

7장 차일드 윈도우

2024년도 1학기 윈도우 프로그래밍

- **학습목표**

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

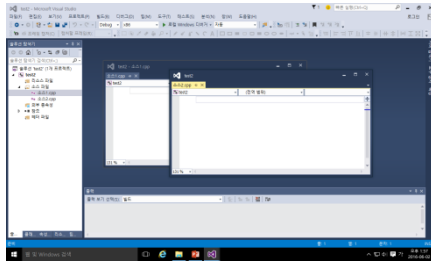
- **내용**

- 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.



MDI 윈도우



분할된 윈도우



차일드 윈도우

차일드 윈도우

• 차일드 윈도우는

- 기존의 윈도우를 만드는 방법으로 만든다
 - **CreateWindow, CreateWindowEx** 함수 사용
- 차일드 윈도우는 각자의 윈도우 클래스를 가진다.
 - 메인 윈도우 윈도우 클래스 외에 차일드 윈도우의 윈도우 클래스를 등록한다:
 - RegisterClass 또는 RegisterClassEx 함수 사용한다.
 - 1개 이상 등록된 윈도우 클래스는 클래스 이름으로 구분한다.
 - 차일드 윈도우는 자신의 윈도우 프로시저를 가질 수 있다. 기존의 윈도우 프로시저와 같은 형태로 만든다.
- 차일드 윈도우의 윈도우 스타일은 **WS_CHILD | WS_VISIBLE** 형태로 설정한다.
 - 위의 두 스타일 외에, **WS_BORDER**나 **WS_THICKFRAME** 등의 스타일을 같이 설정할 수 있다.
 - 가능한 윈도우 스타일
 - WS_OVERLAPPEDWINDOW: WS_CAPTION / WS_HSCROLL / WS_VSCROLL / WS_SYSMENU / WS_MAXIMIZEBOX / WS_MINIMIZEBOX / WS_THICKFRAME / WS_BORDER
 - WS_THICKFRAME: 크기를 바꿀 수 있다
 - WS_BORDER: 테두리만 있고 크기와 위치는 바꿀 수 없다
 - WS_POPUP 스타일은 WS_CHILD와 같이 설정할 수 없다.
- 차일드 윈도우 안에 또 다른 차일드 윈도우를 가질 수 있다.
- 컨트롤들을 차일드 윈도우로 만들 수 있다.

1. 일반적인 차일드 윈도우 만들기

- 일반적인 윈도우를 차일드 윈도우로 만들기

- 윈도우 클래스 등록하여 해당 윈도우 클래스로 차일드 윈도우를 만든다 .

```
int WINAPI wWinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX    wc;
    HWND hWnd;

    //--- 윈도우 클래스를 등록한다.
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = L"ParentClass";
    wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    RegisterClassEx(&wc);

    //--- 차일드 윈도우 클래스를 등록한다.
    wc.hCursor = LoadCursor(NULL, IDC_HELP);
    wc.hbrBackground = (HBRUSH) GetStockObject (GRAY_BRUSH);
    wc.lpszClassName = L"ChildClass";
    wc.lpfnWndProc = ChildProc;
    RegisterClassEx(&wc);

    hWnd = CreateWindow ( L"ParentClass", NULL, WS_OVERLAPPEDWINDOW, 0, 0, 800, 600, NULL, NULL, hInstance, NULL);
    ...
}
```

```
// 클래스 스타일
// 윈도우 프로시저 지정
// 윈도우 클래스 데이터 영역
// 윈도우의 데이터 영역
// 인스턴스 핸들
// 아이콘 핸들
// 사용할 커서 핸들
// 바탕색 브러쉬 핸들
// 메뉴 이름
// 윈도우 클래스 이름

// 윈도우 클래스를 등록
```

```
// 차일드 윈도우 클래스 이름
// 차일드 윈도우 프로시저 지정
// 자식 윈도우 클래스를 등록
```

차일드 윈도우 만들기

- 윈도우 프로시저 : create 메시지 (윈도우가 만들어질 때 불리는 메시지)에서 차일드 윈도우 만들기

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HWND child_hWnd;
    switch (iMsg)
    {
        case WM_CREATE:
            child_hWnd = CreateWindow ( L"ChildClass", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER | WS_THICKFRAME,
                                     10, 10, 400, 500, hWnd, NULL, g_hInst, NULL);

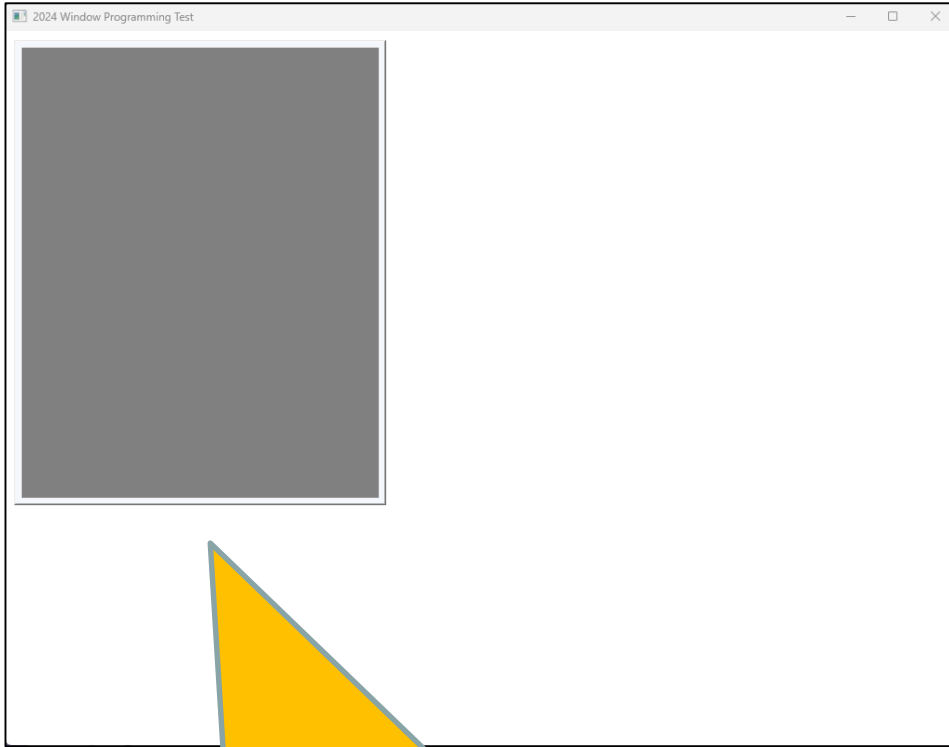
            break;
    }
    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

- 차일드 윈도우 프로시저

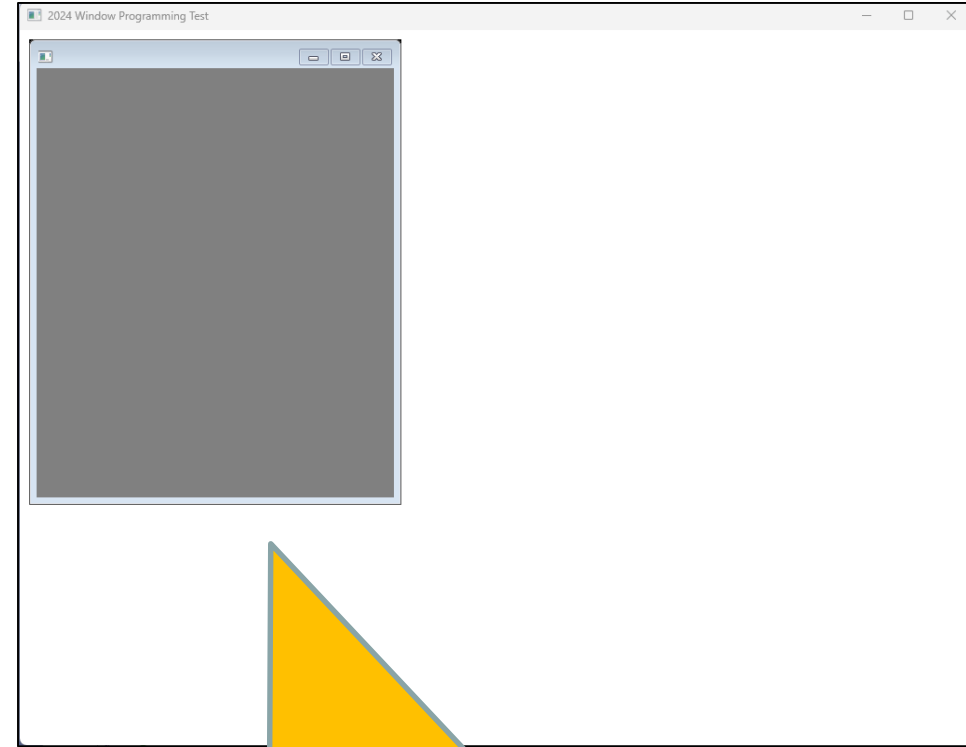
```
LRESULT CALLBACK ChildProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    switch (iMsg) {
        case WM_LBUTTONDOWN:    //-- 차일드 윈도우에 좌측 마우스 버튼을 누른 경우
            MessageBox (hWnd, L"Left Mouse Button", L"Mouse Test ", MB_OK);
            break;
        case WM_DESTROY:
           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

차일드 윈도우 만들기

- 결과 화면



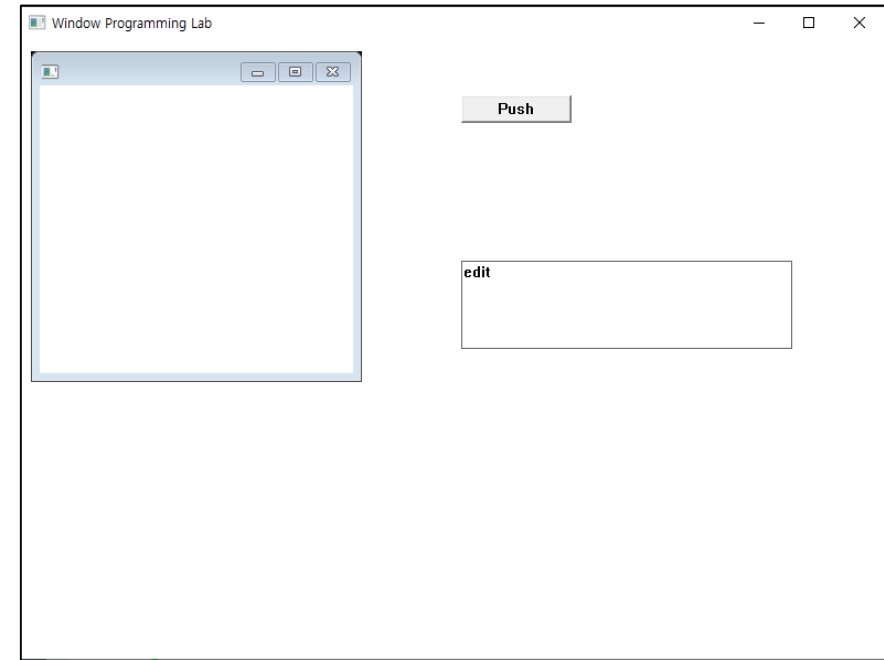
```
child_hWnd = CreateWindow ( L"ChildClass", NULL,  
    WS_CHILD | WS_VISIBLE | WS_BORDER | WS_THICKFRAME,  
    10, 10, 200, 500, hWnd, NULL, g_hInst, NULL);  
/-- 기본 테두리가 있는 차일드 윈도우 만들기
```



```
child_hWnd = CreateWindow ( L"ChildClass", NULL,  
    WS_CHILD | WS_VISIBLE | WS_OVERLAPPEDWINDOW,  
    10, 10, 200, 500, hWnd, NULL, g_hInst, NULL);  
/-- 다른 스타일로 차일드 윈도우 만들기
```

2. 컨트롤 차일드 윈도우

- 대화상자에서 사용했던 컨트롤:
 - 윈도우로 부모 윈도우 아래의 자식 윈도우로 존재한다.
 - 기본적으로 **CreateWindow** 함수로 자식 윈도우를 만든다.
 - **컨트롤의 윈도우 클래스는 이미 등록되어 있다.**
 - 예) 버튼 → button
 - 예) 에디트 박스 → edit
 - **컨트롤 윈도우의 아이디는 정수로 직접 정의한다.**
 - 컨트롤의 아이디를 정수로 정의한다.
 - 예) #define IDC_BUTTON 100
 - 예) #define IDC_EDIT 101
 - 컨트롤 윈도우도 CreateWindow 함수로 만든다.
 - 따로 윈도우 프로시저를 만들지 않고 커맨드 메시지로 처리한다.



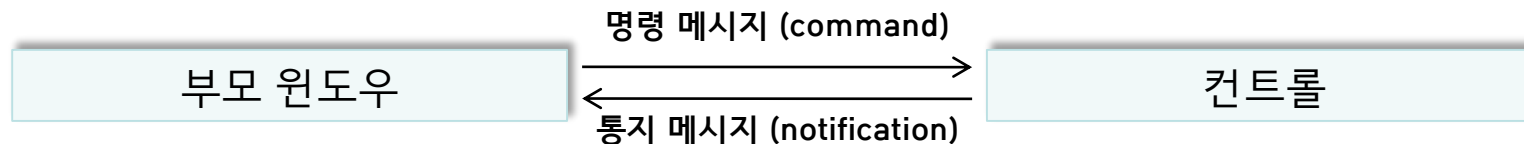
2. 컨트롤 차일드 윈도우

- 컨트롤 차일드 윈도우 클래스 이름: 문자열로 클래스 이름 정의

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



2. 컨트롤 차일드 윈도우

- 차일드 윈도우 만들기

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, 에디트 박스 클래스: edit, 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) 버튼 차일드 윈도우: WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON
- x, y: 윈도우 위치 좌표값
- nWidth, nHeight: 윈도우 크기 (폭, 높이)
- **hWndParent**: 부모 윈도우 핸들
- **hMenu**: 차일드 윈도우 아이디
- hInstance: 인스턴스 핸들

1) 버튼 컨트롤 만들기

- 버튼의 종류

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_AUTOCHECKBOX	자동 체크 박스
	BS_AUTO3STATE	3가지 상태를 가지는 자동 체크 박스
	BS_RADIOBUTTON	라디오 버튼
	BS_AUTORADIOBUTTON	자동 라디오 버튼
	BS_GROUPBOX	그룹 박스

버튼 만들기

- 버튼 만들기

```
#define IDC_BUTTON 100
```

```
HWND hButton;
```

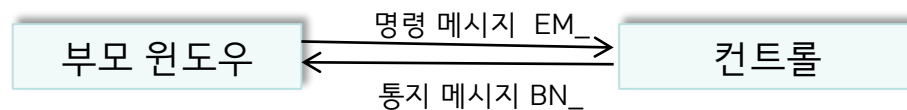
```
hButton = CreateWindow (L"button", L"OK", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 200, 0, 100, 25, hwnd,  
                        (HMENU) IDC_BUTTON, g_hInst, NULL);
```

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

통지 정보	의미
BN_CLICKED	버튼 위에서 마우스가 클릭되었을 때
BN_DBLCLK	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;

    switch (iMsg)
    {
        case WM_CREATE:
            hButton = CreateWindow ( L"button", L"OK",                                //--- 버튼의 윈도우 클래스 이름은 "button"
                                   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,          //--- 차일드 윈도우이고 누르는 형태의 버튼 스타일
                                   200, 0, 100, 25, hWnd, (HMENU) IDC_BUTTON, g_hInst, NULL);
            break;

        case WM_COMMAND:
            switch(LOWORD(wParam)) {
                case IDC_BUTTON:
                    hDC = GetDC(hWnd);
                    TextOut (hdc, 0, 100, L"Hello World", 11);
                    ReleaseDC (hWnd, hDC);
                    break;
            }
            break;
    }
}
```

체크박스 만들기

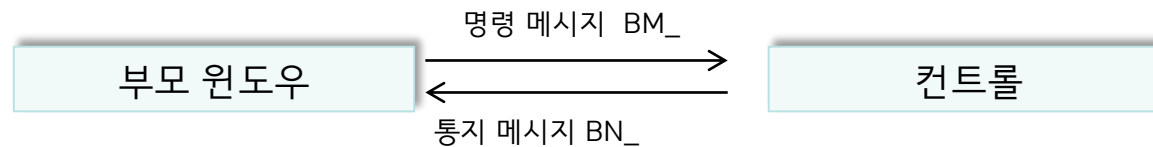
- 체크박스 만들기

```
#define IDC_CHECK 101
```

```
HWND hCheck;
```

```
hCheck = CreateWindow ( L"button", L"check box test", WS_CHILD | WS_VISIBLE | BS_CHECKBOX, 100, 0, 100, 25, hwnd,  
                      (HMENU) IDC_CHECK, g_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
<code>BM_GETCHECK</code>	체크박스가 현재 체크되어 있는 상태인지 조사	<ul style="list-style-type: none">• <code>BST_CHECKED</code>: 현재 체크되어 있다.• <code>BST_UNCHECKED</code>: 현재 체크되어 있지 않다.• <code>BST_INDETERMINATE</code>: 체크도 아니고 비 체크도 아닌 상태
<code>BM_SETCHECK</code>	체크 박스의 체크 상태를 변경, wParam에 변경할 체크상태를 보내준다	

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

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

체크박스 만들기

- 체크박스 버튼 컨트롤 (수동)

```
#define IDC_BUTTON 101
```

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

```
    switch (iMsg) {
```

```
        case WM_CREATE:
```

```
            hCheck = CreateWindow ( L"button", L"Grid",
```

```
                WS_CHILD | WS_VISIBLE | BS_CHECKBOX,
                10, 210, 180, 40, hWnd, (HMENU) IDC_BUTTON,
                g_hInst, NULL);
```

```
//--- 윈도우 클래스 이름은 button
```

```
//--- 차일드 윈도우, 수동 체크 박스
```

```
            break;
```

```
        case WM_COMMAND:
```

```
            switch (LOWORD(wParam) ) {
```

```
                case IDC_BUTTON5: // grid check box
```

```
                    if (SendMessage (hCheck, BM_GETCHECK, 0, 0) == BST_UNCHECKED) {
```

```
                        SendMessage (hCheck, BM_SETCHECK, BST_CHECKED, 0);
```

```
                        cList[0] = 1;
```

```
                    }
```

```
                    else {
```

```
                        SendMessage (hCheck, BM_SETCHECK, BST_UNCHECKED, 0);
```

```
                        cList[0] = 0;
```

```
                    }
```

```
                    break;
```

```
            }
```

```
        }
```

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

```
    }
```


체크박스 만들기

- 체크박스 버튼 컨트롤 (자동)

```
#define IDC_BUTTON 101
```

```
LRESULT CALLBACK WndProc ( HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
static HWND hCheck;
```

```
static int cList[2];
```

```
switch (iMsg) {
```

```
    case WM_CREATE:
```

```
        hCheck = CreateWindow ( L"button", L"Grid",
```

```
                                WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
```

```
                                10, 210, 180, 40, hWnd, (HMENU) IDC_BUTTON,
```

```
                                g_hInst, NULL);
```

```
// 윈도우 클래스 이름은 button
```

```
// 차일드 윈도우, 자동 체크 박스
```

```
        break;
```

```
    case WM_COMMAND:
```

```
        switch (LOWORD(wParam) ) {
```

```
            case IDC_BUTTON5:
```

```
// grid check box: 자동 체크박스로 만들어 체크박스의 상태만 조사한다.
```

```
                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 만들기

```
#define IDC_RADIO 102
```

```
HWND hRadio;
```

```
hRadio = CreateWindow ( L"button", L"radio button test", 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**
 - **BS_RADIOBUTTON 스타일:** 라디오 버튼 상태가 수동으로 바뀌어진다.
 - **BS_AUTORADIOBUTTON 스타일:** 라디오 버튼 상태가 자동으로 바뀌어진다.
 - **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

```
#define ID_R1    102
#define ID_R2    103
#define ID_R3    104

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, 5, 5, 120, 110, hWnd, (HMENU)0, g_hInst, NULL);

        //--- 버튼 만들기: 그룹 1
        r1= CreateWindow (L"button", L"Rectangle", WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON | WS_GROUP,
                        10, 20, 100, 30, hWnd, (HMENU) ID_R1, g_hInst, NULL);
        r2= CreateWindow (L"button", L"Ellipse", WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON, 10, 50, 100, 30, hWnd, (HMENU) ID_R2, g_hInst, NULL);
        r3= CreateWindow (L"button", L"Line", WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON, 10, 80, 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

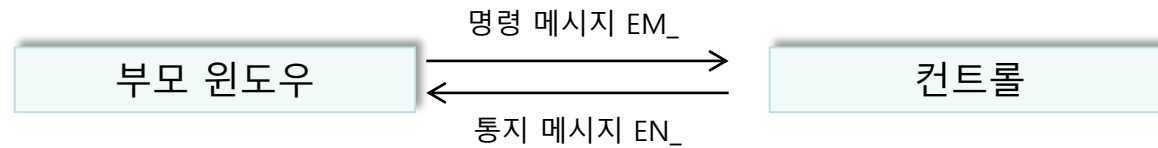
- 에디트 컨트롤 만들기

```
#define IDC_EDIT 200
```

```
HWND hEdit;
```

```
hEdit = CreateWindow ( L"edit", L"editing", 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 스타일과 통지 메시지

	클래스 이름	스타일	의미
에디트 컨트롤의 윈도우 스타일	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.

	메시지	의미
에디트 컨트롤의 통지 메시지	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 200 // 에디트 컨트롤의 ID

HWND hButton, hEdit;
char str[100];

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    case WM_CREATE:
        //--- 박스 주위에 테두리가 있는 에디트 컨트롤스타일
        hEdit = CreateWindow (L"edit", L"editing", WS_CHILD | WS_VISIBLE | WS_BORDER, 0, 0, 200, 25, 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, 0, 100, str, strlen(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_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

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

콤보 박스 만들기

```
#define IDC_BUTTON  100
#define IDC_EDIT    200
#define IDC_COMBO  300           //--- 콤보 박스 컨트롤의 ID

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

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    switch (iMsg)
    {
        case WM_CREATE:
            //--- 크기가 200x300인 콤보박스
            hCombo = CreateWindow (L"combobox", NULL, WS_CHILD | WS_VISIBLE | CBS_DROPDOWN, 0, 100, 200, 300, hWnd, (HMENU) IDC_COMBO, g_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 (strcmp(str, ""))
                        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

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 얻어 오는 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 lParam: 텍스트 스트링의 시작 주소

리스트 박스 만들기

```
#define ID_LISTBOX 400

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

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    int i;
    switch(iMsg) {
        case WM_CREATE:
            /*--- 크기가 100x200인 리스트 박스
            hList=CreateWindow (L"listbox", NULL, WS_CHILD|WS_VISIBLE|WS_BORDER|LBS_STANDARD , 0, 0, 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, iMsg, wParam, lParam));
}
```

3. 윈도우 분할하기

- 메인 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.)



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 썩 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 ; // 윈도우 클래스 변수 선언

//--- 메인 윈도우 클래스 생성 및 등록
wndClass.cbSize = sizeof(wndclass) ;
wndClass.style = CS_HREDRAW | CS_VREDRAW ;
wndClass.lpfnWndProc = WndProc; // 메인 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) ;
wndCclass.lpszClassName = L"Window Class Procedure"; // 메인윈도우 클래스 이름: 클래스 구분자
wndClass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
RegisterClassEx(&wndClass); // 메인 윈도우 클래스 등록

//--- 차일드 윈도우 클래스 생성 및 등록 : 차일드를 위해 wndClass를 재사용
wndClass.lpfnWndProc = ChildWndProc; // 차일드윈도우 프로시저
wndClass.lpszMenuName = NULL ;
wndClass.lpszClassName = L"Child Window Class Procedure"; // 차일드윈도우 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.
- 두 차일드 윈도우는 같은 윈도우 프로시저를 사용한다.
- 각각의 윈도우에 타이머를 설정: 원이 우측으로 이동

```
HWND ChildHwnd[2];           //--- 차일드 윈도우 핸들을 전역변수로 선언
```

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

            ChildHwnd[0] = CreateWindowEx ( WS_EX_CLIENTEDGE, L"Child Window Class Procedure", NULL,
                                             WS_CHILD | WS_VISIBLE, 0, 0, rectView.right, rectView.bottom/2-1, hWnd, NULL, hInst, NULL );

            ChildHwnd[1] = CreateWindowEx ( WS_EX_CLIENTEDGE, L"Child Window Class Procedure", NULL,
                                             WS_CHILD | WS_VISIBLE, 0, rectView.bottom/2+1, rectView.right, rectView.bottom/2-1, hWnd, NULL, hInst, NULL );
            break;
    }
}
```

- **추가할 수 있는 스타일**

- WS_CLIPCHILDREN: 부모 윈도우의 무효화 영역에 자식 윈도우가 포함되어도 이들에게 WM_PAINT 메시지를 보내지 않는다.
- WS_CLIPSIBLINGS: 부모 윈도우에 발생한 WM_PAINT 메시지가 자식들에게 전달되는 경우 자식 윈도우들 역시 불필요한 WM_PAINT 메시지를 처리하는 것을 방지한다.

윈도우 분할: 윈도우 프로시저

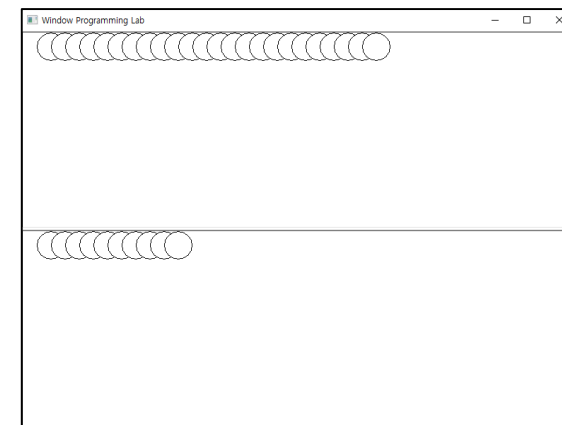
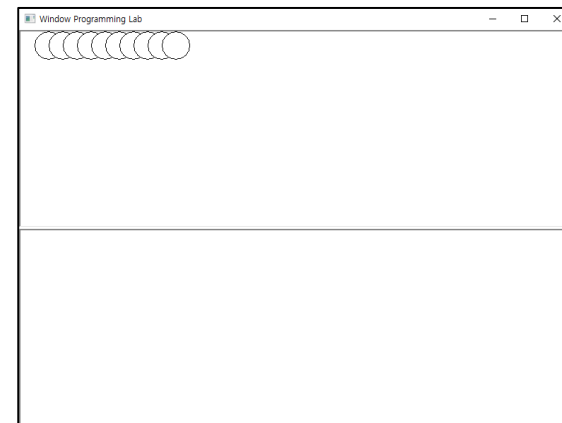
- 차일드 윈도우 프로시저

```
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];
    int select;

    switch (iMsg) {
        case WM_TIMER:
            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])          //-- 어떤 차일드 윈도우인지 찾기
                select = 0;

            else
                select = 1;
            flag[select] = 1 - flag[select];
            if (flag[select])
                SetTimer(hWnd, select, 100, NULL);
            else
                KillTimer(hWnd, select);
            break;
    }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```



윈도우 분할: 윈도우 다루기 함수들

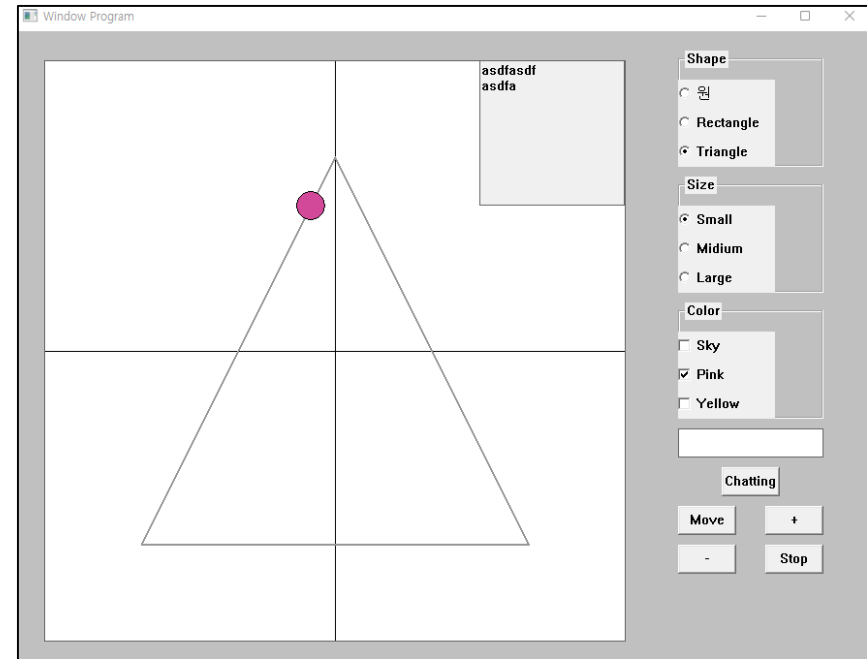
- 윈도우 다루기 함수

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 hWnd) / 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
BOOL IsChild (HWND hWndParent, HWND hWnd);	hWnd 윈도우가 차일드 윈도우인지 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

- 좌측에 차일드 윈도우를 만들고 중앙을 원점으로 x축과 y축 좌표계를 그린다.
 - 좌표계에 선택된 형태의 경로를 그린다.
 - 원이 경로를 따라 이동한다.
- 우측에 컨트롤을 놓는다.
 - 라디오 버튼: 경로 종류 (원 / 사각형 / 삼각형)
 - 라디오 버튼: 원의 크기 대 / 중 / 소
 - 색상: 움직이는 원의 색상 3종류
 - 에디트 박스: 문자열 입력
 - 버튼1: 에디트 박스의 문자열을 채팅 창에 출력
 - 버튼2: 원 이동
 - 버튼3: 이동 속도 증가
 - 버튼4: 이동 속도 감소
 - 버튼5: 이동 멈춤



• 차일드 윈도우와 컨트롤을 이용하여 연결된 그림 만들기

- 화면의 좌측에는 그림을 그릴 차일드 윈도우를 설정한다.
- 차일드 윈도우 좌/우에는 그림을 연결할 화살표 버튼을 놓는다.
 - 화살표 버튼을 누르면 좌우로 이동한다. (이미지를 좌우로 붙인다.)
 - 이미지 버튼을 만들 경우: 윈도우 스타일에 BS_BITMAP 을 추가하고, 대화상자의 컨트롤과 마찬가지로 SendMessage 함수로 비트맵 이미지를 버튼 위에 올려놓는다.

```
hButton[0] = CreateWindow ( L"button", L"Move", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON | BS_BITMAP, 10, 550, 50, 50, hwnd, (HMENU)IDC_BUTTON0, g_hInst, NULL);
```

```
hBit = LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP1));
```

```
SendMessage (hButton[0], BM_SETIMAGE, 0 /*IMAGE_BITMAP*/, (LPARAM)hBit);
```

- 화면의 우측에는 콤보 박스 또는 리스트 박스를 이용하여 사용할 이미지 리스트를 선택할 수 있게 한다.
- 이미지를 선택 후 선택 버튼을 눌러 선택된 이미지를 좌측의 차일드 윈도우에 놓는다.
- 최대 10개의 이미지 (좌우로 움직일 수 있는 이미지)를 연결하여 놓을 수 있도록 한다.
- 컨트롤 만들기
 - 버튼

- Select: 리스트 박스에서 이미지 선택
- Move: 이미지가 좌측으로 움직인다.
- Stop: 움직임이 멈춘다.
- Done: 이미지 연결이 완성

• 에디트 박스

- 현재 선택된 이미지의 이름 출력

