

2013년 윈도우 프로그래밍

# 제 2장 윈도우 기본 입출력

## ▶ 학습목표

- ▶ 윈도우 화면에 출력하기 위해 디바이스 컨텍스트 개념을 이해할 수 있다.
- ▶ 텍스트를 출력하는 기본 함수를 사용할 수 있다.
- ▶ 기본 도형을 화면에 출력할 때 필요한 요소와 함수를 사용할 수 있다.

## ▶ 내용

- ▶ 출력 영역 얻기
- ▶ 텍스트 출력하기
- ▶ 키보드 메시지 처리하기
- ▶ Caret 이용하기
- ▶ 직선, 원, 사각형, 다각형 그리기

# 헝가리언 표기법

## ▶ 헝가리언 표기법

- ▶ 변수명을 만들 때, 변수명 앞에 데이터형 접두어를 붙이는 것
- ▶ 변수가 무엇을 의미하고 어떤 데이터 타입을 갖는지 알 수 있다.
- ▶ Win32 API 프로그래밍 시 많은 프로그래머들이 이 방법을 즐겨 쓴다.

접두어 (prefix)	데이터 타입	접두어 (prefix)	데이터 타입
a	배열 (array)	i	인덱스 (index)
b	BOOLEAN	l	Long int
ch	문자 (character)	lp	Long pointer
cb	바이트개수(count of bytes)	n	Int
dw	Unsigned long (DWORD)	sz	NULL로 끝나는 문자열
h	핸들 (handle)	w	Unsigned int (WORD)

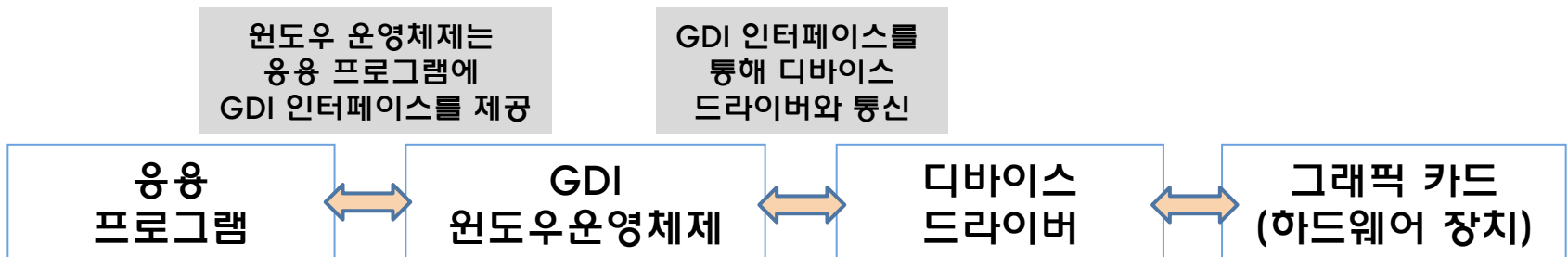
# 윈도우 메시지

메시지	내용	메시지	내용
WM_CREATE	윈도우가 생성될 때 발생	WM_RBUTTONDOWN	마우스 오른쪽 버튼을 누르면 발생
WM_ACTIVE	윈도우가 활성화될 때 또는 비활성화되면 발생	WM_RBUTTONUP	마우스 오른쪽 버튼을 눌렀다가 떴을 때 발생
WM_NCACTIVATE	윈도우의 비작업영역의 활성화 또는 비활성화시 발생 (윈도우 타이틀바 색상 제어)	WM_MOUSEMOVE	마우스가 움직이고 있으면 발생
WM_DESTROY	윈도우가 파괴되기 직전에 발생	WM_NCHITTEST	마우스가 움직이고 있으면 발생. 마우스의 아이콘을 제어하기 위해 사용
WM_PAINT	윈도우가 다시 그려져야 하면 발생	WM_SETCURSOR	마우스의 아이콘을 재설정해야 할 때 발생
WM_LBUTTONDOWN	마우스 왼쪽 버튼을 누르면 발생	WM_TIMER	타이머 설정 시 주기적으로 발생
WM_LBUTTONUP	마우스 오른쪽 버튼을 눌렀다가 떴을 때 발생	WM_COMMAND	메뉴, 버튼, 엑셀러레이터 선택 시 발생

# 1절. 출력 영역 얻기

## ▶ GDI (Graphic Device Interface)

- ▶ 윈도우 운영체제상에서 제공하는 응용 프로그램과 그래픽 장치 간의 인터페이스
  - ▶ 윈도우 내부에 설정되어 있는 그래픽 장치와 연결하여 제어하는 역할



- ▶ 디스플레이, 프린터, 기타 장치에 대한 그래픽 출력을 위하여 응용 프로그램이 사용할 수 있는 함수와 그에 관련된 구조를 제공
- ▶ GDI 객체에는 펜, 브러시, 폰트, 팔레트, 비트맵, 리전
  - ▶ 선 그리기, 칼라 처리 등 그래픽을 다루기 위한 함수의 모음

# 1절. 출력 영역 얻기

- ▶ 디바이스 컨텍스트(Device Context)
  - ▶ 그래픽 관련한 선택 정보를 모아 놓은 구조체
    - ▶ 간단한 출력은 디폴트 사양 이용
    - ▶ 세밀한 출력은 관련 선택정보(option) 변경
- ▶ 윈도우에서 출력 장치에 무언가 출력하기 위해서는 반드시 DC가 필요, DC 핸들을 얻은 후 출력한다.
  - ▶ 윈도우의 화면 메모리에 그리고 그것들을 윈도우 운영체제에서 출력시켜준다.
  - ▶ 이러한 화면 메모리를 제어하는 것이 DC
  - ▶ 모든 그래픽 출력에 있어서 각각의 윈도우는 모두 DC 핸들(HDC)을 얻어야 한다.
  - ▶ DC 핸들은 출력대상을 나타내는 구분번호로 생각
  - ▶ 모든 GDI 함수들은 첫 번째 인자로 DC 핸들을 필요로 한다.
- ▶ DC의 유형
  - ▶ 화면출력을 위한 디스플레이 DC
  - ▶ 프린터나 플로터 출력을 위한 프린터 DC
  - ▶ 비트맵 출력을 위한 메모리 DC
  - ▶ 디바이스 정보를 얻기 위한 정보 DC

# 디바이스 컨텍스트

## ▶ 핸들

- ▶ 프로그램에서 현재 사용중인 객체 (윈도우, 커서, 아이콘, 메뉴 등)들을 구분하기 위해 윈도우 OS가 부여하는 고유 번호
  - ▶ 어떤 대상에 붙여진 레이블 (LABEL): 대상이란 내가 조작할 타겟
  - ▶ 32비트 정수형
  - ▶ 핸들값은 접두어 h로 시작한다.
  - ▶ 핸들은 운영체제가 발급하며 사용자는 사용만 한다.

# 디바이스 컨텍스트

- ▶ DC를 얻고 해제하기
  - ▶ BeginPaint()와 EndPaint()
    - ▶ WM\_PAINT 메시지 부분에서 사용
  - ▶ GetDC()와 ReleaseDC()
    - ▶ 잠시 출력할 때 사용
  - ▶ CreateDC()와 DeleteDC()
    - ▶ DC를 만들어서 사용
    - ▶ 출력목적이 아니라 DC의 정보를 얻고자 할 때 사용



# 디바이스 컨텍스트

옵션	기능
선 그리기 (펜)	선을 그리거나 영역의 경계선을 그릴 때 사용 선의 색, 두께, 형태 등을 지정 디폴트는 검은색 1픽셀 실선
영역 채우기 (브러시)	어떤 영역의 내부를 채울 때 사용 채우기 색, 채우기 패턴 등을 지정 디폴트는 흰색
글꼴 (폰트)	문자를 출력할 때 사용하며 색, 모양, 크기 등을 지정 디폴트는 시스템 폰트
팔레트	화면에 출력할 수 있는 색에 제한을 받을 경우, 실제로 화면에 출력할 색의 수 등을 지정
리전	임의의 도형을 그리는 것과 관련된 옵션을 설정
비트맵	비트맵 그림 파일에 관한 옵션

# 디바이스 컨텍스트

- ▶ 윈도우의 크기가 변경되었을 때나 다른 윈도우에 가려져 있다가 드러날 때 등 화면에 출력된 결과가 깨질 수 있다.
- ▶ OS는 깨진 화면을 복구해주지 않는다.
- ▶ 단지 OS는 화면이 깨질 때 마다 WM\_PAINT메시지를 발생시켜준다.
- ▶ 그래서 출력은 반드시 WM\_PAINT메시지 아래에서 해야 한다!!
- ▶ 그래야 화면이 깨질 때 마다 WM\_PAINT메시지가 발생하고 그 아래에 작성한 소스가 다시 실행되어 화면이 복구된다!!

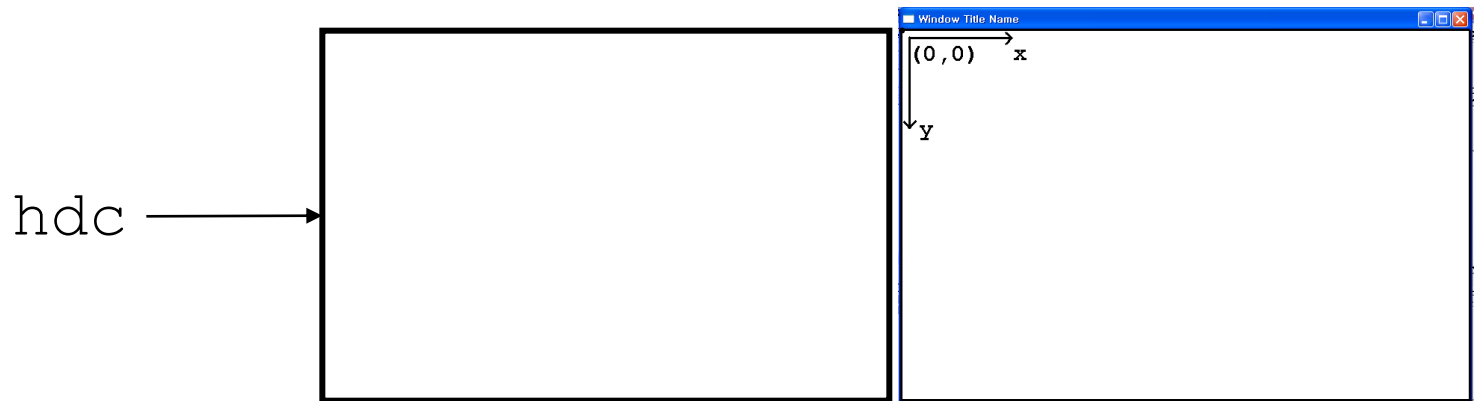
# 디바이스 컨텍스트 핸들

## ▶ HDC

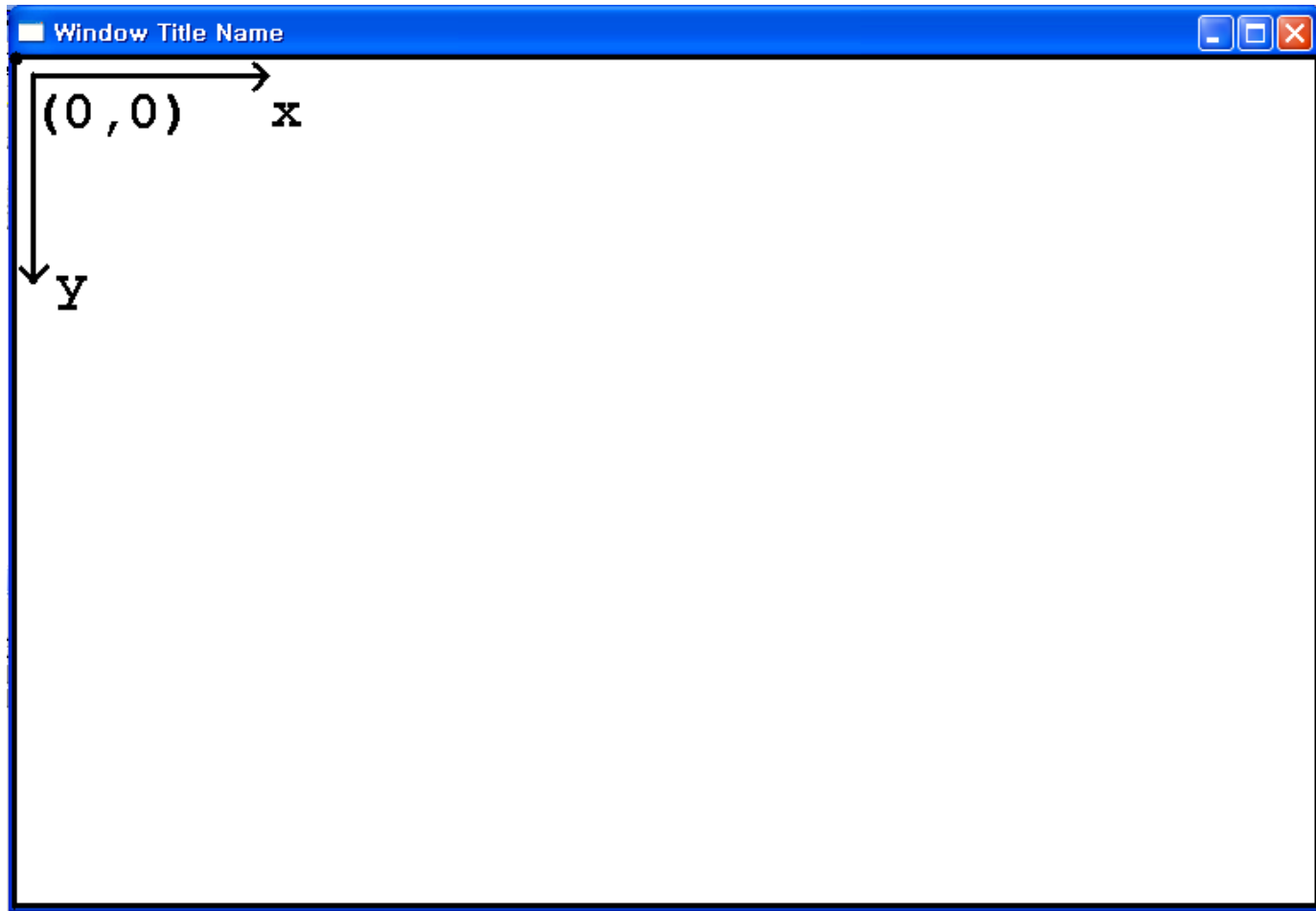
- ▶ 디바이스 컨텍스트 핸들이라고 부르고, 출력할 영역을 지정할 수 있는 타입
- ▶ 화면의 경우 윈도우로부터 얻어 옴

## ▶ HDC hdc;

- ▶ hdc 변수는 출력할 영역을 얻어오면 얻어온 영역을 지정할 수 있음



# 기본 좌표계



# 디바이스 컨텍스트 얻어오기

방법	기능
BeginPaint()와 EndPaint():	WM_PAINT메시지와 함께 사용
GetDC()와 ReleaseDC():	잠시 출력할 때 사용
CreateDC()와 DeleteDC():	DC를 만들어 사용
GetWindowDC()와 ReleaseDC():	비클라이언트 영역을 그리고자할 때 WM_NCPAINT메시지와 함께 사용
CreateIC()와 DeleteDC():	DC에 출력하지 않고 정보만 얻고자 할 때 사용
CreateCompatibleDC()와 DeleteDC():	이미 있는 DC와 같은 또 하나의 DC만들 때 사용. 보통 디스플레이를 이용한 메모리 DC를 만들 때 사용

# 디바이스 컨텍스트 얻어오기

- **BeginPaint()**  
HDC BeginPaint(  
    HWND hwnd;  
    PAINTSTRUCT \*lpPaint;  
);  
    hwnd: 생성된 윈도우의 핸들값  
    lpPaint : 출력될 영역에 대한 정보를 저장한 구조체 공간에 대한 주소
- **GetDC()**  
HDC GetDC(  
    HWND hwnd;  
);  
    hwnd: 생성된 윈도우의 핸들값

# 디바이스 컨텍스트 얻어오기

- PAINTSTRUCT 구조체

```
typedef struct tagPAINTSTRUCT {  
    HDC hdc;                //그리고자 하는 DC 핸들  
    BOOL fErase;            // 배경을 다시 그릴 지 구분  
                            // 0이 아니면 다시 그린다.  
    RECT rcPaint;           // 다시 그릴 사각형의 왼쪽 하단 좌표값  
    BOOL fRestore;  
    BOOL fIncUpdate;  
    BYTE rgbReserved[16]; }  
PAINTSTRUCT;
```

## 2-1 디바이스 컨텍스트 얻어오기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM  
    wParam, LPARAM lParam)  
{  
  
    HDC hdc;  
    PAINTSTRUCT ps;  
    switch (iMsg)  
    {  
        case WM_PAINT:  
            hdc = BeginPaint (hwnd, &ps) ;  
            // 이곳에서 출력이 이루어짐  
            EndPaint (hwnd, &ps) ;  
            break;  
        case WM_DESTROY:  
            PostQuitMessage ( 0 );  
            break;  
    }  
    return DefWindowProc (hwnd, iMsg, wParam, lParam);  
}
```



# WM\_PAINT 메시지 발생 경우

- 클라이언트 영역이 다시 그려져야 할 필요가 있을 때 윈도우즈 OS가 보내는 메시지로 대부분의 출력은 이 메시지 처리부에서 작성해야 한다.
  - 윈도우의 클라이언트 영역 중 일부가 무효화(invalid)되면 OS가 이 메시지를 큐에 넣어준다.
- 다음과 같은 경우에 OS는 WM\_PAINT 메시지를 프로그램에 전달한다.
  - 윈도우가 처음 생성되었을 때
  - 윈도우의 위치가 이동되었을 때
  - 윈도우의 크기가 변경되었을 때
  - 최대, 최소화되었을 때
  - 다른 윈도우에 가려져 있다가 드러날 때
  - 파일로부터 데이터를 출력할 때
  - 출력된 데이터의 일부분을 스크롤, 선택, 변화시킬 때
  - InvalidateRect(), InvalidateRgn() 함수를 호출하여 강제로 화면을 무효화시킬 때

화면의 그래픽 일부가 사라져서 다시 출력해야 할 필요가 있는 영역을 무효화(invalid) 영역이라 하고 이러한 경우를 화면이 무효화되었다고 한다.

# WM\_PAINT 메시지 발생 경우

- 이 메시지를 받았을 때 해당 프로그램은 화면 복구를 위해 클라이언트 영역 전체 또는 무효화된 부분만 다시 그려야 한다.
  - OS는 화면이 무효화될 때 클라이언트 영역을 복구해 주지 않는 대신에 이 메시지를 보내 줌으로써 해당 프로그램에게 다시 그려야 할 시점을 알려 준다.
  - 따라서 클라이언트 영역에 출력한 정보는 모두 저장해 두어야 복구가 가능하다.
- WM\_PAINT 메시지는 모든 메시지 중에서 우선 순위가 가장 낮다.
  - GetMessage() 함수는 메시지 큐에 WM\_PAINT 메시지가 있더라도 다른 메시지가 대기 중이면 그 메시지를 먼저 처리한다.
  - WM\_PAINT 메시지는 큐에 대기 중인 다른 메시지가 없고 무효화 영역이 존재할 때만 윈도우 프로시저로 보내진다.
- WM\_PAINT 메시지는 한번에 하나만 메시지 큐에 들어갈 수 있다.
  - 만약 무효화 영역이 생겼는데 WM\_PAINT 메시지가 이미 메시지 큐에 있으면 기존의 무효화 영역과 새 무효화 영역의 합으로 새로운 무효화 영역이 설정된다.

# WM\_PAINT 메시지 발생 경우

- 해당 윈도우 프로시저에서 이 메시지를 처리하지 않으면 이 메시지는 DefWindowProc() 함수가 처리한다.
  - 이 함수는 무효영역을 모두 유효화(valid)하며 다시 그리기는 하지 않는다.
- 만약 비 클라이언트 영역도 그려져야 한다면 WM\_NCPAINT 메시지를 전달하며, 배경을 지워야 한다면 WM\_ERASEBKGND 메시지를 전달한다.
- WM\_PAINT 메시지에서 그리기를 할 때는 BeginPaint() 와 EndPaint() 함수를 사용해야 한다.
  - 이 두 함수는 WM\_PAINT 메시지 내에서만 사용되며 다시 그려야 할 영역에 대한 정확한 좌표를 조사하며 무효영역을 유효화하고 캐럿을 숨기거나 배경을 지우는 등의 꼭 필요한 동작을 한다.

## 2절. 텍스트 출력하기

- 한 점 기준 텍스트 출력

BOOL **TextOut** (HDC hdc, int x, int y,  
LPCTSTR lpString, int nLength);

- HDC hdc: BeginPaint()나 GetDC()를 통해 얻어온 DC핸들
- int x, y: 텍스트를 출력할 좌표의 x값과 y값
- LPCTSTR lpString: 출력할 텍스트
- int nLength: 출력할 텍스트의 길이

## 2-2 윈도우에 “Hello World” 출력하기

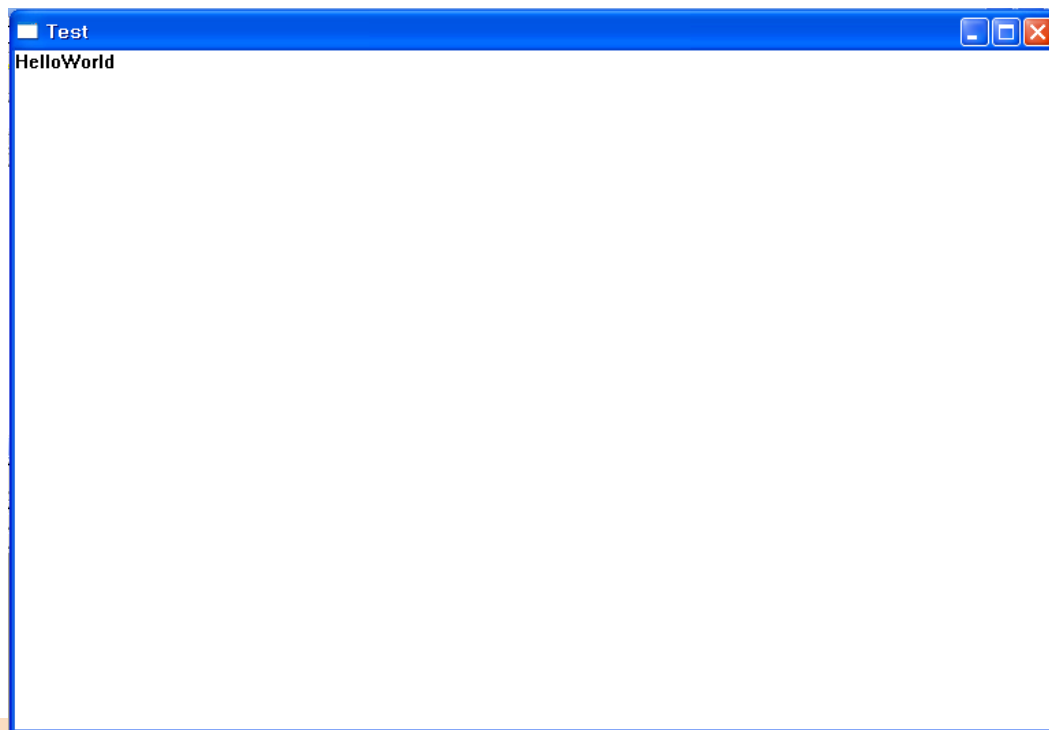
case WM\_PAINT:

hdc = BeginPaint (hwnd, &ps) ;

TextOut(hdc, 0,0, " HelloWorld" ,10);

EndPaint (hwnd, &ps) ;

break;



# 박스 영역에 텍스트 출력 함수

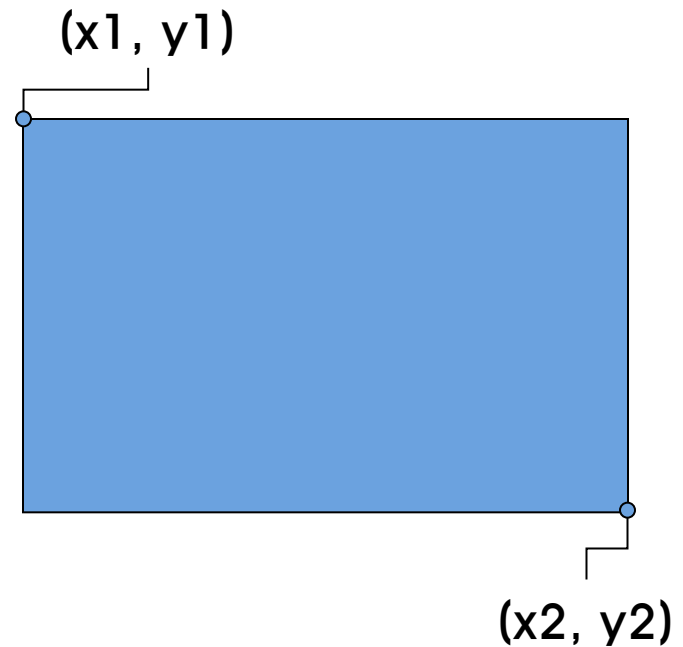
```
▶ int DrawText(  
    HDC hdc,  
    LPCSTR lpString,  
    int nLength,  
    LPRECT lpRect,  
    UINT Flags
```

```
);
```

- ▶ Hdc: BeginPaint()나 GetDC()를 통해 얻어온 DC핸들
- ▶ lpString: 출력 문자열
- ▶ nLength: 문자열 길이
- ▶ lpRect: 문자열을 출력할 박스영역 구조체의 주소 (RECT \*)
- ▶ Flags: 출력 방법

# RECT 구조체

```
typedef struct tagRECT {  
    LONG left;           // x1  
    LONG top;            // y1  
    LONG right;          // x2  
    LONG bottom;         // y2  
} RECT;
```



# DrawText() 출력 방법 Flag

- DT\_SINGLELINE: 박스 영역 안에 한 줄로 출력
- DT\_LEFT: 박스 영역 내에서 왼쪽 정렬
- DT\_CENTER: 박스 영역 내에서 가운데 정렬
- DT\_RIGHT: 박스 영역 내에서 오른쪽 정렬
- DT\_VCENTER: 박스 영역의 상하에서 가운데 출력
- DT\_TOP: 박스 영역의 상하에서 윗쪽에 출력
- DT\_BOTTOM: 박스 영역의 상하에서 아랫쪽에 출력
- DT\_CALCRECT: 문자열을 출력한다면 차지할 공간의 크기 측정

예:

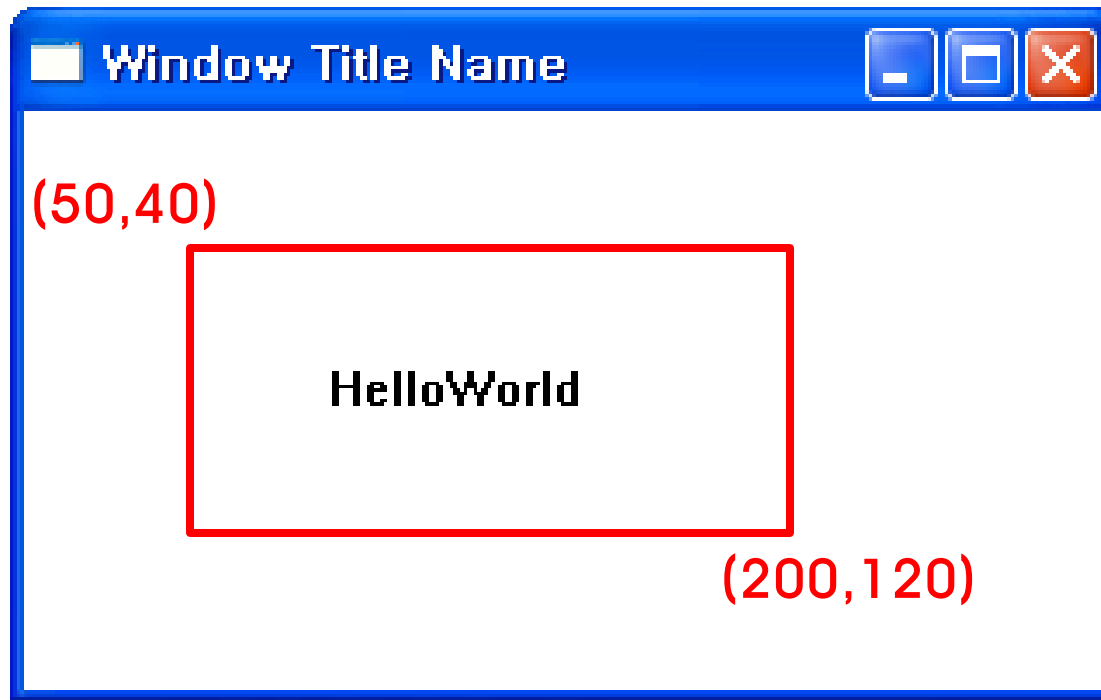
- DT\_TOP | DT\_CENTER | DT\_SINGLELINE
- 사각형 영역의 윗쪽 가운데에 한줄로 출력



## 2-3 DrawText() 함수 이용하기

```
RECT rect;
switch (iMsg)
{
    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        rect.left = 50; // 사각형 정의
        rect.top = 40;
        rect.right = 200;
        rect.bottom = 120;
        DrawText(hdc,"HelloWorld",10,&rect,
            DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        EndPaint (hwnd, &ps) ;
        break; // 한 라인, 수직/수평 중앙
```

## 2-3 DrawText() 함수 이용하기



# 문자 출력시 배경색, 전경색 모드 지정

- COLORREF SetBkColor (HDC hdc, COLORREF crColor);
  - hdc: 디바이스 컨텍스트 핸들
  - crColor: 배경색
- COLORREF SetTextColor (HDC hdc, COLORREF crColor);
  - hdc: 디바이스 컨텍스트 핸들
  - crColor: 문자색
- Int SetBkMode (HDC hdc, int iBkMode);
  - hdc: 디바이스 컨텍스트 핸들
  - iBkMode: 배경 모드 (OPAQUE/TRANSPARENT)
- 사용 예)  
    SetTextColor (hdc, RGB(255, 0, 0));  
    DrawText (hdc, "HelloWorld", 10, &rect,  
              DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

# 문자 출력시 배경색, 전경색 모드 지정

– 윈도우의 색 지정: RGB(Red, Green, Blue) 삼원색 사용

COLORREF color;

COLORREF: DWORD 형태의 타입

```
COLORREF RGB {  
    BYTE byRed,           // 색상 중 붉은 색  
    BYTE byGreen,         // 색상 중 초록 색  
    BYTE byBlue           // 색상 중 푸른 색  
}
```

사용 예)

COLORREF Color;

Color = RGB (255, 0, 0);

# 연습문제 2-1

## ▶ 제목

- ▶ 텍스트 출력하는 프로그램 작성하여 실행하기

## ▶ 내용

- ▶ 윈도우를 800 \* 600 크기로 띄운다.
- ▶ 윈도우를 띄우고 윈도우를 세로 4등분, 가로 5등분하여 각각의 등분에 해당 위치의 행렬 값을 출력한다. (TextOut 사용하기)

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)

## 연습문제 2-2

### ▶ 제목

- ▶ 텍스트 출력하는 프로그램 작성하여 실행하기

### ▶ 내용

- ▶ 연습문제 2-1의 문제를 DrawText() 함수를 이용하여 각 구간의 중앙에 위치 행렬값과 본인의 이름을 출력한다.

## 연습문제 2-3

### ▶ 문자의 색상과 배경 색상 바꾸기

- ▶ 각 칸에서 글씨의 색을 문자의 색상과 배경색상을 서로 보색으로 설정하여 출력한다.

## 연습문제 2-4

### ▶ 제목

#### ▶ 구구단 출력하기

### ▶ 내용

- ▶ 화면에 2단부터 9단까지의 구구단을 출력한다.
- ▶ DrawText 함수를 사용한다.

$$2 \times 1 = 2$$

$$2 \times 2 = 4$$

$$2 \times 3 = 6$$

$$2 \times 4 = 8$$

...

$$2 \times 9 = 18$$

$$3 \times 1 = 3$$

$$3 \times 2 = 6$$

$$3 \times 3 = 9$$

$$3 \times 4 = 12$$

...

$$3 \times 9 = 27$$

$$9 \times 1 = 9$$

$$9 \times 2 = 18$$

$$9 \times 3 = 27$$

$$9 \times 4 = 36$$

...

$$9 \times 9 = 81$$

### 3절. 키보드 메시지 처리하기



전달 변수	iMsg	wParam	lParam
내용	키보드 메시지	가상 키 값	부가 정보
값	<ul style="list-style-type: none"> <li>• WM_KEYDOWN</li> <li>• WM_CHAR</li> <li>• WM_KEYUP</li> </ul>	A, B, C, ... 1, 2, 3, ... !, @, #, ... VK_BACK VK_RETURN VK_LEFT ...	<ul style="list-style-type: none"> <li>• 스캔코드</li> <li>• 키 반복 횟수</li> <li>• 확장키 코드</li> <li>• 이전 키 상태</li> </ul>

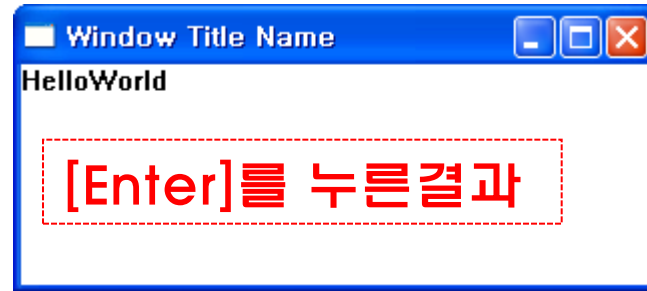
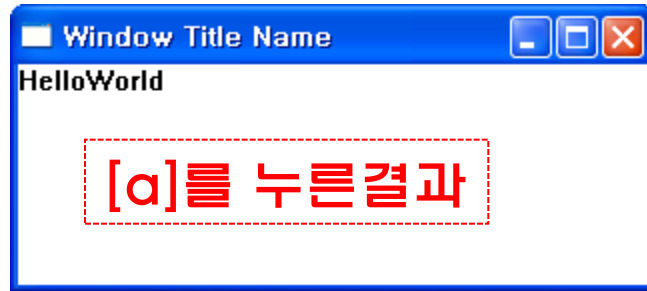


# 키보드 메시지 종류

- ▶ WM\_KEYDOWN
  - ▶ 키보드에서 키를 눌렀을 때 발생하는 메시지
- ▶ WM\_KEYUP
  - ▶ 키보드에서 키를 눌렀다가 떼어지면 발생하는 메시지
- ▶ WM\_CHAR
  - ▶ 문자 키를 눌렀을 때 발생하는 메시지
  - ▶ 'a' 키를 눌렀을 때: WM\_KEYDOWN 메시지 발생하고, 다음으로 WM\_CHAR 메시지 발생
- ▶ 윈도우 프로시저의 인수 값
  - ▶ wParam: 이벤트가 발생시킨 키의 가상키 값 (문자 혹은 특수 문자의 아스키 값)
  - ▶ lParam: 스캔 코드, 키 반복 횟수 같은 부가 정보

## 2-4 WM\_KEYDOWN 메시지 처리하기

```
case WM_KEYDOWN: // 키가 눌렸을 때
    hdc = GetDC(hwnd);
    TextOut(hdc, 0,0," HelloWorld" ,10);
    ReleaseDC(hwnd,hdc);
    break;
```



눌린 키에 관계없이 결과 출력

## 2-5 입력 문자 처리하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg,
                           WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    char str[100];

    switch (iMsg) {
    case WM_CHAR:
        hdc = GetDC(hwnd);
        str[0] = wParam;           // 입력문자 : : WinProc의 매개변수로 들어 올
        str[1] = '\0';             // 문자열은 null( '\0' )로 끝남
        TextOut(hdc,0,0,str,strlen(str));
        ReleaseDC(hwnd,hdc);
        break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

## 2-6 입력 문자열 처리하기

```
static char str[100];  
static int count;
```

```
case WM_CREATE :  
    count = 0;  
    return 0 ;  
case WM_CHAR:  
    hdc = GetDC(hwnd);  
    str[count++] = wParam;  
    str[count] = '\0';  
    TextOut(hdc,0,0,str,strlen(str));  
    ReleaseDC(hwnd,hdc);  
    break;
```

# 실습에서의 문제점

1. 다른 윈도우에 의해 가렸다가 나타나면 출력되었던 내용이 사라짐
  - ▶ 원인: WM\_PAINT 발생 -> 화면을 지워 버림
  - ▶ 해결방법: WM\_PAINT에 대한 case문을 만들어야 함
2. Control 문자 처리 불가
  - ▶ Enter, Backspace 등과 같은 문자 처리 불가
  - ▶ 가상키 사용하여 처리 필요

## 2-7 WM\_PAINT 메시지 처리하기

// 1차적으로 문자열을 출력

case WM\_CHAR:

hdc = GetDC(hwnd);

str[count++] = wParam;

str[count] = '\0';

TextOut(hdc, 0, 0, str, strlen(str)); // 중복

ReleaseDC(hwnd, hdc);

break;

// 화면이 가렸다 지워지면 다시 문자열을 출력

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

TextOut(hdc, 0, 0, str, strlen(str)); // 중복

EndPaint(hwnd, &ps);

break;

# WM\_PAINT 강제 발생 함수

▶ `BOOL InvalidateRgn(  
    HWND hWnd,  
    HRGN hRgn,  
    BOOL bErase  
);`

- ▶ `hWnd`: 수정될 영역이 포함된 윈도우의 핸들값
- ▶ `hRgn`: 수정될 영역에 대한 핸들값으로 (NULL이면 클라이언트 영역 전체를 수정)
- ▶ `bErase`: 수정될 영역을 모두 삭제하고 다시 그리게 될지 아니면 수정되는 부분만 추가할지를 나타내는 BOOL 값. (TRUE - 모두 삭제, FALSE - 삭제하지 않음)

▶ `BOOL InvalidateRect (  
    HWND hWnd,  
    const RECT* lpRect,  
    BOOL bErase  
);`

- ▶ `hRect`: 수정될 영역에 대한 영역 핸들 값
- ▶ `lpRect`: 영역 좌표 (NULL이면 전체 영역)
- ▶ `bErase`: `BeginPaint()`를 위해 플래그 (TRUE - 다음에 호출되는 `BeginPaint`에서 배경을 현재 브러시로 그리게 된다. FALSE - 배경 브러시에 상관없이 배경을 그리지 않는다.)

▶ `InvalidateRgn`이나 `InvalidateRect` 함수를 쓰면 윈도우의 업데이트를 하게된다

## 2-8 문자 저장과 출력 구분하기

case WM\_CHAR:

str[count++] = wParam; // 문자 저장

str[count] = '\0';

InvalidateRect(hwnd, NULL, FALSE); // WM\_PAINT 메시지 발생

break;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

TextOut(hdc, 0, 0, str, strlen(str)); // 문자 출력

EndPaint(hwnd, &ps);

break;



## 두 번째 문제 해결

- ▶ 화면에 표시되지 않는 제어문자는 가상키 테이블 이용

가상키	내용	가상키	내용
VK_CANCEL	Ctrl+Break	VK_END	End
VK_BACK	Backspace	VK_HOME	Home
VK_TAB	Tab	VK_LEFT	좌측 화살표
VK_RETURN	Enter	VK_UP	위쪽 화살표
VK_SHIFT	Shift	VK_RIGHT	우측 화살표
VK_CONTROL	Ct기	VK_DOWN	아래쪽 화살표
VK_MENU	Alt	VK_INSERT	Insert
VK_CAPITAL	Caps Lock	VK_DELETE	Delete
VK_ESCAPE	Esc	VK_F1 ~ VKF10	F1-F10
VK_SPACE	Space	VK_NUMLOCK	Num Lock
VK_PRIOR	Page Up	VK_SCROLL	Scroll Lock
VK_NEXT	Page Down		

## 2-9 Backspace 키 입력 처리하기

```
case WM_CHAR:  
    if (wParam == VK_BACK)  
        count--;  
    else  
        str[count++] = wParam;  
    str[count] = '\0';  
    InvalidateRect(hwnd, NULL, FALSE);  
    break;
```

## 2-10 엔터 키 입력 처리하기

```
static int count, yPos;
```

```
case WM_PAINT:
```

```
    hdc = BeginPaint(hwnd, &ps);
```

```
    TextOut(hdc, 0, yPos, str, strlen(str));
```

```
    EndPaint(hwnd, &ps);
```

```
    break;
```

```
case WM_CHAR:
```

```
    if (wParam == VK_BACK) count--;
```

```
    else if (wParam == VK_RETURN)
```

```
    {
```

```
        count = 0;
```

```
        yPos = yPos + 20;
```

```
    }
```

```
    else str[count++] = wParam;
```

## 4절. Caret(커서) 이용하기

▶ Caret: 키보드 입력 시 깜박거리는 커서

▶ Caret 만들고 보이기

```
CreateCaret(hwnd, NULL, width, height);  
ShowCaret(hwnd);
```

▶ Caret의 위치 설정하기

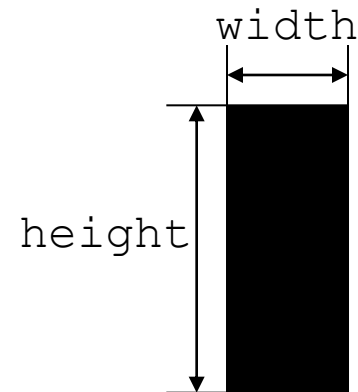
```
SetCaretPos(x,y);
```

▶ Caret 감추기

```
HideCaret(hwnd);
```

▶ Caret 삭제하기

```
DestroyCaret();
```



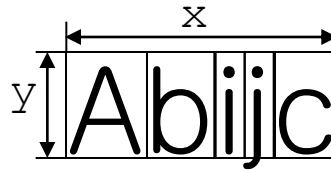
# Caret 위치 정하기

- ▶ 문자열 “Abijc” 를 굴림체로 출력하고 ‘c’ 뒤에 caret 위치를 정한다고 가정

문자열을 저장하고 있는 문자 배열

A	b	i	j	c
---	---	---	---	---

화면상에 출력



- ▶ X의 길이를 알아야 caret 위치 정함
- ▶ 예를 들어 문자열 출력 위치가 (100,200)이라고 하면 caret의 위치는 (100+x, 200)이다.

# 출력될 문자열 폭 구하기

- ▶ `BOOL GetTextExtentPoint(`  
    `HDC hdc,`  
    `LPCTSTR lpString,       // 출력될 문자`  
    `int cbString,           // 몇 번째 문자 뒤에 커서 출력`  
    `LPSIZE lpSize   // 문자열의 크기(폭, 높이) -> 얻어 오는 값`  
);
- ▶ `LPSIZE SIZE *;`  
    `struct tagSIZE {       // 문자열의 폭과 높이 저장`  
        `LONG cx;`  
        `LONG cy;`  
    `} SIZE;`

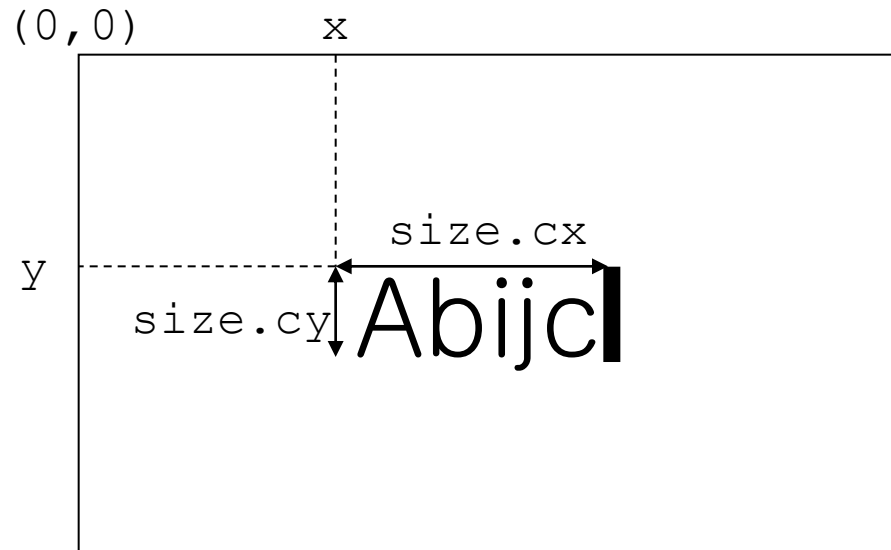
# “Abijc” 폭 구하기

```
SIZE size;
```

```
GetTextExtentPoint(hdc, "Abijc", 5, &size);
```

```
TextOut(hdc, x, y, " Abijc" , 5);
```

```
SetCaretPos(x + size.cx, y); // x좌표에 출력문자열 길이 합산
```



## 2-11 Caret 표시

```
static SIZE size;
case WM_CREATE :
    CreateCaret(hwnd, NULL, 5, 15);
    ShowCaret(hwnd); // 빈 화면에 캐럿 표시
    count = 0;
    return 0 ;
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    GetTextExtentPoint(hdc, str, strlen(str), &size);
    TextOut(hdc,0,0,str,strlen(str));
    SetCaretPos(size.cx, 0);
    EndPaint(hwnd, &ps);
    break;
case WM_DESTROY :
    HideCaret(hwnd);
    DestroyCaret();
    PostQuitMessage (0) ;
    return 0 ;
```



## 연습문제 2-5

### ▶ 제목

- ▶ Caret이 있는 10라인 메모장

### ▶ 내용

- ▶ Caret이 있는 10라인까지 입력 받을 수 있는 메모장을 작성
- ▶ 입력 받을 때 한 줄은 최대 99자 까지 저장 가능해야 하고 최대 10줄
- ▶ 엔터키: 다음 줄로 이동하여 작성 가능
- ▶ 백스페이스키: 한 줄에서 문자를 삭제한다.
- ▶ 특정키 (예, esc)를 누르면 화면이 다 지원진다.
- ▶ 또다른 특정키 (예, 함수키)를 누르면 저장했던 문자열을 화면에 출력한다.

## 연습문제 2-6

### ▶ 제목

- ▶ 키보드 입력 받으면 문자열 이동하기

### ▶ 내용

- ▶ 화면의 임의의 위치에 “키보드 입력 연습문제” 라는 문자열을 출력한다.
- ▶ 키보드 입력에 따라 위의 문자열을 좌 / 우 / 상 / 하 로 이동하여 출력하도록 한다.
- ▶ 화면의 가장자리에 도달하면 더 이상 이동하지 않는다.
- ▶ 문자열이 입력되면 이전에 입력되어 출력된 문자열을 삭제되고 새로운 위치에 문자열이 출력되어야 한다.

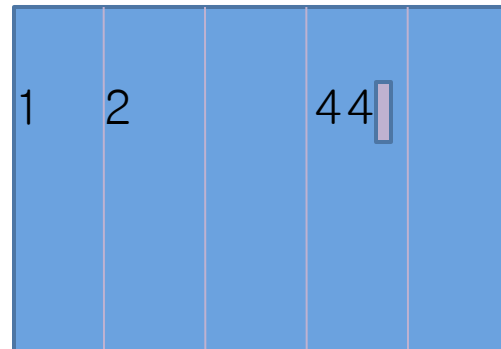
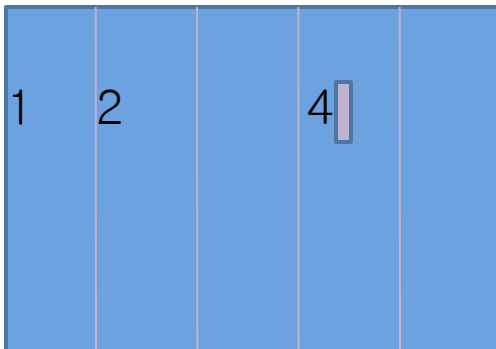
## 연습문제 2-7

### ▶ 제목

- ▶ Caret이 있는 숫자 쓰기

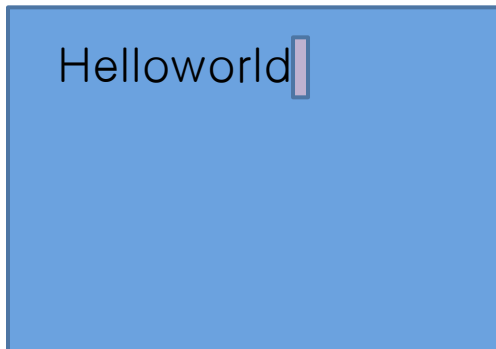
### ▶ 내용

- ▶ 윈도우를 띄우고, 가로로 5등분하고 각 등분에 1, 2, 3, 4, 5를 쓸 수 있게 한다.
- ▶ 첫 번째 Caret은 첫 번째 등분에 있고, 사용자가 1~5까지의 숫자를 입력하면 해당 등분에 해당 위치에 숫자를 쓰고, 그 뒤에 Caret이 있다.
- ▶ 각 등분의 끝에 도달하면 다음 줄에 숫자가 계속 쓰여진다. (옵션)
- ▶ 6~0은 %5의 숫자에 해당되는 부분에 적도록 한다.

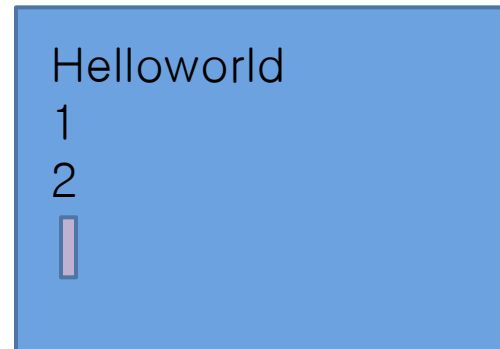


## 연습문제 2-8

- 제목
  - 숫자와 문자 구별하여 출력하기
- 내용
  - 윈도우를 띄우고, 사용자가 문자 또는 숫자를 입력받는다.
  - 문자를 입력받으면 caret이 같은 줄에 놓이게 되고,
  - 숫자를 입력받으면 caret이 다음 줄에 놓이게 된다.
  - Q/q를 입력받으면 프로그램을 종료한다.



Helloworld



Helloworld  
1  
2

## 5절. 직선, 원, 사각형, 다각형 그리기

1. 직선 그리기
2. 원 그리기
3. 사각형 그리기
4. 다각형 그리기
5. 선 속성 바꾸기
6. 면 색 바꾸기

# 직선 그리기

## 1. 직선의 시작점으로 이동하기

```
BOOL MoveToEx(  
    HDC hdc,  
    int X1,  
    int Y1,  
    LPPOINT lpPoint // 이전의 좌표, 사용 안함  
);
```

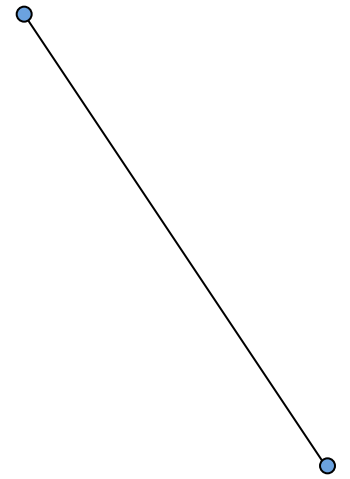
(X1, Y1)



## 2. 직선의 종착점까지 직선 그리기

```
BOOL LineTo(  
    HDC hdc,  
    int X2,  
    int Y2  
);
```

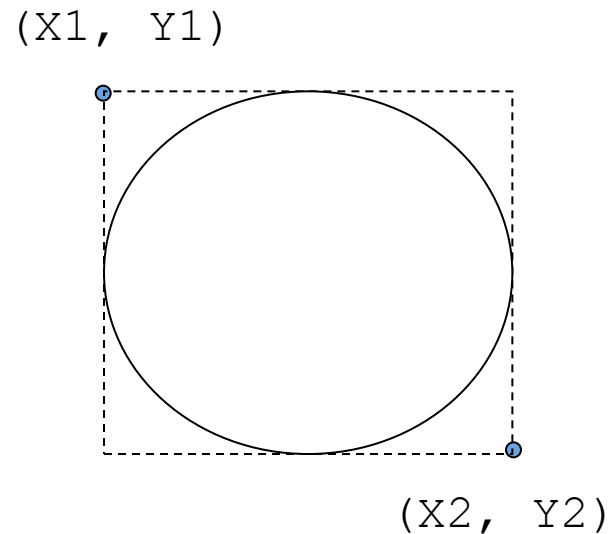
(X2, Y2)



# 원 그리기

▶ 두 점의 좌표를 기준으로 만들어진 가상의 사각형에 내접하는 원을 그림

```
▶ BOOL Ellipse(  
    HDC hdc,  
    int X1,    // left  
    int Y1,    // top  
    int X2,    // right  
    int Y2     // bottom  
);
```



## 2-12 원 그리기

case WM\_PAINT :

hdc = BeginPaint (hwnd, &ps) ;

Ellipse(hdc, 0, 0, 40, 40); // 박스의 좌표, 중심점 (20,20)

EndPaint (hwnd, &ps) ;

return 0 ;

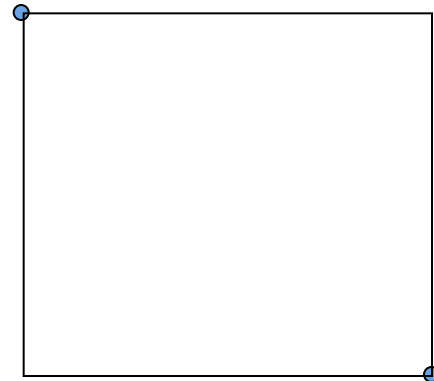


# 사각형 그리기

▶ 두 점의 좌표를 기준으로 수평수직 사각형을 그림

```
▶ BOOL Rectangle(  
    HDC hdc,  
    int X1,    // left  
    int Y1,    // top  
    int X2,    // right  
    int Y2     // bottom  
);
```

(X1, Y1)



(X2, Y2)

## 2-13 사각형 그리기

```
case WM_PAINT :  
    hdc = BeginPaint (hwnd, &ps) ;  
    Rectangle(hdc, 0, 0, 40, 40);  
    EndPaint (hwnd, &ps) ;  
    return 0 ;
```

### ▶ 사각형 그리기 다른 함수들

채워진 사각형 그리기:

```
int FillRect (HDC hdc, CONST RECT *lprc, HBRUSH hbr)
```

외곽선 사각형 그리기:

```
int FrameRect (HDC hdc, CONST RECT *lprc, HBRUSH hbr);
```

## 2-13 사각형 그리기

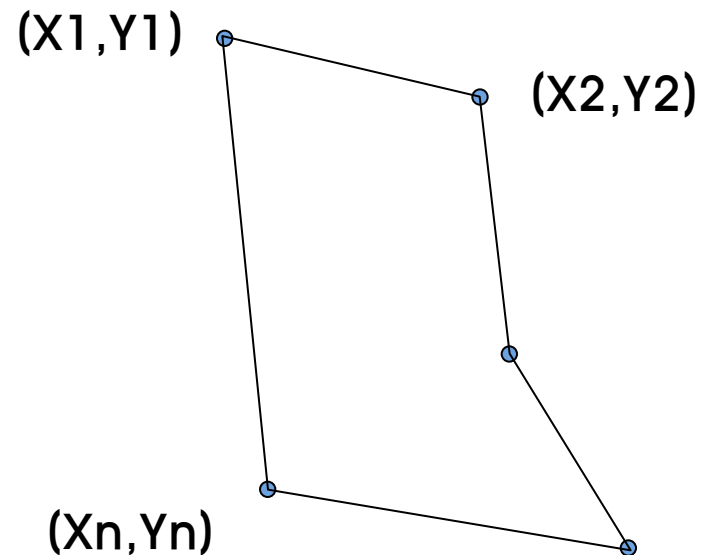
- ▶ **BOOL OffsetRect (LPRECT lprc, int dx, int dy);**
  - ▶ 주어진 Rect를 dx, dy만큼 이동한다.
- ▶ **BOOL InflateRect (LPRECT lprc, int dx, int dy);**
  - ▶ 주어진 Rect를 dx, dy만큼 늘이거나 줄인다.
- ▶ **BOOL IntersectRect (LPRECT lprcDst, CONST RECT \*lprcSrc1, CONST RECT \*lprcSrc2);**
  - ▶ 두 RECT가 교차되었는지 검사한다.
- ▶ **BOOL UnionRect (LPRECT lprcDest, CONST RECT \*lprcSrc1, CONST RECT \*lprcSrc2);**
  - ▶ 두 RECT를 union 시킨다.
- ▶ **BOOL PtInRect (CONST RECT \*lprc, POINT pt);**
  - ▶ 특정 좌표 pt가 lprc 영역 안에 있는지 검사한다.

# 다각형 그리기

- ▶ 연속되는 여러 점의 좌표를 직선으로 연결하여 다각형을 그림

```
BOOL Polygon(  
    HDC hdc,  
    CONST POINT *lppt,  
    int cPoints // 꼭지점의 수  
);
```

```
typedef struct tagPOINT {  
    LONG x;  
    LONG y;  
} POINT;
```

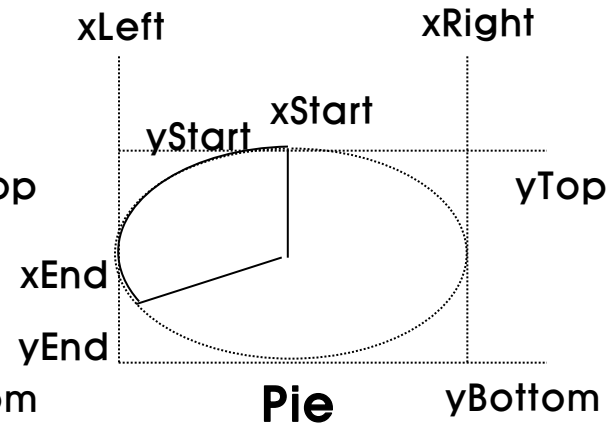
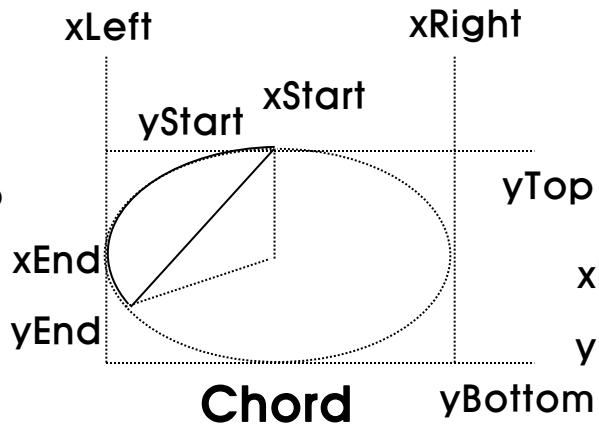
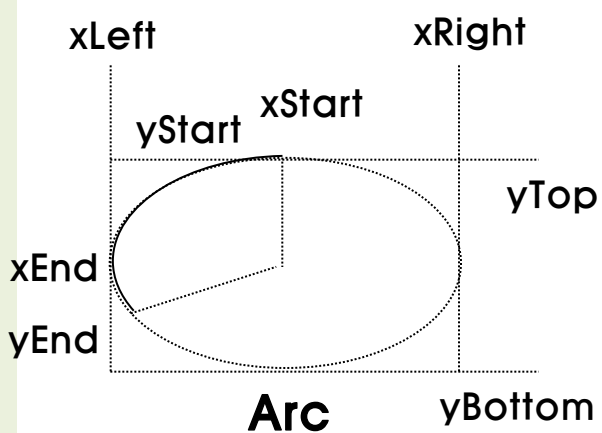
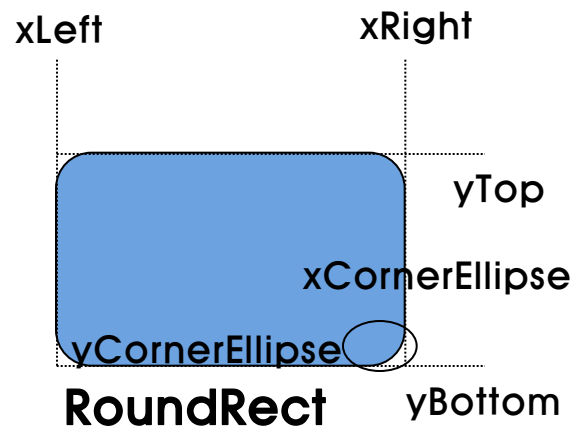
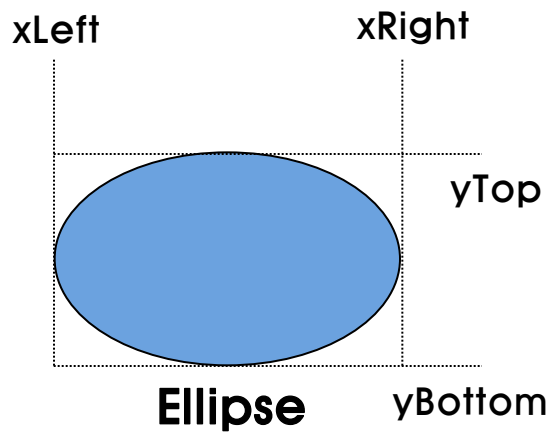
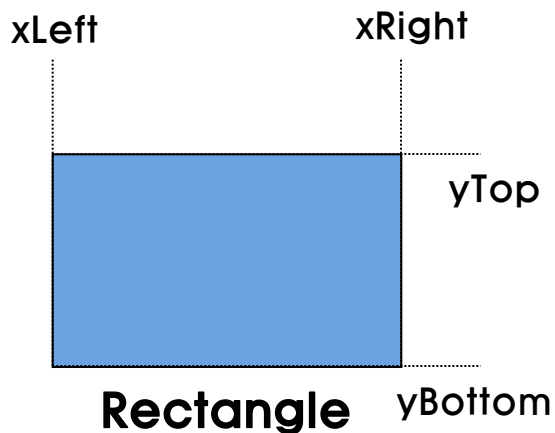


## 2-14 다각형 그리기

```
POINT point[10] = {{10,20}, {100,30}, {500,200},  
                  {600, 300}, {200, 300}};
```

```
case WM_PAINT :  
    hdc = BeginPaint (hwnd, &ps) ;  
    Polygon(hdc, point, 5); // 5각형  
    EndPaint (hwnd, &ps) ;  
    return 0 ;
```

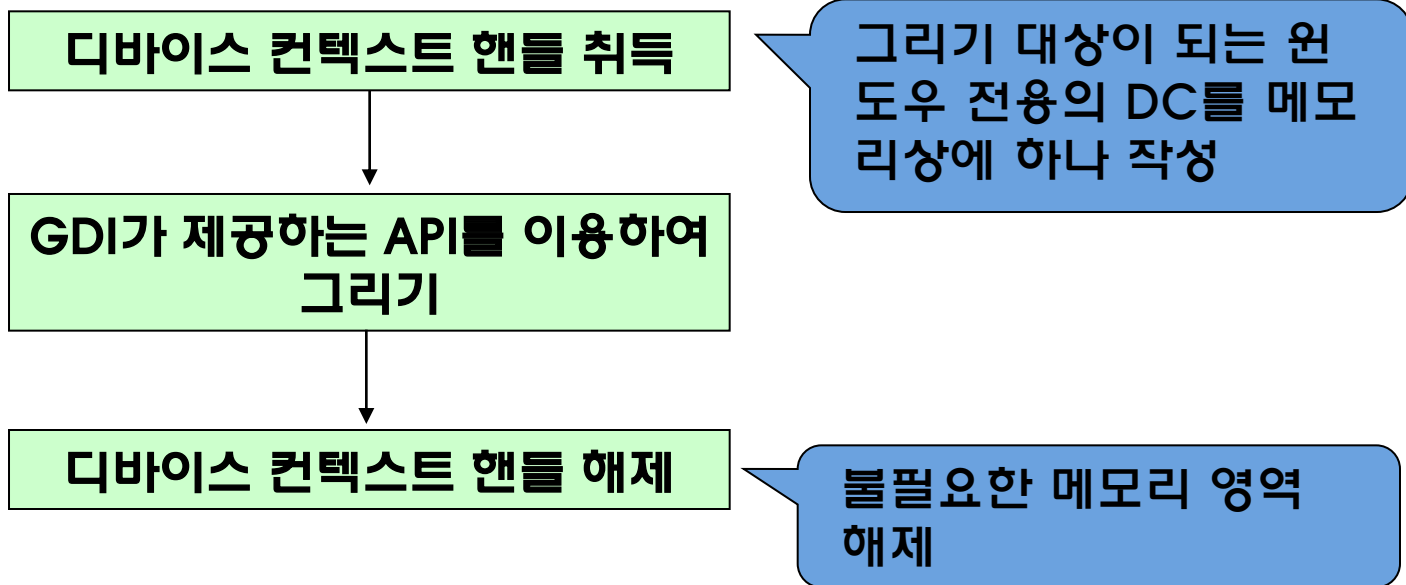
# 도형 그리기



# GDI 객체들

## ▶ GDI (Graphic Device Interface)

- ▶ 화면, 프린터 등의 모든 출력장치를 제어하는 윈도우 모듈
- ▶ GDI 오브젝트: 그림을 그리는 데 필요한 도구 (펜, 브러쉬 등)
- ▶ GDI를 사용한 그리기 순서



# 다양한 그래픽 출력 방법

- ▶ 윈도우즈에서 기본적으로 제공하는 GDI객체인 스톱 객체는 따로 생성할 필요가 없어서 편리하지만 다양하게 그래픽을 하기에는 부족
  - ▶ 다양한 출력을 위해서는 GDI객체를 만들어서 사용
  - ▶ GDI객체를 만드는 함수는 앞의 "Create"로 시작하는 함수



# GDI 객체들

GDI 오브젝트	핸들타입	의미	디폴트 값
펜	HPEN	선을 그을 때	검정색의 가는 실선
브러시	HBRUSH	면을 채울 때	흰색
폰트	HFONT	문자 출력 글꼴	시스템 글꼴
비트맵	HBITMAP	비트맵 이미지	X
팔레트	HPALETTE	팔레트	X
리전	HRGN	화면상의 영역	X

# 다양한 그래픽 출력 방법

## ▶ 일반적으로 다양한 그래픽을 하기 위한 절차

- ▶ **DC 생성**: 우선 `BeginPaint()`나 `GetDC()` 등의 함수를 이용하여 DC를 생성
- ▶ **GDI 객체 생성**: `GetStockObject()`로 스톡 객체나 `Create`로 시작하는 함수로 GDI객체를 생성
- ▶ **객체 선택**: 만든 GDI객체를 `SelectObject()`함수로 선택하고, 이 함수의 리턴값인 이전의 GDI객체를 그리기 작업을 다한 후 DC를 원상 복구할 목적으로 보관
- ▶ **설정**: `SetBkColor()`함수로 배경색을 설정하거나 `Set`으로 시작하는 함수들을 이용하여 다양한 설정
- ▶ **그리기**: GDI함수를 이용하여 그리기 작업
- ▶ **GDI 객체 복구**: 그리기 작업이 다 끝난 후에는 보관해놓았던 이전 GDI객체를 `SelectObject()`함수로 선택하여 이전 DC상태로 복구
- ▶ **생성 객체 삭제**: 생성한 GDI객체를 `DeleteObject()`함수로 삭제
- ▶ **DC 소멸**: 생성했던 DC를 `EndPaint()`나 `ReleaseDC()`함수로 소멸

# 다양한 그래픽 출력 방법



# GDI 객체 생성

## ▶ GDI객체를 만드는 함수

### ▶ 펜:

- ▶ CreatePen, CreatePenIndirect

### ▶ 브러시:

- ▶ CreateBrushIndirect, CreateDIBPatternBrush, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush

### ▶ 폰트(글꼴):

- ▶ CreateFont, CreateFontIndirect

### ▶ 리전:

- ▶ CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect

### ▶ 비트맵:

- ▶ CreateBitmap, CreateBitmapIndirect, CreateCompatibleBitmap, CreateDIBitmap, CreateDIBSection

# 펜 : 선 다루기

- ▶ 선의 굵기 정보와 색상정보를 가지는 펜 핸들을 생성

```
hPen = CreatePen(PS_DOT, 1, RGB(255,0,0));
```

```
HPEN CreatePen( int fnPenStyle, int nWidth,  
                COLORREF crColor);
```

- ▶ 그림을 그릴 화면인 DC에 펜 핸들을 등록

```
HPEN SelectObject (HDC hdc, HPEN pen);
```

- ▶ 그림 그리기를 마친 후 생성된 펜 핸들은 삭제

```
DeleteObject (HPEN pen);
```

# CreatePen

## HPEN CreatePen( int fnPenStyle, int nWidth, COLORREF crColor);

## ▶ 첫 번째 인자: 펜 스타일

PS_SOLID	_____
PS_DASH	- - - - -
PS_DOT	. . . . .
PS_DASHDOT	- . - . - .
PS_DASHDOTDOT	- . - . - .

▶ 두 번째 인자: 펜의 굵기로 단위는 픽셀

- ▶ 세 번째 인자: 색상을 표현하기 위해 COLORREF 값을 제공하며, RGB()함수로 만듦

## COLORREF RGB(int Red,int Green,int Blue)

-> Red, Green, Blue에는 0~255 사이의 정수 값을 사용

## 2-15 빨간 점선으로 원 그리기

```
HPEN hPen, oldPen;
```

```
case WM_PAINT :
```

```
    hdc = BeginPaint (hwnd, &ps) ;
```

```
    hPen = CreatePen(PS_DOT, 1, RGB(255,0,0));
```

```
    oldPen = (HPEN)SelectObject(hdc, hPen); // 새로운 펜 사용 선언
```

```
    Ellipse(hdc, 20,20, 300,300); // 새로운 펜으로 원을 그림
```

```
    SelectObject(hdc, oldPen); // 이전의 펜으로 돌아감
```

```
    DeleteObject(hPen);
```

```
    EndPaint (hwnd, &ps) ;
```

```
    return 0 ;
```

# 면 색상 변경

- ▶ 면의 색상정보를 가지는 브러시 핸들을 만들어 준다  
`HBRUSH CreateSolidBrush(COLORREF crColor);`
- ▶ 그림그릴 화면인 디바이스 컨텍스트에 브러시 핸들을 등록  
`HBRUSH SelectObject(HDC hdc, HBRUSH brush);`
- ▶ 그림 그리기를 마친 후 생성된 브러시 핸들은 삭제한다  
`DeleteObject(HBRUSH);`



## 2-16 빨간면의 원 그리기

HBRUSH hBrush, oldBrush;

case WM\_PAINT :

hdc = BeginPaint (hwnd, &ps) ;

hBrush = CreateSolidBrush(RED);

oldBrush = (HBRUSH)SelectObject(hdc, hBrush);

Ellipse(hdc, 20,20, 300,300);

SelectObject(hdc, oldBrush);

DeleteObject(hBrush);

EndPaint (hwnd, &ps) ;

return 0 ;

# GDI 객체 핸들 구하기

- ▶ **HGDIOBJ SelectObject** (HDC hDC, HGDIOBJ hgdiobj);
  - ▶ 독자적으로 생성한 펜, 브러시 및 폰트를 설정한다.
  - ▶ 리턴 값은 원래의 오브젝트 값
- ▶ **HGDIOBJ GetStockObject** (int fnObject);
  - ▶ 윈도우가 제공하는 펜 브러시 및 폰트를 취득한다.
    - ▶ 윈도우에서 기본적으로 제공하는 GDI 객체를 스톡 객체(stock object)라고 한다.
    - ▶ 윈도우가 제공해주므로 따로 생성하지 않고 사용할 수 있으며 해제하지 않아도 됨
  - ▶ fnObject: BLACK\_BRUSH / DKGRAY\_BRUSH / DC\_BRUSH / GRAY\_BRUSH / HOLLOW\_BRUSH / LTGRAY\_BRUSH / NULL\_BRUSH / WHITE\_BRUSH / BLACK\_PEN / DC\_PEN / WHITE\_PEN
- ▶ **BOOL GetClientRect** (HWND hWnd, LPRECT lprc);
  - ▶ 클라이언트의 영역을 취득한다.

# GDI 객체 핸들하기

## ▶ 윈도우가 제공하는 회색 브러시를 사용하여 사각형을 그리는 경우

```
HBRUSH MyBrush, OldBrush;
```

```
MyBrush = (HBRUSH) GetStockObject (GRAY_BRUSH);
```

```
OldBrush = (HBRUSH) SelectObject (hdc, MyBrush);
```

```
Rectangle (hdc, 50, 50, 300, 200);
```

```
SelectObject (hdc, OldBrush);
```

## ▶ 파란색 색상의 테두리를 가진 사각형을 그리는 경우

```
HPEN MyPen, OldPen;
```

```
MyPen = CreatePen (PS_SOLID, 5, RGB(0, 0, 255));
```

```
OldPen = (HPEN) SelectObject (hdc, MyPen);
```

```
Rectangle (hdc, 50, 50, 300, 200);
```

```
SelectObject (hdc, OldPen);
```

```
DeleteObject (MyPen);
```

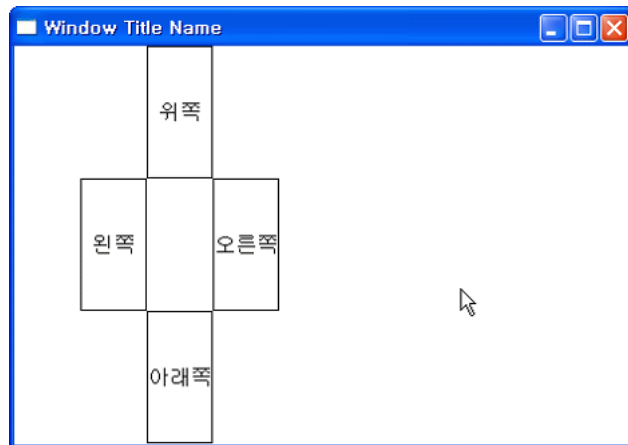
## 연습문제 2-9

### ▶ 제목

- ▶ 방향키가 눌러졌는지 체크하는 프로그램

### ▶ 내용

- ▶ 윈도우화면에 방향화살표키 4개에 대한 사각형을 그려줌
- ▶ 키보드에서 방향화살표를 누르면 누르고 있는 동안 해당되는 사각형이 빨간면을 가진 사각형으로 변함
- ▶ 누르고 있던 키를 놓으면 사각형은 원래대로 돌아감



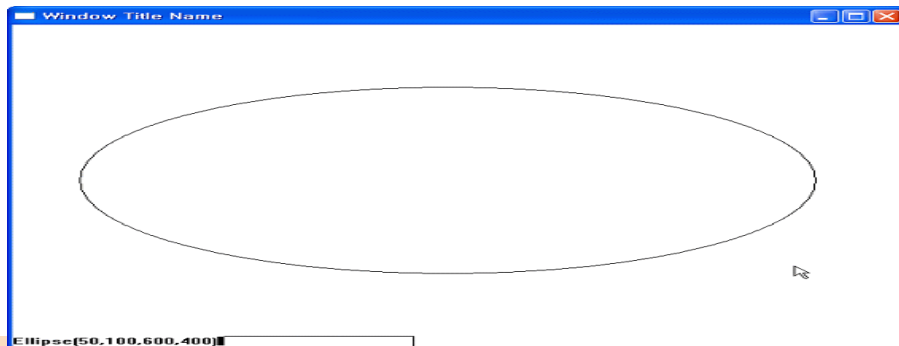
## 연습문제 2-10

### ▶ 제목

- ▶ 명령에 따라 그림을 그리는 프로그램

### ▶ 내용

- ▶ 윈도우 화면 아랫단 중앙에 문자열 한 줄 입력 받을 수 있는 글상자 역할을 할 사각형을 배치
- ▶ 사각형 내에는 Caret이 나타나서 명령어 입력을 기다림
- ▶ 명령어는 다섯개의 숫자로, 그릴 도형의 형태와 좌표값 4개가 입력된다.
- ▶ 예를 들어,
  - ▶ 직선일 경우는 [1 10 10 200 150]
  - ▶ 원일 경우는 [2 0 0 50 50]
  - ▶ 사각형일 경우는 [3 0 0 100 200]
  - ▶ 첫 번째 숫자는 도형의 종류, 2~5번째 숫자는 좌상우하 좌표값



- 1: Line
- 2: Ellipse
- 3: Rectangle

## 연습문제 2-11

## 연습문제 2-12