CLU 言語入門　＃ 3

久野　靖

1993.5.25

1　CLU の組み込みの構造型

さて、前回輪読でやった構造型ですが、CLU にももちろんある。まずは配列から。CLU では配列は

```
array[T]
```

という型。T のところには要素の型が入る。例えば array[int]、array[string]、などなど。ということは、添字の型を指定する余地がない。添字は常に int 型ということになっている。そして、大きさも指定する余地がない。ということは、大きさも可変になっている。さっそく例題。

```
% sam07.clu -- reverse file upside-down

start_up = proc()
  as = array[string]
  a:as := as$new()                % 下限 1 大き 0 の配列 を作る
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  while ~stream$empty(pi) do
    as$addh(a, stream$getl(pi))          % 上限に要素を追加
  end
  for i:int in int$from_to_by(as$high(a), as$low(a), -1) do
    stream$putl(po, a[i])             % as$fetch(a, i) と同等
  end
end start_up
```

しかしこれは最初に逆向きにくっつけておけばもう少し楽。

```
% sam07a.clu -- reverse file upside-down

start_up = proc()
  as = array[string]
  a:as := as$new()
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  while ~stream$empty(pi) do
    as$addl(a, stream$getl(pi))          % 下限に要素を追加
  end
```

```
  for s:string in as$elements(a) do
    stream$putl(po, s)
  end
end start_up
```

あるいは、上限につけといて取り除きながら打つというのも可能。

```
% sam07b.clu -- reverse file upside-down

start_up = proc()
  as = array[string]
  a:as := as$new()
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  while ~stream$empty(pi) do
    as$addh(a, stream$getl(pi))
  end
  while ~as$empty(a) do
    stream$putl(po, as$remh(a))
  end
end start_up
```

このように、様々な操作が用意されていて使い分けられるのがいいと思うのですが…

2　2次元配列

さて、いつも大きさが空だと不便ですね?　そこで大きさを指定して作る操作 `fill` があります。例えば大きさ 10 の配列を作る。

```
% sam08.clu -- array example

start_up = proc()
  ai = array[int]
  a:ai := ai$fill(1, 10, 1234)
  a[4] := 0
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  for i:int in ai$indexes(a) do
    stream$putright(po, int$unparse(i), 5)
    stream$putright(po, int$unparse(a[i]), 8)
    stream$putc(po, '\n')
  end
end start_up
```

これを動かすと…

```
% a.out
    1    1234
    2    1234
```

```
 3    1234
 4       0
 5    1234
 6    1234
 7    1234
 8    1234
 9    1234
10    1234
```

これはよいですよね。じゃあ 2 次元だとどうなるか。

```
% sam08a.clu -- array example 2dim

start_up = proc()
  ai = array[int]
  aai = array[array[int]]
  a:ai := ai$fill(1, 10, 1234)
  aa:aai := aai$fill(1, 10, a)
  aa[4][4] := 0
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  for i:int in aai$indexes(aa) do
    stream$putright(po, int$unparse(i), 5)
    for x:int in ai$elements(aa[i]) do
      stream$putright(po, int$unparse(x), 5)
    end
    stream$putc(po, '\n')
  end
end start_up
```

実行すると:

```
% a.out
    1 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    2 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    3 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    4 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    5 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    6 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    7 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    8 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
    9 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
   10 1234 1234 1234    0 1234 1234 1234 1234 1234 1234
```

つまり。fill だと 1 個の要素がすべての箱によって共有された配列ができてしまう。だから、1 箇所書き変わると全部書き変わるわけですね。これを避けるには、fill_copy を使えばよい。(わかりますね?)

最後になりましたが、「配列定数」のようなものが書けます。その形式は

型指定 [下限: 式,…]

なお下限は指定しなくてもいい。例えば次の通り。

```
% sam08b.clu -- array example 2dim

start_up = proc()
  ai = array[int]
  aai = array[array[int]]
  a:ai := ai$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  aa:aai := aai$fill_copy(1, 10, a)
  aa[4][4] := 0
  pi:stream := stream$primary_input()
  po:stream := stream$primary_output()
  for i:int in aai$indexes(aa) do
    stream$putright(po, int$unparse(i), 5)
    for x:int in ai$elements(aa[i]) do
      stream$putright(po, int$unparse(x), 5)
    end
    stream$putc(po, '\n')
  end
end start_up
```

A 構造型の操作 (その 1)

A.1 array[T] 型

以下 at = array[T] とする。

```
at$create = proc(int) returns(at)   -- 下限を指定して作る
at$new = proc() returns(at)         -- create(1)
at$predict = proc(int, int) returns(at) -- 下限と「大きさの準備」
at$low = proc(at) returns(int)      -- 下限
at$high = proc(at) returns(int)     -- 上限
at$size = proc(at) returns(int)     -- 要素数
at$empty = proc(at) returns(bool)   -- size = 0?
at$set_low = proc(at)               -- 下限の変更
at$trim = proc(at, int, int) signals(bounds,negative_size) -- ちょん切る
at$fill = proc(int, int, T) returns(at) signals(negative_size)
at$fill_copy = proc(int, int, T) returns(at) signals(negative_size)
at$fetch = proc(at, int) returns(T) signals(bounds) -- 要素アクセス
at$bottom = proc(at) returns(T) signals(bounds) -- 最初の要素
at$top = proc(at) returns(T) signals(bounds) -- 最後の要素
at$store = proc(at, int, T) signals(bounds) -- 格納
at$addh = proc(at, T)
at$addl = proc(at, T)
at$remh = proc(at) returns(T) signals(bounds)
at$reml = proc(at) returns(T) signals(bounds)
at$elements = iter(at) yields(T)
at$indexes = iter(at) yields(int)
at$equal = proc(at, at) returns(bool)
at$similar = proc(at, at) returns(bool)
at$similar1 = proc(at, at) returns(bool)
```

```
at$copy = proc(at) returns(at)
at$copy1 = proc(at) returns(at)
```

練習問題 次のような CLU プログラムを書け。

- ファイルを 2 回重複するフィルタ。
- 1 行目は 1 回、2 行目は 2 回…、のように重複するフィルタ
- 行を辞書順に並べかえるフィルタ
- 入力行に行番号をふった上で内容の辞書順に並べ買えるフィルタ