# DirectX 資料①

初期化編

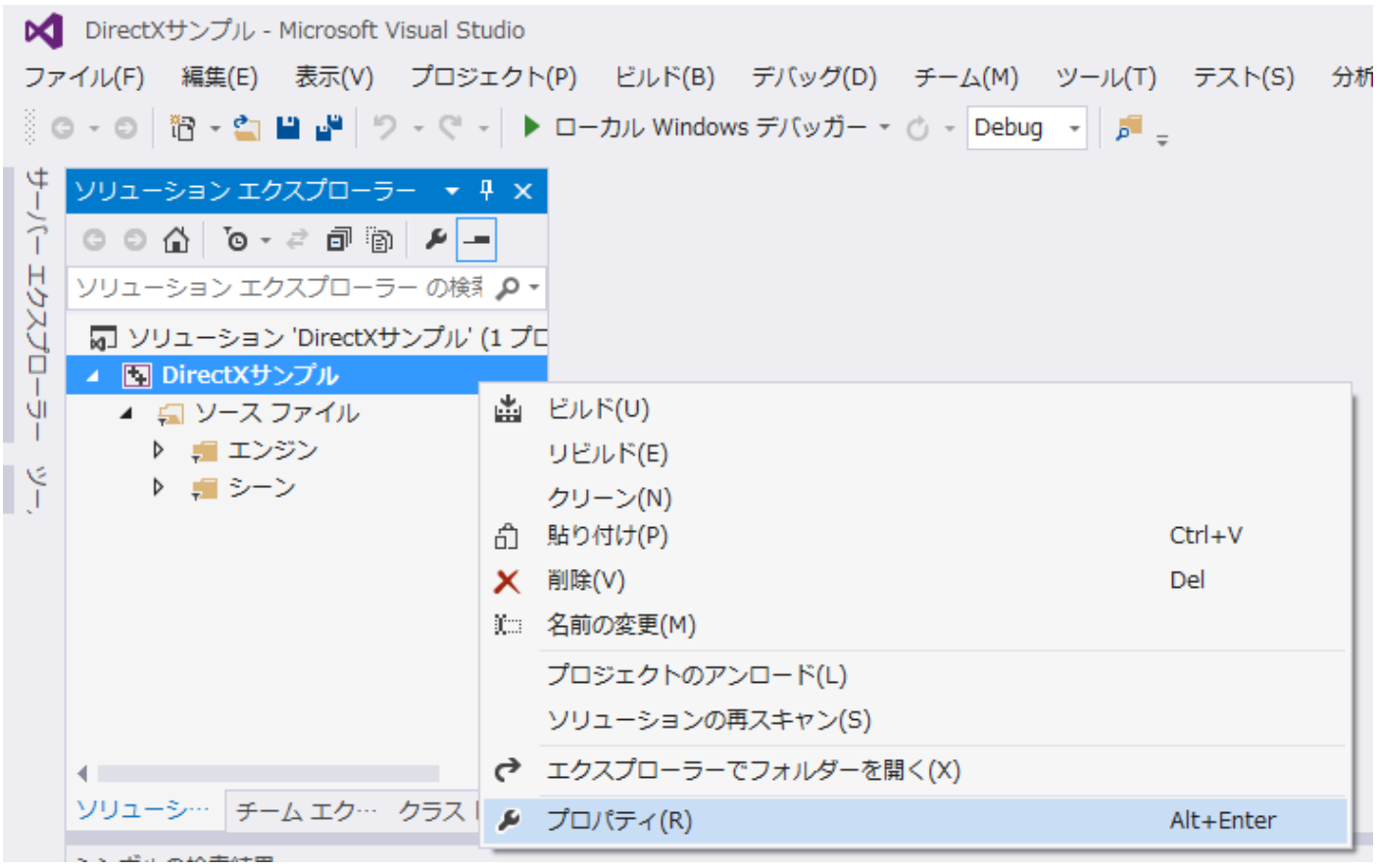
ゲームエンジニア科２年

# 1. VisualStudio の設定

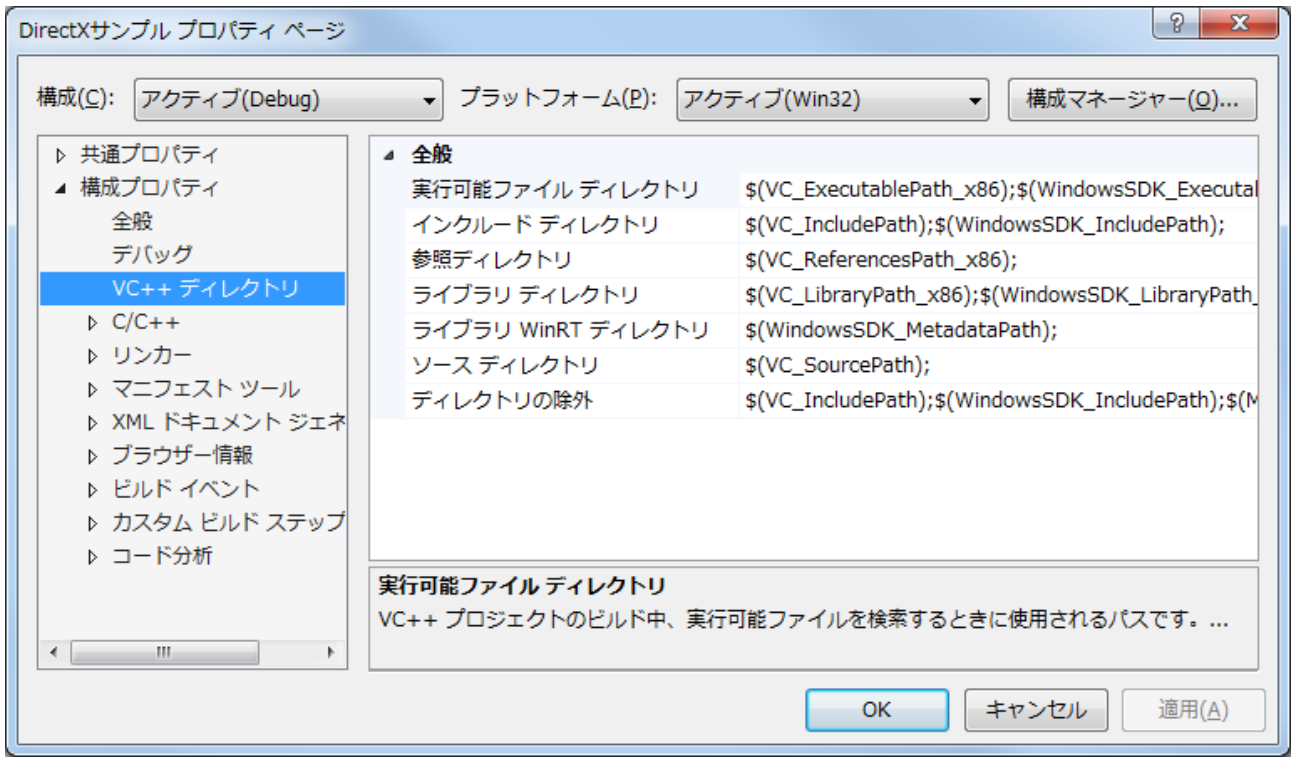
VisualStudio で DirectX を使ったプログラムを作る場合はこの設定をしなければならない。

DirectX に必要なファイルが PC のどこにあるのかを伝えるもの。

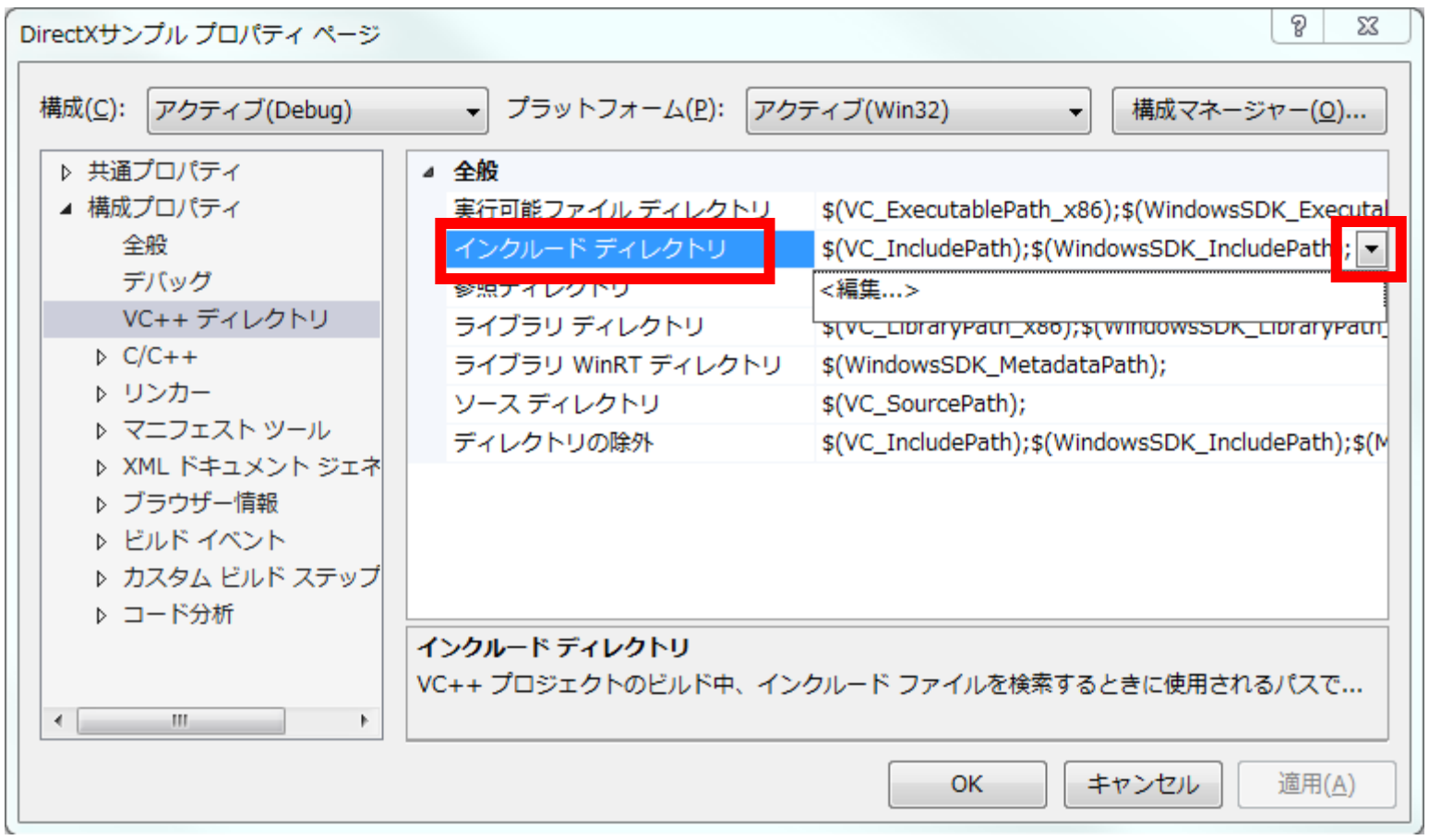
## ① ソリューションエクスプローラーのプロジェクト名を右クリックして「プロパティ」を選択



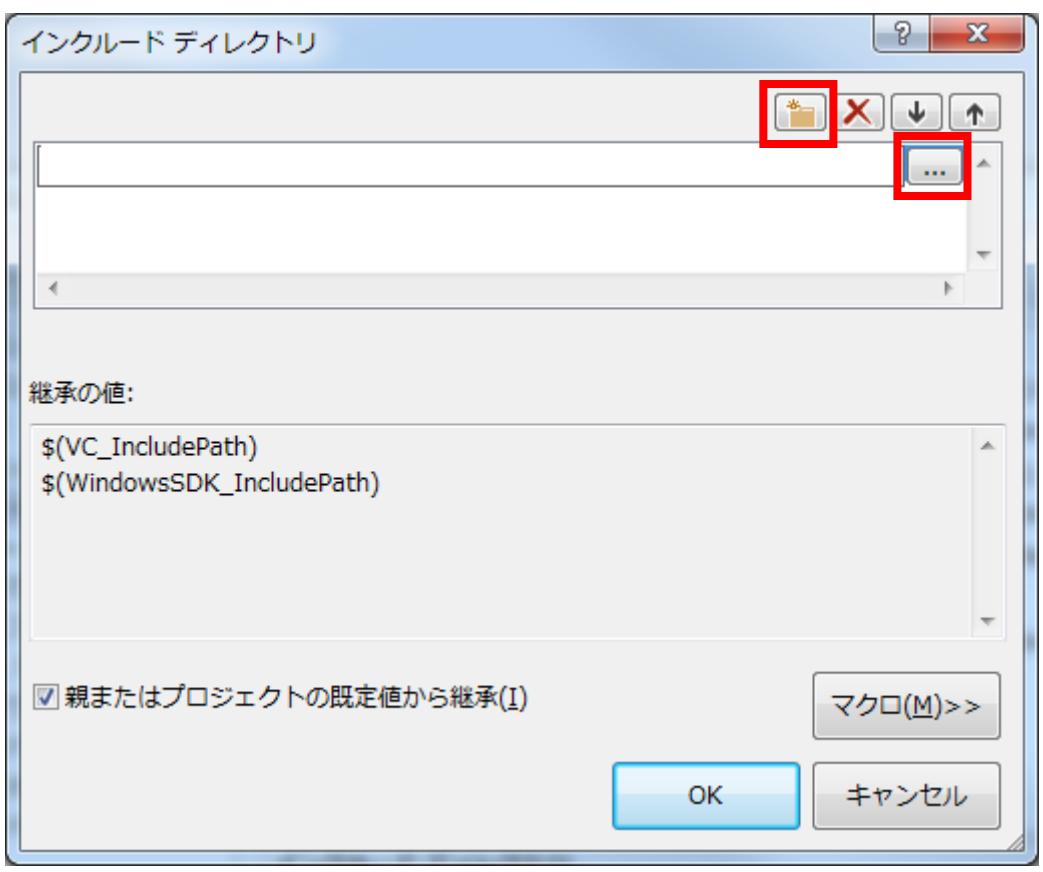

## ② 「VC++ ディレクトリ」を選択



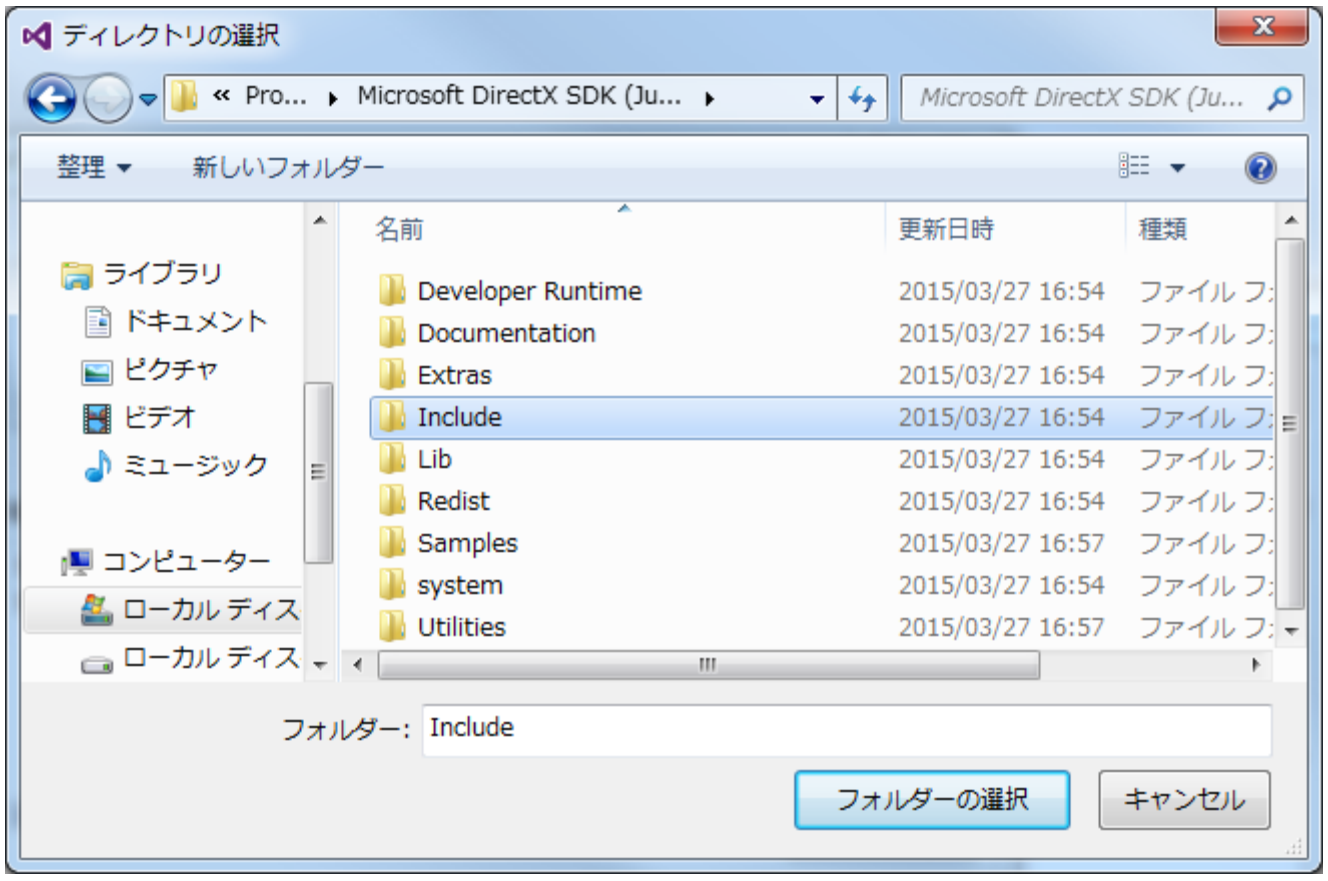
③ 「インクルードディレクトリ」を選択し、右端の▼を押して＜編集＞をクリック



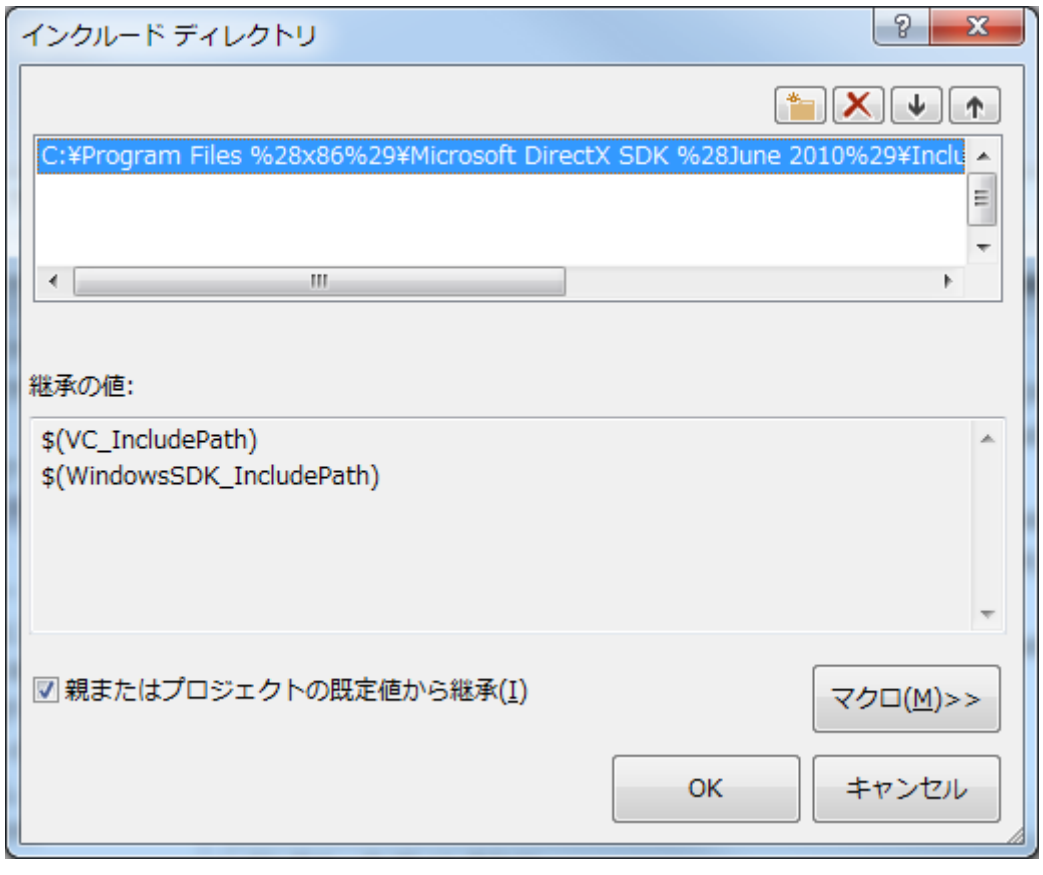

④ 「新しい行」ボタンを押して、右端の「・・・」をクリック



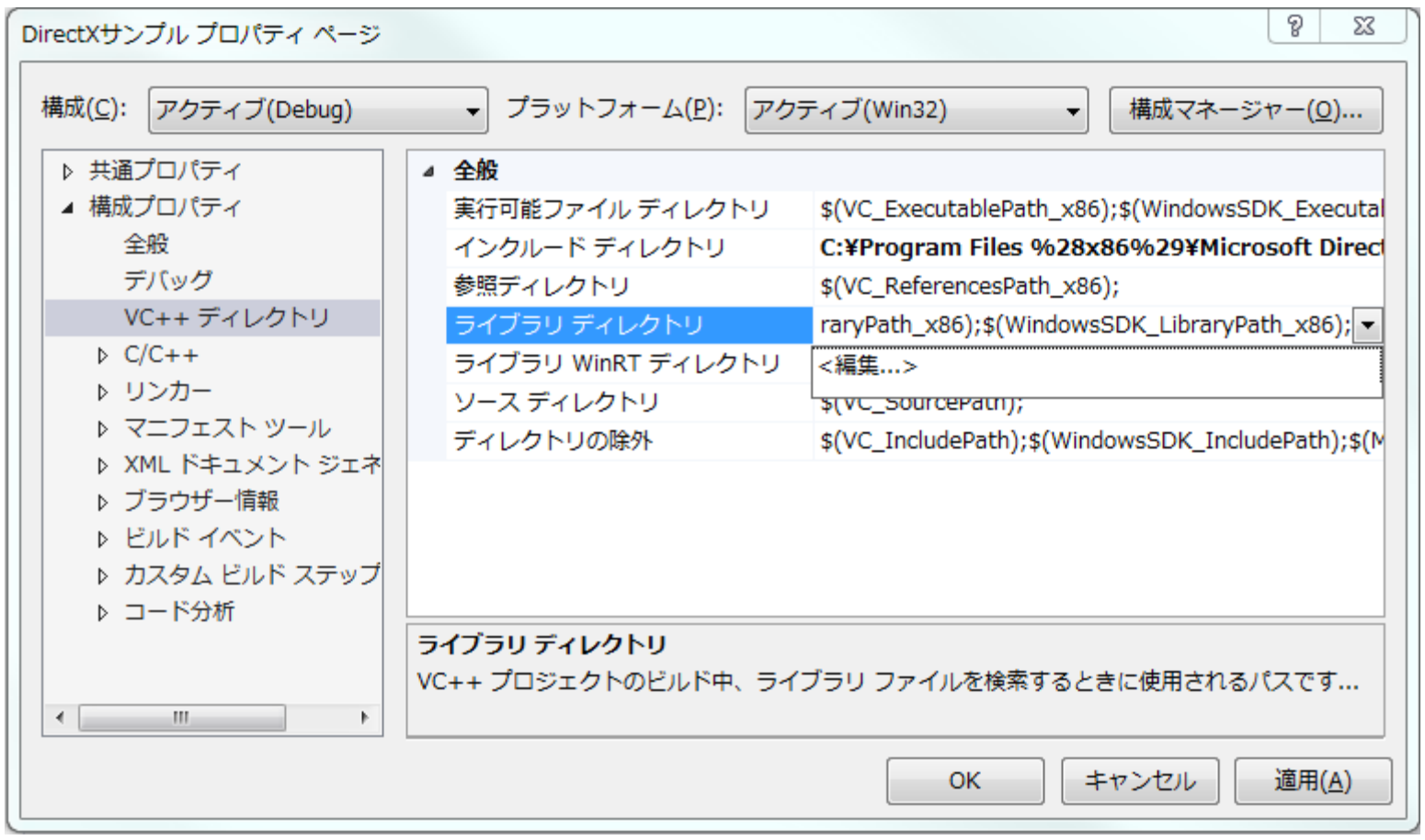
⑤ 「C:¥Program Files (x86)¥Microsoft DirectX SDK (June 2010)¥Include」を選択。



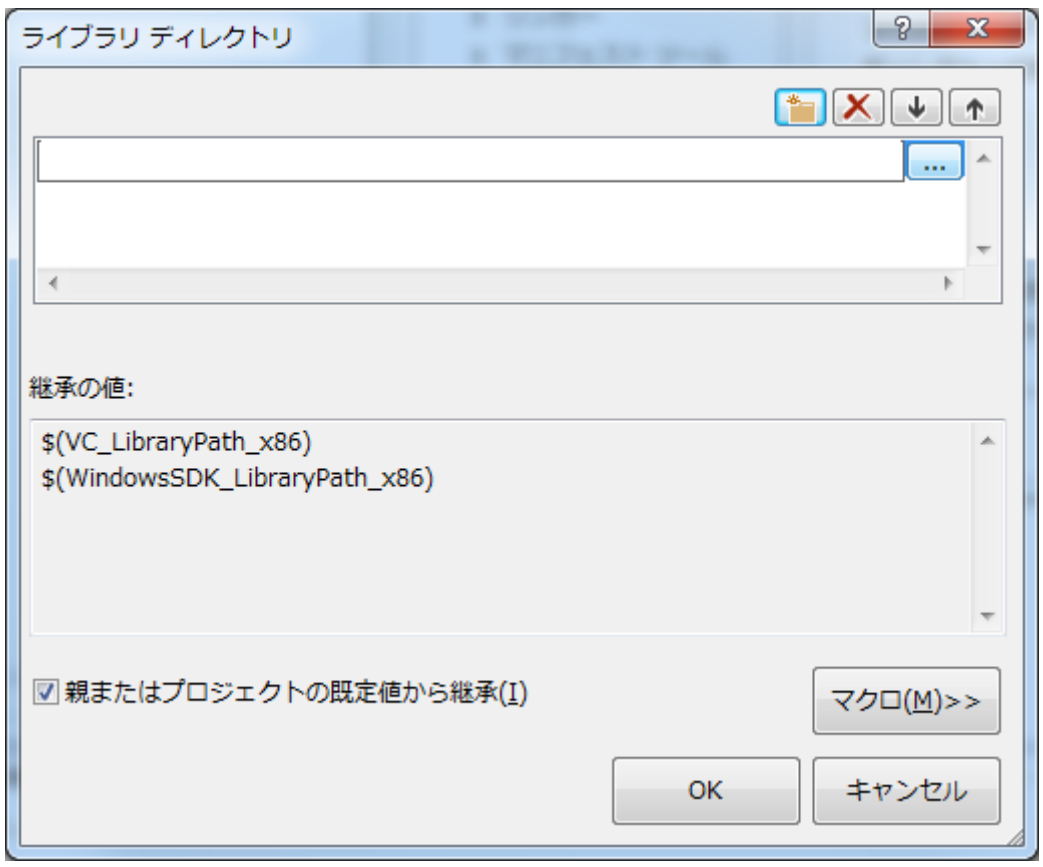

⑥ 「OK」ボタン



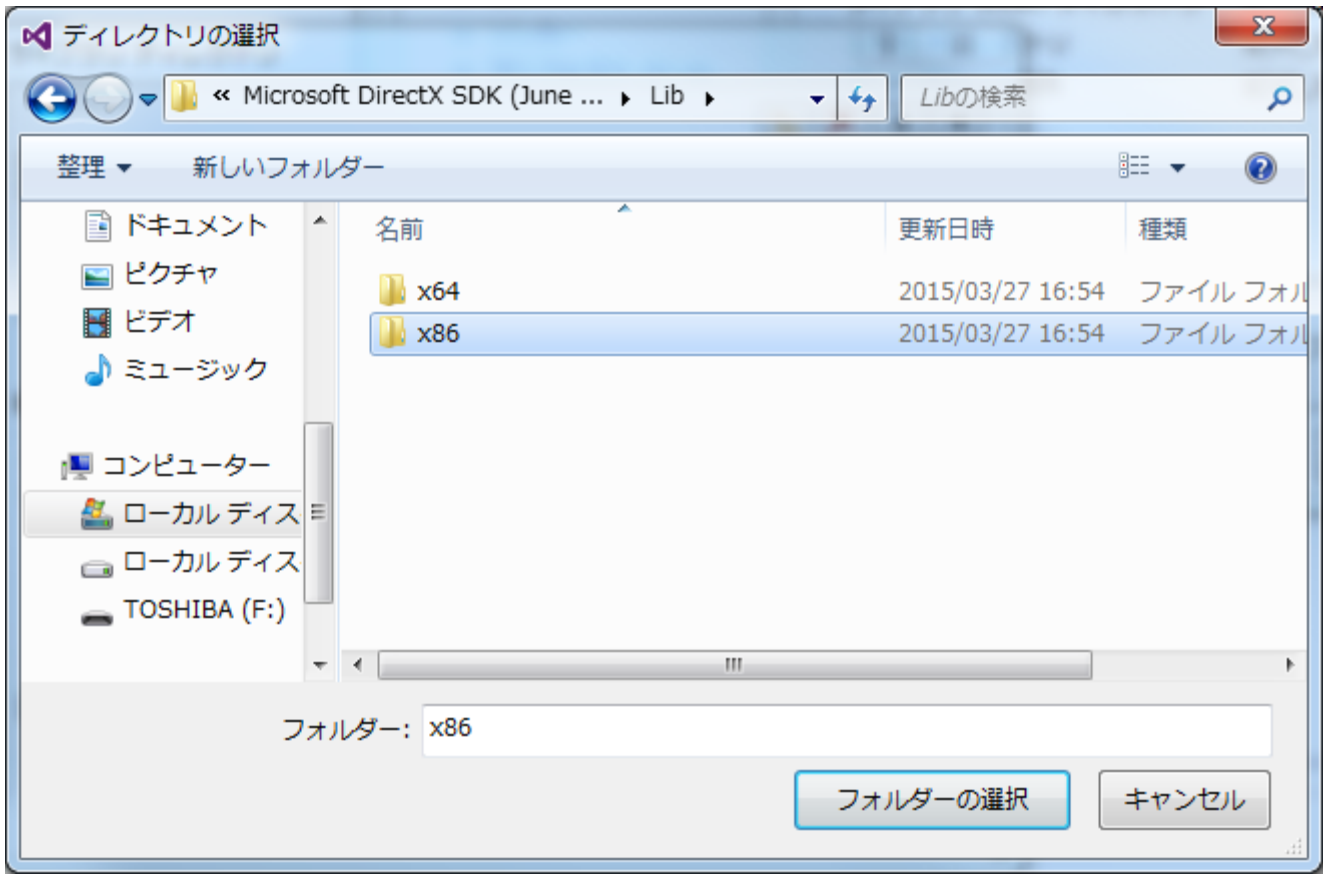
⑦ 「ライブラリディレクトリ」を選択し、右端の▼を押して＜編集＞をクリック




⑧ 「新しい行」ボタンを押して、右端の「・・・」をクリック



⑨ 「C:¥Program Files (x86)¥Microsoft DirectX SDK (June 2010)¥Lib¥x86」を選択



# 2. Direct3D 初期化

## ① インクルードとライブラリのロード

Direct3D を扱うためには必須

**<Global.h>**

```
//---------------インクルード----------------------
#include <windows.h>
#include <d3dx9.h>

//---------ライブラリファイルのロード-------------
#pragma comment(lib,"d3d9.lib")
#pragma comment(lib,"d3dx9.lib")
```

## ② Direct3D オブジェクト作成

今回は Direct3D に関する処理は Game クラスに含めてしまう。（別途 Direct3D クラスを作ってもよい）

Direct3D の本体となる変数をつくり初期化するための関数を用意する。

<Game.h>

```
//―――――――――――――――――――――――――――――
//  ゲームクラス
//―――――――――――――――――――――――――――――
class Game
{
    //Direct3D オブジェクト
    LPDIRECT3D9 pD3d;

    //シーンオブジェクト
    SceneBase* scene[SC_MAX];
public:
    //コンストラクタ
    Game();

    //デストラクタ
    ~Game();

    //DirectX 初期化
    HRESULT InitD3d(HWND);

    //衝突判定処理
   void Hit();
```

**<Game.cpp>**

```
//-----------------------------------------
//  Direct3D 初期化
//-----------------------------------------
HRESULT Game::InitD3d(HWND hWnd)
{
    // 「Direct3D」オブジェクトの作成
    if (NULL == (pD3d = Direct3DCreate9(D3D_SDK_VERSION)))
    {
        MessageBox(0, "Direct3D の作成に失敗しました", "", MB_OK);
        return E_FAIL;
    }

    return S_OK;
}
```

## ③ Direct3D デバイスオブジェクト作成

Direct3D デバイスは「ゲーム画面」のこと。

Windows プログラムのデバイスコンテキストと似たイメージ。

これも Game クラスで制御するが、いろいろな場所で扱うことになるので変数はグローバルで用意しておく。

**<Global.h>**

```
//Direct3D デバイスオブジェクト
extern LPDIRECT3DDEVICE9   g_pDevice;
```

**<Game.cpp>**

```
//----------------グローバル変数----------------
GAME_SCENE          g_gameScene;    //現在のゲームシーン
LPDIRECT3DDEVICE9   g_pDevice;      //Direct3D デバイスオブジェクト
```

デバイスオブジェクトの作成処理は、先ほど作った InitD3d 関数の中に含めてしまおう。（もちろん別関数を作ってもよい）

まずは、いろいろと設定するための構造体変数を用意する。

**<Game.cpp>**

```
//-----------------------------------------
//  Direct3D 初期化
//-----------------------------------------
HRESULT Game::InitD3d(HWND hWnd)
{
            :
            :
    // 「DIRECT3D デバイス」オブジェクトの作成
    D3DPRESENT_PARAMETERS d3dpp;
```

そのメンバに各設定をしていく。(各項目の説明は授業で)

**<つづき>**

```
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    d3dpp.BackBufferFormat =       D3DFMT_UNKNOWN;
    d3dpp.BackBufferCount =        1;
    d3dpp.SwapEffect =             D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed =               TRUE;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
```

その設定をもとにデバイスオブジェクトを作成する。

**<つづき>**

```
    pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                       D3DCREATE_HARDWARE_VERTEXPROCESSING,&d3dpp, &g_pDevice);
```

これでデバイスオブジェクトが完成する。

しかし、環境によっては Direct3D が作成できない場合がある。

その場合は、パフォーマンスを落とした設定で再度作成を試みる。

```
    if (FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, Wnd,
                        D3DCREATE_HARDWARE_VERTEXPROCESSING,    &d3dpp, &g_pDevice)))
    {
        MessageBox(0, "HAL モードで DIRECT3D デバイスを作成できません¥nREF モードで再試行します", NULL, MB_OK);

        pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, hWnd,
                           D3DCREATE_HARDWARE_VERTEXPROCESSING, &d3dpp, &g_pDevice);
    }
}
```

これでも失敗する場合がある。そのときは潔くあきらめよう。

```
    if (FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                              D3DCREATE_HARDWARE_VERTEXPROCESSING, &d3dpp, &g_pDevice)))
    {
        MessageBox(0, "HAL モードで DIRECT3D デバイスを作成できません¥nREF モードで再試行します", NULL, MB_OK);
        if (FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, hWnd,
                                  D3DCREATE_HARDWARE_VERTEXPROCESSING, &d3dpp, &g_pDevice)))
        {
            MessageBox(0, "DIRECT3D デバイスの作成に失敗しました", NULL, MB_OK);
            return E_FAIL;
        }
    }

    return S_OK;
}
```

### ④ 初期化処理を呼ぶ

今作った InitD3d 関数を呼び出そう。

ウィンドウが完成してからでないとデバイスオブジェクトは作れないので、WM_CREATE メッセージのところで呼ぶ。

**<Main.cpp>**

```
//■初期設定
case WM_CREATE:

    //ランダムの準備
    srand((unsigned)time(NULL));

    //ゲームオブジェクトを作成
    game = new Game;

    //Direct3D の初期化
    game->InitD3d(hWnd);

    //タイマーをセット
    SetTimer(hWnd, TIMER_ID, TIMER_INTERVAL, NULL);

    break;
```

一応ここでエラーが出ないか確認しておこう。（実行結果は特に変化無い）

## 3. 解放処理

Direct3D オブジェクトと Direct3D デバイスオブジェクトは、それぞれ「LPDIRECT3D9 型」と「LPDIRECT3DDEVICE9 型」である。DirectX で使う変数の型には「**LP**」で始まるものがたくさん出てくるが、これらは**"＊"を付けなくてもポインタ**になる。（LP はロングポインターの略）

ポインタを new した場合は必ず delete しなければならなかったが、同様に LP で始まる型の変数を使ったら解放処理が必要になる。解放処理の書き方は**変数名->Release()**で、基本的に解放処理の順番は使った順番の逆にする。

**<Game.cpp>**

```
Game::~Game()
{
    for (int i = 0; i < SC_MAX; i++)
    {
        delete scene[i];
    }

    //DirectX 解放
    g_pDevice->Release();
    pD3d->Release();
}
```

# 4. 描画処理

Windows プログラムでは、タイマーを使って一定の間隔で InvalidateRect 関数を呼び、
WM_PAINT メッセージが発せするたびにデバイスコンテキストに文字や画像を描画していた。
**Direct3D を使った描画処理はまったく異なったものになる。**

## ① 描画処理を削除

まずは、「タイマー」「デバイスコンテキスト」に関する処理を全て削除しよう。

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp)
{
    HDC         hdc;    //デバイスコンテキストハンドル
    PAINTSTRUCT ps;     //ペイントストラクト

    switch (msg)
    {
    //■初期設定
    case WM_CREATE:
            :
            :
        //タイマーをセット
        SetTimer(hWnd, TIMER_ID, TIMER_INTERVAL, NULL);
        break;

    //■描画処理
    case WM_PAINT:
            :
            :
        break;

    //■設定した時間になった（1秒間に60回発生）
    case WM_TIMER:
            :
            :
        break;


    //■ウィンドウが消された
    case WM_DESTROY:
```

今まで描画関係の関数は引数でデバイスコンテキストハンドルを受け渡ししていた。
これがいらなくなるので、Game クラス、SceneBase クラス、UnitBase クラス、各ユニットクラスの
Render 関数から引数を削除しよう。

**<各ヘッダー>**

```
    //描画処理
    //引数：なし
    void Render(HDC hdc);
```

**<各ソース>**

```
void ○○○○○○::Render(HDC hdc)
{
```

当然これらの関数を呼んでいるところも修正が必要になる。エラーがないか再度確認。

## ② メッセージループを変更

描画処理をすべて消してしまった。新しい描画処理はどこに書くべきだろうか。

Windows プログラムというのは、何かあったらメッセージが発生して、それに対応した処理を行う。
逆に言えば、何も起こらなければ何もしない。

しかし、ゲームの場合は基本的に**何もしなくても常に画面が更新される**。
つまり、何も起こらなくても（ウィンドウプロシージャが呼ばれなくても）画面を更新する必要がある。

これを実現するためにメッセージループ自体を変更してしまう。

**<main.cpp>**

```
int WINAPI WinMain(HINSTANCE hCurInst, HINSTANCE hPrevInst, LPSTR lpsCmdLine, int nCmdShow)
{
    MSG msg;
    BOOL bRet;

    //ウィンドウクラスの登録
    if (!InitApp(hCurInst))
        return FALSE;

    //ウィンドウ生成
    if (!InitInstance(hCurInst, nCmdShow))
        return FALSE;

    // メッセージを取得
    ZeroMemory(&msg, sizeof(msg));
    while (msg.message != WM_QUIT)
    {
        if (PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
            //ゲームの更新
            game->Update();

            //ゲーム画面の描画
            game->Render();
        }
    }
    return (int)msg.wParam;
}
```

メッセージを取得する関数が「GetMessage」から「**PeekMessage**」に変わった。
これはよりゲームに向いている関数である。
PeekMessage 関数は、特にメッセージがなければ FALSE を返す。

つまり、特にメッセージが無いときはゲームの更新を行うようにしている。

## ③ 画面をクリア

Game クラスの Render 関数を修正する必要がある。

今までは BeginPaint や EndPaint 関数を使っていたが、それと似たようなことを書かなければならない。

まずは何かを描画する前に画面をクリアする。

**<Game.cpp>**

```
void Game::Render()
{
    //画面をクリア
    g_pDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 255), 1.0f, 0);

    //ゲーム画面の描画
    scene[g_gameScene]->Render();
}
```

この例では画面を青く塗りつぶしている。(好きな色に変更してよい)

ただ塗りつぶすだけではなく、**Ｚバッファ**もクリアしている。

## ④ 描画開始と描画終了

描画処理を BeginScene 関数と、EndScene 関数で挟む。

**<Game.cpp>**

```
void Game::Render()
{
    //画面をクリア
    g_pDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 255), 1.0f, 0);

    //描画開始
    g_pDevice->BeginScene();

    //ゲーム画面の描画
    scene[g_gameScene]->Render();

    //描画終了
    g_pDevice->EndScene();
}
```

今までの BeginPaint・EndPaint と同様のものと考えてよい。

ただし、何らかの理由で BeginScene が失敗することもある。
その場合は何かを描画しようとするとエラーになるし、EndScene を呼んでもエラーになる。

対策として次のようにしておこう。

```
    //画面をクリア
    g_pDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 255), 1.0f, 0);

    //描画開始
    if (SUCCEEDED(g_pDevice->BeginScene()))
    {
        //ゲーム画面の描画
        scene[g_gameScene]->Render();

        //描画終了
        g_pDevice->EndScene();
    }
}
```

これで、BeginScene が成功したときだけゲーム画面の描画を行うようになる。

## ⑤ フリップ

Direct3D では２枚の画面（フロントバッファとバックバッファ）を用意し、バックバッファの方に画像を描画する。
全ての画像を描画したら、バックバッファとフロントバッファを交換する必要がある。
この作業を**フリップ**という。

**<Game.cpp>**

```
void Game::Render()
{
    //画面をクリア
    g_pDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 255), 1.0f, 0);

    //描画開始
    if (SUCCEEDED(g_pDevice->BeginScene()))
    {
        //ゲーム画面の描画
        scene[g_gameScene]->Render();

        //描画終了
        g_pDevice->EndScene();
    }

    //フリップ
    g_pDevice->Present(NULL, NULL, NULL, NULL);
}
```

以上で Direct3D を使う全ての準備が整った。
実行すると青い画面が表示されるはず。