デカルト言語のプログラム例：calc

この文書では、デカルト言語のプログラム例として式を計算する calc プログラムを取り上げ説明する。

- calc プログラムは標準入力から入力された計算式を計算して結果を標準出力に結果を表示する。
- 計算式に使えるのは数値(浮動小数点数)、演算子(+−*/)、括弧()である。
- 計算式は逆ポーランド記法に変換してから、rpnf 述語で実行される。

1. プログラムのコーディングスタイル

構文解析を行うプログラムでは、以下に示すように構文部分と操作部分を分けたコーディングスタイルで記述すると分かりやすいプログラムが書ける。

構文部分	操作部分
〈構文ヘッド〉	
〈構文ボディ〉	〈操作述語〉

例)

〈文〉

〈文1〉

〈文 1 操作・出力〉

〈文2〉

〈文 2 操作・出力〉

;

2. calc の戦略

calc は、人間の入力する文字列で表現された計算式を構文解析して、要素ごとに分解した後に、計算しやすいように要素を組み合わせて再構成して計算を実行する。

具体的には、以下のような動作を実施する。

1) 計算式を1行入力する。
2) 入力された計算式を構文解析して、逆ポーランド記法に変換する。
3) 変換された式を rpnf 述語の引数として述語を合成する。
4) 合成した述語を実行する。
5) 「1. から」繰り返す

3. 構文の定義

calc の構文を EBNF(拡張バッカス記法)で記述すると以下のようになる。

EBNF(拡張バッカス記法)による構文

```
expr        =  expradd
expradd     =  exprmul { "+"  exprmul  |  "-"  exprmul }
exprmul     =  exprID  { "*"  exprID  |  "/"  exprID }
exprID      =  "+" exprterm | "-"  exprterm  |  exprterm
exprterm    =  "("  expr  ")" | 数字列
```

EBNF(拡張バッカス記法)の構文を、デカルト言語によって、以下のように書き直す。ほぼ一対一に対応して変換できることがわかる。

デカルト言語による構文

```
<expr>      <expradd>;
<expradd>   <exprmul> { "+" <exprmul> | "-" <exprmul> };
<exprmul>   <exprID> { "*" <exprID> | "/" <exprID> };
<exprID>    "+" <exprterm> | "-" <exprterm> | <exprterm> ;
<exprterm>  "(" <expr> ")" | ::sys <FNUM #t> ;
```

この構文では、演算子の優先度は以下の順になる。

```
高い        ( )
            単項+, 単項-
            * , /
低い        + , -
```

4. 行入力

キーボードより1行入力するためには sys モジュールの getline 述語を使う。

::sys 〈getline #line 呼出述語〉

#line に入力した行が設定され、それを呼出述語の入力ファイルとして呼び出す。
前章で作成した〈expr〉述語を呼び出す場合は以下のように記述する。

::sys 〈getline #line 〈expr〉〉

5. グローバル変数

途中の演算結果を保存するのにグローバル変数を使用する。
グローバル変数に値を設定するには、setvar 述語を使う。

::sys 〈setvar 変数名 設定値〉

グローバル変数から値を取り出すには以下のようにする。

〈変数名 #変数〉

グローバル変数名の値が、#変数に設定される。

6. ライブラリ append 述語の呼び出し

::list 〈append 連結リスト変数 リスト1 リスト 2〉

リスト 1 とリスト 2 を連結して、連結リスト変数に設定される。

7. 逆ポーランド記法演算

〈rpn 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、変数に結果を設定する。
rpn は、整数の計算を行い、rpnf は浮動小数点数の計算を行う。

8. calc の完成ソース

```
?<include list>;     // list ライブラリのインクルード

<calc #result>
        <print "calc : ">   // プロンプトの表示
        // 1行入力して構文解析<expr>を実行する
        // 結果はグローバル変数 exprlist に設定される
        ::sys <getline #line
                ::sys <setvar exprlist ()> // グローバル変数の初期化
                <expr>
        >
        // 結果を exprlist から取り出して rpnf で演算し結果を表示する
        <exprlist #x>
        <rpnf #result #x>
        <print " = "  #result>
        ;
<expr>
        <expradd>
        ;
<expradd>
                <exprmul>
                {  "+" <exprmul> // +演算子と合致
                                // exprlist に+を追加する
                                <exprlist #x>
                                ::list <append #list #x ("+")>
                                ::sys <setvar exprlist #list>
                 |
                   "-" <exprmul> // -演算子と合致
                                // exprlist に-を追加する
                                <exprlist #x>
                                ::list <append #list #x ("-")>
                                ::sys <setvar exprlist #list>
                }
        ;
```

```
<exprmul>
                <exprID>
                {     "*" <exprID> // *演算子と合致
                                    // exprlist に*を追加する
                                    <exprlist #x>
                                    ::list <append #list #x ("*")>
                                    ::sys <setvar exprlist #list>
                  |
                    "/" <exprID>// /演算子と合致
                                    // exprlist に/を追加する
                                    <exprlist #x>
                                    ::list <append #list #x ("/")>
                                    ::sys <setvar exprlist #list>
                }
        ;
<exprID>
          "+" <exprterm>
         |
          "-" <exprterm>
                                    <exprlist #x>
                                    ::list <append #list #x ("-1" "*")>
                                    ::sys <setvar exprlist #list>
         |
          <exprterm>
        ;
<exprterm>
          "("  <expr>  ")"
        |
          ::sys <FNUM #t>
                                    <exprlist #x>
                                    ::list <append #list #x  (#t)>
                                    ::sys <setvar exprlist #list>
        ;
?{{ <calc #line> }};  //  全体を実行する。{}で 2 重に囲むのは、エラーで失敗しても終了させないため。
```

実行例:

```
$ descartes calc
calc :
10+20*30/(2+1)
 =  210
calc :
```