4장
메시지 처리 유형

김성영교수
금오공과대학교
컴퓨터공학부

메시지 처리 유형

- 윈도우에서 발생하는 이벤트에 대응하는 윈도우 메시지를 처리
  - 2장 및 3장의 실습예제
- 자식 윈도우에서 발생한 이벤트를 부모 윈도우에서 처리
  - WM_COMMAND 메시지 처리
  - 실습 4.2
- 자식 윈도우에서 발생한 사건 유형을 구분하여 처리
  - Notification code 처리
  - 실습 4.7
- Notification code의 한계를 극복하여 처리
  - 실습 4.9
- SendMessage() 혹은 PostMessage()를 통한 메시지 처리
  - 실습 4.11

실습 4.1

- 버튼 윈도우를 포함한 프로그램을 작성하자.
  - Windows 운영체제가 미리 만들어둔 윈도우 클래스를 사용하자!!


```
LRESULT CALLBACK WndProc( . . . )
{
    static HWND hBtn1;
    switch( msg )
    {
    case WM_CREATE:
        hBtn1 = CreateWindow( "BUTTON", "click",
                        WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
                        0, 0, 100, 100,
                        hwnd, NULL, _hInstance, NULL
                    );
        break;
    . . .
}
```

메시지 처리 유형 : 두 번째

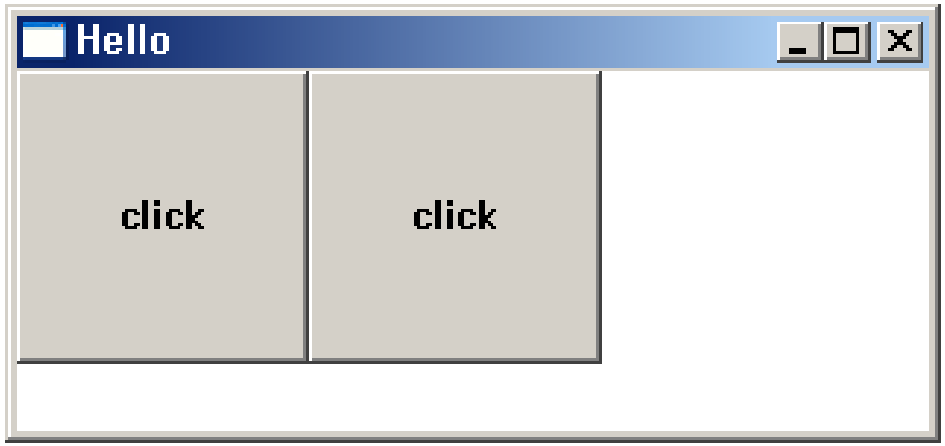
- 컨트롤 윈도우에서 발생한 이벤트를 부모 윈도우에서 처리
  - WM_COMMAND 메시지 처리
  - 컨트롤 윈도우
    - Windows 운영체제가 정의해놓은 윈도우

실습 4.2 – 유형 II

- 버튼을 클릭하면 메시지 박스를 출력하자.
  - WM_COMMAND 메시지를 사용하자!!

실습 4.3

- 두 개의 버튼을 추가하자.
  - 각 버튼에 대해 서로 다른 메시지 박스를 출력하자!!



두 개 이상의 컨트롤 윈도우 구분 (1)

- WM_COMMAND 메시지의 부가정보를 통해 구분

- WM_COMMAND 메시지가 발생하는 경우
  - 메뉴에서 메뉴항목을 선택했을 때
  - 단축키가 눌렸을 때
  - 컨트롤 윈도우에서 부모 윈도우에게 notification 코드를 보낼 때

http://msdn.microsoft.com/ko-kr/library/ms647591(en-us,VS.85).aspx

두 개 이상의 컨트롤 윈도우 구분 (2)

- WM_COMMAND 부가정보
  - `int wNotifyCode = HIWORD( wParam );`
  - `int wID = LWORD( wParam );`
  - `HWND hwndCtl = (HWND)lParam;`

Message Source	wParam (high word)	wParam (low word)	lParam
메뉴	0	Menu identifier (IDM_*)	0
단축키	1	Accelerator identifier (IDM_*)	0
컨트롤	**Control-defined notification code**	**Control identifier**	**Handle to the control window**

http://msdn.microsoft.com/ko-kr/library/ms647591(en-us,VS.85).aspx

두 개 이상의 컨트롤 윈도우 구분 (2)

- 자식 윈도우의 식별자(identifier) 설정

HWND CreateWindow(…, HMENU *hMenu*, …)

■ 9번째 인자 (hMenu)
- WS_POPUP 또는 WS_OVERLAPPED 스타일 윈도우: 메뉴 핸들
- 정수값 부여: 자식 윈도우의 식별자

```
hBtn1 = CreateWindow(
    "BUTTON",
    "click",
    WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
    0, 0, 100, 100,
    hwnd, (HMENU)888, _hInstance, NULL
);
```

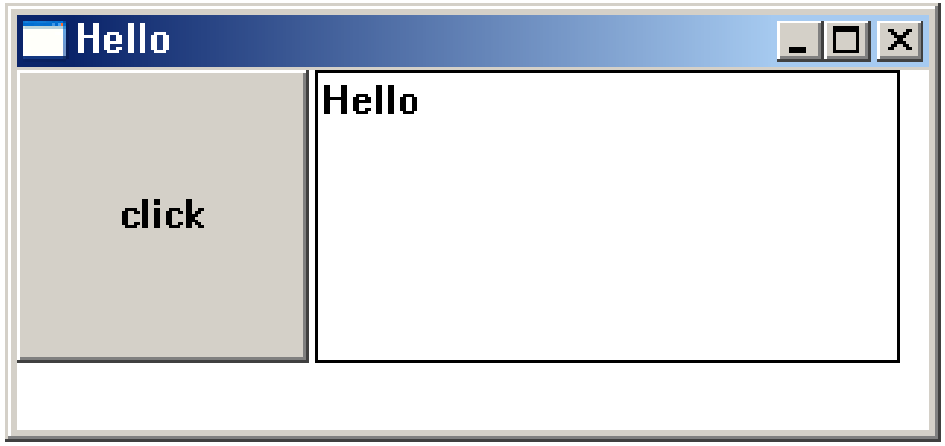
컨트롤 윈도우의 종류

스타일	의미
BUTTON	**버튼 윈도우**
COMBOBOX	콤보박스 윈도우
EDIT	**에디트 윈도우 (키보드 입력을 통해 문자열 편집이 가능)**
LISTBOX	리스트박스 윈도우 (문자열 목록을 가지며 선택된 문자열 표시)
RichEdit	리치에디트 윈도우 (에디트 윈도우 보다 풍부한 편집기능 보유)
SCROLLBAR	스크롤바 윈도우
STATIC	**스태틱 윈도우 (간단한 문자열이나 사각형 출력 용도)**

http://msdn.microsoft.com/en-us/library/bb773169(v=VS.85).aspx

실습 4.4

- 버튼과 에디트 윈도우를 추가하자.
  - 참고
    - 에디트 윈도우는 WS_BORDER 스타일 사용



에디트 윈도우

- 문자열 편집에 필요한 다양한 키(문자, 방향키 등)를 사용 가능

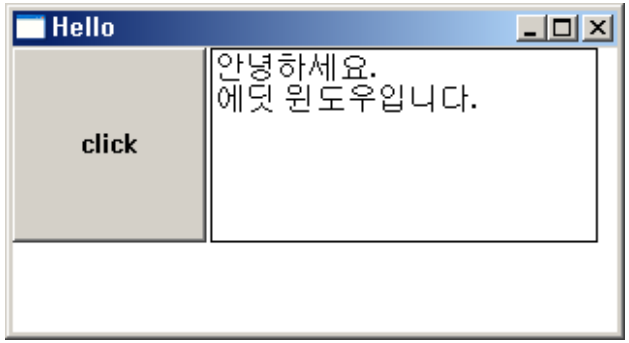
스타일	의미
ES_AUTOHSCROLL	라인 끝에 문자를 입력했을 때 10문자씩 자동으로 오른쪽 스크롤
ES_AUTOVSCROLL	마지막 라인에서 엔터키를 눌렀을 때 자동으로 아래로 스크롤
ES_MULTILINE	여러 라인을 가질 수 있도록 설정
ES_NUMBER	숫자만 입력되도록 함
ES_PASSWORD	입력 문자열을 모두 '*' 로 표시, 비밀번호 입력에 사용
ES_READONLY	입력이 불가능하고 읽기만 가능하게 설정

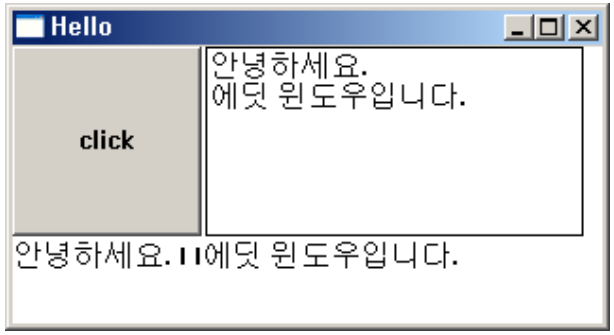
에디트 윈도우 스타일 적용

```
hEdt1 = CreateWindow( . . .,
      WS_VISIBLE|WS_CHILD|WS_BORDER|ES_AUTOHSCROLL|ES_AUTOVSCROLL|ES_MULTILINE,
      . . .
      );
```

실습 4.5

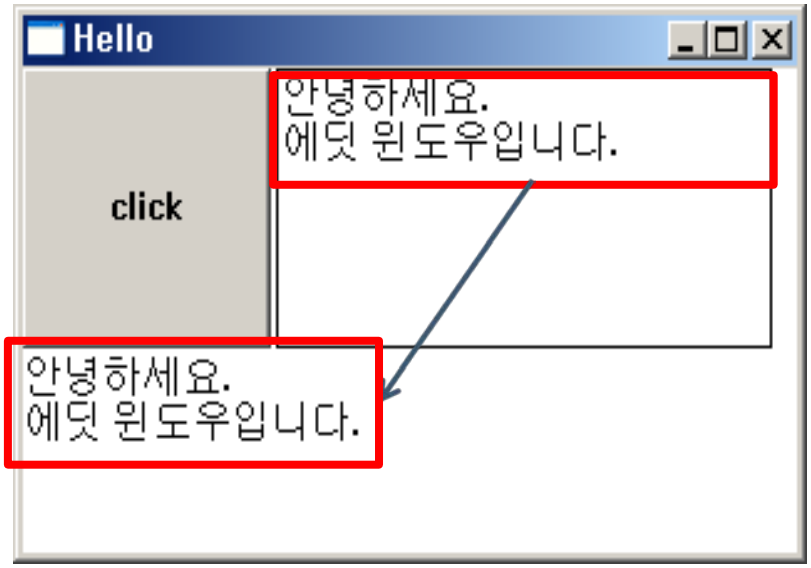
- 버튼을 누르면 에디트 윈도우의 내용을 메인 윈도우에 출력하자.
  - 참고
    - 이전 슬라이드의 에디트 윈도우 스타일 적용
    - GetWindowText( )함수 사용





실습 4.6

- 에디트 윈도우에 여러 줄을 입력한 경우 동일하게 출력되록 하자.
  - 힌트: 기지를 발휘하자.



NOTIFICATION 코드

- 컨트롤 윈도우는 사건의 유형을 부모 윈도우에게 통보
  - 노티피케이션(Notification) 코드 사용
- WM_COMMAND 메시지의 wParam으로부터 추출
  - 버튼 윈도우 코드: BN_*
  - 에디트 윈도우 코드: EN_*

노티피케이션 코드

```
int wNotifyCode;
wNotifyCode = HIWORD( wParam );
```

http://msdn.microsoft.com/en-us/library/bb773169(v=VS.85).aspx

실습 4.7 – 유형 III

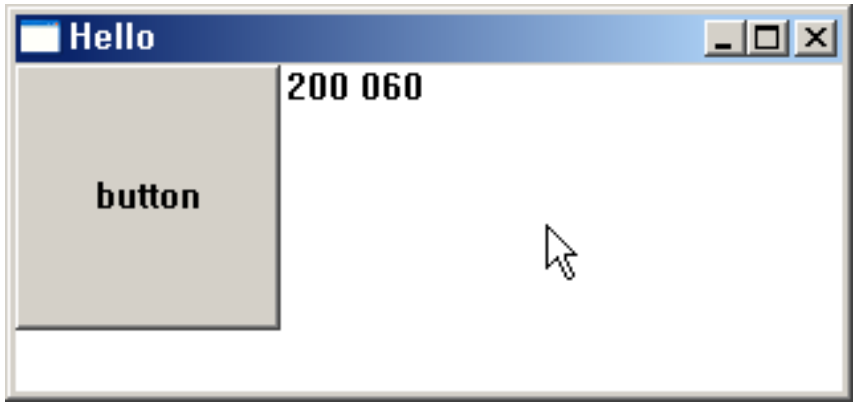
- 첫 번째 에디트 윈도우에 문자를 입력하면 곧바로 두 번째 에디트 윈도우에 그 내용을 출력하자.
  - 참고
    - 에디트 윈도우의 EN_CHANGE 식별 코드 사용

노티피케이션 코드 이벤트 처리의 한계

- 컨트롤 윈도우 메시지 처리에서 필요하다고 판단되는 것들만 일부 구현함
  - 예: BN_CLICKED = WM_LBUTTONDOWN + WM_LBUTTONUP
- 프로그래머가 요구하는 모든 이벤트 유형을 다루기에는 부족함

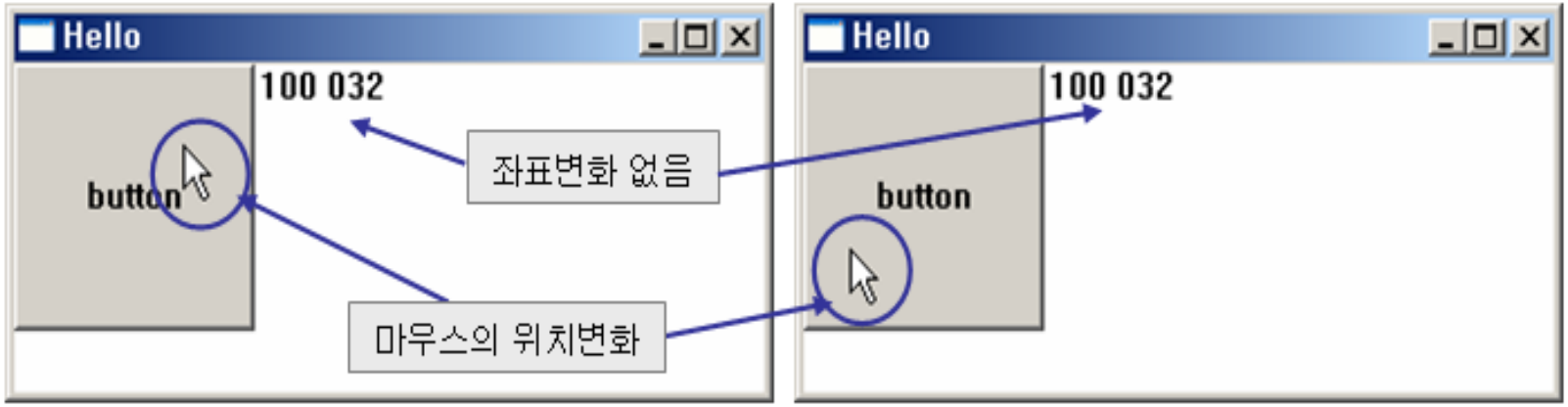
실습 4.8

- 마우스가 움직일 때 마우스 커서의 좌표를 메인 윈도우에 출력하자.



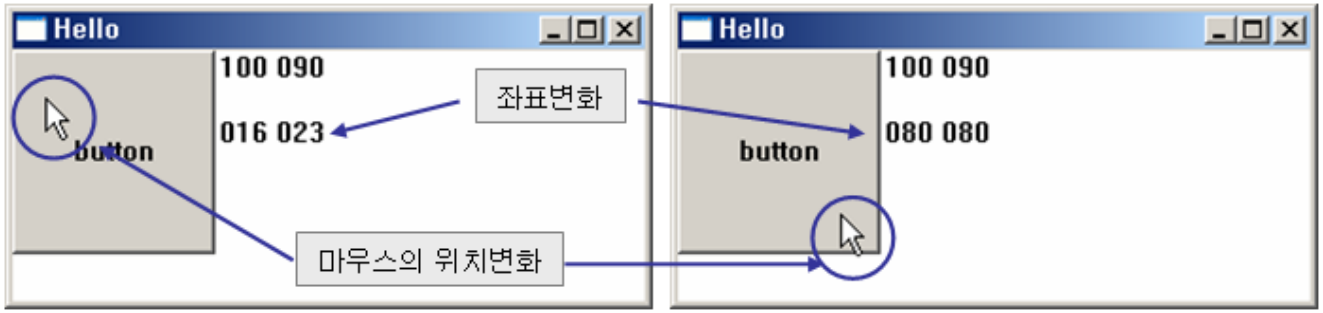
문제점

- 증상
  - 버튼 윈도우 내부에서는 이벤트 처리가 안 됨
- 문제점
  - 버튼 윈도우의 메시지 처리 함수를 확인할 수 없음
  - 버튼 윈도우의 MOUSEMOVE 이벤트 처리에 대한 노티피케이션 코드는 존재하지 않음



실습 4.9 – 유형 IV

- 버튼 윈도우 상에서 마우스를 움직이면 부모 윈도우 영역에 좌표를 출력하자.



처리 단계

1. 기존 버튼 윈도우 메시지 처리 함수의 주소 찾기
   - `LONG GetWindowLong( HWND hWnd, int nIndex );`
2. 버튼 윈도우에 대한 새로운 메시지 처리 함수 작성
3. 새로운 함수에서 WM_MOUSEMOVE 메시지 처리
4. 기존 버튼 윈도우 메시지 처리 함수의 호출
   - `LRESULT CallWindowProc( WNDPROC lpPrevWndFunc, HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam );`
5. 기존 함수를 새로 작성한 메시지 처리 함수로 대체
   - `LONG SetWindowLong( HWND hWnd, int nIndex, LONG dwNewLong );`

GetWindowLong / SetWindowLong / CallWindowProc

▌GetWindowLong

본 함수는 지정된 윈도우에 대한 정보를 검색해 준다. nIndex는 다음과 같이 정의되어 있다. 검색된 정보는 모두 LONG으로 형-변환되어 반환되므로 필요한 형으로 형-변환 한다. 예를 들어 nIndex가 GWL_WNDPROC인 경우 WNDPROC로 형-변환하여 사용한다.

▌SetWindowLong

본 함수는 GetWindowLong의 반대로, 값을 설정하는데 사용된다. 세 번째 인자에 설정 값을 주는데, 요구하는 타입이 LONG임에 주의한다.

■ SetWindowLong

본 함수는 첫 번째 인자로 호출하고자 하는 메시지처리 함수의 주소 인자가 포함되어 있는 것만 제외하고는 기존의 WndProc() 함수와 동일하다.

실습 4.10

- 두 개의 윈도우를 생성하고 자식 윈도우에서 마우스를 움직일 때 현재 위치에 X표시를 출력하자.

자식윈도우 프로시저에서 WM_MOUSEMOVE 메시지 처리

실습 4.11 – 유형 V

- 부모 윈도우에서 마우스를 움직이면 자식 윈도우의 해당 좌표에 X표시를 출력하자.

방법 1.

- WM_MOUSEMOVE 메시지 처리시 출력 윈도우만 자식 윈도우로 설정함

방법 2.

- 접근 방법
  - 부모 윈도우에서 이벤트 처리 및 자식 윈도우의 코드 수행
- 처리 방법
  - 부모가 자식 윈도우 프로시저를 호출 함으로 써 구현 가능
  - 직접 호출이 불가능하므로 부모에서 자식 윈도우 프로시저에 메시지를 보냄
    - SendMessage( ) 함수 사용

SendMessage( )

```
LRESULT SendMessage( HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam );
```

실습 4.12

- 부모 윈도우에서의 마우스 움직임 방향과 직각으로 자식 윈도우에 X표시를 출력하자.

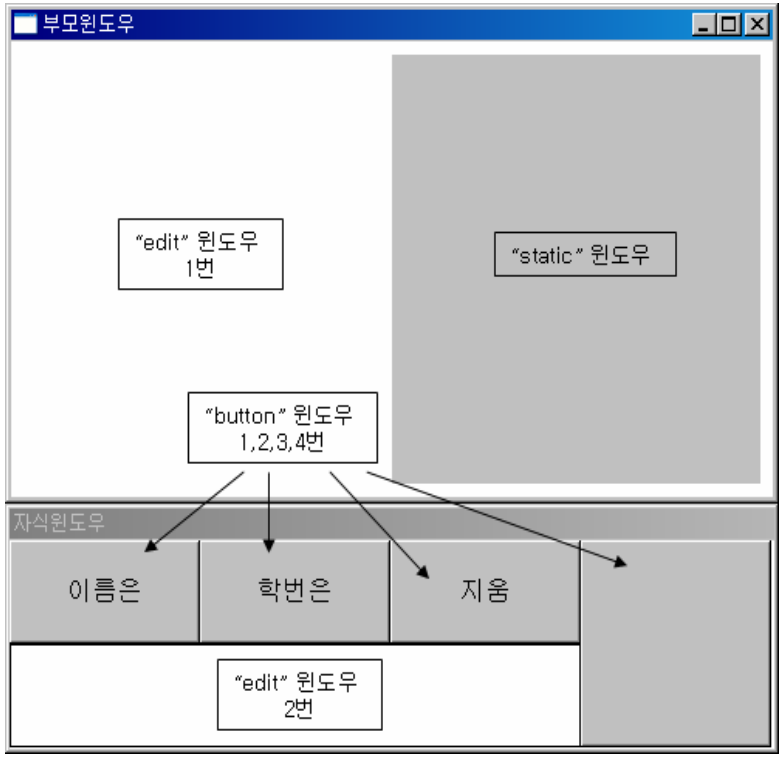
메시지 전달

- SendMessage( )
  - 특정 윈도우 프로시저 직접 호출
- PostMessage( )
  - 메시지를 애플리케이션 메시지 큐에 전달 (간접 호출)

PostMessage()

```
BOOL PostMessage( HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam );
```

윈도우 프로그래밍 정리 (HW)



- 버튼 1, 2, 3을 클릭하면 각 명령에 대한 답변을 edit 1에 출력한다.
- 버튼 4에서 마우스를 움직이면 파란 점을 static 윈도우의 비례위치에 출력한다.
- 버튼 4를 클릭하면 static 윈도우에 그려진 내용을 지운다.
- Edit 2에 문자를 입력하면 바로 edit 1의 화면에 같은 내용을 출력한다.
- 부모 윈도우를 움직이면 자식 윈도우가 따라서 움직인다.
- 부모 윈도우의 크기를 변경하면 자식 윈도우의 가로 크기를 동일한 크기로 변경한다.