第 5 章 計算の方法

Ⅰ. 計算の方法

モデル化されたデータに対する種々の操作は、普通“計算”と呼ばれる。

ここでは、計算の一種である計数(*counting*)を例にとり、$\emptyset$ でない有限集合 A の要素数 $n(A)$ を求める計算を考える。この計算は、「A に対してどのような処理ができるか」に依存する。

1. 取り出し型

集合に対して、

・その集合が空集合 $\emptyset$ であるか否かを判定する

・その集合の要素を 1 つ減らす

という 2 つの処理ができるとする。この時、$n(A)$ を求めるには、〈答〉という変数を用いて、

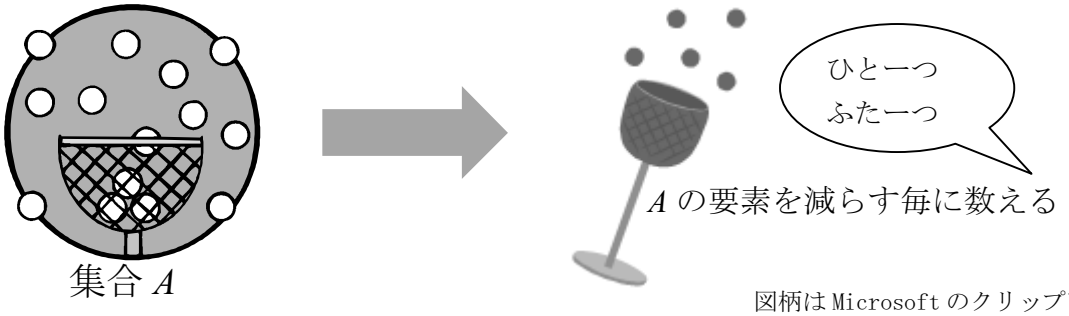
・まず〈答〉を 0 にする

・$A \neq \emptyset$ である間、次の処理を繰り返す

　・A の要素を 1 つ減らす

　・〈答〉を 1 増やす

という計算を行えばよい。これは、運動会の玉入れの籠から 1 つずつ玉を取り出していって、その度に大きな声で 1 つずつ数えていくのと同じ感覚である。最後に玉が無くなったときに言った数値が、籠に入っていた玉の数である。(玉の数を保持しておく変数が〈答〉である)



図柄は Microsoft のクリップアートより

2. 分割型

集合に対して、

・その集合の要素が 1 つであるか否かを判定する

・その集合の要素が 2 つ以上ならば、$\emptyset$ でない 2 つの集合に分割する

という 2 つの処理ができるとする。この時、$n(A)$ を求めるには、〈答〉という変数を用いて、

・A の要素が 1 つならば〈答〉は 1

・A の要素が 2 つ以上ならば、次のようにする

　・A を B と C に分割する（但し B も C も $\emptyset$ でない）

　・〈答〉= B の要素数 + C の要素数

という計算を行えばよい。ここでは、もし A の要素が 2 つ以上であれば、B と C とに分割してそれぞれの要素数を求める。B や C についても同様の処理をしてどんどん細かく分割していくので、いつかは必ず要素数が求まるという訳である。

以上で用いた〈答〉のように値が色々と変化するものを**変数(*valiable*)**といい、その名前である“答”という文字列を**変数名**という。これは、値が一定である数学の変数(*parameter*)とは異なる。

Ⅱ. 計算の記述

1. 条件判断と代入

コンピュータの計算で重要なのは**条件判断**と**反復処理**である。ここでは、この 2 つについて細かく見てみる。

具体的な計算の例として、次の問の答えを求めてみる。

> 2008 年の八十八夜(立春を 1 日目として 88 日目)は何月何日か。
> 但し 2008 年の立春は 2 月 4 日で、2008 年は閏年である。

考え方としては、次のようになる。

- 八十八夜は、仮に 2 月 91 日 (= 2 月 4 日 + 88 日 − 1 日)とおける。
  ここで、日数を表す変数〈日数〉を用意し、〈日数〉を 91 にする。
- 2 月は 29 日までなので、〈日数〉から 29 を引き、「3 月」に進む。
  この時点で、八十八夜は 3 月 62 日と考えられる。
- 3 月は 31 日までなので、〈日数〉から 31 を引き、「4 月」に進む。
  この時点で、八十八夜は 4 月 31 日と考えられる。
- 4 月は 30 日までなので、〈日数〉から 30 を引き、「5 月」に進む。
  この時点で、八十八夜は 5 月 1 日と考えられる。
- 5 月は 31 日までなので、確かに八十八夜は 5 月 1 日である。

ここで、変数〈日数〉に対する操作が色々と出てきたが、変数に何らかの値を設定する操作を**代入(*assignment*)** という。これは、文字の消去を行う数学の代入(*substitution*) とは異なる。

以降、ある変数に値を代入する操作を次のように表す。

変数 ← 式

例えば、上の「〈日数〉を 91 にする」であれば

〈日数〉← 91

であり、「〈日数〉から 29 を引く」であれば

〈日数〉←〈日数〉− 29

である。(右辺の〈日数〉は代入前の〈日数〉の値を表す)

上で行った計算を、**if** や **then** といった言葉を使った記法で表してみる。具体的には、

「もし条件 X が成り立つなら操作 Y を行い、成り立たないならば Z を行う」

という処理を、

```
if X
    then Y
    else Z
endif
```

と表すのである。慣れないと難しいかもしれないが、要は書き方の問題である。このように、条件 X に応じて操作が変わる処理を**条件付き処理(*conditional processing*)** という。

尚、**else** Z は不要なら省略できる。

この記法で八十八夜の問題を書き直してみる。右側は、同じ内容を日本語で書いたものである。

```
〈日数〉← 91
if〈日数〉> 29
  then〈日数〉←〈日数〉− 29
    if〈日数〉> 31
      then〈日数〉←〈日数〉− 31
        if〈日数〉> 30
          then〈日数〉←〈日数〉− 30
            if〈日数〉> 31
              then (6 月以降の処理)
              else ″5 月〈日数〉日″と表示
            endif
          else ″4 月〈日数〉日″と表示
        endif
      else ″3 月〈日数〉日″と表示
    endif
  else ″2 月〈日数〉日″と表示
endif
```

```
〈日数〉を 91 にする。
もし〈日数〉が 29 を超えていれば、
  〈日数〉から 29 を引き、
    もし〈日数〉が 31 を超えていれば、
      〈日数〉から 31 を引き、
        もし〈日数〉が 30 を超えていれば、
          〈日数〉から 30 を引き、
            もし〈日数〉が 31 を超えていれば、
              6 月以降の処理をする。
              そうでなければ″5 月〈日数〉日″と表示する。

          そうでなければ″4 月〈日数〉日″と表示する。

      そうでなければ″3 月〈日数〉日″と表示する。

  そうでなければ″2 月〈日数〉日″と表示する。
おしまい。
```

因みに、行の頭を字下げ(*indentation*)しているのは、**if** や **then** などの対応関係の構造を見やすくするためである。

2. 反復処理と添字つき変数

上で書いた処理は、似たような内容を少しずつ変えて何度も何度もだらだらと同じように書いており、非常に冗長で長い(この文のような文を冗長な文という)。

それでは嫌なので、スッキリと纏めて書きたいと思う。上の処理では、

```
if〈日数〉> (m 月の日数)
    then 〈日数〉− (m 月の日数)
         (m + 1 月の処理)
    else ″m 月〈日数〉日″と表示
endif
```

という処理を繰り返している。そこで、m 月の日数を d_m（$d_1 = 31, d_2 = 29, \cdots, d_{12} = 31$）とし、

「条件 X が成り立っている間、操作 Y を繰り返す」

という処理を

```
while X do
    Y
done
```

と表す構文を導入する。

このように、何度も同じ処理を繰り返す処理を**反復処理(*repetitive processing*)**という。

この構文を用いて、八十八夜の問題を書き直してみる。

〈日数〉← 91	〈日数〉を 91 にする。
m ← 2	m (月を表す変数)を 2 にする。
while 〈日数〉> d_m **do**	〈日数〉が m 月の日数を超えている間は何度もでも、
〈日数〉← 〈日数〉− d_m	〈日数〉から m 月の日数を引き、
m ← m + 1	m を 1 増やす(次の月に進む)
done	という処理を繰り返す。
″m 月〈日数〉日″ と表示	処理が終わったら ″m 月〈日数〉日″ と表示する。

格段に短く、読み易くなった。
ここで、d_m のように添字をつけた変数とその値のことを**配列(*array*)** と呼ぶ。配列は、その全体を 1 つのものとして扱ったり、個々の要素それぞれを変数として扱ったりできる。
因みに、行の頭を字下げ(*indentation*)しているのは、**if** や **then** などの対応関係の構造を見やすくするためである。

第 5 章はここまで。