제 2장 윈도우 기본 입출력 2

2023년 1학기 윈도우 프로그래밍

6. 키보드 메시지 처리하기

- 키보드로 입력한 정보 받기

<table>
<tr><th>메시지</th><th>내용</th><th>윈도우 프로시저 인수값</th><th>실행 형태</th></tr>
<tr><td>WM_KEYDOWN</td><td>키보드에서 키를 눌렀을 때 발생하는 메시지</td><td rowspan="2">• wParam: 가상키코드값
• lParam: 키보드 상태에 대한 정보들 (반복 실행 횟수, 검사 코드, 확장 키 플래그, 컨텍스트 코드, 이전 키-상태 플래그 및 전환 상태 플래그)</td><td rowspan="3">'a'키를 눌렀을 때:
WM_KEYDOWN 메시지 발생하고, 다음으로
WM_CHAR 메시지 발생</td></tr>
<tr><td>WM_KEYUP</td><td>키보드에서 키를 눌렀다가 떼면 발생하는 메시지</td></tr>
<tr><td>WM_CHAR</td><td>문자 키를 눌렀을 때 발생하는 메시지</td><td>• wParam: 입력된 문자값
• lParam: 키보드 상태에 대한 정보들 (반복 실행 횟수, 검사 코드, 확장 키 플래그, 컨텍스트 코드, 이전 키-상태 플래그 및 전환 상태 플래그)</td></tr>
</table>

가상키 코드 테이블

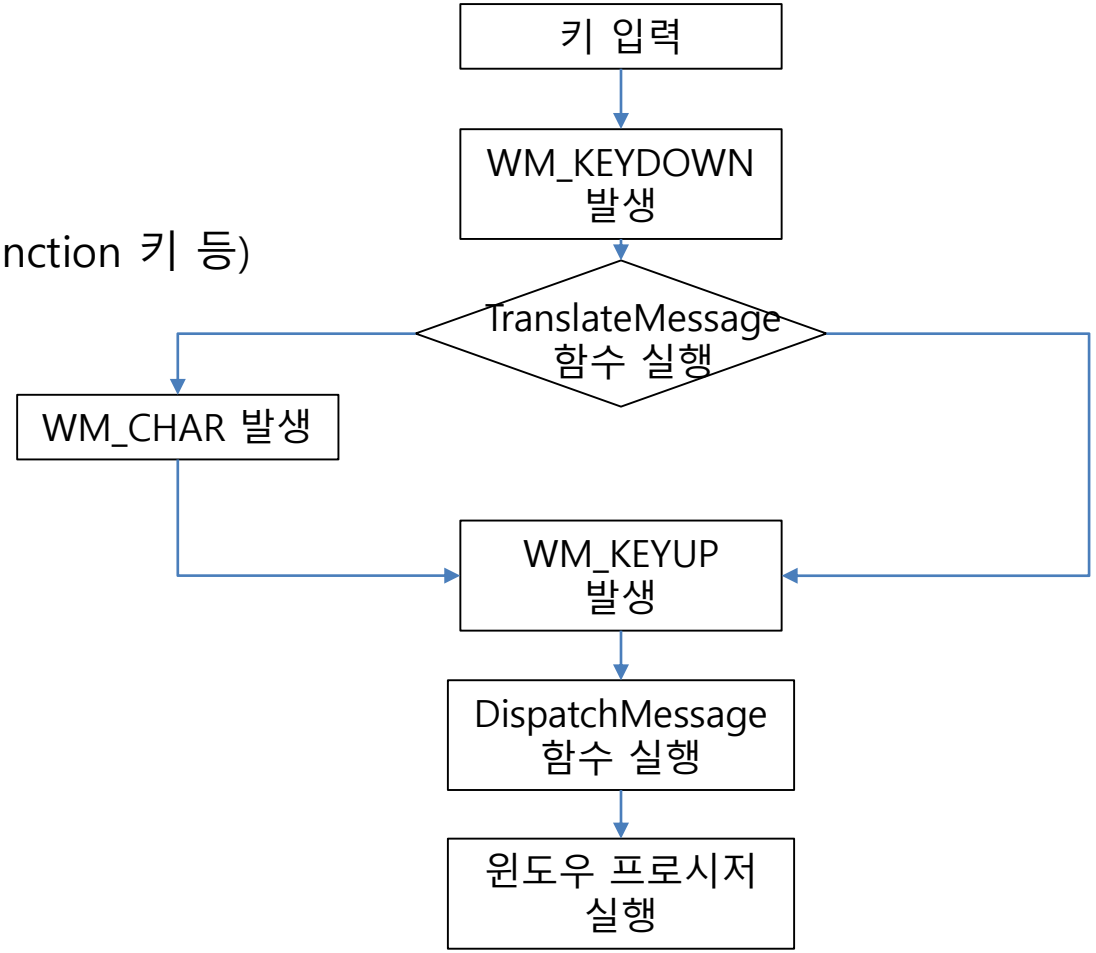
- **가상키 코드 테이블**
  - WM_KEYDOWN 메시지에서 윈도우로 전달되는 문자가 아닌 키 값 : wParam로 전달된다.
  - ALT 키, 윈도우 키, 한영 전환키 등의 특수 키 몇 가지는 제외된다.

가상키	내용	가상키	내용
VK_CANCEL	Ctrl+Break	VK_END	End
VK_BACK	Backspace	VK_HOME	Home
VK_TAB	Tab	VK_LEFT	좌측 화살표
VK_RETURN	Enter	VK_UP	위쪽 화살표
VK_SHIFT	Shift	VK_RIGHT	우측 화살표
VK_CONTROL	Ctrl	VK_DOWN	아래쪽 화살표
VK_MENU	Alt	VK_INSERT	Insert
VK_CAPITAL	Caps Lock	VK_DELETE	Delete
VK_ESCAPE	Esc	VK_F1 ~ VKF10	F1-F10
VK_SPACE	Space	VK_NUMLOCK	Num Lock
VK_PRIOR	Page Up	VK_SCROLL	Scroll Lock
VK_NEXT	Page Down		

메시지 처리: 키보드 눌렀을 때 발생하는 메시지

- **키보드를 눌렀을 때**
  - WM_KEYDOWN 메시지 발생 → 문자인 경우 WM_CHAR 메시지 발생
  - WM_KEYDOWN 메시지: 단순히 키가 눌림/안눌림을 판단, 대소문자 구분없음
  - WM_CHAR 메시지 : 입력된 문자에 대한 처리

- **메시지 사용**
  - WM_CHAR: 문자 키 일 때 사용
  - WM_KEYDOWN: 문자 이외의 키 (home, 화살표, page키, function 키 등)



WM_CHAR 메시지 처리: 입력 문자 처리하기

- **키보드로 문자 입력:**
  - 키보드로 문자를 입력하고 str[0]에 저장한다.
  - 저장한 문자를 화면에 출력

- **사용 예)**

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
	HDC hdc;
	TCHAR str[100];

	switch (iMsg) {
		case WM_CHAR:
			hdc = GetDC(hwnd);                    //--- GetDC 함수를 사용하여 dc를 얻어옴
			str[0] = wParam;                      //--- 입력문자 : : WinProc의 매개변수 wParam로 들어 옴
			str[1] = '₩0';                        //--- 문자열은 null('₩0')로 끝남
			TextOut (hdc, 0, 0, str, lstrlen(str));
			ReleaseDC (hwnd, hdc);
			break;
	}
	return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

WM_CHAR 메시지 처리: 입력 문자열 처리하기

- **키보드로 문자 입력:**
  - 입력한 문자를 순서대로 str에 저장하고 문자열로 만든다
  - 그 문자열을 화면에 출력

- **사용 예)**

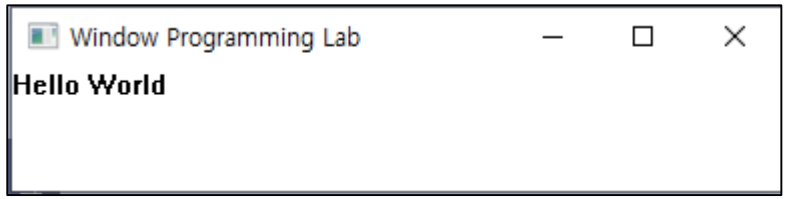
```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
	HDC hdc;
	static  TCHAR str[100];
	static int count;

	switch (iMsg) {
		case WM_CREATE :
			count = 0;
			break ;

		case WM_CHAR:
			hdc = GetDC(hwnd);                    //--- GetDC 함수를 사용하여 DC를 얻어옴
			str[count++] = wParam;                //--- 문자저장 후 인덱스 증가
			str[count] = '₩0';                    //--- 문자열은 null('₩0')로 끝남
			TextOut (hdc, 0, 0, str, lstrlen(str));
			ReleaseDC (hwnd,hdc);
			break;
	}
	return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

WM_KEYDOWN 메시지 처리

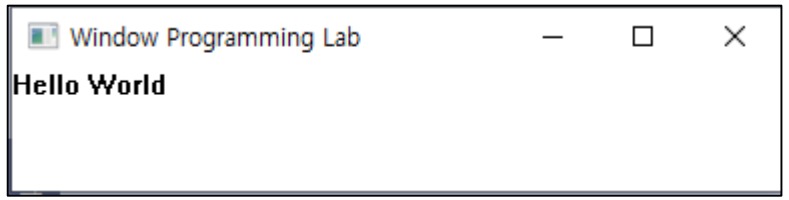
- 키를 누르면 문자열을 출력

- 사용 예)

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

    switch (iMsg ) {
       case WM_KEYDOWN:                                  //--- 키가 눌렸을 때
            hdc = GetDC (hwnd);
            TextOut (hdc, 0, 0, L"HelloWorld", 10);      //--- (0, 0) 위치에 "HelloWorld" 문자열을 출력
            ReleaseDC (hwnd, hdc);
            break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```



[a]를 누른결과



[Enter]를 누른결과

눌린 키에 관계없이 결과 출력

WM_CHAR 메시지 처리 예제

- **키보드로 문자 입력:**
  - 백스페이스 키와 엔터 키 입력 예제

- **사용 예 1)**

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc;
        static int count = 0;
        static TCHAR str[80];

        switch (iMsg) {
           case WM_PAINT:
                   hdc = BeginPaint (hwnd, &ps);
                   TextOut (hdc, 0, 0, str, lstrlen(str));                    //--- 문자 출력
                   EndPaint (hwnd, &ps);
                   break;

           case WM_CHAR:
                   hdc = GetDC (hwnd);
                   if (wParam == VK_BACK)                                     //--- 백스페이스: 마지막 문자열 삭제
                           count--;
                   else                                                       //--- 그 외에는 문자를 문자열 뒤에 붙인다.
                           str[count++] = wParam;
                   str[count] = '₩0';                                         //--- 마지막 문자로 널 추가
                   TextOut (hdc, 0, 0, str, lstrlen(str));
                   ReleaseDC (hwnd, hdc);
                   break;
      }
        return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

WM_CHAR 메시지 처리 예제

- **사용 예 2)**

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc;
        static int count = 0;
        static TCHAR str[80];
        static int yPos=0;

        switch (iMsg) {
          case WM_PAINT:
                hdc = BeginPaint(hwnd, &ps);
                TextOut(hdc, 0, yPos, str,strlen(str));
                EndPaint(hwnd, &ps);
          break;

          case WM_CHAR :
                hdc = GetDC (hwnd);
                if (wParam == VK_BACK)                  //--- 백스페이스를 입력하면
                        count--;                        //---  한 칸 삭제
                else if (wParam == VK_RETURN)           //--- 엔터키를 입력하면: 문자열을 다음줄에 출력
                {
                        count = 0;                      //--- 인덱스 변경
                        yPos = yPos + 20;               //--- y 위치 변경: 한 줄 아래에 출력
                }
                else
                        str[count++] = wParam;          //--- 그 외에는 문자를 문자열 맨 뒤에 붙인다.
                str[count] = '₩0';
                TextOut (hdc, 0, yPos, str, lstrlen(str));
                ReleaseDC (hwnd, hdc);
          break;
        }
        return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

문자열 출력: WM_PAINT 메시지 처리하기

- **코드가 중복되는 문제점 발생**

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        HDC hdc;
        static TCHAR str[100];
        static int count = 0;

        switch (iMsg) {
                case WM_CHAR:                                   //--- 1차적으로 문자열을 출력
                        hdc = GetDC(hwnd);
                        str[count++] = wParam;
                        str[count] = '₩0';
                        TextOut (hdc, 0, 0, str, lstrlen(str));          //=== 중복
                        ReleaseDC(hwnd, hdc);
                        break;

                case WM_PAINT:                                  //--- 화면이 가렸다 지워지면 다시 문자열을 출력
                        hdc = BeginPaint(hwnd, &ps);
                        TextOut (hdc, 0, 0, str, lstrlen(str));          //=== 중복
                        EndPaint(hwnd, &ps);
                        break;
    }
        return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

WM_PAINT 강제 발생 함수

- **WM_PAINT 메시지를 발생시키는 함수**

BOOL InvalidateRect (HWND hWnd, const RECT* lpRect, BOOL bErase);
- hRect: 수정될 영역에 대한 영역 핸들 값
- lpRect: 영역 좌표 (NULL이면 전체 영역을 수정)
- bErase: BeginPaint()를 위해 플래그
  - **TRUE**: 다음에 호출되는 BeginPaint에서 배경을 먼저 지운 후 작업 영역을 그리게 된다.
  - **FALSE:** 배경 브러시에 상관없이 배경을 지우지 않는다.

BOOL InvalidateRgn (HWND hWnd, HRGN hRgn, BOOL bErase);
- hWnd: 수정될 영역이 포함된 윈도우의 핸들값
- hRgn: 수정될 영역에 대한 핸들값 (NULL이면 클라이언트 영역 전체를 수정)
- bErase: 수정될 때 배경을 모두 삭제할지 안 할지를 결정하는 BOOL 값.
  - **TRUE**: 배경이 삭제된다
  - **FALSE:** 배경이 그대로 남아있다.

- **InvalidateRgn이나 InvalidateRect함수를 호출하면 클라이언트의 특정 영역을 무효화**
  - 사용자가 화면의 내용을 변경하여 다시 그리기가 필요할 때 이 함수를 호출한다.
  - 즉, 윈도우의 업데이트를 하게 된다.

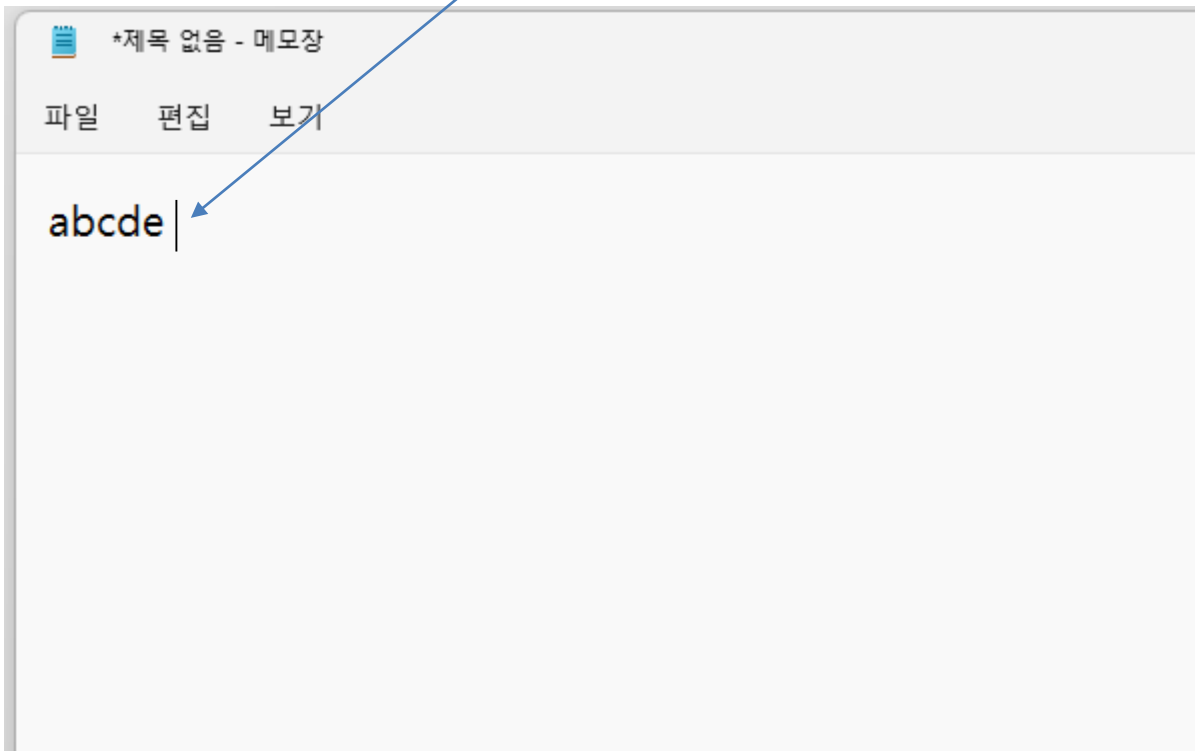
문자 저장과 출력 구분하기

- **사용 예) InvalidateRect 함수를 호출하여 출력 하기**

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
      static TCHAR str[100];
      static int count;

    switch (iMsg) {
      case WM_CHAR:
                  str[count++] = wParam;                          //--- 문자 저장
                  str[count] = '₩0';
                  InvalidateRect (hwnd, NULL, TRUE);              //--- 직접 출력하지 않고 WM_PAINT 메시지 발생
                  break;

      case WM_PAINT:
                  hdc = BeginPaint(hwnd, &ps);
                  TextOut (hdc, 0, 0, str, lstrlen(str));        //--- 문자 출력
                  EndPaint(hwnd, &ps);
                  break;
    }
      return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

7. Caret(캐럿) 이용하기

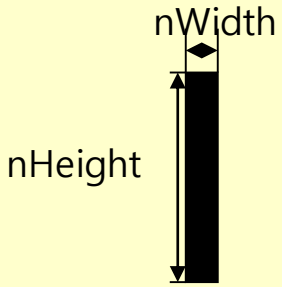
- **Caret (캐럿):**
  - 메모장이나 워드프로세서에서 키보드 입력 시 글자를 입력할 위치에 깜박거리는 커서
  - 캐럿이 있으면 어느 위치에 문자가 입력되는지를 알 수 있다.
  - 키보드 포커스는 오직 하나의 프로그램만이 가질 수 있고, 캐럿도 시스템에 하나만 존재.
  - 리소스로 사용하지 않고 API 함수를 호출하여 캐럿을 만들어서 원하는 위치에 붙일 수 있다.



Caret 이용하기

- **캐럿 관련 함수들**

- BOOL CreateCaret (HWND hwnd, HBITMAP hBitmap, int nWidth, int nHeight);
  - 캐럿 만들기 함수
  - HWND hwnd: 캐럿을 놓을 윈도우 핸들
  - HBITMAP hbitmap: 비트맵 캐럿
  - int nWidth, nHeight: 캐럿의 폭과 높이
- BOOL ShowCaret (HWND hwnd);
  - 캐럿 출력하기
  - HWND hwnd: 캐럿이 출력될 윈도우 핸들
- BOOL SetCaretPos (int x, int y);
  - 캐럿 위치 설정하기
  - int x, int y: 캐럿의 x, y 위치
- BOOL GetCaretPos (LPPOINT lpPoint);
  - 캐럿의 위치를 조사하기
  - LPPOINT lpPoint: 캐럿의 위치를 가져온다.
- BOOL SetCaretBlinkTime (UINT uMSeconds);
  - 캐럿의 깜빡임 속도를 설정
  - UINT uMSeconds: 밀리세컨 단위의 깜빡임 속도
- BOOL HideCaret (HWND hwnd);
  - 캐럿 감추기
  - HWND hwnd: 캐럿이 놓여있는 윈도우 핸들
- BOOL DestroyCaret ();
  - 캐럿 삭제하기



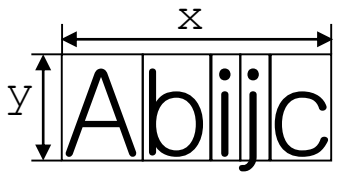
Caret 위치 정하기

- **문자열 "Abijc"를 굴림체로 출력하고 'c'뒤에 caret 위치를 정한다고 가정**

문자열을 저장하고 있는 문자 배열

A	b	i	j	c

화면상에 출력



- **x의 길이를 알아야 caret 위치 정함**
  - 예) 문자열 출력 위치가 (100,200)이라고 하면 caret의 위치는 (100+x, 200)이다.

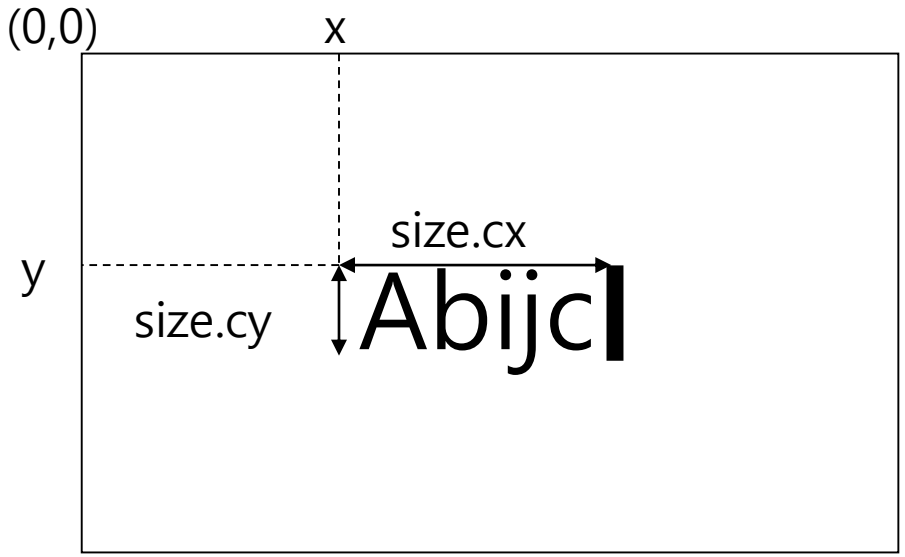
BOOL **GetTextExtentPoint32** (HDC hdc, LPCTSTR lpString, int cbString, LPSIZE lpSize);
- 문자열의 폭 구하기 함수
- HDC hdc: DC 핸들
- lpString: 크기를 측정할 문자열
- cbString: 정수로 문자열의 몇 번째 문자까지 크기를 측정할 지 알려준다
- lpSize: 문자열의 크기 (폭, 높이) -> 얻어오는 값

- **GetTextExtentPoint32A 또는 GetTextExtentPoint32W 함수 사용 가능**

출력될 문자열 폭 구하기

- **크기 저장 구조체 사용**

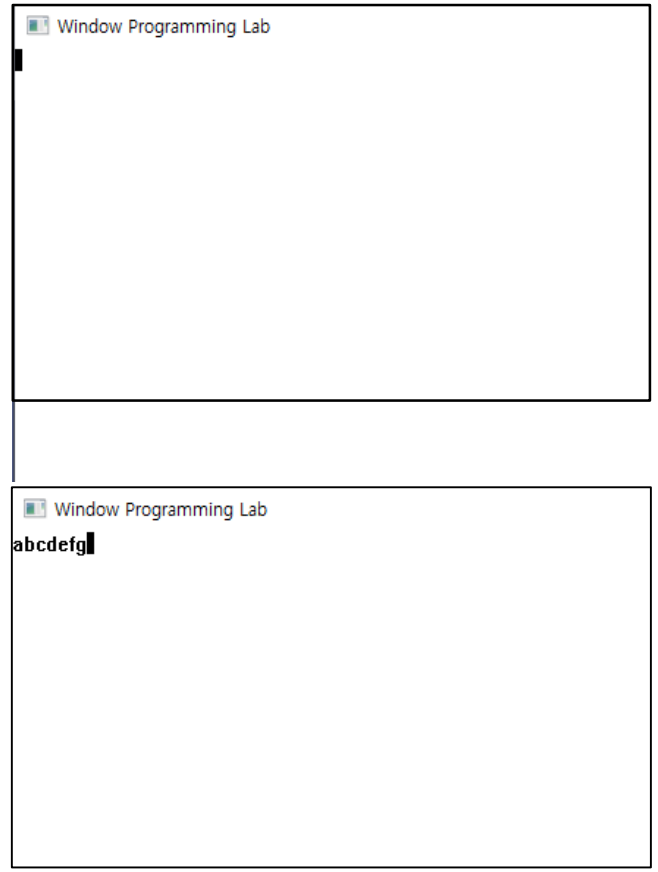
```
struct tagSIZE {                        //--- 문자열의 폭과 높이 저장
        LONG cx;
        LONG cy;
} SIZE;
```




```
SIZE size;

GetTextExtentPoint32 (hdc, L"Abijc", 5, &size);
TextOut (hdc,  x, y, "Abijc", 5);
SetCaretPos (x + size.cx, y);          //--- x좌표에 출력문자열 길이 합산
```

Caret 표시

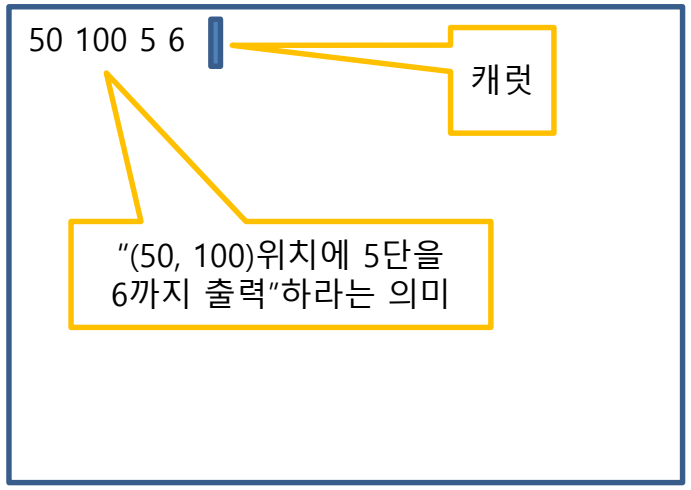
```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static SIZE size;
    static TCHAR str[100];
    static int count;                          //--- 문자열의 인덱스로 사용할 값

    switch (iMsg) {
            case WM_CREATE :
                CreateCaret (hwnd, NULL, 5, 15);                    //--- 캐럿 만들기
                ShowCaret (hwnd);                                   //--- 빈 화면에 캐럿 표시
                count = 0;
                break;
            case WM_CHAR:
                str[count++] = wParam;                              //--- 문자저장 후 인덱스 증가
                str[count] = '₩0';                                  //--- 문자열은 null('₩0')로 끝남
                InvalidateRect (hwnd, NULL, TRUE);                  //--- WM_PAINT 메시지 발생
                break;
            case WM_PAINT:
                hdc = BeginPaint(hwnd, &ps);
                GetTextExtentPoint32 (hdc, str, lstrlen(str), &size);   //--- 문자열 길이 알아내기
                TextOut(hdc,0,0,str,strlen(str));
                SetCaretPos (size.cx, 0);                           //--- 캐럿 위치하기
                EndPaint(hwnd, &ps);
                break;
            case WM_DESTROY :
                HideCaret (hwnd);                                   //--- 캐럿 숨기기
                DestroyCaret ();                                    //--- 캐럿 삭제하기
                PostQuitMessage (0) ;
                break;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

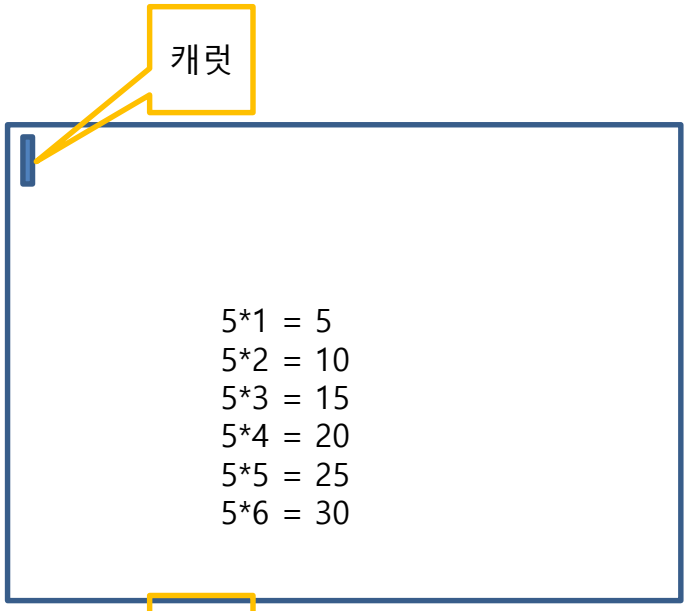


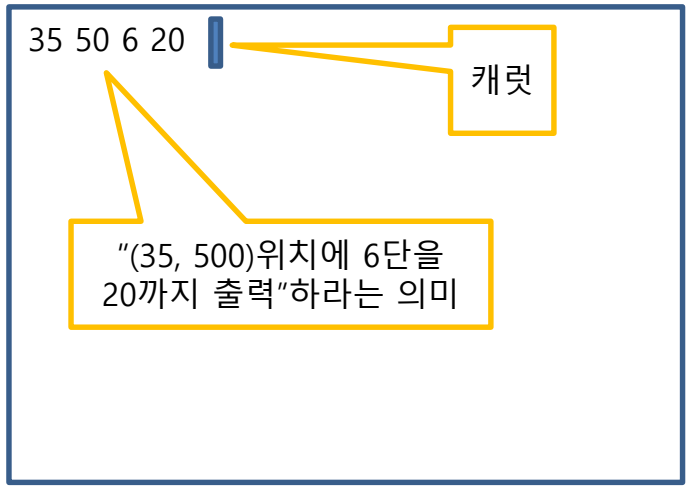
실습 2-4

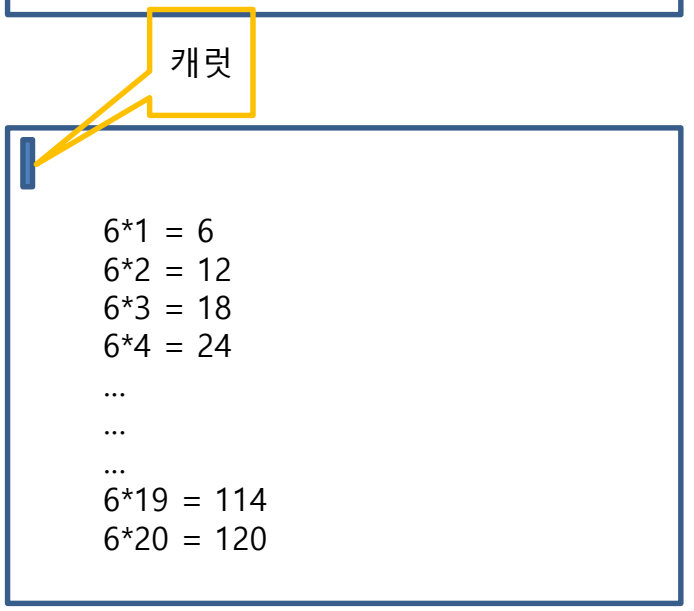
- **키보드 입력하여 구구단 출력하기**
  - 화면을 띄운다.
  - 좌측 상단에 명령어를 받는다. 캐럿을 붙여서 어디에 입력하는지 알 수 있도록 한다.
    - X: x축 좌표값
    - Y: y축 좌표값
    - N: 단 수 (19단까지 입력받도록 한다.)
    - M: 해당 단을 몇까지 출력할 지 설정하는 숫자 (1 과 30 사이의 숫자를 입력받도록 한다)
  - 입력한 위치 (X, Y)에 단수(N)의 구구단을 M 줄만큼 출력한다..
  - 출력 후 명령어를 다시 받을 수 있도록 한다.
  - 0을 입력하면 프로그램을 종료한다.

실습 2-4


50 100 5 6
캐럿
"(50, 100)위치에 5단을
6까지 출력"하라는 의미

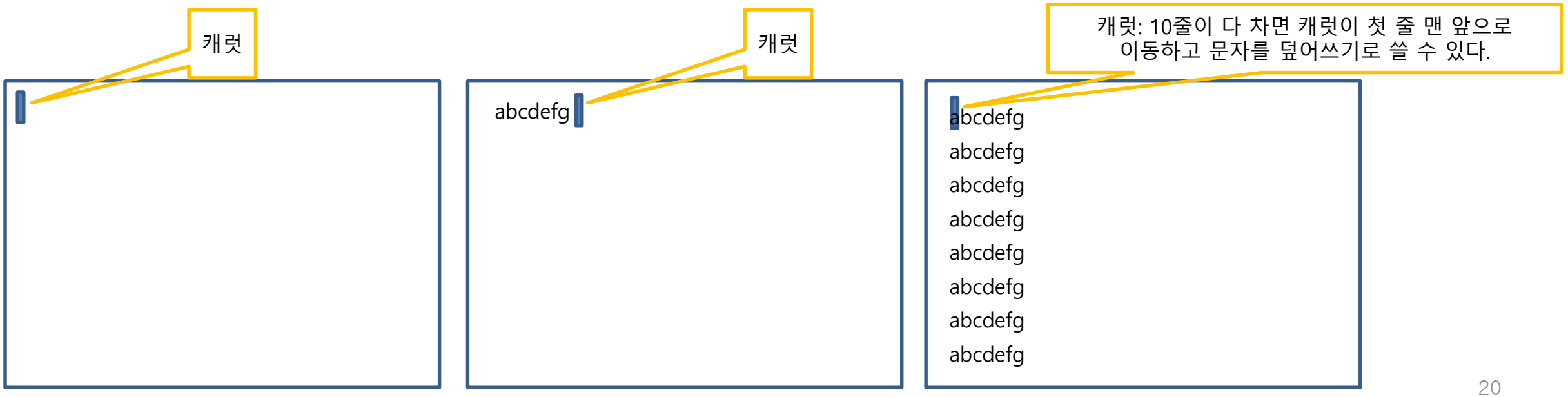

캐럿
5*1 = 5
5*2 = 10
5*3 = 15
5*4 = 20
5*5 = 25
5*6 = 30


35 50 6 20
캐럿
"(35, 500)위치에 6단을
20까지 출력"하라는 의미


캐럿
6*1 = 6
6*2 = 12
6*3 = 18
6*4 = 24
…
…
…
6*19 = 114
6*20 = 120

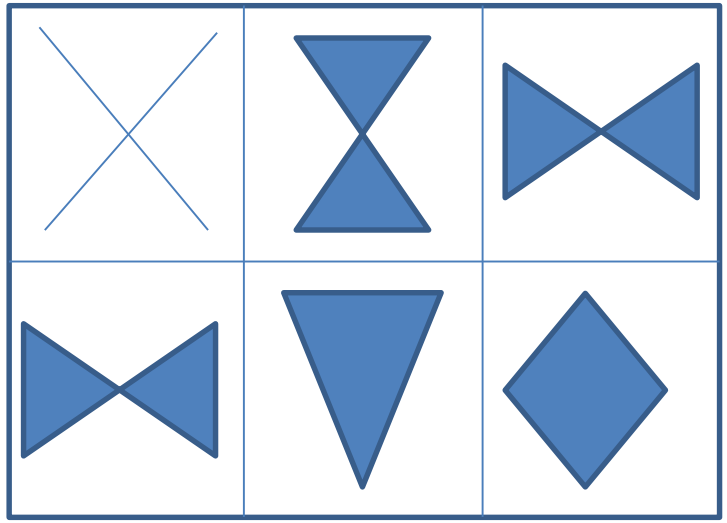
실습 2-5

- **캐럿을 이용한 메모장 만들기**
  - Caret이 있는 메모장을 작성
  - 메모장은 10라인까지, 한 줄은 최대 30자 까지 저장 가능
  - 윈도우를 띄우면 좌측 상단에 캐럿이 깜빡이고 있다. 그 위치에서부터 문자를 입력한다.
  - 문자가 30자 이상 되면 다음 줄로 내려간다. 캐럿도 같이 내려간다.
  - 문자열이 10줄이 다 차면, 캐럿이 첫 줄, 첫 번째로 이동해서 다시 입력할 수 있다. 첫 줄로 이동하여 입력하면 기존에 있던 문자 위에 덮어쓰기가 된다.
  - 문자 키 외에
    - 엔터키: 캐럿이 아랫줄 맨 앞으로 이동하고 이동된 자리에 다시 입력할 수 있다.
    - 백스페이스: 캐럿을 한 칸 앞으로 이동하고 그 자리에 다시 문자를 입력할 수 있다. 첫 줄 맨 앞의 위치에서는 앞으로 이동할 수 없다.

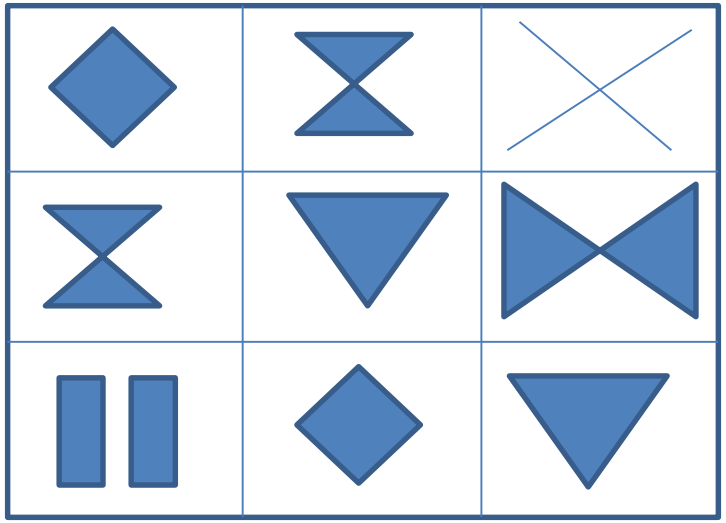


실습 2-6

- **화면에 문자 그리기**
  - 윈도우를 (800, 600) 크기로 만든 후, 화면의 가로 세로를 랜덤의 숫자로 나눈다.
    - 랜덤의 숫자는 2 ~ 10 사이의 숫자로 만든다.
  - 각 등분에 워밍업 8번 문제의 6개의 그림을 임의로 배치하여 그린다.
    - X, 역삼각형, 마름모, 나비, 모래시계, 2개 사각형
    - 그림을 그리는 문자는 여섯 개의 그림마다 다른 문자를 사용한다. 문자의 색과 배경색은 다양하게 설정한다.
    - 그림의 크기와 위치는 관계없고, 각 등분 안에만 그리면 된다.
  - 프로그램을 종료 후, 다시 시작하면 재배치된 그림이 그려진다.



가로 3등분, 세로 2등분을 하여 각 등분에 6개의 그림중 랜덤으로 선택되어 그려졌다.
(가로, 세로의 선은 안 그려도 됨: 단지 등분 표시임)



가로 3등분, 세로 3등분을 하여 각 등분에 6개의 그림이 랜덤으로 선택되어 그려졌다.
(가로, 세로의 선은 안 그려도 됨: 단지 등분 표시임)