Chapter 02

윈도우 기본 입출력

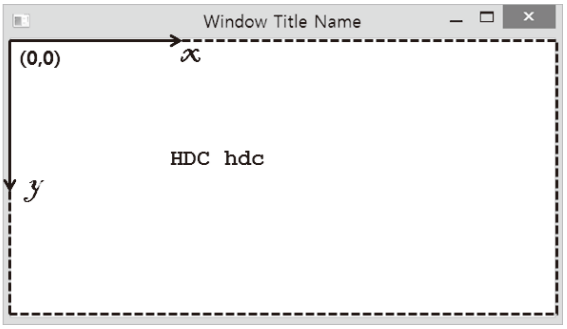
학습목표

- 윈도우 화면에 출력하기 위한 디바이스 콘텍스트의 개념을 이해할 수 있다.
- 텍스트를 출력하는 기본 함수를 사용할 수 있다.
- 기본 도형을 화면에 출력할 때 필요한 요소와 함수를 사용할 수 있다.

01 출력 영역 얻기

2장에서는 1장에서 만든 기본 윈도우 화면에 다양하게 출력하는 방법을 살펴보자. 텍스트를 출력하거나 그림을 그리려면 먼저 커널에서 출력 영역을 얻어오는 과정이 필요하다. 출력 영역을 얻어야 텍스트나 그림 등을 출력할 수 있는데, 기본적인 출력 영역은 [그림 2-1]과 같은 윈도우 프레임 안의 흰색 사각형이다.

그림 2-1 디바이스 콘텍스트와 기본 좌표계



출력 영역에서 좌표는 왼쪽 상단 모서리를 원점으로 한다. x축 값은 오른쪽으로 갈수록 커지고 y축 값은 아래쪽으로 갈수록 커진다. 하지만 좌표계의 원점이나 y축 방향은 변경할 수 있다. 기본 단위는 픽셀이지만 센티미터, 밀리미터, 인치 등 다양한 단위로 나타낼 수도 있다.

출력을 위해 얻어온 화면 영역을 디바이스 콘텍스트(device context, DC)라고 한다. 디바이스 콘텍스트는 변수에 저장해야 하는데, 이때 변수 타입은 HDC이다. HDC 타입은 메모리 영역을 관리하며 메모리 영역에는 얻어온 화면 영역에 대한 속성 값을 저장할 수 있다. 즉 HDC 타입의 변수를 hdc라고 선언한 뒤 hdc에 디바이스 콘텍스트를 얻어와 저장하고, 화면 얻어오기 함수를 이용해 얻은 화면을 hdc에 저장하면 된다.

디바이스 콘텍스트를 얻는 방법은 다양하지만 여기서는 두 가지만 소개할 것이다. BeginPaint( ) 함수를 이용하는 방법과 GetDC( ) 함수를 이용하는 방법이 그것이다.

■ BeginPaint( ) 함수로 디바이스 콘텍스트 얻기

디바이스 콘텍스트를 얻어오는 함수 BeginPaint( )

```
HDC BeginPaint(
    HWND hwnd,
    PAINTSTRUCT *lpPaint
);
```

- HWND hwnd : 생성한 윈도우의 핸들 값
- PAINTSTRUCT *lpPaint : 출력 영역에 대한 정보를 저장할 PAINTSTRUCT 구조체의 주소

BeginPaint( ) 함수는 [그림 2-1]의 점선 사각형 영역인 윈도우의 클라이언트 영역을 디바이스 콘텍스트로 할당해 핸들 값을 반환한다. BeginPaint( ) 함수로 디바이스 콘텍스트 핸들을 얻는 방법은 WM_PAINT 메시지가 발생했을 때만 사용해야 하고, 다른 메시지는 GetDC( ) 함수를 이용한다. BeginPaint( ) 함수를 이용할 때 매개변수로 주어진 PAINTSTRUCT 구조체에는 출력 영역(디바이스 콘텍스트)에 대한 상세 정보가 저장되어 돌아온다.

PAINTSTRUCT 구조체

```
typedef struct tagPAINTSTRUCT {
     HDC hdc;
     BOOL fErase;
     RECT rcPaint;
     BOOL fRestore;
     BOOL fIncUpdate;
     BYTE rgbReserved[32];
} PAINTSTRUCT, *PPAINTSTRUCT;
```

- hdc : 출력 영역(디바이스 콘텍스트)에 대한 핸들 값
- fErase : 배경 삭제 여부를 가리키는 불(bool) 값이다. 참(true)이면 응용 프로그램이 직접 배경을 삭제해야 하는데, 그렇게 하려면 윈도우 클래스를 만들 때 hbrBackground 멤버를 NULL로 채워야 함
- rcPaint : RECT 구조체로, 출력 영역의 왼쪽 상단 꼭짓점과 오른쪽 하단 꼭짓점의 좌표 저장
- fRestore, fIncUpdate, rgbReserved : 시스템에서 사용

BeginPaint( ) 함수를 이용해 디바이스 콘텍스트를 얻어와 출력을 마친 다음에는 반드시 EndPaint( ) 함수를 호출해 출력의 끝을 나타내야 한다. 이때 함수 호출에 사용하는 인수는

BeginPaint( ) 함수와 같다.

디바이스 콘텍스트 반환 함수 EndPaint( )

```
BOOL EndPaint(
     HWND hwnd,
     PAINTSTRUCT *lpPaint
);
```

- HWND hwnd : 생성한 윈도우의 핸들 값
- PAINTSTRUCT *lpPaint : 출력 영역에 대한 정보를 저장할 PAINTSTRUCT 구조체의 주소

■ GetDC( ) 함수로 디바이스 콘텍스트 얻기

디바이스 콘텍스트를 얻는 두 번째 방법은 GetDC( ) 함수를 이용하는 것이다. 인수로는 윈도우의 핸들 값을 넣고 윈도우의 클라이언트 영역에 대한 디바이스 콘텍스트를 반환한다. GetDC( ) 함수를 이용해 디바이스 콘텍스트를 얻어와 출력한 다음에는 반드시 ReleaseDC( ) 함수를 호출해 출력을 마쳤음을 알려야 한다.

디바이스 콘텍스트를 얻어오는 함수 GetDC( )

```
HDC GetDC(
    HWND hwnd // 생성한 윈도우의 핸들 값
);
```

디바이스 콘텍스트 반환 함수 ReleaseDC( )

```
int ReleaseDC(
     HWND hwnd, // 생성한 윈도우의 핸들 값
     HDC hdc // 반환하는 디바이스 콘텍스트의 핸들 값
);
```

[실습 2-1]은 BeginPaint( ) 함수를 이용해 디바이스 콘텍스트를 얻는 예제로, WM_PAINT 메시지, 즉 윈도우가 화면에 나타나는 시점에 디바이스 콘텍스트를 얻어오는 것이다. 출력 방법은 아직 다루지 않았으므로 디바이스 콘텍스트를 얻는 방법만 실습한다. 이제부터 실습에 나와 있지 않은 소스코드는 1장의 연습문제 1번과 동일하다고 보면 된다. 예를 들어 [실습 2-1]은 43행부터 시작하는데 01~42행은 1장의 연습문제 1번과 동일하다.

실습 2-1 디바이스 컨테스트 얻기

```
43  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
44                           WPARAM wParam, LPARAM lParam)
45  {
46      HDC         hdc;
47      PAINTSTRUCT ps;
48
49      switch (iMsg)
50      {
51     case WM_CREATE:
52          break;
53     case WM_PAINT:
54          hdc = BeginPaint(hwnd, &ps);
55          // 이곳에서 출력이 이뤄짐
56          EndPaint(hwnd, &ps);
57          break;
58     case WM_DESTROY:
59          PostQuitMessage(0);
60          break;
61      }
62      return DefWindowProc(hwnd, iMsg, wParam, lParam);
63  }
```

46, 47행 : 디바이스 콘텍스트를 저장할 변수 hdc와 출력 영역(디바이스 콘텍스트)에 대한 상세 정보를 저장할 ps 변수를 선언한다.

53, 54행 : WM_PAINT 발생 시 출력할 영역인 디바이스 콘텍스트를 BeginPaint( ) 함수를 이용해 얻어온다.

55행 : 출력 방법은 아직 다루지 않았으므로 일단 비워둔다.

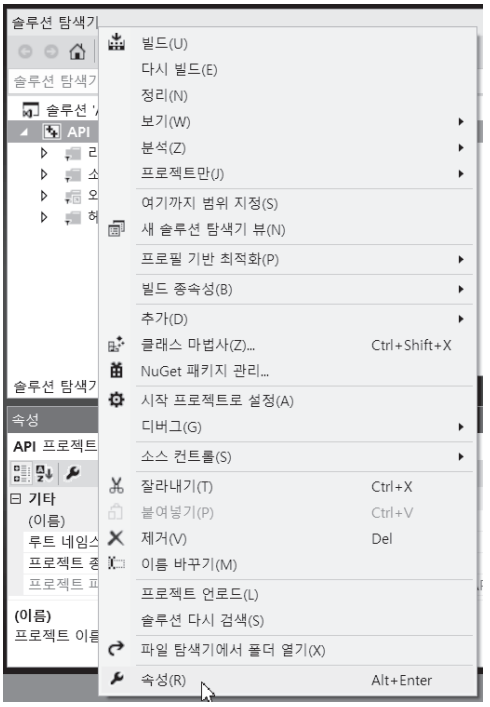
56행 : 출력을 마친 후 EndPaint( ) 함수를 이용해 디바이스 콘텍스트를 반환한다.

02 문자 집합

프로그램 입출력은 문자의 입력과 출력이 중요한 비중을 차지한다. 문자의 입출력을 위해 먼저 문자 집합에 대한 개념을 알고 있어야 한다. 프로젝트를 만들 때 선택된 문자 집합과 프로그램에서 사용하는 문자 집합이 서로 일치하지 않으면 에러가 발생하기 때문에 잘 알아둬야 한다. 1장의 연습문제 1번에서는 문자열 상수에 매크로 _T()를 적용했는데, 만약 이 매크로를 적용하지 않았다면 에러가 발생했을 것이다. 이 절에서는 프로젝트에 대한 문자 집합은 어떤 것이 있으며, 어떻게 설정하는지 살펴볼 것이다.

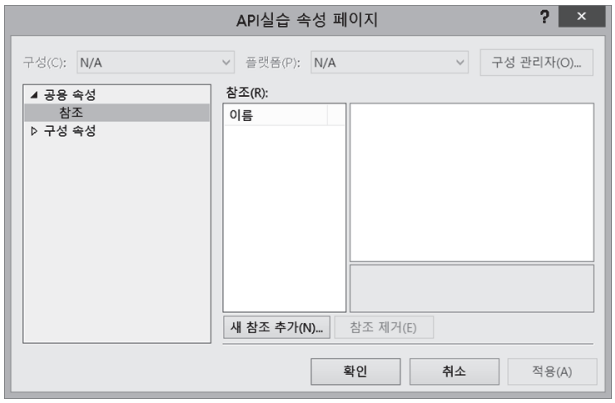
❶ [그림 2-2]와 같이 생성된 프로젝트의 이름에서 마우스 오른쪽 버튼을 누르고 [속성]을 선택한다.

그림 2-2 프로젝트의 속성 선택



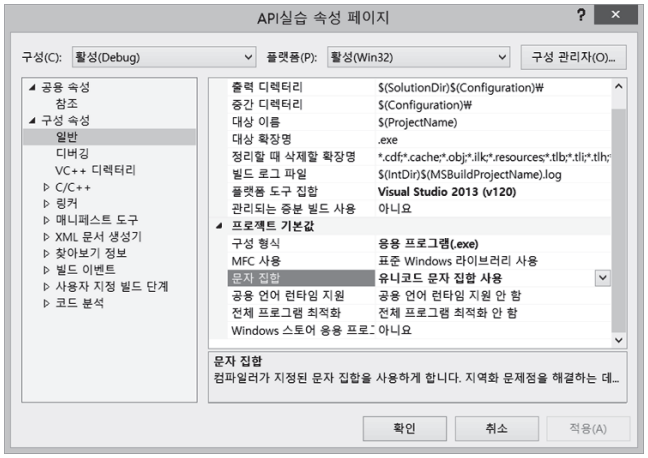
❷ 생성 중인 프로젝트의 속성 페이지가 등장한다. 속성 페이지에는 프로젝트에 대한 다양한 정보의 설정이 나와 있어 매우 유용하다. 여기서 모든 속성 정보를 다루기는 어려우므로 문자 집합에 대한 부분만 살펴본다.

그림 2-3 속성 페이지



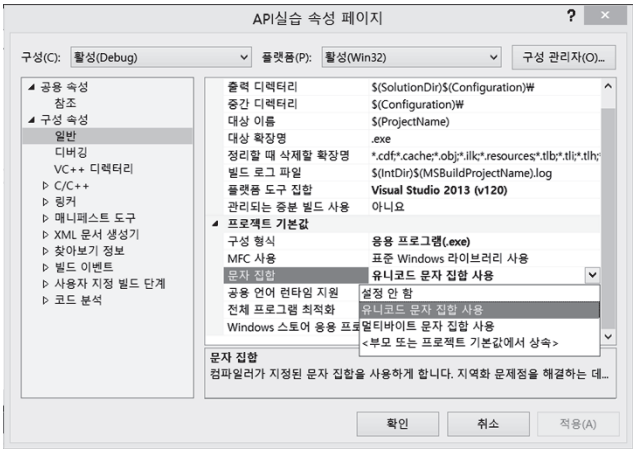
❸ 속성 페이지에서 [구성 속성]을 선택하고 하위 메뉴 중 [일반]을 선택한다. 그러면 오른쪽 분할 영역에서 [문자 집합]이라는 속성을 볼 수 있다.

그림 2-4 문자 집합의 기본 설정



❹ [문자 집합]의 기본 설정이 '유니코드 문자 집합 사용'으로 되어 있는데, 비주얼 스튜디오 2005 버전 이후부터는 문자 집합의 기본 값이 이처럼 설정되어 있다. 우리가 사용할 수 있는 다른 문자 집합은 어떤 것이 있는지 클릭해서 알아보자.

그림 2-5 사용 가능한 문자 집합



[그림 2-5]와 같이 사용 가능한 문자 집합은 세 가지, 즉 '설정 안 함', '유니코드 문자 집합 사용', '멀티바이트 문자 집합 사용'이 있다. 먼저 문자 집합에 대한 기본적인 이해가 필요하다.

키보드에 있는 모든 영문자와 특수문자는 1바이트에 저장할 수 있다. 이는 영문자와 특수문자를 다 합해도 255개를 넘지 않기 때문이다. 컴퓨터는 서양에서 처음 등장했기 때문에 1바이트에 문자 하나를 저장하는 데 전혀 문제가 없었다.

■ 문자 집합 설정 안 함

예를 들어 다음과 같이 선언하면 각 공간에 하나의 문자가 저장되는 방식이다. 문자 집합을 '설정 안 함'으로 할 경우에 해당한다.

```
char str[15] = "I love you";
```

I		I	o	v	e		y	o	u	₩0				

■ 멀티바이트 문자 집합(MBCS) 사용

한글이나 한자의 경우 문자의 개수가 매우 많아서 영어처럼 1바이트에 한 글자를 저장할 수 없다. 그래서 이러한 동양의 문자를 저장하기 위해 멀티바이트 문자 집합(MBCS)이라는 개념이 나왔다. 이는 한 글자를 저장하기 위해 2바이트 이상을 사용할 수 있게 해주는 것으로, 예를 들어 다음과 같이 선언하고 문자열을 저장하면 한글은 2바이트에, 영문자와 특수문자는 1바이트에 저장된다.

```
char str[15] = "나는 love";
```

나		는		l	o	v	e	₩0				

그런데 이때 실제 글자의 개수와 공간의 크기가 일치하지 않는 문제가 발생한다. 따라서 문자열에서 몇 번째 글자가 무엇인지 알기 위해 한글이 몇 자이고 그 외 문자가 몇 자인지 알고 있어야 한다. 예를 들어 위 문자열에서 뒤의 네 자 'love'를 삭제하려면 각 바이트에 한글이 저장되어 있는지 영어가 저장되어 있는지 알아야 한다. 멀티바이트 문자 집합은 낭비되는 공간이 없기 때문에 메모리 사용 측면에서 매우 효율적이라는 장점이 있으며, 특히 주로 영어로 된 문자열을 처리할 때 불필요하게 공간을 낭비하지 않는다.

■ 유니코드 문자 집합 사용

한글을 저장하는 데 유니코드를 이용할 수도 있다. 유니코드에서는 영어나 한글 구분 없이 모든 문자를 2바이트에 저장한다. 즉 한글은 2바이트를 모두 사용하며, 영문자나 특수문자는 1바이트만 사용하고 나머지 공간은 사용하지 않는 방식이다. 이렇게 하면 멀티바이트에서 발생하는 단점을 피할 수 있다. 문자열에서 몇 번째 글자를 얻어내기 위해 위치에 2를 곱하면 되기 때문이다. 위와 마찬가지로 문자열에서 뒤의 네 자만 삭제하기 위해 문자열 전체 공간 중 뒤에서 8바이트 앞에 NULL 문자를 넣으면 된다. 하지만 영어가 많이 사용되는 프로그램에서는 메모리의 낭비가 많아진다는 단점이 있다.

C 언어 프로그램은 WCHAR라는 자료형을 제공하는데 이는 유니코드 문자열을 위한 자료형이다. 따라서 문자열을 유니코드로 저장하려면 기존의 char가 아닌 WCHAR를 사용해야 한다.

다음과 같이 선언했다고 가정해보자.

```
WCHAR str[15] = L "나는 love";
```

나		는		₩0	l	₩0	o	₩0	v	₩0	e	₩0	₩0

여기서 문자열 상수 앞에 L을 넣은 것은 유니코드로 변환하기 위함이다. 기본 문자열 상수는 모두 멀티바이트이기 때문이다. 한글을 저장하는 것은 멀티바이트의 경우와 동일하며, 영문자나 특수문자에도 2바이트를 제공해 앞쪽에 문자를 저장하고 뒤쪽에는 NULL 문자를 저장한다.

단, 프로그램을 작성할 때 어떤 문자 집합을 사용할지는 미리 결정해야 한다. 중간에 바꾸면 코드

를 수정해야 하기 때문이다. 예를 들어 영어권 국가를 대상으로 한 프로그램이 개발 중에 변경되었다면 작성한 것을 수정해야 하는데, 이를 해결하기 위해 TCHAR를 제공하고 있다. TCHAR는 두 문자 집합의 중간 형태로 프로젝트 속성에서 설정된 문자 집합에 따라 문자열을 처리하는 부분을 멀티바이트 또는 유니코드로 자동 변경해준다. 따라서 앞의 예는 다음과 같이 선언할 수 있다.

```
TCHAR str[15] = _T("나는 love");
```

참고로 이 책에서는 자료형이 동일한 TCHAR와 tchar_t를 혼용한다. 이와 같이 작성된 코드는 프로젝트 속성의 문자 집합 설정 값에 따라 자동으로 변경된다. 멀티바이트이면 char str[15] = "나는 love";으로 변경되고, 유니코드로 설정되어 있다면 WCHAR str[15] = L"나는 love";으로 변경된다. 여기서 주의해야 할 점은 TCHAR를 이용하기 위해 소스코드 맨 윗줄에 다음과 같이 헤더 파일을 포함해야 한다는 것이다.

```
#include <TCHAR.H>
```

한 가지 팁을 주자면, 프로그램을 작성할 때 모든 문자열 상수에 매크로 _T()를 적용하는 것이 안전하다.

이제 문자열을 가리키는 포인터형 변수들 간의 차이를 알아보자. 프로그램을 작성하다 보면 다음과 같은 자료형을 자주 접하게 된다.

```
LPSTR, LPCSTR, LPTSTR, LPCTSTR, LPWSTR, LPCWSTR
```

C 언어 프로그래밍만 하다가 이와 같은 자료형을 처음 마주하면 복잡하게 느껴질 수 있지만 여러 번 사용하다 보면 쉽게 적응할 수 있을 것이다. 위의 자료형을 이미 배운 문자형과 포인터형을 이용해 나타내면 다음과 같다.

API 자료형	같은 의미의 자료형	설명
LPSTR	char *	ANSI 코드 문자열에 대한 포인터형
LPCSTR	Const char *	ANSI 코드 문자열에 대한 포인터 상수형
LPTSTR	TCHAR *	TCHAR 코드 문자열에 대한 포인터형
LPCTSTR	Const TCHAR *	TCHAR 코드 문자열에 대한 포인터 상수형
LPWSTR	WCHAR *	유니코드 문자열에 대한 포인터형
LPCWSTR	Const WCHAR *	유니코드 문자열에 대한 포인터 상수형

앞의 자료형에는 모두 LP가 붙어 있는데 이는 'long pointer'의 약자이다. long이라는 말은 과거 16비트 머신일 때 만들어진 것인데 이제는 무시해도 된다. LP가 붙어 있는 자료형은 모두 포인터형이라고 생각하면 된다. 그다음에 C가 붙은 것과 붙지 않은 것이 있는데 C가 붙은 자료형은 Const, 즉 상수를 의미하기 때문에 이렇게 선언된 포인터 변수의 값은 수정할 수 없다. 그리고 STR가 모두 들어 있는데 이것은 문자열이라는 의미이다. 즉 앞의 자료형은 공통적으로 문자열을 가리키는 포인터형이다. W가 붙은 자료형은 유니코드 문자열에 대한 포인터형이다. 마지막으로 T는 유니코드 또는 멀티바이트 둘 다를 지원한다는 의미이다. 만약 환경이 멀티바이트이면 멀티바이트로, 유니코드이면 유니코드 자료형으로 변하게 된다. T가 붙은 자료형에 문자열을 저장할 때는 매크로 _T()를 이용하기 바란다.

03 텍스트 출력하기

이 절에서는 텍스트를 출력하는 방법을 살펴보자. 텍스트를 출력할 때 주로 사용하는 함수는 TextOut( )과 DrawText( )인데 이 중 TextOut( ) 함수가 좀 더 간단하다.

3.1 텍스트 출력 함수 : TextOut( )

TextOut( ) 함수는 특정 위치에 문자열을 출력하기 위한 것으로 문자 집합에 따라 TextOutA나 TextOutW로 변환된다. 만약 멀티바이트 문자 집합을 사용하고 있다면 TextOutA를 이용하고, 유니코드를 사용하고 있다면 TextOutW를 이용한다.

텍스트 출력 함수 TextOut( )

```
BOOL TextOut(HDC hdc, int x, int y, LPCTSTR lpString, int nLength);
```

- HDC hdc : BeginPaint( )나 GetDC( ) 함수로 얻어온 화면 영역
- int x, y : 텍스트를 출력할 위치의 x 좌표와 y 좌표
- LPCTSTR lpString : 출력할 텍스트 문자열
- int nLength : 출력할 텍스트 길이

예를 들어 TextOut(hdc, 0, 0, _T("Hello 안녕"), _tcslen(_T("Hello 안녕")));은 화면의 (0, 0) 위치에 'Hello 안녕'을 출력한다. 출력 문자열의 길이는 _tcslen( ) 함수를 이용하는데, 그 이유는 문자 집합에 따라 위의 문자열 길이가 달라지기 때문이다. 멀티바이트이면 한글을 두 자씩 계산해 총 열 자가 되지만, 유니코드이면 문자의 종류에 상관없이 모두 한 자로 계산하기 때문에 여덟 자가 된다. 이와 같이 프로그램이 문자 집합 종류의 영향을 받으면 일관성이 없어서 프로그래밍을 어렵게 만든다. 그러므로 문자 집합 종류와 무관한 _tcslen( ) 함수를 이용한다.

[실습 2-2]는 출력할 영역을 얻어온 다음 TextOut( )을 이용해 화면의 (0, 0)에 'Hello 안녕'을 출력하는 예제이다. 진하게 표시한 부분은 [실습 2-1]에 새로 추가한 것인데, 프로그래밍 학습

에서는 이렇게 달라진 부분을 살펴보는 것이 아주 중요하다.

실습 2-2 윈도우에 'Hello 안녕' 출력하기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                               WPARAM wParam, LPARAM lParam)
46  {
47      HDC            hdc;
48      PAINTSTRUCT    ps;
49
50      switch (iMsg)
51      {
52      case WM_CREATE:
53           break;
54      case WM_PAINT:
55           hdc = BeginPaint(hwnd, &ps);
56           TextOut(hdc, 100, 100, _T("Hello 안녕"), _tcslen(_T("Hello 안녕")));
57           EndPaint(hwnd, &ps);
58           break;
59      case WM_DESTROY:
60           PostQuitMessage(0);
61           break;
62      }
63      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
64  }
```

47, 48행 : HDC 타입의 hdc 변수와 PAINTSTRUCT 타입의 구조체 변수 ps를 만든다. ps는 화면에 관한 다양한 정보를 얻어오는 데 사용하지만 여기서는 사용하지 않는다.

52, 53행 : 1장에서 설명한 것처럼 WndProc( ) 함수에 다양한 메시지가 전달된다. 먼저 도착하는 메시지는 윈도우가 처음 만들어졌을 때 발생하는 WM_CREATE이다. 여기서는 WM_CREATE에 대해 어떤 일도 하지 않고 switch 문을 끝냈다.

54~58행 : 윈도우가 화면에 등장하면서 WM_PAINT 메시지가 발생한다. 이때 디바이스 콘텍스트를 얻어온 다음 TextOut( ) 함수를 이용해 화면인 hdc에 'Hello 안녕'을 출력한다. 출력을 마친 뒤에는 출력을 마쳤음을 알린다.

59~61행 : 윈도우의 〈닫기〉 버튼을 누르거나 종료하면 WM_DESTROY 메시지가 발생한다. 여기서는 PostQuitMessage(0)을 수행하고 있는데 이는 WinMain( )의 while 문에 있는 GetMessage( )

함수가 0을 반환하게 한다. 즉 while 문이 거짓이므로 루프가 종료되고 WinMain( )이 끝난다. 이는 전체 프로그램을 종료시킨다.

3.2 텍스트 출력 함수 : DrawText( )

DrawText( ) 함수는 TextOut( )처럼 텍스트를 화면에 출력하지만 한 점의 좌표를 주고 출력하는 것이 아니라 박스 영역을 지정하고 그 안에 출력한다. 사용하는 매개변수는 TextOut( ) 함수와 대체로 유사하지만, 영역의 좌표를 전달하는 매개변수가 있다는 것과 영역의 어느 위치에 출력할지를 알리는 플래그 값이 있다는 것이 다르다.

텍스트 출력 함수(영역 지정) DrawText( )

```
int DrawText(HDC hdc, LPCTSTR lpString, int nLength, LPRECT lpRect, UINT Flags);
```

- HDC hdc : 화면 영역을 가리키는 변수
- LPCTSTR lpString : 출력할 텍스트 문자열
- int nLength : 출력할 문자열 길이
- LPRECT lpRect : LPRECT는 RECT *와 같은 것으로, 문자열을 출력할 박스 영역의 좌표가 저장된 RECT 타입 변수의 주소 값
- UINT Flags : 영역의 어느 위치에 어떻게 출력할지를 알려주는 플래그 값으로 미리 정의된 상수 사용

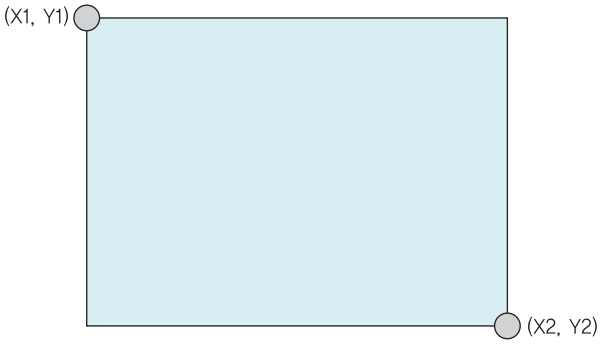
박스 영역을 지정할 때는 RECT 구조체를 사용한다.

RECT 구조체

```
typedef struct tagRECT {
LONG left; // X1
LONG top; // Y1
LONG right; // X2
LONG bottom; // Y2
} RECT;
```

박스 영역을 그림으로 나타내면 [그림 2-6]과 같은 직사각형이 된다. 직사각형의 좌표를 저장하려면 대각선으로 마주 보는 좌표가 2개 필요하다. RECT 구조체에서는 왼쪽 상단 꼭짓점과 오른쪽 하단 꼭짓점 좌표를 이용해 RECT 변수를 만든다. 이를 위해 left, top, right, bottom이라는 4개의 정수 필드를 사용한다.

그림 2-6 RECT 영역



X1의 값은 가장 왼쪽 경계를 나타내므로 left에 저장하고, Y1의 값은 가장 위쪽 경계를 나타내므로 top에 저장한다. 마찬가지로 right에는 X2를, bottom에는 Y2를 저장한다. Flags는 박스에 문자열을 어떤 형태로 어느 위치에 출력할지를 알려주는 상수이다. 플래그로 사용할 수 있는 값에는 다음과 같은 것들이 있다.

DrawText( ) 함수의 플래그 값

- DT_SINGLELINE : 박스 영역에 한 줄로 출력
- DT_LEFT : 박스 영역에서 왼쪽 정렬
- DT_CENTER : 박스 영역에서 가운데 정렬
- DT_RIGHT : 박스 영역에서 오른쪽 정렬
- DT_VCENTER : 박스 영역의 상하에서 가운데에 출력
- DT_TOP : 박스 영역의 위쪽에 출력
- DT_BOTTOM : 박스 영역의 아래쪽에 출력
- DT_CALCRECT : 문자열을 출력한다면 차지할 공간의 크기 측정
- DT_NOCLIP : 문자열이 길어서 사각형을 넘어가도 모두 출력

박스 중앙에 한 줄로 출력할 때는 DT_SINGLELINE | DT_CENTER | DT_VCENTER를 사용한다. 만약 DT_SINGLELINE을 사용하지 않으면 개행 문자인 '\n'에서 다음 줄로 행을 바꿔 출력한다. DT_TOP, DT_BOTTOM, DT_VCENTER는 DT_SINGLELINE과 같이 사용해야 정렬이 된다. 다음은 DrawText( ) 함수를 이용해 'Hello 안녕'을 가상의 사각형 중앙에 출력하는 예제이다.

실습 2-3 DrawText( ) 함수 이용하기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                              WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49      RECT            rect;
50
51      switch (iMsg)
52      {
53      case WM_CREATE:
54          break;
55      case WM_PAINT:
56          hdc = BeginPaint(hwnd, &ps);
57          rect.left = 50;
58          rect.top = 40;
59          rect.right = 200;
60          rect.bottom = 120;
61          DrawText(hdc, _T("HelloWorld"), 10, &rect,
62              DT_SINGLELINE | DT_CENTER | DT_VCENTER);
63          EndPaint(hwnd, &ps);
64          break;
65      case WM_DESTROY:
66          PostQuitMessage(0);
67          break;
68      }
69      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
70  }
```

47~49행 : hdc와 ps 변수는 앞서 설명한 내용과 같다. rect 변수는 RECT 구조체로 텍스트 출력 영역을 지정한다.

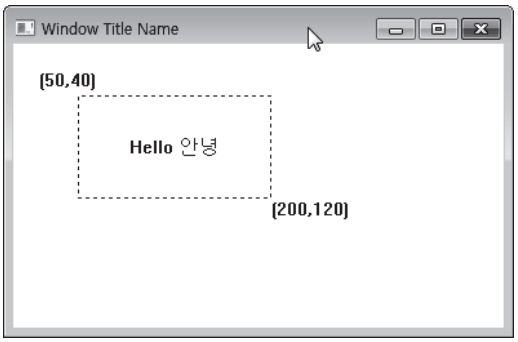
53, 54행 : [실습 2-2]와 같이 먼저 오는 메시지는 WM_CREATE이다. 여기서도 break 문으로 switch 문을 빠져나가 WndProc( )를 마친다.

55, 56행 : WM_PAINT 메시지가 오면 BeginPaint( ) 함수로 화면 영역을 얻어온다.

57~60행 : rect 변수의 left, top, right, bottom 필드에 값 50, 40, 200, 120을 저장한다.

61, 62행 : DrawText( ) 함수를 이용해 rect 영역에 'Hello 안녕'을 출력한다. 출력할 때 사용하는 플래그의 조합은 rect 영역의 중앙에 한 줄로 출력한다는 것을 의미한다.

63, 64행 : EndPaint( )를 이용해 출력을 마쳤음을 알린다.



실행 화면에 나타난 좌표와 점선 사각형은 박스 영역을 표시하기 위한 것으로 실제 출력 화면에는 나타나지 않는다. 이 박스의 중앙에 텍스트를 한 줄로 출력한다.

04 키보드 메시지 처리하기

지금까지는 문자열 상수를 화면에 일방적으로 출력하는 방법을 배웠는데, 이 절에서는 키보드로 입력한 정보를 받는 방법을 살펴보자. 키보드에서 발생하는 주요 이벤트는 키를 누르거나 떼는 것이다. 키보드에서 이벤트가 발생하면 윈도우 프로시저인 WndProc( )에 [그림 2-7]과 같은 키보드 메시지, 가상키 값, 부가 정보와 같은 내용이 전달된다. 그러면 윈도우 프로시저에서는 다음과 같은 변수를 기반으로 메시지를 처리한다.

그림 2-7 키보드에서 메시지와 함께 전달되는 정보

전달 변수	내용	값
iMsg	키보드 메시지	WM_KEYDOWN WM_CHAR WM_KEYUP
wParam	가상키 값	A, B, C, … 1, 2, 3, … !, @, #, … VK_BACK VK_RETURN VK_LEFT …
lParam	부가 정보	스캔 코드 키 반복 횟수 확장 키 코드 이전 키 상태

키보드에 의해 다음과 같은 메시지가 발생한다.

- WM_KEYDOWN : 키보드의 키를 눌렀을 때 발생
- WM_CHAR : 키보드의 문자키를 눌렀을 때 발생
- WM_KEYUP : 키보드에서 키를 눌렀다가 뗐을 때 발생

WM_KEYDOWN 메시지는 키보드의 키를 눌렀을 때 발생하고, WM_CHAR 메시지는 키보드의 문자키를 눌렀을 때 발생한다. 그렇다면 A 키를 눌렀을 때 어떤 메시지가 발생할까? WM_KEYDOWN 메시지가 발생하고 다음으로 WM_CHAR 메시지가 발생할 것이다. 또한 WM_KEYUP은 키를 눌렀다가 뗐을 때 발생하는 메시지이다. 이 밖에도 여러 가지 메시지가 있지만 이 세 가지를 주로 사용한다.

[그림 2-7]과 같이 키보드 메시지와 함께 wParam 변수에는 이벤트를 발생시킨 키의 가상키 값이 저장된다. 즉 영문자 또는 숫자를 나타내는 문자나 특수문자가 아스키코드 값으로 전달되는 것이다. 그러나 Enter, Esc, Back Space 방향키의 경우에는 [표 2-1]과 같은 가상키의 코드 값으로 전달된다.

표 2-1 가상키의 종류

가상키	내용	가상키	내용
VK_CANCEL	Ctrl + Break	VK_END	End
VK_BACK	Back Space	VK_HOME	Home
VK_TAB	Tab	VK_LEFT	←
VK_RETURN	Enter	VK_UP	↑
VK_SHIFT	Shift	VK_RIGHT	→
VK_CONTROL	Ctrl	VK_DOWN	↓
VK_MENU	Alt	VK_INSERT	Insert
VK_CAPITAL	Caps Lock	VK_DELETE	Delete
VK_ESCAPE	Esc	VK_F1 ~ VK_F10	F1 ~ F10
VK_SPACE	Space Bar	VK_NUMLOCK	Num Lock
VK_PRIOR	Page Up	VK_SCROLL	Scroll Lock
VK_NEXT	Page Down		

마지막으로 lParam에는 스캔 코드, 키 반복 횟수 같은 부가 정보가 저장 및 전달되는데 이에 대

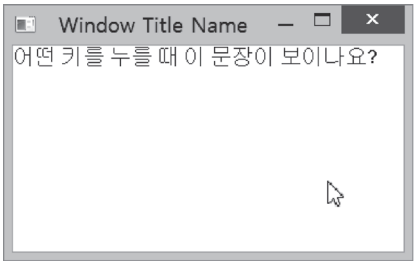
한 자세한 설명은 생략한다. 이어서 메시지를 처리하는 예제를 살펴보자.

실습 2-4 WM_CHAR 메시지 처리하기

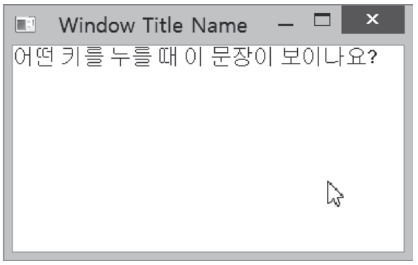
```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                               WPARAM wParam, LPARAM lParam)
46  {
47      HDC   hdc;
48
49      switch (iMsg)
50      {
51      case WM_CREATE:
52           break;
53      case WM_CHAR:
54           hdc = GetDC(hwnd);
55           TextOut(hdc, 0, 0, _T("어떤 키를 누를 때 이 문장이 보이나요?"), 21);
56           ReleaseDC(hwnd, hdc);
57           break;
58      case WM_DESTROY:
59           PostQuitMessage(0);
60           break;
61      }
62      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
63  }
```

54, 56행 : GetDC( ) 함수를 이용해 HDC를 얻어온다. 앞에서는 WM_PAINT 메시지를 처리하는 case 문에서 BeginPaint( )를 이용해 HDC를 얻어오고, EndPaint( )를 이용해 출력을 마쳤다. 하지만 WM_PAINT가 아닌 다른 메시지에서는 GetDC( ) 함수를 이용해야 한다. 출력을 마칠 때는 ReleaseDC( ) 함수를 이용한다. 이처럼 사용하는 함수는 달라도 기능은 같다. 다른 윈도우로 가렸다가 나타내거나, 최소화했다가 원상 복귀했을 때 WM_PAINT 메시지를 처리하는 case 문에서 출력한 내용은 계속 볼 수 있다. 하지만 GetDC( ) 함수로 얻어와 화면에 출력하는 case 문의 내용은 모두 사라진다.

55, 56행 : 키보드를 눌렀을 때 hdc에 화면 영역을 얻어오고 화면의 (0, 0) 좌표에 '어떤 키를 누를 때 이 문장이 보이나요?'를 출력한다. 키보드의 어떤 키를 눌러도 결과는 같다. 눌린 키가 무엇이든 상관없이 모두 '어떤 키를 누를 때 이 문장이 보이나요?'를 출력한다.



A 키를 누른 결과



Enter 키를 누른 결과

다음은 눌린 키보드의 문자키를 알아내어 화면에 문자를 출력하는 예제이다.

실습 2-5 입력 문자 처리하기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                           WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      static TCHAR    str[100];
49
50      switch (iMsg)
51      {
52      case WM_CREATE:
53          break;
54      case WM_CHAR:
55          hdc=GetDC(hwnd);
56          str[0] = wParam;
57          str[1] = NULL;
58          TextOut(hdc, 0, 0, str, _tcslen(str));
59          ReleaseDC(hwnd, hdc);
60          break;
61      case WM_DESTROY:
62          PostQuitMessage(0);
63          break;
64      }
65      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
66  }
```

48행 : 문자를 저장하기 위한 변수 str를 선언했다. 2절에서 설명한 바와 같이 TCHAR 자료형으로 선

언해 유니코드 환경이든 멀티바이트 환경이든 모두 가능하게 했다. 배열의 크기를 반드시 100으로 할 필요는 없으며 여기서는 2로 설정해보자.

54, 55행 : WM_CHAR는 문자키를 누르면 발생하는 메시지이다. case 문에서 WM_KEYDOWN 메시지를 처리해도 되지만, 화면에 표시할 문자키에 대한 처리이므로 WM_CHAR 메시지 처리로 했다. 키보드의 문자키를 눌렀을 때 hdc 변수에 화면을 얻어온다.

56~58행 : 미리 선언한 str 문자열에 입력된 문자를 저장해 문자열을 만든다. 코드를 보고 짐작할 수 있겠지만 키를 눌렀을 때 가상키 값은 wParam에 저장되어 온다. 즉 WM_CHAR 메시지의 번호는 iMsg에 저장되고 문자의 가상키 값은 wParam에 저장되어 온다. A 키를 눌렀다면 str에는 'a'라는 문자열이 저장되므로 화면의 (0, 0) 좌표에 'a'를 출력하는 것이다. _tcslen( )은 TCHAR 문자열의 길이를 알려주는 함수로 문자 집합이 어떤 것으로 결정되든 상관없이 문자열의 길이를 알려준다.

[실습 2-4]와 같은 이유로 [실습 2-5]에서 출력된 내용도 다른 윈도우로 가렸다가 나타내거나 최소화했다가 원상 복귀하면 모두 사라진다. 생성한 윈도우가 최소화되었다가 나타나면 화면을 깨끗이 지우고 WM_PAINT 메시지를 발생시키기 때문이다. [실습 2-5]에서는 마지막에 누른 문자 하나만 보여주고 그 전에 입력한 문자는 사라진다.

이번에는 입력한 문자를 계속 저장했다가 화면에 문자열을 출력하는 간단한 텍스트 입력 예제를 살펴보자.

실습 2-6 입력 문자열 처리하기

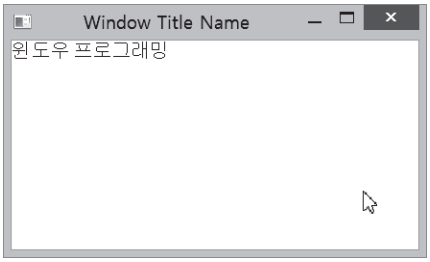
```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                           WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      static TCHAR    str[100];
49      static int      count;
50
51      switch (iMsg)
52      {
53      case WM_CREATE:
54          count = 0;
55          break;
```

```
56      case WM_CHAR:
57          hdc=GetDC(hwnd);
58          str[count++] = wParam;
59          str[count] = NULL;
60          TextOut(hdc, 0, 0, str, _tcslen(str));
61          ReleaseDC(hwnd, hdc);
62          break;
63      case WM_DESTROY:
64          PostQuitMessage(0);
65          break;
66      }
67      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
68  }
```

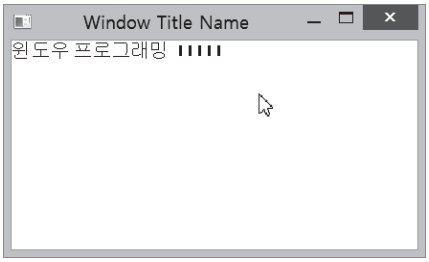
49행 : [실습 2-5]와 달리 str 문자열과 count 정수를 static 변수로 선언했다. 메시지가 발생했을 때 저장한 내용을 다음 메시지가 발생해도 계속 유지하기 위해서이다. 메시지 하나를 처리하고 나면 그 메시지 때문에 저장한 내용은 static 변수나 전역변수에 저장하지 않는 한 모두 잃게 된다. 메시지 처리를 마치면 WndProc( ) 함수가 종료되기 때문이다.

54행 : 윈도우가 처음 만들어졌을 때 count 변수를 0으로 초기화한다. count 변수는 입력된 문자의 개수를 알려준다.

56~62행 : 문자키를 누를 때마다 WM_CHAR 메시지가 발생하고 입력된 문자는 wParam에 저장되어 오는데, 이를 str[count]에 저장하고 count 변수를 증가시킨다. 그런 다음 (0, 0) 좌표에 str를 출력한다.



일반 문자키 입력 후 화면



Enter, Back Space 키 입력 후 화면

[실습 2-6]을 실행한 후 문자를 입력하면 왼쪽 상단부터 입력한 문자가 순서대로 나타난다. 하지만 Enter나 Back Space 키 같은 컨트롤키 입력에는 작동하지 않는다.

[실습 2-6]의 경우 기본적인 기능을 수행하지만 몇 가지 문제가 있다. 이 중 가장 시급한 문제 두 가지와 그 해결책을 알아보자.

❶ 윈도우를 최소화했다가 다시 나타내면 출력한 내용이 사라진다.
→ WM_PAINT 메시지가 발생하기 때문이므로 WM_PAINT 메시지를 처리하는 case 문을 만든다.

❷ Enter나 Back Space 같은 키를 처리할 수 없다.
→ 가상키 테이블을 사용하거나 DrawText( ) 함수를 이용해 처리한다.

❶번 문제는 윈도우를 최소화했다가 원상 복귀하면 화면에 출력된 내용이 모두 사라진다는 것인데, 이와 같은 상황은 WM_PAINT 메시지가 발생하면서 화면에 이미 출력된 내용을 무효화, 즉 전체 내용을 삭제하기 때문이다. WM_PAINT 메시지가 발생하면 화면의 내용이 모두 지워지고, WndProc( ) 함수의 WM_PAINT 메시지를 처리하는 case 문에서 출력하는 것만 남는다. 따라서 이전에 출력한 내용이 화면에 계속 남아 있도록 하려면 출력한 내용을 WM_PAINT 메시지에서 다시 출력해야 한다.

❷번 문제는 Enter나 Back Space 키가 처리되지 않는다는 것이다. 물론 Tab이나 Delete 키도 사용할 수 없다. (두 경우 모두 원리가 같으므로 Tab이나 Delete 키에도 쉽게 응용할 수 있을 것이다.) Enter나 Back Space 키를 눌렀을 때 이에 대한 가상키 코드 값이 wParam에 저장되고, 이것은 단지 하나의 문자로 문자 배열 str에 저장된다. TextOut( ) 함수는 출력하는 문자 배열에 Enter나 Back Space 문자가 있어도 무시하고 그냥 출력만 한다.

다음은 WM_PAINT 메시지가 발생해 화면이 삭제되는 문제를 해결하는 예제이다.

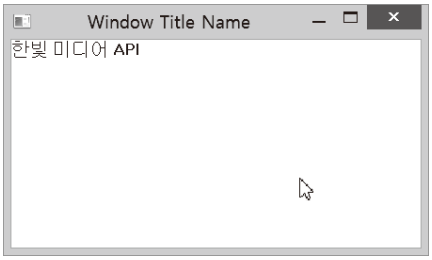
실습 2-7 WM_PAINT 메시지 처리하기

```
45  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
46                           WPARAM wParam, LPARAM lParam)
47  {
48      HDC             hdc;
49      PAINTSTRUCT     ps;
50      static TCHAR    str[100];
51      static int      count;
52
53      switch (iMsg)
```

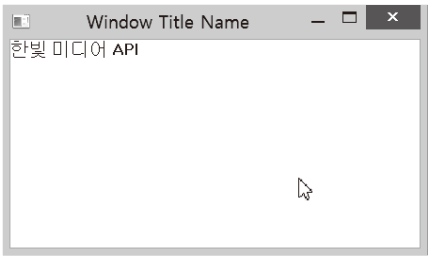
```
54      {
55      case WM_CREATE:
56          count = 0;
57          break;
58      case WM_PAINT:
59          hdc = BeginPaint(hwnd, &ps);
60          TextOut(hdc, 0, 0, str, _tcslen(str));
61          EndPaint(hwnd, &ps);
62          break;
63      case WM_CHAR:
64          hdc=GetDC(hwnd);
65          str[count++] = wParam;
66          str[count] = NULL;
67          TextOut(hdc, 0, 0, str, _tcslen(str));
68          ReleaseDC(hwnd, hdc);
69          break;
70      case WM_DESTROY:
71          PostQuitMessage(0);
72          break;
73      }
74      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
75  }
```

50행 : str 배열 변수가 static으로 선언되어 이전에 저장한 내용을 모두 그대로 유지한다.

58~62행 : [실습 2-6]과 비슷하지만 WM_PAINT 메시지를 처리하는 case 문이 추가되었다. 즉 WM_PAINT 메시지가 발생해 화면의 내용이 모두 사라졌다면 str에 저장된 문자열을 (0, 0) 좌표에 출력하는 것이다.



초기 텍스트 입력 화면



최소화했다가 원상 복귀한 화면

WM_PAINT 메시지가 발생했을 때는 BeginPaint( ) 함수를 이용해 출력 영역을 얻어오고 다른 메시지는 GetDC( ) 함수를 이용한다. 그리고 화면 출력을 마쳤을 때는 각각 EndPaint( )와 ReleaseDC( ) 함수를 이용해 알린다.

[실습 2-7]에서는 텍스트를 출력하는 부분이 두 곳이다. 그렇다면 TextOut( ) 호출 두 번을 하나로 모을 수 없을까? WM_CHAR 메시지에서 출력하는 부분을 삭제하고 WM_PAINT 메시지를 발생시켜 출력하면 가능하다. 화면의 특정 영역을 수정하는 InvalidateRgn( ) 함수가 있는데, 이것은 클라이언트의 특정 영역을 무효화하므로 다시 그리고 싶을 때 발생시킨다. 특정 영역에 WM_PAINT 메시지를 강제로 발생시키는 효과가 있다.

화면 영역 수정 함수 `InvalidateRgn()`

```
BOOL InvalidateRgn(
     HWND hWnd,
     HRGN hRgn,
     BOOL bErase
);
```

- `hWnd` : 수정 영역이 포함된 윈도우의 핸들 값
- `hRgn` : 수정 영역에 대한 핸들 값으로 NULL을 주면 클라이언트 영역 전체를 수정하는데, 여기서는 주로 NULL 값 이용
- `bErase` : 수정 영역을 모두 삭제하고 다시 그릴지, 수정 부분만 추가할지를 나타내는 불(bool) 값으로, 참(true)이면 모두 삭제하고 거짓(false)이면 삭제하지 않음

[실습 2-8]에서는 입력 문자를 저장하는 곳과 출력하는 곳을 나눈다. 그리고 문자열 출력을 위한 TextOut( ) 함수 호출을 WM_PAINT 메시지 처리에서만 한다.

실습 2-8 문자 저장과 출력 구분하기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                       WPARAM wParam, LPARAM lParam)
46  {
47       HDC            hdc;
48       PAINTSTRUCT    ps;
49       static TCHAR   str[100];
50       static int     count;
51
```

```
52      switch (iMsg)
53      {
54      case WM_CREATE:
55          count = 0;
56          break;
57      case WM_PAINT:
58          hdc = BeginPaint(hwnd, &ps);
59          TextOut(hdc, 0, 0, str, _tcslen(str));
60          EndPaint(hwnd, &ps);
61          break;
62      case WM_CHAR:
63          str[count++] = wParam;
64          str[count] = NULL;
65          InvalidateRgn(hwnd, NULL, TRUE);
66          break;
67      case WM_DESTROY:
68          PostQuitMessage(0);
69          break;
70      }
71      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
72  }
```

59행 : 문자 배열 str에 저장된 내용을 출력하는 일은 WM_PAINT 메시지 처리 부분이 담당한다. 문자가 키보드를 통해 하나씩 들어올 때마다 WM_CHAR 메시지 처리에서 str에 저장한 후, 화면에 출력하도록 InvalidateRgn( ) 함수를 호출한다.

63, 64행 : [실습 2-7]과 달리 WM_CHAR 메시지 처리 부분에서는 키보드로 입력한 문자를 str 문자 배열에 저장만 하고 출력하지는 않는다. 대신 InvalidateRgn( ) 함수를 호출한다.

65행 : InvalidateRgn( ) 함수 호출에서는 세 번째 매개변수가 TRUE이므로 윈도우 화면 전체를 무효화해 화면의 내용을 삭제하고 WM_PAINT 메시지를 발생시킨다.

그러나 [실습 2-8]도 문제가 있는데, 화면 전체를 강제로 지우기 때문에 출력 내용이 많으면 화면이 깜박거린다. 이는 비트맵을 배운 다음 더블 버퍼링이라는 기법을 사용하거나, 세 번째 매개변수를 FALSE로 변경해 화면의 변경된 부분만 수정하도록 해결할 수 있다.

또한 Enter나 Back Space 키 또는 방향키처럼 많이 사용하지만 문자키가 아닌 컨트롤키 처리에도 문제가 있다. 컨트롤키는 볼 수 있는 문자로 표시할 수 없기 때문에 가상키라는 상수로 처리한다.

[표 2-1]에서 가상키에 해당하는 매크로와 내용을 살펴보았는데, 각 내용에 해당하는 키 입력이 있는지 확인하는 방법은 WM_KEYDOWN이나 WM_CHAR 메시지가 올 때 wParam에 저장되어 오는 내용이 가상키와 같은 값인지 비교하는 것이다. 예를 들어 Back Space 키를 누르면 wParam의 내용은 VK_BACK과 같다. 따라서 가상키 처리는 WM_CHAR나 WM_KEYDOWN 메시지 처리 case 문에서 해당 가상키와 wParam이 같은지 비교하는 방식을 사용한다.

[실습 2-9]는 키가 입력되었을 때 마지막에 입력한 글자를 삭제하는 예제이다.

실습 2-9 Back Space **키 입력 처리하기**

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                               WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49      static TCHAR    str[100];
50      static int      count;
51
52      switch (iMsg)
53      {
54      case WM_CREATE:
55          count = 0;
56          break;
57      case WM_PAINT:
58          hdc = BeginPaint(hwnd, &ps);
59          TextOut(hdc, 0, 0, str, _tcslen(str));
60          EndPaint(hwnd, &ps);
61          break;
62      case WM_CHAR:
63          if(wParam == VK_BACK && count > 0) count--;
64          else str[count++] = wParam;
65          str[count] = NULL;
66          InvalidateRgn(hwnd, NULL, TRUE);
67          break;
68      case WM_DESTROY:
69          PostQuitMessage(0);
70          break;
71      }
72      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
73  }
```

62~67행 : 입력 문자열을 출력하는 프로그램과 기능이 같다. WM_CHAR 메시지를 처리하는 case 문에 wParam 내용과 VK_BACK이 같은지 확인하는 부분만 추가되었다.

63행 : 입력한 문자가 VK_BACK이고 count가 양수이면 count 값을 1 감소시킨다. 여기서 count가 양수라는 것은 삭제할 수 있는 문자가 존재한다는 의미이다.

65행 : 63행에서 1 감소된 count는 마지막에 입력 문자가 저장된 곳의 인덱스 값이다. 따라서 마지막에 입력된 문자가 str[count] = NULL에 의해 삭제된다.

66행 : InvalidateRgn( ) 함수를 호출해 화면을 삭제하고 str 문자열의 새로운 내용을 출력한다.

[실습 2-9]를 문자 집합이 멀티바이트일 경우 실행해보면 Back Space 키에 의해 한글 한 자가 완전히 지워지지 않는 것을 볼 수 있다. 한글은 Back Space 키를 두 번 눌렀을 때 한 자가 삭제된다. 이는 한글이 2바이트, 영어가 1바이트를 차지하기 때문이다. 그러나 유니코드의 경우에는 모든 문자가 2바이트에 저장되기 때문에 한글이든 영어든 상관없이 한 자씩 삭제가 가능하다.

[실습 2-10]은 Enter 키가 입력된 경우, 이후에 입력된 문자열의 출력 위치를 변경하는 예제이다.

실습 2-10 Enter **키 입력 처리하기**

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                               WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49      static TCHAR    str[100];
50      static int      count, yPos;
51
52      switch (iMsg)
53      {
54      case WM_CREATE:
55          count = 0;
56          yPos = 0;
57          break;
58      case WM_PAINT:
59          hdc = BeginPaint(hwnd, &ps);
60          TextOut(hdc, 0, yPos, str, _tcslen(str));
61          EndPaint(hwnd, &ps);
```

```
62            break;
63        case WM_CHAR:
64            if (wParam == VK_BACK && count > 0) count--;
65            else if (wParam == VK_RETURN)
66            {
67                count = 0;
68                yPos = yPos + 20;
69            }
70            else str[count++] = wParam;
71            str[count] = NULL;
72            InvalidateRgn(hwnd, NULL, TRUE);
73            break;
74        case WM_DESTROY:
75            PostQuitMessage(0);
76            break;
77        }
78        return(DefWindowProc(hwnd, iMsg, wParam, lParam));
79    }
```

65~67행 : Enter 키가 입력되었을 때 str 문자열에 저장된 문자를 모두 버리고 새로 입력받을 수 있도록 한다.

68행 : 이후에 입력되는 문자열은 화면에 출력할 때 y 좌표가 20 증가되므로 다음 행에 출력되도록 한다.

[실습 2-10]의 경우, 다음 행으로 내려가기는 하지만 이전에 입력된 모든 내용을 잃어버리는 문제가 있다. 이 문제를 해결하기 위해서는 기본적으로 2차원 배열을 선언해 입력된 모든 것을 저장해둬야 한다. 하지만 TextOut( )이 아닌 DrawText( )를 이용한다면 Enter 키를 쉽게 처리할 수 있다. [실습 2-11]에서는 DrawText( )를 이용해 문자열 출력을 함으로써 Enter 키를 간단하게 처리했다.

실습 2-11 DrawText( ) 함수를 이용해 Enter 키 입력 처리하기

```
44    LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                                WPARAM wParam, LPARAM lParam)
46    {
47        HDC             hdc;
```

```
48      PAINTSTRUCT    ps;
49      static TCHAR   str[1000];
50      static int     count, yPos;
51      RECT rt = { 0, 0, 1000, 1000 };
52      switch (iMsg)
53      {
54      case WM_CREATE:
55           count = 0;
56           yPos = 0;
57           break;
58      case WM_PAINT:
59           hdc = BeginPaint(hwnd, &ps);
60           DrawText(hdc, str, _tcslen(str), &rt, DT_TOP|DT_LEFT);
61           EndPaint(hwnd, &ps);
62           break;
63      case WM_CHAR:
64           if (wParam == VK_BACK && count > 0) count--;
65           else str[count++] = wParam;
66           str[count] = NULL;
67           InvalidateRgn(hwnd, NULL, TRUE);
68           break;
69      case WM_DESTROY:
70           PostQuitMessage(0);
71           break;
72      }
73      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
74 }
```

51행 : DrawText를 이용하려면 사각형 영역의 좌표를 저장하기 위한 RECT 변수가 필요하다. 그래서 RECT형으로 rt 변수를 선언하고 (0, 0, 1000, 1000)을 초깃값으로 저장했다. 이 사각형은 윈도우의 왼쪽 상단을 기준으로 가로세로가 1000인 정사각형 모양이다.

60행 : 문자열은 rt 사각형 내의 위에서부터 왼쪽 정렬로 출력된다. 여기서 출력 옵션에 DT_SINGLELINE을 추가하면 한 줄로 출력하라는 뜻이므로 Enter 키가 효과를 발휘할 수 없다.

63~68행 : Enter 키가 입력되었을 때 이전 실습과는 달리 str 문자열에 그냥 저장한다. 이는 DrawText()가 개행 문자를 만나며 아래 행으로 내려가기 때문이다. 여기서는 Enter 키를 입력해도 count를 0으로 변경하는 일이 없기 때문에 지금까지 입력한 모든 문자열이 출력된다.

05 캐럿 이용하기

메모장이나 일반 워드프로세서에서는 글자를 입력할 위치에 깜박이는 커서가 나타나는데 이것을 캐럿이라 한다. 캐럿이 있으면 어느 위치에 문자가 입력되는지를 알 수 있다. 이 절에서는 지금까지 실습한 예제 프로그램에서는 나타나지 않은 캐럿을 추가해보자.

캐럿은 다음과 같은 함수를 통해 이용할 수 있다.

캐럿 관리 함수

- 캐럿 만들고 보이기

```
BOOL CreateCaret(HWND hwnd, HBITMAP Bitmap, int width, int height);
BOOL ShowCaret(HWND hwnd);
```

- 캐럿 위치 설정하기

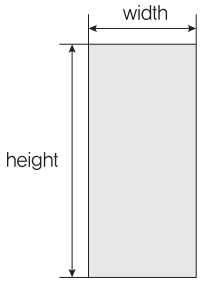
```
BOOL SetCaretPos(int x, int y);
```

- 캐럿 감추기

```
BOOL HideCaret(HWND hwnd);
```

- 캐럿 삭제하기

```
BOOL DestroyCaret();
```

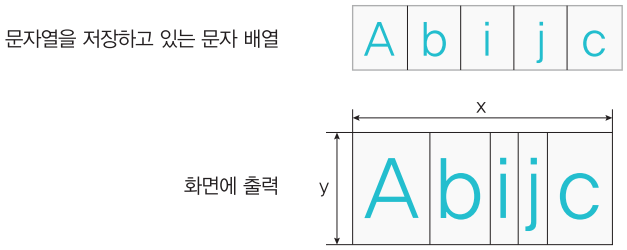


캐럿은 CreateCaret( ) 함수로 만든다. 첫 번째 매개변수(hwnd)는 캐럿이 나타날 윈도우 핸들 값이다. 이는 WinMain( )에서 만든 윈도우의 핸들로 메시지가 올 때 WndProc( )에 값이 전달된다. 두 번째 매개변수(Bitmap)는 캐럿의 내부를 칠할 내용이다. 비트맵으로 칠하고 싶으면 비트맵 핸들을 넣고 NULL을 넣으면 검은색이 된다. 세 번째와 네 번째 매개변수(width와 height)는 정수로 캐럿의 너비와 높이이다.

캐럿의 위치는 SetCaretPos( ) 함수로 설정하는데 매개변수(x와 y)는 캐럿이 위치할 좌표이다. 포커스가 다른 윈도우로 넘어가 캐럿을 감춰야 할 때는 HideCaret( ) 함수를 이용한다. 마지막으로 프로그램을 종료할 때는 DestroyCaret( ) 함수로 캐럿을 삭제한다.

그럼 문자열을 출력한 후 특정 문자 뒤에 캐럿을 출력하려면 어떻게 해야 하는지 살펴보자. 문자열 Abijc를 특정 크기의 굴림으로 출력하고 c 뒤에 캐럿을 출력한다고 해보자. 문자열을 저장하는 배열 변수의 모양은 [그림 2-8]과 같다.

그림 2-8 문자열 크기 측정



각 공간마다 문자가 하나씩 저장되는데, 이 문자열을 화면에 출력하면 각 문자가 차지하는 공간의 크기가 서로 다르다. 대문자는 차지하는 영역이 가장 크고 소문자도 문자의 모양에 따라 공간의 크기가 서로 다르다. 그렇기 때문에 특정 문자 뒤에 캐럿을 출력할 때 문자가 몇 번째 문자인지로 위치를 계산하면 값이 틀릴 수 있으니 화면에서의 실제 길이를 측정해야 한다.

위의 Abijc 문자열을 특정 크기의 굴림으로 출력한다면 너비가 x 픽셀, 높이가 y 픽셀이다. 이 문자열을 화면의 (100, 200) 좌표에 출력하면 c 뒤의 위치는 (100+x, 200)이고, 이곳에 캐럿을 출력하면 c 뒤에서 깜박인다. 따라서 캐럿의 위치를 얻으려면 문자열이 차지하는 크기를 측정해야 하는데, 이때 사용하는 함수가 GetTextExtentPoint( )이다.

문자열 크기 측정 함수 GetTextExtentPoint( )

```
BOOL GetTextExtentPoint(
     HDC        hdc,
     LPCTSTR    lpString,
     int        cbString,
     LPSIZE     lpSize
);

• LPSIZE = SIZE *
• struct tagSIZE {
       LONG     cx;
       LONG     cy;
• } SIZE;
```

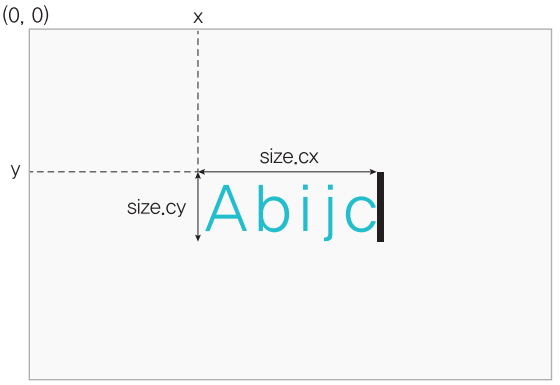
GetTextExtentPoint( ) 함수를 이용해 문자열이 화면에서 차지하는 공간이 얼마나 큰지 측정할 수 있다. 이 함수에서는 다음과 같은 매개변수를 사용한다.

- **첫 번째 매개변수(hdc)** : 캐럿을 출력할 화면이다. 화면을 함수에 알리는 이유는 화면에 현재 설정된 폰트와 크기 정보가 저장되어 있기 때문이다.
- **두 번째 매개변수(lpString)** : 크기를 측정하는 문자열이다. LPCTSTR는 Const TCHAR *와 같다고 볼 수 있기 때문에 문자열의 시작 주소나 문자열 상수를 쓸 수 있다.
- **세 번째 매개변수(cbString)** : 정수로 문자열의 몇 번째 문자까지 크기를 측정할지를 알린다. 문자열을 모두 측정할 수도 있지만 일부만 측정할 수도 있으므로 이 정숫값을 이용한다. 값이 2이면 앞에서부터 두 번째 문자까지 크기를 측정한다.
- **네 번째 매개변수(lpSize)** : cx와 cy라는 정수 필드가 있는 SIZE 구조체이다. 문자열의 너비와 높이를 측정해 각각 cx와 cy에 저장한다.

예를 들어 문자열 Abijc를 출력하고 c 뒤에 캐럿이 나타나는 프로그램을 살펴보자.

그림 2-9 캐럿의 위치 측정 예

```
SIZE size
GetTextExtentPoint(hdc, _T("Abijc"), 5, &size);
TextOut(hdc, x, y, _T("Abijc"), 5);
SetCaretPos(x+size.cx, y);
```



먼저 SIZE 구조체 변수 size를 선언했는데 이 변수에 문자열의 너비와 높이가 저장된다. 그리고 GetTextExtentPoint( ) 함수에 Abijc와 값 5를 준다. 그러면 문자의 너비는 size.cx에, 높이는 size.cy에 저장되어 돌아온다. 이어서 TextOut( ) 함수를 이용해 화면의 (x, y) 좌표에 문자열을

출력한다. c 문자 바로 뒤의 좌표는 (x+size.cx, y)가 된다. [그림 2-9]에서 보듯이 y 좌표는 아래로 내려간 것이 아니므로 변함이 없고, x 좌표만 문자열의 너비만큼 오른쪽으로 이동한다. 이제 그 위치에 캐럿의 위치를 잡으면 된다.

이번에는 앞서 배운 [실습 2-8]의 내용을 기반으로 텍스트를 한 줄 입력할 수 있는 프로그램에 캐럿을 넣어보자.

실습 2-12 캐럿이 있는 텍스트 입력 처리하기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                                WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49      static TCHAR    str[100];
50      static int      count;
51      static SIZE     size;
52
53      RECT rt = { 0, 0, 1000, 1000 };
54      switch (iMsg)
55      {
56      case WM_CREATE:
57          CreateCaret(hwnd, NULL, 5, 15);
58          ShowCaret(hwnd);
59          count = 0;
60          break;
61      case WM_PAINT:
62          hdc = BeginPaint(hwnd, &ps);
63          GetTextExtentPoint(hdc, str, _tcslen(str), &size);
64          TextOut(hdc, 0, 0, str, _tcslen(str));
65          SetCaretPos(size.cx, 0);
66          EndPaint(hwnd, &ps);
67          break;
68      case WM_CHAR:
69          if (wParam == VK_BACK && count > 0) count--;
70          else str[count++] = wParam;
71          str[count] = NULL;
72          InvalidateRgn(hwnd, NULL, TRUE);
73          break;
```

```
74        case WM_DESTROY:
75            HideCaret(hwnd);
76            DestroyCaret();
77            PostQuitMessage(0);
78            break;
79        }
81  }
```

57, 58행 : WM_CREATE 메시지를 처리하는 case 문에서 맨 먼저 캐럿을 만든다. 캐럿의 크기는 너비가 5픽셀, 높이가 15픽셀이다. 좀 더 정확하게 하려면 화면에 설정된 폰트 정보를 얻어와 캐럿의 크기를 정해야 하지만 이 과정은 여기서 다루기에 복잡하므로 생략한다. [실습 2-12]를 실행했을 때 글자 크기보다 캐럿이 너무 작으면 수치를 크게 조정하면 된다.

59행 : 정수 변수 count를 초기화한다.

61~67행 : WM_PAINT 메시지를 처리하는 case 문에서는 str에 저장된 내용을 출력하고 캐럿을 출력된 내용 뒤에 그려준다. 처음에는 str 문자 배열의 내용이 없어서 캐럿만 화면 왼쪽 상단 모서리에서 깜박인다.

68~73행 : 문자를 입력하면 WM_CHAR 메시지가 발생하고 해당 case 문에서 str 문자 배열에 입력된 문자를 저장한다. 그리고 WM_PAINT 메시지를 강제로 발생시켜 str 내용을 출력하고 캐럿의 위치를 변경한다.

69, 70행 : 키보드로 문자를 입력하면 WM_CHAR 메시지가 도착한다. 먼저 입력된 문자가 Back Space 인지 검사해 맞으면 count 변수를 감소시키고, 아니면 입력된 문자를 str 문자 배열 끝에 넣는다.

71, 72행 : 문자열의 끝을 알리는 NULL을 넣고 InvalidateRgn( )을 호출하면 WM_PAINT 메시지가 발생한다.

74~78행 : 프로그램을 종료하면 생성했던 캐럿을 삭제한다. 앞서 설명한 것과 같이 PostQuitMessage(0)은 WinMain( )의 GetMessage( ) 함수가 0의 값을 강제로 반환하도록 만들어준다. 이는 while 문을 멈추게 하고 프로그램이 종료된다.

06 직선, 원, 사각형, 다각형 그리기

6.1 직선 그리기

직선은 두 점과 그 사이를 연결하는 선분으로 이뤄진다. 따라서 직선을 그리려면 시작점과 끝점의 좌표를 나타내야 한다. Win32 API를 이용해 직선을 그리는 방법은 두 단계이다. 첫 번째 단계는 시작점으로 포인트를 옮기는 것이다. 이 과정에는 MoveToEx( ) 함수를 사용하며 시각적인 변화가 없다. 두 번째 단계는 직선을 그리는 것이다. 현재 위치부터 지정된 점까지 직선을 그리는 데 사용하는 함수는 LineTo( )이다.

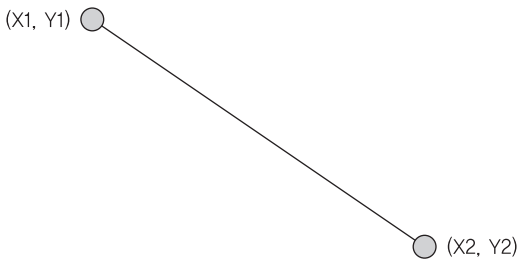
직선 그리기 함수

- 직선의 시작점으로 이동하기

```
BOOL MoveToEx(
    HDC hdc,
    int X1,
    int Y1,
    LPPOINT lpPoint
);
```

- 직선의 끝점까지 직선 그리기

```
BOOL LineTo(
    HDC hdc,
    int X2,
    int Y2
);
```



MoveToEx( ) 함수의 매개변수는 각각 다음과 같은 의미이다.

- **첫 번째 매개변수(hdc)** : 화면 영역을 가리킨다.
- **두 번째와 세 번째 매개변수(X1과 Y1)** : 옮겨갈 위치의 좌표 값으로 정숫값이다.

- **네 번째 매개변수(lpPoint)** : LPPOINT 타입으로 POINT *와 같다고 보면 된다. POINT 타입은 한 점의 좌표를 저장할 수 있는 구조체 타입으로 필드는 LONG형인 x, y 2개가 존재한다. 그러므로 MoveToEx( ) 함수의 네 번째 매개변수는 한 점의 좌표를 저장하는 POINT 변수의 주소이다. 여기에는 포인트가 이전에 위치했던 좌표를 저장하지만 거의 사용하지 않는다. 따라서 네 번째 매개변수에는 NULL 값을 넣어도 무방하다.

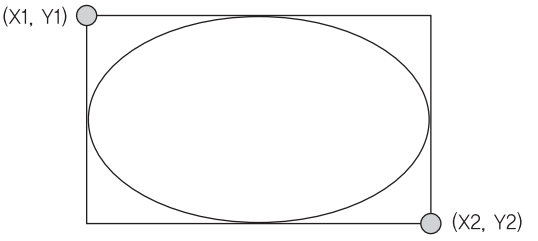
LineTo( ) 함수는 매개변수가 3개이다. 첫 번째 매개변수는 화면 영역을 가리키는 hdc이고, 두 번째와 세 번째 매개변수는 끝점의 좌표이다.

6.2 원 그리기

원은 일반적으로 중심점과 반지름으로 구성된다. 하지만 한 점을 중심으로 반지름을 이용해 원을 그리면 타원 형태는 그릴 수 없다. 윈도우 API에서는 타원까지 그릴 수 있는 원 그리기 함수인 Ellipse( )를 제공한다. Ellipse( ) 함수에는 기본적으로 2개의 좌표 값을 넣어야 한다. 두 좌표를 기준으로 가상의 사각형이 그려지고 실제로 나타나는 원은 그 사각형에 내접하는 형태이다. 가상의 사각형이 정사각형이면 완전히 동그란 원이 되고, 가상의 사각형이 정사각형이 아니라 직사각형이면 타원이 된다. 직선을 그리거나 텍스트 출력처럼 원을 그리는 Ellipse( ) 함수에도 화면 영역을 알리는 hdc 값이 주어져야 한다.

원 그리기 함수

```
BOOL Ellipse(
     HDC hdc,
     int X1,
     int Y1,
     int X2,
     int Y2
);
```



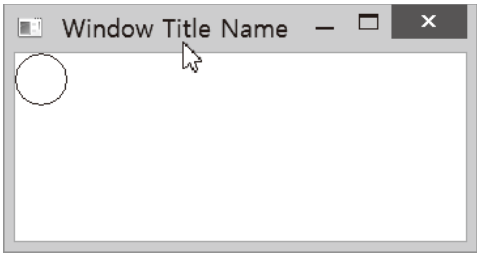
[실습 2-13]은 Ellipse( ) 함수를 이용해 원을 그리는 예제이다.

실습 2-13 원 그리기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                           WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49
50      switch (iMsg)
51      {
52      case WM_CREATE:
53          break;
54      case WM_PAINT:
55          hdc = BeginPaint(hwnd, &ps);
56          Ellipse(hdc, 0, 0, 40, 40);
57          EndPaint(hwnd, &ps);
58          break;
59      case WM_DESTROY:
60          PostQuitMessage(0);
61          break;
62      }
63      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
64  }
```

55행 : 원을 그리기 위해 화면 영역을 hdc에 얻어온다.

56행 : 두 좌표 (0, 0), (40, 40)을 기준으로 원을 그린다. 가상의 사각형이 정사각형이기 때문에 정원을 그리는데, 중심점의 좌표는 (20, 20)이고 반지름은 20이다.

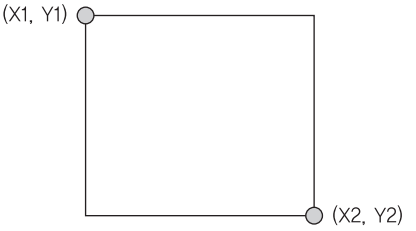


6.3 사각형 그리기

사각형을 그리는 방법은 원과 비슷하지만 여기서 그리는 사각형은 직사각형이다. 다른 형태의 일반 사각형은 다각형을 그리는 함수를 이용할 수 있다. 직사각형은 RECT 구조체와 같이 2개의 점으로 만드는데, 주어진 두 점은 직사각형의 대각선 방향 꼭짓점 역할을 한다. 직사각형을 그리는 함수는 Rectangle( )이다.

직사각형 그리기 함수

```
BOOL Rectangle(
     HDC hdc,
     int X1,
     int Y1,
     int X2,
     int Y2
);
```



Rectangle( ) 함수는 화면 영역에 해당하는 hdc를 첫 번째 매개변수로 하고 두 번째부터 다섯 번째 매개변수는 정수 좌표 값이다. (X1, Y1)이 하나의 점이고 (X2, Y2)가 대각선 방향의 점이며, 두 점을 기준으로 직사각형을 그린다. Rectangle( ) 함수를 이용해 직사각형을 그리는 예제를 살펴보자.

실습 2-14 사각형 그리기

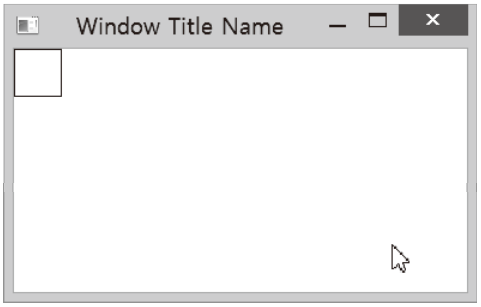
```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                           WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49
50      switch (iMsg)
51      {
52      case WM_CREATE:
53          break;
54      case WM_PAINT:
55          hdc = BeginPaint(hwnd, &ps);
56          Rectangle(hdc, 0, 0, 40, 40);
```

```
57          EndPaint(hwnd, &ps);
58          break;
59      case WM_DESTROY:
60          PostQuitMessage(0);
61          break;
62      }
63      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
64  }
```

55행 : 프로그램이 시작되고 WM_CREATE 메시지 이후에 WM_PAINT 메시지가 도착하면 BeginPaint( ) 함수를 이용해 화면 영역을 hdc에 얻어온다. 화면 영역을 얻어오면 그림을 그리거나 텍스트를 출력할 수 있다.

56행 : (0, 0)과 (40, 40)을 기준으로 하는 사각형을 그린다. 한 변의 길이가 40인 정사각형이 그려질 것이다.

57행 : 사각형 그리기를 마친 다음 EndPaint( )로 화면 영역을 반환한다.



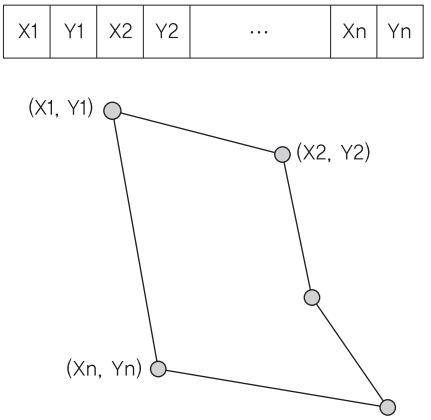
6.4 다각형 그리기

다각형이나 다각선은 여러 개의 점으로 이뤄지며 이웃하는 점 사이에 직선이 존재한다. 다각형과 다각선의 차이는 마지막 점과 시작점이 연결되었느냐 그렇지 않느냐이다. 앞서 간단히 언급했듯이 한 점의 좌표를 저장하는 구조체는 POINT이다. 따라서 POINT 구조체에는 정수형 필드인 x와 y가 존재한다. 다각형을 그리는 함수는 Polygon( )이고 다각선을 그리는 함수는 Polyline( )이다. Polyline( ) 함수는 Polygon( )과 이름만 다르고 형태가 같으므로 따로 설명하지 않겠다.

다각형 그리기 함수

```
• BOOL Polygon(
      HDC      hdc,
      CONST POINT *lppt,
      int      cPoints
  );

• typedef struct tagPOINT {
      LONG x;
      LONG y;
  } POINT;
```



Polygon( ) 함수에는 먼저 화면 영역에 해당하는 hdc를 첫 번째 매개변수로 넣고 두 번째 매개변수는 다각형 꼭짓점의 좌표로 한다. 꼭짓점의 좌표는 POINT 타입의 배열에 꼭짓점 순으로 저장된다. Polygon( ) 함수의 두 번째 매개변수는 저장된 꼭짓점 배열의 시작 주소이다. 즉 제공된 주소부터 POINT 타입의 꼭짓점이 저장되어 있다. 그러나 다각형을 그릴 때 지정 주소부터 몇 개의 POINT 값을 이용하는지는 아직 나타나지 않았다. Polygon( ) 함수의 세 번째 매개변수는 다각형의 꼭짓점 개수를 나타낸다. [실습 2-15]는 Polygon( ) 함수를 이용해 화면에 다각형을 그리는 예제이다.

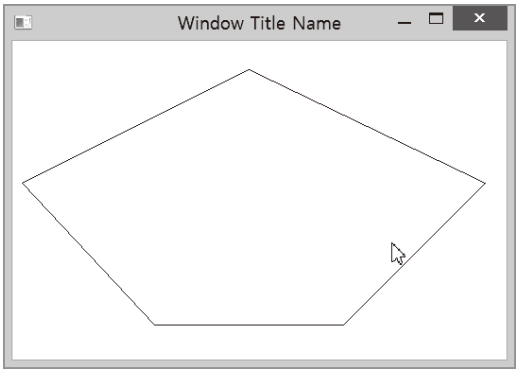
실습 2-15 다각형 그리기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                               WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT     ps;
49      POINT           point[10] = { {10, 150}, {250, 30}, {500, 150},
50                                        {350, 300}, {150, 300}};
51
52      switch (iMsg)
53      {
54      case WM_CREATE:
```

```
55            break;
56        case WM_PAINT:
57            hdc = BeginPaint(hwnd, &ps);
58            Polygon(hdc, point, 5);
59            EndPaint(hwnd, &ps);
60            break;
61        case WM_DESTROY:
62            PostQuitMessage(0);
63            break;
64        }
65        return(DefWindowProc(hwnd, iMsg, wParam, lParam));
66    }
```

49, 50행 : 꼭짓점의 좌표를 저장할 POINT 타입 배열 변수를 선언한다. 여기서는 좌표를 최대 10개 저장할 수 있는 point 배열을 지역변수로 선언하고 초깃값을 5개만 지정했다.

56~60행 : WM_PAINT 메시지가 발생했을 때 화면을 얻어오고 hdc에 꼭짓점이 5개인 다각형을 그린다. 다각형은 면이 있으므로 내부가 색으로 채워지는데 따로 지정하지 않으면 기본 색인 흰색으로 채워진다. 반면에 선의 기본 색은 검정이다.



6.5 선 속성 바꾸기

지금까지 선의 굵기나 색 등 다양한 속성을 지정하지 않은 채 기본 선으로 직선, 사각형, 원, 다각형을 그렸는데 여기서는 선의 속성을 변경하는 방법을 살펴보자. 선의 속성 정보는 펜 핸들 변수에 저장할 수 있다.

펜 생성/등록/삭제 함수

- 펜 생성하기

```
HPEN CreatePen(
     int fnPenStyle,
     int nWidth,
     COLORREF crColor
);
```

- 펜 등록하기

```
HPEN SelectObject(
     HDC hdc,
     HPEN pen
);
```

- 펜 삭제하기

```
DeleteObject(HPEN pen);
```

선의 속성 정보를 펜 핸들 변수에 저장하려면 먼저 펜 핸들 변수를 만들고 CreatePen( ) 함수를 이용해 펜 핸들 변수에 속성 정보를 저장해야 한다. CreatePen( ) 함수는 다음 3개의 매개변수를 이용해 속성(펜 유형, 굵기, 색상)을 지정한다.

- **fnPenStyle** : 펜 유형으로 아래 매크로 상수 중 하나를 사용한다. 단, 펜의 두께가 2 이상이면 실선만 지원한다.

PS_SOLID	————————	PS_DASH	— — — — — — — — — —
PS_DOT	·····	PS_DASHDOT	—·—·—·—·—·—
PS_DASHDOTDOT	—··—··—··—		

- **nWidth** : 펜의 굵기를 나타내는 정수로 단위는 픽셀이다. 값이 크면 펜의 굵기가 두꺼워진다.
- **crColor** : 선의 색을 표현한다. COLORREF 타입이 필요하고 이를 만들기 위해 RGB( ) 매크로를 이용한다. RGB( ) 매크로는 이름에서 알 수 있듯이 빨강(Red), 초록(Green), 파랑(Blue)을 위한 정숫값 3개를 필요로 한다. 세 가지 색을 모아서 하나의 색을 만들며, 정숫값은 0~255인데 0이면 해당 색을 사용하지 않고 255이면 최대로 사용한다. 255가 최댓값인 이유는, 색이 24비트이고 24비트를 삼등분하면 빨강, 초록, 파랑에 8비트씩 할당되는데 8비트가 표현할 수 있는 최대 정수가 255이기 때문이다.

```
COLORREF RGB(int Red,int Green,int Blue)
```

펜 핸들 변수에 선의 속성 값을 지정하면 펜 핸들 변수를 출력 영역인 디바이스 콘텍스트에 등록해야 한다. 디바이스 콘텍스트에는 펜 하나를 등록할 수 있으며, 디바이스 콘텍스트에 직선, 사각형, 원, 다각형 등을 그릴 때 이 펜을 이용한다.

펜을 등록할 때 사용하는 함수는 SelectObject( )이다. 첫 번째 매개변수는 화면을 가리키는 디바이스 콘텍스트 핸들이고, 두 번째 매개변수는 등록을 원하는 펜 핸들 변수이다. 명심해야 할 점은 펜으로 그리기를 마친 뒤에는 반드시 DeleteObject( ) 함수를 이용해 펜 핸들을 삭제해야 한다는 것이다. 펜을 만들고 원을 그리는 예제를 살펴보자.

실습 2-16 빨간 점선으로 원 그리기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                           WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT ps;
49      HPEN        hPen, oldPen;
50
51      switch (iMsg)
52      {
53      case WM_CREATE:
54          break;
55      case WM_PAINT:
56          hdc = BeginPaint(hwnd, &ps);
57          hPen = CreatePen(PS_DOT, 1, RGB(255, 0, 0));
58          oldPen = (HPEN)SelectObject(hdc, hPen);
59          Ellipse(hdc, 20, 20, 150, 150);
60          SelectObject(hdc, oldPen);
61          DeleteObject(hPen);
62          EndPaint(hwnd, &ps);
63          break;
64      case WM_DESTROY:
65          PostQuitMessage(0);
66          break;
67      }
68      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
69  }
```

49행 : 펜을 만들려면 펜 핸들 변수를 먼저 선언해야 하므로 HPEN 타입 변수 hPen, oldPen을 지역 변수로 선언했다. 하지만 변수를 만들었다고 해서 펜이 만들어진 것은 아니고 CreatePen( ) 함수를 이용해 펜의 다양한 속성 값을 가진 펜 핸들을 만들어야 한다.

57행 : WM_PAINT 메시지가 발생할 때 CreatePen( ) 함수를 이용해 펜을 만든다. 만든 펜은 점선이고 굵기는 1픽셀, 색은 빨간색이다. 만들어진 펜을 펜 핸들 변수 hPen에 저장한다.

58행 : hPen은 SelectObject( ) 함수를 이용해 화면 hdc에 등록한다. 새로운 펜을 hdc에 등록하면 SelectObject( ) 함수는 hdc에 등록했던 펜을 반환한다. 여기서는 반환한 펜을 oldPen에 저장한다.

59행 : 등록한 빨간 펜으로 원을 그린다.

60행 : 원 그리기를 마치면 화면 hdc에 받아둔 펜 핸들을 다시 등록해야 한다. SelectObject( ) 함수는 이때도 등록된 빨간 펜을 반환하지만 그 값을 받을 필요가 없기 때문에 저장하지 않고 있다.

61행 : 만들었던 hPen을 DeleteObject( ) 함수를 이용해 삭제한다.

6.6 면 색 바꾸기

면이 있는 도형은 원, 사각형, 다각형이다. 면이 있는 도형은 펜으로 테두리를 그리고 내부는 브러시로 칠한다. 면을 칠하려면 색상 정보를 가진 브러시 핸들을 만들어야 하는데, 이때 사용하는 함수는 CreateSolidBrush( )이다.

내부를 칠하는 브러시의 속성은 색상 정보 하나면 충분하므로 CreateSolidBrush( ) 함수의 매개변수는 COLORREF 타입의 값이다. 펜을 만들고 펜 핸들 변수에 넣은 것처럼 브러시를 만든 다음 브러시 핸들 변수에 저장해야 한다.

SelectObject( ) 함수를 이용해 브러시 핸들 변수를 디바이스 콘텍스트(출력 영역)에 등록한다.

펜과 마찬가지로 SelectObject( ) 함수는 이전에 등록한 브러시를 디바이스 콘텍스트에 반환한다. 필요하면 브러시 핸들 변수를 만들어 저장하지만 그럴 필요가 없을 때는 저장하지 않는다.

그림 그리기를 마친 뒤 브러시 핸들이 더 이상 필요 없으면 DeleteObject( ) 함수를 이용해 삭제한다. 여기서 주의할 점은 펜 핸들을 등록 또는 삭제하는 함수나 브러시 핸들을 등록 또는 삭제하는 함수가 같다는 것이다. 왜냐하면 SelectObject( )나 DeleteObject( ) 함수의 매개변수는 void 타입이라 어떤 것이든 받을 수 있기 때문이다.

브러시 생성/등록/삭제 함수

- 브러시 생성하기

```
HBRUSH CreateSolidBrush(COLORREF crColor);
```

- 브러시 등록하기

```
HBRUSH SelectObject(HDC hdc, HBRUSH brush);
```

- 브러시 삭제하기

```
DeleteObject(HBRUSH);
```

이번에는 면이 빨간색이고 테두리가 검은색인 원을 그리는 프로그램을 만들어보자.

실습 2-17 면이 빨간 원 그리기

```
44  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
45                           WPARAM wParam, LPARAM lParam)
46  {
47      HDC             hdc;
48      PAINTSTRUCT ps;
49      HBRUSH          hBrush, oldBrush;
50
51      switch (iMsg)
52      {
53      case WM_CREATE:
54          break;
55      case WM_PAINT:
56          hdc = BeginPaint(hwnd, &ps);
57          hBrush = CreateSolidBrush(RGB(255, 0, 0));
58          oldBrush = (HBRUSH)SelectObject(hdc, hBrush);
```

```
59          Ellipse(hdc, 20, 20, 150, 150);
60          SelectObject(hdc, oldBrush);
61          DeleteObject(hBrush);
62          EndPaint(hwnd, &ps);
63          break;
64      case WM_DESTROY:
65          PostQuitMessage(0);
66          break;
67      }
68      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
69  }
```

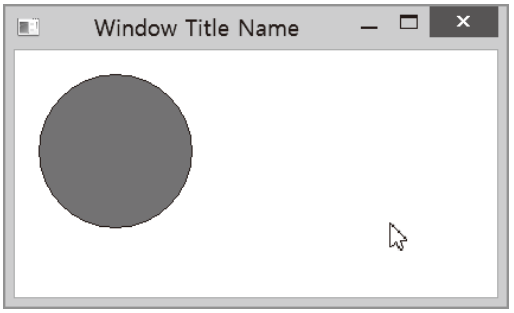
49행 : 빨간색 브러시를 만들기 위해 브러시 핸들 변수를 선언한다. 펜 핸들 변수를 사용할 때처럼 oldBrush 변수도 선언한다.

57행 : WM_PAINT 메시지에서 빨간색 브러시를 만들어 hBrush 변수에 저장한다.

58행 : SelectObject( ) 함수를 이용해 화면 hdc에 등록하고 이전에 등록한 브러시는 oldBrush에 받아둔다. 이제 화면인 디바이스 콘텍스트 hdc에 빨간색 브러시가 등록되어 면을 가진 도형을 그리면 내부가 빨간색으로 칠해진다.

59행 : 원을 그린다. 펜을 따로 만들어 등록하지 않았으므로 기본인 1픽셀 두께의 검은색 실선 펜을 이용한다. oldBrush에 받아둔 브러시는 디바이스 콘텍스트 hdc에 있던 기본 브러시로서 흰색이다.

60, 61행 : 그림 그리기를 마친 뒤 SelectObject( ) 함수를 이용해 받아둔 oldBrush를 등록하고 만들었던 hBrush 브러시는 삭제한다. 삭제하지 않으면 WM_PAINT 메시지가 발생할 때마다 브러시를 만들기 때문에 메모리의 소비가 점점 커진다.



요약

1 화면에 출력하는 기본 방법

- 출력을 하려면 디바이스 콘텍스트를 얻어와야 한다.
- 출력 영역은 x-y 좌표계로 원점은 왼쪽 상단 모서리이며, x축은 오른쪽으로 갈수록 값이 커지고 y축은 아래쪽으로 갈수록 값이 커진다.
- 화면에 직선, 원, 사각형, 다각형, 텍스트를 출력할 수 있다.
- 출력하는 도형이나 직선의 선 속성을 변경하려면 HPEN 개체를 만들어 등록해야 한다.
- 출력하는 도형의 면 속성을 변경하려면 HBRUSH 개체를 만들어 등록해야 한다.

2 문자 집합 설정

- 영문자나 특수문자는 1바이트면 충분하지만 한글과 같은 문자는 2바이트가 필요하다.
- 멀티바이트는 한글에 2바이트를 할당하는 방식인데 문자들이 차지하는 공간이 일관성이 없기 때문에 처리할 때 어려움이 있다.
- 유니코드는 모든 문자를 2바이트에 저장하기 때문에 공간 사용이 효율적이지 못하다.
- 비주얼 스튜디오는 기본적으로 유니코드를 지원한다.
- 문자 집합 환경의 영향을 받지 않으려면 TCHAR를 이용하는 것이 좋다.

3 키보드에서 발생하는 메시지

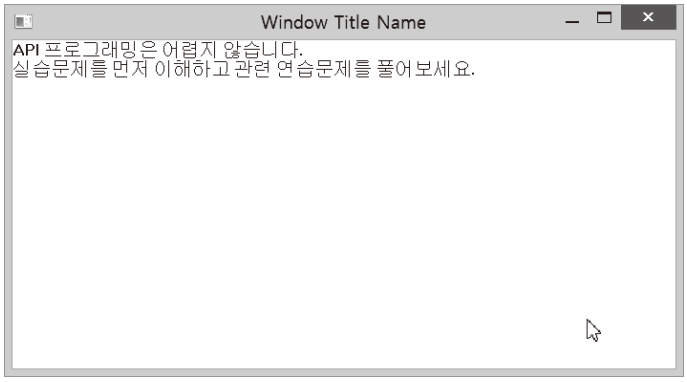
- 키보드에서 발생하는 메시지는 WM_CHAR, WM_KEYDOWN, WM_KEYUP 등이 있다.
- 키보드 메시지를 발생시킨 키 값을 얻어오려면 wParam의 내용을 보아야 한다.
- wParam에 저장된 키 값은 가상키 값으로 저장되어 있다.
- 키보드로 입력되는 문자를 화면에 출력할 때 그 위치를 나타내는 것을 캐럿이라고 한다.
- 캐럿은 디바이스 콘텍스트에 설정된 글꼴 크기에 따라 크기를 정하고 문자가 입력됨에 따라 위치를 변경해야 한다.

연습문제

1 좌표 (100, 100)에 'I love you'를 출력하는 프로그램을 작성해보자. TextOut( ) 함수와 DrawText( ) 함수를 각각 이용해 텍스트를 출력한다.

2 [실습 2-5]에서는 문자의 출력 위치를 (0, 0)으로 고정했는데, 여기서는 난수를 발생시켜서 문자키가 입력될 때마다 출력 위치가 변경되도록 수정해보자. 난수를 발생시키는 함수는 rand( )이다.

3 본문에서 배운 내용을 바탕으로 10행까지 입력받을 수 있는 메모장을 작성해보자. 한 행에는 최대 99자까지 들어가고 최대 10행까지 입력받을 수 있다. 만약 한 행이 99자가 넘으면 자동으로 개행이 되게 하라.



▶ **HINT** 아직 배우지 않은 내용은 Enter 키의 입력 처리이다. Enter 키가 입력되었는지 알려면 WM_CHAR 또는 WM_KEYDOWN이 왔을 때 조건(wParam==VK_RETURN)이 참인지 확인한다.

[실습 2-8]에서는 TextOut(hdc, 0, 0, str, _tcslen(str));과 같이 (0, 0) 좌표에 출력했지만 여기서는 Enter 키를 입력할 때마다 y 좌표의 값을 증가시킨다. 이렇게 하려면 먼저 현재 몇 행에 출력하는지를 알리는 변수가 필요하며 이 변수 값을 바탕으로 좌표를 재설정한다. 예를 들어 Enter 키를 한 번 입력해 텍스트를 2행에 입력하는 중이라면 행을 알리는 변수 값이 1이다. 그리고 문자열을 2행 출력하는 좌표는 한 행의 높이가 20픽셀이라고 가정하면 (0, 1*20)이다. 같은 방법으로 3행은 (0, 2*20)에 출력한다. 각 행의 내용을 계속 저장해 유지하려면 문자열 저장 변수를 2차원 문자열로 선언해야 한다. 입력된 문자열을 배열에 저장하는데, 만약 99개 문자 이상이 Enter 키 입력 전에 들어온다면 100번째부터는 다음 행에 저장해야 한다.

4 [실습 2-11]에서는 Back Space 키를 누르면 한 자씩 정상적으로 삭제되는 것을 확인할 수 있다. 문자열을 여러 행 입력한 다음 Back Space 키로 아래 행부터 하나씩 삭제해보자. 아래 행의 모든 문자가 삭제되고 나면 바로 윗 행 마지막 문자부터 한 자씩 삭제된다. 그런데 [실습 2-11]에서는 아래 행에서 윗 행으로 올라갈 때 행 끝 부분에서 Back Space 키를 한 번 더 눌러야 된다. 여기서는 이 문제를 해결해 프로그램을 작성해보자.

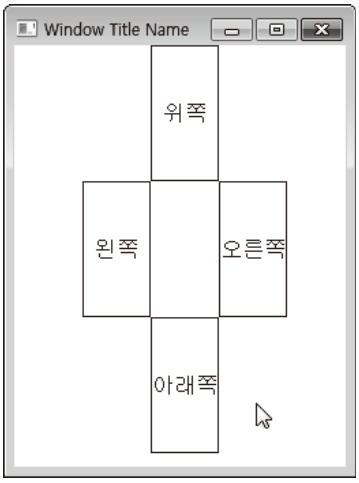
▶ **HINT** Back Space 키로 삭제할 문자가 개행 문자이면 한 문자를 추가로 삭제하게 해준다.

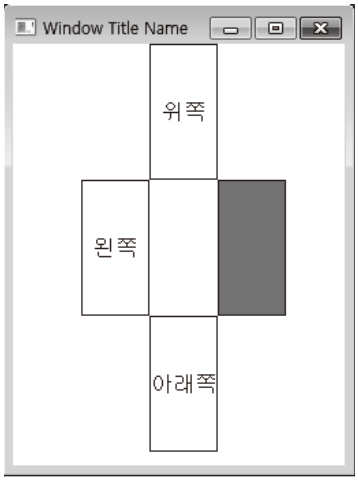
5 연습문제 3번의 최대 10행까지 입력받을 수 있는 메모장에 캐럿을 추가해 사용자가 입력받기 쉽게 만들어보자.

▶ **HINT** Enter나 Back Space 키 입력 시 캐럿의 위치를 주의해야 하는데, 이는 현재 입력된 문자의 개수를 기억하는 count 변수를 이용해 해결할 수 있다. 그리고 Enter 키로 입력되는 문장의 출력이 아래로 내려올 때 캐럿도 내려와야 하는데, 이것은 캐럿의 y 좌표 값을 변경해 해결한다.

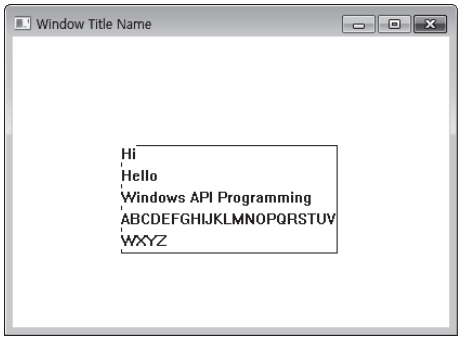
6 펜 핸들과 브러시 핸들을 이용해 원을 그리는 프로그램을 작성해보자. 펜은 노란색에 굵기가 1픽셀인 파선이며, 브러시는 분홍색이다. 중심 좌표는 (200, 120)이고 반지름은 20이다.

7 화면에 방향키 4개에 대한 사각형을 그린다. 키보드 방향키를 누르면 해당되는 사각형 면이 빨간색으로 변하고 글이 사라지며, 눌렀던 키를 놓으면 사각형이 원래대로 돌아가고 글도 나타나도록 만들어보자.





※8 화면에 사각형 글상자를 만들어보자. 사각형 왼쪽 상단에 캐럿이 나타나 문자의 입력을 기다린다. 글자를 입력하다가 글상자 경계를 만나면 캐럿이 다음 행으로 내려간다. Enter 키를 입력해도 캐럿이 다음 행으로 내려간다. 글상자가 가득 차면 더 이상 입력을 받지 않고 사각형 모양과 캐럿이 사라진다.



※9 화면 하단 중앙에 문자열을 한 행 입력받을 수 있는 글상자를 배치한다. 사각형에는 캐럿이 나타나 문자열의 입력을 기다린다. 입력할 수 있는 문자열은 직선 그리기, 원 그리기, 사각형 그리기 API 함수와 비슷하다. Ellipse(0, 0, 50, 50)을 입력하고 Enter 키를 누르면 중심 좌표가 (25, 25)이고 반지름이 25인 원이 나타난다. Line(10, 10, 200, 150)을 입력하면 (10, 10)에서 (200, 150)까지 직선을 그린다. Rectangle(0, 0, 100, 200)을 입력했을 때는 가로가 100, 세로가 200인 직사각형이 나타나야 한다.