

# 제 2장 윈도우 기본 입출력

2019년 1학기 윈도우 프로그래밍

## 2장 학습 목표

### • 학습목표

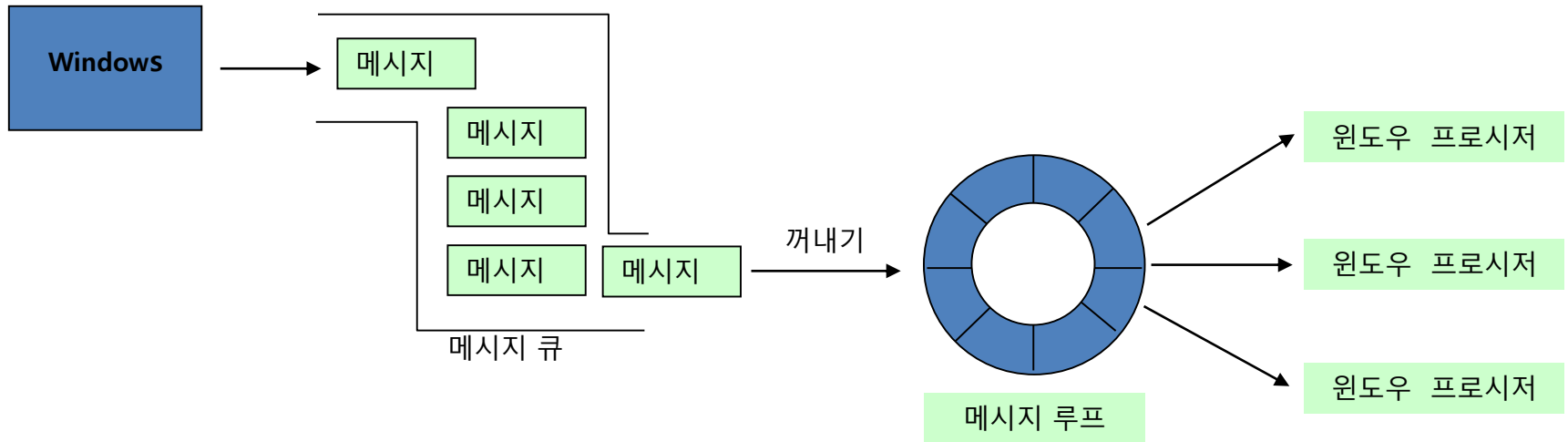
- 윈도우 화면에 출력하기 위해 디바이스 컨텍스트 개념을 이해할 수 있다.
- 텍스트를 출력하는 기본 함수를 사용할 수 있다.
- 기본 도형을 화면에 출력할 때 필요한 요소와 함수를 사용할 수 있다.

### • 내용

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

# 메시지 기반 프로그램

- 메시지 기반의 프로그램 진행



- 윈도우에서 발생하는 이벤트에 의한 메시지를 처리하며 프로그램이 진행된다.

# 윈도우 메시지들

- 마우스/키보드 메시지, 윈도우 메시지, 시스템 메시지 등 수백 개의 메시지가 발생되어 처리됨

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 비 작업영역의 활성화 또는 비 활성화시 발생 (윈도우 타이틀 바 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, 액셀러레이터 선택 시 발생

# 헝가리언 표기법

- 헝가리언 표기법

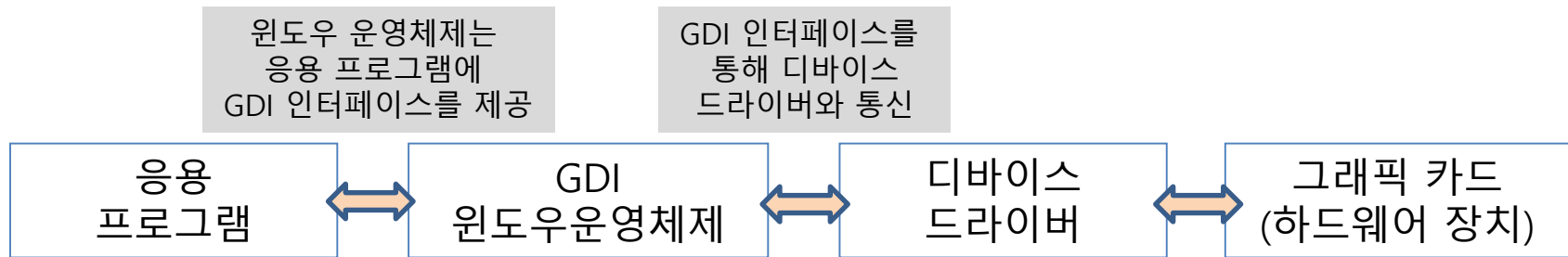
- Microsoft사의 Charles Simonyi에 의해 개발된 형식
- 변수명을 만들 때, 변수명 앞에 데이터형 접두어를 붙이는 것
- 변수가 무엇을 의미하고 어떤 데이터 타입을 갖는지 알 수 있다.
- API 프로그래밍 시 많은 프로그래머들이 이 방법을 즐겨 쓴다.

접두어 (prefix)	데이터 타입	접두어 (prefix)	데이터 타입
a	배열 (array)	i	int, 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)

# 1. 출력 영역 얻기

- **GDI (Graphic Device Interface)**

- 윈도우 운영체제상에서 제공하는 응용 프로그램과 그래픽 장치간의 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 등이 있음
- 선 그리기, 칼라 처리 등 그래픽을 다루기 위한 함수의 모음

# 디바이스 컨텍스트: 출력 영역 얻기

- 디바이스 컨텍스트 (DC, Device Context)

- 윈도우가 제공하는 화면을 사용할 수 있는 권한
- 그래픽 관련한 선택 정보 (폰트, 색상, 굵기, 무늬, 출력 방법 등)를 모아 놓은 구조체
  - GDI에 의해서 관리된다.
  - 간단한 출력은 디폴트 사양 이용
  - 세밀한 출력은 관련 선택정보(option) 변경

- 윈도우에서 출력 장치에 무언가 출력하기 위해서는 반드시 DC가 필요, DC 핸들을 얻은 후 해당 DC에 데이터를 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 얻어오기

## • 디바이스 컨텍스트 얻어오는 방법

- 다양한 함수 호출을 통하여 디바이스 컨텍스트를 얻어온다.

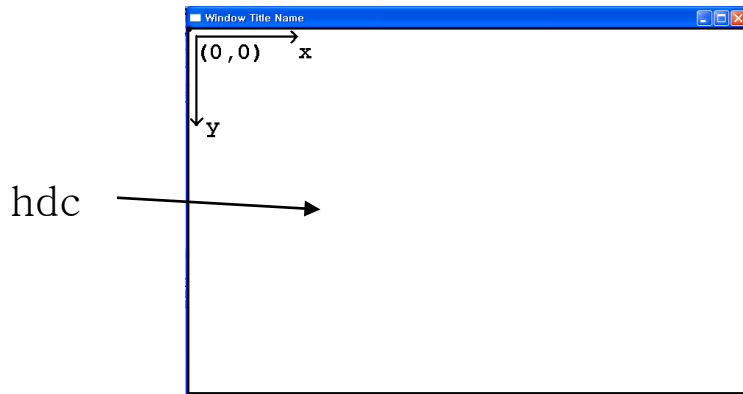
호출 함수	기능
BeginPaint() EndPaint():	WM_PAINT메시지와 함께 사용
GetDC() ReleaseDC():	잠시 출력할 때 사용 (WM_PAINT 메시지 외에서 사용)
CreateDC() DeleteDC():	DC를 만들어 사용
GetWindowDC() ReleaseDC():	비 클라이언트 영역을 그리고자 할 때 WM_NCPAINT메시지와 함께 사용
CreateIC() DeleteDC():	DC에 출력하지 않고 정보만 얻고자 할 때 사용
CreateCompatibleDC() DeleteDC():	이미 있는 DC와 같은 또 하나의 DC만들 때 사용. 보통 디스플레이를 이용한 메모리 DC를 만들 때 사용



# 디바이스 컨텍스트 얻어오기

- HDC hdc

- 디바이스 컨텍스트 핸들이라고 부르고, 출력할 영역을 지정할 수 있는 타입
- 화면의 경우 윈도우로부터 얻어 옴
- hdc 변수는 출력할 영역을 얻어오면 얻어온 영역을 지정할 수 있음



- 일반 메시지에서 DC를 얻어오는 GetDC() 함수 / 해제하는 ReleaseDC () 함수

## HDC GetDC (hWnd);

- HWND hwnd; // hwnd: 생성된 윈도우의 핸들값
- 리턴 값으로 디바이스 컨텍스트 핸들을 얻어온다.

## int ReleaseDC (hWnd, hDC);

- HWND hWnd;
- HDC hDC;

# 디바이스 컨텍스트 얻어오기

- WM\_PAINT 메시지에서 DC를 얻어오는 BeginPaint() 함수 / 해제하는 EndPaint () 함수

## HDC BeginPaint (hWnd, \*lpPaint);

- HWND hwnd;

- PAINTSTRUCT \*lpPaint;

//생성된 윈도우의 핸들값

//출력될 영역에 대한 정보를 저장한

// 구조체 공간에 대한 주소

## BOOL EndPaint (HWND hWnd, \*lpPaint);

- HWND hwnd;

- PAINTSTRUCT \*lpPaint;

## – PAINTSTRUCT 구조체

```
typedef struct tagPAINTSTRUCT {
```

```
    HDC hdc;
```

```
    BOOL fErase;
```

```
    RECT rcPaint;
```

```
    BOOL fRestore;
```

```
    BOOL fIncUpdate;
```

```
    BYTE rgbReserved[16];
```

```
} PAINTSTRUCT;
```

//그리고자 하는 DC 핸들

// 배경 삭제 여부를 가리킨다: 0이 아니면 다시 그린다.

// 다시 그릴 사각형의 좌표값

# 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 dc를 이용하여 출력이 이루어짐

            EndPaint (hwnd, &ps) ;

            break;

        case WM_TIMER:
            hdc = GetDC (hwnd);
            //-- dc를 이용하여 출력 수행

            ReleaseDC (hwnd, hdc);

            break;

        case WM_DESTROY:
            PostQuitMessage ( 0 );

            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# 메시지 처리: WM\_PAINT 메시지

- **WM\_PAINT 메시지: 클라이언트 영역이 다시 그려져야 할 필요가 있을 때 윈도우 OS가 보내는 메시지, 대부분의 출력은 이 메시지 처리부분에서 작성해야 한다.**
  - 윈도우의 크기가 변경되었을 때나 다른 윈도우에 가려져 있다가 드러날 때 등 화면에 출력된 결과가 깨질 수 있다.
  - 화면의 그래픽 일부가 깨지거나 다시 출력해야 할 필요가 있는 영역을 **무효화(Invalid) 영역**이라 하고 이러한 경우를 화면이 무효화 되었다고 한다.
  - OS는 깨진 화면을 복구해주지 않는다.
  - 단지 OS는 화면이 깨질 때 마다 WM\_PAINT 메시지를 발생시켜준다.
  - 즉, 윈도우의 클라이언트 영역 중 일부가 무효화(Invalid)되면 OS가 WM\_PAINT 메시지를 큐에 넣어준다.
  - 그래서 **출력은 WM\_PAINT 메시지 아래에서 해야 한다!!**
  - 그래야 화면이 깨질 때 마다 WM\_PAINT 메시지가 발생하고 그 아래에 작성한 소스가 다시 실행되어 화면이 복구된다!
- **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()**, 또는 **InvalidateRgn()** 함수를 호출하여 강제로 화면을 무효화시킬 때

# 메시지 처리: WM\_PAINT 메시지

- 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\_PAINT메시지는 모든 메시지 중에서 우선 순위가 가장 낮다.
  - GetMessage()함수는 메시지 큐에 WM\_PAINT메시지가 있더라도 다른 메시지가 대기 중이면 그 메시지를 먼저 처리한다.
  - WM\_PAINT메시지는 큐에 대기중인 다른 메시지가 없고 무효화 영역이 존재할 때만 윈도우 프로시저로 보내진다.
- WM\_PAINT메시지는 한번에 하나만 메시지 큐에 들어갈 수 있다.
  - 만약 무효화 영역이 생겼는데 WM\_PAINT메시지가 이미 메시지 큐에 있으면 기존의 무효화 영역과 새 무효화 영역의 합으로 새로운 무효화 영역이 설정된다.
- 다른 메시지에서도 출력은 가능하다!

## 메시지 처리: WM\_PAINT 메시지

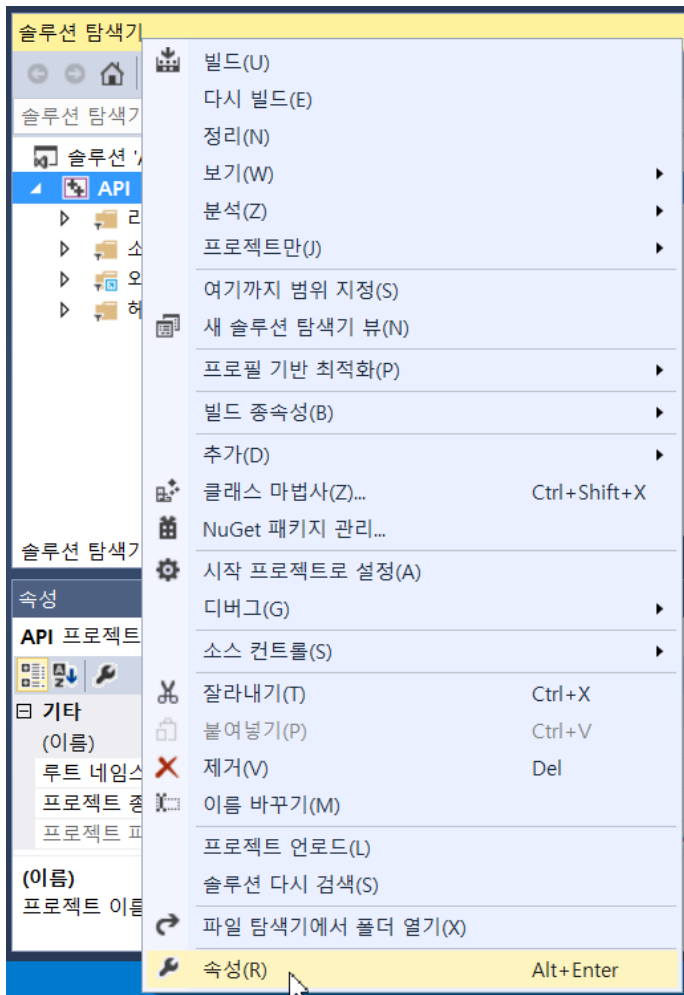
- 해당 윈도우 프로시저에서 이 메시지를 처리하지 않으면 이 메시지는 DefWindowProc() 함수가 처리한다.
  - 이 함수는 무효영역을 모두 유효화(valid)하며 다시 그리기는 하지 않는다.
- 만약 비 클라이언트 영역도 그려져야 한다면 WM\_NCPAINT 메시지를 전달하며, 배경을 지워야 한다면 WM\_ERASEBKGND 메시지를 전달한다.
- WM\_PAINT 메시지에서 그리기를 할 때는 BeginPaint()와 EndPaint() 함수를 사용해야 한다.
  - 이 두 함수는 WM\_PAINT 메시지 내에서만 사용되며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

# 메시지 처리: WM\_CREATE / WM\_DESTROY 메시지

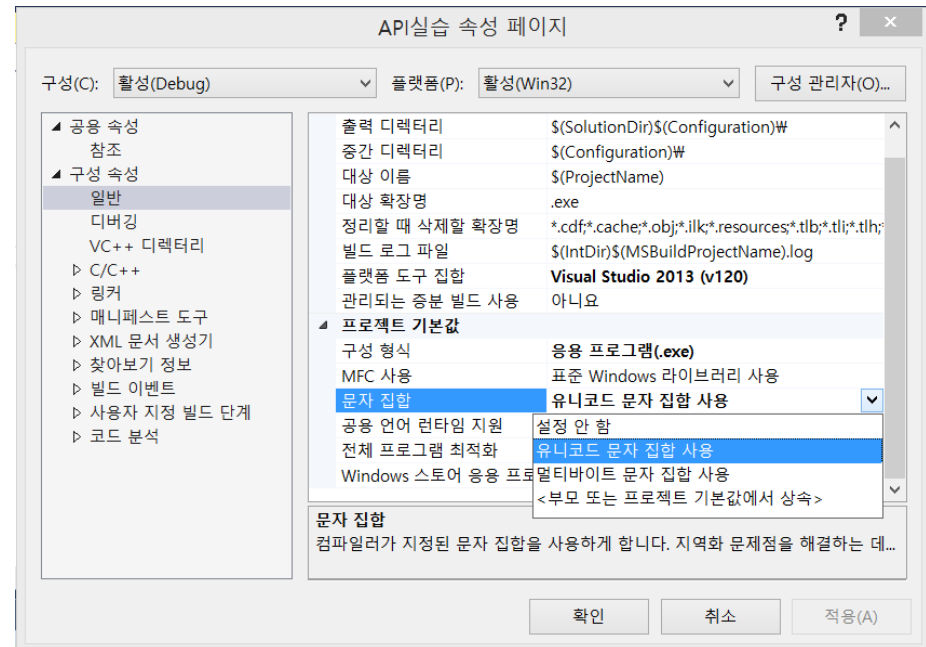
- 윈도우가 생성될 때: **WM\_CREATE** 메시지
  - CreateWindowEx 함수에 의해 윈도우가 생성될 때 호출되는 메시지
  - 메모리에 윈도우를 생성한 후 화면에 보이기 전에 보내지며 주로 윈도우에 관련된 초기화 작업을 할 때 사용됨
  - 윈도우 동작을 위한 메모리 할당, 리소스 생성, 차일드 컨트롤 생성, 윈도우 속성 초기화 작업에 이 메시지가 사용된다.
  - CreateWindowEx 함수는 이 메시지를 완전히 처리한 후에 리턴
- 프로그램을 종료할 때: **WM\_QUIT** 메시지
  - PostQuitMessage () 함수를 호출하여 프로그램을 종료한다.
  - 프로그램을 종료하고 윈도우를 파괴할 때 호출되는 메시지
    - WM\_DESTROY: 사용자가 시스템 메뉴를 더블 클릭하거나 Alt+F4를 눌러 프로그램을 끝내려고 할 때 발생하는 메시지
    - WM\_QUIT: PostQuitMessage 함수를 호출하면 발생한다. **WM\_QUIT** 메시지가 입력되면 메시지 루프의 GetMessage 함수 리턴값이 False가 되어 프로그램이 종료된다. 윈도우에 전달되는 메시지는 아님.
  - 윈도우를 닫는 함수
    - BOOL DestroyWindow (HWND hWnd);
      - hWnd 윈도우를 파괴한다. 단, 이 함수로 다른 스레드에서 생성한 윈도우를 파괴할 수는 없다

## 2. 문자 집합

- 현재 설정된 문자 집합 보기
  - 프로젝트 이름에서 마우스 오른쪽 버튼을 누른 후, 속성 선택



- 기본 설정은 유니코드
  - 일반->문자 집합 확인





# 멀티 바이트 문자 집합(MBCS) 사용

- 멀티바이트 형의 문자 집합

- 한 글자를 저장하기 위해 2바이트 이상을 사용할 수 있다.
- 실제 글자의 개수와 공간의 크기가 일치하지 않을 수 있다.
- 메모리 낭비가 없어 효율적인 장점

- 영어 알파벳만 사용 했을 때

- `char str[15] = "I love you";`

I		I	o	v	e		y	o	u	₩0				
---	--	---	---	---	---	--	---	---	---	----	--	--	--	--

- 한글 혼용(멀티 바이트) 했을 때

- `char str[15] = "나는 love";`

나	는		I	o	v	e	₩0							
---	---	--	---	---	---	---	----	--	--	--	--	--	--	--

# 유니코드 문자 집합 사용

- 유니코드형의 문자 집합

- 영어나 한글 구분없이 모든 문자를 2바이트 사용하여 저장
- 위치를 쉽게 알 수 있다.
- 메모리 낭비가 심하다

- 한글 혼용 했을 때

- `wchar str[15] = L"나는 love";` //L: 유니코드로 변환

나	는		₩0		₩0	o	₩0	v	₩0	e	₩0	₩0
---	---	--	----	--	----	---	----	---	----	---	----	----

# 유니코드 문자 집합 사용

- 문자집합 설정을 어떻게 해도 처리되는 자료형

- TCHAR 형
- 멀티바이트이면 TCHAR는 char로 변환
- 유니코드이면 TCHAR는 wchar로 변환

- 사용 예

```
#include <TCHAR.H>
```

```
TCHAR str[15] = _T("나는 love");
```

# 문자열 자료형

- 문자열 자료형

API 자료형	같은 의미의 자료형	설명
LPSTR	char *	ANSI 코드 문자열 포인터
LPCSTR	const char *	ANSI 코드 문자열 포인터 상수
LPTSTR	TCHAR *	TCHAR 문자열 포인터
LPCTSTR	const TCHAR *	TCHAR 문자열 포인터 상수
LPWSTR	WCHAR *	유니코드 문자열 포인터
LPCWSTR	const WCHAR *	유니코드 문자열 포인터 상수

- LP : long pointer
- C : const
- T : TCHAR (유니코드, 멀티바이트 모두 지원)
- W: WCHAR (유니코드 문자열)
- STR : 문자열

### 3. 텍스트 출력하기

- 한 점 기준 텍스트 출력 함수

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

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

# 윈도우에 “Hello World” 출력하기

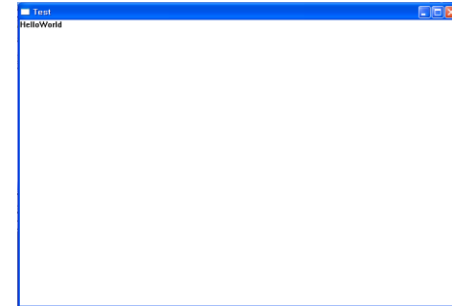
- 윈도우 프로시저 함수 안에서 WM\_PAINT 메시지에서 출력 처리

LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```
{
    HDC hdc;
    PAINTSTRUCT ps;

    switch (iMsg)
    {
        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps) ;
            TextOut (hdc, 0, 0, "HelloWorld", 10);
            EndPaint (hwnd, &ps) ;
            break;

        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



# 박스 영역에 텍스트 출력 함수: DrawText ()

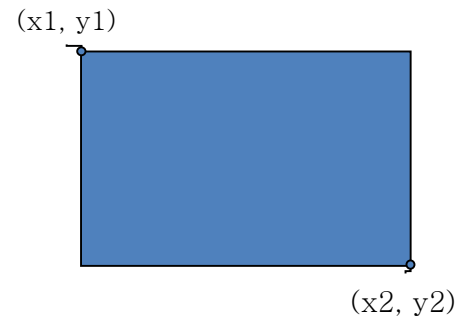
- 박스 영역에 텍스트 출력 함수

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

- HDC hdc: BeginPaint()나 GetDC()를 통해 얻어온 DC핸들
- LPCSTR lpString: 출력 문자열
- int nLength: 문자열 길이
- LPRECT lpRect: 문자열을 출력할 박스영역 구조체의 주소 (RECT \*)
- UINT Flags: 출력 방법

- RECT 구조체

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



# DrawText() 출력 방법

- **DrawText 함수의 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 위쪽에 출력
  - DT\_BOTTOM: 박스 영역의 상하에서 아래쪽에 출력
  - DT\_CALCRECT: 문자열을 출력한다면 차지할 공간의 크기 측정
- **예: 1개 이상의 설정을 하는 경우**
  - DT\_TOP | DT\_CENTER | DT\_SINGLELINE: 사각형 영역의 위쪽 가운데에 한 줄로 출력



# DrawText() 함수 사용하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
```

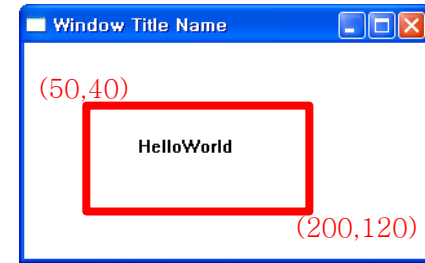
```
{
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;

    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;

        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



```
DrawText (hdc, "HelloWorld", 10, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
// 한 라인, 수직/수평 중앙
```

# 문자 출력시 배경색, 전경색 모드 지정

- 문자 색 및 배경색 변경 해주는 함수

**COLORREF SetBkColor** (HDC hdc, COLORREF crColor); // 문자의 배경색 설정

- hdc: 디바이스 컨텍스트 핸들
- crColor: 배경색

**COLORREF SetTextColor** (HDC hdc, COLORREF crColor); // 문자 색 설정

- hdc: 디바이스 컨텍스트 핸들
- crColor: 문자색

**int SetBkMode** (HDC hdc, int iBkMode); // 배경 모드 설정

- hdc: 디바이스 컨텍스트 핸들
- iBkMode: 배경 모드 (OPAQUE/TRANSPARENT)

- 사용 예)  
앞 페이지의 예에서

```
SetTextColor (hdc, RGB (255, 0, 0)); // 이후의 문자는 빨간색으로 출력된다.  
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 RGB (BYTE R, BYTE G, BYTE B);

- R, G, B: 빛의 3원색으로 0 ~ 255 사이의 정수값
- 0 ~ 16777215 사이의 색상값 설정

COLORREF: DWORD 형태의 타입

```
COLORREF RGB {  
    BYTE byRed,           // 색상 중 빨간 색  
    BYTE byGreen,         // 색상 중 초록 색  
    BYTE byBlue           // 색상 중 파란 색  
}
```

- 사용 예)

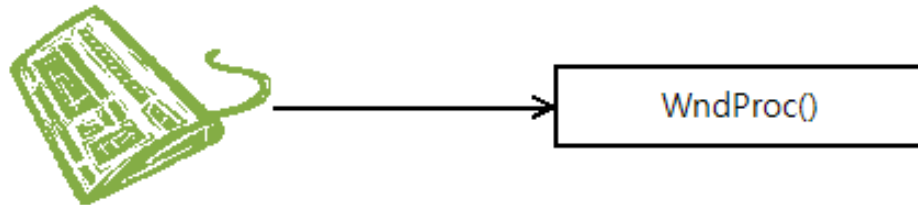
```
COLORREF text_color ;  
text_color = RGB (255, 0, 0);
```

```
SetTextColor (hdc, text_color );
```

```
DrawText (hdc, "HelloWorld", 10, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
```

## 4. 키보드 메시지 처리하기

- 키보드로 입력한 정보 받기



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>

# 메시지 처리: 키보드 눌렀을 때 발생하는 메시지

- **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 회수 같은 부가 정보

# 메시지 처리: 키보드 눌렀을 때 발생하는 메시지

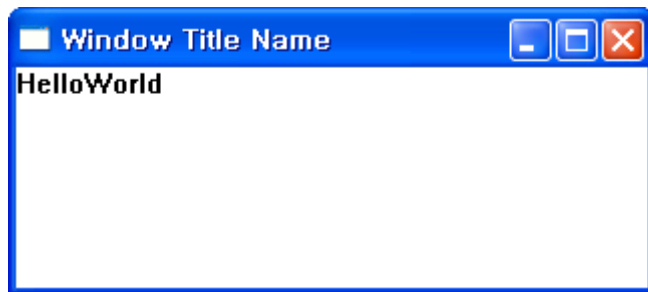
메시지	내용	윈도우 프로시저 인수값	비고
WM_KEYDOWN	키보드에서 키를 눌렀을 때 발생하는 메시지	<b>wParam:</b> 이벤트가 발생시킨 키의 가상키 값 (문자 혹은 특수 문자의 아스키 값)  <b>lParam:</b> 스캔 코드, 키 반복 회수 같은 부가 정보	'a'키를 눌렀을 때: WM_KEYDOWN 메시지 발생하고, 다음으로 WM_CHAR 메시지 발생
WM_KEYUP	키보드에서 키를 눌렀다가 떼어지면 발생하는 메시지		
WM_CHAR	문자 키를 눌렀을 때 발생하는 메시지		

- 위의 메시지를 윈도우 프로시저 함수에서 처리한다.

# WM\_KEYDOWN 메시지 처리하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;

    switch (iMsg) {
        case WM_KEYDOWN: // 키가 눌렸을 때
            hdc = GetDC(hwnd);
            TextOut(hdc, 0,0,"HelloWorld",10);
            ReleaseDC(hwnd,hdc);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



[a]를 누른결과



[Enter]를 누른결과

눌린 키에 관계없이 결과 출력

# 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 옴
            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);
}
```



# 입력 문자열 처리하기

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

    switch (iMsg) {
        case WM_CREATE :
            count = 0;
            break ;
        case WM_CHAR:
            hdc = GetDC(hwnd);
            str[count++] = wParam;           // 문자저장 후 인덱스 증가
            str[count] = '\0';              // 문자열은 null('\0')로 끝남
            TextOut (hdc, 0, 0, str, strlen(str));
            ReleaseDC (hwnd,hdc);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# 앞의 코드에서의 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
  - 해결방법: 가상키 사용하여 처리 필요

# WM\_PAINT 메시지 처리하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    static char str[100];
    static int count = 0;

    switch (iMsg) {
        // 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;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# WM\_PAINT 강제 발생 함수

- WM\_PAINT 메시지를 발생시키는 함수

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

- hWnd: 수정될 영역에 대한 영역 핸들 값
- lpRect: 영역 좌표 (NULL이면 전체 영역을 수정)
- bErase: BeginPaint()를 위해 플래그 (TRUE - 다음에 호출되는 BeginPaint에서 배경을 먼저 지운 후 작업 영역을 그리게 된다. FALSE - 배경 브러시에 상관없이 배경을 지우지 않는다.)

**BOOL InvalidateRgn (HWND hWnd, HRGN hRgn, BOOL bErase );**

- hWnd: 수정될 영역이 포함된 윈도우의 핸들값
- hRgn: 수정될 영역에 대한 핸들값 (NULL이면 클라이언트 영역 전체를 수정)
- bErase: 수정될 때 배경을 모두 삭제할지 안 할지를 결정하는 BOOL 값. (TRUE - 배경이 삭제됨, FALSE - 배경이 그대로 남아있음)

- InvalidateRgn**이나 **InvalidateRect** 함수를 호출하면 클라이언트의 특정 영역을 무효화하므로 다시 그리기가 필요할 때 이 함수를 호출한다. 즉, 윈도우의 업데이트를 하게 된다

# 문자 저장과 출력 구분하기

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

    switch (iMsg) {
        case WM_CHAR:
            str[count++] = wParam;           // 문자 저장
            str[count] = '\0';
            InvalidateRect (hwnd, NULL, TRUE); // 직접 출력하지 않고 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;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

## 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	Ctrl	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		

# Backspace 키 입력 처리하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static int count = 0;
    static char str[80];

    switch (iMsg) {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            TextOut(hdc,0,0,str,strlen(str));           // 문자 출력
            EndPaint(hwnd, &ps);
            break;

        case WM_KEYDOWN:
            if (wParam == VK_BACK)
                count--;
            else
                str[count++] = wParam;
            str[count] = '\0';
            InvalidateRect (hwnd, NULL, TRUE);         // WM_PAINT 메시지 발생
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# Enter 키 입력 처리하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static int count = 0;
    static char str[80];
    static int yPos;

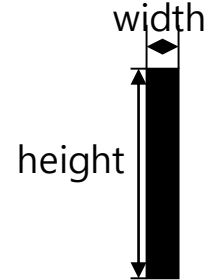
    switch (iMsg) {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            TextOut(hdc, 0, yPos, str, strlen(str));
            EndPaint(hwnd, &ps);
            break;

        case WM_KEYDOWN :
            if (wParam == VK_BACK)
                count--;
            else if (wParam == VK_RETURN)
            {
                count = 0;
                yPos = yPos + 20;
            }
            else
                str[count++] = wParam;
            InvalidateRect (hwnd, NULL, TRUE);    // WM_PAINT 메시지 발생
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



## 5. Caret(커서) 이용하기

- **Caret:** 키보드 입력 시 글자를 입력할 위치에 깜박거리는 커서
  - 캐럿이 있으면 어느 위치에 문자가 입력되는지를 알 수 있다.



**BOOL CreateCaret** (hwnd, NULL, width, height);

- 캐럿 만들기 함수
- HWND hwnd: 캐럿을 놓을 윈도우 핸들
- HBITMAP hbitmap: 비트맵 캐럿
- int x, int y: 캐럿의 x, y 위치

**BOOL ShowCaret** (hwnd);

- 캐럿 보이기
- HWND hwnd: 캐럿이 보일 윈도우 핸들

**BOOL SetCaretPos** (x, y);

- 캐럿 위치 설정하기
- int x, int y: 캐럿의 x, y 위치

**BOOL HideCaret** (hwnd);

- 캐럿 감추기
- HWND hwnd: 캐럿이 보일 윈도우 핸들

**BOOL DestroyCaret** ();

- 캐럿 삭제하기

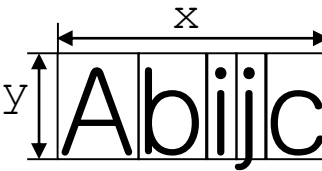
# Caret 위치 정하기

- 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
  - 예) 문자열 출력 위치가 (100,200)이라고 하면 caret의 위치는 (100+x, 200)이다.

# 출력될 문자열 폭 구하기

BOOL **GetTextExtentPoint** (HDC hdc, LPCTSTR lpString, int cbString, LPSIZE lpSize );

- 문자열의 폭 구하기 함수
- HDC hdc: DC 핸들
- lpString: 크기를 측정할 문자열
- cbString: 정수로 문자열의 몇 번째 문자까지 크기를 측정할 지 알려준다
- lpSize: 문자열의 크기 (폭, 높이) -> 얻어오는 값

- struct **tagSIZE** { // 문자열의 폭과 높이 저장  
    LONG cx;  
    LONG cy;  
} SIZE;

## 예) "Abijc" 폭 구하기

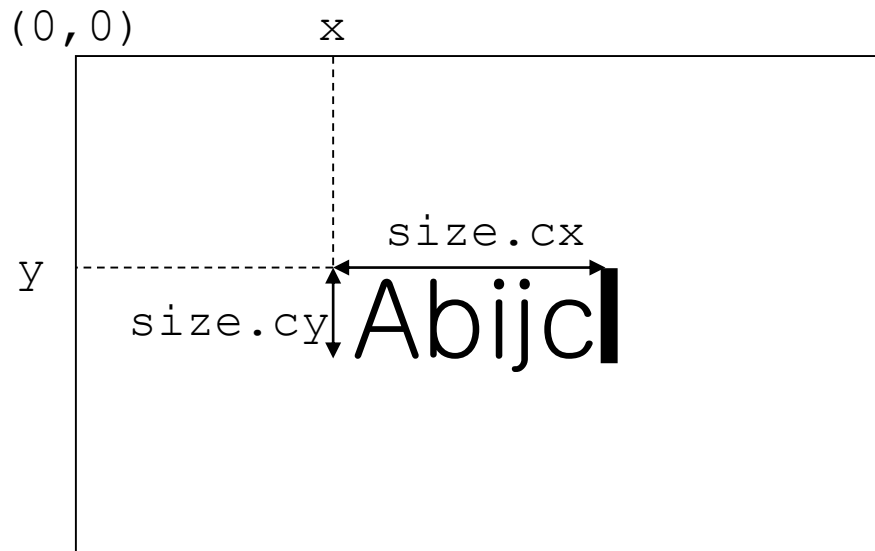
```
SIZE size;
```

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

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

```
SetCaretPos (x + size.cx, y);
```

// x좌표에 출력문자열 길이 합산



# Caret 표시

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static SIZE size;

    switch (iMsg) {
        case WM_CREATE :
            CreateCaret (hwnd, NULL, 5, 15);           // 캐럿 만들기
            ShowCaret (hwnd);                          // 빈 화면에 캐럿 표시
            count = 0;
            return 0 ;

        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            GetTextExtentPoint (hdc, str, strlen(str), &size); // 문자열 길이 알아내기
            TextOut(hdc,0,0,str,strlen(str));
            SetCaretPos (size.cx, 0);                  // 캐럿 위치하기
            EndPaint(hwnd, &ps);
            break;

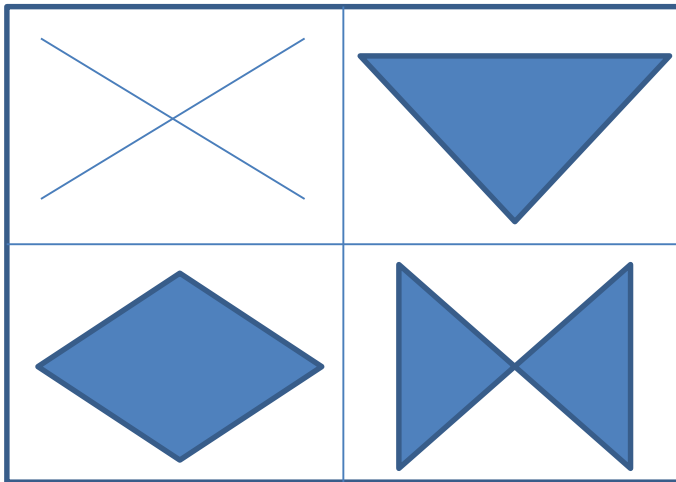
        case WM_DESTROY :
            HideCaret (hwnd);                          // 캐럿 숨기기
            DestroyCaret ();                            // 캐럿 삭제하기
            PostQuitMessage (0) ;
            return 0 ;

    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

## 실습 2-1

- 화면에 문자 출력하기

- 화면을 사등분 한다.
- 각 등분에 각각 X, 역삼각형, 마름모, 나비 모양을 임의의 문자로 출력한다.
- 각 모양을 나타내는 문자의 색과 배경색을 다르게 설정한다.
- 키보드 명령:
  - n/N: 화면의 각등분에 위의 네 모양이 임의의 순서로 다시 출력된다.
  - q/Q: 프로그램을 종료한다.



옆의 모양을 문자로 그린다.

## 실습 2-2

- 키보드 입력하여 구구단 출력하기
  - 화면을 띄운다.
  - 좌측 상단에 명령어를 받는다
    - x: 가로 위치
    - Y: 세로 위치
    - N: 단 수
  - 입력한 위치에 단수의 구구단을 출력한다.
  - 출력 후 명령어를 다시 받을 수 있도록 한다.
  - 0을 입력하면 프로그램을 종료한다.

50 100 5

5\*1 = 5  
5\*2 = 10  
5\*3 = 15  
...  
5\*9 = 45





## 6. 직선, 원, 사각형, 다각형 그리기

- 직선 그리기
- 원 그리기
- 사각형 그리기
- 다각형 그리기
- 선 속성 바꾸기
- 면 색 바꾸기

# 직선 그리기

- 직선의 시작점으로 이동하기 함수

```
BOOL MoveToEx (HDC hdc, int X1, int Y1, LPPOINT lpPoint );
```

- hdc: DC 핸들
- X1, Y1: 직선의 시작점 (x와 y 좌표값)
- lpPoint: 이전의 좌표, 사용 안함

(X1, Y1)

(X2, Y2)

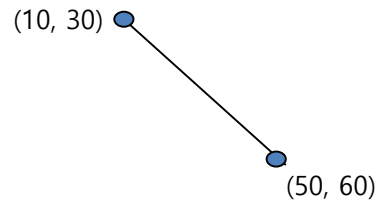
- 직선의 끝점까지 직선 그리기 함수

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

- hdc: DC 핸들
- X2, Y2: 직선의 끝점

```
case WM_PAINT :
```

```
    hdc = BeginPaint (hwnd, &ps) ;  
    ); MoveToEx (hdc, 10, 30, NULL);  
    LineTo (hdc, 50, 60);  
    EndPaint (hwnd, &ps) ;  
    return 0 ;
```

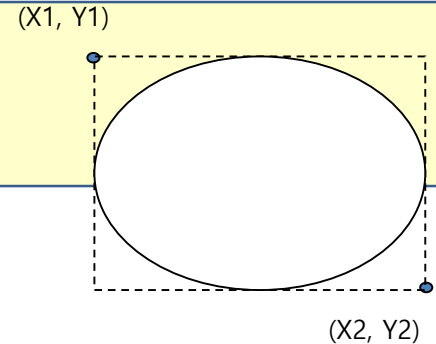


# 원 그리기

- 두 점의 좌표를 기준으로 만들어진 가상의 사각형에 내접하는 원을 그리는 함수

**BOOL Ellipse (HDC hdc, int X1, int Y1, int X2, int Y2 );**

- X1, Y1: 좌측 상단 좌표값 (x와 y 최소값)
- X2, Y2: 우측 하단 좌표값 (x와 y 최대값)



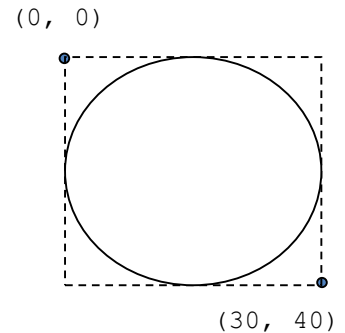
case WM\_PAINT :

hdc = BeginPaint (hwnd, &ps) ;

**Ellipse(hdc, 0, 0, 30, 40);** // 박스의 좌표, 중심점 (15,20)

EndPaint (hwnd, &ps) ;

return 0 ;

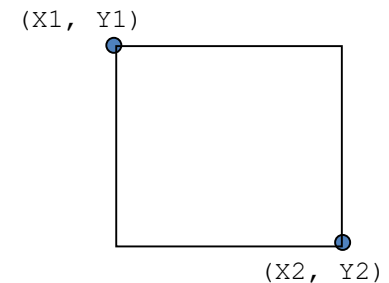


# 사각형 그리기

- 두 점의 좌표를 기준으로 수평수직 사각형을 그림

**BOOL Rectangle** (HDC hdc, int X1, int Y1, int X2, int Y2 );

- X1, Y1: 좌측 상단 좌표값 (x와 y 최소값)
- X2, Y2: 우측 하단 좌표값 (x와 y 최대값)



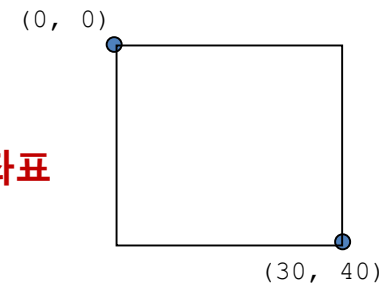
case WM\_PAINT :

hdc = BeginPaint (hwnd, &ps) ;

**Rectangle(hdc, 0, 0, 30, 40);** // 원을 둘러싼 사각형의 좌표

EndPaint (hwnd, &ps) ;

return 0 ;



# 사각형 그리기

- 사각형 그리기 다른 함수들
  - 내부가 채워진 사각형 그리기:

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

- hDC: DC 핸들
- lprc: 사각형의 좌표값
- hbr: 내부 색

- 외곽선 사각형 그리기:

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

- hDC: DC 핸들
- lprc: 사각형의 좌표값
- hbr: 외곽선 색

# 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 lprcDest, CONST RECT \*lprcSrc1, CONST RECT lprcSrc2);**

- 두 RECT (lprcSrc1, lprcSrc2)가 교차되었는지 검사한다.
- lprcDest: 교차된 RECT 부분의 좌표값이 저장된다.

**BOOL UnionRect (LPRECT lprcDest, CONST RECT \*lprcSrc1, CONST RECT \*lprcSrc2)**

- 두 RECT (lprcSrc1, lprcSrc2) 를 union 시킨다.
- lprcDest: 두 사각형을 합한 사각형의 좌표값이 저장된다.

**BOOL PtInRect (CONST RECT \*lprc, POINT pt);**

- 특정 좌표 pt가 lprc 영역 안에 있는지 검사한다.

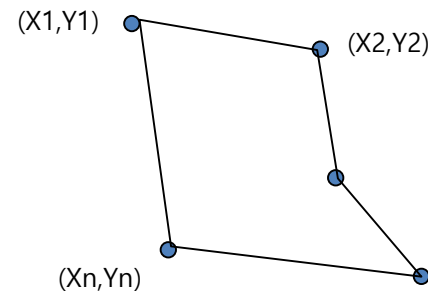
# 다각형 그리기

- 연속되는 여러 점의 좌표를 직선으로 연결하여 다각형을 그림

**BOOL Polygon** (HDC hdc, CONST POINT \*lppt, int cPoints );

- hdc: DC 핸들
- lppt: 다각형 꼭지점의 좌표 값이 저장된 리스트
- cPoints: 꼭지점의 개수

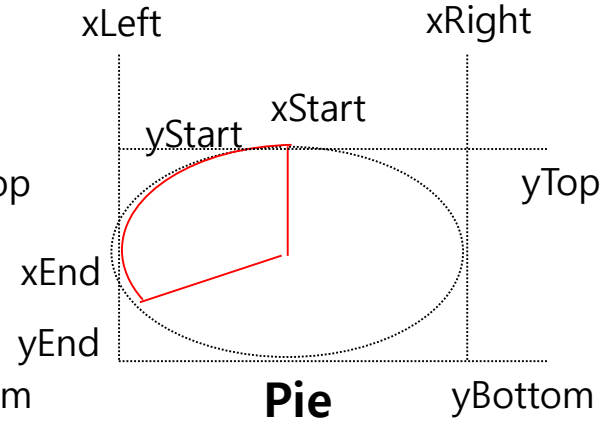
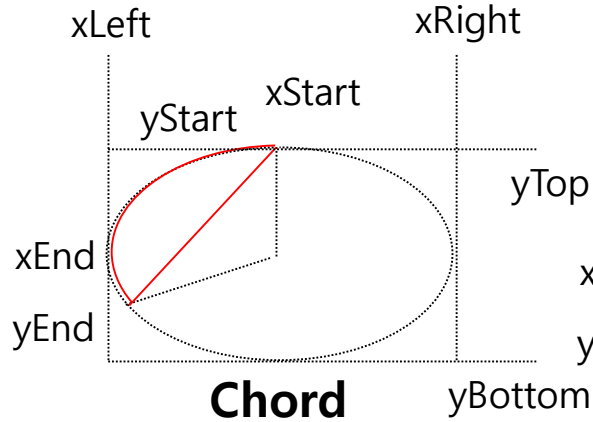
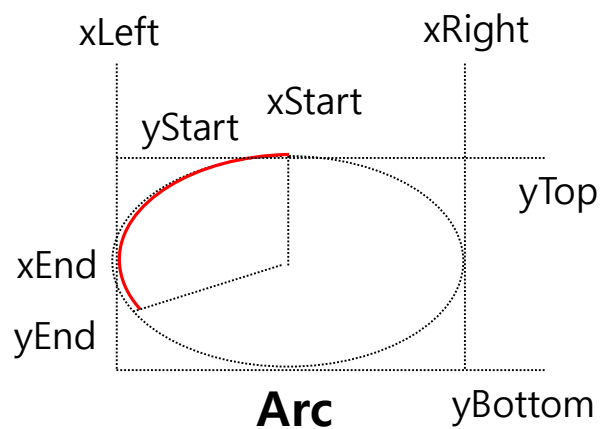
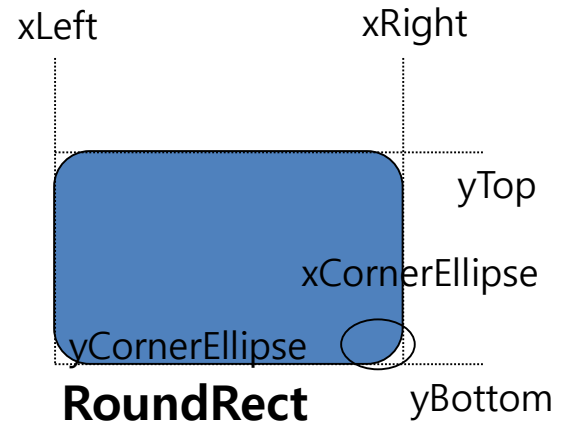
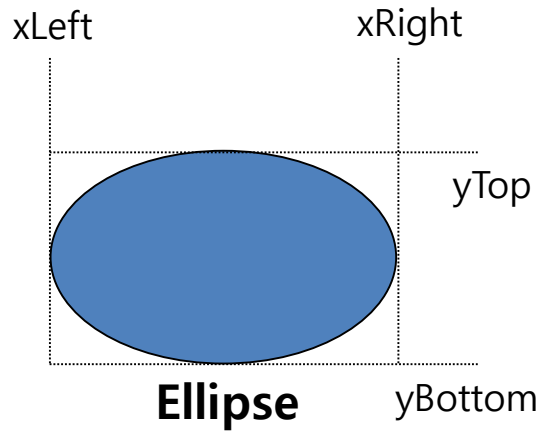
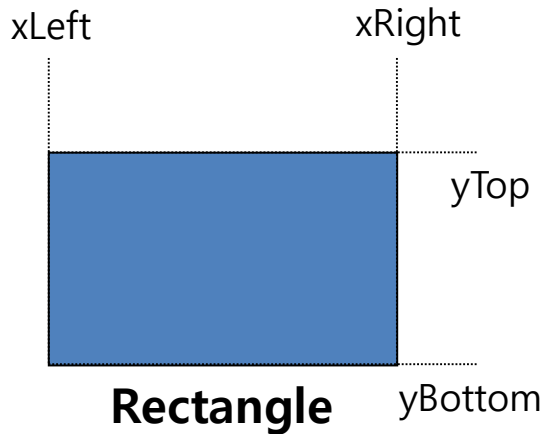
```
typedef struct tagPOINT {  
    LONG x;  
    LONG y;  
} POINT;
```



```
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 ;
```

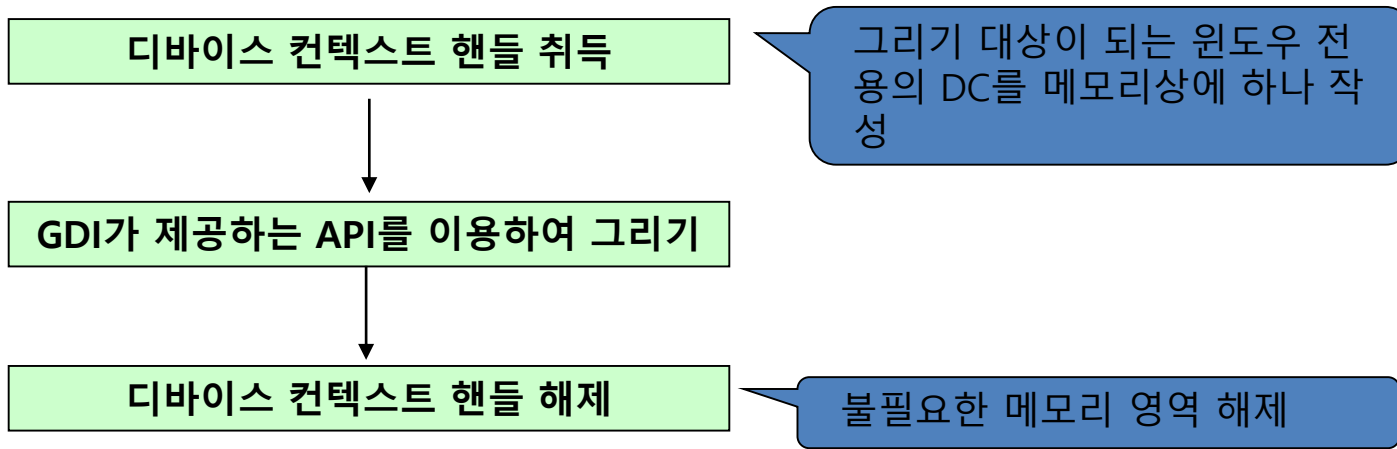
# 도형 그리기





# 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



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

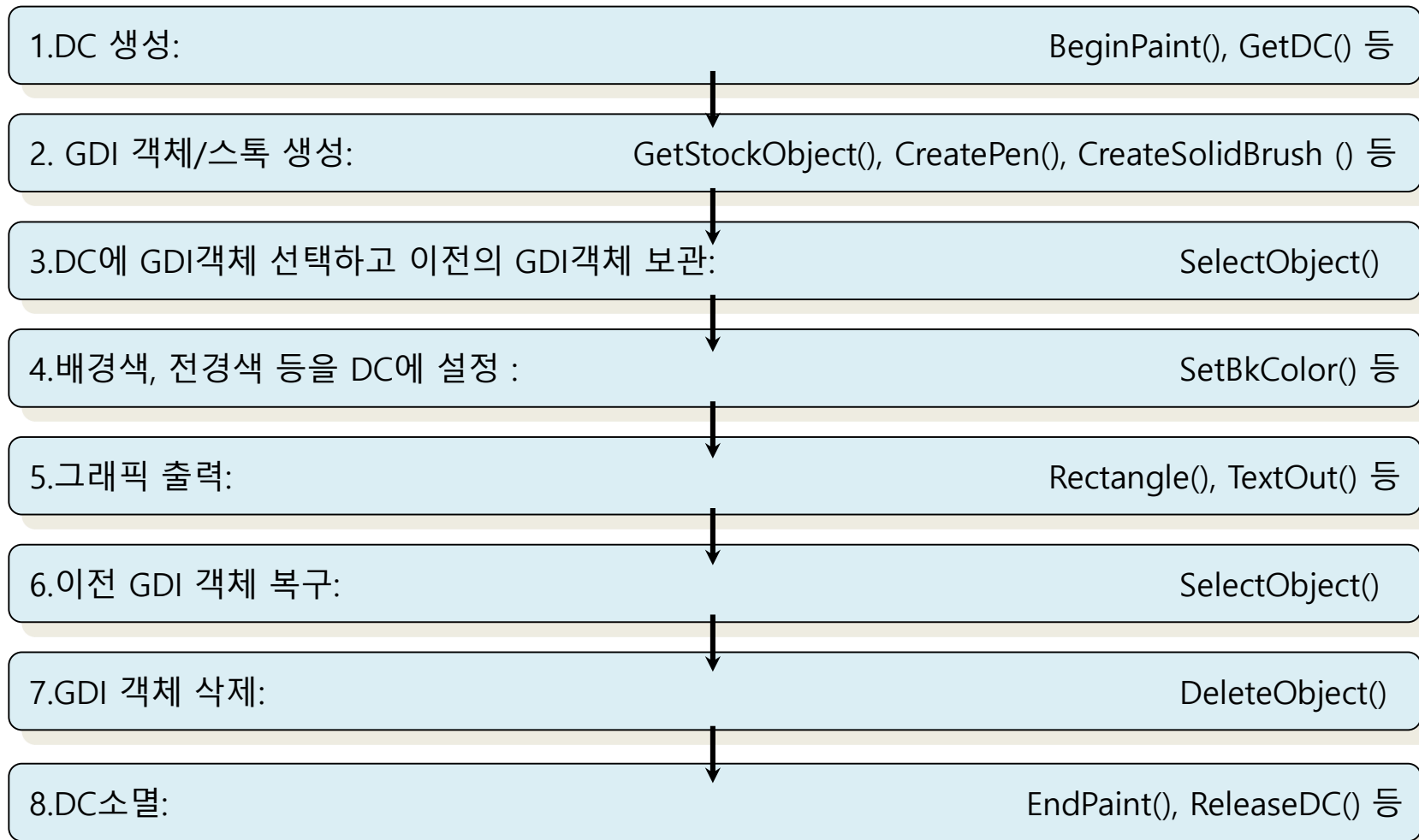
# GDI 객체들

GDI 오브젝트	의미	내용	핸들타입	디폴트 값
펜	선을 그을 때	선을 그리거나 영역의 경계선을 그릴 때 사용 선의 색, 두께, 형태 등을 지정 디폴트는 검은색 1픽셀 실선	HPEN	검정색의 가는 실선
브러시	면을 채울 때	어떤 영역의 내부를 채울 때 사용 채우기 색, 채우기 패턴 등을 지정 디폴트는 흰색	HBRUSH	흰색
폰트	문자 출력 글꼴	문자를 출력할 때 사용하며 색, 모양, 크기 등을 지정 디폴트는 시스템 폰트	HFONT	시스템 글꼴
비트맵	비트맵 이미지	비트맵 그림 파일에 관한 옵션	HBITMAP	X
팔레트	팔레트	화면에 출력할 수 있는 색에 제한을 받을 경우, 실제로 화면에 출력할 색의 수 등을 지정	HPALETTE	X
리전	화면상의 영역	임의의 도형을 그리는 것과 관련된 옵션을 설정	HRGN	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 Create로 시작하는 함수로 GDI객체를 생성
  - **객체 선택 및 보관**: 만든 GDI객체를 SelectObject()함수로 선택하고, 이 함수의 리턴값인 이전의 GDI객체를 그리기 작업을 다한 후 DC를 원상 복구할 목적으로 보관
  - **설정**: SetBkColor()함수로 배경색을 설정하거나 Set으로 시작하는 함수들을 이용하여 다양한 설정
  - **그래픽 출력**: 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 함수  
`HPEN CreatePen ( int fnPenStyle, int nWidth, COLORREF crColor);`
- 그림을 그릴 화면인 DC에 펜 핸들을 등록 함수  
`HPEN SelectObject (HDC hdc, HPEN pen);`
- 그림 그리기를 마친 후 생성된 펜 핸들은 삭제 함수  
`BOOL DeleteObject (HPEN pen);`

# CreatePen

- **펜 만들기 함수**

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

- fnPenStyle: 펜 스타일
- nWidth: 펜의 굵기로 단위는 픽셀
- crColor: 색상을 표현하기 위해 COLORREF 값을 제공하며, RGB() 함수로 만들

PS\_SOLID

---

PS\_DASH

-----

PS\_DOT

.....

PS\_DASHDOT

\_\_\_\_\_

PS\_DASHDOTDOT

.....

- **COLORREF RGB (int Red, int Green, int Blue)**

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

COLORREF 는 색상을 표현하는 자료형 (R, G, B 3가지 색으로 표현)

# 빨간 점선으로 원 그리기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    HPEN hPen, oldPen;

    switch (iMsg) {
        case WM_PAINT :
            hdc = BeginPaint (hwnd, &ps) ; // DC 얻어오기

            hPen = CreatePen (PS_DOT, 1, RGB(255,0,0)); // GDI: 펜 만들기
            oldPen = (HPEN) SelectObject (hdc, hPen); // 새로운 펜 선택하기

            Ellipse(hdc, 20,20, 300,300); // 선택한 펜으로 도형 그리기

            SelectObject (hdc, oldPen); // 이전의 펜으로 돌아감
            DeleteObject (hPen); // 만든 펜 객체 삭제하기

            EndPaint (hwnd, &ps) ; // DC 해제하기
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



# 면 색상 변경

- 면의 색상정보를 가지는 브러시한들을 만들어 주는 함수

**HBRUSH CreateSolidBrush (COLORREF crColor);**

- crColor: 면의 색상 값

- 그림그릴 화면인 디바이스컨텍스트에 브러시한들을 등록

**HBRUSH SelectObject** (HDC hdc, HBRUSH brush);

- 그림 그리기를 마친 후 생성된 브러시한들은 삭제한다

**BOOL DeleteObject** (HBRUSH);

# 빨간면의 원 그리기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    HBRUSH hBrush, oldBrush;

    switch (iMsg) {
        case WM_PAINT :
            hdc = BeginPaint (hwnd, &ps) ; // DC 얻어오기

            hBrush = CreateSolidBrush (RGB(255,0,0)); // GDI: 브러시 만들기
            oldBrush = (HBRUSH) SelectObject (hdc, hBrush); // 새로운 브러시 선택하기

            Ellipse(hdc, 20,20, 300,300); // 선택한 브러시로 도형 그리기

            SelectObject (hdc, oldBrush); // 이전의 브러시로 돌아가기
            DeleteObject (hBrush); // 만든 브러시 객체 삭제하기

            EndPaint (hwnd, &ps) ; // DC 해제하기
            return 0 ;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# GDI 객체 핸들 구하기 함수들

- 독자적으로 생성한 펜, 브러시 및 폰트를 설정하는 함수

**HGDIOBJ SelectObject** (HDC hDC, HGDIOBJ hgdiobj);

- hDC: DC 핸들값
- Hgdiobj: GDI의 객체
- 리턴 값은 원래의 오브젝트 값

- 윈도우가 제공하는 펜 브러시 및 폰트를 취득하는 함수
  - 스톡 오브젝트: 윈도우에서 기본적으로 제공하는 GDI 객체
  - 윈도우가 제공해주므로 따로 생성하지 않고 사용할 수 있으며 해제하지 않아도 됨.

**HGDIOBJ GetStockObject** (int fnObject);

- 리턴 값은 가져올 스톡 오브젝트 핸들 값
- fnObject: 가져올 스톡 오브젝트의 속성
  - BLACK\_BRUSH / DKGRAY\_BRUSH / DC\_BRUSH / GRAY\_BRUSH / HOLLOW\_BRUSH / LTGRAY\_BRUSH / NULL\_BRUSH / WHITE\_BRUSH / BLACK\_PEN / DC\_PEN / WHITE\_PEN

- 클라이언트의 영역을 취득하는 함수

**BOOL GetClientRect** (HWND hWnd, LPRECT lprc);

- 클라이언트의 영역을 취득한다.
- hWnd: 윈도우 핸들
- lprc: 윈도우 영역의 좌표값

# GDI 객체 핸들하기

- 윈도우가 제공하는 회색 브러시를 사용하여 사각형을 그리는 경우 → 객체를 만들지 않고 윈도우가 제공하는 객체 (스톡 객체) 사용

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 → 파란색의 펜 객체를 만들어 사용

HPEN MyPen, OldPen;

MyPen = **CreatePen** (PS\_SOLID, 5, RGB(0, 0, 255));

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

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

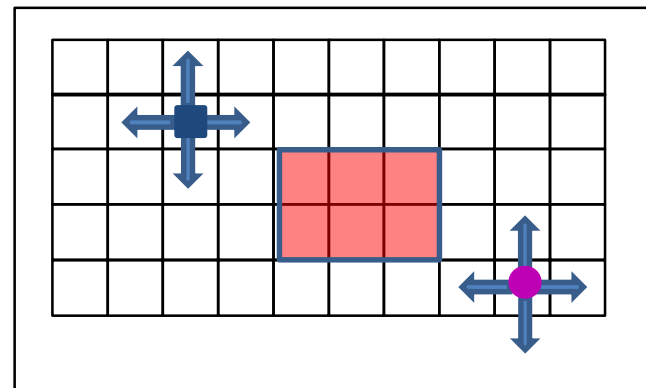
**SelectObject** (hdc, OldPen);

**DeleteObject** (MyPen);

## 실습 2-4

### • 돌 이동하기

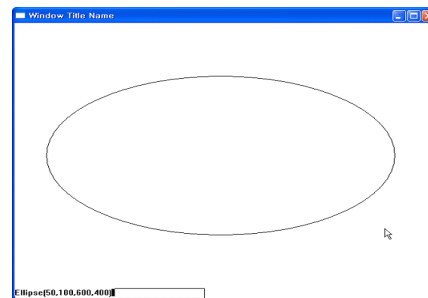
- 화면에 20x20 칸을 그린다.
- 칸의 중간 4x4 칸은 다른 색으로 표시되어 있다.
- 임의의 위치에 두 개의 다른 색을 가진 돌을 그린다.
  - 다른색으로 표시된 영역 이외의 부분에 돌이 생긴다.
- 2인 플레이로 각각 다른 키보드 명령어를 이용하여 자신의 돌을 칸에 맞추어 좌우상하 이동한다.
- 플레이는 한 명씩 번갈아 하도록 한다.
  - 같은 돌을 두 번 이동하려고 하면 에러 메시지 출력
- 돌이 다른 색으로 표시된 영역에 들어가면
  - 그 돌은 움직이지 못하고 새로운 위치에 다시 돌이 만들어 지고 그 돌을 이동한다.
- 키보드 명령
  - 2인의 돌 이동 키보드
  - r/R: 새롭게 시작
  - q/Q: 프로그램 종료



## 실습 2-5

### • 명령에 따라 그림을 그리는 프로그램

- 윈도우화면 아랫단 중앙에 문자열 한 줄 입력 받을 수 있는 글상자 역할을 할 사각형을 배치
- 사각형 내에는 Caret이 나타나서 명령어 입력을 기다림
- **명령어는 여섯 개의 숫자로:** 그릴 도형의 형태, 좌표 값 4개, 두께가 입력된다.
  - 도형의 형태: 1 - 직선, 2 - 원, 3 - 삼각형, 4 - 사각형
- 예를 들어,
  - 직선일 경우는 [1 10 10 200 150 2] => 직선을 (10, 10)에서 (200, 150)에 두께 2로 그린다.
  - 원일 경우는 [2 0 0 50 50 3] => 원을 (0, 0)과 (50, 50) 구간에 3 두께로 그린다.
  - 삼각형일 경우는 [3 0 0 100 200 2] => 삼각형을 (0, 0)과 (100, 200)에 2 두께로 그린다.
  - 사각형일 경우는 [4 0 0 100 200 1] => 사각형을 (0, 0)과 (100, 200)에 1 두께로 그린다.
  - 첫 번째 숫자는 도형의 종류, 2~5번째 숫자는 좌표값, 6번째 숫자는 두께
- **문자 명령어를 넣어서 그린 도형의 두께를 늘리거나 줄이도록 한다.**
  - **명령어 + / -** : 두께를 특정 두께까지 늘리거나 줄인다.
  - **화살표**: 도형을 좌우상하로 이동
  - 도형 테두리 또는 내부의 색깔을 랜덤하게 설정한다.
  - 화면에 도형이 없을 경우에는 **에러 메시지**를 출력한다.



- 1: 직선
- 2: 원
- 3: 삼각형
- 4: 사각형

