

## ● 組み込み述語

組み込み述語は、ライブラリモジュールと異なり、モジュール名を指定しないで呼び出します。

---

### <alt 述語...>

省略可能な述語実行を示します。内部で呼び出された述語の実行結果に関わらずに必ずTrueで成功する述語です。  
"[述語...]"とも書けます。  
EBNF記法の[]として構文解析で使います。

<assert 節>  
<asserta 節>  
<assertz 節>

プログラムの1行の単位である節を追加します。  
assertaは先頭に追加し、assertzは最後に追加します。  
assertはassertaと同じで先頭に追加します。

### <cd パス>

カレントディレクトリを引数で指定したパスに移す。

<compare 比較式>  
<comparef 比較式>

比較式を評価。  
compareは整数、comparefは不動小数点数の数式の比較を行う。

### <dir>

カレントディレクトリを表示します。

<eachproc (要素変数 要素リスト) 並列実行する述語 ...>  
<eachproc 結果変数 (要素変数 要素リスト) 並列実行する述語 ...>

eachproc述語は、並列プロセス数を指定して並列実行する機能です。  
指定されたリストの要素のプロセスが並列に生成されます。  
eachprocで生成された全プロセスが終了すると、呼び出したeachproc述語に制御が戻り、全プロセスの結果を回収します。

<edit ファイル名>  
環境変数DEDITORPATHまたはEDITORに設定されたエディタを使ってファイル名を編集します。

### <eq 引数1 引数2>

引数1と引数2の比較し、同一ならばtrueになります。

### <erase 節名>

節名に該当するプログラム、変数、オブジェクトをすべて削除します。

<exit>

実行を途中で中断する。

<f 変数 述語リスト>

<func 変数 述語リスト>

述語リストの中に含まれている関数述語を実行評価して  
値を置き換えます。すべてを実行後に結果を変数に設定  
します。

<findall 述語...>

述語を実行して、すべての解を求めます。

<firsteachproc (要素変数 要素リスト) 並列実行する述語 ...>

<firsteachproc 結果変数 (要素変数 要素リスト) 並列実行する述語 ...>

firsteachproc述語は、並列プロセス数を指定して並列実行する機能です。  
指定されたリストの要素のプロセスが並列に生成されます。  
firsteachprocで生成された全プロセスが一つでもtrueで終了すると、  
呼び出したfirsteachproc述語に制御が戻り、最初に終了したプロセスの  
結果を回収します。残りのプロセスは処理を打ち切られます。

<firstfor (変数 実行回数) 述語...>

<firstfor (変数 初期値 最終値) 述語...>

<firstfor 結果変数 (変数 実行回数) 述語...>

<firstfor 結果変数 (変数 初期値 最終値) 述語...>

firstfor述語は、for文と同じように動作しますが、最初にtrueで終了した  
述語の結果を結果変数に設定して、以後の処理を打ち切ります。

<firstforeach (変数 リスト) 述語...>

<firstforeach 結果変数 (変数 リスト) 述語...>

firstforeach述語は、foreach文と同じように動作しますが、最初にtrueで終了  
した

述語の結果を結果変数に設定して、以後の処理を打ち切ります。

<firstnewproc (プロセス番号変数 プロセス数) 並列実行する述語 ...>

<firstnewproc (プロセス番号変数 プロセス番号初期値 プロセス番号終り値) 述語 ...>

<firstnewproc 結果変数 (プロセス番号変数 プロセス数) 並列実行する述語 ...>

<firstnewproc 結果変数 (プロセス番号変数 プロセス番号初期値 プロセス番号終り値)  
述語

...>

firstnewproc述語は、並列プロセス数を指定して並列実行する機能です。  
指定された数のプロセスが並列に生成されます。  
firstnewprocで生成されたプロセスが一つでもtrueで終了すると、  
呼び出したnewproc述語に制御が戻り、最初に終了したプロセスの  
結果を回収します。残りのプロセスは処理を打ち切られます。

<for (変数 実行回数) 述語...>

<for (変数 初期値 最終値) 述語...>

<for 結果変数 (変数 実行回数) 述語...>

<for 結果変数 (変数 初期値 最終値) 述語...>

指定された回数、引数の述語を実行します。  
変数には、順に数が設定されます。実行回数が指定された場合は0からの値が、初期値が指定された場合はその値から実行回数または最終値を超えない値まで、1ずつ増加させます。  
述語は1ターンの実行後にすべての変数のバインドがクリアされて最初から実行されます。  
結果変数が指定されている場合には、実行された最初の述語の第一引数を順に集めたものが結果変数にリストとして返されます

<foreach (変数 リスト) 述語...>

<foreach 結果変数 (変数 リスト) 述語...>

リストの要素ごとに引数の述語を実行します。  
変数には、リストの値が順に設定されます。  
述語は1ターンの実行後に実行中の変数のバインドがクリアされて最初から実行されます。  
結果変数が指定されている場合には、実行された最初の述語の第一引数を順に集めたものが結果変数にリストとして返されます

<help>

<help 項目名>

ヘルプマニュアルを表示する。  
引数なしで実行すると、組み込み述語とsysモジュールのライブラリの述語の一覧を表示する。

<include ファイル名>

ファイル名のライブラリからモジュールを読み込みます。

<is 項目1 項目2>

項目1と項目2をユニフィケーションし、同一になった場合にはtrueを返します。  
項目が変数の場合には、値の代入となります。

<let 変数 = 数式>

<letf 変数 = 数式>

<letc 変数 = 数式>

数式を計算します。  
左辺が変数の場合は、計算結果を代入します。  
左辺が数値の場合は、計算結果と等しいか判定します。  
letは、整数の計算を行い、letfは浮動小数点数の計算を行います。  
letcは複素数を計算します。

<list [変数]>

プログラムリストを表示します。

<load ファイル名>

ファイル名のプログラムを読み込みます。

<loop 述語...>

述語実行の繰り返しを示します。内部で呼び出された述語が失敗するとループを抜けます。述語は1ターンの実行後に実行中の変数のバインドがクリアされて最初から実行されます。述語の実行結果に関わらずに必ずTrueで成功する述語です。"{述語...}"とも書けます。EBNF記法の{}として構文解析で使います。

<ls>

カレントディレクトリを表示します。

<map (変数 リスト) 述語...>

リストの要素ごとに引数の述語を実行します。変数には、リストの値が順に設定されます。述語は1ターンの実行後に実行中の変数のバインドがクリアされて最初から実行されます。

<module 変数>

現在使用しているモジュールを変数に設定します。

<new>

プログラムをすべてクリアします。

<newproc (プロセス番号変数 プロセス数) 並列実行する述語 ...>

<newproc (プロセス番号変数 プロセス番号初期値 プロセス番号終り値) 述語 ...>

<newproc 結果変数 (プロセス番号変数 プロセス数) 並列実行する述語 ...>

<newproc 結果変数 (プロセス番号変数 プロセス番号初期値 プロセス番号終り値) 述語 ...>

newproc述語は、並列プロセス数を指定して並列実行する機能です。指定された数のプロセスが並列に生成されます。newprocで生成された全プロセスが終了すると、呼び出したnewproc述語に制御が戻り、全プロセスの結果を回収します。

<nop ...>

引数に対して何も操作をしません。この述語は何も操作しない(No Operation)です。常にtrueを返します。

<not 述語...>

述語実行がtrueの場合はfalseを返します。falseの場合はtrueを返します。unknownの場合はunknownを返します。

<noteq 引数1 引数2>

引数1と引数2の比較し、同一でなければtrueになります。

<obj オブジェクト名 述語...>

指定されたオブジェクトまたはモジュールを使って、述語を実行します。  
オブジェクトとモジュールは、同等のものとして扱われるためこの2つは同じものです。  
省略形として“::オブジェクト <述語...>”と記述できます。

<or 述語 述語 述語 ...>

引数の述語を先頭から実行し、trueになったところで処理を打ち切りtrueを返します。  
引数の述語を先頭から実行し、falseになったところで処理を打ち切りfalseを返します。  
すべての述語がunknownの場合は、unknownを返します。  
EBNF記法の | として構文解析で使います。

<print リスト>

リストを出力した後、改行します。

<writeln リスト>と同じである。

<printf リスト>

リストを出力します。  
printとの違いはリストの要素の間に空白が入らないことと自動的に改行しないことです。

<printlist リスト>

括弧を外して、リストを出力します。

<printlistnl リスト>

括弧を外して、要素毎に改行しながら、リストを出力します。

<pwd [変数]>

カレントディレクトリを変数に設定します。

<quit>

プログラムの実行を中止して終了します。

<quote 述語>

引数の関数述語の評価を抑止します。

<retract 節名>

節名に該当するプログラム、変数、オブジェクトをすべて削除します。

<retractpred ヘッド>

ヘッドに該当するプログラムをすべて削除します。

<rpn 変数 逆ポーランド式>  
<rpnf 変数 逆ポーランド式>  
<rpnf 変数 逆ポーランド式>

逆ポーランド式を計算して、変数に結果を設定します。  
rpnは、整数の計算を行い、rpnfは浮動小数点数の計算を行います。  
rpnfは複素数の計算を行います。

<save ファイル名>

ファイル名のプログラムを書き込みます。

<self 変数>

変数に自身のオブジェクト名を設定する。

<super 変数>

変数に継承しているオブジェクト名のリストを設定する。

<timeout 時間 述語>

設定した時間以内に、述語の実行が終了しない場合は  
処理を打ち切り、unknownを返します。  
設定時間はマイクロ秒単位です。

<troff>

デバッグトレースをオフにします。

<tron>

デバッグトレースをオンにします。

<true>

<false>

<unknown>

true, false, unknownを返す

<unify モジュール名 述語...>

指定されたオブジェクトまたはモジュールを使って、述語を  
実行します。  
オブジェクトとモジュールは、同等のものとして扱われるため  
この2つは同じものです。  
省略形として“::オブジェクト <述語...>”と記述できます。

<warn リスト>

リストをエラー出力に出力します。  
出力後に改行します。

<x 述語...>

通常は何も行わない。

バックトラックしたときに引数の述語を実行する。

<!>

カット演算子

<% \_ フォーマット 引数>

printfのためにフォーマットの整形をします。

<¥ \_ 文字>

エスケープ文字

EBNF 構文解析

<TOKEN 変数 述語...>

入力の構文解析述語実行後に、得られたtokenを変数に設定します。

<SKIPSPACE>

入力のスペースをスキップします。

<C [変数]>

入力を一文字変数に設定します。

<N [変数]>

入力が数字であった場合は、変数に設定します。  
違う場合はunknownを返します。

<A [変数]>

入力がASCII文字であった場合は、変数に設定します。  
違う場合はunknownを返します。

<AN [変数]>

入力がASCII文字か数字であった場合は、変数に設定します。  
違う場合はunknownを返します。

<^>

行の先頭とマッチする。

<\$>

行の最後とマッチする。

<\* [変数]>

任意の文字列とマッチする。

<CR>

入力がCR改行であった場合には、trueを返します。  
違う場合はunknownを返します。

#### <CNTL [変数]>

入力がCNTL文字であった場合には、変数に設定します。  
違う場合はunknownを返します。

#### <EOF>

入力がEOF (End Of File) である場合はtrueを返します。  
違う場合はunknownを返します。

#### <SPACE>

入力がスペースである場合はtrueを返します。  
違う場合はunknownを返します。

#### <PUNCT>

アルファベット、数字以外の文字である場合はtrue  
を返します。

#### <WORD [変数]>

任意の文字列で、アルファベット、数字、“\_”以外  
の文字列の場合はunknownを返します。

#### <NUM [変数]>

入力の整数を変換して、変数に設定します。

#### <FNUM [変数]>

入力の浮動小数点数を変換して、変数に設定します。

#### <GETTOKEN 変数>

直前の構文解析の結果であるトークンを変数に設定  
します。

#### <ID [変数]>

入力の文字列（先頭はアルファベット、それ以外  
は数字も可）、合致すれば変数に設定します。

#### <INDENT 変数>

カレント行のインデントのスペースの長さを変数に設定します。  
タブは、8文字と数えます。マルチバイト文字の空白は2文字と数えます。  
引数に変数ではなく、数の場合は、カレント行のインデントのスペースの長さ  
と比較し一致していればtrueになります。

#### <RANGE 変数 文字 1 文字 2>

#### <NONRANGE 変数 文字 1 文字 2>

文字 1 と文字 2 の範囲に含まれるならばtrue  
となります。



#### <NEXTCHAR 変数>

1文字先読みして、値を変数に設定します。

#### <NEXTINDENT 変数>

カレント行の次の行のインデントのスペースの長さを変数に設定します。タブは、8文字と数えます。マルチバイト文字の空白は2文字と数えます。引数が変数ではなく、数の場合は、カレント行の次の行のインデントのスペースの長さと比較し一致していればtrueになります。

#### <NEXTSTR 文字列>

先読みして、文字列と等しいか判定します。

#### <NEXTSYNTAX プログラム>

先読みして、プログラムを実行します。  
実行後には、ファイルの読み込み位置は開始時の位置に戻ります。

#### <NOTNEXTSTR 文字列>

先読みして、文字列と等しいか判定します。  
等しくないときにtrueとなります。

#### <NOTNEXTSYNTAX プログラム>

先読みして、プログラムを実行します。  
構文解析が失敗したときにtrueとなります。  
実行後には、ファイルの読み込み位置は開始時の位置に戻ります。

#### <NULLLINE>

空行かどうか判定します。

#### <SKIP 文字列>

文字列までの構文解析を行わずにスキップします。

#### <SKIPCR>

改行までの構文解析を行わずにスキップします。

### オブジェクトの操作

#### <newObj 名前>

新しい名前のオブジェクトを生成する。

#### <cloneObj 新オブジェクト 名前>

名前のオブジェクトを複製して新オブジェクトを生成する。

#### <delObj 名前>

名前のオブジェクトを削除する。

<method 名前>

新しいメソッドを名前で作成する

<methoda 名前>

新しいメソッドを作成して、オブジェクトの先頭に追加する

<methodz 名前>

新しいメソッドを作成して、オブジェクトの末尾に追加する。

<delmethod 名前>

指定された名前のメソッドを削除する。

<delmethoda 名前>

指定された名前の先頭のメソッドを削除する。

<delmethodz 名前>

指定された名前の末尾のメソッドを削除する。

<setVar 変数名 値>

グローバル変数に値を設定します。  
グローバル変数は、述語として以下の形式で登録  
されます。  
<変数名 値>;

<setArray 変数名 値 インデックス>

グローバル変数配列に値を設定します。  
グローバル変数は、述語として以下の形式で登録  
されます。  
<変数名 値 インデックス>;

<delVar 変数名>

変数名に合致する変数が削除されます。

<delArray 変数名 INDEX>

変数名に合致する配列変数が削除されます。

<変数名 値 インデックス>;

多次元の配列を仕様する場合は、インデックスを  
リストとして指定します。

<変数名 値 (インデックス1 インデックス2 ...)>

インデックスは数以外にもいかなる種類や形式で  
も良いです。

以上。