

쉽게 배우는 MFC 윈도우 프로그래밍

Visual C++ 2019

2015·2017 버전 사용 가능



4장. 화면 출력

목차

01 화면 출력 기초

02 CDC 클래스

03 GDI 객체

GDI와 디바이스 컨텍스트

■ 윈도우 운영체제에서 출력 시스템을 설계 시 고려 사항

- 장치 변경에 따른 프로그램 수정 없음
 - 모니터, 비디오 카드, 프린터 등 출력에 사용되는 주변 장치가 변경되더라도 프로그램을 수정할 필요가 없어야 한다.
- 프로그램 출력 영역 제약
 - 여러 프로그램이 화면을 분할해서 사용하므로 각각의 프로그램이 출력하는 영역에 제약을 가해야 한다.
 - 화면이나 기타 출력 장치에 직접 접근(Direct Access)하거나 독점해서 사용(Exclusive Use)하는 것을 운영체제 수준에서 막아야 한다.

GDI와 디바이스 컨텍스트

■ GDI(Graphics Device Interface)

- 윈도우 운영체제의 하위 시스템 중 하나로 DLL로 존재
- 응용 프로그램의 요청을 받아 실제 출력 장치인 모니터나 프린터에 출력하는 역할을 담당

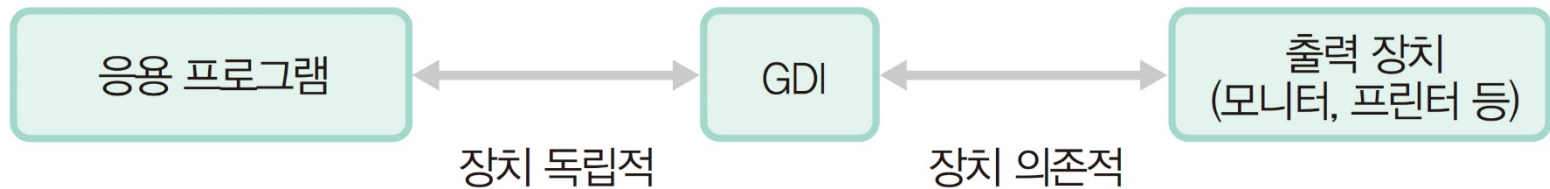


그림 4-1 GDI의 역할

GDI와 디바이스 컨텍스트

■ 윈도우 응용 프로그램이 화면에 출력 시 고려 사항

- 클라이언트 영역에 출력하려면 출력 대상 윈도우의 위치를 알아야 한다.
- 화면에 윈도우가 여러 개 있을 때, 출력 결과가 다른 윈도우 영역을 침범하지 않아야 한다.
- 현재 출력할 화면이 다른 윈도우에 가려졌다면 출력할 수 없어야 한다.

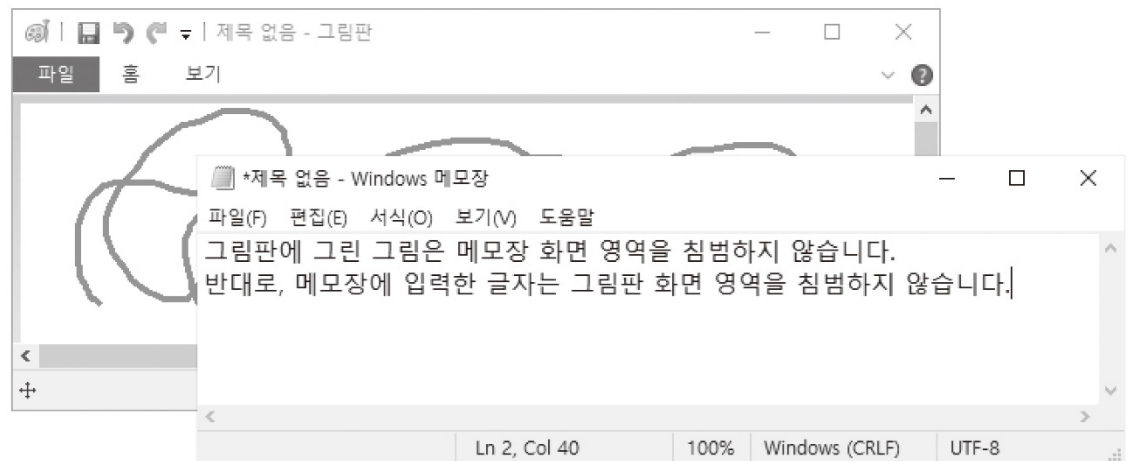


그림 4-2 화면 출력 상황

GDI와 디바이스 컨텍스트

■ 디바이스 컨텍스트(DC, Device Context)

- GDI가 생성하고 관리하는 데이터 구조체
- 멀티태스킹 GUI 환경에서 발생할 수 있는 복잡한 상황들을 신경쓰지 않고 장치에 자유롭게 출력 가능

GDI와 디바이스 컨텍스트

■ 윈도우 응용 프로그램의 출력 과정(1/2)

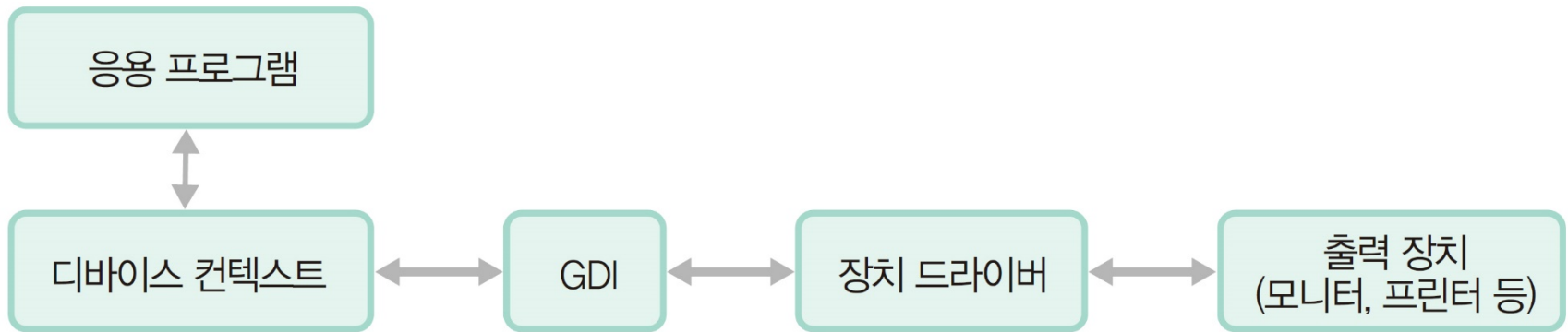


그림 4-3 윈도우 응용 프로그램의 출력 과정

GDI와 디바이스 컨텍스트

■ 윈도우 응용 프로그램의 출력 과정 (2/2)

- 운영체제에 디바이스 컨텍스트를 요청
- 요청을 받은 운영체제의 GDI가 내부적으로 디바이스 컨텍스트를 만든 후, 디바이스 컨텍스트를 가리키는 핸들(HDC 타입)을 응용 프로그램에 돌려줌
- 응용 프로그램은 (필요하다면) 받은 디바이스 컨텍스트의 속성을 일부 변경한다. 그런 다음, 디바이스 컨텍스트 핸들을 윈도우 API 함수에 전달하여 출력(→장치 독립적)을 요청. 이 요청은 다시 운영체제의 GDI에 전달됨
- GDI가 디바이스 컨텍스트에 포함된 정보를 토대로 다양한 상황을 고려하여 출력(→장치 의존적). 이때 장치별 출력을 위해 장치 드라이버를 사용함

무효 영역의 개념

■ 화면을 다시 그려야 하는 상황

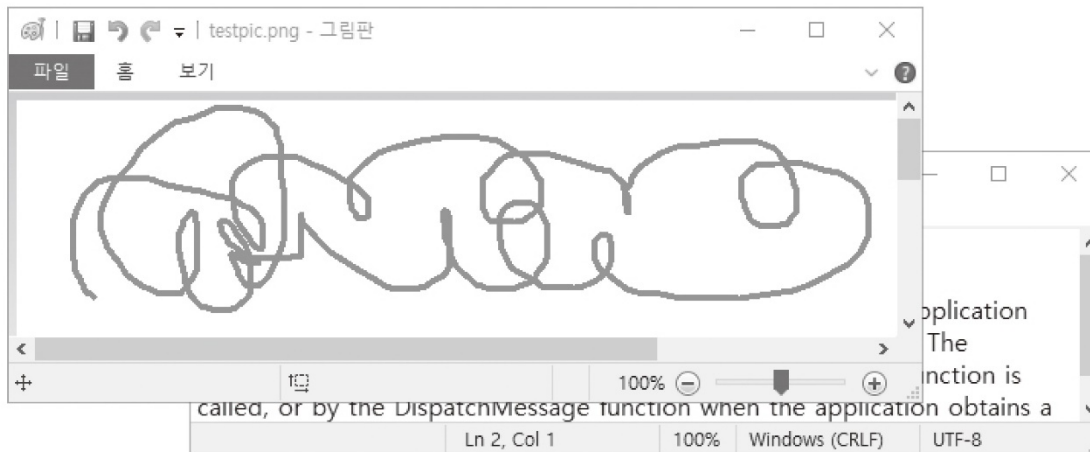
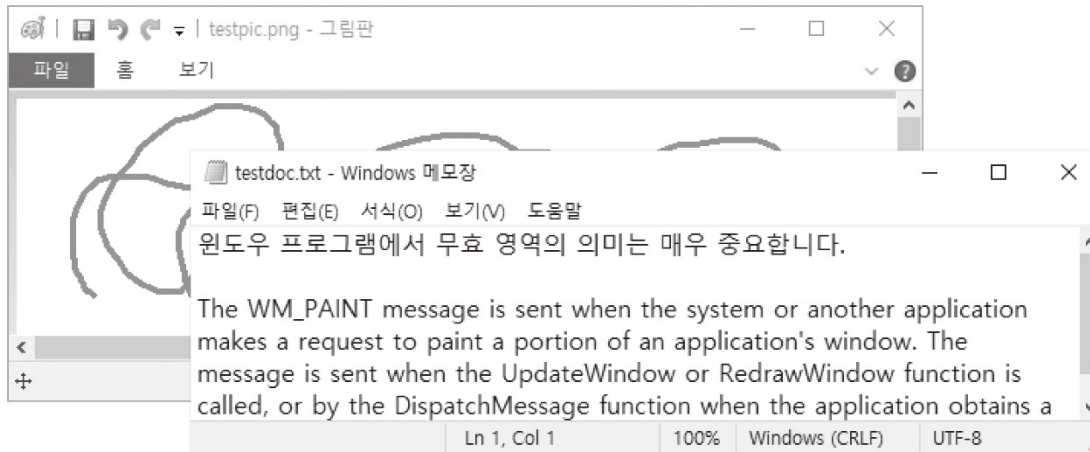


그림 4-4 화면을 다시 그려야 하는 상황

무효 영역의 개념

■ WM_PAINT 메시지 처리 방식

- HelloSDK 프로그램

```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    TextOut(hdc, 100, 100, str, strlen(str));
    EndPaint(hwnd, &ps);
    return 0;
```

- HelloMFC 프로그램

```
void CMainFrame::OnPaint()
{
    CPaintDC dc(this); /* 생성자에서 ::BeginPaint() 호출! */
    TCHAR* msg = _T("Hello, MFC");
    dc.TextOut(100, 100, msg, strlen(msg));
} /* 소멸자에서 ::EndPaint() 호출! */
```

무효 영역의 개념

■ WM_PAINT 메시지 발생 상황

- 윈도우가 생성될 때
- 윈도우가 최소화 또는 최대화되었을 때
- 윈도우의 크기가 변경될 때
- 다른 윈도우가 가렸다가 드러날 때

■ 무효 영역 생성 함수

```
void CWnd::Invalidate(BOOL bErase = TRUE);
```

```
void CWnd::InvalidateRect(LPCRECT lpRect, BOOL bErase = TRUE);
```

```
void CWnd::InvalidateRgn(CRgn* pRgn, BOOL bErase = TRUE);
```

다양한 디바이스 컨텍스트 클래스

■ SDK 프로그램 출력 과정

- 운영체제에 디바이스 컨텍스트를 요청해서 얻음
- 디바이스 컨텍스트를 전달 인자로 사용해 API 함수를 호출하여 출력
- 디바이스 컨텍스트 사용이 끝났음을 운영체제에 알림

■ MFC 프로그램 출력 과정

- 디바이스 컨텍스트 객체를 생성
- 객체의 멤버 함수를 호출하여 출력

다양한 디바이스 컨텍스트 클래스

■ MFC 클래스 계층도

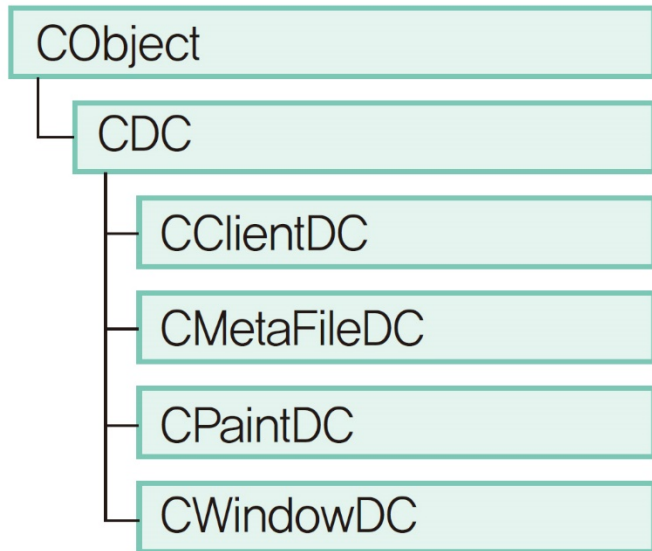


그림 4-5 MFC 클래스 계층도

다양한 디바이스 컨텍스트 클래스

표 4-1 디바이스 컨텍스트 클래스(CDC 파생 클래스)

클래스 이름	용도
CPaintDC	클라이언트 영역에 출력할 때 (WM_PAINT 메시지 핸들러에서만 사용)
CClientDC	클라이언트 영역에 출력할 때 (WM_PAINT 메시지 핸들러를 제외한 다른 모든 곳에서 사용)
CWindowDC	윈도우 전체 영역(클라이언트 영역+비클라이언트 영역)에 출력할 때
CMetaFileDC	메타파일(Metafile)에 출력할 때

CPaintDC 클래스

■ CPaintDC

- WM_PAINT 메시지 핸들러에서만 사용 가능
- 클라이언트 영역에만 출력 가능

■ 사용 예

```
void CChildView::OnPaint()  
{  
    CPaintDC dc(this);  
}
```

CPaintDC 사용하기

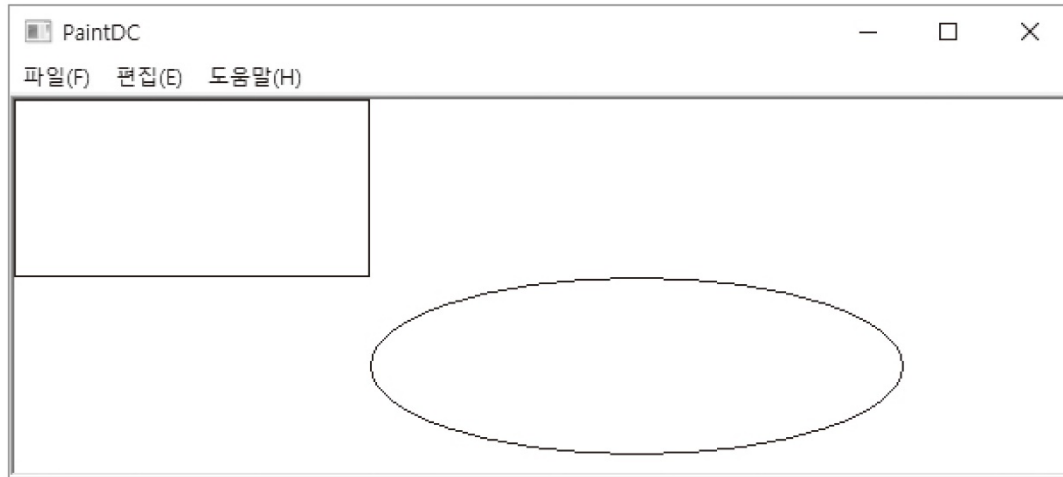


그림 4-6 실행 결과

- CChildView::OnPaint() 함수에 다음 코드를 추가

```
void CChildView::OnPaint()  
{  
    CPaintDC dc(this);  
}
```

CClientDC 사용하기

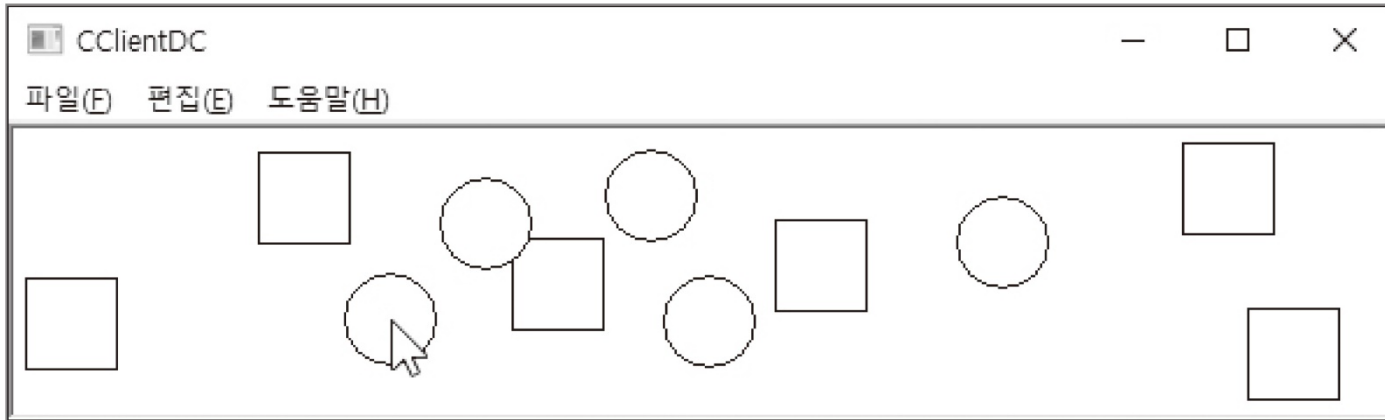


그림 4-7 실행 결과

CClientDC 사용하기

‘메시지’를 선택하면 메시지 핸들러를 추가/삭제할 수 있다.

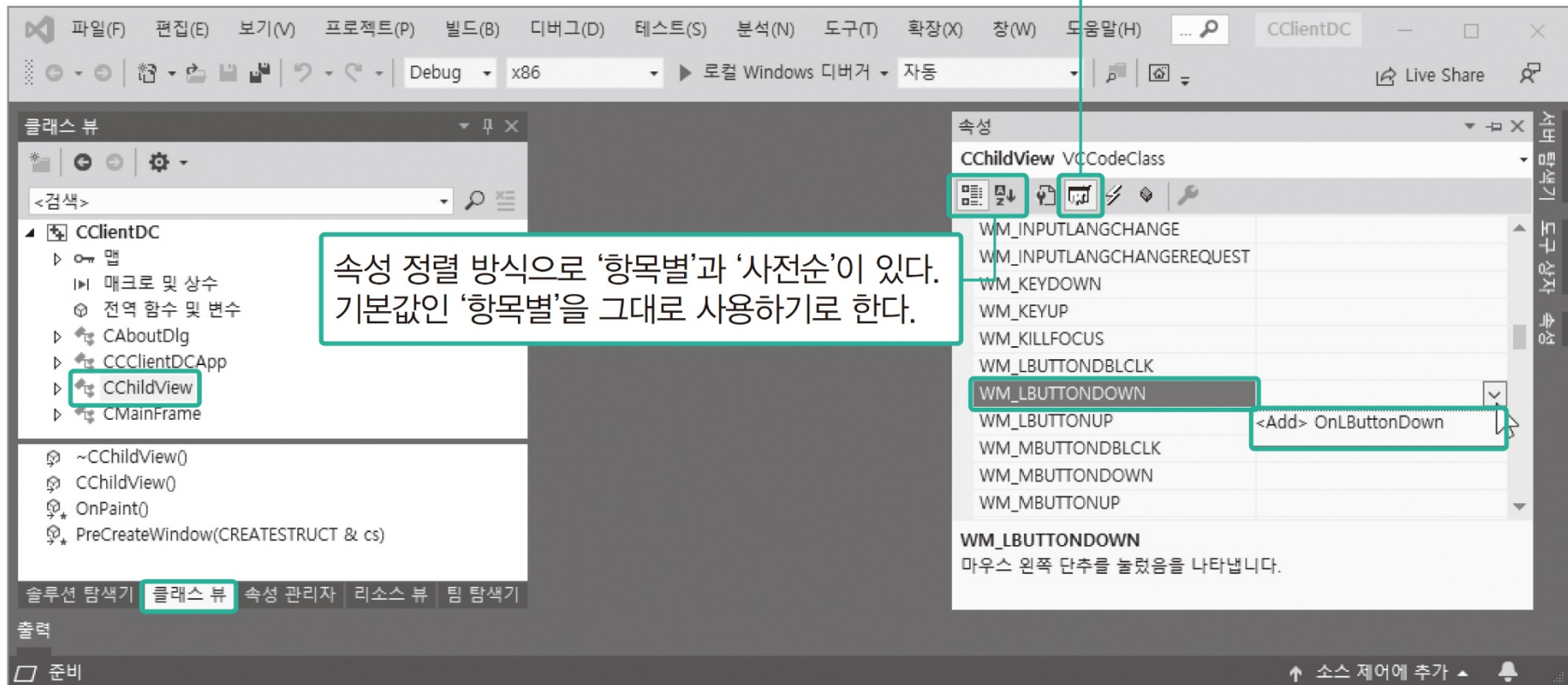


그림 4-8 마우스 메시지 핸들러 추가

CClientDC 사용하기

■ OnLButtonDown()

```
void CChildView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    dc.Rectangle(point.x - 20, point.y - 20, point.x + 20, point.y + 20);
}
```

■ OnRButtonDown()

```
void CChildView::OnRButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    dc.Ellipse(point.x - 20, point.y - 20, point.x + 20, point.y + 20);
}
```

CWindowDC 클래스

■ CWindowDC

- 윈도우 전체(클라이언트+비클라이언트)영역에 출력
- CPaintDC/CClientDC 클래스와 사용법 동일
- 단, 좌표의 원점 위치가 다름

■ 디바이스 컨텍스트별 원점 위치

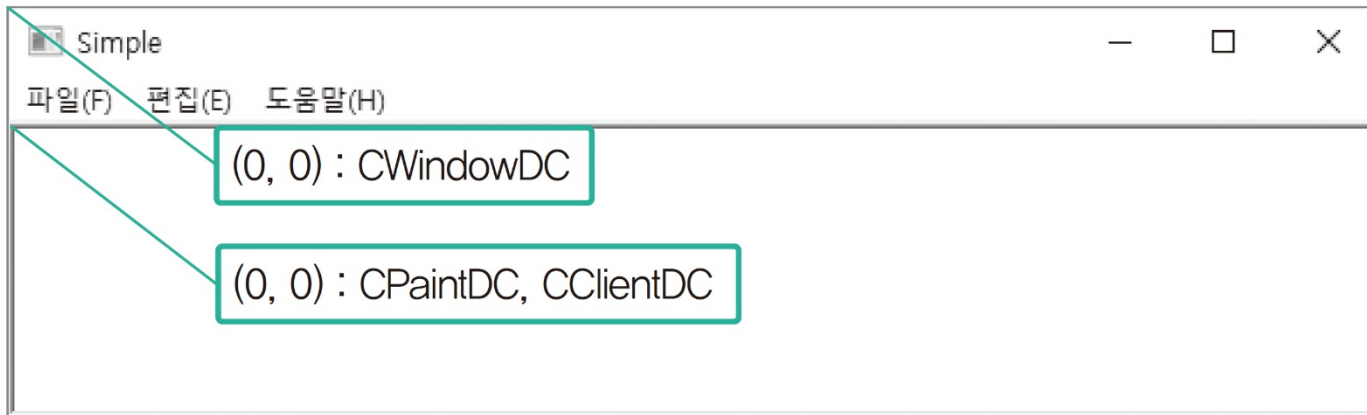


그림 4-9 디바이스 컨텍스트별 원점 위치

CWindowDC 사용하기

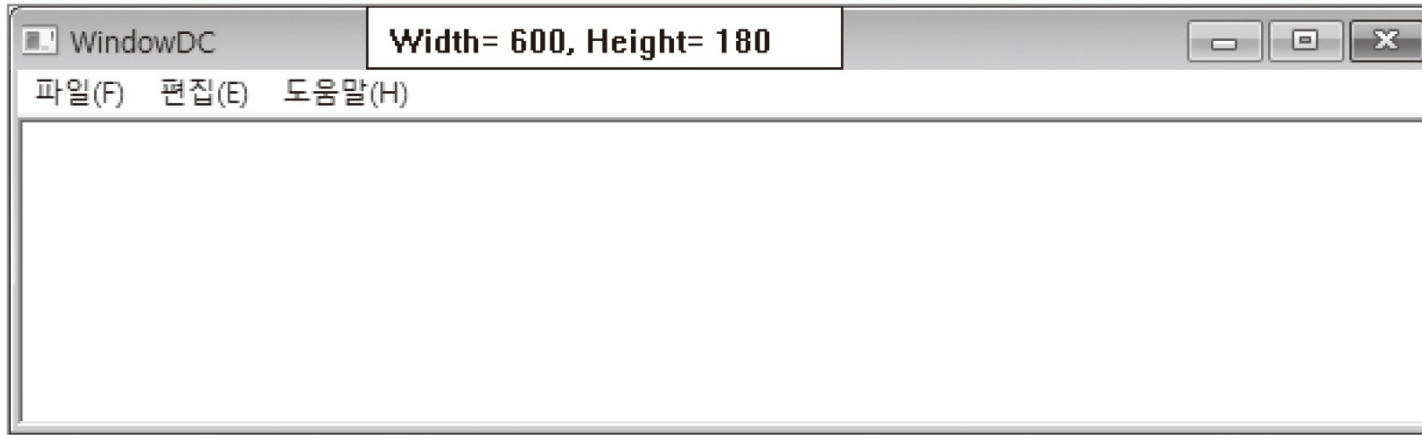


그림 4-10 실행 결과

CWindowDC 사용하기

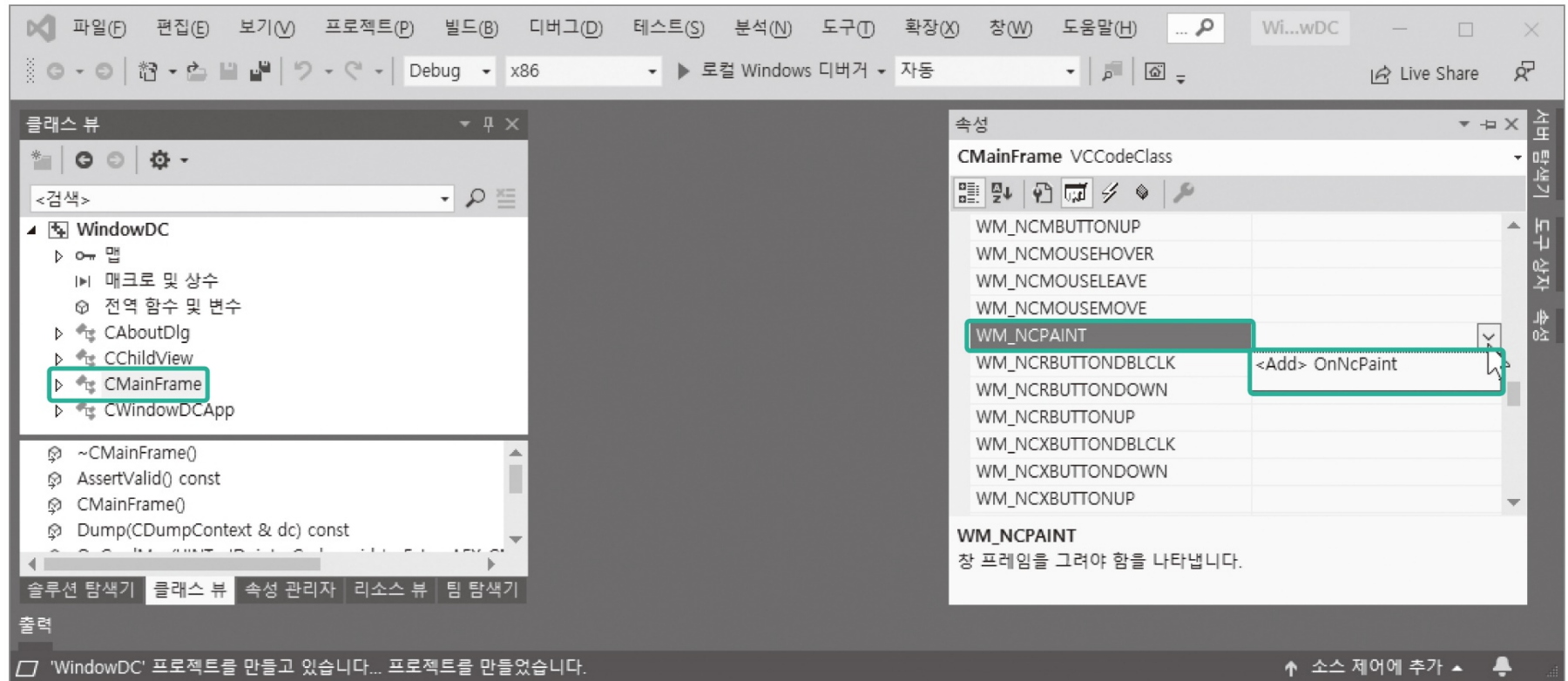


그림 4-11 WM_NCPAINT 메시지 핸들러 추가

CClientDC 사용하기

■ CMainFrame::OnNcPaint()

```
01 void CMainFrame::OnNcPaint()
02 {
03     // 운영체제가 자동으로 비클라이언트 영역을 그리게 한다.
04     CFrameWnd::OnNcPaint();
05
06     // 타이틀바에 직사각형을 그린다.
07     CWindowDC dc(this);
08     int height = ::GetSystemMetrics(SM_CYSIZE)+GetSystemMetrics(SM_CYSIZEFRAME);
09     dc.Rectangle(150, 0, 350, height + 1);
10
11     // 메인(프레임) 윈도우의 크기를 얻는다.
12     CRect rect;
13     GetWindowRect(&rect);
14
15     // 타이틀바에 크기를 출력한다.
16     CString str;
17     str.Format(_T("Width=%4d, Height=%4d"), rect.Width(), rect.Height());
18     dc.TextOut(160, 7, str);
19 }
```

CMetaFileDC 클래스

■ 메타파일(Metafile)

- GDI 명령을 저장할 수 있는 파일

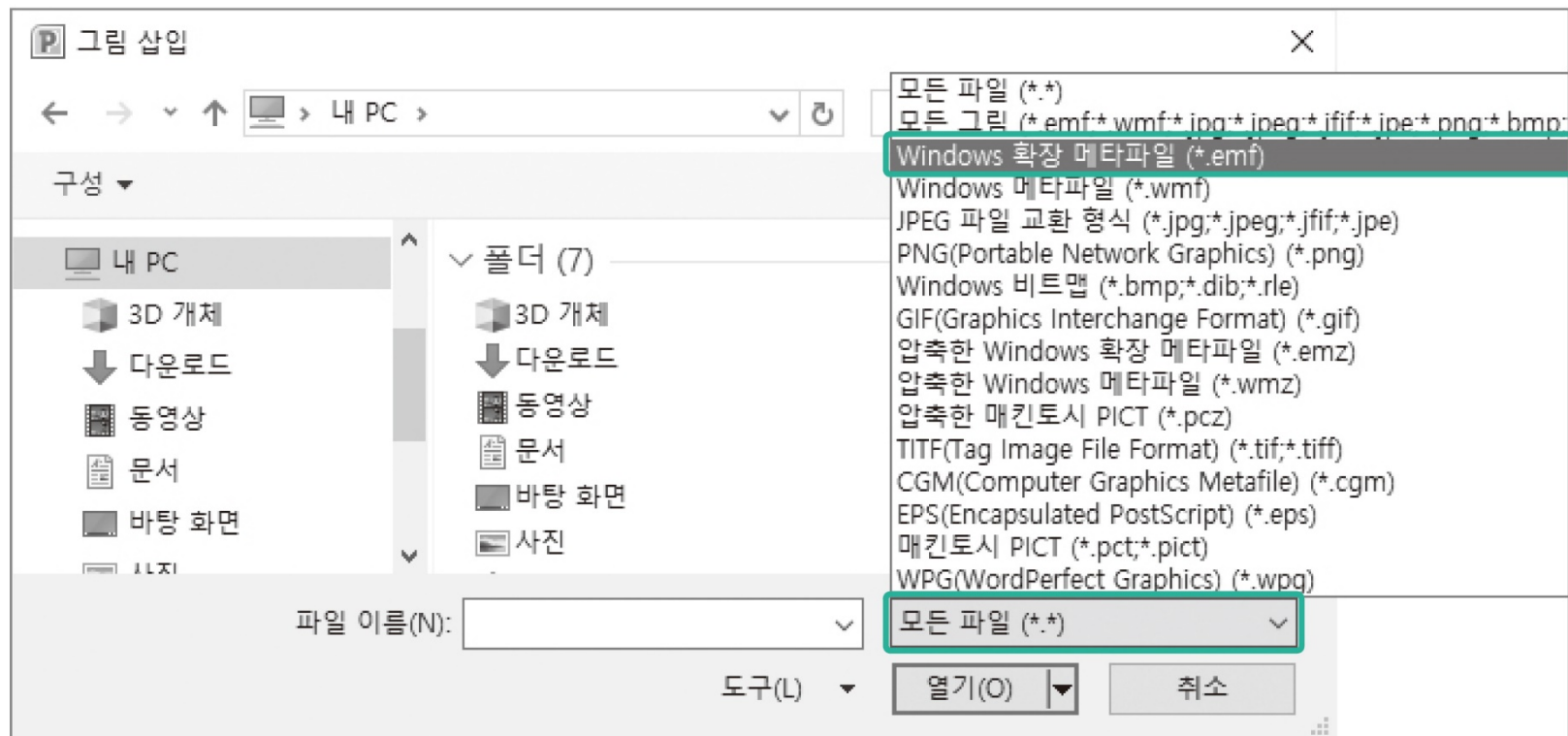


그림 4-12 메타파일 사용 예

CMetaFileDC 클래스

■ CMetaFileDC 사용 방법

- CMetaFileDC 객체를 만들고, CMetaFileDC::CreateEnhanced() 함수로 초기화
- 확장 메타파일 객체를 일반 디바이스 컨텍스트 객체로 간주하여 출력 함수를 호출
- CMetaFileDC::CloseEnhanced() 함수를 호출하여 확장 메타파일 핸들(HENHMETAFILE 타입)을 얻음
- 확장 메타파일 핸들을 CDC::PlayMetaFile() 함수에 전달하여 재생하면 저장된 GDI 명령들이 수행됨
- 확장 메타파일 사용이 끝나면 핸들을 ::DeleteEnhMetaFile() API 함수에 전달하여 삭제

CMetaFileDC 사용하기

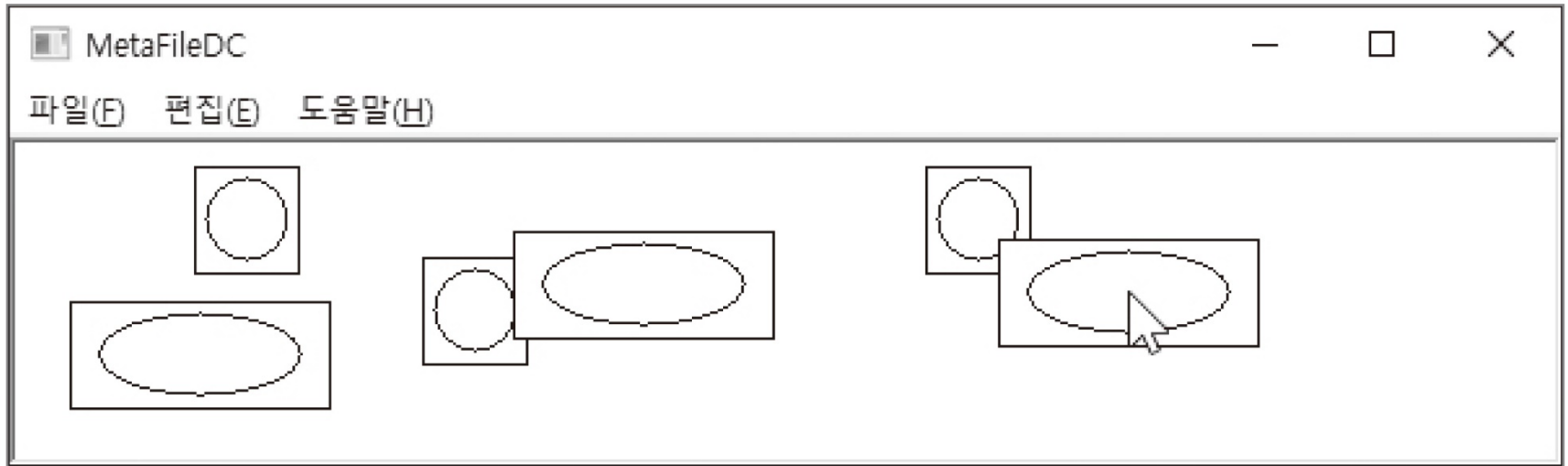


그림 4-13 실행 결과

CMetaFileDC 사용하기

```
class CChildView : public CWnd
{
// 생성입니다.
public:
    CChildView();
// 특성입니다.
public:
    HENHMETAFILE m_hmf; // 메타파일 핸들
```

CMetaFileDC 사용하기

```
CChildView::CChildView()
{
    // 확장 메타파일 객체를 생성하고 초기화한다.
    CMetaFileDC dc;
    dc.CreateEnhanced(NULL, NULL, NULL, NULL);
    // 멤버 함수를 호출하여 출력한다.
    dc.Rectangle(0, 0, 10, 10);
    dc.Ellipse(1, 1, 9, 9);
    // 확장 메타파일 핸들을 얻는다.
    m_hmf = dc.CloseEnhanced();
}
```

```
CChildView::~~CChildView()
{
    // 확장 메타파일을 삭제한다.
    ::DeleteEnhMetaFile(m_hmf);
}
```

CMetaFileDC 사용하기

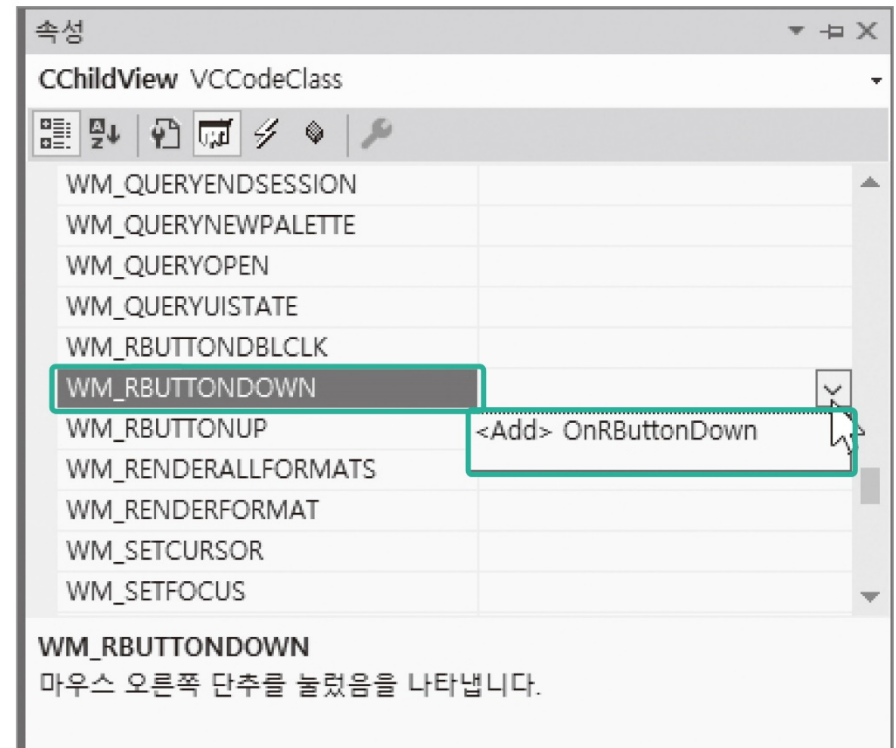
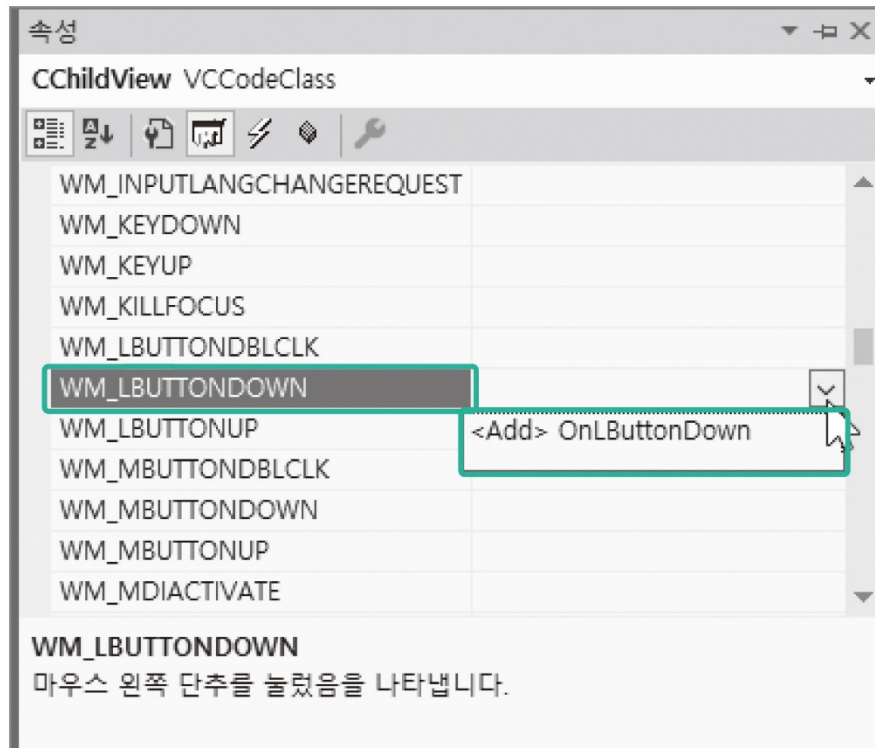


그림 4-14 두 개의 메시지 핸들러 추가

CMetaFileDC 사용하기

```
void CChildView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    CRect rect(point.x - 20, point.y - 20, point.x + 20, point.y + 20);
    dc.PlayMetaFile(m_hmf, &rect);
}
```

```
void CChildView::OnRButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    CRect rect(point.x - 50, point.y - 20, point.x + 50, point.y + 20);
    dc.PlayMetaFile(m_hmf, &rect);
}
```

그리기 함수

■ 점 찍기

표 4-2 그리기 함수 : 점

이름	기능
GetPixel()	화면 (x, y) 지점의 색을 얻는다.
SetPixel()	화면 (x, y) 지점에 특정 색상의 점을 찍고 원래 점의 색을 리턴한다.
SetPixelV()	SetPixel() 함수와 출력은 같지만 원래 점의 색을 리턴하지 않으므로 속도가 좀 더 빠르다.

그리기 함수

■ 점 찍기

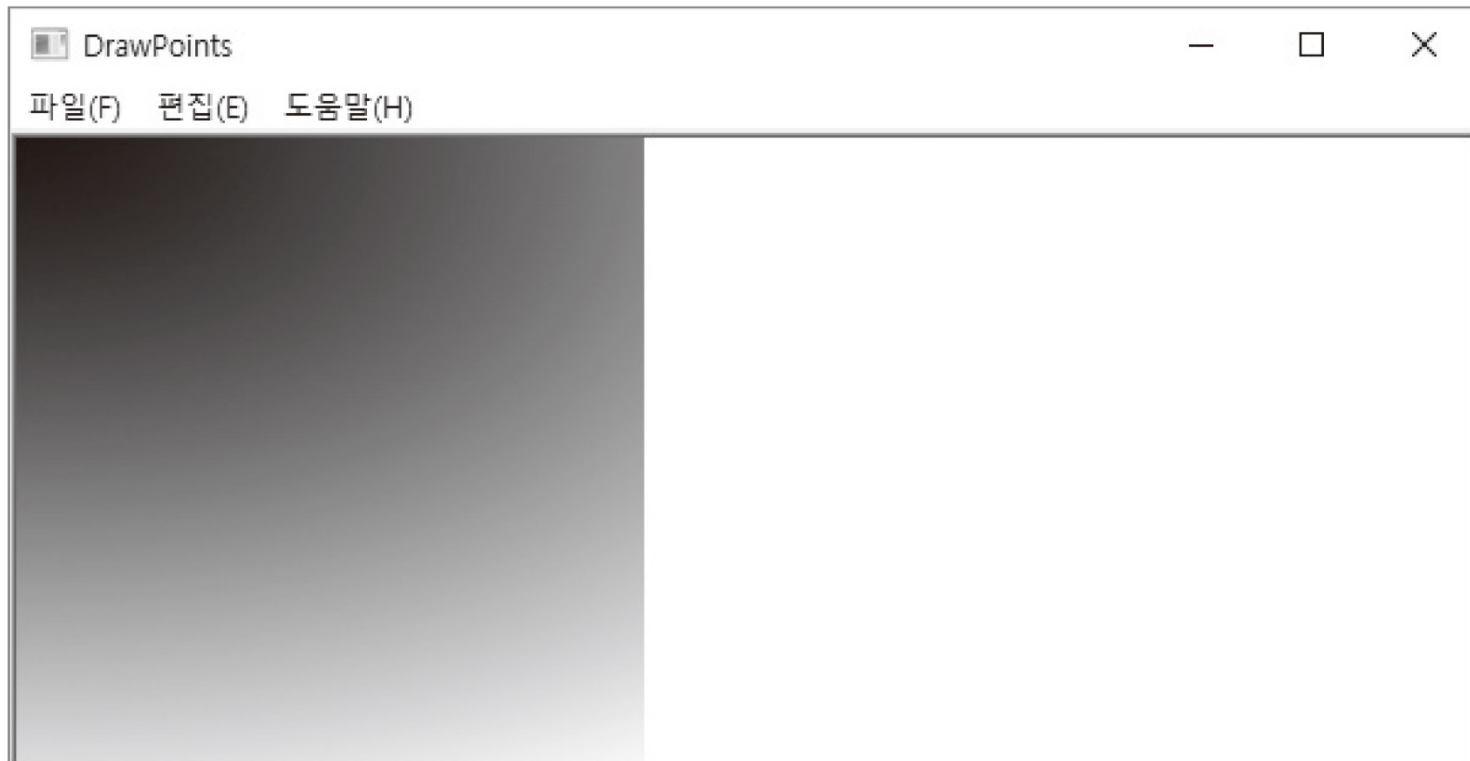


그림 4-15 실행 결과

그리기 함수

■ 점 찍기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);
    for (int x = 0; x < 256; x++)
        for (int y = 0; y < 256; y++)
            dc.SetPixelV(x, y, RGB(x, y, 0));
}
```

```
COLORREF RGB (
    BYTE byRed, // 빨간색 값
    BYTE byGreen, // 초록색 값
    BYTE byBlue // 파란색 값
);
```

그리기 함수

■ 선 그리기

- ① 현재 위치를 $(x1, y1)$ 으로 옮긴다.
 - 현재 위치는 디바이스 컨텍스트의 속성 중 하나
- ② 현재 위치에서 $(x2, y2)$ 까지 선을 그린다.
 - 완료되면 현재 위치는 $(x2, y2)$ 로 자동 변경됨

① `dc.MoveTo(x1, y1);`

② `dc.LineTo(x2, y2);`

그리기 함수

■ 선 그리기

표 4-3 그리기 함수 : 선

이름	기능
MoveTo()	현재 위치를 (x, y) 위치로 옮긴다.
LineTo()	현재 위치에서 (x, y) 위치까지 선을 그리고, 현재 위치를 (x, y)로 변경한다.
Polyline()	POINT 구조체 배열로 전달된 점들을 차례로 이어서 선을 그린다. 참고 많은 데이터 점들을 이어서 그래프를 그릴 때 유용하다.

[실습 4-6] 선 그리기

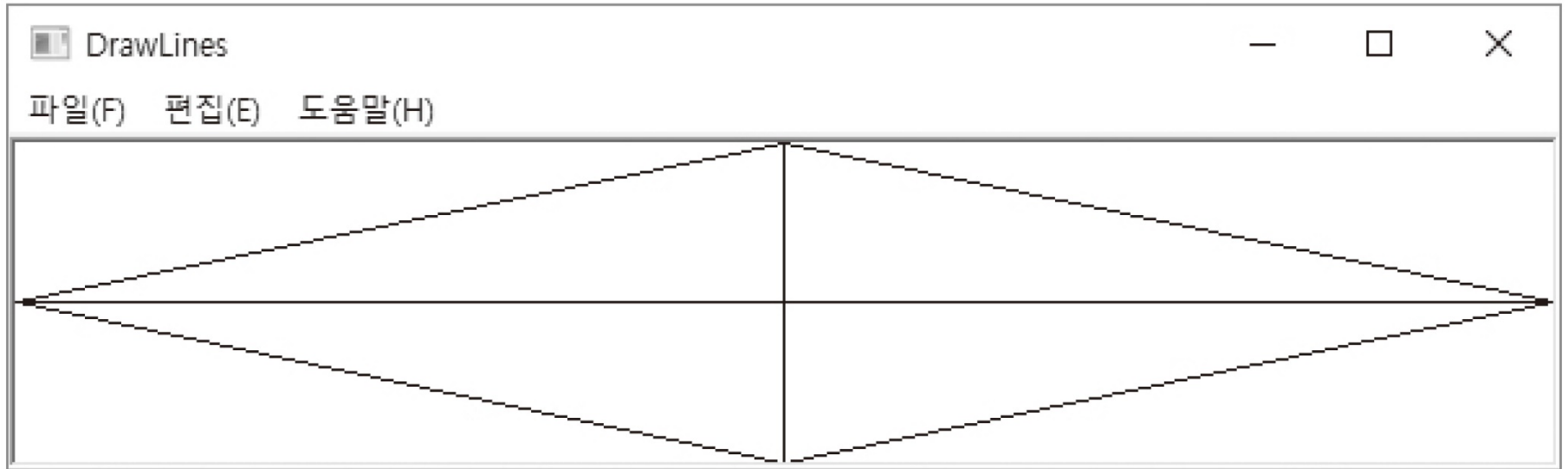


그림 4-16 실행 결과

[실습 4-6] 선 그리기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);

    // 클라이언트 영역의 좌표를 얻는다.
    CRect rect;
    GetClientRect(&rect);

    // 수평선과 수직선을 그린다.
    dc.MoveTo(0, rect.bottom / 2);
    dc.LineTo(rect.right, rect.bottom / 2);
    dc.MoveTo(rect.right / 2, 0);
    dc.LineTo(rect.right / 2, rect.bottom);

    // 마름모꼴을 그린다.
    POINT points[] = { {rect.right / 2, 0}, {rect.right, rect.bottom / 2},
        { rect.right / 2, rect.bottom}, {0, rect.bottom / 2}, {rect.right / 2, 0} };
    dc.Polyline(points, 5);
}
```

그리기 함수

■ 도형 그리기

표 4-4 그리기 함수 - 도형

이름	기능
Rectangle()	직사각형을 그린다.
Ellipse()	직사각형에 내접하는 타원을 그린다.
RoundRect()	테두리가 둥근 직사각형을 그린다.
Polygon()	POINT 구조체 배열로 전달된 점들을 차례로 이어서 다각형을 그린다.

그리기 함수

■ 도형 그리기

`RoundRect(int x1, int y1, int x2, int y2, int x3, int y3);`

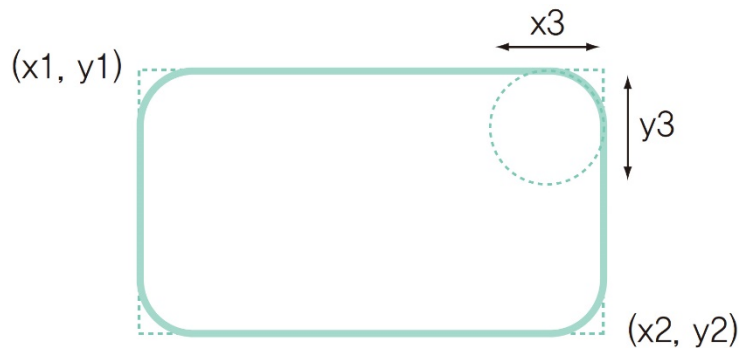


그림 4-17 RoundRect() 함수

// 외접 직사각형의 좌상단점과 우하단점 좌표를 지정한다.

`BOOL Ellipse(int x1, int y1, int x2, int y2);`

// 외접 직사각형의 좌표를 RECT 또는 CRect 객체로 지정한다.

`BOOL Ellipse(LPCRECT lpRect);`

텍스트 함수

■ 텍스트 함수

표 4-5 텍스트 함수

이름	기능
TextOut()	(x, y) 위치에 문자열을 출력한다.
DrawText()	직사각형 영역 내부에 문자열을 출력한다.
SetTextColor()	글자의 색을 바꾼다.
SetBkColor()	글자의 배경색을 바꾼다.
SetTextAlign()	기준 위치에 대한 문자열 정렬 방식을 설정한다.

```
int DrawText(  
    const CString& str,  
    LPRECT lpRect,  
    UINT nFormat  
);
```

[실습 4-7] 텍스트 출력하기

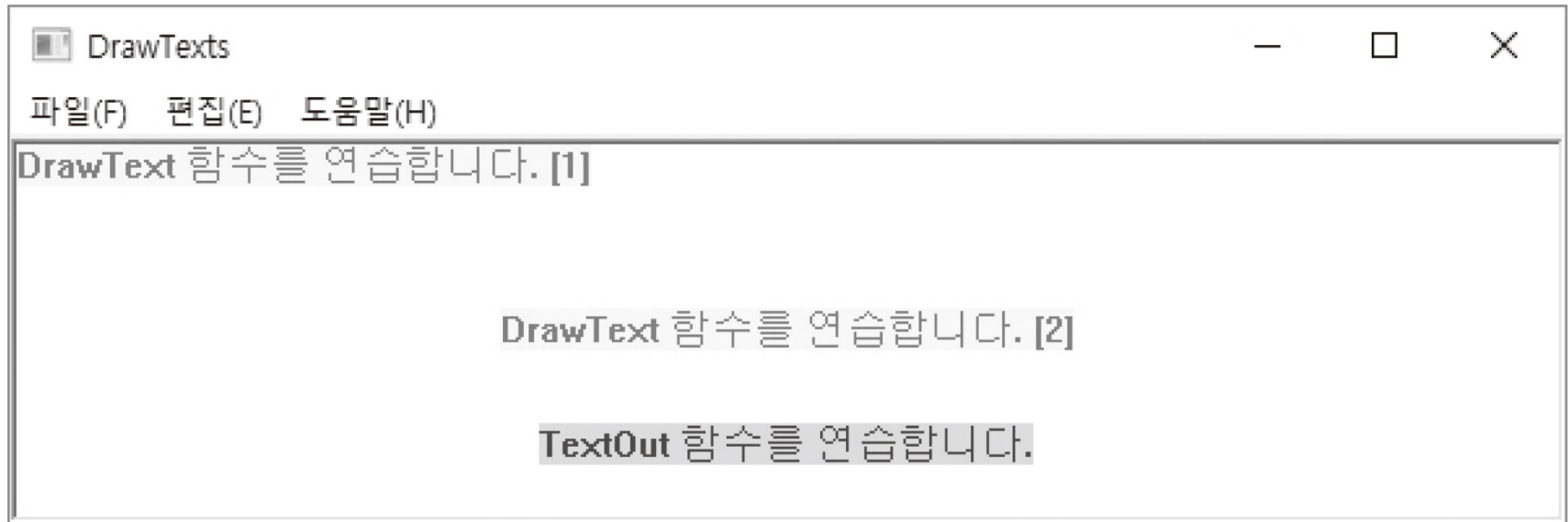


그림 4-18 실행 결과

[실습 4-7] 텍스트 출력하기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);

    CRect rect;
    GetClientRect(&rect);

    dc.SetTextColor(RED); // 글자를 빨간색으로 설정
    dc.SetBkColor(YELLOW); // 배경을 노란색으로 설정
    dc.DrawText(CString("DrawText 함수를 연습합니다. [1]"), &rect, 0);
    dc.DrawText(CString("DrawText 함수를 연습합니다. [2]"), &rect,
        DT_CENTER | DT_VCENTER | DT_SINGLELINE);

    dc.SetTextAlign(TA_CENTER); // 가운데 정렬 방식으로 변경
    dc.SetTextColor(BLUE); // 글자를 파란색으로 설정
    dc.SetBkColor(GREEN); // 배경을 초록색으로 설정
    dc.TextOut(rect.right / 2, 3 * rect.bottom / 4,
        CString("TextOut 함수를 연습합니다."));
}
```

매핑 모드

■ 매핑 모드

표 4-6 매핑 모드

매핑 모드	단위	x축	y축
MM_TEXT	1픽셀	→ +x	↓ +y
MM_LOMETRIC	0.1mm	→ +x	↓ -y
MM_HIMETRIC	0.01mm	→ +x	↓ -y
MM_LOENGLISH	0.01인치	→ +x	↓ -y
MM_HIENGLISH	0.001인치	→ +x	↓ -y
MM_TWIPS	1/1440인치	→ +x	↓ -y
MM_ISOTROPIC	사용자 정의(가로/세로 길이 같음)	사용자 정의	사용자 정의
MM_ANISOTROPIC	사용자 정의(가로/세로 길이 다름)	사용자 정의	사용자 정의

매핑 모드

■ MM_TEXT

- 디바이스 컨텍스트 최초 생성시 자동 설정된 매핑 모드
- 논리 단위와 장치 단위가 1:1로 대응
 - 함수에 사용하는 단위 = 픽셀

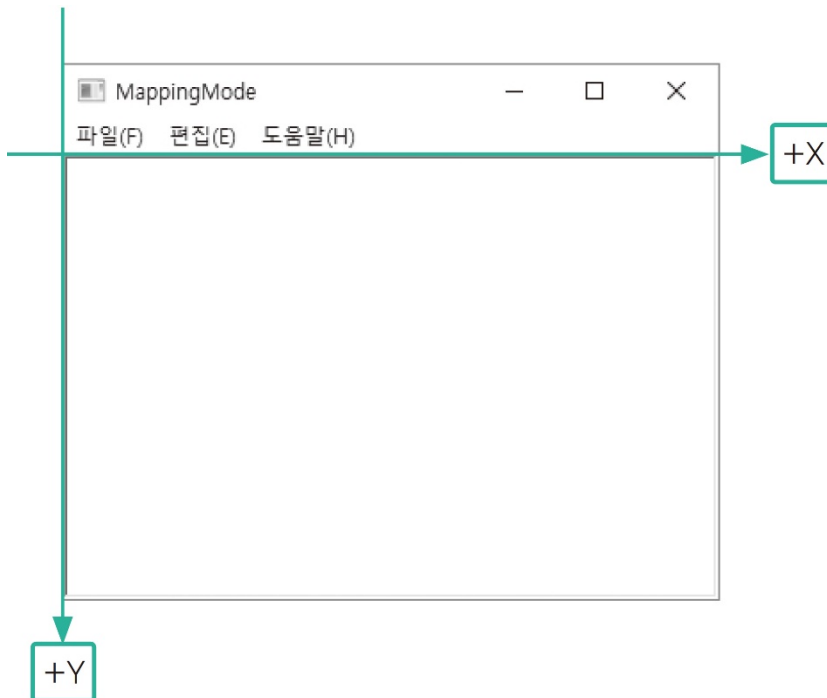
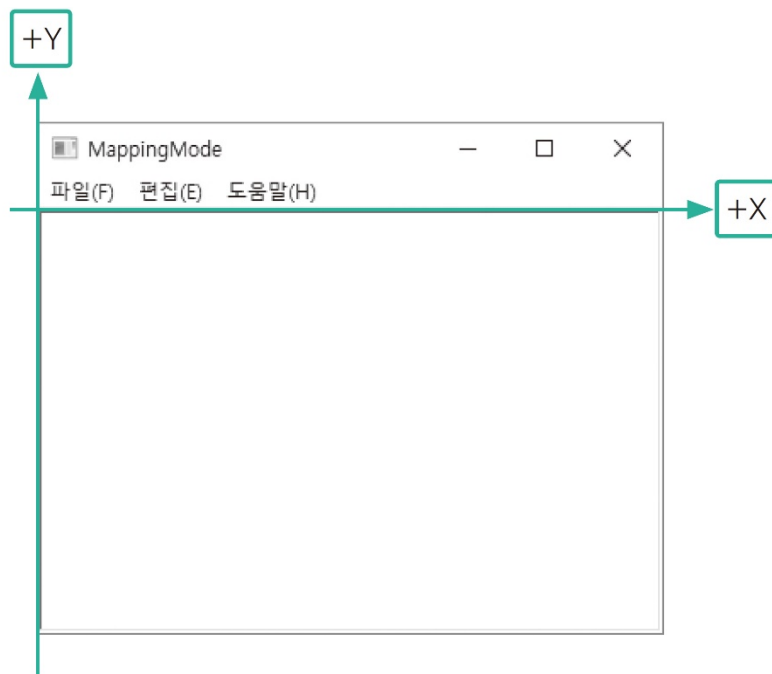


그림 4-19 MM_TEXT

매핑 모드

■ MM_LOMETRIC~MM_TWIPS

- 논리 단위가 장치 단위로 변환될 때, 출력장치의 종류에 관계 없이 물리적 길이가 항상 일정하도록 설계됨
- 장치 독립적인 출력 가능



매핑 모드

■ MM_ISOTROPIC, MM_ANISOTROPIC

- 논리 단위와 장치 단위의 변환 관계를 사용자가 정의 가능
- 프로그래밍 가능한 매핑 모드

[실습 4-8] 매핑 모드 사용하기 1

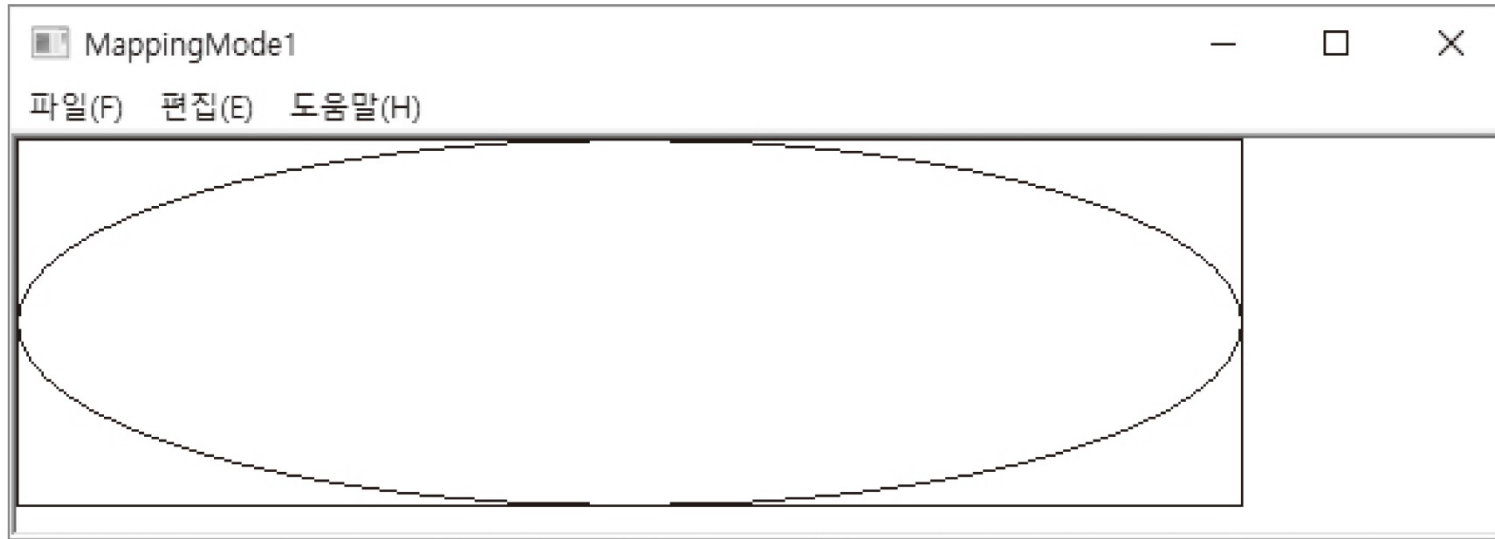


그림 4-21 실행 결과

[실습 4-8] 매핑 모드 사용하기 1

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);
    dc.SetMapMode(MM_LOMETRIC); // 매핑 모드 변경
    dc.Rectangle(0, 0, 1000, -300);
    dc.SetMapMode(MM_HIMETRIC); // 매핑 모드 변경
    dc.Ellipse(0, 0, 10000, -3000);
}
```

[실습 4-9] 매핑 모드 사용하기 2

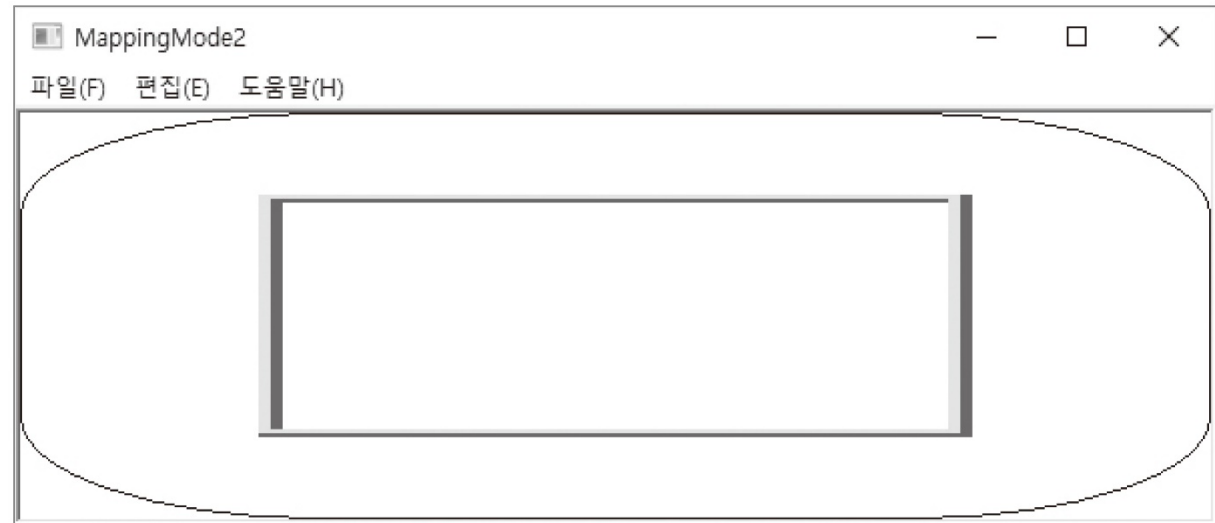


그림 4-23 실행 결과

[실습 4-9] 매핑 모드 사용하기 2

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);
    CRect rect;
    GetClientRect(&rect);
    dc.SetMapMode(MM_ANISOTROPIC);
    dc.SetWindowExt(100, 100);
    dc.SetViewportExt(rect.Width(), rect.Height());
    dc.RoundRect(0, 0, 100, 100, 50, 50);
    dc.DrawEdge(CRect(20, 20, 80, 80),
        BDR_SUNKENINNER | BDR_RAISEDOUTER, BF_RECT);
}
```

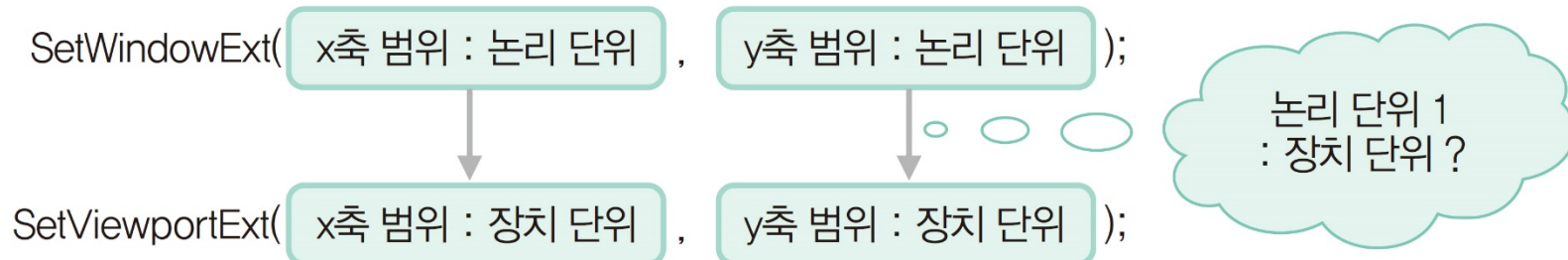


그림 4-22 SetWindowExt()와 SetViewportExt() 함수의 관계

좌표 변환

■ 논리 좌표 ↔ 장치 좌표

```
void CDC::LPtoDP(LPPOINT lpPoints, int nCount=1);  
void CDC::DPtoLP(LPPOINT lpPoints, int nCount=1);
```

■ 스크린 좌표 ↔ 클라이언트 좌표

```
void CWnd::ScreenToClient(LPPOINT lpPoint);  
void CWnd::ClientToScreen(LPPOINT lpPoint);
```

좌표 변환

■ 스크린 좌표와 클라이언트 좌표

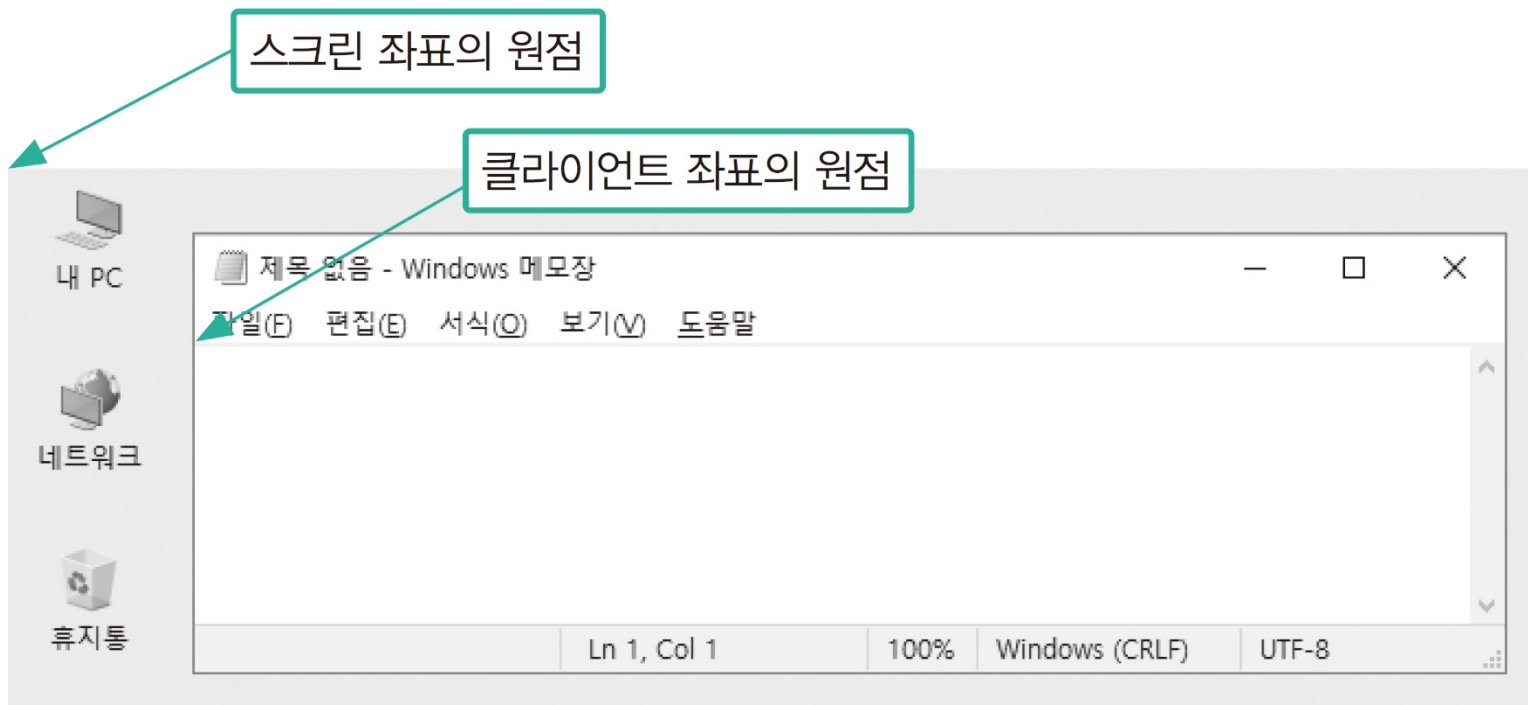


그림 4-24 스크린 좌표와 클라이언트 좌표

좌표 변환

좌표 변환 함수 간의 관계

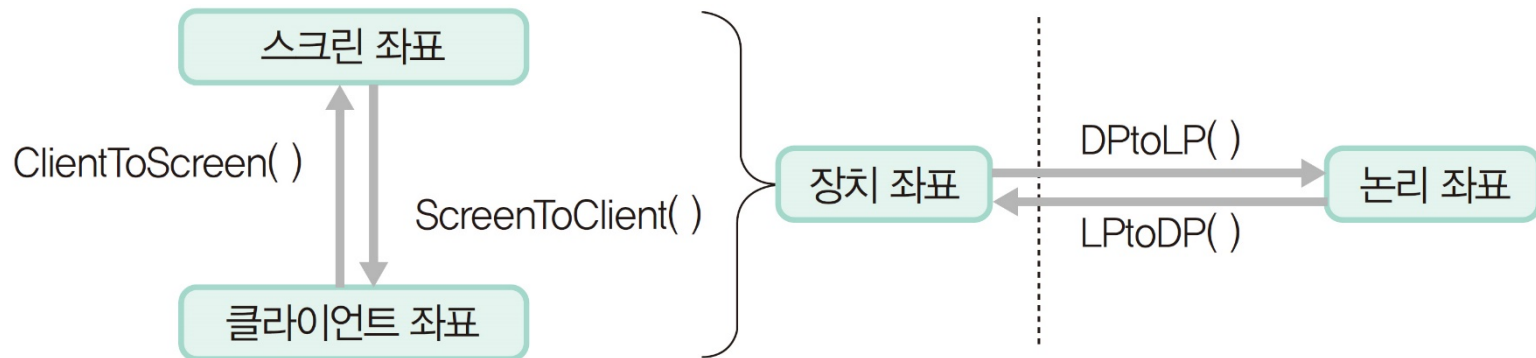


그림 4-25 좌표 변환 함수 간의 관계

속성 함수

표 4-7 속성 함수

속성	초기값	속성을 얻는 함수	속성을 변경하는 함수
텍스트 색상	검은색	GetTextColor()	SetTextColor()
배경 색상	흰색	GetBkColor()	SetBkColor()
배경 모드	OPAQUE	GetBkMode()	SetBkMode()
매핑 모드	MM_TEXT	GetMapMode()	SetMapMode()
그리기 모드	R2_COPYPEN	GetROP2()	SetROP2()
현재 위치	(0, 0)	GetCurrentPosition()	MoveTo()
펜	BLACK_PEN	SelectObject()	SelectObject()
브러시	WHITE_BRUSH	SelectObject()	SelectObject()
폰트	SYSTEM_FONT	SelectObject()	SelectObject()
비트맵	없음	SelectObject()	SelectObject()
팔레트	없음	SelectPalette()	SelectPalette()
리전	없음	SelectObject()	SelectObject()

그리기 모드

표 4-8 그리기 모드

그리기 모드	연산	그리기 모드	연산
R2_NOP	$D = D$	R2_MERGENOTPEN	$D = \sim S \mid D$
R2_NOT	$D = \sim D$	R2_MASKNOTPEN	$D = \sim S \& D$
R2_BLACK	$D = \text{BLACK}$	R2_MERGE PEN	$D = D \mid S$
R2_WHITE	$D = \text{WHITE}$	R2_NOTMERGE PEN	$D = \sim(D \mid S)$
R2_COPYPEN	$D = S$	R2_MASKPEN	$D = D \& S$
R2_NOTCOPYPEN	$D = \sim S$	R2_NOTMASKPEN	$D = \sim(D \& S)$
R2_MERGE PENNOT	$D = \sim D \mid S$	R2_XORPEN	$D = S \wedge D$
R2_MASKPENNOT	$D = \sim D \& S$	R2_NOTXORPEN	$D = \sim(S \wedge D)$

[실습 4-10] 배경 모드 변경하기

```
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle |= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS,
        ::LoadCursor(nullptr, IDC_ARROW), (HBRUSH)GetStockObject(GRAY_BRUSH), nullptr);

    return TRUE;
}
```

[실습 4-10] 배경 모드 변경하기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);
    dc.SetBkMode(OPAQUE); // 이 줄을 주석 처리해도 결과는 같음
    dc.TextOut(100, 50, CString(" OPAQUE 모드 [1] "));

    dc.SetBkMode(TRANSPARENT);
    dc.TextOut(100, 100, CString(" TRANSPARENT 모드 "));

    dc.SetBkMode(OPAQUE);
    dc.SetBkColor(RGB(0, 255, 0)); // 배경을 초록색으로 설정
    dc.TextOut(100, 150, CString(" OPAQUE 모드 [2] "));
}
```

[실습 4-10] 배경 모드 변경하기

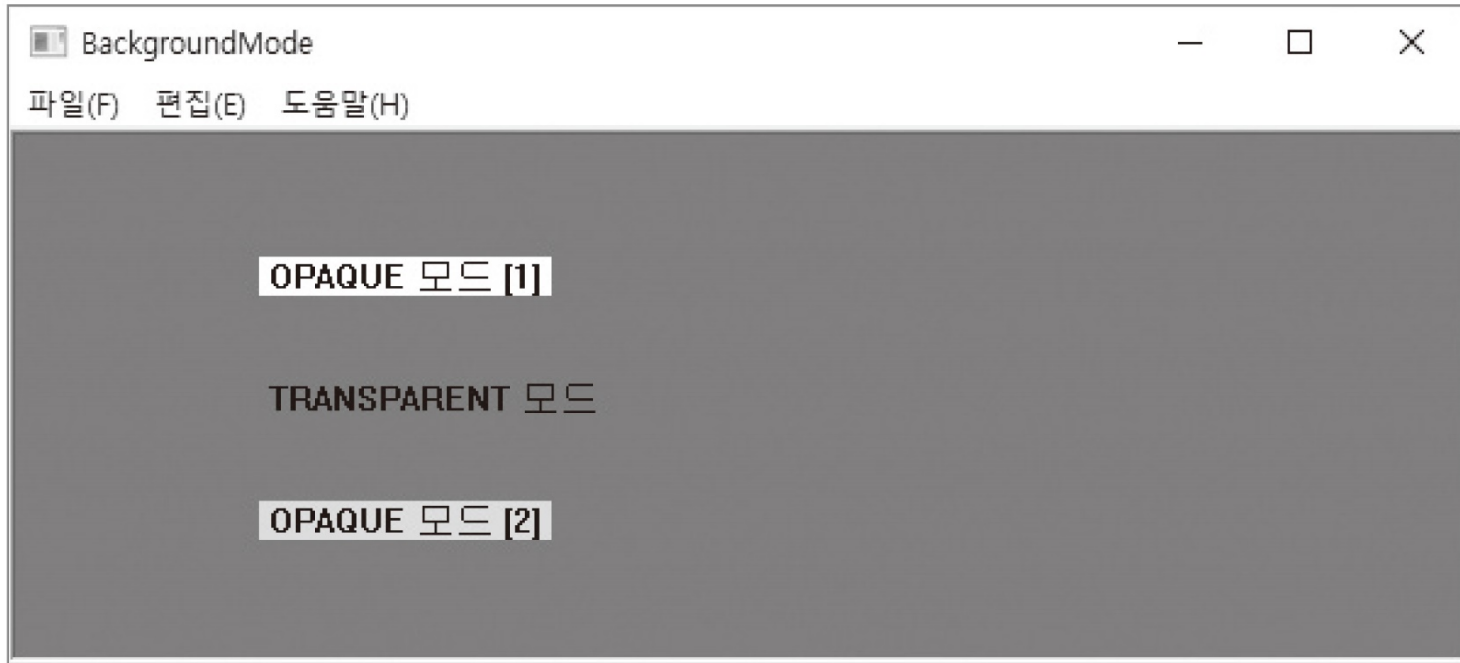


그림 4-26 실행 결과

GDI 객체

■ GDI 객체

- GDI에서 출력할 때 사용하는 도구

■ 종류

표 4-9 GDI 객체

GDI 객체	용도	클래스 이름
펜	선을 그릴 때	CPen
브러시	면의 내부를 채울 때	CBrush
폰트	글자를 출력할 때	CFont
비트맵	그림을 출력할 때	CBitmap
팔레트	출력할 색의 집합을 다룰 때	CPalette
리전	다양한 형태의 면을 정의할 때	CRgn

GDI 객체

■ MFC 클래스 계층도

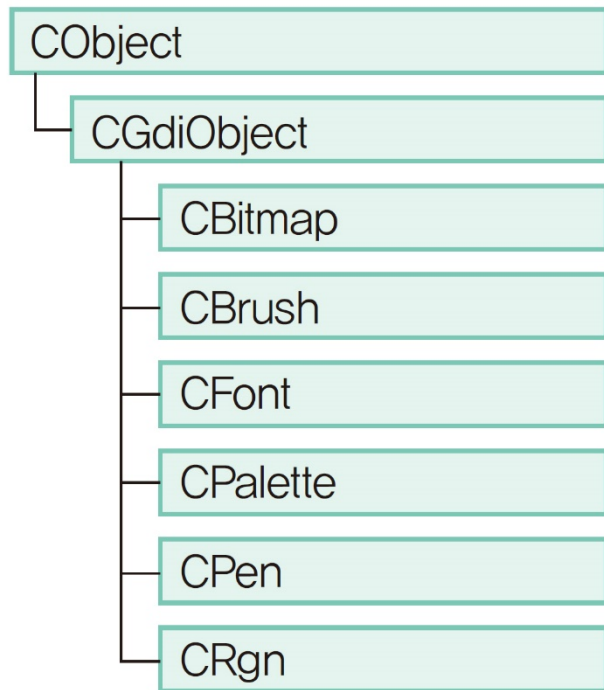


그림 4-27 MFC 클래스 계층도

GDI 객체

■ GDI 객체 사용 방법

- GDI 객체를 스택에 생성
- 생성된 GDI 객체를 CDC::SelectObject() 함수에 전달하여 디바이스 컨텍스트에 선택
- CDC 클래스 멤버 함수를 호출하여 출력
- 이전의 GDI 객체를 CDC::SelectObject() 함수에 전달하여 복원
- 함수가 끝나면 GDI 객체의 소멸자가 자동으로 호출되어 파괴됨

CPen* SelectObject(CPen* pPen); // 새로운 펜 전달, 이전의 펜 리턴

CBrush* SelectObject(CBrush* pBrush); // 새로운 브러시 전달, 이전의 브러시 리턴

CFont* SelectObject(CFont* pFont); // 새로운 폰트 전달, 이전의 폰트 리턴

CBitmap* SelectObject(CBitmap* pBitmap); // 새로운 비트맵 전달, 이전의 비트맵 리턴

int SelectObject(CRgn* pRgn); // 새로운 리전 전달, 생성된 리전의 타입 리턴

펜

■ 펜 생성 방법 1

```
CPen pen(PS_SOLID, 2, RGB(255, 0, 0)); // 폭이 2인 빨간색 펜을 만든다.
```

■ 펜 생성 방법 2

```
CPen pen;  
pen.CreatePen(PS_SOLID, 2, RGB(255, 0, 0)); // 폭이 2인 빨간색 펜을 만든다.
```

펜

■ 예

```
CPaintDC dc(this); // 디바이스 컨텍스트 객체를 만든다.  
CPen pen(PS_SOLID, 1, RGB(0, 0, 255)); // 펜 객체를 만든다.  
CPen* pOldPen = dc.SelectObject(&pen); // 새로운 펜 선택, 이전 펜 저장  
dc.Rectangle(100, 100, 200, 200); // 직사각형을 그린다.  
dc.SelectObject(pOldPen); // 이전 펜 복원, 사용하던 펜 선택 해제
```

```
CPaintDC dc(this); // 디바이스 컨텍스트 객체를 만든다.  
CPen pen(PS_SOLID, 1, RGB(0, 0, 255)); // 펜 객체를 만든다.  
dc.SelectObject(&pen); // 새로운 펜을 선택한다.  
dc.Rectangle(100, 100, 200, 200); // 직사각형을 그린다.
```

[실습 4-11] 펜 사용하기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);

    // 다양한 펜 종류를 연습한다.
    int nPenStyle[] = { PS_SOLID, PS_DASH,
        PS_DOT, PS_DASHDOT, PS_DASHDOTDOT,
        PS_NULL, PS_INSIDEFRAME };
    TCHAR* PenStyle[] = { _T("PS_SOLID"), _T("PS_DASH"),
        _T("PS_DOT"), _T("PS_DASHDOT"), _T("PS_DASHDOTDOT"),
        _T("PS_NULL"), _T("PS_INSIDEFRAME") };
    dc.SetTextAlign(TA_BASELINE);
    for (int i = 0; i < sizeof(nPenStyle) / sizeof(nPenStyle[0]); i++) {
        CPen pen(nPenStyle[i], 1, RGB(0, 0, 255));
        dc.SelectObject(&pen);
        dc.TextOut(50, 25 + i * 25, PenStyle[i], lstrlen(PenStyle[i]));
        dc.MoveTo(200, 25 + i * 25);
        dc.LineTo(500, 25 + i * 25);
    }
}
```

[실습 4-11] 펜 사용하기

```
CPen blackpen(PS_SOLID, 1, RGB(0, 0, 0)); // 폭 1, 검은색 PS_SOLID 펜
```

```
dc.SelectObject(&blackpen);
```

```
dc.Rectangle(50, 200, 150, 300); // 폭과 높이가 100인 직사각형
```

```
CPen pen1(PS_SOLID, 20, RGB(255, 0, 0)); // 폭 20, 빨간색 PS_SOLID 펜
```

```
dc.SelectObject(&pen1);
```

```
dc.Ellipse(50, 200, 150, 300); // 지름이 100인 원
```

```
dc.SelectObject(&blackpen);
```

```
dc.Rectangle(250, 200, 350, 300); // 폭과 높이가 100인 직사각형
```

```
CPen pen2(PS_INSIDEFRAME, 20, RGB(255, 0, 0)); // 폭 20, 빨간색 PS_INSIDEFRAME 펜
```

```
dc.SelectObject(&pen2);
```

```
dc.Ellipse(250, 200, 350, 300); // 지름이 100인 원
```

```
}
```

[실습 4-11] 펜 사용하기

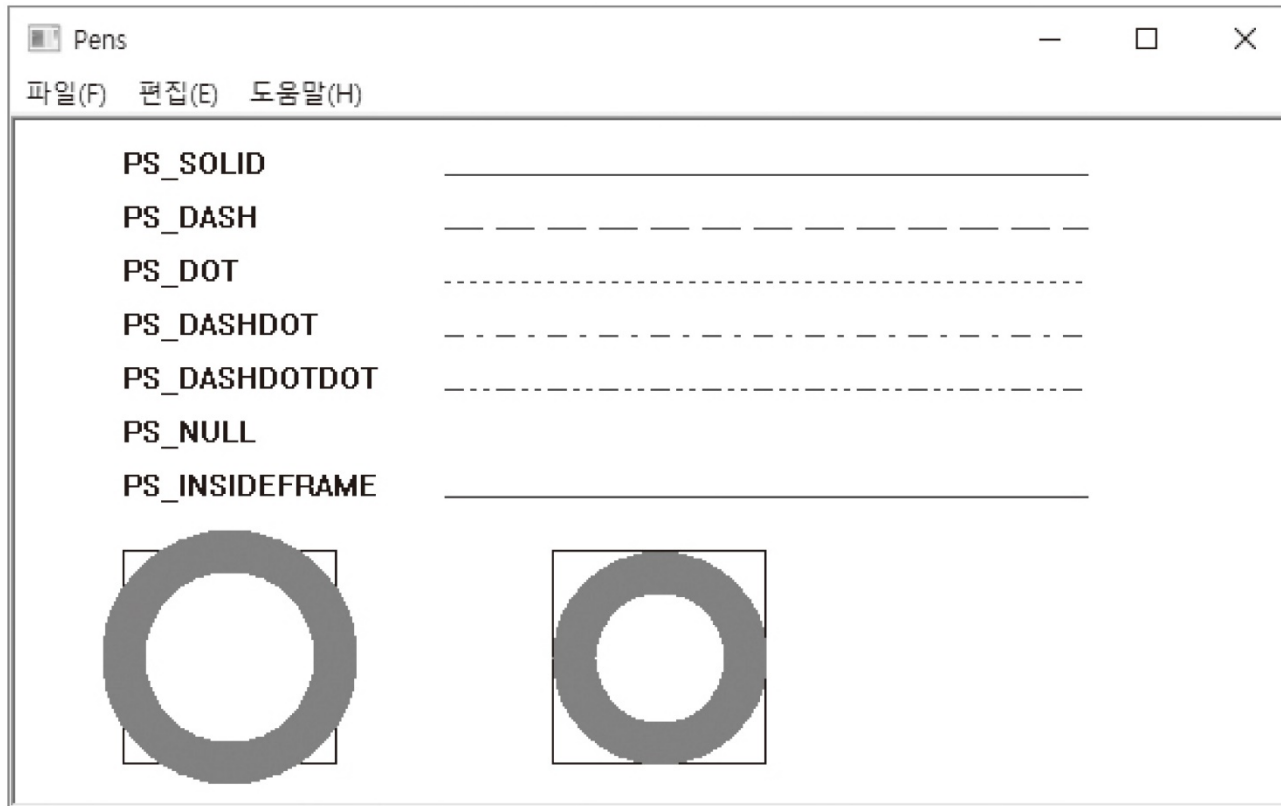


그림 4-28 실행 결과

브러시

■ 브러시 종류

표 4-10 브러시 종류

종류	생성 예
솔리드Solid(속이 채워짐)	<code>CBrush brush(RGB(255, 0, 0));</code>
해치Hatch(교차하는 평행선 무늬)	<code>CBrush brush(HS_DIAGCROSS, RGB(255, 0, 0));</code>
패턴Pattern(비트맵의 반복 무늬)	<code>CBitmap bitmap; // 비트맵 객체 생성 bitmap.LoadBitmap(IDB_BITMAP1); // 비트맵 로드 CBrush brush(&bitmap);</code>

브러시

■ 브러시 사용 예 1

```
CPaintDC dc(this); // 디바이스 컨텍스트 객체를 만든다.  
CBrush brush(RGB(255, 0, 0)); // 브러시 객체를 만든다.  
CBrush* pOldBrush = dc.SelectObject(&brush); // 새로운 브러시 선택, 이전 브러시 저장  
dc.Ellipse(100, 100, 200, 200); // 원을 그린다.  
dc.SelectObject(pOldBrush); // 이전 브러시 복원, 사용하던 브러시 선택 해제
```

■ 브러시 사용 예 2

```
CPaintDC dc(this); // 디바이스 컨텍스트 객체를 만든다.  
CBrush brush(RGB(255, 0, 0)); // 브러시 객체를 만든다.  
dc.SelectObject(&brush); // 새로운 브러시를 선택한다.  
dc.Ellipse(100, 100, 200, 200); // 원을 그린다.
```

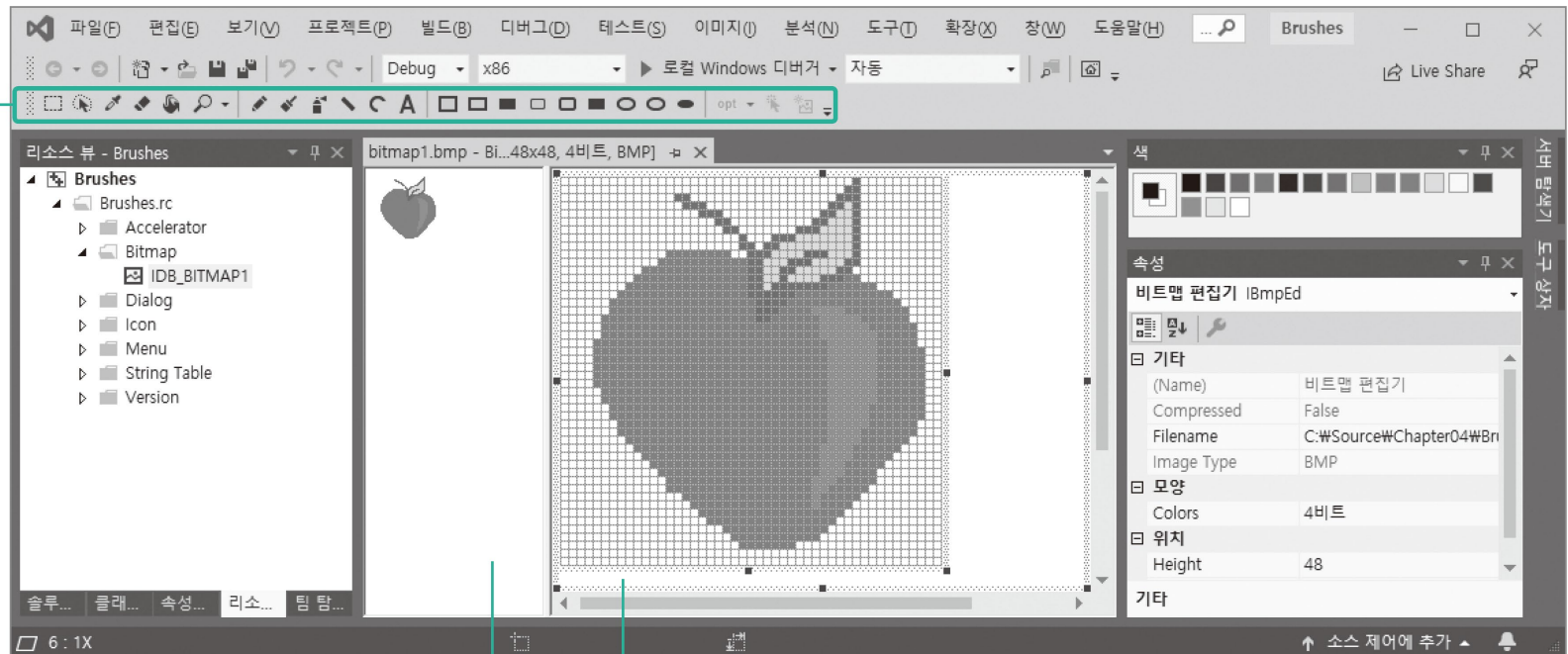
[실습 4-12] 브러시 사용하기



그림 4-29 비트맵 리소스 추가

[실습 4-12] 브러시 사용하기

이미지 편집기



빈 공간을 더블 클릭하면 오른쪽의 속성 창이 열린다.

[실습 4-12] 브러시 사용하기

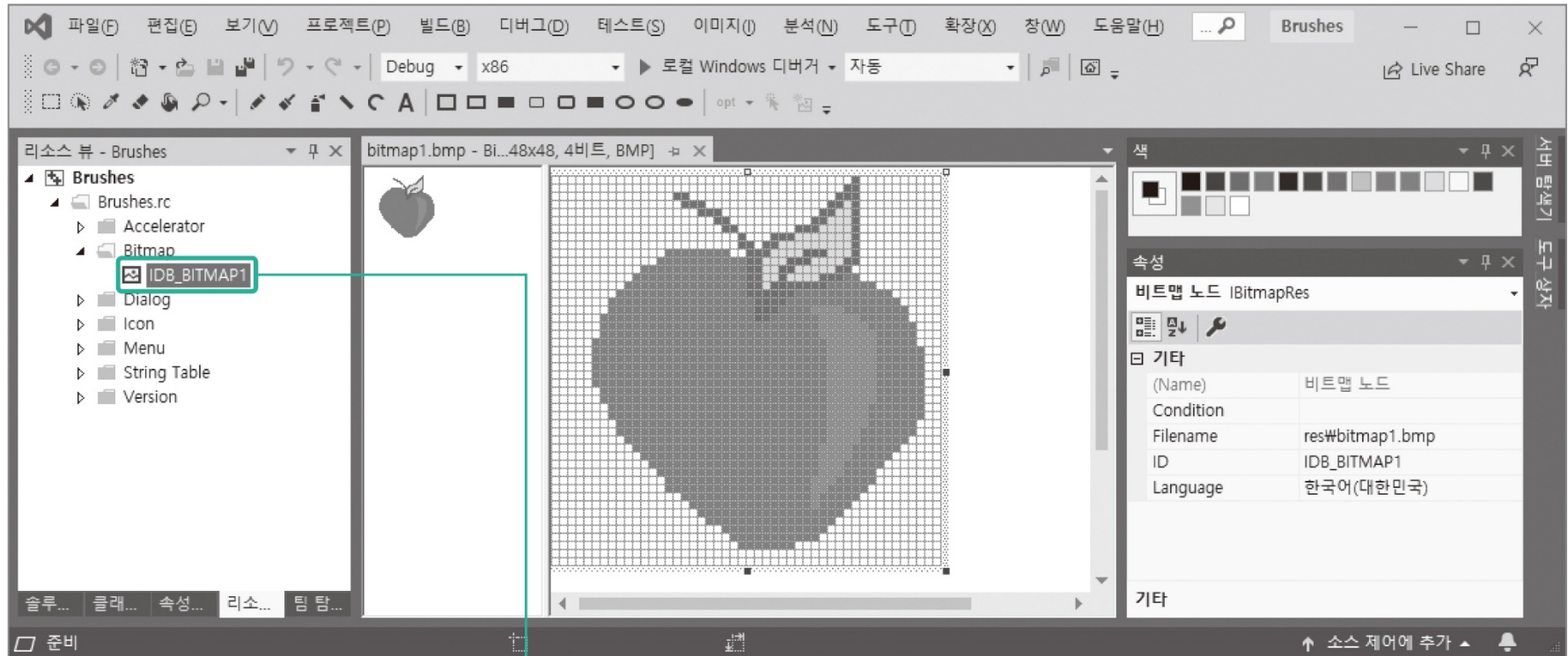


그림 4-30 비트맵 리소스 편집

여기를 클릭하면 오른쪽의 속성 창 내용이 바뀐다.

[실습 4-12] 브러시 사용하기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);

    // 솔리드 브러시
    CBrush brush1(RGB(255, 0, 0));
    dc.SelectObject(&brush1);
    dc.Rectangle(50, 50, 200, 200);

    // 해치 브러시
    CBrush brush2(HS_DIAGCROSS, RGB(255, 0, 0));
    dc.SelectObject(&brush2);
    dc.Ellipse(250, 50, 400, 200);

    // 패턴 브러시
    CBitmap bitmap;
    bitmap.LoadBitmap(IDB_BITMAP1);
    CBrush brush3(&bitmap);
    dc.SelectObject(&brush3);
    dc.RoundRect(450, 50, 600, 200, 50, 50);
}
```

[실습 4-12] 브러시 사용하기

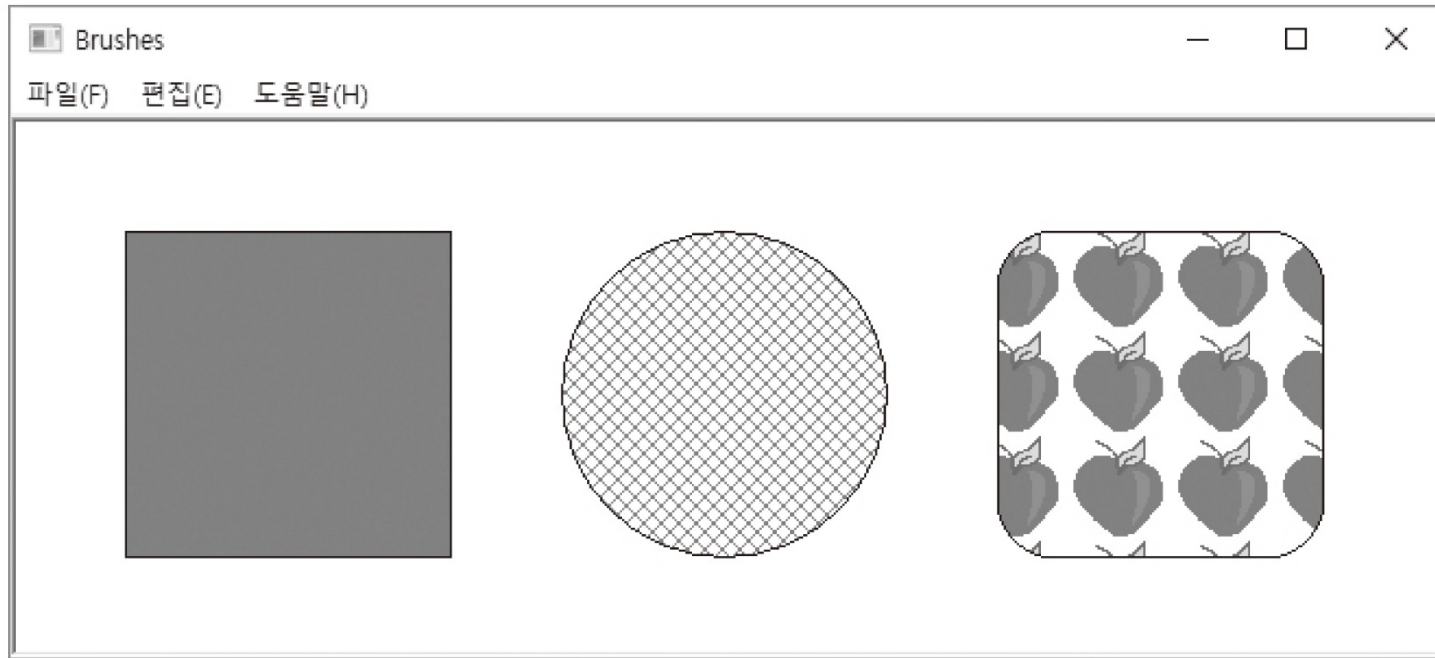


그림 4-31 실행 결과

폰트

■ 폰트 생성 방법

- 폰트 객체 생성

```
CFont font;
```

- 폰트 객체에 대해 Create*() 함수 호출

```
font.CreateFont(...);  
또는 font.CreateFontIndirect(...);  
또는 font.CreatePointFont(...);  
또는 font.CreatePointFontIndirect(...);
```

- 예

```
CPaintDC dc(this); // 디바이스 컨텍스트 객체를 만든다.  
CFont font; // 폰트 객체를 생성한다.  
font.CreatePointFont(400, "Arial"); // 크기가 40(=400/10) 포인트인 Arial 폰트  
dc.SelectObject(&font); // 새로운 폰트를 선택한다.  
dc.TextOut(10, 10, CString("Hello")); // 텍스트를 출력한다.
```

내장 객체

표 4-11 내장 객체

이름	용도
BLACK_PEN	두께가 1인 검은색 펜
WHITE_PEN	두께가 1인 흰색 펜
NULL_PEN	투명 펜
BLACK_BRUSH	검은색 브러시
DKGRAY_BRUSH	어두운 회색 브러시
GRAY_BRUSH	회색 브러시
LTGRAY_BRUSH	밝은 회색 브러시
HOLLOW_BRUSH 또는 NULL_BRUSH	투명 브러시
SYSTEM_FONT	윈도우 운영체제가 사용하는 폰트 예) 메뉴, 대화 상자, ...

[실습 4-13] 내장 객체 사용하기

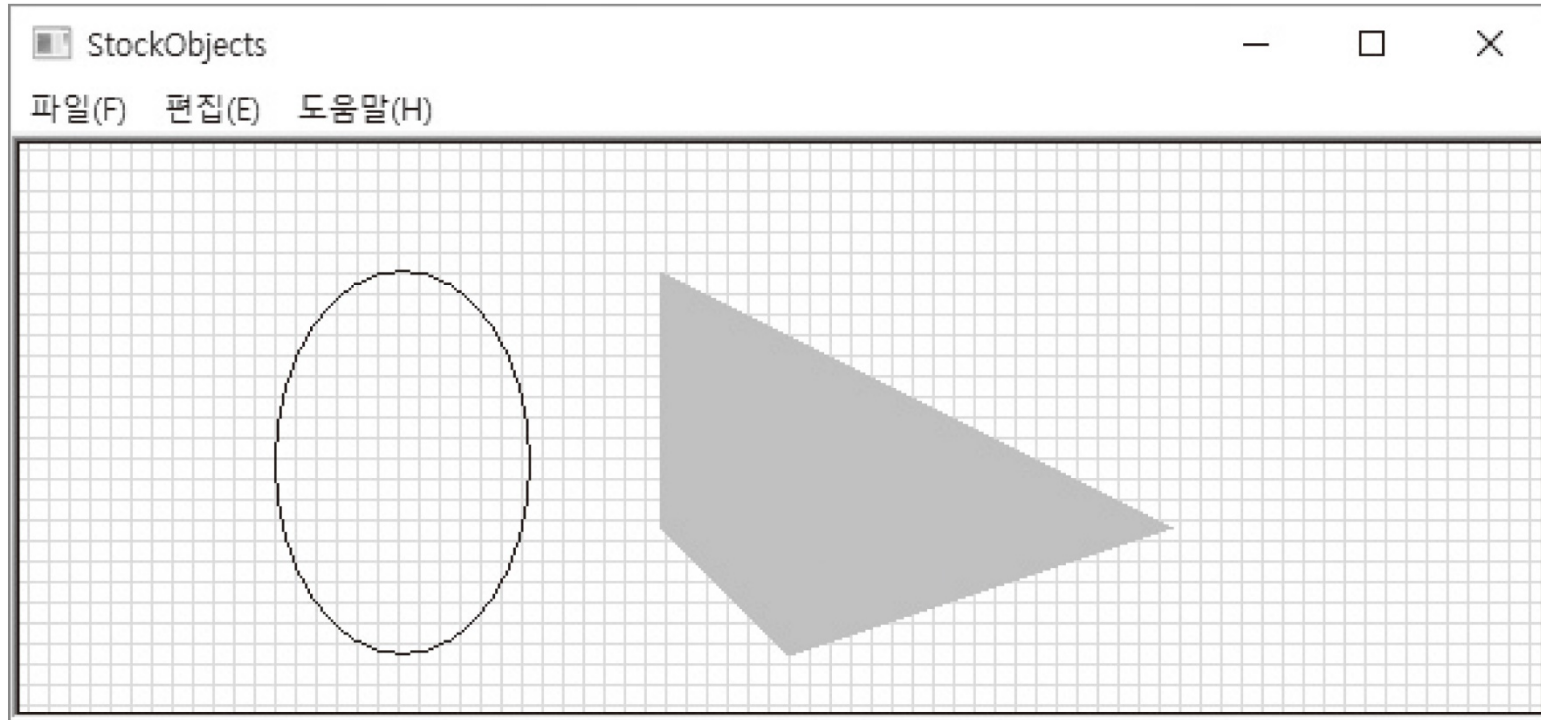


그림 4-32 실행 결과

[실습 4-13] 내장 객체 사용하기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);

    // 클라이언트 영역을 초록색 해치 브러시로 채운다.
    CRect rect;
    GetClientRect(&rect);
    CBrush brush(HS_CROSS, RGB(0, 255, 0));
    dc.SelectObject(&brush);
    dc.Rectangle(&rect);

    // 경계선이 검은색이고 내부가 비어 있는 타원을 그린다.
    dc.SelectStockObject(BLACK_PEN);
    dc.SelectStockObject(NULL_BRUSH);
    dc.Ellipse(100, 50, 200, 200);

    // 경계선이 없고 내부가 밝은 회색으로 채워진 다각형을 그린다.
    dc.SelectStockObject(NULL_PEN);
    dc.SelectStockObject(LTGRAY_BRUSH);
    POINT points[] = { {250, 50}, {450, 150}, {300, 200}, {250, 150} };
    dc.Polygon(points, 4);
}
```

[실습 4-13] 내장 객체 사용하기

```
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;
    cs.dwExStyle |= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS,
        ::LoadCursor(nullptr, IDC_ARROW), nullptr, nullptr); // NULL 배경 브러시
    return TRUE;
}
```

비트맵

■ 비트맵 정보

```
int CBitmap::GetBitmap(BITMAP* pBitMap);
```

```
typedef struct tagBITMAP {  
    int bmType;  
    int bmWidth; // 비트맵의 폭(픽셀)  
    int bmHeight; // 비트맵의 높이(픽셀)  
    int bmWidthBytes;  
    BYTE bmPlanes;  
    BYTE bmBitsPixel;  
    LPVOID bmBits;  
} BITMAP;
```

비트맵

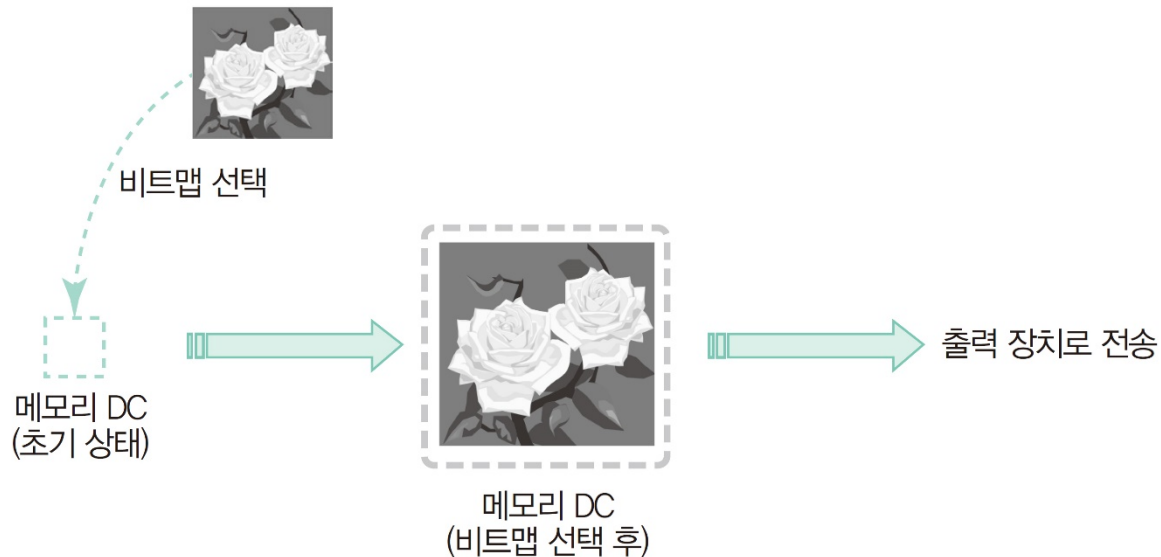
■ 비트맵 정보 확인 방법

```
CBitmap bitmap;  
bitmap.LoadBitmap(IDB_BITMAP1);  
BITMAP bmpinfo;  
bitmap.GetBitmap(&bmpinfo);  
TRACE(_T("가로 = %d, 세로 = %d\n"), bmpinfo.bmWidth, bmpinfo.bmHeight);
```

비트맵

■ 비트맵 출력 절차

- CDC::CreateCompatibleDC() 함수를 이용하여 메모리 디바이스 컨텍스트를 만듦
- CDC::SelectObject() 함수를 이용하여 비트맵을 메모리 디바이스 컨텍스트에 선택
- CDC::BitBlt()나 CDC::StretchBlt() 함수를 이용하여 화면에 출력



비트맵

■ 비트맵 출력 함수_BitBlt() 함수

```
BOOL CDC::BitBlt(int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc,
                 int ySrc, DWORD dwRop);
```

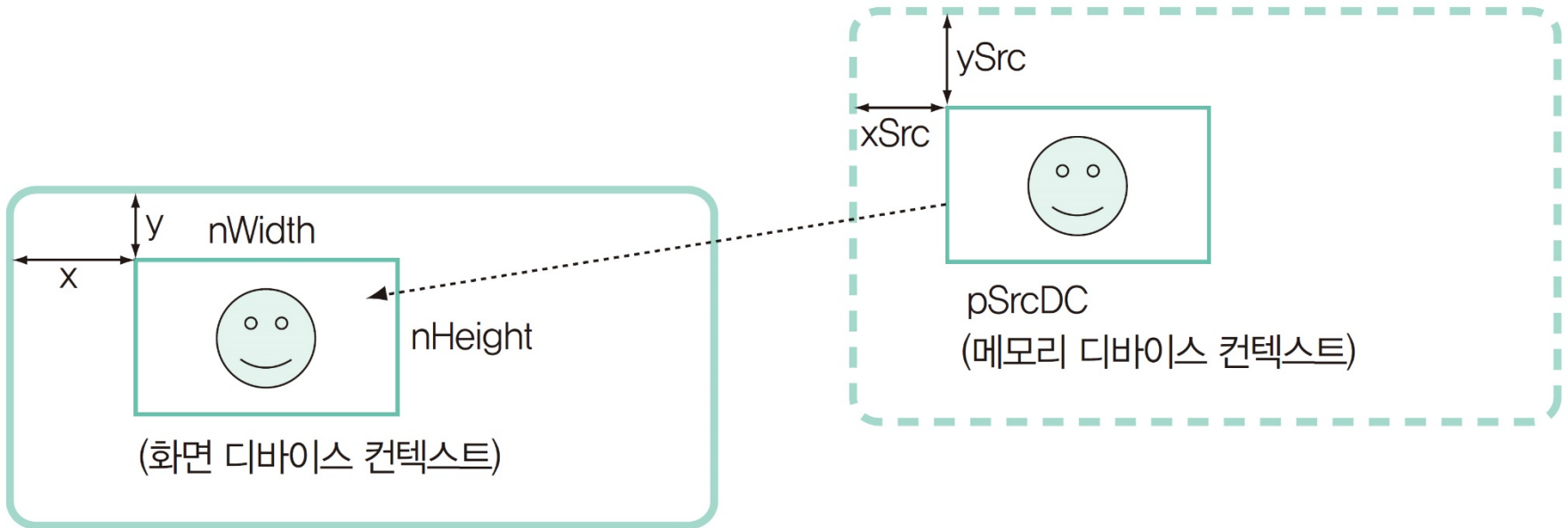


그림 4-34 BitBlt() 함수

비트맵

■ 비트맵 출력 함수 StretchBlt() 함수

```
BOOL CDC::StretchBlt(int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int xSrc,
                    int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwRop);
```

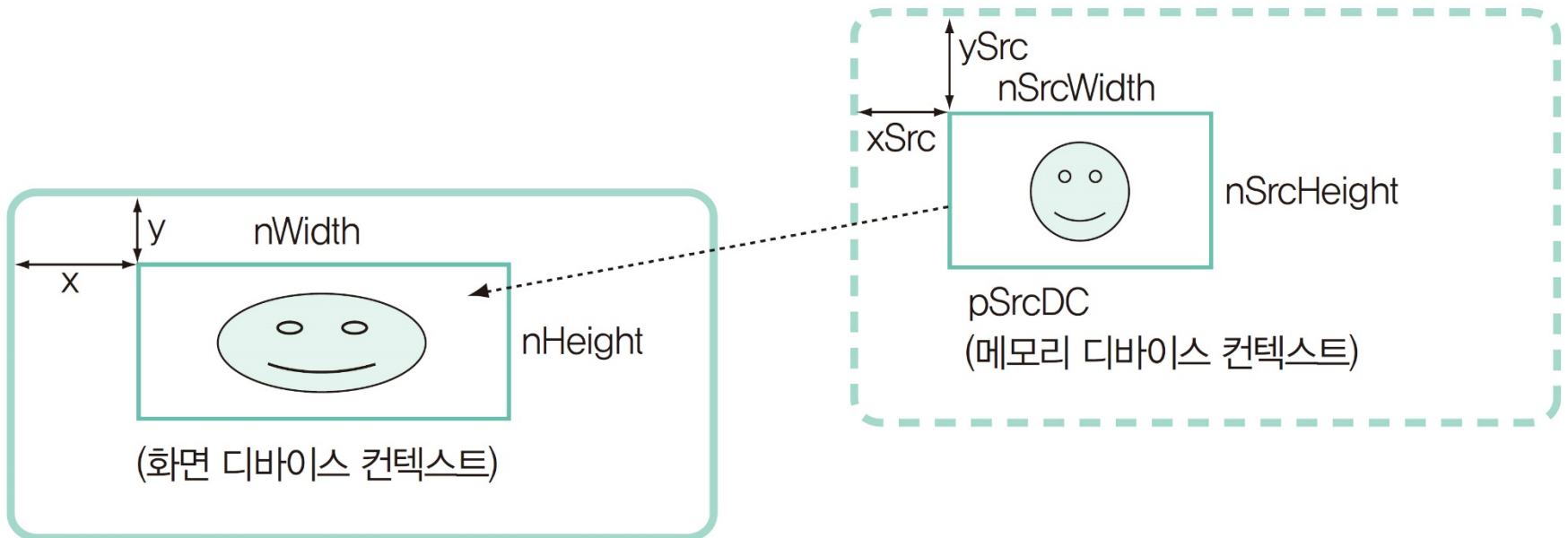


그림 4-35 StretchBlt() 함수

[실습 4-14] 비트맵 출력하기

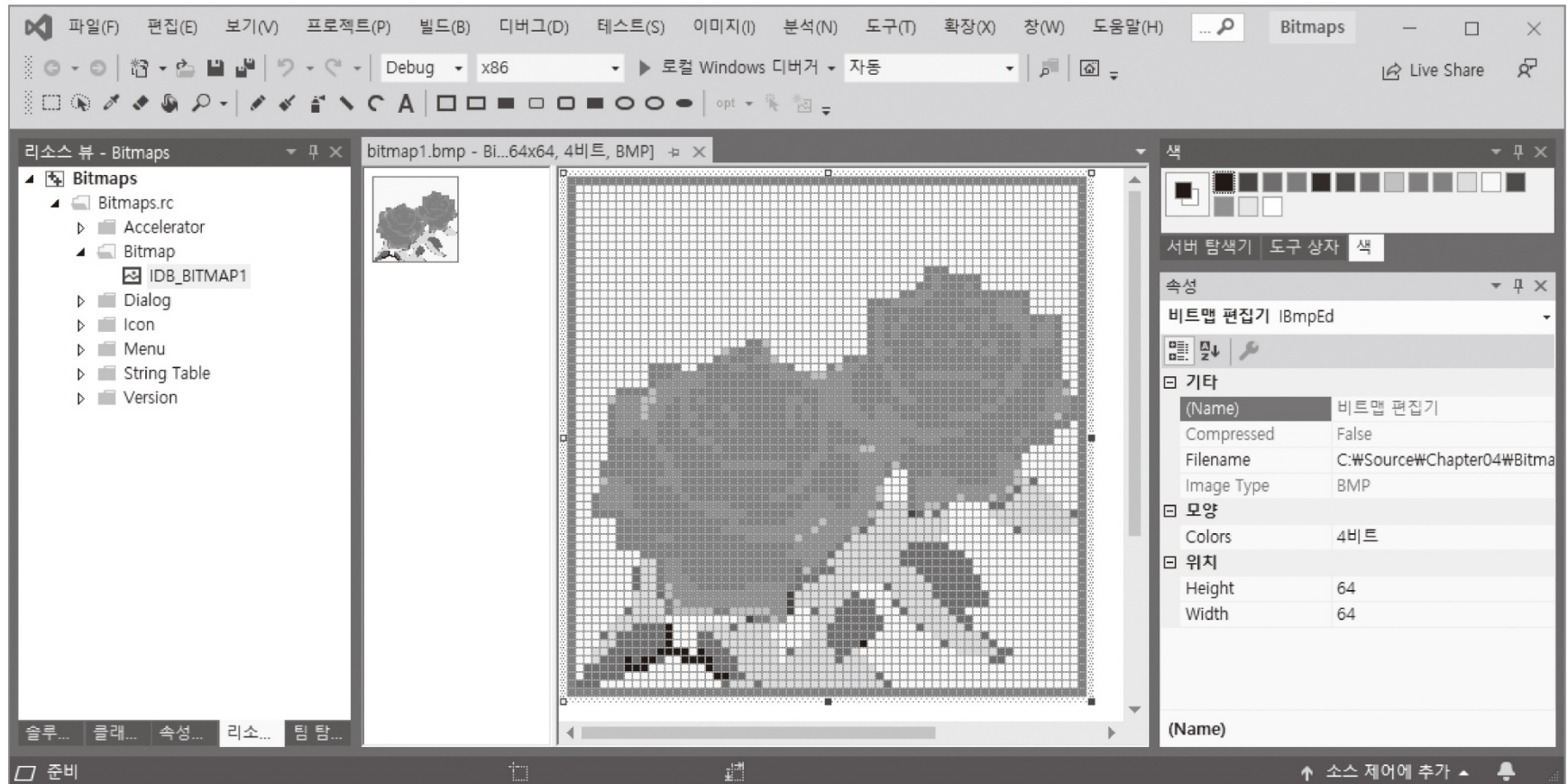


그림 4-36 비트맵 리소스 편집

[실습 4-14] 비트맵 출력하기

```
void CChildView::OnPaint()
{
    CPaintDC dc(this);

    // 비트맵 리소스를 로드한 후 크기 정보를 얻는다.
    CBitmap bitmap;
    bitmap.LoadBitmap(IDB_BITMAP1);
    BITMAP bmpinfo;
    bitmap.GetBitmap(&bmpinfo);

    // 메모리 DC를 만든 후 비트맵을 선택해 넣는다.
    CDC dcmem;
    dcmem.CreateCompatibleDC(&dc);
    dcmem.SelectObject(&bitmap);

    // 비트맵을 화면에 출력한다.
    dc.BitBlt(10, 10, bmpinfo.bmWidth, bmpinfo.bmHeight,
        &dcmem, 0, 0, SRCCOPY);
    dc.StretchBlt(10, 100, bmpinfo.bmWidth * 4, bmpinfo.bmHeight * 2,
        &dcmem, 0, 0, bmpinfo.bmWidth, bmpinfo.bmHeight, SRCCOPY);
}
```

[실습 4-14] 비트맵 출력하기

```
// 메모리 DC에 그림을 그린 후 다시 화면에 출력한다.  
dcmem.Rectangle(5, 5, 15, 15);  
dc.BitBlt(350, 10, bmpinfo.bmWidth, bmpinfo.bmHeight,  
    &dcmem, 0, 0, SRCCOPY);  
dc.StretchBlt(350, 100, bmpinfo.bmWidth * 4, bmpinfo.bmHeight * 2,  
    &dcmem, 0, 0, bmpinfo.bmWidth, bmpinfo.bmHeight, SRCCOPY);  
}
```

[실습 4-14] 비트맵 출력하기

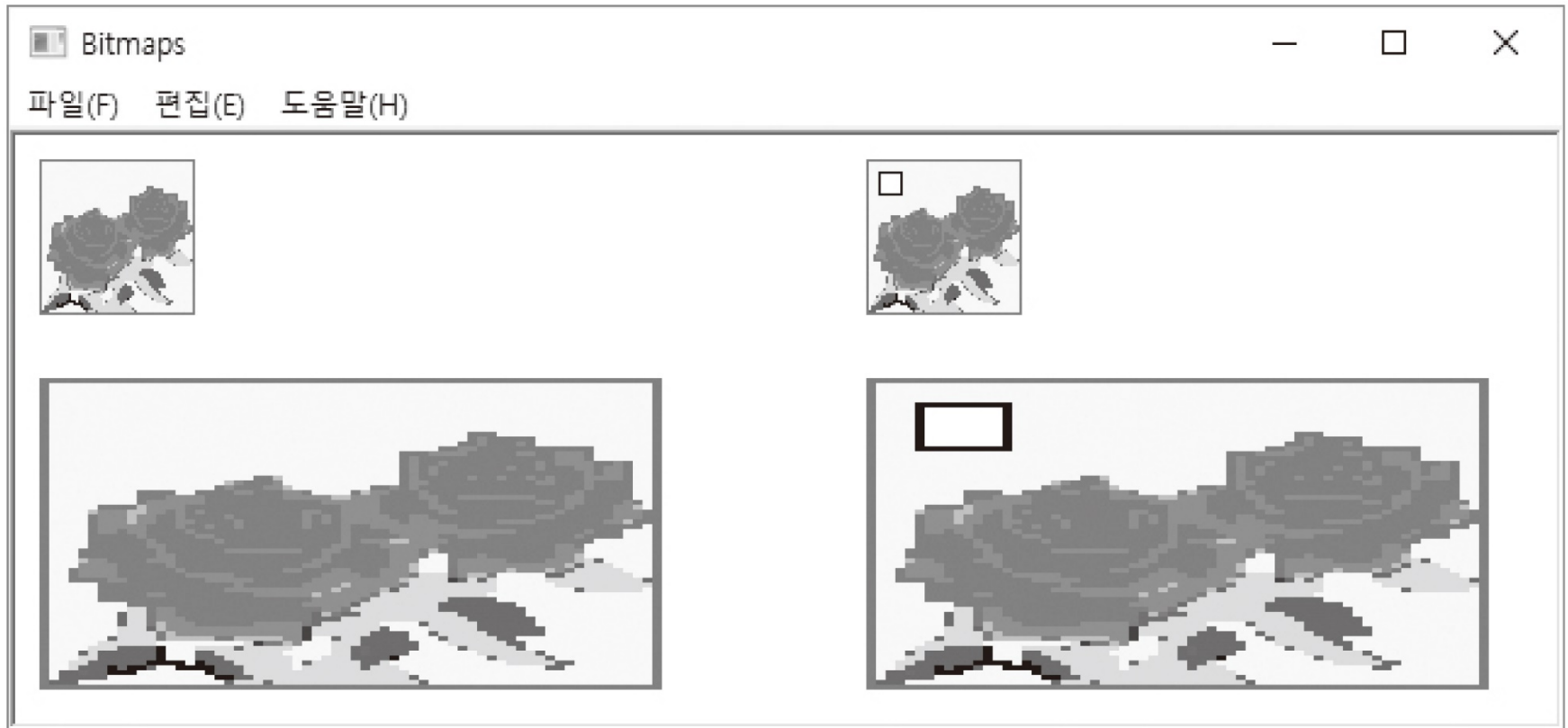


그림 4-37 실행 결과

리전

표 4-12 CRgn 클래스 주요 멤버 함수

분류	관련 함수	기능 설명
리전 객체 생성	CRgn()	리전 객체를 초기화한다.
리전 초기화	CreateRectRgn()	직사각형 리전을 만든다.
	CreateEllipticRgn()	타원형 리전을 만든다.
	CreateRoundRectRgn()	모서리가 둥근 직사각형 리전을 만든다.
	CreatePolygonRgn()	다각형 리전을 만든다.
리전 결합	CombineRgn()	두 개의 리전을 결합하여 한 개의 리전을 만든다.
리전 테스트	PtInRegion()	점이 리전 내부에 있는지 확인한다.
	RectInRegion()	직사각형이 리전 내부에 있는지 확인한다.

리전

표 4-13 리전 활용 함수

분류	관련 함수	기능 설명
리전 자체 출력	CDC::FillRgn() CDC::PaintRgn()	브러시로 리전 내부를 채운다.
	CDC::FrameRgn()	브러시로 리전에 테두리를 그린다.
	CDC::InvertRgn()	리전 내부 색상을 반전시킨다.
화면 출력 제한	CWnd::InvalidateRgn()	클라이언트 영역에서 리전이 나타내는 부분만 무효화한다.
	CDC::SelectClipRgn()	디바이스 컨텍스트로 출력하는 내용을 리전 내부로 제한한다.
	CWnd::SetWindowRgn()	윈도우가 화면에 보이는 영역을 리전 내부로 제한한다.

리전

```
01 void CChildView::OnPaint()
02 {
03     CPaintDC dc(this);
04     CRgn rgn1, rgn2, rgn3;
05     rgn1.CreateRectRgn(0, 0, 1, 1); // 임의로 만든 리전
06     rgn2.CreateRectRgn(10, 10, 100, 100);
07     rgn3.CreateEllipticRgn(50, 50, 200, 200);
08     rgn1.CombineRgn(&rgn2, &rgn3, RGN_XOR);
09     CBrush brush(RGB(0, 0, 255));
10     dc.FillRgn(&rgn1, &brush);
11 }
```

