The example of a program of the Descartes language: calc

This document takes up and explains the calc program which calculates a formula as an example of a program of the Descartes language.

- A calc program calculates the formula inputted from standard input, and displays a result for a result on standard output.
- Numerical value (floating point number), operator (+-*/), and parenthesis () can be used for a formula.
- After changing a formula into a reverse Polish notation, it is performed in a rpnf predicate.

1. Coding Style of Program

In the program which analyzes syntax, if the coding style which divided a part for a syntax portion and a final controlling element describes as shown below, an intelligible program can be written.

```
       Syntax portion          Operation portion
_____________________________ _______________

<HEAD >
       <BODY>                  <Operation>

Example:
<Pred>
       <Pred1>
                               <Pred1 opertion>
       <Pred2>
                               <Pred2 operaition>
       ;
```

2. Strategy of Calc

After analyzing the syntax of the formula expressed by the character string which man inputs and decomposing for every element, calc is reconstructed combining an element and performs calculation so that it may be easy to calculate. Specifically, the following operation is carried out.

1) Input the formula of one line.
2) Analyze the syntax of the inputted formula and change into a reverse Polish notation.
3) Compound a predicate for the changed formula as an argument of a rpnf predicate.
4) Perform the compound predicate.
5) "From 1." It repeats.

3. Definition of Syntax

It is as follows when EBNF (extended Backus Naur form) describes the syntax of calc.

Syntax of EBNF(extended Backus Naur form)

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

The syntax of EBNF (extended Backus Naur form) is rewritten as follows with the Descartes language. It turns out that it is convertible corresponding to about 1 to 1.

Syntax by the Descartes language

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

By this syntax, the priority of a operator becomes the following order.

```
HIGH      ( )
          Unary +, unary -
          * , /
LOW       + , -
```

4. Line Input

In order to input one line from a keyboard, the getline predicate of a sys module is used.

```
::sys <getline #line Called-predicate >
```

#line inputted into line is set up and call it as an input file of a call predicate.
When calling the <expr> predicate created for the preceding chapter, it describes as follows.

```
::sys <getline #line <expr>>
```

5. Global Variable

A global variable is used for saving an intermediate operation result.
A setVar predicate is used in order to set a value as a global variable.

```
<setVar VAR VALUE>
```

It is performed as follows for taking out a value from a global variable.

```
<VAR #VAR>
```

The value of a global variable name is set as #VAR.

6. Call of Library Append Predicate

```
::list <append #VAR LIST1 LIST2>
```

LIST1 and LIST2 are connected and it is set as #VAR.

7. Reverse Polish Notation Operation

<rpn VAR RPN >
<rpnf VAR RPN >

A reverse Poland style is calculated and a result is set as a variable.
rpn calculates an integer and rpnf calculates a floating point number.

8. Completion Sauce of Calc

```
?<include list>;    // Include of a list library

<calc #result>
       <print "calc : ">   // The display of a prompt
       // One line is inputted and syntax analysis <expr> is performed.
       // A result is set as global variable exprlist.
       ::sys <getline #line
              <setVar exprlist ()>
              <expr>
       >
       // A result is taken out from exprlist,
       // it calculates by rpnf, and a result is displayed.
       <exprlist #x>
       <rpnf #result #x>
       <print " = "  #result>
       ;
<expr>
       <expradd>
       ;
<expradd>
              <exprmul>
              {  "+" <exprmul> // + operator
                           // + is added to exprlist.
                           <exprlist #x>
                           ::list <append #list #x ("+")>
                           <setVar exprlist #list>
               |
                 "-" <exprmul> // - operator
                           // - is added to exprlist
                           <exprlist #x>
                           ::list <append #list #x ("-")>
                           <setVar exprlist #list>
              }
       ;
```

```
<exprmul>
             <exprID>
             {   "*" <exprID> // * operator
                         // * is added to exprlist
                         <exprlist #x>
                         ::list <append #list #x ("*")>
                         <setVar exprlist #list>
               |
                  "/" <exprID>// / operator
                         // / is added to exprlist
                         <exprlist #x>
                         ::list <append #list #x ("/")>
                         <setVar exprlist #list>
             }
       ;
<exprID>
         "+" <exprterm>
        |
         "-" <exprterm>
                         <exprlist #x>
                         ::list <append #list #x ("-1" "*")>
                         <setVar exprlist #list>
        |
         <exprterm>
       ;
<exprterm>
         "("  <expr>  ")"
       |
         <FNUM #t>
                         <exprlist #x>
                         ::list <append #list #x  (#t)>
                         <setVar exprlist #list>
       ;
?{{ <calc #line> }};
```

Example of execution :

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