8장
윈도우에서의 그리기

김성영교수
금오공과대학교
컴퓨터공학부

그래픽 장치 인터페이스 객체

- **GDI 객체**

객체의 핸들타입	MFC 클래스	속 성
HPEN	CPen	선 속성
HBRUSH	CBrush	채우기 속성
HFONT	CFont	글꼴 속성
HBITMAP	CBitmap	비트맵 영상 속성
HPALETTE	CPalette	팔레트 속성
HRGN	CRgn	영역 속성

GDI 클래스는 CGdiObject 클래스로부터 상속
GDI 객체의 핸들을 할당할 수 있는 멤버변수를 가짐

```
class CGdiObject : public CObject
{
        HANDLE    m_hObject;   // GDI 객체에 대한 핸들
}
```

- 윈도우가 GDI 객체를 만들어 각 객체의 속성에 기본값 설정

- 선을 그리는 경우
  - 시작점과 끝점만 제공하면 선을 그릴 수 있음

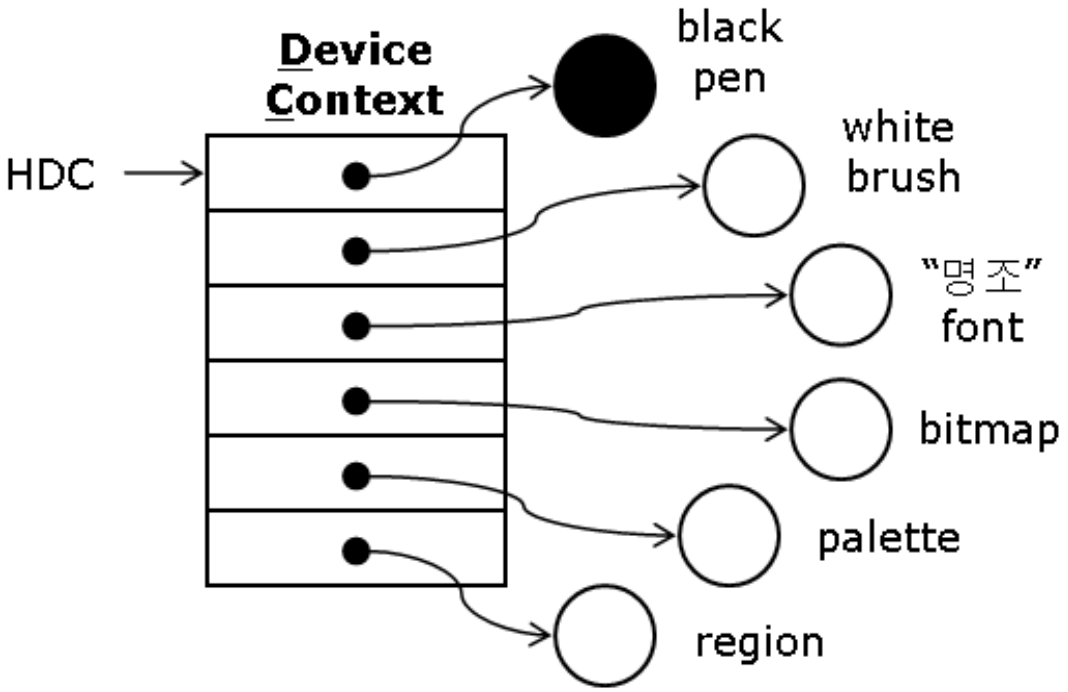
```
MoveTo( int x, int y )  // 지정한 위치로 이동
LineTo( int x, int y )   // 현재 위치에서 지정된 위치까지 선을 그음
```

- 사각형을 그리는 경우

```
Rectangle( int x1, int y1, int x2, int y2 ) //각진 사각형
RoundRect( int x1, int y1, int x2, int y2 , int x3, int y3 ) // 모서리가 둥근 사각형
```

- **CDC 클래스**

  - 디바이스 컨텍스트(Device Context, DC) 구조체
  - **HDC**는 DC를 가리키는 핸들



```
class CDC : public CObject
{
        HDC m_hDC;      // DC에 대한 핸들
        BOOL LineTo( int x, int y );
        BOOL Rectangle( int x1, int y1, int x2, int y2 );
}
```

멤버변수 : HDC
멤버함수 : Rectangle() 등의 그리기 함수

CDC 클래스를 이용한 그리기
→ GDI 객체를 가리키고 있는 DC를 준비

```
CDC* pDC;
pDC = GetDC();              // DC 요청
pDC->Rectangle(0, 0, 100, 100);   //그리기 관련 함수 (GetDC와 ReleaseDC 사이)
ReleaseDC(pDC);             // DC 반환
```

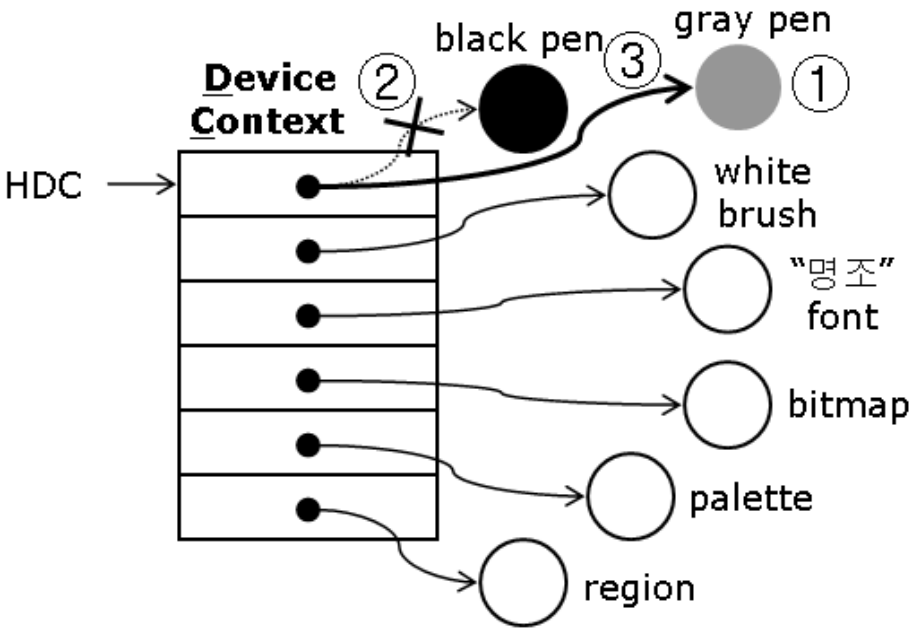
CDC: http://msdn.microsoft.com/ko-kr/library/4acfw2ha(VS.80).aspx

그리기의 전형적인 형태

기본으로 설정된 속성 값을 바꾸고자 할 때

➡ 기존에 생성된 객체의 속성을 바꾸는 대신 새로운 객체 생성

검은색 선을 회색으로 바꾸려면..



① 회색 속성을 가진 펜 객체 생성

② DC는 검은색 펜 객체를 가리키던 연결선을 끊음

③ DC는 새로 생성한 회색 펜 객체를 연결

- SelectObject()함수
  - ②, ③과정 처리
  - GDI 객체별로 존재
  - 기존 GDI 객체의 주소 값 반환

```
CPen* SelectObject( CPen* pPen );

CBrush* SelectObject( CBrush* pBrush );

CFont* SelectObject( CFont* pFont );

CBitmap* SelectObject( CBitmap* pBitmap );

int SelectObject( CRgn* pRgn );
```

- 윈도우가 생성해 놓은 DC값이 바뀔 수도 있으므로 GetDC() 함수에 의해 할당된 DC는 독점적으로 사용
- 사용이 끝나면 ReleaseDC() 함수로 사용하지 않음을 알림

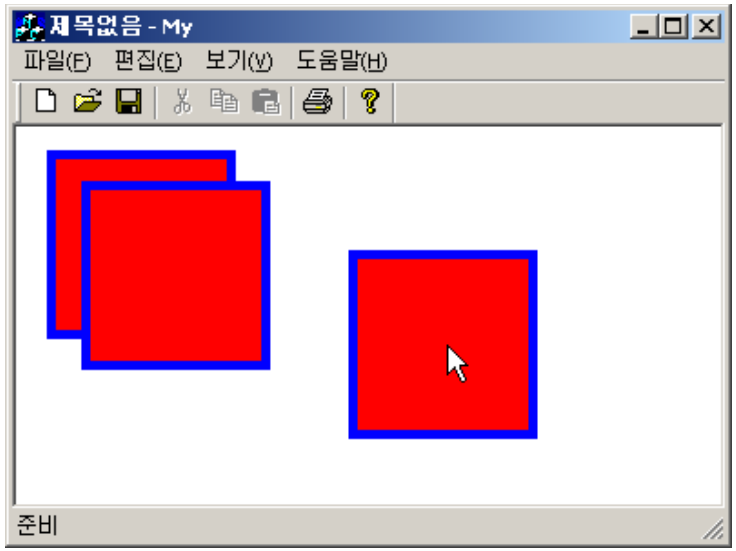
- **출력작업이 끝나면 DC를 이전 상태로 되돌림**
  - **SelectObject()함수를 다시 호출하여 이전 GDI객체 복원**
- **새로 생성하여 사용한 객체는 메모리에서 제거**

```
CDC* pDC;
pDC = GetDC();
CPen* oldPen, pen;
pen.CreatePen( PS_SOLID, 1, RGB(128,128,128) );   // 회색 펜 생성
oldPen = pDC->SelectObject( &pen );               //새로 생성한 회색 펜 선택

pDC->Rectangle( 0, 0, 100, 100 );                 //회색 테두리를 가진 사각형 그림

pDC->SelectObject( oldPen );                      //이전 검은 펜 복구
pen.DeleteObject( );                              //생성한 펜을 제거
ReleaseDC( pDC );
```

- **CClientDC 클래스를 사용하여 위 코드를 아래와 같이 대체가능**
  - **-생성자에서 GetDC(), 소멸자에서 ReleaseDC()호출**
  - **- 단, CClientDC 클래스는 클라이언트 영역만 담당**

```
CClientDC dc( this );
pen.CreatePen( PS_SOLID, 1, RGB(128,128,128) );
oldPen = dc. SelectObject( &pen );

dc.Rectangle( 0, 0, 100, 100 );

dc.SelectObject( oldPen );
pen.DeleteObject( );
```

실습 8.1



마우스 클릭 위치를 중심으로 스타일을 갖는 사각형을 그려보자.

마우스 클릭 처리는 CMyView 클래스의 WM_LBUTTONDWN 메시지 핸들러에서 처리

화면출력 준비 / 사각형 그리기는 Rectangle()함수 이용

MyView.cpp

```
void CMyView::OnLButtonDown( UINT nFlags, CPoint point )
{
        CDC* pDC;
        pDC = GetDC( );   // DC 준비

        CRect rt;
        rt.left = point.x - 50;
        rt.top = point.y - 50;
        rt.right = point.x + 50;
        rt.bottom = point.y + 50;

        pDC->Rectangle( rt );  //사각형 그림

        ReleaseDC(pDC);  // DC 반환

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

브러쉬 객체 생성은 CreateSolidBrush()함수 사용

MyView.cpp

```
void CMyView::OnLButtonDown( UINT nFlags, CPoint point )
{
        . . .
        CPen pen, *oldPen;  // 펜 객체 선언
        pen.CreatePen( PS_SOLID, 5, RGB(0,0,255) );  // 파란 펜 생성
        oldPen = pDC->SelectObject( &pen );  // 파란 펜 선택 & 이전 펜 저장

        CBrush br, *oldBr;  // 브러시 객체 선언
        br.CreateSolidBrush( RGB(255,0,0) );  // 빨간 브러쉬 생성
        oldBr = pDC->SelectObject( &br );  // 빨간 브러쉬 선택 & 이전 브러쉬 저장

        pDC->Rectangle(rt);

        pDC->SelectObject(oldPen);          // 이전 펜으로 복귀
        pDC->SelectObject(oldBr);           // 이전 브러쉬로 복귀

        pen.DeleteObject();         // 펜 객체 제거
        br.DeleteObject();          // 브러쉬 객체 제거
        . . .
}
```

참고

- 자주 사용하는 GDI 객체는 미리 생성하고 상수형태로 제공

BLACK_BRUSH, DKGRAY_BRUSH, GRAY_BRUSH
LTGRAY_BRUSH, NULL_BRUSH, WHITE_BRUSH

BLACK_PEN, NULL_PEN, WHITE_PEN

```
virtual CGdiObject* SelectStockObject( int nindex );
```

WM_PAINT 메시지

실습 8.2

[실습 8.1]에 이어서 윈도우가 다른 윈도우에 가려졌다가 다시 나타났을때 사각형이 지워지지 않도록 해보자.

- WM_LBUTTONDOWN 메시지 핸들러 코드를 OnDraw() 함수로 이동
- DC는 OnDraw() 함수의 인자에 포함된 것을 사용
- 출력위치는 WM_LBUTTONDOWN 메시지 핸들러의 인자로 넘어온 좌표를 OnDraw()에서 사용 → 멤버변수로 선언

MyView.h

```
class CMyView : public CView
{
private:
        CPoint  m_point; //마우스 클릭위치 저장
}
```

MyView.cpp

```
CMyView::CMyView( )
{
        m_point = CPoint( -100, -100 );  //화면에서 보이지 않도록 초기화
}
void CMyView::OnDraw( CDC* pDC )
{
        CMyDoc* pDoc = GetDocument( );
        ASSERT_VALID( pDoc );

        CRect rt;
        rt.left = m_point.x - 50;
        rt.top = m_point.y - 50;
        rt.right = m_point.x + 50;
        rt.bottom = m_point.y + 50;
```

```
        CPen pen, *oldPen;
        pen.CreatePen( PS_SOLID, 5, RGB(0,0,255) );
        oldPen = pDC->SelectObject( &pen );

        CBrush br, *oldBr;
        br.CreateSolidBrush( RGB(255,0,0) );
        oldBr = pDC->SelectObject( &br );

        pDC->Rectangle( rt );

        pDC->SelectObject( oldPen );
        pDC->SelectObject(o ldBr );

        pen.DeleteObject();
        br.DeleteObject();
}
void CMyView::OnLButtonDown( UINT nFlags, CPoint point )
{
        m_point = point;  // 현재 마우스 위치 저장
        CView::OnLButtonDown( nFlags, point );
}
```

- 프로그램 실행 후 왼쪽 버튼을 눌러도 반응 없음
  → 명시적으로 WM_PAINT 메시지를 발생시키는 **UpdateWindow()** 함수 호출

MyView.cpp

```
void CMyView::OnLButtonDown( UINT nFlags, CPoint point )
{
        m_point = point;   //현재 마우스 위치 저장
        UpdateWindow( );  //명시적으로 WM_PAINT 메시지 발생
        CView::OnLButtonDown( nFlags, point );
}
```

이전에 그려졌던 것과 비교해서 무효한 영역이 있을 때만 새로 그림

무효화 영역을 임의로 만들기 위해서는 CWnd클래스의 Invalidate() 또는 InvalidateRect()함수를 사용

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

MyView.cpp

```
void CMyView::OnLButtonDown( UINT nFlags, CPoint point )
{
        m_point = point;   //현재 마우스 위치 저장
        Invalidate();      //윈도우 영역을 무효화 시킴
        UpdateWindow( );  //명시적으로 WM_PAINT 메시지 발생 (생략 가능)
        CView::OnLButtonDown( nFlags, point );
}
```

참고

- 클릭한 모든 마우스 좌표에 대한 사각형을 다시 그리고자 한다면?

집합 클래스 (Collection Class) 사용
: 배열(array), 리스트(list), 맵(map)

CArray	**CArray<CPoint, CPoint&> array;**
CByteArray	**CByteArray array;**

```
CArray<CPoint, CPoint&> array;
array.SetSize( 5 );

for( int i=0; i<5; i++ ) {
        CPoint pt( i, i*5 );
        array[i] = pt;
}
```

비트맵

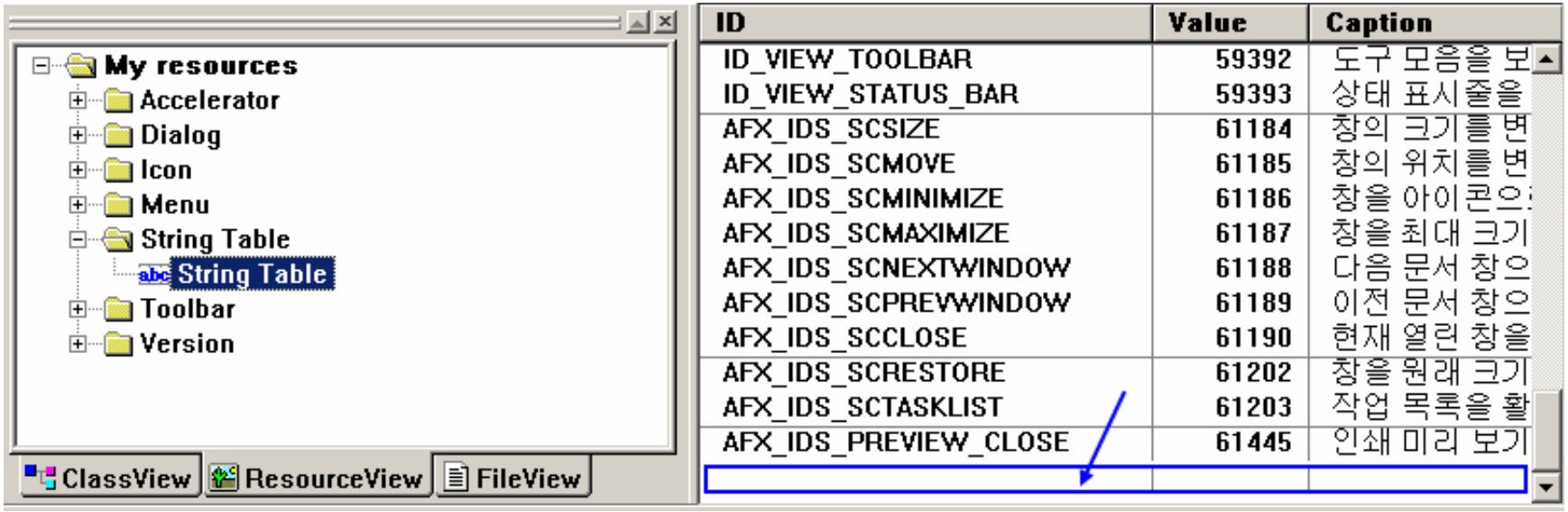
- 비트맵 객체는 픽셀단위로 표현된 영상을 다룸
- 리소스 편집기를 이용해 비트맵을 생성 및 출력

실습 8.3

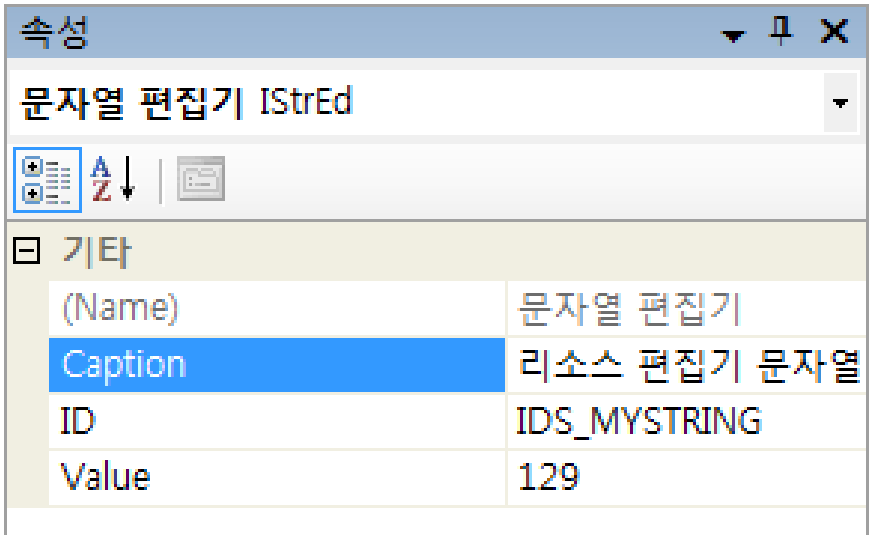
마우스 클릭 위치에 문자열을 출력하는 프로그램을 작성하자.
단, 출력할 문자열은 리소스로 등록한 것을 사용하고 도큐먼트-뷰 구조에 충실하게 작성한다.

도큐먼트-뷰 구조에 따른 코딩

- 리소스 편집기에서 문자열 리소스 추가
  - 리소스뷰 탭에서 String Table을 선택
  - 문자열 테이블의 마지막 빈 칸을 클릭



- 속성창
  - 문자열의 아이디 “IDS_MYSTRING” 입력
  - Caption “리소스 편집기 문자열입니다” 입력



- 도큐먼트-뷰 구조
  - 도큐먼트 – 데이터 처리
  - 뷰 – 출력 담당

도큐먼트 클래스에 리소스에 있는 문자열을 저장하기 위한 변수 선언

MyView.h

```
class CMyDoc : public CDocument
{
public:
        CString  m_strMsg;
}
```

리소스의 문자열을 읽어오는 함수

CString::LoadString(UINT *nID*);

도큐먼트 – 데이터 처리

사용할 초기 데이터의 로딩은 도큐먼트가 초기화될 때 하는 것이 적절 →
CMyDoc::OnNewDocument( ) 함수에서 처리

MyDoc.cpp

```
BOOL CMyDoc::OnNewDocument( )
{
        if ( !CDocument::OnNewDocument() )
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        m_strMsg.LoadString( IDS_MYSTRING );  // 리소스에서 문자열을 읽어옴

        return TRUE;
}
```

뷰 - 출력

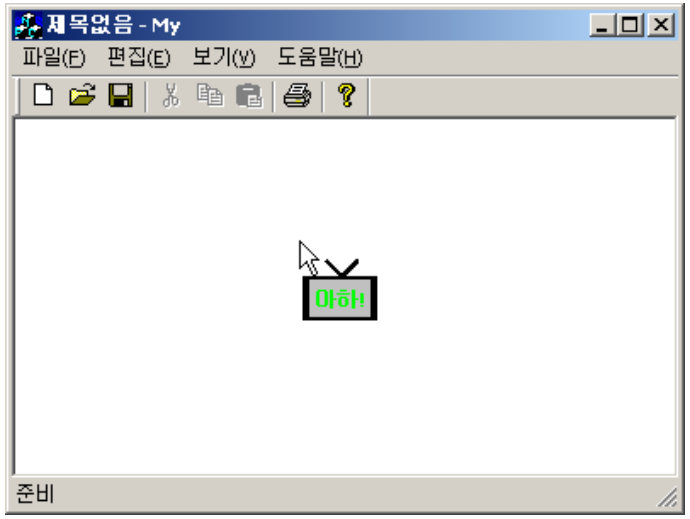
마우스 왼쪽 버튼을 누르면 출력되도록 WM_LBUTTONDOWN 메시지 핸들러에 코드 추가

GetDocument() – 도큐먼트 객체 주소 획득

MyView.cpp

```
void CMyView::OnLButtonDown( UINT nFlags, CPoint point )
{
        CString strMsg;
        CMyDoc* pDoc;
        pDoc = GetDocument( );
        strMsg = pDoc->m_strMsg;

        CClientDC dc( this );
        dc.TextOut( point.x, point.y, strMsg, strMsg.GetLength() );

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

실습 8.4

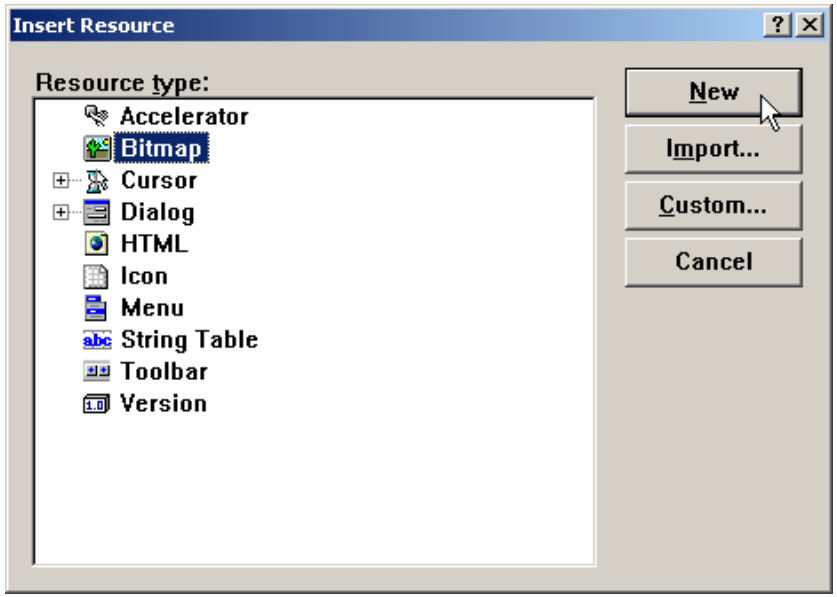


마우스 클릭 위치에 다음 그림과 같은 영상을 출력해보자.

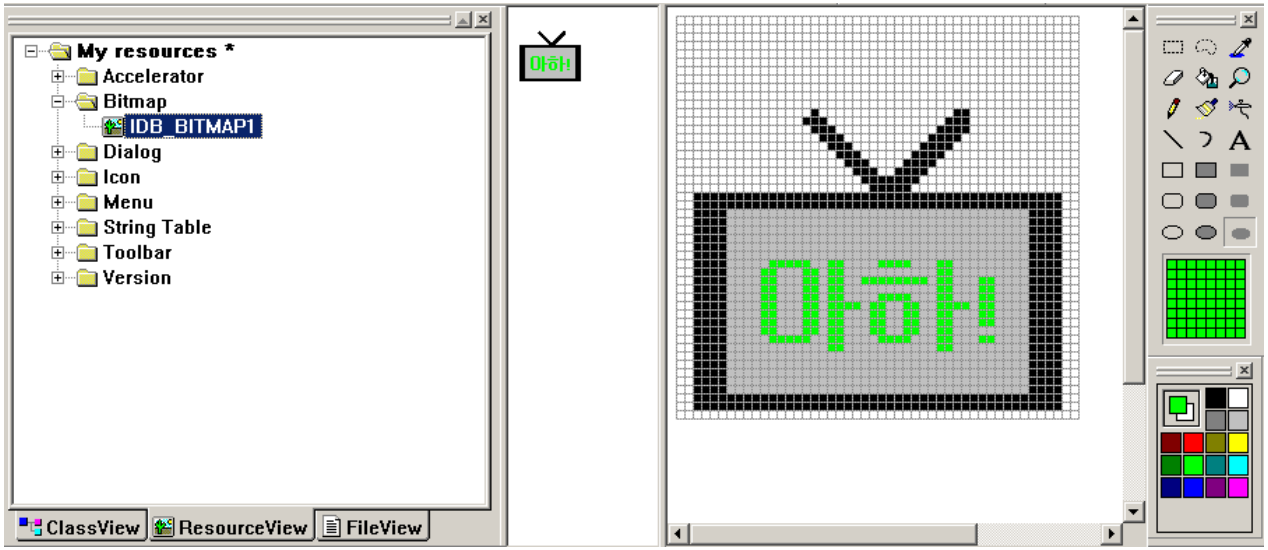
비트맵 리소스 편집

비트맵도 문자열과 같이 하나의 **리소스** – 리소스 편집기 사용

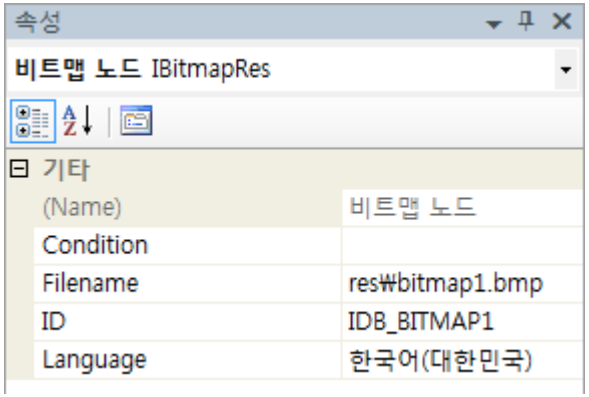
- 리소스 추가 대화상자
  - 리소스뷰 탭의 My.rc에서 마우스 오른쪽 버튼을 눌러 컨텍스트 메뉴를 활성화
  - 리소스 추가 (Insert..) 항목 클릭



- 비트맵을 선택하고 New 버튼클릭 → 비트맵 편집창



- 편집내용은 IDB_BITMAP1로 설정



비트맵 클래스를 통해 비트맵 리소스를 읽어옴

BOOL LoadBitmap(UINT *nIDResource*)

비트맵 영상은 화면에 보이지 않게 일반 메모리에 먼저 옮겨진 후, 화면 메모리로 한꺼번에 복사하여 빠르게 출력

메모리로 비트맵을 선택하는 방법은 메모리 DC 이용

```
CBitmap bmp;
bmp.LoadBitmap( IDB_BITMAP1 );

CDC* pDC;  // 화면 DC
pDC = GetDC( );

CDC memDC;  // 메모리 DC
memDC.CreateCompatibleDC( pDC );   //화면 DC와 호환이 되도록 생성
memDC.SelectObject( &bmp );
```

메모리 DC의 형식은 화면 DC와 호환되도록 CreateCompatibleDC() 함수를 이용해 생성

비트맵 관련정보 – BITMAP 구조체에 저장

```
typedef struct tagBITMAP {
  LONG        bmType;        //0으로 설정
  LONG        bmWidth;       //비트맵의 픽셀단위 가로크기
  LONG        bmHeight       //비트맵의 픽셀단위 세로크기
  LONG        bmWidthBytes;  //비트맵의 바이트단위 가로크기 (짝수)
  WORD        bmPlanes;      //컬러 판의 수
  WORD        bmBitsPixel;   //필셀 당 할당된 비트 수
  LPVOID      bmBits;        //영상이 저장된 시작 주소
} BITMAP;
```

BITMAP 정보는 비트맵 클래스의 GetObject()함수를 통해 얻어 올 수 있다

```
CBitmap bmp;
bmp.LoadBitmap( IDB_BITMAP1 );

BITMAP bm;
bmp.GetObject( sizeof(bm), &bm );        //비트맵 영상 정보 얻기

int width, height;
width = bm.bmWidth;
height = bm.bmHeight;
```

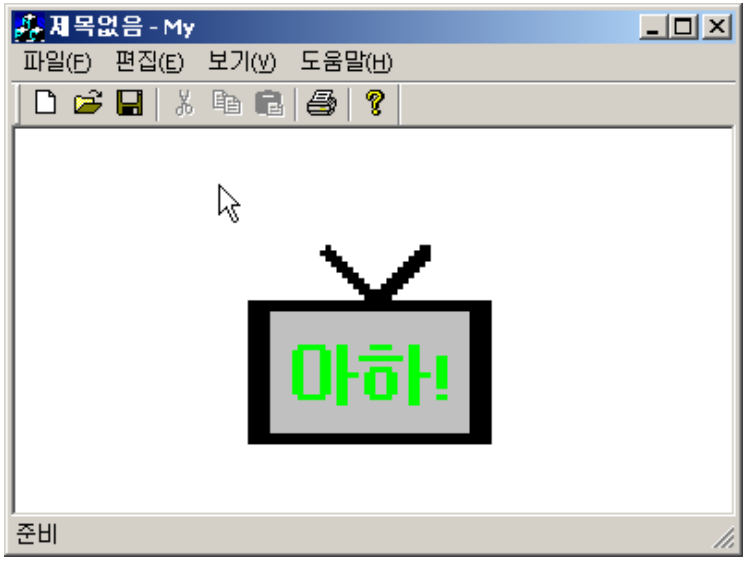
메모리 DC의 내용을 화면 DC로 복사 – BitBlt()

MyView.cpp

```
void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
	CBitmap bmp;
	bmp.LoadBitmap( IDB_BITMAP1 );

	CDC* pDC;
	CDC memDC;

	pDC = GetDC( );

	memDC.CreateCompatibleDC( pDC );  // 복사를 위해 화면 DC와 호환되도록 생성
	memDC.SelectObject( &bmp );// 메모리에 그림

	BITMAP bm;
	bmp.GetObject( sizeof(bm), &bm );  // 비트맵 관련 정보 획득
```

```
        pDC->BitBlt(
                point.x,            //출력 가로위치
                point.y,            //출력 세로위치
                bm.bmWidth,         //비트맵 가로크기
                bm.bmHeight,        //비트맵 세로크기
                &memDC,             //원본(메모리) DC의 주소
                0,                  //복사 시작할 비트맵의 가로 좌표
                0,                  //복사 시작할 비트맵의 세로 좌표
                SRCCOPY             //원본을 그대로 복사
        );

        memDC.DeleteDC( );  // 메모리 DC 해제
        ReleaseDC( pDC );  // 화면 DC 해제

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

실습 8.5



비트맵을 다시 작성하지 않고 비트맵을 확대해 보자.

- **비트맵 크기 변경 출력**

비트맵 크기 변화는 메모리 DC의 내용을 화면 DC로 복사하는 과정에서 CDC 클래스의 StretchBlt()함수를 통해 조정 가능

```
BOOL StretchBlt(
                    int x,          // 출력 가로위치
                    int y,          // 출력 세로위치
                    int nWidth,     // 출력 비트맵 가로크기
                    int nHeight,    // 출력 비트맵 세로크기
                    CDC* pSrcDC,    // 원본(메모리) DC의 주소
                    int xSrc,       // 복사 시작할 비트맵의 가로좌표
                    int ySrc,       // 복사 시작할 비트맵의 세로좌표
                    int nSrcWidth,  // 원본 비트맵 가로크기
                    int nSrcHeight, // 원본 비트맵 세로크기
                    DWORD dwRop     // 그리기 모드
        );
```

메모리 DC의 내용을 확대하여 화면 DC로 복사 – StretchBlt()

MyView.cpp

```
void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
	CBitmap bmp;
	bmp.LoadBitmap( IDB_BITMAP1 );

	CDC* pDC;
	CDC memDC;

	pDC = GetDC( );

	memDC.CreateCompatibleDC( pDC );  // 복사를 위해 화면 DC와 호환되도록 생성
	memDC.SelectObject( &bmp );// 메모리에 그림

	BITMAP bm;
	bmp.GetObject( sizeof(bm), &bm );  // 비트맵 관련 정보 획득
```

```
        pDC->StretchBlt(
                point.x,            //출력 가로위치
                point.y,            //출력 세로위치
                bm.bmWidth*3,       //출력 비트맵 가로크기
                bm.bmHeight*3,      //출력 비트맵 세로크기
                &memDC,             //원본(메모리) DC의 주소
                0,                  //복사 시작할 비트맵의 가로 좌표
                0,                  //복사 시작할 비트맵의 세로 좌표
                bm.bmWidth,         //원본 비트맵 가로크기
                bm.bmHeight,        //원본 비트맵 세로크기
                SRCCOPY             //원본을 그대로 복사
        );

        memDC.DeleteDC( );  // 메모리 DC 해제
        ReleaseDC( pDC );  // 화면 DC 해제

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

화면 맵핑 모드

맵핑모드	논리단위	X출 증가 방향	Y축 증가 방향
MM_TEXT	픽셀	오른쪽	아래
MM_LOMETRIC	0.1 mm	오른쪽	위
MM_HIMETRIC	0.01 mm	오른쪽	위
MM_LOENGLISH	0.1 inch	오른쪽	위
MM_HIENGLISH	0.01 inch	오른쪽	위
MM_TWIPS	1 / 1440 inch	오른쪽	위
WW_ISOTROPIC	지정가능	선택가능	선택가능
MM_ANISOTROPIC	지정가능	선택가능	선택가능

- **MM_TEXT: 기본설정 모드**
  - **화면 좌측상단이 (0,0)이고 오른쪽으로 갈수록 x축 값이 증가하고 아래로 갈수록 y축 값이 증가**
- **MM_LOMETIC ~ MM_TWIPS**
  - **물리 장치의 종류에 관계없이 물리적 길이가 항상 일정하도록 설계**
- **MM_ISOTROPIC, MM_ANISOTROPIC**
  - **프로그래밍 가능한 매핑 모드**

- **MM_ANISOTROPIC을 사용한 방법 1**

```
CRect rect;
GetClientRect( &rect );

CDC* pDC;
pDC = GetDC( );
pDC->SetMapMode( MM_ANISOTROPIC );  //맵핑 모드 설정
// 논리단위 설정
pDC->SetWindowExt( 300, 300 );
// 물리단위 설정
pDC->SetViewportExt( rect.Width(), rect.Height() );

pDC->Ellipse( 0, 0, 300, 300 );
pDC->Rectangle( 50, 50, 250, 250 );

ReleaseDC( pDC );
```

- **MM_ANISOTROPIC을 사용한 방법 2**

```
CDC* pDC;
pDC = GetDC( );
pDC->SetMapMode( MM_ANISOTROPIC );  //맵핑 모드 설정

// 디스플레이 크기의 1/3 크기로 논리단위 설정
pDC->SetWindowExt( xDisplay/3, yDisplay/3 );

// 디스플레이 크기 그대로 물리단위 설정
pDC->SetViewportExt( xDisplay, yDisplay );
```

- **SetMapMode() – 맵핑모드 설정**
- **xDisplay 및 yDisplay – 디스플레이 해상도**
- **SetWindowExt() – 화면출력단위가 되는 논리단위 설정**
- **SetViewPortExt() – 물리단위 화면크기 설정**

디스플레이 해상도를 얻는 방법

```
xDisplay = pDC->GetDeviceCaps( HORZRES );      //픽셀단위의 가로크기
yDisplay = pDC->GetDeviceCaps( VERTRES );      //픽셀단위의 세로크기
```

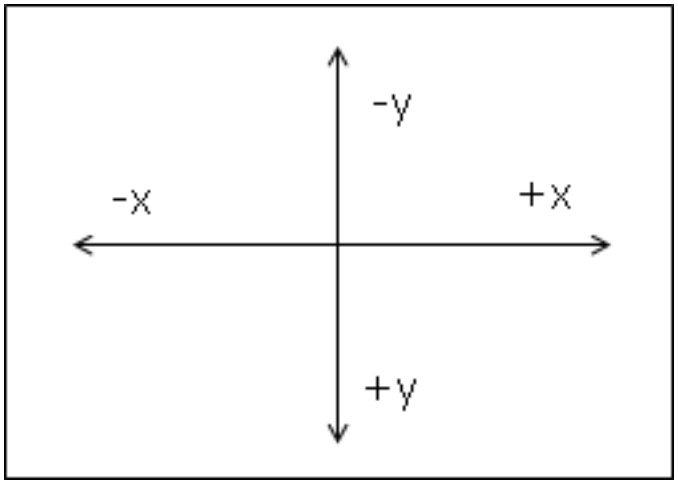
마우스를 클릭했을 때 인자로 넘어오는 마우스 좌표는 물리단위인 픽셀단위 → 물리단위를 논리단위로 변환하는 DPtoLP( )함수를 사용해 해결 (CDC 클래스의 멤버함수)

```
void DPtoLP(LPPOINT lpPoints, int nCount = 1) const; // 변환할 점이 여러 개인 경우
void DPtoLP(LPRECT lpRect) const;                    // RECT 좌표변환
void DPtoLP(LPSIZE lpSize) const;                    // SIZE 좌표변환

void LPtoDP(LPPOINT lpPoints, int nCount = 1) const;
void LPtoDP(LPRECT lpRect) const;
void LPtoDP(LPSIZE lpSize) const;
```

SetViewPortOrg(), SetWindowOrg() 함수를 이용하면 좌표계의 원점과 축 증가 방향 변경 가능

MM_TEXT 모드에서 다음 코드는 아래와 같이 화면의 중심이 원점이 되도록 변경

```
pDC->SetViewPortOrg( xDisplay/2, yDisplay/2 );
```



물리적인 원점 및 논리적인 원점 : 화면중심

논리적인 원점 설정을 추가 하여 중앙으로 변경된 물리적인 원점이 화면의 -1/2 : 논리적인 원점이 오른쪽 하단

```
pDC->SetViewPortOrg( xDisplay/2, yDisplay/2 );
pDC->SetWindowOrg( -xDisplay/2, -yDisplay/2 );
```