7장 차일드 윈도우

2020년도 1학기 윈도우 프로그래밍

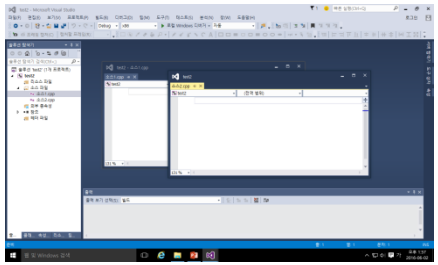
학습 목표

- **학습목표**
  - 차일드 윈도우 만들기
  - 버튼, 에디트 박스, 콤보 박스 등 컨트롤 윈도우를 활용할 수 있다.
  - 윈도우 분할하기

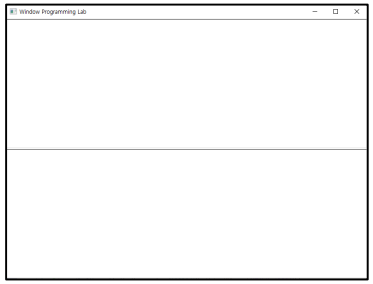
- **내용**
  - 차일드 윈도우를 활용하여 컨트롤 윈도우 만들기
  - 윈도우를 분할하여 차일드 윈도우로 사용하기

여러 개의 윈도우 만들기

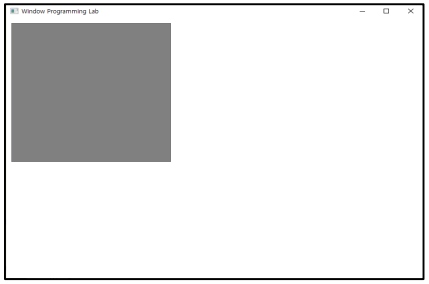
- **1개 이상의 윈도우를 만드는 방법**
  - MDI (Multiple Document Interface): 여러 개의 문서를 여러 개의 화면에 출력하는 형태
    - 예) MS 워드, 엑셀, 비주얼 스튜디오 같은 형태



  - 윈도우 분할: 기존의 윈도우를 여러 개의 자식 윈도우로 분할하는 형태



  - 차일드 윈도우: 부모 윈도우 아래의 자식 윈도우를 생성하는 형태
    - 사용자가 만든 윈도우를 부모 윈도우로 두고, 차일드 윈도우를 만든다.



차일드 윈도우

- **차일드 윈도우는**
  - 기존의 윈도우를 만드는 방법으로 만든다
    - CreateWindow, CreateWindowEx 함수 사용
  - 차일드 윈도우는
    - 각자의 윈도우 클래스를 가진다: 1개 이상의 윈도우 클래스를 등록한다.
      - RegisterClass, RegisterClassEx 함수 사용
      - **차일드 윈도우 클래스는 클래스 이름으로 구분한다.**
    - **각자의 윈도우 프로시저를 가질 수 있다.**
      - **윈도우 프로시저는 차일드 윈도우 클래스에 등록하여 사용한다.**
  - 차일드 윈도우 안에 또 다른 차일드 윈도우를 가질 수 있다.
  - 윈도우 스타일을 **WS_CHILD | WS_VISIBLE** 형태로 설정한다.
    - **WS_POPUP 스타일은 WS_CHILD와 같이 설정할 수 없다.**
  - **컨트롤들을 차일드 윈도우로 만들 수 있다.**

  - 부모 윈도우 가져오기
    - HWND GetParent (HWND hWnd);
  - 부모 윈도우 변경하기
    - HWND SetParent (HWND hWndChild, HWND hWndNewParent);
  - 소유 윈도우 가져오기
    - HWND GetWindow (HWND hWnd, UINT uCmd);
      - uCmd: GW_OWNDR / GW_CHILD…

차일드 윈도우 만들기

- **차일드 윈도우 만들기**
  - **차일드 윈도우 클래스**
    - **차일드 윈도우 클래스**: 등록한 윈도우 클래스로 차일드 윈도우 클래스를 설정
    - **차일드 윈도우 스타일**: **WS_CHILD | WS_VISIBLE** 스타일을 기본으로 설정
      - 위의 두 스타일 외에, **WS_BORDER나 WS_THICKFRAME** 등의 스타일을 같이 설정할 수 있다.
      - WS_CHILD 스타일과 WS_POPUP 스타일을 같이 사용할 수 없다.
  - **윈도우 스타일**
    - WS_OVERLAPPEDWINDOW: WS_CAPTION / WS_HSCROLL / WS_VSCROLL / WS_SYSMENU / WS_MAXIMIZEBOX / WS_MINIMIZEBOX / WS_THICKFRAME / WS_BORDER
      - WS_OVERLAPPEDWINDOW: 기존의 윈도우와 같은 형태로 생성
      - WS_THICKFRAME: 크기를 바꿀 수 있다
      - WS_BORDER: 테두리만 있고 크기와 위치는 바꿀 수 없다
      - WS_POPUPWINDOW: WS_POPUP | WS_BORDER | WS_SYSMENU
        - 팝업 스타일의 차일드 윈도우: WS_POPUPWINDOW | WS_VISIBLE | WS_CAPTION

    - 또는 확장된 윈도우 스타일을 설정할 수 있는 **CreateWindowEx** 함수를 사용한다.
    - HWND **CreateWindowEx** (**DWORD dwExStyle**, LPCTSTR lpClassName, LPCTSTR lpWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWndParent, HMENU hMenu, HANDLE hInstance, LPVOID lpParam);
      - dwExStyle
        - WS_EX_CLIENTEDGE: 작업영역이 쑥 들어간 음각 모양으로 만든다.
        - WS_EX_WINDOWEDGE: 양각 모양의 경계선을 가진 윈도우를 만든다.
        - WS_EX_DLGMODALFRAME: 이중 경계선을 가진 윈도우를 만든다.

차일드 윈도우 만들기

- **일반적인 윈도우를 차일드 윈도우로 만들기**
  - **차일드 윈도우 클래스 등록: WinMain 함수**

```
WNDCLASSEX      wc;

//--- 윈도우 클래스를 등록한다.
wc.cbSize = sizeof(WNDCLASSEX);
wc.style = CS_HREDRAW | CS_VREDRAW;                          // 클래스 스타일
wc.lpfnWndProc = WndProc;                                    // 윈도우 프로시저 지정
wc.cbClsExtra = 0;                                           // 윈도우클래스 데이터 영역
wc.cbWndExtra = 0;                                           // 윈도우의 데이터 영역
wc.hInstance = hInstance;                                    // 인스턴스 핸들
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);                  // 아이콘 핸들
wc.hCursor = LoadCursor(NULL, IDC_ARROW);                    // 사용할 커서 핸들
wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);      // 바탕색 브러쉬 핸들
wc.lpszMenuName = NULL;                                      // 메뉴 이름
wc.lpszClassName = "ParentClass;                             // 윈도우 클래스 이름
wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
RegisterClassEx(&wc);                                        // 윈도우 클래스를 등록

//--- 차일드 윈도우 클래스를 등록한다.
wc.hCursor = LoadCursor(NULL, IDC_HELP);
wc.hbrBackground = (HBRUSH) GetStockObject (GRAY_BRUSH);
wc.lpszClassName = "ChildClass";                             // 차일드 윈도우 클래스 이름
wc.lpfnWndProc = ChildProc;                                  // 차일드 윈도우 프로시저 지정
RegisterClassEx(&wc);                                        // 자식 윈도우 클래스를 등록
```

차일드 윈도우 만들기

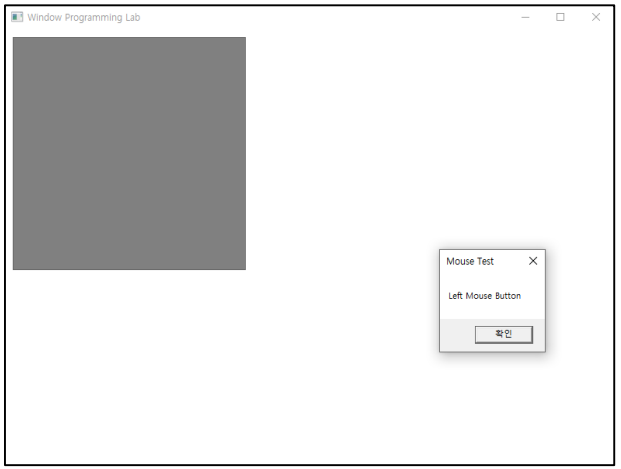
- **윈도우 프로시저 만들기**

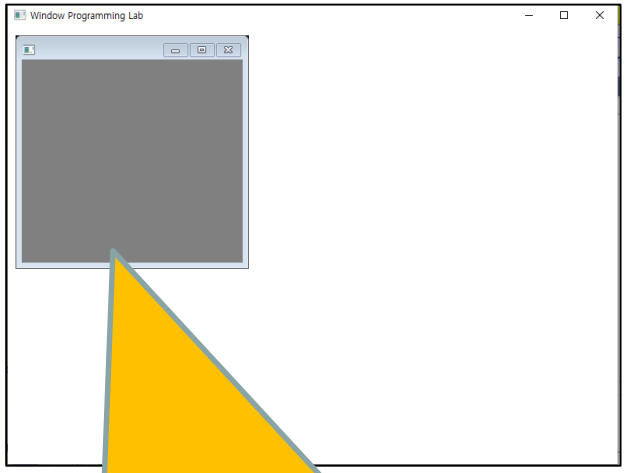
```
LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  HWND child_hWnd;
  switch (uMsg)
  {
            case WM_CREATE:                    //--- 메인윈도우가 생성될 때 자식 윈도우 생성
               child_hWnd = CreateWindow ( " ChildClass ", NULL, WS_CHILD | WS_VISIBLE |
                                          WS_BORDER, 10, 10, 200, 500, hWnd, NULL, g_hInst, NULL);
       break;
  }
   return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

- **차일드 윈도우 프로시저 만들기**

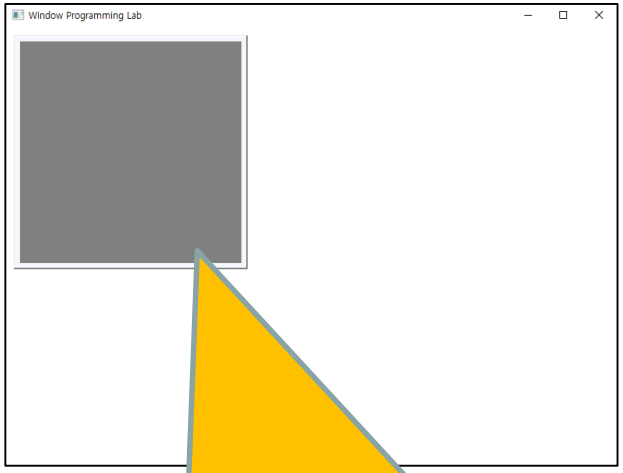
```
LRESULT CALLBACK ChildProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg) {
        case WM_LBUTTONDOWN:              //--- 마우스 좌측 버튼을 누른 경우
              MessageBox (hWnd, L"Left Mouse Button", L"Mouse Test ", MB_OK);
              break;
        case WM_DESTROY:
              PostQuitMessage(0);
              break;
   }
   return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

차일드 윈도우 만들기






```
child_hWnd = CreateWindow ( "ChildClass",
        NULL, WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_OVERLAPPEDWINDOW,
        10, 10, 200, 500,
        hWnd, NULL, g_hInst, NULL);
```




```
child_hWnd = CreateWindow ( "ChildClass",
        NULL, WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_THICKFRAME,
        10, 10, 200, 500,
        hWnd, NULL, g_hInst, NULL);
```

컨트롤 차일드 윈도우

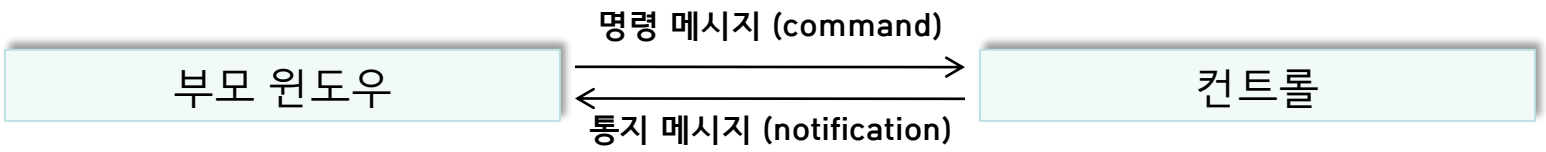
- **대화상자에서 사용했던 컨트롤:**
  - 윈도우로 부모 윈도우 아래의 자식 윈도우로 존재한다.

- **컨트롤의 스타일**

컨트롤	컨트롤 차일드 윈도우 클래스 이름	스타일	명령 메시지	통지 메시지
버튼	"button"	BS_	BM_	BN_
리스트 박스	"listbox:	LBS_	LB_	LBN_
콤보 박스	"combobox"	CBS_	CB_	CBN_
에디트	"edit"	ES_	EM_	EN_

- **컨트롤에서 발생 메시지: WM_COMMAND 메시지**

메시지를 보낸 곳	wParam		lParam
	HIWORD	LOWORD	
컨트롤	컨트롤에 따른 통지정보	컨트롤 ID	컨트롤 핸들값



컨트롤 차일드 윈도우

- **기본적으로 CreateWindow 함수로 자식 윈도우를 만든다.**
  - HWND CreateWindow (LPCTSTR lpClassName, LPCTSTR lpWindowName,
    DWORD dwStyle, int x, int y, int nWidth, int nHeight,
    HWND hWndParent, HMENU hMenu,
    HANDLE hInstance, PVOID lpParam);

    - **lpClassName**: 차일드 윈도우 클래스 이름
      - 예) 버튼 클래스: button
      - 예) 에디트 박스 클래스: edit
      - 예) 리스트 박스: listbox
    - **lpWindowName**: 윈도우 캡션
    - **dwStyle**: 윈도우 스타일
      - 윈도우 스타일 | 차일드 스타일: **WS_CHILD | WS_VISIBLE** 항상 포함하고 고유의 차일드 윈도우 스타일을 함께 설정
      - 예) **WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON**
    - **x, y, nWidth, nHeight**: 윈도우 위치 (x, y), 크기 (폭, 높이)
    - **nWndParent**: 부모 윈도우 핸들
    - **hMenu**: **차일드 윈도우 아이디**
    - **hInstance**: 인스턴스 헨들

1) 버튼 컨트롤

- **버튼 윈도우 클래스: “button”**
- **버튼의 스타일**

윈도우 클래스 이름	윈도우 스타일	버튼 내용
button	BS_PUSHBUTTON	푸시 버튼
	BS_DEFPUSHBUTTON	디폴트 푸시 버튼
	BS_CHECKBOX	체크 박스
	BS_3STATE	3가지 상태를 가지는 체크 박스
	BS_AUTOCEHCKBOX	자동 체크 박스
	BS_AUTO3STATE	3가지 상태를 가지는 자동 체크 박스
	BS_AUTORADIOBUTTON	자동 라디오 버튼
	BS_GROUPBOX	그룹 박스

버튼 만들기

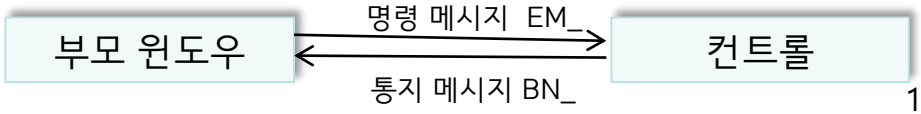
- **버튼 만들기**

```
#define IDC_BUTTON 100

HWND hButton;
hButton = CreateWindow  (L"button",  L"확인",
          WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 200, 0, 100, 25,
          hwnd, (HMENU) IDC_BUTTON, g_hInst, NULL);
```

  - 버튼 클래스: **button**
  - 윈도우 스타일: **WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON**
  - 버튼 이벤트 (버튼 통지)

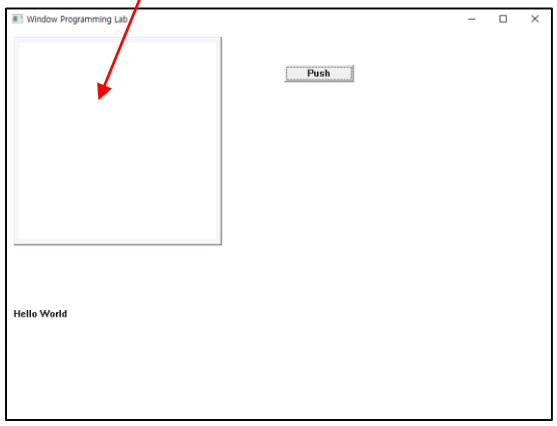
Notify	Meaning
BN_CLICKED	버튼 위에서 마우스가 클릭되었을 때
BN_DBCLK	버튼 위에서 마우스가 더블 클릭되었을 때
BN_SETFOCUS	버튼 위 마우스 커서가 올 때
BN_KILLFOCUS	버튼 위에서 마우스가 벗어날 때
BN_PAINT	버튼 내부를 Drawing 할 때

  - WM_COMMAND 메시지
    - HIWORD(wParam): 통지 코드
    - LOWORD(wParam): 컨트롤의 ID



버튼 만들기

```
#define IDC_BUTTON 100                  // 버튼 컨트롤의 ID

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  HDC hdc;
  static HWND hButton;
  HWND child_hWnd;

  switch (iMsg)
  {
       case WM_CREATE:
            child_hWnd = CreateWindow ( L"ChildClass", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER | WS_THICKFRAME,
                         10, 10, 300, 300, hWnd, NULL, g_hInst, NULL);        //--- 좌측상단의 차일드 윈도우

            hButton = CreateWindow ( L"button",  L"Push",                     //--- 버튼의 윈도우 클래스 이름은 "button"
                         WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,               //--- 차일드 윈도우,  누르는 형태의 버튼 스타일
                         400, 50, 100, 25, hwnd,
                         (HMENU) IDC_BUTTON, hInst, NULL);
      break;

`     case WM_COMMAND:
            switch(LOWORD(wParam)) {
                  case IDC_BUTTON:
                        hdc = GetDC(hwnd);
                        TextOut (hdc, 10, 400, L"Hello World", 11);
                        ReleaseDC (hwnd, hdc);
                        break;
            }
      break;
    }
}
```



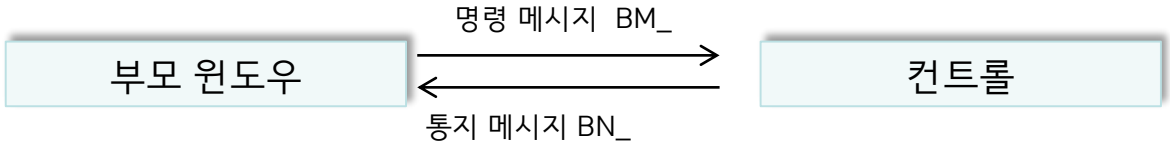
체크박스 만들기

- **체크박스 윈도우 클래스: “button”**
- **체크박스 스타일: BS_CHECKBOX 또는 BS_AUTOCHECKBOX**
- **체크박스 만들기**

```
#define IDC_CHECK 200

HWND hCheck;
hCheck = CreateWindow  (L“button " , L " 체크 박스 테스트 " ,
        WS_CHILD | WS_VISIBLE | BS_CHECKBOX, 100, 0, 100, 25,
        hwnd, (HMENU) IDC_CHECK, hInst, NULL);
```

- 체크 박스 클래스: **button**
- 체크 박스 스타일: **WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX**
  - **BS_CHECKBOX 스타일:** 체크 상태를 수동으로 바꿔준다.
  - **BS_AUTOCHECKBOX 스타일:** 체크 상태가 자동으로 바뀌어 진다.

- 체크박스 통지 메시지:
  - 차일드 윈도우 → 부모 윈도우: **BN_CLICKED** 메시지를 보낸다.



체크박스 만들기

- 부모 윈도우가 체크 박스의 현재 상태를 알아보거나 상태를 바꾸고자 할 때 차일드 윈도우로 메시지를 보낸다.
  - SendMessage 함수를 이용하여 차일드 윈도우로 메시지를 보낸다.
  - 보내는 메시지: BM_GETCHECK / BM_SETCHECK

메시지	의미	리턴 값 또는 체크 박스 상태
BM_GETCHECK	체크박스가 현재 체크되어 있는 상태인지 조사	• BST_CHECKED: 현재 체크되어 있다. • BST_UNCHECKED: 현재 체크되어 있지 않다. • BST_INDETERMINATE: 체크도 아니고 비 체크도 아닌 상태
BM_SETCHECK	체크 박스의 체크 상태를 변경, wParam에 변경할 체크상태를 보내준다	

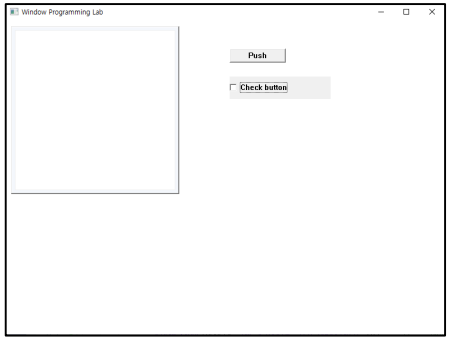
LRESULT SendMessage (HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);

- 메시지를 메시지 큐에 넣지 않고 바로 윈도우 프로시저로 보냄
- hWnd: 메시지를 전달받을 윈도우 핸들
- Msg : 전달할 메시지
- wParam, lParam 메시지의 추가적 정보, 메시지에 따라 다른 정보 반환

체크박스 만들기

- **체크박스 버튼 컨트롤 (수동)**

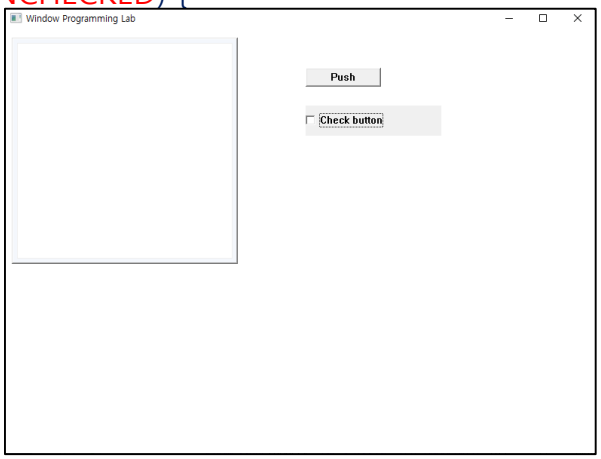
```
#define IDC_CHECK 110

LRESULT CALLBACK WndProc ( HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
   static HWND hCheck;
   static int cList[2];

   switch (iMsg) {
        case WM_CREATE:
          //--- 앞의 차일드 윈도우와 버튼을 만들었다고 가정
          hCheck = CreateWindow ( L"button", L"Check button",          //--- 윈도우 클래스 이름은 button
                                  WS_CHILD | WS_VISIBLE | BS_CHECKBOX,          //--- 차일드 윈도우, 수동 체크 박스
                                  10, 210, 180, 40, hwnd, (HMENU) IDC_CHECK, hInst, NULL);
          break;

      case WM_COMMAND:
            switch (LOWORD(wParam) ) {
              case IDC_CHECK:                                                    //---체크 박스가 선택되었는데
                  if  (SendMessage (hCheck, BM_GETCHECK, 0, 0) == BST_UNCHECKED) {  //--- 선택되지 않았다면
                        SendMessage (hCheck, BM_SETCHECK, BST_CHECKED, 0);          //--- 선택 상태로 바꾸기
                        cList[0] = 1;
                  }
                  else {                                                             //--- 아니라면: 선택상태라면
                        SendMessage (hCheck, BM_SETCHECK, BST_UNCHECKED, 0);        //--- 해제 상태로 바꾸기
                        cList[0] = 0;
                  }
                  break;
              }
              break;
       }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```



체크박스 만들기

- **체크박스 버튼 컨트롤 (자동)**

```
#define IDC_CHECK 110

LRESULT CALLBACK WndProc ( HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
   static HWND hCheck;
   static int cList[2];

   switch (iMsg) {
        case WM_CREATE:
            //--- 앞의 차일드 윈도우와 버튼을 만들었다고 가정
              hCheck = CreateWindow ( L"button " , L"Check button",          //--- 윈도우 클래스 이름은 button
                                      WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,  //--- 차일드 윈도우, 자동 체크 박스
                                      10, 210, 180, 40, hwnd, (HMENU) IDC_CHECK,
                                      hInst, NULL);
               break;

     case WM_COMMAND:
              switch (LOWORD(wParam) ) {
              case IDC_CHECK:                          //--- 체크박스: 자동 체크박스로 만들어 체크박스의 상태만  조사한다.
                                                       //--- SendMessage 를 하지 않는다.
                    if  (SendMessage (hCheck, BM_GETCHECK, 0, 0) == BST_UNCHECKED) {
                          cList[0] = 1;
                    }
                    else {
                          cList[0] = 0;
                    }
                    break;
               }
       }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```



라디오 버튼 만들기

- **라디오 버튼 윈도우 클래스: “button”**
- **라디오 버튼 스타일: BS_AUTORADIOBUTTON**
- **라디오 버튼 만들기**

```
#define IDC_RADIO 300

HWND hRadio;
hRadio = CreateWindow ( "button", “라디오 버튼 테스트",
                WS_CHILD |WS_VISIBLE | BS_AUTORADIOBUTTON,
                100, 0, 100, 30, hWnd, (HMENU) IDC_RADIO, g_hInst, NULL);
```

- 라디오 버튼 클래스: **button**
- 라디오 버튼 스타일: **WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON**
  - **그룹의 시작을 위하여 첫번째 라디오 버튼에 WS_GROUP 스타일 추가한다.**

- 라디오 버튼 그룹 박스 만들기
  - 그룹 박스 클래스: **button**
  - 그룹 박스 컨트롤 스타일: **BS_GROUPBOX**

라디오 버튼 만들기

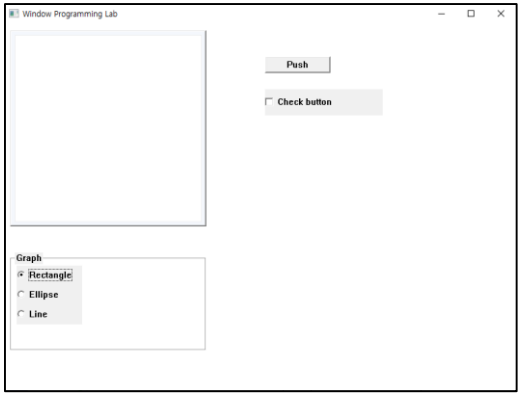
- **라디오 버튼 초기화 함수**

BOOL **CheckRadioButton** (HWND hDlg, int nIDFirstButton, int nIDLastButton,
int nIDCheckButton);

- 처음 선택될 라디오 버튼 선택
- hDlg: 라디오 버튼을 가지는 부모 윈도우(또는 대화상자)의 핸들
- nIDFirstButton : 각각 그룹의 시작 버튼 아이디
- nIDLastButton: 각각 그룹의 끝 버튼 아이디
- nIDCheckButton: 선택될 버튼의 아이디

라디오 버튼 만들기

```
#define ID_R1       200
#define ID_R2       210
#define ID_R3       220

LRESULT CALLBACK WndProc ( HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HWND r1, r2, r3;
        static int shape;

        switch (iMsg) {
        case WM_CREATE:
            //--- 그룹 박스로 윈도우 만들기
            CreateWindow (L"button", L"Graph", WS_CHILD | WS_VISIBLE | BS_GROUPBOX,
                                        10, 350, 300, 150, hWnd, (HMENU)0, g_hInst, NULL);
            //--- 버튼 만들기: 그룹 1
            r1= CreateWindow (L"button", L"Rectangle", WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON | WS_GROUP,
                                20, 370, 100, 30, hWnd, (HMENU) ID_R1, g_hInst, NULL);
            r2= CreateWindow (L"button", L"Ellipse", WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                                20, 400, 100, 30, hWnd, (HMENU) ID_R2, g_hInst, NULL);
            r3= CreateWindow (L"button", L"Line", WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                                20, 430, 100, 30, hWnd, (HMENU) ID_R3, g_hInst, NULL);

            CheckRadioButton (hWnd, ID_R1, ID_R3, ID_R1);
            break;

        case WM_COMMAND:
            switch (LOWORD (wParam)) {
                case ID_R1:   shape = 1;    break;
                case ID_R2:   shape = 2;    break;
                case ID_R3:   shape = 3;    break;
            }
            break;
        }

}
```



2) 에디트 컨트롤 만들기

- **에디트 컨트롤 클래스: “edit”**
- **에디트 컨트롤 스타일: WS_BORDER 포함하고, 추가 스타일은 뒷 페이지에**
- **에디트 컨트롤 만들기**

```
#define IDC_EDIT        101

HWND hEdit;
hEdit = CreateWindow  ( "edit", "에디팅",
                        WS_CHILD | WS_VISIBLE | WS_BORDER,
                        0, 0, 200, 25, hwnd, (HMENU) IDC_EDIT, hInst, NULL);
```

  - 에디트 컨트롤 클래스: **edit**
  - 에디트 컨트롤 스타일: WS_CHILD | WS_VISIBLE | WS_BORDER

에디트 컨트롤 만들기

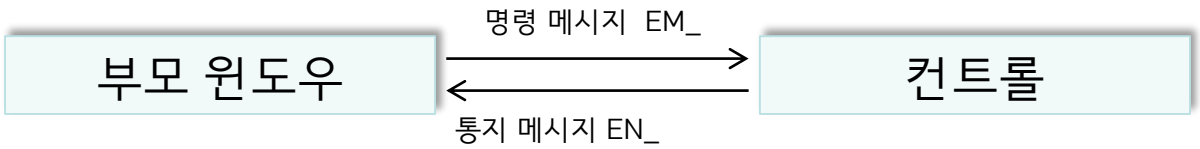
- **에디트 컨트롤 윈도우 스타일**

클래스 이름	스타일	의미
edit	ES_AUTOHSCROLL	수평 스크롤을 지원
	ES_AUTOVSCROLL	여러 줄을 편집할 때 수직 스크롤을 지원
	ES_LEFT	왼쪽 정렬
	ES_RIGHT	오른쪽 정렬
	ES_CENTER	중앙 정렬
	ES_LOWERCASE	소문자로 변환하여 표시
	ES_UPPERCASE	대문자로 변환하여 표시
	ES_MULTILINE	여러 줄을 편집
	ES_READONLY	읽기 전용, 편집할 수 없다.
	ES_PASSWORD	입력되는 모든 문자를 *로 보여준다.

에디트 컨트롤 만들기

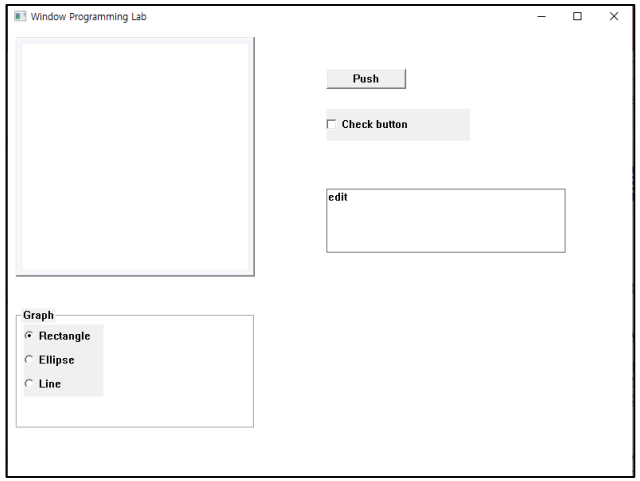
- 에디트 컨트롤 메시지 통지

메시지	의미
EN_CHANGE	Editbox의 내용이 변경된 후 발생 (화면에 갱신된 후)
EN_UPDATE	Editbox 내용이 변경되려고 할 때 발생 (사용자가 타이프한 후 화면에 갱신되기 직전에 발생)
EN_SETFOCUS	포커스를 받을 때 발생
EN_KILLFOCUS	포커스를 잃을 때 발생
EN_HSCROLL/EN_VSCROLL	수평 / 수직 스크롤바 클릭
EN_MAXTEXT	지정한 문자열 길이를 초과
EN_ERRSPACE	메모리 부족



에디트 컨트롤 만들기

```
#define IDC_BUTTON 100
#define IDC_EDIT       101                          // 에디트 컨트롤의 ID


HWND hButton, hEdit;
TCHAR str[100];

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
      case WM_CREATE:
            //--- 박스 주위에 테두리가 있고 여러줄 입력을 받을 수 있는 에디트 컨트롤스타일
            hEdit = CreateWindow ( L"edit", L"edit", WS_CHILD | WS_VISIBLE | WS_BORDER | ES_MULTILINE,
                                                      400, 200, 300, 30, hWnd, (HMENU)IDC_EDIT, g_hInst, NULL);
            break;

      case WM_COMMAND:
            switch(LOWORD(wParam)) {
                  case IDC_BUTTON:
                        GetDlgItemText(hwnd, IDC_EDIT, str, 100);
                        hdc = GetDC (hwnd);
                        TextOut(hdc, 500, 400, str, lstrlen(str));
                        ReleaseDC(hwnd, hdc);
                  break;
            }
            break;
      }
      return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



3) 콤보 박스 만들기

- **콤보박스 클래스: "combobox"**
- **콤보 박스 스타일**

스타일	의미
CBS_SIMPLE	에디트만 가진다.
CBS_DROPDOWN	에디트와 리스트 박스를 가진다.
CBS_DROPDOWNLIST	리스트 박스만 가지며 에디트에 항목을 입력할 수는 없다
CBS_AUTOHSCROLL	콤보 박스에서 항목을 입력할 때 자동 스크롤

- **콤보박스의 다운 버튼을 눌렀을 때**
  - CBN_DROPDOWN 통지가 보내진다.

메시지	의미
CBN_DBLCLK	콤보 박스를 더블클릭하였다.
CBN_DROPDOWN	콤보박스의 에디트영역 우측의 버튼을 누르면 리스트박스가 열린다.
CBN_ERRSPACE	메모리가 부족하다.
CBN_KILLFOCUS	키보드 포커스를 잃었다.
CBN_SELCANCEL	사용자가 선택을 취소하였다.
CBN_SELCHANGE	사용자에 의해 선택이 변경되었다.
CBN_SETFOCUS	키보드 포커스를 얻었다.

콤보 박스에 전달되는 메시지

- **부모 윈도우가 콤보 박스에 보내는 메시지 (SendMessage 함수 사용)**

메시지	의미	전달 값
CB_ADDSTRING	콤보 박스에 텍스트를 아이템으로 추가하는 메시지로써 리스트의 마지막에 추가된다.	wParam: 사용하지 않음 lParam: 텍스트 스트링의 시작 주소
CB_DELETESTRING	콤보 박스에 있는 아이템들 중 하나를 삭제하는 메시지	wParam: 삭제하기 원하는 아이템의 인덱스로 0부터 시작한다. lParam: 0
CB_GETCOUNT	콤보 박스의 아이템 리스트에 들어 있는 아이템의 개수를 얻기 위한 메시지로 개수 값은 SendMessage()함수가 리턴한다.	wParam: 0 lParam: 0
CB_GETCURSEL	현재 선택된 아이템의 인덱스 번호를 얻기 위한 메시지로 인덱스 번호는 SendMessage()함수가 리턴한다.	wParam: 0 lParam: 0
CB_SETCURSEL	콤보 박스 컨트롤의 텍스트 편집 공간에 지정한 항목의 텍스트를 보여준다.	wParam: 나타내고자 하는 항목의 인덱스 번호 lParam: 사용않음
CB_GETLBTEXT	콤보박스의 리스트에서 문자열을 가져온다. 리턴값은 문자열의 길이	wParam: 검색할 문자열의 인덱스 번호 lParam: 문자열을 받는 버퍼에 대한 포인터

콤보 박스 만들기

```
#define IDC_BUTTON      100
#define IDC_EDIT        101
#define IDC_COMBO       102             //--- 콤보 박스 컨트롤의 ID

HWND hButton, hEdit, hCombo;
 TCHAR str[100];

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  switch (iMsg)
  {
   case WM_CREATE:
                hCombo = CreateWindow (L"combobox", NULL, WS_CHILD | WS_VISIBLE | CBS_DROPDOWN, // 콤보 박스
                        0, 100, 200, 300, hwnd, (HMENU) IDC_COMBO, hInst, NULL);
      return 0;
   case WM_COMMAND:
         switch(LOWORD(wParam)) {
           case ID_COMBOBOX:
                  switch (HIWORD(wParam)) {
                        case CBN_SELCHANGE:
                                i= SendMessage (hCombo, CB_GETCURSEL,0,0);
                                SendMessage (hCombo, CB_GETLBTEXT, i, (LPARAM)str);
                                SetWindowText (hWnd, str);
                        break;
                   }
                 break;
           case IDC_BUTTON:
                 GetDlgItemText (hwnd, IDC_EDIT, str, 100);
                 if (lstrcmp(str, L""))
                        SendMessage (hCombo,CB_ADDSTRING,0,(LPARAM)str);
                 break;
     }
     break;
```

4) 리스트 박스 만들기

- 리스트 박스 클래스 이름: listbox
- 리스트 박스 스타일

스타일	의미
LBS_MULTIPLESEL	여러개의 항목을 선택할 수 있도록 한다. 이 스타일을 적용하지 않으면 디폴트로 하나만 선택할 수 있다.
LBS_NOTIFY	사용자가 목록중 하나를 선택했을 때 부모 윈도우로 통지 메시지를 보내도록 한다.
LBS_SORT	추가된 항목들을 자동 정렬하도록 한다.
LBS_OWNERDRAW	문자열이 아닌 비트맵이나 그림을 넣을 수 있도록 한다
LBS_STANDARD	LBS_NOTIFY | LBS_SORT | WS_BORDER (가장 일반적인 스타일)

- 리스트 박스에서 메시지가 발생했을 때 부모 윈도우로 보내는 통지 메시지

메시지	의미
LBN_DBLCLK	리스트 박스를 더블클릭하였다.
LBN_ERRSPACE	메모리가 부족하다.
LBN_KILLFOCUS	키보드 포커스를 잃었다.
LBN_SELCANCEL	사용자가 선택을 취소하였다.
LBN_SELCHANGE	사용자에 의해 선택이 변경되었다.
LBN_SETFOCUS	키보드 포커스를 얻었다.

리스트 박스에 전달되는 메시지

- **부모 윈도우가 리스트 박스에게 보내는 메시지 (SendMessage 함수 사용)**

메시지	의미	전달 값
LB_ADDSTRING	리스트 박스에 텍스트를 아이템으로 추가하는 메시지로써 리스트의 마지막에 추가된다.	**wParam**: 사용하지 않음 **lParam**: 텍스트 스트링의 시작 주소
LB_DELETESTRING	리스트 박스에 있는 아이템들 중 하나를 삭제하는 메시지	**wParam**: 삭제하기 원하는 아이템의 인덱스로 0부터 시작한다. **lParam**: 0
LB_GETCOUNT	리스트 박스의 아이템 리스트에 들어 있는 아이템의 개수를 얻기 위한 메시지로 개수 값은 SendMessage()함수가 리턴한다.	**wParam**: 0 **lParam**: 0
LB_GETCURSEL	현재 선택된 아이템의 인덱스 번호를 얻기 위한 메시지로 인덱스 번호는 SendMessage()함수가 리턴한다.	**wParam**: 0 **lParam**: 0
LB_GETTEXT	아이템 리스트중 wParam에서 지정한 인덱스 아이템의 텍스트를 얻어오는 메시지	**wParam**: 얻어올 아이템의 인덱스 번호 **lParam**: 얻어온 텍스트를 저장할 버퍼의 시작 주소
LB_INSERTSTRING	리스트 박스에 텍스트를 아이템으로 리스트 중간에 추가하는 메시지	**wParam**: 아이템 리스트중 추가될 위치의 인덱스 번호, -1이면 마지막에 추가 **lParam**: 추가할 텍스트 스트링의 시작 주소

리스트 박스 만들기

```
#define ID_LISTBOX 100

TCHAR Items[][15]={L"First", L"Second", L"Third", L"Fourth"};
TCHAR str[128];
HWND hList;

LRESULT CALLBACK WndProc  (HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
        int i;
        switch(iMessage) {
        case WM_CREATE:
                hList=CreateWindow (L"listbox", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER | LBS_STANDARD ,
                                10, 10, 100, 200, hWnd, (HMENU) ID_LISTBOX, g_hInst, NULL);
                for ( i=0; i<4; i++)
                        SendMessage (hList, LB_ADDSTRING, 0, (LPARAM)Items[i]);
                return 0;

        case WM_COMMAND:
                switch (LOWORD(wParam))
                {
                        case ID_LISTBOX:
                                switch (HIWORD(wParam))
                                {
                                        case LBN_SELCHANGE:
                                                i=SendMessage (hList, LB_GETCURSEL,0,0);
                                                SendMessage (hList, LB_GETTEXT, i, (LPARAM)str);
                                                SetWindowText (hWnd, str);
                                           break;
                                }
                } return 0;
        }
        return (DefWindowProc (hWnd, iMessage, wParam, lParam));
}
```

윈도우 분할하기

- **프레임 윈도우를 분할하여 차일드 윈도우 관리**
  - 분할된 윈도우는 자식 윈도우이지만 팝업 윈도우는 아니므로 자식 윈도우에 타이틀 바를 포함하는 독립적인 프레임이 존재하지는 않는다.
  - 분할 윈도우도 메인 윈도우와 같은 방법으로 생성하고, **CreateWindowEx** 함수를 사용한다. (CreateWindowEx 함수를 사용하면 윈도우 가장자리의 스타일을 설정할 수 있다.)
  - 차일드 윈도우 스타일: WS_CHILD | WS_VISIBLE

윈도우 분할: 차일드 윈도우 생성 함수

- **윈도우 생성 함수**

```
HWND CreateWindowEx (
        DWORD dwExStyle,          // 생성되는 확장 윈도우의 스타일
        LPCTSTR lpClassName,      // 등록된 윈도우클래스
        LPCTSTR lpWindowName,     // 윈도우 타이틀 텍스트
        DWORD dwStyle,            // 기본 윈도우 스타일
        int x,                    // 생성 윈도우 위치의 x값
        int y,                    // 생성 윈도우 위치의 y값
        int nWidth,               // 생성 윈도우의 너비
        int nHeight,              // 생성 윈도우의 높이
        HWND hWndParent,          // 부모 윈도우 핸들
        HMENU hMenu,              // 사용될 메뉴의 핸들
        HINSTANCE hInstance,      // 어플리케이션 인스턴스
        LPVOID lpParam
        );
```

  - dwExStyle 스타일:

스타일	내용
WS_EX_DLGMODALFRAME	이중 경계선을 가진 윈도우를 만든다
WS_EX_WINDOWEDGE	양각 모양의 경계선을 가진 윈도우를 만든다.
WS_EX_CLIENTEDGE	작업영역이 쑥 들어간 음각 모양으로 만든다.
WS_EX_MDICHILD	MDI 차일드 윈도우를 만든다.
WS_EX_OVERLAPPEDWINDOW	(WS_EX_WINDOWEDGE | WS_EX_CLIENTEDGE)복합 속성

윈도우 분할: 차일드 윈도우 클래스 등록

- **윈도우 클래스 등록**

```
WNDCLASSEX   wndclass ;                          // 변수 선언

//--- 메인 윈도우 클래스 생성 및 등록
wndclass.cbSize                 = sizeof(wndclass) ;
wndclass.style                  = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc            = MainWndProc;                  //---  메인 윈도우 (부모 윈도우) 프로시저
wndclass.cbClsExtra             = 0 ;
wndclass.cbWndExtra             = 0 ;
wndclass.hInstance              = hInstance ;
wndclass.hIcon                  = LoadIcon(NULL,IDI_APPLICATION);
wndclass.hCursor                = LoadCursor(NULL,IDC_ARROW) ;
wndclass.hbrBackground          = (HBRUSH)GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName           = MAKEINTRESOURCE(IDR_MENU1) ;
wndclass.lpszClassName          = L"Window Class Name";          //--- 메인윈도우 클래스 이름: 클래스 구분자
Wndclass.hIconSm                = LoadIcon(NULL,IDI_APPLICATION);
RegisterClassEx(&wndclass);                                      //--- 메인 윈도우 (부모 윈도우) 클래스 등록


//--- 차일드 윈도우 클래스 생성 및 등록 : 차일드를 위해 wndclass를 재사용
Wndclass.lpfnWndProc            = ChildWndProc;                  //--- 차일드 윈도우 프로시저
Wndclass.lpszMenuName            = NULL ;
Wndclass.lpszClassName          = L"Child Window Class Name";    //--- 차일드 윈도우 클래스 이름: 클래스 구분자
RegisterClassEx(&wndclass);                                      //--- 차일드 윈도우 클래스 등록
```

윈도우 분할: 윈도우 프로시저

- **메인 윈도우 프로시저**
  - 메인 윈도우를 상하로 이등분하여 차일드 윈도우를 2개 만든다,
  - 각각의 윈도우에 타이머를 설정: 원이 우측으로 이동

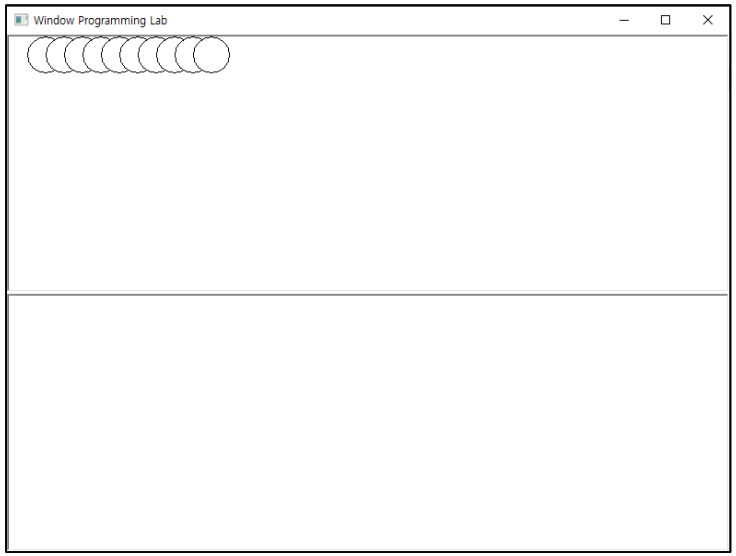
```
HWND ChildHwnd[2];

LRESULT CALLBACK MainWndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  RECT rectView;
  switch (iMsg)
  {
        case WM_CREATE:
                GetClientRect (hwnd, &rectView);

                //--- 윈도우의 상단에 만든 차일드 윈도우: 윈도우 클래스는 Child Window Class Name 사용
                ChildHwnd[0] = CreateWindowEx ( WS_EX_CLIENTEDGE,
                        "Child Window Class Name", NULL,
                        WS_CHILD | WS_VISIBLE ,
                        0, 0, rectView.right, rectView.bottom/2-1,
                        hwnd, NULL, hInst, NULL );

                //--- 윈도우의 하단에 만든 차일드 윈도우: 윈도우 클래스는 Child Window Class Name 사용
                ChildHwnd[1] = CreateWindowEx ( WS_EX_CLIENTEDGE,
                        "Child Window Class Name", NULL,
                        WS_CHILD | WS_VISIBLE,
                        0, rectView.bottom/2+1, rectView.right, rectView.bottom/2-1,
                        hwnd, NULL, hInst, NULL );
                break;
}
```

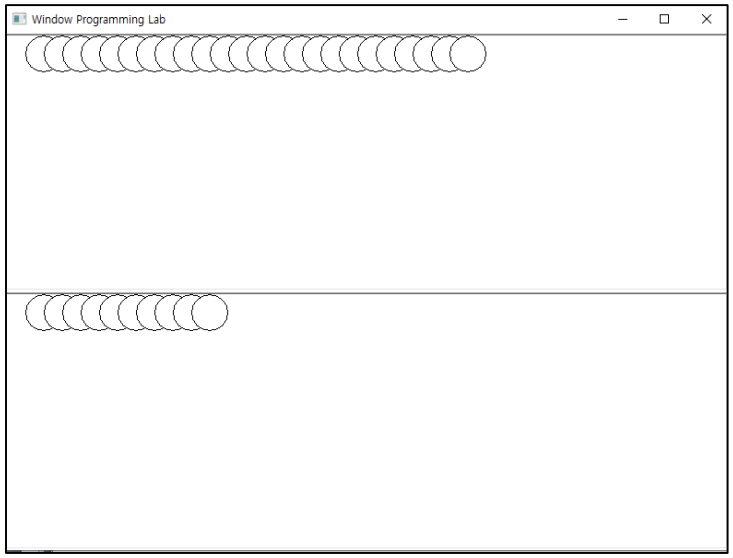
윈도우 분할: 윈도우 프로시저

- **차일드 윈도우 프로시저**
  - 차일드 윈도우 2개가 같은 윈도우 클래스 사용 ➔ 같은 윈도우 프로시저 사용

```
LRESULT CALLBACK ChildWndProc (HWND hwnd, UINT iMsg, WPARAM wParam,LPARAM lParam)
{
    HDC hdc;
    static int x[2]={20,20}, y[2]={20,20}, flag[2];
    static int select;

 switch (iMsg) {
       case WM_TIMER:                              //--- 원이 우측으로 이동하는 애니메이션
            hdc = GetDC(hwnd);
            x[wParam] = x[wParam] + 20;
            Ellipse(hdc, x[wParam]-20, y[wParam]-20, x[wParam]+20, y[wParam]+20);
            ReleaseDC(hwnd, hdc);
            break;

       case WM_LBUTTONDOWN:
            if (hwnd == ChildHwnd[0])              //--- 사용하는 윈도우가 차일드윈도우 0이라면
                 select = 0;
            else                                   //--- 사용하는 윈도우가 차일드윈도우 1이라면
                 select = 1;
            flag[select] = 1 - flag[select];
            if (flag[select])                      //--- 현재 차일드 윈도우에 타이머 설정
                 SetTimer(hwnd, select, 100, NULL);
            else
                 KillTimer(hwnd, select);
            break;
    }
 return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

윈도우 분할: 윈도우 프로시저





윈도우 분할: 윈도우 다루기 함수들

- 윈도우 다루기 함수

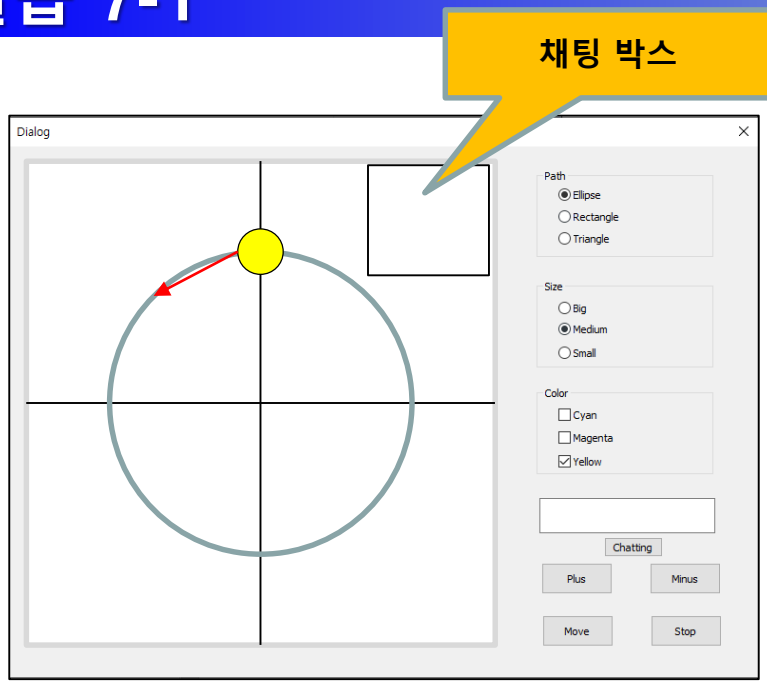
함수 원형	함수 소개
BOOL **MoveWindow** (HWND hWnd, int x, int y, int nWidth, int nHeight, BOOL bRepaint);	윈도우의 위치와 크기를 변경하는 함수
HWND **SetCapture** (HWND hWnad) / HWND **ReleaseCapture** ();	마우스를 윈도우 내에 캡처하는 함수 / 마우스 캡처를 해제하는 함수
HWND **SetFocus** (HWND hWnd); / HWND **GetFocus** ();	키보드 포커스를 설정하여 윈도우를 활성화 해주는 함수 / 키보드 포커스를 가진 윈도우 핸들 반환 함수 차일드 윈도우를 만들 때 차일드 윈도우에 포커스를 주어야 할 때 SetFocus 함수를 사용
BOOL **IsChild** (HWND hWndParent, HWND hWnd);	hWnd 윈도우가 차일드 윈도인지 확인
HWND **GetWindow** (HWND hWnd, UINT uCmd);	uCmd 관계를 가지고 있는 윈도우 핸들을 얻는 함수
HWND **GetParent** (HWND hWnd);	부모 윈도우 핸들을 얻는 함수
HWND **FindWindow** (LPCSTR lpClassName, LPCSTR lpWindowName);	윈도우 클래스 이름을 가진 윈도우를 찾는 함수

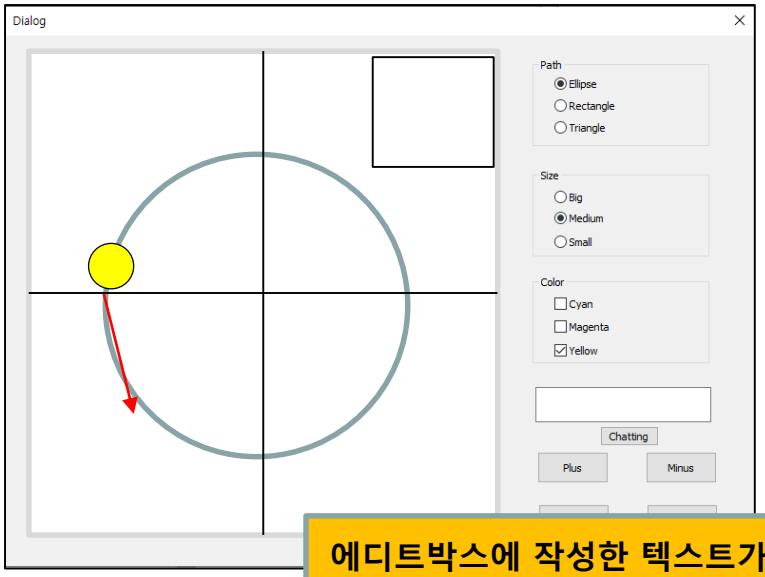
실습 7-1

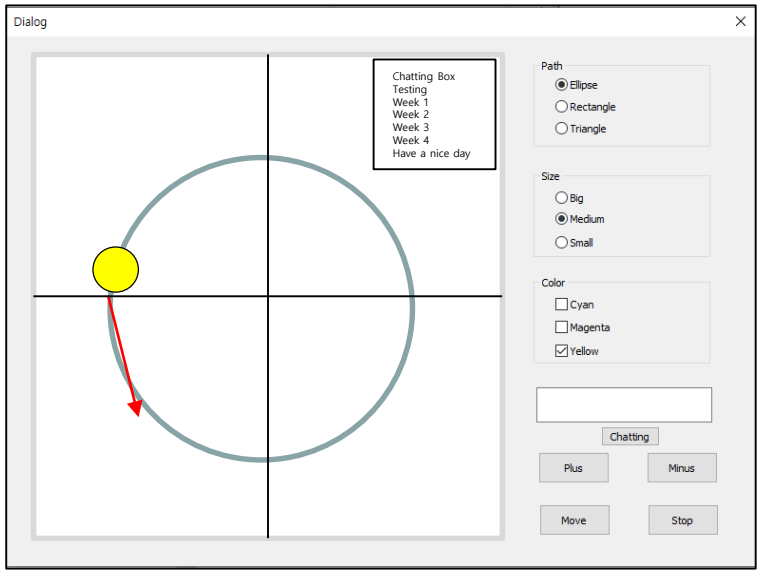
- **공전하는 원 만들기**
  - 메인 윈도우의 좌측:
    - 그림을 그리는 일반적인 차일드 윈도우:
      - 중앙을 원점으로 x축과 y축 좌표계를 그린다.
      - 좌표계를 그리고 선택된 형태의 도형으로 궤도를 그린다.
      - 원이 궤도를 따라 이동한다.
    - 우측 상단에 에디트 박스 (채팅박스): 우측에디트박스의 문자열을 가져와순서대로 출력하도록 한다.

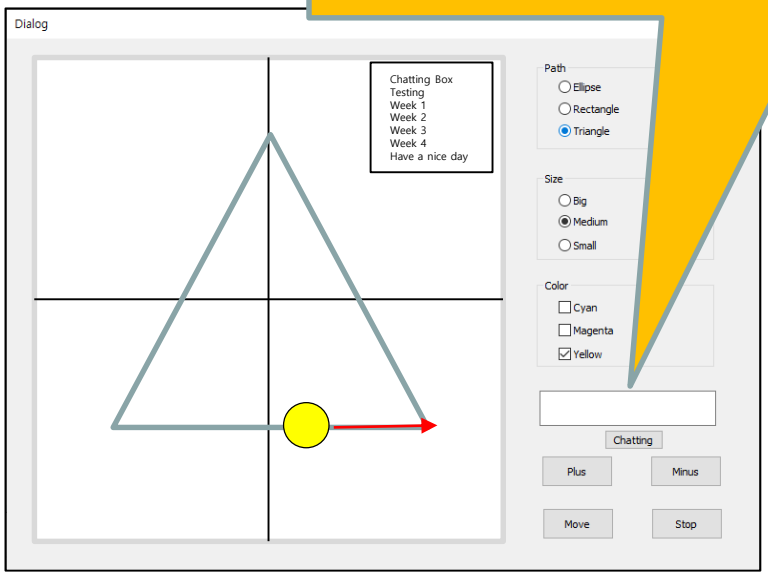
  - 우측에 차일드 윈도우를 사용하여 컨트롤을 놓는다.
    - 라디오 버튼1: 공전 궤도 종류 (원 / 사각형 / 삼각형)
    - 라디오 버튼2: 궤도의 크기 대 / 중 /소
    - 체크박스: 이동 원의 색을 Cyan/Magenta/Yellow
    - 에디트박스: 문자열 입력, 입력된 문자열은 좌측의 채팅박스에 순서대로 출력된다.
    - 버튼1: 에디트 박스의 텍스트가 채팅박스에 출력된다
    - 버튼2: 원이 그려진 경로에 따라 이동한다. 다시 누르면 반대 방향으로 이동한다.
    - 버튼3: 원의 이동 속도를 증가한다.
    - 버튼4: 원의 이동속도를 감소한다. 계속 감소하면 0이 된다.
    - 버튼5: 종료 버튼

실습 7-1


채팅 박스
Dialog
Path
Ellipse
Rectangle
Triangle
Size
Big
Medium
Small
Color
Cyan
Magenta
Yellow
Chatting
Plus
Minus
Move
Stop


Dialog
Path
Ellipse
Rectangle
Triangle
Size
Big
Medium
Small
Color
Cyan
Magenta
Yellow
Chatting
Plus
Minus
에디트박스에 작성한 텍스트가 채팅 버튼을 누르면 채팅 박스에 출력된다.


Dialog
Chatting Box
Testing
Week 1
Week 2
Week 3
Week 4
Have a nice day
Path
Ellipse
Rectangle
Triangle
Size
Big
Medium
Small
Color
Cyan
Magenta
Yellow
Chatting
Plus
Minus
Move
Stop


Dialog
Chatting Box
Testing
Week 1
Week 2
Week 3
Week 4
Have a nice day
Path
Ellipse
Rectangle
Triangle
Size
Big
Medium
Small
Color
Cyan
Magenta
Yellow
Chatting
Plus
Minus
Move
Stop

실습 7-2

- **차일드 윈도우와 컨트롤을 이용하여 연결된 그림 만들기: 횡으로 길게 연결된 배경 이미지 만들기**
  - 화면의 좌측에는 그림을 그릴 차일드 윈도우를 설정한다.
  - 최대 6개의 이미지 (좌우로 움직일 수 있는 이미지)를 연결하여 놓을 수 있도록 한다.
    - 6개의 버퍼가 있어, 각 버퍼에 이미지를 붙여 넣을 수 있다.
  - 차일드 윈도우 좌/우에는 그림을 연결할 화살표 버튼을 놓는다.
    - **화살표 버튼을 누르면 좌우로 이동한다. (좌 또는 우의 배경에 이미지를 좌우로 붙인다.) 즉,**
    - **좌방향 버튼:** 현재 차일드 윈도우에 출력된 배경 순서의 좌측으로 이동
      - 예) 현재 차일드 윈도우 3번 배경 ➔ 좌방향 버튼 ➔ 2번 배경
    - **우방향 버튼**: 현재 차일드 윈도우에 출력된 배경 순서의 우측으로 이동
      - 예) 현재 차일드 윈도우 3번 배경 ➔ 우방향 버튼 ➔ 4번 배경
  - 화면의 우측에는 리스트 박스를 이용하여 사용할 이미지 리스트를 나열하고, 선택할 수 있게 한다.
  - 이미지를 선택 후 선택 버튼을 눌러 선택된 이미지를 좌측의 차일드 윈도우에 출력한다.

  - **버튼**
    - 선택: 리스트 박스에서 이미지 선택
    - 완성: 이미지 연결이 완성. 더 이상 이미지를 추가할 수 없음.
    - 이동: 이미지가 좌측으로 움직인다.
    - 멈춤: 움직임이 멈춘다.
  - **에디트 박스 (좌측의 배경을 그리는 차일드 윈도우 위쪽)**
    - 현재 선택된 배경의 순서 출력

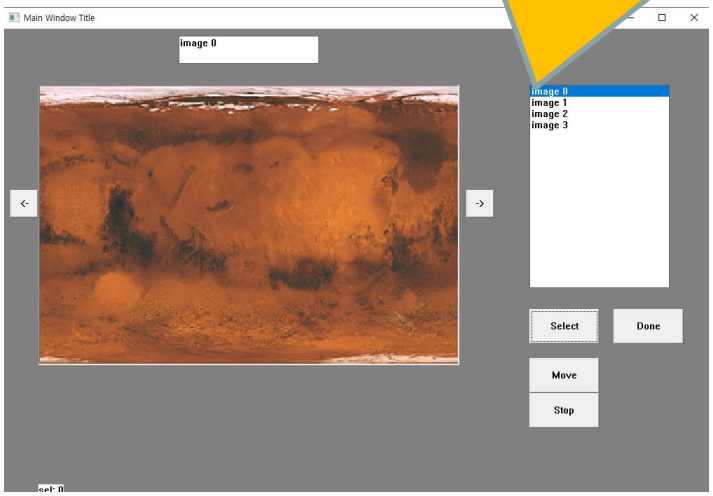
실습 7-2

현재 이미지 번호를 출력하는 에디트 박스: 현재 0번 버퍼

현재 이미지에서 이전 버퍼로 이동

현재 이미지에서 다음 버퍼로 이동

0번 이미지를 선택한 후 Select 버튼을 누름 ➔ 좌측 차일드 윈도우에 image0 그림출력



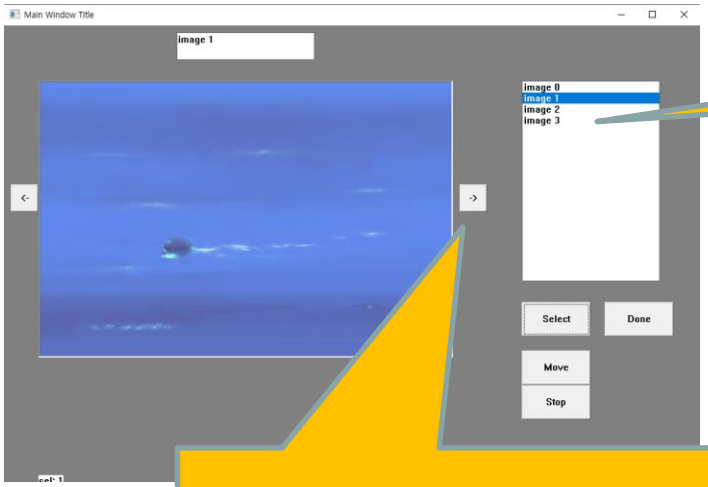
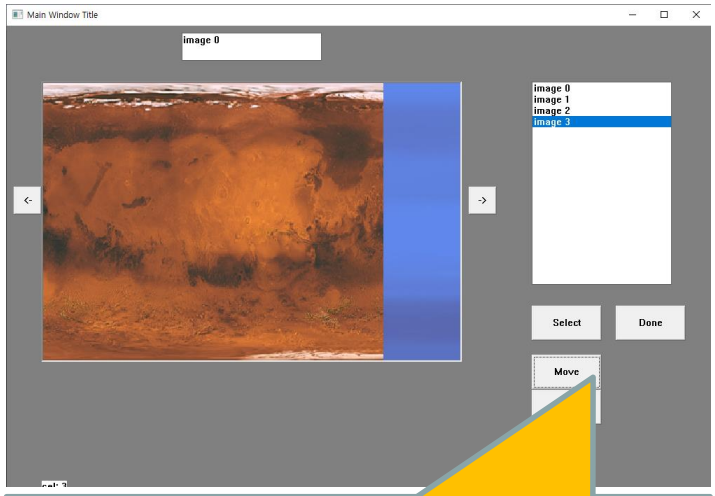


Image1을 선택한 후 Select 버튼을 누름 ➔ 좌측 차일드 윈도우에 image1그림 출력

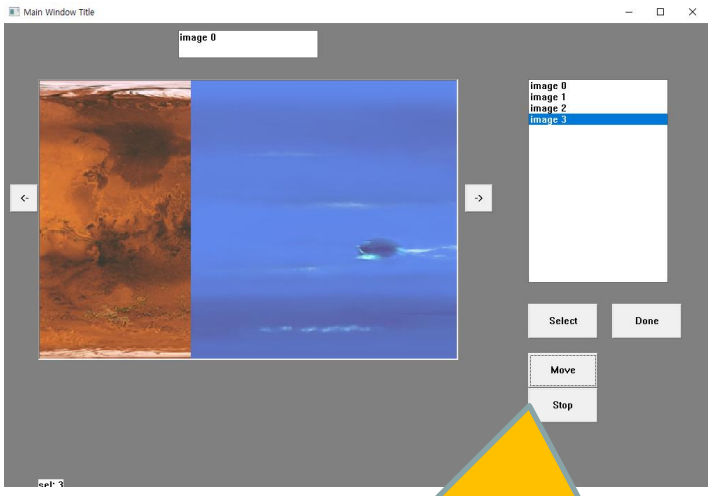
현재 0번 버퍼와 1번 버퍼에 각각 image0과 image1이 저장되어 있음. Done 버튼을 누르면 더 이상 이미지를 추가할 수 없음

우측 화살표 버튼을 누르면 ➔ 위의 에디트 박스에 1번 버퍼로 설정되어 있음

실습 7-2



Move 버튼을 누르면 → 좌측 이미지가 왼쪽으로 흘러간다
(0번 이미지 ➔ 1번 이미지 ➔ 2번 이미지 ➔ … ➔ 마지막 이미지 ➔ 첫번째 이미지 ➔ …)



Stop 버튼을 누를 때까지 계속 흘러간다.

- **이미지 버튼을 만들 경우: 윈도우 스타일에 BS_BITMAP 을 추가하고, 대화상자의 컨트롤과 마찬가지로 SendMessage 함수로 비트맵 이미지를 버튼 위에 올려놓는다.**

```
hButton[0] = CreateWindow ( "button",  "1-이동", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON
                          |BS_BITMAP, 10, 550, 50, 50, hwnd, (HMENU)IDC_BUTTON0, hInst, NULL);
hBit = LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP1));
SendMessage (hButton[0], BM_SETIMAGE, 0 /*IMAGE_BITMAP*/, (LPARAM)hBit);
```

이번주에는

- **이번주에는**
  - 차일드 윈도우 만들기
    - 컨트롤 만들기
    - 윈도우 분할하기

- **다음주에는**
  - 파일 입출력
  - 사운드 사용하기
  - 맵툴 만들기

- **이번주에는 1차시 강의만 있습니다. 실습은 2문제입니다!!**

- **이번주도 수고하셨습니다. 다음주에 만나요~**