第 1 章

思考の道具

「言語は思考を規定する」という．実際プログラマは思考の道具としてプログラミング言語をよく用いるし，そのプログラマの好みのプログラミング言語の影響を強く受ける．手続き的に問題を解くのが好きなプログラマもいれば，宣言的に問題を解くのが好きなプログラマもいる．問題の複雑さを「オブジェクト」というプログラム単位に押し込むのが好きなプログラマもいれば，「クロージャ」という別のプログラム単位に押し込むのが好きなプログラマもいる．

一方で，古代ギリシャから連綿と続く数学者たちは，数学という言葉で問題を考えることを好む．数学はわずかな方言の違いを無視すれば驚くほど統一された言語である．

プログラマが使う言葉すなわちプログラミング言語と，数学者たちが使う後はだいたい乖離しており，その両者の歩み寄りは虚しいものであることが多かった．ここで言いたいのはプログラマが限られた文字セットしか使えないとか，スクリーン上の行という制約に縛られているとか，そのような表面的なことではない．そうではなく，思考の様式が，プログラミング「文化」と数学「文化」で異なるという意味である．

プログラミング文化と数学文化の大きな違いの一つは，リアルタイムに起こるイベントの取り扱い方法である．プログラマはほとんどいつもリアルタイムに起こるイベントに対応しないといけない．一方で，数学者たちはそのようなイベントを「不純なもの」として理論から取り除く．例えば変数 x があるときは 0 だがあるときは 1 であるというのは，甚だ不純なものとして数学者の間では認識されるものだ．

第 2 章

数学的準備

この章では今後本書で登場する数学的概念の記法を決めておく．

2.1 定数と文字定数

定数はアラビア数字で記述するものとする．定数が負の場合はマイナス記号 $(-)$ を用いる．定数が実数の場合は小数点を用いる場合がある．例えば $-1, 0, 1.2$ は定数である．

文字定数とはコード化された文字のことである．文字コードを定数として扱うといささか読みづらくなるため，該当する文字をシングルクォートで囲むことにする．例えば ‘A’, ‘b’, ‘.’ は文字定数である．

2.2 変数

変数はアルファベット（ローマ文字またはギリシア文字）の小文字 1 文字で表すことにする．変数名には任意個のプライム記号 $(')$ をつけても良いし，添え字をつけても良いこととする．

2.3 演算子

演算子には数学記号を割り当てる．単項演算子には論理否定 $(\neg)$ とマイナス $(-)$ がある．二項演算子のうちよく使われるものは和 $(+)$ である．二項演算子は多数あるので，その都度説明する．

演算の優先順位を明示的に与えるために括弧が用いられる．

2.4 関数

関数も変数と同じようにアルファベット（ローマ文字またはギリシア文字）の小文字 1 文字で表すことにする．関数名には任意個のプライム記号 $(')$ をつけても良いし，添え字をつけても良いこととする．

関数引数には括弧を付けない．我々はよく引数 x をとる関数 f を $f(x)$ と書くが，括弧は冗長なので今後は fx と書くことにする．引数 x を関数 f に「食わせる」ことを関数適用と呼ぶ．

複数引数をとる関数を我々はよく $f(x, y)$ と書くが，これも括弧が冗長なので今後は fxy と書くことにする．この場合式 fxy は左を優先して結合するものとする．つまり

$$fxy = (fx)y$$

である．引数に「飢えた」関数 (fx) を部分適用された関数と呼ぶ．

関数は合成できる．関数 f と関数 g があって，その合成を $f \cdot g$ と書くとき

$$(f \cdot g)x = f(gx)$$

である．関数合成の演算子 $\cdot$ は関数適用よりも優先順位が高いものとすれば $(f \cdot g)x$ は単に $f \cdot gx$ と書ける．この記法は括弧の数を減らすためにしばしば用いられる．

関数合成とは逆に，関数適用を行う演算子も考えておくと括弧の数を減らすのに便利である．関数適用演算子 $\$$ を次のように定義しておく．

$$f \,\$\, gx = f(gx) \tag{2.1}$$

演算子 $\$$ の優先順位は足し算演算子よりも低いものとする．よって $f(x+1)$ は $f \,\$\, x + 1$ と書くこともできる．

二項演算子とは 2 引数関数の特別な場合であると考えてよい．関数 r を次のように中置する記法を定義しておく．

$$x \,`r`\, y = rxy \tag{2.2}$$

バッククオート記号 $(`)$ を使っているのはプライム記号 $(')$ と区別するためである．

既存の二項演算子 $\circ$ は次のようにして通常の関数として使えるものとする．

$$(\circ)xy = x \circ y \tag{2.3}$$

2.5 ラムダ式

引数 x をとり値 $x+1$ を返すラムダ式は次のように書くことにする．

$$\backslash x \to x + 1 \tag{2.4}$$

この式はチャーチのオリジナルの論文の記法であれば $\hat{x}\,.\,x+1$ と書かれたところであり，現在でも多くの書物で $\lambda x\,.\,x+1$ と記述される．しかし我々はすべてのギリシア文字を変数名のために予約しておきたいのと，ピリオド記号 (.) が関数合成演算子 $(\cdot)$ と非常に紛らわしいため，上述の記法を用いる．

複数引数をとるラムダ式は例えば

$$\backslash xy \to x + y$$

のように書く．

2.6 関数定義

ラムダ式を用いた関数の定義が可能である．例えば引数 x をとり値 $x+1$ を返す関数 f は

$$f = \backslash x \to x + 1 \tag{2.5}$$

と定義できる．この省略形として

$$fx = x + 1 \tag{2.6}$$

と書いても良いこととする．

関数にスペシャルバージョンがある場合は列挙する．例えば引数が 0 の場合は特別に戻り値も 0 であり，その他の場合は関数 f と同じ振る舞いをする関数 f' を考える．この時 f' は次のように定義することになる．

$$\begin{aligned} f'0 &= 0 \\ f'x &= x + 1 \end{aligned} \tag{2.7}$$

関数定義に場合分けが必要な場合は「ガード」を用いる．例えば引数の値が負の場合は -1 を，0 の場合は 0 を，それ以外の場合は関数 f と同じ振る舞いをする関数 f'' は

$$\begin{aligned} f''x\,|_{x<0} &= -1 \\ |_{x\equiv 0} &= 0 \\ |_{\text{otherwise}} &= x + 1 \end{aligned} \tag{2.8}$$

という風に定義することにする．

2.7 条件式

数学者のいう条件式とはスペシャルバージョンやガードを一般化したもので，if 節と otherwise 節からなるものである．例を挙げる．

$$fx = \begin{cases} 0 & \text{if } x \equiv 0 \\ x + 1 & \text{otherwise} \end{cases} \tag{2.9}$$

この書き方は行を意識せざるをえないプログラミング言語とは相性がよろしくない．そこで次のように書くものとする．

$$fx = \mathbf{if}\, x \equiv 0\, \mathbf{then}\, 0\, \mathbf{else}\, x + 1 \tag{2.10}$$

2.8 型

型とは変数が取りうる値に与えた制約のことである．数学者はしばしば変数 x が整数であることを $x \in \mathbb{Z}$ のように書く．

今我々が考慮しておくべき型は，論理型 ($\mathbb{B}$)，整数型 ($\mathbb{Z}$)，及び実数型 ($\mathbb{R}$) である．括弧内に示した記号は数学者たちが慣用的に用いているものである．

2.9 リスト

同じ型の値を一列に並べたものはリストである．例えば 0 から始まり 10 まで続く整数のリストは $[0,1,2,3,4,5,6,7,8,9,10]$ と書く．等差数列に限って，簡略化した書き方が許されるものとする．0 から 10 までのリストは $[0,1\cdots 10]$ と書いても良い．

より高度なリスト，例えば 0 から 10 までの平方数のリストは

$$\left[x^2\,|_{x\in[0,1\cdots 10]}\right]$$

のように書く．

リストは無限個の要素を持っても良い．例えば自然数全体を表すリスト n は

$$n = [1, 2\cdots]$$

のように定義して良い．

型 $\mathfrak{a}$ のリストの型は $[\mathfrak{a}]$ で表す．

2.10 タプル

複数の変数を束ねたものをタプルと呼ぶ．変数 x と変数 y からなるタプルは (x, y) と書く．いま

$$t = (x, y)$$

としたとき，タプルの中身を取り出すには

$$(t_1, t_2) = t$$

のようにしてパタンマッチングを用いる．