オブジェクト指向と
ゲームプログラミング

▷　フレームワーク編 － 第４回　ウィンドウプロシージャ　◁

課　　題

ゲームアプリケーションクラスにウィンドウプロシージャを追加しましょう。

(１)CGameAppクラスにメッセージを処理する機能を追加します。

メッセージを処理するには、メッセージループとウィンドウプロシージャが必要です。以下のプロトタイプをCGameAppクラスのprivateなメンバに追加しましょう。

```
// メッセージ処理
bool MessageLoop();

static LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WindowProc(const HWND hWnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam);
```

MessageLoop関数は、メッセージループを確立する関数です。WndProc関数とWindowProc関数がウィンドウプロシージャです。
なぜ、ウィンドウプロシージャが２つあるのかというと、WNDCLASSEX構造体のlpfnWndProcメンバにstaticでないメンバ関数を指定することができないという問題を解決するためです。

変数や引数に、関数(のアドレス)を指定する場合、staticでないメンバ関数を指定できない場合が多くあります。これは、staticでないメンバ関数には、引数にthisポインタが暗黙のうちに追加されるためです。グローバル関数やstaticなメンバ関数は、プロトタイプで指定した引数のみ関数に渡されます。それに対し、staticでないメンバ関数は、引数にthisポインタが追加され、「thisコール」という形式で関数が呼び出されます。引数が１つ増え、さらに呼び出し形式が異なってしまうので、コンパイルエラーとなるのです。

この問題を解決するため、staticな関数とstaticでない関数の２つ用意します。staticな関数ならエラーになりません。しかし、staticな関数では、staticなメンバにしかアクセスできないという制限があります。そこで、あらかじめthisポインタをstaticな関数でもアクセスできる場所に格納しておき、それをとおしてstaticな関数からstaticでない関数を呼び出します。そして、その中で本命の処理をする、ということを行います。

(２)クラス関数WndProc内でCGameAppクラスのthisポインタを使用できるようにするため、以下のメンバをCGameAppクラスのprivateに追加しましょう。

```
static CGameApp*    m_pThis;      // 自身へのポインタ
```

さらに、以下のプログラムを適切な場所に追加しましょう。

```
CGameApp*    CGameApp::m_pThis = NULL;
```

(３)CGameAppクラスのコンストラクタに以下のプログラムを追加しましょう。

```
m_pThis = this;    // thisポインタの保存
```

(４)CGameAppクラスのソースファイルにMessageLoop関数を実装します。以下のプログラムの足りない部分を補って完成させましょう。

```
/*****************************************************************************************************/
/*                                          メッセージループ                                         */
/*****************************************************************************************************/
bool ????????::MessageLoop()
{
    MSG    msg;
    while(true) {
        const BOOL    ret = ::??????????(&msg, NULL, 0, 0);  // メッセージの取得
        if(ret == 0 || ret == -1)
```

```
            return false;
        ::TranslateMessage(&msg);                       // メッセージの変換
        ::??????????????(&msg);                         // メッセージの送出
    }

    return true;
}
```

MessageLoop関数は、メッセージループを確立する関数です。WM_QUITメッセージを受信した場合またはエラーが発生した場合は、アプリケーションの終了を示すためにfalseを返し、それ以外の場合は続行を示すtrueを返します。

(５)CGameAppクラスのソースファイルにWndProc関数を実装します。この関数は、(３)で保存したthisポインタを介し、実際にメッセージを処理する関数(WindowProc関数)を呼び出すだけの関数です。
以下のプログラムの足りない部分を補って完成させましょう。

```
/*****************************************************************************************************/
/*                                        ウィンドウプロシージャ                                       */
/*****************************************************************************************************/
LRESULT CALLBACK ????????::WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    return m_pThis->WindowProc(hWnd, uMsg, wParam, lParam);
}
```

(６)CGameAppクラスのソースファイルにWindowProc関数を実装します。この関数が、実際にメッセージを処理する関数になります。
以下のプログラムの足りない部分を補って完成させましょう。

```
/*****************************************************************************************************/
/*                                          メッセージ処理                                           */
/*****************************************************************************************************/
LRESULT ????????::WindowProc(const HWND hWnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam)
{
    return ::DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

(７)CGameWndFrameクラスをCGameAppクラスに集約します。

ウィンドウはアプリケーションの一部であり、このような関係を「part-of関係」と呼びます。ソースコード上では、あるクラスのメンバ変数の型に、別のクラスを指定している状態となります。これを集約や合成と呼びます。
以下のプログラムを適切な場所に追加し、CGameWndFrameオブジェクトを適切なクラスに集約させましょう。

－追加１－
```
#include "GameWndFrame.hpp"
```

－追加２－
```
CGameWndFrame   m_Frame;        // ウィンドウフレーム
```

(８)CGameAppオブジェクトの作成時に、メンバ変数m_Frameを初期化するため、以下のコンストラクタ初期化子を適切な場所に追加しましょう。

```
m_Frame()
```

(９)以下のアクセス関数をCGameAppクラスに追加しましょう。

```
// アクセス関数
HWND GetHWnd() const { return m_Frame.GetHWnd(); }
```

GetHWnd関数により、「m_Frame.GetHWnd()」を「GetHWnd()」と短縮して記述することができます。また、CGameAppクラスの外でも、ウィンドウのハンドルを取得することができるようになります。

(10)アプリケーションの初期化時にウィンドウを生成するようにします。

ウィンドウを生成するには、CGameWndFrame::Create関数を呼び出します。このとき、引数にインスタンスのハンドルとウィンドウプロシージャを渡します。ウィンドウプロシージャは、(５)で作成しました。しかし、インスタンスのハンドルはプログラムの起動時にWinMain関数に渡されるものの、CGameAppクラスではアクセスできません(ただし、GetModuleHandle関数を呼び出すと取得できます)。
そこで、CGameApp::Initialize関数を以下のように変更します。

```
bool Initialize(const HINSTANCE hInstance);
```

こうすれば、CGameApp::Initialize関数内でインスタンスのハンドルを扱うことができます。これでCGameWndFrame::Create関数を呼び出すことができるようになります。

以下のプログラムの足りない部分を補ってInitialize関数を完成させましょう。

```
bool CGameApp::Initialize(const HINSTANCE hInstance)
{
    // ウィンドウ生成
    if(m_Frame.??????(hInstance, WndProc) == false)
        return false;

    return true;    // 成功
}
```

(11)CGameApp::Initialize関数の引数が変更になったので、WinMain関数の「アプリケーション初期化」を以下のように変更しましょう。

```
    // アプリケーション初期化
    if(App.Initialize(hInstance) == false)
        return -1;
```

(12)アプリケーションの解放時にウィンドウを破棄します。以下のプログラムの足りない部分を補ってRelease関数を完成させましょう。

```
void CGameApp::Release()
{
    m_Frame.???????();      // ウィンドウ破棄
}
```

(13)アプリケーションのメイン処理にメッセージループを追加します。以下のプログラムの足りない部分を補ってMain関数を完成させましょう。

```
int CGameApp::Main()
{
    // メインループ
    while(true) {
        // メッセージループ
        if(???????????() == false)
            break;
    }

    return 0;       // 正常終了
}
```

※今回の課題だけでは、プログラムが不完全なため終了しません。ウィンドウは消えますが、プログラムはメモリ上に存在しています。プログラムの完全な終了は、Ctrl+Shift+Esc(WinNT/2000/XP)でタスクマネージャを呼び出して行います。この不具合は次回の課題で修正します。