ゲームプログラミング

▶　第１回　Windowsプログラムの基礎　◀

Windowsプログラミングに必要となる基本的な知識を解説します。

WindowsAPI

Windowsで動作するアプリケーションのプログラムは、API(アプリケーション・プログラム・インタフェース)と呼ばれる関数群を組み合わせて実現したい機能を構築していきます。

APIは、Windowsが提供する機能を呼び出すための関数群です。Ｃ言語の関数と同様の形式をとり、「引数」と「戻り値」があります。引数は動作指示、戻り値は動作結果という役割を持っています。APIでは戻り値も重要で、基本的に戻り値を調べて以降の処理に生かします。もし、エラーであった場合は処理を中断したり、エラー処理を行うようにします。

Windowsが提供するほぼすべての機能は、APIを呼び出すだけで簡単に使用することができます。

```
// APIの例…MessageBox関数(メッセージボックスを表示)
MessageBox(NULL, "続けますか？", "タイトル", MB_ICONEXCLAMATION | MB_OKCANCEL);
```

以下のようにAPIの戻り値を変数に代入したり、if文などの条件に使用することができます。

```
// 変数nRetにMessageBox関数の戻り値(押されたボタン)を代入
int    nRet = MessageBox(NULL, "OKを押してください", "タイトル", MB_ICONEXCLAMATION | MB_OKCANCEL);

// 押されたボタンで分岐
if(IDOK == MessageBox(NULL, "OKを押してください", "タイトル", MB_ICONEXCLAMATION | MB_OKCANCEL)) {
    // OKボタンが押された
} else {
    // ｷｬﾝｾﾙボタンが押された(またはエラーが発生した)
}
```

データ型

Windowsプログラミングでは、Ｃ／Ｃ++のintやdoubleといった基本的なデータ型のほかに、独自のデータ型を多く使用します。独自のデータ型は、ヘッダファイル<windows.h>やその関連ヘッダファイル(windef.h)で定義されていて、windows.hをインクルードすることによりほとんどのデータ型が使用できるようになっています。よく使うデータ型は以下のものです。

型	サイズ	値の範囲	備　　考
void/VOID	―	な　し	
LPVOID	32	32ビットのアドレスを格納	void*と同じ
bool	8	true, false	
BOOL	32	TRUE, FALSE	bool型とは互換性なし。実体はint型
char/CHAR	8	-128～127	
BYTE	8	0～255	unsigned charと同じ
LPBYTE	32	32ビットのアドレスを格納	unsigned char*と同じ
wchar_t	16	UNICODE文字型	1文字を16ビットで表すunicode文字を格納
WCHAR	16	UNICODE文字型	wchar_tとほとんど同じ
TCHAR	8/16	UNICODEまたはchar文字	コンパイル時にcharとwchar_t切り替えられる
short	16	-32768～32767	
WORD	16	0～65535	unsigned shortと同じ
LPWORD	32	32ビットのアドレスを格納	unsigend short*およびWORD*と同じ
int/INT	32	-2147483648～2147483647	
UINT	32	0～4294967295	unsigend intを省略した記述方法
long/LONG	32	-2147483648～2147483647	
LPLONG	32	32ビットのアドレスを格納	long*と同じ

型	サイズ	値の範囲	備　　考
DWORD	32	0～4294967295	unsigned longと同じ
LPDWORD	32	32ビットのアドレスを格納	unsigned long*およびDWORD*と同じ
__int64	64	-9223372036854775808 ～ 9223372036854775807	符号あり64ビット整数
DWORDLONG	64	0～18446744073709551615	符号なし64ビット整数
float/ FLOAT	32	1.175494351E-38 ～ 3.402823466E+38	
double	64	2.2250738585072014E-308 ～ 1.7976931348623158E+308	Windowsではlong double型も同等
LPSTR	32	文字列のアドレスを格納	char*と同じ。文字列へのポインタを格納
LPCSTR	32	文字列のアドレスを格納	const char*と同じ。固定文字列へのポインタ
LPTSTR	32	文字列のアドレスを格納	コンパイル時にchar*とwchar_t*を切り替えられる
LPCTSRT	32	文字列のアドレスを格納	LPTSTRの固定文字列版。文字列は変更不可
HANDLE	32	32ビット整数値	リソースを識別する値（ハンドル）を格納

※「サイズ」はデータを格納するのに必要なビット数を表しています。

・HANDLE型

HANDLE(ハンドル)型は、Windowsが管理するリソース(ウィンドウ、ファイル、フォント、ビットマップ、カーソルなど)の情報テーブルへアクセスするための値です。おもに、リソースを識別したり、利用するときに必要となります。

HANDLE型には、HINSTANCE型やHDC型といった多数の派生型が存在します。どの型も32ビットの値を格納しますが、実体はint型であったり、LPVOID型であったり、ある構造体のアドレスであったりとさまざまです。たとえば、HWND型はウィンドウを管理ために使われる32ビットの整数値であり、HBITMAPはビットマップを管理するために使われる32ビットの整数値です。HWND，HBITMAP，HFONT，HMODULE，HINSTANCEなど、"H"で始まる型は、ほとんどがハンドル型です。

アプリケーションは、Windowsからハンドルを取得することで間接的にリソースを参照するようになっています。こうすると、他のアプリケーションに動作を切り替えたときに、自アプリケーションのリソースが仮想メモリに退避されても(格納場所が変わっても)見失わないように設計されているのです。

APIからは、デバイスやリソースを直接制御することができないようになっています。デバイスやリソースからハンドルを取得し、それをAPIに渡すことによって間接的に制御するようになっています。

命名規則

Windowsプログラムでは、関数や変数の名前の付け方もC／C++と異なる部分があります。

関数の命名規則は、動詞とこれに続く名詞で構成します。動詞と名詞の最初の文字は大文字にします。APIもこの規則に従って命名されています。

ウィンドウを生成する関数…CreateWindow関数
ウィンドウを破棄する関数…DestoryWindow関数

変数の命名規則は、ハンガリー表記法というMicrosoft社が提唱する独特の方法をとる場合が多くあります。この規則は批判も多くありますが、Microsoft社が提供するヘルプファイルやサンプルプログラムは、ほとんどがこの方法で記述されています。

ハンガリー表記法では、データ型を変数名に埋め込む方法をとります。これをプリフィックス文字といいます。変数名は、小文字のプリフィックス文字と大文字で始まる名前となります。たとえば、int型の変数名は、名前の先頭にデータ型を表す"n"をつけ、その後に名前が続きます(nCountやnMax)。代表的なデータ型のプリフィックス文字は、以下のものを使うのが一般的です。

ﾌﾟﾘﾌｨｯｸｽ	データ型	ﾌﾟﾘﾌｨｯｸｽ	データ型	ﾌﾟﾘﾌｨｯｸｽ	データ型
b	bool,BOOL	h	ハンドル	w	WORD
c	char(１文字)	l	long,LONG	sz	文字列
d	double	lp	ポインタ	lpsz	文字列へのポインタ
dw	DWORD	n	short,int,INT	rgb	RGB構造体
f	float,ﾋﾞｯﾄﾌｨｰﾙﾄﾞ	pt	POINT構造体	by	BYTE

WinMain関数

C／C++プログラムでは、main関数から実行が開始されますが、Windowsプログラムではmain関数から実行が開始されます。一般的なWindowsプログラムにはmain関数がありません。

通常、WinMain関数は、以下のような形式を取ります。

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nShowCmd)
```

プログラムが起動されると、WinMain関数はWindowsから４つのパラメータを受け取ります。それぞれ以下の情報が格納されます。

HINSTANCE	hInstance……プログラムを識別するための値（インスタンスハンドル）
HINSTANCE	hPrevInstance…常にNULL。Windows3.1との互換性のために残っています
LPSTR	lpszCmdLine……コマンドラインオプション(/Pなどの起動時のオプション)
int	nShowCmd……メインウィンドウの表示モード(最大化・最小化・通常表示など)

特に最初のパラメータ「インスタンスハンドル」はいくつかのAPIで必要になります。

WinMain関数は、WINAPIという呼び出し規約(コンパイル時の関数呼び出しの規約)を使ってコンパイルされなければならないので、戻り値intと関数名WinMainの間に「WINAPI」がついています。WINAPIのかわりに「APIENTRY」や「PASCAL」が使われることもあります。

○ 課題 ○

(１)新規プロジェクトを"Win32 Application(空のプロジェクト)"で作成し、以下のプログラムを入力して実行結果を確認しましょう。

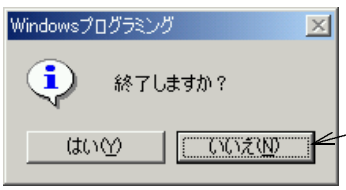
```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nShowCmd)
{
    MessageBox(NULL, "メッセージ", "タイトル", MB_ICONEXCLAMATION | MB_OKCANCEL);

    return 0;
}
```

(２)Win32API プログラマーズ リファレンス(API32JPN.HLP)またはMSDNライブラリでMessageBox関数を調べ、以下のメッセージボックスと同じものを作成しましょう。

Win32API プログラマーズ リファレンスは、"S"キーを押すと検索モードになります。MSDNライブラリの場合は左側の"キーワード(N)"をクリックすると同様の状態になります。ここで関数名を入力し、目的のAPIを検索します。



※「いいえ(N)」が選択されている状態にしましょう。

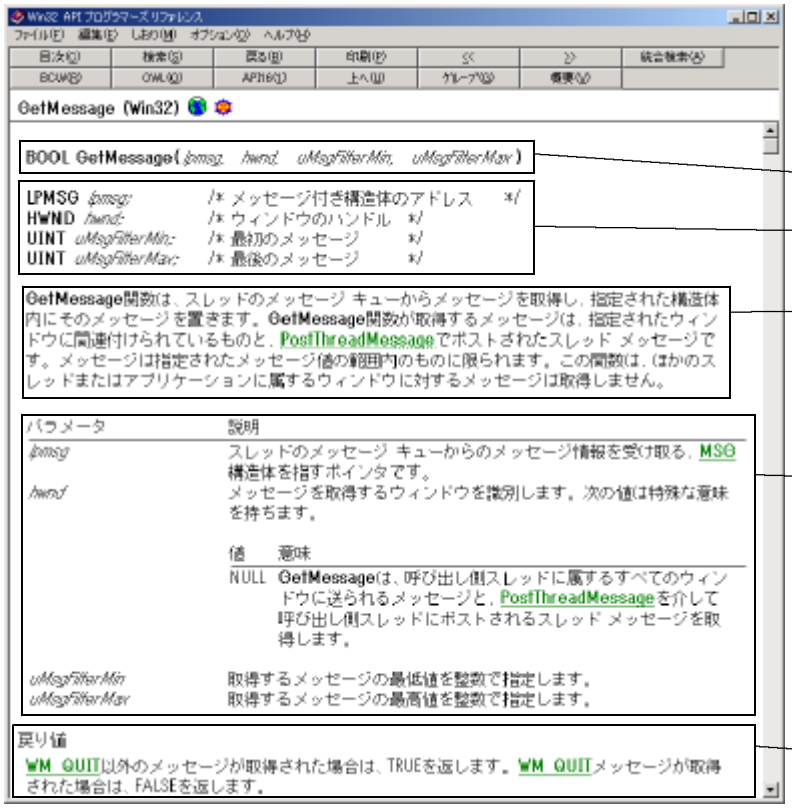
ヒント：複数の値を組み合わせるときは、"|"を使います。(例：MB_ICONEXCLAMATION | MB_OKCANCEL)

(３)(２)で作成したメッセージボックスの"はい"ボタンが押されるまで、プログラムが終了しないようにしてみましょう。

ヒント：MessageBox関数の戻り値をループ終了条件に利用します。

資料　－APIリファレンス－

APIリファレンスは以下のようになっています。独特の表現が使われていますが、APIやDirectXのメソッドに関する有益な情報が記述されているので、読みこなせるようにしましょう。



API(関数)のプロトタイプ

パラメータの型とその説明

APIの詳細な説明

パラメータについての解説。
渡すべき値が詳細に記述されています。

関数の戻り値の説明