デカルト言語の概要

1. prologとの類似性

デカルト言語は、1階述語論理を基にしているため、prolog言語
と多くの共通点を持ちます。
そこで、リストの連結するプログラムでデカルト言語とprolog
言語を比較してみましょう。

デカルト言語では以下のようになります。

```
<append #Z () #Z>;
<append (#W : #Z1) (#W : #X1) #Y> <append #Z1 #X1 #Y>;
```

prologでは以下のようになります。

```
append(Z, [], Z).
append( [W | Z1], [W | X1], Y) :- append(Z1, X1, Y).
```

どちらの例でも、第2引数と第3引数のリストを連結し、
第1引数に結果を返します。

違いがお分かりでしょうか。
1) 述語は<>で括られる。
2) 引数の区切りは空白を使う。
3) prologでは最後にピリオド"."を置くが、デカルト言語で
   はセミコロン";"を置く。
4) リストは[]ではなく()を使う。
5) リストを分割する"|"が、デカルト言語では":"である。
6) ヘッド部とボディ部の区切りにprologでは":-"を使うが
   デカルト言語では何も無い。
7) デカルト言語では変数には"#"が付く。

デカルト言語で、上記appendを実行するには、次のように
?を付けた述語で実行します。

```
?<append #x (a b) (c d)>;
```

2. 注釈コメント
注釈(コメント)には次の3種類があります。
- //から行末まで
- #から行末まで
- /* */に囲まれた範囲

3. 数値計算

数の計算には、整数ではlet述語, 浮動小数点ではletf述語を
使います。

```
<let  #x = 1 + 2>;
<letf #f = 1.1 + 0.3*(2.3-1.2)>;
```

数式内では、後述する関数述語が使えます。

```
    <letf #f = <sin #x1 3.14>+<cos #x2 3.14>>;
```

letは省略することができます。以下は２つとも同一です。

```
    <let #z = #x + #y>;
    <#z = #x + #y>;
```

注) letfは省略できません。

4. 関数述語

let, letf, f, funcなど述語の引数は関数述語として評価され、
関数述語の返す関数値は第1引数です。
返り値の変数は、無名変数"_"を使うと便利です。

```
    <letf #x = <sin _ <cos _ 3.14>>>;
```

let, letfの中で使えるのは、数値を返す関数のみです。

```
    <f #x <car #x1 <cdr _ (a b c)>>>;
```

fはfuncの別名であり、まったく同じ働きをします。
また、fは引数としてリストを取ることが可能であり、リスト
の要素に関数述語が含まれる場合は、すべて評価された後に
関数値として返されます。

```
    <f #x (This is a ::sys <getline _>)>;
```

(上記で"::sys"とあるのは後述するライブラリの呼び出しを
表し、sysモジュールのgetline述語の呼び出しを意味します。)

5. ライブラリ

ライブラリの呼び出しは以下のような形式で行います。

::ライブラリモジュール名 〈述語〉
<unify ライブラリモジュール名 〈述語〉>
<obj ライブラリモジュール名 〈述語〉>

これら3種類の呼び出し方法は同一の動作内容です。

6. オブジェクト指向

オブジェクトは以下のような形式で定義します。

```
:: < オブジェクト名
        プログラムまたはinheirt 継承オブジェクト
>;
```

例として鳥、ペンギン、鷹のオブジェクト例を以下に
示します。

```
::<bird
        <fly>;
        <walk>;
>;
```

```
::<penguin
        <fly>
                <!><false>;
        <swim>;
        inherit bird;
>;

::<hawk
        inherit bird;
>;
```

オブジェクトの呼び出し方は、ライブラリの呼び出し方法
と同じです。以下を試してみてください。

```
?::bird <swim>;
?::penguin <swim>;
?::bird <walk>;
?::penguin <walk>;
?::bird <fly>;
?::penguin <fly>;
?::penguin <run>;

?::hawk <fly>;
?::hawk <walk>;
?::hawk <swim>;
```

7. EBNF記法

構文解析のためのEBNF記法が使えます。

```
<名前>  "田中";
<名前>  "佐藤";

<name #x>
        "私" "は"
        <名前> ::sys<GETTOKEN #x>
        "です"
        ["。"]
        ;

? ::sys<getline _ <name #name>>;
私は佐藤です。

result --
(<obj sys <getline 私は佐藤です。 <name 佐藤>>>)
-- true
```
