仕様・アルゴリズム・プログラム

2003年10月20日

海谷 治彦

問題の定義 ～ 仕様(書)

- プログラム(コンピュータソフトウェア)はある種の問題解決のために構築・利用される.
- よって，開発の前には，どのような問題のためのプログラムなのかを明確にするのが必要.
- 問題を文書化したものを仕様(書)と呼ぶ.

問題の例

- 数列を入力すると平均と標準偏差を計算.
- 数列を大きい順に並べ替える.
- 整数の平方根を求める.
- 名前とパスワードを入れると口座残高を照会できる.
- マウスで二点クリックすると，その二点間に線を引く.
- URLを入力すると，ホームページが見える.
- かな文字を入れてスペースを押すと漢字に変換される.

問題の本質 ～ 関数

- **$(y_1, y_2, y_3 \ldots) = f(x_1, x_2, x_3 \ldots.)$**
  - いわゆる数学の関数と同じ.
  - ただし，入出力ともにベクトルとして一般化.
- $x_1, x_2, x_3 \ldots$: 入力，これをもとに何か処理.
- f: 関数の名前，他の区別するためにつける.
- $y_1, y_2, y_3 \ldots$: 出力，処理結果.

問題の仕様化

- 基本的には，入力データと出力データの関係を文章で書く.
- 永続的なデータの参照が必要な場合もある.
  - 例: 住所録の更新等.
  - 永続データを入力として読み，それを変更して出力として吐くという風な見方をしてもよい.
- 形式的(数学的)にかいてもよい.
- 基本的・直感的な演算は利用してよい.
  - ＋－÷×等
  - 文字列の連結等

仕様書の例

- 「数列を入力すると平均と標準偏差を計算．」
  - 入力データをx_1~x_nとして，出力データをyとすると，$(x_1+\ldots+x_n)/n=y$ となる．
- 「マウスで二点クリックすると，その二点間に線を引く．」
  - 始点をx_1, y_1, 終点をx_2, y_2 ，現在の画面をgとすると，(x_1,y_1)〜(x_2,y_2)を結ぶ線に近接するドット描画を追加したgを出力する．

問題の典型的種類

- 単純に入力から出力を計算する.
  - 平均計算の例は典型例
- 計算に入力以外の情報が別途必要な場合.
  - 入力のユーザー名，パスワードのみでは銀行口座の残高を照会できない.
    - ユーザー名と残高表の対データが別途必要.
- 対話型の計算.
  - 関数を連続的に呼び出しとみなしてよい，
  - 場合によっては，入力以外の情報が必要な場合もある.
    - 関数の呼び出し順番によって，問題が変わる等.

問題解決の手続き: アルゴリズム

- 問題解決するには，原始的な命令をある規則に従って実行するしかない.
  - 特にコンピュータの場合.
  - 人間の場合「めのこ」でできちゃう場合もあり.
- 「ある問題を解くための命令列と実行順序」をアルゴリズムと考えてよい.
- 問題を解決する手続き(手順)は複数通りあるため，問題とアルゴリズムは別物.

WhatとHow

- What: 何を解決するか？
  - 仕様書は，この情報のみ記述するのがよい.
  - 例: 数を大きい順に並べる(降順整列).
- How: どうやって解決するか？
  - アルゴリズム，プログラムは，このHowを記述するのに使う.
  - 例: クイックソート，バブルソート，なんとかソート … 整列アルゴリズムは多数存在する.

アルゴリズムの現実

- プログラムは変数の読み書き命令の連続実行によって計算を進める.
- よって，アルゴリズムも以下の語彙・命令を使って記述されるのが普通.
  - 変数のような記憶保持をするもの
  - 集合や列，木等の基本的なデータ構造
  - 代入
  - 基本的な演算: ＋ー÷×値入れ替え(swap)等
  - 基本的な条件判断: 大小，包含の真偽等

問題とアルゴリズムのギャップ

- あるアルゴリズムが，ある問題の解決に使えるとは一見して解らない場合もある.
- そのようなアルゴリズムは数学的な性質等をベースとして設計されているためである.
- 何故，そのアルゴリズムがその問題を解決するかの理由を理解するのも大切だが，
- 確かにそのアルゴリズムがその問題を解決することを確認する程度でも最初は良いだろう.

ギャップの例

- ユークリッドの互除法 (超有名な方法)
- **問題**: 正整数 n, m の最大公約数 z を求める.
- **理論**: 最大公約数の計算を gcd(x,y)で表すと，以下の事実が数学的に知られている.
  - gcd(a,a)=a
  - gcd(a,b) = gcd(b,a)
  - a>b の場合，gcd(a,b)=gcd(a-b, b) かつ 0<a-b<a
  - 上記の変換規則により，gcd への入力値は単調減少する.
- 上記の数学的性質を知らないと，それをもとにしたアルゴリズムの意味はさっぱりわからない.

アルゴリズムの例

- 問題: 正整数ペアの最大公約数を計算
  - 入力: x, y
  - 出力: z
- 手順
  1. x と y が等しければ, zの値をxと同じにして処理を終える.
  2. x < y ならば, x と y の値を入れ替える.
  3. x を x-y に入れ替える.
  4. 手順 1.に戻る.

Cのプログラム例 (参考)

```
int x, y, z;

    while(1){
        if(x==y){
            z=x;
            break;
        }
        if(x<y){
        int c;
            c=x; x=y; y=c;
        }
        x=x-y;
    }
```

アルゴリズムのトレース

- アルゴリズム内で示された手順が問題を解決するかどうかを例題で確かめる.
- 本当のところなら問題クラス全てを解決できることを示す必要がある.
  - 前述のgcdの例なら，全ての正整数のペアに対して，正しい結果を計算することを示す必要がある.
  - が，本授業ではそこまで要求しない.
- 要は人間コンピュータになってみる.

トレースの例: gcd(12,15)

手順 (参考)

1. x と y が等しければ, zの値をxと同じにして処理を終える.
2. x < y ならば, x と y の値を入れ替える.
3. x を x-y に入れ替える.
4. 手順 1.に戻る.

1. 開始 x: 12, y:15
2. 手順1 x: 12, y: 15
3. 手順2 x: 15, y: 12
4. 手順3 x: 15-12, y: 12
5. 手順1 x: 3, y: 12
6. 手順2 x:12, y:3
7. 手順3 x: 9, y:3
8. 手順1 x: 9, y:3
9. 手順2 x:9, y:3
10. 手順3 x:6, y:3
11. 手順1 x:6, y: 3
12. 手順2 x:6, y:3
13. 手順3 x:3, y:3
14. 手順1 z=3
15. 修了

かなりたるい.
が, 計算手続きの確認にはなる.
何故, これが解決になるのかは,
トレースしても直接はわからない.

アルゴリズムとプログラムの違い

- 詳細はHPを参照.

- 一般に「アルゴリズムはより抽象的に」と言われる.
- しかし，どっちにしろ手続き型計算モデルを使っているので，あんまり変わらない場合が多い.

以上

整数の平方根を求める問題

- 問題: 非負整数の平方根を求めよ. ただし, 解が整数とならない場合は小数点以下を切り捨てよ.
- 理論: 以下の数学的事実が知られている.