プログラム言語論(7 or 6補)
パラメータ渡し

櫻井彰人

本題の前に

- Parameter:
  - 1650s in geometry, from Modern Latin *parameter* (1630s), from Greek *para-* "beside, subsidiary" (see **para-** (1)) + *metron* "measure" (from PIE root ***me-** (2) "to measure").
  - A geometry term until 1920s when it yielded sense of "measurable factor which helps to define a particular system" (1927). Common modern meaning (influenced by *perimeter*) of "boundary, limit, characteristic factor" is from 1950s. Related: *Parametric*.
- Argument:
  - early 14c., "statements and reasoning in support of a proposition or causing belief in a doubtful matter," from Old French *arguement* "reasoning, opinion; accusation, charge" (13c.), from Latin *argumentum* "a logical argument; evidence, ground, support, proof," from *arguere* "make clear, make known, prove" (see **argue**). Sense passed through "subject of contention" (1590s) to "a quarrel" (by 1911), a sense formerly attached to **argumentation**.
  - The OED cites an example from Chaucer (c. 1386) under the following definition: "Astr. and Math. The angle, arc, or other mathematical quantity, from which another required quantity may be deduced, or on which its calculation depends." The etymology is pretty much that given by Rhomboid, above.

https://www.etymonline.com/word/parameter
https://ask.metafilter.com/78617/Why-argument
https://english.stackexchange.com/questions/144141/what-is-the-sense-of-using-word-argument-for-inputs-of-a-function

制御構造の抽象化

- 命令型言語の抽象化には、データ面だけでなく制御面にもある
- どんな言語も制御の抽象化を行っている
  - 選択
  - 繰り返し
- これらは、プログラムの文レベルでの抽象化
  - “ユニット”レベルの抽象化も存在する
  - 手続き、関数、コルーチン、繰り返し子や生成子

手続きと関数

- 単純な “call and return” という制御を提供する
  - master/slave の関係
  - すなわち非対称である
- このユニットレベルの制御抽象化には次の共通的な特徴がある:
  - 名前 (呼んだり、起動したりするときにこれで指定する)
  - 入力パラメータの集合 (引数を供給するため)
  - 出力パラメータの集合 (結果を戻すため)
- このパラメータが実際の値とどのように連絡するかを調べる.
  - パラメータ渡しの機構

Call by Copy

- 仮引数と実引数は、記憶域を共有しない
- 3つの下位概念がある:
  - call-by-value, call-by-result, call-by-value-result
- **Call by value（値渡し）**
  - 呼出し側が実引数を評価（計算）する
  - 呼び出される側の局所変数として扱われる仮引数に、この値は代入される
  - 呼出側へ情報を返すことはない (仮引数への代入は、局所変数への代入と変わるところはない)

  Ada の *in* mode である

  Java の 引数引渡し機構である
- 値渡しはは広く使用されている
  - 問題もある: 例えば、100Mバイトの配列を渡すとどうなる?

Call by Copy (続)

- **Call by result（結果渡し）**
  - 呼出し時には、変数間の結合は起こらない
  - 仮引数は、局所変数として振舞う
  - 戻るときに, 仮引数に入っている最終値が実引数にコピーされる

  Ada の *out* mode 引数がこれである
- **Call by value-result（値-結果渡し）**
  - 仮引数は、実引数の値で初期化され、局所変数として振舞う
  - 仮引数の最終値は、戻り時に、実引数に代入される

  Ada の *in out* mode 引数がこれにあたる
- 以上の機構に共通する特徴は、呼出側の環境・文脈で実引数を評価することと、それをコピー（代入）によって受け渡すことである

Call by Reference（参照渡し）

- 参照渡しは、インプリメントが易しい
  - 呼出側は、実引数の番地を渡す
  - 実引数と仮引数は、alias の関係にある（同じ番地を別の名前で参照する）
- 効率的 (100Mbyte の配列は、一文字と同速度で渡される)
- 失うのは、参照の透明性 referential transparency
  - 一つの変数を変更すると他の変数が変わってしまう
- 参照渡しと値渡しは、非常に異なった方法であり、異なる結果となる
- 言語によっては、参照渡しをするには明示的に宣言する必要がある (eg Pascal).

Call by need

- 引数の中には、時によっては使われないものもある
- そうした場合には、"Call by Need" という引数渡しが有効
- "call by need" では, 仮引数が最初に参照されたとき、それの評価が行われる（対応する実引数が計算される）
- ある呼び出しで、一度も参照されなければ、評価されない
  - しばしば "lazy evaluation"（遅延評価）と呼ばれる
- "Call by need" は、ポインタが使われているとき、引数の一部をいつ更新していいか、非常に判断が難しい
- 従って、これが使用されるのは、関数型言語のように、値が実際上変更できないような言語（こうした場合、非常に明確に、意味が定まるの）である

Call by Name（名前渡し）

- この機構の "標準的な" 説明は次の通り:
  - "手続き・関数（呼び出され側）のプログラムを、呼出した場所に、字面通りコピーする.
  - その時、仮引数の場所には、実引数を、字面通り、書く"
- 実際には、これを言葉通り行うことはせず、中間変数を用いて、同様の効果を出す
- メカニズム
  - 仮引数は、呼出側のある場所（変数等）を示す
  - 仮引数と実引数の間の結合は、呼出し時に行われるのではない
  - 結合は、仮引数が参照されたとき、その度毎に、行われる
- この表現は、実際、上記の "標準的な" 記述の実際的な実装となっている
  - 記述は非常に簡明であるが、何が起こるかを考えると、非常に分かり難い機構である

Call by Name 例 1

- 典型的 "swap" 手続き:

```
procedure
  swap(a, b : T);
  temp : T;
  begin temp := a;
    a := b;
    b := temp;
  end;
```

- *swap(i, a[i]);* とすると何がおこるか?

```
temp := i;
i:= a[i];
a[i] := temp;
```

- 意外か?
- 字面で置き換えるというのは非常に自然な考えに思える（値だ、番地だと言うより）. しかし、実際はそうではない!
- 更なる注: 仮引数の評価は、呼出側が参照する環境*referencing environment* で行われる

Call by Name 例 2

```
procedure x ...
  c : integer;
  procedure swap(a, b : T);
    temp : T;
    begin temp := a;
      a := b;
      b := temp;
      c := c + 1;
    end;
  procedure v ...
    c, d : integer;
    begin ...
      swap(c,d);
  end
```

```
temp := c;
c := d;
d := temp;
c := c + 1
```

Call by Name 例 3

- "call by name" 機構の精妙さを堪能するために、"**Jensen's Device**" をみてみよう
- **Jensen's Device** は単純な関数であるが、次のようなことに使える:
  - 行列積
  - ベクトルの内積
  - sin(x), cos(x) や他の三角関数
  - Taylor 展開
  - ...
- この魔法は、たった一つの関数が、これらに共通していること、そして色々な機能が実現できることは、引数として渡すものを様々に変えることができることによっている。
  - 当然ながら、関数本体は、そう自明ではない

Jensen's Device

```
real procedure sum( expr, index, lb, ub );
  value lb,ub;
  real expr;
  integer index, lb, ub;
  begin real temp := 0.0;
    for index := lb step 1 until ub do
      temp := temp + expr;
    sum := temp;
  end sum
```

引数のモードと型の宣言

まるで
expr * (ub - lb+1)
を計算しているよう

例: sum(a[i]*b[i], i, 1, 25)
for i := 1 step 1 until 25 do
temp := temp + a[i]*b[i];
* これはベクトルの内積であり、行列積の一部!!

例

- 下記のプログラムについて、引数の受渡し機構をかえたときに得られる結果を比較せよ

```
real procedure p(x,y,z);
  integer x,y,z;
  begin
    x := 2;
    y := x+1;
    z := x+1;
  end;
begin
  x := 1;
  a := [1,2,1,4];
  p(x,x,a[x]);
end
```

配列 *a* の初期化

憶えておくべきこと

- 引数の受渡し機構には色々ある
- 別の引数受渡し機構を用いると、別の結果が得られる