제 5장 비트맵과 애니메이션

2022년 1학기 윈도우 프로그래밍

학습 목표

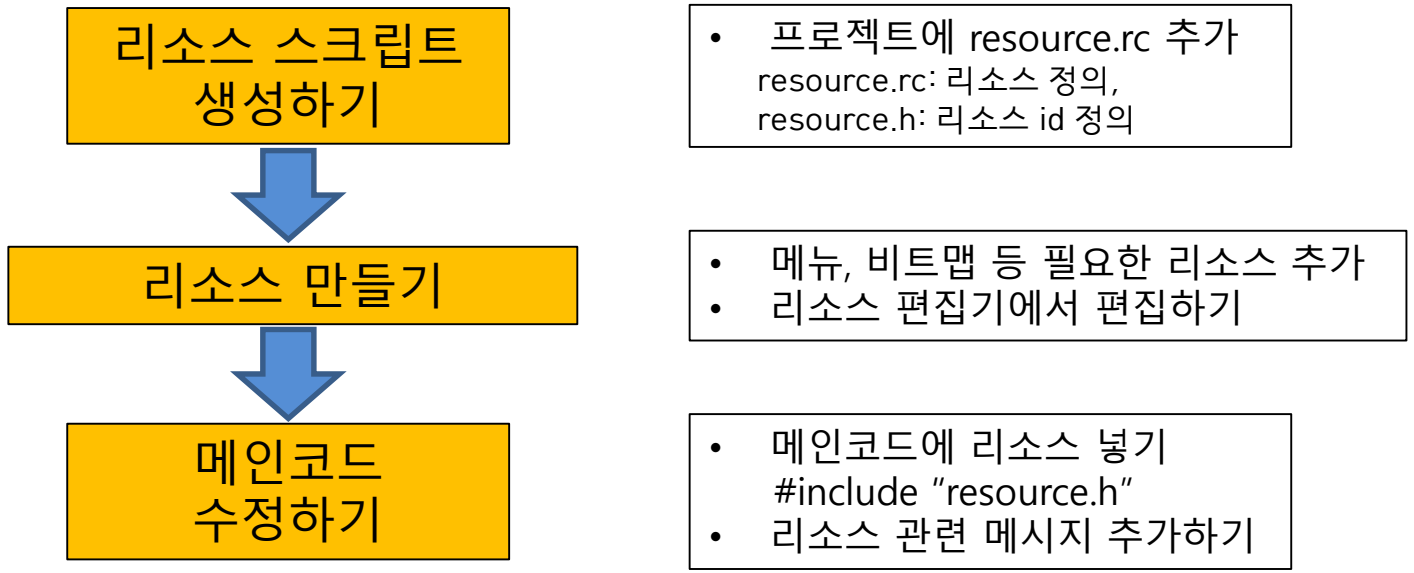
- **학습 목표**
  - 비트맵 형식의 그림 파일을 불러 화면에 출력할 수 있다.
  - 더블 버퍼링 기법을 이용해 비트맵 그림 파일로 애니메이션을 만들 수 있다

- **내용**
  - 비트맵
  - 애니메이션 만들기
  - 더블 버퍼링 사용하기
  - 마스크 이용하기

1. 리소스 사용하기

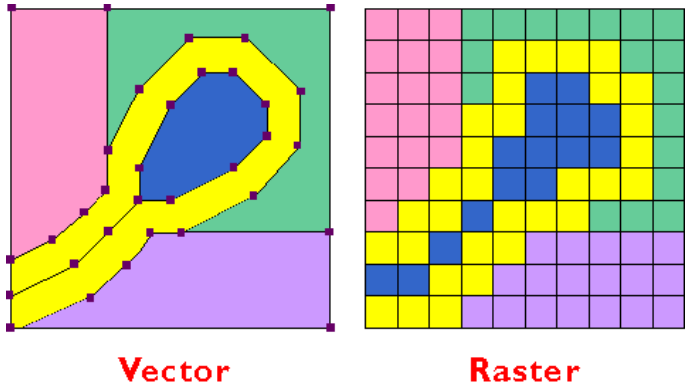
- **리소스 (Resource)**
  - 메뉴, 아이콘, 커서, 다이얼로그, 액셀러레이터, 비트맵 등 사용자 인터페이스를 구성하는 자원들로 읽기 전용 정적 데이터를 말한다.
  - 리소스는 프로그램 실행 중 변경되지 않는다.
  - C/C++과 같은 언어로 관리하지 않고 리소스 스크립트(Resource Script; .rc)파일로 관리한다.
  - 윈도우 프로그래밍에서는 리소스로 소스코드와 별도로 만들고 컴파일하며, **링크 시 최종 실행파일에 합쳐진다.**

- **리소스 사용하기**



2. 비트맵

- **컴퓨터 이미지(image)**
  - 전자적인 형태로 만들어지거나 복사되고, 저장된 그림이다.
  - 2차원 그래픽스 종류: 래스터 그래픽스(raster graphics), 벡터 그래픽스(vector graphics)
    - **래스터 그래픽스 (raster graphics)**
      - 비트맵 이미지로 그래픽 객체들을 구성하고 있는 픽셀 (PIXEL, Picture Element) 값들을 그대로 저장하는 방식
      - 그림의 크기가 변형되면 화질이 떨어진다.
      - 그림의 복잡도에 관계없이 그림파일의 크기는 일정하다.
      - 포토샵, 페인터 같은 소프트웨어에서 사용
    - **벡터 그래픽스 (vector graphics):**
      - 이미지를 표현하기 위하여 수학 방정식을 기반으로 점, 직선, 곡선, 다각형과 같은 물체를 사용하는 방법
      - 벡터 그래픽 파일에는 선을 그리기 위해 각 비트들이 저장되지 않고 연결될 일련의 점의 위치가 저장된다.
      - 파일 크기가 작아지며 변형이 용이한 특징을 갖는다.
      - 벡터 형식으로 저장된 이미지를 메타파일이라고도 한다.
      - 일러스트레이터, 각종 3D 프로그램



비트맵

- 이미지 종류

이미지 포맷	특징
BMP (*.BMP, *.DIB)	Microsoft Windows Device Independent Bitmap 윈도우의 표준 그래픽 파일. 거의 모든 프로그램에서 지원하기 때문에 사용이 간편 용량이 크고, 레이어와 알파채널을 지원하지 않는다.
GIF (*.GIF)	Graphics Interchange Format의 약자로 미국의 통신업체인 CompuServe에서 개발 최대 256색까지 저장할 수 있는 비손실 압축 형식으로 웹에서 가장 널리 쓰이는 파일 포맷 다중 프레임 애니메이션을 구현 투명 이미지 지원
JPEG (*.JPG)	Joint Picture Experts Group의 약자로 이미지 압축 기술자 모임인 JPEG 위원회에서 개발 압축률이 높아 적은 용량으로 고화질의 사진을 저장할 수 있지만 이미지 품질은 하락 레이어와 알파채널을 지원하지 않는다.
TIFF (*.TIF, *.TIFF)	Tag Image File Format의 약자 응용프로그램 간 그래픽 데이터 교환을 목적으로 개발된 형식으로 사용자가 고쳐서 쓸 수 있다. 알파채널 지원, 다양한 압축방법 제공 파일 용량이 크다
PNG (*.PNG)	Portable Network Graphics의 약자로 GIF와 JPEG의 장점을 합친 파일 포맷 투명 효과, 압축률이 우수 비손실 압축을 사용하므로 재편집 할 때 유용
	그 외, PSD 파일, PDF 파일, RAW 파일, AI 파일 등

비트맵

- **윈도우 OS에서 지원하는 비트맵은 두가지이다.**
  - 윈도우 3.0 이전에 사용하던 DDB(Device Dependent Bitmap)
  - 현재 많이 사용하는 DIB(Device Independent Bitmap)

- **DDB는 DIB에 비해 간단하며 DC에 바로 선택될 수 있는 비트맵**
  - 프로그램 내부에서만 사용되는 비트맵의 경우에 많이 사용한다.
  - 장치에 의존적이기 때문에 원래 만들어진 장치 이외에서 출력할 경우 원래대로 출력되지 않을 수 있다.
  - 외부 비트맵파일(.bmp)을 프로그램에 불러와 그래픽 작업을 수행하거나 다양한 영상처리 효과를 주는 프로그램을 만드는 경우에는 장치에 독립적이고 훨씬 다양한 기능을 가지고 있는 DIB를 더 많이 사용한다

비트맵

- 비트맵 구조체 (DDB 비트맵)

```
typedef struct tagBITMAP {
        LONG bmType;            // 비트맵 타입: 0
        LONG bmWidth;           // 비트맵의 넓이 (픽셀 단위)
        LONG bmHeight;          // 비트맵의 높이 (픽셀 단위)
        LONG bmWidthBytes;      // 각 스캔 라인의 바이트 수
        WORD bmPlanes;          // 색상 판의 숫자
        WORD bmBitsPixel;       // 각 픽셀당 색상을 위한 비트수
        LPVOID bmBits;          // 비트맵을 가리키는 포인터
    } BITMAP, *PBITMAP;
```

비트맵

- 비트맵 구조체 (DIB 비트맵)

<픽셀 당 1, 2, 4, 8비트 비트맵>

<픽셀 당 16, 24, 32 비트 비트맵>

비트맵

- 비트맵 구조체 (DIB 비트맵)

BITMAPFILEHEADER
(비트맵 파일에 대한 정보)
	typedef struct tag BITMAPFILEHEADER {
	WORD bfType; // 비트맵 파일 확인 (BM 타입)
	DWORD bfSize; // 비트맵 파일 크기
	WORD bfReserved1; // 0
	WORD bfReserved2; // 0
	DWORD bfOffBits; // 실제 비트맵 데이터 값과 헤더의 오프셋 값
	} BITMAPFILEHEADER, *PBITMAPFILEHEADER;

BITMAPINFOHEADER
(비트맵 자체에 대한 정보)
	typedef struct tag BITMAPINFOHEADER{
	DWORD biSize; // BITMAPINFOHEADER 구조체 크기
	LONG biWidth; // 비트맵의 가로 픽셀 수
	LONG biHeight; // 비트맵의 세로 픽셀 수
	WORD biPlanes; // 장치에 있는 색상 면의 개수 (반드시 1)
	WORD biBitCount; // 한 픽셀을 표현할 수 있는 비트의 수
	DWORD biCompression; // 압축 상태 지정 (BI_RGB: 압축되지 않은 비트맵
	DWORD biSizeImage; // 실제 이미지의 바이트 크기 (비 압축 시 0)
	LONG biXPelsPerMeter; // 미터당 가로 픽셀 수
	LONG biYPelsPerMeter; // 미터당 세로 픽셀 수
	DWORD biClrUsed; // 색상테이블의 색상 중 실제 비트맵에서 사용되는
	// 색상수 (0 : 모든 색상을 사용
	DWORD biClrImportant; // 비트맵을 출력하는데 필수적인 색상인덱스수
	// (0 : 모든 색상이 사용되어야 함)
	} BITMAPINFOHEADER, *PBITMAPINFOHEADER;

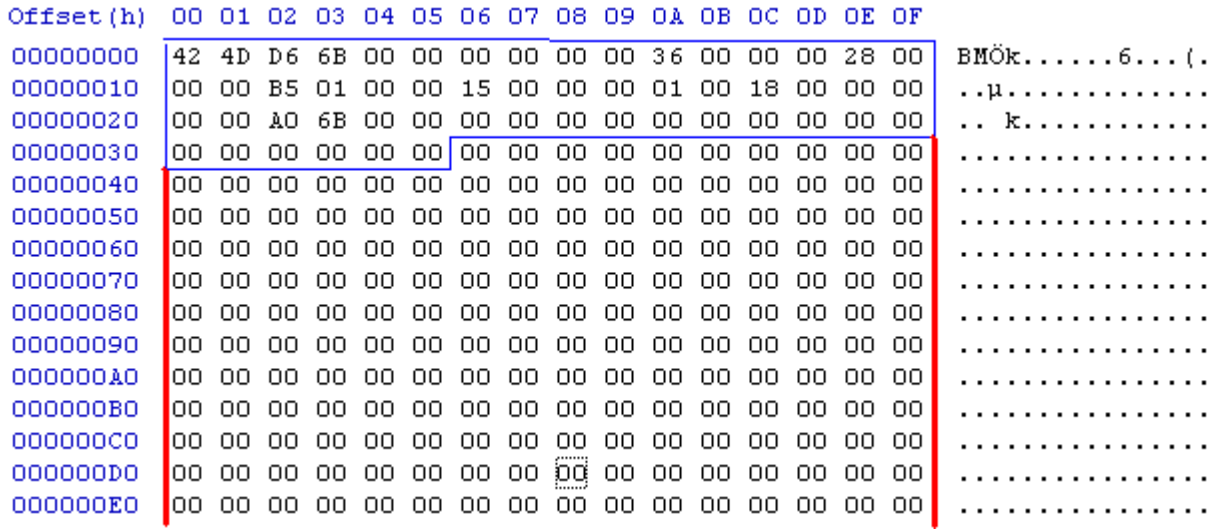
RGBQUAD 배열
(색상 테이블)
	typedef struct tag RGBQUAD {
	BYTE rgbBlue; // 파란색
	BYTE rgbGreen; // 초록색
	BYTE rgbRed; // 빨강색
	BYTE rgbReserved; // 예약된 값: 0
	} RGBQUAD;

color/index 배열 (픽셀 데이터)	24비트 비트맵 파일이 픽셀인 경우 BGRBGRBGR… (B: blue, G: green, R: red가 각각 1바이트씩, BGR이 한 픽셀)

비트맵

- 비트맵 읽기
  - **비트맵 이미지는 LoadBitmap() 함수 또는 LoadImage() 함수로 읽는다.**
    - LoadBitmap 함수:
      - 리소스로 불러온 이미지 파일만 읽기가 가능한다.
    - LoadImage 함수:
      - 리소스로 불러온 이미지 파일 또는 이미지 파일로도 읽기가 가능한다.

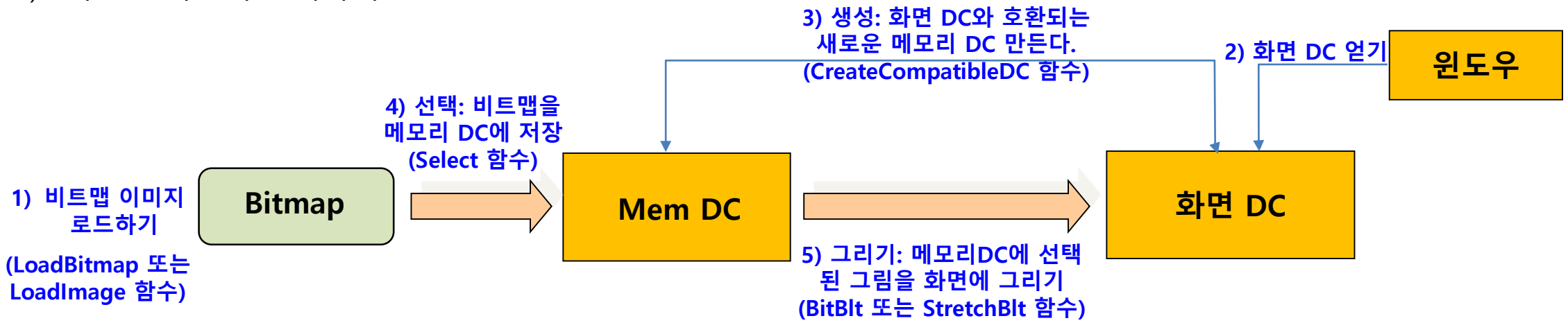
  - 비트맵 이미지를 사용하기 위해서
    - 이미지 편집기에서 직접 이미지 만들어 활용하기
    - 이미 만들어져 있는 이미지를 다운로드 받아서 활용하기



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	42	4D	D6	6B	00	00	00	00	00	00	36	00	00	00	28	00	BMÖk....6...(.
00000010	00	00	B5	01	00	00	15	00	00	00	01	00	18	00	00	00	..µ.....
00000020	00	00	A0	6B	00	00	00	00	00	00	00	00	00	00	00	00	.. k.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

비트맵 화면에 출력하기

- 비트맵 화면에 출력하기
  1) 비트맵 로드하기
  2) 화면 디바이스 컨텍스트 얻기
  3) 메모리 디바이스 컨텍스트 만들기
  4) 비트맵 사용하기 위해 선택하기
  5) 비트맵 화면에 출력하기



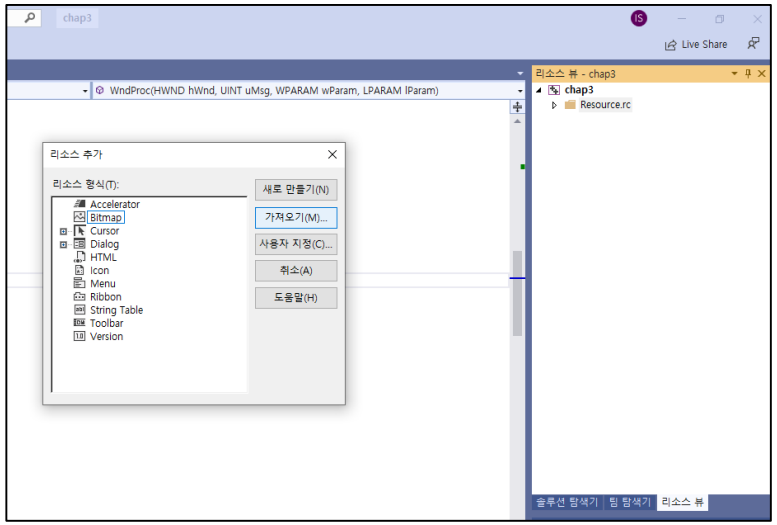
1) 비트맵 로드하기: 리소스로 사용하기

- **리소스 파일 작성**
  - 방법: 소스 파일 작성과 유사
  - "C++ Source" 대신에 Resource Script 선택
  - 리소스 파일 이름 명시

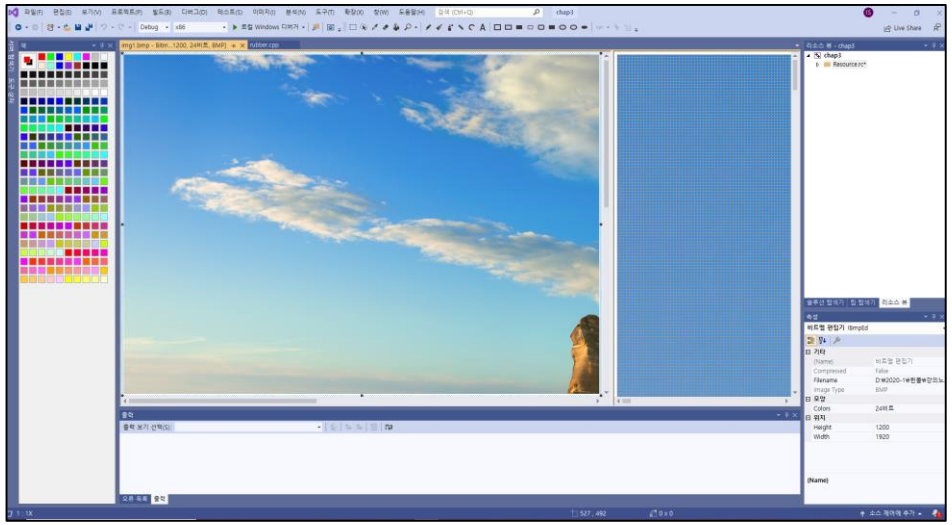
- **비트맵을 리소스로 사용하려면**
  - 새로운 비트맵 리소스 만들기
    - 새로 만들기:직접 만들기 – 리소스 편집기에서 이미지 만들기
    - 불러오기: 이미 만들어진 이미지 활용하기 – 기존의 비트맵 이미지를 읽어서 사용하기

비트맵 로드하기: 리소스로 사용하기

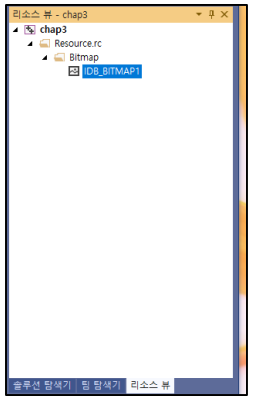
- **Visual Studio 2019 환경**
  - 리소스에서 비트맵 추가
  - 비트맵 편집창

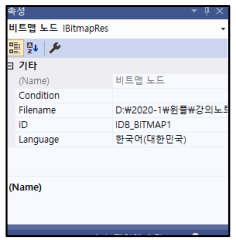


비트맵 불러오기



불러온 비트맵 편집창

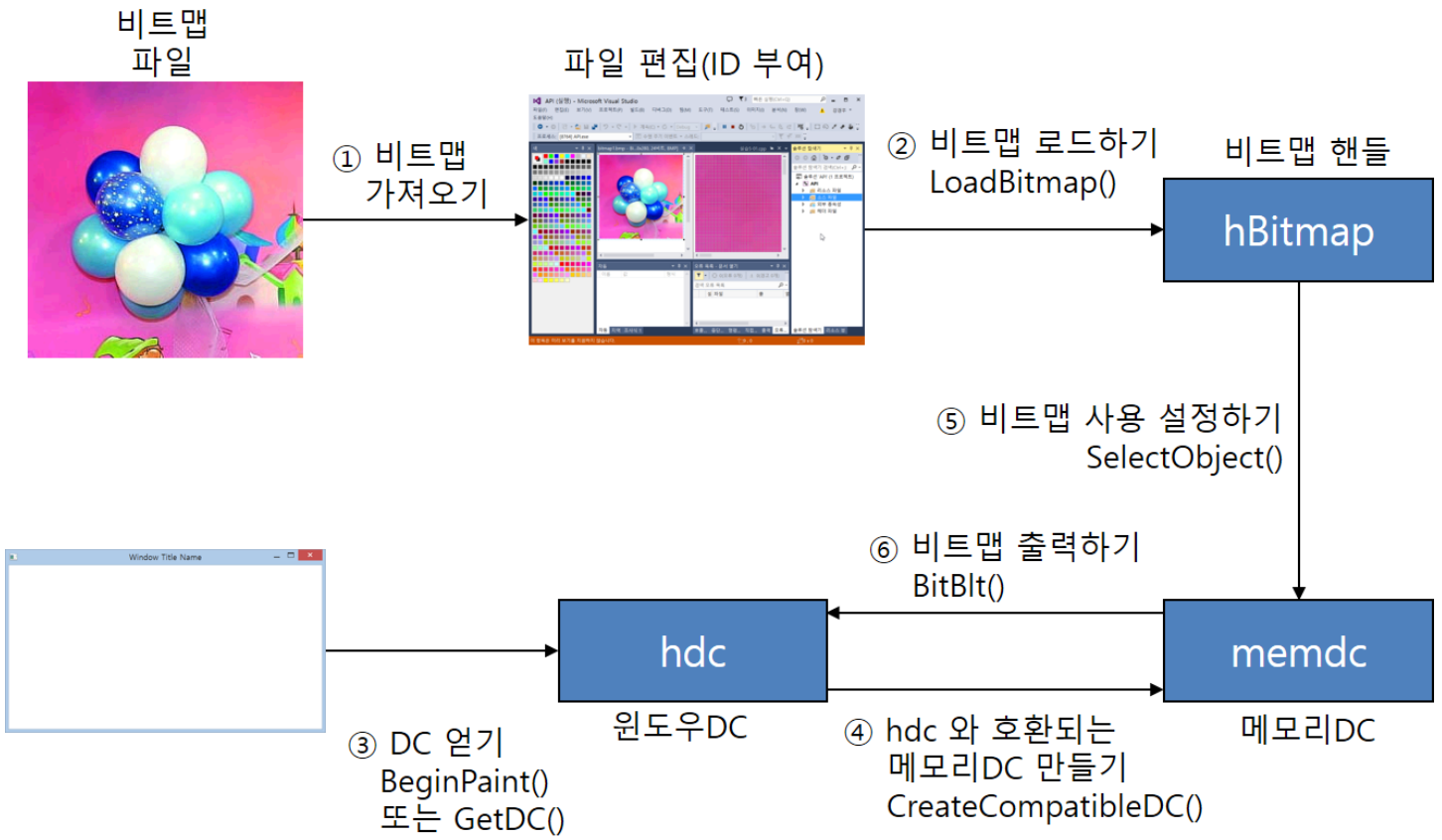




리소스뷰와 비트맵 속성창

비트맵 로드하기: 리소스로 사용하기

- 비트맵 이미지를 출력하기



비트맵 로드하기: LoadBitmap () 함수

- **비트맵 읽기**
  - 비트맵을 읽어올 때: LoadBitmap 함수를 사용
    - 리소스로 불러온 이미지 파일만 읽기가 가능한다. 리소스 파일에 추가된 이미지는 빌드하면 실행 파일에 포함된다.
    - 이 함수로 읽은 비트맵은 DDB로 변경된다.

HBITMAP LoadBitmap (HINSTANCE hInstance, LPCTSTR lpBitmapName);
- 리소스를 사용하여 비트맵 로드
  - HINSTANCE hInstance: 어플리케이션 인스턴스 핸들
  - LPCTSTR lpBitmapName: 비트맵 리소스 이름

- **비트맵 읽기 예)**
  - LoadBitmap 함수 사용: 리소스로 사용

```
HBITMAP hBitmap;
hBitmap = LoadBitmap (hInstance, MAKEINTRESOURCE(IDB_BITMAP1));        //--- IDB_BITMAP1: 리소스로 저장된 이미지
```

2) 화면 디바이스 컨텍스트 얻기

- **디바이스 컨텍스트 얻기**
  - 기존의 방식으로 DC 얻기

```
HDC hdc = GetDC(hwnd);
또는
HDC hdc = BeginPaint(hwnd, &ps);
```

3) 메모리 디바이스 컨텍스트 만들기

- **메모리 디바이스 컨텍스트 (메모리 DC)**
  - 화면 DC와 동일한 특성을 가지며 그 내부에 출력 표면을 가진 메모리 영역
    - 화면 DC에서 사용할 수 있는 모든 출력을 메모리 DC에서 할 수 있다.
    - 메모리 DC에 먼저 그림을 그린 후 사용자 눈에 그려지는 과정은 보여주지 않고 메모리 DC에서 작업을 완료한 후 그 결과만 화면으로 고속 복사한다.
    - **비트맵도 일종의 GDI 오브젝트이지만 화면 DC에서는 선택할 수 없으며 메모리 DC만이 비트맵을 선택할 수 있어서 메모리 DC에서 먼저 비트맵을 읽어온 후 화면 DC로 복사한다.**

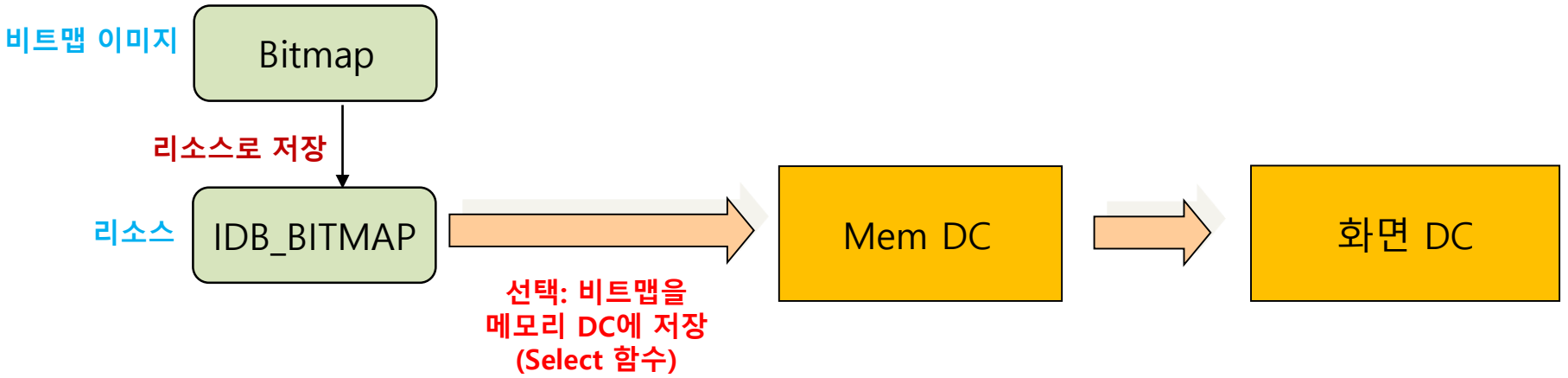
    - 메모리 DC를 만들 때: CreateCompatibleDC 함수 사용

HDC CreateCompatibleDC (HDC hdc);
- 주어진 DC와 호환되는 메모리 DC를 생성해서 리턴한다.
- HDC hdc: 주어진 DC

4) 비트맵 선택하기

- **비트맵 선택**
  - 메모리 DC를 만든 후에는 읽어온 비트맵을 메모리 DC에 선택해 준다.
  - 선택하는 방법: SelectObject 함수를 사용

HGDIOBJ SelectObject (HDC hDC, HGDIOBJ hgdiobj);
- hDC: DC 핸들값
- hgdiobj: GDI의 객체
- 리턴 값은 원래의 오브젝트 값



비트맵: 리소스로 사용하기

- **비트맵 이미지 읽어 화면에 출력하기**

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc, hMemDC;
    HBITMAP hBitmap;

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

            hBitmap = LoadBitmap (hInstance, MAKEINTRESOURCE(IDB_BITMAP1));      //--- 리소스로 읽은 비트맵을 로드하기

            hMemDC = CreateCompatibleDC (hdc);                                   //--- 1. 주어진 DC와 호환되는 DC를 생성
            SelectObject (hMemDC, hBitmap);                                      //--- 2. 새로 만든 DC에 그림을 선택한다

            BitBlt (hdc, 0, 0, 320, 320, hMemDC, 0, 0, SRCCOPY);                 //--- 3. DC간 블록 전송을 수행한다
             ...
}
```

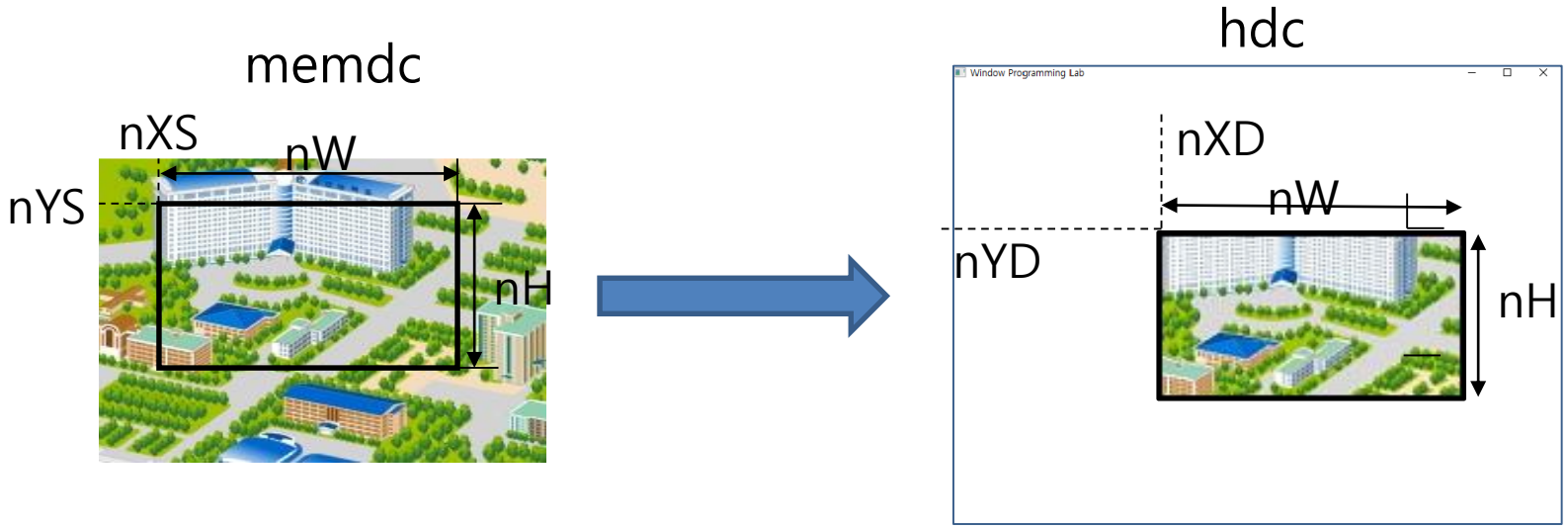
5) 비트맵 출력하기: BitBlt()

- **BitBlt 함수**
  - DC간의 영역끼리 고속 복사 수행
  - 메모리DC에 그려져 있는 비트맵을 화면 DC로 복사하여 비트맵을 화면에 출력
  - 1:1로 영역 복사

BOOL BitBlt (HDC hdc, int nXD, int nYD, int nW, int nH, HDC memdc, int nXS, int nYS, DWORD dwRop);

- DC 간의 영역 고속 복사
- 메모리 DC의 표면에 그려져 있는 비트맵을 화면 DC로 복사하여 비트맵을 화면에 출력
  - hdc: 복사 대상 DC
  - nXD, nYD: 복사 대상의 x, y 좌표 값
  - nW, nH: 복사 대상의 폭과 높이
  - memdc: 복사 소스 DC
  - nXS, nYS: 복사 소스의 좌표
  - dwRop: 래스터 연산 방법
    - BLACKNESS : 검정색으로 칠한다.
    - DSTINVERT: 대상의 색상을 반전시킨다.
    - MERGECOPY: 이미지 색상과 현재 선택된 브러시를 AND 연산시킨다.
    - MERGEPAINT: 반전된 이미지와 화면의 색을 OR 연산시킨다.
    - NOTSRCCOPY: 소스값을 반전시켜 칠한다.
    - SRCPAINT: 소스와 대상의 OR연산 값으로 칠한다.
    - **SRCCOPY**: 소스값을 그대로 칠한다.
    - SRCAND: 소스와 대상의 AND연산 값으로 칠한다.
    - WHITENESS: 흰색으로 칠한다.

BitBlt() → 1 : 1 영역 복사




```
BitBlt (hdc, nXD, nYD, nW, nH, memdc, nXS, nYS, SRCCOPY);
```

- **비트맵 출력 후, 메모리 DC와 비트맵 해제**

BOOL DeleteDC (HDC hdc);

- 생성한 메모리 DC를 제거한다.
  - HDC hdc: 제거 할 DC

BOOL DeleteObject (GDIOBJ hObject);

- 생성한 비트맵 객체를 제거한다.
  - GDIOBJ hObject: 제거할 객체

비트맵 출력

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
        HDC hdc, memdc ;
        PAINTSTRUCT ps ;
        static HBITMAP hBitmap;

        switch (iMsg) {
                case WM_CREATE:
                    hBitmap = (HBITMAP) LoadBitmap ( hInstance,  MAKEINTRESOURCE (IDB_BITMAP1));    //---1) 비트맵 로드하기
                    break;

                case WM_PAINT:
                    hdc = BeginPain t(hwnd, &ps);                          //--- 2) 화면 DC 얻어오기
                    memdc=CreateCompatibleDC (hdc);                        //--- 3) 메모리 DC 만들기
                    SelectObject (memdc, hBitmap);                         //--- 4) 비트맵 선택하기
                    BitBlt (hdc, 0, 0, 330, 240, memdc, 0, 0, SRCCOPY);    //--- 5) 비트맵 출력하기
                                                                           //--- memdc 에 있는 그림에서 (0, 0) 위치에 (320, 240) 크기로 그리기
                                                                           //--- SRCCOPY : 바탕색과 관계없이 소스값을 그대로 그리기
                    DeleteDC (memdc);
                    EndPaint (hwnd, &ps);
                    break;

                case WM_DESTROY:
                    DeleteObject (hBitmap);
                    PostQuitMessae (0);
                    break;
        }
        return DefWindowProc (hWnd, iMessage, wParam, lParam);
}
```

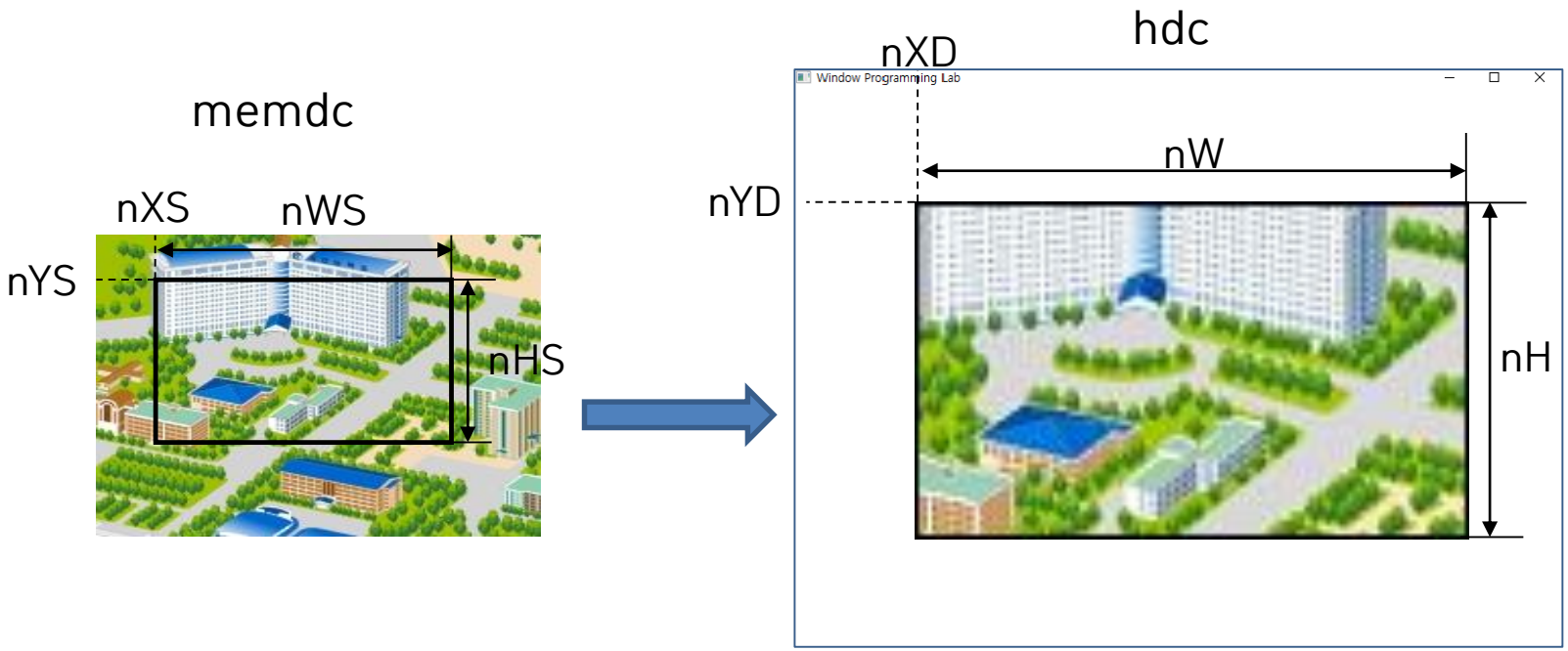
5) 비트맵 출력하기: StretchBlt()

- **StretchBlt 함수**
  - BitBlt 함수와 유사하게 DC간에 비트맵을 전송하여 복사
  - 복사원의 크기와 높이를 따로 지정하여 확대 및 축소 복사할 수 있다.

BOOL StretchBlt (HDC hdc, int nXD, int nYD, int nW, int nH,
HDC memdc, int nXS, int nYS, int nWS, int nHS, DWORD dwRop);

- DC간의 이미지 확대 또는 축소하여 복사
  - hdc: 복사대상 DC
  - nXD, nYD: 복사대상 DC x, y 좌표값
  - nW, nH: 복사대상 DC의 폭과 높이
  - memdc: 복사소스 DC
  - nXS, nYS: 복사소스 DC의 x, y 좌표값
  - nWS, nHS복사소스 DC의 폭과 높이
  - dwRop: 래스터 연산 방법
    - BLACKNESS : 검정색으로 칠한다.
    - DSTINVERT: 대상의 색상을 반전시킨다.
    - MERGECOPY: 이미지 색상과 현재 선택된 브러시를 AND 연산시킨다.
    - MERGEPAINT: 반전된 이미지와 화면의 색을 OR 연산시킨다.
    - NOTSRCCOPY: 소스값을 반전시켜 칠한다.
    - SRCPAINT: 소스와 대상의 OR연산 값으로 칠한다.
    - SRCCOPY: 소스값을 그대로 칠한다.
    - SRCAND: 소스와 대상의 AND연산 값으로 칠한다.
    - WHITENESS: 흰색으로 칠한다.

StretchBlt() → 확대, 축소 영역 복사



StretchBlt (hdc, nXD, nYD, nW, nH, memdc, nXS, nYS, nWS, nHS, SRCCOPY);

비트맵 출력

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
	HDC hdc, memdc ;
	PAINTSTRUCT ps ;
	static HBITMAP hBitmap;

	switch (iMsg) {
		case WM_CREATE:
			hBitmap = (HBITMAP) LoadBitmap ( hInstance,  MAKEINTRESOURCE (IDB_BITMAP1));
			break;
		case WM_PAINT:
			hdc = BeginPain t(hwnd, &ps);
			memdc=CreateCompatibleDC (hdc);
			SelectObject (memdc, hBitmap);
			StretchBlt (hdc, 100, 0, 160, 120, memdc, 0, 0, 320, 240, SRCCOPY);
                                        //--- 메모리 DC에 있는 그림에서 (0, 0)위치에서 (320, 240) 크기의 그림을
                                        //--- 화면의 (100, 0)위치에 (160, 120) 크기로 이미지 색 그대로 그리기
			DeleteDC (memdc);
			EndPaint (hwnd, &ps);
			break;
		case WM_DESTROY:
			DeleteObject (hBitmap);
			PostQuitMessae (0);
			break;

	}
	return DefWindowProc (hWnd, iMessage, wParam, lParam);
}
```

1-1) 비트맵 로드하기: LoadImage () 함수 (리소스 대신 이미지로 직접 사용하기)

- **비트맵 로드하기**
  - 비트맵을 읽어올 때: LoadImage 함수를 사용
    - 리소스로 불러온 이미지 파일 또는 이미지 파일 자체로도 읽기가 가능한다.

HANDLE **LoadImage** (HINSTANCE hInstance, LPCTSTR lpBitmapName, UINT uType,
int cxDesired, int cyDesired, UINT fuLoad);

- 비트맵, 아이콘, 커서를 로드 (LoadIcon, LoadCursor, LoadBitmap 함수를 통합)

  - HINSTANCE hInstance: 어플리케이션 인스턴스 핸들
  - LPCTSTR lpBitmapName: 비트맵 리소스 이름
    - 비트맵을 리소스 또는 파일 이름으로 사용할 수 있음.
    - 파일이름인 경우 fuLoad에 LR_LOADFROMFILE 플래그를 설정
  - UINT uType: 불러올 이미지의 종류
    - **IMAGE_BITMAP**: 비트맵 불러오기
    - IMAGE_ICON: 아이콘 불러오기
    - IMAGE_CURSOR: 커서 불러오기
  - int cxDesired, cyDesired: 아이콘이나 커서를 읽을 경우 아이콘이나 커서의 너비, 높이
  - UINT fuLoad: 플래그 설정
    - LR_DEFAULTCOLOR: 기본 플래그로 이미지를 흑백으로 불러오지 않도록 함
    - **LR_LOADFROMFILE**: lpBitmapName을 리소스 대신 파일이름을 사용해 불러옴
    - LR_DEFAULTSIZE: cxDesired, cyDesired 인자가 0인 경우 시스템 지정값 사용
    - **LR_CREATEDIBSECTION**: uType 인수에서 IMAGE_BITMAP을 사용한 경우 호환 비트맵이 아닌 DIB 섹션 비트맵으로 불러옴

비트맵 로드하기: LoadImage () 함수

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
        HDC hdc, memdc ;
        PAINTSTRUCT ps ;
        static HBITMAP hBitmap;

        switch (iMsg) {
            case WM_CREATE:
                hBitmap = (HBITMAP) LoadImage (hInstance, TEXT("image.bmp"), IMAGE_BITMAP, 0, 0,
                                                          LR_LOADFROMFILE | LR_CREATEDIBSECTION);        //--- 리소스 대신 비트맵 직접 로드
                break;

            case WM_PAINT:
                hdc = BeginPain t(hwnd, &ps);
                memdc=CreateCompatibleDC (hdc);                                  //--- 메모리 DC 생성하기
                SelectObject (memdc, hBitmap);                                   //--- 비트맵 이미지 선택하기
                StretchBlt (hdc, 100, 0, 160, 120, memdc, 0, 0, 320, 240, SRCCOPY);        //--- 비트맵 출력하기
                DeleteDC (memdc);
                EndPaint (hwnd, &ps);
                break;

            case WM_DESTROY:
                  DeleteObject (hBitmap);
                  PostQuitMessae (0);
                  break;
        }
        return DefWindowProc (hWnd, iMessage, wParam, lParam);
}
```

GetObject (): 그림 크기 알아내기

- 그림 크기 알아내기

int GetObject (HGDIOBJ hgdiobj, int cbBuffer, LPVOID lpvObject);

- 객체의 정보 알아오기
  - HGDIOBJ hgdiobj: GDI 오브젝트 핸들
  - int cbBuffer: 오브젝트 버퍼의 크기에 관한 정보
  - LPVOID lpvObject: 오브젝트 정보 버퍼를 가리키는 포인터

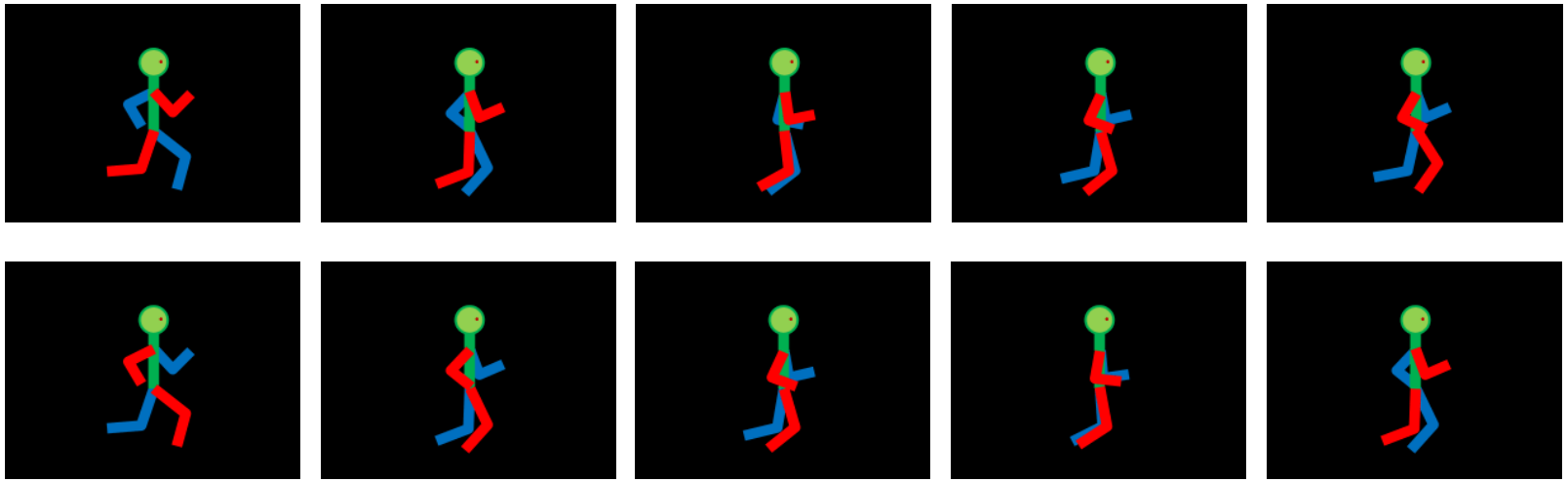
– 사용 예)

```
BITMAP bmp;                    //--- DDB 포맷의 비트맵 이미지 타입
int mWidth, mHeight;

GetObject (hBitmap, sizeof(BITMAP), &bmp);
mWidth = bmp.bmWidth;
mHeight = bmp.bmHeight;
```

```
typedef struct tagBITMAP {
        LONG bmType;            // 비트맵 타입: 0
        LONG bmWidth;           // 비트맵의 넓이 (픽셀 단위)
        LONG bmHeight;          // 비트맵의 높이 (픽셀 단위)
        LONG bmWidthBytes;      // 각 스캔 라인의 바이트 수
        WORD bmPlanes;          // 색상 판의 숫자
        WORD bmBitsPixel;       // 각 픽셀당 색상을 위한 비트수
        LPVOID bmBits;          // 비트맵을 가리키는 포인터
    } BITMAP, *PBITMAP;
```

3. 비트맵 애니메이션

- **애니메이션**
  - 각 시점에 다른 그림을 그려서 움직이는 효과를 얻는다.
    - 프레임(Frame): 움직이는 그림 중 하나의 동작이 그려진 이미지
  - 애니메이션 동작은 **타이머로** 처리한다.
  - 매 타이머의 주기에 각 프레임을 표시하여, 각 동작에 하나의 프레임 만을 보여준다.
  - 애니메이션 이미지들이 한 파일에 저장되어 있을 때는 한 프레임씩 이동하면서 필요한 부분을 잘라내어 번갈아 표시한다. 오프셋 개념을 이용한다
    - 오프셋 (Offset): 동일 오브젝트 안에서 오브젝트 처음부터 주어진 요소나 지점까지의 변위차를 나타내는 정수형



비트맵 애니메이션

```
HBITMAP RunBit[10];

LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
	static int xPos=0;

	switch (iMsg)
	{
	case WM_CREATE:
		RunBit[0]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R1));		//-- 필요한 애니메이션 이미지들을 로드하기
		RunBit[1]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R2));
		RunBit[2]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R3));
		. . . . . .
		RunBit[8]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R9));
		RunBit[9]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R10));
		SetTimer(hwnd, 1, 100, NULL);
		break;
	case WM_TIMER:
		xPos += 10;                                                    //--- 애니메이션의 x 위치 변경하기
		if (xPos > 800)		xPos = 0;
		InvalidateRect (hwnd, NULL, true);
		return 0;
	case WM_PAINT:
		hdc = BeginPaint(hwnd, &ps);
		Animation (xPos, 300, hdc);                                    //--- (xPos, 300) 위치에 애니메이션 그리기
		EndPaint(hwnd, &ps);
		break;
	case WM_DESTROY:
		for (int i = 0; i < 10; i++)		DeleteObject (RunBit[i]);
		PostQuitMessage (0);
	}
	return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

비트맵 애니메이션

//--- 10개의 이미지를 사용하여 애니메이션 구현

```
void Animation (int xPos, int yPos, HDC hdc)
{
	HDC    memdc;
	HBITMAP hBit, oldBit;
	static int count;
	int i;

	count++;
	count = count % 10;

	hBit = LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_BACK));	//--- 배경 이미지 로드하기

	memdc = CreateCompatibleDC (hdc);

	oldBit = (HBITMAP) SelectObject (memdc, hBit);			//--- 배경 이미지를 메모리 DC에 올리기
	BitBlt (hdc, 0, 0, 800, 600, memdc, 0, 0, SRCCOPY);

	(HBITMAP) SelectObject (memdc, RunBit[count]);			//--- 순서대로 전경 이미지를 메모리 DC에 올리기
	BitBlt (hdc, xPos, yPos, 64, 64, memdc, 0, 0, SRCCOPY);

	SelectObject (memdc, oldBit);

	DeleteDC (memdc);						//--- 메모리 DC와 로드한 배경 이미지 삭제하기
	DeleteObject (hBit);
}
```

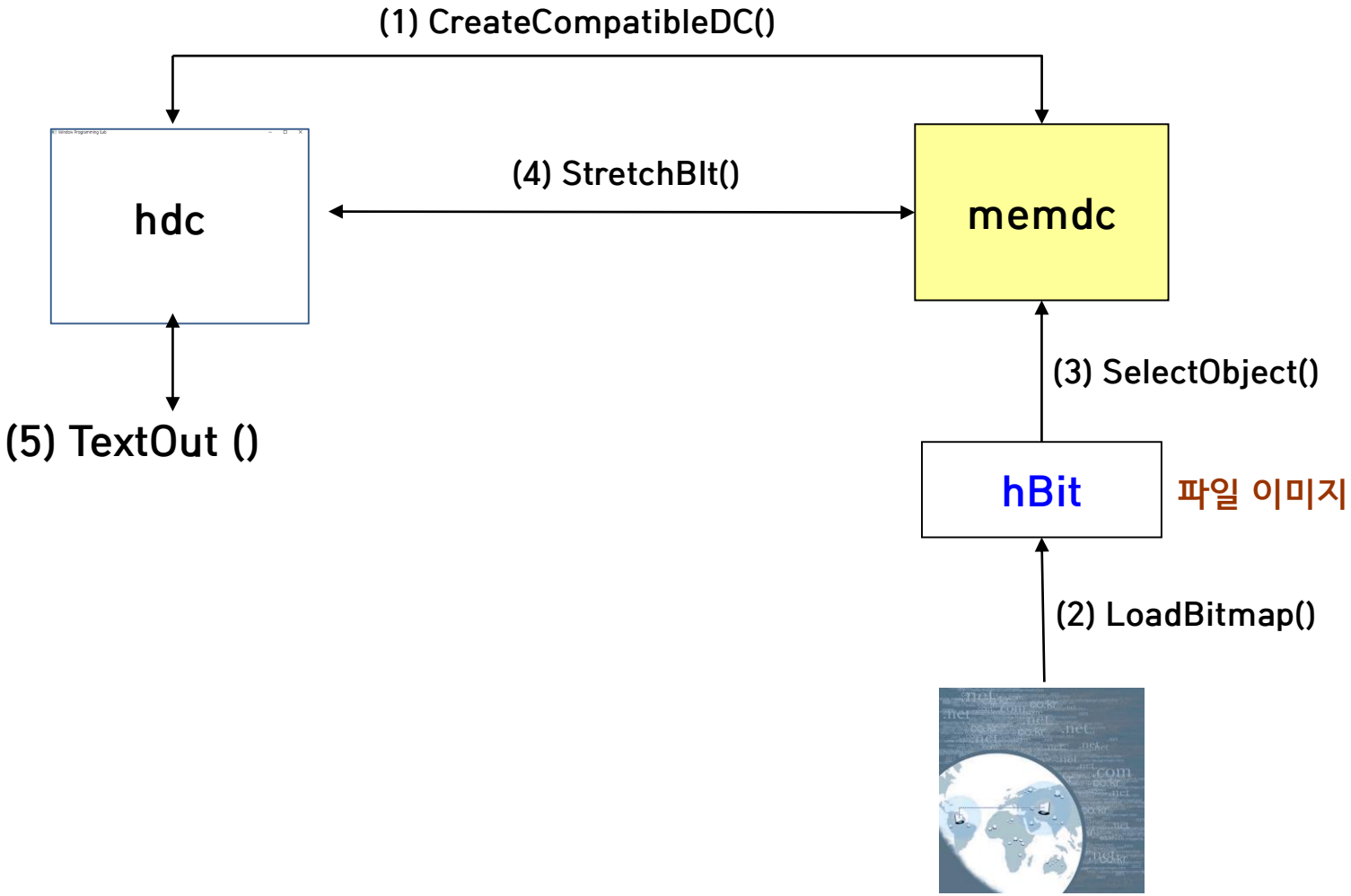
4. 더블 버퍼링

- **비트맵 이미지 여러 개를 이용하여 애니메이션을 나타낼 때**
  - 이미지를 순서대로 화면에 출력
  - 예를 들어 풍경 위에 날아가는 새를 표현한다면
    1. 풍경 이미지를 먼저 출력
    2. 그 다음에 새 이미지를 출력
    3. 날아가는 모습을 나타내고자 한다면 풍경 이미지 출력과 새 이미지 출력을 번갈아 가며 계속 수행
  - 이미지의 잦은 출력으로 인해 화면이 자주 깜박거리는 문제점

- **문제점 해결**
  - 메모리 디바이스 컨텍스트를 하나 더 사용
  - 추가된 메모리 디바이스 컨텍스트에 그리기를 원하는 그림들을 모두 출력한 다음 화면 디바이스 컨텍스트로 한꺼번에 옮기는 방법을 이용

- **추가된 메모리 디바이스 컨텍스트가 추가된 버퍼 역할을 하기 때문에 이 방법을 더블버퍼링이라 부름**

배경화면 위로 움직이는 글: 기존 방법



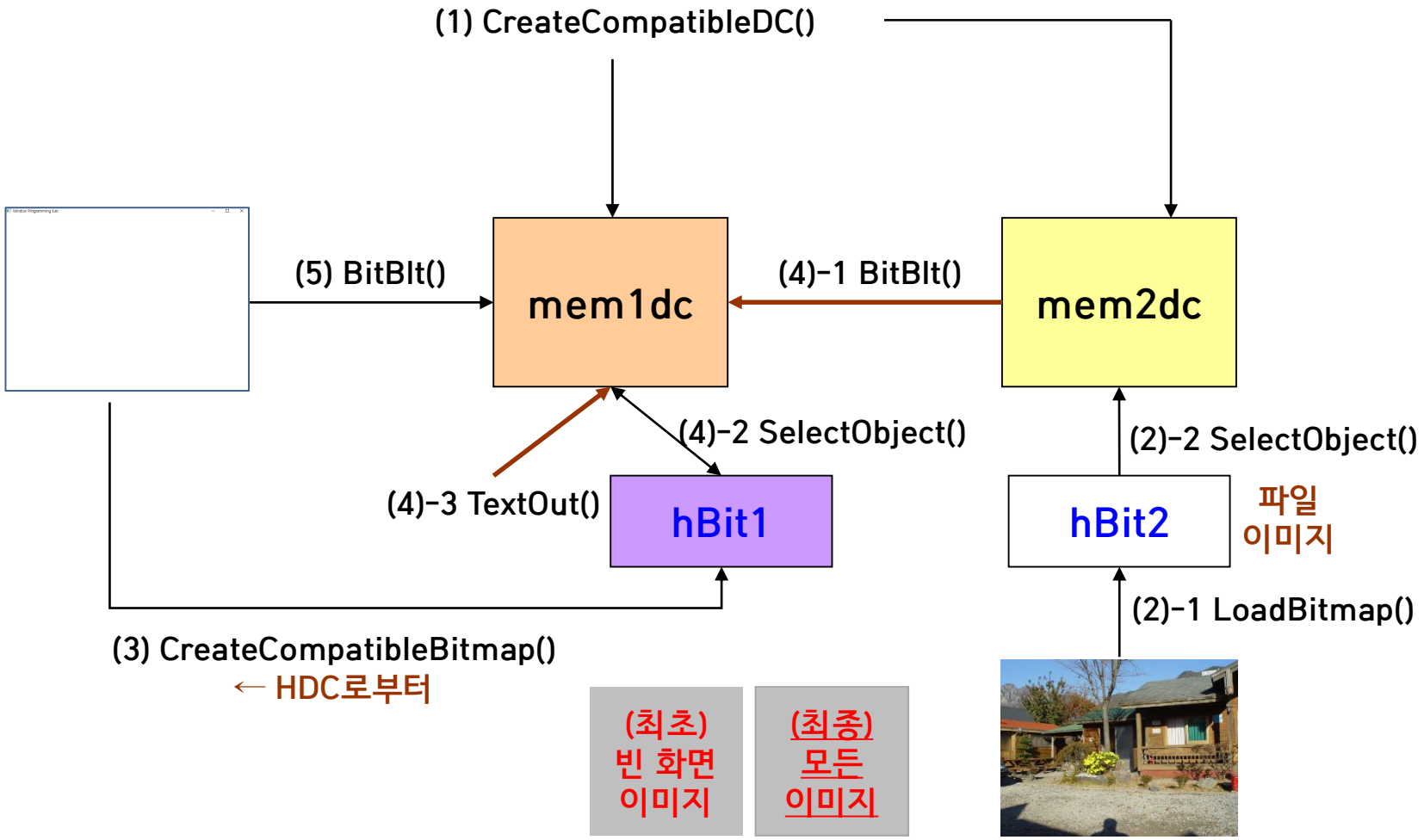
배경화면 위로 움직이는 글

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc, memdc;
        static HBITMAP hBit, oldBit;
        TCHAR word[] = L"움직이는 그림";

        switch(iMsg) {

                case WM_CREATE:
                    yPos = -30;                                    //--- -30: 글자의 높이 고려
                    GetClientRect (hwnd, &rectView);
                    SetTimer (hwnd, 1, 70, NULL);
                    hBit=LoadBitmap (hInstance, MAKEINTRESOURCE(IDB_BITMAP1));
                    break;

                case WM_TIMER:                                     //--- Timeout 마다 y좌표 변경 후, 출력 요청
                    yPos += 5;
                    if (yPos > rectView.bottom)
                                yPos = -30;
                    InvalidateRect (hwnd, NULL, true);
                    break;
```

배경화면 위로 움직이는 글

```
        case WM_PAINT:
            hdc=BeginPaint(hwnd, &ps);

        //--- 이미지 로드
            hBit=LoadBitmap (hInstance, MAKEINTRESOURCE(IDB_BITMAP1));
            memdc = CreateCompatibleDC (hdc);

        //--- 이미지 출력
            oldBit=(HBITMAP)SelectObject(memdc, hBit);

        //--- 메모리 DC -> 화면 DC(hdc)로 이동, 출력
            StretchBlt (hdc, 0, 0, rectView.right, rectView.bottom,          memdc, 0, 0, rectView.right, rectView.bottom, SRCCOPY);
            SelectObject (memdc, oldBit);
            DeleteDC (memdc);

        //--- 문자열 출력
            TextOut(hdc, 200, yPos, word, strlen(word));

            EndPaint(hwnd, &ps);
            break;

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

더블 버퍼링


(1) CreateCompatibleDC()
(5) BitBlt()
mem1dc
(4)-1 BitBlt()
mem2dc
(4)-2 SelectObject()
(2)-2 SelectObject()
(4)-3 TextOut()
hBit1
hBit2
파일
이미지
(2)-1 LoadBitmap()
(3) CreateCompatibleBitmap()
← HDC로부터
(최초)
빈 화면
이미지
(최종)
모든
이미지

배경화면 위로 움직이는 글: 더블버퍼링 적용

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

    HDC hdc, memdc;
    HDC mem1dc, mem2dc;
    static HBITMAP hBit1, hBit2;
    HBITMAP oldBit1, oldBit2;
    ...
    TCHAR word[] = L"더블 버퍼링 실습";


    switch(iMsg) {
        case WM_CREATE:
            yPos = -30;
            GetClientRect(hwnd, &rectView);
            SetTimer(hwnd, 1, 70, NULL);

            //--- hBit2에 배경 그림 로드, 나중에 mem2dc에 hBit2 그림 설정
            hBit2 = LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BITMAP4));
            break;
```

배경화면 위로 움직이는 글: 더블버퍼링

```
case WM_TIMER:
        yPos += 5;
        if (yPos > rectView.bottom)                yPos = -30;
        hdc = GetDC(hwnd);

        if (hBit1 == NULL)                                          //--- hBit1을 hdc와 호환되게 만들어준다.
        hBit1 = CreateCompatibleBitmap (hdc, 1024, 768);

        //--- hdc와 호환되는 mem1dc를 만들어준다.
        mem1dc = CreateCompatibleDC (hdc);

        //---  mem1dc와 호환되는 mem2dc를 만들어준다.
        mem2dc = CreateCompatibleDC (mem1dc);

        //--- mem2dc의 비트맵을 mem1dc에 옮기고, mem1dc를 hdc로 옮기려고 함
        //--- hBit1의 이미지를 mem1dc로, hBit2의 이미지를 mem2dc로 선택
        oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);          //--- mem1dc에는 hBit1
        oldBit2 = (HBITMAP) SelectObject (mem2dc, hBit2);          //--- mem2dc에는 hBit2: hBit2에는 배경 그림이 저장되어 있음

        //--- mem2dc에 있는 배경그림을 mem1dc에 옮긴다.
        BitBlt(mem1dc, 0, 0, 1024, 768, mem2dc, 0, 0, SRCCOPY);

        SetBkMode (mem1dc, TRANSPARENT);
        TextPrint (mem1dc, 200, yPos, word);                       //--- mem1dc에 텍스트 출력

        //--- 저장한 비트맵 핸들값을 DC에 원상복귀, 생성된 MDC 삭제
        SelectObject (mem2dc, oldBit2);                 DeleteDC (mem2dc);
        SelectObject (mem1dc, oldBit1);                 DeleteDC (mem1dc);
        ReleaseDC (hwnd, hdc);
        InvalidateRgn (hwnd, NULL, false);          //--- 다시 그리기를 할 때 배경 이미지를 지우지 않고 출력하기 위해 마지막 인자를 false로 설정
        break;
```

배경화면 위로 움직이는 글: 더블버퍼링

```
    case WM_PAINT:
            GetClientRect (hwnd, &rectView);
            hdc = BeginPaint (hwnd, &ps);

            mem1dc = CreateCompatibleDC (hdc);

            //--- hBit1에는 배경과 텍스트가 출력된 비트맵이 저장되어 있다.  이 비트맵을 mem1dc에 선택
            oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);

            //--- mem1dc에 있는 내용을 hdc에 복사한다.
            BitBlt (hdc, 0, 0, 1024, 768, mem1dc, 0, 0, SRCCOPY);

            SelectObject (mem1dc, oldBit1);
            DeleteDC (mem2dc);
            EndPaint (hwnd, &ps);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam)
}
```

배경화면 위로 움직이는 글 (더블버퍼링)

HDC CreateCompatibleDC (HDC hdc);

- 주어진 DC와 호환되는 메모리 DC를 생성해 준다.
- 주어진 DC가 사용하는 출력장치의 종류나 출력장치가 사용중인 그래픽 드라이버 정보를 가지고 새로운 DC 를 만든다.
- hdc와 동일한 방법으로 그림을 그리지만 화면에 출력은 되지 않는다.

  - HDC hdc: 주어진 DC

HBITMAP CreateCompatibleBitmap (HDC hdc, int nWidth, int nHeight);

- hdc와 호환되는 비트맵을 생성하여 반환하는 함수
- 화면 DC와 호환되게 만들어야 한다. (메모리 DC와 호환되게 만들면 1비트 색상수를 사용하는 비트맵을 생성하게 된다.)
- CreateCompatibleDC로 생성한 DC를 사용하려면 비트맵 객체를 만들어서 연결하여 사용
- 생성된 비트맵은 hdc와 호환되는 어떤 메모리 DC에서도 선택되어질 수 있다.

  - HDC hdc: DC 핸들
  - int nWidth/nHeight: 작성하는 비트맵의 가로/세로 사이즈

더블 버퍼링 사용하여 도형 그리기

- **도형 그리기에서 더블 버퍼링 사용하기**

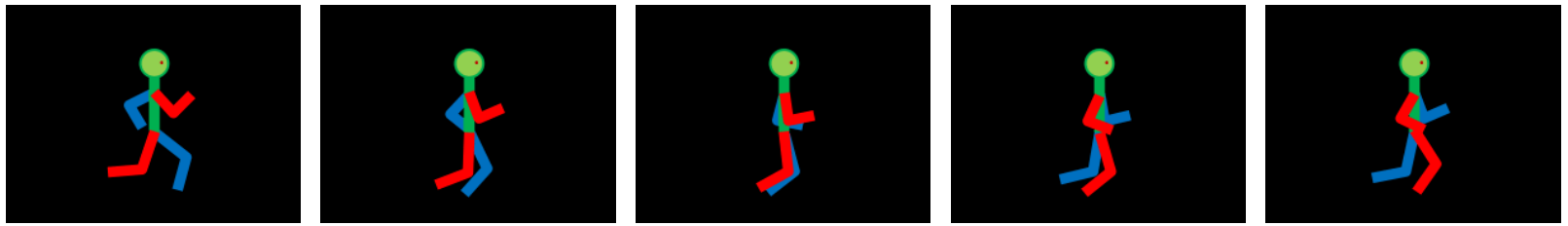
```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
	static HDC mdc;
	static HBITMAP hBitmap;

	switch (iMsg) {

	case WM_CREATE:
		hdc = GetDC(hwnd);
		mdc = CreateCompatibleDC(hdc);
		hBitmap = CreateCompatibleBitmap(hdc, rt.right, rt.bottom);
		SelectObject(mdc, (HBITMAP)hBitmap);
		ReleaseDC(hwnd, hdc);
	break;

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

		//--- 모든 객체는 메모리DC에 그리기
		Rectangle (mdc, 0, 0, rt.right, rt.bottom);
		DrawBlock (mdc, blockYnum + 1, blockXnum);
		Rectangle (mdc, rect.left, rect.top, rect.right, rect.bottom);

		Ellipse (mdc, s_ellipse.left, s_ellipse.top, s_ellipse.right, s_ellipse.bottom);

		//--- 메모리DC에 그려진 그림을 화면 DC에 복사하기
		BitBlt (hdc, 0, 0, rt.right, rt.bottom, mdc, 0, 0, SRCCOPY);

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

```
	case WM_TIMER:
		//--- 좌표값 변화
		s_ellipse.left += xStep;
		s_ellipse.right += xStep;
		s_ellipse.top += yStep;
		s_ellipse.bottom += yStep;

		//--- 공과 윈도우 테두리의 충돌체크
		if (s_ellipse.right > width)
		xStep *= -1;
		if (s_ellipse.bottom > height)
		yStep *= -1;
		InvalidateRect(hwnd, NULL, false);
		break;

	case WM_DESTROY:
		DeleteDC(mdc);
		DeleteObject(hBitmap);

		PostQuitMessage(0);
		return 0;
	}

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

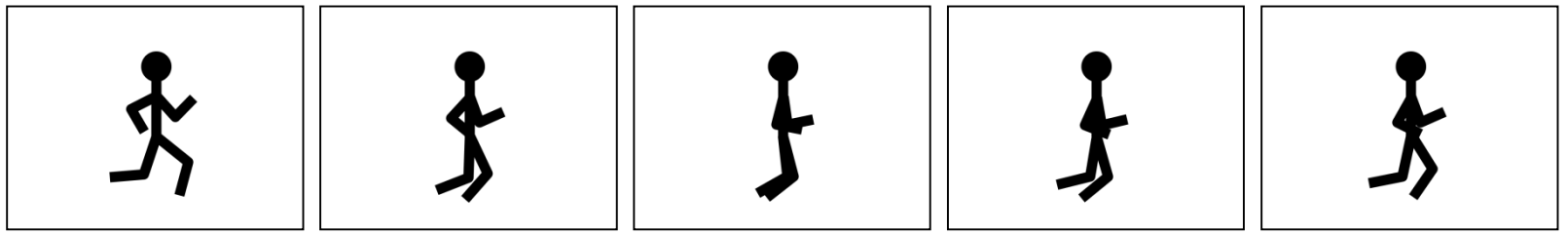
5. 비트맵 마스크

- 사각형의 비트맵 이미지에서 원하는 부분만을 사용하고 싶을 때, 그리려는 비트맵 이미지 부분에 마스크를 씌운다.

  – 필요한 이미지:
    - 비트맵 이미지
    - 출력하고자 하는 부분을 흑색 처리한 마스크



비트맵 이미지



마스크 이미지

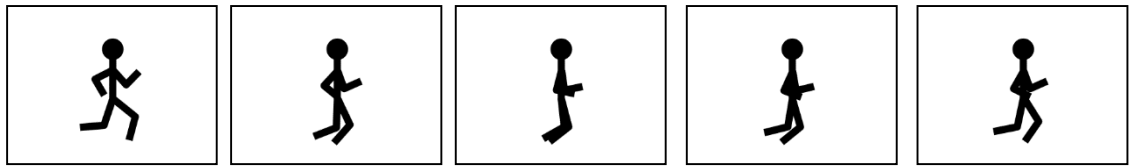
비트맵 마스크

- **처리 방법:**
  - 각 프레임의 동작마다 마스크 이미지와 출력하고자 하는 소스 이미지의 그림을 각각 두번씩 씌워주어야 한다.

1. 소스의 원하는 부분을 흑백으로 처리한 마스크를 배경 그림과 **AND** 연산 → 배경 이미지에 흑색 마스크가 그려진다.

   BitBlt (hdc, x, y, size_x, size_y, BitmapMaskDC, mem_x, mem_y, **SRCAND**);
   - SRCAND: 소스와 대상의 AND 연산값으로 칠한다.
     - » 마스크와 배경이미지의 AND 연산

배경 이미지

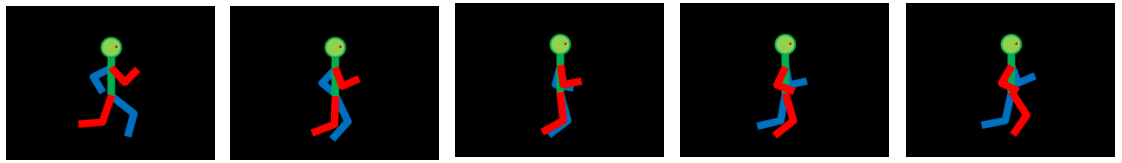


마스크 이미지

2. 여기에 원하는 그림을 배경 그림과 OR 연산 → 배경과 합성된 이미지로 나타나게 된다.

   BitBlt (hdc, x, y, size_x, size_y, hBitmapFrontDC, mem_x, mem_y, **SRCPAINT**);
   - SRCPAINT: 소스와 대상의 OR 연산값으로 칠한다.
     - » 출력하고자 하는 이미지와 배경이미지의 OR 연산

배경 이미지



비트맵 이미지

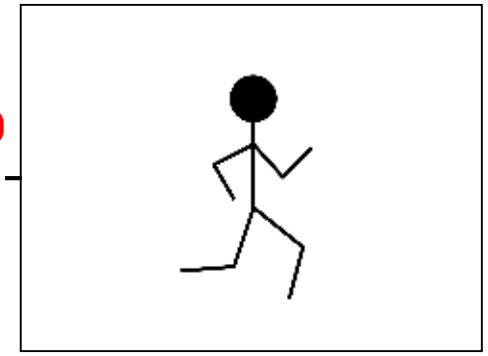
비트맵 마스크

1. 마스크 그리기: AND 연산

컬러(X, X, X)

흰색(255,255,255)

AND



검은색(0, 0, 0)

흰색(255,255,255)
AND 컬러(X , X , X)

컬러(X , X , X)

검은색(0, 0, 0)
AND 컬러 (X, X, X)

검은색(0, 0, 0)

비트맵 마스크

2. 캐릭터 그리기: OR 연산

컬러(X, X, X)
검은색(0, 0, 0)

OR

검은색(0, 0, 0)

컬러(X, X, X)

		검은색(0,	0,	0)
OR		컬러 (X,	X,	X)
		컬러 (X,	X,	X)

비트맵 마스크

3. 결과: 배경 위에 캐릭터가 올려진 결과 이미지

비트맵 마스크

```
HBITMAP       RunBit[5], Mask[5];

LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc;
        PAINTSTRUCT ps;
        static int xPos=0, yPos;

        switch (iMsg)
        {
        case WM_CREATE:
                RunBit[0]= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_R1));     //--- 애니메이션 이미지 로드하기: 5개
                RunBit[4]= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_R5));

                Mask[0] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_M1));     //--- 마스크 이미지 로드하기: 5개
                Mask[4] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_M5));
                SetTimer (hwnd, 1000, 100, NULL);
                break;
        case WM_PAINT:
                hdc = BeginPaint (hwnd, &ps);
                Animation (xPos, yPos, hdc);
                EndPaint (hwnd, hdc);
                break;
        case WM_TIMER:
                xPos += 10;
                InvalidateRect (hwnd, NULL, false);
                break;
        case WM_DESTROY:
                for (i = 0; i < 10; i++) {
                    DeleteObject (RunBit[i]);      DeleteObject (Mask[i]);
                }
                PostQuitMessage (0);
                break;
        }
        return DefWindowProc (hWnd, iMessage, wParam, lParam);
}
```

비트맵 마스크

```
void Animation (int xPos, int yPos, HDC hdc)
{
	HDC	memdc;
	static int count;
	int	i;

	count++;

	memdc = CreateCompatibleDC(hdc);
	hBit = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP5));	// 배경 이미지
	(HBITMAP)SelectObject(memdc, hBit);
	BitBlt(hdc, 0, 0, 819, 614, memdc, 0, 0, SRCCOPY);

	(HBITMAP) SelectObject (memdc, Mask[count]);
	BitBlt (hdc, xPos, yPos, 180, 240, memdc, 0, 0, SRCAND);

	(HBITMAP) SelectObject (memdc, RunBit[count]);
	BitBlt (hdc, xPos, yPos, 180, 240, memdc, 0, 0, SRCPAINT);

	DeleteDC (memdc);
	DeleteObject (hBit);
}
```

마스크를 그린다.

캐릭터를 그린다.

6. 투명 비트맵 처리

- **비트맵의 일부를 투명하게 처리하여 투명색 부분은 출력에서 제외한다.**
  - 1개의 특정 색을 투명하게 설정한다.
  - BitBlt 함수나 StretchBlt 함수 대신 사용할 수 있다.

BOOL TransparentBlt (HDC hdcDest, int nXOriginDest, int nYOriginDest, int nWidthDest, int hHeightDest,
HDC hdcSrc, int nXOriginSrc, int nYOriginSrc, int nWidthSrc, int nHeightSrc, UINT crTransparent);

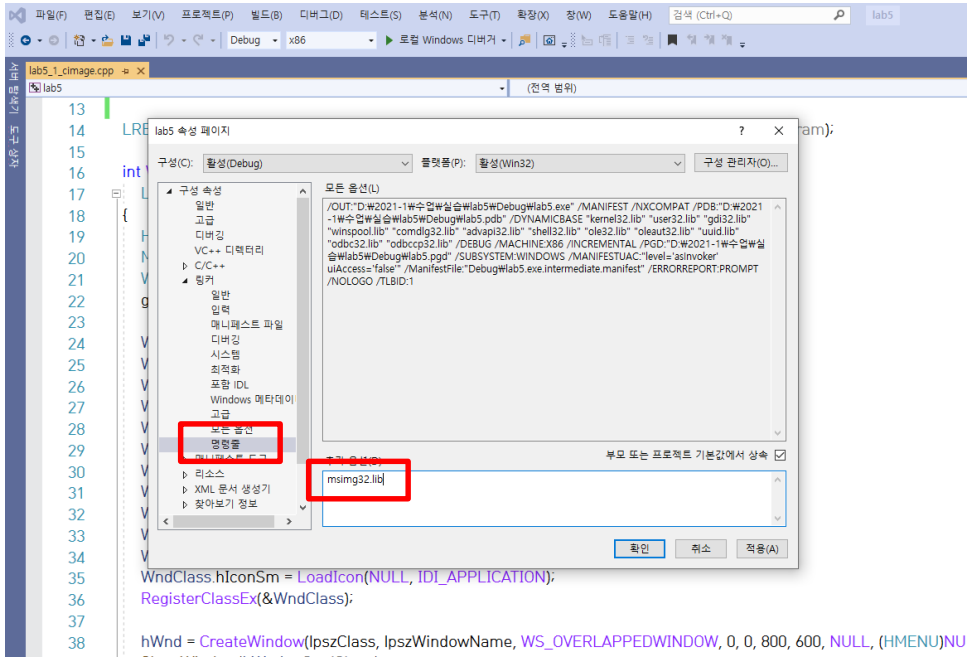
- 비트맵의 특정 색을 투명하게 처리하는 함수
  - HDC hdcDest: 출력할 목표 DC 핸들
  - int nXOriginDest : 좌측 상단의 x 좌표값
  - int nYOriginDest : 좌측 상단의 y 좌표값
  - int nWidthDest : 목표 사각형의 넓이
  - int hHeightDest : 목표 사각형의 높이

  - HDC hdcSrc : 소스 DC 핸들
  - int nXOriginSrc : 좌측 상단의 x 좌표값
  - int nYOriginSrc : 좌측 상단의 y 좌표값
  - int nWidthSrc : 소스 사각형의 넓이
  - int nHeightSrc : 소스 사각형의 높이

  - UINT crTransparent : 투명하게 설정할 색상

투명 비트맵 처리

- 라이브러리 추가
  - **msimg32.lib 라이브러리**를 링크한다.
    - 프로젝트에 추가: 프로젝트 속성 → 링커 → 명령줄에서 라이브러리 추가
    - 또는 전처리기를 사용: #pragma comment (lib, "msimg32.lib")




```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc, memdc ;
         PAINTSTRUCT ps ;
         static HBITMAP hBitmap;

         switch (iMsg) {

         case WM_CREATE:
              hBitmap = (HBITMAP) LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BITMAP7));
              break;

         case WM_PAINT:
              hdc = BeginPain t(hwnd, &ps);
              memdc=CreateCompatibleDC (hdc);
              SelectObject (memdc, hBitmap);

              TransparentBlt (hdc, 0, 0, 100, 100, memdc, 10, 50, 100, 100, RGB(0, 0, 0) );      //--- 마지막 인자를 RGB(0, 0, 0)으로 설정:  검정색을 투명하게 설정한다

              DeleteDC (memdc);
              EndPaint (hwnd, &ps);
              break;
         }
         return DefWinProc (hwnd, iMsg, wParam, lParam);
}
```

그 외 여러 비트맵 함수들

함수 설명	함수 프로토타입
점 찍기	COLORREF **SetPixel** (HDC hdc, int x, int y, COLORREF color);
점의 색상 알아보기	COLORREF **GetPixel** (HDC hdc, int x, int y);
지정한 사각 영역을 채색한다. (현재 DC에 선택되어 있는 브러시와 화면의 색상을 논리 연산)	BOOL **PatBlt** (HDC hdc, int x, int y, int nWidth, int nHeight, DWORD dwrop);
흑백 마스크 이미지를 사용하여 소스와 목적 색상을 혼합한다.	BOOL **MaskBlt** (HDC hdcDest, int xDest, int yDest, int width, int height, HDC hdcSrc, int xSrc, int ySrc, HBITMAP hbmMask, int xMask, int yMask, DWORD rop);
사변형 모양의 이미지 전송 (이미지 회전 가능)	BOOL **PlgBlt** (HDC hdcDest, const POINT *lpPoint, HDC hdcSrc, int xSrc, int ySrc, int width, int height, HBITMAP hbmMask, int xMask, int yMask);

반투명 픽셀 설정
 	
BLENDFUNCTION 구조체:
typedef struct BLENDFUNCTION {
BYTE BlendOp;
BYTE BlendFlags;
BYTE SourceConstantAlpha;
BYTE AlphaFormat;
};
	BOOL **AlphaBlend** (HDC hdcDest, int xoriginDest, int yoriginDest, int wDest, int hDest,
	HDC hdcSrc, int xoriginSrc, int yoriginSrc, int wSrc, int hSrc, BLENDFUNCTION ftn);
	
	//--- BLENDFUNCTION 설정 예) ➔ msimg32.lib 링크
	BLENDFUNCTION bf;
	
	bf.AlphaFormat = 0; // 일반 비트맵: 0, 32비트 비트맵: AC_SRC_ALPHA
	bf.BlendFlags = 0; // 무조건 0
	bf.BlendOp = AC_SRC_OVER; // 무조건 AC_SRC_OVER: 원본과 대상 이미지를 합침
	bf.SourceConstantAlpha = 127; // 투명도(투명 0 - 불투명 255)
	AlphaBlend (hDC, 0, 0, w, h, hMemDC, 0, 0, w, h, bf);

참고: CImage 클래스 사용하기

- **CImage는 그림 관련 클래스**
  - ATL에서 추가된 이미지 관리 클래스
    - ATL (Active Template Library): 작고 빠른 구성 요소 개체 모델 (COM, Component Object Model)을 만들 수 있도록 해주는 템플릿 기반의 C++ 클래스 집합
  - CImage는
    - 비트맵 관리 클래스, 비트맵 정보를 내부에서 보유
    - 여러 종류의 이미지 포맷을 지원: bmp 파일 외에 확장하여 png, jpg, gif 등의 다양한 포맷을 지원한다.
    - API 에서 지원되는 비트맵 관련 다양한 함수들이 지원된다: BitBlt, StretchBlt, TransparentBlt, AlphaBlend 같은 그림 관련 함수들이 지원된다.
  - atlImage.h 를 포함해야 한다.
  - 자세한 설명은
    - https://docs.microsoft.com/en-us/cpp/atl-mfc-shared/reference/cimage-class (영문)
    - https://msdn.microsoft.com/ko-kr/library/bwea7by5(v=vs.120).aspx (한글)

CImage 클래스 사용하기

- **CImage는 그림 관련 클래스**
  - 사용 가능한 public method들
    - Create (int nWidth, int nHeight, int nBitPerPixel, DWORD dwFlags);
      - Cimage 비트맵 생성
    - Destroy ();
      - Cimage 개체와 비트맵 삭제
    - Load (LPCTSTR pszFileName);
      - 이미지를 로드한다.
    - LoadFromResource (HINSTANCE hInstance, LPCTSTR pszResourceName);
      - 비트맵 리소스에서 이미지를 로드한다.
    - GetHeight (); GetWidth ();
      - 이미지 픽셀에서 높이/폭 값 리턴
    - Draw (HDC hDestDC, int xDest, int yDest, int nDestWidth, int nDestHeight, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight);
      - 소스 사각형에서 대상 사각형으로 비트맵을 복사
    - BitBlt (HDC hDestDC, int xDest, int yDest, int nDestWidth, int nDestHeight, int xSrc, int ySrc, DWORD dwROP);
      - 소스 디바이스 컨텍스트에서 목적지 장치 컨텍스트로 비트맵을 복사
    - StretchBlt (HDC hDestDC, int xDest, int yDest, int nDestWidth, int nDestHeight, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwROP);
      - 소스 디바이스 컨텍스트에서 목적지 장치 컨텍스트로 비트맵을 크기 변경하여 복사
    - AlphaBlend (HDC hDestDC, int xDest, int yDest, int nDestWidth, int nDestHeight, int xSrc, int ySrc, int nSrcWidth, int nSrcHeight, BYTE bSrcAlpha = 0xff, BYTE bBlendOp = AC_SRC_OVER);
      - 투명하거나 반투명 이미지

CImage 클래스 사용하기

- **사용 예) Cimage를 사용하여 비트맵 출력하기**

```
#include <atlImage.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,  LPARAM lParam)
{
	PAINTSTRUCT ps;
	HDC hdc;
	CImage img;

	switch (message)
	{
		case WM_PAINT:
			hdc = BeginPaint (hWnd, &ps);
			img.Load (TEXT("bitmap1.png"));                    //--- bitmap1.png 파일을 로드하기
			nWidth = img.GetWidth();
			nHeight = img.GetHeight();

			img.Draw (hdc, 0, 0, rect.right, rect.bottom, 0, 0, nWidth, nHeight);
			//--- img.BitBlt (hdc, 0, 0, rect.right, rect.bottom, 0, 0, SRCCOPY);
			//--- img.StretchBlt (hdc, rect, SRCCOPY);

			EndPaint (hWnd, &ps);
			img.Destroy ();
		break;
	}
}
```

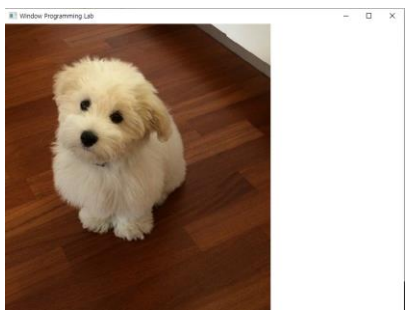
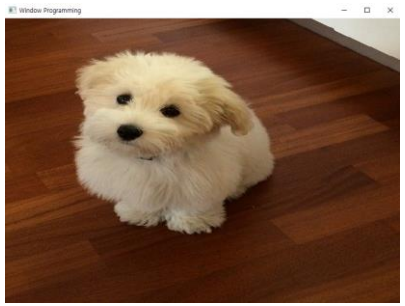
CImage 클래스 사용하기

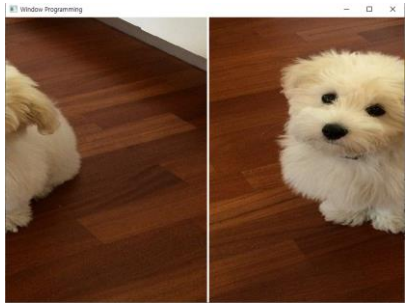
- **사용 예) 더블 버퍼링 사용하기: 비트맵1을 배경으로 그리고 비트맵2가 튕기는 애니메이션 그리기**

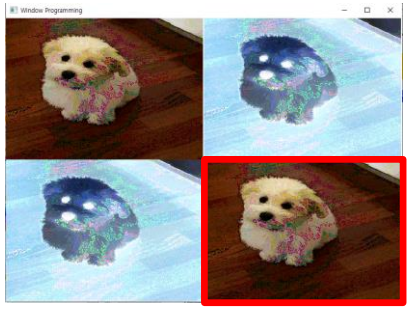
```
#include <atlImage.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,  LPARAM lParam)
{
	static Cimage img, imgSprite;
	static int xPos=0, yPos=0;
	static RECT rect;
	switch (message)
	{
		case WM_CREATE:
			img.Load (L"bitmap1.bmp");                          //--- background
			imgSprite.Load (L"bitmap2.bmp");                    //--- sprite image
			GetClientRect (hWnd, &rect);
		break;
		case WM_PAINT:
			hdc = BeginPaint(hwnd, &ps);
			hBitmap = CreateCompatibleBitmap (hdc, rect.right, rect.bottom);
			memdc = CreateCompatibleDC (hdc);
			(HBITMAP) SelectObject (memdc, hBitmap);
			w = img.GetWidth();
			h = img.GetHeight();
			img.Draw (memdc, 0, 0, rect.right, rect.bottom, 0, 0, w, h);          //--- 메모리 DC에 배경 그리기
			imgSprite.Draw (memdc, xPos, yPos, 100, 100, 0, 0, 100, 100);         //--- 메모리 DC에 스프라이트 그리기
			BitBlt (hdc, 0, 0, rect.right, rect.bottom, memdc, 0, 0, SRCCOPY);    //--- 메모리 DC의 그림을 화면 DC에 복사하기
			DeleteObject (hBitmap);
			DeleteDC (memdc);
			EndPaint (hwnd, &ps);
		break;
		case WM_TIMER:
			xPos += 5;
			yPos += 5;
			InvalidateRect (hwnd, NULL, false);
		break;
	}
}
```

실습 5-1

- **윈도우에 배경 그림 넣기**
  - 그림을 화면에 출력한다.
    - 원래의 비트맵 크기대로 화면에 출력한다. (윈도우보다 작은 크기의 비트맵 사용)
  - 다음의 명령어를 실행한다.
    - **r/R**: 색상을 반전/원래로 출력시킨다.
    - **a/A**: 윈도우에 빈 공간 없이 배경 그림을 그린다.
    - **2**: 화면의 가로 세로를 각각 2등분하여 4 구간에 각각 다른 래스터 연산을 적용하여 그린다.
      - 화면에 그림이 그려지는 래스터 연산을 적용하도록 한다.
    - **3:** 화면의 가로 세로를 각각 3등분하여 9 구간에 다른 래스터 연산을 적용하여 그린다.
      - 화면에 그림이 그려지는 래스터 연산을 적용하도록 한다.
    - **←/→**: 화면의 이미지를 좌/우로 조금씩 이동한다. 이동방향으로 안 그려지는 부분은 반대편에 그려진다.
      - 화면이 등분되어 1개 이상의 그림이 그려졌을 때는 해당 등분의 그림을 왼쪽 마우스 버튼으로 선택하여 그 그림만 이동한다.
      - 선택되었다는 것을 표시하도록 한다.
    - **+/-**: 비트맵의 크기를 조금씩 확대/축소해서 그린다.
    - **q/Q**: 프로그램 종료

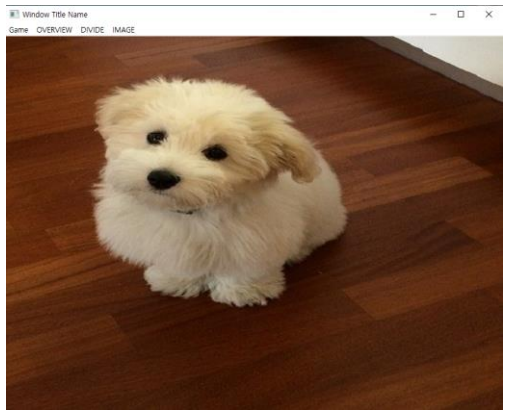



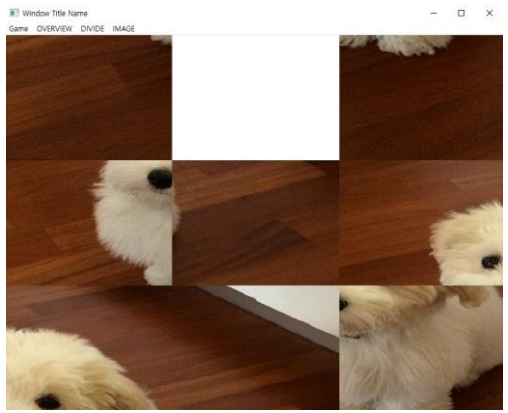




실습 5-2

- **조각 퍼즐 맞추기**
  - 2개의 밑 그림을 사용한다. 메뉴를 이용하여 그림을 선택하게 한다.
    - 그림 퍼즐 한 칸이 비어있고, 다른 퍼즐을 움직여서 그림 맞추기를 진행한다.
    - 퍼즐이 맞으면 게임 종료 메시지 박스를 띄운다.
  - **메뉴**:
    - **그림**: 그림 1 / 그림 2 → 밑 그림 1 / 밑 그림 2
    - **그림 나누기**: 3 / 4 / 5 → 밑그림의 가로와 세로를 숫자에 맞게 분할하여 랜덤하게 화면에 배치
    - **게임**: 게임 시작 / 전체 그림 보기 / 반전 그림 / 게임 종료
      - 게임 시작 메뉴를 선택하면 한 칸 빈 채로 퍼즐이 랜덤하게 배치되어 출력된다.
      - 전체 그림 보기: 밑그림이 전체가 그려진다.
      - 게임 종료: 퍼즐 이동이 안된다.
  - **마우스 입력:**
    - **왼쪽 마우스를 누른채로 드래그 하면 빈 칸 주변의 칸 중 선택된 방향의 칸이 이동된다.**
      - 마우스를 아래방향으로 드래그하면 빈 칸의 위쪽의 칸이 아래로 이동한다.
      - 마우스를 오른쪽으로 드래그하면 빈 칸의 왼쪽의 칸이 오른쪽으로 이동한다.
    - **이동은 단번에 이동하지 않고 조금씩 미끄러지듯이 이동한다.**
  - **키보드 입력: 메뉴의 항목이 키보드로 실행될 수 있다.**
    - **S**: 게임 시작
    - **F**: 전체 그림 보기
    - **Q**: 게임 종료
    - **1**: 첫 번째 밑그림
    - **2**: 두 번째 밑그림
    - **3**: 그림 나누기 (3x3)
    - **4**: 그림 나누기 (4x4)
    - **5**: 그림 나누기 (5x5)



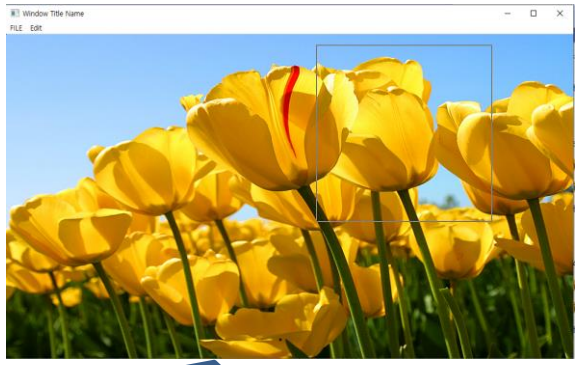


실습 5-3

- **부분 이미지를 돋보기 보기**
  - 2개 이미지를 사용할 수 있게 한다.
  - 화면에 디폴트로 설정된 이미지가 그려져 있다.
  - 왼쪽 마우스를 클릭하고 드래그하여 사각형(돋보기)을 그린다. (사각형은 고무줄 효과를 가진다)
  - **기본 키보드 명령**
    - **1:** 첫번째 이미지
    - **2:** 두번째 이미지
    - **3**-돋보기1: 돋보기 내의 그림이 조금씩 확대해서 보인다.
    - **4**-돋보기2: 돋보기 내의 그림이 조금씩 축소해서 보인다.
    - **0**-제자리: 돋보기 내의 그림이 원래의 크기로 보인다.
    - **c**-복사: 돋보기 내의 그림이 복사된다.
    - **p**-붙여 넣기: 돋보기 내의 그림이 랜덤한 위치에 같은 크기로 복사된다.
    - **F**-전체 붙여넣기: 돋보기 내의 그림이 화면 전체에 복사된다. 다시 누르면 원래 크기로 바뀐다.
  - **그 외의 키보드 명령**
    - **좌/우/상/하 화살표**: 돋보기의 위치가 좌/우/상/하로 이동된다. 붙여넣기한 곳의 그림은 새로운 돋보기 위치의 그림으로 그려진다.
    - **m/n**: 돋보기 사각형이 작아진다/커진다.
    - **h/v**: 붙여넣기한 곳의 그림의 방향이 좌↔우 / 위↔아래로 반전된다.
    - **x/X**: 붙여넣기한 곳의 그림이 왼쪽/오른쪽으로 조금씩 이동한다.
    - **y/Y**: 사각형의 y축 크기가 축소/확대한다.
    - **a/A**: 돋보기 사각형이 튕기기를 한다. 사각형 내의 그림이 튕기기에 따라 바뀐다. (벽돌깨기의 공처럼 네 가장자리에 대해 튕기기 진행)
    - **r/R**: 리셋한다 (돋보기 사각형이 없어지고 다시 돋보기 사각형을 입력받게 한다)
    - **q/Q**: 프로그램 종료
  - 사각형(돋보기)를 오른쪽 마우스로 클릭한채로 드래그하면 사각형의 위치가 바뀐다.

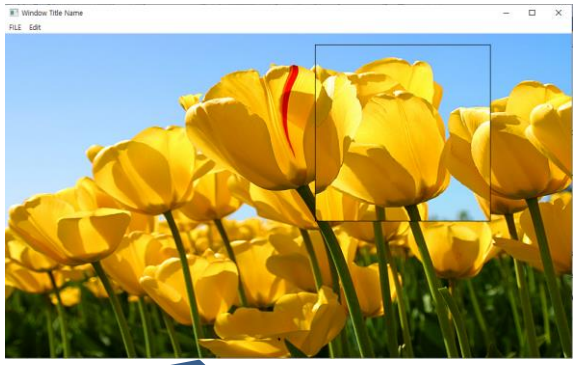
실습 5-3

1



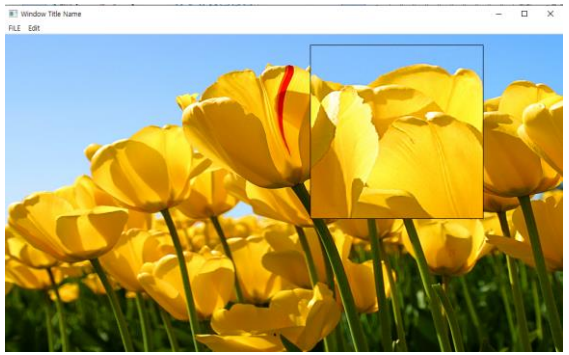
고무줄 효과가 있는 사각형을 그려 영역을 선택한다. (돋보기영역)

2



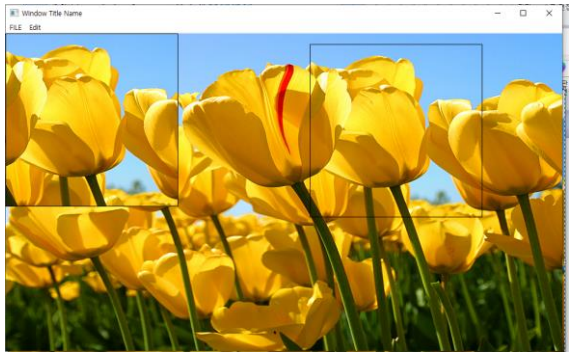
1: 돋보기 영역을 확대

3



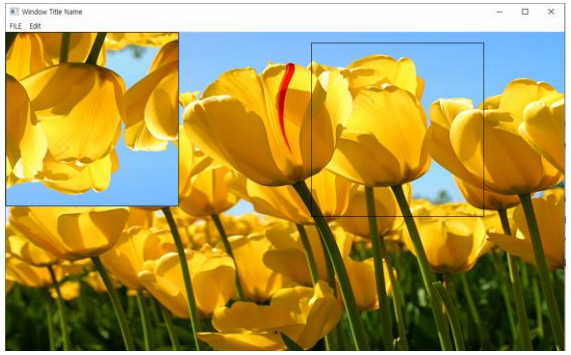
2: 돋보기 영역을 확대

4



c, p(복사, 붙여넣기): 돋보기 영역을 복사하여 붙여넣기를 했다.

5



v: 붙여넣기한 곳의 이미지가 상하 반전하였다.

6

h: 붙여넣기한 곳의 이미지가 좌우 반전하였다.

7

n: 돋보기 영역이 커졌다. 붙여넣기 한 곳의 영역도 같이 커진다.

8

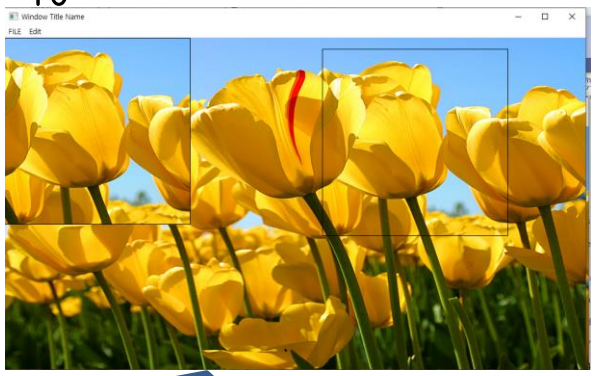
m: 돋보기 영역이 작아졌다. 붙여넣기한 곳의 영역도 같이 작아진다.

실습 5-3

9

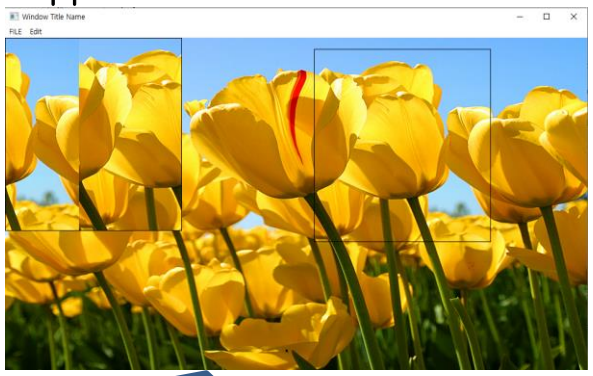
f: 돋보기 영역의 그림이 전체 배경으로 바뀐다.

10



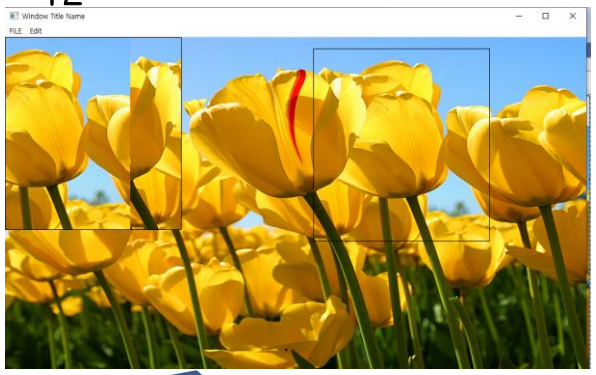
F: 다시 f를 눌러 배경화면을 원래 그림으로 바꾼다.

11



X: 붙여넣기한 곳의 그림이 왼쪽으로 이동한다.

12



x: 붙여넣기한 곳의 그림이 오른쪽으로 이동한다.

13

아래쪽 화살표: 돋보기가 아래쪽으로 이동한다. 붙여넣기한 곳의 그림이 새로운 돋보기 위치에 따라 바뀐다.

14

오른쪽 화살표: 돋보기가 오른쪽으로 이동한다. 붙여넣기한 곳의 그림이 새로운 돋보기 위치에 따라 바뀐다.

다음 시간에는

- 애니메이션 추가 실습 진행