0. . . の旅

0.1　自然数から整数への継承

私たちはものを数えるとき

$$1,\ 2,\ 3,\ 4,\ 5,\ \ldots,\ 10,\ \ldots,\ 100,\ \ldots \tag{1}$$

と数えるのが普通である。これらの数はどこまでも大きく数えられるし、どこまでいっても終りというものがない。数学の世界では、ここに登場した数を**自然数**と呼んでいる。

まず、約束がなされたことに気づいただろうか。(1) の数を自然数と呼ぶ約束のことだ。ここで、何でそう呼ぶの？とか、0 は自然数じゃないの？とかの疑問が浮かぶかもしれないね。数学では “何々のことを何々と決める” ということが頻繁にでてくる。そしてその決め方が自然と納得できるものもあれば、場合によってはどうしてそう決めるのか不思議に思うことがあるだろう。約束—すなわち**定義**—というものは、大体の雰囲気で決める場合もあれば、深い意味があってそう決める場合がある。君たちが戸惑うのは、深い意味があって定義されることがらだろう。その場合は、定義に納得いかないこともあるはずだが、深い意味があるだけに当面は謎のままになってしまうものだ。その定義に関する内容を深く理解したとき、謎が氷解することが往々にしてあるので、楽しみはとっておくのがよいだろう。

自然数という呼び名は、大体の雰囲気から妥当と思われる。人間が自然発生的に使い出した数が 1, 2, 3, . . . だから。

ちょっと待ってほしい。われわれが現在、自然に使う数字は 0, 1, 2, 3, . . . ではないか。それなら 0 を含めて自然数と呼ぶほうが自然だろう。こう考える人はいるかな。たしかに、そのとおり。それは間違った考えではない。ただ、ここでは習慣に従い、(1) が自然数であると定義しておこう。あとで分かるように、0 は特別扱いしたい数だから。

ところで私たちが普段使う数には -1, -5 などもある。このような数は 0 より小さい数を表すために作られた数だ。自然界には 0 より小さいものはないといってよいだろう。何もない状態が 0 の状態だから、もうこれ以上なくなる状態はありえないのだ。ところが人々は 0 より小さい状態を概念として作ってしまった。借金や気温や水深などはそのよい例になっている。

$$\begin{array}{lcl} 5{,}000\text{ 円の借金} & \to & -5000\text{ 円} \\ \text{氷点下 } 8\ ^\circ\mathrm{C} & \to & -8\ ^\circ\mathrm{C} \\ \text{水深 } 120\ \mathrm{m} & \to & -120\ \mathrm{m} \end{array}$$

0 より小さい数は、自然数に − の**符号**をつけて表している。1 や 5 は、0 より 1 や 5 だけ大きい数ととらえるならば、−1 や −5 は、0 より 1 や 5 だけ小さい数ととらえることができる。数に − の符号をつけて、0 より小さい数を表すことは新たな約束となる。しかし、ここでは約束もさることながら、数の継承がなされたことに注意を払ってほしいのだ。

もし、負の数を新たに定義するだけなら a, b, c, ... という "数" を作れば済む。ところが実際は、自然数を土台に −1, −2, −3, ... と数を拡張している。つまり自然数を継承したわけだ。継承するということは、自然数の持つ性質も受け継ぐことを意味している。例えば数の間隔は、負の数でも 1 ずつだし、大きい数字を使うほど 0 から離れた数になることなどがそうだ。

結局、数は 0 を中心にして

$$\ldots,\ -10,\ \ldots,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ \ldots,\ 10,\ \ldots$$

と並んでいることになる。そしてこれらの数を**整数**と呼び、さまざまな数の基準として使うことにするわけである。

それではこの辺で、コンピュータに何かさせてみよう。

programming list [printout.cpp]

```
1: #include <iostream>
2:
3: int main() {
4:     int n;
5:     n = 123;
6:     std::cout << "a number is " << n << std::endl;
7:
8:     return 0;
9: }
```

`#include <iostream>` と `int main()` については後で詳しく触れるが、とにかくプログラムには必ず必要なものだと覚えておいてほしい。

このプログラムは、画面に "a number is 123" と表示し、**復帰改行**—改行してカーソル位置を行の先頭に移動—するだけのもので、そのことを記述したのが `{` と `}` の間に書かれた文である。プログラムは `{` と `}` の間をひとつのグループのように扱う。ここでは 4:–8:行の命令がひとまとめにされているのである。

まず 4:行目に `int n;` がある。これは**変数** `n` を整数値で扱うことを指示している。`int` は次に続く変数が、整数値であることを明示するためのキーワードになっている。従って `n` が `1.23` のよう

な小数になることはない。小数を扱うにはそのためのキーワードで指示する必要があるが、それは後で登場する。いずれにせよ変数を使うには、その変数が整数なのか小数なのかを明らかにしなくてはならない。`int` は `n` が整数**型**であることをコンピュータに知らせるキーワードなのだ。また `;` は**文**の終わりを示す記号で、この行には必要である。うっかり忘れるとコンピュータはエラーを返してくる。プログラムは多くの文が集まってできているので、大抵の命令の後には`;` が付くものだと考えておいてほしい。

5:行目には `n = 123;` がある。多くのプログラミング言語では、`=` は左辺と右辺が等しいことを意味しない。右辺の値を左辺の変数に**代入**することを意味する。従ってここでは、`n` に `123` を代入することを指示している。そして、この時点で `n` は `123` の値を持ったことになる。この行も文であるから、当然のごとく `;` が付いている。

さあ、6:行目に行こう。こいつは少々厄介だが一言で済ませるなら、`cout << "何がし"` は**標準出力**—今は Terminal の画面—に "何がし" を出力する命令である。`cout` の前に `std::`がくっついているのは、`cout` が **std 名前空間**に置かれているからで、要するに `std::cout` で 1 つの命令になっていると思えばよい。ぶっちゃけ、

```
std::cout << "何がし"
```

が文字を画面に表示するための書式である。ちなみに、`cout` は "characters output" を意味する。

すると今コンピュータは `std::cout << "a number is "`まで見て、画面に（is の直後の空白も含めて）"a number is " を出力したくてウズウズしている。でも、後ろに `<<` が続いているので、出力すべきものがまだあるのだ。それは `n` だ。しかし、`n` には `123` が代入されていたはずだから、実際は `123` に取って代わって出力されることになる。さらに `std::endl` が出力されるのだが、これは復帰改行を指示する命令だ。"end line" のことである。以上の結果、画面には "a number is 123" がさん然と輝き、Terminal は次のコマンドを受け付けるため待機することになる。

これで分かるように、例えば `std::cout << "a number is " + n + std::endl;` のような書き方で出力がまとめられることはない。複数のものを出力するには、`<<` を反復使用するのだ。

8:行目の `return 0;` は、今のところ最後に書くものである、程度の意識でよいだろう。**関数**については後で述べるが、今書いたのは `main` という名前の関数が実行されるプログラムである。数学で言うところの関数は、中学校でも勉強したであろう $y = 2x$ みたいな関係式を指す。$y = 2x$ なら $x = 5$ のとき $y = 10$ が返ってくるように、**C++**で言うところの関数も何らかの値を返すものである。`main` の前に `int` があるのは、関数 `main` が整数値を返すことを示している。で、この関数がしたことは文字列を出力したのであって整数値を返したのではない。そこで `return 0;` によって `0` を返したのである。

このように、ある部分ではプログラミングは易しいものだが、ある部分では非常に気難しい面も

ある。この本では気難しいことは省いておこう。まあ、それでも多少のプログラミング作法は身に付くだろうから。それに何が気難しいかは、ほんの一握りの好奇心があれば誰にでも体験できることだ。例えば今の例では、`n = 123;` の `n` の値をいろいろ変えたり、`return` 文を書かないとどうなるか試すことで、多少の好奇心は満たされるだろう。だが、好奇心は満たされても不満が募るかもしれない。不満の解消はしばらく待ってもらうしかないが。

TRY! 5:行目の `n = 123;` を `n = -123;` や `n = 1.23;` に変えて実行するとどんなことが起きるか?

TRY! 6:行目で、`<< std::endl` を記述しないとどうなるだろう。復帰改行の意味がよく分かるはずだ。