ゲームプログラミング

▶　DirectInput－第４回　デバイス状態の取得　◀

入力デバイスの状態は、デバイスオブジェクトのGetDeviceStateメソッドで取得します。

デバイス状態の取得

入力デバイスのボタンの状態や軸の傾きといった情報の取得は、デバイスオブジェクトのGetDeviceStateメソッドで行います。このメソッドが返す情報は、SetDataFormatメソッドで設定したものです。定義済みのデータ形式を設定した場合は、キーボードは256バイトの配列、マウスはDIMOUSESTATE構造体またはDIMOUSESTATE2構造体に情報が返されます。
このメソッドで取得できる情報は、ボタンは「押されている(押しっぱなし)」「押されていない(離されている)」、軸は前回からの座標の差または傾きの角度です。ボタンや軸が「押された」「離された」という情報は、このメソッドでは取得できないので、デバイスバッファを使って取得します。

GetDeviceStateメソッド
－説明－
GetDeviceStateメソッドは、デバイスから直接データを取得します。

－パラメータ－
１つ目の引数は、２つ目の引数(情報を受け取る領域)のバイト数です。
２つ目の引数は、デバイスの状態を受け取る領域のアドレスです。

－戻り値－
成功した場合はDI_OK、それ以外はエラーの原因をエラーコードで返します。

SetDataFormatメソッドで定義済みの変数を指定した場合、このメソッドが返す形式は以下のようになります。

定義済み変数	対応デバイス	メソッドが返す情報
c_dfDIKeyboard	キーボード	256バイトの配列
c_dfDIMouse	マウス	DIMOUSESTATE構造体
c_dfDIMouse2	マウス	DIMOUSESTATE2構造体
c_dfDIJoystick	ジョイスティック	DIJOYSTATE構造体
c_dfDIJoystick2	ジョイスティック (フォースフィードバック対応)	DIJOYSTATE2構造体

キーボードの状態取得

GetDeviceStateメソッドがキーボードから取得する情報は、キーが「押されている」と「押されていない」というものです。１回のメソッド呼び出しで256種類のキー状態が取得できます。この情報は256バイトの配列(charまたはBYTE配列)に格納されます。
キーボードの状態を取得する場合、GetDeviceStateメソッドの１つ目の引数は256、２つ目の引数はキー状態を格納する配列の先頭アドレスを指定します。

```
// キーボード状態の取得
LPDIRECTINPUTDEVICE8   lpDIDKeyboard;   // キーボードのデバイスオブジェクト(初期化済みとする)
BYTE                   byKeyState[256]; // キー状態を格納する配列
lpDIDKeyboard->GetDeviceState(256, byKeyState);
```

上記のようにGetDeviceStateメソッドを呼び出すと、256バイトの配列にキーの状態が格納されます。配列の要素１つに対して１つのキーが割り当てられています。特定のキーの状態を取り出すには、あらかじめ定義されている「デバイス定数」を添字として使用します。たとえば、Enter(Return)キーは"DIK_RETURN"、'A'キーは"DIK_A"を添字とします(DirectInputでボタンや軸を指定するときに使用するデバイス定数は、ヘルプに詳しく記述されています。DirectX8のヘルプでは、目次→DirectInput→DirectInput C/C++ リファレンス→デバイス定数で調べることができます)。

配列に格納された数値から「押されている」「押されていない」という状態を調べることができます。配列の１要素は、charまたはBYTE形式なので８ビットあります。このうち上位１ビットで判定します。ここが１ならキーが押されていることを表し、０ならキーが押されていないことを表します。プログラムではビット論理積演算子"&"を使ったマスク処理で上位１ビットを取り出します。たとえば、Enterキーが押されているかどうかを調べるには、以下のようにします。

```
// Enterキーが押されているかを調べる
if(0 != (byKeyState[DIK_RETURN] & 0x80))
    // Enterキーは押されている
```

マウスの状態取得

GetDeviceStateメソッドがマウスから取得する情報は、マウスの変化量とボタンの状態です。変化量は、ｘ座標、ｙ座標、ｚ座標(ホイール)が前回の取得位置からどのくらい動いたのかという情報です。これらの情報はDIMOUSESTATE構造体またはDIMOUSESTATE2構造体に格納されます。２つの構造体の違いは取得できるボタン数で、前者は４ボタン、後者は８ボタンまで取得できます。

マウスの状態を取得する場合、GetDeviceStateメソッドの１つ目の引数はsizeof演算子で構造体のサイズを指定し、２つ目の引数は状態を格納する構造体変数のアドレスを指定します。

```
// マウス状態の取得
LPDIRECTINPUTDEVICE8   lpDIDMouse;      // マウスのデバイスオブジェクト(初期化済みとする)
DIMOUSESTATE           diMouseState;    // マウスの状態を格納する構造体変数
lpDIDMouse->GetDeviceState(sizeof(diMouseState), &diMouseState);
```

上記のようにGetDeviceStateメソッドを呼び出すと、DIMOUSESTATE構造体にマウスの状態が格納されます。この構造体のメンバと格納される情報は以下のようになっています。

```
struct DIMOUSESTATE {
    LONG lX;                // ｘ座標の変化量
    LONG lY;                // ｙ座標の変化量
    LONG lZ;                // ｚ座標(ホイール)の変化量
    BYTE rgbButtons[4];     // ボタンの状態(DIMOUSESTATE2構造体では要素数8)
};
```

ボタンはキーボードと同じようにBYTE配列になっており、特定のボタンを指定するにはボタンの番号を添字にします。「押されている」「押されていない」という情報も上位１ビットで判定します。

```
// マウスのボタン０(通常は左ボタン)が押されているかを調べる
if(0 != (diMouseState.rgbButtons[0] & 0x80))
    // ボタン０は押されている
```

マウスカーソルの座標はDirectInputでは取得できません。移動量から計算するか、APIのGetCursorPos関数で取得します。ただし、排他占有モードの場合はこの関数では取得できません。

```
// マウスカーソル座標取得
POINT   ptMouse;
GetCursorPos(&ptMouse);
ScreenToClient(hWnd, &ptMouse); // ウィンドウ座標に変換。フルスクリーンアプリケーションでは不要
```

○ 課題 ○

(１)以下のプログラムは、キーボードのキー状態を取得するDIGetKeyboardState関数です。関数の仕様をよく読み、プログラムを完成させましょう。

DIGetKeyBoardState関数
－説明－
DIGetKeyboardState関数は、キーボードのキー状態を取得し、指定された領域に格納します。

－パラメータ－
BYTE byKeyState[]…キー状態を格納する要素数256のBYTE配列の先頭アドレス

－戻り値－
取得に成功した場合はtrue、それ以外はfalseを返す。

－追加１－

```
/*****************************************************************************************************/
/*                                          キーボード状態取得                                         */
/*****************************************************************************************************/
bool DIGetKeyboardState(BYTE byKeyState[])
{
    ZeroMemory(byKeyState, 256);    // 配列をゼロクリア

    if(NULL == g_lpDIDevice8[DID_KEYBOARD]) {
        OutputDebugString("*** Error - キーボード未初期化(DIGetKeyboardState)\n");
        return false;
    }

    const HRESULT  hr = ※ここは各自考えましょう(ヒント：キーボードの状態取得);
    if(DI_OK != hr) {
        if(DIERR_INPUTLOST == hr)
            g_lpDIDevice8[DID_KEYBOARD]->Acquire();     // アクセス権の再取得
        OutputDebugString("*** Error - キーボード状態取得失敗(DIGetKeyboardState)\n");
        return false;
    }

    return true;
}
```

(２)以下のプログラムは、マウスの状態を取得するDIGetMouseState関数です。関数の仕様をよく読み、プログラムを完成させましょう。

DIGetMouseState関数

－説明－

DIGetMouseState関数は、マウスの状態を取得し、指定された領域に格納します。

－パラメータ－

DIMOUSESTATE& diMouseState…マウスの状態を格納するDIMOUSESTATE構造体変数

－戻り値－

取得に成功した場合はtrue、それ以外はfalseを返す。

－追加２－

```
/*****************************************************************************************************/
/*                                          マウス状態取得                                             */
/*****************************************************************************************************/
bool DIGetMouseState(DIMOUSESTATE& diMouseState)
{
    ZeroMemory(&diMouseState, sizeof(diMouseState));    // 構造体をゼロクリア

    if(NULL == g_lpDIDevice8[DID_MOUSE]) {
        OutputDebugString("*** Error - マウス未初期化(DIGetMouseState)\n");
        return false;
    }

    const HRESULT  hr = ※ここは各自考えましょう(ヒント：マウスの状態取得);
    if(DI_OK != hr) {
        if(DIERR_INPUTLOST == hr)
            g_lpDIDevice8[DID_MOUSE]->Acquire();    // アクセス権の再取得
        OutputDebugString("*** Error - マウス状態取得(DIGetMouseState)\n");
        return false;
    }

    return true;
}
```

(３)以下のプログラムを完成させ、カーソルキーが押されたらキャラクターが移動するようにしましょう。なお、このプログラムはGameFunc.cppのTest関数とTestProc関数に作成するものとします。

```
/*****************************************************************************************************/
/*                                          機能テスト                                               */
/*****************************************************************************************************/
void Test()
{
    DDLoadFromFile(DDS_OFFSCRN1, "BG.bmp");     // 背景読み込み
    DDLoadFromFile(DDS_OFFSCRN2, "Chara.bmp");  // キャラクター読み込み
    DDSetColorKey(DDS_OFFSCRN2, DDCKEY_SRCBLT, RGB(0, 0, 0));

    GameFunc = TestProc;
}

/*****************************************************************************************************/
/*                                       機能テストメイン処理                                        */
/*****************************************************************************************************/
void TestProc()
{
    static POINT   ptChara   = {0, 0};  // キャラクター座標
    static int     nCharaPtn = 0;       // キャラクターアニメパターン

    // キャラクター移動
    BYTE   byKeyState[256];
    DIGetKeyboardState(byKeyState);
    ※ここは各自考えましょう
    －ヒント－
      カーソルキーの上下左右の状態にあわせ、
      キャラクターの座標を増減します。
      キャラクターのｘ座標は ptChara.x
      キャラクターのｙ座標は ptChara.y
      に格納されています。


    // 背景転送
    DDBltFast(DDS_BACKBUF, 0, 0, DDS_OFFSCRN1, NULL, DDBLTFAST_WAIT);

    // キャラクター転送
    RECT   rcChara;
    rcChara.left   = nCharaPtn * 120;
    rcChara.top    = 0;
    rcChara.right  = rcChara.left + 120;
    rcChara.bottom = rcChara.top  + 120;
    DDBltFast(DDS_BACKBUF, ptChara.x, ptChara.y,
              DDS_OFFSCRN2, &rcChara, DDBLTFAST_SRCCOLORKEY | DDBLTFAST_WAIT);

    DDFlip();

    nCharaPtn = (nCharaPtn + 1) % 5;    // キャラクターパターン更新

    Sleep(15);
}
```

(４)(３)のプログラムを変更し、マウスの左ボタンが押されているときはキャラクターを左に、マウスの右ボタンが押されているときはキャラクターを右に動かすようにしましょう。

(５)(３)のプログラムを変更し、マウスカーソルの座標にキャラクターを表示するようにしましょう。

資料－参照型変数について－

C++には、C言語に存在しない参照型という変数が存在します。C言語では、関数に構造体を渡す場合、ポインタを利用するのが一般的です。しかし、->演算子の概念が多少複雑であり、また未初期化ポインタや無効なポインタが渡されたときには不正な参照が発生してしまいます。
そこで、C++ではポインタに似て異なる「参照」という概念が導入されました。

```
int     a = 10;
int&    b = a;    // bはaの参照。変数名は別でも同じ変数を参照する

b = 9999;         // bはaの参照なので、bを9999にするとaも9999になる
```

参照型は、"&"を伴って宣言された変数で、上記のように必ず宣言と同時に初期化が必要になります。参照型の変数は、同じ変数に別の名前を付けるような効果が得られます。名前は別でも同じ変数を参照しているので、一方を変更するともう一方も変更されます。
参照型は、構造体やクラスの変数を関数の引数にするときに使用します。これは、不正なポインタによるバグや、コピーを作成するオーバーヘッドを少なくするためです。なお、参照型で渡された引数を関数内で変更すると、当然呼び出し元の変数も書き換えられます。