Windows API를 이용한 윈도우 만들기

Windows 운영체제의 게임을 만들기 위해선 먼저 운영체제에 대해서 알아야 합니다. 일단 기본적인 것들을 생각해 보세요. 가장 먼저 윈도우를 하나 띄워야 합니다. 이번 장에서 윈도우를 띄운다는 간단한 개념에서 출발하여 차차 다른 장에서 추가적인 운영체제의 기능들을 사용하는 방법을 익혀나갈 것입니다. 코드가 길다고 해서 당황하지 마세요, 이제 시작일 뿐입니다.

Windows 운영체제에서 GUI 프로그래밍을 하는 방법에는 여러가지가 있지만, 가장 기본적인방법은 Windows API를 사용하는 것입니다. Windows API는 Microsoft에서 제공하는 Windows 운영체제에서 사용 가능한 기능들을 모은 API(Application Programming Interface)입니다. API는 프로그래밍에 필요한 기능들을 모아둔 도구라고 생각하면 됩니다. 기본적으로 윈도우를 띄우고 GDI라는 기능을 통해서 그래픽을 그릴 수 있지만, GDI는 CPU를 사용하기 때문에 속도가 느립니다. 단순 윈도우를 관리하는 것뿐만 아니라 레지스트리(Registry)나 파일 입출력, 기타 Windows 운영체제와 관련된 모든 기능들이 들어 있습니다. 기본적으로 Visual Studio를 사용한다면 Windows SDK가 설치되는데, 그 안에 들어 있습니다. 아니면 Microsoft 홈페이지에서 다운로드 할 수도 있습니다[1].

준비가 되었다면 이제 프로그래밍을 시작해 봅시다, 먼저 프로젝트를 생성해야 합니다. 주의할 점은 프로젝트를 생성할 때 Win32(혹은 Win64) 콘솔 응용프로그램이 아닌 Win32 프로젝트를 선택해야 한다는 것입니다. 콘솔 응용프로그램을 선택한 경우 자동적으로 콘솔 창을 생성하는 과정이 추가되고, int main(void) 형식의 함수가 진입점 함수가 됩니다. 그러나 Win32 프로젝트는 이와 다릅니다. 생성 단계 중 프로젝트를 빈 프로젝트로 지정해서 잡다한 코드가 생기지 않게 합니다.

먼저 Windows.h를 포함해야 합니다. 그리고 진입점 함수의 형식은 기존의 콘솔 응용프로그램과는 다릅니다. 일단 아래의 예제를 따라 치고 컴파일 한 뒤 실행해 보세요.

```
#include <Windows.h>

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE, LPSTR, int)
{
	MessageBox(0, L"Hello World", L"Unipen", MB_ICONINFORMATION);
	return 0;
}
```

예제 1. 메시지 상자를 띄우는 예제(SimpleMessageBox.cpp)

먼저 Windows.h 를 포함합니다. 그 뒤에 진입점 함수인 `WinMain`을 선언합니다. Windows 응용프로그램에서는 콘솔 응용 프로그램과 달리 진입점 함수의 이름이 main()이 아닌 WinMain() 입니

[1] Windows 7 용 Windows SDK: http://www.microsoft.com/en-us/download/details.aspx?id=8279

다. 함수의 형태는 위와 같고 반환형은 콘솔 응용프로그램의 `main()`과 같이 종료 코드를 나타냅니다. `WINAPI` 는 매크로로서 `__stdcall`로 지정되어 있습니다. 이는 함수 옵션인데 아직은 몰라도 됩니다. 인자들은 4개가 있는데, 인자들의 형식에 대해서 먼저 알아봅시다.

Windows API의 모든 자료형들은 대문자로 지어져 있습니다. 즉 HINSTANCE와 LPSTR은 자료형입니다. 그 중에서 H로 시작하는 자료형들은 핸들(Handle)이라는 타입으로써, 어떠한 다른 객체를 가리키는 일종의 포인터와 같은 개념입니다. 그러나 실제로 그 값이 무엇을 의미하는지는 우리가 알 수 없습니다. Windows 에는 Windows XP, Windows 7, Windows 8 등등의 많은 운영체제가 있습니다. 모든 OS가 같은 구조를 갖고 있지는 않습니다. 그렇다고 하더라도 동일한 API로 동작해야 개발자들이 편리하게 어플리케이션을 개발할 수 있을 것입니다. 그렇기 때문에 핸들을 통해서 자세한 정보를 은닉한 것입니다.

HINSTANCE는 인스턴스, 즉 응용 프로그램을 나타내는 핸들입니다. 첫 번째 인자로 온 HINSTANCE 값이 현재 응용 프로그램을 나타내는 핸들을 의미합니다. 두 번째 인자는 이전에 실행된 응용 프로그램의 핸들이라고 하는데, Windows가 32Bit 운영체제로 넘어오면서 더 이상은 아무런 값도 주어지지 않습니다. 세 번째 인자는 응용 프로그램을 실행할 때 넘어온 명령 인자들입니다. `LPSTR`은 `const char*` 타입입니다. 네 번째는 윈도우를 생성했을 때 어떻게 표시할 것인지를 나타내는 값입니다. 사실상 모든 인자를 무시해도 되긴 하지만 편의상 첫 번째 인자는 남겼습니다. 첫 번째 인자는 반드시 필요하지만 `(HINSTANCE)GetModuleHandle(0)` 과 같은 방법으로 얻을 수 있습니다.

함수의 본문에서는 MessageBox 라는 함수를 호출합니다. 이 함수는 메세지 상자를 띄우는 함수입니다. 첫 번째 인자는 넘어가고 두 번째 인자는 표시할 문자열, 세 번째 인자는 메시지 상자의 제목이고, 네 번째 인자는 메시지 상자의 종류인데 여기서는 `MB_ICONINFORMATION` 을 주어 알림 상자로 표시하게 하였습니다.

문자열 상수 앞에 대문자 'L'을 붙였는데 이는 절대로 실수가 아닙니다. 문자열 상수 앞에 L을 붙이게 되면 그 문자열은 문자당 16비트인 유니코드로 인식됩니다. 문자를 저장하는 자료형도 char 가 아닌 wchar_t 라는 새로운 2바이트 자료형입니다. 즉 L"Hello World" 는 const wchar_t* 타입입니다.

그림 1. MessageBox() 함수로 띄운 메시지 상자

윈도우 만들기

그렇다면 이번엔 윈도우를 띄워 볼 차례입니다. 다음 예제를 봅시다.

```
#include <Windows.h>

//
// 메세지 프로시저 함수
// 이벤트가 발생했을 때 이 함수가 호출된다.
LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp)
{
     switch(msg){
     // 윈도우가 파괴되었을 때
     case WM_DESTROY:
            // 나가라!
            PostQuitMessage(0);
            return 0;

     // 처리되지 않은 메시지는
     default:
            // 기본 처리 함수를 이용한다.
            return DefWindowProc(hwnd, msg, wp, lp);
     }
}

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE, LPSTR, int)
{
     //
     // 윈도우 클래스를 등록한다.
     WNDCLASS wc = {0};
     wc.lpszClassName = L"Unipen Window";
     wc.style = CS_VREDRAW | CS_HREDRAW;
     wc.lpfnWndProc = WindowProc;
     wc.hInstance = hinst;
     wc.hCursor = LoadCursor(NULL, IDC_ARROW);
     wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
     RegisterClass(&wc);

     HWND hwnd = CreateWindow (
            wc.lpszClassName,   // 창 클래스
            L"SimpleWindow",    // 창 제목
            WS_OVERLAPPEDWINDOW,// 창 스타일
            CW_USEDEFAULT,             // 화면에서의 X 좌표
            CW_USEDEFAULT,             // 화면에서의 Y 좌표
```

```
            CW_USEDEFAULT,              // 가로 크기
            CW_USEDEFAULT,              // 세로 크기
            0,                                  // 부모 윈도우
            0,                                  // 메뉴
            hinst,                      // 인스턴스 핸들
            0                                   // 초기 전달 값
            );
      //
      // 윈도우 창을 띄운다.
      ShowWindow(hwnd, SW_SHOW);

      //
      // 메시지 루프를 돈다.
      MSG msg = {0};
      while(GetMessage(&msg, 0, 0, 0))
      {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
      }
      return 0;
}
```

예제 2. 간단한 윈도우 창 띄우기(SimpleWindow.cpp)

간단한 윈도우 창을 띄우기 위함인데 소스가 길고 알 것도 많습니다. Windows API에 대해서는 많이 다룰 건 아니기 때문에 이것만 외워두면 됩니다! 윈도우 생성에는 2단계를 거칩니다. 첫째는 윈도우 클래스를 등록하는 것이고, 둘째는 윈도우를 생성하는 것입니다. 인자가 많아서 그렇지 딱히 어렵진 않습니다. WindowProc이라는 함수가 추가 되었는데 일단은 WinMain 함수부터 봅시다. 윈도우 클래스란 윈도우에 대한 공통적인 정보가 담긴 데이터를 말합니다. 비단 우리가 생각하는 제목 있고 최소화 최대화 닫기 버튼이 달린 창만이 윈도우가 아닙니다. 그것은 프레임(Frame)이라고 한다. 버튼, 텍스트 입력 창, 리스트, 드롭다운 메뉴 등 모든 것이 윈도우입니다. 이런 윈도우들은 서로 공통적인 부분이 있습니다. Windows API는 초창기 윈도우 운영체제를 만들기 위해 개발되었기 때문에 굉장히 낡았어요. 그래서 이런 불편한 점들이 있습니다.

윈도우 클래스를 등록하는 방법은 WNDCLASS 구조체의 필드에 정보를 채워 넣은 뒤에 RegisterClass() 함수를 통해서 등록하는 것입니다. 구조체의 필드 중 몇 가지만 알아보겠습니다. 나머지 자세한 정보는 MSDN(Microsoft Software Development Network)[2]에 잘 설명되어 있습니다. 영어가 부담스럽다면 soen.kr 같은 곳에서 제공하는 튜토리얼을 보면 됩니다. 일단 위에서 대입한 필드들만 살펴보도록 하겠습니다.

---

[2] Microsoft Developer Network, Microsoft, http://msdn.microsoft.com/

- lpszClassName: 윈도우 클래스를 구분하기 위한 문자열입니다.
- style: 윈도우 클래스의 스타일을 지정하는 항목입니다. 속성은 CS_ 로 시작하며, 여러 개의 속성은 or('|') 연산자로 묶어서 전달해야 합니다. 이 상수들은 각각이 2의 배수로 되어 있습니다. 예를 들어 CS_VREDRAW 는 1이고, CS_HREDRAW는 2인데, 이 둘은 2진수로 각각

  00000001

  00000010

  입니다. 이들을 or 연산하면(CS_HREDRAW | CS_VREDRAW)

  00000011

  이 됩니다. 그러면 여기서 CS_HREDRAW 속성이 있는지 파악할 때 아래처럼 and 연산하면 알아낼 수 있습니다.

  ```
  if( value & CS_HREDRAW )
  ```

  CS_HREDRAW 는 윈도우의 가로 크기나 위치가 변경될 때 윈도우의 내부를 다시 그리라는 의미이며, CS_VREDRAW는 윈도우의 세로 크기나 위치가 변경될 때 윈도우의 내부를 다시 그리라는 의미의 매크로 상수입니다.
- lpfnWndProc: 윈도우를 클릭하거나, 옮기거나, 기타 응용 프로그램에서 다양한 이벤트(Event)가 발생했을 때 이를 처리해야 합니다. 그를 위해서 처리할 코드를 적은 함수를 줘야 합니다. 위 예제에서 WindowProc 함수가 바로 그 역할을 담당합니다. 이벤트가 발생하면 운영체제는 우리가 lpfnWndProc에 저장한 WindowProc() 함수를 실행하게 됩니다.
- hCursor: 윈도우의 커서 아이콘을 지정합니다. LoadCursor() 함수로 윈도우가 제공하거나 자신이 제작한 커서로 직접 지정할 수 있습니다. 운영체제가 제공하는 기본 커서를 사용하려면 첫 번째 인자에 NULL을 주고, 두 번째 인자에 IDC_로 시작하는 커서의 종류를 지정해 주어야 합니다. IDC_ARROW는 윈도우의 기본 화살표 모양의 커서 모양입니다.
- hInstance: 윈도우가 소속될 응용 프로그램의 핸들입니다.
- hbrBackground: 윈도우의 배경 색깔입니다. GetStockObject(WHITE_BRUSH)를 통해서 흰색 브러시를 얻어왔고, 이를 HBRUSH 타입으로 캐스팅해서 대입했습니다.

이후 CreateWindow 함수를 봅시다. 이 함수는 윈도우를 생성해서 반환해 주는 함수입니다. 이 함수들의 인자들이 굉장히 많지만 힘내도록 합시다(하다 보면 외워져요).

1. lpClassName: 적용할 윈도우 클래스의 이름을 넣어주어야 합니다.
2. lpWindowName: 만들 윈도우의 제목을 지정합니다.
3. dwStyle: 만들 윈도우의 스타일을 지정합니다. WS_로 시작하며, 예제에서는

WS_OVERLAPPEDWINDOW 로 지정했는데, 이는 평범한 윈도우를 말합니다. 크기 조절이 가능하고, 최대화/최소화/닫기 버튼이 존재하고, 제목이 있습니다. 작업 표시줄에서 우 클릭을 하면 시스템 팝업메뉴도 뜹니다.

4. x: 윈도우의 화면에서의 X좌표를 지정합니다. CW_USEDEFAULT는 무작위로 한 위치에 생성하라는 의미입니다.
5. Y: 윈도우의 화면에서의 Y좌표를 지정합니다. 역시나 CW_USEDEFAULT가 쓰입니다.
6. nWidth: 윈도우의 너비를 지정합니다. 역시나 CW_USEDEFAULT
7. nHeight: 윈도우의 높이를 지정합니다. 마찬가지로 CW_USEDEFAULT로 임의의 크기를 지정했습다.
8. hWndParent: 부모 윈도우를 지정합니다. 윈도우들은 모두 부모-자식 관계로 이루어져 있습니다. 예를 들어 윈도우에 버튼이 있다면 버튼은 윈도우의 자식이고, 윈도우는 버튼의 부모가 됩니다. 최상위 윈도우는 바탕화면 윈도우입니다. 여기서 0을 지정하면 바로 바탕화면 윈도우를 부모로 지정하게 됩니다.
9. hMenu: 메뉴를 지정하는 인자입니다. 메뉴는 안 사용하기 때문에 0을 넣었습니다.
10. hInstance: 인스턴스 핸들을 넣어주면 됩니다.
11. lpParam: 사용자가 필요로 하는 윈도우 생성 인자인데, 지금은 쓸 일이 없습니다.

그렇게 열심히 만든 윈도우의 핸들이 반환됩니다. 타입은 HWND이다, 이후에 ShowWindow 함수를 통해서 스크린에 나타냅니다. 두 번째 인자 SW_SHOW는 화면에 나타내라는 의미입니다. 만약 SW_HIDE를 준다면, 화면에서 사라지고 작업 표시줄에서도 나타나지 않습니다.

이후에는 메시지 루프를 도는데 이는 대부분의 GUI 프로그램이 가지는 패턴입니다. GUI 응용프로그램은 콘솔과는 다르게 다양한 이벤트(Event)에 대처해야 합니다. 키보드가 눌리거나, 마우스가 눌리거나 움직이거나 드래그를 할 수도 있습니다. 또한 사용자가 드래그해서 윈도우의 위치를 옮겨버리거나, 크기를 늘리거나 닫아서 꺼버릴 수도 있습니다. 이런 것들을 이벤트라고 합니다.

MSG 구조체에는 이러한 이벤트 정보를 담은 메시지가 담기게 됩니다. GetMessage 함수는 메세지를 받을 때까지 scanf 함수처럼 대기하게 됩니다. 이후 메시지가 전달되면 그 메시지가 종료메세지인 경우 FALSE(0)을, 아닐 경우 TRUE(1)를 반환하기 때문에 종료한 경우 반복문이 끝나게 됩니다. 이후 TranslateMessage 함수는 받은 메시지를 전달하고, DispatchMessage 함수가 처리하게 됩니다. 이 두 함수는 반드시 호출 되어야 합니다. 그렇게 전달받은 메시지를 처리하는 기능을 우리가 위에서 선언한 WindowProc 함수가 담당하게 됩니다. 왜냐하면 윈도우 클래스에 lpfnWndProc 인자에 WindowProc 함수를 넣었기 때문입니다. 처리해야 할 메시지가 발생한다면

우리가 선언한 WindowProc 함수에 인자를 주어서 호출하게 됩니다. 이렇게 우리가 직접 호출하는 게 아닌 시스템에서 호출하는 함수를 콜백 함수(Callback Function)이라고 합니다.

WindowProc 함수의 반환형은 LRESULT 인데 이는 32비트 정수형입니다. 함수의 처리 결과를 의미하는데 그 결과는 처리하는 메시지에 따라 다릅니다. 함수의 첫 번째 인자는 메시지를 받은 윈도우, 두 번째 인자는 메세지의 종류이고, 세 번째와 네 번째는 메시지의 부가적인 정보를 나타냅니다. 함수의 몸체에서는 switch구문을 통해서 메시지의 종류에 따른 처리를 하게 됩니다. 이 메시지의 상수는 WM_으로 시작하며, 굉장히 많은 종류의 메시지가 있습니다. 그 중 WM_DESTROY는 윈도우 객체가 사라질 때 그 직전에 호출됩니다. 예제에선 PostQuitMessage() 함수를 호출하는데, 이는 종료 메시지를 전달하는 함수입니다. 이 함수가 호출되면 WinMain 함수에서 진행한 메시지 루프가 종료되게 됩니다. WM_CREATE 라는 메시지는 윈도우가 CreateWindow() 함수로 정상적으로 실행되게 된다면 전달되는 메시지입니다. 한번 switch 문에 아래 코드를 입력해서 확인해 봅시다.

```
case WM_CREATE:
            MessageBox(0, L" 윈 도 우 가  생 성 됐 습 니 다 !" , 0, 0);
            return 0;
```

default 구문을 통해서 처리되지 않은 함수들은 DefWindowProc 이라는 함수가 대신 처리해줍니다. 이 함수는 윈도우가 처리해야 할 기능들을 처리해 줍니다. 예를 들어 단순히 윈도우의 닫기 버튼을 누르는 것만으로는 윈도우 객체가 파괴되지 않습니다. 윈도우 객체를 파괴하지 않으면 계속해서 메모리 상에 존재하게 됩니다. DestroyWindow() 함수를 통해서 파괴해야 하는데, DefWindowProc 함수는 그 역할을 수행해 줍니다. 그렇기 때문에 윈도우가 파괴될 때 전달되는 WM_DESTROY 메시지가 나타나게 된 것입니다. 만약 닫기 버튼이 눌려도 윈도우 객체를 파괴하고 싶지 않다면 닫기 버튼이 눌릴 때 발생하는 WM_CLOSE 메시지를 따로 처리해 주어야 할 것입니다. 혹은 style에 CS_NOCLOSE를 지정해 주면 됩니다.

길고 장황했습니다. 위 예제를 실행하면 아래의 결과가 나옵니다.

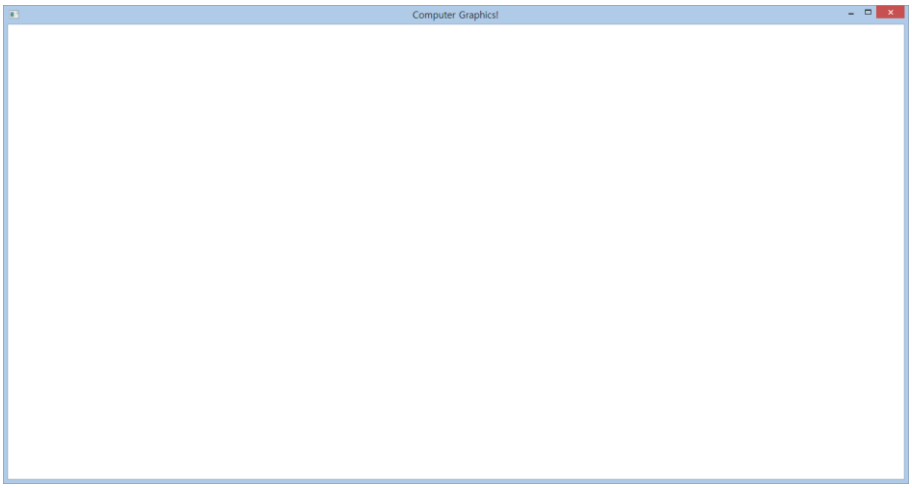

그림 2. 윈도우 프레임

윈도우의 크기와 위치를 바꾸기

크기, 좌표를 모두 CW_USEDEFAULT로 했습니다. 이 경우 크기는 보통 화면 크기보다 좀 작게 나옵니다. 좌표 값은 아무렇게나 나오지 않고 나름 규칙성 있게 화면 크기를 고려하여 왼쪽 위에 주로 나오게 됩니다. 윈도우의 크기와 위치는 SetWindowPos() 함수로 지정할 수 있습니다. SetWindowPos() 함수의 마지막 인자를 통해 크기나 위치 하나만 바꾸거나 둘 다 바꿀 수도 있습니다. 둘 다 바꾼다면 마지막 인자에 0을 주면 됩니다. 아래 코드는 예제의 일부분입니다.

메모 포함[HK1]: 크기, 클라이언트 영역, 위치 이동과 같은 내용 추가해야 할 듯

```
HWND hwnd = CreateWindow (… );

// 위치는 바꾸지 않고 크기만 바꾼다(SWP_NOMOVE).
SetWindowPos(hwnd, NULL,
       0, 0,
       450, 300, // 가로, 세로
       SWP_NOMOVE );
// 크기는 바꾸지 않고 위치만 바꾼다(SWP_NOSIZE).
SetWindowPos(hwnd, NULL,
       450, 300, // X, Y
       0, 0,
       SWP_NOSIZE );
// 윈도우 창을 띄운다.
ShowWindow(hwnd, SW_SHOW);
```

예제 3. ControlWindow.cpp

클라이언트 영역

프레임에는 클라이언트 영역(Client Area)가 존재합니다. 클라이언트 영역이란 아래 그림처럼 제목 바, 겉 경계 등을 제외한 내부 영역을 말합니다.

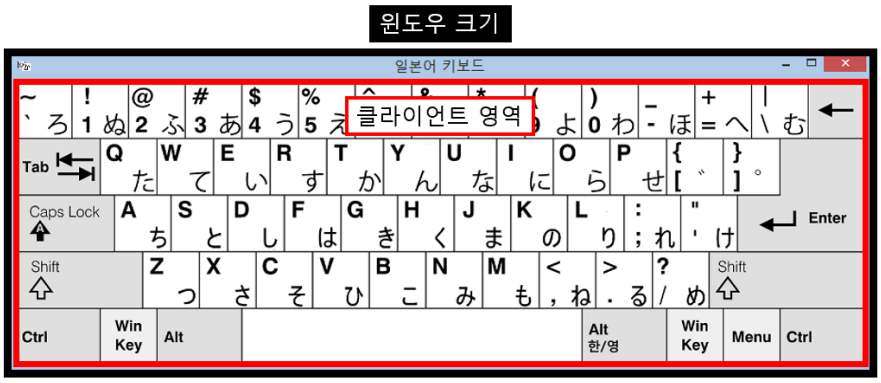

그림 3. 프레임의 클라이언트 영역

우리가 CreateWindow() 함수나 SetWindowPos() 와 같은 함수는 바깥 검정 부분의 크기를 지정하지 내부 클라이언트 영역의 크기를 지정하지는 않습니다. 다행히 이런 점을 고려하여, 특정 클라이언트 영역의 크기가 나오는 윈도우 크기를 계산해주는 함수가 있습니다. 바로 AdjustWindowRect() 함수입니다. 아래 예제에서는 AdjustWindowRect() 함수를 통해 가로 800 세로 600의 클라이언트 영역을 지닌 윈도우를 생성합니다.

```
	DWORD windowStyle = WS_OVERLAPPEDWINDOW;
	RECT rect = {// 원하는 크기는 가로 800, 세로 600이다.
		0, 0,
		800, 600
	};

	// AdjustWindowRect() 함수를 통해서
	// 특정 스타일을 가진 윈도우의 클라이언트 영역의 크기를 구한다
	// 마지막 인자는 메뉴의 여부로, 없기 때문에 false를 준다.
	AdjustWindowRect(&rect, windowStyle, false);

	// 가로, 세로 계산하기
	DWORD width = rect.right - rect.left;
	DWORD height= rect.bottom- rect.top;

	HWND hwnd = CreateWindow (
```

```
            wc.lpszClassName,          // 창 클래스
            L"AdjustWindow",            // 창 제목
            windowStyle,               // 창 스타일
            CW_USEDEFAULT,             // 화면에서의 X 좌표
            CW_USEDEFAULT,             // 화면에서의 Y 좌표
            width,                       // 가로 크기
            height,                     // 세로 크기
            0,                         // 부모 윈도우
            0,                         // 메뉴
            hinst,                     // 인스턴스 핸들
            0                          // 초기 전달 값
            );
```

예제 4. AdjustWindow.cpp

이번 장에서는 윈도우를 생성하는 방법에 대해 알아보고, 생성한 윈도우의 위치나 크기를 바꿔 보았습니다. 다음 장에서는 생성한 윈도우에 그림을 그려서 게임을 제작하기 위한 기초를 알아보도록 하겠습니다.