計算機プログラミング'99　＃2

久野　靖*

1999.9.8

1　はじめに

今回はまず、前回の演習の例解を読んでみましょう。プログラムを多く書き、多く読むことが上達のコツです。また、それぞれの例題でのポイントも (あれば) 挙げてみましょう。

演習 6b

6a は 6b ができれば同じことだから略。疑似コードは次の通り。

- 実数 x、y を入力する。
- x+y、x-y、x*y、x/y を出力する。

で、Java では次のようになる。

```
import java.io.*;

class r1ex6b {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("x = "); System.out.flush();
      String line = in.readLine();
      double x = (new Double(line)).doubleValue();
      System.out.print("y = "); System.out.flush();
      line = in.readLine();
      double y = (new Double(line)).doubleValue();
      System.out.println("x + y = " + (x+y));
      System.out.println("x - y = " + (x-y));
      System.out.println("x * y = " + (x*y));
      System.out.println("x / y = " + (x/y));
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

*筑波大学大学院経営システム科学専攻

自分が作ったのと違う、と思いました? もちろん、それで当然であり、どっちが正解ということはない。ちょうど日本語で同じことを言うのにさまざまな言い方があるのと同じこと。ただし、スマートだとか短いとか分かりやすいとかそういった点で違いはあるかもしれない。美観の問題! だから自分の好みもとりまぜて、考えて選ぶこと。

その他注意してほしい点:

- まず、字下げ (左側に空白を入れること) をしない人がいるけど、これは絶対やめた方がよい。たとえば、

```
if(...) {       if(...) {
  ....            ....
  ....            ....
  ....            ....
}               }
```

  左のように書いてあれば、どこまでが「もし〜ならば」の範囲かすぐ分かる。しかし右のようにべったり揃えてしまうとそれが分からないし、たとえば間違って「}」をつけ忘れたり別の場所に入れてしまうともう大混乱になる。たかがスペースキーを打つ手間を惜しんで後で間違い探しに 1 時間も掛けるのは阿呆みたいでしょ? プログラムは「美しく」書こう。

- 入力のとき、まず読んだ 1 行を変数 `line` に入れているけど、x を読む時と y を読む時で同じ `line` を使っている。この場合は変数が 1 つなので 2 回目は宣言が要らない (つまり頭の「`String`」というのが不要)。もちろん、1 回目と 2 回目で別の変数を使ってもよい。変数が少ないほど効率がよい? と思うかも知れないけど、こんなのはほとんど (または全く) 関係ない。美しい方を選ぼう。

- 実はいったん変数 `line` に行を入れなくても、

```
Double x = (new Dobule(in.readLine())).doubleValue();
```

  のように 1 行で済ませることもできた。その方が好きな人はそうしてもよい。

- 和、差、商、積をいったん変数に入れていない。もちろん、前回の例題のように変数に入れてもよいが、上のようにいきなり出力してもよい。一般に変数が書けるところには代りに任意の計算式を書いてもよい。

- ところで、式を丸かっこで囲んでいるのは、「`"...." + x + y`」とすると足し算は行われず全部文字列として連結されてしまうから。

演習 6c

6c は 6b と似たようなもんですね。疑似コードは次の通り。

- 実数 r と h を入力する。
- $\frac{\pi r^2 h}{3}$ を出力する。

Java は次の通り。今度は入力を 1 行にしてみました。その方が短くていい?

```
import java.io.*;

class r1ex6c {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("r = "); System.out.flush();
```

```
      double r = (new Double(in.readLine())).doubleValue();
      System.out.print("h = "); System.out.flush();
      double h = (new Double(in.readLine())).doubleValue();
      System.out.println("volume : " + (Math.PI * r * r * h / 3.0));
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

2 枝分かれについて

演習 7a

7a は 2 つの枝分かれですから例題とほとんど同じ。まず疑似コード。

- 実数 a、b を入力する。
- もし a > b であれば、
- 　 a を出力する。
- そうでなければ、
- 　 b を出力する。
- 枝分かれおわり。

Java では次の通り。

```
import java.io.*;

class r1ex7a {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("a = "); System.out.flush();
      double a = (new Double(in.readLine())).doubleValue();
      System.out.print("b = "); System.out.flush();
      double b = (new Double(in.readLine())).doubleValue();
      if(a > b) {
        System.out.println("larger : " + a);
      } else {
        System.out.println("larger : " + b);
      }
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

しかし、次のような「別解」はどうだろう。

- 実数 a、b を入力する。
- max ← a
- もし b > max であれば、

- 　max ← b
- 枝分かれおわり。
- max を出力する。

これの Java 版は次のとおり。

```
import java.io.*;

class r1ex7a1 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("a = "); System.out.flush();
      double a = (new Double(in.readLine())).doubleValue();
      System.out.print("b = "); System.out.flush();
      double b = (new Double(in.readLine())).doubleValue();
      double max = a;
      if(b > max) {
        max = b;
      }
      System.out.println("larger : " + max);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

どっちが好みですか? これもどちらが正解ということではない。

演習 7b

7b はもうちょっとややこしい。まず考えるのは、a と b の大きい方はどっちか決めて、それぞれの場合についてそれを c と比べるというもの。

- 実数 a、b、c を入力する。
- もし a > b であれば、
- 　もし a > c であれば、
- 　　a を出力する。
- 　そうでなければ、
- 　　c を出力する。
- 　枝分かれおわり。
- そうでなければ、
- 　もし b > c であれば、
- 　　b を出力する。
- 　そうでなければ、
- 　　c を出力する。
- 　枝分かれおわり。

- 枝分かれおわり。

うーむ大変だ。これを Java にしておく。

```
import java.io.*;

class r1ex7b {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("a = "); System.out.flush();
      double a = (new Double(in.readLine())).doubleValue();
      System.out.print("b = "); System.out.flush();
      double b = (new Double(in.readLine())).doubleValue();
      System.out.print("c = "); System.out.flush();
      double c = (new Double(in.readLine())).doubleValue();
      if(a > b) {
        if(a > c) {
          System.out.println("largest : " + a);
        } else {
          System.out.println("largest : " + c);
        }
      } else {
        if(b > c) {
          System.out.println("largest : " + b);
        } else {
          System.out.println("largest : " + c);
        }
      }
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

こうなると字下げしてないとごちゃごちゃになるでしょう? しかし字下げしてあってもこれはかなり苦しい。ときに、先の別解から発展させるとどうだろう?

- 実数 a、b、c を入力する。
- max ← a
- もし `b > max` であれば、
- 　　max ← b
- 枝分かれおわり。
- もし `c > max` であれば、
- 　　max ← c
- 枝分かれおわり。
- max を出力する。

この方がすっきりしているでしょう? Java でも次のとおり。

```
import java.io.*;

class r1ex7b1 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("a = "); System.out.flush();
      double a = (new Double(in.readLine())).doubleValue();
      System.out.print("b = "); System.out.flush();
      double b = (new Double(in.readLine())).doubleValue();
      System.out.print("c = "); System.out.flush();
      double c = (new Double(in.readLine())).doubleValue();
      double max = a;
      if(b > max) {
        max = b;
      }
      if(c > max) {
        max = c;
      }
      System.out.println("largest : " + max);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

今度はどちらが好みですか? 一般には、枝分かれの中に枝分かれを入れるよりは、枝分かれを並べるだけで済ませられればその方が分かりやすいといえる。ただし次の節も参照。

3 多方向の枝分かれ

演習 7c

7c は 3 通りに分かれるのだから、if の中にまた if が入るのはやむを得ない。しかし、ちょっと工夫すると分かりやすくなる。Java コードから見ていただく。

```
import java.io.*;

class r1ex7c {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("x = "); System.out.flush();
      double x = (new Double(in.readLine())).doubleValue();
      if(x > 0.0) {
        System.out.println("positive.");
      } else if(x < 0.0) {
        System.out.println("negative.");
      } else {
```

```
        System.out.println("zero.");
      }
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

これは実は最初の if の else のすぐ後ろに次の if がくっついた、という形をしているが、こういうパターンを利用するとプログラムが分かりやすくなる。順序が前後したが、疑似コードだと次のようになる。

- 実数 x を入力する。
- もし `x > 0` ならば、
- 　　「positive.」と出力。
- そうでなくて `x < 0` ならば、
- 　　「negative.」と出力。
- そうでなければ、
- 　　「zero.」と出力。
- 枝分かれおわり。

一般に「そうでなくて〜ならば、」は何回現われてもよい。またそのどれもが成り立たない場合は「そうでなければ」に来るが、この部分は不要ならなくてもよい。これを Java にする場合は次の形になる。

```
if(...) {
  ...
} else if(...) {
  ...
} else if(...) {
  ...
} else {
  ...
}
```

なお、これはあくまでも Java の if 文の「else の後にすぐ次の if をくっつけた」というパターンなだけで、特別な文というわけではない。

4　繰り返しについて

演習 8a

これは単に 2 ずつ引いていけばよい。

- 整数 x を入力する。
- `x > 1` である間繰り返し:
- 　　`x ← x - 2`
- ここまで繰り返し。
- もし $x = 0$ ならば、
- 　　「偶数」と出力。
- そうでなければ、

- 「奇数」と出力。
- 枝分かれ終わり。

Java で書くと次の通り。

```
import java.io.*;

class r1ex8a {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("x = "); System.out.flush();
      int x = (new Integer(in.readLine())).intValue();
      while(x > 1) {
        x = x - 2;
      }
      if(x == 0) {
        System.out.println("even.");
      } else {
        System.out.println("odd.");
      }
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

いくつか注意を。

- なお、資料に書いてあったのだが、「等しい」は Java では「==」で表す。ただの「=」だと「変数に値を入れる」意味になってしまうので注意。
- ところで、繰り返しの条件に注意。「x > 1 の間繰り返す」という条件で、しかも繰り返しの中では x は小さくなる方向に変化するから、いつかは x が十分小さくなってこの繰り返しは確実に止まる。さらに、繰り返しが止まったときは「$x \leq 1$」が成り立っているから、つまり x は 1 であるか 0 であるかのいずれか、になっている。なのでそのどちらかを調べれば奇数か偶数かが分かる。このように、繰り返しを使う時には
  - 最終的に成り立って欲しい条件を決め、
  - 繰り返しの内側ではその条件が成り立ちやすい方向に変化を起こす

  ようにする。これらが守られていないと、繰り返しが無限に続いたり、繰り返しが終わった時に役に立つ値が計算されていない、ということになる。
- たとえば、人生で金持ちになるための手順も同様である。
  - 手持ちの金額が G 円未満である間繰り返し、
  - 手持ちの金を増やす※。
  - ここまで繰り返し。

  ただし問題は、人生では※を確実に行う手順が知られていないことである。
- ところで、x が正の整数でないと、当然このプログラムは正しく動作しない。それでいいのか、と思うかも知れないが、それは構わない。プログラムを作る時はあくまでも「プログラムが動作すべき条件が

満たされた時に正しく動作するプログラムを作る」ことだけが求められる。余計なオマケのために労力を費すのがいいかどうかは必ずしも分からない。

ただし、自分でプログラムの動作を定める時には、正の整数以外でも扱いたいかどうか、十分考えてよいと思うように決めること。

5 for 文による繰り返しについて

演習 8b

8b は要するに y を x 回足せばよい。これを行うのにいろいろな方法があるが、普通はもう 1 つ変数を用意して、これを 1 ずつ足しながら回数を数えていく。

- 整数 x、y を入力する。
- seki ← 0。
- i ← 0。
- `i < x` である間繰り返し:
- 　seki ← seki + y。
- 　i ← i + 1。
- ここまで繰り返し。
- seki を出力する。

これを Java にすると次の通り。

```
import java.io.*;

class r1ex8b {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("x = "); System.out.flush();
      int x = (new Integer(in.readLine())).intValue();
      System.out.print("y = "); System.out.flush();
      int y = (new Integer(in.readLine())).intValue();
      int seki = 0;
      int i = 0;
      while(i < x) {
        seki = seki + y;
        i = i + 1;
      }
      System.out.println("product is: " + seki);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

ここでもいくつか注意を。

- 上のコードでは「x 回繰り返す」のに、i を 0、1、2、…、$x-1$ まで変化させている。このように「0 から数える」のは計算機ではよく使う。

- まず、変数 seki を最初 0 にしておき、次に y ずつ増やしていることに注意。合計を求めるには「`seki = seki + y`」のようにして「足し込んで」行くのが定石。なお、これはよくでて来るパターンなので「`seki += y`」と書くこともできる。他に「引き込む」「掛け込む」等もあり。
- 特に「1 足す」「1 引く」はよく使うのでさらに短く「`++i`」「`--i`」と書くこともできる。

さて、上で出て来た i のように「数を数える」ための変数を「カウンタ」と呼ぶ。カウンタを 0、1、2、…、$N-1$ まで変化させて繰り返す、というのはとてもよく使うので疑似コードで次のように書くことにする。

- 変数 i を 0 から $N-1$ まで変化させながら繰り返し:
- 　 ....
- 繰り返し終わり。

また、Java でもこの部分は while 文の代りに for 文を使って書くことが多い。

```
for(int i = 0; i < n; ++i) {
    ....
}
```

for 文は実は while 文の「親戚」で、ただしカウンタ用の変数の初期設定と毎回の更新 (加算) を一緒に書いて見やすくできる、というだけのものである。なお、上のようにカウンタ変数の宣言を for の中に入れた場合は、その変数は for の内部でのみ使える。

これを使って先の例題を書き直すと次の通り。

- 整数 x、y を入力する。
- seki ← 0。
- 変数 i を 0 から x-1 まで変化させながら繰り返し:
- 　 seki ← seki + y。
- ここまで繰り返し。
- seki を出力する。

この方が見やすいでしょう? Java では次の通り。

```
import java.io.*;

class r1ex8b1 {
  public static void main(String args[]) {
    try {
      DataInputStream in = new DataInputStream(System.in);
      System.out.print("x = "); System.out.flush();
      int x = (new Integer(in.readLine())).intValue();
      System.out.print("y = "); System.out.flush();
      int y = (new Integer(in.readLine())).intValue();
      int seki = 0;
      for(int i = 0; i < x; ++i) {
        seki = seki + y;
      }
      System.out.println("product is: " + seki);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

演習 8c

8c は前の問題の「足す」を「掛ける」にしたようなもの。

- 整数 n を入力する。
- seki ← 1。
- 変数 i を 0 から n-1 まで変化させながら繰り返し:
- 　　seki ← seki * 2。
- ここまで繰り返し。
- seki を出力する。

では Java にすると。

```
import java.io.*;

class r1ex8c {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("n = "); System.out.flush();
      int n = (new Integer(in.readLine())).intValue();
      long seki = 1;
      for(int i = 0; i < n; ++i) {
        seki = seki * 2;
      }
      System.out.println("result: " + seki);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

なお、seki を long にしたのは、その方が大きいべきまで計算できるから。int だとどうなるか試してみるとよい。

6　演習 9 と 10

疑似コードは前回の資料に載っていたので略。説明は口頭で。

```
import java.io.*;

class r1ex9 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("x = "); System.out.flush();
      int x = (new Integer(in.readLine())).intValue();
      System.out.print("y = "); System.out.flush();
      int y = (new Integer(in.readLine())).intValue();
```

```
      while(x != y) {
        if(x > y) {
          x = x - y;
        } else {
          y = y - x;
        }
      }
      System.out.println("GCD : " + x);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

なお、演習 10 の方は資料の一部で「$c^2 < x$」になっているというミスプリがありました。すいません。

```
import java.io.*;

class r1ex10 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.print("x = "); System.out.flush();
      double x = (new Double(in.readLine())).doubleValue();
      double a = 0.0;
      double b = x;
      while((b - a) > 0.0000001) {
        double c = 0.5 * (a + b);
        if(c*c > x) {
          b = c;
        } else {
          a = c;
        }
      }
      System.out.println("square root of " + x +" : " + a);
    } catch(Exception e) { System.err.println("!!" + e); }
  }
}
```

7 値とオブジェクト

前回説明し損なった、値とオブジェクトの区別という話からはじめよう。Java が扱うデータには大きく分けると「値」と「オブジェクト」の 2 種類があり、その区別は次のようになっている。

- 値 — 数値 (四則演算の対象になるもの) と、文字。四則演算程度の機能だけを持つ。単純なデータ。
- オブジェクト — さまざまな「もの」を表すデータ。込み入った構造や機能を持つことができる。クラスによって定義され (てい) る。言い替えればクラスとはオブジェクトの種類である。

値としては `int`(整数)、`long`(倍精度整数)`char`(文字)、`float`(実数)、`double`(倍精度 — 精度の高い — 実数)、などがある。一方、文字列などは複雑な構造を持つ (中に文字がたくさん詰まっている) のでオブジェク

トで表す。ところが困ったことに、整数や文字などを表すオブジェクトも別途ある。これは「値」の方はデータの変換などの込み入った機能が入れられないため。そのような値とクラスを次に示しておく。クラス名は大文字で始まることに注意。

種別	値	クラス
真偽値	boolean	Boolean
整数	int	Integer
倍精度整数	long	Long
文字	char	Character
実数	float	Float
倍精度実数	double	Double

基本的に、「値」に対してできることは各種の演算と自動的な文字列への変換だけで、それ以上の機能はすべて「クラス」の方の機能として用意されている。具体的な演算としては次のものがある。

```
四則演算      +   -   *   /   %  ←剰余
ビット演算    &   |   ^   ~
シフト演算    <<  >>
論理演算      &&  ||  !
比較演算      ==  !=  <   >   <=   >=
代入演算      =   +=  -=  *=  …
増減演算      ++  --
```

なお、ビット演算とシフト演算は整数をビット列とみなして演算するもの (実際にはたとえば「00111000」は 64+32+16=122 という数を表す)。

```
 00111000 & 00011100 → 00011000
 00111000 | 00011100 → 00111100
 00111000 ^ 00011100 → 00100100
~00111000            → 11000111
 00111000 >> 2       → 00001110
 00111000 << 1       → 01110000
```

あと、論理演算は「〜かつ〜」「〜または〜」「〜でない」を表す。

```
a > b && a > c   → 「a が b より大きく、かつ a が c より大きい」
x != 0 || !(a > c) → 「x が 0 でないか、または a が c より大きくない」
```

8 JDK のバージョンの違い

さて、ここで Java のバージョンの違いを説明しておこう。主要な Java のバージョンは次のものがある (GSSM ホームページからはどの版の API ドキュメントも見られるようになっている)。

- JDK 1.0.2 — Java が本格的に普及し始めたころの安定バージョンで、1 年前くらいまでは結構使われていた。各種ブラウザもずっとこのバージョンの Java を内蔵していた。「Java 言語入門」で取り上げている。
- JDK 1.1.x — 2 年半前に出たが、1 年前から各種ブラウザでもこの版が搭載されるようになった。
- JDK 1.2.x 以降 — 「Java 2」とも呼ばれる現在の最新版。「続・Java 言語入門」はこちらに準拠している。

我々のところでは、Solaris マシン (smm、smp、smo 等「3 文字の」ホスト名のマシン) では JDK 1.2.x、FreeBSD マシンでは JDK 1.1.x を動かしている。この講義の範囲では JDK 1.1.x と JDK 1.2.x の違いが問題にならない部分だけを取り扱う。

演習 1 API ドキュメント中の java.io.PrintStream の箇所を眺めてみよ。知っている/使ったことのあるメソッドの説明を読め。

演習 2 API ドキュメント中の java.lang.String の箇所を眺めてみよ。文字列に対してだいたいどのような操作ができるのか自分なりに整理してみよ。

9 String オブジェクト

さて、お話ばかりではつまらないので、とりあえずよくお目に掛かって色々役に立つクラスである String クラスを題材にちょっと遊んでみよう。String は文字列を表すクラスである (これに対し、文字は char 型の値。「"..."」は文字列。「'...'」は文字)。

まずは「左右ひっくり返し」の例題を見てみよう。疑似コードを示す。

- 無限に繰り返し:
- 文字列 str を入力。
- str が空文字列なら、ループを抜け出す。
- res ← 空文字列。
- i を str の長さ-1 から 0 まで変えながら繰り返し:
- res の末尾に str の i 番目の文字を連結。
- ここまで繰り返し。
- res を出力。
- ここまで繰り返し。

先にも述べたように、計算機では数は 0 から数えることが多いので、文字列も長さ L の文字列であれば、その中には 0 番目、1 番目、…、$L-1$ 番目の文字が含まれると考える。Java のコードは次の通り。

```
import java.io.*;

class R2Sample1 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        String res = "";
        for(int i = str.length()-1; i >= 0; --i) {
          res += str.charAt(i);
        }
        System.out.println(res);
      }
    } catch(Exception e) { System.err.println("!!"+e); }
```

```
    }
  }
```

では説明を。

- 1 回入力したらおしまいではつまらないので、このプログラムは空行 ([RET] のみ) を入力するまで繰り返し処理を行う。
- 「while(true) ... 」では、条件が「真」という定数なので無限に繰り返すことになる。
- 文字列はオブジェクトなので「==」では比較できない。メソッド `equals()` を使うこと。ここでは空文字列と比較している。
- `break` というのは新しい文で「ループから抜け出す」という動作を行う。無限ループなのでこれがないと終わらない。なお、if 文の枝として文が 1 つしかない場合は「{ ... } 」で囲まないでもよい。
- 文字列の長さはメソッド `length()` で調べられる。ここでは for 文を「$L-1$ から 0 まで順に繰り返し」で使うので i を 1 つずつ減らして行く。
- くっつけるのは「+」だが、ここでは「くっつけ込む」ので「+=」を使っている。

動かしてみよう。

```
% javac R2Sample1.java
% java R2Sample1
String> baka
akab
String> aho
oha
String>    ← [RET] だけを打った
%
```

なかなか面白いでしょう?

演習 3 上の例題をそのまま打ち込んで動かせ。

演習 4 次の Java プログラムを作れ。String クラスのメソッドを活用すること。

a. 入力文字列をすべて大文字にして表示する。(ヒント: toUpperCase() を使う)
b. 入力文字列に現われる「a」をすべて「*」にする。(ヒント: replace() を使う)
c. 入力文字列に現われる「a、e、i、o、u」をすべて「*」にする。(ヒント: replace() を 5 回使う)
d. 次のような三角形を表示する。(ヒント: substring() を使う)

```
String> abcde
abcd
abc
ab
a
```

e. 次のような三角形を表示する。

```
String> abcde
abcd
 bcd
  cd
   d
```

f. 次のような巡回表示を行う。

```
String> abcde
bcda
cdab
dabc
abcd
```

10 Javaの文法

プログラムの正確な書き方は、その言語の「文法」によって定められている。クラスについてはまだ説明していないので、ここではメソッド以下の部分の文法を簡単に示す。なお「[...]」は「あってもなくてもよい」、「|」は「または」を表す。

```
メソッド ::= [public] [static] 型指定 メソッド名(引数部) { 文… }
型指定   ::= 型名 | クラス名 | 型名[] | クラス名[]
文       ::= 宣言文 | 式文 | if文 | while文 | for文 | { 文… }
           | break; | continue;
宣言文   ::= 型指定 変数名 [= 式];
式文     ::= 式;
if文     ::= if(式) 文 [else 文]
while文  ::= while(式) 文
for文    ::= for(式; 式; 式) 文
           | for(型指定 変数名 = 式; 式; 式) 文
式       ::= 式 演算子 式 | 演算子 式 | 式[式] | 変数名 | リテラル
           | クラス名.メソッド名(式,..) | 式.メソッド名(式,..)
           | クラス名.変数名 | 式.変数名 | ( 式 )
リテラル ::= 整定数[l] | 実定数[f] | "..." | '...'
```

整数のリテラルは「l」がついたのはlong、そうでないのはint。実数のリテラルは「f」がついたのはfloat、そうでないのはdouble。なお、演算子については先に挙げた。「a = b;」の「=」はあくまでも代入演算子であることに注意。

11 配列

次は配列について説明しよう。先に出て来た文字列は「文字の並び」を表す特別なクラスだったが、「整数の並んだもの」や「実数の並んだもの」等、一般に「同じ型が並んだもの」を表すプログラミング言語の機能を「配列」と呼ぶ。

Javaでは配列の型は元の型の後ろに「[]」をつけて表す。たとえば「int[]」は整数の並んだ配列、「String[]」は文字列の並んだ配列ということになる。

次に、配列を使うにはその配列型の変数を宣言した上で、大きさを指定して配列オブジェクトを用意しなければならない。具体的には次のような感じになる。

```
int[] a = new int[100];  ←要素数100の配列を用意
int a[] = new int[100];  ←実はこのように「[]」を後に置いてもよい
```

「new」を使うことから分かるように、配列も一種のオブジェクトである(ただし書き方がやや特別な形になっている)。

いちど用意してしまえば、配列の個々の要素は 1 つの変数と同様に扱える。ここで「どの要素か」を指定するのに `[...]` の中に式を書いて指定する (これを添字と呼ぶ)。たとえば上の例だと `a[0]`〜`a[99]` という要素があることになる (例によって 0 番目から数えるのに注意)。

では、配列に整数をいくつか (0 が現われるまで、ただし最大 100 個まで) 読み込み、それを逆順に表示するという例題。

- a ← 大きさ 100 の整数配列。
- count ← 0。
- `count+1 < a` の要素数である間繰り返し:
- 　v に整数を入力する。
- 　もし `v == 0` ならば、繰り返しを終わる。
- 　a[count] ← v。
- 　count を 1 ふやす。
- ここまで繰り返し。
- 変数 i を count-1 から 0 まで変えながら繰り返し:
- 　a[i] を出力する。
- ここまで繰り返し。

なお、最初のループの条件は「次に 1 個増やしたらもう配列に入り切らなくなるならループをやめる」ということ。このように、配列は最初に大きさを指定して作ってしまうので、指定した以上の要素を使わないように注意しながらプログラムを作る必要がある。Java コードは次の通り。

```
import java.io.*;

public class R2Sample2 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      int[] a = new int[100];
      int count = 0;
      while(count+1 < a.length) {
        System.out.print("" + count + "> "); System.out.flush();
        int v = (new Integer(in.readLine())).intValue();
        if(v == 0) break;
        a[count] = v;
        ++count;
      }
      for(int i = count-1; i >= 0; --i) {
        System.out.println(a[i]);
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

なお、「a.length」というのは、a が配列オブジェクトの場合、その変数 length を調べれば配列の要素数が分かるのでこれを参照しているもの。

演習 5 上記のプログラムをそのまま打ち込んで動かせ。

演習 6 下記のような Java プログラムを作れ。

a. 配列に整数を読み込み、その最大値と最少値を出力する。
b. 配列に整数を読み込み、その合計と平均を出力する。
c. 配列に整数を読み込み、その平均より大きい要素を出力する。

演習 7 下記の疑似コードは、配列に整数を読み込み、それを小さい順に並べて出力するものである。この疑似コードを Java に直して動かせ。またなぜこれで小さい順に並べて出力できるのか説明せよ。

- a ← 大きさ 100 の整数配列。
- count ← 0。
- `count+1 < a` の要素数である間繰り返し:
- 　v に整数を入力する。
- 　もし `v == 0` ならば、繰り返しを終わる。
- 　a[count] ← v。
- 　count を 1 ふやす。
- ここまで繰り返し。
- 変数 i を 0 から count-1 まで変えながら繰り返し:
- 　変数 k を 1 から count-1 まで変えながら繰り返し:
- 　　もし `a[k-1] > a[k]` ならば、
- 　　x ← a[k-1]。
- 　　a[k-1] ← a[k]。
- 　　a[k] ← x。
- 　枝分かれ終わり。
- 　ここまで繰り返し。
- ここまで繰り返し。
- 変数 i を 0 から count-1 まで変えながら繰り返し:
- 　a[k] を出力。
- ここまで繰り返し。

12　ファイルの読み書き

さて、せっかく文字列や配列が分かったので、もう 1 つの「内容が並んでいるもの」としてファイルの読み書きをやろう。ファイルを読み書きするには、次のクラスを使う。

- `FileInputStream` — ファイルからバイト単位で読み込む
- `FileOutputStream` — ファイルへバイト単位で書き出す

これらのクラスの API を見てみると、new でファイル名を文字列として指定すればよいことが分かる。ここでは文字列を固定する代りに、main への引数、つまりプログラム起動時にコマンド引数として渡したものを利用することにしよう。1 行ずつ読み書きするには、これらをそれぞれ InputStreamReader / OutputStreamWriter (文字単位での読み書き)、BuffferedReader / PrintWriter (1 行単位での読み書き) に「包んで」使用する。

```
import java.io.*;

public class R2Sample3 {
```

```
  public static void main(String args[]) {
    try {
      BufferedReader in = new BufferedReader(
        new InputStreamReader(new FileInputStream(args[0])));
      PrintWriter out = new PrintWriter(
        new OutputStreamWriter(new FileOutputStream(args[1])));
      String str;
      while((str = in.readLine()) != null) {
        out.println(str);                        /****/
      }
      in.close(); out.close();
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

これを動かすときは次のようにする。

```
% javac R2Sample3.java
% java R2Sample3 test.txt test.out  ←入力と出力を指定
```

13 漢字を正しく扱う

ところで、これと先の「左右ひっくり返し」を組み合わせるため、上の例だいの「/****/」の行を次のように差し替えたとする。

```
String res = "";
for(int i = str.length()-1; i >= 0; --i) {
  res += str.charAt(i);
}
out.println(res);
```

これを日本語のファイルに対して動かすと、日本語が化けてしまってうまく行かない。なぜだろう?

実は、上のプログラムでは Reader や Writer は「1 バイトが 1 文字だと思って」(つまり「英語を扱ってると思って」) 動作してしまっているので、日本語の中のバイトや制御文字列も全部左右反転されてしまってめちゃくちゃになる。Java では (C や C++と違って)1 文字が 16 ビットあり日本語をきちんと「文字」単位で扱えるのだが、ただし正しく日本語を扱うためには、InputStreamReader や OutputStreamWriter を生成するときに追加の引数で「符号化のしかた」を指定する必要がある。現在のところ、日本語がらみでは次の 5 つが使える。

- JIS — JIS コード
- SJIS — Shift-JIS コード
- EUCJIS — 日本語 EUC コード
- JISAutoDetect — 上の 3 つのどれであるかを自動判別 (入力のみ)
- UTF8 — UNICODE 用のファイル形式である UTF8

ここでは入力は自動判別、出力は JIS にしてみよう。

```
import java.io.*;

public class R2Sample4 {
  public static void main(String args[]) {
```

```
    try {
      BufferedReader in = new BufferedReader(
        new InputStreamReader(new FileInputStream(args[0]), "JISAutoDetect"));
      PrintWriter out = new PrintWriter(
        new OutputStreamWriter(new FileOutputStream(args[1]), "JIS"));
      String str;
      while((str = in.readLine()) != null) {
        String res = "";
        for(int i = str.length()-1; i >= 0; --i) {
          res += str.charAt(i);
        }
        out.println(res);
      }
      in.close(); out.close();
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

これで漢字のファイルもきちんと扱える。

14 おまけ: ファイルでなくネットワークから読むには…

では最後に、ファイルではなく任意の URL から読むのをやってみよう (ただし外のインターネットから読むためには utogw で動かさないといけない)。それには、URL の文字列をもとに URL オブジェクトを作り、そのメソッド openStream() を使うとその URL の内容を読むストリームが返される。あとはこれまで学んだことそのままでよい。

```
import java.io.*;
import java.net.*;

public class R2Sample4c {
  public static void main(String args[]) {
    try {
      BufferedReader in = new BufferedReader(
        new InputStreamReader((new URL(args[0])).openStream(),
          "JISAutoDetect"));
      PrintWriter out = new PrintWriter(
        new OutputStreamWriter(System.out, "JIS"));
      String str;
      while((str = in.readLine()) != null) {
        String res = "";
        for(int i = str.length()-1; i >= 0; --i) {
          res += str.charAt(i);
        }
        out.println(res);
      }
      in.close(); out.close();
    } catch(Exception e) { System.err.println("!!"+e); }
  }
```

```
}
```

なお、このプログラムでは出力は標準出力に出るようにしてある。

演習 8 ここまでの例題を参考に、次のようなプログラムを作成せよ。

a. ファイルを「上下さかさま」に変換する。ファイル名を 2 つ指定。
b. 上と同様だが、ただし出力ファイルを指定しない場合は標準出力に出す。
c. 上と同様だが、ただしオプションで指定するとファイルの代りに指定した URL から読み込む。
d. 上に加えて、「左右反転」と「上下反転」を自由に組み合わせて変換する。オプションの指定方法は自分で自由に設計してよい。