計算機プログラミングI 2005　久野クラス　＃3

久野　靖*

2005.10.21

はじめに

いただいたレポートを拝見していると、皆様順調に課題をこなされているようで嬉しく思います (大変だという人はいますが、それはまだプログラミングの感覚が不慣れだからで、段々スムーズになると思います)。ところで、**無限ループ (止まらないループ) を作ってしまったときの止め方を知らないと困りますね?! それは「Ctrl+C」(Ctrl キーを押しながら「C」のキーを打つ) で止められます。** 覚えておきましょう。あと、コンパイラのエラーメッセージの見かたが難しいという意見があるようですので、参考ページへのリンクを本クラスのサイトに追加しました (でも非常に多数の種類のメッセージがあるのですぐ分かるというわけには行きません。やっぱり修行になっちゃうのですよね)。

さて、本日はまず前回の課題の解説と併せて、制御構造などで追加すべき点を説明します。また、本日の新しい内容として次のものを取り上げます。

- 繰り返し中での入力とデータ処理
- 配列とその操作

これらが分かるとだいぶ多様なプログラムが作れるようになると思います。ここまでで値の計算は一応終わりにして、次回はオブジェクト関連の話題に進む予定です。

1　演習問題の解説 — 枝分かれについて

1.1　演習 2a

演習 2a は 2 つの枝分かれですから例題とほとんど同じ。まず疑似コードを見よう。

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

Java では次の通り。

```
import java.io.*;

public class r2ex2a {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = ");
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = ");
    double b = (new Double(in.readLine())).doubleValue();
    double max;
```

*筑波大学大学院経営システム科学専攻

```
    if(a > b) {
      max = a;
    } else {
      max = b;
    }
    System.out.println("larger is: " + max);
  }
}
```

ところで、要点部分だけ示すとして、次のようなコードを書いてコンパイラに怒られていた人がいたが、どう思いますか?

```
double max;
if(a > b) { max = a; }
if(a < b) { max = b; }
```

これをコンパイルすると「変数 max は初期化されていない可能性があります。」といわれる。つまり max は最初に値を入れずに宣言だけして、a と b の大小に応じてどちらかが入るから…もし等しかったら? その時は max は「何も入らないまま」になりますね? それはまずいでしょう、というのがこのメッセージの意味。しかし実は。

```
double max;
if(a > b) { max = a; }
if(a <= b) { max = b; }
```

これでも同じエラーが出る。実は Java コンパイラは「この 2 つの条件の少なくとも片方は必ず成り立つ」と推論してくれるほど賢くない。このような場合にエラーメッセージを消すには max に初期値を入れておけばいい…

```
double max = 0.0;
if(a > b) { max = a; }
if(a <= b) { max = b; }
```

けど、気持ち悪いでしょう? やっぱり「どちらか片方を実行」する場合はこういうのではなく「if-else」を使いましょう、ということ。

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.*;

public class r2ex2a1 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = ");
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = ");
    double b = (new Double(in.readLine())).doubleValue();
    double max = a;
    if(b > max) {
      max = b;
    }
    System.out.println("larger is: " + max);
  }
}
```

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。

1.2 演習 2b — 枝分かれの入れ子

しかし演習 2b 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、
- 　　max ← a。
- 　そうでなければ、
- 　　max ← c。
- 　枝分かれおわり。
- そうでなければ、
- 　もし b > c であれば、
- 　　max ← b。
- 　そうでなければ、
- 　　max ← c。
- 　枝分かれおわり。
- 枝分かれおわり。
- max を出力する。

うーむ大変だ。これを Java にしておく。

```
import java.io.*;

public class r2ex2b {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = ");
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = ");
    double b = (new Double(in.readLine())).doubleValue();
    System.out.print("c = ");
    double c = (new Double(in.readLine())).doubleValue();
    double max;
    if(a > b) {
      if(a > c) {
        max = a;
      } else {
        max = c;
      }
    } else {
      if(b > c) {
        max = b;
      } else {
        max = c;
      }
    }
    System.out.println("largest : " + max);
  }
}
```

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。
- 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.*;

public class r2ex2b1 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = ");
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = ");
    double b = (new Double(in.readLine())).doubleValue();
    System.out.print("c = ");
    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);
  }
}
```

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。また、この方法では入力の数 N がいくつになっても簡単に対処できるという利点がありますね。なお、if 文や「もし」の疑似コードが 1 行に書かれているが、この場合はこちらのの方が見やすいと思ったので。

1.3 多方向の枝分かれ

演習 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.*;

public class r2ex2c {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    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.");
    }
  }
}
```

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。

1.4 文字列を変数に入れる

ところで、上の例解では枝分かれの中で出力することでメッセージを切替えていた。そうじゃなくて、これまでのように最後の 1 個所で出力したければどうしたらいいだろう? それは出力したい文字列を変数に入れておけばよい。Java のコードだけ示す。お分かりになります?

```
import java.io.*;

public class r2ex2c1 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    double x = (new Double(in.readLine())).doubleValue();
    String mesg;
    if(x > 0.0) {
      mesg = "positive.";
    } else if(x < 0.0) {
      mesg = "negative.";
    } else {
      mesg = "zero.";
    }
    System.out.println(x + " is " + mesg);
  }
}
```

1.5 複合条件

ところで、演習をやっていて「A かつ B、のような複合条件はどう書けばよいのか」という質問があった。それには次のようにする。

α かつ β → α && β
α または β → α || β
α でない → ! α

丸かっこでくくることで、これらを組み合わせたいくらでも複雑な条件を書くことができる。複合条件と多方向の枝分かれがあると、「3 つの最大」問題はたとえば次のようにも書ける。これならすっきりしていますか?

```
import java.io.*;

public class r2ex2b2 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = ");
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = ");
    double b = (new Double(in.readLine())).doubleValue();
    System.out.print("c = ");
    double c = (new Double(in.readLine())).doubleValue();
    double max;
    if(a > b && a > c) {
```

```
      max = a;
    } else if(b > c) {
      max = b;
    } else {
      max = c;
    }
    System.out.println("largest : " + max);
  }
}
```

ただし、この方法でも N が 4、5 と増えてくると条件の中の比較演算が増えて、一般に N^2 に比例してしまう。もっともだからいけないというわけではなく、N の個数がいくつと決まっていればこのやり方を使ってもいいかも知れない。

2 演習問題の解説 — 繰り返しについて

2.1 演習 3a — 繰り返しの使い方

演習 3a であるが、これは前に説明したように 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:
- $\quad x \leftarrow x - 2$
- ここまで繰り返し。
- もし $x = 0$ ならば、
- $\quad$「偶数」と出力。
- そうでなければ、
- $\quad$「奇数」と出力。
- 枝分かれ終わり。

Java で書くと次の通り。

```
import java.io.*;

public class r2ex3a {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    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.");
    }
  }
}
```

このような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。つまり、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。それでいいのかどうか? 次のどれが適切だと思う? (解説は講義で。)

(1) このままで適切である。

(2) 正でない整数が来たらその旨警告して止まるのが適切である。

(3) 正でない整数でも正しく奇数偶数が判定できるようにしておくのが適切である。

2.2 演習 3b — for 文による繰り返し

演習 3b 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.*;

public class r2ex3b {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    int x = (new Integer(in.readLine())).intValue();
    System.out.print("y = ");
    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);
  }
}
```

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 while 文の「親戚」だが、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。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.*;

public class r2ex3b1 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    int x = (new Integer(in.readLine())).intValue();
    System.out.print("y = ");
    int y = (new Integer(in.readLine())).intValue();
    int seki = 0;
    for(int i = 0; i < x; ++i) { seki += y; }
    System.out.println("product is: " + seki);
  }
}
```

2.3 演習 3c — 整数の限界

演習 3c は 3b と同様で、足す代りに 2 を掛けていく。Java のコードだけ示そう。

```
import java.io.*;

public class r2ex3c {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("n = ");
    int n = (new Integer(in.readLine())).intValue();
    int power = 1;
    for(int i = 0; i < n; ++i) { power = power * 2; }
    System.out.println("2 raised by " + n + " is: " + power);
  }
}
```

ところで、これでやってみると 30 まではうまく行くが 31 でマイナスになってしまいますね? Java では `int` は 32 ビットで表されている。「1」を 2 倍するということは、全体を 1 ビット左へシフトすることと等しい。ということは、31 回シフトすると一番上の桁に「1」が来る。この桁は符号桁なので、マイナスになる。もう 1 回 2 倍すると、「1」が左から出て行ってしまうので 0 になってしまう。まあそれはともかく、整数では誤差はない代りにおよそ -2^{31}〜$+2^{31}$ くらいの範囲しか扱えないことは覚えておいて欲しい。

なお、Java では倍精度整数を表す `long` 型 (クラスの方は `Long`) もある。こちらは 64 ビットなのでおよそ -2^{63}〜$+2^{63}$ の範囲が扱える。よかったら上のプログラムを

```
long power = 1;
```

のように変更して動かして確認しよう。

2.4 演習 4

疑似コードは前回の資料に載っていたので略。

```
import java.io.*;

public class r2ex4 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    int x = (new Integer(in.readLine())).intValue();
    System.out.print("y = ");
    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);
  }
}
```

なぜこれで最大公約数が求まるのだろう? 次のように考えてみるとよい。なお、x と y は正の整数であるものとします。

- $x = y$ であれば、最大公約数は x そのもの。当然ですね。
- $x > y$ であれば、x と y の最大公約数は $x - y$ と y の最大公約数に等しい。[1]
- したがって、$x - y$ を改めて x とおいて、x と y の最大公約数を求めればよい。
- $x < y$ の場合も同様。
- この手順の反復ごとに、x または y のどちらかがより小さくなるが、0 以下にはならない (大きい方から小さい方を引くから)。
- ということは、この反復は有限回で止まる。
- ということは、そのとき $x = y$ が成り立ち、x が一番最初の x と y の最大公約数に等しい。

どうですか、繰り返しを使うときは「必ず止まって、止まった時には求める状況が成り立っている」ように設計する、という意味が分かります?

2.5 演習 5

これも疑似コードは略。

```
import java.io.*;

public class r2ex5 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = ");
    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;
```

[1] 証明: 最大公約数を G とおくと、x も y も G の整数倍なのだから、$x - y$ もまた G の整数倍である。ということは、G は $x - y$ と y の公約数である (最大かどうかはまだ分からない)。ところで、もし最大公約数で「なかった」とすると、最大公約数 $H(> G)$ が別にあるわけで、H は y の約数かつ $x - y$ の約数。ということは、H は $x - y + y = x$ の約数でもある。これは G が x と y の最大公約数であるということに矛盾する。従って G は $x - y$ と y の最大公約数でもある。

```
      }
    }
    System.out.println("square root of " + x +" : " + a);
  }
}
```

このプログラムでなぜ「平方根を求める」ことができるか考えてみよう。ただし x は 1 より大きな数とする。

- x の平方根を R とすると、$0 \leq R < x$ である。
- $a = 0$、$b = x$ とおくのだから、$a \leq R < b$ である。
- $c = \frac{a+b}{2}$ としたとき、$c^2 > x$ であれば $a < R < c$ であるので、c を改めて b と置いたとき、$a \leq R < b$ である。そうでなければ、$c \leq R < b$ であるので、c を改めて a と置いたとき、$a \leq R < b$ である。
- $b - a$ は、反復ごとに a と b の平均 c を a、b のいずれかと置き換えるので、毎回半分ずつに小さくなっていく。ということは、いつかは必ず 0.0000001 以下になって停止する。
- このときも $a \leq R < b$ は成り立っているので、つまり a と R の差は 0.0000001、ということはその誤差以下で R が求まったと言える。

これも「1 より小さな数でも求まるように」したいかどうか。考えてみよう。

2.6 演習 6 — 論理型

素数、やってみましたか? 疑似コードは次の通り。

- N を入力する。
- sosu ← 「真」。
- i を 2 から $N - 1$ まで変化させながら繰り返し:
- 　　もし N が i で割り切れるならば、sosu ←「偽」
- 繰り返しおわり。
- sosu を出力。

この sosu は真偽値 (真/偽のいずれかだけを表す) で、Java では `boolean` 型を使い、また「真」は `true`、「偽」は `false` で表す。もっとも、皆様は習ってないから整数の 1/0 で表したりしたと思うけれど、それはそれで結構。なお、この変数 sosu は最初に「真」を入れておいて、どこかで素数でないと思ったら「偽」にして、最後に結果がどちらか見る。このような使い方の変数のことを「旗」(flag) と呼ぶ。では Java コードを見てみよう。

```
import java.io.*;

public class r2ex6 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("n = ");
    int n = (new Integer(in.readLine())).intValue();
    boolean sosu = true;
    for(int i = 2; i < n; ++i) {
      if((n % i) == 0) { sosu = false; }
    }
    System.out.println("prime number : " + sosu);
  }
}
```

前にも説明したが、「`%`」は剰余演算子 (割った余りを返す) なので「`(n % i) == 0`」で「`n` が `i` で割り切れる」という意味になる。

2.7 演習 7 — プログラムの改良

さて、演習 7 は演習 6 の部分を「ループの中に取り込んで」作ることができる。疑似コードは略して Java の方を見てみよう。

```
import java.io.*;

public class r2ex7 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("m = ");
    int m = (new Integer(in.readLine())).intValue();
    for(int n = 1; n <= m; ++n) {
      boolean sosu = true;
      for(int i = 2; i < n; ++i) {
        if((n % i) == 0) { sosu = false; }
      }
      if(sosu) { System.out.println(n); }
    }
  }
}
```

変数 sosu の使い方が「sosu だったらその数を打ち出す」に変っているのに注意。ところで、これでやると私の手元では 17000 までやるのに 10 秒くらい掛かっている。これを工夫して速くするにはどうしたらいいだろう? たとえば次のようなことが考えられる。

- 2 は別に打ち出すことにして、候補として 3 以上の奇数だけ調べる (候補の数が半分になる!)。
- 割ってみる数も 3 以上の奇数だけ試す (割ってみる数も半分に!)。
- $N-1$ まで試さなくても、$\sqrt{N}$ まで試せば十分 ($\sqrt{N}$ より大きい因数があるなら、必ず $\sqrt{N}$ より小さい因数もあるわけだから)。
- 割り切れると分かったところでループを止めてその先は調べないようにする。

これらの改良を施した版を次に示す。

```
import java.io.*;

public class r2ex71 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("m = ");
    int m = (new Integer(in.readLine())).intValue();
    System.out.println(2);
    for(int n = 3; n <= m; n += 2) {
      boolean sosu = true;
      int limit = (int)(Math.sqrt(n) + 1);
      for(int i = 3; i <= limit; i += 2) {
        if((n % i) == 0) { sosu = false; break; }
      }
      if(sosu) { System.out.println(n); }
    }
  }
}
```

この「`break;`」という文は、この文を含んでいる繰り返し (for ループまたは while ループ) を直ちに終わらせる (ループから抜け出す)、という機能を持っている。そんなの習ってないって? すいません、ただしこの問題の場合はこれを使わなくても「`break;`」の代りに「`i = limt + 1;`」とすれば同様に終わらせることができる。でもまあ、ループを終わりたいのなら「`break;`」を使うのが素直なので覚えておこう。

で、これだと、17000 までやるのに私の手元で約 1 秒だった。つまり、10 倍のスピードアップ! ところで、さらに次のような改良もあり得ますね?

- 3 からの奇数全部で割ってみる代りに、これまでに見つかった素数でだけ割ってみれば十分。

素数は数が大きくなるとかなりまばらにしかないので、これで割ってみる数が減らせる。ただし残念なことに、これまでに学んだやり方では「見つかった素数を覚えておいて利用する」ことができない! 実は、後で学ぶ「配列」を使うとこれができるので、ぜひこの改良にチャレンジしてみて欲しい。

3　再度、値とオブジェクトについて

3.1　値とオブジェクトの違い

前回もちょっと触れたが、重要な点なので再度値とオブジェクトの話をしよう。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。

以下ではまず値の種類について復習し、その後で値の演算についてまとめ、続いて前回あとまわしにしたおまじないの説明をする。

3.2　値の種類と演算

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`
バイト	`byte`	`Byte`
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。

値に対する具体的な演算をすべて挙げると、次のものがある。

四則演算	`+`	`-`	`*`	`/`	`%`	←剰余
ビット演算	`&`	`|`	`^`	`~`		← and、or、xor、0/1 反転
シフト演算	`<<`	`>>`	`>>>`			←左シフト、右シフト、算術シフト
論理演算	`&&`	`||`	`!`			
比較演算	`==`	`!=`	`<`	`>`	`<=`	`>=`
代入演算	`=`	`+=`	`-=`	`*=`	…	
増減演算	`++`	`--`				

ビット演算、シフト演算はあまり使わないので当面は忘れてよい。演算子には「強さの順」がある。例えば乗算は加算より強いので、「1 + 2 * 3」は「1 + (2 * 3)」と同じになる。演算子を強い順に並べると次のようになる (丸かっこでくくったものは同等)。

`(++ -- ! ~)` → `(* / %)` → `(+ -)` → シフト →`(< <= > >=)` → `(== !=)` → `&` → `^` → `|` → `&&` → `||` → 代入演算

4　繰り返しを使った複数データの処理

繰り返しは前回一応やったが、もう少し練習した方がよいので、今度は繰り返しを使ってデータを次々に読むというパターンのプログラムを見てみることにする。今回の例題は、任意個数の数値を入力して、その合計を表示するといういかにも例題らしい:-) プログラムである。

- sum ← 0.0
- 無限に繰り返し:
- 　　値 x を入力。
- 　　もし x = 0 ならば繰り返しを抜け出す。
- 　　sum ← sum + x。
- ここまで繰り返し。
- sum を出力。

無限に繰り返す? それでは終わらないじゃないか、と思うかも知れないが、その中で「抜け出す」を実行すればちゃんとループを終わることができるから別に構わない。こういう無限ループはプログラムによってはなかなか便利である。では Java にしたものを示す。

```
import java.io.*;

public class R3Sample1 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    double sum = 0.0;
    while(true) {
      System.out.print("x> ");
      double x = (new Double(in.readLine())).doubleValue();
      if(x == 0.0) { break; }
      sum = sum + x;
    }
    System.out.println("sum = " + sum);
  }
}
```

こうすれば数がいくつあっても合計できるでしょう? そして、合計を求めるときに「0」を加えたいとは普通思わないので、「0」を入れたらおしまいの印、というのはなかなかうまくできている。別案として、データの個数を最初に入力してもらい、あとはその数だけデータを読み込む、というのもあるが、こちらは人間にとってはあまり使いやすくない (なぜか分かりますか?)。

演習 1 上のプログラムをそのまま打ち込んで動かせ。

演習 2 動いたら次のようなバリエーションのプログラムを作ってみよ。上の例題を直接直すかコピーして直して作るとよい。

a. 任意個数のデータを入力し (0 で終わり)、その平均値を表示。
b. 任意個数のデータを入力し (0 で終わり)、その最大値を表示。
c. 任意個数のデータを入力し (0 で終わり)、その最大値および最大が何番目に現われたかを表示。ただし複数回最大値が現われた場合は最初に現われたものが何番目かを表示すること。
d. 任意個数のデータを入力し (0 で終わり)、その最大値および最大が何番目に現われたかを表示。ただし複数回最大値が現われた場合は最後に現われたものが何番目かを表示すること。

5 配列

上のやり方で、とりあえず「次々に足して行く」とか「とりあえず今まで分かっているうちで最大を覚えておく」という形で複数のデータを扱うことはできる。しかし、たとえば一連のデータに対して繰り返し計算をするとなると、それを全部一度に変数に保持して置かなければならないので、これまでの方法ではうまく行かない。本日 2 番目の話題として、そのような場合に利用する機能である「配列」について説明しよう。

「配列」とは、「同じ種類 (型) のデータを沢山ならべたもの」というプログラミング言語用語である。並べる型は何でもよい。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];  ←変数定義ではこの書き方も可 (C 言語から伝承)
```

「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、それら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 < 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 R3Sample2 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    int[] a = new int[100];
    int count = 0;
    while(count < a.length) {
      System.out.print(count + "> ");
      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]);
    }
  }
}
```

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 3 上記のプログラムをそのまま打ち込んで動かせ。動いたらこれを参考に下記のような Java プログラムを作れ。

a. 配列に整数を読み込み、その最大値と最小値を出力する。

b. 配列に整数を読み込み、最大値が何番目か (複数あればそれら全部の番号) を出力する。なお先頭を 0 番目とする。

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 4 下記の疑似コードは、配列に整数を読み込み、それら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 < 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[i] を出力。
- ここまで繰り返し。

演習 5 下記の疑似コードは、配列に整数を読み込み、それら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 < 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。
- 　　i ← count。
- 　　i > 0 かつ a[i-1] > v である間繰り返し:
- 　　a[i] ← a[i-1]。
- 　　i ← i - 1。
- 　　ここまで繰り返し。
- 　　a[i] ← 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:
- 　　a[i] を出力。
- ここまで繰り返し。

演習 6 配列を使って「N 未満の素数を全部打ち出す」プログラムを「これまでに分かっている素数でだけ割って見る」ように改良し、どれくらい速くなったか調べよ。

演習 7 次のような構想 (これはまだ疑似コードではない!) に従って「N 未満の素数を全部打ち出す」プログラムを作り、速さを評価せよ。

- 論理値 (boolean) 型が並んだ要素数 N の配列を作り、全部「真」に初期化。
- 2、4、6、…と、2 の倍数番目の部分を「偽」に変更。
- 3、6、9、…と、3 の倍数番目の部分を「偽」に変更。
- 同様に、素数の倍数番目を「偽」に変更していく。
- 最後に、「真」で残っているところを順に調べ何番目かを出力。

演習 8 演習 4 や 5 の「並べ換え」はデータの量が少ないので時間計測には不向きだった。そこで、入力する代りに

```
int v = (int)(1000000*Math.random()); // 0〜999999 の乱数生成
```

という文を使ってデータを生成し、少し多め (1 万個とか) のデータで時間計測を行い、演習 4 と演習 5 のやり方の速度を比較してみよ。また、もっと速くする工夫があれば試してみよ。

演習 9 その他、配列を活用する、自分の腕にあった (不当に易しくない) プログラムを考案し、作成せよ。

A　本日の課題 3A

「演習 2」で動かしたプログラムを含む小レポートを今日中に久野までメールで送ってください。

1. Subject: は「Report 3A」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. 「演習 2」で動かしたプログラムどれか 1 つのソース。
4. 以下のアンケートの回答。

   Q1. 繰り返しを使ったプログラムに慣れましたか。

   Q2. 配列について学びましたが、使えそうですか。

   Q3. 本日の全体的な感想と今後の要望をお書きください。

B　次回までの課題 3B

次回までの課題は「演習 2a」〜「演習 9」の (小) 課題からプログラムを 2 つ以上作り、報告すること。ただし 5〜9 から最低 1 つは選んで欲しい。レポートは授業開始時刻の **10 分前** までに、上記と同様に久野までメールで送付してください。

1. Subject: は「Report 3B」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. 選んだプログラム 1 つのソース。
4. その簡単な説明。
5. もう 1 つのプログラムのソース。
6. その簡単な説明。あれば考察等も。
7. 下記のアンケートの回答。

   Q1. 配列が使いこなせるようになりましたか。

   Q2. 今回もまた、疑似コードを書くのと、Java に直すのと、打ち込んで動かすのとで掛かった手間の比率を教えてください。

   Q3. 課題に対する感想と今後の要望をお書きください。