

# 제 2장 윈도우 기본 입출력

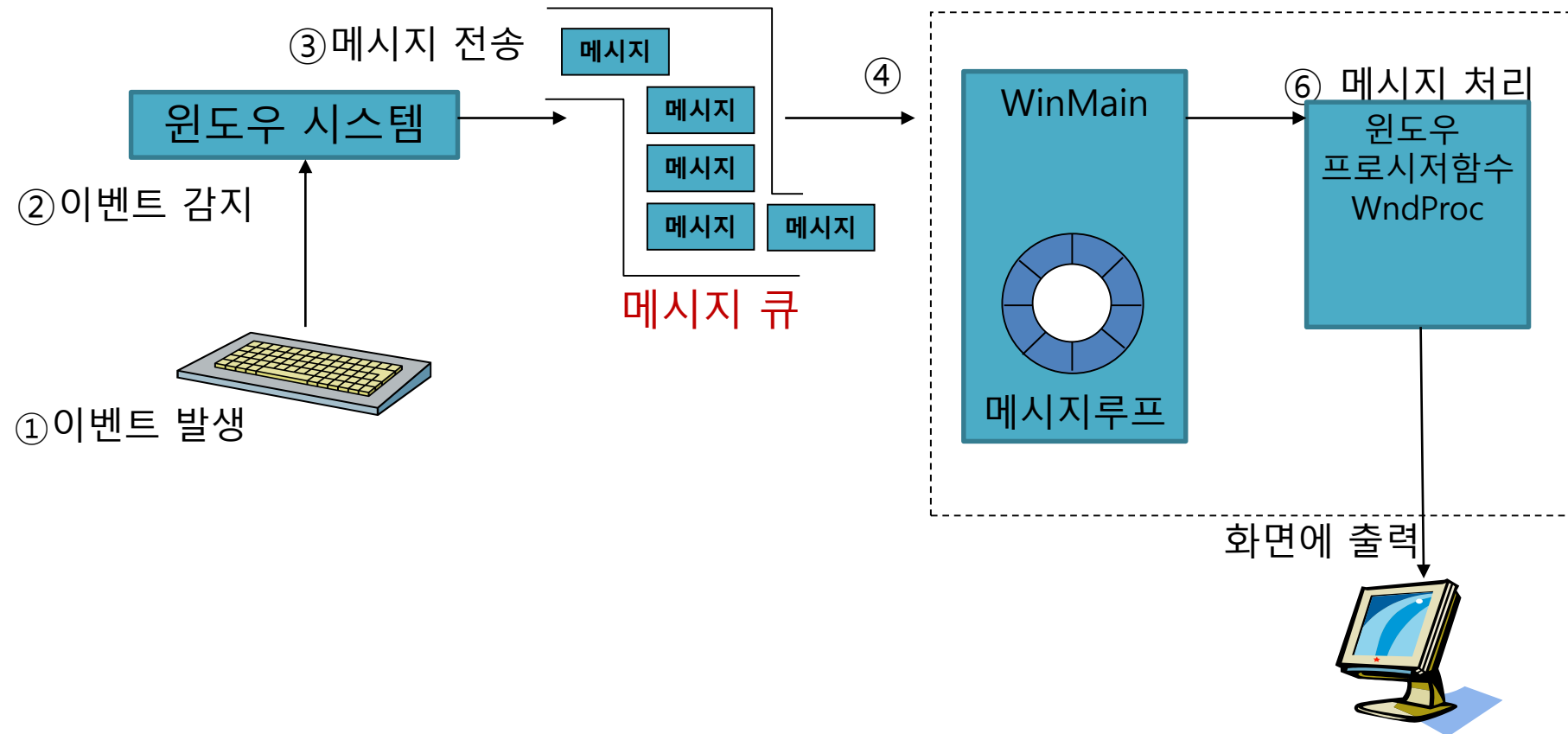
2022년 1학기 윈도우 프로그래밍

## 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
  - Caret 이용하기
  - 직선, 원, 사각형, 다각형 그리기

# 윈도우 프로그램 특징

- 메시지 기반의 프로그램 진행



- 윈도우 프로그램은 윈도우에서 발생하는 이벤트에 의한 메시지를 처리하며 프로그램이 진행된다.

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

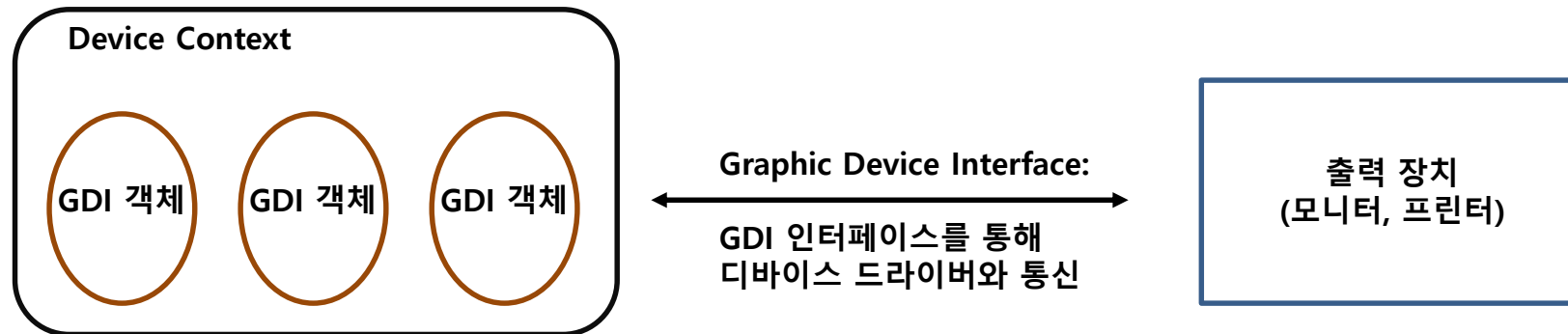
메시지	내용	메시지	내용
WM_CREATE	윈도우가 생성될 때 발생	WM_RBUTTONDOWN	마우스 오른쪽 버튼을 누르면 발생
WM_NCACTIVATE	윈도우의 비 작업영역의 활성화 또는 비 활성화시 발생 (윈도우 타이틀 바 색상 제어)	WM_RBUTTONUP	마우스 오른쪽 버튼을 떼면 발생
WM_DESTROY	윈도우가 파괴되기 직전에 발생	WM_MOUSEMOVE	마우스가 움직이고 있으면 발생
WM_PAINT	윈도우가 다시 그려져야 하면 발생	WM_SETCURSOR	마우스의 아이콘을 재설정해야 할 때 발생
WM_LBUTTONDOWN	마우스 왼쪽 버튼을 누르면 발생	WM_TIMER	타이머 설정 시 주기적으로 발생
WM_LBUTTONUP	마우스 왼쪽 버튼을 떼면 발생	WM_COMMAND	메뉴, 버튼, 액셀러레이터 선택 시 발생

- LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
  - hwnd: 메시지가 발생한 윈도우 핸들
  - iMsg: 발생한 메시지 (위의 메시지들이 전달된다)
  - wParam, lParam: 메시지를 처리하기 위해 필요한 데이터들, 메시지들마다 다른 값이 전달된다.

# 1. 출력 영역 얻기

- **GDI (Graphic Device Interface)**

- 화면, 프린터와 같은 모든 출력 장치를 제어하는 인터페이스
- 윈도우 프로그램에서의 모든 출력은 GDI를 통해서 화면과 프린터로 나가게 되어 있다.
- GDI 객체: 그래픽 출력에 사용되는 도구
  - GDI 객체 종류: 펜, 브러시, 비트맵, 폰트 등
  - GDI 객체들은 핸들을 이용해서 사용한다.
- DC (Device Context): GDI 오브젝트를 모아놓은 것
- GDI는 현재 DC에 선택되어 있는 GDI 오브젝트를 사용한다.



# 디바이스 컨텍스트: 출력 영역 얻기

- **DC (Device Context)**

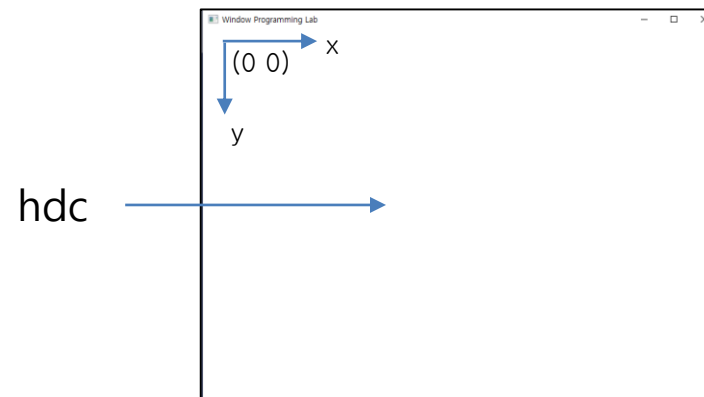
- 디스플레이 또는 프린터와 같은 출력에 필요한 정보를 포함하는 윈도우 데이터 구조체
  - 그래픽 관련한 선택 정보 (폰트, 색상, 굵기, 무늬, 출력 방법 등)를 모아 놓은 구조체
- 모든 그리기는 디바이스 컨텍스트 개체를 통해 수행된다.
  - 디바이스 컨텍스트는 GDI 모듈에 의해서 관리된다.
  - 간단한 출력은 디폴트로 설정된 속성 (예를 들어 선을 그린다면 선의 색상, 굵기, 모양 등) 이용
  - 속성은 얻어온 DC에서 변경 가능: 선의 형태, 색깔, 굵기, 모드, 위치 등의 속성 변경 가능

- 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 구분 번호로 생각
- 모든 GDI 함수들은 첫 번째 인자로 DC 핸들을 필요로 한다.

- DC의 유형

- 화면 출력을 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.

DC 얻어오는 함수	DC 해제하는 함수	기능
BeginPaint()	EndPaint():	WM_PAINT 메시지와 함께 사용
GetDC()	ReleaseDC():	WM_PAINT 메시지 외의 일반 메시지에서 DC를 얻어올 때 사용
CreateDC()	DeleteDC():	DC를 만들어 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

- 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 hWnd, HDC hDC);

- HWND hWnd: 해제할 DC에 대한 윈도우 핸들
- HDC hDC: 해제할 DC 핸들



# 디바이스 컨텍스트 얻어오기

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

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

- HWND hwnd: 생성된 윈도우의 핸들값
- PAINTSTRUCT \*lpPaint: 출력될 영역에 대한 정보를 저장한 구조체 공간에 대한 주소
- 리턴 값으로 디바이스 컨텍스트 핸들을 얻어온다

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

- HWND hwnd: 생성된 윈도우의 핸들값
- PAINTSTRUCT \*lpPaint: 출력될 영역에 대한 정보를 저장한 구조체 공간에 대한 주소

- PAINTSTRUCT 구조체

```
typedef struct tagPAINTSTRUCT {  
    HDC hdc;                //그리고자 하는 DC 핸들  
    BOOL fErase;            // 배경 삭제 여부를 가리킨다: 0이 아니면 다시 그린다.  
    RECT rcPaint;           // 다시 그릴 사각형의 좌표값  
    BOOL fRestore;  
    BOOL fIncUpdate;  
    BYTE rgbReserved[16];  
} PAINTSTRUCT;
```

# 디바이스 컨텍스트 얻어오기

- 사용 예)

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    ...
}

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

    switch (iMsg)
    {
        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps) ;
            EndPaint (hwnd, &ps) ;
            break;

        case WM_TIMER:
            hdc = GetDC (hwnd);
            ReleaseDC (hwnd, hdc);
            break;

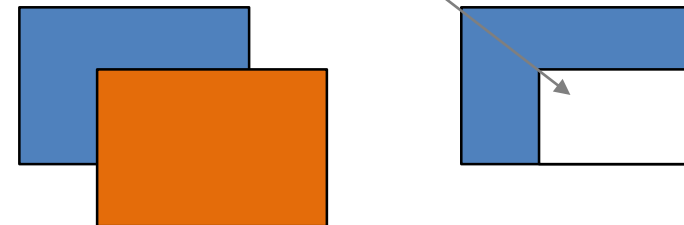
        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

//--- WM\_PAINT 메시지: dc를 얻어 그리기 진행  
//--- BeginPaint 함수를 사용하여 DC를 얻는다

//--- WM\_TIMER 메시지: dc를 얻어 출력 진행  
//--- GetDC 함수를 사용하여 DC를 얻는다

## 2. 메시지 처리: WM\_PAINT 메시지

- 클라이언트 영역에 그릴 필요가 있을 때: **WM\_PAINT** 메시지
  - 원도우의 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

- WM\_PAINT 메시지는
  - 이 메시지를 받았을 때 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.
- 해당 윈도우 프로시저에서 이 메시지를 처리하지 않으면 이 메시지는 DefWindowProc()함수가 처리한다.
  - 이 함수는 무효 영역을 모두 유효화(valid)하며 다시 그리기는 하지 않는다.
- WM\_PAINT메시지에서 그리기를 할 때는 BeginPaint()와 EndPaint()함수를 사용해야 한다.
  - 이 두 함수는 WM\_PAINT메시지 내에서만 사용된다.
  - 다시 그려야 할 영역에 대한 정확한 좌표를 조사하며 무효 영역을 유효화하고 캐릿을 숨기거나 배경을 지우는 등의 꼭 필요한 동작을 한다.
- 다른 메시지에서도 출력은 가능하다!

### 3. 메시지 처리: WM\_CREATE / WM\_QUIT 메시지

- 윈도우가 생성될 때: **WM\_CREATE** 메시지
  - CreateWindow 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.
  - CreateWindow 함수는 이 메시지를 완전히 처리한 후에 리턴
- 윈도우를 파괴할 때: **WM\_DESTROY** 메시지
- 프로그램을 종료할 때: **WM\_QUIT** 메시지
  - 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.
  - **PostQuitMessage ()** 함수를 호출하여 프로그램을 종료한다.
    - 파괴되는 윈도우가 메인 윈도우일 경우에는 반드시 PostQuitMessage 함수를 호출하여 메시지 루프를 종료하도록 한다.
  - 윈도우를 닫는 함수
    - **BOOL DestroyWindow (HWND hWnd);**
      - hWnd 윈도우를 파괴한다. 단, 이 함수로 다른 스레드에서 생성한 윈도우를 파괴할 수는 없다

### 3. 메시지 처리: WM\_CREATE / WM\_QUIT 메시지

- 프로그램 종료

**VOID PostQuitMessage (int nExitCode);**

- 스레드 메시지 큐에 WM\_QUIT 메시지를 붙이고 즉시 리턴
- WM\_QUIT 메시지를 큐에 붙임으로써 시스템에게 이 스레드가 종료될 것이라는 것을 미리 알려준다.
- 메시지 루프는 보통 WM\_QUIT 메시지를 받으면 종료하도록 되어 있으므로 이 함수를 호출하면 메시지 루프가 종료된다.

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

- 사용 예)

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

```
{
    HDC hdc;
    PAINTSTRUCT ps;

    switch (iMsg)
    {
        case WM_CREATE:                                     //--- WM_CREATE 메시지: 윈도우 생성될 때 호출, 필요한 초기화 작업
            break;

        case WM_PAINT:                                       //--- WM_PAINT 메시지: dc를 얻어 그리기 진행
            hdc = BeginPaint (hwnd, &ps) ;                  //--- 필요한 그리기를 실행한다.

            EndPaint (hwnd, &ps) ;

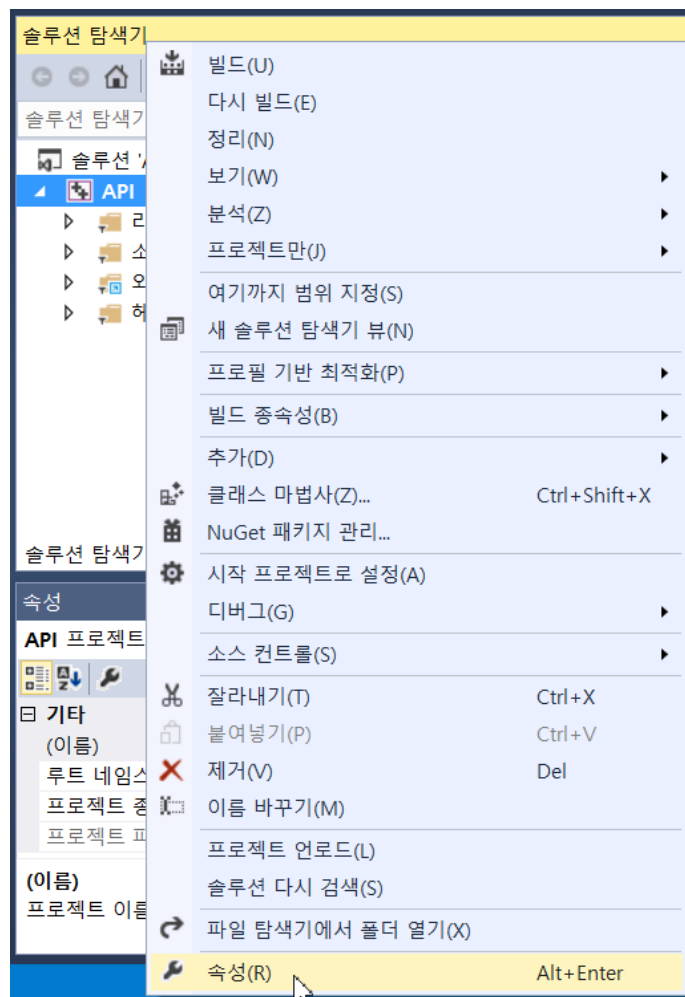
            break;

        case WM_DESTROY:                                     //--- 프로그램 종료
            PostQuitMessage ( 0 );

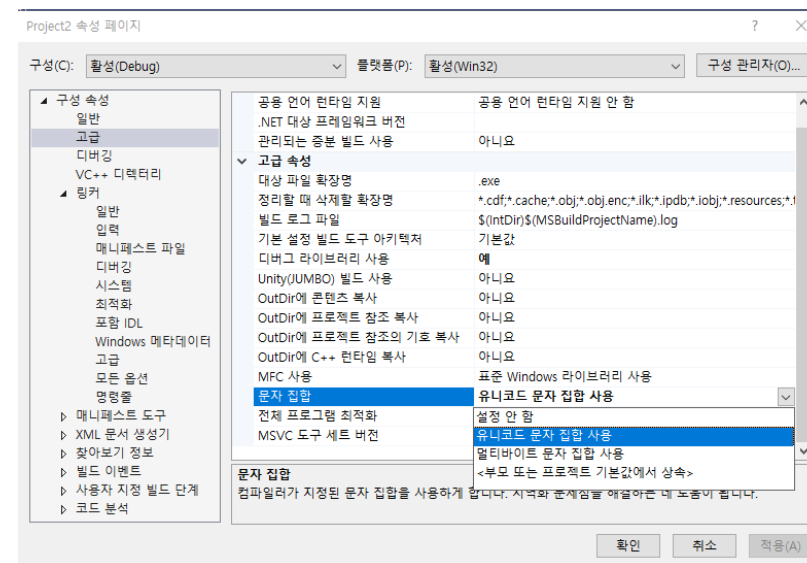
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# 4. 문자 집합

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



- 기본 설정은 유니코드
  - 고급 → 문자집합 → 유니코드/멀티바이트





# 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_1[15] = _T("나는 winple");
```

```
TCHAR str_2[15] = L"나는 winple";
```

```
TCHAR str_3[15] = TEXT("나는 winple");
```

```
// 위의 세개의 매크로 함수 모두 사용 가능
```

# 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

## 5. 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

- 문자열 관련 함수들:

	멀티바이트	TCHAR	유니코드	사용 예
문자열 복사	strcpy	_tcscopy	wcscopy	strcpy(a, b)
문자열을 c만큼 복사	strncpy	_tcsncpy	wcsncpy	strncpy (a, b, c)
문자열 길이	strlen	_tcslen	wcslen	strlen(a)
문자열 붙이기	strcat	_tcscat	wcscat	strcat (a, b)

# 윈도우에 “Hello World” 출력하기

- 원도우 프로시저 함수 안에서 WM\_PAINT 메시지에서 출력 처리

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    ...
}
```

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    TCHAR str[100] = L"Second Hello World";
    int num;

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

            num = wcslen(str);
            TextOut (hdc, 0, 0, L"Hello World", strlen("Hello World"));

            TextOut (hdc, 0, 100, str, num);
            EndPaint (hwnd, &ps);
            break;

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



# 박스 영역에 텍스트 출력 함수: DrawText ()

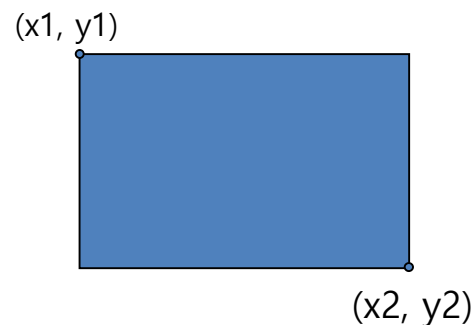
- 박스 영역에 텍스트 출력 함수

```
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 (세로에 해당하는 플래그 사용 시 함께 사용)
  - 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\_SINGLELINE 과 함께 사용)
  - DT\_TOP: 박스 영역의 상하에서 위쪽에 출력
  - DT\_BOTTOM: 박스 영역의 상하에서 아래쪽에 출력
  - DT\_WORDBREAK: 단어가 박스 영역의 오른쪽 끝에 닿으면 자동 개행되도록 한다.
- **사용 예) 1개 이상의 설정을 하는 경우: 사각형 영역의 가운데 출력**

```
TCHAR szText[] = _T("Hello world");  
RECT rect = {0, 0, 800, 600};  
  
DrawText (hdc, szText, _tcslen(szText), &rect, DT_VCENTER | DT_CENTER | DT_SINGLELINE);
```
- **사용 예) 긴 문자열을 사각 영역에 맞추어 출력하는 경우**

```
TCHAR szText[] = _T("HelloWorld HelloWorld HelloWorld HelloWorld HelloWorld");  
RECT rect = {0, 0, 20, 600};  
  
DrawText (hdc, szText, _tcslen(szText), &rect, DT_WORDBREAK); //
```



# DrawText() 함수 이용하기

- DrawText 함수를 이용하여 HelloWorld 출력

```
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, L"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);
}
```



//--- 한 라인, 수직/수평 중앙

# <문자열 만들기>

- 문자열 만들기

- int **wsprintf** (LPTSTR lpOut, LPCTSTR lpFmt...);
  - 서식화된 문자열을 버퍼에 저장한다.
- 사용 예)

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    TCHAR lpOut[100];
    int x=0, y=0;

    switch (iMsg)
    {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            wsprintf (lpOut, L"%d * %d = %d", x, y, x * y);
            TextOut (hdc, x, y, lpOut, lstrlen (lpOut));
            EndPaint (hwnd, &ps);
            break;

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

# 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

- 사용 예)  
앞 페이지의 예에서

```
SetTextColor (hdc, RGB (255, 0, 0));           //--- 문자 출력 이전에 문자 색상 설정, 설정 이후의 문자는 빨간색으로 출력된다.  
DrawText (hdc, L"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:RGB 색상을 지정하는데 사용되는 DWORD 형태의 타입
  - 16진수 형태
  - 0x00bbggrr의 값으로 저장됨
  - r, g, b 값을 얻으려면 각각 GetRValue, GetGValue, GetBValue 함수를 사용
- RGB 매크로 함수는 빨강, 초록, 파랑 색의 값을 인자로 받아 COLORREF 타입의 색상값으로 만든다.

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

- 사용 예)

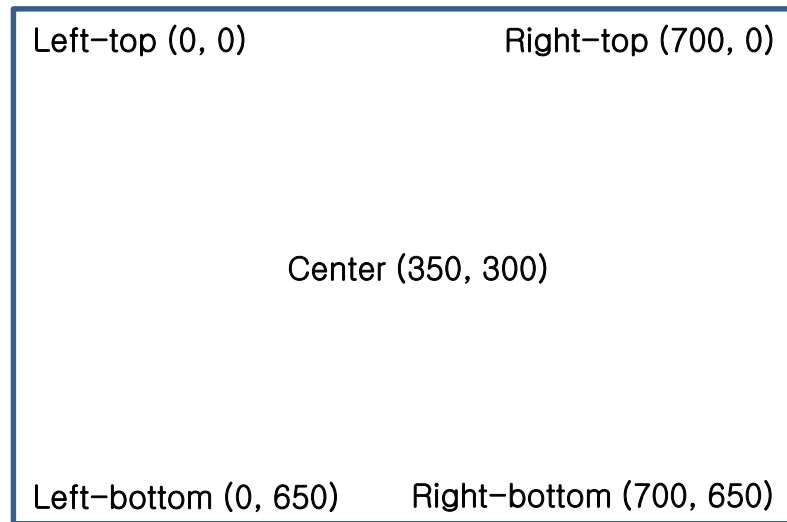
```
COLORREF text_color ;  
text_color = RGB (255, 0, 0);
```

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

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

## 실습 2-1

- 화면의 코너와 중앙에 문자 그리기
  - 윈도우를 띄운다
  - 윈도우의 네 코너와 중앙에 문자를 그린다.
  - 문자 내용은 각각 "Left-top (0, 0)", "Right-top (700, 0)", "Left-bottom (0, 650)", "Right-bottom(700, 650)", "Center(350, 300)"를 출력한다.
    - (0, 0), (700, 0), (0, 650), (700, 650), (350, 300): 각각의 문자가 시작하는 좌표값 (x, y)



## 실습 2-2

- 화면을 등분하여 문자 그리기
  - 윈도우를 띄운다.
  - 화면의 가로 세로를 각각 2등분하여 4개의 구간으로 만든 후 각 구간에 문자 그리기
  - 각 구간의 문자색과 배경색은 랜덤한 색으로 설정한다.

abcdefghijklmn opqrstuvwxyz abcdefghijklmno pqrstuvwxyzab	ABCDEFGHIJKLMN OPQRSTUVWXYZ VWXYZABCDE FGHIJKLMNOP
ABCDEFGHIJKLMN OPQRSTUVWXYZ VWXYZABCDE FGHIJKLMNOP	abcdefghijklmn opqrstuvwxyz abcdefghijklmno pqrstuvwxyzab

## 6. 키보드 메시지 처리하기

- 키보드로 입력한 정보 받기

메시지	내용	윈도우 프로시저 인수값	실행 형태
WM_KEYDOWN	키보드에서 키를 눌렀을 때 발생하는 메시지	<ul style="list-style-type: none"><li><b>wParam</b>: 가상키코드값</li><li><b>lParam</b>: 키보드 상태에 대한 정보들 (반복 실행 횟수, 검사 코드, 확장 키 플래그, 컨텍스트 코드, 이전 키-상태 플래그 및 전환 상태 플래그)</li></ul>	'a'키를 눌렀을 때: WM_KEYDOWN 메시지 발생하고, 다음으로 WM_CHAR 메시지 발생
WM_KEYUP	키보드에서 키를 눌렀다가 떼면 발생하는 메시지		
WM_CHAR	문자 키를 눌렀을 때 발생하는 메시지	<ul style="list-style-type: none"><li><b>wParam</b>: 입력된 문자값</li><li><b>lParam</b>: 키보드 상태에 대한 정보들 (반복 실행 횟수, 검사 코드, 확장 키 플래그, 컨텍스트 코드, 이전 키-상태 플래그 및 전환 상태 플래그)</li></ul>	

## 가상키 코드 테이블

- WM\_KEYDOWN 메시지에서 윈도우로 전달되는 문자가 아닌 키 값 : wParam로 전달된다.
- ALT 키, 윈도우 키, 한영 전환키 등의 특수 키 몇 가지는 제외된다.

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 ~ VK_F10	F1-F10
VK_SPACE	Space	VK_NUMLOCK	Num Lock
VK_PRIOR	Page Up	VK_SCROLL	Scroll Lock
VK_NEXT	Page Down		



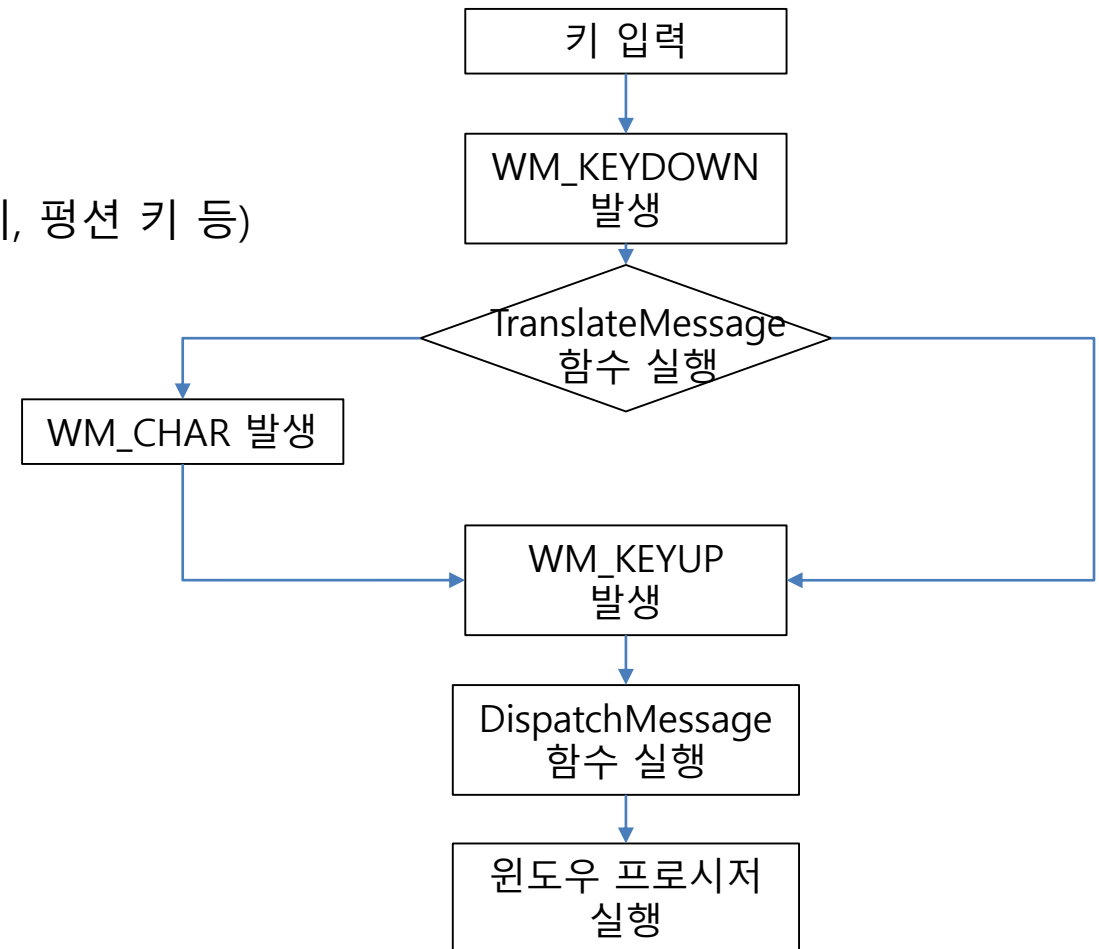
# 메시지 처리: 키보드 눌렀을 때 발생하는 메시지

- 키보드를 눌렀을 때

- WM\_KEYDOWN 메시지 발생 → 문자인 경우 WM\_CHAR 메시지 발생
- WM\_KEYDOWN 메시지: 단순히 키가 눌림/안눌림을 판단, 대소문자 구분없음
- WM\_CHAR 메시지 : 입력된 문자에 대한 처리

- 메시지 사용

- WM\_CHAR: 문자 키 일 때 사용
- WM\_KEYDOWN: 문자 이외의 키 (home, 이동키, 페이지 키, 평션 키 등)



# WM\_CHAR: 입력 문자 처리하기

- 키보드로 문자 입력:
  - 키보드로 문자를 입력하고 str[0]에 저장한다.
  - 저장한 문자를 화면에 출력
- 사용 예)

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

```
{
```

```
    HDC hdc;
```

```
    TCHAR str[100];
```

```
    switch (iMsg) {
```

```
        case WM_CHAR:
```

```
            hdc = GetDC(hwnd);
```

```
            str[0] = wParam;
```

```
            str[1] = '\0';
```

```
            TextOut (hdc, 0, 0, str, lstrlen(str));
```

```
            ReleaseDC (hwnd, hdc);
```

```
            break;
```

```
    }
```

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

```
}
```

```
//--- GetDC 함수를 사용하여 dc를 얻어옴
```

```
//--- 입력문자 : WinProc의 매개변수 wParam로 들어옴
```

```
//--- 문자열은 null('\0')로 끝남
```

# WM\_CHAR: 입력 문자열 처리하기

- 키보드로 문자 입력:
  - 입력한 문자를 순서대로 str에 저장하고 문자열로 만든다
  - 그 문자열을 화면에 출력
- 사용 예)

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

```
{
    HDC hdc;
    static TCHAR str[100];
    static int count;

    switch (iMsg) {
        case WM_CREATE :
            count = 0;
            break ;

        case WM_CHAR:
            hdc = GetDC(hwnd);
            str[count++] = wParam;
            str[count] = '\0';
            TextOut (hdc, 0, 0, str, lstrlen(str));
            ReleaseDC (hwnd,hdc);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

//--- GetDC 함수를 사용하여 DC를 얻어옴  
//--- 문자저장 후 인덱스 증가  
//--- 문자열은 null('\0')로 끝남

# WM\_KEYDOWN 메시지 처리하기

- 키를 누르면 문자열을 출력
- 사용 예)

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

```
{
```

```
    HDC hdc;
```

```
    PAINTSTRUCT ps;
```

```
    switch (iMsg ) {
```

```
        case WM_KEYDOWN:
```

```
            hdc = GetDC (hwnd);
```

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

```
            ReleaseDC (hwnd, hdc);
```

```
            break;
```

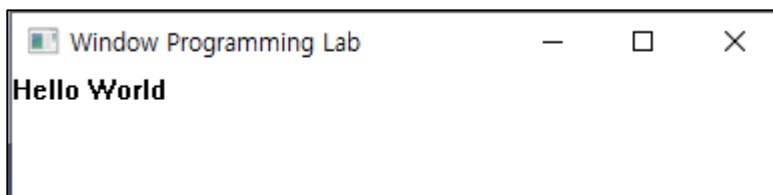
```
    }
```

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

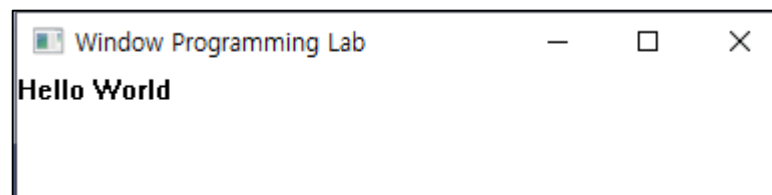
```
}
```

```
//--- 키가 눌렸을 때
```

```
//--- (0, 0) 위치에 "HelloWorld" 문자열을 출력
```



[a]를 누른결과



[Enter]를 누른결과

눌린 키에 관계없이 결과 출력

# WM\_CHAR 예제

- 키보드로 문자 입력:
  - 백스페이스 키와 엔터 키 입력 예제

- 사용 예 1)

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

```
{
    HDC hdc;
    static int count = 0;
    static TCHAR str[80];

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

        case WM_CHAR:
            hdc = GetDC (hwnd);
            if (wParam == VK_BACK)                             //--- 백스페이스: 마지막 문자열 삭제
                count--;

            else                                                //--- 그 외에는 문자를 문자열 뒤에 붙인다.
                str[count++] = wParam;

            str[count] = '\0';
            TextOut (hdc, 0, 0, str, lstrlen(str));           //--- 마지막 문자로 널 추가
            ReleaseDC (hwnd, hdc);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

# WM\_CHAR 예제

- 사용 예 2)

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

```
{
```

```
    HDC hdc;
```

```
    static int count = 0;
```

```
    static TCHAR str[80];
```

```
    static int yPos=0;
```

```
    switch (iMsg) {
```

```
        case WM_PAINT:
```

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

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

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

```
        break;
```

```
        case WM_CHAR :
```

```
            hdc = GetDC (hwnd);
```

```
            if (wParam == VK_BACK)
```

```
                count--;
```

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

```
            {
```

```
                count = 0;
```

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

```
            }
```

```
            else
```

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

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

```
            ReleaseDC (hwnd, hdc);
```

```
        break;
```

```
    }
```

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

```
}
```

```
//--- 백스페이스를 입력하면
```

```
//--- 한 칸 삭제
```

```
//--- 엔터키를 입력하면: 문자열을 다음줄에 출력
```

```
//--- 인덱스 변경
```

```
//--- y 위치 변경: 한 줄 아래에 출력
```

```
//--- 그 외에는 문자를 문자열 맨 뒤에 붙인다.
```

# 문자열 출력: WM\_PAINT 메시지 처리하기

- 코드가 중복되는 문제점 발생

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

```
{  
    HDC hdc;  
    static TCHAR str[100];  
    static int count = 0;  
  
    switch (iMsg) {  
        case WM_CHAR:                                     //--- 1차적으로 문자열을 출력  
            hdc = GetDC(hwnd);  
            str[count++] = wParam;  
            str[count] = '\0';  
            TextOut (hdc, 0, 0, str, lstrlen(str));        //=== 중복  
            ReleaseDC(hwnd, hdc);  
            break;  
  
        case WM_PAINT:                                     //--- 화면이 가렸다 지워지면 다시 문자열을 출력  
            hdc = BeginPaint(hwnd, &ps);  
            TextOut (hdc, 0, 0, str, lstrlen(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 무효화

- 사용자가 화면의 내용을 변경하여 다시 그리기가 필요할 때 이 함수를 호출한다.
- 즉, 윈도우의 업데이트를 하게 된다.



# 문자 저장과 출력 구분하기

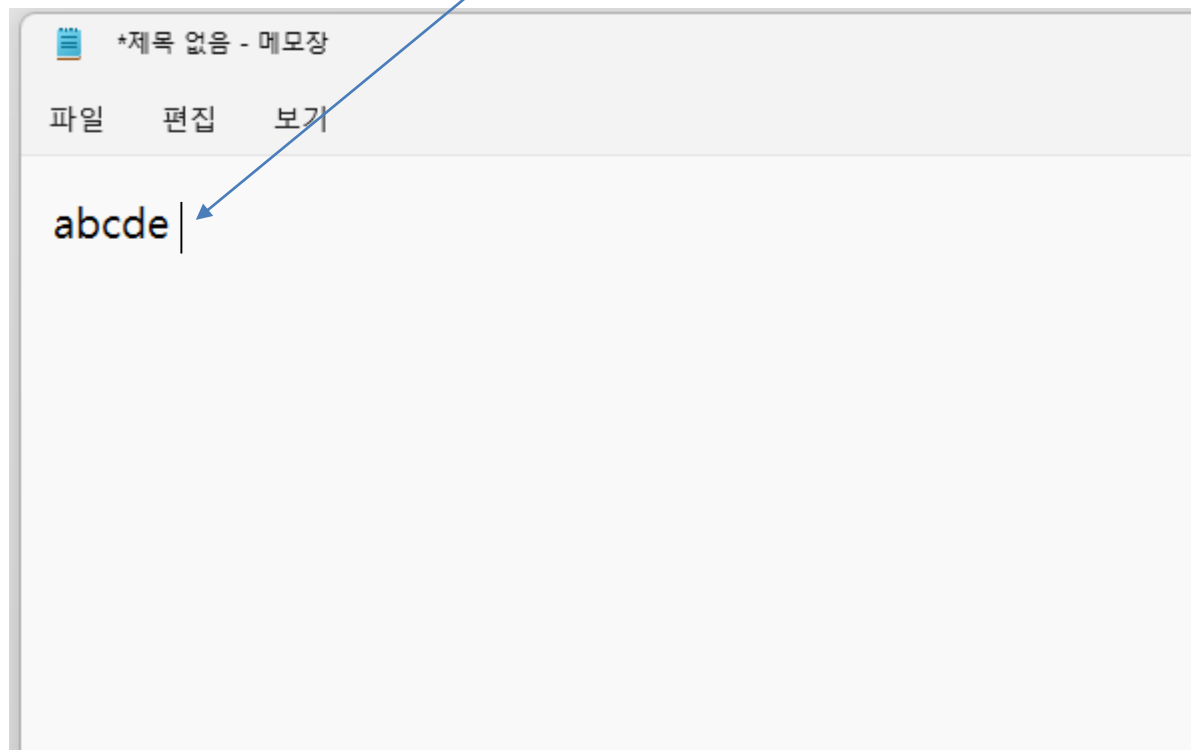
```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static TCHAR str[100];
    static int count;

    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, lstrlen(str)); //--- 문자 출력
            EndPaint(hwnd, &ps);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

## 7. Caret(캐럿) 사용하기

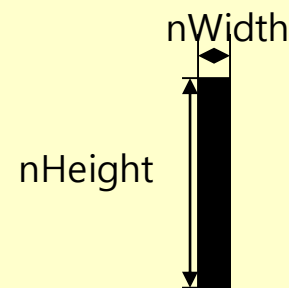
- Caret (캐럿):
  - 메모장이나 워드프로세서에서 키보드 입력 시 글자를 입력할 위치에 깜박거리는 커서
  - 캐럿이 있으면 어느 위치에 문자가 입력되는지를 알 수 있다.
  - 키보드 포커스는 오직 하나의 프로그램만이 가질 수 있고, 캐럿도 시스템에 하나만 존재.
  - 리소스로 사용하지 않고 API 함수를 호출하여 캐럿을 만들어서 원하는 위치에 붙일 수 있다.



# Caret 이용하기

## • 캐럿 관련 함수들

- **BOOL CreateCaret** (HWND hwnd, HBITMAP hBitmap, int nWidth, int nHeight);
  - 캐럿 만들기 함수
  - HWND hwnd: 캐럿을 놓을 윈도우 핸들
  - HBITMAP hbitmap: 비트맵 캐럿
  - int nWidth, nHeight: 캐럿의 폭과 높이
- **BOOL ShowCaret** (HWND hwnd);
  - 캐럿 출력하기
  - HWND hwnd: 캐럿이 출력될 윈도우 핸들
- **BOOL SetCaretPos** (int x, int y);
  - 캐럿 위치 설정하기
  - int x, int y: 캐럿의 x, y 위치
- **BOOL GetCaretPos** (LPPPOINT lpPoint);
  - 캐럿의 위치를 조사하기
  - LPPPOINT lpPoint: 캐럿의 위치를 가져온다.
- **BOOL SetCaretBlinkTime** (UINT uMSeconds);
  - 캐럿의 깜빡임 속도를 설정
  - UINT uMSeconds: 밀리세컨 단위의 깜빡임 속도
- **BOOL HideCaret** (HWND hwnd);
  - 캐럿 감추기
  - HWND hwnd: 캐럿이 놓여있는 윈도우 핸들
- **BOOL DestroyCaret** ();
  - 캐럿 삭제하기



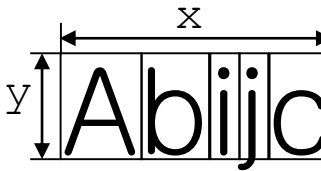
# Caret 위치 정하기

- 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.

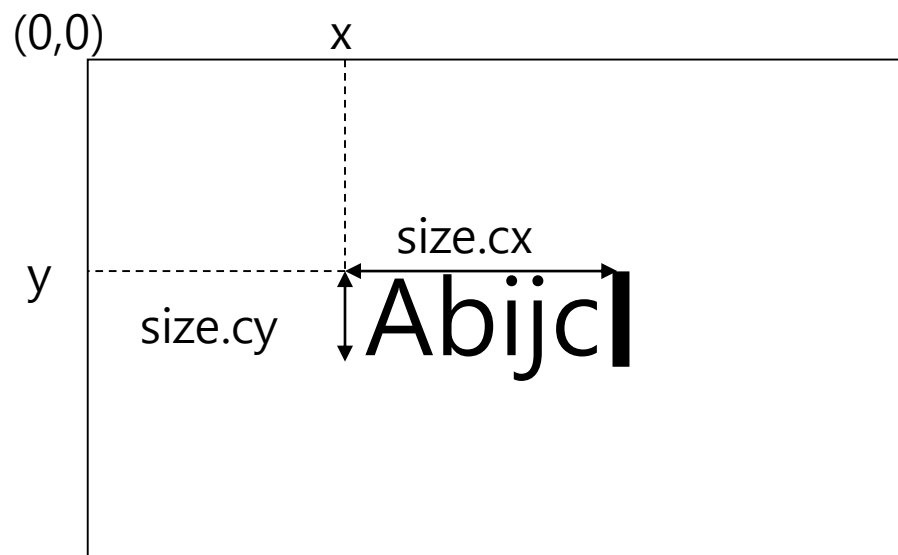
```
BOOL GetTextExtentPoint32 (HDC hdc, LPCTSTR lpString, int cbString, LPSIZE lpSize );
```

- 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

# 출력될 문자열 폭 구하기

- 크기 저장 구조체 사용

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



```
SIZE size;
```

```
GetTextExtentPoint32 (hdc, "Abijc", 5, &size);
```

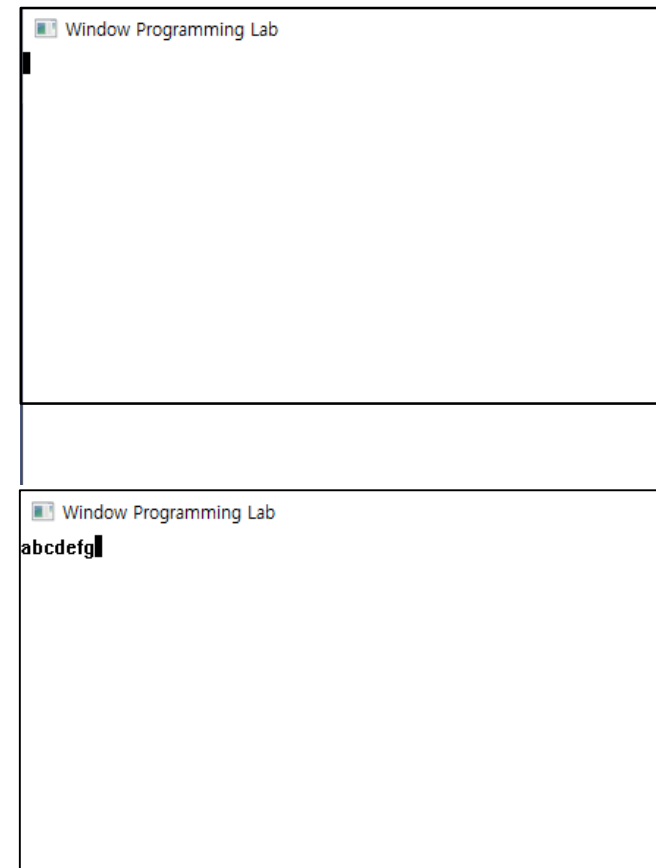
```
TextOut (hdc, x, y, "Abijc", 5);
```

```
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;
    static TCHAR str[100];
    static int count;           //--- 문자열의 인덱스로 사용할 값

    switch (iMsg) {
        case WM_CREATE :
            CreateCaret (hwnd, NULL, 5, 15);           //--- 캐럿 만들기
            ShowCaret (hwnd);                          //--- 빈 화면에 캐럿 표시
            count = 0;
            break;
        case WM_CHAR:
            str[count++] = wParam;                     //--- 문자저장 후 인덱스 증가
            str[count] = '\0';                          //--- 문자열은 null('\0')로 끝남
            InvalidateRect (hwnd, NULL, TRUE);          //--- WM_PAINT 메시지 발생
            break;
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            GetTextExtentPoint32 (hdc, str, lstrlen(str), &size);           //--- 문자열 길이 알아내기
            TextOut(hdc,0,0,str,lstrlen(str));
            SetCaretPos (size.cx, 0);                  //--- 캐럿 위치하기
            EndPaint(hwnd, &ps);
            break;
        case WM_DESTROY :
            HideCaret (hwnd);                          //--- 캐럿 숨기기
            DestroyCaret ();                            //--- 캐럿 삭제하기
            PostQuitMessage (0) ;
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

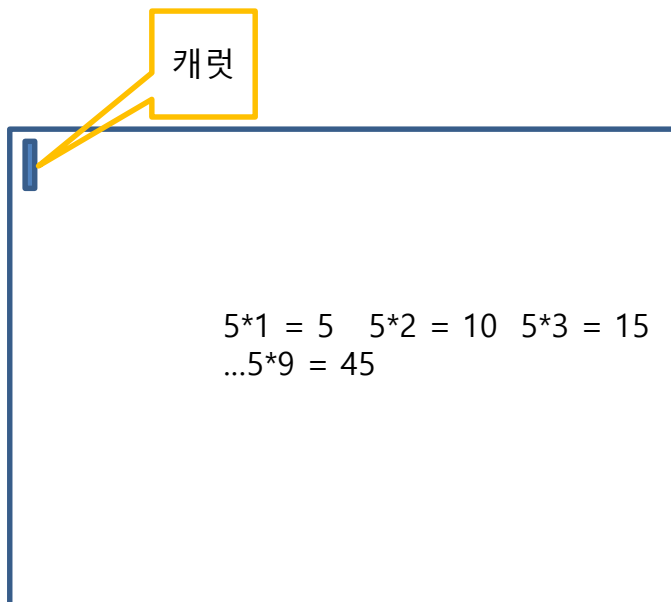
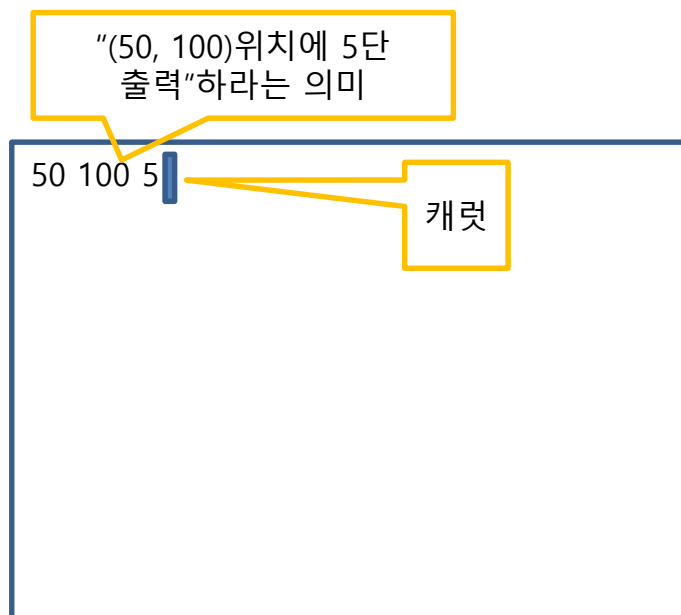


- 화면에 문자 그리기

- 47

### • 키보드 입력하여 구구단 출력하기

- 화면을 띄운다.
- 좌측 상단에 명령어를 받는다. 캐럿을 붙여서 어디에 입력하는지 알 수 있도록 한다.
  - X: x축 좌표값
  - Y: y축 좌표값
  - N: 단 수 (19단까지 입력받도록 한다.)
- 입력한 위치 (X, Y)에 단수(N)의 구구단을 길게 출력한다. 한 줄을 최대 40줄로 설정한다.
- 출력 후 명령어를 다시 받을 수 있도록 한다.
- 0을 입력하면 프로그램을 종료한다.





# 이번 시간 학습내용

- 이번주 학습 내용
  - GDI (Graphic Device Interface)
  - DC (Device Context)
  - DC를 얻어오는 함수들
  - 메시지:
    - WM\_PAINT
    - WM\_CREATE
    - WM\_QUIT
  - 문자 출력함수
    - `BOOL TextOut (HDC hdc, int x, int y, LPCTSTR lpString, int nLength);`
    - `int DrawText ( HDC hdc, LPCSTR lpString, int nLength, LPRECT lpRect, UINT Flags);`
  - 문자 입력 메시지
    - WM\_CHAR, WM\_KEYDOWN, WM\_KEYUP
  - WM\_PAINT 메시지 발생 함수
    - `BOOL InvalidateRect (HWND hWnd, const RECT* lpRect, BOOL bErase );`
    - `BOOL InvalidateRgn (HWND hWnd, HRGN hRgn, BOOL bErase );`
  - 캐럿 만들기