計算機プログラミング I (2)

変数とデータ型

2000 年 10 月 17 日 増原 英彦

練習問題について

練習問題を解いた人は、ニュースグループlectures.g00.cp1-masuhara-W-Tue-5 に投稿するか、電子メールでmasuhara-cp1@lecture.ecc.u-tokyo.ac.jp まで送って下さい。優秀作品を講義の web ページに掲示します。

1 計算式

練習 1-4 に出てきたSystem.out.println(2+4); のように、数式を (「"」で囲まずに) 書くと計算が行われる。

例えば、「10 万円を八十二銀行のスーパー定期で 10 年間預けたときの残高はいくらになるか?」という問題を考える。10 年定期の利率は 0.45% だそうなので、電卓を使うのであれば以下のように計算するだろう。

$$100,000 \times \overbrace{1.0045 \times \cdots \times 1.0045}^{10}$$

従って、プログラムとしては次のようなものを書けばよかろう。

```
1  class Saving { // 預金残高の計算
2    public static void main(String[] args) {
3      // 10 万円を年利 0.45%で 10 年間預けたときの金額
4      System.out.println(100000 * 1.0045 * 1.0045 * 1.0045 * 1.0045 * 1.0045 *
5                                  1.0045 * 1.0045 * 1.0045 * 1.0045 * 1.0045);
6    }
7  }
```

(4,5 行目の「*」は Java 言語における掛け算。なお、以降の例ではクラスやメソッドの宣言 (1,2 行目) を略す。)

練習 **2-1:** (計算) 半径 15 の円と半径 3.14 の円があったとする。このとき、それぞれの円の面積と円周を計算するプログラムを書け。ただし、円周率は 3.14 だとする。

2 変数

さて、ここで「昔の郵便局の定額預金は年利 6 % もあったんだってね。もしそのときに預けていたら、いくらになったのかな?」と思ったとしよう。その場合、上のプログラムの 4,5 行目を修正して下のようにすればよい。

```
// 10 万円を年利 6%で 10 年間預けたときの金額
System.out.println(100000 * 1.06 * 1.06 * 1.06 * 1.06 * 1.06 *
                           1.06 * 1.06 * 1.06 * 1.06 * 1.06);
```

しかし、このやり方では、金利を変える度にプログラムを 10 箇所修正しなければならない。似たような例はいくらでもある。例えば、練習 2-1 をもっと精度よく計算するために円周率を 10 桁にして計算したくなったという場合もそうだろう。さらに言えば、プログラムの中にたまたま半径が 3.14 の円があった場合、それを円周率だと思って変えてしまう危険もある。

一般化すると、次のようなことが言える:

> 与えられた問題の中で**1** つの値として用いられているもの (上の例で言えば利率や円周率) は、後からまとめて変えられるように、プログラム中でも **1** つの値として扱うべきだ。

プログラミング言語では、変数を使うことで、値に名前をつけることができる。これによって、上の問題が解決できる。Java 言語では、プログラム中で変数を使う場合には、変数の種類・名前・値を宣言しておく必要がある。また、数式中に変数の名前を書けば、そこに値を書いたのと同じことになる。

以下は、最初の例の 3–5 行目を変数を使って書き直したものである。

```
1  double principal = 100000; // 元金
2  double interest = 1.0045; // 10 年定期の年利
3  // 10 年後の金額
4  double balance = principal * interest * interest * interest * interest
5                             * interest * interest * interest * interest
6                             * interest * interest;
7  System.out.println(balance);  // 表示
```

- 1 ~ 6 行目は変数の宣言である。
  - double は宣言されている変数に入れる値の種類が倍精度浮動小数点数であることを表わしている。どのような種類があるかは 3 節で説明する。
  - principal, interest は変数の名前である。変数の名前は、
    - 先頭がアルファベットまたは _ (下線)
    - 2 文字目以降がアルファベット、_ (下線)、数字のどれか

    であれば何でもよい。例えば、interest, MyAccountBalance, price_of_PlayStation2 などは全て変数名として使える。

    もちろん変数名として i や x のようなものも使えるが、その値が表わす意味が分かるようにしておくことが望ましい。
  - 「=」の後ろは、その変数が持つ値。
  - 最後の「;」は宣言の終りを表わす。
- 4 ~ 7 行目では、変数を使った式がある。式を計算するときは、principal, interest という変数名はその変数が持っている値 (ここでは 100000 や 1.0045) とみなされる。

このように、数値を直接書かずに変数を用いることで、ある値が使われている場所を明確にすることができる。そのため、プログラムを人間が理解することや、後から値を変更することが簡単になる。

練習 **2-2:** (変数)　練習 2-1 で作ったプログラムを変数を使って書き直し、円周率を 3.1415926535 としてみよ。

3　データ型

コンピュータが扱うことのできる情報には、数値・文章・音・映像など様々な種類がある。プログラム中でデータを扱う場合には、どのような種類の情報であるかを明示して扱う必要がある。このような情報の種類のことをデータ型とよぶ。例えば、変数を宣言する場合には、「その変数にはどのような種類の情報を入れるか」を与め宣言する必要がある。

3.1　情報の表現

様々な種類の情報は、コンピュータの内部では、数値 (の並び) として扱われている。例えば、「世」という文字は 19990 という数値[1]として、「世界の国からこんにちは。」という文字の並び (文字列と呼ぶ) には、19990, 30028, 12398, 22269, 12363, 12425, 12371, 12435, 12395, 12385, 12399, 12290 という数値の並びとして処理されている。

また、画像や音のようなアナログ情報は、点の明るさや一定時間間隔の音圧などを、有限の整数の並びで表わしている。このことは、音楽を記録した CD と「ファイナルファンタジー」を記録した CD-ROM が同じメディアを使っていることからも想像がつくだろう。

結局、様々な情報は数値の (並び) として表わされる。この数値は、ハードウェア (電子回路) 上では、電気信号の有無として表わされている。その方法を見てみよう。

電気信号の有無/コインの表裏/「理科」と「文科」のように二者択一的な情報の単位をビット(bit) という。通常、1 ビットの情報は 0 と 1 を使って書き表す。より複雑な情報は、複数のビットを使って表わす。

例えば、理科 1 類 ~ 3 類、文科 1 類 ~ 3 類までの 6 つの科類を表わすのであれば、3 ビットあれば充分で、000=理科 1 類, 001=理科 2 類, 010=理科 3 類, 011=文科 1 類, 100=文科 2 類, 101=文科 3 類のようにする。(これは例であって、他の割り当て方も考えられる。また、110 と 111 は使っていないことに注意。) 同様にして、「1 から 100 までの整数」も何ビットかを使って表わすことができるだろう。

整数をビットの並びで表わすやり方で最もよく使われているものは、その数を**2** 進数によって書き、各桁の 0,1 をビットに対応させるものである。2 進数で書かれた数、例えば 01101 は、下から $0, 1, 2, 3, \cdots$ 桁目が (10 進数で) $2^0, 2^1, 2^2, 2^3, \cdots$ の重みを持っているので、

$$0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$$

となる。一般化すると、n 桁の 2 進数 $b_{n-1}b_{n-2}\cdots b_0$ は $\sum_{i=0}^{n-1} 2^i b_i$ という数を表わしている。

負の数も表わすようにするためには、n ビットを使って、(1) 0 から $2^{n-1}-1$ までの正の数は、そのまま 2 進数に直したもので (2) -1 から -2^{n-1} までの負の数 x は、$(2^n - x)$ を 2 進数に直したもので表わす。(この方法を**2** の補数表現という。) 例えば、3 ビットで表わす場合は

[1] このように、文字に数値を対応させる規則を 文字コード体系 という。英文字の文字コード体系には ASCII, EBCDIC などが、日本字のコード体系には JIS, シフト JIS, EUC などがある。最近では、各国の文字を一つのコード体系にまとめたものもあり、Java 言語ではそのようなコード体系である Unicode を使うことができる。

10 進数	-4	-3	-2	-1	0	1	2	3
2 進数	100	101	110	111	000	001	010	011

という対応になる。

実数の表わし方　コンピュータで実数を扱う場合は、浮動小数点数という方法がとられることが多い。これは、工学や物理学の世界で 3.0×10^8 のように、仮数 (3.0) と指数 (10^8) という 2 つの数によって表わすものである。実際の表わし方には様々なものがあるが、仮数に 23 ビット、指数に 8 ビット、符号 (正負) に 1 ビットの合計 32 ビットで表わす単精度浮動小数点数や、合計 64 ビットで表わす倍精度浮動小数点数が使われることが多い。

3.2　基本データ型

上で見たように、数を表わす方法だけでも沢山の方法がある。一般に、沢山のビット数を使う表現をとれば、より広い範囲の値を扱うことができる。しかし、必要以上に沢山のビット数を使うと、それだけ扱うデータの量や計算時間が増えるため、適切な方式の表わし方を選ぶことが重要である。

プログラム言語には、変数や式の値がとり得る範囲と、その表し方を示す概念として、データ型(data type) というものがある。特に、数値や文字などの基本的なものは、基本データ型と呼ばれる。Java 言語には次のような基本データ型がある。

名前	種類	ビット長	値の範囲
byte	整数	8	-128 から 127
short	整数	16	-32768 から 32767
char	整数 (文字)	16	0 から 65535 (に対応する文字)
int	整数	32	約 -21 億 から 約 21 億
long	整数	64	約 -922 京 から 約 922 京
float	実数	32	約 $\pm 3.40282347 \times 10^{\pm 38}$ の範囲
double	実数	64	約 $\pm 1.79769313486231570 \times 10^{\pm 308}$ の範囲
boolean	真偽値	1	true か false

練習 2-3: (桁あふれ)　Java 言語では、整数の計算は 32 ビット (`int` 型) で行われるのが普通である。(例外: `long` 型を使った計算。) 計算結果が 32 ビットを超える値になったとき、何が起きるかを調べよ。

3.3　複合データ型

x 座標と y 座標の組、文字列、行列のように、基本データ型を組み合わせて作られている値を扱うための型は複合データ型と呼ばれる。

例えば、`public static void main(String[] args)` の下線部は、「String 型の配列」という型を表わし、String 型は文字の並びを 1 つのデータとして扱うための型である。これらについては後の回で説明する。

3.4 変数の型

色々な型のデータも、コンピュータの内部では 0 と 1 のビット列になっている。そのため、内部のデータを見ただけでは、それが整数なのか実数なのか文字なのかは分からない。

そのようなビット列を「どの型の値」と見なして扱うかを指示するために、変数を宣言する際には型を一緒に与える必要がある。

そのように指定した型の変数には、その型に適合する値しか入れることができない。従って、

```
int pi = 3.14; // 円周率?
```

のような宣言はエラーになる。一方、

```
int numberOfCups = 3;                                  // コップ3個
double volumePerCup = 0.3;                             // コップの容積
double totalVolume = numberOfCups * volumePerCup; // 全容積
```

のような宣言はエラーにならない。これは、ある型の値を「その型を含むようなより広い型」の値だと思っても問題がないため、自動的に変換してくれるようになっているためである。

練習 **2-4:** (型の広さ) `int` 型の変数に `double` 型の値を入れてもエラーになるが、その逆はエラーにならない。このようにエラーにならない組み合わせはどのようなものがあるかを調べよ。

4　アルゴリズムの考え方

例題:　2 進数を 10 進数に直す方法を考え、プログラムを作る。ただし、元の 2 進数が「1101」であれば、下のように 4 つの変数によって与えられるものとする。また、負の数は考えない。

```
int bit3 = 1; // 2進数 1101 の 3桁目
int bit2 = 1; //              2桁目
int bit1 = 0; //              1桁目
int bit0 = 1; //              0桁目
```

このような複雑な問題を解く場合には、

問題を小さな問題に分割し、それぞれを解く

のが基本的な方法である。

問題の分割:　上で見たように、2 進数 $b_{n-1}\cdots b_0$ は、$\sum_{i=0}^{n-1} 2^i b_i$ という数を表わしている。この計算をするには、(1) 各 i について $2^i b_i$ を計算する部分 (2) 全ての i についての和を求める部分についてそれぞれ考えればよい。

$2^i b_i$ の計算:　単純には、以下のようにすればよい。

```
int base_i = 2 * 2 * ··· * 2 * 1; // i桁目の基数
int decimal_i = base_i * bit_i; // i桁目の値
```

和を求める。　これも単純に、それぞれの桁を足し合わせて以下のようにすればよい。

```
// 各桁の和
int decimal = decimal_{n-1} + decimal_{n-2} + ··· + decimal_0;
```

結局、プログラムとしては、図 1 のようになる。

プログラムの工夫　上のプログラムでは、i 桁目の基数を求めるのに、それぞれ 2 を i 回掛けていた。4 桁の 2 進数ぐらいであれば気にならないが、何十桁もの 2 進数を変換しようという場合、プログラムを書くのも大変になるし、同じ計算を何度行うという無駄がある。

ここで、$\mathrm{base}_{i+1} = 2 * \mathrm{base}_i$ の関係があることに注目すると、base_{i+1} を求めるときには、前に計算した base_i の値を使うことができるため、i 桁目で計算した基数を流用できる。よって、プログラムは次のように改良できる。

```
int base0 = 1;                // 0桁目の基数
int decimal0 = base0 * bit0; //       の値
int base1 = 2 * base0;        // 1桁目の基数
int decimal1 = base1 * bit1; //       の値
int base2 = 2 * base1;        // 2桁目の基数
int decimal2 = base2 * bit2; //       の値
int base3 = 2 * base2;        // 3桁目の基数
int decimal3 = base3 * bit3; //       の値
```

```
1  int bit3 = 1; // 2進数 1101 の 3桁目
2  int bit2 = 1; //               2桁目
3  int bit1 = 0; //               1桁目
4  int bit0 = 1; //               0桁目
5
6  int base0 = 1;               // 0桁目の基数
7  int decimal0 = base0 * bit0; //      の値
8  int base1 = 2 * 1;           // 1桁目の基数
9  int decimal1 = base1 * bit1; //      の値
10 int base2 = 2 * 2 * 1;       // 2桁目の基数
11 int decimal2 = base2 * bit2; //      の値
12 int base3 = 2 * 2 * 2 * 1;   // 3桁目の基数
13 int decimal3 = base3 * bit3; //      の値
14
15 // 各桁の和を計算して表示
16 int decimal = decimal3 + decimal2 + decimal1 + decimal0;
17 System.out.print("10進数では"); System.out.println(decimal);
```

図 1: 2 進数を 10 進数に変換するプログラム

練習 2-5: (掛け算の回数) n 桁目の 2 進数を変換する場合、最初のプログラムと改良後のプログラムでは、掛け算が何回行われるか?

練習 2-6: (10 進数を 2 進数に変換する) 10 進数の値を n 桁の 2 進数に変換する方法を考え、プログラムを作れ。ただし、もとの数は

```
int decimalNumber = 123; // 10進数の値
```

のようにして与えられるものとする。

ヒント: (1) ある数を 2 進数にしたときの 0 桁目は、その数を 2 で割ったときの余りになる。(2) ある数を 2 進数にしたときの 1 桁目以上 (0 桁目以外) は、その数を 2 で割った商を 2 進数にしたもの。という性質を利用して、以下のようにすればよい:

- 2 進数の 0 桁目をもとの数から求める
- 2 進数で 1 桁目より上に対応する (10 進) 数を計算する
- 2 進数の 1 桁目をもとの数から求める
- 2 進数で 2 桁目より上に対応する (10 進) 数を計算する
- ...

5　アンケート (第2回 10月17日)

※ 線から下を切り離し、あてはまるところに丸をつけ、教卓に提出して下さい。

1. 今日の授業内容は
   a. ほとんど知っていた　b. だいたい知っていた　c. あまり知らなかった　d. ほとんど知らなかった
2. 授業の進度は　a. 速い　b. やや速い　c. 丁度よい　d. やや遅い　e. 遅い
3. 自分にとって授業のレベルは　a. 高い　b. やや高い　c. 丁度よい　d. やや低い　e. 低い
4. その他、自由に感想を書いて下さい
   (

)

ありがとうございました。可能な限り、次回以降の授業に反映します。