BREW & EZアプリ
ゲームプログラミング

○　総集編　シューティングゲームの作成　○

BREW

　BREW(Binary Runtime Environment for Wireless)は、アメリカのQUALCOMM(クアルコム)社が発表した、CDMA方式の携帯電話向けアプリケーション開発環境(BREW SDK：Software Development Kit)および実行環境(BREW AEE：Application Execution Environment)で、アプリケーションの配信や課金のためのシステムも備えています。

　BREWは、携帯電話間の仕様の違いを吸収し、さまざまな機種に対応できるように設計されています。世界的にもプログラマの多い、Ｃ／Ｃ++を採用していますが、BREW用JavaVMの搭載によってJavaも利用できるように設計されています。
　アプリケーション開発者は、さまざまなBREW携帯電話端末向けに、極めて移植性の高い、ゲーム、ブラウザ、電話帳などといった各種アプリケーションを開発することができます。

　現在、QUALCOMMは世界最大の顧客を抱えています。日本では、KDDIが同社製のチップと通信ソリューションの提供を受け、2003年からEZアプリ(BREW)という名称でサービスを開始しています。また、QUALCOMMは、Vodafone向けにもチップを提供しているだけでなく、BREWクライアントを搭載したFOMA端末を開発すべく、NTTドコモと協力しています。

BREWの特徴

　BREWには、以下のような優れた特徴があります。

・軽量
　BREW実行環境は、極めて少ないメモリで軽快に動作するように実装されています。幅広く、多種多様な携帯電話端末に搭載される可能性を秘めています。

・世界標準
　BREW APIの仕様は世界で共通です。従って、BREWで開発されたアプリケーションは、基本的に世界中のすべてのBREW対応携帯電話端末で動作します。仕様差がある場合でも、BREWアプリケーションの移植は比較的スムーズに行えます。

・拡張性
　エクステンション機構により、サードパーティがBREWのAPIを拡張できます。この仕組みを利用して、携帯電話端末の機能そのものをソフトウェア的にカスタマイズできます。

・自在性
　BREWでは、携帯電話端末(BREW実行環境)の機能をAPIで直接制御でき、アセンブラの知識を駆使したプログラミングが部分的に可能です。端末機の内部機能をフルに活かした、自在性のある自由度の高いアプリケーションを開発できます。Javaでは実現不可能なこともBREWでは可能となるケースが多々あります。

・オープン性
　BREW開発環境は、無料で配布されています。また、Visual C++などの一般的な開発環境で作成することができ、BREW上にJavaVMやFlash Playerなどのようなアプリケーション・プラットフォームを実装することもできます。多種多様なアプリケーションをあたかもBREWアプリケーションのように動かすことができるように設計されています。

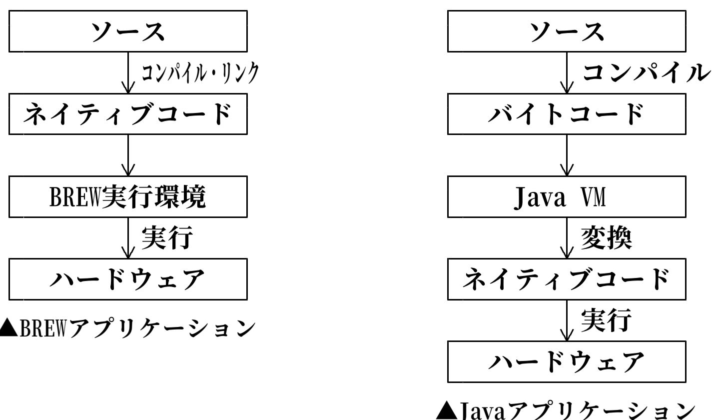
・ネットワーキング
　TCP/IPが使えるので、エンドトゥエンドの柔軟性のあるネットワークプログラミングが可能です。Push型の情報配信アプリケーションやインスタントメッセンジャーのようなP2P型のアプリケーションを開発することもできます。

BREWのしくみ

　BREWプログラミングでは、BREW APIを使用し、Ｃ／Ｃ++でプログラムを書きます。BREW APIは、BREW実行環境の機能を呼び出すための関数群です。これは、携帯Javaプログラミングが Java APIを使用してJavaでプログラムを書くのとまったく同じです。

作成したプログラムは、C／C++コンパイラおよびリンカにより、BREW実行環境がそのまま解釈・実行できるネイティブコードに変換されます。

BREWアプリケーションは、直接ハードウェアを制御するのではなく、機種間の違いを吸収するBREW実行環境を利用して実行されますが、ネイティブコードであるため、バイトコードをネイティブコードに変換しながら実行するJavaアプリケーションに比べ、高速に実行されます。



▲BREWアプリケーション

▲Javaアプリケーション

BREW SDK

BREWアプリケーションの開発を行うには、BREW開発キット(BREW SDK)が必要です。BREW SDKには、以下のツールが含まれています。

- BREW AEEヘッダーファイル
- BREW AEEソースファイル
- BREW APIリファレンスおよびオンラインヘルプ
- BREWドキュメント
- BREWシミュレータ
- MIFエディタ
- リソースエディタ
- 圧縮画像作成ツール
- デバイスコンフィギュレータ
- サンプルアプリ

BREW SDKをインストールすれば、すぐにBREWアプリケーションの開発を行うことができます。また、Visual C++と統合することができるので、統合開発環境を利用した開発も行えます。

BREWアプリケーションの配布方法

BREWアプリケーションを配信するためには、KDDIのEZアプリ(BREW)公式コンテンツプロバイダとなり、当該アプリケーションがKDDIの検証試験に合格する必要があります。残念ながら、BREWアプリケーションを自由に配信することはできません。これは、認証式とすることで安全性を高めです。

課　題

BREWアプリケーションを開発するのに必要なソフトウェアをインストールしましょう。

(１)あらかじめ、Visual C++をインストールしておきます。なお、BREW SDKのインストーラは、Visual C++ 2005には対応していません(ヘッダーファイルなどのツール群は使用できます)。

(２)BREW SDKをインストールします。BREW SDKは、ファイルが配布されておらず、Qualcommのサイトからオンラインインストールしなければなりません。また、オンラインインストーラを起動するには、メールアドレスの登録が必要になります。
　Qualcommのサイト(http://brew.qualcomm.com/brew/ja/)にアクセスしましょう。

(３)左側の「関連リンク」の下にある「ディベロッパー」をクリックし、ディベロッパーリソースのページに移動します。左側の「ディベロッパー」の中段にある「ダウンロード BREWツール」をクリックしましょう。

(４)ここは英語のページになります。メールアドレスが登録済みの場合は、メールアドレスを入力して「Submit」ボタンをクリックします。未登録の場合は、「click here」のリンクをクリックします。

(５)登録フォームに以下の必要事項(会社名も必要です)を入力し、「I agree」にチェックを入れて「Continue」ボタンをクリックしましょう。

First Name	名
Last Name	姓
Email Address	メールアドレス
Company Name	会社名
Address	住所
City	都道府県
Zip Code/Postal Code	郵便番号
Country	国

(６)(４)と(５)に成功すると、BREWの開発・商用化ツールのページに移動し、BREW SDKがインストールできるようになります。「BREW SDK 3.1」をクリックしましょう。

(７)BREW SDK 3.1のダウンロードのページに移動します。「BREW SDKR 3.1」の「日本語 インストール」をクリックしましょう(ActiveXを利用してイントールを行うので、使用可能にしておきましょう)。

(８)「BREWR ダウンロードページにようこそ」というページに移動したら、「クリックしてインストールを開始する」をクリックします。インストールに必要なファイルがダウンロードされた後、インストーラが起動するので、指示にしたがってインストールしましょう。

(９)BREW SDK Toolsをインストールします。ツールには、MIFエディタやリソースエディタなどが含まれています。BREW SDK 3.1のダウンロードのページに戻り、「BREWR SDK Tools」の「日本語 インストール」をクリックしましょう(これもActiveXを利用します)。以後の手順は、(８)と同じです。

(10)アドインをイントールします。BREWの開発・商用化ツールのページに移動しましょう。

(11)「各種開発ツール」をクリックし、その他の開発ツールのページに移動します。「BREWR Add-Ins 3.0 for Microsoft Visual Studio」の「インストール」をクリックしましょう(これもActiveXを利用します)。以後の手順は、(８)と同じです。

(12)KDDIのサイトから、EZアプリを作成するのに必要なファイルをダウンロードします。KDDIのEZアプリ技術情報サイト(http://www.au.kddi.com/ezfactory/tec/spec/brew.html)にアクセスしましょう。

(13)BREW 2.1用の「A5501Tデバイス構成ファイル」と「A5503SAデバイス構成ファイル」をダウンロードして解凍し、BREW SDKをインストールしたフォルダの「Devices」フォルダにコピーしましょう(softkey.barは、なくてもかまいません)。※現時点では、ダウンロードできなくなっています

(14)BREW APIリファレンスのショートカットを作成します。BREW SDKをインストールしたフォルダの「Help」フォルダにある「BREWAPIReference.chm」のショートカットを適切な場所に作成しましょう。

BREWモジュール

BREWアプリケーションは、実機にダウンロードできるようになるまで時間が掛かるので、開発初期はSDKに付属のBREWシミュレータで動作確認を行います。シミュレータで実行するBREWアプリケーションの実体は、拡張子がdllのファイルです。これをBREWモジュールといいます。

BREWモジュールを実行するには、BREWモジュールに関する基本的な情報を記述したMIFファイルが必要です。MIFファイルはBREWモジュールと同じ名前でなければなりません。たとえば、SampleGame.dllモジュールには、SampleGame.mifというMIFファイルが必要になります。２つのファイルは、必ずペアで扱います。どちらかが欠けてしまったり、正しい場所に保存されていない場合、そのアプリケーションは起動できません。

BREWモジュールには、複数のBREWアプリケーションを含ませることができます。このBREWアプリケーションを、正確にはBREWアプレットといい、携帯電話のアプリ一覧に表示されるアイコンひとつひとつに対応します。

BREWアプレットは、クラスIDを持ちます。クラスIDとは、BREWアプレットを識別するための32ビットの値で、BREWアプレットごとに一意に割り当てられます。このIDは、正式にはQualcommやKDDIに申請して発行してもらうものですが、開発の段階ではどんなIDを使用しても構いません。

課　題

BREWアプリ用のプロジェクトを作成しましょう。

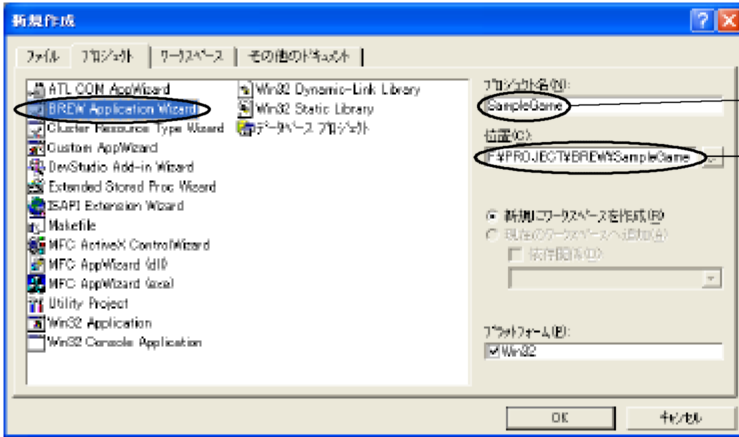
(１)BREWアプリのプロジェクトを保存するフォルダを作成しましょう(※日本語は使用できません)。このフォルダに、プロジェクトのフォルダとMIFファイルを保存します(MIFファイルは各プロジェクトのフォルダではなく、ここで作成したBREWプロジェクト全体のフォルダに保存します)。

```
BREWプロジェクト用フォルダ
│
├プロジェクトＡ用フォルダ
│  ├プロジェクトＡ.c
│  ├プロジェクトＡ.brh
│  ├プロジェクトＡ.bar
│  └プロジェクトＡ.dll
│
├プロジェクトＢ用フォルダ
│  ├プロジェクトＢ.c
│  ├プロジェクトＢ.brh
│  ├プロジェクトＢ.bar
│  └プロジェクトＢ.dll
│
├プロジェクトＡ.mif
└プロジェクトＢ.mif
```

(２)BREWアプリのプロジェクトを作成します。Visual C++を起動しましょう。

(３)メニューの「ファイル(F)」→「新規作成(N)」を選択しましょう。新規作成ダイアログが表示されます。

（４）「ﾌﾟﾛｼﾞｪｸﾄ」タブの「BREW Application Wizard」を選択した状態で、「位置(C)」に(１)で作成したフォルダを設定します。その後、「ﾌﾟﾛｼﾞｪｸﾄ名(N)」を入力しましょう。



プロジェクト名を入力(英数字と_のみ)

「BREWアプリのプロジェクトを保存するフォルダ」＋「プロジェクト名」であることを確認

正しく設定できたら、OKボタンをクリックします。

（５）「BREW Application Wizard - ｽﾃｯﾌﾟ１ / 2」が表示されます。

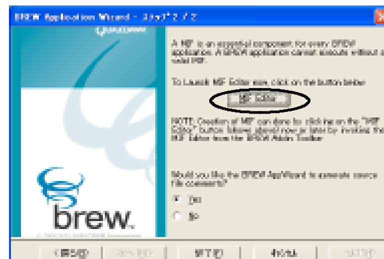
ここでは、BREWアプリケーションで使用する機能を選択します。以下のような機能があります。

項目名	機　能
File	ファイル入出力
Network	ネットワーク通信
Database	データベース
TAPI/SMS	電話制御／ショートメッセージ
Sound	サウンド

使用する機能を選択すると、機能を提供するAPIのヘッダーファイルが、生成されるソースコードにインクルードされます。なお、いつでも手動でインクルードすることができるので、ここで設定しなくてもかまいません。

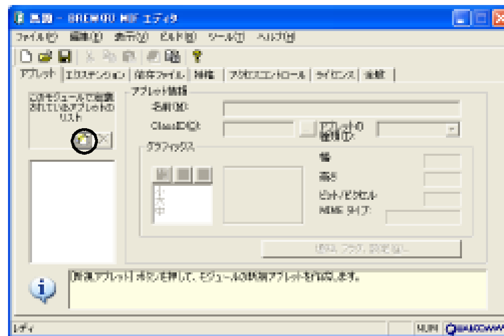
今回は、いずれの機能も選択しないで、そのまま次へボタンをクリックしましょう。

（６）「BREW Application Wizard - ｽﾃｯﾌﾟ２ / 2」が表示されます。



ここでは、MIFファイルを作成することができます。中央の「MIF Editor」ボタンをクリックしましょう。

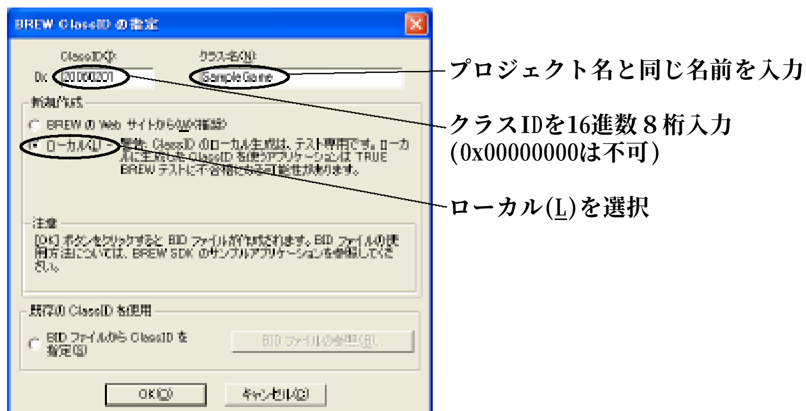
(７)MIFエディタが起動します(起動しない場合もあります。そのときは手動で起動してください)。



ここでは、MIFファイルを作成することができます。「新規ｱﾌﾟﾚｯﾄ」ボタンをクリックします。

(８)「ClassIDの生成」ダイアログが表示されます。

ここでは、クラスIDを作成することができます。「名前(N)」にプロジェクト名と同じ名前を入力し、「ローカル(L)」を選択します。「ClassID(I)」に適当な値を16進数で８桁入力し、「生成(G)」ボタンをクリックします。「BREW ClassID をローカルに生成してよろしいですか?」というメッセージが表示されるので、「はい」を選択します。



(９)bidファイルの保存ダイアログが表示されるので、(４)で作成したプロジェクトのフォルダに保存しましょう。

(10)MIFエディタに戻ります。ここでは、作成するアプリケーションの名前や種類、アイコンを設定します。

①ClassIDが設定されていることを確認します。設定されていない場合はBIDファイルの参照ボタンをクリックし、(９)で作成したbidファイルを選択します。

②「名前(M)」にアプリケーションの名前を入力します。プロジェクト名と同じでなくてもかまいません。ここで設定した名前が携帯電話のアプリ一覧に表示されます。ただし、日本語は正しく表示されません。

③「ｱﾌﾟﾚｯﾄの種類(T)」を設定します。

④アイコンを設定する場合は、「ｱｲｺﾝ(I)」にアイコンとなる画像のファイルを設定します。

※モジュールに複数のアプレットを含める場合は、(７)から(10)を繰り返します。

(11)MIFファイルを保存します。メニューの「ファイル(F)」→「名前を付けて保存(A)」を選択し、「プロジェクト名.mif」(BREW MIF v 2 ﾌｧｲﾙ)という名前で、(１)で作成したフォルダに保存しましょう。

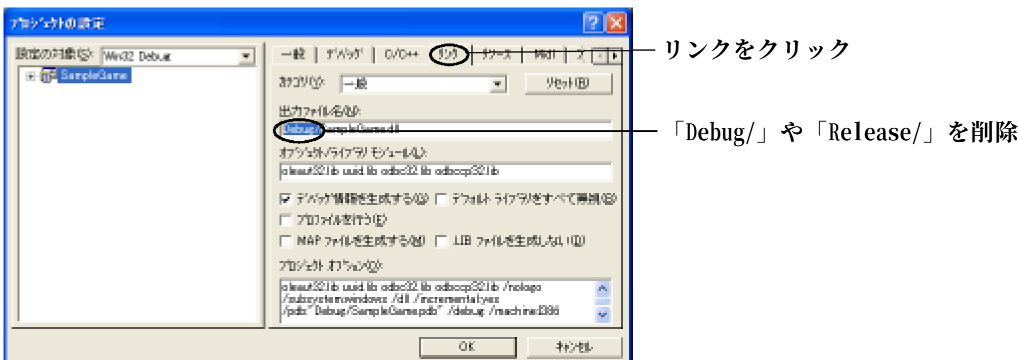
(12)MIFエディタを終了してVisual C++に戻り、「BREW Application Wizard - ｽﾃｯﾌﾟ2 / 2」ダイアログの終了ボタンをクリックしましょう。プロジェクトの情報が表示されるので、OKボタンをクリックすると、ソースファイルが自動生成されます。

(13)プロジェクトの設定を行います。

ビルドされるモジュールファイル(.dll)は、(４)で作成したプロジェクトのフォルダ直下になければ実行できません。Visual C++では、DebugフォルダやReleaseフォルダにファイルがビルドされるので、プロジェクトの設定を変更します。

メニューの「ﾌﾟﾛｼﾞｪｸﾄ(P)」→「設定(S)」(VC++2002以降はﾌﾟﾛｼﾞｪｸﾄ(P)→ﾌﾟﾛﾊﾟﾃｨ(P))を選択してプロジェクトの設定ダイアログを表示し、リンクタブをクリックすると、リンク時の設定が行えます。

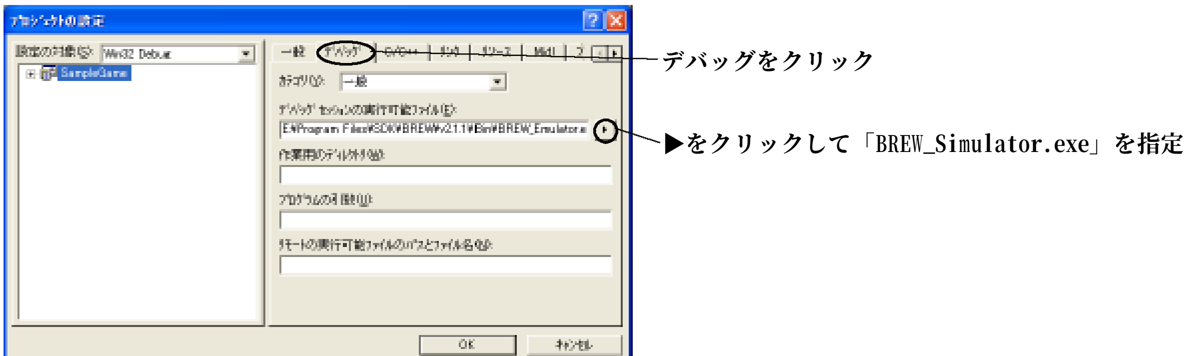
「出力ﾌｧｲﾙ名(N)」の「Debug/」や「Release/」を削除しましょう。



(14)実行ファイルの設定を行います。

BREWモジュールファイルなどの拡張子がDLLのファイルは、アプリケーションから呼び出される関数を集めたもので、そのままでは実行できません。Visual C++でも、DLLファイルを直接実行できないので、代わりにBREWシミュレータを起動させ、そこからDLLファイルを呼び出すようにします。

(13)のプロジェクトの設定ダイアログのデバッグタブをクリックすると、実行ファイルの設定が行えます。「ﾃﾞﾊﾞｯｸﾞ ｾｯｼｮﾝの実行可能ﾌｧｲﾙ(E)」の▶ボタン(VC++2002以降は構成―ﾃﾞﾊﾞｯｸﾞ―コマンドラインの参照)をクリックし、シミュレータの実行ファイル「BREW_Simulator.exe」を選択します。このファイルは、BREW SDKをインストールしたフォルダの「bin」フォルダにあります。



(15)プログラムを実行します。Shift+F5キーを押し、BREWシミュレータを起動しましょう。

BREWシミュレータの初回起動時は、古い機種で起動します。機種を変更するには、BREWシミュレータのメニューから「ﾌｧｲﾙ(F)」→「ﾃﾞﾊﾞｲｽの読み込み(L)」を選択し、「A5501T.qsc」を開きます。
機種が変更されたら、実行したいアプリケーションを左右で選んで選択(Enter)キーをクリックすると、プログラムが起動します。

(16)デバイスの設定を変更します。

シミュレータの画面が右下にずれている場合があります。これは、機種を定義する「BREWデバイスコンフィギュレータドキュメントファイル」(拡張子.qsc)の設定が正しくないためです。一部の機種のファイルでは、このように、画面やボタンの設定が若干ずれている場合があります。

デバイスの設定は、BREW SDK 2.1に付属の「BREWデバイスコンフィギュレータ」というツールで編集するか、テキストエディタで直接編集することにより、自由に変更することができます。

A5501T.qscをテキストエディタで開き、８８行目を以下のように変更しましょう。

誤：SCREEN 67 167 307 465 AVS_SCREEN_0 240 298 0.000000 0.000000 INCH 16 1
↓
正：SCREEN 66 163 306 461 AVS_SCREEN_0 240 298 0.000000 0.000000 INCH 16 1

BREWプロジェクトの内容

BREW Application Wizardでプロジェクトを作成すると、以下の３つのソースファイルが生成されます。

- AEEAppGen.c
- AEEModGen.c
- *プロジェクト名*.c

AEEAppGen.cとAEEModGen.cは、BREW SDKに付属しているソースファイルで、BREWアプリケーションやBREW実行環境の機能を呼び出すためのインタフェースの基本的な処理、つまりBREWアプリの基礎となるプログラムが記述されています。この２つのファイルは、BREW Application Wizardで生成した複数のBREWプロジェクトから共有されるので、絶対に編集してはいけません。

プログラマが編集するのは、*プロジェクト名*.cだけです。このファイルの名前は、プロジェクトの名前と同じになります。また、自動生成される関数や構造体の名前にも、プロジェクト名が付加されます。自動生成される関数は、AEEClsCreateInstance関数、*プロジェクト名*_HandleEvent関数、*プロジェクト名*_InitAppData関数、*プロジェクト名*_FreeAppData関数の４つです。

AEEClsCreateInstance関数は、アプリケーションの実行が開始された時点で呼び出されます。この関数では、グローバルな変数(アプレット構造体)を格納する領域を確保する処理と、*プロジェクト名*_InitAppData関数を呼び出すという２つの処理を行っています。この関数を編集する必要はなく、のこり３つの関数をアプリケーションに合うように編集します。

*プロジェクト名*_HandleEvent関数は、アプリケーションの実行中に起きたイベントを処理するためのイベントハンドラと呼ばれる関数です。BREWアプリケーションでは、この関数を中心に処理が進められます。
BREWプログラミングは、Windowsプログラミングと同様のスタイルが採用されています。アプリケーションは、実行中であっても休止しています。キー操作やアプリケーションの状態遷移といったイベントが発生すると、BREW実行環境がそれを検出し、該当するアプリケーションのイベントハンドラに通知します。このとき、*プロジェクト名*_HandleEvent関数が呼び出され、実行が再開されます。
*プロジェクト名*_HandleEvent関数では、通知されたイベントのうち、処理する必要があれば適切な処理を行い、不要なイベントであれば何もせずに実行権をシステムに戻します。携帯Javaプログラミングのように、自由にプログラムを記述するのではなく、発生したイベントに対してどのような処理を行うかを記述するのです。このようなスタイルをイベントドリブン型といいます。

プロジェクト名_InitAppData関数は、AEEClsCreateInstance関数から呼び出される関数です。この関数には、BREW実行環境の機能を呼び出すためのインタフェースの生成といった、システム関連の初期化処理を記述します。変数の初期化は、別のタイミングで行うので、この関数では行いません。

プロジェクト名_FreeAppData関数は、アプリケーションが終了するときに呼び出される関数です。この関数には、プロジェクト名_InitAppData関数の逆の処理、つまりインタフェースの解放といった、システム関連の解放処理を記述します。

メインループ

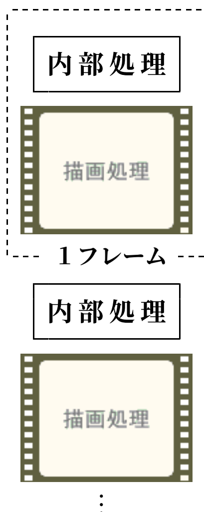
ゲームプログラムは、「ものの状態の集まり」と考えることができます。キャラクターの座標やパラメータ、背景、画面エフェクトなどすべて、それぞれ「もの」の状態です。ゲームプログラムとは、このいろいろな「もの」の状態を時間に沿って更新し、それを画面に描画するプログラムと考えることができます。
つまり、ゲームプログラムの処理を非常に単純化すると、

- 「もの」の状態を更新(内部処理)
- 「もの」を画面に描画(描画処理)

という２つにまとめることができます。
この２つの処理の繰り返しがメインループになります。このループ１回ぶんを「１フレーム」と呼びます。１フレームごとに、キャラクタの位置や状態、背景やそのほかの表示状態などを更新し、その状態を反映して描画を行う、という処理を繰り返していきます。

フレームレート

ゲームプログラムでは、メインループ(正確には画面の更新)を一定の間隔で実行します。実行速度は環境によって異なるので、速い環境ほど高速に動作できます。
１秒間に何回画面を更新するのかをフレームレート(FPS:Frame Per Second)と呼びます。60FPSの場合、１秒間に60回画面を更新しているという意味になり、１フレームを約16.67ミリ秒の間隔で表示しています。一般的な家庭用ゲームでは60FPSです。BREW対応携帯電話では、30FPS以上で処理する能力を持ちます。



タイマ

BREWでは、携帯Javaプログラミングと違い、メインループをループで構成することができません。これは、イベントが起きたときに必要な処理だけを行い、その後直ちに実行権をシステムに戻さなくてはならないという、イベントドリブン型のスタイルを採用しているためです(BREWでは、ひとつのイベント処理に多大な時間を費やすと、アプリケーションが強制終了させられます)。

メインループは、一定の間隔で実行されれば良いので、ループでなくても実現できます。たとえば、タイマという機能を用いる方法です。タイマは、一定の間隔で特定の動作を行うという機能です。

BREWのタイマは、「指定した時間」の経過後に、「指定した関数」を１度だけ実行させることができます。タイマの「指定した時間」をフレームの間隔に、「指定した関数」を内部処理および描画処理を行う関数に設定します。ゲームが終わるまでタイマを起動し続ければ、ループで構成するのとなんら変わりのない動作が得られます。

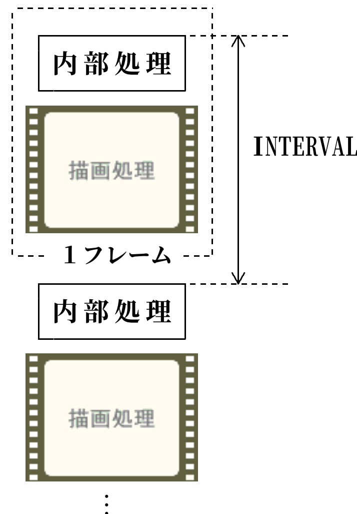
課　題

メインループを作成しましょう。

(１)プロジェクト名.cに、フレームレートを制御するための定数を定義します。定数FPSおよびINTERVALを「#include」群の下に追加しましょう。

```
/*===========================================================================
INCLUDES AND VARIABLE DEFINITIONS
=========================================================================== */
#include "AEEModGen.h"          // Module interface definitions
#include "AEEAppGen.h"          // Applet interface definitions
#include "AEEShell.h"           // Shell  interface definitions

#include "プロジェクト名.bid"

/*===========================================================================
CONSTANT NUMBERS
=========================================================================== */
#define  FPS       30
#define  INTERVAL  1000 / FPS
```

定数FPSが１秒あたりのフレーム数、定数INTERVALが現フレームの先頭から次フレームの先頭までの間隔です。



(２)メインループを記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*=======================================================================
FUNCTION プロジェクト名_MainLoop

DESCRIPTION
    This is the MainLoop for this Game.

PROTOTYPE:
    static void プロジェクト名_MainLoop(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none
```

```
RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
static void プロジェクト名_MainLoop(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(3)内部処理を記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*===========================================================================
FUNCTION プロジェクト名_Proc

DESCRIPTION
    This is the InternalProcessing for this Game. All game processing to this app are handled in this
    function.

PROTOTYPE:
    void プロジェクト名_Proc(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none

RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
void プロジェクト名_Proc(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(4)描画処理を記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*===========================================================================
FUNCTION プロジェクト名_Draw

DESCRIPTION
    This is the DrawingProcessing for this Game. All game processing to this app are handled in this
    function.

PROTOTYPE:
    void プロジェクト名_Draw(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none

RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
void プロジェクト名_Draw(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(５)タイマ起動を記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*===========================================================================
FUNCTION プロジェクト名_SetTimer

DESCRIPTION
    This is the SetTimer for this Game. The timer to move this game at regular intervals is started.

PROTOTYPE:
    void プロジェクト名_SetTimer(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none

RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
void プロジェクト名_SetTimer(プロジェクト名* pMe)
{
}
```

※「*プロジェクト名*」は作成するプロジェクト名(半角英数字)に置き換えてください

(６)(２)～(５)で作成した関数のプロトタイプを宣言します。以下のプログラムをコメント「Function Prototypes」の下に追加しましょう。

```
/*-------------------------------------------------------------------
Function Prototypes
-------------------------------------------------------------------*/
static  boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                          uint16 wParam, uint32 dwParam);
boolean プロジェクト名_InitAppData(プロジェクト名* pMe);
void    プロジェクト名_FreeAppData(プロジェクト名* pMe);

static void プロジェクト名_MainLoop(プロジェクト名* pMe);

void プロジェクト名_Proc(プロジェクト名* pMe);
void プロジェクト名_Draw(プロジェクト名* pMe);
void プロジェクト名_SetTimer(プロジェクト名* pMe);
```

※「*プロジェクト名*」は作成するプロジェクト名(半角英数字)に置き換えてください

(７)タイマを起動する処理を記述します。

タイマは、IShellインタフェースのISHELL_SetTimer関数で設定します。この関数を呼び出すと、タイマが設定されます。現在の時間から指定した時間(ミリ秒)経過後にタイマが作動し、指定した関数が実行されます。BREWのタイマは、１回しか作動しないので、タイマを繰り返すには、再度関数を呼び出し、再設定する必要があります。

以下のプログラムを適切な場所に追加しましょう。

```
ISHELL_SetTimer(pMe->pIShell, INTERVAL, プロジェクト名_MainLoop, pMe);
```

ISHELL_SetTimer関数の引数は、先頭から「①IShellインタフェース」(※インタフェースは１７ページで説明します)、「②タイマ作動までの時間」、「③時間経過後に実行する関数」、「④引数③の関数に渡す変数のポインタ」です。上記の場合、定数INTERVAL経過後に、*プロジェクト名*_MainLoop関数が１回だけ実行されます。

(８)メインループを作成します。

　メインループでは、以下の３つの処理を行います。
　　１．タイマの起動
　　２．内部処理
　　３．描画処理

　上記１～３がプロジェクト名_MainLoop関数で実行されるように、適切なプログラムを追加しましょう。

ヒント１：プログラムを追加するのは、プロジェクト名_MainLoop関数
ヒント２：(５)、(３)、(４)
ヒント３：３つの関数を呼び出しますが、それらの関数の引数は１つで、すべて「pMe」

イベント

　BREW実行環境は、実行中のアプリケーションを監視しています。キーが押されたり、アプリケーションが中断されたりといったイベントが発生すると、該当するアプリケーションにイベントが起きたことを通知します。各アプリケーションは、送出されてきたイベントを正しく処理しないと、強制終了させられてしまうことがあります。

イベント	発　生　条　件
EVT_APP_START	アプリケーションが起動されたとき(AEEClsCreateInstance関数の実行後)
EVT_APP_STOP	アプリケーションが終了を要求したとき
EVT_APP_SUSPEND	アプリケーションの実行が一時中断されようとしているとき
EVT_APP_RESUME	中断されたアプリケーションが再開されようとしているとき
EVT_KEY_PRESS	キーが押されたとき
EVT_KEY_RELEASE	キーが離されたとき

▲おもなイベントと発生条件

イベントハンドラ

　BREWアプリケーションは、イベントハンドラと呼ばれる特別な関数を準備しなければなりません。この関数は、BREW実行環境から送出されてくるイベントを処理するための関数です。プロジェクト名_HandleEvent関数が該当します。

　プロジェクト名_HandleEvent関数は、以下のように送出されてきたイベントをswitch文で分岐させ、それぞれの処理を行います。イベントは十数種類ありますが、すべてのイベントを処理する必要はありません。必要のないイベントの場合は、FALSEを返してイベントハンドラから抜けます。イベントを処理した場合は、決められた戻り値を返します。

```
// イベントハンドラ(イベントを処理する関数)
static boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                         uint16 wParam, uint32 dwParam)
{
    switch(eCode) {
    case EVT_APP_START:
        ※アプリケーションが起動されたときの処理(変数の初期化など)
        return TRUE;

    case EVT_APP_STOP:
        ※アプリケーションを終了するときの処理
        return TRUE;
```

```
        case EVT_KEY_PRESS:
            ※キーが押されたときの処理
            return TRUE;

        default:
            break;
        }

        return FALSE;
    }
```

イベントハンドラは、アプリケーションが何も処理していないときだけ実行されます。イベントの処理中だったり、タイマで指定された関数を実行中のときは、発生したイベントは処理待ち状態になります。処理待ち状態のイベントが一定時間処理されなかったり、一定数を超えた場合、BREW実行環境は、そのアプリケーションを強制終了させてしまいます。したがって、イベントやタイマ起動時の処理は、あまり長時間行うことはできません。目安としては、１秒程度といわれています。

Handle_Event関数

イベントハンドラ「*プロジェクト名*_HandleEvent関数」のプロトタイプは、以下のようになっています。

```
static boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                          uint16 wParam, uint32 dwParam);
```

最初のstaticは、ほかのソースファイルと関数名が重複してもエラーにならないようにあるだけで、動作が変わるなどの意味はありません。

イベントハンドラが呼び出されると、BREW実行環境から４つの引数が渡されます。それぞれ、以下のような情報が格納されています。

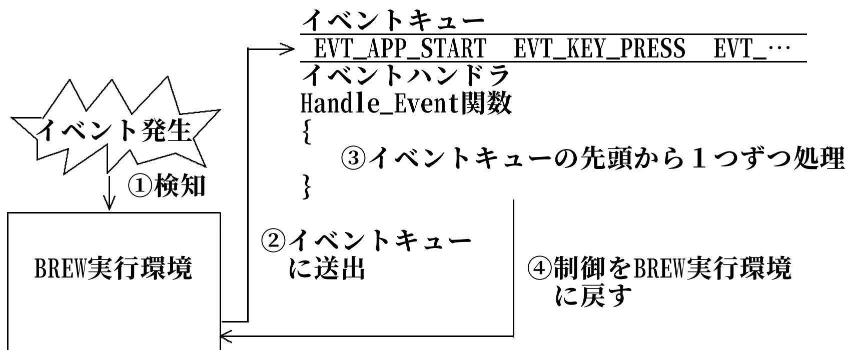
*プロジェクト名**	pMe………アプレット構造体(へのポインタ)
AEEEvent	eCode……送出されたイベントの種類
uint16	wParam …イベントの追加情報。内容はイベントによって異なります
uint32	dwParam…イベントの追加情報。内容はイベントによって異なります

BREWプログラミングは、イベントに応答しながら処理を行うことからイベントドリブン(イベント駆動型)プログラミングと呼ばれています。アプリケーションが独自に動作するのではなく、BREW実行環境から取得したイベントをトリガ(きっかけ)として動作するためです。

このような複雑な構造になっているのは、BREW実行環境が複数のアプリケーションを最適にスケジュールし、安定して協調動作させるためです(Javaは、イベントを処理するプログラムとメイン処理のプログラムを同時に実行できるマルチスレッド型なので、このような制限はありません)。

旧来のMS-DOS(コンソール)などのアプリケーションでは、main関数からプログラムの実行が始まり、プログラマが記述した流れにそってプログラムが実行されます。プログラムの中で文字を表示する必要があれば、そこでprintf関数などを実行することによって文字出力が行われます。MS-DOSアプリケーションでは、プログラマが記述したコードのとおりに処理が進んでいきます。
ところがBREWアプリケーションでは、プログラマが書くコードは、BREW実行環境から必要なときに呼び出されるようになっています。BREWプログラミングの世界では、プログラマはプログラムの流れを取り仕切るのではなく、イベントを受け取ったときやタイマが起動したときの動作という局部的なコードを記述するのです。「プログラムの流れはこうしよう」という考え方ではなく、「このイベントが発生したらこうしよう」という考え方になっています。

このようなシステムからのメッセージに応答するようなプログラム構造は、キーボードやマウスからの入力がない限り処理を行わないワープロやメーラーといったビジネスアプリケーションと非常に相性が良いのですが、常にリアルタイムで動作しなければならないゲームプログラムと相性が良いとはいえません。



▲BREWアプリケーションの基本プログラミングモデル

課　題

変数やデータの初期化および解放を行う関数と、アプリケーションが休止および復帰状態になったときの関数を作成しましょう。

(１)変数やデータの初期化を行う関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
// ゲームデータ初期化
boolean プロジェクト名_InitGameData(プロジェクト名* pMe)
{
    return TRUE;
}
```

※「*プロジェクト名*」は、作成したプロジェクトの名前にそって置き換えてください

(２)(１)のプロトタイプを宣言します。以下のプログラムを適切な場所に追加しましょう。

```
boolean プロジェクト名_InitGameData(プロジェクト名* pMe);
```

(３)変数やデータの解放を行う関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
// ゲームデータ解放
void プロジェクト名_FreeGameData(プロジェクト名* pMe)
{
}
```

(４)(３)のプロトタイプを適切な場所に作成しましょう。

(５)アプリケーションの起動時に、変数やデータを初期化します。

アプリケーションが使用する変数やデータは、未初期化バグを防ぐため、アプリケーションが起動したときに0やNULLなどで初期化しておきます。
アプリケーションが起動し、AEEClsCreateInstance関数から抜けると、EVT_APP_STARTイベントが発生します。変数やデータは、このイベントで初期化します。

EVT_APP_STARTイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション起動
static boolean プロジェクト名_OnAppStart(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    if(※ここは各自考えましょう(pMe) == FALSE)  // 初期化
        return FALSE;

    プロジェクト名_MainLoop(pMe);              // メインループ
    return TRUE;
}
```

(６)(５)のプロトタイプを宣言します。以下のプログラムを適切な場所に追加しましょう。

```
static boolean プロジェクト名_OnAppStart(プロジェクト名* pMe, uint16 wParam, uint32 dwParam);
```

(７)アプリケーションの終了時に、変数やデータを解放します。

アプリケーションが使用した変数やデータは、アプリケーション終了時に自動的に解放されますが、独自にメモリを確保したり、画像などを読み込んだ場合は、プログラマが解放しなければなりません。
アプリケーションが終了するときは、プロジェクト名_FreeAppData関数が呼び出される前に、EVT_APP_STOPイベントが発生します。メモリや画像などの解放は、このイベントで行います。

EVT_APP_STOPイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション終了
static boolean プロジェクト名_OnAppStop(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    ※ここは各自考えましょう;
    return TRUE;
}
```

(８)(７)のプロトタイプを適切な場所に作成しましょう。

(９)イベントハンドラでEVT_APP_STARTイベントとEVT_APP_STOPイベントが処理されるようにします。
プロジェクト名_HandleEvent関数を以下のように変更しましょう。

```
static boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                              uint16 wParam, uint32 dwParam)
{
    switch (eCode) {
      case EVT_APP_START:   return プロジェクト名_OnAppStart(pMe, wParam, dwParam);
      case EVT_APP_STOP:    return プロジェクト名_OnAppStop (pMe, wParam, dwParam);
   }

   return FALSE;
}
```

(10)アプリケーションの一時中断とその復帰に対応させます。

実行中のBREWアプリケーションは、電話がかかってきたり、メールを着信したりすると、実行が一時中断される場合があります。一時中断された状態では、ゲームを進めることができないので、タイマを停止し、ゲームが進行しないようにします。

アプリケーションが一時中断されたり中断から復帰した場合は、イベントよって通知されます。アプリケーションは、これらのイベントを利用することにより、一時中断とその復帰に対応させることができます。

アプリケーションが一時中断されたときのイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション中断
static boolean プロジェクト名_OnAppSuspend(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    // タイマキャンセル
    ISHELL_???????????(pMe->pIShell, プロジェクト名_MainLoop, pMe);
    return TRUE;
}
```

(11)(10)のプロトタイプを適切な場所に作成しましょう。

(12)アプリケーションが中断から復帰したときのイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション再開
static boolean プロジェクト名_OnAppResume(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    // タイマ設定
    ISHELL_????????(pMe->pIShell, 0, プロジェクト名_MainLoop, pMe);
    return TRUE;
}
```

(13)(12)のプロトタイプを適切な場所に作成しましょう。

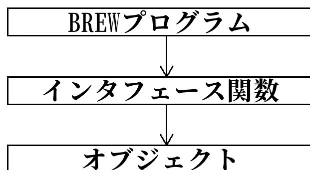
(14)イベントハンドラでアプリケーションの一時中断とその復帰イベントが処理されるようにします。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
    case EVT_???_???????: return プロジェクト名_OnAppSuspend(pMe, wParam, dwParam);
    case EVT_???_??????:  return プロジェクト名_OnAppResume (pMe, wParam, dwParam);
```

インタフェース

BREW APIは、BREW実行環境の機能を呼び出すための関数群ですが、その実体は、「オブジェクト」に作用する「インタフェース関数」です。
オブジェクトやインタフェースといった用語は、オブジェクト指向で用いるものですが、BREWのそれらは、若干意味が異なります。

BREWのオブジェクトとは、BREW APIが内部で使うデータのことをいいます。そして、そのデータを操作するのがインタフェース関数です。インタフェース関数の１つめの引数は、必ず操作対象となるオブジェクトを渡すようになっています。BREWでは、オブジェクトとインタフェース関数をまとめたものをインタフェースと呼びます。



たとえば、前回の課題で使用したISHELL_SetTimer関数は、IShellインタフェースのSetTimerという機能を呼び出しています。
IShellインタフェースは、BREW実行環境を操作するためのインタフェースです。このインタフェースには、BREW実行環境を管理するためのデータと、その機能を呼び出すための多数のインタフェース関数が備えられています。ISHELL_SetTimer関数は、１度だけ起動するタイマを設定するという動作をとります。つまり、この関数はBREW実行環境のタイマを設定している、ということになります。

```
ISHELL_SetTimer(pMe->pIShell, 引数…);
```

ISHELL_SetTimer：インタフェース関数　pMe->pIShell：オブジェクト

IDisplayインタフェースとIGraphicsインタフェース

画面に描画を行うためのインタフェースとしてIDisplayインタフェースとIGraphicsインタフェースが用意されています。２つのインタフェースは、以下のように機能が異なります。

ｲﾝﾀﾌｪｰｽ	画面のクリア	ﾌﾚｰﾑの更新	ﾋﾞｯﾄﾏｯﾌﾟの転送	塗りつぶし	文字描画	点の描画	線の描画	矩形描画	三角形の描画	多角形の描画	円の描画	色の反転	ﾊﾞｯｸﾗｲﾄの制御
IDisplay	◎	◎	○(等倍)	○	○	×	△	○	×	×	×	○	○
IGraphics	○	○	○(拡縮)	○	×	○	○	◎	○	○	○	×	×

また、同じような動作をするにもかかわらず、関数名や性能が異なったり、制限のある場合もあるので、ドキュメントやシミュレータ(実機)で動作を確認してからどちらを使うか決定する必要があります。

インタフェースの生成と解放

IShellインタフェースとIDisplayインタフェースは、どのようなアプリケーションでもほぼ確実に使用するため、アプリケーションの起動時に、AEEClsCreateInstance関数で自動的に生成されます。
そのほかのインタフェースは、IShellインタフェースのISHELL_CreateInstance関数で生成する必要があります。引数にインタフェースのクラスIDを渡すと、そのインタフェースがメモリに生成され、オブジェクトが返されます。たとえば、IGraphicsインタフェースを生成するには、クラスID「AEECLSID_GRAPHICS」を渡します。

```
// IGrapicsインタフェースの生成
IGraphics  *pIGraphics;
ISHELL_CreateInstance(pMe->pIShell, AEECLSID_GRAPHICS, &pIGraphics);
```

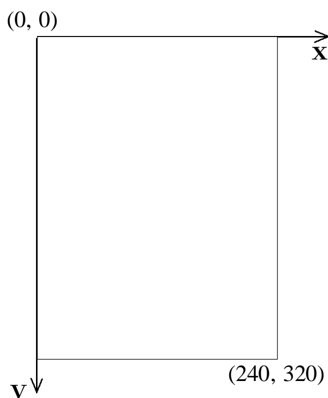
関数が成功すると、メモリにBREWの描画機能にアクセスするためのIGraphicsインタフェースが生成され、そのオブジェクトがpIGraphics変数に格納されます。以後、IGRAPHICS_で始まるインタフェース関数を呼び出せば、描画に関する機能を使用できます。

ISHELL_CreateInstance関数で生成したインタフェースが不用になったら、そのインタフェース関数のRelease関数を呼び出し、メモリから解放しなければなりません。

```
// IGraphicsインタフェースの解放
IGRAPHICS_Release(pIGraphics);
```

スクリーン座標

2Dでグラフィックを描画する場合、座標の指定はスクリーン座標で行います。スクリーン座標とは、画面のピクセルに１対１に対応する座標系のことです。スクリーン座標の原点は、左上にあります。また、ｙ軸の向きが逆になっていることも大きな特徴です。



QVGA対応端末では、２４０×３２０ドットの解像度を持ちますが、上部にバッテリー残量やアンテナ感度を表示するため、実際には、A5501Tで２４０×２９８ドット、A5503SAで２４０×２９１ドットとなります。

グローバル変数の定義

BREWでは、システム構造の制限により、グローバル変数およびstaticな変数は、使用することができません(ただし、シミュレータでは使用できます)。しかも、イベントハンドラ(プロジェクト名_HandleEvent関数)などのように、関数の引数があらかじめ決められていることもあり、必要なデータを関数に渡すことができない場合があります。このような場合、どこからでも参照できるグローバル変数なしでは、あらゆる場面で参照しなければならない変数にアクセスできないということになります。

BREWでは、このような場合、その変数を*プロジェクト名*.cで定義されている「*プロジェクト名*」という構造体(これをアプレット構造体といいます)のメンバ変数として定義します。
アプレット構造体は、AEEClsCreateInstance関数によってアプリケーションの起動時に生成され、この関数を除く、自動生成された３つの関数の最初の引数になっています。BREWでは、*プロジェクト名*_HandleEvent関数を中心に処理が進められるので、この関数からほかの関数にアプレット構造体を渡すことにより、あらゆる関数で必要となるデータを参照することができます。

課　題

グラフィックを描画するための準備をしましょう。

(１)グラフィックは、IDisplayインタフェースまたはIGraphicsインタフェースで描画します。IGraphicsインタフェースを用いるには、専用のヘッダファイルが必要になります。以下のプログラムを適切な場所に追加しましょう。

```
#include "AEEGraphics.h"    // Graphics interface definitions
```

(２)IGraphicsインタフェースを格納する変数をアプレット構造体に追加します。以下のプログラムを適切な場所に追加しましょう。

```
IGraphics   *pIGraphics; // give a standard way to access the Graphics interface
```

ヒント：// add your own variables here...

(３)プログラムの開始時に、IGraphicsインタフェースを取得します。以下のプログラムを適切な場所に追加しましょう。なお、「？」は各自考えてください。

```
// IGraphicsインタフェース生成
ISHELL_??????????????(pMe->pIShell, AEECLSID_????????, &pMe->pIGraphics);
if(pMe->pIGraphics == NULL)
    return FALSE;
```

IGraphicsインタフェースを生成し、そのオブジェクトをアプレット構造体のpIGraphicsメンバに格納します。以後、「pMe->pIGraphics」と記述することでIGraphicsオブジェクトを得ることができます。

ヒント：追加する場所は、１４ページのどこかに記述されています

(４)プログラムの終了時に、IGraphicsインタフェースを解放します。以下のプログラムを適切な場所に追加しましょう。なお、「？」は各自考えてください。

```
// IGraphicsインタフェース解放
if(pMe->pIGraphics != NULL) {
    IGRAPHICS_???????(pMe->pIGraphics);
    pMe->pIGraphics = NULL;
}
```

ヒント：// insert your code here for freeing any resources you have allocated...

(５)画面を更新する処理を追加します。以下のプログラムをプロジェクト名_Draw関数の最後に追加しましょう。

```
// 画面更新
IDISPLAY_Update(pMe->pIDisplay);
```

IDisplayインタフェースまたはIGraphicsインタフェースのUpdate関数を実行すると、前回のUpdate関数呼び出し以後に行ったすべての描画命令の結果が画面に表示されます。よって、この関数は、現フレームのすべての描画が終わったあとに呼び出します。

画像の読み込み

BREWでは、画像はファイルまたはリソースから読み込むことができます。対応している形式は、BMP、PNG、JPEGです。ただし、BMPを１とすると、PNGは４、JPEGは１００の時間が掛かります。また、透過色をサポートしているインタフェースはIBITMAPだけなので、キャラクターなどで透明色を扱うには、BMP形式しか選択肢がありません。なお、BREWで推奨されているBMPの形式は、圧縮なしの色深度１，２，４，８ビットのものです(大きさには特に制限がなく、容量の限り読み込めます)。

ファイルは、BREWがサポートするファイルシステムのことです。携帯Javaと違い、BREWにはPCのようなファイルシステムが存在します。このファイルシステムは、ファイル名の最大長が64文字であるという制限を除けば、ディレクトリも作成できます。ファイルを扱うインタフェースにはIFileMgrやIFileなどがあります。

リソースは、アプリで使う表示文字列や画像など、実行コード以外の資源をひとつにまとめたものです。リソースは、ひとつのアプリにひとつである必要はなく、リソースを持たないアプリや２つ以上のリソースを使用するアプリを作成することもできます。BREWのリソースは、拡張子が.barで、BREW SDKに含まれるBREWリソースエディタで作成することができます。

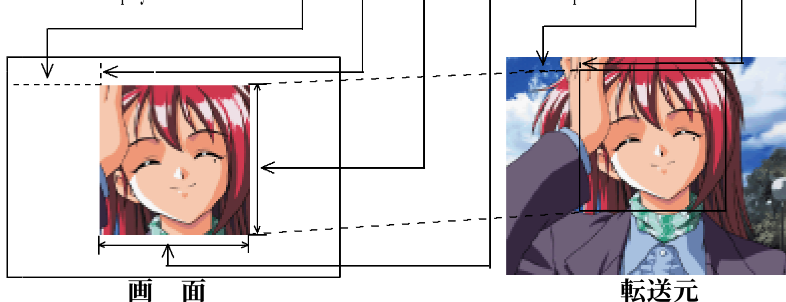
リソースから画像を読み込む関数には、ISHELL_LoadResBitmap関数とISHELL_LoadResImage関数があります。ISHELL_LoadResBitmap関数は、リソースからBMP形式の画像を読み込み、それを制御するためのIBitmapインタフェースを返します。ISHELL_LoadResImage関数は、リソースから画像を読み込み、それを制御するためのIImageインタフェースを返します。
IBitmapインタフェースはBMP形式しか扱えませんが、IImageはすべての形式を扱うことができます。ただし、IBitmapインタフェースはビットマップの管理と描画に、IImageインタフェースはアニメーションに重点を置いており、２つのインタフェースがサポートする関数は、かなりの違いがあります。

```
IBitmap    *pBmpChara;         // ビットマップを格納するインタフェース
IImage     *pImgAniChara;      // イメージを格納するインタフェース

// リソースからビットマップを読み込む
pBmpChara = ISHELL_LoadResBitmap(pMe->pIShell, "resource.bar", IDB_CHARA);
                                 IShellオブジェクト  リソース      リソースID

// リソースから画像を読み込む
pImgAniChara = ISHELL_LoadResImage(pMe->pIShell, "resource.bar", IDI_ANICHARA);
```

画像の描画

ビットマップを画面に表示する関数には、IDISPLAY_BitBlt関数があります。

```
// ビットマップの描画
IDISPLAY_BitBlt(pMe->pIDisplay, 160, 50, 150, 160, pBmpChara, 82, 9, AEE_RO_COPY);
                IDisplayオブジェクト                IBitmapオブジェクト  ラスタオペレーションコード
```



最後の引数は、「ラスタオペレーションコード」と呼ばれるもので、転送元と転送先のビットマップデータをどのように結合するかという指定です。通常はAEE_RO_COPYを使用して単純に転送元から転送先にコピーするだけですが、画像を反転、くり抜き、合成といった効果を付加することができます。

ラスタオペレーションコード	演　　算	意　　味
AEE_RO_COPY	転送先 = 転送元	転送元を上書き
AEE_RO_NOT	転送先 = ~転送元	転送元のビット列を反転して上書き
AEE_RO_OR	転送先 = 転送先 | 転送元	転送先と転送元のビット列のORをとる
AEE_RO_XOR	転送先 = 転送先 ^ 転送元	転送先と転送元のビット列のXORをとる
AEE_RO_MERGENOT	転送先 = ~転送元 | 転送先	転送元のビット列を反転し、転送先とORをとる
AEE_RO_ANDNOT	転送先 = ~転送元 & 転送先	転送元のビット列を反転し、転送先とANDをとる
AEE_RO_TRANSPARENT		転送元の指定された色が転送されなくなる

▲ラスタオペレーションコード

IBitmapインタフェースは、ビットマップ間で転送を行うIBITMAP_BltIn関数とIBITMAP_BltOut関数をサポートしているほか、IBITMAP_QueryInterface関数でIDIBインタフェースを取得することによりピクセルに直接アクセスすることもできるので、ビットマップに様々な加工を行うことができます。IImageインタフェースには、このような機能はありません。

IImageインタフェースでは、IIMAGE_Draw関数またはIIMAGE_DrawFrame関数で画面に描画します。IImageインタフェースは、アニメーション機能に重点を置いているので、IBitmapインタフェースに比べ、用意されている関数はかなり異なります。

透過色

IBitmapインタフェースは、透過色をサポートしています。透過色を適用するには、はIBITMAP_SetTransparencyColor関数で透明にしたい色(転送したくない色)を指定し、ビットマップの描画および転送時にラスタオペレーションコード「AEE_RO_TRANSPARENT」を指定すれば、設定した色が転送されなくなります。ただし、IBITMAP_RGBToNative関数に渡す色は、デバイスに読み込まれた後の、それぞれのデバイスに合わせた色でなければならないので、MAKE_RGBマクロではなく、IBITMAP_RGBToNative関数で作成します。

```
NativeColor    color;

// 透過色設定(何度も行う必要はありません)
color = IBITMAP_RGBToNative (pBmpChara, MAKE_RGB(0, 0, 0));
IBITMAP_SetTransparencyColor(pBmpChara, color);

// 透過色を用いた転送
IDISPLAY_BitBlt(pMe->pIDisplay, 0, 0, 60, 60, pBmpChara, 0, 0, AEE_RO_TRANSPARENT);
```

画像の解放

IBitmapインタフェースに読み込まれているビットマップが不用になったら、IBITMAP_Release関数で解放します。同じように、IImageインタフェースに読み込まれている画像が不用になったら、IImage_Release関数で解放します。
解放されたインタフェースは、再び生成または初期化するまで使用できないので、インタフェースを格納していた変数にはNULLを代入し、無効であることを明示するようにします。こうしておくと、解放したインタフェースを初期化しないで不正使用したことによる、不安定な動作を起こすという深刻なバグを防ぐことができます。

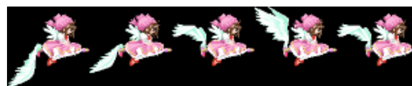
```
// 画像の解放
if(pBmpChara != NULL) {          // ポインタチェック。NULLの場合は解放しない(できない)
    IBITMAP_Release(pBmpChara);  // インタフェースの解放
    pBmpChara = NULL;            // NULLを代入し、無効なインタフェースであることを示す
}
```

課　題

画像を読み込み、描画しましょう。

(１)以下のような背景とプレイヤーの画像を作成し、それぞれ256色以下のビットマップ形式で保存しましょう。

背景(BG.bmp:240×298)

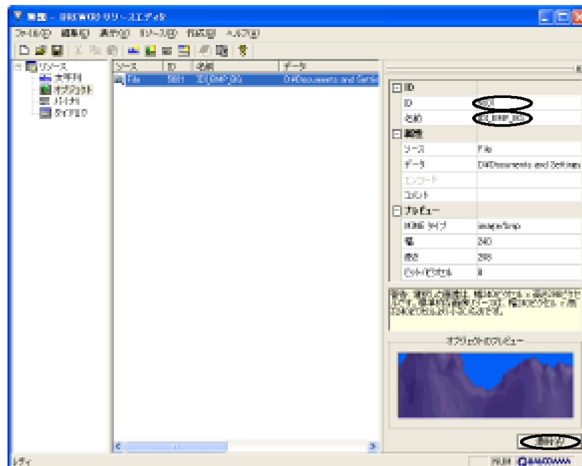


プレイヤー(Player.bmp)

(２)リソースを作成します。BREWリソースエディタを起動しましょう。

(３)リソースにビットマップを登録します。メニューから「リソース(R)→新規オブジェクト(O)」を選択しましょう。リソースに、画像を登録する領域(File 5001 IDI_OBJECT_5001)が作成されます。

(４)画像を登録します。「File 5001 IDI_OBJECT_5001」という文字列をダブルクリックしましょう。画像ファイルを選択するダイアログが表示されるので、背景となるビットマップファイルを選択します。



選択した画像によっては、警告になる場合があります。画像サイズ以外の警告は、画像の形式がBREW準拠ではないということになり、実行時に画像が正しく扱われるかどうかは、機種依存となります。

(５)画像のプロパティを設定します。

IDには数字を、名前には英数字と'_'を、ほかのリソースと重複しないように入力します(ただし、リソースファイルが異なる場合は、重複してもかまいません)。IDに「5001」、名前に「IDI_BMP_BG」と入力し、下の適用ボタンをクリックしましょう。

これらは、リソースのヘッダファイル(.brh)で以下のように定義され、ISHELL_LoadResBitmap関数とISHELL_LoadResImage関数で必要になります。

```
#define IDI_BMP_BG      5001
```

(６)(３)から(５)を参考に、キャラクターとなる画像をリソースに登録しましょう。

(７)brxファイルを保存します。メニューから「ﾌｧｲﾙ(F)→保存(S)」を選択し、プロジェクトのフォルダに保存しましょう(名前は適当でかまいませんが、プロジェクトと同じ名前が無難です。また、(８)で生成されるbarファイルは、大文字のファイル名だと読み込みエラーになるので、英字は小文字で指定しましょう)。

※brxファイルは、リソースエディタが読み込み可能なリソースのファイルです。このファイルをリソースエディタで読み込めば、リソースを編集することができます。

(８)リソースのヘッダファイルとbarファイルを作成します。メニューから「作成(B)→リソーススクリプトのコンパイル(C)」を選択しましょう。選択後、(７)で指定したフォルダにリソースのヘッダファイル(*brx*のファイル名.brh)とリソースファイル(*brx*のファイル名.bar)が作成されます。

(９)ビットマップを扱うために必要なヘッダファイルをインクルードします。以下のプログラムを適切な場所に追加しましょう。

```
#include "AEEBitmap.h"          // Bitmap interface definitions
#include "AEEStdLib.h"

#include "brxのファイル名.brh"
```

(10)アプレット構造体で、ビットマップを制御するための変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
IBitmap   *pBmpBG;
```

(11)ビットマップを読み込みます。以下のプログラムを完成させ、*アプリケーション名*_InitGameData関数の適切な場所に追加しましょう。

```
// 背景読み込み
pMe->pBmpBG = ISHELL_?????????????(pMe->???????, "brxのファイル名.bar", IDI_BMP_BG);
if(pMe->pBmpBG == NULL)
    return FALSE;
```

※画像は、ゲームで必要となるデータなので、InitGameData関数で読み込みます。

(12)ゲーム終了時に、ビットマップが解放されるようにします。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
// 背景解放
if(pMe->pBmpBG != NULL) {
    IBITMAP_???????(pMe->pBmpBG);
    pMe->pBmpBG = ????;
}
```

※以下のようなマクロを定義しておくと、解放処理を１行にまとめることができます。

```
#define BITMAP_RELEASE(x) {if(x!=NULL) {IBITMAP_Release(x); x=NULL;}}
```

(13)背景を描画してみましょう。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
// 背景描画
IDISPLAY_??????(pMe->pIDisplay, 0, 0, 240, 298, pMe->pBmpBG, 0, 0, AEE_RO_????);
```

(14)(11)～(14)を参考に、プレイヤーを読み込み、１コマ目だけ描画しましょう。

(15)プレイヤーに透過色を設定しましょう。

課　題

プレイヤーをアニメーションさせましょう。

(１)「何パターン目の画像を描画するのか」を保存する変数plyAniを宣言します。以下の変数をアプレット構造体の適切な場所で宣言しましょう。

```
int   plyAni;     // プレイヤーアニメーションカウンタ
```

(２)内部処理で描画するパターンを更新します。

次に描画するパターンは、変数plyAniで管理します。変数plyAniを０から「最大コマ数－１」まで１ずつ増やしていき、これを越えたら０に戻します。たとえば、アニメーションパターンが５パターンある場合は、変数plyAniは０，１，２，３，４となり、５になったら０に戻します。こうすることにより、プレイヤーのアニメーションを循環させます。

以下のプログラムを内部処理の適切な場所に追加しましょう。ただし、斜体の部分は、画像のパターン数に合わせて書き換えてください。

```
// プレイヤーアニメーション処理
pMe->plyAni++;
if(pMe->plyAni >= 5)
    pMe->plyAni = 0;
```

※上記のプログラムは、%演算子を用いて以下のように書き換えることができます。

```
// プレイヤーアニメーション処理
pMe->plyAni = (pMe->plyAni + 1) % 5;
```

(３)プレイヤーを描画する部分をアニメーション対応にします。プレイヤー描画を以下のように変更しましょう。なお、「？」の部分は各自考えてください。また、斜体の部分も画像の幅と高さに合わせて書き換えてください。

```
// プレイヤー描画
IDISPLAY_BitBlt(pMe->pIDisplay, 0, 0, 60, 60,
                pMe->pBmpChara, pMe->?????? * 60, 0, AEE_RO_TRANSPARENT);
```

(４)スリープモードに入らないようにしましょう。

BREWでは、３０秒間キーパッドの操作がない場合、バッテリを節約のため、スリープモードに入ります。スリープモードに入ると、タイマーは、指定した時間よりも長い時間経過した後に関数を起動します。このため、ゲームの動作が非常に遅くなります。
これを回避するには、EVT_APP_NO_SLEEPイベントを処理します。このイベントの受信したら、TRUEを返すと、スリープモードに入って欲しくないことを端末に伝えることができます(ただし、実際に受け入れるかどうかは、ベンダーの判断に任されています)。

以下のプログラムを完成させ、適切な場所に追加しましょう。

```
case EVT_???_??_?????:      return ????;
```

キー入力情報の取得

BREWには、どのキーが押されているかを取得するAPIはありません。その代わりに、キーが押されたり離されたりすると、イベントとしてイベントハンドラに通知されます。

キーに関するイベントは、キーが押されたときに発生するEVT_KEY_PRESS、キーが離されたときに発生するEVT_KEY_RELEASE、キー処理(発生する条件は機種によって異なります)イベントのEVT_KEYがあります。たとえば、数字１キーを押し、しばらく押し続けてから離すと、イベントハンドラには以下のイベントが送られます。

```
EVT_KEY_PRESS (wParam = AVK_1)  と  EVT_KEY (wParam = AVK_1, dwParam = 0)
    ↓
EVT_KEY (wParam = AVK_1, dwParam = KB_AUTOREPEAT)
    ↓
EVT_KEY (wParam = AVK_1, dwParam = KB_AUTOREPEAT)
    ↓
  ………
    ↓
EVT_KEY_RELEASE (wParam = AVK_1, dwParam = 0)
```

これらのイベントが発生すると、イベントハンドラのwParamには、AVK_SELECTやAVK_CLRといった、どのキーが押されたかを示すコードが渡されます。それぞれのキーに対してひとつのイベントが発生するので、イベントをそのまま利用しても、同時押しを判別することはできません。

ゲームプログラムでは、EVT_KEY_PRESSイベント(またはEVT_KEYイベントとdwParam)とEVT_KEY_RELEASEイベントを使用します。同時押しを判別するため、キーイベントの処理ではキーが押された、キーが離されたといったフラグを立てるだけにします。内部処理で各キーのフラグを判別し、キャラクターの移動などの処理に役立てます。

なお、エミュレータでは、キーボードのキーリピート機能により、キーを押しっぱなしにいるにもかかわらず、EVT_KEY_PRESSイベントが何度も発生してしまいます。

課　題

キーの入力によってキャラクターを移動させましょう。

(１)キーのフラグを格納する構造体を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
// キーフラグ構造体
typedef struct _KEY {
    boolean   UP;        // ↑
    boolean   DOWN;      // ↓
    boolean   LEFT;      // ←
    boolean   RIGHT;     // →
    boolean   SELECT;    // セレクト
    boolean   CLR;       // クリア
    boolean   NUM0;      // ０
    boolean   NUM1;      // １
    boolean   NUM2;      // ２
    boolean   NUM3;      // ３
    boolean   NUM4;      // ４
    boolean   NUM5;      // ５
    boolean   NUM6;      // ６
    boolean   NUM7;      // ７
    boolean   NUM8;      // ８
    boolean   NUM9;      // ９
    boolean   STAR;      // ＊
    boolean   POUND;     // ＃
} KEY;
```

主要なキーごとにフラグを割り当てます。条件を満たしていればTRUE、そうでなければFALSEを格納します。

(２)キーが押されているフラグ、キーが押されたフラグ、キーが離されたフラグを格納する変数を宣言します。以下の変数をアプレット構造体に追加しましょう。

```
    KEY   keyPress;        // キープレス情報(キーが押されたフラグ)
    KEY   keyRelease;      // キーリリース情報(キーが離されたフラグ)
    KEY   keyDown;         // キーダウン情報(キーが押されているフラグ)
```

(３)KEY構造体のフラグをクリアする関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
// キーフラグクリア
void プロジェクト名_KeyClear(KEY* pKey)
{
    MEMSET(pKey, 0, sizeof(KEY));
}
```

(４)(３)のプロトタイプを適切な場所に作成しましょう。

(５)アプリケーションが中断から復帰したとき、キーフラグを無効に(初期化)します。以下のプログラムを完成させ、*プロジェクト名*_OnAppResume関数の適切な場所に追加しましょう。

```
    // キー情報の消去
    プロジェクト名_????????(&pMe->keyPress);
    プロジェクト名_????????(&pMe->keyRelease);
    プロジェクト名_????????(&pMe->keyDown);
```

(６)キーが押された、キーが離されたという情報は、１フレームごとに消去します。定期的に消去しないと、キーの状態とフラグにズレが生じ、正しい情報が得られません。
以下のプログラムを完成させ、メインループの最後に追加しましょう。

```
プロジェクト名_????????(&pMe->keyPress);      // キープレスフラグ消去
プロジェクト名_????????(&pMe->keyRelease);    // キーリリースフラグ消去
```

(７)キーが押されたときのイベントの処理を作成します。このイベントは、キーが押されたことを示すので、「キーが押されたフラグ」をTRUEに、「キーが押されているフラグ」もTRUEにします。
以下のプログラムを完成させ、適切な場所に追加しましょう。

```
// キープレスイベント処理
static boolean プロジェクト名_OnkeyPress(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    switch(??????) {
      case AVK_UP:
        pMe->keyDown .UP     = ここは各自考えましょう;
        pMe->keyPress.UP     = ここは各自考えましょう;
        break;

      case AVK_DOWN:
        pMe->keyDown .DOWN   = ここは各自考えましょう;
        pMe->keyPress.DOWN   = ここは各自考えましょう;
        break;

      case AVK_LEFT:
        pMe->keyDown .LEFT   = ここは各自考えましょう;
        pMe->keyPress.LEFT   = ここは各自考えましょう;
        break;

      case AVK_RIGHT:
        pMe->keyDown .RIGHT  = ここは各自考えましょう;
        pMe->keyPress.RIGHT  = ここは各自考えましょう;
        break;

      case AVK_SELECT:
        pMe->keyDown .SELECT = ここは各自考えましょう;
        pMe->keyPress.SELECT = ここは各自考えましょう;
        break;

      case AVK_CLR:
        pMe->keyDown .CLR    = ここは各自考えましょう;
        pMe->keyPress.CLR    = ここは各自考えましょう;
        break;

      case AVK_0:
        pMe->keyDown .NUM0   = ここは各自考えましょう;
        pMe->keyPress.NUM0   = ここは各自考えましょう;
        break;

      case AVK_1:
        pMe->keyDown .NUM1   = ここは各自考えましょう;
        pMe->keyPress.NUM1   = ここは各自考えましょう;
        break;

      case AVK_2:
        pMe->keyDown .NUM2   = ここは各自考えましょう;
        pMe->keyPress.NUM2   = ここは各自考えましょう;
        break;

      case AVK_3:
        pMe->keyDown .NUM3   = ここは各自考えましょう;
        pMe->keyPress.NUM3   = ここは各自考えましょう;
        break;

      case AVK_4:
        pMe->keyDown .NUM4   = ここは各自考えましょう;
        pMe->keyPress.NUM4   = ここは各自考えましょう;
        break;

      case AVK_5:
        pMe->keyDown .NUM5   = ここは各自考えましょう;
        pMe->keyPress.NUM5   = ここは各自考えましょう;
```

```
        break;

      case AVK_6:
        pMe->keyDown .NUM6   = ここは各自考えましょう;
        pMe->keyPress.NUM6   = ここは各自考えましょう;
        break;

      case AVK_7:
        pMe->keyDown .NUM7   = ここは各自考えましょう;
        pMe->keyPress.NUM7   = ここは各自考えましょう;
        break;

      case AVK_8:
        pMe->keyDown .NUM8   = ここは各自考えましょう;
        pMe->keyPress.NUM8   = ここは各自考えましょう;
        break;

      case AVK_9:
        pMe->keyDown .NUM9   = ここは各自考えましょう;
        pMe->keyPress.NUM9   = ここは各自考えましょう;
        break;

      case AVK_STAR:
        pMe->keyDown .STAR   = ここは各自考えましょう;
        pMe->keyPress.STAR   = ここは各自考えましょう;
        break;

      case AVK_POUND:
        pMe->keyDown .POUND  = ここは各自考えましょう;
        pMe->keyPress.POUND  = ここは各自考えましょう;
        break;
    }

    return TRUE;
}
```

(８)(７)のプロトタイプを適切な場所に作成しましょう。

(９)キーが離されたときのイベントの処理を作成します。このイベントは、キーが離されたことを示すので、「キーが離されたフラグ」をTRUEに、「キーが押されているフラグ」をFALSEにします。
(８)を参考に以下のプログラムを完成させ、適切な場所に追加しましょう。

```
// キープレスイベント処理
static boolean プロジェクト名_OnKeyRelease(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    switch(??????) {


    ※case文は、問題文および(７)を参考に、各自作成してください


    }

    return TRUE;
}
```

(10)(９)のプロトタイプを適切な場所に作成しましょう。

(11)イベントハンドラでキーが押されたイベントとキーが離されたイベントが処理されるようにします。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
    case EVT_???_?????:    return プロジェクト名_OnKeyPress  (pMe, wParam, dwParam);
    case EVT_???_???????: return プロジェクト名_OnKeyRelease(pMe, wParam, dwParam);
```

(12)キーの状態によって、プレイヤーを移動させましょう。
・ヒント１　プレイヤーの座標を保持する変数が必要です

```
int    plyX;
int    plyY;
```

・ヒント２　ゲームで使用する変数の初期化関数で、プレイヤーの座標を初期化します

・ヒント３　上下左右に対応するキーが押されたら、プレイヤーの座標を増減します

・ヒント４　キーが押されているかどうかの判定は、以下のように行います

```
// 移動
if(pMe->KeyDown.LEFT == TRUE)   // 左
    (左が押されている)
```

・ヒント５　ヒント３，４は内部処理で行います

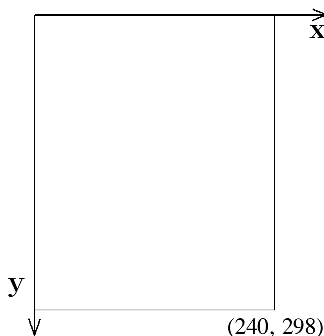
・ヒント６　プレイヤーを描画するIDISPLAY_BitBlt関数の２つ目と３つ目の引数を変更します

(13)プレイヤーの移動範囲を画面内だけにしましょう。
・ヒント１
　座標は、画像の左上を指しています。

・ヒント２
　ｘ座標の最小値は、原点の0です。この値未満になるということは、原点より左側にいるということになり、画面外にいることになります。この場合は、ｘ座標を0にし、画面左に戻します。

・ヒント３
　ｘ座標の最大値は、画面右端の240になりそうですが、実は違います。ｘ座標は、キャラクターの左端の座標なので、ここが240の場合は、すでに画面外に出ています。キャラクターの右端(左端 + 幅)が240を越えているかどうかを調べるか、キャラクターの左端が、240から幅を引いた値(240 - 幅)を越えているかを調べます。越えている場合は、ｘ座標を240から幅を引いた値にします。

・ヒント４
　ｙ座標も同様に考えます。



課　題

　ボス敵を表示し、プログラムによって動かしましょう。