Chapter 02

윈도우 기본 입출력

* 학습목표

• 윈도우 화면에 출력하기 위해 디바이스 컨텍스트 개념을 이해할 수 있다.

• 텍스트를 출력하는 기본 함수를 사용할 수 있다.

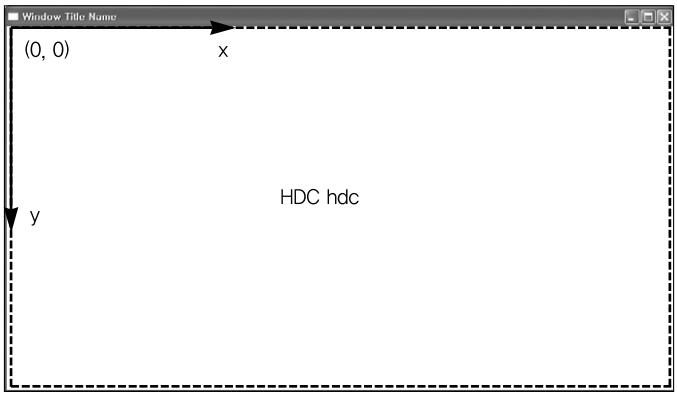
• 기본 도형을 화면에 출력할 때 필요한 요소와 함수를 사용할 수 있다.

Section

01 출력 영역 얻기

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

출력 영역에서 좌표는 좌측상단 모서리를 원점으로 한다. X축 값은 오른쪽으로 갈수록 커지고 Y축 값은 아래쪽으로 갈수록 커진다. 기본 단위는 픽셀이지만 좌표계의 원점도 변경할 수 있고 Y축 방향도 위로 변경할 수 있다. 또한, 단위도 cm나 mm, 인치 등으로 다양하게 나타낼 수 있다.



[그림 2-1] 디바이스 컨텍스트와 기본 좌표계

출력을 위해 얻어온 화면 영역을 디바이스 컨텍스트라 한다. 디바이스 컨텍스트는 변수에 저장해야 하는데 변수 타입은 HDC이다. HDC 타입이 메모리 영역을 관리하며, 메모리 영역에는 얻어온 화면 영역에 대한 속성값을 저장할 수 있다.

HDC 타입의 변수를 hdc라고 선언하면 hdc에 디바이스 컨텍스트를 얻어와 저장할 수 있

다. 화면 얻어오기 함수를 이용해 얻은 화면을 hdc에 저장한다.

디바이스 컨텍스트를 얻는 방법은 다양하지만 여기서는 두 가지만 소개한다. 다음과 같은 형식의 BeginPaint( ) 함수를 이용하는 방법과 나중에 알아볼 GetDC( ) 함수를 이용하는 방법이다.

디바이스 컨텍스트 얻기 함수 : BeginPaint( )

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

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

BeginPaint( ) 함수는 [그림 2-1]의 점선 사각형 영역인 윈도우의 클라이언트 영역을 디바이스 컨텍스트로 할당해 핸들값을 반환한다. BeginPaint( ) 함수로 디바이스 컨텍스트 핸들을 얻어오는 방법은 WM_PAINT 메시지가 발생했을 때만 사용해야 하고, 다른 메시지는 GetDC( ) 함수를 사용한다. BeginPaint( ) 함수를 이용할 때 매개변수로 주어진 PAINTSTRUCT 구조체에는 출력 영역(디바이스 컨텍스트)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 컨텍스트)에 대한 핸들값
- `fErase` : 배경 삭제 여부를 가리키는 불(Bool) 값이다. 참(True)이면 응용 프로그램이 직접 배경을 삭제해 주어야 하는데, 그렇게 하려면 윈도우 클래스를 만들 때 hbrBackground 멤버를 NULL로 채워야 함
- `rcPaint` : RECT 구조체로, 출력 영역의 좌측 상단 꼭지점과 우측 하단 꼭지점의 좌표 저장
- `fRestore, fIncUpdate, rgbReserved` : 시스템에서 사용

BeginPaint( ) 함수를 이용해 디바이스 컨텍스트를 얻어와 출력을 마친 후에는 반드시 EndPaint( ) 함수를 호출해야 한다. EndPaint( ) 함수는 출력의 끝을 나타내고, 함수 호출에 사용하는 인수는 BeginPaint( ) 함수와 같다.

디바이스 컨텍스트 반환 함수 : EndPaint( )

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

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

디바이스 컨텍스트를 얻는 두 번째 방법은 GetDC( ) 함수를 이용하는 것이다. 인수로는 윈도우의 핸들값을 주고 윈도우의 클라이언트 영역에 대한 디바이스 컨텍스트를 반환한다. GetDC( ) 함수를 이용해 디바이스 컨텍스트를 얻어와 출력한 후에는 반드시 ReleaseDC( ) 함수를 호출해 출력을 마쳤음을 알려야 한다.

디바이스 컨텍스트 얻어오는 함수 : GetDC( )

```
HDC GetDC(
     HWND hwnd    // 생성한 윈도우의 핸들값
);
```

디바이스 컨텍스트 반환 함수 : ReleaseDC( )

```
int ReleaseDC(
     HWND hwnd,   // 생성한 윈도우의 핸들값
     HDC  hdc     // 반환하는 디바이스 컨텍스트 핸들값
);
```

[실습 2-1]은 BeginPaint( )를 이용해 디바이스 컨텍스트를 얻어오는 예제다. 여기서는 WM_PAINT 메시지, 즉 윈도우가 화면에 나타나는 시점에 디바이스 컨텍스트를 얻어온다. 출력 방법은 아직 다루지 않았으므로 여기서는 디바이스 컨텍스트를 얻어오는 방법만 실습한다.

실습 2-1 디바이스 컨텍스트 얻어오기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06
07      switch(iMsg)
08      {
09      case WM_CREATE:
10          break;
11      case WM_PAINT:
12          hdc = BeginPaint(hwnd, &ps);
13          // 이곳에서 출력이 이루어짐
14          EndPaint(hwnd, &ps);
15          break;
16      case WM_DESTROY:
17          PostQuitMessage(0);
18          break;
19      }
20      return DefWindowProc(hwnd, iMsg, wParam, lParam);
21  }
```

11~12행 : WM_PAINT 발생 시 출력할 영역인 디바이스 컨텍스트를 BeginPaint( ) 함수를 이용해 얻어온다.

13행 : 출력 방법은 아직 다루지 않았으므로 일단 비워둔다.

14행 : 출력을 마친 후 EndPaint( ) 함수를 이용해 디바이스 컨텍스트를 반환한다.

Section

2 텍스트 출력하기

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

텍스트 출력 함수

```
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, "HelloWorld", 10);은 화면의 (0, 0) 위치에 'HelloWorld'를 출력한다. 그리고 출력 문자열의 길이는 10이다.

[실습 2-2]는 출력할 영역을 얻어온 다음, TextOut( )을 이용해 화면의 (0, 0)에 'Hello World'를 출력하는 예제다. 진하게 표시한 부분이 [실습 2-1]에 새로 추가한 부분인데, 프로그래밍 학습에서는 이렇게 달라진 부분을 살펴보는 것이 무척 중요하다.

실습 2-2 윈도우에 'HelloWorld' 출력하기

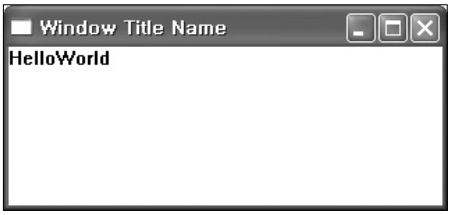
```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06
07      switch(iMsg)
08      {
09      case WM_CREATE:
10          break;
11      case WM_PAINT:
12          hdc = BeginPaint(hwnd, &ps);
13          TextOut(hdc, 0, 0, "HelloWorld", 10);
14          EndPaint(hwnd, &ps);
15          break;
16      case WM_DESTROY:
17          PostQuitMessage(0);
18          break;
19      }
20      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
21  }
```

04~05행 : HDC 타입의 hdc 변수와 PAINTSTRUCT 타입의 구조체 변수 ps를 만든다. ps는 화면에 관한 다양한 정보를 얻어오는 데 사용하지만 여기서는 사용하지 않는다.

09~10행 : 1장에서 설명한 것처럼 WndProc( ) 함수에 다양한 메시지가 전달된다. 먼저 도착하는 메시지는 윈도우가 처음 만들어졌을 때 발생하는 WM_CREATE다. 여기서는 WM_CREATE에 대해 어떤 일도 하지 않고 switch 문을 끝내고 있다.

11~15행 : 윈도우가 화면에 등장하면서 WM_PAINT 메시지가 발생한다. 이때 디바이스 컨텍스트를 얻어온 후, TextOut( ) 함수를 이용해 화면인 hdc에 'HelloWorld'를 출력한다. 출력을 마친 후에는 출력을 마쳤음을 알린다.

16~18행 : 윈도우의 닫기 버튼을 누르거나 종료하면 WM_DESTROY 메시지가 발생한다. 여기서는 PostQuitMessage(0)를 수행하고 있는데 이것은 WinMain( )의 while 문에 있는 GetMessage( ) 함수가 0을 반환하게 한다. 즉, while 문이 거짓이 되므로 루프는 종료하고 WinMain( )은 끝난다. 이는 전체 프로그램을 종료시킨다.



DrawText( )는 TextOut( ) 함수처럼 텍스트를 화면에 출력하지만, 한 점의 좌표를 주고 출력하게 하는 것이 아니라 박스 영역을 지정하고 그 안에 출력한다. 그리고 사용하는 매개변수가 TextOut( ) 함수의 매개변수와 대체로 유사하지만, 영역 좌표를 전달하는 매개변수가 있다는 것과 영역의 어느 위치에 출력할 지를 알리는 플래그 값이 있다는 점이 다르다.

텍스트 출력 함수(좌표 지정)

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

- `HDC hdc` : 화면 영역을 가리키는 변수
- `LPCSTR lpString` : 출력할 텍스트 문자열
- `int nLength` : 출력할 문자열의 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 출력할 지를 알려주는 플래그 값, 미리 정의된 상수 사용

박스 영역을 지정할 때는 RECT 구조체를 사용한다.

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

RECT 구조체

박스 영역을 그림으로 나타내면 [그림 2-2]와 같은 직사각형이 된다. 직사각형의 좌표를 저장하려면 대각선으로 마주보는 좌표가 두 개 필요하다. RECT 구조체에서는 좌측상단 꼭지점과 우측하단 꼭지점 좌표를 이용해 RECT 변수를 만든다. 이를 위해 left, top, right, bottom이라는 네 개의 정수 필드가 있다.

[그림 2-2] RECT 영역

X1의 값은 가장 왼쪽 경계를 나타내므로 left에 저장하고, Y1은 가장 위쪽 경계를 나타내므로 top에 저장한다. 마찬가지로 right에는 X2를, bottom에는 Y2를 저장한다. Flags는 박스에 문자열을 어떤 형태로 어느 위치에 출력할 지를 알려주는 상수다. 플래그로 사용할 수 있는 값에는 다음 값들이 있는데, 박스 중앙에 한 줄로 출력할 때는 DT_SINGLELINE | DT_CENTER | DT_VCENTER를 사용한다.

- 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 가운데 출력
- DT_TOP : 박스 영역의 위쪽에 출력
- DT_BOTTOM : 박스 영역의 아래쪽에 출력
- DT_CALCRECT : 문자열을 출력한다면 차지할 공간의 크기 측정

DrawText( ) 함수의 플래그 값

[실습 2-3]은 DrawText( ) 함수를 이용해 'HelloWorld' 문자열을 가상의 사각형 중앙에 출력하는 예제다.

실습 2-3 DrawText( ) 함수 이용하기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06      RECT        rect;
07
08      switch (iMsg)
09      {
10      case WM_CREATE:
11          break;
12      case WM_PAINT:
13          hdc = BeginPaint(hwnd, &ps);
14          rect.left = 50;
15          rect.top = 40;
16          rect.right = 200;
17          rect.bottom = 120;
18          DrawText(hdc, "HelloWorld", 10, &rect,
19                   DT_SINGLELINE | DT_CENTER | DT_VCENTER);
20          EndPaint(hwnd, &ps);
21          break;
22      case WM_DESTROY:
23          PostQuitMessage(0);
24          break;
25      }
26      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
27  }
```

04~06행 : hdc와 ps 변수는 앞서 설명한 내용과 같다. rect 변수는 RECT 구조체로, 텍스트 출력 영역을 지정한다.

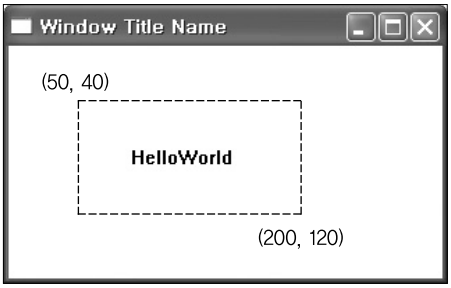
10~11행 : 먼저 오는 메시지는 [실습 2-2]처럼 WM_CREATE다. 여기서도 return 문으로 switch 문을 빠져나가 WndProc( )를 마친다.

12~13행 : WM_PAINT 메시지가 오면 BeginPaint( ) 함수로 화면 영역을 얻어온다.

14~17행 : rect 변수의 left, top, right, bottom 필드에 50, 40, 200, 120 값을 저장한다.

18~19행 : DrawText( ) 함수를 이용해 rect 영역에 'HelloWorld'를 출력한다. 출력할 때 사용하는 플래그 조합은 rect 영역의 중앙에 한 줄로 출력함을 의미한다.

20~21행 : EndPaint( )를 이용해 출력을 마쳤음을 알린다.



실행 화면에서 나타난 점선 사각형은 박스 영역을 표시하기 위한 것으로 실제 출력 화면에는 나타나지 않는다. 이 박스 중앙에 텍스트를 한 줄 출력한다.

Section

03 키보드 메시지 처리하기

지금까지는 문자열 상수를 화면에 일방적으로 출력하는 방법을 배웠다. 여기서는 키보드로 입력한 정보를 받는 방법을 배운다. 키보드에서 발생하는 주요 이벤트는 키를 누르거나 떼는 것이다. 이벤트가 발생하면 윈도우 프로시저인 WndProc( )에 [그림 2-3]과 같은 정보가 전달되는데, 이 정보는 키보드 메시지, 가상키 값, 부가 정보 등이다. 그러면 윈도우 프로시저에서 다음과 같은 변수를 기반으로 메시지를 처리한다.

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

[그림 2-3] 키보드에서 전달되는 메시지

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

키보드 메시지

WM_KEYDOWN 메시지는 키보드의 아무 키나 눌렀을 때 발생한다. WM_CHAR 메시지는 키보드의 문자키를 눌렀을 때 발생한다. 그러면 a 키를 누르면 어떤 메시지가 발생할까? 먼저 WM_KEYDOWN 메시지가 발생하고 다음으로 WM_CHAR 메시지가 발생한다. WM_KEYUP은 키를 눌렀다가 뗄 때 발생하는 메시지다. 이외에도 여러 메시지가 있지만 이 세 가지를 주로 사용한다. [그림 2-3]과 같이 키보드 메시지와 함께 wParam 변수에는 이벤트를 발생시킨 키의 가상키 값이 저장된다. 알파벳이나 숫자를 나타내는 문자나 특수문자가 아스키 코드 값으로 전달된다. 그러나 Enter 키나 Esc, Back Space, 방향키 경우에는 [표 2-1]과 같은 가상키의 코드 값으로 전달된다.

[표 2-1] 가상키 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 저장되어 전달되는데 자세한 설명은 생략한다.

[실습 2-4]는 메시지를 처리하는 예제다.

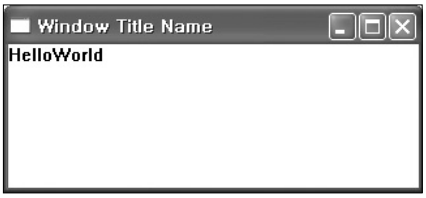
실습 2-4 WM_CHAR 메시지 처리하기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC  hdc;
05
06      switch(iMsg)
07      {
08      case WM_CREATE:
09          break;
10      case WM_CHAR:
11          hdc = GetDC(hwnd);
12          TextOut(hdc, 0, 0, "HelloWorld", 10);
13          ReleaseDC(hwnd, hdc);
14          break;
15      case WM_DESTROY:
16          PostQuitMessage(0);
17          break;
18      }
19      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
20  }
```

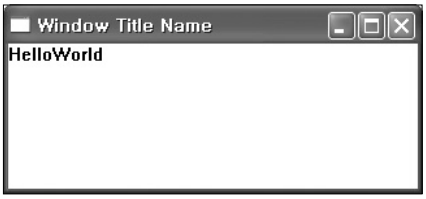
11행, 13행 : GetDC( ) 함수를 이용해 HDC를 얻어온다. 앞에서는 WM_PAINT 메시지를 처리하는 case 문에서 BeginPaint( )를 이용해 HDC를 얻어오고 EndPaint( )를 이용해 출력을 마쳤다. 하지만 WM_PAINT가 아닌 다른 메시지에서는 GetDC( ) 함수를 이용해야 한다.

출력을 마칠 때는 ReleaseDC( ) 함수를 이용한다. 사용하는 함수는 달라도 기능은 같다. 단지, 다른 윈도우로 가렸다가 나타나거나, 최소화했다가 원상 복귀했을 때 WM_PAINT 메시지를 처리하는 case 문에서 출력한 내용은 계속 볼 수 있지만, GetDC( )로 얻어와 화면에 출력하는 case 문의 내용은 모두 사라진다.

12행 : 키보드를 눌렀을 때 hdc에 화면 영역을 얻어오고 화면의 (0, 0) 좌표에 'HelloWorld'를 출력한다. 키보드의 어떤 키를 눌러도 결과는 같다. 눌린 키의 문자가 어떤 것인지 구분하지 않고 모두 'HelloWorld'를 출력한다.



a 키를 누른 결과



Enter 키를 누른 결과

[실습 2-5]는 눌러진 키보드의 문자키를 알아내 화면에 문자를 출력하는 예제다.

실습 2-5 입력 문자 처리하기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC          hdc;
05      static char  str[100];
06
07      switch(iMsg)
08      {
09      case WM_CREATE:
10          break;
11      case WM_CHAR:
12          hdc=GetDC(hwnd);
13          str[0] = wParam;
14          str[1] = '\0';
15          TextOut(hdc, 0, 0, str, strlen(str));
16          ReleaseDC(hwnd, hdc);
17          break;
18      case WM_DESTROY:
19          PostQuitMessage(0);
20          break;
21      }
22      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
23  }
```

11~12행 : [실습 2-4]와 다르게 WM_CHAR 메시지를 처리하는 case 문이 있다. WM_CHAR은 문자키를 누르면 발생하는 메시지다. WM_KEYDOWN 메시지를 case 문에서 처리해도 되지만 화면에 표시할 문자키에 대한 처리이므로 WM_CHAR 메시지 처리로 변경하였다. 키보드의 문자키를 눌렀을 때 hdc 변수에 화면을 얻어온다.

13~15행 : 미리 선언한 str 문자열에 입력된 문자를 저장해 문자열을 만든다. 코드를 보고 짐작할 수 있겠지만 키를 눌렀을 때 키의 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 들어간다. 따라서 화면의 (0, 0) 좌표에 'a' 를 출력한다.

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

[실습 2-6]은 입력한 문자를 계속 저장했다가 화면에 문자열을 출력하는 간단한 텍스트 입력 예제다.

실습 2-6 입력 문자열 처리하기

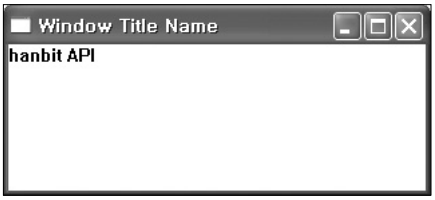
```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam,LPARAM lParam)
03  {
04      HDC         hdc;
05      static char str[100];
06      static int  count;
07
08      switch(iMsg)
09      {
10      case WM_CREATE:
11          count = 0;
12          break;
13      case WM_CHAR:
14          hdc=GetDC(hwnd);
```

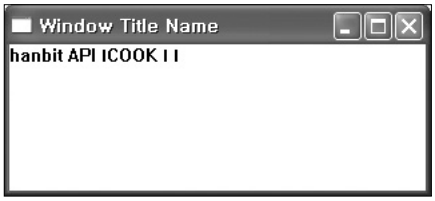
```
15          str[count++] = wParam;
16          str[count] = '\0';
17          TextOut(hdc, 0, 0, str, strlen(str));
18          ReleaseDC(hwnd, hdc);
19          break;
20      case WM_DESTROY:
21          PostQuitMessage(0);
22          break;
23      }
24      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
25  }
```

06행 : [실습 2-5]와 달리 str 문자열과 count 정수를 static 변수로 선언하였다. 메시지가 발생했을 때 저장한 내용을 다음 메시지가 발생해도 계속 유지하기 위해서다. 메시지 하나를 처리하고 나면 그 메시지 때문에 저장한 내용은 static 변수나 전역변수에 저장하지 않는 한 모두 잃는다. 왜냐하면 메시지 처리를 마치면 WndProc( ) 함수가 종료하기 때문이다.

11행 : 윈도우가 처음 만들어졌을 때 count 변수를 0으로 초기화한다. count 변수는 입력된 문자의 개수를 알려준다.

13~19행 : 문자키가 눌러질 때마다 WM_CHAR 메시지가 발생하고 입력된 문자는 wParam에 저장되어 온다. 이를 str[count]에 저장하고 count 변수를 증가시킨다. 그런 다음 (0, 0) 좌표에 str을 출력한다.





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

[실습 2-6]은 기본 기능을 하지만 몇 가지 문제가 있었다. 가장 시급한 문제 두 가지와 해결책은 다음과 같다.

- 다른 윈도우로 가렸다가 나타나면 출력한 내용이 사라진다.
  해결 방법 : WM_PAINT 메시지 발생 때문이므로 WM_PAINT 메시지를 처리하는 case문을 만든다.
- Enter나 Back Space 같은 키를 처리할 수 없다.
  해결 방법 : 가상키 테이블을 사용해 처리한다.

첫 번째는 다른 윈도우로 가렸다가 나타나거나, 윈도우를 최소화했다가 원상 복귀하면 화면에 출력된 내용이 모두 사라진다는 것이다. 이와 같은 상황은 WM_PAINT 메시지가 발생하면서 화면에 이미 출력된 내용을 무효화, 즉 전체 내용을 삭제하기 때문이다. WM_PAINT 메시지가 발생하면 화면의 내용이 모두 지워지며 WndProc( ) 함수의 WM_PAINT 메시지를 처리하는 case 문에서 출력하는 것만 남는다. 따라서 이전에 출력한 내용을 화면에 계속 남기려면 출력한 내용을 WM_PAINT 메시지에서 다시 출력해 주어야 한다.

두 번째로 Enter나 Back Space 키가 처리되지 않고 있다. 물론, 이 외에도 Tab이나 Delete 키도 사용할 수 없다. 여기서는 Enter나 Back Space 키 처리만 배우지만 원리는 같으므로 Tab이나 Delete 키에도 쉽게 응용할 수 있을 것이다. Enter나 Back Space 키를 눌렀을 때 이들에 대한 가상키 코드값이 wParam에 저장되고 이것은 단지 하나의 문자로 문자 배열 str에 저장된다. TextOut( )이나 DrawText( ) 함수는 출력하는 문자 배열에 Enter나 Back Space 문자가 있어도 무시하고 그냥 출력만 한다.

[실습 2-7]은 WM_PAINT 메시지가 발생해 화면이 삭제되는 문제를 해결한 예제다.

실습 2-7 WM_PAINT 메시지 처리하기

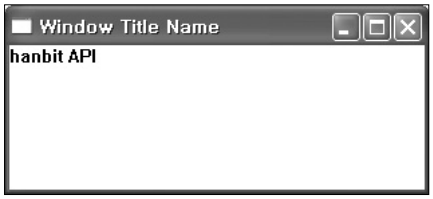
```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06      static char str[100];
07      static int  count;
08
09      switch(iMsg)
10      {
```

```
11      case WM_CREATE:
12          count = 0;
13          break;
14      case WM_PAINT:
15          hdc = BeginPaint(hwnd, &ps);
16          TextOut(hdc, 0, 0, str, strlen(str));
17          EndPaint(hwnd, &ps);
18          break;
19      case WM_CHAR:
20          hdc=GetDC(hwnd);
21          str[count++] = wParam;
22          str[count] = '\0';
23          TextOut(hdc, 0, 0, str, strlen(str));
24          ReleaseDC(hwnd, hdc);
25          break;
26      case WM_DESTROY:
27          PostQuitMessage(0);
28          break;
29      }
30      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
31  }
```

14~18행 : [실습 2-6]과 비슷하지만 WM_PAINT 메시지를 처리하는 case 문이 추가되었다. 즉, WM_PAINT 메시지가 발생하여 화면의 내용이 전부 사라졌다면 (0, 0) 좌표에 str에 저장된 문자열을 출력해 주는 것이다.

06행 : str 배열 변수는 static으로 선언되어 이전에 저장한 내용을 모두 그대로 유지한다.

초기 텍스트 입력 화면



최소화했다가 원상복귀한 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. 그럼 TextOut( ) 두 개를 하나로 모을 수 없을까? 가능하다. WM_CHAR 메시지에서 출력하는 부분을 삭제하고 WM_PAINT 메시지를 발생시켜 출력하면 된다. 화면의 특정 영역을 수정하는 InvalidateRgn( ) 함수가 있다. 이것은 클라이언트의 특정 영역을 무효화하므로 다시 그리고 싶을 때 발생시킨다. 특정 영역에 WM_PAINT 메시지를 강제로 발생시키는 효과가 있다.

화면 영역 수정 함수

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

- hWnd : 수정 영역이 포함된 윈도우의 핸들값
- hRgn : 수정 영역에 대한 핸들값으로 NULL을 주면 클라이언트 영역 전체를 수정. 여기서는 주로 NULL 값 이용
- bErase : 수정 영역을 모두 삭제하고 다시 그릴지, 수정 부분만 추가할지를 나타내는 불(BOOL) 값. 참(TRUE)이면 모두 삭제하고 거짓(FALSE)이면 삭제하지 않음

[실습 2-8]에서는 입력 문자를 저장하는 곳과 출력하는 곳을 나누고 있다. 여기서는 문자열 출력을 위한 TextOut( ) 함수 호출을 WM_PAINT 메시지 처리에서만 하고 있다.

실습 2-8 문자 저장과 출력 구분하기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06      static char str[100];
07      static int  count;
08
09      switch(iMsg)
10      {
11      case WM_CREATE:
```

```
12          count = 0;
13          break;
14      case WM_PAINT:
15          hdc = BeginPaint(hwnd, &ps);
16          TextOut(hdc, 0, 0, str, strlen(str));
17          EndPaint(hwnd, &ps);
18          break;
19      case WM_CHAR:
20          str[count++] = wParam;
21          str[count] = '\0';
22          InvalidateRgn(hwnd, NULL, TRUE);
23          break;
24      case WM_DESTROY:
25          PostQuitMessage(0);
26          break;
27      }
28      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
29  }
```

16행 : 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.

20행 : [실습 2-7]과는 달리 WM_CHAR 메시지 처리 부분에서는 키보드로 입력한 문자를 str 문자 배열에 저장만 하고 출력은 하지 않는다. 대신 InvalidateRgn( ) 함수를 호출한다.

22행 : InvalidateRgn( ) 함수는 윈도우 화면을 무효화시켜 화면의 내용을 삭제하고 WM_PAINT 메시지를 발생시킨다.

하지만 이 방법도 문제가 있다. 화면을 강제로 지우기 때문에 출력 내용이 많으면 화면이 깜박거린다. 이는 비트맵을 배운 후에 더블 버퍼링이라는 기법으로 해결할 수 있다.

두 번째 문제는 Enter나 Back Space 키 또는 방향키처럼 많이 사용하지만 문자키가 아닌 컨트롤 키 처리다. 이 키는 볼 수 있는 문자로 표시할 수 없기 때문에 가상키라는 상수로 처리한다. [표 2-1]에서 가상키에 해당하는 매크로와 내용을 살펴보았다. 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 내

용은 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 Back Space 키 입력이 들어왔을 때 마지막에 입력한 글자를 삭제하는 예제다.

실습 2-9 백스페이스 키 입력 처리하기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06      static char str[100];
07      static int  count;
08
09      switch(iMsg)
10      {
11      case WM_CREATE:
12          count = 0;
13          break;
14      case WM_PAINT:
15          hdc = BeginPaint(hwnd, &ps);
16          TextOut(hdc, 0, 0, str, strlen(str));
17          EndPaint(hwnd, &ps);
18          break;
19      case WM_CHAR:
20          if(wParam == VK_BACK) count--;
21          else str[count++] = wParam;
22          str[count] = '\0';
23          InvalidateRgn(hwnd, NULL, TRUE);
24          break;
25      case WM_DESTROY:
26          PostQuitMessage(0);
27          break;
28      }
29      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
30  }
```

19~24행 : 입력 문자열을 출력하는 프로그램과 기능은 같다. WM_CHAR 메시지 처리 case 문에 wParam 내용과 VK_BACK이 같은지 확인하는 부분만 추가되었다.

20행 : 입력한 문자가 VK_BACK이면 count 값을 1 감소시킨다.

22행 : 20행에서 1 감소된 count는 마지막에 입력 문자가 저장된 곳의 인덱스 값이다. 따라서 str[count] = '\0'에 의해 마지막에 입력된 문자가 삭제된다.

23행 : InvalidateRgn( ) 함수를 호출해 화면을 삭제하고 str 문자열의 새로운 내용을 출력한다.

[실습 2-10]은 Enter 키 입력이 들어온 경우 이후에 입력된 문자열의 출력 위치를 변경하는 예제다.

실습 2-10 엔터 키 입력 처리하기

```
01  LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
02                           WPARAM wParam, LPARAM lParam)
03  {
04      HDC         hdc;
05      PAINTSTRUCT ps;
06      static char str[100];
07      static int  count, yPos;
08
09      switch(iMsg)
10      {
11      case WM_CREATE:
12          count = 0;
13          yPos = 0;
14          break;
15      case WM_PAINT:
16          hdc = BeginPaint(hwnd, &ps);
17          TextOut(hdc, 0, yPos, str, strlen(str));
18          EndPaint(hwnd, &ps);
19          break;
20      case WM_CHAR:
21          if(wParam == VK_BACK) count--;
22          else if(wParam == VK_RETURN)
23          {
24              count = 0;
25              yPos = yPos + 20;
```

```
26          }
27          else str[count++] = wParam;
28          str[count] = '\0';
29          InvalidateRgn(hwnd, NULL, TRUE);
30          break;
31      case WM_DESTROY:
32          PostQuitMessage(0);
33          break;
34      }
35      return(DefWindowProc(hwnd, iMsg, wParam, lParam));
36  }
```

24행 : Enter 키 입력이 들어왔을 때 str 문자열에 저장된 문자를 모두 버리고 새로 입력받을 수 있도록 한다.

25행 : 다음에 입력되는 문자열은 화면에 출력할 때 y 좌표를 20 증가해주므로 다음 행에 출력하도록 한다.

요약

1 화면에 출력하는 기본 방법

- 출력을 위해서는 디바이스 컨텍스트를 얻어와야 한다.
- 출력 영역은 X-Y 좌표계로 원점은 좌측상단 모서리며 X축은 오른쪽으로 갈수록 값이 커지고 Y축은 아래로 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.

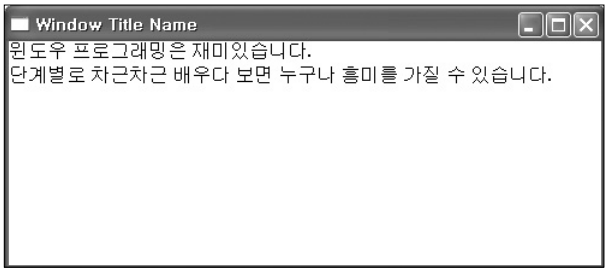
2 키보드에서 발생하는 메시지

- 키보드에서 발생하는 메시지에는 WM_CHAR, WM_KEYDOWN. WM_KETUP 등이 있다.
- 키보드 메시지를 발생시킨 키 값을 얻어오려면 wParam의 내용을 봐야 한다.
- wParam에 저장된 키 값은 가상키 값으로 저장되어 있다.
- 키보드로 입력되는 문자를 화면에 출력할 때 위치를 나타내는 것을 Caret이라고 부른다.
- Caret은 디바이스 컨텍스트에 설정된 글꼴 크기에 따라 크기를 정해주고 문자가 입력됨에 따라 위치를 변경해야 한다.

연습문제

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

2 본문에서 배운 내용을 기반으로 10행까지 입력받을 수 있는 메모장을 작성해보자. 입력받을 때 한 행은 최대 99자까지 가능하고, 모두 10행까지 입력받을 수 있다.



힌트

아직 배우지 않은 내용은 Enter 키의 입력 처리다. 이를 위해서는 wParam==VK_RETURN 조건이 참인지 확인하여, 참이면 다음 행에서 출력하도록 해야 한다.

[실습 2-8]에서는 TextOut(hdc, 0, 0, str, strlen(str));과 같이 (0, 0) 좌표에 출력하였지만 여기서는 Enter 키를 입력할 때마다 y 좌표의 값을 증가시켜준다. 이렇게 하려면 먼저 현재 몇 행에 출력하는지를 알리는 변수가 필요하다. 이 변수 값을 기반으로 좌표를 재설정한다. 예를 들어, Enter 키를 한 번 입력해 텍스트를 2행에 입력하는 중이라면 행을 알리는 변수값은 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.

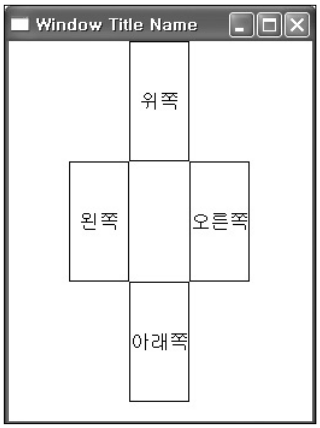
3 연습문제 2번의 10행까지 입력받을 수 있는 메모장에 Caret을 추가하여 사용자가 입력받기 쉽게 해보자.

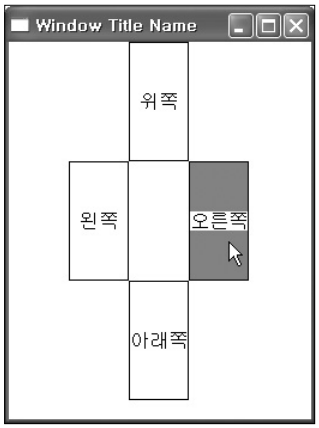
힌트

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

4 펜 핸들과 브러시 핸들을 이용해 원을 그리는 프로그램을 작성해보자. 여기서 만들 펜은 빨간색이고, 굵기가 2픽셀인 실선이며, 브러시는 파란색이다. 중심 좌표는 (20, 20)이고 반지름은 20이다.

5 화면에 방향키 4개에 대한 사각형을 그린다. 키보드 방향키를 누르면 해당되는 사각형 면이 빨간색으로 변한다. 눌렀던 키를 놓으면 사각형이 원래대로 돌아가도록 만들어보자.





6 화면에 사각형 글상자를 만들어보자. 사각형 좌측상단에 Caret이 나타나 문자 입력을 기다린다. 글자를 입력하다가 글상자 경계를 만나면 Caret은 아래 행으로 내려간다. Enter 키를 입력해도 Caret이 다음 행으로 내려간다. 글상자가 가득차면 더 이상 입력을 받지 않는다.

7 화면 하단 중앙에 문자열을 한 행 입력받을 수 있는 글상자를 배치한다. 사각형에는 Caret이 나타나 문자열 입력을 기다린다. 입력할 수 있는 문자열은 직선 그리기, 원 그리기, 사각형 그리기 API 함수와 비슷하다. Ellipse(0, 0, 50, 50)으로 입력하고 Enter 키를 누르면 중심 좌표가 (25, 25)이고 반지름이 25인 원이 나타난다. Line(10, 10, 200, 150)으로 입력하면 (10, 10)에서 (200, 150)까지 직선을 그린다. Rectangle(0, 0, 100, 200)은 가로가 100, 세로가 200인 직사각형을 그려보자.