6903. ポーランド記法変換*

大駒誠一，中西正和
(慶応義塾大学工学部)

できるだけ簡単に数式をポーランド記法に変換することを考え，LISP を使って次のような方法でプログラムをした．

すなわち

(被演算子　演算子
　被演算子)　　(1)

というパターンを

被演算子　被演算子
　演算子　　(2)

というパターンに置きかえるだけである．被演算子はまた(1)の形でありうる．

そのかわり変換すべき数式は各演算子ごとに一対の括弧をつけておかなければならない．こうすると演算子の強さを考慮にいれる必要がなくなりプログラムが非常に簡単になる．

この例では逆ポーランド記法に変換しているが，(2)で演算子を左端にすれば正ポーランド記法になることはいうまでもない．AXLE[3]ではこれと同様のことを簡単な2つのステートメントでやっているが（そのうち1つは end ステートメント），そのかわり長ったらしい Assertion Table を必要としている．

TRACE 中の

n ARGUMENTS OF POLISH

n VALUE OF POLISH

というのは POLISH 関数が *n* レベルで再帰的に呼ばれたときの引数と値をその下に示している．

```
*JOB
*OPCONTROL:NUMTRACE
POLISH[X] = [ATOM[X]->CONS[X;NIL];
        T->APPEND[POLISH[CAR[X]];APPEND[POLISH[CADDR[X]];CONS[CADR[X];NIL]]]].
POLISH[(A + B)];
POLISH[((A + B) * C)];
POLISH[(1 / (((1 - (Z ** 2)) * (1 - ((K * Z) ** 2))) ** 0.5))];
TRACE[(POLISH)];
POLISH[(X = (Y - (3.14159 * Z)))].

 FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
POLISH
((A + B))

 END OF EVALQUOTE, VALUE IS..
(A B +)

 FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
POLISH
(((A + B) * C))

 END OF EVALQUOTE, VALUE IS..
(A B + C *)

 FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
POLISH
(( 1 / ((( 1 - (Z **  2)) * ( 1 - ((K * Z) **  2))) **  0.5
      )))

 END OF EVALQUOTE, VALUE IS..
( 1  1 Z  2 **  -  1 K Z *  2 **  -  *  0.5 ** /)

 FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
TRACE
((POLISH))

 END OF EVALQUOTE, VALUE IS..
NIL

 FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
POLISH
((X = (Y - ( 3.14159 * Z))))

 0  ARGUMENTS OF POLISH
(X = (Y - ( 3.14159 * Z)))

 1  ARGUMENTS OF POLISH
X

 1  VALUE OF POLISH
(X)

 1  ARGUMENTS OF POLISH
(Y - ( 3.14159 * Z))

 2  ARGUMENTS OF POLISH
Y

 2  VALUE OF POLISH
(Y)

 2  ARGUMENTS OF POLISH
( 3.14159 * Z)

 3  ARGUMENTS OF POLISH
3.14159

 3  VALUE OF POLISH
( 3.14159)

 3  ARGUMENTS OF POLISH
Z

 3  VALUE OF POLISH
(Z)

 2  VALUE OF POLISH
( 3.14159 Z *)

 1  VALUE OF POLISH
(Y  3.14159 Z * -)

 0  VALUE OF POLISH
(X Y  3.14159 Z * - =).

 END OF EVALQUOTE, VALUE IS..
(X Y  3.14159 Z * - =)

RUN TIME   0 MIN     820 MSEC

     **** END OF KLISP JOB! ****

((POLISH) (A B +) (A B + C *) ( 1  1 Z  2 **  -  1 K Z *  2
      **  -  *  0.5 **  /) NIL (X Y  3.14159 Z * - =))

JOB TIME   0 MIN   15540 MSEC
```

定義式

評価式

TRACE しない場合

以下 TRACE の結果

KLISP[2] ではプログラムを M-式で書いても S-式で書いても受けつけてくれる.

参考文献

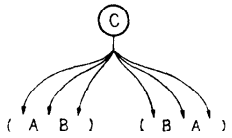
1) J. McCarthy: "LISP 1.5 Programmer's Manual", The M. I. T. Press.
2) 中西正和: "KLISP 説明書", 慶応義塾大学工学部計算センター
3) K. Cohen and J. H. Wegstein: "An Axiomatic Language for String Transformations" Comm. ACM, Vol. 8, No. 11 (1965)

(昭和43年11月4日受付)

6904. 順列の生成*

大駒誠一, 中西正和
(慶応義塾大学工学部)

LISP を使った順列を作るプログラムである. 順列の作り方にはいろいろあるが, ここでは n 個の相異なる要素のすべての順列は, n-1 個の順列を作って, その各順列の要素と要素との間および各順列の両端に n 個目の要素をつけ加えるという方法で作っている (**第1図**).



第1図 $n=3$ の場合

このために補助関数を2つ定義しているが, その機能は TRACE の結果を見ればあきらかであろう.

このプログラムでは数をかぞえることとプログラム機能はどうも LISP らしくないので使わないというのを大原則として作った.

実際の順列の生成に要した時間は, TOSBAC-3400 (16K 語) KLISP システムを使って, 3要素のとき 0.36秒, 4要素のとき 1.84秒, 5要素のとき Garbage Collector が1回働いて 8.80秒であった. 6要素のときは Garbage Collector が13回働いたが, 記憶装置がパンクしてできなかった.

TRACE 中で

n ARGUMENTS OF func

n VALUE OF func

は func という関数が n レベルで再帰的に呼ばれたときの引数と値をその下に示している.

参考文献

1) J. McCarthy: "LISP 1.5 Programmer's Manual", The M. I. T. Press.
2) 中西正和: "KLISP 説明書" 慶応義塾大学工学部計算センター

(昭和43年11月4日受付)