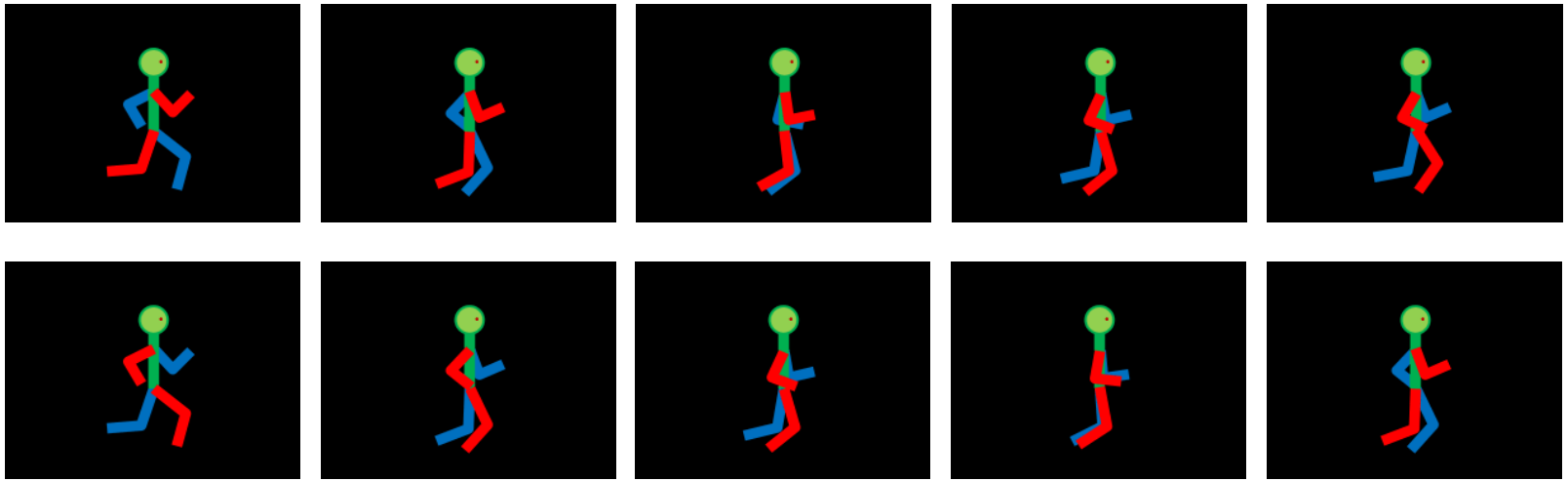
제 5장 비트맵과 애니메이션 1

2020년 1학기 윈도우 프로그래밍

비트맵 애니메이션

- **애니메이션**
  - 각 시점에 다른 그림을 그려서 움직이는 효과를 얻는다.
    - 각 시점의 한 그림: 프레임(Frame)
  - 애니메이션 동작은 **타이머로** 처리한다.
  - 매 타이머의 주기에 각 프레임을 표시하여, 각 동작에 하나의 프레임만을 보여준다.
    - 프레임 별 그림은 각각 다른 그림에 저장할 수 있다.
    - 프레임 별 그림은 한 개의 큰 그림에 한꺼번에 저장하고 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;                          // 타이머가 호출될 때 x 위치 변경
                if (xPos > 800)
                        xPos = 0;
                InvalidateRect (hwnd, NULL, true);
                return 0;
        case WM_PAINT:
                hdc = BeginPaint(hwnd, &ps);
                Animation (xPos, 300, hdc);          // 애니메이션 함수 호출
                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;

	RunBit[0]= LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_R1));	//--- 애니메이션 이미지 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));

	memdc = CreateCompatibleDC (hdc);	//--- 메모리 DC 생성

	hBit = LoadBitmap (hInst, MAKEINTRESOURCE(IDB_BITMAP_BACK));	//--- 배경 이미지 로드 (1)
	oldBit = (HBITMAP) SelectObject (memdc, hBit);	//--- 배경 이미지 선택하기
	BitBlt (hdc, 0, 0, 800, 600, memdc, 0, 0, SRCCOPY);	//--- 배경 이미지 그리기


	(HBITMAP) SelectObject (memdc, RunBit[count]);	// 애니메이션 이미지 선택하기 (2)
	BitBlt (hdc, xPos, yPos, 64, 64, memdc, 0, 0, SRCCOPY);	//--- 메모리 DC의 그림을 화면 DC에 출력하기
	SelectObject (memdc, oldBit);

	count++;	//--- 애니메이션 이미지 순서 카운트 (3)
	count = count % 10;

	for (i = 0; i < 10; i++)	//--- 생성한 객체 삭제하기
		DeleteObject (RunBit[i]);
	DeleteDC (memdc);
	DeleteObject (hBit);
}
```

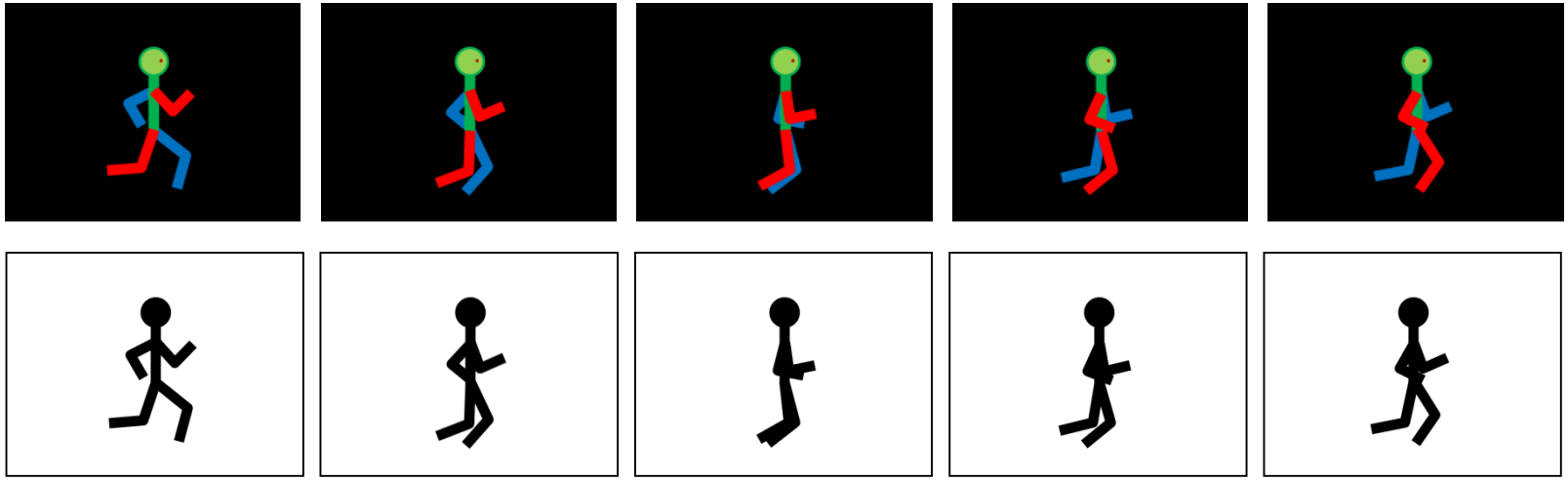
비트맵 애니메이션

- **결과 화면**

사람이 그려진 이미지가 x축으로 조금씩 이동하면서 프레임별로 출력된다.
문제점: 사람만 그려져야 하는데, 배경 그림도 모두 그려진다.

비트맵 마스크

- 사각형의 비트맵 이미지에서 원하는 부분만을 사용하고 싶을 때, 그리려는 비트맵 이미지 부분에 마스크를 씌운다.
  - 필요한 이미지:
    - 비트맵 이미지
    - 출력하고자 하는 부분을 검정색 처리한 마스크



비트맵 마스크

- **처리 방법:**
  - 각 프레임의 동작마다 마스크 처리와 소스 프레임의 그림을 각각 두번씩 씌워주어야 한다.
    - hBitmapMaskDC: 마스크 이미지가 선택되어 있는 메모리 DC
    - hBitmapFrontDC: 애니메이션 소스 이미지가 선택되어 있는 메모리 DC

1. 소스의 원하는 부분을 흑백으로 처리한 패턴을 배경 그림과 AND 연산 → 배경 이미지에 검정색 그림만이 그려진다.

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

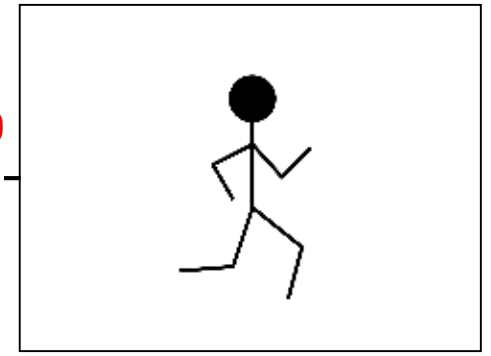
2. 여기에 원하는 그림을 배경 그림과 OR 연산 → 배경과 합성된 이미지로 나타나게 된다.

   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)

AND



흰색(255,255,255)

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++;   count %= 5;

        RunBit[0]= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_R1));     //--- 애니메이션 이미지
        . . .
        RunBit[4]= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_R5));

        Mask[0] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_M1));     //--- 마스크 이미지
        . . . . . .
        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);
}
```

마스크를 그린다.

캐릭터를 그린다.

투명 비트맵 처리

- **비트맵의 일부를 투명하게 처리하여 투명색 부분은 출력에서 제외한다.**
  - BitBlt 함수나 StretchBlt 함수 대신 사용한다.
  - 투명색을 설정하여 비트맵을 그린다.

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

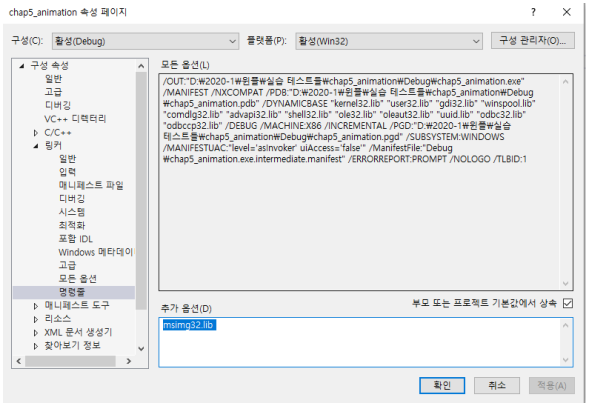
- 비트맵의 특정 색을 투명하게 처리하는 함수
  - HDC hdcDest: 출력할 목표 DC 핸들
  - int nXOriginDest : 좌측 상단의 x 좌표값
  - int nYOriginDest : 좌측 상단의 y 좌표값
  - int nWidthDest : 목표 사각형의 넓이
  - int hHeightDest : 목표 사각형의 높이

  - 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

```
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);

}
```

- **라이브러리 추가**
  - **msimg32.lib**를 링크한다.
  - 속성 → 링커 → 명령줄에서 라이브러리 추가



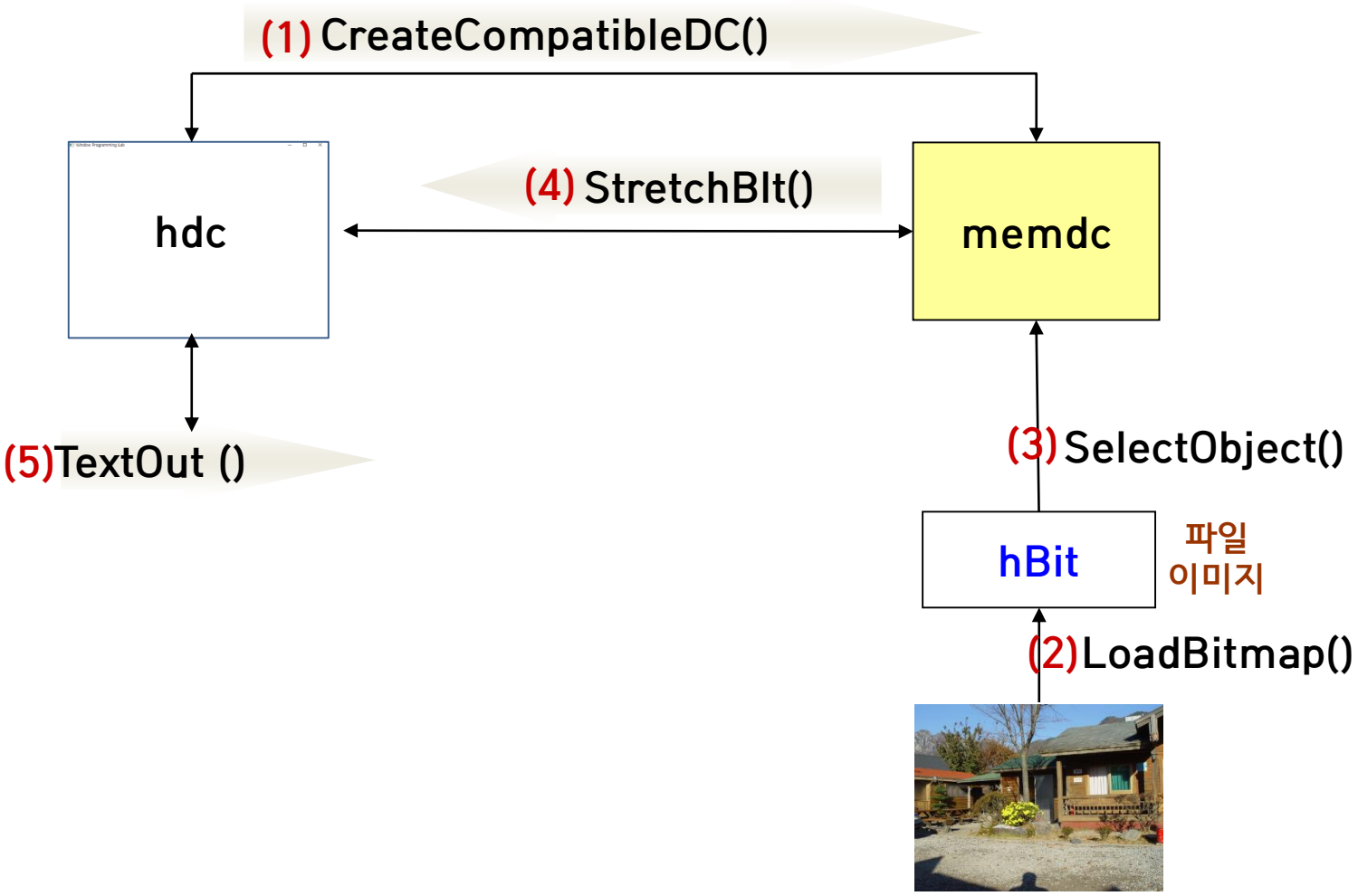
3. 더블 버퍼링

- **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 (더블 버퍼링 없이 출력)



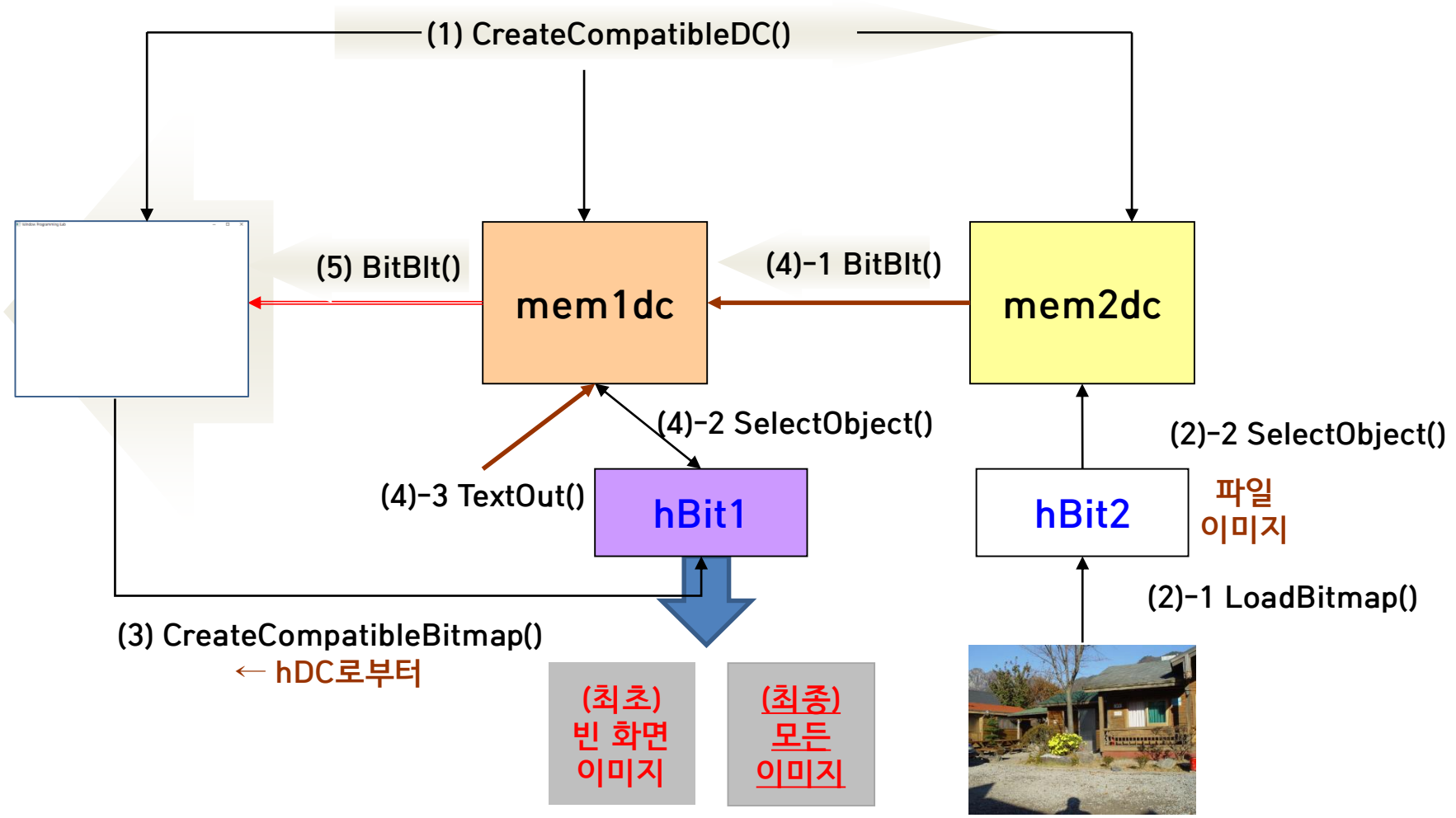
배경화면 위로 움직이는 글

```
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 = 0;                                   //--- 문자열을 출력할 y 위치
                    GetClientRect(hwnd, &rectView);
                    SetTimer(hwnd, 1, 70, NULL);
                    hBit=LoadBitmap (hInstance, MAKEINTRESOURCE(IDB_BITMAP1));
                    break;

                case WM_TIMER:                                  //--- Timer 생성시 마다 y좌표 변경 후, 출력 요청
                    yPos += 5;
                    if (yPos > rectView.bottom)
                                yPos = 0;
                    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);

        //--- 메모리 dc에 이미지 선택
            oldBit=(HBITMAP)SelectObject(memdc, hBit);

        //--- 메모리 DC -> 화면 DC(hdc)로 출력 (1)
            StretchBlt (hdc, 0, 0, rectView.right, rectView.bottom,
                                        memdc, 0, 0, rectView.right, rectView.bottom, SRCCOPY);
            SelectObject (memdc, oldBit);
            DeleteDC (memdc);

        //--- 문자열 출력 (2)
            TextOut(hdc, 200, yPos, word, strlen(word));

            EndPaint (hwnd, &ps);
            break;

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

배경화면 위로 움직이는 글 (더블버퍼링)



배경화면 위로 움직이는 글 (더블버퍼링)

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

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


	switch(iMsg) {
	case WM_CREATE:
		yPos = 0;
		GetClientRect(hwnd, &rectView);
		SetTimer(hwnd, 1, 70, NULL);

		// hBit2에 배경 그림 로드, 나중에 mem2dc에 hBit2 그림 설정
		hBit2 = LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BITMAP1));
		break;
```

배경화면 위로 움직이는 글 (더블버퍼링)

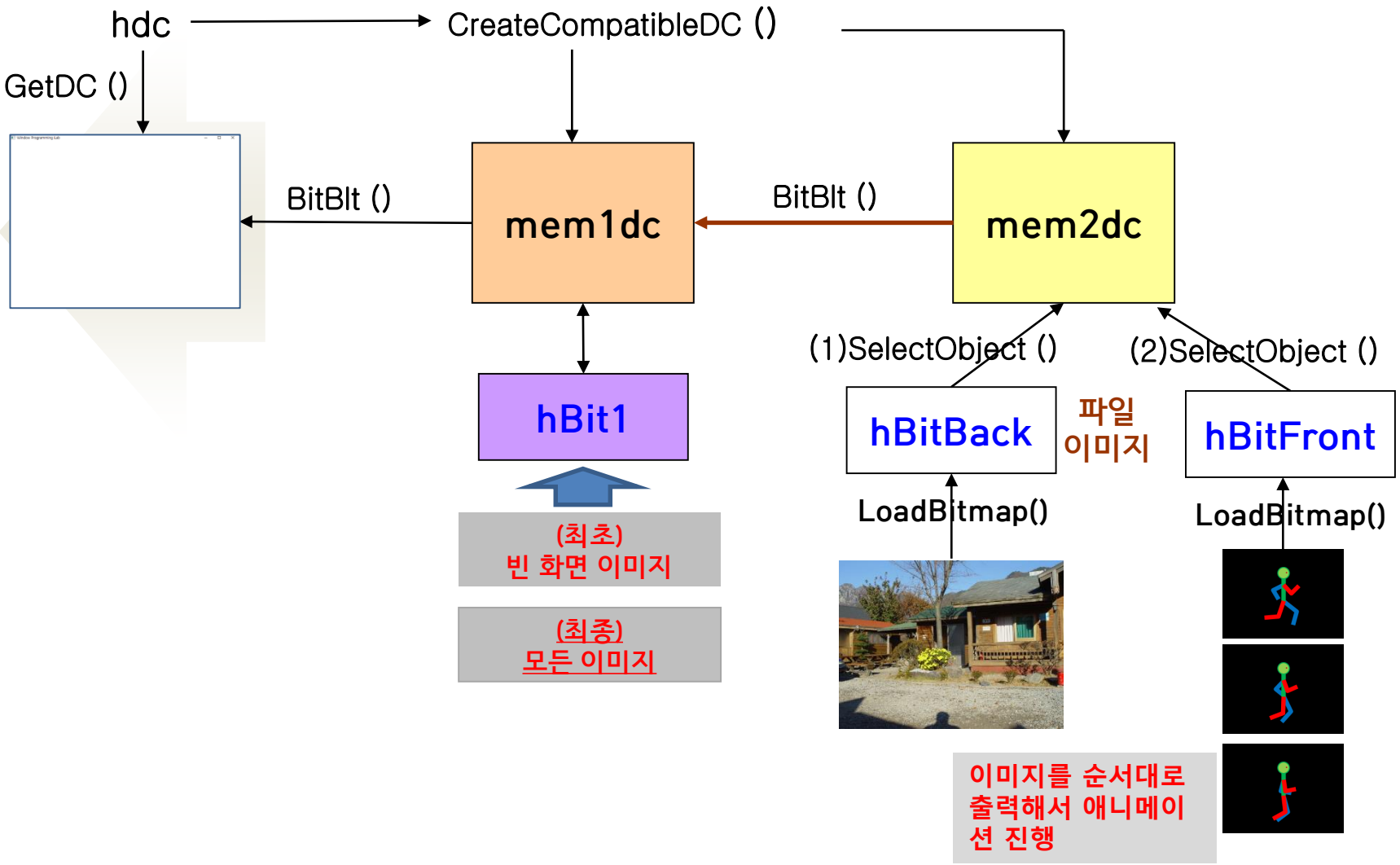
```
case WM_TIMER:
        yPos += 5;
        if (yPos > rectView.bottom)                yPos = 0;

        hdc = GetDC(hwnd);

        if (hBit1 == NULL)                                          //--- hBit1을 이미 생성하지 않았다면 새로 생성
              hBit1 = CreateCompatibleBitmap (hdc, 1024, 768);      //--- hBit1을 hdc와 호환되게 만들어준다.
                                                                    //--- 모든 이미지를 저장할 예정 (1)

        //--- hdc와 호환되는 mem1dc를  만들어준다. (2)
        mem1dc = CreateCompatibleDC (hdc);

        //---  mem1dc와 호환되는 mem2dc를 만들어준다.    (3)
        mem2dc = CreateCompatibleDC (mem1dc);

        //--- mem1dc에 hBit1 이미지를선택, mem2dc에 hBit2 이미지를선택 (4)
        oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);
        oldBit2 = (HBITMAP) SelectObject (mem2dc, hBit2);      //--- hBit2에는 배경 이미지가 저장되어 있음

        //--- mem2dc에 있는 배경그림을 mem1dc로 옮긴다. (5)
        BitBlt (mem1dc, 0, 0, 1024, 768, mem2dc, 0, 0, SRCCOPY);

        //--- mem1dc에 텍스트 출력 (6)
        SetBkMode(mem1dc, TRANSPARENT);
        TextPrint (mem1dc, 200, yPos, word);

        //--- 저장한 비트맵 핸들값을 DC에 원상복귀, 생성된 memdc들 삭제, hBit1은 삭제하면 안됨!! (7)
        SelectObject(mem2dc, oldBit2);                  DeleteDC(mem2dc);
        SelectObject(mem1dc, oldBit1);                  DeleteDC(mem1dc);
        ReleaseDC(hwnd, hdc);
        InvalidateRect (hwnd, NULL, false);    //--- 그리기 메시지를 호출: 마지막 인자는 false로 설정 (8)
        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 있음
        oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);

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

        SelectObject(mem1dc, oldBit1);
        DeleteDC(mem1dc);
        EndPaint(hwnd, &ps);
        break;

case WM_DESTROY:
        if ( hBit1) DeleteObject (hBit1);
        if ( hBit2) DeleteObject (hBit2);
        KillTimer (hwnd, 1);
        PostQuitMessage (0);
        break;
}
return DefWindowProc (hwnd, iMsg, wParam, lParam)
}
```

더블 버퍼링 사용하여 애니메이션 진행 예

- 배경을 그리고 그위에 3개 프레임으로 구성된 애니메이션을 그리기



더블 버퍼링 사용하여 애니메이션 진행 예

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

    HDC hdc, memdc;                    //--- hdc: 화면 dc, memdc: 페인트 메시지에서 사용할 메모리 dc
    HDC mem1dc, mem2dc;                //--- 타이머 메시지에서 사용할 메모리 dc
    static HBITMAP hBit1, hBitBack, hBitFront[3] , oldBit1, oldBit2;
    static int frontIndex = 0;
    static int xPos = 0, yPos = 100;
    TCHAR word[] = L"더블 버퍼링 실습: 캐릭터 애니메이션";
    static SIZE imgSize;

    switch(iMsg) {
    case WM_CREATE:
            xPos = 0;
            GetClientRect(hwnd, &rectView);
            SetTimer(hwnd, 1, 70, NULL);

            //--- hBitBack: 배경 이미지
            //--- hBitFront[3]: 애니메이션 될 3개의 이미지
            hBitBack = LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BACKGROUND));
            hBitFront[0] = LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BITMAP1));
            hBitFront[1] = LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BITMAP2));
            hBitFront[2] = LoadBitmap ( hInstance, MAKEINTRESOURCE(IDB_BITMAP3));
            frontIndex = 0;
            imgSize.x = 50;
            imageSize.y = 60;
            break;
```

더블 버퍼링 사용하여 애니메이션 진행 예

```
case WM_TIMER:
        xPos += 5;
        if (xPos > rectView.right)   xPos = 0;
        hdc = GetDC(hwnd);

        if (hBit1 == NULL)          //--- hBit1을 hdc와 호환되게 만들어준다: 여기에 이미지를 모아서 저장하려고 함
                hBit1 = CreateCompatibleBitmap (hdc, 800, 600);

        //--- mem1dc를 hdc와 호환되도록 만들어준다.  더블 버퍼로 사용할 메모리 DC
        mem1dc = CreateCompatibleDC (hdc);

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

        //--- hBit1 비트맵을 mem1dc에 선택함, mem1dc의 이미지를 hdc로 출력하려고 함
        oldBit1 = (HBITMAP) SelectObject (mem1dc, hBit1);  // mem1dc에는 hBit1

        //--- 배경 이미지를 먼저 그림: mem2dc에 있는 배경그림을 mem1dc에 옮긴다.
        oldBit2 = (HBITMAP) SelectObject (mem2dc, hBitBack);              // mem2dc에는 hBit2
        BitBlt (mem1dc, 0, 0, 1024, 768, mem2dc, 0, 0, SRCCOPY);

        //--- 캐릭터 이미지를 mem2dc에 선택한 후, 배경이 그려진 mem1dc 위에 그린다.
        oldBit2 = (HBITMAP) SelectObject (mem2dc, hBitFront[frontIndex]);
        TransparentBlt (mem1dc, xPos, yPos, imgSize.x, imageSize.y, mem2dc, 0, 0, imgSize.x, imgSize.y, RGB(0, 0, 0));

        frontIndex++;               frontIndex %= 3;          //--- 애니메이션 순서 인덱스

        //--- 저장한 비트맵 핸들값을 DC에 원상복귀, 생성한 객체들 중 사용하지 않는 객체들 삭제
        SelectObject(mem2dc, oldBit2);                        DeleteDC(mem2dc);
        SelectObject(mem1dc, oldBit1);                        DeleteDC(mem1dc);
        ReleaseDC(hwnd, hdc);
        InvalidateRect(hwnd, NULL, false);
        break;
```

더블 버퍼링 사용하여 애니메이션 진행 예

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

        memdc = CreateCompatibleDC (hdc);

        //--- hBit1에는 배경과 텍스트가 출력된 비트맵이 저장, mem1dc에 설정
        oldBit1 = (HBITMAP) SelectObject (memdc, hBit1);

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

        SelectObject(memdc, oldBit1);
        DeleteDC(memdc);
        EndPaint(hwnd, &ps);
        break;

case WM_DESTROY:
        if ( hBit2) DeleteObject (hBit2);
        KillTimer (hwnd, 1);
        PostQuitMessage (0);
        break;
}
return DefWindowProc (hwnd, iMsg, wParam, lParam)
}
```

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;

UINT g_nSpriteX;                  //--- 스프라이트 가로
UINT g_nSpriteY;                  //--- 스프라이트 세로
UINT g_nSpriteCount;              //--- 스프라이트 전체 인덱스
UINT g_nSpriteCurrent;            //--- 현재 스프라이트 인덱스

//--- 캐릭터 (스프라이트) 이미지 그리기: OnDraw 함수에서 호출함
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 );
}

//--- 이미지 그리기: WM_PAINT 메시지에서 호출함
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(더블 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

- **트윈이 있는 팩맨 만들기 (실습 3-1 활용)**
  - 배경 이미지를 출력한다.
  - 팩맨이 입을 움직이며 이동하고 있다.
    - 팩맨은 입의 움직임을 애니메이션 한다.
  - 화면의 임의의 위치에 먹이가 나타나고 먹이는 제자리에서 좌우 또는 위아래로 이동하고 있다.
    - 먹이는 한개씩 순서대로 나타나고 최대 20개가 나타나도록 한다.
  - 팩맨이 먹이를 먹으면 먹이는 사라지고 팩맨의 크기가 약간 커진다.

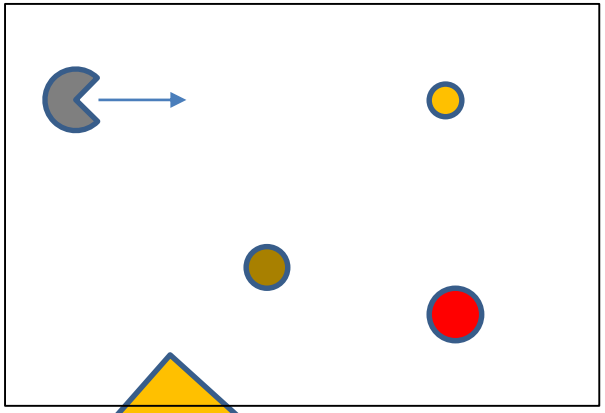
  - 키보드 명령어
    - 좌/우/상/하 키: 팩맨의 이동방향을 좌/우/상/하로 바꾼다.
    - j/J: 팩맨이 점프한다.
    - e/E: 팩맨 크기가 확대됐다가 제자리로 돌아간다..
    - s/S: 팩맨 크기가 줄어들었다가 제자리로 돌아간다.
    - t/T: 팩맨이 복사되어 트윈 팩맨이 원래의 팩맨의 뒤를 따라온다. 꼬리처럼 붙어있지 않고 약간 떨어져서 팩맨의 뒤를 따라오고 원래 팩맨이 이동하면 그 뒤를 그대로 따라온다. 최대 3개의 트윈 팩맨이 만들어진다.

  - 왼쪽 마우스:
    - 마우스가 팩맨을 클릭하면 팩맨의 다른 애니메이션이 보여지고 임의의 다른 방향으로 이동한다.
    - 잠시 후 원래의 팩맨 애니메이션이 다시 나타난다.

  - 팩맨과 배경은 비트맵 이미지를 사용한다. 먹이는 원 같은 도형으로 사용해도 된다.

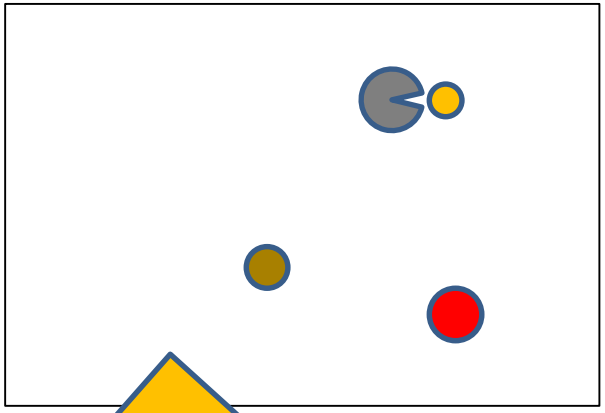
실습 5-4

1



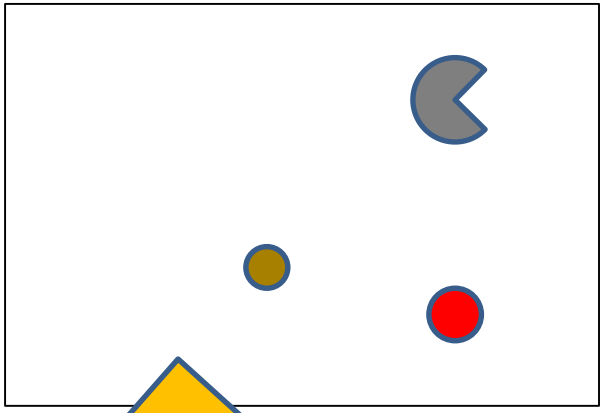
팩맨이 있고 먹이들이 있다. 팩맨은 우측으로 이동하고 있다.

2



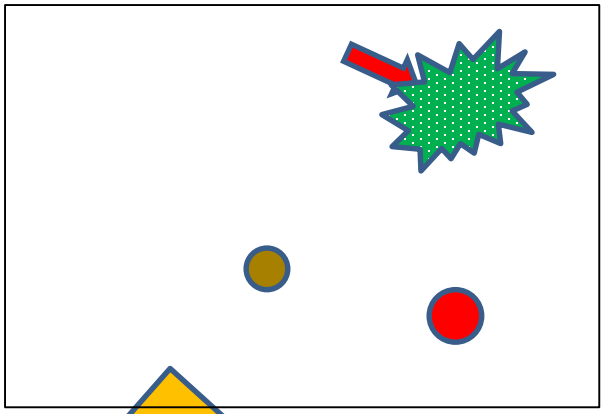
팩맨이 애니메이션 되며 이동하고 먹이를 먹는다.

3



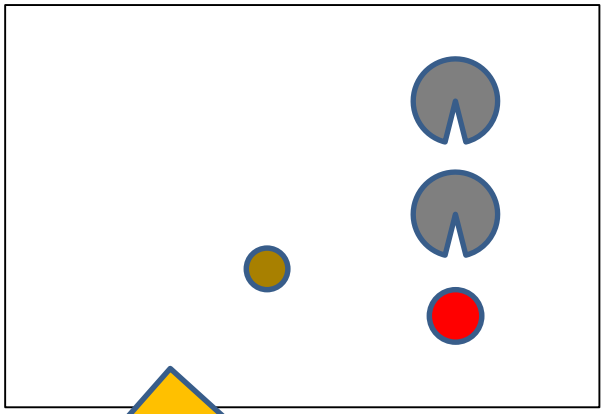
팩맨의 크기가 커졌다.

4



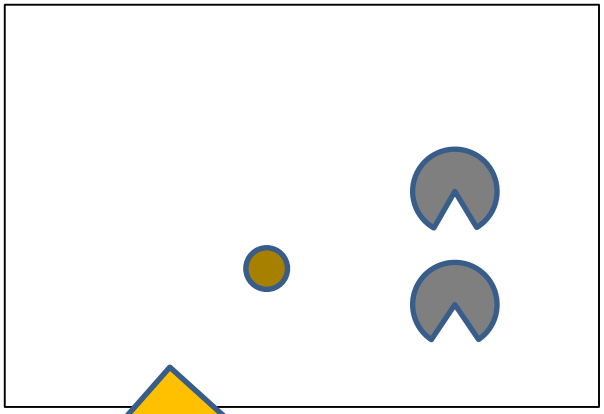
마우스로 클릭하자 다른 애니메이션이 출력된다.

5



T 명령어에 의해 트윈 팩맨이 생겼다.

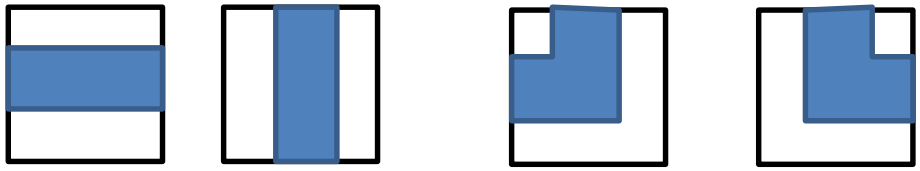
6



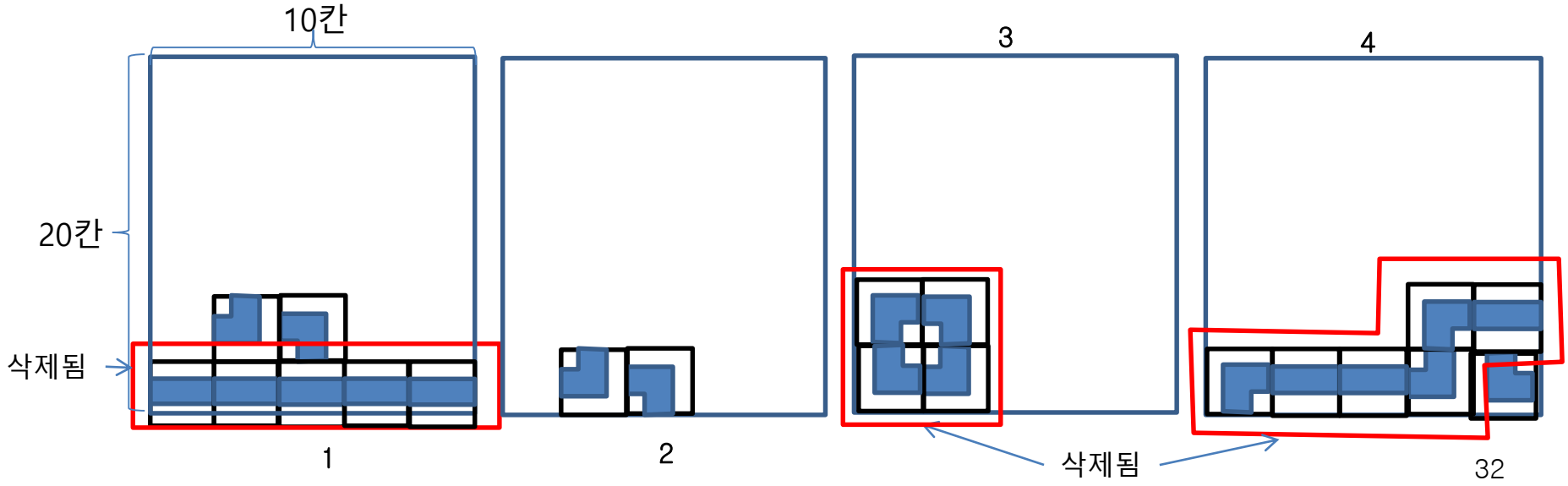
트윈 팩맨이 앞의 팩맨을 뒤따른다.

실습 5-5

- **미니 테트리스 만들기**
  - 테트리스 보드를 만든다. (폭 10칸, 높이 20칸)
  - 다음의 네 종류의 블록이 위에서 임의의 순서대로 떨어진다. (블록은 비트맵 이용)



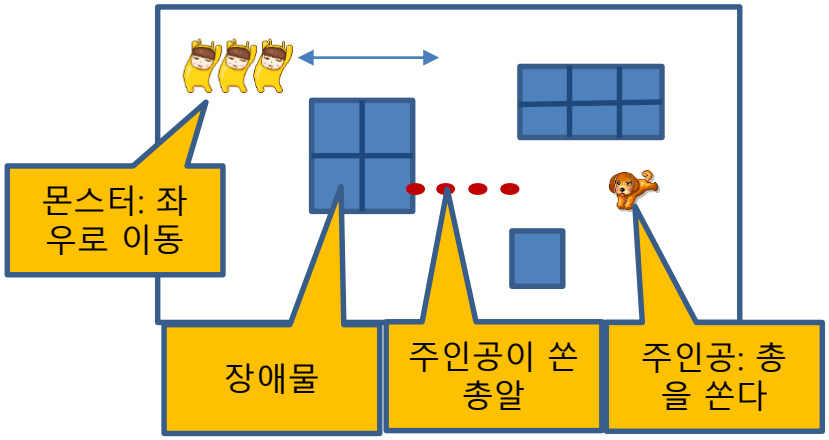
  - **블록은 space 키보드를 누르면 90도씩 반시계 방향 또는 시계방향으로 회전한다.**
  - 삭제 조건
    - ① 블록의 **줄무늬 모양이 가로로 한 줄이 되면 그 줄은 삭제된다.** ➔ 50%
    - ② 블록의 **줄무늬 모양이 네모가 되면 그 네모는 삭제된다.** ➔ 30%
    - ③ **블록의 연결된 줄무늬의 시작과 끝이 가장자리에 닿으면 그 줄은 삭제된다.** ➔ 20%
    - ④ **추가 삭제 조건: 보너스 점수**

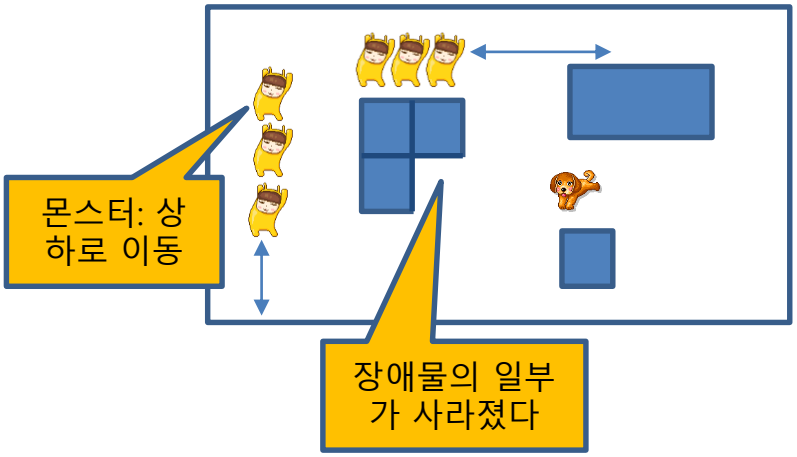


실습 5-6

- **슈팅 게임 만들기**
  - 화면에 3개 이상의 장애물을 랜덤한 위치에 놓는다.
  - 애니메이션 되는 주인공이 화면의 랜덤한 위치에서 출발하고 정해진 방향으로 자동이동한다.
  - 몬스터들이 나타나서 일정한 방향으로 이동한다.
    - 몬스터는 항상 3개가 나타나고 일렬로 나타난다.
    - 좌우로 또는 위아래로 이동하고 가장자리에 도달하면 반대 방향으로 다시 이동한다.
  - 주인공은 총을 쏴서 몬스터를 죽일 수가 있고, 몬스터는 죽으면 사라지는 애니메이션을 보이며 없어진다.
  - 장애물도 총알에 특정 숫자 이상 맞으면 사라지게 한다.
  - 캐릭터, 몬스터, 장애물은 비트맵을 사용한다.
    - 주인공 캐릭터는 장애물과 충돌체크 하고, 몬스터는 장애물 통과한다.

  - **키보드 명령**:
    - 좌우상하 키보드: 주인공의 방향을 바꾼다.
    - 스페이스: 주인공이 총을 쏘아 몬스터를 죽인다. (총알의 최대 개수는 임의로 지정한다.)
    - q/Q: 게임 종료





5장 학습내용

- **이번주 학습 내용**
  - 비트맵을 사용한 애니메이션
    - 더블 버퍼링
  - 5장의 전반주에는
    - 비트맵 다루기

- **다음주에는**
  - 대화상자와 컨트롤

- **이번주도 수고하셨습니다. 다음주에 만나요!**