J3W for Linux 入門講座

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

第7回　objviewの使用法

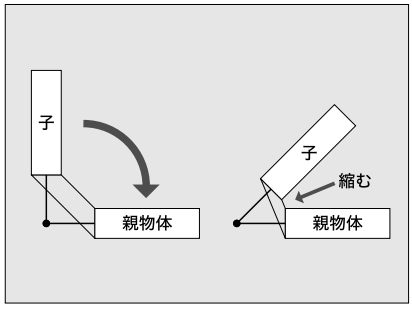
これまでは、幾何学的な形状の「硬そうな」物体ばかりを例にしてきましたが、今回は、ウネウネとした動物的な動きをする物体を作成してみましょう。次に、物体の形を個別に確認するためのツールである objview.j3c の使い方を解説します。

■スキン

「腕を曲げる」動作を考えると、肘の皮膚は伸び縮みしていることが分かります。動物の体は骨とそれを取り巻く筋肉、そして皮膚でできています。関節の角度が変化しても、表面の皮膚が伸縮することによって関節の部分で分離することはありません。J3Wでは多角形の頂点を登録する場合に、親の頂点を含めることができます。子物体と親物体の距離が変わると多角形が伸縮します（図1）。

階層構造の物体は、頂点を持つ親オブジェクトが存在する場合、多角形の頂点番号に負の数を使用して親オブジェクトの頂点を指定することができます。独立して運動する物体間には、多角形（スキン）を張ることができます。このとき親オブジェクトの頂点は負の数を使用して指定するため、親の0番頂点は指定できないことに注意してください。親1つに複数の子を持たせる場合も同様に、それぞれの子が親の頂点を指定できます。なお、子は直接の親の頂点だけを参照できます。

【図1】腕を曲げる動作を多角形で表した様子



■インスタンスの再帰的生成

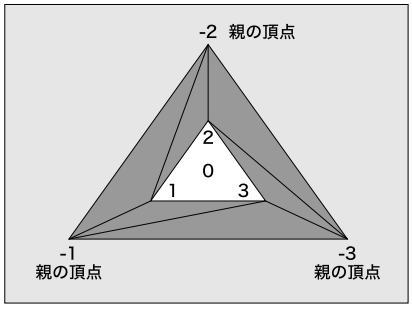
J3W付属のタコ（taco.j3c）の足の部分は、インスタンスを再帰的に生成しています。階層構造の上位（親）の頂点を使って面（スキン）を次々と登録していき、階層構造の下位に自分自身を生成しています。リスト1 に示す skin.j3c の Common クラス自身は実体化せずにTop、Child、Baseクラスに継承され、スキンで構成された三角柱を生成する機能を提供しています。

Commonクラスは、三角形を1つ登録した後、親の頂点を使って面を登録して三角柱を生成しています（図2）。三角柱の側面に三角形を張ることで、上下の三角形が「ねじれる」ことができるようになっています。最初の頂点（頂点番号が0）は子インスタンスから参照できないため、原点に置いていますが、使用しません。

Base クラスが階層構造の基部になるクラスで、正三角形になるように頂点を登録してChildクラスを生成しています。Childクラスを生成するとき、RXレジスタ変数に三角形の大きさ、RYレジスタ変数に再帰的に生成する数、RZレジスタ変数に色を設定します。

Childクラスは、生成時のレジスタ変数の値に従って三角柱を生成しますが、先端に行く（X座標が負）ほど細くなるように、次に生成するChildクラスのインスタンスに渡す大きさの値（RX）を小さくしています。また、再帰的に生成する数（RY）を1つ減らしています。RYレジスタ変数が「0」の場合には、Top クラスを生成して再帰から抜けます。こうしてChildクラスは、自分自身を再帰的に呼び出して、階層構造を形成しながら多くのインスタンスを生成しています。

【図2】リスト1（skin.j3c）のCommonクラスによる三角柱の生成



枝を分岐させて再帰的に生成し、先端部に葉が付くようなオブジェクトも同じように作成できると思います。

動きの設定は、Childクラスで単にピッチ回転を繰り返すだけですが、ねじれを加えているため動物的なちょっと無気味な動きをします。

■オブジェクトビューア

形状を確認しながら物体を定義できると、ストレスなくプログラミングを進めることができると思います。目的としている本来のプログラムとは独立して、作成中の物体の形状を確認するために、objview.j3c というツールが用意されています。

リスト1の skin.j3c は物体を作成するクラスだけですが、プログラムとして完成させるためには、いろいろな初期設定をしたり、視点を持つ物体を作成する必要があります。

J3C で3D アプリケーションを作成する場合の基本的な手順は次のようになります。まず最初に、個々の物体の形状と動作を独立して作成します。次に、部品として用意された物体を空間に追加していくことで世界を組み立てます。

各部品のクラスを objview.j3c に組み込むと、部品の形状や動きを確認して完成することができます。objview.j3c には、インポートした model.j3c で定義済みの model クラスを継承して、キー入力に従って移動や回転する MyModel クラスがあります。この model.j3c 中で、任意のクラスを継承するようにmodelク

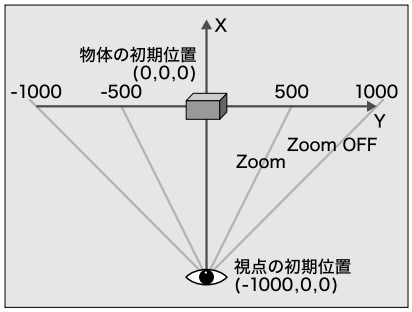
ラスを定義することで、部品としての物体の形状を確認するプログラムができます。

model.j3cで定義される物体は、ワールド座標系の原点（変更可能）にあり、視点はX座標が-1000の位置（南に1000離れた位置）にあります（図3）。初期状態で画面に表示される範囲は、左右（Y座標）に2000、上下（Z座標）に1500になります。ズームした場合は、それぞれ1000と750の範囲が見えることになります。視点や物体を移動させた場合には、表示範囲は変化します。

物体の移動速度は、objview.j3cのMyModelクラスのEVENTメソッドで決められており、1回のキー入力で100の距離を0.1秒間で移動します。視点は、距離50を0.1秒間で移動します。どちらも回転速度は3度/0.1秒となっています。小さい物体を近くで見る場合には、見失わないように移動速度を小さくする必要があるかもしれません。

model.j3cのmodelクラスは、物体のクラ

【図3】ワールド座標系にあるmodel.j3cで定義された物体



【リスト1】skin.j3c

```
// 共通の基底クラス
class Common {
    volatile int radius, color, count;

    // 正三角形の頂点を登録
    int Def_Tri(int radius, int x) {
        volatile int P, Q, R;

        PushRegisters();
        R = radius;
        P = sqrt(radius * radius * 3) / 2;
        Q = radius / 2;
        RX = x;
        RZ = -R; RY = 0;  Point();      // #0;
        RZ = Q;  RY = P;  Point();      // #1
        RZ = Q;  RY = -P; Point();      // #2
        PopRegisters();
    }

    // 親物体の頂点との間に面（スキン）を張る
    int def_shape(int r, int color) {
        DefPoint(0, 0, 0);          // 0 頂点はダミー
        Def_Tri(r, -200);
        DefPlane(color, 3, 2,  3, -3);  //
        DefPlane(color, 3, 2, -3, -2);  //      -2
        DefPlane(color, 3, 1,  2, -2);  //       2
        DefPlane(color, 3, 1, -2, -1);  //    1     3
        DefPlane(color, 3, 3,  1, -1);  // -1        -3
        DefPlane(color, 3, 3, -1, -3);  //
    }

    int INIT() {}
    int RUN () {
    Wait();
    }
    int EVENT () {}
}

//  先端部：階層構造の最下位
class Top extends Common {
    int INIT() {
        color = 4;
        radius = RX;
        def_shape(radius, color);
        DefPlane(color, 3, 3, 2, 1);
        ClearRegisters();
        RX = -100;
        SetPosition();              // 初期位置と角度
    }
}
```

↘

```
// 中間部：オブジェクトを再帰的生成
class Child extends Common {
    int initialize() {
        color = RZ;
        radius = RX;
        count = RY;
        def_shape(radius, color);
        ClearRegisters();
        RX = -100;
        SetPosition();              // 初期位置と角度
    }
    int INIT () {
        initialize();
        RX = radius - 5;
        RY = count - 1;
        RZ = color;
        if (count > 0) child(Child, 50, 4, 6)
        else child (Top, 50, 4, 7);// Top の生成
        RotBank(3000, 80);
        RotPitch(2000, 60);         // 2秒間で回転
    }

    int RUN () {                    // 動きを定義
        RotPitch(3000, -120);       // 3秒間で回転
        RotPitch(2000, 120);
    }
}

// 階層の根元
class Base extends Common {
    int initialize() {
        ClearRegisters();
        RP = -720;
        SetPosition();              // 初期位置と角度
        DefPoint(0, 0, 0);          // ダミー頂点
        Def_Tri(radius, 0);
        DefPlane(color, 3, 1, 2, 3);
    }
    int INIT() {
        radius = 300;               // 基部の太さ
        color = 2;
        initialize();
        RX = radius;
        RY = 20;qi                  // 中間部の長さは10個
        RZ = color;
        child(Child, 50, 4, 6);     // 子オブジェクト
    }
}
```

↗

スを継承するか、または物体のクラスを子オブジェクトとして生成します。modelクラスを継承するMyModelは、modelクラスのRUNとEVENTメソッドを書き換えることで、作成される物体をキー入力で操作できるようになります（表1）。

実際にmodelクラスを作成してobjviewの使い方を見ていきましょう。作業環境は、今月号の本誌付録CD-ROMのsample.tar.gzを展開してできるskin.j3c、m_skin.j3c、m_skin2.j3cをj3w-643/j3c_scriptディレクトリにコピーしているものとします。

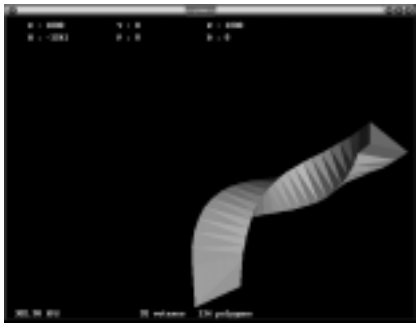
リスト2に示すm_skin.j3cでは、リスト1で作成したskin.j3cをインポートしています。skin.j3c中のBaseクラスは、物体の階層構造の根元を作るクラスになっています。modelクラスより下位の階層に物体を生成するため、modelクラス自身は形を持つ必要がありません。単にインスタンスの位置をワールド座標系の原点に置いて、子オブジェクト（階層構造の下位の物体）としてBaseクラスのインスタンスを生成します。

skin.j3cで定義されている物体が大きいため、初期値であるワールド座標系の原点位置より遠ざかった位置で、さらに下に移動した位置に配置しています。背景の色も黒に変更してみました。こうしてskin.j3cをobjview.j3cに組み込んでできたプログラムは、「目に見えないテーブルの上に物体を置いて、テーブルを動かすことによって形状を色々な角度から確認する」形式になります。

m_skin.j3cをmodel.j3cにコピーした後、objview.j3cをコンパイルします。model.j3cを変更してobjview.j3cをコンパイルしても、いつもobjview.j3dという実行ファイルができてしまいます。objview.j3dの名前を変更しておくと、別のmodel.j3cを設定してコンパイルしても上書きされません。実行例1のようにしてobjskin.j3dを実行すると、画面1のようになります。ちょっと不気味な動きをすると思いませんか？

リスト3はmodelクラスがBaseクラスを継承しています。リスト2とは異なり、modelクラス自体が形を持つ物体となります。この方法では、Baseクラス自身を継承によって書き換えるため、より柔軟な設定を試すことができますが、継承元のクラス（この場合はBaseクラス）の内容を理解して、必要な変更を加える必

【画面1】objskin.j3dの実行画面



【実行例1】リスト2のコンパイルと実行

```
$ cp m_skin.j3c model.j3c
$ j3cc objview.j3c
$ mv objview.j3d objskin.j3d
$ j3w objskin.j3d
```

【表1】objviewのキー操作

操作対象	キー	動作
物体の操作	←	左回転
	→	右回転
	↑	下向き回転
	↓	上向き回転
	L	バンク反時計回り
	K、R	バンク時計回り
	F	前進
	B	後退
	W	上移動
	X	下移動
	S	左移動
	D	右移動
	I	元の位置（原点）に戻る
カメラの操作	Z	拡大
	Q	縮小
	U	上向き回転
	N	下向き回転
	H	左回転
	J	右回転
	O	前進
	P	後退
表示	T	ワイヤーフレーム表示
	Y	通常の表示
	ESC	終了

【リスト2】m_skin.j3c

```
import "skin.j3c";

class model {
      int INIT() {
            BackgroundColor(0x000);     // 背景は黒
            NewObject(0,0);             // インスタンス自身は形状を持たない
            ClearRegisters();
            RX = 1000;
            RZ = 1000;
            SetPosition();              // (1000, 0, 1000) に配置
            child(Base, 50, 4, 1);
      }
      int RUN() {}                      // MyModel で上書きされる
      int EVENT() {}                    // MyModel で上書きされる
}
```

【リスト3】m_skin2.j3c

```
import "skin.j3c";

class model extends Base {
      int INIT() {
            NewObject(4, 1);
            radius = 300;           // 基部の太さ
            color = 2;              // 色
            initialize();
            RX = radius;
            RY = 20;                // 中間部の長さは20個
            RZ = color;
            child(Child, 50, 4, 6); // 子オブジェクト
            ClearRegisters();
            RX = 1000;
            RZ = 1000;
            RP = -720;
            SetPosition();          // (1000, 0, 1000) に配置
      }
}
```

要があります。

コンパイルの際には、実行例2のようにm_skin2.j3cをmodel.j3cにコピーした後、objview.j3cをコンパイルして、objview.j3dをobjskin2.j3dにリネームしておきます。実行してみると、objskin.j3dとはキー操作が異なることに気が付かれると思います。objskin.j3dではMy_modelというテーブルの上に立てた物体を操作しているのに対して、objskin2.j3dではテーブル自身が物体となって90度向きを変えて立ち上がっています。

j3w-643/j3c_script/objviewディレクトリには、他にも表2に示すように、いろいろな物体用のmodel.j3cがm_XXX.j3cというファイル名で収録されていますので、objview.j3cと組み合わせて試してみてください。また、objview.j3cの構造を簡単に表3にまとめました。objview.j3cをカスタマイズするときの参考にしてください。静止した物体だけではなく、自立して運動している物体もobjview.j3cに組み込んで表示することができます。いろいろな形や動きを簡単に試すことができると思います。

【実行例2】リスト3のコンパイルと実行

```
$ cp m_skin2.j3c model.j3c
$ j3cc objview.j3c
$ mv objview.j3d objskin2.j3d
$ j3w objskin.j3d
```

■終わりに

J3C言語を使った三次元アニメーションの作成法を一通り解説してきました。次回は最終回として、J3Wのネイティブな言語であるアセンブリでのプログラミングを紹介する予定です。

【表2】objviewディレクトリ

ファイル	内容
falcon02.j3c	F-16クラス
m_falcon.j3c	falcon02.j3c用のmodel.j3c
taco.j3c	タコの形状クラス
weed.j3c	海草クラス
m_taco.j3c	taco.j3cとweed.j3c用のmodel.j3c
lhand.j3c	左手クラス
m_hand.j3c	lhand.j3c用のmodel.j3c
body.j3c	POSEの人体クラス
m_body.j3c	body.j3c用のmodel.j3c
tubo.j3c	回転体で作成した壷クラス
m_tubo.j3c	tubo.j3c用のmodel.j3c

【表3】objview.j3cのクラスの動作

クラス	動作
MyModel	modelを継承して物体を生成する
	RUNとEVENTメソッドをオーバーライドしてmodelの動作を変更する
	RUNメソッドでインスタンス番号を設定して、メッセージを待つ
	EVENTメソッドでキー入力によるコマンドを実行する
Camera	インスタンス番号を設定する
	視点用に物体を生成する
	(-1000, 0, 0)に位置を設定する
	Seeを実行して視点を取得する
	メッセージを受信したら対応した動作をする
main	背景色の設定する
	フレームレート表示用のクラスを起動する
	MyModelを起動して物体を表示する
	視点用にカメラを生成する
	グラフックウィンドウを開く
	使用法を表示する
	キー入力からカメラや物体にメッセージを送信する
model	別に定義したクラスを継承する
	INITを再定義して位置を変更する
	RUN、EVENTはMyModelで再定義される

Column

J3Wの内部構造の解説

前回（2002年2月号）のコラムで、「シンボル管理に苦労した」と書きましたが、j3cコンパイラのソースのうちで、動作が分かりにくい部分を簡単に説明します。

■コンパイラのシンボル管理

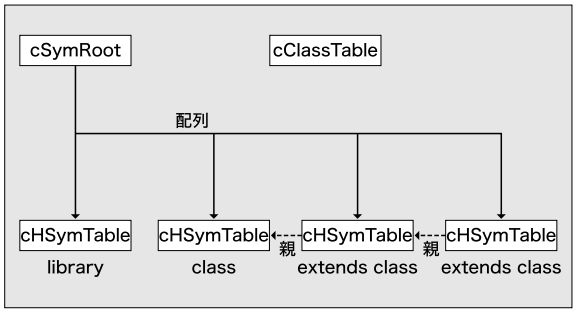
シンボル管理に関連する（C++の）クラスには、cClassTable、cSymTable、cHSymTable、cSymRootの4つがあります（図A）。紛らわしいので、これ以降の「クラス」という記述は、すべて「J3C言語におけるクラス」として説明していきます。

cClassTableが最も簡単で、宣言されたクラス名、ライブラリ名、インポートファイル名を構文解析時に登録しています。登録時に、インポートファイル名なら0、クラス名なら1、ライブラリ名なら2を種類別に設定しています。

cSymTableは、単一のクラスまたはライブラリ用のシンボル管理テーブルで、シンボル名、種類、値を保持します。シンボルには、ライブラリ名、クラス名、メソッド名、引数名、定数名、局所変数名の6種類があります。値は、変数の場合は変数のアドレス、定数の場合は、定数値をそのまま登録しています。局所変数名の場合はブロックのネストの数を登録します。cSymTableは、cHSymTableに継承されて、クラス名とスーパークラス（継承元）へのポインタを追加した形で使われます。

J3Cの複数のクラスやライブラリのシンボルを扱うため、cSymRootがcHSymTableへのポインタを配列として持つようになっています。cSymRootは、構文解析中のクラスのシンボルを管理しているcHSymTableへのポインタ（Current）を持っていて、実際のシンボルの登録、検索は、cSymRoot経由でのみ行われます。構文解析時に、新たなクラスに出会うとcSymRoot経由でcHSymTableを生成して、そのポインタを配列に登録し、さらにcClassTableに名前とクラスかライブラリの種別を登録します。クラスの定数、変数、メソッドのスコープ（名前の有効範囲）はcSymRootによって適切に管理されるため、構文解析側ではクラスの継承関係によるスコープを気にすることなくシンボルにアクセスできるようにしています。

【図A】シンボル管理に関連する（C++の）クラス



■コンパイラの組み込み関数処理

J3C言語の組み込み関数は、Cでいうところの「ライブラリ」としてリンクされるのではなく、1つのj3w（仮想CPU）の命令に翻訳されます。現在、115個の組み込み関数が用意されていますが、引数の数や引数の種類（レジスタまたは定数）、関数の結果を式で使うか、といった組み込み関数ごとの情報をj3wの命令に翻訳するために、組み込み関数の書式をコンパイラが知っている必要があります。この組み込み関数の書式処理はcParamFormat（j3c_func.h）が担当しています。

115種類の関数の書式は、関数ごとに32ビット整数で表現しています（ParamFormatTable）。これらは上位30ビットを2ビットずつ使って引数の種類を持っています。1は式、2は文字列、3は定数式を表し、最大15個の引数を指定しています。従って、引数の数に制限のない多角形の登録などの命令でも、最大引数は15に制限されるようになっています。最下位のビットは、引数を使わずレジスタ変数を入出力に使う命令を表していますが、実際には使っていません。下位の2ビット目は算術関数やInKeyなどの返り値のある関数の第1引数（レジスタ）を隠す目的で使用しています。

ソースコードを単語単位で読み込むscan->getsym()に注意して、記事末の構文図式を参考にcParse::parseから追っていくと、j3cの仕組みが理解できると思います。（水谷純）

J3C 構文リファレンス ②

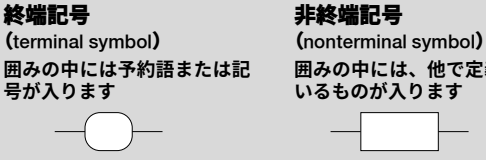
J3Cは、3Dアニメーションキット J3W のコンパイラです。J3Cは、Java風の文法を持つオブジェクト指向言語として設計されています。J3Cの構文リファレンスを掲載します。

構文図式の見方

■構文規則（syntactic rule）

コンピュータ言語の文法は、構文規則によって決定されます。構文は、左から右へ、上から下へ矢印をたどって読んでいきます。

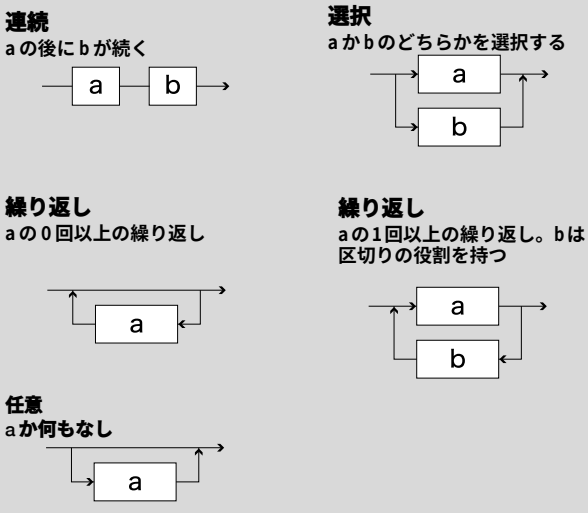
構文規則の左側にある名前のことを、「非終端記号(nonterminal symbol)」と呼びます。また、`class`や`for`などの予約語、＝や＋などの記号で表される単語そのもののことを「終端記号(terminal symbol)」と呼びます。終端記号と非終端記号は、構文図では次のように表します。



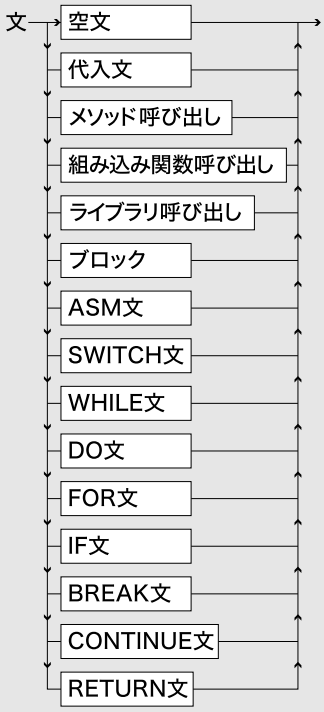
構文規則を順次適用していくと、すべての非終端記号は終端記号に行き着きます。

■構文図式の基本形

構文図式の基本形は次のようになっています。「連続」、「選択」、「繰り返し」を組み合わせて文法を記述します。



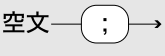
文



文としては、次の14種類が使用可能です。IF文以外は、C言語など他の言語に共通する文法となっています。

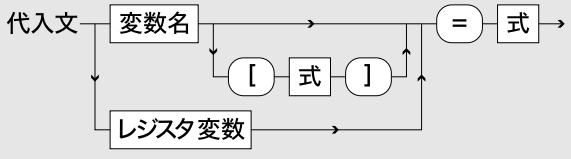
空文

文が必要な位置にセミコロンを置くと、何もしない空文とすることができます。



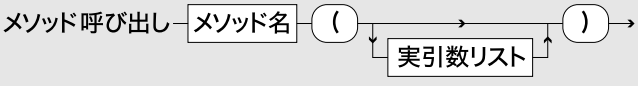
代入文

代入文は、左辺に用意した局所変数、データメンバおよびそれらの配列、またはレジスタ変数に、右辺の式の値を代入します。代入文は値を返しません。



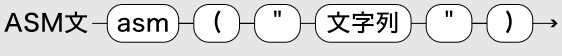
メソッド呼び出し

メソッドを文として呼び出した場合、返り値は無視されますが、代入文の右辺や式中で使用された場合には、メソッド中で最後に評価された式の値が返り値になります。

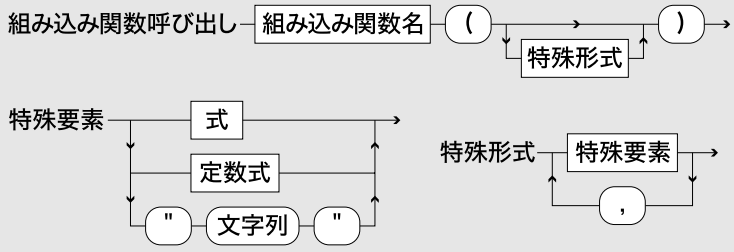


ASM文

ASM文で指定された文字列は、そのままj3cの出力としてアセンブラソースに埋め込まれます。j3dasmの入力として有効な文字列を指定する必要があります。



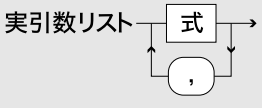
組み込み関数呼び出し

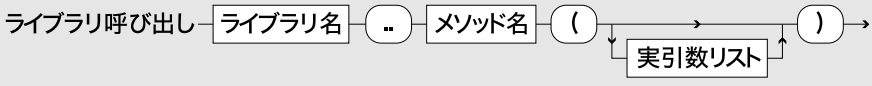


組み込み関数呼び出しは、仮想マシンJ3Wの機能を使用するためにJ3C内に組み込まれている関数です。引数は、通常のメソッドやライブラリの呼び出しとは異なり、形式は組み込み関数ごとに違います。

ライブラリ呼び出し

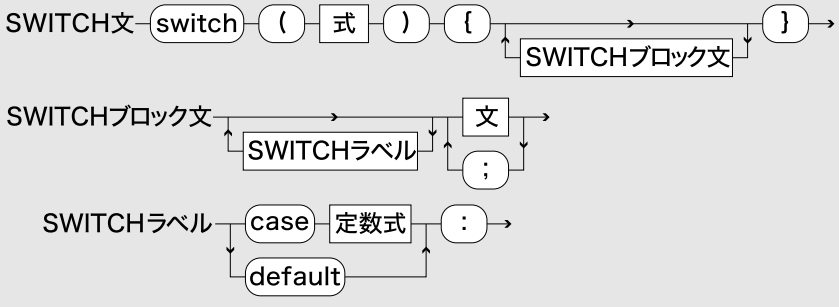
library宣言されているライブラリが持つメソッドを呼び出します。ライブラリ呼び出しは、ライブラリが宣言済みである限り、どのクラスまたはライブラリから参照することもできます(グローバルスコープを持つ)。実引数リストは、メソッド呼び出し、ライブラリ呼び出しのときの引数です。引数の数は0以上、任意の個数が指定可能です。





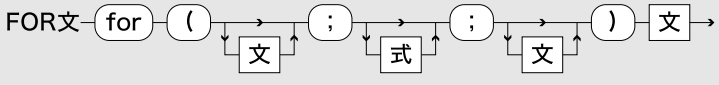
SWITCH文

SWITCH文は、C、C++、JAVAと同じように、式の値で分岐する場合に使用します。BREAK文を使用しない場合、分岐したラベル以降を上から下へ順に実行します。BREAK文が存在するとSWITCH文の実行は終了します。caseラベルに一致する値がなく、「default:」が存在する場合にはその部分が実行されます。



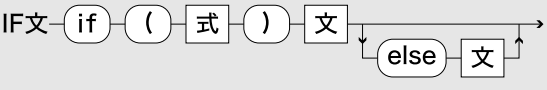
FOR文

FOR文は、決まった回数を繰り返します。FOR文に使用する変数(制御変数)は、事前に宣言しておく必要があります。また、加算は「i++」ではなく「i=i+1」という形式にする必要があります。



IF文

IF文だけが、C、C++、JAVAの文法とわずかに異なり、ELSE節の前にセミコロンを置くことができません。

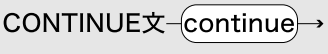


BREAK文

BREAK文は、最も内側のループ(FOR、DO、WHILE、SWITCH)から抜けます。BREAK文は、SWITCH文中でCASEの区切りとして使用します。

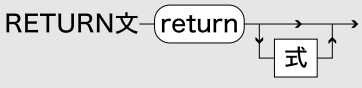
CONTINUE文

CONTINUE文は、ループ(FOR、DO、WHILE)内で、CONTINUE文の後の処理を飛ばして次のループの先頭から開始します。



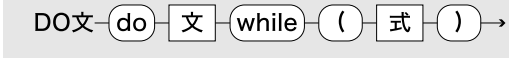
RETURN文

すべてのメソッドは、最後に評価した式の値が返ります。RETURN文は式を評価する目的で使用します。式の伴なわないRETURN文は何もしません。処理の流れを制御する目的では使用できません。メソッドの最後で返す値を明示するために使用します。メソッド定義の途中でRETURN文を使用してもメソッドは終了しません。



DO文

DO文は、C、C++、JAVAと同じように、繰り返し処理に使用します。式の値が真(非0)の間、繰り返し文が実行されます。WHILE文と異なり、文は少なくとも1度は実行されます。



WHILE文

WHILE文は、C、C++、JAVAと同じように、繰り返し処理に使用します。式の値が真(非0)の間、繰り返し文が実行されます。