プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 6 (制御構造)

構造化プログラミング

Dijkstra, "GO TO CONSIDERED HARMFUL" (1968)

- go to 文を多用したプログラムは理解しづらい.
- かわりに, より「構造的」な制御文を使うべき.
- if-then-else, while, for, case ...

制御構造 FORTRAN

```
10  IF (X .GT. 0.000001) GO TO 20
    X=-X
11  Y=X*X-SIN(Y)/(X+1)
    IF (X .LT. 0.000001) GO TO 50
20  IF (X*Y .LT. 0.000001) GO TO 30
    X=X-Y-Y
30  X=X+Y
    ...
50  CONTINUE
    X=A
    Y=B-A+C*C
    GO TO 11
```

スパゲッティ・コード (Mitchell, "Concepts in PL", 2003 より)

リストの要素の積を返す関数

```
open List;;  (* おまじない *)
let rec mult x =
  if x=[] then 1
  else (hd x) * (mult (tl x))
in mult [2; -3; 5]  ;;
```

hd はリストの先頭要素を取る関数.
tl はリストの先頭要素を除いた残り (リスト) を取る関数.
`mult [2;-3;5] => -30`

例外的な状況

mult 関数で、リストの要素に負の数があったら，(積ではなく) その要素を返すことにしたい.

```
let rec mult2 x =
  if x=[] then 1
  else if (hd x) < 0 then (hd x)
  else (hd x) * (mult2 (tl x))
in mult2 [2; -3; 5]  ;;
```

これではうまくいかない.

例外的な状況

```
let rec mult3 x =
  if x=[] then 1
  else if (hd x) < 0 then (hd x)
  else
     let res = mult3 (tl x) in
     if res < 0 then res
     else (hd x) * res
in mult3 [2; -3; 5]  ;;
```

再帰呼出しをするごとに,「例外的な状況が起きたかどうか」をチェックしないといけない.
⇒プログラムが読みづらくなるし，効率も悪い.

例外機構の利用

```
exception Negative of int ;; (例外の宣言)

try
  let rec mult4 x =
    if x=[] then 1
    else if (hd x) <= 0 then
        raise (Negative (hd x))
    else (hd x) * (mult4 (tl x))
  in
    mult4 [2; -3; 5]
with
  Negative n -> n ;;
```

raise (例外の発生) を実行すると，対応する try-with (例外の処理) まで一気にジャンプする.

例外機構: 別の例

map 関数の利用:

```
open List;;
let inc x = x +. 1.0;;
map inc [1.0; 2.0; 3.0];;
map sqrt [1.0; 2.0; 3.0];;
map sqrt [1.0; -2.0; 3.0];;
```

負の数があったら，例外を出したい.

```
exception Neg of float;;
let f r =
   if r < 0.0 then raise (Neg r)
   else sqrt r ;;
try
    map f [1.0; -2.0; 3.0]
with
    Neg r -> [r] ;;
```

例外機構

- 深い関数呼出しから一気に抜ける.
- 「内から外」の方向のみ可能.
- 例外の種類に名前を付け，発生した例外と一致する名前の「受け手」まで飛ぶ.
- 例外の発生と受け手は，動的に対応付けられる.

例外機構は動的束縛 (やや高度な内容)

```
exception E3;;
try
  let f x = raise E3 in
  let g x =
    try
       f 10
    with E3 -> 20
  in
    g 0
with E3 -> 30;;
```

上記の答は「20」である．(「30」ではない)

いろいろな言語の例外機構

例外機構〜大域脱出機構

- C: setjmp(), longjmp()
- C++: try-catch, throw
- Java: try-catch-finally, throw
- ML: try-with (または handle), raise

現代のプログラム言語では，例外機構を持つことはほとんど必須と考えられる。

継続 (continuation); 発展的な内容

実行時の「残りの計算」を表す概念.

```
let rec fact n =
  if n=0 then 1
  else n * (fact (n-1))
in fact 10
;;
let rec fact2 n k =
 if n=0 then k 1
 else
   fact2 (n-1) (fun x -> k (n*x))
in fact2 10 (fun x -> x)
```

fact2 は fact と同じ計算

```
fact2 10 (fun x -> x)
fact2 9 (fun x -> 10*x)
fact2 8 (fun x -> 10*9*x)
...
```

継続

例: `fact2 9 (fun x -> 10*x)` の第二引数
「残りの計算」を，関数で表すことにより，末尾呼び出しの形に変形できた．(「継続渡し方式」(Continuation Passing Style) のプログラム)
例外機構の表現:

```
let mult5 x =
  let rec f x k =
    if x=[] then (k 1)
    else if (hd x < 0) then
       (hd x)
    else f (tl x) (fun v -> k ((hd x)*v))
in f x (fun v -> v)
```

```
mult5 [1;5;-3]
=> f [1;5;-3] (fun v->v)
=> f [5;-3] (fun v->1*v)
=> f [-3] (fun v->5*1*v)
=> -3
```


継続の利用

miniC 言語のミステリー:

- miniC 言語には、例外機構等のコントロール抽象の仕組みはない。
- miniC 言語インタープリタは、継続渡し方式で実装。

C/miniC 言語の return 式:

```
int f (int x) {
  ...
  for (i=0; i<100; i++} {
    ...
    if (...) return 1;
  }
  ...
  y = x ? (return 10) : 5;
  ...
}
```

そのほかにも、continue など。

継続

- 継続渡し方式: 継続を関数で表現したプログラミングスタイル.
- 一方，継続を直接扱うことにより、継続渡し方式と同等のプログラムを簡潔に書けるようにした言語もある.
  - call/cc (call-with-current-continuation; Scheme, SML/NJ, Ruby)
  - call/cc を使うと，例外機構以外に，バックトラックやコルーチンなど種々の制御を表現できる.

まとめ

- 構造化プログラミングとプログラムの制御構造
- 例外機構
- (継続)

問題:

- GO TO 文 (無制限のジャンプ命令) を乱用すると，なぜ，有害なのだろうか.
- 以下の C プログラムは、どういう点が意図通りでないか、説明せよ。

```
int mult2 (int* x, int len) {
  if (len==0) return 1;
  else if (*x < 0) { return *x;}
  else return *x * (mult2(x+1,len-1));
}
```

- (発展課題) Scheme の call/cc や Ruby の yield などの制御機構を調べ，特徴を述べなさい.