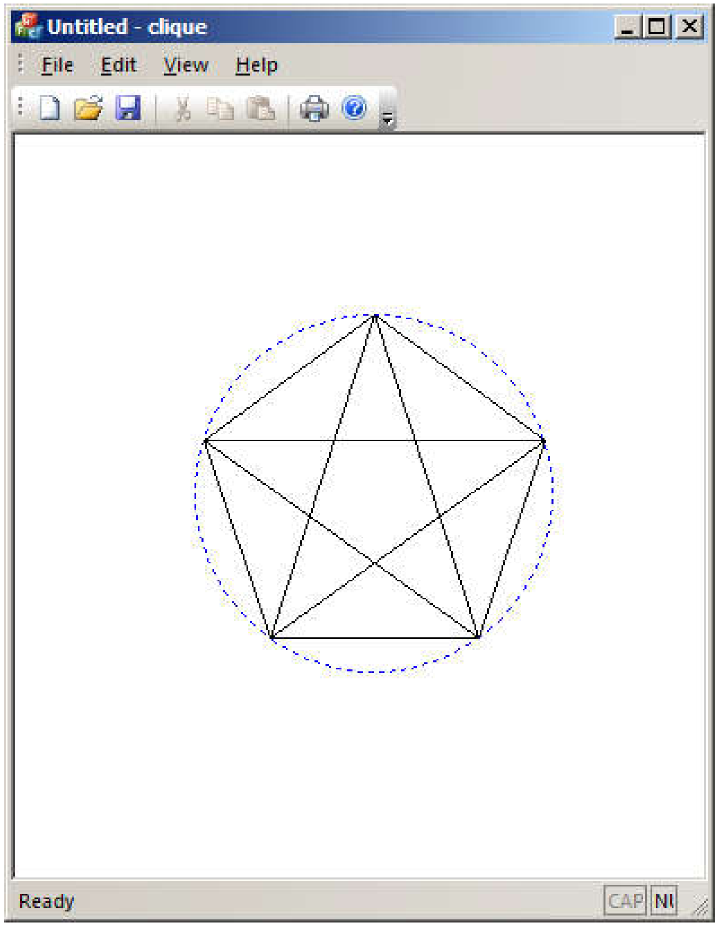
Exercise: Sine Wave

- Write a program to draw the sine wave from 0 degree to 720 degree.
- Recall that you learned in Calculus class that, you can approximate a smooth curve by a series of line segments.



Exercise: Drawing a Polygon

- Use LineTo() and Ellipse() to draw the following figure.
- Be advised to design your drawing function to be generic enough, because a few weeks later you will re-use this function to draw polygons with 5 vertices, 7 vertices, or 3 vertices.



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 aRect (250,50,300,100);
   CPoint Start(275,100);
   CPoint End(250,75);
   pDC->Arc(&aRect, Start, End);
}
```

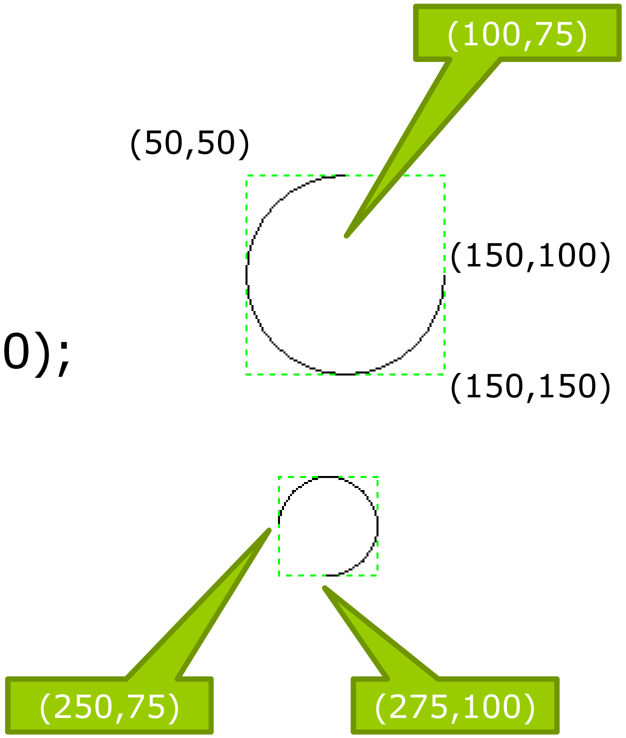
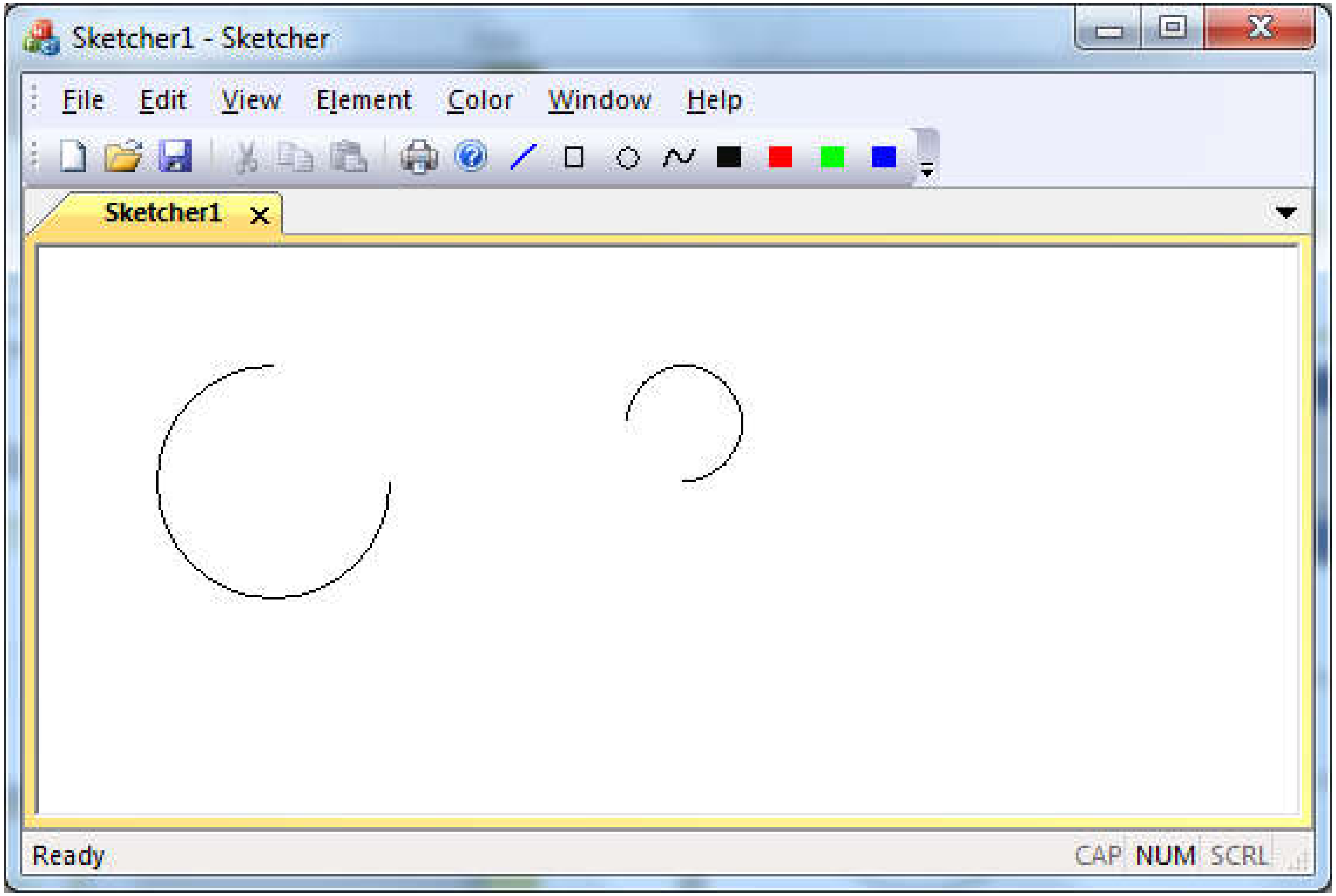
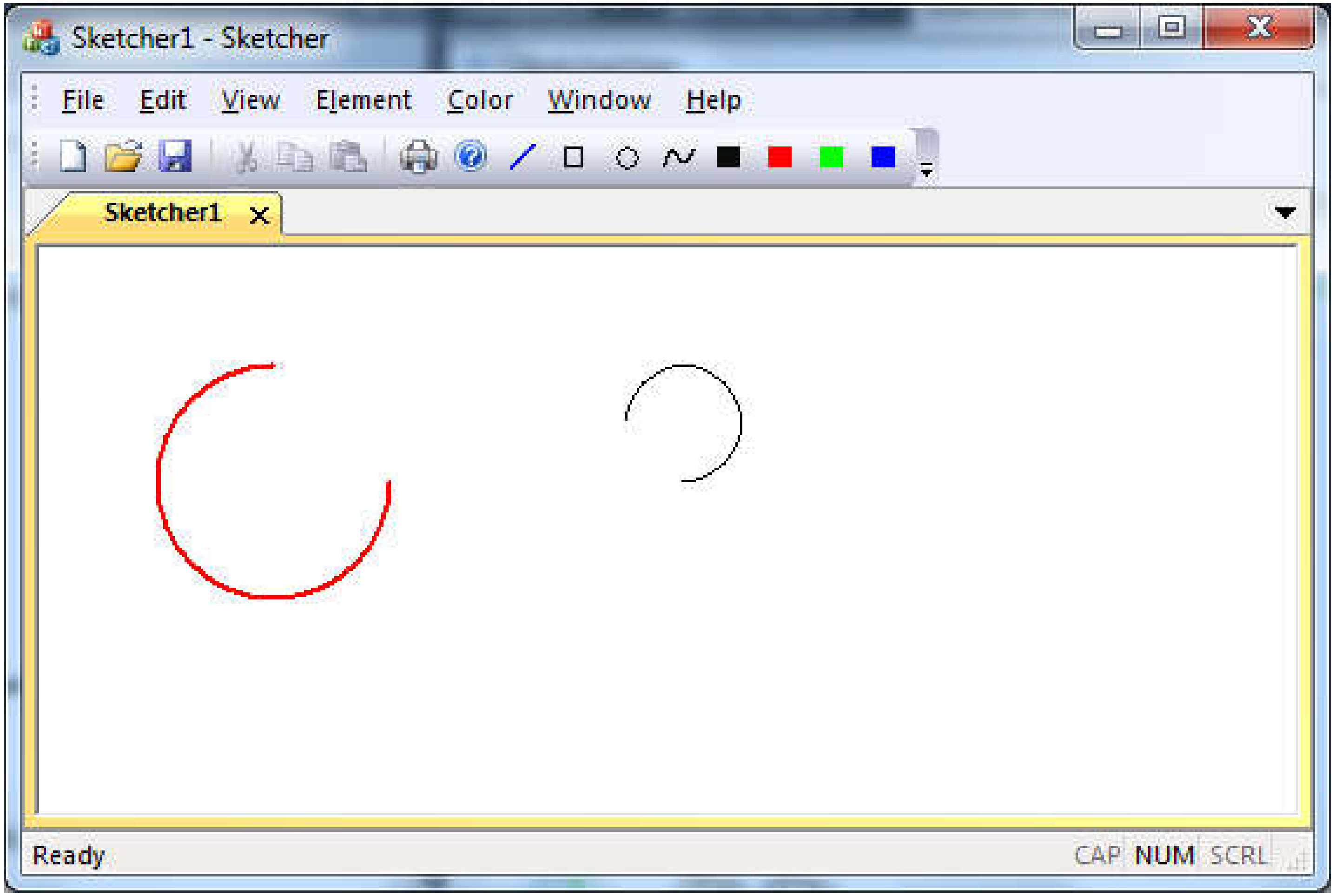


Figure 14-5 (P.673)



Drawing in Color



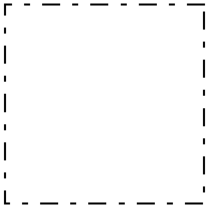
Using a Pen

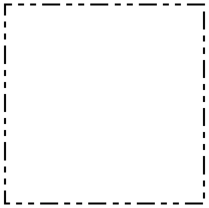
- Declare a pen object and initialize it as a red solid pen drawing a line 2 pixels wide (P.675)

```
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 aRect(250,50,300,100);
CPoint Start(275,100);
CPoint End(250,75);
pDC->Arc(&aRect, Start, End);
```

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



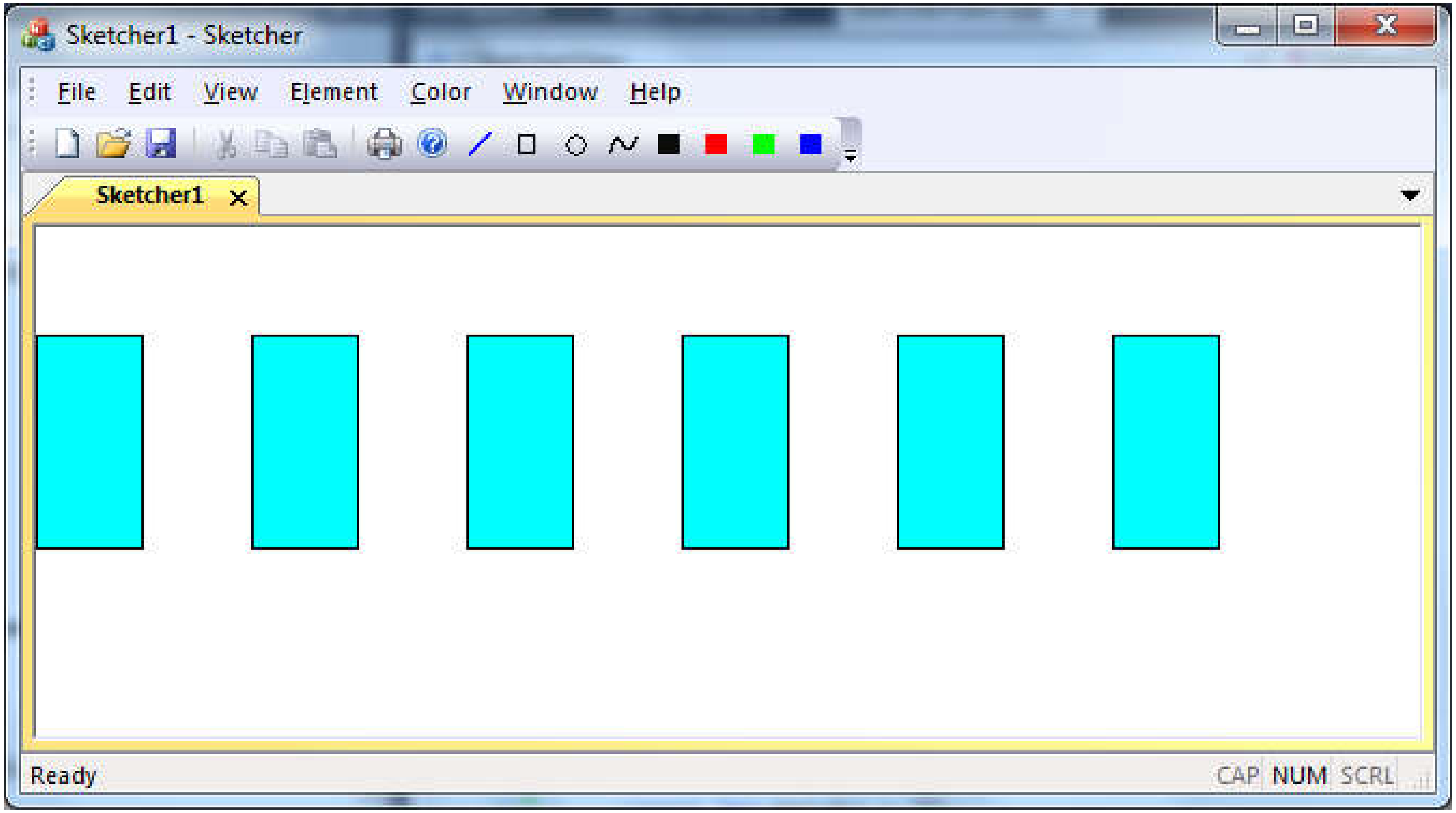


Creating a Brush (P.676)

- 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).

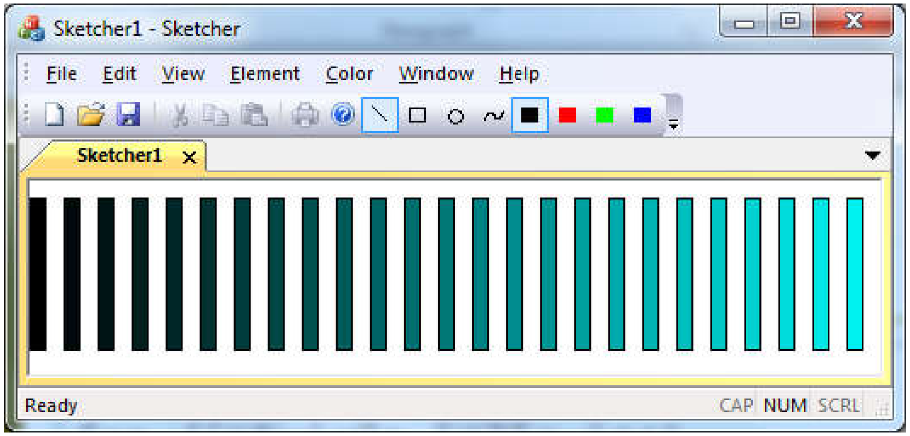
```
CBrush aBrush(RGB(0,255,255));

CBrush* pOldBrush =
 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


Sketcher1 - Sketcher
File
Edit
View
Element
Color
Window
Help
Sketcher1
Ready
CAP
NUM
SCRL

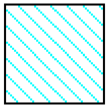
DeleteObject() (P.676)

```
CBrush aBrush;
for (int i=0; i<25; i++)
{
    aBrush.CreateSolidBrush(RGB(0,i*10,i*10));
    CBrush* pOldBrush = pDC->SelectObject(&aBrush);
    pDC->Rectangle(i*20, 10, i*20+10, 100);
    aBrush.DeleteObject();
}
```

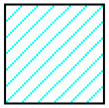


Hatching Style (P.676)

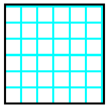
- HS_HORIZONTAL
- HS_VERTICAL
- HS_FDIAGONAL



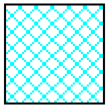
- HS_BDIAGONAL



- HS_CROSS



- HS_DIAGCROSS

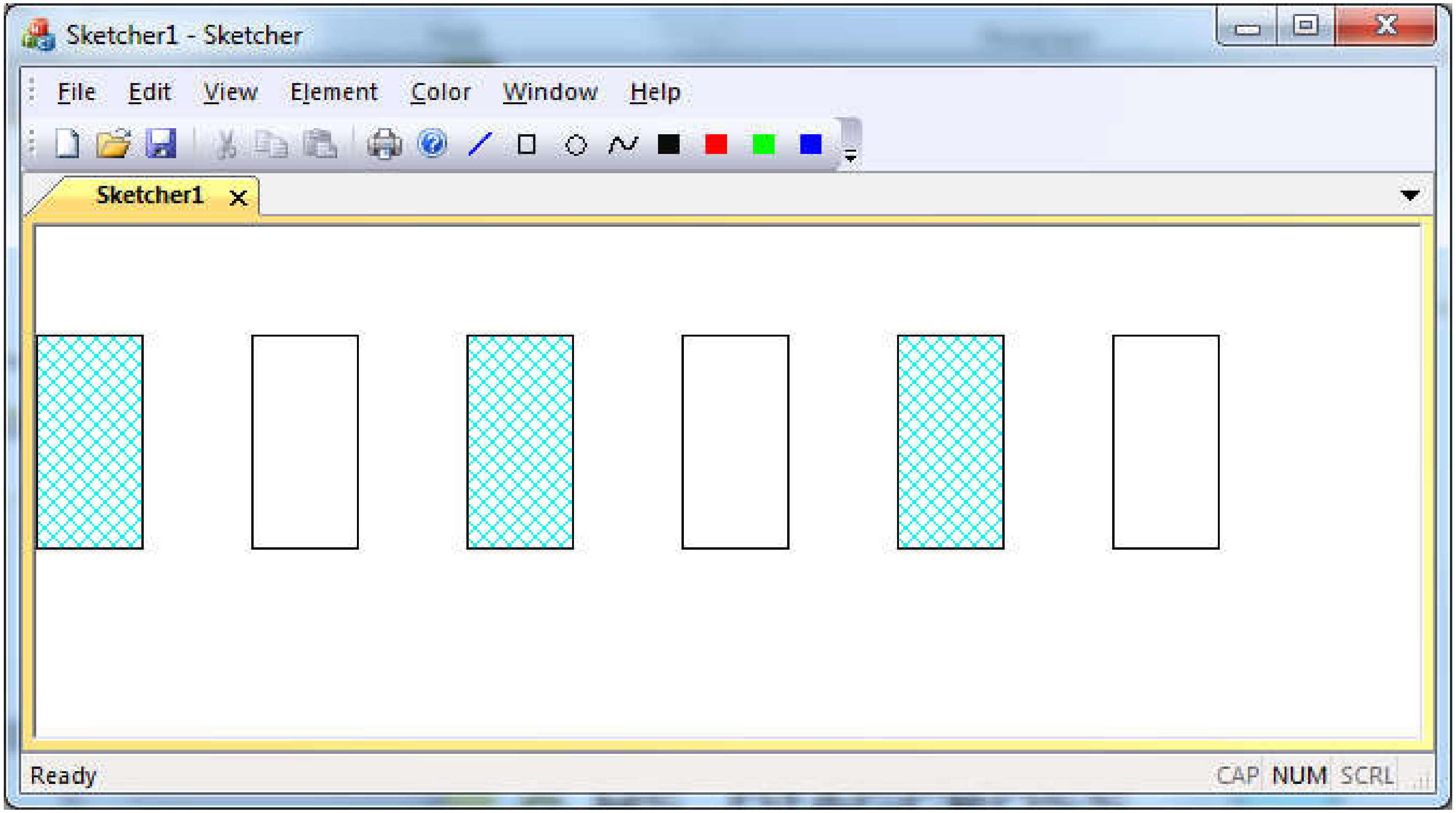


```
CBrush aBrush;
aBrush.CreateHatchBrush(HS_DIAGCROSS,
  RGB(0,255,255));
CBrush* pOldBrush = pDC->SelectObject(&aBrush);
```

SketcherView.cpp

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

          CBrush aBrush(HS_DIAGCROSS, RGB(0,255,255));
          CBrush* pOldBrush =
                    pDC->SelectObject(&aBrush);

          const int width = 50;
          const int height = 50;
          int i;
          for (i=0; i<6; i+=2)
                    pDC->Rectangle(i*2*width, 50,i*2*width+50, 150);

          pDC->SelectObject(pOldBrush);
          for (i=1; i<6; i+=2)
                    pDC->Rectangle(i*2*width, 50,i*2*width+50, 150);
}
```

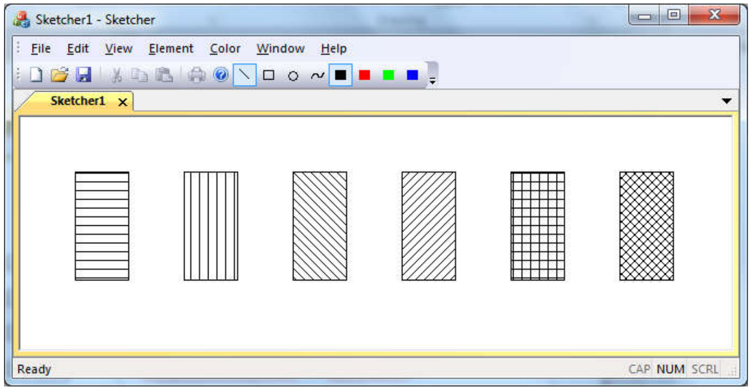
A Hatched Brush



The BrushHatch enumeration

```
typedef enum
{
 HS_HORIZONTAL = 0x00000000,
 HS_VERTICAL = 0x00000001,
 HS_FDIAGONAL = 0x00000002,
 HS_BDIAGONAL = 0x00000003,
 HS_CROSS = 0x00000004,
 HS_DIAGCROSS = 0x00000005
} BrushHatch;
```

```
CBrush aBrush;
for (int i=0; i<6; i++)
{
aBrush.CreateHatchBrush(i,
RGB(0,0,0));
CBrush* pOldBrush = pDC-
>SelectObject(&aBrush);
pDC->Rectangle(i*100+50, 50,
i*100+100, 150);
aBrush.DeleteObject();
}
```

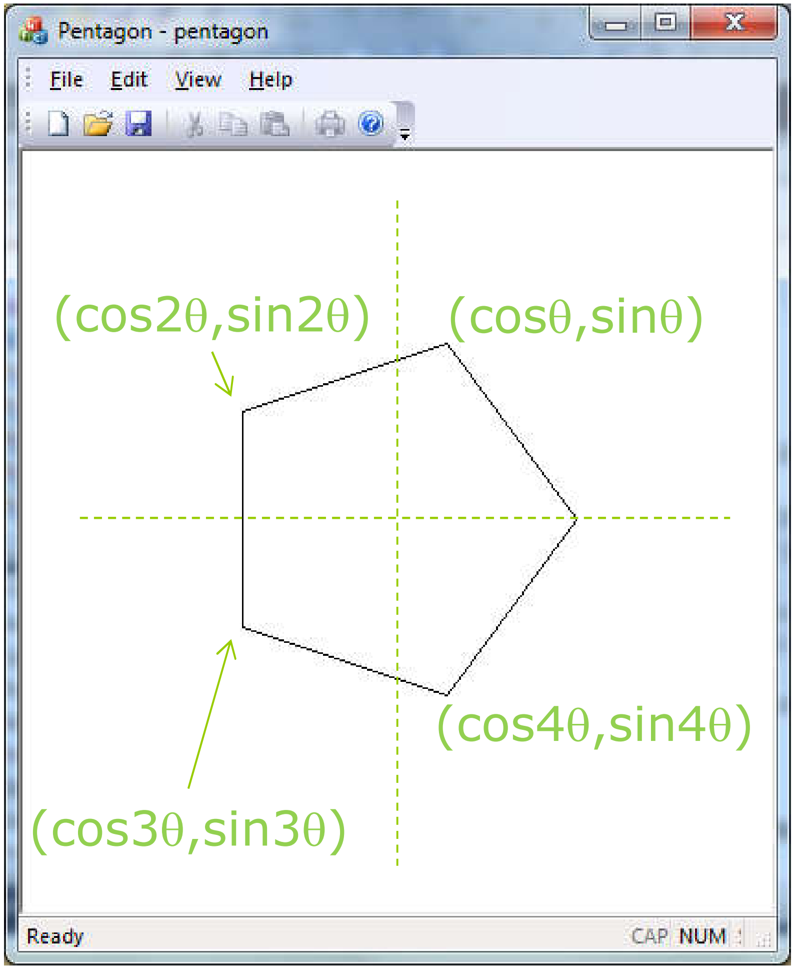


Summary

- The client coordinate system
- Drawing in the client area
- Device contexts
- Mapping modes
- Drawing in a window
  - Line, Rectangle, Ellipse, Arc
- Pen
- Brush

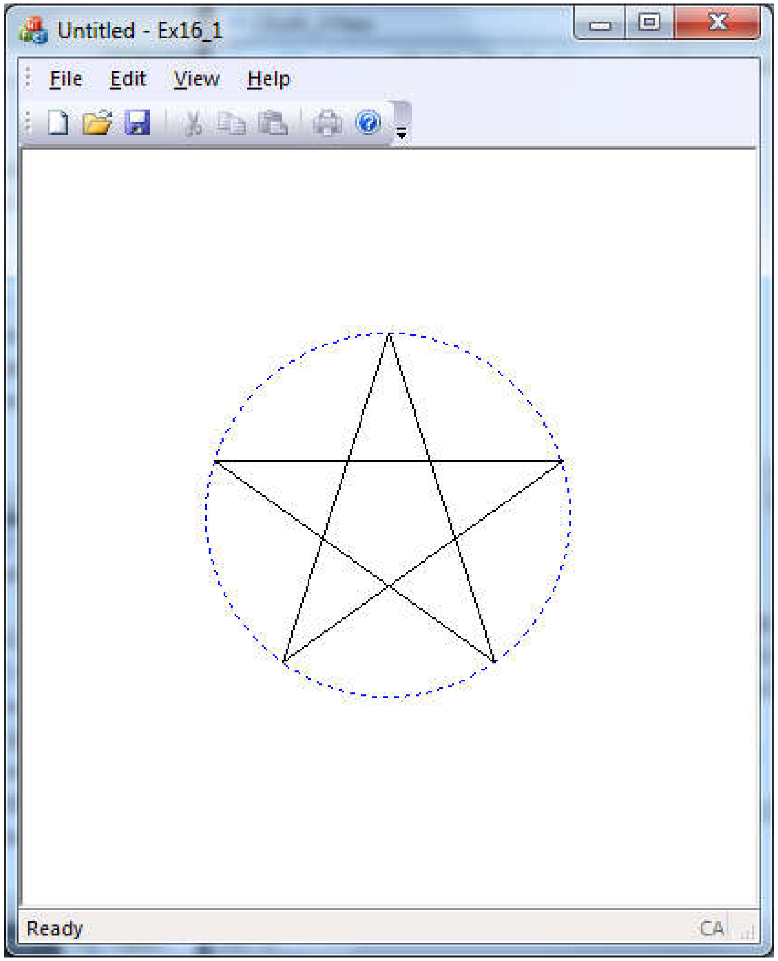
Homework: Pentagon

- Draw a pentagon like this.
- You may need to include <cmath> if you want to call the sin/cos functions.



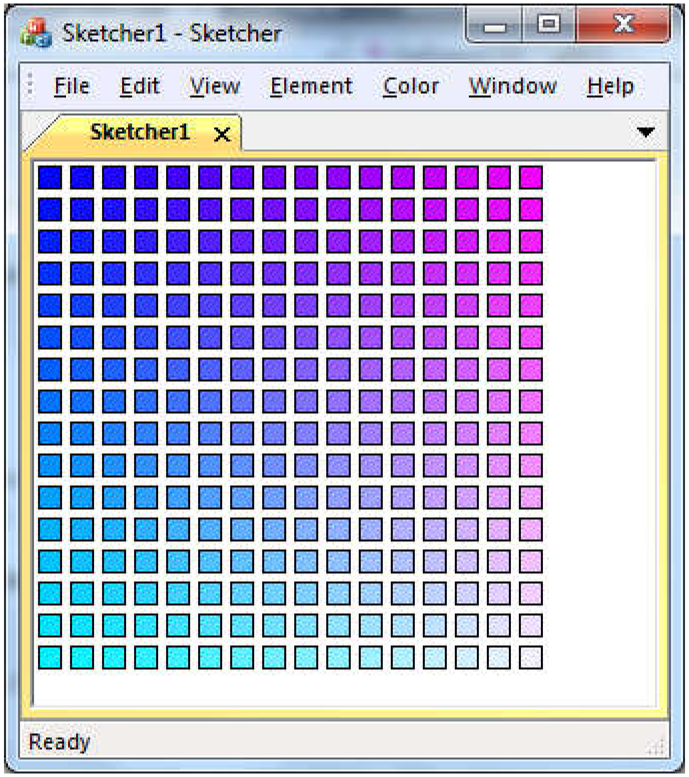
Homework

- Use `LineTo()` and `Ellipse()` to draw the following figure.
- Hint: You may need to include `<cmath>` to utilize the `sin()` and `cos()` functions.



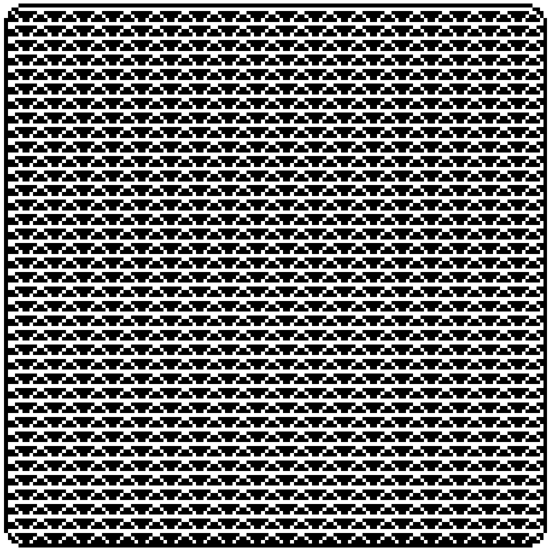
Exercise: Colorful Bricks

- Use CreateSolidBrush() to write a program generating the output as shown in this figure.



Exercise: Create a Patterned Brush

```
// Create a hatched bit pattern.
WORD HatchBits[8] = { 0x18, 0x24, 0x42, 0xFF, 0x18, 0x24, 0x42, 0xFF, };

// Use the bit pattern to create a bitmap.
CBitmap bm;
bm.CreateBitmap(8,8,1,1, HatchBits);

// Create a pattern brush from the bitmap.
CBrush brush;
brush.CreatePatternBrush(&bm);

// Select the brush into a device context, and draw.
CBrush* pOldBrush = (CBrush*)pDC->SelectObject(&brush);
pDC->RoundRect(CRect(50, 50, 200, 200), CPoint(10,10));

// Restore the original brush.
pDC->SelectObject(pOldBrush);
```



如何學好程式設計

整理筆記

學以致用