04. 델타타임을 이용한 속도 제어

게임은 시간을 통해 제어함. 어떤 프레임에는 연산량이 많아 오래 걸릴수도 있고 어떤 프레임에는 연산할 것이 별로 없어 짧게 끝날수도 있음.

매 프레임마다 간격이 절대 일정할 수 없기 때문에 시간을 이용!

그렇다면 시간 간격을 사용하려면 어떻게 해야하는가?

→ CPU는 초당 몇 백만번의 연산을 할 수 있음. CPU 주파수에 따라 1초당 몇 번의 진동이 일어나는지. 즉, 초당 Tick 수를 얻어오는 함수가 Query Performance Frequency.
고해상도 타이머의 진동수(1초당 진동수)를 반환함.

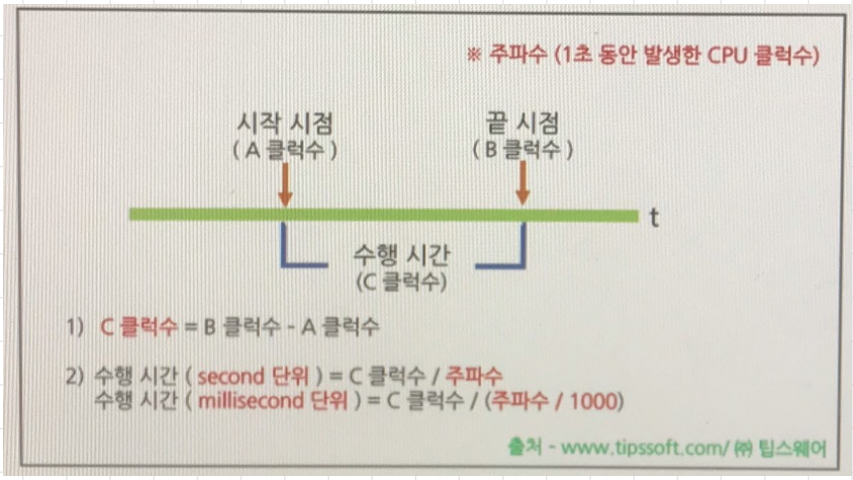
이 함수는 LARGE_INTEGER를 인자로 받는다.

```
typedef union _LARGE_INTEGER {
    struct {
        DWORD LowPart;  ①
        LONG HighPart;
    } DUMMYSTRUCTNAME;
    struct {
        DWORD LowPart;  ②
        LONG HighPart;
    } u;
    LONGLONG QuadPart;  ③
} LARGE_INTEGER;
#endif //MIDL_PASS
```

LARGE_INTEGER는 이런 형식의 공용체.
(①,②,③이 메모리를 공유함.)
이때 QueryPerformanceFrequency는 가장 큰 크기인 QuadPart 부분에 Tick 값을 넣어줌. (타이머의 주파수)

그다음 현재 프레임의 카운터를 알아내기 위해 QueryPerformanceCounter 함수를 이용. 해당 함수는 고해상도 타이머의 현재 CPU 클럭수를 얻어옴. 인자에 함수가 호출된 시점의 고해상도 타이머 값을 저장해줌.

위 진동수를 아래와 같은 계산을 통해 시간으로 변환



거리/속력/시간으로 생각해보면

$$\text{시간} = \frac{\text{움직인 거리 (이전 부터 현재 까지의 클럭 수)}}{\text{속력 (고정된 주파수)}}$$

B : 10만
A : 5만
주파수 : 100만

CPU는 초당 100만 클럭인데 이전 시점에서 현재 시점 까지 10만 - 5만 = 5만 클럭 만 진동함.

$$\text{시간} = \frac{5\text{만}}{100\text{만}} = \frac{1}{20}\text{초}$$

즉, 이전에서 현재까지 $\frac{1}{20}$초 흐름.

<실제코드>

```cpp
#include "pch.h"
#include "framework.h"
#include "AssortrockWINAPI.h"

#define MAX_LOADSTRING 100

typedef struct _tagRectangle
{
    float l, t, r, b;
}RECTANGLE, *PRECTANGLE;

// Global Variables:
HINSTANCE hInst;                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

HWND g_hWnd;
HDC  g_hDC;
bool g_bLoop = true;
RECTANGLE g_tPlayerRC = { 100, 100, 200, 200 };

// 시간을 구하기 위한 변수들
LARGE_INTEGER g_tSecond;
LARGE_INTEGER g_tTime;
float         g_fDeltaTime;
```

RECTANGLE* rect
=
PRECTANGLE rect

RECTANGLE 구조체를 가리키는 포인터 타입 이름을 PRECTANGLE로 함.

```cpp
    // 화면 그리기용 DC 생성.
    g_hDC = GetDC( g_hWnd );

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_ASSORTROCKWINAPI));

    MSG msg;

    QueryPerformanceFrequency( &g_tSecond );
    // 이전 Tick(초기 Tick)
    QueryPerformanceCounter( &g_tTime );

    // Main message loop:
    while( g_bLoop )
    {
        if ( PeekMessage( &msg, nullptr, 0, 0, PM_REMOVE ) )
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        // 윈도우 데드 타임일 경우
        else
        {
            Run();
        }
    }

    ReleaseDC( g_hWnd, g_hDC );

    return (int) msg.wParam;
}
```

CPU 주파수(초당 클럭수 얻어오기)

Run() 들어가기전 초기(이전) 지점 클럭수 저장

```cpp
void Run()
{
    // DeltaTime
    LARGE_INTEGER tTime;
    // 현재 프레임의 틱을 뽑아옴.
    QueryPerformanceCounter( &tTime );

    // (현재 Tick - 이전 Tick) / 초당 Tick
    g_fDeltaTime = static_cast<float> ((tTime.QuadPart - g_tTime.QuadPart)) / g_tSecond.QuadPart;

    g_tTime = tTime;

    // 슬로우 모션 ( 타임 스케일 이용 )
    static float fTimeScale = 1.f;

    if( GetAsyncKeyState( VK_F1 ) & 0x8000 )
    {
        fTimeScale -= g_fDeltaTime;
        if( fTimeScale < 0.f )
        {
            fTimeScale = 0.f;
        }
    }

    if( GetAsyncKeyState( VK_F2 ) & 0x8000 )
    {
        fTimeScale += g_fDeltaTime;
        if( fTimeScale > 1.f )
        {
            fTimeScale = 1.f;
        }
    }

    // 플레이어 초당 이동속도 : 300에 30%만큼 더 빠르게 움직임.(아이템 먹었을 때 이동 속도 퍼센트 단위로 올리기)
    float fSpeed = (300 + 300 * 0.3f) * g_fDeltaTime * fTimeScale;

    RECT clientRect;
    GetClientRect( g_hWnd, &clientRect );

    if( GetAsyncKeyState( 'D' ) & 0x8000 )
    {
        g_tPlayerRC.l += fSpeed;
        g_tPlayerRC.r += fSpeed;
        if( g_tPlayerRC.r > clientRect.right )
        {
            g_tPlayerRC.r = clientRect.right;
            g_tPlayerRC.l = clientRect.right - 100;
        }
    }
```

현재 클럭수 이전 클럭수에 저장

타임 스케일을 이용, 슬로우 모드와 일시정지 기능을 구현 가능.

호출 간격이 $\frac{1}{20}$인 경우, $\frac{1}{20} \times 300 = 15$

즉, 한 번 호출에 15만큼 움직이며,

1초에 20번 호출하므로, 초당 $20 \times 15 = 300$

만큼 움직인다고 볼 수 있다.