제5장 단축키와 비트맵

2014년 윈도우 프로그래밍

- 학습목표
  - 메뉴에 단축키를 설정할 수 있다.
  - 비트맵 형식의 그림 파일을 불러 화면에 출력할 수 있다.
  - 더블 버퍼링 기법을 이용해 비트맵 그림 파일로 애니메이션을 만들 수 있다

- 내용
  - 단축키
  - 비트맵
  - 더블 버퍼링

1절. 단축키

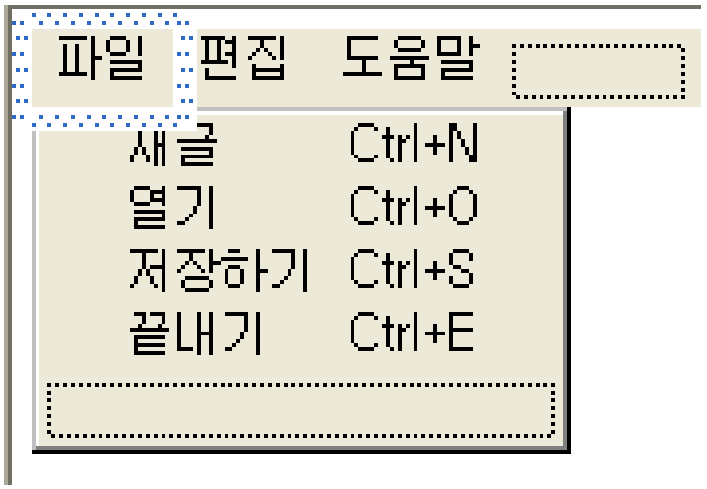
- **단축키**
  - 메뉴항목을 선택하지 않고 키보드의 단축키 만으로 메뉴의 기능을 수행하게 하는 기능

- **설정방법**
  - 메뉴의 속성창에서 Caption에 단축키 표시
  - 새로운 accelerator 추가
    - 2010 환경: 리소스추가에서 accelerator 새로 만들기 선택
  - 단축키 맵핑
  - 단축키 설정

5-1 메뉴에 단축키 설정하기

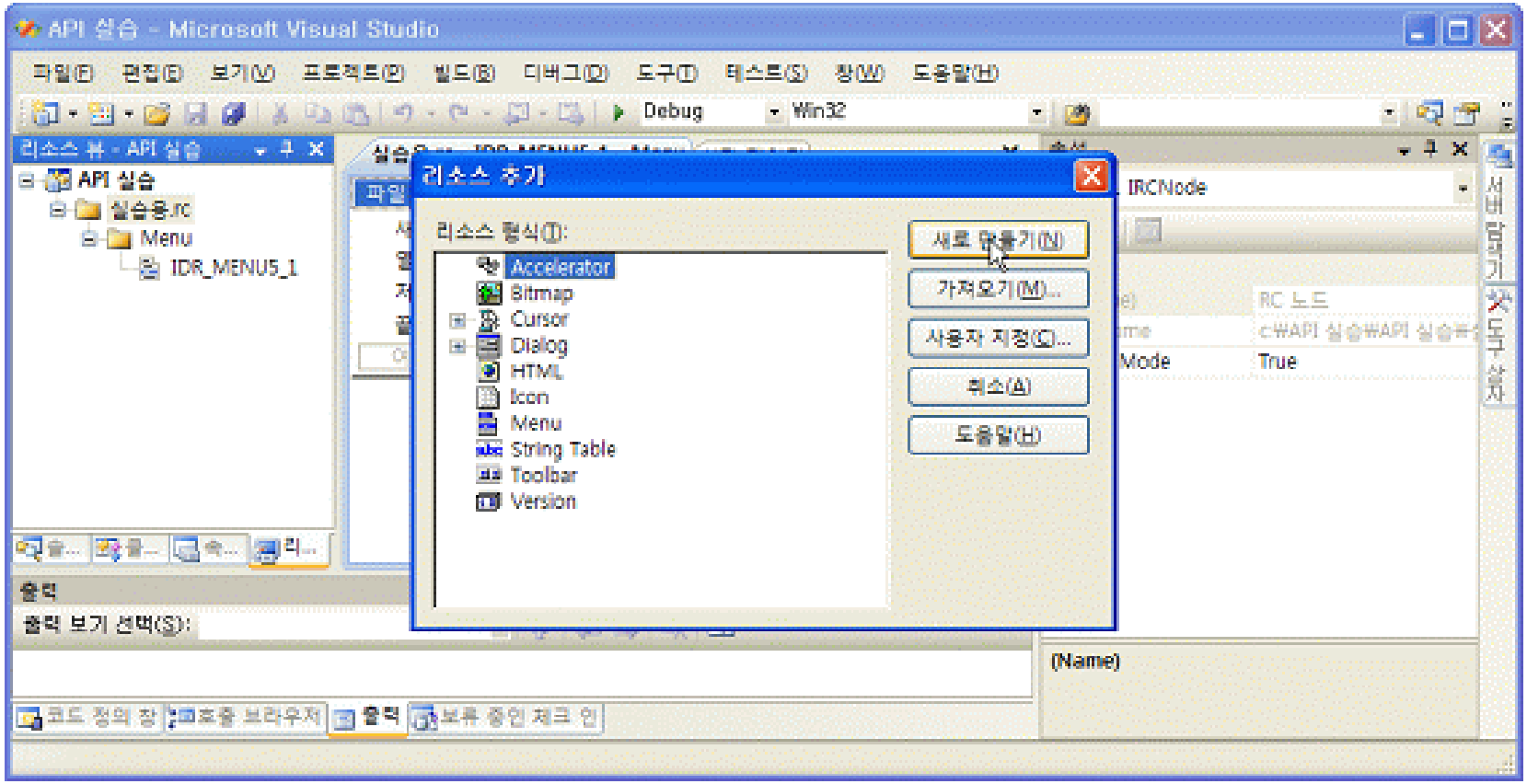
- 메뉴항목 설정표를 참고하여 메뉴항목 Caption에 단축키 표시

Caption	ID	속성
파일		Pop-up
새글₩tCtrl+N	ID_FILENEW	디폴트
열기₩tCtrl+O	ID_FILEOPEN	디폴트
저장하기₩tCtrl+S	ID_FILESAVE	디폴트
끝내기₩tCtrl+E	ID_EXIT	디폴트

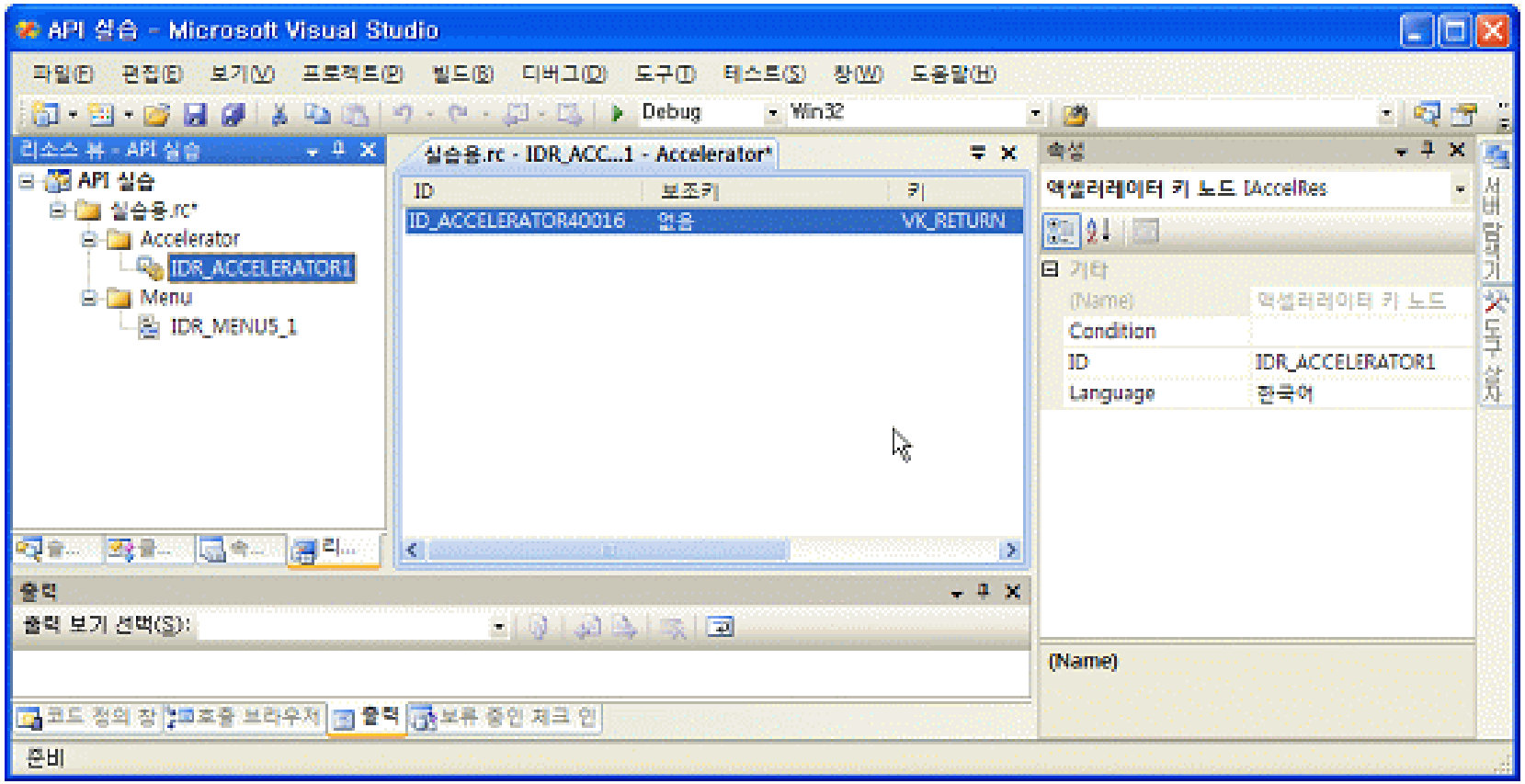


단축기 리소스 (Accelerator) 추가

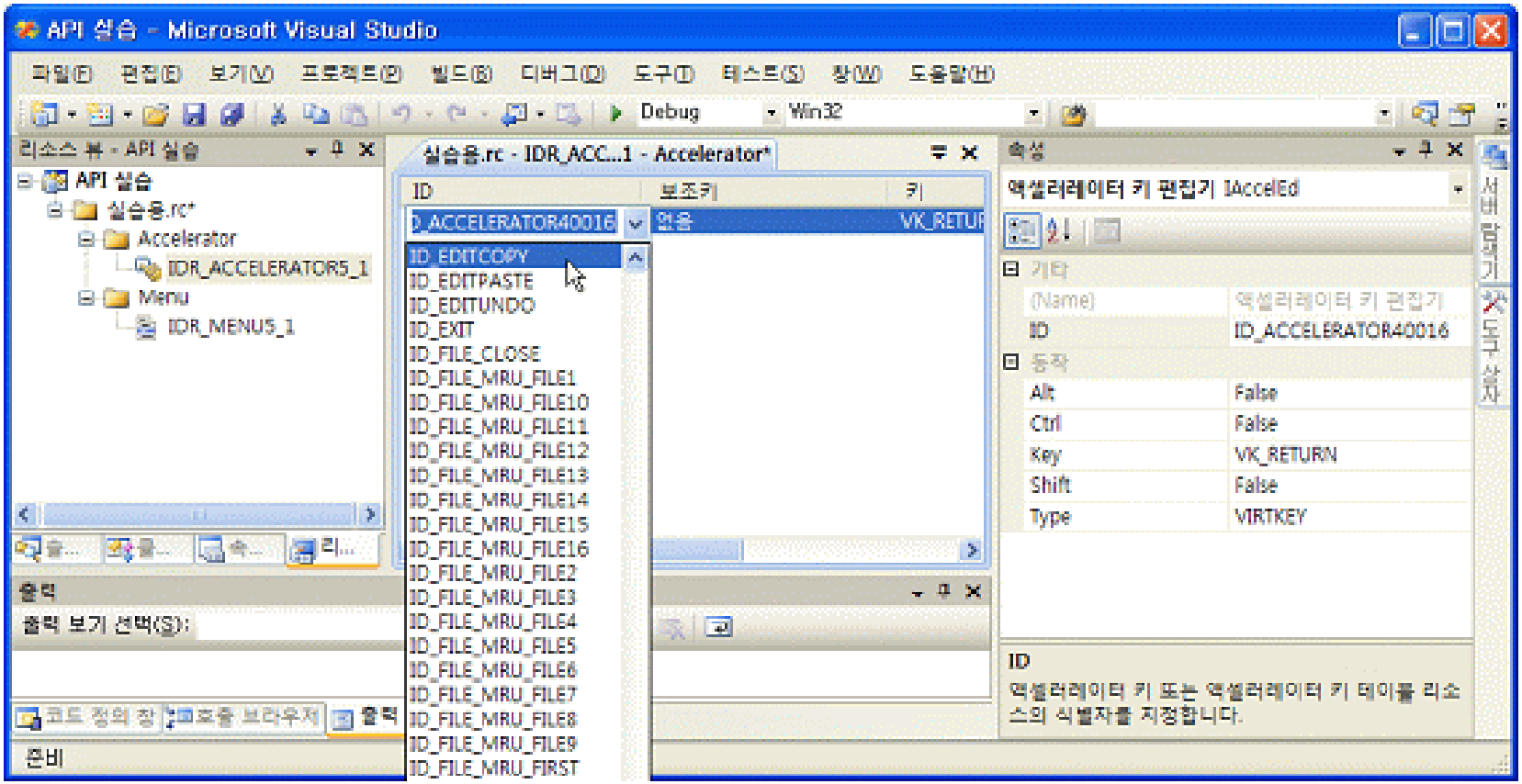
- Visual Studio 2010 환경



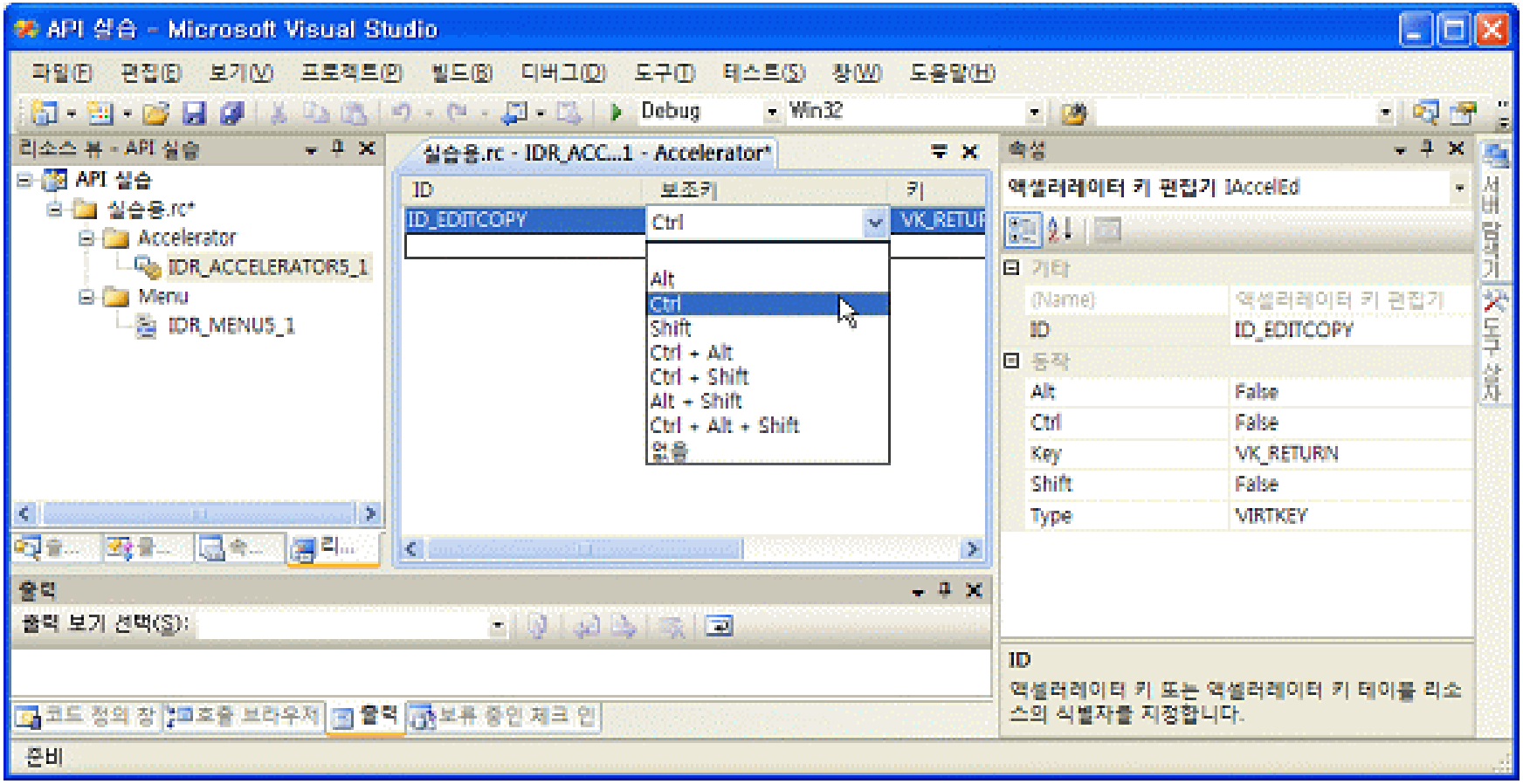
단축기 리소스 (Accelerator) 추가



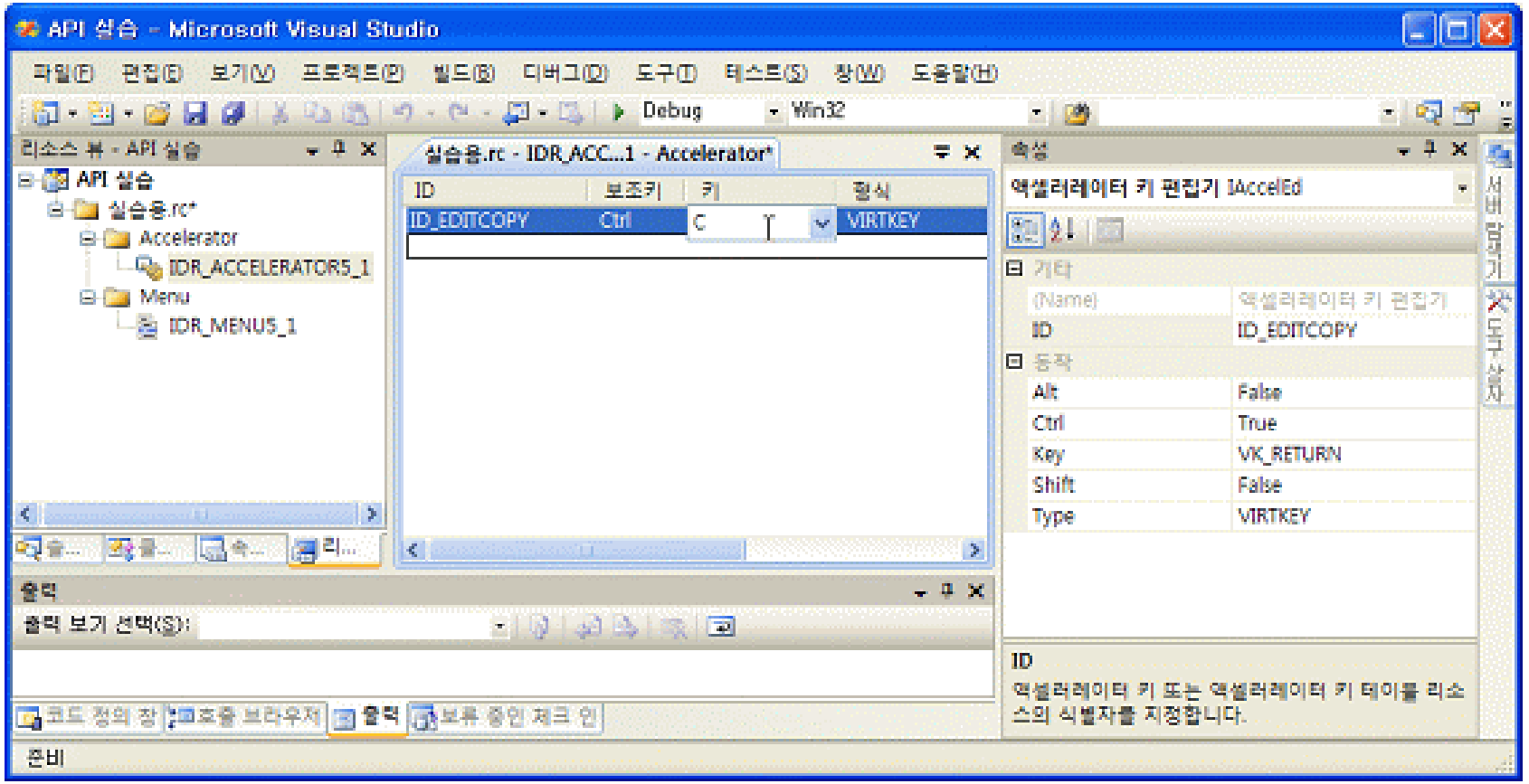
메뉴 ID 선택


API 실습 - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 도구(T) 테스트(S) 창(W) 도움말(H)
Debug
Win32
리소스 뷰 - API 실습
API 실습
실습용.rc*
Accelerator
IDR_ACCELERATOR5_1
Menu
IDR_MENU5_1
실습용.rc - IDR_ACC...1 - Accelerator*
ID
보조키
키
D_ACCELERATOR40016
VK_RETUR
ID_EDITCOPY
ID_EDITPASTE
ID_EDITUNDO
ID_EXIT
ID_FILE_CLOSE
ID_FILE_MRU_FILE1
ID_FILE_MRU_FILE10
ID_FILE_MRU_FILE11
ID_FILE_MRU_FILE12
ID_FILE_MRU_FILE13
ID_FILE_MRU_FILE14
ID_FILE_MRU_FILE15
ID_FILE_MRU_FILE16
ID_FILE_MRU_FILE2
ID_FILE_MRU_FILE3
ID_FILE_MRU_FILE4
ID_FILE_MRU_FILE5
ID_FILE_MRU_FILE6
ID_FILE_MRU_FILE7
ID_FILE_MRU_FILE8
ID_FILE_MRU_FILE9
ID_FILE_MRU_FIRST
속성
액셀러레이터 키 편집기 IAccelEd
기타
(Name)
액셀러레이터 키 편집기
ID
ID_ACCELERATOR40016
동작
Alt
False
Ctrl
False
Key
VK_RETURN
Shift
False
Type
VIRTKEY
ID
액셀러레이터 키 또는 액셀러레이터 키 테이블 리소스의 식별자를 지정합니다.
출력
출력 보기 선택(S):
준비

보조키 선택


API 실습 - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 도구(T) 테스트(S) 창(W) 도움말(H)
Debug
Win32
리소스 뷰 - API 실습
API 실습
실습용.rc*
Accelerator
IDR_ACCELERATOR5_1
Menu
IDR_MENU5_1
실습용.rc - IDR_ACC...1 - Accelerator*
ID
보조키
키
ID_EDITCOPY
Ctrl
VK_RETUR
Alt
Ctrl
Shift
Ctrl + Alt
Ctrl + Shift
Alt + Shift
Ctrl + Alt + Shift
없음
속성
액셀러레이터 키 편집기 IAccelEd
기타
(Name)
액셀러레이터 키 편집기
ID
ID_EDITCOPY
동작
Alt
False
Ctrl
False
Key
VK_RETURN
Shift
False
Type
VIRTKEY
ID
액셀러레이터 키 또는 액셀러레이터 키 테이블 리소스의 식별자를 지정합니다.
출력
출력 보기 선택(S):
준비

키 선택


API 실습 - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 도구(T) 테스트(S) 창(W) 도움말(H)
Debug
Win32
리소스 뷰 - API 실습
API 실습
실습용.rc*
Accelerator
IDR_ACCELERATOR5_1
Menu
IDR_MENU5_1
실습용.rc - IDR_ACC...1 - Accelerator*
ID
보조키
키
형식
ID_EDITCOPY
Ctrl
C
VIRTKEY
속성
액셀러레이터 키 편집기 IAcceled
기타
(Name)
액셀러레이터 키 편집기
ID
ID_EDITCOPY
동작
Alt
False
Ctrl
True
Key
VK_RETURN
Shift
False
Type
VIRTKEY
출력
출력 보기 선택(S):
ID
액셀러레이터 키 또는 액셀러레이터 키 테이블 리소스의 식별자를 지정합니다.
준비

단축키 설정

- 단축키를 프로그램에 연동하기
- 새글, 열기에 메시지 박스 출력

WinMain 부분

```
HACCEL hAcc;
. . . 중략 . . .
hAcc=LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR_ACCELERATOR5_1));
while (GetMessage (&msg, NULL, 0, 0))
{
  if(!TranslateAccelerator(hwnd,hAcc,&msg)) // 단축키 -> 메뉴 ID로 인식
  {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
  }
}
```

WinProc 부분

단축키에 대한 처리 = 메뉴와 동일한 처리

2절. 비트맵

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

2절. 비트맵

- 컴퓨터에서 영상(image)은 전자적인 형태로 만들어지거나 복사되고, 저장된 그림이다.
- 영상은 벡터그래픽(vector graphics)이나 래스터그래픽(raster graphics) 형태가 있다.

- 윈도우즈에서는 래스터 형식으로 저장된 영상을 비트맵(bitmap), 벡터형식으로 저장된 영상을 메타파일(metafile)이라고 한다.
  - 비트맵은 각 픽셀(PIXEL: PICture ELement, 화소)을 표시하기 위한 공간과 색상으로 정의된다.
    - 비트맵은 영상을 표현할 때 이미 결정된 주사선으로 구성된 래스터 영상을 이용하기 때문에, 사용자가 영상의 크기를 바꾸게 되면 선명도가 떨어지게 된다.
  - 벡터그래픽은 주어진 2차원이나 3차원 공간에 선이나 형상을 배치하기 위해 일련의 명령어들이나 수학적 표현을 통해 디지털 영상을 만든다.
    - 벡터 그래픽 파일에는 선을 그리기 위해 각 비트들이 저장되지 않고 연결될 일련의 점의 위치가 저장된다.
    - 그래서 파일크기가 작아지며 변형이 용이한 특징을 갖는다.

2절. 비트맵

- 윈도우즈 OS에서 지원하는 비트맵은 두가지이다.
  - 윈도우즈 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

2절. 비트맵

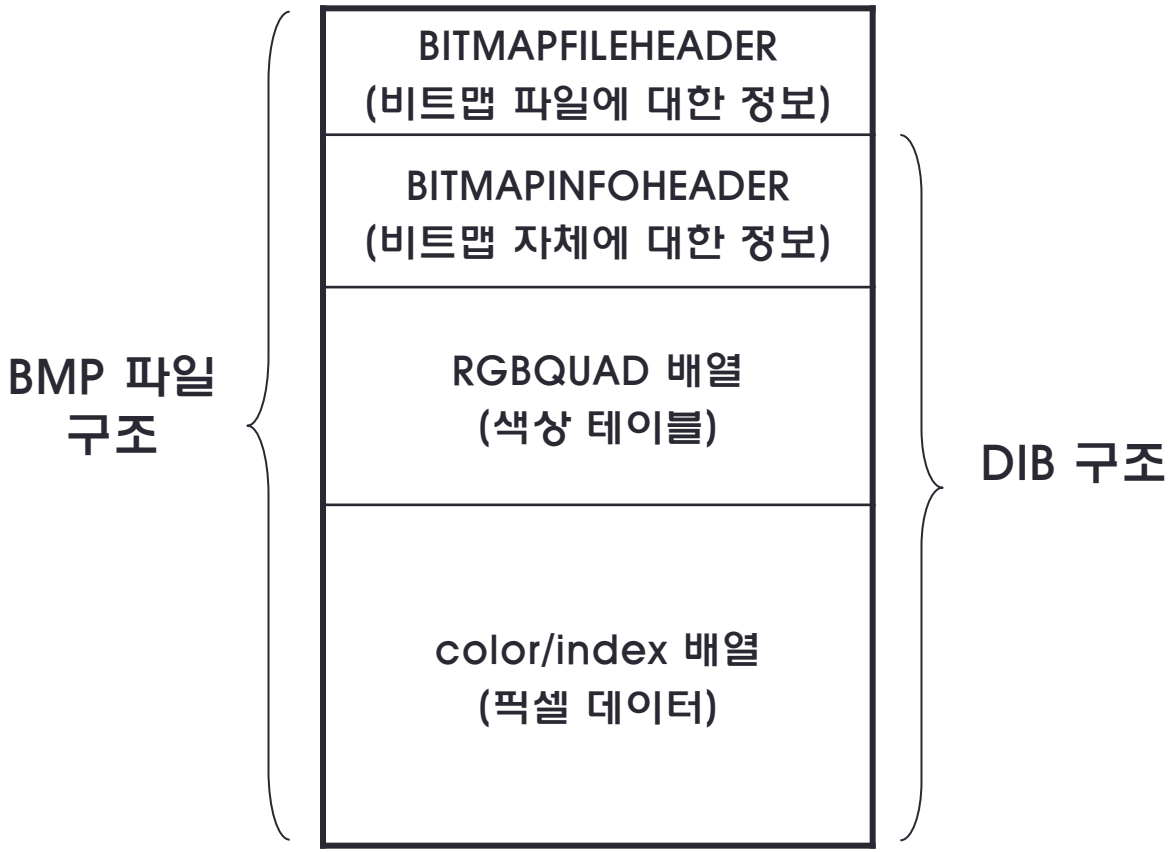
- 비트맵 구조체 (DDB 비트맵)

```
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 읽기
  - DC는 DDB 비트맵 타입만이 선택된다.
  - 리소스 에디터에 의해서 만들어지는 비트맵 리소스들은 DIB 비트맵
  - LoadBitmap() 함수로 읽는다: 이 함수로 읽은 비트맵은 DDB로 변경된다.

2절. 비트맵

- 비트맵 구조체 (DIB 비트맵)



2절. 비트맵

- 비트맵 구조체 (DIB 비트맵)

```
typedef struct tagBITMAPFILEHEADER {
    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;
```

2절. 비트맵

```
typedef struct tagBITMAPINFOHEADER{
    DWORD   biSize;             // BITMAPINFOHEADER 구조체 크기
    LONG    biWidth;            // 비트맵의 가로 픽셀 수
    LONG    biHeight;           // 비트맵의 세로 픽셀 수
    WORD    biPlanes;           // 장치에 있는 색상면의 개수 (반드시 1)
    WORD    biBitCount;         // 한 픽셀을 표현할 수 있는 비트의 수
    DWORD   biCompression;      // 압축 상태 지정 (BI_RGB: 압축되지 않은 비트맵
                                // BI_RLE8: 8비트 압축, BI_RLE4: 4비트 압축
    DWORD   biSizeImage;        // 실제 이미지의 바이트 크기
                                // (압축되지 않은 경우는 0)
    LONG    biXPelsPerMeter;    // 미터당 가로 픽셀 수
    LONG    biYPelsPerMeter;    // 미터당 세로 픽셀 수
    DWORD   biClrUsed;          // 색상테이블의 색상중 실제 비트맵에서 사용되는
                                // 색상수 (0 : 비트맵은 사용할 수 있는 모든색상을
                                // 사용,그 외 : RGBQUAD구조체배열의 크기는
                                // 이 멤버의 크기만큼 만들어짐)
    DWORD   biClrImportant;     // 비트맵을 출력하는데 필수적인 색상수
                                // (0 : 모든 색상이 사용되어야 함)

} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

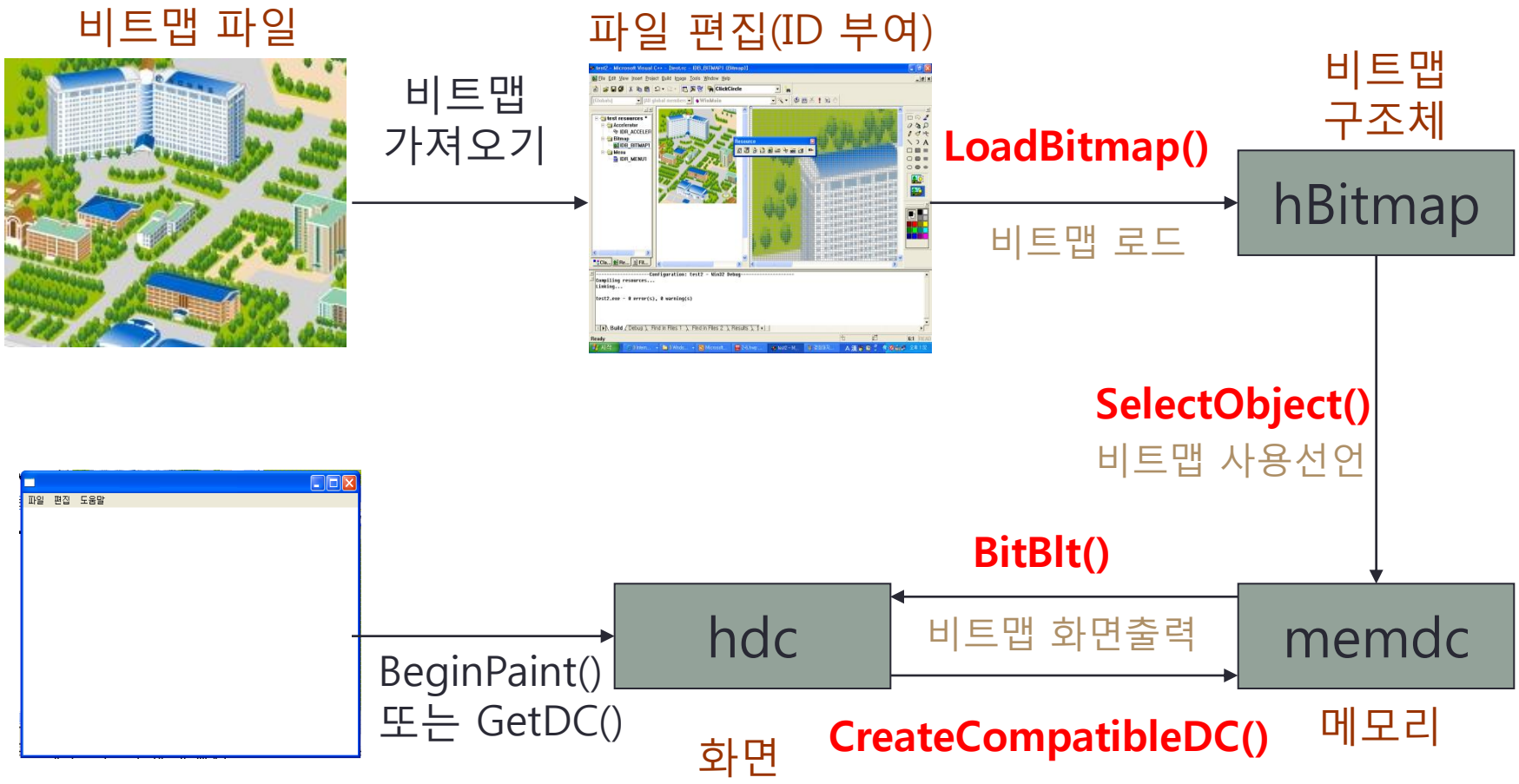
2절. 비트맵

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

이미지 만들기

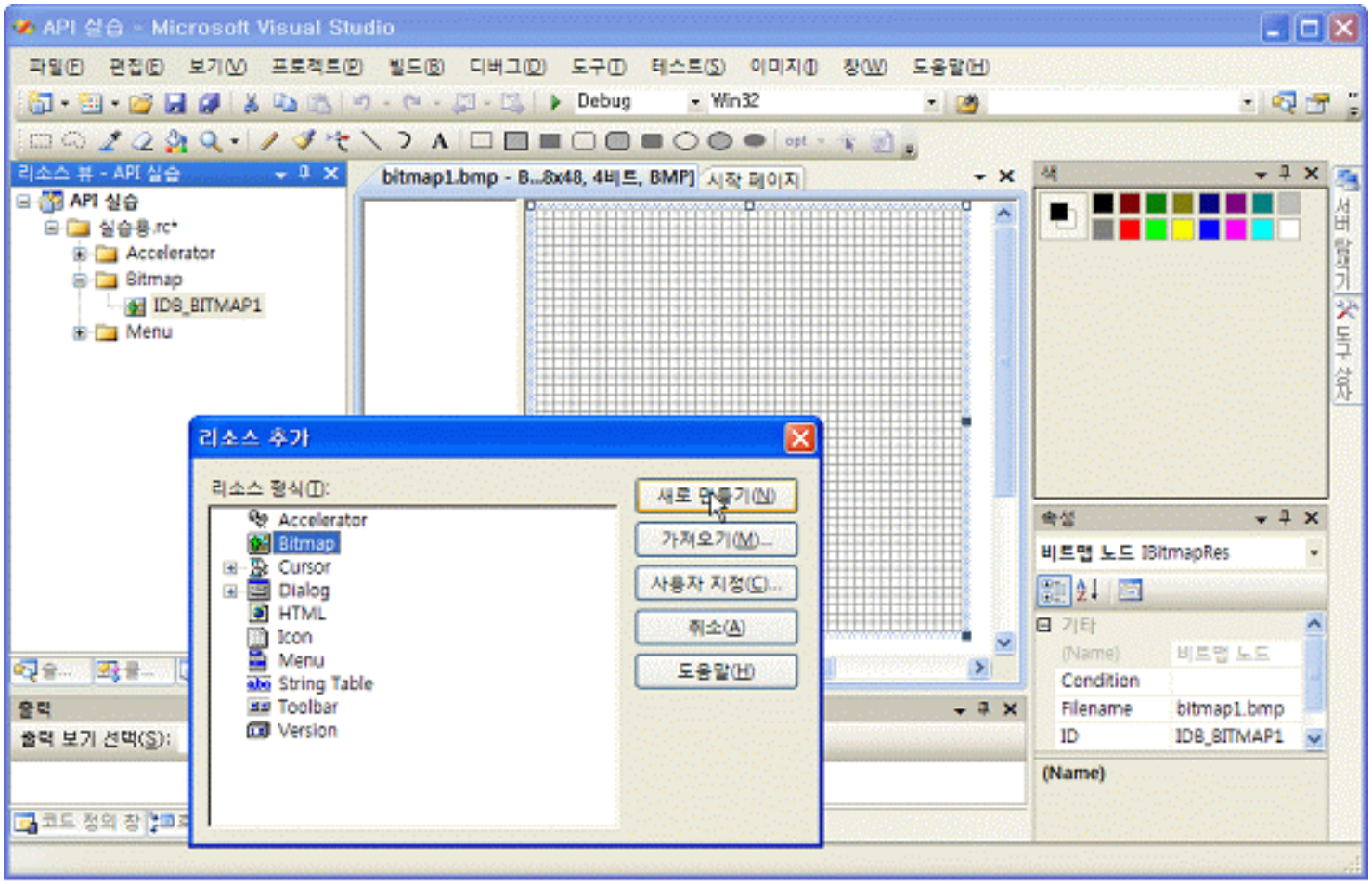
- 프로젝트 작성
  - 윈도우 응용
- 리소스 파일 작성
  - 방법: 소스 파일 작성과 유사
  - ”C++ Source” 대신에 Resource Script 선택
  - 리소스 파일 이름 명시
- 이미지 만들기 및 불러오기
  - 새로 만들기: 리소스 도구상자 이용
  - 불러오기: Visual Studio의 리소스뷰에서 import

비트맵 출력하기


비트맵 파일
비트맵
가져오기
파일 편집(ID 부여)
LoadBitmap()
비트맵 로드
비트맵
구조체
hBitmap
SelectObject()
비트맵 사용선언
BitBlt()
비트맵 화면출력
hdc
memdc
BeginPaint()
또는 GetDC()
화면
CreateCompatibleDC()
메모리

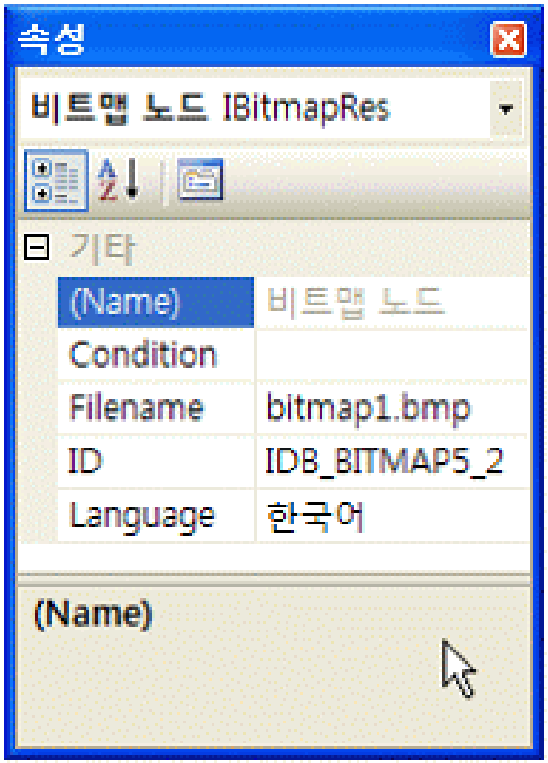
비트맵 만들기

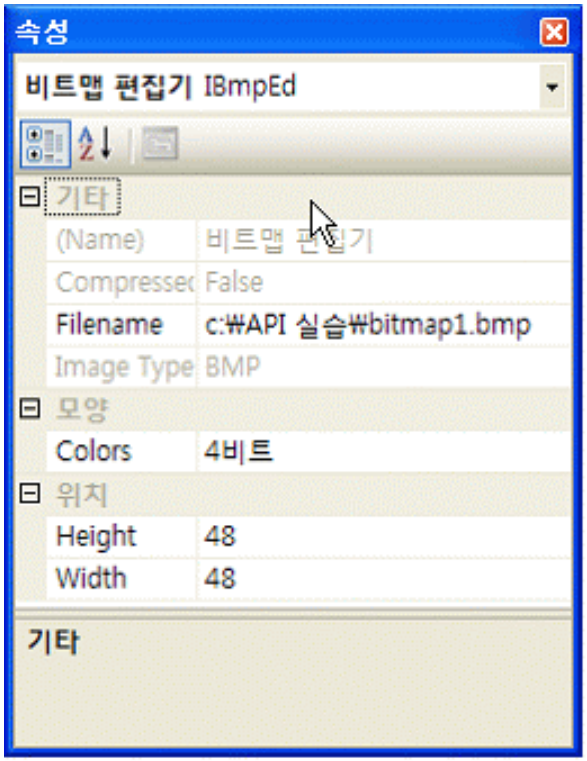
- Visual Studio 2010 환경



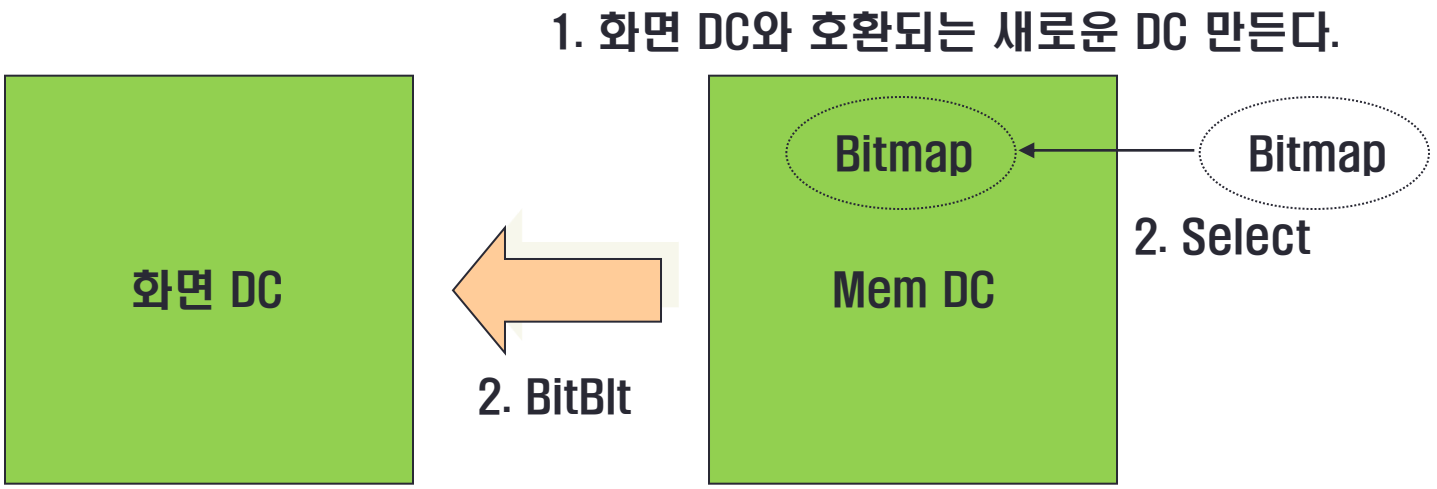
비트맵 속성

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





비트맵 읽기



- HDC hMemDC;
  - 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: 주어진 DC

비트맵 읽기

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

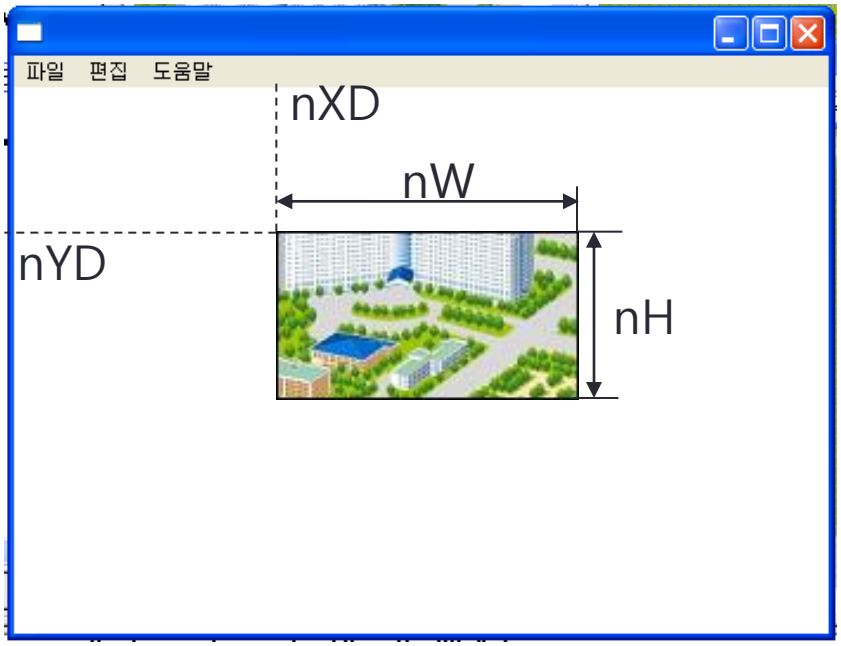
  - HBITMAP LoadBitmap (HINSTANCE hInstance, LPCTSTR lpBitmapName);
    - 비트맵 로드
    - hInstance: 어플리케이션 인스턴스 핸들
    - lpBitmapName: 비트맵 리소스 이름

BitBlt() -> 1 : 1 Copy

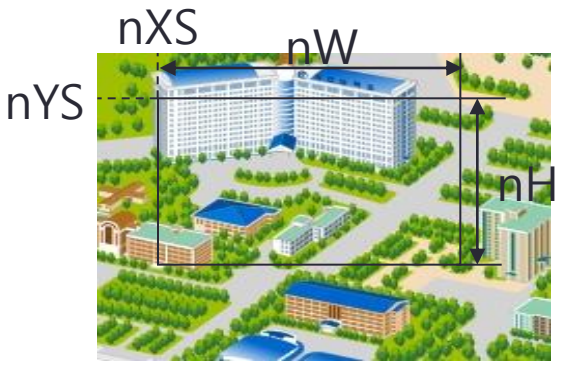
- DC 간의 영역 고속 복사

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

hdc



memdc



BitBlt() -> 1 : 1 Copy

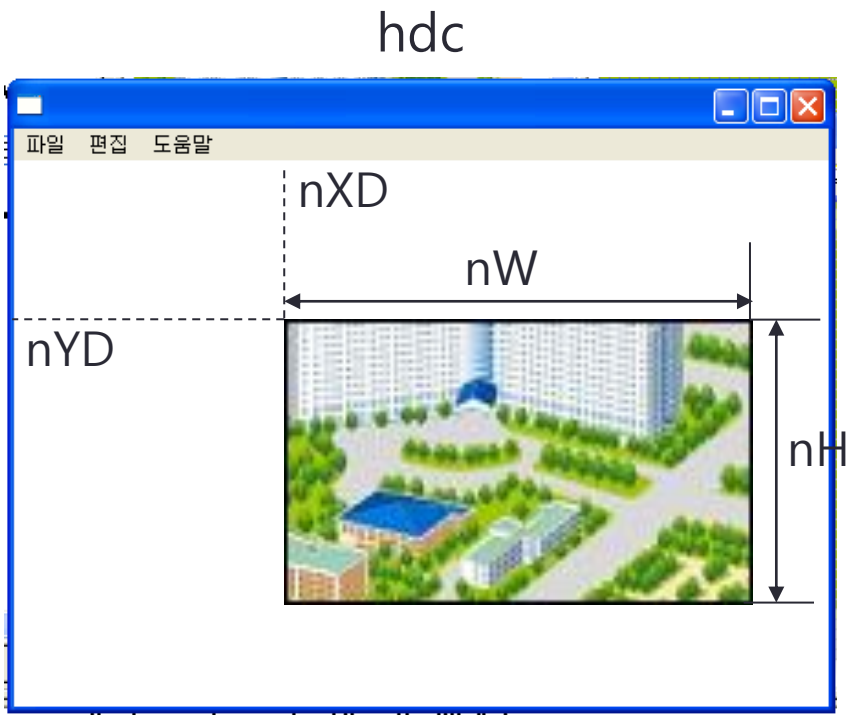
- BOOL BitBlt (HDC hdc, int nXD, int nYD, int nW, int nH,
  HDC memdc, int nXS, int nYS,
  DWORD dwRop);
  - DC간의 영역끼리 고속 복사 수행 (메모리 DC 표면의 비트맵을 화면 DC로 복사)
  - hdc: 복사 대상 DC
  - nXD, nYDest: 복사 대상의 x, y 좌표 값
  - nW, nHeight: 복사 대상의 폭과 높이
  - memdc: 복사 소스 DC
  - nXS, nYS: 복사 소스의 좌표
  - dwRop: 래스터 연산 방법
    - BLACKNESS : 검정색으로 칠한다.
    - DSTINVERT: 대상의 색상을 반전시킨다.
    - NOTSRCCOPY: 소스값을 반전시켜 칠한다.
    - SRCPAINT: 소스와 대상의 OR연산 값으로 칠한다.
    - SRCCOPY: 소스값을 그대로 칠한다.
    - SRCAND: 소스와 대상의 AND연산 값으로 칠한다.
    - WHITENESS: 흰색으로 칠한다.
- 비트맵 출력 후, 메모리 DC와 비트맵 해제 (DeleteDC / DeleteObject)

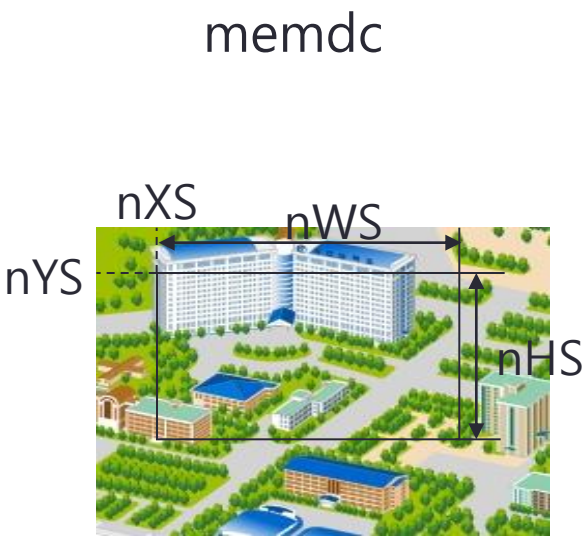
5-2 비트맵 출력

```
HDC hdc, memdc ;
PAINTSTRUCT ps ;
static HBITMAP hBitmap;

switch (iMsg) {

case WM_CREATE:
   hBitmap = (HBITMAP)LoadBitmap ( hInstance,
                                   MAKEINTRESOURCE(IDB_BITMAP5_2));
   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);
                                   // SRCCOPY : 바탕색을 무시하고 그려라

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

StretchBlt() : 확대, 축소 Copy

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





StretchBlt() : 확대, 축소 Copy

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

  - hdc: 복사대상 DC
  - nXD, nYD: 복사대상 DC x, y 좌표값
  - nW, nH: 복사대상 DC의 폭과 높이
  - HDC memdc: 복사소스 DC
  - nXS, nYS: 복사소스 DC의 x, y 좌표값
  - nWS, nHS복사소스 DC의 폭과 높이
  - dwRop: 래스터 연산 방법

GetObject (): 그림 크기 알아내기

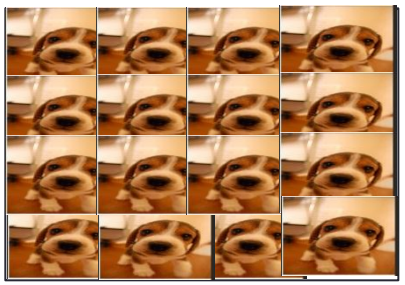
- 그림 크기 알아내기
  - int GetObject (HGDIOBJ hgdiobj, int cbBuffer,
    LPVOID lpvObject);
    - HGDIOBJ hgdiobj: GDI 오브젝트 핸들
    - int cbBuffer: 오브젝트 버퍼의 크기에 관한 정보
    - LPVOID lpvObject: 오브젝트 정보 버퍼를 가리키는 포인터

```
BITMAP bmp;
int mWidth, mHeight;

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

실습 5-1

- 제목
  - 메뉴를 이용하여 윈도우에 배경 그림 넣기
- 내용
  - 그림을 프로그램에서 출력한다.
  - 메뉴:
    - 전체 화면:
      - StretchBlt()를 이용하여 윈도우에 빈공간 없이 배경그림을 그린다.
      - 윈도우 크기가 변경되어도 윈도우 화면 전체에 배경그림이 나와야 한다.
    - 바둑판 모양:
      - 서브 메뉴: 3*3 / 4*4/ 5*5 (세 종류의 바둑판 모양)
      - 비트맵 파일이 윈도우 화면에 연속해서 나타나게 함으로 바둑판 모양으로 윈도우 배경에 나타나도록 프로그램을 작성한다.
  - 메뉴에 단축키를 넣는다.

원본 이미지



실습 5-2

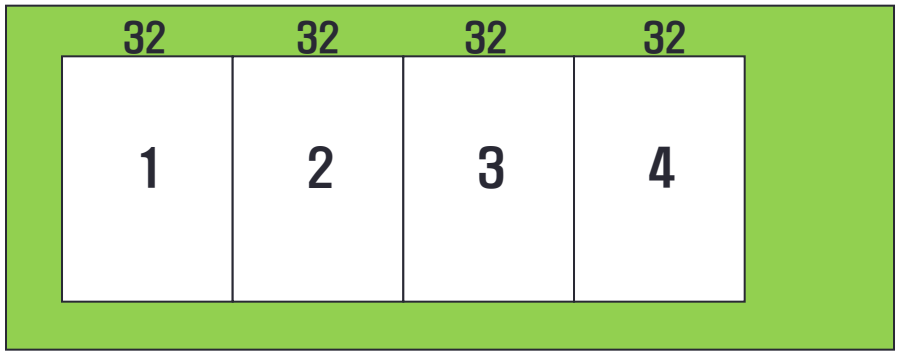
- 제목
  - 그림을 복사하여 붙여넣기

- 내용
  - 윈도우 화면위에서 마우스를 가지고 복사할 영역에 해당하는 사각형을 그리고 (고무줄 효과 사용하기) 메뉴의 복사하기 (단축키 Ctrl+v)를 누르면 사각형 내의 이미지가 복사된다.
  - 마우스를 복사할 위치에 클릭한 후 메뉴의 붙여넣기 (단축키 Ctrl+v)를 누르면 사각형 안에 있던 이미지는 윈도우내 다른 위치에 복사되어 나타나야 된다.

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



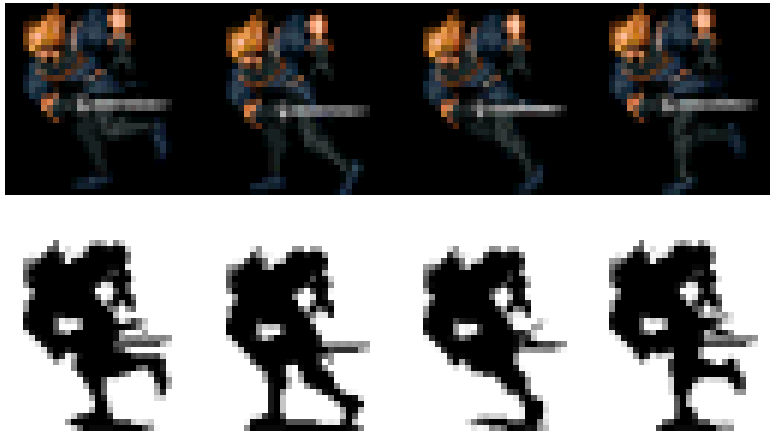
비트맵 애니메이션

- 사용 예:

```
static int FrameNo = 0;
POINT PtSrc;
PtSrc.x = 32 * FrameNo;
PtSrc.y = 0;

FrameNo++;
FrameNo = FrameNo % 4;

DrawBitmap (hdc, hBitmapFont, Pos, size, PtSrc, SRCPAINT);
```

비트맵 마스크

- 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.
    - 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

    - 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

투명 비트맵 처리

- 비트맵의 일부를 투명하게 처리하여 투명색 부분은 출력에서 제외한다.

```
BOOL TransparentBlt (
        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

- msimg32.lib를 링크한다. (속성 → 링커 → 명령줄에서 라이브러리 추가)

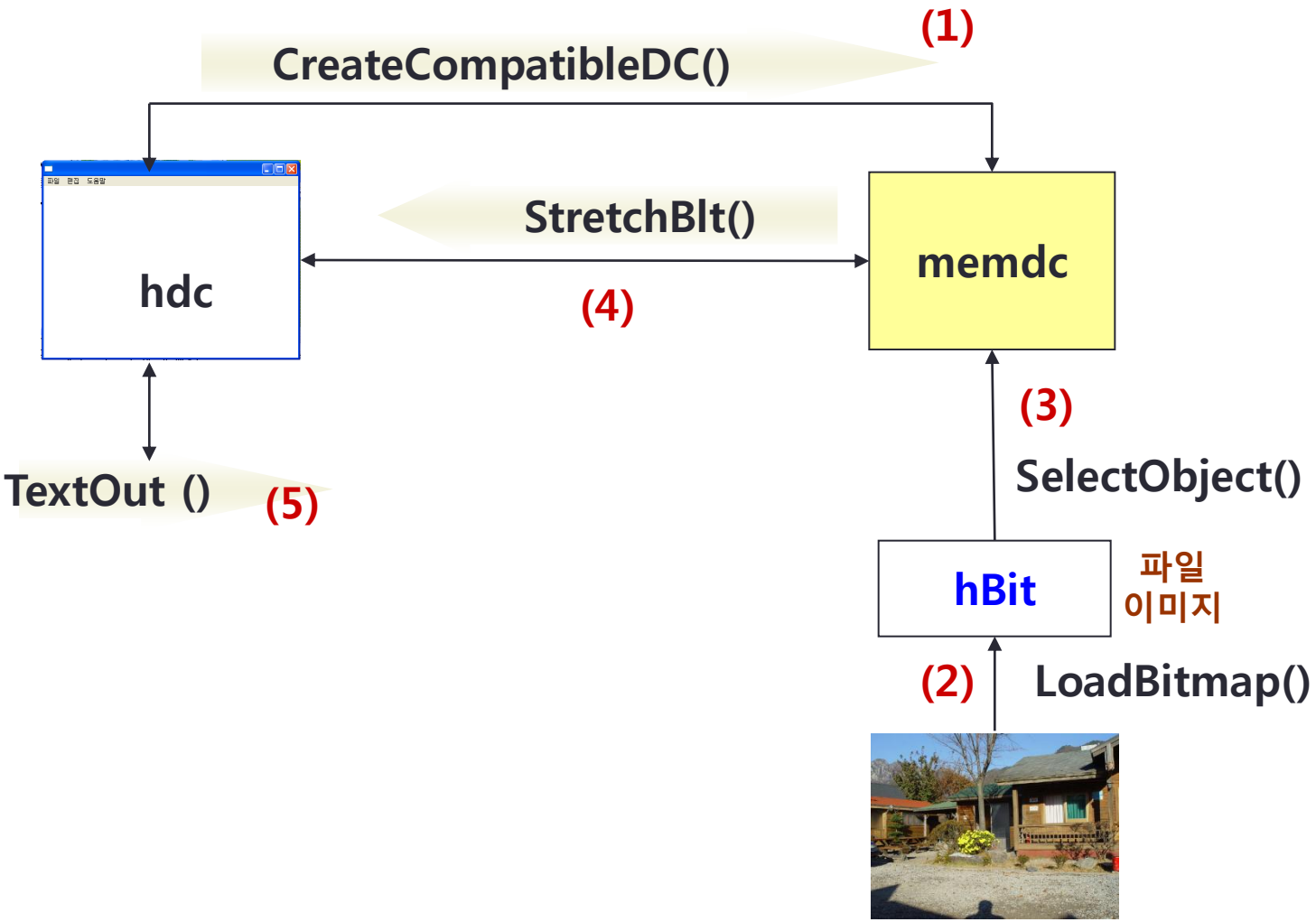
```
HDC hdc, memdc ;
PAINTSTRUCT ps ;
static HBITMAP hBitmap;

switch (iMsg) {

case WM_CREATE:
   hBitmap = (HBITMAP) LoadBitmap ( hInstance,
                                    MAKEINTRESOURCE(IDB_BITMAP5_2));
   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;
```

3절. 더블 버퍼링

- 비트맵 이미지 여러개를 이용하여 동영상을 나타낼때
  - 이미지를 순서대로 화면 디바이스 컨텍스트에 출력
  - 예를 들어 풍경위에 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 번갈아가며 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

5-3 배경화면위로 움직이는 글



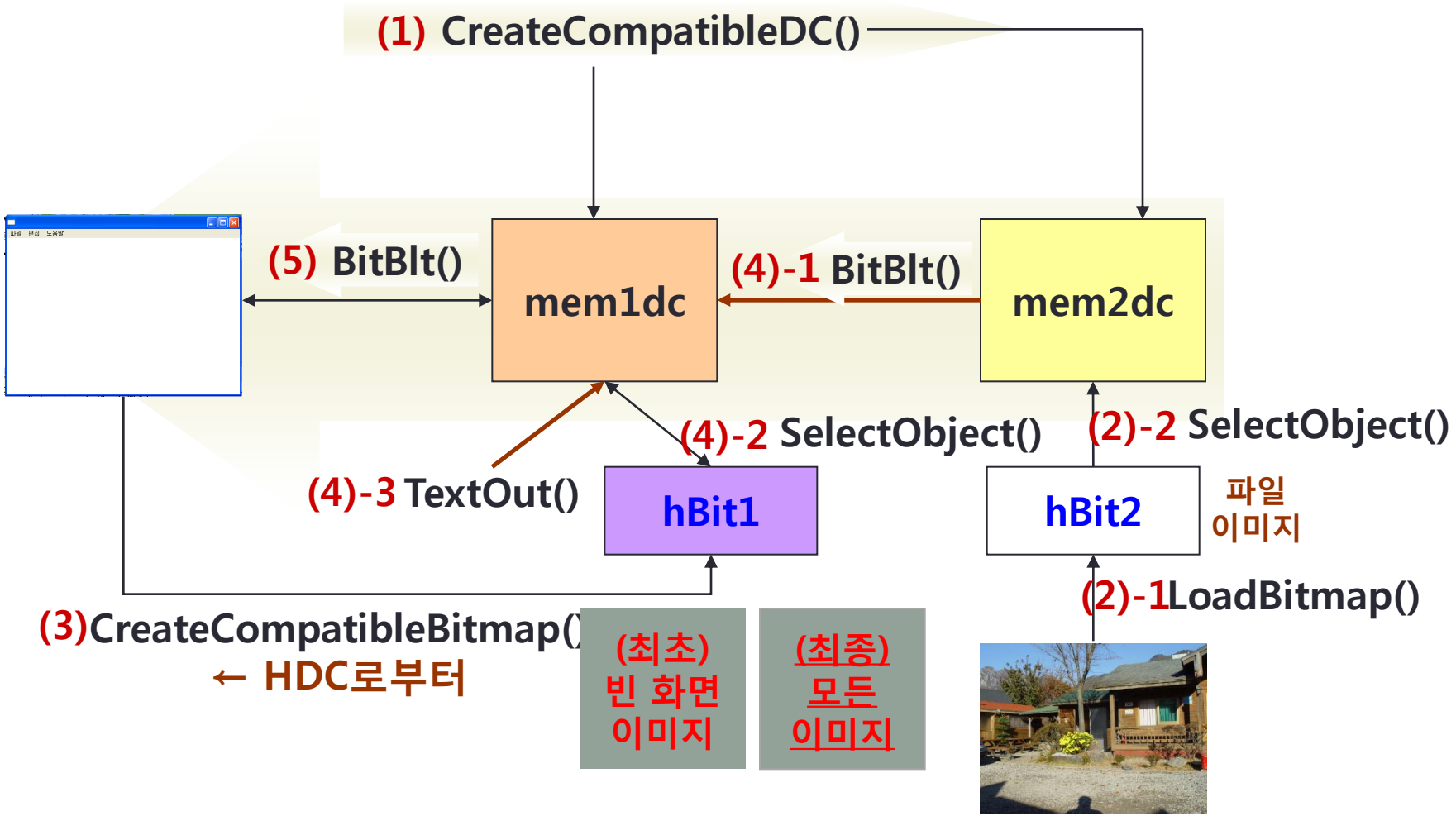
5-3 배경화면 위로 움직이는 글

```
HDC hdc, memdc;
static HBITMAP hBit, oldBit;
…
char word[] = "대한민국 화이팅";

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);
  return 0;
```

5-3 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);
  return 0;
```

5-4 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)-1LoadBitmap()
(3)CreateCompatibleBitmap()
← HDC로부터
(최초)
빈 화면
이미지
(최종)
모든
이미지

5-3 배경화면위로 움직이는 글 (더블버퍼링)

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

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_BITMAP5_4));
        break;
```

5-3 배경화면위로 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
```

5-3 배경화면위로 움직이는 글 (더블버퍼링)

```
    // 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);
    return 0;

  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);
    return 0;
```

실습 5-4

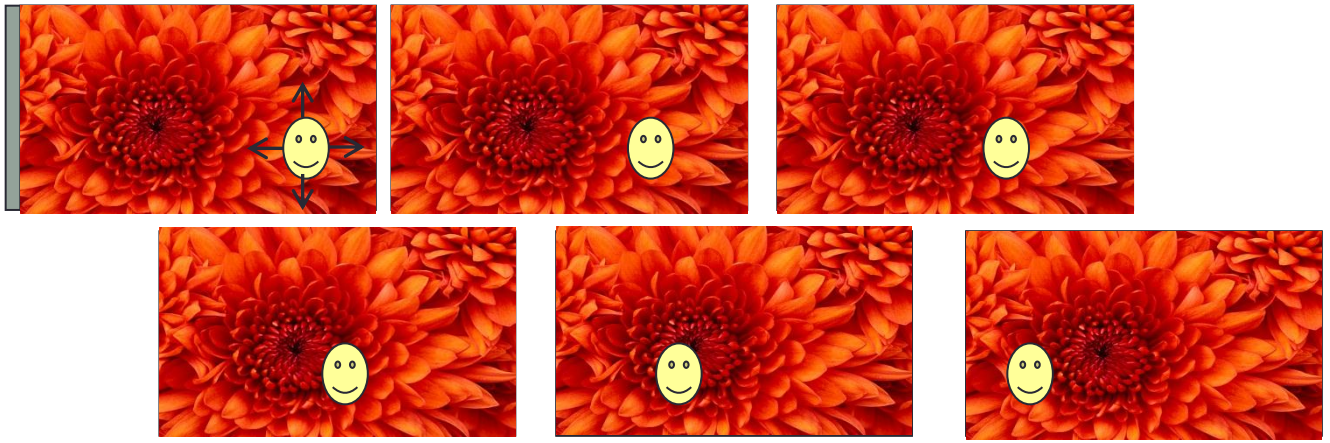
- 제목
  - 쥐따라 다니는 고양이

- 내용
  - 화면에 고양이가 그려져 있다.
  - 마우스의 왼쪽 버튼을 누르면 마우스 커서 위치에 쥐 그림을 출력하고 버튼을 떼면 쥐가 사라진다. 누른 채 드래그 할 수 있다.
  - 쥐가 화면상에 나타나면 고양이는 쥐를 잡기 위하여 움직이기 시작한다.
  - 쥐와 고양이 이미지는 원하는 이미지 사용 가능

실습 5-5

- 제목
  - 스프라이트 이동 애니메이션 구현

- 내용
  - 배경 이미지를 출력한다.
  - 스프라이트 이미지를 출력하고 키보드를 이용하여 스프라이트가 방향 전환을 한다.
    - 좌우상하 키: 스프라이트 캐릭터가 좌 우 하 상으로 이동한다.
    - j/J: 스프라이트가 점프한다.



실습 5-6

실습 5-7

실습 5-8

실습 5-9