제 5장 비트맵과 애니메이션

2019년 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

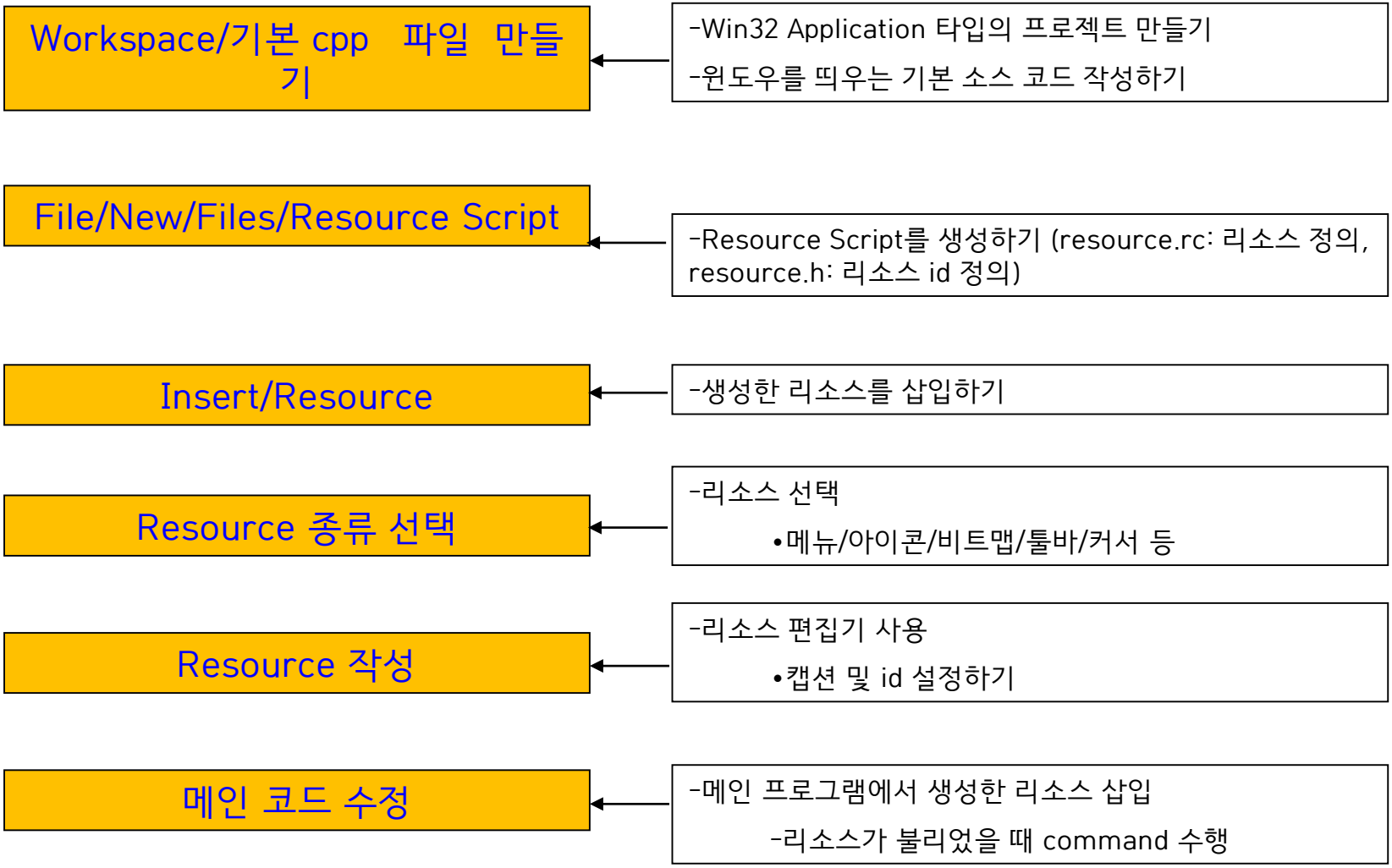
- **내용**
  - 비트맵
  - 애니메이션 만들기
  - 더블 버퍼링 사용하기

리소스 개요

- **리소스(resource)**
  - 메뉴, 아이콘, 커서, 다이얼로그, 액셀러레이터, 비트맵, 문자열, 버전정보 등 사용자 인터페이스를 구성하는 자원들로 읽기 전용 정적 데이터를 말한다.
  - 리소스는 프로그램 실행 중 변경되지 않는 정적 데이터
  - C/C++과 같은 언어로 관리하지 않고 리소스 스크립트(Resource Script; .rc)파일로 관리한다.

- **윈도우 프로그램의 큰 특징 중 하나가 소스코드와 리소스가 분리**
  - DOS기반 프로그래밍에서는 리소스를 정의할 때 복잡한 배열 등을 만들어 사용하거나 외부 파일로 만들어 둔 후 프로그램 실행 시에 읽어 들이도록 했다.
  - 그러나 윈도우 프로그래밍에서는 이런 데이터들을 리소스로 만들어 놓고 소스코드와 별도로 컴파일하며, **링크 시 최종 실행파일에 합쳐진다.**

리소스 개요


Workspace/기본 cpp 파일 만들기
-Win32 Application 타입의 프로젝트 만들기
-윈도우를 띄우는 기본 소스 코드 작성하기
File/New/Files/Resource Script
-Resource Script를 생성하기 (resource.rc: 리소스 정의, resource.h: 리소스 id 정의)
Insert/Resource
-생성한 리소스를 삽입하기
Resource 종류 선택
-리소스 선택
•메뉴/아이콘/비트맵/툴바/커서 등
Resource 작성
-리소스 편집기 사용
•캡션 및 id 설정하기
메인 코드 수정
-메인 프로그램에서 생성한 리소스 삽입
-리소스가 불리었을 때 command 수행

2. 비트맵

- **컴퓨터 이미지(image)**
  - 전자적인 형태로 만들어지거나 복사되고, 저장된 그림이다.
  - 2차원 그래픽스 종류: 벡터 그래픽스(vector graphics), 래스터 그래픽스(raster graphics)

    - **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

    - **래스터 그래픽스 (raster graphics)**
      - 비트맵 이미지로 그래픽 객체들을 구성하고 있는 픽셀 (PIXEL, Picture Element) 값들을 그대로 저장하는 방식
      - 그림의 크기를 확대하면 화질이 떨어진다.
      - 그림의 복잡도에 관계없이 그림파일의 크기는 일정하다.
      - 포토샵, 페인터 같은 소프트웨어에서 사용

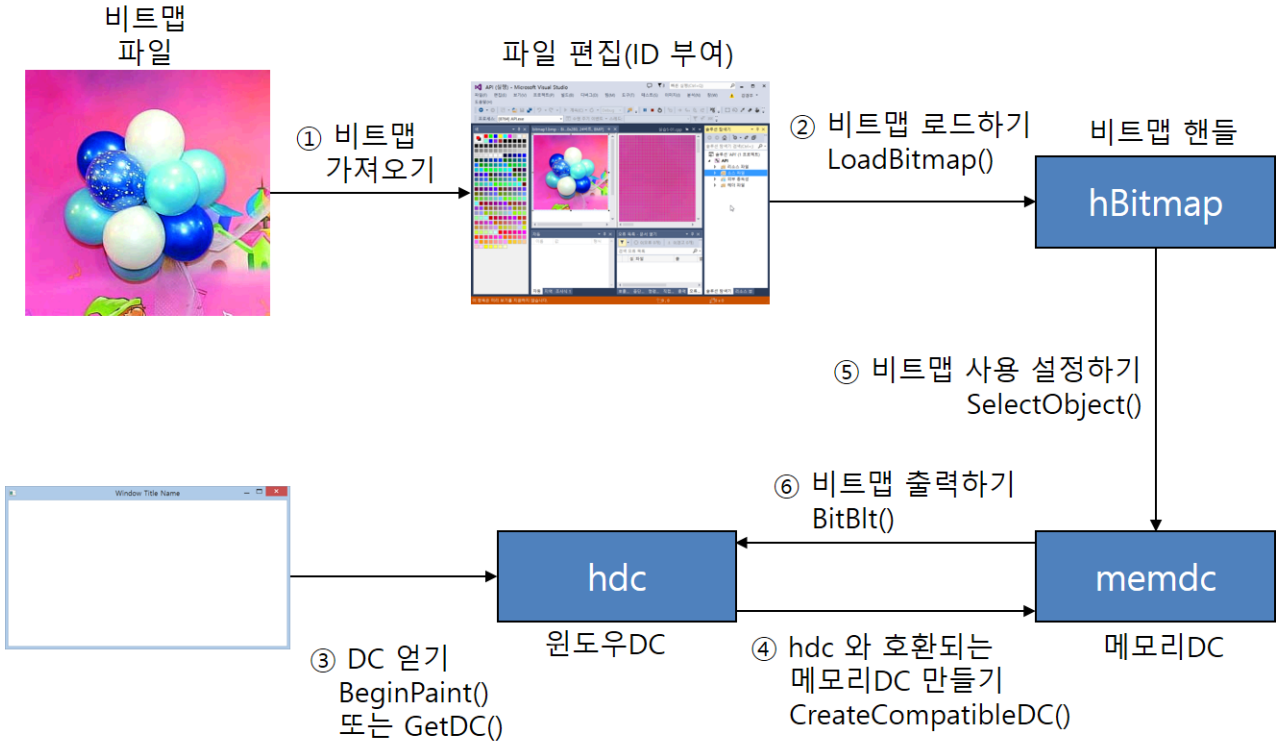
비트맵

- 이미지 종류

이미지 포맷	특징
BMP	윈도우의 표준 그래픽 파일. 거의 모든 프로그램에서 지원하기 때문에 사용이 간편 압축을 하지 않기 때문에 용량이 크다 윈도우의 배경파일 등에 BMP파일이 사용
GIF	그래픽 파일의 하나로 256색 이하의 그래픽에 가장 최적화 웹사이트의 아이콘 등에 많이 사용 바탕화면이 투명한 이미지 등에도 사용되며 간단한 애니메이션 등에 이용
JPG	가장 많이 사용되는 이미지 파일 압축률이 높아 적은 용량으로 고화질의 사진을 저장할 수 있다
TIFF	응용프로그램 간 그래픽 데이터 교환을 목적으로 개발된 형식 전자 출판 등에 이용,
PNG	인터넷 및 온라인 서비스에 사용되는 GIF 포맷 대체 이미지 투명 효과, 압축률이 우수
	그 외, PSD 파일, PDF 파일, RAW 파일, AI 파일 등

비트맵

- 새로운 비트맵 이미지 만들기
- 이미지 파일을 비트맵형태로 읽어오기
  - 자체적으로 만든 이미지 활용 또는
  - 이미 만들어진 이미지 활용
- 비트맵 출력하기



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)**

BITMAPFILEHEADER (비트맵 파일에 대한 정보)
BITMAPINFOHEADER (비트맵 자체에 대한 정보)
RGBQUAD 배열 (색상 테이블)
color/index 배열 (픽셀 데이터)

```
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;
```

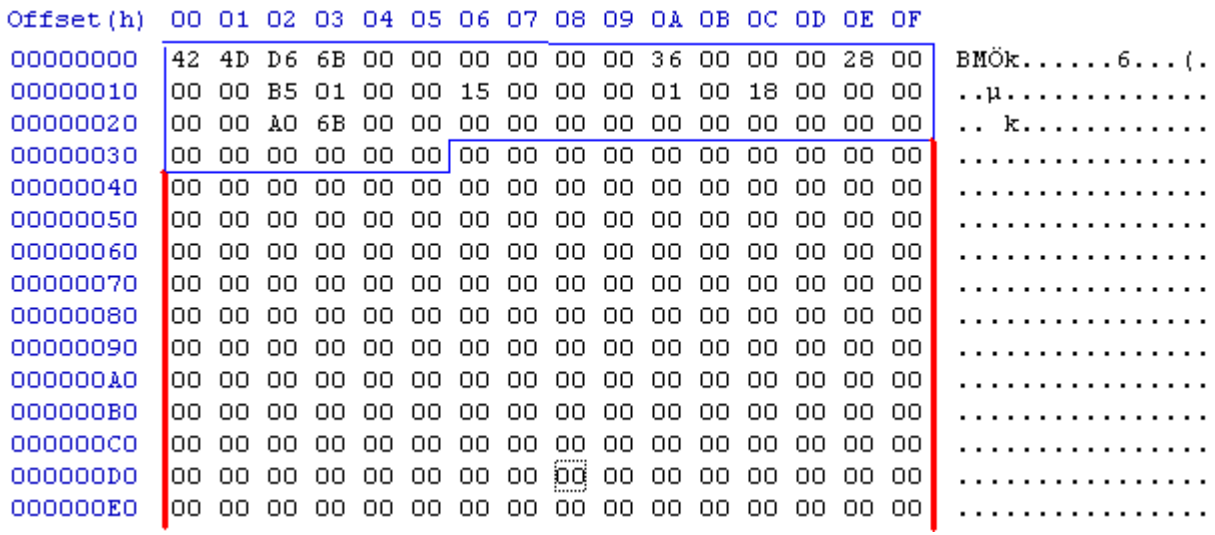
```
typedef struct tag BITMAPINFOHEADER{
    DWORD   biSize;                 // BITMAPINFOHEADER 구조체 크기
    LONG    biWidth;                // 비트맵의 가로 픽셀 수
    LONG    biHeight;               // 비트맵의 세로 픽셀 수
    WORD    biPlanes;               // 장치에 있는 색상 면의 개수 (반드시 1)
    WORD    biBitCount;             // 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 : 비트맵은 사용할 수 있는 모든 색상을
                                    // 사용,그 외 : RGBQUAD구조체배열의 크기는
                                    // 이 멤버의 크기만큼 만들어짐)
    DWORD   biClrImportant;         // 비트맵을 출력하는데 필수적인 색상수
                                    // (0 : 모든 색상이 사용되어야 함)
} BITMAPINFOHEADER,   *PBITMAPINFOHEADER;
```

```
typedef struct  tag RGBQUAD {
    BYTE    rgbBlue;                // 파란색
    BYTE    rgbGreen;               // 초록색
    BYTE    rgbRed;                 // 빨강색
    BYTE    rgbReserved;            // 예약된 값: 0
} RGBQUAD;
```

비트맵

- 비트맵 읽기
  - 리소스 에디터에 의해서 만들어지는 비트맵 리소스들은 DIB 비트맵
  - 비트맵 이미지는 LoadBitmap() 함수로 읽는다.
    - 이 함수로 읽은 비트맵은 DDB로 변경된다.

  - 이미지 파일을 비트맵 형태로 읽어오는 방법
    - 자체적으로 만든 이미지 활용
    - 이미 만들어진 이미지 활용



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	.....

리소스 만들기

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

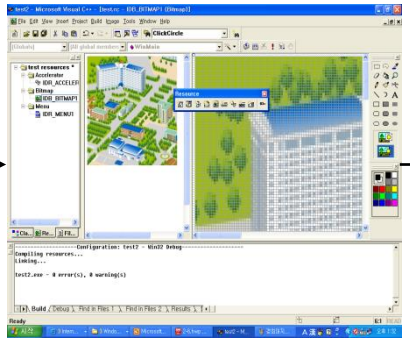
- **이미지 만들기 및 불러오기**
  - 새로 만들기: 리소스 도구상자에서 새로 만들기 이용
  - 불러오기: 리소스 도구상자에서 가져오기 이용

리소스 만들기: 비트맵 출력하기

비트맵 파일

비트맵
가져오기

리소스에서 파일 편집(ID 부여)



LoadBitmap()

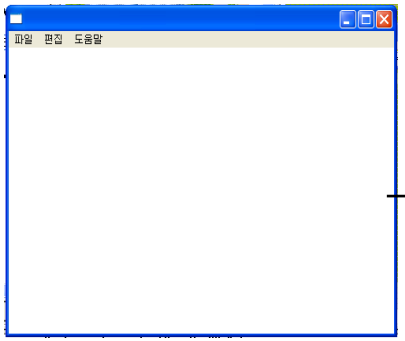
비트맵 로드

비트맵 핸들

hBitmap

비트맵 사용선언

SelectObject()



DC 얻기
BeginPaint()
또는 GetDC()

hdc

화면

BitBlt()

비트맵 화면출력

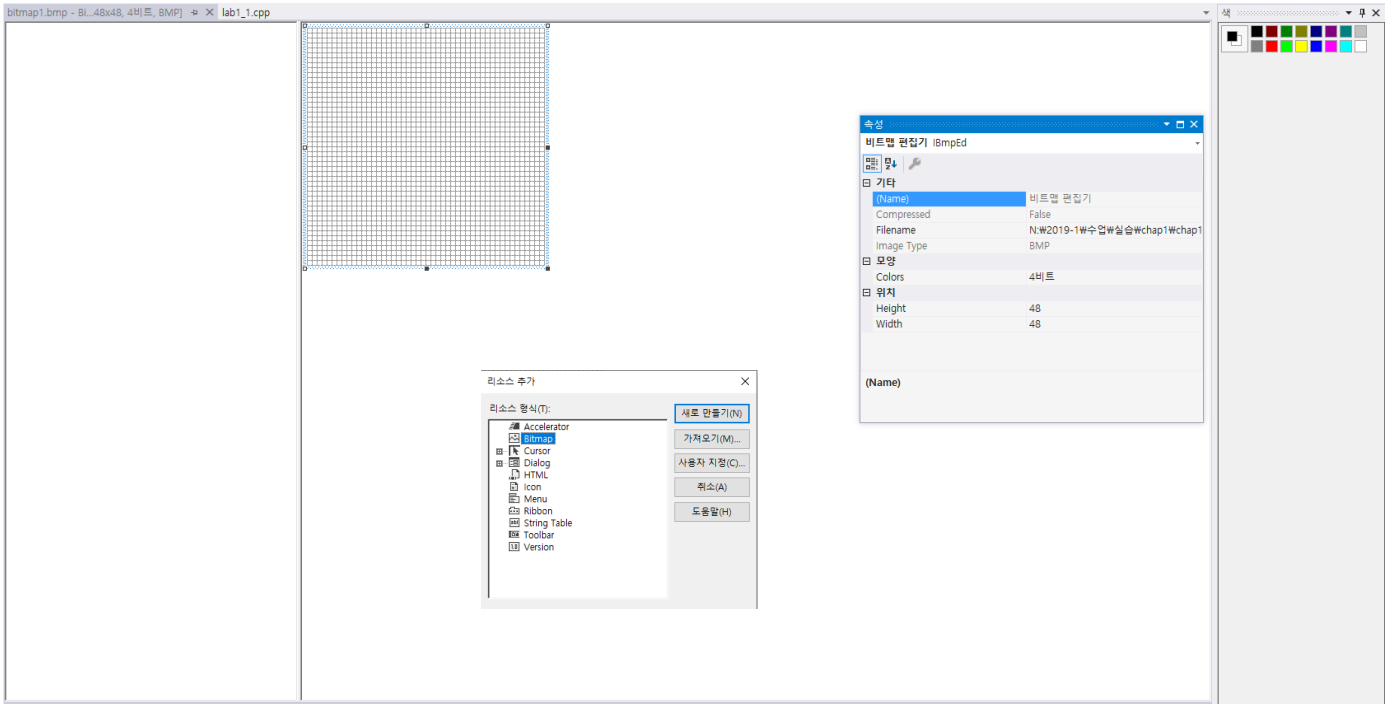
memdc

메모리DC 만들기

CreateCompatibleDC()

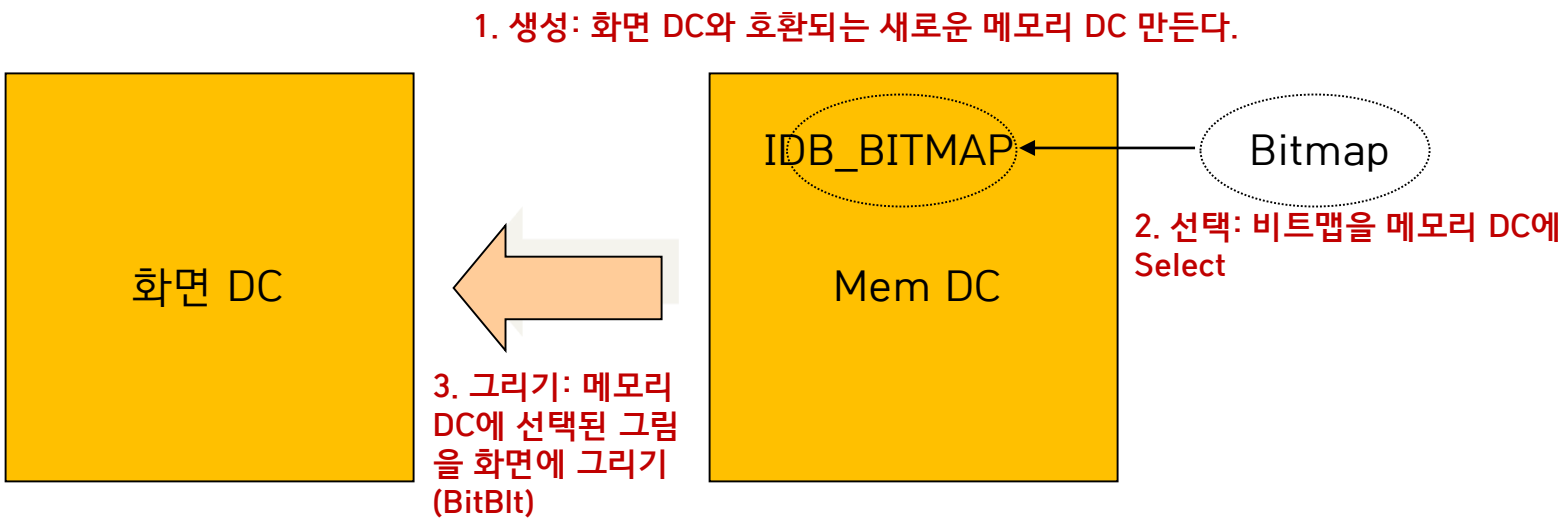
리소스 만들기: 비트맵 만들기

- Visual Studio 2017 환경



- 비트맵 속성
  - ID: 비트맵에 대한 식별자, 수정가능
  - Width, Height: 비트맵의 크기, 수정가능
  - Colors: 사용하는 컬러수로 32비트까지 지원 가능
  - File name: 비트맵파일을 저장할 파일이름

비트맵 읽기




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

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

            hBitmap = LoadBitmap (hInstance, MAKEINTRESOURCE(IDB_BITMAP));   // 비트맵 로드
            hMemDC = CreateCompatibleDC (hdc);                    // 주어진 DC와 호환되는 DC를 생성
            SelectObject (hMemDC, hBitmap);                       // 새로 만든 DC에 그림을 선택한다
            BitBlt (hdc, 0, 0, 320, 320, hMemDC, 0, 0, SRCCOPY);  // DC간 블록 전송을 수행한다
            ...
}
```

비트맵 읽기

- 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

HDC CreateCompatibleDC (HDC hdc);

- 주어진 DC와 호환되는 메모리 DC를 생성해 준다.
- HDC hdc: 주어진 DC

비트맵 읽기

- **비트맵 읽고 선택**
  - 비트맵을 읽어올 때: LoadBitmap 함수를 사용

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

- 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

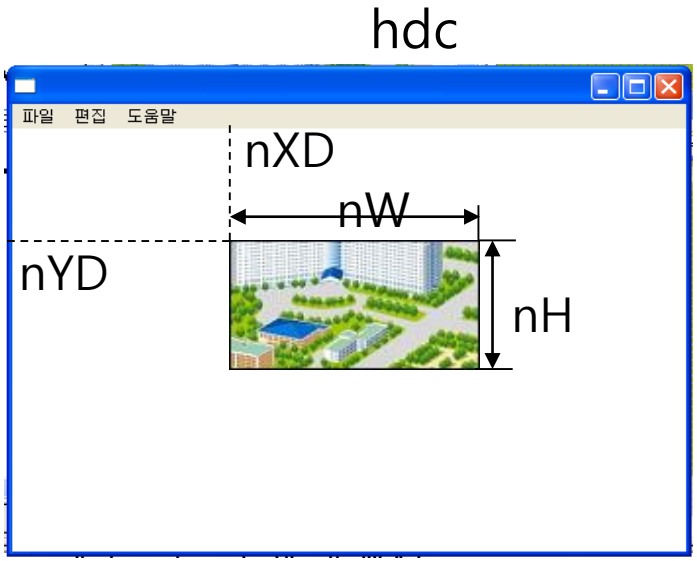
비트맵 출력: BitBlt() → 1 : 1 Copy

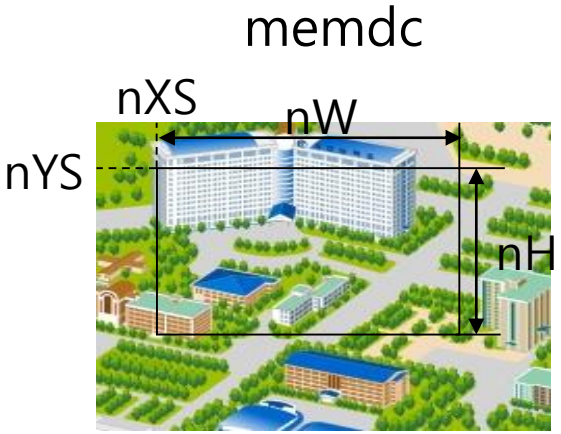
- **BitBlt 함수**

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, nYDest: 복사 대상의 x, y 좌표 값
  - nW, nHeight: 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 Copy





- **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));
                    break;

                case WM_PAINT:
                    hdc = BeginPain t(hwnd, &ps);
                    memdc=CreateCompatibleDC (hdc);
                    SelectObject (memdc, hBitmap);
                    BitBlt (hdc, 0, 0, 332, 240, memdc, 0, 0, SRCCOPY);
                              // memdc 에 있는 그림에서 (0, 0) 위치로부터 (0, 0)위치에 (322, 240) 크기로 그려라
                              // SRCCOPY : 바탕색을 무시하고 소스값을 그대로 그리기
                    DeleteDC (memdc);
                    EndPaint (hwnd, &ps);
                    break;
        }
        return DefWindowProc (hWnd, iMessage, wParam, lParam);
}
```

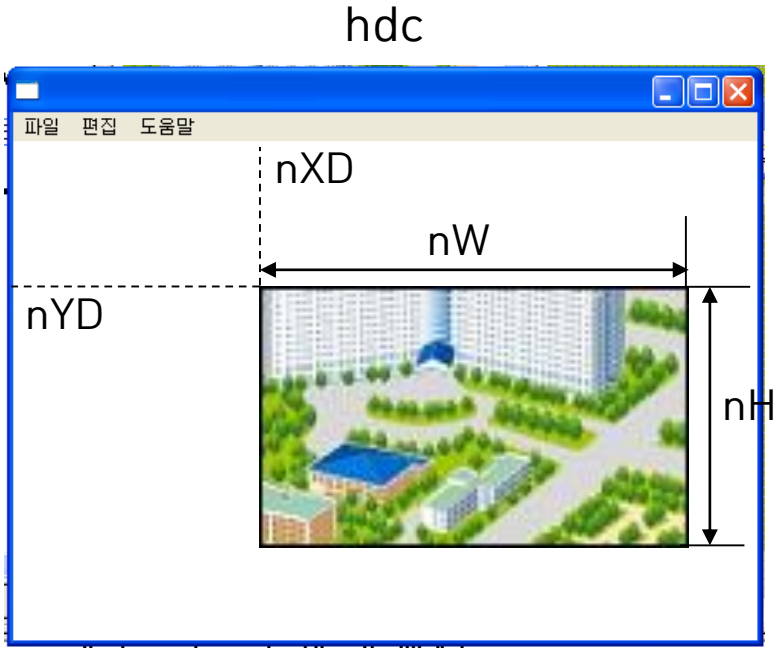
비트맵 출력: StretchBlt() → 확대, 축소 Copy

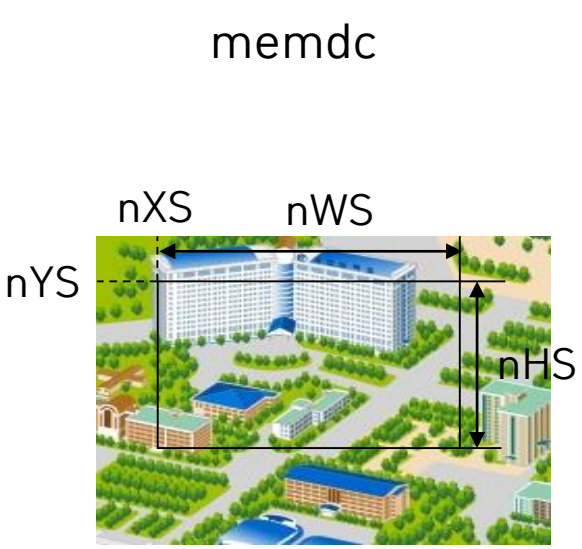
- **StretchBlt 함수**

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 Copy





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, 100, 100, memdc, 0, 0, 320, 240, SRCCOPY);
                                // 메모리 DC에 있는 그림에서 (0, 0)위치에서 (320, 240) 크기의 그림을
                                // 화면의 (100, 0)위치에 (100, 100) 크기로 그려라
                    DeleteDC (memdc);
                    EndPaint (hwnd, &ps);
                    break;
        }
        return DefWindowProc (hWnd, iMessage, wParam, lParam);
}
```

GetObject (): 그림 크기 알아내기

- 그림 크기 알아내기

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

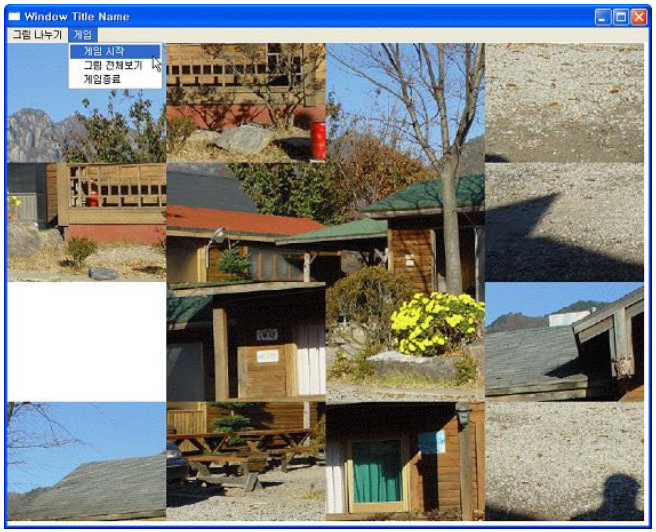
- 객체의 정보 알아오기
- HGDIOBJ hgdiobj: GDI 오브젝트 핸들
- int cbBuffer: 오브젝트 버퍼의 크기에 관한 정보
- LPVOID lpvObject: 오브젝트 정보 버퍼를 가리키는 포인터

- 사용 예)

```
BITMAP bmp;
int mWidth, mHeight;

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

실습 5-1

- **조각 퍼즐 맞추기**
  - 1개 이상의 밑 그림을 사용한다. 키보드를 이용하여 그림을 선택하게 한다.
  - 3, 4, 5를 선택하면 그림을 숫자에 맞게 분할하여 랜덤하게 화면에 배치한다.
  - S를 선택하면 아래와 같이 그림 하나가 비고 게임이 시작된다.
  - 마우스를 이용하여 주변의 조각을 클릭하여 조각 그림을 이동한다.
    - **이동은 단번에 이동하지 않고 조금씩 이동한다.**
  - 게임 중에 F를 선택하면 완성된 형태를 보여준다.
  - 그림 맞추기가 완성되면 맞추기 완성 메시지를 출력한다.



- 키보드 명령어:
  - S: 게임시작
  - F: 그림전체보기
  - Q: 종료
  - 1: 첫 번째 밑그림
  - 2: 두 번째 밑그림
  - 3: 그림 나누기 3*3
  - 4: 그림 나누기 4:4
  - 5: 그림 나누기 5*5

실습 5-2

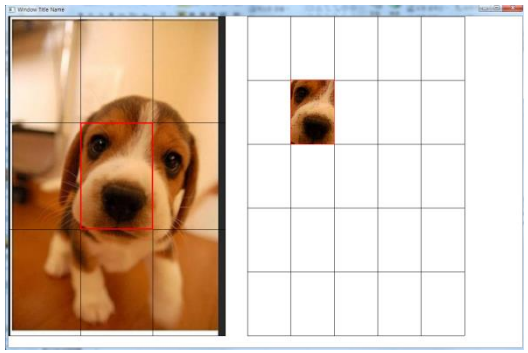
- **이미지 한 칸씩 복사하기**
  - 이미지: 한쪽 면에 이미지를 그리고 가로와 세로를 각각 3등분한다.
  - 그림판: 다른쪽 면에 복사할 그림판을 만들고 가로와 세로를 디폴트로 각각 3등분 한다.

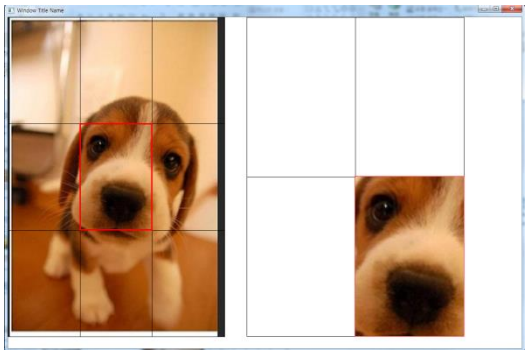
  - 그림판의 등분은 키보드 명령으로 바꿀 수 있다.
    - 1/2/3/4/5: 그림판 가로/세로 각각 1칸/2칸/3칸/4칸/5칸
  - 그림판의 빈 칸에 이미지를 놓을 수 있다.

  - 왼쪽 마우스: 이미지의 한 칸을 왼쪽 마우스 버튼을 클릭하고 드래그 하여 그림을 끌어서 그림판 칸으로 이동하면 이미지가 마우스가 놓여지는 칸에 복사된다.
    - 오른쪽 그림판에서 왼쪽 마우스 버튼을 이미지의 네 귀퉁이에 클릭 후 드래그 하면 이미지의 크기가 커진다. (마우스 버튼을 놓으면 칸에 맞춰진 상태로 확대된다.)

  - 오른쪽 마우스: 그림판에 복사된 그림은 그림판에서 오른쪽 마우스 버튼의 선택 드래그를 이용하여 위치를 바꿀 수 있다. 즉, 그림판에 복사된 그림을 오른쪽 마우스 버튼을 눌러 선택하고 드래그 하면 이미지가 이동하고, 마우스를 놓으면 마우스가 놓여진 칸에 맞춰서 그려진다.

이미지 **그림판**

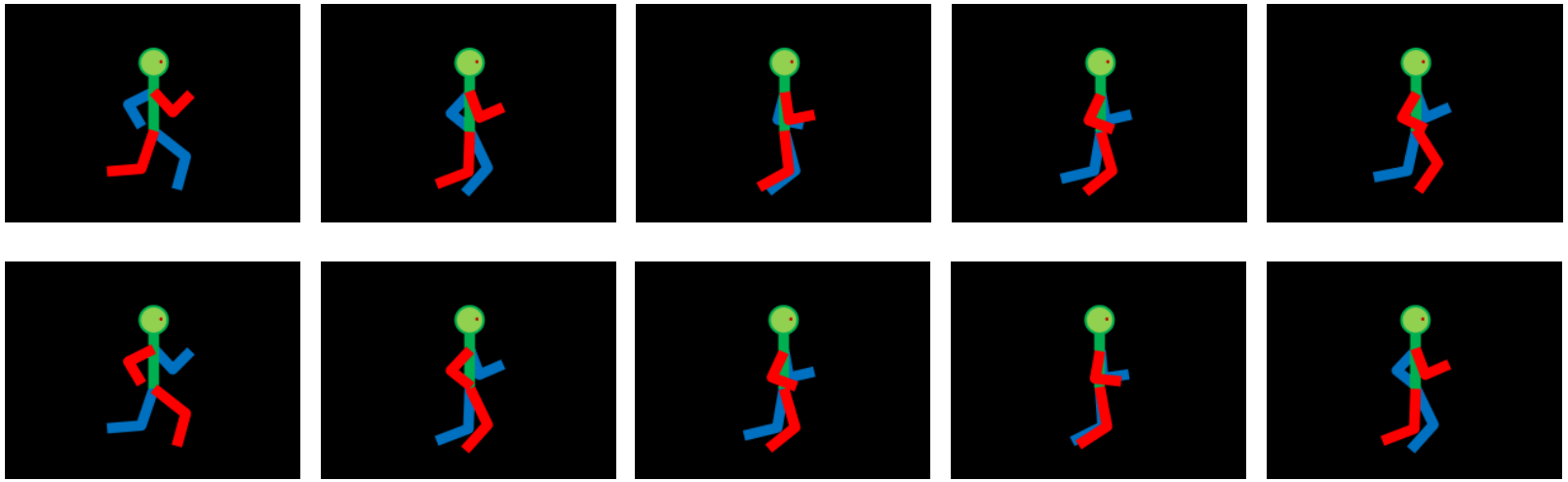




실습 5-3

비트맵 애니메이션

- 애니메이션
  - 각 시점에 다른 그림을 그려서 움직이는 효과를 얻는다. 프레임(Frame)
  - 애니메이션 동작은 **타이머로** 처리한다.
  - 매 타이머의 주기에 각 프레임을 표시하여, 각 동작에 하나의 프레임만을 보여준다.
  - 한 프레임씩 이동하면서 필요한 부분을 잘라내어 번갈아 표시한다. 오프셋 개념을 이용한다



비트맵 애니메이션

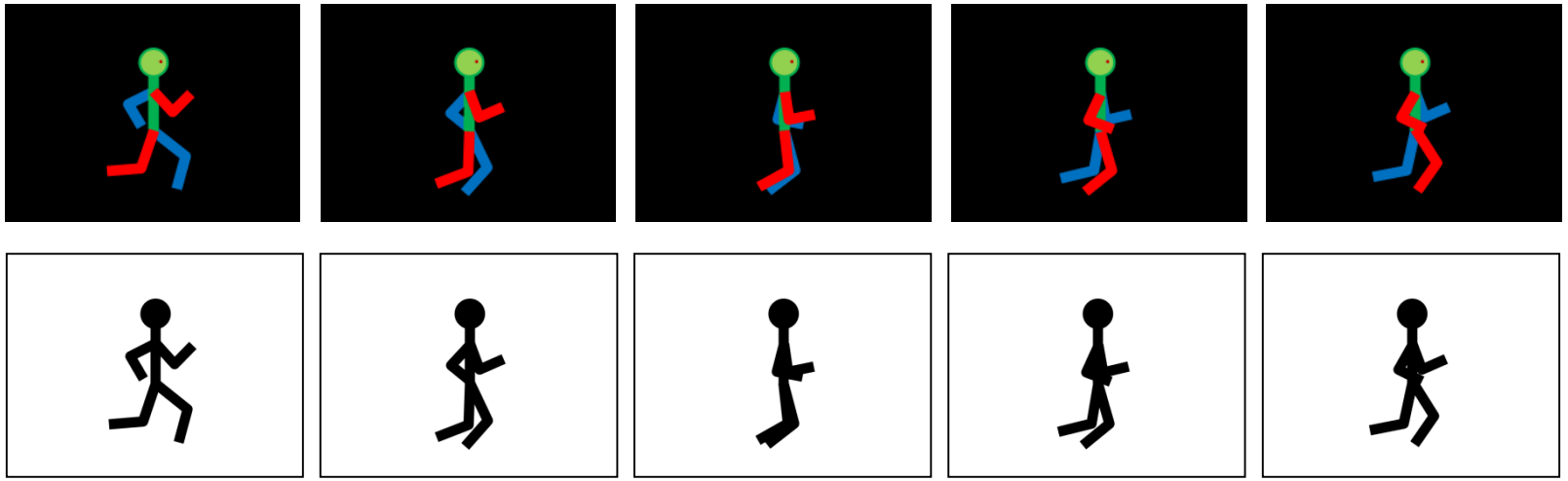
```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc;
        PAINTSTRUCT ps;
        static int xPos;

        switch (iMsg)
        {
        case WM_CREATE:
                xPos = 0;
                SetTimer(hwnd, 1, 100, NULL);
                break;
        case WM_TIMER:
                xPos += 10;
                if (xPos > 800)
                        xPos = 0;
                InvalidateRect (hwnd, NULL, true);
                return 0;
        case WM_PAINT:
                hdc = BeginPaint(hwnd, &ps);
                Animation (xPos, 300, hdc);
                EndPaint(hwnd, &ps);
                break;
        }
        return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

비트맵 애니메이션

```
// 10개의 이미지를 사용하여 애니메이션 구현
void Animation(int xPos, int yPos, HDC hdc)
{
	HDC    memdc;
	HBITMAP RunBit[10], hBit, oldBit;
	static int count;
	int i;

	count++;
	count = count % 10;
	RunBit[0]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R1));
	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));

	memdc = CreateCompatibleDC (hdc);
	hBit = LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_BACK));

	oldBit = (HBITMAP) SelectObject (memdc, hBit);
	BitBlt (hdc, 0, 0, 800, 600, memdc, 0, 0, SRCCOPY);

	(HBITMAP) SelectObject (memdc, RunBit[count]);
	BitBlt (hdc, xPos, yPos, 64, 64, memdc, 0, 0, SRCCOPY);
	SelectObject (memdc, oldBit);

	for (i = 0; i < 10; i++)
		DeleteObject (RunBit[i]);
	DeleteDC (memdc);
	DeleteObject (hBit);
}
```

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 마스크

- 처리 방법:
  - 각 프레임의 동작마다 마스크 처리와 소스 프레임의 그림을 각각 두번씩 씌워주어야 한다.
    1. 소스의 원하는 부분을 흑백으로 처리한 패턴을 배경 그림과 AND 연산 → 배경 이미지에 흑색 그림만이 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

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

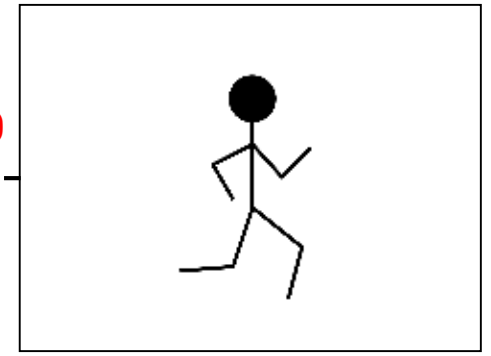
       BitBlt (hdc, x, y, size_x, size_y, hBitmapFrontDC, mem_x, mem_y, SRCPAINT);
       - SRCPAINT: 소스와 대상의 OR 연산값으로 칠한다.
         - » 출력하고자하는 이미지와 배경이미지의 OR 연산

비트맵 마스크

- 마스크 그리기 (AND 연산)

흰색(255,255,255)

AND



컬러(X, X, X)

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

- 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

- 결과

비트맵 마스크

```
void Animation (int xPos, int yPos, HDC hdc)
{
        HDC         memdc;
        HBITMAP     RunBit[5], hBit, Mask[5];
        static int  count;
        int         i;
        count++;
        // 애니메이션 이미지
        RunBit[0]= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_R1));
         . . .
        RunBit[4]= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_R5));
        // 마스크 이미지
        Mask[0] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_M1));
        . . . . . .
        Mask[4] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_M5));

        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);

        for (i = 0; i < 10; i++)
        {
            DeleteObject(RunBit[i]);
            DeleteObject(Mask[i]);
        }
        DeleteDC(memdc);
        DeleteObject(hBit);
}
```

투명 비트맵 처리

- 비트맵의 일부를 투명하게 처리하여 투명색 부분은 출력에서 제외한다.

```
BOOL TransparentBlt ( hdcDest, int nXOriginDest, int nYOriginDest, int nWidthDest,
        int hHeightDest, 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.lib를 링크한다.
  - 속성 → 링커 → 명령줄에서 라이브러리 추가

```
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) );      // 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	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)

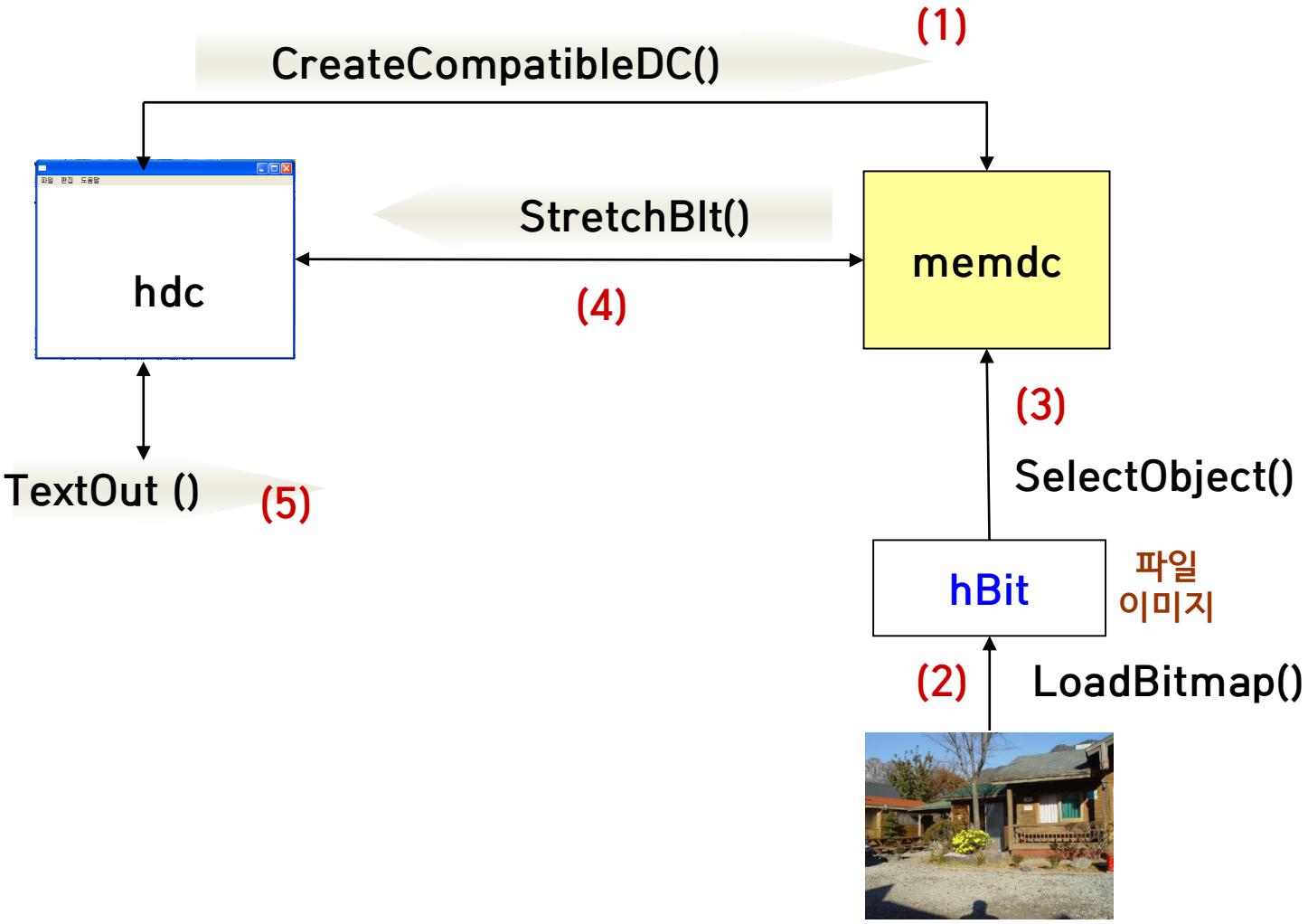
3. 더블 버퍼링

- **비트맵 이미지 여러 개를 이용하여 동영상을 나타낼 때**
  - 이미지를 순서대로 화면 디바이스 컨텍스트에 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



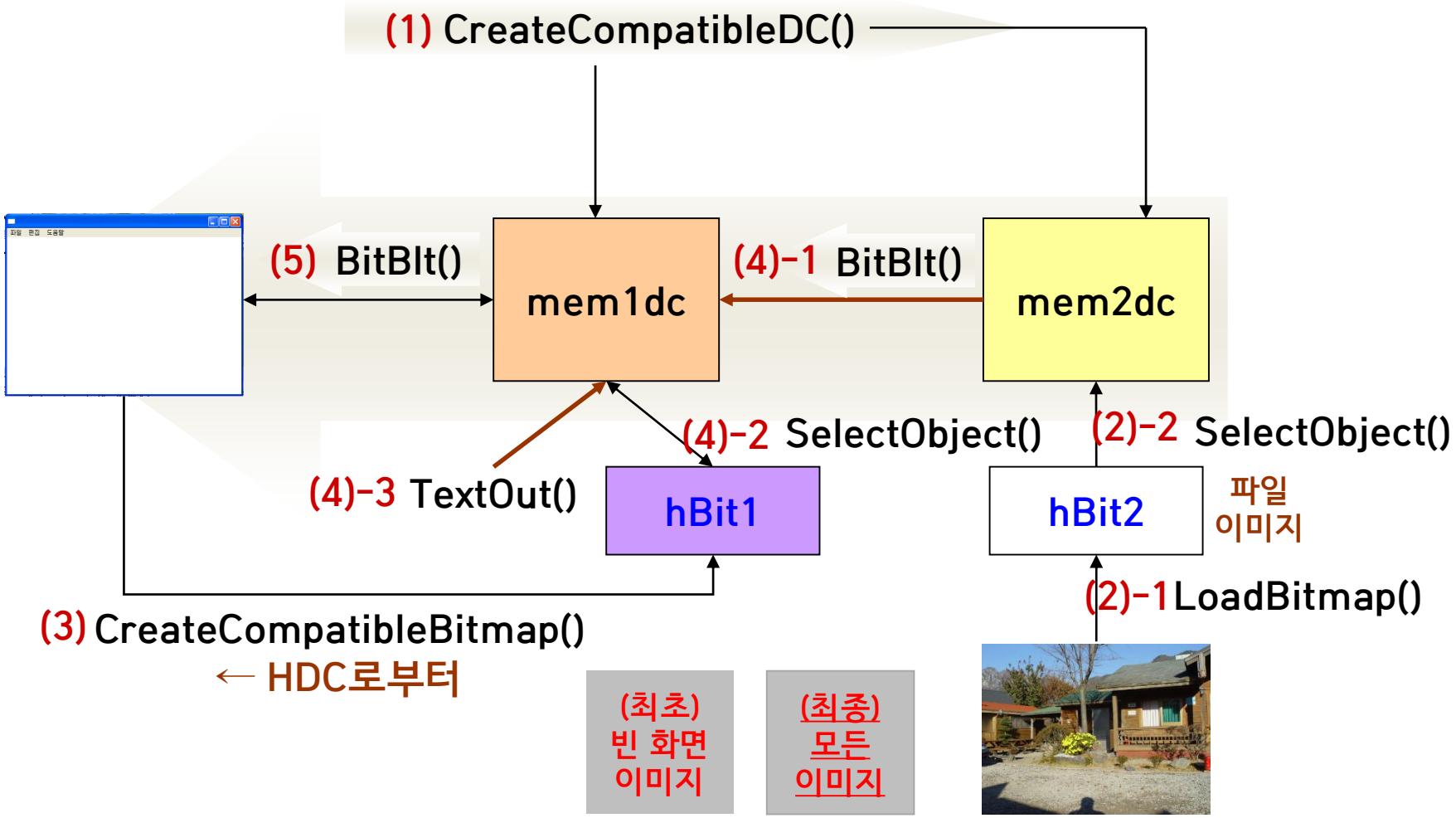
배경화면 위로 움직이는 글

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

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

        HDC hdc, memdc;
        Static HDC hdc, mem1dc, mem2dc;
        static HBITMAP hBit1, hBit2, oldBit1, oldBit2;
        …
        char word[] = "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);

        // hdc에서 mem1dc를 호환되도록 만들어준다.
        mem1dc = CreateCompatibleDC (hdc);

        //  mem1dc에서 mem2dc를 호환이 되도록 만들어준다.
        mem2dc = CreateCompatibleDC (mem1dc);

        // mem2dc의 비트맵을 mem1dc에 옮기고, mem1dc를 hdc로 옮기려고 함
        oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);          // mem1dc에는 hBit1
        oldBit2 = (HBITMAP) SelectObject (mem2dc, hBit2);          // mem2dc에는 hBit2

        // 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);
        break;
```

배경화면 위로 움직이는 글 (더블버퍼링)

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

        mem1dc = CreateCompatibleDC (hdc);
        // hBit1에는 배경과 텍스트가 출력된 비트맵이 저장, mem1dc에 설정
        oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);

        // mem1dc에 있는 내용을 hdc에 뿌려준다.
        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)

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

- hdc와 호환되는 비트맵을 생성하여 반환하는 함수
- 생성된 비트맵은 hdc와 호환되는 어떤 메모리 DC에서도 선택되어질 수 있다.
- HDC hdc: DC 핸들
- int nWidth/nHeight: 작성하는 비트맵의 가로/세로 사이즈

HDC CreateCompatibleDC (HDC hdc);

- 주어진 DC와 호환되는 메모리 DC를 생성해 준다.
- HDC hdc: 주어진 DC

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
  - 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.

CImage 클래스 사용하기

- **CImage는 그림 관련 클래스**
  - 사용 가능한 public method들
    - Create (int nWidth, int nHeight, int nBPP, 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)

```
#include <atlImage.h>
#define CLIENT_WIDTH 400            // 클라이언트 너비
#define CLIENT_HEIGHT 400           // 클라이언트 높이

CImage g_cimgTest;
CImage g_cimgBackBuff;
Cimage g_cimgExplosion;

UINTg_nSpriteX;                     // 스프라이트 가로
UINTg_nSpriteY;                     // 스프라이트 세로
UINTg_nSpriteCount;                 // 스프라이트 전체 인덱스
UINTg_nSpriteCurrent;               // 현재 스프라이트 인덱스

void DrawSprite(HDC hDC, POINT ptStart = POINT { 0, 0 })
{
    UINT nSpriteWidth  = g_cimgExplosion.GetWidth()  / g_nSpriteX;
    UINT nSpriteHeight = g_cimgExplosion.GetHeight() / g_nSpriteY;

    UINT xCoord = g_nSpriteCurrent % g_nSpriteX;
    UINT yCoord = g_nSpriteCurrent / g_nSpriteX;

    g_cimgExplosion.Draw (hDC, ptStart.x, ptStart.y, nSpriteWidth, nSpriteHeight.
            xCoord * nSpriteWidth, yCoord * nSpriteHeight, nSpriteWidth,
    nSpriteHeight );
}

void OnDraw(HWND hWnd)
{
    RECT rcClient;
    GetClientRect (hWnd, &rcClient);

    PAINTSTRUCT ps;
    HDC hdc = BeginPaint (hWnd, &ps);
    HDC memDC = g_cimgBackBuff.GetDC ();

    // 초기화
    FillRect (memDC, &rcClient, (HBRUSH)(GetStockObject(WHITE_BRUSH)));

    // 테스트 이미지 그리기 : 50, 50 위치에 100, 100 사이즈로
    g_cimgTest.Draw (memDC, 50, 50, 100, 100);

    // 폭발 스프라이트 그리기
    DrawSprite (memDC, POINT {100, 0});

    g_cimgBackBuff.Draw (hdc, 0, 0);
    g_cimgBackBuff.ReleaseDC ();
    EndPaint (hWnd, &ps);
}
```

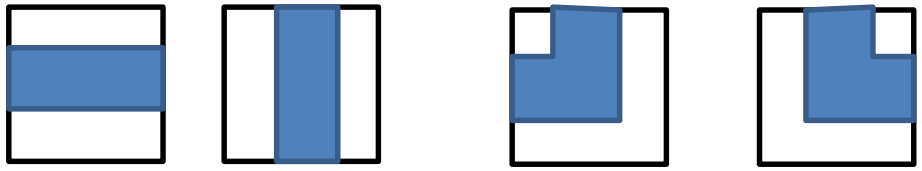
```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                                  LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
        g_cimgTest.Load (_T("Test.png")); // 테스트 이미지 로드
        g_cimgExplosion.Load (_T("explosion_01.png")); // 폭발 스프라이트 로드

        // 폭발 스프라이트 그림을 보고 직접 인자 설정.
        g_nSpriteX = 9;
        g_nSpriteY = 9;
        g_nSpriteCount = 66;
        g_nSpriteCurrent = 0;

        // 이미지 로드 성공여부 판단
        if (g_cimgTest.IsNull() || g_cimgExplosion.IsNull())
        {
            MessageBox(hWnd, _T("Image Load Fail!") , _T("cimage") , MB_OK);
        }

        // 백 버퍼 생성
        g_cimgBackBuff.Create (CLIENT_WIDTH, CLIENT_HEIGHT, 24, 0);

        SetTimer (hWnd, 0, 60, NULL);
        break;

    case WM_TIMER:
        (++g_nSpriteCurrent) %= g_nSpriteCount;
        InvalidateRect (hWnd, NULL, false);
    break;

    case WM_PAINT:
        OnDraw (hWnd);
    break;

    case WM_DESTROY:
        g_cimgBackBuff.Destroy ();
        PostQuitMessage (0);
    break;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);

}
```

실습 5-4

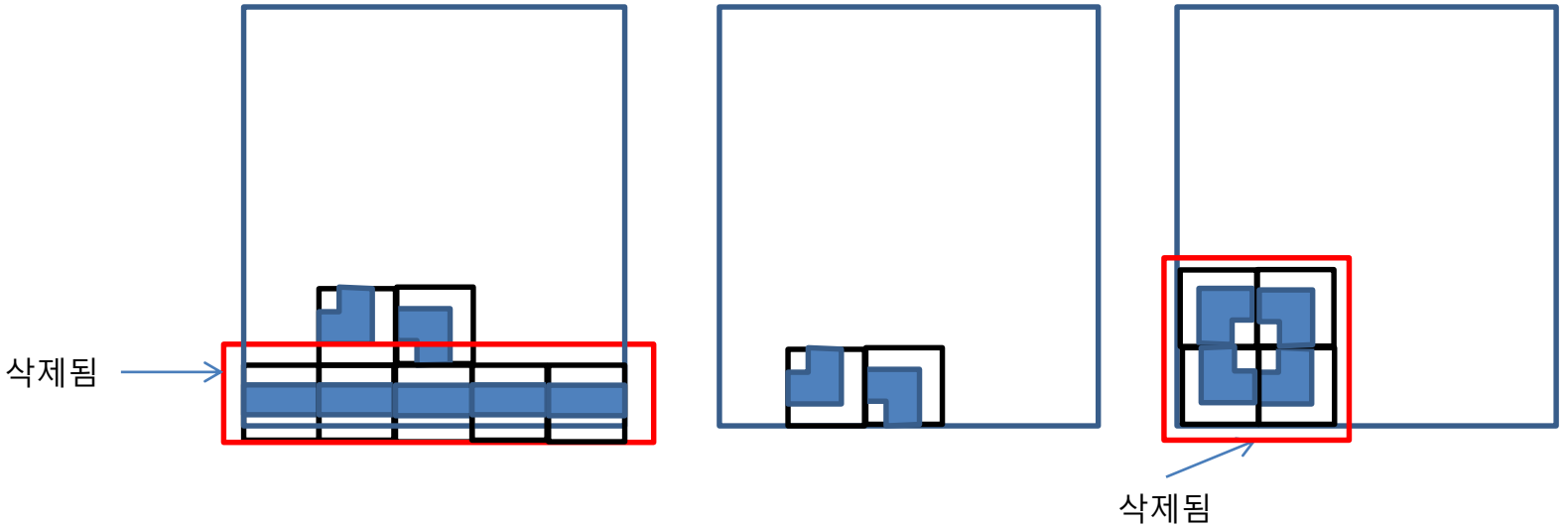
- **이동하며 마우스에 반응하는 이미지 그리기**
  - 배경 이미지를 출력한다.
  - 캐릭터 스프라이트가 애니메이션 되고 특정 방향으로 이동하고 있고, 키보드를 이용하여 스프라이트가 방향을 바꾼다.
    - 좌/우/상/하 키: 스프라이트 캐릭터가 좌/우/상/하로 이동한다.
    - j/J: 스프라이트가 점프한다.
    - e/E: 이미지 크기가 확대됐다가 제자리로 돌아간다..
    - s/S: 이미지 크기가 줄어들었다가 제자리로 돌아간다.
    - t/T: 스프라이트 이미지를 복사하여 다른 곳에 애니메이션 한다. 위의 키보드 명령어를 입력하면 스프라이트 이미지 모두 명령어를 수행한다. (twin 이미지가 생긴다) 1개 이상의 트윈 이미지가 만들어진다. (최대 5개까지)
  - 왼쪽 마우스:
    - 마우스가 이미지를 클릭하면 다른 애니메이션이 보여지고 다른 곳으로 이동한다.
    - 이동된 위치에서 다시 원래의 캐릭터 애니메이션이 나타난다.
    - 복사된 스프라이트 이미지에 마우스 명령은 각각 따로 수행된다.
  - 오른쪽 마우스:
    - 오른쪽 마우스를 이미지 위에 클릭하면, 이미지는 움직이던 방향으로 빠른 속도로 진행하고 화면 밖으로 나가면 반대편에서 나타나 계속 이동 후 원래 자리에 도착하면 멈춘다.

실습 5-5

- **미니 테트리스 만들기**
  - 테트리스 보드를 만든다. (가로 10칸, 세로 20칸)
  - 다음의 네 종류의 블록이 위에서 임의의 순서대로 떨어진다.



  - **블록은 space 키보드를 누르면 90도씩 반시계 방향 또는 시계방향으로 회전한다.**
  - 삭제 조건
    - 블록의 **줄무늬 모양이 가로로 한 줄이 되면 그 줄은 삭제된다.** 위의 줄이 아래로 내려온다.
    - 블록의 **줄무늬 모양이 네모가 되면 삭제된다.**
  - 블록은 비트맵을 이용한다.



실습 5-6