

# 제 2장 윈도우 기본 입출력

2024년 1학기 윈도우 프로그래밍

## 2장 학습 목표

- **학습목표**

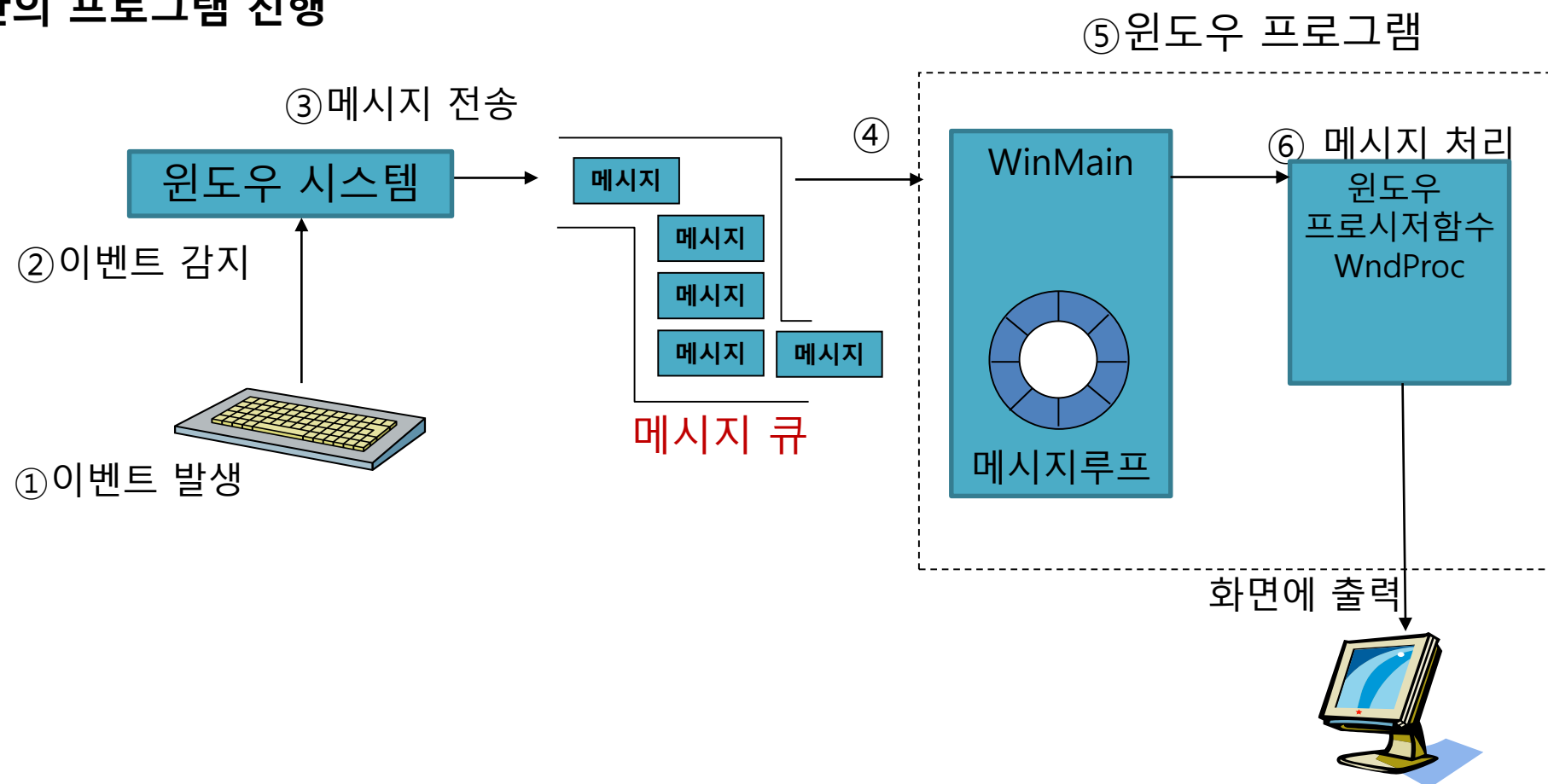
- 윈도우 화면에 출력하기 위해 디바이스 컨텍스트 개념을 이해할 수 있다.
- 텍스트를 출력하는 기본 함수를 사용할 수 있다.
- 기본 도형을 화면에 출력할 때 필요한 요소와 함수를 사용할 수 있다.

- **내용**

- 출력 영역 얻기
- 텍스트 출력하기
- 키보드 메시지 처리하기
- Caret 이용하기
- 직선, 원, 사각형, 다각형 그리기

# 윈도우 프로그램 특징

- 메시지 기반의 프로그램 진행



- 윈도우 프로그램은 윈도우에서 발생하는 이벤트에 의한 메시지를 처리하며 프로그램이 진행된다.

# 윈도우 메시지들

- 마우스/키보드 메시지, 윈도우 메시지, 시스템 메시지 등 수백 개의 메시지가 발생되어 처리됨

```
#include <windows.h>          //--- 윈도우 헤더 파일
#include <tchar.h>

HINSTANCE g_hInst;
LPCTSTR lpszClass = L"Window_Class Name";
LPCTSTR lpszWindowName = L"Window Programming Lab";

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WNDCLASSEX WndClass;
    g_hInst = hInstance;

    WndClass.cbSize = sizeof(WndClass);
    WndClass.style = CS_HREDRAW | CS_VREDRAW;
    WndClass.lpfnWndProc = (WNDPROC)WndProc;
    WndClass.cbClsExtra = 0;
    WndClass.cbWndExtra = 0;
    WndClass.hInstance = hInstance;
    WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    WndClass.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    WndClass.lpszMenuName = NULL;
    WndClass.lpszClassName = lpszClass;
    WndClass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    RegisterClassEx(&WndClass);

    hWnd = CreateWindow (lpszClass, lpszWindowName, WS_OVERLAPPEDWINDOW, 0, 0, 800, 600, NULL, (HMENU)NULL, hInstance, NULL);
    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);

    while (GetMessage (&Message, 0, 0, 0)) {
        TranslateMessage (&Message);
        DispatchMessage (&Message);
    }
    return Message.wParam;
}
```

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hDC;

    //--- 메시지 처리하기
    switch (uMsg) {
        case WM_CREATE:
            break;
        case WM_PAINT:
            hDC = BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
    return DefWindowProc (hWnd, uMsg, wParam, lParam);
}
```

## - 메인 함수

**int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)**

- hInstance: 현재 실행중인 어플리케이션의 인스턴스 핸들
- hPrevInstance: 동일한 어플리케이션이 실행중일 경우 이전에 실행된 프로그램의 인스턴스 핸들.
- lpszCmdLine: 커맨드라인 상에서 프로그램 구동 시 전달된 문자열
- nCmdShow: 윈도우가 화면에 출력될 형태

## - 메시지 처리 함수

**LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)**

- hWnd: 메시지가 발생한 윈도우 핸들
- iMsg: 발생한 메시지 (위의 메시지들이 전달된다)**
- wParam, lParam: 메시지를 처리하기 위해 필요한 데이터들, 메시지들마다 다른 값이 전달된다.

## 윈도우 메시지들

- 마우스/키보드 메시지, 윈도우 메시지, 시스템 메시지 등 수백 개의 메시지가 발생되어 처리됨

WM\_ACTIVATE  
WM\_ACTIVATEAPP  
WM\_APP  
WM\_APPCOMMAND  
WM\_CHAR  
WM\_CHARTOITEM  
WM\_CHILDACTIVATE  
WM\_CLEAR  
WM\_CLOSE  
WM\_COMMAND  
WM\_COMPACTING  
WM\_COMPAREITEM  
WM\_COPY  
WM\_CREATE  
WM\_CTLCOLORBTN  
WM\_CTLCOLORDLG  
WM\_CTLCOLOREDIT  
WM\_CTLCOLORLISTBOX  
WM\_CTLCOLORSCROLLBAR  
WM\_CTLCOLORSTATIC  
WM\_CUT  
WM\_DEADCHAR  
WM\_DELETEITEM  
WM\_DESTROY  
WM\_DRAWITEM  
WM\_DROPFILES  
WM\_ENABLE  
WM\_ENDSESSION  
WM\_ENTERIDLE  
WM\_ERASEBKGND  
WM\_FONTCHANGE  
WM\_GETDLGCODE  
WM\_GETFONT  
WM\_GETMINMAXINFO  
WM\_GETTEXT  
WM\_GETTEXTLENGTH  
WM\_HSCROLL

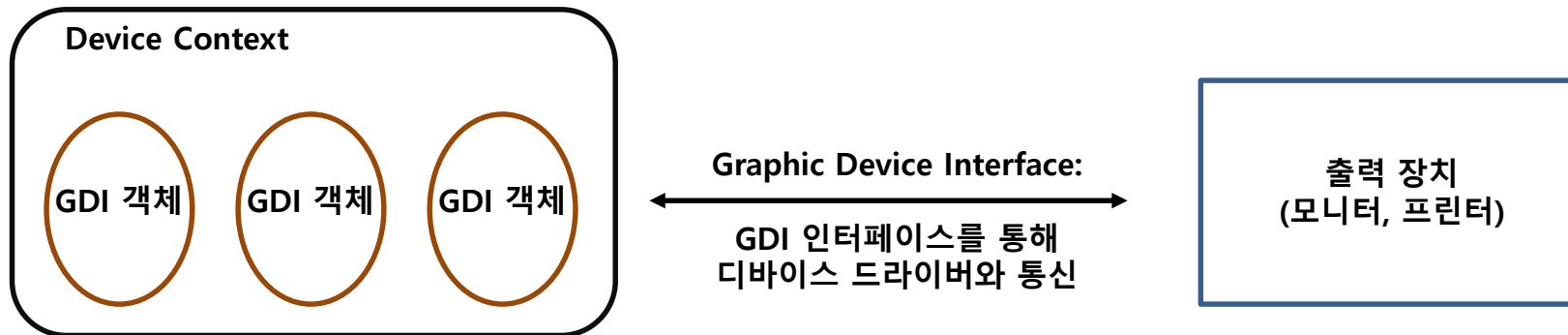
WM\_ICONERASEBKGND  
WM\_INITDIALOG  
WM\_INITMENU  
WM\_INITMENUPOPUP  
WM\_KEYDOWN  
WM\_KEYUP  
WM\_KILLFOCUS  
WM\_LBUTTONDBLCLK  
WM\_LBUTTONDOWN  
WM\_LBUTTONUP  
WM\_MENUCHAR  
WM\_MENUSELECT  
WM\_MOUSEACTIVATE  
WM\_MOUSEMOVE  
WM\_MOVE  
WM\_NCACTIVATE  
WM\_NCCALCSIZE  
WM\_NCCREATE  
WM\_NCDESTROY  
WM\_NCHITTEST  
WM\_NEXTDLGCTL  
WM\_NULL  
WM\_OTHERWINDOWCREATED  
WM\_OTHERWINDOWDESTROYED  
WM\_PAINT  
WM\_PAINTCLIPBOARD  
WM\_PAINTICON  
WM\_PALETTECHANGED  
WM\_PARENTNOTIFY  
WM\_PASTE  
WM\_QUERYDRAGICON  
WM\_QUERYENDSESSION

WM\_QUERYNEWPALETTE  
WM\_QUERYOPEN  
WM\_QUIT  
WM\_RBUTTONDOWN  
WM\_RBUTTONUP  
WM\_RENDERFORMAT  
WM\_SETCURSOR  
WM\_SETFOCUS  
WM\_SETFONT  
WM\_SERREDRAW  
WM\_SETTEXT  
WM\_SHOWWINDOW  
WM\_SIZE  
WM\_SIZECLIPBOARD  
WM\_SYSCHAR  
WM\_SYSCOLORCHANGE  
WM\_SYSCOMMAND  
WM\_SYSDEADCHAR  
WM\_SYSKEYDOWN  
WM\_SYSKEYUP  
WM\_TIMECHANGE  
WM\_TIMER  
WM\_UNDO  
WM\_USER  
WM\_VKEYTOITEM  
WM\_VSCROLL  
WM\_WINIINICHANGE

# 1. 출력 영역 얻기

- **GDI (Graphic Device Interface)**

- 화면, 프린터와 같은 모든 출력 장치를 제어하는 인터페이스
- 윈도우 프로그램에서의 모든 출력은 GDI를 통해서 화면과 프린터로 나가게 되어 있다.
- GDI 객체: 그래픽 출력에 사용되는 도구
  - GDI 객체 종류: 펜, 브러시, 비트맵, 폰트 등
  - GDI 객체들은 핸들을 이용해서 사용한다.
- DC (Device Context): GDI 오브젝트를 모아놓은 것
- GDI는 현재 DC에 선택되어 있는 GDI 오브젝트를 사용한다.



# 1. 출력 영역 얻기

- **GDI+**

- C++ 기반으로 클래스의 집합체
- 모든 그리기 함수들은 Graphics 클래스의 멤버 함수로 정의
- 출력 함수들은 그리기에 필요한 값들을 인수로 전달받아 사용한다.
- C++ 의 함수오버로딩 기능을 사용하고 있어서 여러 종류 인수 타입이 지원된다.
- 함수가 종료될 때 자동으로 파괴자가 호출되므로 객체 해제를 직접 작성할 필요가 없다.
- 동적 할당한 경우에는 삭제해야한다.
- 모든 문자열은 유니코드여야 한다.
- GDI에 비해 많은 기능이 추가로 제공된다.
- 여러 다양한 기능이 추가되고 사용도 편리하지만 GDI에 비해 속도가 느리다.
- 이번 학기는 GDI를 기본으로 사용하기로 한다!

# 디바이스 컨텍스트: 출력 영역 얻기

- **DC (Device Context)**

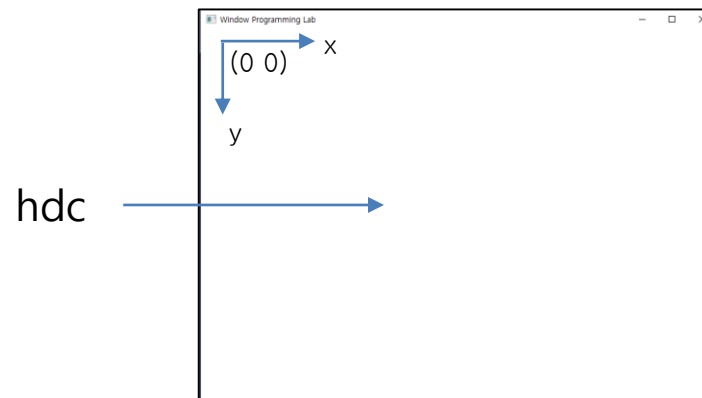
- 디스플레이 또는 프린터와 같은 출력에 필요한 정보를 포함하는 윈도우 데이터 구조체
  - 그래픽 관련한 선택 정보 (폰트, 색상, 굵기, 무늬, 출력 방법 등)를 모아 놓은 구조체
- 모든 그리기는 디바이스 컨텍스트 개체를 통해 수행된다.
  - 디바이스 컨텍스트는 GDI 모듈에 의해서 관리된다.
  - 간단한 출력은 디폴트로 설정된 속성 (예를 들어 선을 그린다면 선의 색상, 굵기, 모양 등) 이용
  - 속성은 얻어온 DC에서 변경 가능: 선의 형태, 색깔, 굵기, 모드, 위치 등의 속성 변경 가능

- 윈도우에서 출력 장치에 무언가 출력하기 위해서는 반드시 DC가 필요, DC 핸들을 얻은 후 해당 DC에 데이터를 출력한다.

- 윈도우의 화면 메모리에 그리고 그것들을 윈도우 운영체제에서 출력시켜준다.
- 이러한 화면 메모리를 제어하는 것이 DC
- **모든 그래픽 출력에 있어서 각각의 윈도우는 모두 DC 핸들(HDC)을 얻어야 한다.**
- DC 핸들은 출력대상을 나타내는 구분 번호로 생각
- 모든 GDI 함수들은 첫 번째 인자로 DC 핸들을 필요로 한다.

- DC의 유형

- 화면 출력을 위한 디스플레이 DC
- 프린터나 플로터 출력을 위한 프린터 DC
- 비트맵 출력을 위한 메모리 DC
- 디바이스 정보를 얻기 위한 정보 DC





# 디바이스 컨텍스트 얻어오기

- 디바이스 컨텍스트 얻어오는 방법
  - 다양한 함수 호출을 통하여 디바이스 컨텍스트를 얻어온다.

DC 얻어오는 함수	DC 반환하는 함수	기능
BeginPaint()	EndPoint():	WM_PAINT메시지에서 DC를 얻거나 반환할 때 사용
GetDC()	ReleaseDC():	WM_PAINT 메시지 외의 일반 메시지에서 DC를 얻거나 반환할 때 사용
CreateDC()	DeleteDC():	DC를 만들어 사용
CreateIC()	DeleteDC():	DC에 출력하지 않고 정보만 얻고자 할 때 사용
CreateCompatibleDC()	DeleteDC():	이미 있는 DC와 같은 또 하나의 DC만들 때 사용. 보통 디스플레이를 이용한 메모리 DC를 만들 때 사용

# 디바이스 컨텍스트 얻어오기

- 일반 메시지에서 DC를 얻어오는 GetDC() 함수 / 해제하는 ReleaseDC () 함수

**HDC GetDC (HWND hWnd);**

- HWND hWnd: 생성된 윈도우의 핸들값
- 리턴 값으로 디바이스 컨텍스트 핸들을 얻어온다.

**int ReleaseDC (HWND hWnd, HDC hDC);**

- HWND hWnd: 해제할 DC에 대한 윈도우 핸들
- HDC hDC: 해제할 DC 핸들

# 디바이스 컨텍스트 얻어오기

- WM\_PAINT 메시지에서 DC를 얻어오는 BeginPaint() 함수 / 해제하는 EndPaint () 함수

**HDC BeginPaint (HWND hWnd, PAINTSTRUCT \*lpPaint);**

- HWND hWnd: 생성된 윈도우의 핸들값
- PAINTSTRUCT \*lpPaint: 출력될 영역에 대한 정보를 저장한 구조체 공간에 대한 주소
- 리턴 값으로 디바이스 컨텍스트 핸들을 얻어온다

**BOOL EndPaint (HWND hWnd, PAINTSTRUCT \*lpPaint);**

- HWND hWnd: 생성된 윈도우의 핸들값
- PAINTSTRUCT \*lpPaint: 출력될 영역에 대한 정보를 저장한 구조체 공간에 대한 주소

- PAINTSTRUCT 구조체

```
typedef struct tagPAINTSTRUCT {  
    HDC hdc;                //---그리고자 하는 DC 핸들  
    BOOL fErase;            //--- 배경 삭제 여부를 가리킨다: 0이 아니면 다시 그린다.  
    RECT rcPaint;           //--- 다시 그릴 사각형의 좌표값  
    BOOL fRestore;  
    BOOL fIncUpdate;  
    BYTE rgbReserved[16];  
} PAINTSTRUCT;
```

# 디바이스 컨텍스트 얻어오기

- 사용 예)

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    ...
}

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT ps;

    switch (iMsg)
    {
        case WM_PAINT:
            hDC = BeginPaint (hWnd, &ps);
            EndPaint (hWnd, &ps);
            break;

        case WM_TIMER:
            hDC = GetDC (hWnd);
            ReleaseDC (hWnd, hDC);
            break;

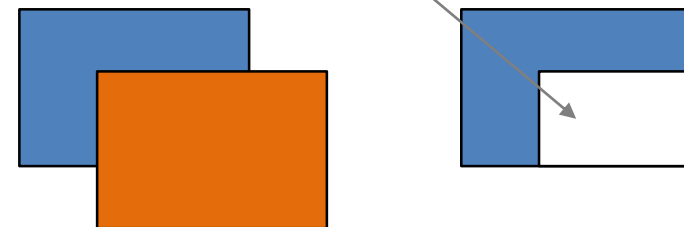
        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
    }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```

//--- WM\_PAINT 메시지: dc를 얻어 그리기 진행  
//--- BeginPaint 함수를 사용하여 DC를 얻는다

//--- WM\_TIMER 메시지: dc를 얻어 출력 진행  
//--- GetDC 함수를 사용하여 DC를 얻는다

## 2. 메시지 처리: WM\_PAINT 메시지

- 클라이언트 영역에 그릴 필요가 있을 때: **WM\_PAINT** 메시지 사용
  - 원도우의 크기가 변경되었을 때나 다른 윈도우에 가려져 있다가 드러날 때 등 화면에 출력된 결과가 깨질 수 있다.
  - 화면의 그래픽 일부가 깨지거나 다시 출력해야 할 필요가 있는 영역을 **무효화(invalid) 영역**이라 하고 이러한 경우를 화면이 무효화 되었다고 한다.
  - OS는 깨진 화면을 복구해주지 않는다.
  - 단지 OS는 화면이 깨질 때 마다 **WM\_PAINT** 메시지를 발생시켜준다.
    - 즉, 윈도우의 클라이언트 영역 중 일부가 무효화(invalid)되면 OS가 WM\_PAINT 메시지를 큐에 넣어준다.
  - 그래서 **출력은 WM\_PAINT 메시지 아래에서 해야 한다!!**
  - 그래야 화면이 깨질 때 마다 WM\_PAINT 메시지가 발생하고 그 아래에 작성한 소스가 다시 실행되어 화면이 복구된다!
- 다음과 같은 경우에 OS는 WM\_PAINT 메시지를 프로그램에 전달한다.
  - 원도우가 처음 생성되었을 때
  - 원도우의 위치가 이동되었을 때
  - 원도우의 크기가 변경되었을 때
  - 최대, 최소화되었을 때
  - 다른 윈도우에 가려져 있다가 드러날 때
  - 파일로부터 데이터를 출력할 때
  - 출력된 데이터의 일부분을 스크롤, 선택, 변화시킬 때
  - InvalidateRect()**, 또는 **InvalidateRgn()** 함수를 호출하여 강제로 화면을 무효화시킬 때



# 메시지 처리: WM\_PAINT 메시지

- WM\_PAINT 메시지는
  - 이 메시지를 받았을 때 프로그램은 화면 복구를 위해 클라이언트 영역 전체 또는 무효화된 부분만 다시 그려야 한다.
    - OS는 화면이 무효화될 때 클라이언트 영역을 복구해 주지 않는 대신에 이 메시지를 보내 줌으로써 해당 프로그램에게 다시 그려야 할 시점을 알려 준다.
    - 따라서 클라이언트 영역에 출력한 정보는 모두 저장해 두어야 복구가 가능하다.
  - WM\_PAINT 메시지는 모든 메시지 중에서 우선 순위가 가장 낮다.
    - GetMessage() 함수는 메시지 큐에 WM\_PAINT 메시지가 있더라도 다른 메시지가 대기 중이면 그 메시지를 먼저 처리한다.
    - WM\_PAINT 메시지는 큐에 대기 중인 다른 메시지가 없고 무효화 영역이 존재할 때만 윈도우 프로시저로 보내진다.
  - WM\_PAINT 메시지는 한번에 하나만 메시지 큐에 들어갈 수 있다.
    - 만약 무효화 영역이 생겼는데 WM\_PAINT 메시지가 이미 메시지 큐에 있으면 기존의 무효화 영역과 새 무효화 영역의 합으로 새로운 무효화 영역이 설정된다.
- 해당 윈도우 프로시저에서 이 메시지를 처리하지 않으면 이 메시지는 [DefWindowProc\(\)](#) 함수가 처리한다.
  - 이 함수는 무효 영역을 모두 유효화(valid)하며 다시 그리기는 하지 않는다.
- WM\_PAINT 메시지에서 그리기를 할 때는 BeginPaint()와 EndPaint() 함수를 사용해야 한다.
  - 이 두 함수는 WM\_PAINT 메시지 내에서만 사용된다.
  - 다시 그려야 할 영역에 대한 정확한 좌표를 조사하며 무효 영역을 유효화하고 캐럿을 숨기거나 배경을 지우는 등의 꼭 필요한 동작을 한다.
- 다른 메시지에서도 출력은 가능하다!

### 3. 메시지 처리: WM\_CREATE / WM\_QUIT 메시지

- 윈도우가 생성될 때: **WM\_CREATE** 메시지
  - CreateWindow 함수에 의해 윈도우가 생성될 때 호출되는 메시지
  - 메모리에 윈도우를 생성한 후 화면에 보이기 전에 보내지며 주로 윈도우에 관련된 초기화 작업을 할 때 사용됨
  - 윈도우 동작을 위한 메모리 할당, 리소스 생성, 차일드 컨트롤 생성, 윈도우 속성 초기화 작업에 이 메시지가 사용된다.
  - CreateWindow 함수는 이 메시지를 완전히 처리한 후에 리턴
- 윈도우를 파괴할 때: **WM\_DESTROY** 메시지
- 프로그램을 종료할 때: **WM\_QUIT** 메시지
  - 프로그램을 종료하고 윈도우를 파괴할 때 호출되는 메시지
    - WM\_DESTROY: 사용자가 시스템 메뉴를 더블 클릭하거나 Alt+F4를 눌러 프로그램을 끝내려고 할 때 발생하는 메시지
    - WM\_QUIT: PostQuitMessage 함수를 호출하면 발생한다. **WM\_QUIT** 메시지가 입력되면 메시지 루프의 GetMessage 함수 리턴값이 False가 되어 프로그램이 종료된다. 윈도우에 전달되는 메시지는 아님.
  - **PostQuitMessage ()** 함수를 호출하여 프로그램을 종료한다.
    - 파괴되는 윈도우가 메인 윈도우일 경우에는 반드시 **PostQuitMessage** 함수를 호출하여 메시지 루프를 종료하도록 한다.
  - 윈도우를 닫는 함수: **DestroyWindow (HWND hWnd);**

### 3. 메시지 처리: WM\_CREATE / WM\_QUIT 메시지

- 프로그램 종료 함수

**VOID PostQuitMessage (int nExitCode);**

- 스레드 메시지 큐에 WM\_QUIT 메시지를 붙이고 즉시 리턴
- WM\_QUIT 메시지를 큐에 붙임으로써 시스템에게 이 스레드가 종료될 것이라는 것을 미리 알려준다.
- 메시지 루프는 보통 WM\_QUIT 메시지를 받으면 종료하도록 되어 있으므로 이 함수를 호출하면 메시지 루프가 종료된다.

**BOOL DestroyWindow (HWND hWnd);**

- hWnd 윈도우를 파괴한다. 단, 이 함수로 다른 스레드에서 생성한 윈도우를 파괴할 수는 없다



# 메시지 처리: WM\_PAINT/WM\_CREATE/WM\_DESTROY

- 사용 예)

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT ps;

    switch (iMsg)
    {
        case WM_CREATE:                                     //--- WM_CREATE 메시지: 윈도우 생성될 때 호출, 필요한 초기화 작업
            break;

        case WM_PAINT:                                       //--- WM_PAINT 메시지: dc를 얻어 그리기 진행
            hDC = BeginPaint (hWnd, &ps) ;                  //--- 필요한 그리기를 실행한다.

            EndPaint (hWnd, &ps) ;

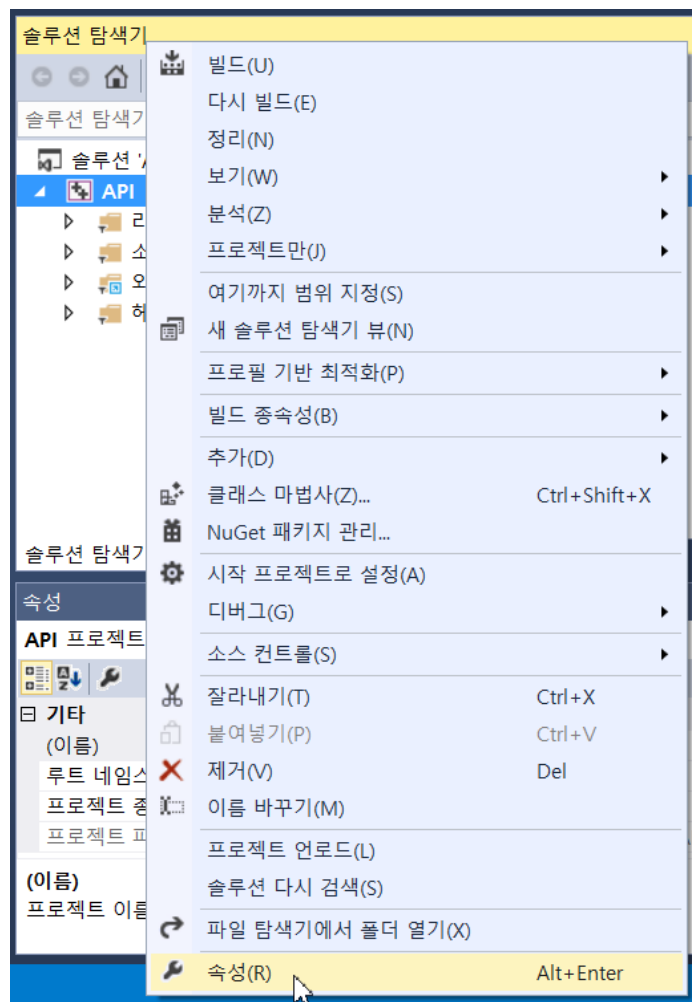
            break;

        case WM_DESTROY:                                     //--- 프로그램 종료
            PostQuitMessage ( 0 );

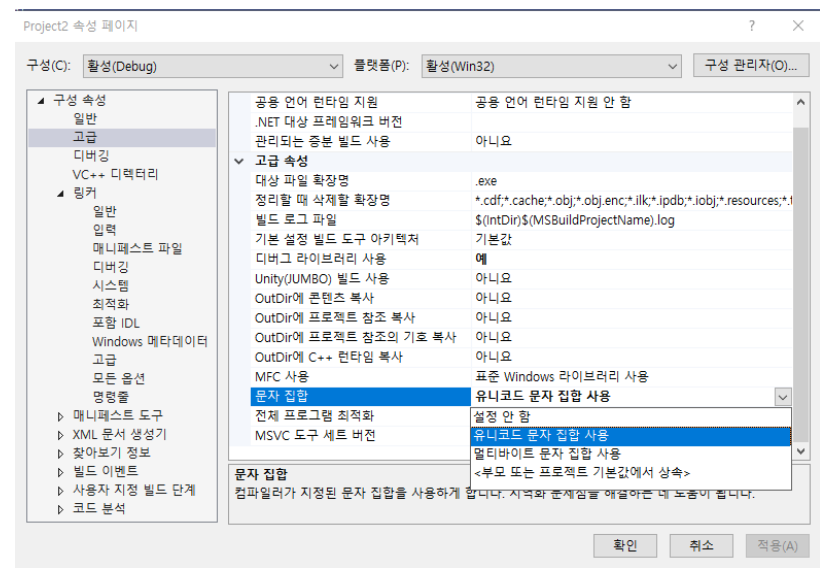
            break;
    }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```

# 4. 문자 집합

- 현재 설정된 문자 집합 보기
  - 프로젝트 이름에서 마우스 오른쪽 버튼을 누른 후, 속성 선택



- 기본 설정은 유니코드
  - 고급 → 문자집합 → 유니코드/멀티바이트



# 멀티 바이트 문자 집합(MBCS) 사용

- 멀티바이트 형의 문자 집합

- 한 글자를 저장하기 위해 2바이트 이상을 사용할 수 있다.
- 실제 글자의 개수와 공간의 크기가 일치하지 않을 수 있다.
- 메모리 낭비가 없어 효율적인 장점

- 영어 알파벳만 사용 했을 때

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

I		I	o	v	e		y	o	u	₩0				
---	--	---	---	---	---	--	---	---	---	----	--	--	--	--

- 한글 혼용(멀티 바이트) 했을 때

- `char str[15] = "나는 love";`

나	는		I	o	v	e	₩0							
---	---	--	---	---	---	---	----	--	--	--	--	--	--	--

# 유니코드 문자 집합 사용

- 유니코드형의 문자 집합
  - 영어나 한글 구분없이 모든 문자를 2바이트 사용하여 저장
  - 위치를 쉽게 알 수 있다.
  - 메모리 낭비가 심하다

- 한글 혼용 했을 때
  - `wchar str[15] = L"나는 love";` //L: 유니코드로 변환

나	는		₩0		₩0	o	₩0	v	₩0	e	₩0	₩0
---	---	--	----	--	----	---	----	---	----	---	----	----

# 유니코드 문자 집합 사용

- 문자집합 설정을 어떻게 해도 처리되는 자료형

- TCHAR 형
- 멀티바이트이면 TCHAR는 char로 변환
- 유니코드이면 TCHAR는 wchar로 변환

- 사용 예

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

```
//--- 헤더 파일 include
```

```
TCHAR str_1[15] = _T("나는 winple");
```

```
TCHAR str_2[15] = L"나는 winple";
```

```
TCHAR str_3[15] = TEXT("나는 winple");
```

//--- 위의 세개의 매크로 함수 (\_T, L, TEXT) 모두 사용 가능

# 문자열 자료형

- 문자열 자료형

API 자료형	같은 의미의 자료형	설명
LPSTR	char *	ANSI 코드 문자열 포인터
LPCSTR	const char *	ANSI 코드 문자열 포인터 상수
LPTSTR	TCHAR *	TCHAR 문자열 포인터
LPCTSTR	const TCHAR *	TCHAR 문자열 포인터 상수
LPWSTR	WCHAR *	유니코드 문자열 포인터
LPCWSTR	const WCHAR *	유니코드 문자열 포인터 상수

- LP : long pointer
- C : const
- T : TCHAR (유니코드, 멀티바이트 모두 지원)
- W: WCHAR (유니코드 문자열)
- STR : 문자열

## 5. 텍스트 출력하기

- 한 점 기준 텍스트 출력 함수

**BOOL TextOut (HDC hDC, int x, int y, LPCTSTR lpString, int nLength);**

- HDC hDC: BeginPaint()나 GetDC()를 통해 얻어온 DC핸들
- int x, int y: 텍스트를 출력할 좌표의 x값과 y값
- LPCTSTR lpString: 출력할 텍스트
- int nLength: 출력할 텍스트의 길이

- 문자열 관련 함수들:

내용	멀티바이트	TCHAR	유니코드	API 문자열	사용 예
문자열 복사	strcpy	_tcscopy	wscopy	lstrcpy	strcpy(a, b)
문자열을 c 만큼 복사	strncpy	_tcsncpy	wcsncpy	lstrcpyn	strncpy (a, b, c)
문자열 길이	strlen	_tcslen	wcslen	lstrlen	strlen(a)
문자열 붙이기	strcat	_tcscat	wcscat	lstrcat	strcat (a, b)

# 윈도우에 “Hello World” 출력하기

- 윈도우 프로시저 함수 안에서 WM\_PAINT 메시지에서 출력 처리

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    ...
}
```

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

```
    HDC hDC;
    PAINTSTRUCT ps;
    TCHAR str[100] = L"Second Hello World";
    int num;
```

```
    switch (iMsg)
    {
```

```
        case WM_PAINT:
            hDC = BeginPaint (hWnd, &ps);
```

```
            num = _tcslen (str);
```

```
            TextOut (hDC, 0, 0, L"Hello World", strlen("Hello World")); //--- 문자열 직접 사용
```

```
            TextOut (hDC, 0, 100, str, num); //--- 변수 사용
```

```
            EndPaint (hWnd, &ps);
            break;
```

```
        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
```

```
    }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```





# 박스 영역에 텍스트 출력 함수: DrawText ()

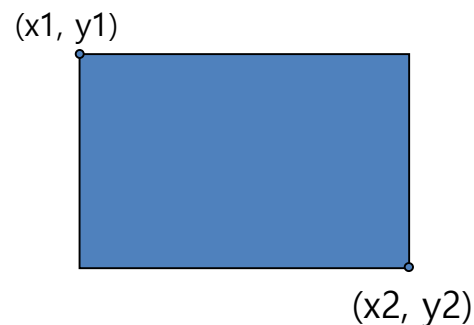
- 박스 영역에 텍스트 출력 함수

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

- HDC hDC: BeginPaint()나 GetDC()를 통해 얻어온 DC핸들
- LPCSTR lpString: 출력 문자열
- int nLength: 문자열 길이
- LPRECT lpRect: 문자열을 출력할 박스 영역 구조체의 주소 (RECT \*)
- UINT Flags: 출력 방법

- RECT 구조체

```
typedef struct tagRECT {  
    LONG left;           //--- x1  
    LONG top;            //--- y1  
    LONG right;          //--- x2  
    LONG bottom;         //--- y2  
} RECT;
```



# DrawText() 출력 방법

- **DrawText 함수의 Flag:**
  - DT\_SINGLELINE: 박스 영역 안에 한 줄로 출력 (세로에 해당하는 플래그 사용 시 함께 사용)
  - DT\_LEFT: 박스 영역 내에서 왼쪽 정렬
  - DT\_CENTER: 박스 영역 내에서 가운데 정렬
  - DT\_RIGHT: 박스 영역 내에서 오른쪽 정렬
  - DT\_VCENTER: 박스 영역의 상하에서 가운데 출력 (DT\_SINGLELINE 과 함께 사용)
  - DT\_TOP: 박스 영역의 상하에서 위쪽에 출력
  - DT\_BOTTOM: 박스 영역의 상하에서 아래쪽에 출력
  - DT\_WORDBREAK: 단어가 박스 영역의 오른쪽 끝에 닿으면 자동 개행되도록 한다.
  - DT\_EDITCONTROL: DT\_WORDBREAK에 OR해주면 영문,한글,숫자 모두 글자 단위로 개행
- **사용 예) 1개 이상의 설정을 하는 경우: 사각형 영역의 가운데 출력**

```
TCHAR szText[] = _T("Hello world");  
RECT rect = {0, 0, 800, 600};  
  
DrawText (hDC, szText, _tcslen(szText), &rect, DT_VCENTER | DT_CENTER | DT_SINGLELINE);
```
- **사용 예) 긴 문자열을 사각 영역에 맞추어 출력하는 경우**

```
TCHAR szText[] = _T("HelloWorld HelloWorld HelloWorld HelloWorld HelloWorld");  
RECT rect = {0, 0, 20, 600};  
  
DrawText (hDC, szText, _tcslen(szText), &rect, DT_WORDBREAK | DT_EDITCONTROL);
```

# DrawText() 함수 이용하기

- DrawText 함수를 이용하여 HelloWorld 출력

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT ps;
    RECT rect;

    switch (iMsg)
    {
        case WM_PAINT:
            hDC = BeginPaint (hWnd, &ps) ;

            rect.left = 50;           //--- 사각형 정의
            rect.top = 40;
            rect.right = 200;
            rect.bottom = 120;

            DrawText (hDC, L"HelloWorld", 10, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
            EndPaint (hWnd, &ps) ;
            break;

        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
    }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```



//--- 한 라인, 수직/수평 중앙

# <문자열 만들기>

- 문자열 만들기

- int **wsprintf** (LPTSTR lpOut, LPCTSTR lpFmt...);
  - 서식화된 문자열을 버퍼에 저장한다.
- 사용 예)

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT ps;
    TCHAR lpOut[100];
    int x=0, y=0;

    switch (iMsg)
    {
        case WM_PAINT:
            hDC = BeginPaint (hWnd, &ps);
            wsprintf (lpOut, L"%d * %d = %d", x, y, x * y);
            TextOut (hDC, x, y, lpOut, lstrlen (lpOut));
            EndPaint (hWnd, &ps);
            break;

        case WM_DESTROY:
            PostQuitMessage ( 0 );
            break;
    }
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```

# 문자 출력 시 배경색, 전경색 모드 지정

- 문자 색 및 배경색 변경 해주는 함수

**COLORREF SetBkColor** (HDC hDC, COLORREF crColor);

- hDC: 디바이스 컨텍스트 핸들
- crColor: 변경 할 배경색

**COLORREF SetTextColor** (HDC hDC, COLORREF crColor);

- hDC : 디바이스 컨텍스트 핸들
- crColor: 변경 할 문자색

- 사용 예) 앞 페이지의 예에서

case **WM\_PAINT**:

hDC = BeginPaint (hWnd, &ps) ;

rect.left = 50; //--- 사각형 정의

rect.top = 40;

rect.right = 200;

rect.bottom = 120;

**SetTextColor** (hDC, RGB (255, 0, 0));

//--- 문자 출력 이전에 문자 색상 설정, 설정 이후의 문자는 빨간색으로 출력된다.

DrawText (hDC, L"HelloWorld", 10, &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

//--- 한 라인, 수직/수평 중앙

EndPaint (hWnd, &ps) ;

break;

# 문자 출력 시 배경색, 전경색 모드 지정

- 윈도우의 색 지정: RGB(Red, Green, Blue) 삼원색 사용

**COLORREF RGB (BYTE R, BYTE G, BYTE B);**

- R, G, B: 빛의 3원색으로 0 ~ 255 사이의 정수값
- 0 ~ 16777215 사이의 색상값 설정

- COLORREF:RGB 색상을 지정하는데 사용되는 DWORD 형태의 타입
  - 16진수 형태
  - 0x00bbggrr의 값으로 저장됨
  - r, g, b 값을 얻으려면 각각 GetRValue, GetGValue, GetBValue 함수를 사용
- RGB 매크로 함수는 빨강, 초록, 파랑 색의 값을 인자로 받아 COLORREF 타입의 색상값으로 만든다.

```
COLORREF RGB {  
    BYTE byRed,           // 색상 중 빨간 색  
    BYTE byGreen,         // 색상 중 초록 색  
    BYTE byBlue           // 색상 중 파란 색  
}
```

- 사용 예)

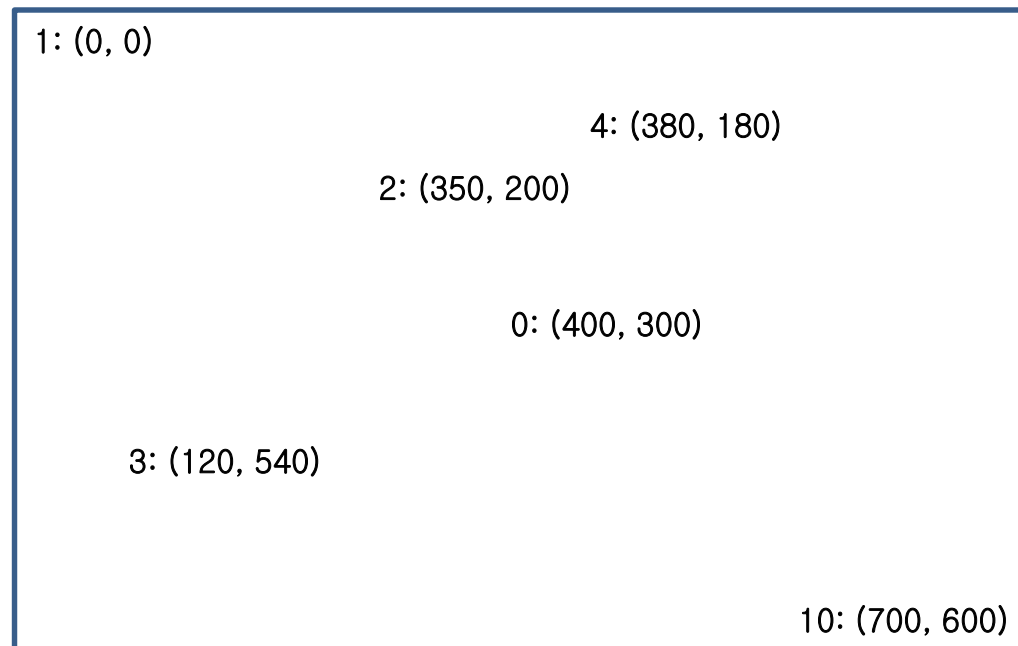
```
COLORREF text_color ;  
text_color = RGB (255, 0, 0);
```

```
SetTextColor (hDC, text_color );
```

```
DrawText (hDC, "HelloWorld", 10, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
```

## 실습 2-1

- 화면의 코너와 중앙에 문자 그리기
  - 윈도우를 800x600 크기로 띄운다
  - 윈도우의 열 개의 위치를 랜덤하게 설정한 후 그 위치에 문장을 쓴다.
    - 랜덤한 위치는 10의 배수로 설정한다.
      - 위치는 중복되지 않고, 앞 뒤로 10이내에 들어가지 않는다.
    - 문자 내용: "1: (0, 0)", "2: (350, 200)", "3: (120, 540)", "4: (380, 180)", ..., "10: (700, 600)"
    - 좌표값: 각각의 문자가 시작하는 위치의 좌표값 (x, y)
  - 윈도우의 중앙에 한 문장을 쓴다.
    - 화면의 중앙에 "0: (400, 300)"이라는 문장을 쓴다.



- 화면을 등분하여 문자 그리기
  - 윈도우를 800x600 크기로 띄운다.
  - 화면의 가로 세로를 각각  $n \times m$  등분하여  $n \times m$ 개의 구간으로 만든 후 각 구간에 문자 그리기
    - 등분 수는  $2 \leq n \leq 10, 2 \leq m \leq 10$  로 랜덤하게 정한다.
  - 각 구간의 문자색과 배경색은 구간 별로 랜덤한 색으로 설정한다.

예)  $n: 3, m: 2$ 일 때 가로를 3등분, 세로를 2등분 한 후 문자를 그린다.

abcdefghijkl hijklmnopqr stuvwxyzab cdefghij	ABCDEFGH IJKLMNOP OPQRSTU VWXY	abcdefghijkl hijklmnopqr stuvwxyzab cdefghij
ABCDEFGH IJKLMNOP OPQRSTU VWXYZAB	abcdefghijkl klmnopqrs tuvwxyzab cdefghij	ABCDEFGH IJKLMNOP OPQRST UVWXY