ゲームプログラミング

▶　基礎編－第５回　WinMain.cppの作成　◀

ウィンドウの生成や破棄、メッセージの処理といったWindowsアプリケーションの基本的な処理を行うWinMain.cppを作成します。

マウスカーソルの消去

ゲームによっては、標準で表示されているマウスカーソルを消したい場合があります。また、マウスカーソルはDirectDrawと相性が悪い場合もあるので、特にフルスクリーンで動作するゲームの場合は、標準のマウスカーソルは消しておいた方が安全です。

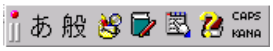
マウスカーソルは、ShowCursor関数で表示状態を変更することができます。Windowsは、内部に「マウスカーソル表示カウンタ」という情報を持っています。このカウンタが０以上のときにマウスカーソルが表示されます。ShowCursor関数は、このカウンタを増減することができるので、カーソルの表示状態を変更することができます。引数にTRUEを指定するとカウンタが１つ増やされ、FALSEなら１つ減らされます。カーソルを消去(非表示)にするには、引数にFALSEを指定し、カウンタが０未満になるまでループさせます。

```
// カーソル消去
while(0 <= ShowCursor(FALSE))
    ;
```

また、DirectInputでマウスを占有することによってもマウスカーソルを消すことができます。

IMEツールバーの消去

画面全体を覆うウィンドウを生成したとき、MS-IMEツールバーが表示されることがあります。



▲MS-IMEツールバー

このツールバーを消去するには、WINNLSEnableIME関数を使用します。１つ目の引数にツールバーが表示されているウィンドウのハンドル、２つ目の引数に表示フラグを指定します。表示フラグがFALSEならツールバーが非表示になり、TRUEなら表示されます。

```
WINNLSEnableIME(hWnd, FALSE);  // hWndはメインウィンドウのハンドル
```

なお、この関数を使用するには、<winnls32.h>をインクルードする必要があります。

スクリーンセーバを起動させない

Windowsの設定でスクリーンセーバが起動するようになっている場合、ゲーム起動中にもかかわらず、条件を満たすと強制的にスクリーンセーバに切り替わります。

ムービー再生中やエンディングのような、キーボードやマウスにいっさい触れないシーンがゲームには多くあります。このようなシーンでは、特にスクリーンセーバの起動条件を満たしやすくなってしまいます。スクリーンセーバは、あるメッセージを処理することによって起動を阻止することができます。

スクリーンセーバが起動するとき、アクティブなアプリケーションに対してWM_SYSCOMMANDメッセージが送信され、追加情報(WPARAM)にSC_SCREENSAVEが格納されてきます。このとき、ウィンドウプロシージャで何も処理せずに１を返すと、スクリーンセーバの起動を阻止することができます。

```
case WM_SYSCOMMAND:
  if(SC_SCREENSAVE == wParam)
    return 1;                                    // スクリーンセーバは起動させない
  return DefWindowProc(hWnd, uMsg, wParam, lParam);    // ほかのコマンドはデフォルト処理
```

○ 課題 ○

プロジェクトに以下のソースファイルWinMain.cppを作成し、追加しましょう。WinMain.cppには、ゲームプログラムのウィンドウ関係の処理を記述します。
WinMain.cppを作成するにあたって、以下の条件を満たすように作成しましょう。また、このソースファイルには足りない部分があるので、「関数解説」を参考にしながら完成させてください。

－条件－
・画面全体を覆うシンプルなウィンドウ(WS_POPUP)を生成
・カーソルを消去
・IMEを消去
・このプログラムがアクティブ状態にあるときは、スクリーンセーバを起動しない
・ESCキーでプログラム終了
・アイコンとバージョン情報を作成

○WinMain.cpp関数解説

InitApp関数
－説明－
InitApp関数は、メインウィンドウの生成やゲームの初期化といった、アプリケーションの初期化処理を行います。

－パラメータ－
HINSTANCE hInstance…アプリケーションインスタンスのハンドル。メインウィンドウの生成やゲームの初期化に必要
const int nShowCmd…ウィンドウ表示モード。メインウィンドウの生成に必要

－戻り値－
すべての初期化処理に成功した場合はtrue，それ以外はfalseを返します。

ReleaseApp関数
－説明－
ReleaseApp関数は、メインウィンドウの破棄やゲームで使用した資源の解放といった、アプリケーションの解放処理を行います。

－パラメータ－
const HWND hWnd…メインウィンドウのハンドル。このウィンドウが破棄されます

－戻り値－
なし

CreateMainWindow関数
－説明－
CreateMainWindow関数は、メインウィンドウの「ウィンドウクラスの登録」、「生成」、「表示」、マウスカーソルの消去、IMEツールバーの消去といった処理を行います。

－パラメータ－
HINSTANCE hInstance…アプリケーションインスタンスのハンドル。メインウィンドウの生成に必要
const int nShowCmd…ウィンドウ表示モード。メインウィンドウの生成時に使用

－戻り値－
関数が成功した場合はメインウィンドウのハンドル、それ以外はNULLを返します。

ウィンドウメッセージを処理する関数群

WndProc関数	メッセージを処理するウィンドウプロシージャです。
OnKeyDown関数	WM_KEYDOWNメッセージを処理します。
OnSysCommand関数	WM_SYSCOMMANDメッセージを処理します。
OnClose関数	WM_CLOSEメッセージを処理します。
OnDestroy関数	WM_DESTROYメッセージを処理します。

―WinMain.cpp―

```
/*
================================================================================
                                ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2003 Hibikino software. All rights reserved.
================================================================================

  【対象ＯＳ】
    Microsoft Windows98/2000以降

  【コンパイラ】
    Microsoft VisualC++ 6.0J ServicePack5

  【プログラム】
    WinMain.cpp
                WinMain

  【履歴】
    * Version    0.00       2002/04/dd  hh:mm:ss  初版

================================================================================
*/

/*****************************************************************************/
/*                                インクルードファイル                       */
/*****************************************************************************/
※ここは各自考えましょう
※ここは各自考えましょう
※ここは各自考えましょう

/*****************************************************************************/
/*                                   定　数                                  */
/*****************************************************************************/
LPCTSTR   MAINWND_CLASS = "GMPROG_MAINWND";                 // ウィンドウクラス名
LPCTSTR   MAINWND_TITLE = "ゲームプログラミング";           // ウィンドウタイトル

/*****************************************************************************/
/*                               外部参照変数                                */
/*****************************************************************************/

/*****************************************************************************/
/*                               グローバル変数                              */
/*****************************************************************************/

/*****************************************************************************/
/*                             プロトタイプ(プライベート)                    */
/*****************************************************************************/
static bool InitApp(const HINSTANCE hInstance, const int nShowCmd);
static void ReleaseApp(const HWND hWnd);

static HWND CreateMainWindow(const HINSTANCE hInstance, const int nShowCmd);

static LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
static int OnKeyDown   (const HWND hWnd, const WPARAM wParam, const LPARAM lParam);
static int OnSysCommand(const HWND hWnd, const WPARAM wParam, const LPARAM lParam);
static int OnClose     (const HWND hWnd, const WPARAM wParam, const LPARAM lParam);
static int OnDestroy   (const HWND hWnd, const WPARAM wParam, const LPARAM lParam);

/*****************************************************************************/
/*                                   WinMain                                 */
/*****************************************************************************/
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nShowCmd)
{
    // アプリケーション初期化
    if(false == ???????(hInstance, nShowCmd))
        return 0;

    // メッセージループ
    MSG   msg;
    ZeroMemory(&msg, sizeof(msg));
    while(true) {
        const BOOL   bRet = GetMessage(&msg, NULL, 0, 0);
        if(0 == bRet || -1 == bRet)
```

```
            break;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }   // while(true)

    return msg.wParam;
}

/*****************************************************************************************************/
/*                                        アプリケーション初期化                                     */
/*****************************************************************************************************/
bool InitApp(const HINSTANCE hInstance, const int nShowCmd)
{
    // メインウィンドウ生成
    const HWND   hWnd = ???????????????(hInstance, nShowCmd);
    if(NULL == hWnd)
        return false;

    return true;
}

/*****************************************************************************************************/
/*                                        アプリケーション解放                                       */
/*****************************************************************************************************/
void ReleaseApp(const HWND hWnd)
{
    ????????????(hWnd);    // メインウィンドウ破棄
}

/*****************************************************************************************************/
/*                                        メインウィンドウ生成                                       */
/*****************************************************************************************************/
HWND CreateMainWindow(const HINSTANCE hInstance, const int nShowCmd)
{
    // ウィンドウクラス登録
    WNDCLASSEX   wcx;
    ZeroMemory(&wcx, sizeof(wcx));
    wcx.cbSize        = sizeof(wcx);
    wcx.style         = 0;
    wcx.lpfnWndProc   = WndProc;
    wcx.cbClsExtra    = 0;
    wcx.cbWndExtra    = 0;
    wcx.hInstance     = hInstance;
    wcx.hIcon         = ※ここは各自考えましょう
    wcx.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wcx.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wcx.lpszClassName = MAINWND_CLASS;
    wcx.lpszMenuName  = NULL;
    wcx.hIconSm       = NULL;
    if(0 == ???????????????(&wcx)) {
        OutputDebugString("*** Error - ウィンドウ登録失敗\n");
        return NULL;
    }

    // メインウィンドウ生成
    HWND   hWnd = ※ここは各自考えましょう
    if(NULL == hWnd) {
        OutputDebugString("*** Error - ウィンドウ生成失敗\n");
        return NULL;
    }

    // カーソル消去
    ※ここは各自考えましょう

    // IME消去
    ※ここは各自考えましょう

    SetFocus(hWnd);                 // フォーカス設定
    ShowWindow(hWnd, nShowCmd);     // ウィンドウ表示
    UpdateWindow(hWnd);             // クライアント領域更新

    return hWnd;
}
```

```
/*****************************************************************************************************/
/*                                          ウィンドウプロシージャ                                   */
/*****************************************************************************************************/
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch(uMsg) {
      // キーダウン
      case WM_KEYDOWN:
        return ?????????(hWnd, wParam, lParam);

      // システムコマンド処理
      case WM_SYSCOMMAND:
        return ※ここは各自考えましょう

      // ウィンドウクローズ
      case WM_CLOSE:
        return ※ここは各自考えましょう

      // ウィンドウ破棄
      case WM_DESTROY:
        return ※ここは各自考えましょう
    }   // switch(uMsg)

    return ※ここは各自考えましょう
}

/*****************************************************************************************************/
/*                                      WM_KEYDOWNメッセージ処理                                     */
/*****************************************************************************************************/
int OnKeyDown(const HWND hWnd, const WPARAM wParam, const LPARAM lParam)
{
    switch(wParam) {
      // ESCキー(終了)
      case ?????????:
        ReleaseApp(hWnd);
        break;
    }   // switch(wParam)

    return 0;
}

/*****************************************************************************************************/
/*                                     WM_SYSCOMMANDメッセージ処理                                   */
/*****************************************************************************************************/
int OnSysCommand(const HWND hWnd, const WPARAM wParam, const LPARAM lParam)
{
    switch(wParam) {
      // スクリーンセーバ起動防止
      case SC_SCREENSAVE:
        return ?;
    }   // switch(wParam)

    return DefWindowProc(hWnd, WM_SYSCOMMAND, wParam, lParam);
}

/*****************************************************************************************************/
/*                                       WM_CLOSEメッセージ処理                                      */
/*****************************************************************************************************/
int OnClose(const HWND hWnd, const WPARAM wParam, const LPARAM lParam)
{
    ReleaseApp(hWnd);

    return 0;
}

/*****************************************************************************************************/
/*                                      WM_DESTROYメッセージ処理                                     */
/*****************************************************************************************************/
int OnDestroy(const HWND hWnd, const WPARAM wParam, const LPARAM lParam)
{
    PostQuitMessage(0);

    return 0;
}
```