0.2 無限

さあ整数が出揃って、いよいよ数学の世界の奥へ入っていく準備ができた。数は整数の他にも、様々な種類のものがあることは知っているだろう。続けていろいろな種類の数を紹介したいけれど、まだ整数についてさえ十分に語っていないのだ。それは何かって？　それは "..." の部分。特に "..., −10" と "10, ..." のところは、単なる省略ではないことを言いたいのである。

ここに使われた "..." は、この先の数を書く意味がないことを示している。なぜなら、書き切ることができないから。当たり前だよね。ところが "..." には、思いもかけない不思議が詰まっている。

君たちは**偶数**と**奇数**は知っているね。自然数に限れば 2, 4, 6, ... が偶数、1, 3, 5, ... が奇数ということになる。ものの集まりを**集合**と呼ぶことにして、偶数の集合を $\mathcal{E}$、奇数の集合を $\mathcal{O}$、そして自然数の集合を $\mathcal{N}$ で表せば

$$\mathcal{E} + \mathcal{O} = \mathcal{N} \tag{1}$$

であることが想像できる。でも、残念ながら違う。想像が事実に合わないことはよくある話。(1) は偶数や自然数のような、**無限集合**にはそぐわない関係なのだ。それについてカントール[1]は次のように説明している。

例えば "りんごが 5 つある" とは、各りんごに自然数 1, 2, 3, 4, 5 が **1 対 1 に対応**していると見る。このとき、りんごの数 (かず) は 5 に等しいと言える。同様のことを偶数と自然数で考えてみよう。すると

$$\begin{array}{lcccccc} \text{偶数} & 2 & 4 & 6 & 8 & 10 & \cdots \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \cdots \\ \text{自然数} & 1 & 2 & 3 & 4 & 5 & \cdots \end{array}$$

のように、各偶数に自然数 1, 2, 3, 4, 5, ... が 1 対 1 に対応していることになる。従って偶数と自然数は同じ数 (かず) だけある、と考えるわけだ。このときに数 (かず) ということばは適切でないように思えるので、カントールは**濃度**という用語を使っている。すると奇数に対しても、奇数と自然数は同じ濃度を持つと言えるのだ。

この用語はまったくもって適切で、私たちの生活感覚にも合っている。なぜなら、同じ濃度の食塩水を混ぜると、同じ濃度を持つ食塩水ができるのだから。そして食塩水の "量" も増えているので、自然数も偶数だけの "量" に比べると、きっと奇数の分だけ "量" が増えているんだろう。

ちょっと不思議な感覚だけど、日常からかけ離れたものにはありがちなことではある。(想像を超える大金持ちの資産) + (想像を超える大金持ちの資産) は、やはり (想像を超える大金持ちの資産) であるのと同様、"..." は私たちの想像を超えた世界ということなのだ。

[1] ゲオルク・カントール (1845–1918)：ドイツの数学者。

それでは、コンピュータは無限にどう対処しているのだろう？　ここで簡単な調査をしておこう。

programming list [parrot.cpp]

```
1: #include <iostream>
2:
3: int main() {
4:     int n;
5:     std::cout << " input N : "; std::cin >> n;
6:     std::cout << "output N : " << n << std::endl;
7:
8:     return 0;
9: }
```

`#include <iostream>` と `main(){}` の部分は、前回と同じく必ず必要になるところだ。

4:行目の `int n;` で整数値が扱える変数 `n` を用意している。

5:行目には `std::cout` と `std::cin` に関する 2 つの文が並列に書いてある。文の区切りは `;` で明確に分かるので、このように 1 行に 2 つ以上の文を書くことも可能だ。実は、1 行 1 文で書くことが望ましいと思うが、`cin` にはお知らせ用の文があるとよいので、あえて 1 行 2 文で書いてあるのだ。

まず、`std::cout` は前節で見たように単に “ input N : ” だけが表示される。`std::cout` 文は `""` 内の文字を素直に表示するので、`input N :` の前後の空白も表示対象になっていることに注意してほしい。なぜこんなものを表示したかといえば、その後の `std::cin` はキーボードからの入力を待つ命令なのだが、まったく馬鹿正直に入力を待つことしかしない。つまり、あらかじめ “ input N : ” のようなお知らせをしない限り、コンピュータがどんな状態でいるか分からないからだ。ちなみに “ input N : ” の前に空白があるのは、次の “output N : ” と文字の位置をそろえたかっただけだ。

さて、その `std::cin` だが、書式は `std::cout` と似ている。ただ、`>>` が逆を向いていることに注意してほしい。キーボードから受け取った値を変数 `n` に代入するのだから、`>>`向きは自然な感覚だろう。変数 `n` に代入された値は、6:行目の `std::cout` へその処理が渡される。出力の仕方は前と同じ書き方だね。

実行すれば分かるが、このプログラムは入力した数を出力するだけだ。本当に？　どうか、いろいろな数を入力してもらいたい。いつでも納得できる結果になっただろうか？

TRY! “ input N : ” の入力要求に対して、いろいろな整数を入力してみよ。特に、32768, -32768 近辺の整数を入力すると何が表示されるか。もし、これで何の変哲もなければ 2147483648, -2147483648 近辺の整数で試してほしい。

それにしても、このプログラムは何て気が利かないと感じるだろう。たったひとつの数を入力し、それを出力するだけなのに、そのたびにプログラムを実行しなくてはならない。どうして連続

して入力をさせてくれないのだろう。ひとつの解決方法は while(1){}によってプログラム文を囲むことである。

programming list [parrot2.cpp]

```
 1: #include <iostream>
 2:
 3: int main() {
 4:     int n;
 5:     while(1) {
 6:         std::cout << " input N : "; std::cin >> n;
 7:         std::cout << "output N : " << n << std::endl;
 8:     }
 9:
10:     return 0;
11: }
```

しかし、この方法はちょっと危険を伴う。なぜ危険かといえば、while(1){}という書式は、{}内の命令を永遠に繰り返すからである。このことは、{}内に重大なエラーを発生させる要因があると、プログラム全体が暴走する危険があるからだ。従って while(1){}で囲む前に、{}内のプログラムが問題なく動くことを確認することは特に重要だ。そしてもっと重要なのは、プログラムが暴走したり**無限ループ**に入り込んだとき、それを止める方法を知っていることである[2]。

さて、プログラムはすべての文を while(1){}で囲んでいないことに注意してほしい。int n; が外に出ている。繰り返したいのは、数の入力と出力の場面である。従って変数 n は、最初に一度だけ用意すればよいので while(1){}の外にある。また、while(1){}の最後—8:行目—に ; を書いてないことにも注意をしてほしい。while(1){}はグループ化をしているのであって文ではないのだ。もっともここに ; があってもエラーにはならない。しかし 11:行目の } の後ろに ; を書いてしまうと、コンピュータは無情にもエラーを返してくれる。ああ、なんて融通の利かない奴なんだろう。

さらに付け加えると、while(1){}が囲んだのは 6:–7:行だから、while(1){ に対応する }—閉じカッコ—は 8:行目のものである。11:行目の } が 5:行目の {—すなわち main() が囲んでいるカッコ—に対応している。このように{}は、必ず**入れ子**の構造をとるのだ。また、while(1){}で 6:–7:行を囲むと同時に字下げも行っている。字下げは見やすさを考えてのことだ。字下げをしなくてもプログラムは正常に動作するが、やはり見やすい書き方を心がけるべきだろう。

[2]Terminal の場合は、ctrl+c または ctrl+z で終了または停止となる。