2. . . の旅

2.1　関数

数学に**関数**は付きものである。関数は特別難しいものではない。記述の仕方に慣れさえすれば、関数はむしろ便利で使いやすいのである。

関数 $f(x) = x^2$ があるとする。これは、f という名の関数に (何がし)^2 の機能を持たせている。そのため関数 f に 5 を与えると、25 という値が返ってくるのだ。また関数 $g(x) = 2^x$ は、g という名の関数に $2^{(何がし)}$ の機能を持たせている。よって関数 g に 10 を与えると、1024 という値が返ってくるのだ。

ここでちょっと変な感じを持った人がいるだろうか？　私が関数 f、g と言ったことに対してだ。だが、これでいい。$f(x)$ と書くのは、関数 f が変数 x を使うことを明らかにするためなのだ。もし皆が皆、変数には x しか使わないと取り決めていたら、$f = x^2$、$g = 2^x$ で十分である。しかし、変数には x 以外の文字を使うことはよくあるし、変数が 2 つ以上ある関数だってある。さらには文字定数を利用する場合があることを思えば、関数名に続けて ( ) を付け加えることが不可欠なのである。

関数の例をあと 2 つ提示しよう。

最初は $f(x,\ y) = x^2 - 3y$ でどうだろう。関数 f は変数に x, y を使う。その結果返ってくるのが $x^2 - 3y$ で計算された値だ。だから $f(5,\ 10)$ と書けば、$x = 5, y = 10$ を代入して計算した値 -5 が返ってくることになる。一般に数学で使う関数は、実数値を受け取って実数値を返すので、$f(0.8,\ 0.1)$ でも計算可能で 0.34 が返ってくる。単純に端数を取り除くのではないことに注意しよう。

2 つ目の関数は $g(x) = [x]$ でどうだろう。見慣れない記号 [ ] は**ガウス記号**[1]と呼ばれる。ガウス記号を簡単に説明すると、『与えられた値を超えない整数のうち、最大のもの』を返す機能を持っている。むむ、難しい表現だ。具体的には $g(3.14)$ と書けば 3 が返され、$g(-3.14)$ と書けば -4 が返されるのだ。

もっとも現在は、同じ意味で $\lfloor x \rfloor$ と書くことも多い。floor 記号と呼ばれる。これとは逆の、『与えられた値を下回らない整数のうち、最小のもの』を返す場合の記号は ceiling 記号と呼ばれ $\lceil x \rceil$

[1] カール・フリードリヒ・ガウス (1777–1855)：ドイツの数学者。

と書く。ここでは floor の関数だけを扱うので、昔ながらのガウス記号を用いることにする。

EX. $g(x) = [x]$ において、$g(10)$ とするといくつが返ってくるか？　また、$g(-10)$ や $g(-8.9)$ ではどうか？

今 2 つの関数を例に出してみたが、値を求めるためにする記述は、いずれも $f(5,\ 10)$ や $g(3.14)$ のように簡便だ。にもかかわらず -5 や 3 の値が返ってくるのは、裏方で f や g がせっせと $5^2 - 3 \times 10$ や $[3.14]$ の計算をしているからである。また、$f(3,\ 8)$ のように別の値を与えれば、それに応じた値が返ってくる。これは、関数がする仕事がきちんと決められているからである。

何のことはない。関数とは、与えられた値を別の値に加工する手続きなのだ。

この旅では**指数関数**を話題に取り上げる。それも、もっとも基本的な $f(x) = 2^x$ にしておこう。しかしその前に、**C++**における関数の仕組みを知るために、x の値を入力すると x^2 の値を表示するプログラムを組んでおこう。

programming list [xsquare.cpp]

```
 1: #include <iostream>
 2:
 3: int sqf(int x) {
 4:     return (x * x);
 5: }
 6:
 7: int main() {
 8:     int n;
 9:     std::cout << "input x of x^2 : "; std::cin >> n;
10:     std::cout << n << "^2 = " << sqf(n) << std::endl;
11:
12:     return 0;
13: }
```

今回のプログラムは、`#include <iostream>` の次に `main()` がない。`main(){}`は 7:行目以降に追いやられ、その前に `int sqf(int x){}`が割り込んでいる。プログラムは常に `main()` から始まる—つまり特別な存在である—ので、`int sqf(int x){}`の場所はどこでもよいけれど、先に書くほうが何かと便利なのでそうしている。

3:–5:行目の `int sqf(int x){}`が、関数 $f(x) = x^2$ の **C++**的表現である。細かく分析すると、左辺の $f(x)$ がプログラムの `int sqf(int x)` で、右辺の x^2 がプログラムの `return (x * x);` に対応している。x の平方（square）を求める関数（function）の意味で、関数名を `sqf` としただけだから、この関数名に特別な意味があるわけではない。数学で使う関数なら $sqf(x)$ だけで済むところが、プログラムでは扱う数値の型が大きな問題となる。そこで `sqf(x)` が整数値を受け取り、整数値を返すことを明確にするために `int sqf(int x)` と書かなくてはならないのだ。`int x` が受け取る変数の型、`int sqf()` が返す値の型を決めている。

数学では $f(x)$ の x に値が代入されたとき、右辺の x^2 により実質的な計算が行われる。プログラムも同じだ。関数 `sqf()` に具体的な値が代入されると、`{}` 内の `(x * x)` により実質的な計算が行われる。黙っていても値を返してくれないので、`return (x * x)` として値を返してやらなくてはならない。これが `int sqf(int x){}`の全貌である。

プログラムの本体は 7:行目から始まる `int main(){}`である。プログラムにさせたいことは、値 x を受け取り x^2 の値を出力することだけだ。

x の値を受け取る変数が必要になるので、8:行目で `int n;` を用意している。

9:行目が値 x をキーボードから値を受け取る場面だ。画面には "input x of x^2 : " と表示され、`std::cin` で値を待ち受ける。画面に x^2 と表示させるのは難しいので、`x^2` と書いて 2 乗の表現にすることが多い。ところが皮肉なことに、**C++**では `x^2` は 2 乗の表現ではなく、まったく別の意味の操作をすることになるのだ。詳しくは旅のずっと後で出会うだろう。だから x^2 は `x * x` で計算させたのである。

10:行目で、9:行目で受け取った値を出力している。例えばキーボードから 3 が入力されると、"3^2 = 9" が表示されるようになっている。これまでにも似たようなことをしてきたね。

TRY! 関数 $f(x) = x^3 - 2x^2 + 3x - 4$ の値を返すようにしてみよ。

TRY! このままだと `sqf()` は、整数値だけしか扱えない関数である。実数値が扱えるように書き換えよ。

関数の使い方を見て、`int main(){}`の中では、キーボードから受け取る変数が `n` に代入されるのだから、その値 `n` を受け取る関数は `int sqf(int n)` とすべきではないかと感じなかっただろうか。そして返される値は `return (n * n)` で返されるべきだと。もちろんそれは正しい考えだし、そうしてもプログラムは正常に動作する。だが、一般には同じ変数名を使わない。

理由のひとつに、関数が再利用を目的に作られていることをあげよう。関数に使う変数をあるプログラムと同じにすると、別のプログラムに同じ関数を使うときに違和感を感じるものなのだ。今回の関数を例にとれば、`int sqf(int n)` と書いてしまうと、関数 `sqf()` はどうしても整数関数の印象を受けてしまう。それは、文字 n が整数変数に使われることが多いからだ。関数 `sqf()` は実数変数でも問題なく動作する。変数を `n` から `x` に変えているのはそのためだ。