CLU 言語入門　＃ 2

久野　靖

1993.5.17

今回は本当は配列やレコードをやりたいのだけれど、まだ基本型の話が終っていない。それより前に、反復子の話をしないといけない。まあ「データ型序説」には型検査とかでいろいろやっているうちに追い付くであろう。

1　反復子 — 目新しい (?) 制御構造

前回やった構文の中に for 文がないなあ、と思った人はいませんか。実は CLU にもちゃんと for 文はある。ところで、一般の (皆様が知っている) 言語における for 文というのは要するに何をするためのものだろうか?

典型的には Pascal の for 文のように「整数変数 i を x から y まで 1 きざみで変えながらループする」ためのものである。ところでなんで「整数」だけの特別扱いなのか?　「数えるといえば整数だから」といえばそうなのだが…

C の for 文は整数を特別扱いする代わりに、<初期設定，終了条件，更新>の組を記述するからより一般的だが、その代わり整数で数える場合にはちょっと煩わしい。Lisp の do もこの精神。ただし Lisp には dotimes という Pascal の for みたいなのもある。ところで Lisp には dolist というのがあるが… これはリストの各要素を列挙しながらループするもので、とっても便利。

そうしてみると、**様々なデータ構造 (ないしデータ型) ごとに、それに関連した「便利な繰り返し方」というのがある**のではないか?　そういう考えのもとに CLU では反復子 (iterator) というのが導入されている。そして for 文は反復子を呼び出すための文、ということになっている。

```
for 変数:型指定,... in 反復子名 (式,...) do
    文...
end
```

で、いちばんよく使う反復子はやっぱり int$from_to(x, y) でしょう。これは Pascal の for と同じ。

```
% sam04.clu -- use of iterator

start_up = proc()
  pi:stream := stream$primary_input()
  for i:int in int$from_to(3, 8) do
      stream$putl(po, int$unparse(i))
  end
end start_up
```

なお、「きざみ」を指定できる from_to_by というのもある。これらを併せて、実は int の操作は次のようになる。

```
int$add = proc(int,int) returns(int) signals(overflow)
int$sub = proc(int,int) returns(int) signals(overflow)
int$mul = proc(int,int) returns(int) signals(overflow)
int$minus = proc(int) returns(int) signals(overflow)
int$div = proc(int,int) returns(int) signals(zero_divide,overflow)
int$mod = proc(int,int) returns(int) signals(zero_divide,overflow)
int$power = proc(int,int) returns(int) signals(negative_exponent,overflow)
int$abs = proc(int) returns(int) signals(overflow)
int$max = proc(int,int) returns(int)
int$min = proc(int,int) returns(int)
int$from_to = iter(int,int) yields(int)
int$from_to_by = iter(int,int,int) yields(int)
int$parse = proc(string) returns(int) signals(bad_format,overflow)
int$unparese = proc(int) returns(string)
int$lt = proc(int,int) returns(bool)
int$gt = proc(int,int) returns(bool)
int$le = proc(int,int) returns(bool)
int$ge = proc(int,int) returns(bool)
int$equal = proc(int,int) returns(bool)
int$similar = proc(int,int) returns(bool)
int$copy = proc(int) returns(int)
```

2 CLU の基本型

2.1 null 型

null 型は nil というただ 1 つの値のみを持つ。何に使うか分かりますか?

```
null$equal = proc(null,null) returns(bool)
null$similar = proc(null,null) returns(bool)
null$copy = proc(null) returns(null)
```

2.2 bool 型

論理型。値は true と false の 2 つだけ。

```
bool$and = proc(bool,bool) returns(bool)
bool$or = proc(bool,bool) returns(bool)
bool$not = proc(bool) returns(bool)
bool$equal = proc(bool,bool) returns(bool)
bool$similar = proc(bool,bool) returns(bool)
bool$copy = proc(bool) returns(bool)
```

なお、cand と cor は「bool 型の操作」ではない特別なもの。理由は前回やりましたね?

2.3 int 型

前述の通り。整数は 1 種類しかない。

2.4 real 型

もちろん、実数型。実数も 1 種類しかない。

```
real$add = proc(real,real) returns(real) signals(overflow,underflow)
real$sub = proc(real,real) returns(real) signals(overflow,underflow)
real$mul = proc(real,real) returns(real) signals(overflow,underflow)
real$minus = proc(real) returns(real)
real$div = proc(real,real) returns(real)
            signals(zero_divide,overflow,underflow)
real$power = proc(real,real) returns(real)
              signals(zero_divide,compex_result,overflow,underflow)
real$abs = proc(real) returns(real)
real$max = proc(real,real) returns(real)
real$min = proc(real,real) returns(real)
real$i2r = proc(int) returns(real) signals(overflow)
real$r2i = proc(real) returns(int) signals(overflow)
real$trunc = proc(real) returns(int) signals(overflow)
real$exponent = proc(real) returns(int) signals(undefined)
real$mantissa = proc(real) returns(real)
real$parse = proc(string) returns(real) signals(bad_format,overflow,underflow)
real$unparese = proc(real) returns(string)
real$lt = proc(real,real) returns(bool)
real$gt = proc(real,real) returns(bool)
real$le = proc(real,real) returns(bool)
real$ge = proc(real,real) returns(bool)
real$equal = proc(real,real) returns(bool)
real$similar = proc(real,real) returns(bool)
real$copy = proc(real) returns(real)
```

2.5 char 型

文字型。リテラルは'' で囲む。

```
char$i2c = proc(int) returns(char) signals(illegal_char) -- コードから文字
char$c2i = proc(char) returns(int) -- 上の逆
char$lt = proc(char,char) returns(bool)
char$gt = proc(char,char) returns(bool)
char$le = proc(char,char) returns(bool)
char$ge = proc(char,char) returns(bool)
char$equal = proc(char,char) returns(bool)
char$similar = proc(char,char) returns(bool)
char$copy = proc(char) returns(char)
```

2.6 string 型

文字列型。リテラルは""で囲む。

```
string$size = proc(string) returns(int) -- 長さ
string$empty = proc(string) returns(bool) -- 長さ 0 か?
string$indexs = proc(string,string) returns(int) -- 検索
string$indexc = proc(string,char) returns(int) -- 検索
string$concat = proc(string,string) returns(string) -- 連結
string$append = proc(string,char) returns(string) -- 文字連結
string$fetch = proc(string,int) returns(char) -- N 文字目
string$rest = proc(string,int) returns(string) signals(bounds) -- N 文字目以降
string$substr = proc(string,int,int) returns(string)
                 signals(bounds,negative_size) -- 部分文字列
string$s2ac = proc(string) returns(array[char]) -- 配列との相互変換
string$ac2s = proc(array[char]) returns(string)
string$s2sc = proc(string) returns(sequence[char]) -- 列との相互変換
string$sc2s = proc(sequence[char]) returns(string)
```

```
string$chars = iter(string) yields(char)
string$lt = proc(string,string) returns(bool)
string$gt = proc(string,string) returns(bool)
string$le = proc(string,string) returns(bool)
string$ge = proc(string,string) returns(bool)
string$equal = proc(string,string) returns(bool)
string$similar = proc(string,string) returns(bool)
string$copy = proc(string) returns(string)
```

3 反復子を用いたもう少しまともな例

```
% sam05.clu -- twice each line

start_up = proc()
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  while ~stream$empty(pi) do
    line:string := stream$getl(pi)
    for c:char in string$chars(line) do
      stream$putc(po, c)
      stream$putc(po, c)
    end
    stream$putc(po, '\n')
  end
end start_up

% sam06.clu -- reverse each line

start_up = proc()
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  while ~stream$empty(pi) do
    line:string := stream$getl(pi)
    for i:int in int$from_to_by(string$size(line), 1, -1) do
      stream$putc(po, line[i])
    end
    stream$putc(po, '\n')
  end
end start_up
```

練習問題 次のような CLU プログラムを書け。

- 「cat -v」のように、入力の各行に一連番号を付加するフィルタ
- 各行のあたまの空白を全部取り除くフィルタ
- 各行のあたまの空白をすべて 2 倍にするフィルタ