J3W入門 for Linux講座

水谷純
http://www.nk.rim.or.jp/~jun/index.html

第2回　J3C言語の特徴

今回は「物体をプログラムする」というJ3C言語の特徴をオブジェクト指向的な側面から解説します。

J3C言語は「物体」を動かすプログラムを作成するための言語です。J3C言語を習得するためには身の回りの「物体」に注意して、「何」を「どのように動かすプログラムとしようか？」と考えて実際にプログラミングしてみることです。身の回りで動くものを見つけて、それがどのように動いているかを観察することで、オリジナルなプログラムのアイデアが見つかると思います。

●インスタンス

前回、1つのクラスは汎用言語の1つのプログラムに相当すると説明しました。複数のクラスは同時に動作することが可能で、全体として1つの3次元空間を構成します。同一のクラスで定義される物体が、同時に複数動作することもできます。そこで動作中のクラス1つをインスタンスと呼んでクラスと区別します。クラスとは、物体の設計図で、インスタンスが物体そのものと考えると理解しやすくありませんか？

図1は独立して動作するクラスの概念図です。コンピュータは複数存在し、どれか1つのクラスを実行しています。同じクラスを複数のコンピュータに実行させることもできます。また実行を終了したクラス(コンピュータは消滅)や、実行されていないクラスはインスタンスではありません。このインスタンスが3次元空間で実体化して運動することで、3Dアニメーションを実現します。

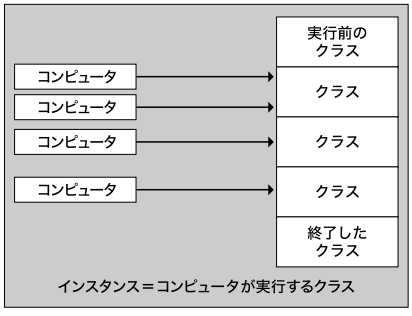
例えば、図2に示す空間ではビルもクルマも道路も山も3次元空間に存在するものすべてが、クラスの定義に従って動作するコンピュータにコントロールされているインスタンスとなります。ビルがたくさん立ち並んでいたとしても、1つのクラスの複数のインスタンスとして簡単に実現できます。空間で実体を持たないインスタンスも使いますが、普通はインスタンスは1つの物体と考えておきましょう。個々のクラスは、別のプログラムのように完全に独立して動作します。複雑な3Dアニメーションでも個々のインスタンスを順に追加することで、他のクラスに影響することなくシーン（3次元空間）を組み立てていくことができます。

J3C言語のクラスとインスタンスという用語の違いを理解していただけましたか？ クラスは遺伝子で、インスタンスはその遺伝子から生まれて実際に生きている個々の生物です。同じ遺伝子をもつ生物が1匹の場合も1000匹の場合もありえます。例えばJ3W付属のサンプルでは、`taco.j3d`のタコの足は6個のクラスを使った120個のインスタンスでできています(画面1)。

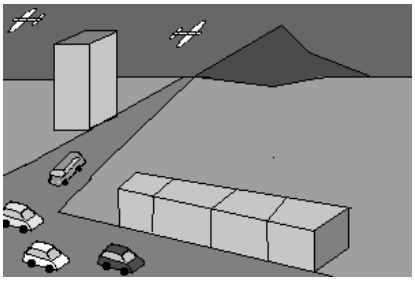
●インスタンスの例

実際に複数のインスタンスを動作させる簡単な例を実行してみましょう。リスト1では、独立した2つのクラスの、それぞれ1つのインスタンスが同時に動作しますが、全体として意味のある動きをします。

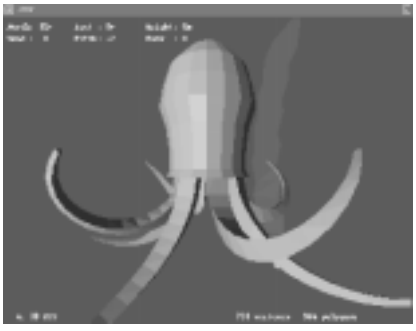
【図1】クラスとインスタンスの関係



【図2】3次元空間内に存在するインスタンス



【画面1】

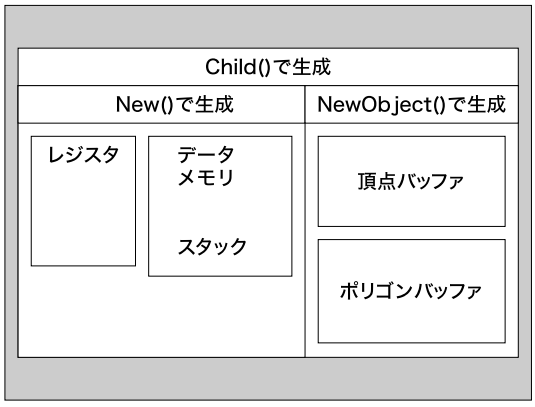


【リスト1】インスタンスの動作例

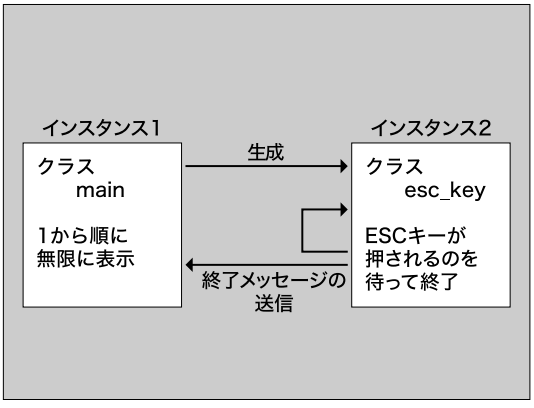
```
class esc_key {
  volatile int key;      //変数宣言
  int INIT() {
    RQ = 9;
  }
  int RUN() {
    key = InKey();       //キーコード入力
    if (key == 0x1B)     //ESCキーなら
      Stop();            //全インスタンスの終了
  }
  int EVENT() {}         //イベント処理はない
}

class main {
  volatile int i;        //変数宣言
  int INIT() {
    new(esc_key, 50);    //esc_keyを実行開始
    i = 0;
  }
  int RUN() {
    Number(i);           //数値の出力
    Char(13);            //改行の出力
    i = i + 1;           //iの値を1増やす
  }
  int EVENT() {}
}
```

【図3】J3W仮想CPUの構造



【図4】インスタンスの関係



この例では、まずmainクラスのINITメソッドが最初に実行され、その中のnew(esc_key, 50)でesc_keyクラスのインスタンスを生成しています。「50個のデータメモリを持つクラスesc_keyのインスタンスを生成して、INITメソッドから実行しなさい」という意味になります。

インスタンスを生成するときには、実行に必要なデータメモリの大きさを指定する必要があります。インスタンスを生成すると、図3に示すJ3Wの仮想CPUが内部的に生成されますが、このときクラスのデータメンバ（変数）と、スタック領域を格納するデータメモリの大きさを指定する必要があります。データメモリの量は、配列や再帰を使用しなければ50か100で十分と思います。必要ならば大きな値を指定してください。1000でも10000でも制限はありません。

ここで2つのクラスのインスタンスの動作を考えます。1つ目のmainクラスのインスタンスは0から順に永久に1を加えてそれを表示し続けます。2つ目のインスタンスはESCキーが押されるまで永久に待ちますが、ESCキーが押されるとStopを実行して、すべてのインスタンスを終了させます。

個々に考えるとどちらも無限ループかそれに近いものですが、2つのプロセスが同時に実行されていると「ESCキーが押されるまで0から順に1を加えてそれを表示する」プログラムとなります。2つ目のインスタンス（今後esc_keyクラスインスタンスと呼びます）はESCキーが押されるとStop()に処理が移り、他のすべてのインスタンスを終了させるためです。図4に示すようにmainクラス（のインスタンス）はesc_keyクラスインスタンスを生成しますが、その後は独立して動作します。esc_keyクラスインスタンスは、ESCキーが押されると、すべてのインスタンスにプログラム終了のメッセージを送信してプログラムは終了します（図4）。

RUN()メソッドは1回実行すると、他のインスタンスに実行を渡します（内部的にThrow組み込み関数を呼び出している）。つまり、各インスタンスのRUNが1度ずつ順に実行されます。すべてのインスタンスがRUNを1回実行するたびに画面更新が行われます。

物体の移動などの時間を指定する組み込み関数では、自動的に必要なタイミングで別のインスタンスに実行が渡されます。時間を指定する組み込み関数では長時間の指定でも問題ありません。一方、INIT、RUN中、またはそこから呼ばれるメソッド中で計算など長時間かかるループを実行すると、別のインスタンスに実行が渡らず、画面の更新もされません。どうしても必要な場合はThrow組み込み関数をループ内に置いて他のインスタンスに実行するチャンスを与えてください。基本的にはループを避けてRUNが繰り返し実行されるよう利用する方がよいでしょう。

【リスト2】キー入力を有効にする

```
class esc_key {
  volatile int key;
  int INIT() {
    RQ = 9;
    GraphMode();
  }
  int RUN() {
    key = RawKey(0x1B); //キーコード入力
    if (key) Stop();    //ESCキーなら終了
  }
  int EVENT() {}        //イベント処理はない
}
```

esc_keyクラスインスタンスの最初の文「RQ = 9」はインスタンスの識別番号を9に設定するための組み込み関数です。他のインスタンスからメッセージを送る場合などに使用しますが、リスト1では使っていません。このesc_keyクラスのインスタンスを、任意のＪ３Ｃのプログラム内でnew(esc_key, 50);を実行してインスタンス化するだけで、ESCキーで強制終了という機能を追加できます。

InKeyによるキー入力はグラフィックウィンドウかテキストウィンドウがアクティブ（フォーカスがあたっている）になっている必要があります。アクティブでなくてもキー入力を有効にするには以下のリスト2を使用してください。

ただし、リスト2のようにRawKey組み込み関数を使うと複数キーの同時入力チェックが可能となるものの、別のウィンドウで実行されているアプリケーション（例えばvi）でESCキーが押されても実行中のJ3Cのプログラムは終了します。RawKey組み込み関数はグラフィックウィンドウが開いている状態（GraphMode実行後）で使う必要があります。

●インスタンスの引数

C++におけるコンストラクタの引数に相当する機能がインスタンスの引数です。1つのクラスから複数の同一のインスタンスを生成しては、同じ物体が同じ位置で同じ動きをすることになって、無意味な場合があります。大きさが異なるとか、速度が異なるといったインスタンスを同一のクラスから生成するために、インスタンスを生成するときに引数を渡すことができます。

インスタンスが生成されるとき、親インスタンスのレジスタ変数（RX,RY,RZ,RH,RP,RB）を引き継ぎます。アセンブリ言語（j3dasm）レベルでは、RL、RH、RP、RB、R1～R6の計13個のレジスタが実際に渡されますが、J3C言語では、RX、RY、RZ、RH、RP、RBの6個のレジスタ変数を利用できます。これらのレジスタ変数に値を設定してnewを実行することで、同じプログラムを複数のインスタンスで共有して、引数により異なった動作をさせることができます。

リスト3は、mainクラスのインスタンスから生成された3つのWaitAndWriteクラスのインスタンスが同時に動作します。WaitAndWriteクラスのインスタンスは、newで生成される時点の親インスタンスのレジスタ変数を引数として受け取って、別々の文字を異なるタイミングで出力します。

●ライブラリ

Cの関数のように定型的な処理を定義するために、クラスの外で関数や定数を宣言するライブラリという機能があります。ライブラリは、クラスと異なりインスタンス化することはありません。スコープはグローバルで複数のクラス

からライブラリに属するメソッドと定数を参照することができます。クラスのようにデータメンバ(クラスの変数)を持つことはできませんが、定数を宣言することはできます。例えば、色を異なるクラス中で統一した名前で指定したい場合には、リスト4のようにライブラリを使います。どのクラスからでも、Color.Blackと指定すれば黒に相当する色番号を名前で指定することができるようになります。

リスト5はライブラリで関数を利用する例です。クラスのデータメンバ(変数)を参照することはできないため、関数が必要とする情報は引数として渡します。どのクラスからでも、このライブラリを使うとError.Message(n)を呼び出すことでnの値に対応したエラーメッセージを表示できます。switch文のcaseブロックの最後にはbreak;が必要なことに注意してください。breakを忘れると、C言語と同じように、その後に続くcaseを実行します。libraryの関数は値を1つしか返せませんが、レジスタ変数をグローバル変数として使うことで複数の値を返すこともできます。

●インポート

インポートは指定したファイルをその位置に挿入する働きをします。汎用的に使用できるクラスやライブラリをファイルに記述して、その機能が必要な場合にインポートします。Cのヘッダファイルのインクルードと同様な使い方になります。同じファイルが何度もインポートされても、最初に1度だけコンパイルされ、2度目からは無視されます。同一ファイルかどうかの判定はファイル名のみで行っているため、別ディレクトリに同名のファイルが存在し、両方のファイルをインポートしようとしても同一ファイルと判定されます。

リスト2のソースをesckey.j3cというファイル名で保存します。リスト6のように別ファイルからimportで指定すると、アセンブル時にesckey.j3cが挿入され、リスト1と同等なプログラム(RawKeyバージョン)となります。このように挿入して使用するファイルをインポートファイルと呼びます。

インポートの利点は、1度作成したクラスやライブラリを別のファイル中に取り込んで再利用できることです。J3C言語はクラスの継承という機能を持っているため、インポートされるファイルには手を加えずに、必要な部分だけをインポートする側のファイルで変更できます。いろいろな形状を持つ物体のクラスを記述したファイルを用意しておくと、それらのファイルをインポートして、必要ならば色や位置や動作を再定義することで、それらの物体を容易に3次元空間に追加できます。

インポートファイルのパスは、

```
import "../../lib/abc.j3c";
```

のようにインポートするファイルからの相対パスで指定することもできます。インポートファイル中に、さらにインポートファイルを記述することも可能です。

■終わりに

今回までの解説でJ3Wに付属している文書とサンプルプログラムが理解しやすくなると思います。次回は3次元に関する話題に移ります。3次元の形状の作成のコツを中心に解説する予定です。

【リスト3】インスタンスの引数の使用例

```
class WaitAndWrite {
  volatile int c, time;
  int INIT () {
    c = RX;                  //インスタンス引数
    time = RY;               //インスタンス引数
  }
  int RUN () {
    Pause(time);             //時間待ち
    Char(c);                 //文字の出力
  }
  int EVENT() {}
}

class main {
  volatile int i;
  int INIT () {
    RX = 'A'; RY = 1000;
    new(WaitAndWrite, 50);   //Aを1秒ごとに出力
    RX = 'B'; RY = 2000;
    new(WaitAndWrite, 50);   //Bを2秒ごとに出力
    RX = 'C'; RY = 1500;
    new(WaitAndWrite, 50);   //Cを1.5秒ごとに出力
    i = 0;
  }
  int RUN () {
    Pause(5000);              //時間待ち(5秒)
    Char(13);                //改行出力
    i = i + 1;
    if (i>=10) Stop();       //終了
  }
  int EVENT() {}
}
```

【リスト4】ライブラリ使用例

```
library Color {
  final int Black      = 0 ;
  final int Blue       = 1 ;
  final int Green      = 2 ;
  final int Aqua       = 3 ;
  final int Red        = 4 ;
  final int Purple     = 5 ;
  final int Yellow     = 6 ;
  final int White      = 7 ;
  final int Gray       = 8 ;
  final int Navy       = 9 ;
  final int Teal       = 10 ;
  final int LightGreen = 11 ;
  final int Brown      = 12 ;
  final int Fkusia     = 13 ;
  final int Orange     = 14 ;
  final int Cream      = 15 ;
  //関数宣言を置いてもよい
}
```

【リスト5】ライブラリで関数を利用する例

```
library Error {
  //ここで変数宣言できない
  //定数宣言は可
  int Message(int i) {
    //ここで局所変数宣言は可能
    switch (i) {
      case 1 :
        String("Error 1");  //文字列表示
        Char(13);           //改行出力
        break;              //忘れないこと
      case 2 :
        String("Error 2");  //文字列表示
        Char(13);           //改行出力
        break;              //忘れないこと
      // 必要ならcaseが続く
      default :
        String("Unknown");  //文字列表示
        Char(13);           //改行出力
        break;              //なくてもいい
    }
  }
  //ここに別の関数定語も可能
}
```

【リスト6】インポートの例

```
import "esckey.j3c";    // esckey.j3cを挿入

class main {
  volatile int i;
  int INIT() {
    new(esc_key, 50);
    i = 0;
  }
  int RUN() {
    Number(i);
    Char(13);
    i = i + 1;
  }
  int EVENT() {}
}
```

Column J3WとPS2LinuxKit

J3Wは、PS2LinuxKitでも、ソースを修正することなくコンパイルと実行を行えます。ただ、PS2のCPUであるEmotion Engineの数値演算コプロセッサでは32ビットの浮動小数点数しかサポートしていないため、doubleを多用しているJ3Wでは非常に遅くなってしまいます。

そこで、doubleをfloatに変更して試してみました。表の3つのヘッダーファイルの先頭付近に「#define double float」の1行を追加します。さらに、xkey.cppの先頭付近の

```
#include <iostream.h>
```

とある行を削除します。

その後コンパイルして実行すると2〜3倍高速化され、使用に耐えられるようになります。行列演算にVPUを使えば、より高速化できるはずです。（水谷純）

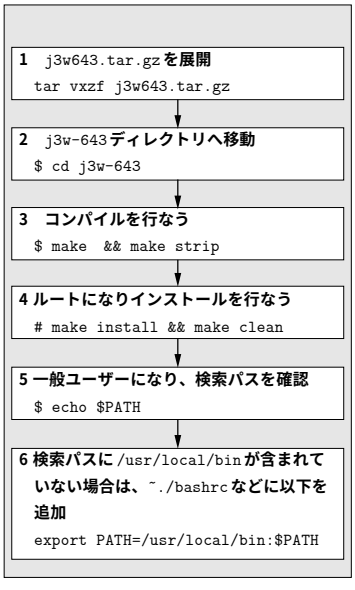
【表】「#define double float」を追加するヘッダーファイル

source/j3w/misc3d.h
source/j3w/hobj3d.h
source/j3w/tpolygon.h

Column J3W for Linuxのインストール

J3Wの最新ソースコード「j3w643.tar.gz」を付録CD-ROMに収録しています。インストールは、流れ図のように行なってください。インストールに関してさらに詳しい情報を得るには、J3W添付のREADME.eucファイルを確認してください。（編集部）

【図】J3Wのインストール手順



Column J3Wの内部構造の解説

前号ではJ3Wの3Dエンジンの概要の紹介までしたので、今回はもう少し詳しく見ていきましょう。J3Wでは、頂点、面、位置、角度といった物体の情報は複数のクラスに分散して格納しています。まずは、頂点と面(多角形）について見てみましょう。TPolygonsクラス（**図A**）は、1つの物体を構成する複数の頂点と多角形を管理しています。TVertexクラスが管理する頂点は物体を構成する上で必要な情報ですが、さらに物体の表面の性質を指定する必要があります。J3Wでは物体は多くの面（多角形、ポリゴン）から構成され、1つの物体の「形」に関する情報の管理がTPolygonsクラスの役割となります(**図A**)。物体を構成する多角形は頂点の情報と面の情報に分かれて、それぞれTVertexクラスとPolygon3Dクラスに保持されています。

●頂点

TVertexクラスは、物体を構成する最も基本的なデータである頂点を管理しています。頂点の座標データは、local、world、eyeという3種類の3次元座標値（x、y、z）を1組にしたVectorSet構造体に格納します。TVertexクラスは、そのVectorSet構造体へのポインタを配列として持つことによって、複数の頂点を管理します。

テーブルの上に、1冊の雑誌が存在するとしましょう。雑誌のサイズが、高さ30cm、幅20cm、厚さ3cmといえば誰でもどんな大きさか理解できますよね。例えば、この雑誌の表紙の右下の角を原点として雑誌を形作る8個の頂点の位置をローカル座標系で指定します。また、テーブルが存在する、床のある1点を基準として雑誌の8個の頂点を表したものがワールド座標系（world）となります。さらに、ワールド座標系のある位置の「目」を基準とする視点座標系（eye）でも同様に8個の頂点の座標を持ちます。「目」も含めて複数の物体がワールド座標系を動き回る状況を考えるとこのように3種類の座標系が必要になります。

●面

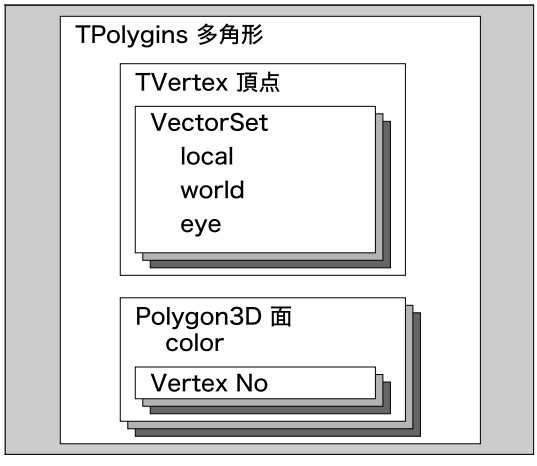
J3Wでは面はすべて多角形で定義します。1つの多角形は頂点座標以外にも、色、反射率、環境光の強度、重心のオフセットという情報を持っています。1つの多角形に関する情報はPolygon3D構造体が持ち、Polygon3D構造体へのポインタの配列をTPolygonsクラスが所有することによって複数の多角形を管理しています。

多角形の頂点は、Polygon3D構造体がVectorSet内の頂点を頂点番号で参照します。その頂点番号を配列で持つことによって、多角形の実際の頂点データにアクセスします。このように頂点と多角形を分離することによって、複数のポリゴンで頂点を共有することができます。

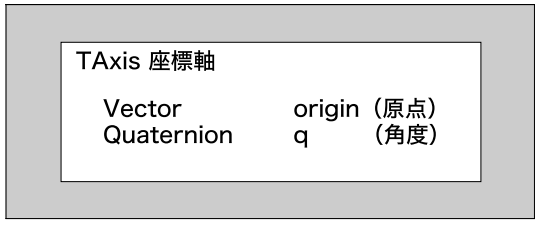
ポリゴンの反射率とはスペキュラ（ハイライト）の強度のことで、値が大きいほど良く光る面となります。環境光の強度は0から100までの値をとり、光があたらない面の明るさを決めます。

J3Wでは視点から遠いポリゴンから塗り潰し法で描画しますが、視点からの距離はポリゴンの重心位置で比較します。小さいポリゴンが、地面などを表す大きいポリゴンの近くにあると位置関係が狂ってしまうことがあります。重心のオフセットは、ポリゴンの重心を強制的に変化させて、そのポリゴンの表示順序を変えるために使います。

【図A】TPolygonsクラスの概要



【図B】TAxis座標軸



●移動と回転

3Dアニメーションの基本となる物体の移動や回転は、物体のローカル座標系の移動や回転として扱います。ローカル座標系の原点の位置をワールド座標系（または親座標系）の座標で指定することで物体の位置が決まり、これを変化させれば物体は移動します。ローカル座標系とワールド座標系（または親座標系）との角度はquaternionの形式で格納されます。この角度を変化させれば物体は回転します。このように物体の回転や移動は単にTAxisに値を設定することで実現します。頂点や面の操作と比べると簡単です(**図B**)。（水谷純）