

# WinAPI

# WinAPI란?

- ▶ 프로그래머들이 Windows에서 동작하는 app을 만들기 위해 제공하는 소스코드
- ▶ 과거에는 Win32 API라고 했지만 이제는 **64비트**를 **사용**하기 때문에 통합하여 **WinAPI**라 부른다

# 핸들(Handle)과 인스턴스(Instance)

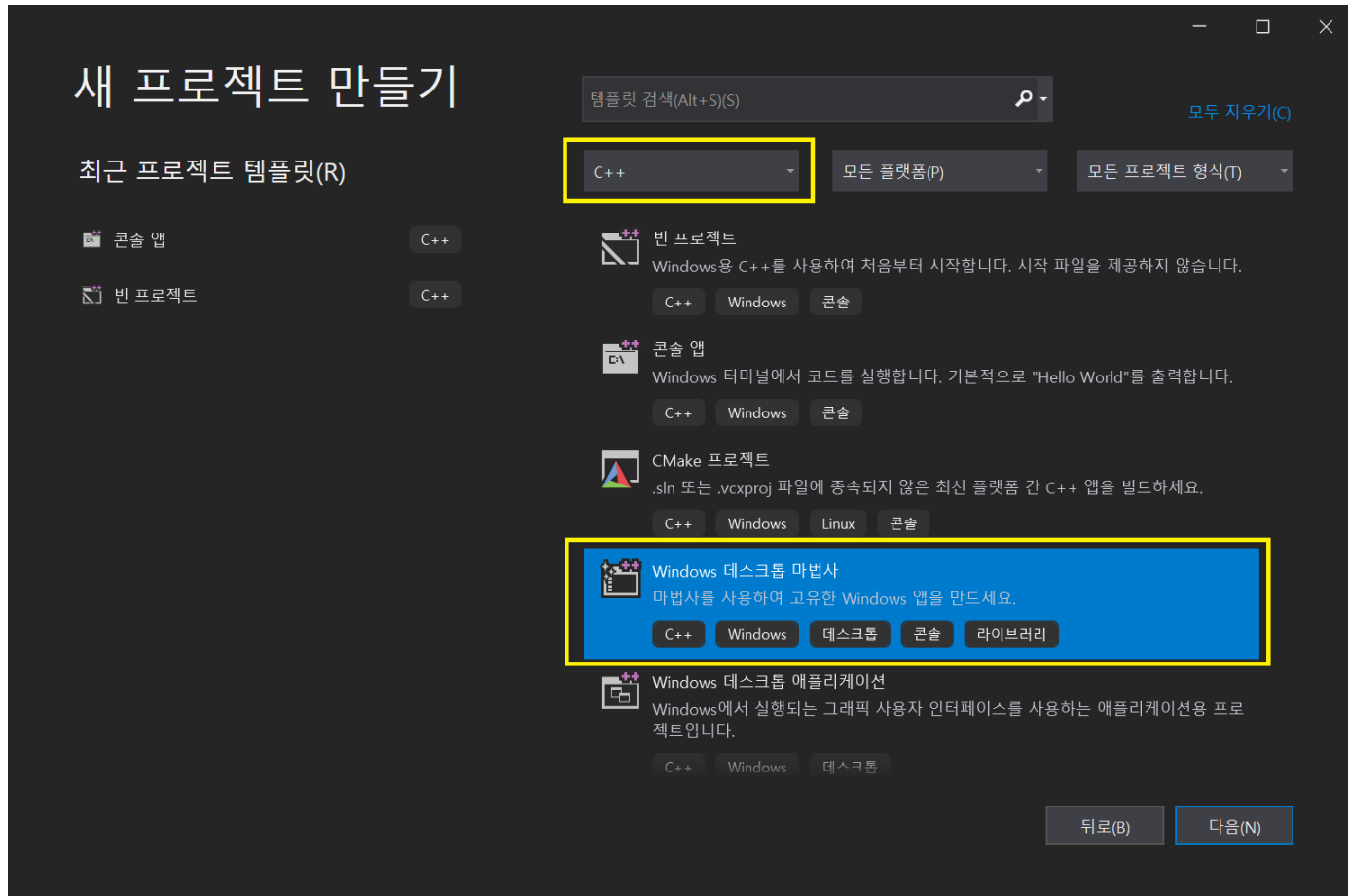
## ▶ 핸들

- 운영체제가 안전하게 리소스를 관리하기 위하여 사용자에게 핸들 값을 알려준다
- 운영체제 내부에 있는 어떤 리소스의 주소를 정수로 치환한 값
- 운영체제가 할당하는 자원의 아이디

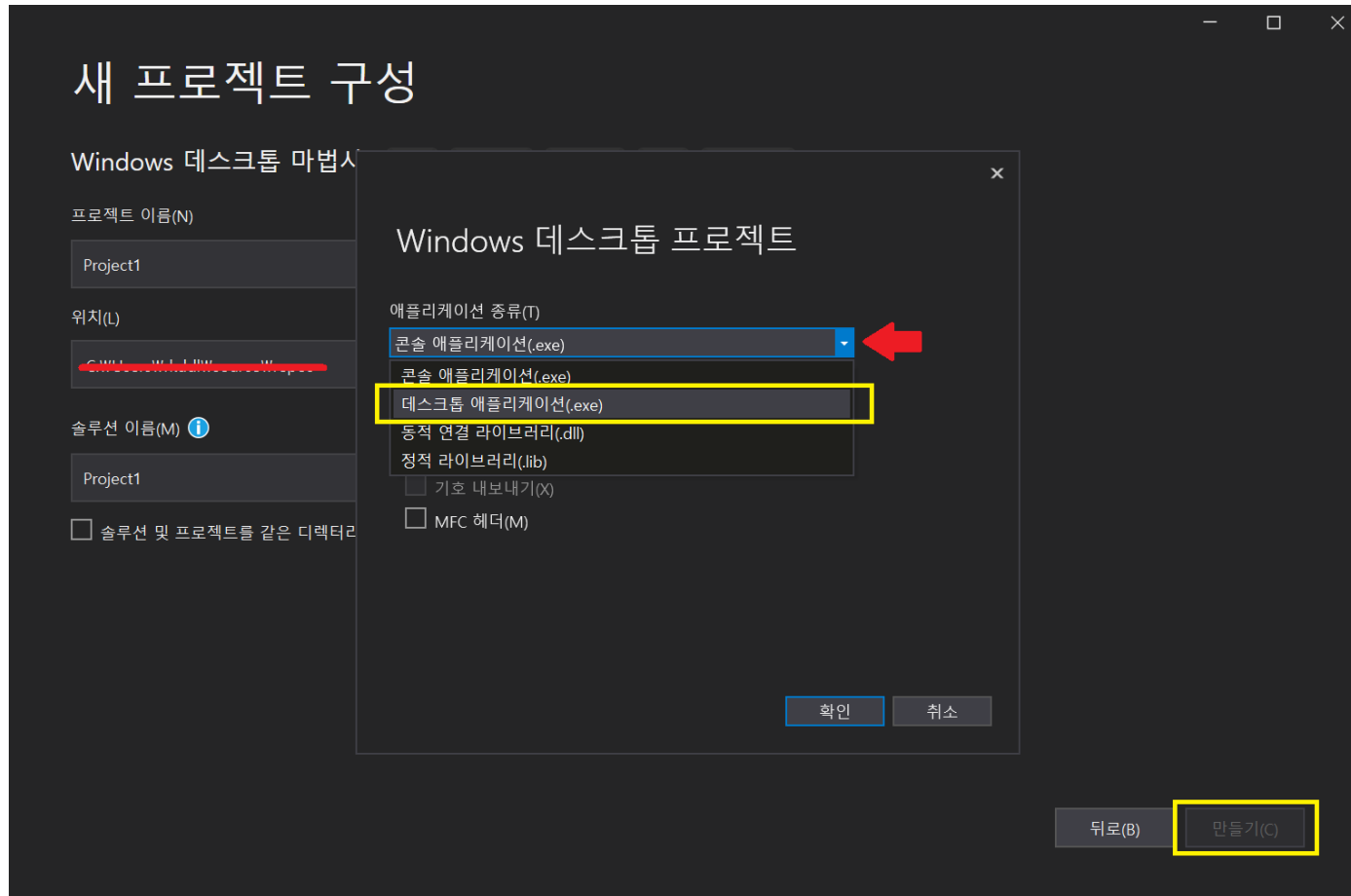
## ▶ 인스턴스

- 응용프로그램(App)의 아이디
- 각 프로그램들은 저마다 다른 인스턴스를 지닌다

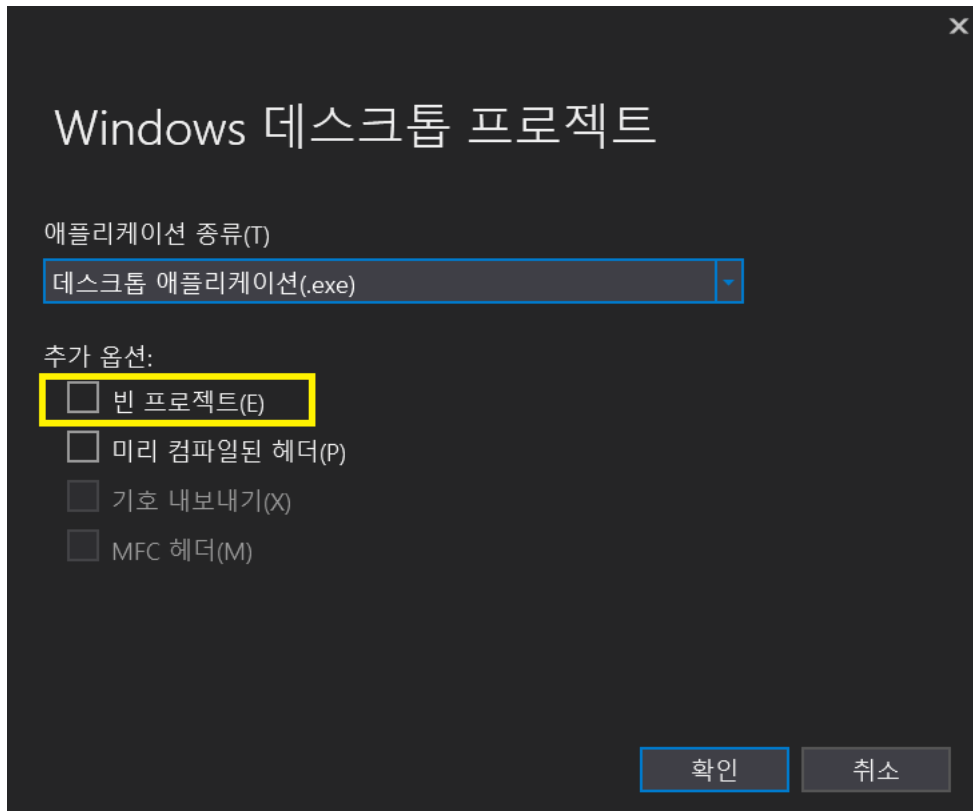
# Win API 프로젝트 만들기



# Win API 프로젝트 만들기



# Win API 프로젝트 만들기



Visual Studio 2017에서 부터는 이전과 **프로젝트 생성** 방식이 다르게 **변경** 되었다

이후 버전에서는 이와 같은 방법으로 Win API 프로젝트를 생성하면 된다

**빈 프로젝트 체크를 해제!! 후 확인**

# 주요 기본 생성 함수

- ▶ **wWinMain()**

- 윈도우 생성과 메시지 처리 등이 되는 메인 함수

- ▶ **MyRegisterClass()**

- 윈도우 구조체로 생성할 윈도우를 정의하여 등록한다

- ▶ **InitInstance()**

- 등록된 윈도우 구조체를 이용하여 윈도우를 만든다

- ▶ **WndProc()**

- 윈도우 메시지가 발생하면 호출되며 해당 메시지 처리를 할 수 있다

# WinAPI의 Main

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ LPWSTR lpCmdLine, _In_ int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    // 전역 문자열을 초기화합니다.
    LoadStringW(hInstance, IDC_WINAPI, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    // 애플리케이션 초기화를 수행합니다:
    if (!InitInstance (hInstance, nCmdShow)) return FALSE;
    MSG msg;
    // 기본 메시지 루프입니다:
    while (true)
    {
        if (PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE))
        {
            if (WM_QUIT == msg.message) break;

            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return (int) msg.wParam;
}
```



# 윈도우 클래스 정의

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance     = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_WINAPI));
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = NULL;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = NULL;

    return RegisterClassExW(&wcex);
}
```

# 윈도우 생성

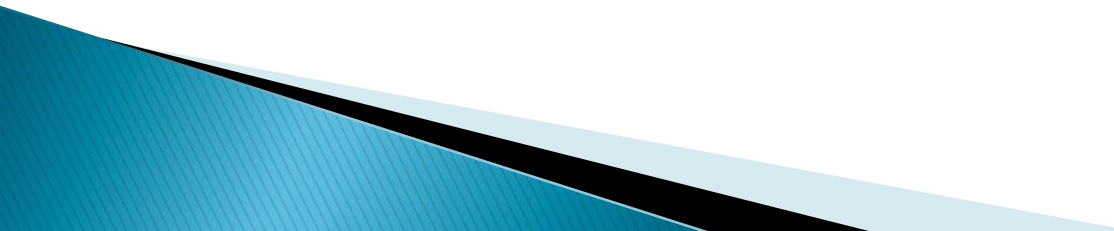
```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance;

    HWND hWnd = CreateWindowW(szWindowClass, L"타이틀 이름!!", WS_CAPTION |
    WS_SYSMENU | WS_MINIMIZEBOX, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr,
    nullptr, hInstance, nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}
```



# 윈도우 프로시저

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        //그리기는 게임루프에서 처리.
        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: 여기에 hdc를 사용하는 그리기 코드를 추가합니다...
            EndPaint(hWnd, &ps);
        }
        break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

# DC(Device Context)

- ▶ 출력에 필요한 모든 정보를 가진 데이터 구조체
- ▶ GDI(Graphic Device Interface)에 의하여 관리된다
- ▶ HDC GetDC(HWND hWnd)
- ▶ int ReleaseDC(HWND hWnd, HDC hDC)
  - GetDC를 이용하여 **DC를 사용 후 반드시 ReleaseDC를 사용하여 해제**

# WM\_PAINT

- ▶ 무효화 영역의 화면이 다시 보이게 되면 **자동으로 호출**이 된다
- ▶ **무효화 영역** : 클라이언트 화면이 최소화 상태나 다른 무언가에 의해 가려져 화면에 보이지 않는 영역
- ▶ HDC WINAPI BeginPaint(HWND hWnd, LPPAINTSTRUCT lpPaint)
- ▶ BOOL WINAPI EndPaint(HWND hWnd, CONST PAINTSTRUCT \*lpPaint)
  - BeginPaint : 무효화 영역을 자동으로 알아와 주며 사용할 DC를 한 번에 발급하여 준다
  - EndPaint : 무효화 영역을 유효화 영역으로 변경하고 DC를 해제하여 준다
- ▶ BOOL InvalidateRect(HWND hWnd, const RECT\* lpRect, BOOL bErase)
  - 지정 영역을 무효화 영역으로 하여, WM\_PAINT를 강제로 호출, 클라이언트 화면을 갱신하게 한다
  - lpRect : 해당 사각형 영역을 무효화 영역으로 한다, NULL일 경우는 클라이언트 전체
  - bErase : true 화면 전체를 지우고 새로 그린다, false 화면을 지우지 않고 그린다

# 문자 출력

- ▶ **BOOL TextOut(HDC hdc, int x, int y, LPCWSTR lpString, int c)**
  - lpString에 있는 문자열을 c(길이)만큼 x, y좌표에 출력하여준다
- ▶ **int DrawText( HDC hdc, LPCTSTR lpString, int nCount, LPRECT lpRect, UINT uFormat)**
  - lpString에 있는 문자열을 nCount(길이)만큼 lpRect(사각형영역)안에 uFormat형식으로 출력하여 준다
  - nCount : **-1** 일 경우 null문자열을 포함하여 **알아서 길이를 지정**해준다
  - uFormat :

값	설명
DT_LEFT	수평 왼쪽 정렬한다.
DT_RIGHT	수평 오른쪽 정렬한다.
DT_CENTER	수평 중앙 정렬한다.
DT_BOTTOM	사각 영역의 바닥에 문자열을 출력한다.
DT_VCENTER	사각 영역의 수직 중앙에 문자열을 출력한다.
DT_WORDBREAK	사각영역의 오른쪽 끝에서 자동 개행되도록 한다.
DT_SINGLELINE	한줄로 출력한다.
DT_NOCLIP	사각 영역의 경계를 벗어나도 문자열을 자르지 않고 그대로 출력한다.

# 문자 출력

## ▶ UINT SetTextAlign(HDC hdc,UINT fMode)

- Text의 중심 좌표를 설정한다
- fMode : 두 개 이상의 플래그로 or연산이 가능
- CP : Current Position



값	설명
TA_TOP	지정한 좌표가 상단좌표가 된다.
TA_BOTTOM	지정한 좌표가 하단 좌표가 된다.
TA_CENTER	지정한 좌표가 수평 중앙 좌표가 된다.
TA_LEFT	지정한 좌표가 수평 왼쪽 좌표가 된다.
TA_RIGHT	지정한 좌표가 수평 오른쪽 좌표가 된다.
TA_UPDATECP	지정한 좌표대신 CP를 사용하며 문자열 출력후에 CP를 변경한다.
TA_NOUPDATECP	CP를 사용하지 않고 지정한 좌표를 사용하며 CP를 변경하지 않는다.

# 문자 출력

```
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    // TODO: 여기에 hdc를 사용하는 그리기 코드를 추가합니다...

    // TextOut()을 이용하여 문자 출력.
    std::string str = "TEST test TEST";
    TextOut(hdc, 10, 10, str.c_str(), str.length());

    // DrawText()를 이용하여 문자 출력.
    str = "TEST test TEST test TEST test TEST test TEST test TEST test TEST test TEST";
    RECT rect = {100, 100, 400, 300};
    DrawText(hdc, str.c_str(), -1, &rect, DT_CENTER | DT_WORDBREAK);

    EndPaint(hWnd, &ps);
}
```



# 문자 출력

- ▶ **COLORREF SetTextColor(HDC hdc, COLORREF color)**
  - 출력 **문자의 색상**을 변경하고 **이전 색상을 리턴** 한다
  - color : **RGB**(?, ?, ?) 값은 0~255의 정수
- ▶ **COLORREF SetBkColor(HDC hdc, COLORREF color)**
  - 출력 **문자의 배경색**을 변경하고 **이전 배경색을 리턴**한다
  - color : **RGB**(?, ?, ?) 값은 0~255의 정수

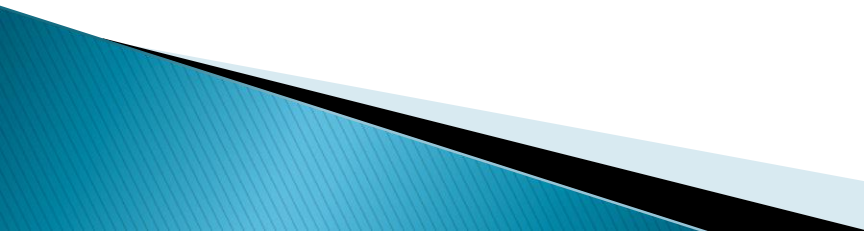
# 문자 출력

```
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);

    SetTextColor(hdc, RGB(255,0,0)); // 문자 색을 붉은색으로 변경.
    SetBkColor(hdc, RGB(0,0,0));    // 문자의 배경색을 검은색으로 변경.

    std::string str = "TEST test TEST test";
    TextOut(hdc, 10, 10, str.c_str(), str.length());

    EndPaint(hWnd, &ps);
}
```



# 그래픽 출력

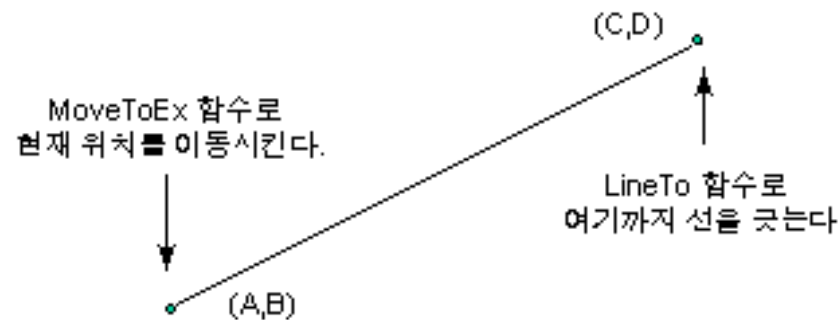
- ▶ **COLORREF SetPixel(HDC hdc, int X, int Y, COLORREF crColor)**
  - x, y 위치에 crColor색의 점을 찍는다
  - crColor : RGB(?, ?, ?) 값은 0~255의 정수

SetPixel(hdc, 10, 10, RGB(255, 0, 0))



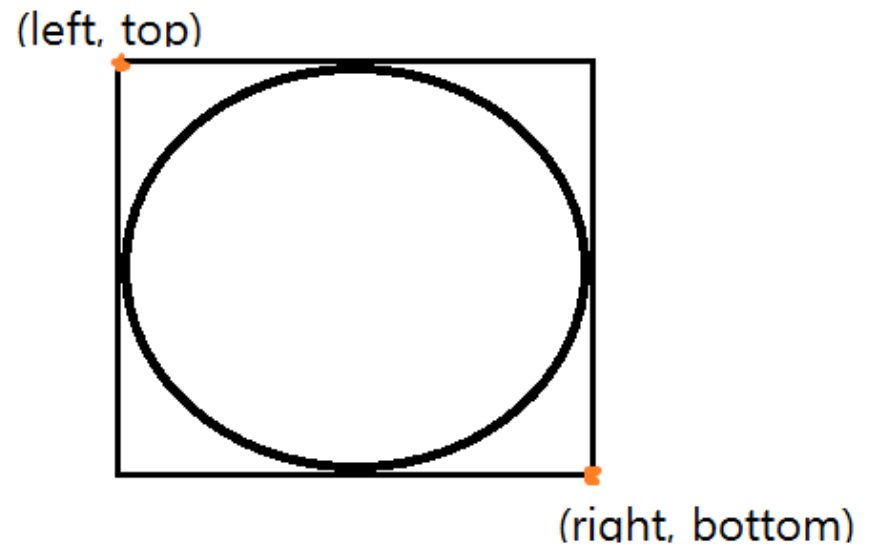
# 그래픽 출력

- ▶ **BOOL MoveToEx(HDC hdc, int X, int Y, LPPOINT lpPoint)**
- ▶ **BOOL LineTo( HDC hdc, int x, int y )**
- MoveToEx로 시작 위치를 정하고 LineTo로 끝점을 정하여 두 점을 잇는 선을 그린다
- lpPoint : NULL 포인터로 사용되지 않는다



# 그래픽 출력

- ▶ **BOOL Rectangle(HDC hdc, int left, int top, int right, int bottom)**
  - 인자로 받은 값에 맞는 사각형을 그린다
- ▶ **BOOL Ellipse(HDC hdc, int left, int top, int right, int bottom)**
  - 인자를 이용해 사각형을 만들고 그 안에 들어가는 원을 그려 준다



# 그래픽 출력

```
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    // SetPixel()로 붉은 점 찍기.
    for (int i = 0; 10 > i; i++)
    {
        SetPixel(hdc, 10 + 10 * i, 10, RGB(255, 0, 0));
    }
    // MoveToEx()와 LineTo()를 이용하여 선 긋기.
    MoveToEx(hdc, 10, 60, NULL);
    LineTo(hdc, 100, 20);
    // Rectangle()을 이용하여 사각형 그리기.
    Rectangle(hdc, 10, 100, 100, 150);
    // Ellipse()를 이용하여 타원 그리기.
    Ellipse(hdc, 10, 160, 100, 210);
    EndPaint(hWnd, &ps);
}
```

# Input

## WM\_CHAR

- 키보드 입력을 받았을 경우 발생
- wParam : **입력한 문자**를 알아온다

```
TCHAR str[256];
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CHAR:
        {
            int len = lstrlen(str);
            str[len] = (TCHAR)wParam;
            str[len + 1] = 0;
            HDC hdc = GetDC(hWnd);
            TextOut(hdc, 100, 100, str, lstrlen(str));
            ReleaseDC(hWnd, hdc);
        }
        break;
    }
    return 0;
}
```

# Input

## WM\_KEYDOWN

- WM\_CHAR처럼 키보드를 입력 받았을 경우 발생
- wParam : 문자가 아닌 **가상키 코드(virtual key code)**를 알려 준다

가상키 코드	값	키
VK_BACK	08	Backspace
VK_TAB	09	Tab
VK_RETURN	0D	Enter
VK_SHIFT	10	Shift
VK_CONTROL	11	Ctrl
VK_MENU	12	Alt
VK_PAUSE	13	Pause
VK_CAPITAL	14	Caps Lock
VK_ESCAPE	1B	Esc
VK_SPACE	20	스페이스
VK_PRIOR	21	PgUp
VK_NEXT	22	PgDn
VK_END	23	End
VK_HOME	24	Home
VK_LEFT	25	좌측 커서 이동키
VK_UP	26	위쪽 커서 이동키
이하생략		이하생략



# Input

## ▶ 마우스 입력 처리 윈도우 메시지(WM\_)

버튼	누름	놓음	더블클릭
좌측	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDBLCLK
우측	WM_RBUTTONDOWN	WM_RBUTTONUP	WM_RBUTTONDBLCLK
중앙	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDBLCLK

## ▶ 마우스 더블클릭

- 마우스 더블클릭 메시지를 받기 위해서는 윈도우 클래스 구성할 때 더블클릭 사용을 알려야 한다

```
wcex.style          = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;  
wcex.lpfnWndProc    = WndProc;  
wcex.cbClsExtra     = 0;  
wcex.cbWndExtra     = 0;  
wcex.hInstance      = hInstance;
```

## ▶ 마우스의 좌표

- $x = \text{LOWORD}(lParam); \quad y = \text{HIWORD}(lParam);$

# 타이머

## ▶ UNIT SetTimer(HWND hWnd, UINT\_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc)

- nIDEvent : 만들어진 타이머의 ID, wParam으로 값을 확인 할 수 있다
- uElapse : 1/1000초 단위로 해당 WM\_TIMER를 호출한다
- lpTimerFunc : 타이머의 CALLBACK

VOID CALLBACK TimerProc(HWND, UINT, UINT\_PTR, DWORD);

## ▶ BOOL KillTimer(HWND hWnd, UINT uIDEvent)

- 타이머는 시스템의 적역 자원이라 한 번 설정해 두면 애플리케이션이 종료되어도 사라지지 않고 계속 남아 있게 된다, 따라서 프로그램 종료 시 타이머를 파괴하여 주지 않으면 계속 해서 CPU를 차지하여 시스템의 성능이 떨어진다
- uIDEvent : SetTimer()에서 사용된 nIDEvent 값

# 타이머

```
char g_buf[256]; // 전역 변수.
SYSTEMTIME time; // 지역 변수.
case WM_CREATE: SetTimer(hWnd, 1, 1000, NULL); break;
case WM_TIMER:
    switch (wParam)
    {
    case 1:
        GetLocalTime(&time);
        sprintf_s(g_buf, "%d년 %d월 %d일 %d시 %d분 %d초", time.wYear, time.wMonth, time.wDay, time.wHour,
            time.wMinute, time.wSecond);
        InvalidateRect(hWnd, NULL, true);
        break;
    }
    break;
case WM_PAINT:
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    TextOutA(hdc, 10, 10, g_buf, strlen(g_buf));
    EndPaint(hWnd, &ps);
    break;
case WM_DESTROY:
    KillTimer(hWnd, 1);
    PostQuitMessage(0);
    break;
```

# 메시지 박스

- ▶ 사용자에게 별도의 조그만 윈도우를 열게 하여 정보전달 등을 하는 장치

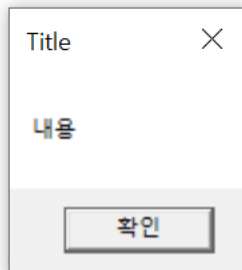
- ▶ **int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType)**

- lpText : 내용
- lpCaption : 메시지 박스의 타이틀

- uType :	<table><tr><th>값</th><th>설명</th></tr><tr><td>MB_ABORTRETRYIGNORE</td><td>Abort, Retry, Ignore 세 개의 버튼이 나타난다.</td></tr><tr><td>MB_OK</td><td>OK버튼 하나만 나타난다.</td></tr><tr><td>MB_OKCANCEL</td><td>OK, Cancel 두 개의 버튼이 나타난다.</td></tr><tr><td>MB_RETRYCANCEL</td><td>Retry, Cancel 두 개의 버튼이 나타난다.</td></tr><tr><td>MB_YESNO</td><td>Yes, No 두 개의 버튼이 나타난다.</td></tr><tr><td>MB_YESNOCANCEL</td><td>Yes, No, Cancel 세 개의 버튼이 나타난다.</td></tr></table>	값	설명	MB_ABORTRETRYIGNORE	Abort, Retry, Ignore 세 개의 버튼이 나타난다.	MB_OK	OK버튼 하나만 나타난다.	MB_OKCANCEL	OK, Cancel 두 개의 버튼이 나타난다.	MB_RETRYCANCEL	Retry, Cancel 두 개의 버튼이 나타난다.	MB_YESNO	Yes, No 두 개의 버튼이 나타난다.	MB_YESNOCANCEL	Yes, No, Cancel 세 개의 버튼이 나타난다.
값	설명														
MB_ABORTRETRYIGNORE	Abort, Retry, Ignore 세 개의 버튼이 나타난다.														
MB_OK	OK버튼 하나만 나타난다.														
MB_OKCANCEL	OK, Cancel 두 개의 버튼이 나타난다.														
MB_RETRYCANCEL	Retry, Cancel 두 개의 버튼이 나타난다.														
MB_YESNO	Yes, No 두 개의 버튼이 나타난다.														
MB_YESNOCANCEL	Yes, No, Cancel 세 개의 버튼이 나타난다.														

# 메시지 박스

```
switch (message)
{
    case WM_LBUTTONDOWN:
        if (MessageBox(hWnd, L"내용", L"Title", MB_OK) == IDOK)
        {
            // 수행 할 내용.
        }
        break;
    ...
}
```



값	설명
IDABORT	Abort 버튼을 눌렀다.
IDCANCEL	Cancel 버튼을 눌렀다.
IDIGNORE	Ignore 버튼을 눌렀다.
IDNO	No 버튼을 눌렀다.
IDOK	OK 버튼을 눌렀다.
IDRETRY	Retry 버튼을 눌렀다.
IDYES	Yes 버튼을 눌렀다.

# 이미지

## ▶ HDC CreateCompatibleDC(HDC hdc)

- 어떤 DC와 호환(compatible)되는 DC를 만든다
- DC의 특성을 대부분 가지고 있지만 출력 장치와는 연결이 안된 DC를 생성
- 화면에 그림을 출력할 것이 아니라 DC와 연결된 비트맵에만 그림을 그릴 경우
- 만들어져 **사용이 끝난 DC**는 **DeleteDC()** 함수를 이용하여 **제거** 해야 한다

## ▶ HGDIOBJ SelectObject(HDC hdc, HGDIOBJ h)

- h : GDI(그래픽 출력에 사용되는 도구) 오브젝트의 핸들
- DC에 저장되어 있는 **GDI 오브젝트의 핸들** 값을 **변경**할 경우 사용
- GDI 오브젝트는 **메모리를 사용하기** 때문에 사용이 끝나면 **DeleteObject()** 함수를 이용하여 **삭제**해 주어야 한다

# 이미지

- ▶ **HANDLE LoadImage(HINSTANCE hInst, LPCWSTR name, UINT type, int cx, int cy, UINT fuLoad)**
  - hInst : 해당 어플리케이션의 인스턴스 핸들, **독립 리소스**를 불러올 경우 **NULL**
  - name : 불러 올 **이미지의 경로**, **MAKEINTRESOURCE** 매크로를 사용하여 리소스를 불러올 수 있다
  - type : 불러올 **이미지의 종류**, **IMAGE\_BITMAP**(비트맵을 불러온다)
  - cx, cy : **0**, 원본 이미지의 크기로 불러온다
  - fuLoad : 옵션 정의 플래그로 **하나 이상의 값**을 지정, **LR\_LOADFROMFILE**(리소스가 아닌 경로를 통하여 이미지를 로드한다)
- ▶ **LoadBitmap()** 함수로 비트맵을 로드 할 수 있으나 **이 후 버전에서 변경이 되거나 지원하지 않을 수 있기에** 사용 하지 않는 것이 좋다

# 이미지

플래그	설명
LR_DEFAULTCOLOR	기본 플래그, 이미지를 흑백으로 불러오지 않게 한다
LR_MONOCHROME	이미지를 <b>흑백</b> 으로 불러온다
LR_LOADFROMFILE	name의 인수를 리소스 대신 <b>파일 경로를 사용하여</b> 불러온다
LR_LOADTRANSPARENT	이미지의 첫 번째 픽셀의 색상을 읽어 색상 테이블에 있는 해당 색상을 <b>윈도우 기본 색상(COLOR_WINDOW)</b> 으로 변경 LR_LOADMAP3DCOLORS와 함께 사용하 경우 이 값이 우선순위가 된다 <b>*이 플래그는 8bpp 이상의 비트맵에서 사용할 수 없다</b>
LR_DEFAULTSIZE	cx 또는 cy 인수가 0인 경우 <b>시스템 지정 값을 사용</b> 이 플래그가 지정되지 않고 cx나 cy 인수가 0인 경우 원본 크기를 사용
LR_VGACOLOR	<b>True VGA</b> 색상을 사용
LR_CREATEDIBSECTION	type 인수에서 <b>IMAGE_BITMAP</b> 을 사용한 경우 호환 비트맵이 아닌 <b>DIB(Device Independent Bitmap)</b> <b>섹션 비트맵</b> 으로 불러온다
LR_SHARED	<b>같은 이미지를 여러 번 불러올 경우, 이미지 핸들을 서로 공유한다</b> 이 플래그를 사용하면 더 이상 필요하지 않을 때 <b>시스템이 리소스를 제거해 준다</b> 이 플래그를 사용하지 않고 같은 이미지를 여러 번 불러오면 각각 서로 다른 핸들 값을 리턴 한다 이 플래그를 사용하지 않을 경우, <b>시스템 아이콘이나 커서</b> 를 불러올 수 없다



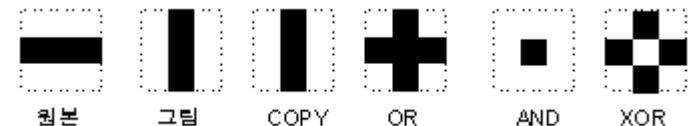
# 이미지

- ▶ BOOL BitBlt(HDC hdc, int x, int y, int cx, int cy, HDC hdcSrc, int x1, int y1, DWORD rop)
- x : 화면 상의 x좌표
- y : 화면 상의 y좌표
- cx : 그려지는 이미지의 너비
- cy : 그려지는 이미지의 높이
- x1 : 원본 이미지의 x좌표
- y1 : 원본 이미지의 y좌표
- rop : 레스터 연산 방법, 어떤 방식으로 그릴 것인가

값	설명
BLACKNESS	대상영역을 검정색으로 가득 채운다.
DSTINVERT	화면을 반전시킨다.
MERGECOPY	소스 비트맵과 대상 화면을 AND 연산한다.
MERGEPAINT	소스 비트맵과 대상 화면을 OR 연산한다.
SRCCOPY	소스 영역을 대상 영역에 복사한다.
WHITENESS	대상영역을 흰색으로 채운다.

- ▶ 원본 이미지의 크기를 변경하지 않는다

## 그리기 모드



# 이미지

- ▶ `BOOL StretchBlt(HDC hdcDest, int xDest, int yDest, int wDest, int hDest, HDC hdcSrc, int xSrc, int ySrc, int wSrc, int hSrc, DWORD rop)`
  - `xDest` : 화면 상의 x좌표
  - `yDest` : 화면 상의 y좌표
  - `wDest` : 화면 상에 그려지는 크기(너비)
  - `hDest` : 화면 상에 그려지는 크기(높이)
  - `xSrc` : 원본 이미지의 x좌표
  - `ySrc` : 원본 이미지의 y좌표
  - `wSrc` : 원본 이미지에서 `xSrc`를 중심으로 그리려고 하는 크기(너비)
  - `hSrc` : 원본 이미지에서 `ySrc`를 중심으로 그리려고 하는 크기(높이)
  - `rop` : 레스터 연산 방법, 어떤 방식으로 그릴 것인가
- ▶ 원본 이미지의 크기를 늘리거나 줄여서 그릴 수 있다

# 이미지

- ▶ 예제 소스의 00.bmp 파일을 프로젝트 폴더에 추가

```
// 원본 이미지의 크기 : 145, 245
```

```
case WM_PAINT:
```

```
{
```

```
    PAINTSTRUCT ps;
```

```
    HDC hdc = BeginPaint(hWnd, &ps);
```

```
    HDC memDC = CreateCompatibleDC(hdc);
```

```
    HBITMAP myBitmap = (HBITMAP)LoadImage(NULL, "00.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
```

```
    auto OldBitmap = (HBITMAP)SelectObject(memDC, myBitmap);
```

```
    BitBlt(hdc, 0, 0, 145, 245, memDC, 0, 0, SRCCOPY);
```

```
    StretchBlt(hdc, 200, 200, 245, 345, memDC, 0, 0, 145, 245, SRCCOPY);
```

```
    SelectObject(memDC, OldBitmap);
```

```
    DeleteObject(myBitmap);
```

```
    DeleteDC(memDC);
```

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

```
}
```

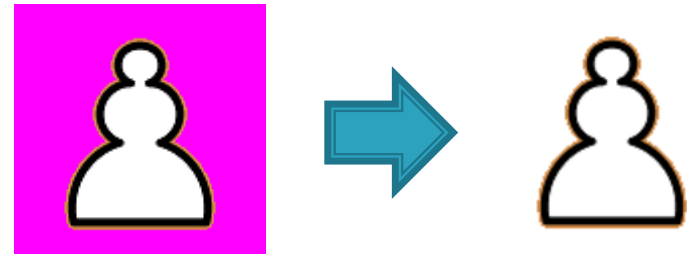
# 투명처리

- ▶ **BOOL TransparentBlt(HDC hdcDest, int xoriginDest, int yoriginDest, int wDest, int hDest, HDC hdcSrc, int xoriginSrc, int yoriginSrc, int wSrc, int hSrc, UINT crTransparent)**
  - xoriginDest : 화면상의 x좌표
  - yoriginDest : 화면상의 y좌표
  - wDest : 화면에 그려지는 크기(너비)
  - hDest : 화면에 그려지는 크기(높이)
  - xoriginSrc : 원본 이미지상의 x좌표
  - yoriginSrc : 원본 이미지상의 y좌표
  - wSrc : 원본 이미지의 설정한 x 좌표로부터 그려질 크기(너비)
  - hSrc : 원본 이미지의 설정한 y 좌표로부터 그려질 크기(높이)
  - crTransparent : 투명하게 처리 할 RGB 색상, 일반적으로 **RGB(255, 0, 255)** 분홍색을 사용
- ▶ 원본 이미지의 크기를 늘리거나 줄여서 그릴 수 있다

# 투명처리

- ▶ 예제 소스의 `block_w_00.bmp` 파일을 프로젝트 폴더에 추가

```
#pragma comment(lib, "msimg32.lib") // TransparentBlt() 함수를 사용하기 위해서는,  
// 해당 라이브러리를 추가하여야 한다.  
  
// 원본 이미지의 크기 : 125, 125  
case WM_PAINT:  
{  
    PAINTSTRUCT ps;  
    HDC hdc = BeginPaint(hWnd, &ps);  
  
    HDC memDC = CreateCompatibleDC(hdc);  
    HBITMAP myBitmap = (HBITMAP)LoadImage(NULL, "block_w_00.bmp", IMAGE_BITMAP, 0, 0,  
    LR_LOADFROMFILE);  
    auto OldBitmap = (HBITMAP)SelectObject(memDC, myBitmap);  
    TransparentBlt(hdc, 100, 100, 50, 50, memDC, 0, 0, 125, 125, RGB(255, 0, 255));  
    SelectObject(memDC, OldBitmap);  
    DeleteObject(myBitmap);  
    DeleteDC(memDC);  
  
    EndPaint(hWnd, &ps);  
}
```



# 충돌체크

```
▶ struct RECT
{
    LONG    left;
    LONG    top;
    LONG    right;
    LONG    bottom;
}
```

RECT 구조체 좌표

(left, top)



(right, bottom)

- ▶ **BOOL PtInRect(RECT \*lprc, POINT pt)**
  - lprc : 체크할 사각형의 영역
  - pt : x, y 좌표, 대상의 위치
- ▶ **BOOL IntersectRect(LPRECT lprcDst, RECT \*lprcSrc1, RECT \*lprcSrc2)**
  - lprcDst : 두 사각형의 영역이 겹쳤을 때 생기는 사각형 영역을 알려준다
  - lprcSrc1, lprcSrc2 : 충돌확인을 할 두 사각형

# 카드 맞추기 게임

[BitMapManager 예제]를 이용하여  
카드 맞추기 게임을 만들어 봅시다

# Input2

## ▶ SHORT GetKeyState(int nVirtKey)

- nVirtKey : 가상키 값
- Caps Lock이나 Num Lock과 같이 **현재 키의 토글 상태**를 알아온다
- 해당 프로그램의 **메시지 입력 큐**에서 얻어온다

## ▶ SHORT GetAsyncKeyState(int vKey)

- vKey : 가상키 값
- **메시지 발생 후의 상태**를 리턴한다
- 반환 값이 비트 형태로 리턴되어 비트 연산을 할 수 있다
- **비동기 처리**가 되어 **다중 키 사용이 가능**해 진다

`if(GetAsyncKeyState(VK_SPACE) & 0x8000)`

반환값	설명
0 (0x0000)	이전에 누른 적이 없고 호출 시점에서 안눌린 상태
0x8000	이전에 누른 적이 없고 호출 시점에서 눌린 상태
0x8001	이전에 누른 적이 있고 호출 시점에서 눌린 상태
1 (0x0001)	이전에 누른 적이 있고 호출 시점에서 안눌린 상태



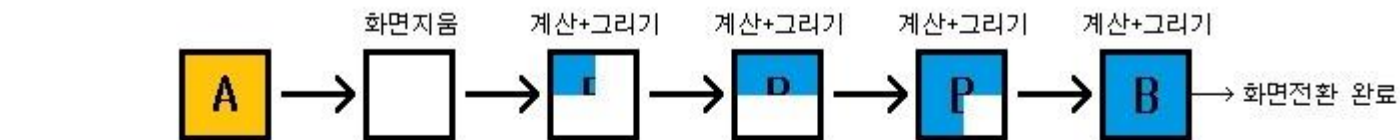
# Input2

```
int g_nGetKeyX = 0, g_nGetAsyncKeyX = 0; // 전역 변수.
case WM_CREATE:
    SetTimer(hWnd, 1, 30, NULL);
    break;
case WM_TIMER:
{
    auto hdc = GetDC(hWnd);
    if (GetKeyState(VK_SPACE))
    {
        g_nGetKeyX += 10;
        TextOut(hdc, g_nGetKeyX, 10, TEXT("GetKeyState"), 11);
    }
    if (GetAsyncKeyState(VK_SPACE))
    {
        g_nGetAsyncKeyX += 10;
        TextOut(hdc, g_nGetAsyncKeyX, 30, TEXT("GetAsyncKeyState"), 16);
    }
    ReleaseDC(hWnd, hdc);
}
break;
case WM_DESTROY:
    KillTimer(hWnd, 1);
    PostQuitMessage(0);
    break;
```

# 더블 버퍼링(Double Buffering)

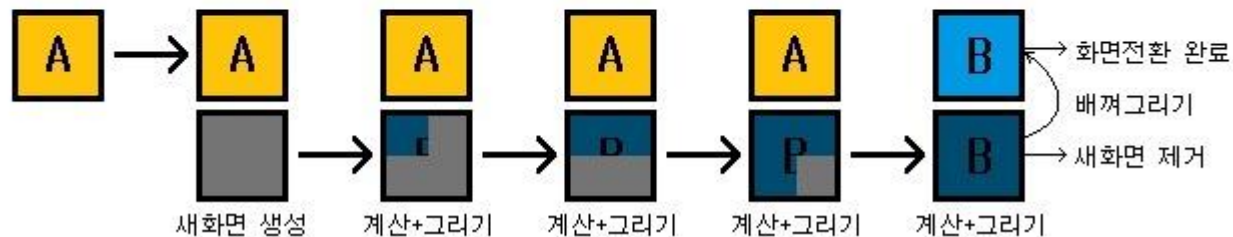
## ▶ 더블 버퍼링이란?

- 이중 버퍼링이라고도 불리며, **전위 버퍼(Primary/Front buffer)**와 **후위 버퍼(Back buffer)**로 구분한다
- 일반적인 방법으로 그리기를 하면 발생하는 화면 **깜빡임(Flickering)** 현상을 **보안**하기 위한 기법
- 그리기를 할 때, 화면에 바로 이미지를 그리는 것이 아닌 **미리 그려둔 전체 화면을 한 번에 덮어씌운다**
- InvalidateRect() 함수의 세 번째 인자는 **false**를 쓰며, 지우지 않고 갱신되게 한다



기존 화면전환 InvalidateRect(hWnd, NULL, TRUE);

더블버퍼링



# 더블 버퍼링(Double Buffering)

```
HBITMAP CreatedIBSectionRe(HDC hdc, int width, int height)
{
    BITMAPINFO bm_info;
    ZeroMemory(&bm_info.bmiHeader, sizeof(BITMAPINFOHEADER));
    bm_info.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    bm_info.bmiHeader.biBitCount = 24; // 컬러 수(color bits) : 1, 4, 8, 16, 24, 31
    bm_info.bmiHeader.biWidth = width; // 너비.
    bm_info.bmiHeader.biHeight = height; // 높이.
    bm_info.bmiHeader.biPlanes = 1;

    LPVOID pBits;
    return CreateDIBSection(hdc, &bm_info, DIB_RGB_COLORS, (void**)&pBits, NULL, 0);
}
```

# 더블 버퍼링(Double Buffering)

```
HBITMAP Chess, Dog; // 전역 변수.
int g_nX = 0; // 전역 변수.
void DoubleBuffer(HWND hWnd, HDC hdc)
{
    RECT windowRect;
    GetWindowRect(hWnd, &windowRect);

    HDC backDC = CreateCompatibleDC(hdc);
    HBITMAP backBitmap = CreateDIBSection(hdc, windowRect.right - windowRect.left, windowRect.bottom - windowRect.top);
    HBITMAP oldBack = (HBITMAP)SelectObject(backDC, backBitmap);

    HDC memDC = CreateCompatibleDC(hdc);
    HBITMAP oldBitmap = (HBITMAP)SelectObject(memDC, Chess);
    TransparentBlt(backDC, 100 + g_nX, 100, 125, 125, memDC, 0, 0, 125, 125, RGB(255, 0, 255));
    SelectObject(memDC, oldBitmap);

    oldBitmap = (HBITMAP)SelectObject(memDC, Dog);
    BitBlt(backDC, 0, 0, 145, 245, memDC, 0, 0, SRCCOPY);
    StretchBlt(backDC, 200, 200, 245, 345, memDC, 0, 0, 145, 245, SRCCOPY);
    SelectObject(memDC, oldBitmap);

    DeleteDC(memDC);
    BitBlt(hdc, 0, 0, windowRect.right - windowRect.left, windowRect.bottom - windowRect.top, backDC, 0, 0, SRCCOPY);
    SelectObject(backDC, oldBack);
    DeleteObject(backBitmap);
    DeleteObject(backDC);
}
```

# 더블 버퍼링(Double Buffering)

```
case WM_CREATE:
    Chess = (HBITMAP)LoadImage(NULL, L"block_w_00.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);
    Dog = (HBITMAP)LoadImage(NULL, L"00.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);
    SetTimer(hWnd, 1, 30, NULL);
    break;
case WM_TIMER:
    if (GetAsyncKeyState(VK_RIGHT)){
        g_nX += 10;
        InvalidateRect(hWnd, NULL, false);}
    if (GetAsyncKeyState(VK_LEFT)){
        g_nX -= 10;
        InvalidateRect(hWnd, NULL, false);}
    break;
case WM_PAINT:
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    DoubleBuffer(hWnd, hdc);
    EndPaint(hWnd, &ps);
    break;
case WM_DESTROY:
    DeleteObject(Chess);
    DeleteObject(Dog);
    KillTimer(hWnd, 1);
    PostQuitMessage(0);
    break;
```

# 더블 버퍼링(Double Buffering)

- ▶ **BOOL GetWindowRect(HWND hWnd, LPRECT lpRect)**
  - 클라이언트의 **화면 만의 크기**를 구해준다
- ▶ **HBITMAP WINAPI CreateDIBSection(HDC hdc, CONST BITMAPINFO \*pbmi, UINT usage, VOID \*\*ppvBits, HANDLE hSection, DWORD offset)**
  - **DIB를 생성해 준다**
  - DIB(Device Independent Bitmap) : 장치 독립적 비트맵, GDI 함수를 사용할 수 없다
  - DIBSection은 DIB를 DDB(Device Dependent Bitmap : 장치 종속적 비트맵)처럼 행동할 수 있게 하여 GDI 함수를 사용할 수 있게 된다
  - pbmi : **만들고자 하는 비트맵의 정보**를 지정하는 구조체
  - usage : **DIB\_RGB\_COLORS**(RGB 값의 배열 사용), **DIB\_PAL\_COLORS**(16비트 인덱스 배열 사용)
  - ppvBits : DIB bit 값의 위치 포인터를 받기 위한 이중 포인터
  - hSection : NULL
  - offset : 0

# 서커스 찰리 게임

더블 버퍼링을 사용하여  
서커스 찰리 게임을 만들어 봅시다