続・東大理系1年生むけCSスパルタ教育
～JavaからRubyへ～(プ会2008.1)

久野　靖*

2008.1.17

1　前回(2007年5月)のあらまし

□ 1年後期理系選択科目「計算機プログラミング」→2006年度から「情報科学」に模様替えしてクラス指定科目に

- 理系1年生全員に科目枠を割り当て(選択しないことは可能だが、選択しない場合そのコマは空きコマになる)
- 「情報科学(CS)」を教えることが目標となる
- 教える内容範囲は「標準スライド」により各担当者に周知(?)。内容はかなり広範囲かつ深い
- 使用するプログラミング言語はRubyに統一。ただしプログラミングの習得は目的でない
- 2006年度については「試行」として上記に準拠しなくてもよい。2007年度から本格施行

□ 久野の選択(2006年度)

- 自分が作った資料でないと教えられないという人なので、ずっと自作資料でJava言語を教えて来た
- プログラミングは書かなければ身につかないという信条なので、毎週2回(当日+次回まで)小課題でプログラムを提出してもらってきた
- 上記方針は維持し、ただし翌年度に備えて内容範囲を「標準スライド」をカバーするものに書き換える。題材を扱う順序は自分で設計
- 内容と言語を一緒に書き換える気力はないので→2006年度はJava言語を使用

□ 久野クラスの実施結果…

- 標準スライドが広く深いため、かなり詰め込みの内容
- にもかかわらず、自分の趣味なところ(言語処理系)のところはさらに内容を上積みしてしまった(つい…)
- 結果、当該箇所は超スパルタになったので学生さんには申し訳ないことをしたと考えている
- 当日課題提出人数の変遷:　1:52 → 2:42 → 3:38 → 4:34 → 5～13:約25で安定。やめる人はさっさとやめて後はやめない
- 学生さんの反応は「難しい」というのは当然あるが、難しいから駄目、というわけではない(さすが東大?)
- マークシートアンケート(東大実施、出席しない人がいるので回答数は18)→総合評価:　大変満足10、ほぼ満足2、普通4、やや不満2。難易度:　易しすぎ1、適切6、やや難しい9、難し過ぎ2。(最後まで出た人という点を割り引く必要あり。)

□ 2006年度に対する久野の考察…

- 標準スライドがハードなのに輪を掛けてスパルタにしてしまったが、学生は好意的だった→ありがたいことです
- ドリトル言語での中学/高校での経験→「難しい」ことはマイナスではなく、難しいことを達成するのは楽しいという生徒が多い(教員が下手を打って達成させ損ねない限りは)
- プログラムを書かせつつCSを教えるのは大いに可能
- 題材としてのCSは奥が深く面白い(たぶん学生にとっても)、という点は改めて見直した

2　2007年度の実施に当たって

□ 東大から示された前年度よりの変更点

- 履修率が良くないので難易度を下げる→難易度と学生の人気とは必ずしも関係ないのでは?　難しくてもやりがいがあり自分のためになると思えば来てくれるという印象
- 標準スライドの内容を削って少なくする→自分もやりすぎたと思ったので少なくするつもりだったが、あんまり削ると面白くなくなるのではと懸念
- 内容は標準スライドの「必須」部分をカバーしていれば順番や扱いは各教員に任される→それはそのつもりで準備してきたので

*筑波大学大学院経営システム科学専攻

- 評価は共通試験+各担当者独自評価→毎回のレポートは維持し、その点数と共通試験とを 50:50 でつけることにする

□ 最大の変更点: 「Java から Ruby へ」

- 久野の信念: 実行時エラー/論理エラー取りに比べてコンパイル時エラー取りははるかに容易→強い型の言語でコンパイル時エラーを取って実行、という経験はぜひとも積んで欲しい☆
- とはいえ、Ruby の使用が指定されたので、Ruby だとどうなるかやってみることは興味津津。☆のため最後に数回 Java をやってもらう
- 資料は前年度に扱う題材をコンバートしたので、基本的には各例題を Ruby に書き直して細かい説明を Ruby 用に直す (結果、Ruby というべきところに Java と書いてあること多々に…)
- いくつかの (けっこう多くの) 箇所で、Java と Ruby でストーリを変更する必要 (後述)

3 2007 年度授業の概要

□ 全体方針

- 標準スライドの内容をカバーするが (ちょっとできてないところもあるかも)、取り上げる順番は変更する
- 取り上げる各内容について、(原則として)Ruby プログラムを用いて確認ないし追体験を行えるようにする (演習課題を用意するということ。ただし演習を全部やるのは無理なので学生の選択)
- そのため、Ruby プログラミングが身につくように、プログラミング的に易→難の順に題材を配置する

□ 各回の授業の運用

- 毎回 (授業時間は 90 分)、おおむね 2 つ程度のトピックを紹介する。資料は授業の数日前に PDF で公開
- 1 つ目について講義/説明した後、それに対応する例題プログラムを「そのまま」打ち込んで動作確認してもらう。
- その後、それを多少手直ししてやるとできる演習が複数用意されているので引き続きやってもらう。(この演習内容を「当日レポート」として提出。) ここまでで大体 50 分〜1 時間 (説明 20+演習 30)
- 頃合いを見計らって演習を中断し、2 つ目のトピックを講義/説明。これに対応する演習から 2 つ選んで次回までのレポートとする。
- 次回レポートの解説を次回講義の最初に行う。この部分では、まず問題を考えてもらう→それに対応する CS 概念を講義、という順が可能

□ レポートとその評価

- 当日課題 (A レポート) は当日 24 時、次回課題 (B レポート) は次回講義 10 分前が締め切り (でも期限は臨機応変に延長)。授業に来る人は 10 人ちょっと、残りは資料で勉強してレポートを出すだけ (完全 e-learning?)
- 課題は資料に「演習」として掲載、8 個くらい (うち半分くらいは数個の小課題から成る) →難易とりまぜ、「自分の腕前に合った課題を選択」するよう促す→とにかく何かはプログラムを書くことを優先
- レポートはメールで提出させ、Web で全員ぶんを公開。公開時にすべてに短いコメントを付けた→互いがやっていることを見られるように

  http://lecture.ecc.u-tokyo.ac.jp/~kuno/is07/report/

- 評価は△ (遅刻や不十分)、○ (普通)、◎ (特に買う点がある) の 3 段階。◎は自由課題的演習の時につける

□ 以下、各回の内容と思ったこと。

- 現在、#1〜#11 まで完了し、標準スライドの「さまざまなプログラミング言語」以外をすべて完了 (Ruby でやる内容はすべて完了)。残り 3 回は「さまざまなプログラミング言語」とおまけ 2 回を Java でやる予定
- 出席人数は 1:30 → 2〜11:約 20 で安定

3.1 #1: アルゴリズムとプログラム/数値の表現

□ 計算の手順がアルゴリズムで、それをコンピュータで実行できるように表記したものがプログラムだ、という話 (前期の復習)

- Ruby の「三角形の面積」プログラムを提示しそのまま動かさせる。

```
def triarea(w, h)
  s = (w * h) / 2.0
  return s
end
```

- 動いたら「2 数の和」「円柱の体積」など同様の (単純計算の) プログラムを書かせる
- すべての例題はメソッドの形で書き、エディタでファイルに打ち込ませ、irb でロードして引数をつけて呼び出し実行させる

- 前年の Java に比べると、入力する行数が圧倒的に少なく、おまじないが少ないため、最初の演習に掛かる労力は非常に少なくなった。これはさすがだと思った。
- しかしそれでナメていたため、「エディタでファイルに打ち込む」というのを丁寧に説明せずにはまった。学生はそういうことをしたことがないので苦労していた (が、1 週間経ったらあとはまったく OK)

□ 数値の表現について、整数の 2 の補数表現と、実数の浮動少数点表現の説明。数の有限性/有限精度、誤差 (情報落ち、桁落ち) の説明

- Java のときは整数があふれて負になるとかを体験する課題を出していたが、Ruby は勝手に多倍長にするためこれができない。しかたないので時間計測法を提示して「ある値から先は計算時間が長くなるが、そのある値はいくつか調べよ」という課題にした。
- 浮動少数点の誤差を実際に観察する課題。これは Java でも同じ。非常に緻密に調べてくれた学生もいるし、適当にやって誤差がありました、程度の学生もいた
- 全体として、単なる計算のコードだけでできる課題なので初回の B 課題にぴったり

3.2 #2: 制御構造/数値積分

□ 前述の通り、課題として出した整数演算時間の段差と実数の誤差の問題について解説

□ 制御構造がないと何もできないので、とにかくまず if 文を教える。

- 例題は「絶対値の計算」で、1 つの動作に複数の書き方があることを強調

```
def abs1(x)
  if x < 0
    result = -x
  else
    result = x
  end
  return result
end

def abs2(x)
  if x < 0
    return -x
  else
    return x
  end
end

def abs3(x)
  result = x
  if x < 0 then result = -x end
  return result
end
```

- これを打って動かした後「2 数のより大きいものを打ち出す」「3 数の〃」を演習してもらう。3 つ、4 つになると初心者はそれなりに苦労するので楽しんでもらえる

□ if 文が終わったら while 文を説明し、「積分の公式を知らなくても積分が求められる」と言って数値積分の概念を説明する

- 例題は while 文を使って 2 次関数の定積分 (正解が分かっている) を求める→誤差がバラバラ→dx=W/N を累計しても誤差のためループが N+1 回になったりすることを示す→回数を決めてループしカウンタから x を求める→計数ループを教える→刻みを小さくするとそれに応じて安定に誤差が減っていることを示す

```
def integ1(a, b, n)
  dx = (b - a) / n;
  s = 0.0
  x = a
  while x < b
    y = x**2        # 関数 f(x) の計算
    s = s + y * dx
    x = x + dx
  end
  return s
end

def integ2(a, b, n)
  dx = (b - a) / n;
  s = 0.0
  n.times do |i|
    x = a + i * (b - a) / n;
    y = x**2        # 関数 f(x) の計算
    s = s + y * dx
  end
  return s
end
```

- 単調増加な関数で左端公式 (?) だと常に小さいことを説明し、ヒント (中点公式、台形公式、両者の混合の考え方) を示して次回までの演習問題としてやらせる
- このほか、sin/cos のテーラ展開式を示してこれをループで計算してみるという問題も用意
- さらに制御構造の演習として、素数の判定と素数の列挙 (ナイーブに全部割ってみる方式) の課題も出し、速度改良してみよと指示。この課題は例年、学生に人気がある (計算量の前フリにもよい)
- 計数ループは「数値.times do ... end」を教えてしまい、ついに現時点まで for 文を教えていない。Ruby なんだから Ruby らしい方がいいじゃん?

3.3 #3: f(x)=0 の求解/データ型

□ 前回課題の説明として、シンプソンの公式による積分の説明と打ち切り誤差の話をする。テーラ展開式をそのまま計算すると誤差でめちゃくちゃになる話もする

□ f(x)=0 の解も数値的に解けるよという話をしてから、単調増加関数を前提として区間 2 分法の説明とニュートン法の説明を (あらすじだけ) して、そのコードを自力で演習として書いてもらう

- これまでは例題をちょっと手直し程度だったが、この演習ではゼロからコードを書いてもらう。でも難しいので、その場で穴埋め程度になるようにヒントを出した

□ 基本データ型 (数値、文字、論理値、nil) と複合データ型 (配列、レコード、オブジェクト) の説明

- Ruby だと文字列がオブジェクトで文字型がないので苦しい。また、シンボル型があるのでどうしようかと思ったが説明してしまう (レコードを作る時に必要)
- 配列については沢山つかうので具体的例題で説明。配列内容の合計を取る程度
- 素数列挙の改良として割ってみるのを$\sqrt{}$までででやめる等を例示し、その後「素数を配列に取っておいて素数だけで割ってみる」「エラトステネスのふるい」の考えを示して作ってみる課題を説明
- このあたりで、繰り返し、枝分かれ、配列を使ったプログラムがそれなりに書けるようになった感じ

3.4 #4: 手続き (関数) と副作用/レコードと画像

□ 前回課題の説明として素数列挙の問題を中心にプログラム例を解説

□ メソッドを分けることで「抽象化」がなされていたことを後づけで指摘。またグローバル変数の概念を説明し、グローバル変数を書き換えると副作用のあるメソッドになることを説明。続いて再帰関数の概念を GCD(最大公約数) で説明 (前にループによる最大公約数をやっているので)

```
def gcd(x, y)
  if x == y
    return x
  elsif x > y
    return gcd(x-y, y)
  else
    return gcd(x, y-x)
  end
end
```

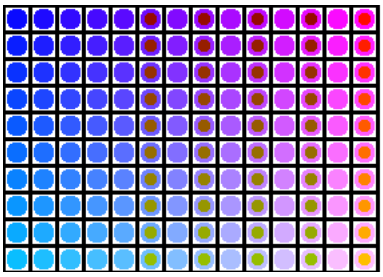
- 階乗、組み合わせの数、再帰的定義、非負整数の 2 進表現の 3 つについて再帰的定義を提示し、再帰メソッドを書かせる演習
- 例題と同様のパターンで書けばできるのですんなり再帰に入れるという印象

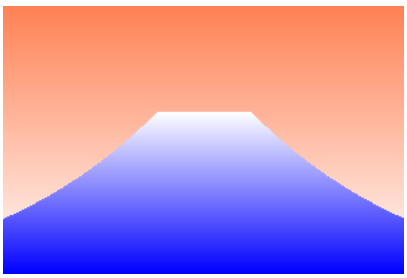
□ レコード型について解説し「Pixel = Struct.new(:r, :g, :b)」でピクセル値を表すレコードを例示。これの 2 次元配列で画像を表現できることを説明

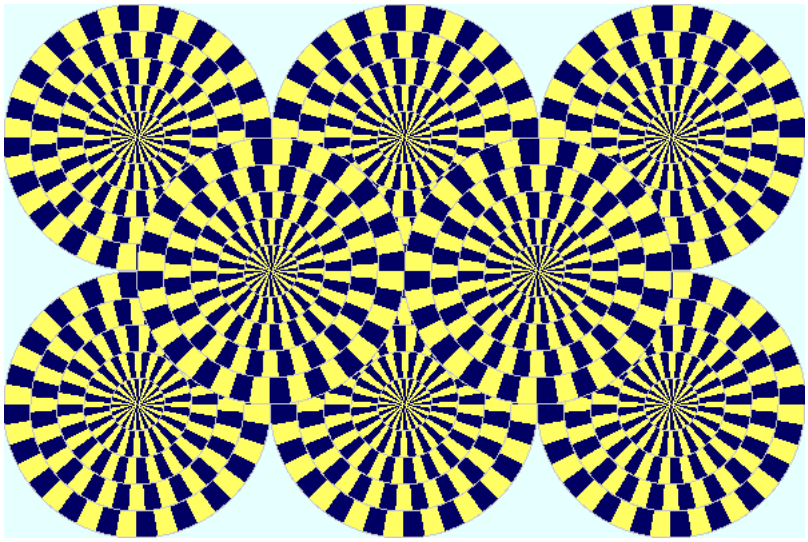
- initimage(画像データの初期化)、writeimage(ファイルへの PPM 形式での書き出し)、fillcircle(円を塗りつぶし) を示し、この程度で画像が作れることを示したあと、長方形や直線などを作る課題と「美しい絵」を作る課題を提示

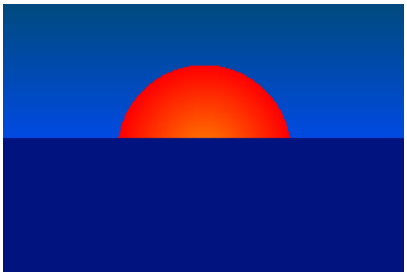
```
def mypicture
  initimage
  fillcircle(150, 30, 60, 255, 100, 70)
  fillcircle(190, 100, 30, 100, 200, 80)
  writeimage("t.ppm")
end
```

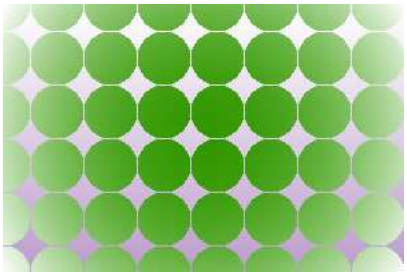
□ 「美しい画像」について、簡単な絵でも構わないとしたが、学生は皆いろいろ頑張ってくれた。

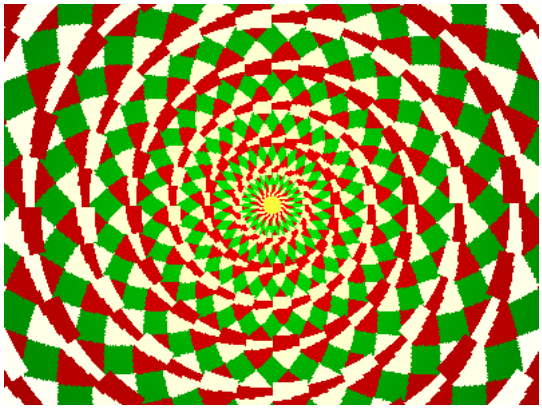


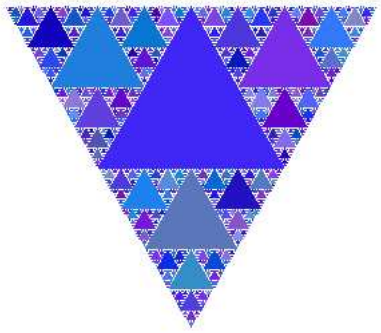


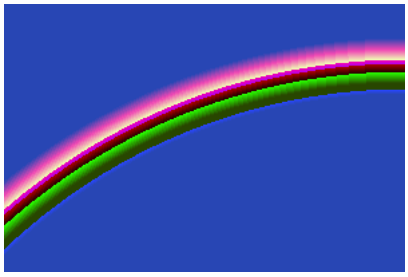


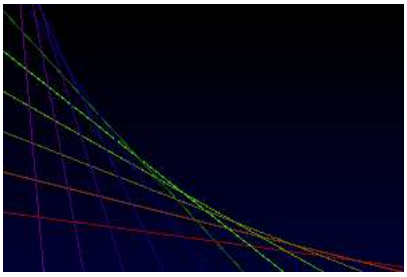


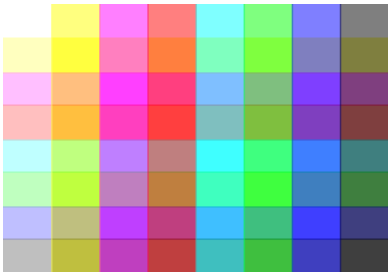


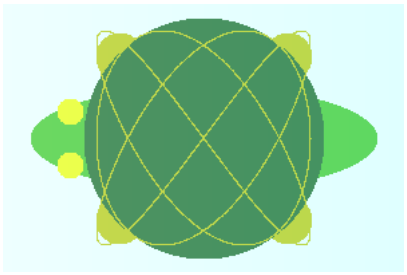


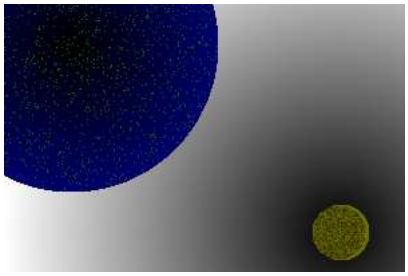


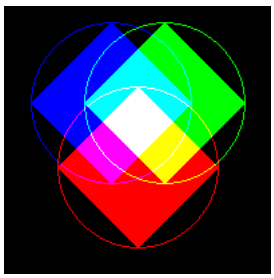


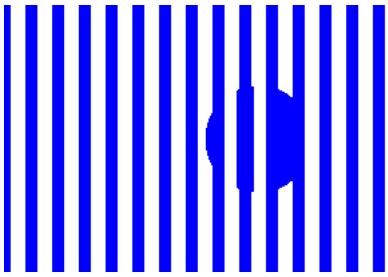


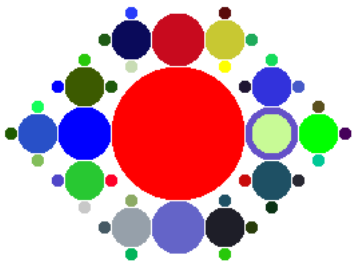


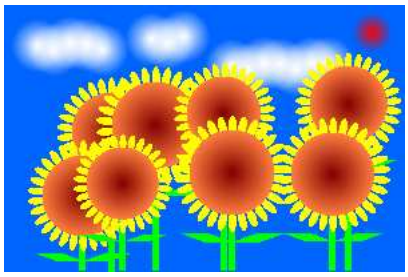


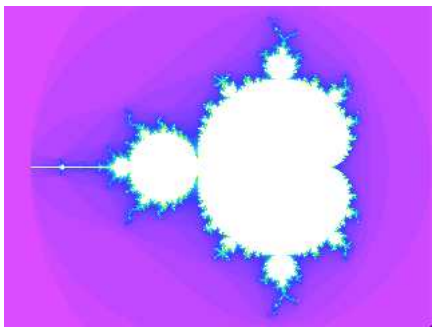


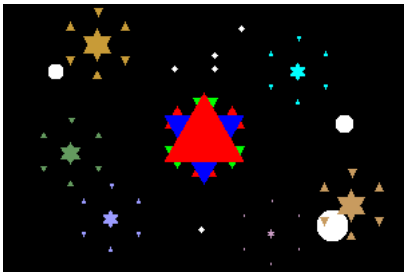












3.5 #5: さまざまな整列手法/時間計算量

- □ 配列の整列について#3 で出した課題の解説として単純選択法、バブルソートを示し、N を変化させて時間計測を行う演習
- □ マージソート、クイックソートを解説
- □ 時間計算量の考え方を解説
- □ ビンソート、基数ソートの考え方を解説し実装する課題を提示
- □ その他の題材として最大公約数のユークリッド互除法、フィボナッチのループ版と 22 行列版、組み合わせのパスカル三角形版を考え方だけ示し実装と時間計算量の計測/見当を課題として提示

3.6 #6: 連立方程式の数値解法/ランダムアルゴリズム

- □ 前回課題の各種アルゴリズムの実装例と時間計算量分析
- □ 連立一次方程式のガウス消去法を説明し、Gauss-Jordan に直す演習。また Jacobi 法の原理と実装例を説明しテスト実行による収束の検討と Gauss-Seidel 法に直す演習
- □ 乱数と擬似乱数、ランダムアルゴリズム、モンテカルロ法による数値積分について説明し適当な関数の積分と試行数による精度分析を課題として提示
- □ 乱数を使ったゲームの例として「数当て」を説明し、任意のゲームを作ってにみるという課題を提示

```
def kazuate
  a = (rand(10000)+10000).to_s[1..4]; count = 0
  while true do
    printf("your guess? ")
    s = gets; hit = 0; blow = 0
    4.times do |i|
      4.times do |j|
        if s[i] == a[j] then
          if i == j then hit = hit + 1
          else           blow = blow + 1
          end
        end
      end
    end
```

```
    if hit == 4 then puts "you win!"; return end
    count = count + 1
    if count > 9 then
      puts "you lose! answer = #{a}."; return
    end
    puts "hit = #{hit}, blow = #{blow}."
  end
end
```

□ ゲームを作りたい人はかなり頑張ってゲームを作り提出してくれた (文字だけでやる RPG とか)

3.7 #7: 常微分方程式の数値解法/オブジェクト指向

□ 常微分方程式とは何か整理した後、Euler 法、Runge-Kutta 法 (2 次) の考え方とコード、および Runge-Kutta 法 (4 次) の考え方と公式までを示し、精度を検討するかまたは 4 次の実装をする演習

□ オブジェクト指向とはプログラムが扱うデータを「もの」として考える「考え方」であると説明し、クラス、インスタンス、インスタンス変数、インスタンスメソッドなどの言語機構/概念を解説

- 例題として「犬」のクラスを示しその変更を演習として提示

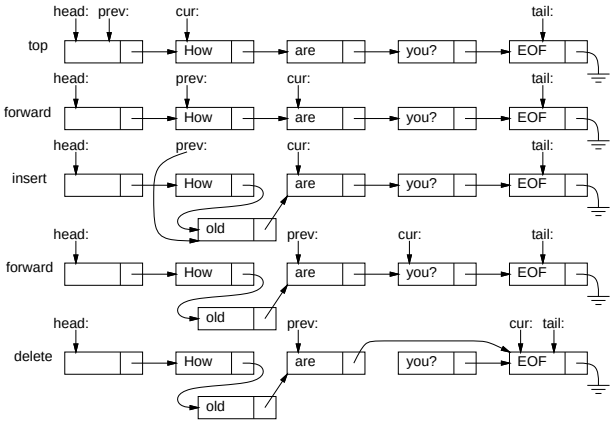
```
class Dog
  def initialize(name)
    @name = name; @speed = 0.0; @count = 3
  end
  def talk
    puts('my name is ' + @name)
  end
  def addspeed(d)
    @speed = @speed + d
    puts('speed = ' + @speed.to_s)
  end
  def setcount(c)
    @count = c
  end
  def bark
    @count.times do puts('Vow! ') end
  end
end
```

- 「有理数」クラスを示しその演算追加と「複素数」クラスの作成を演習として提示

3.8 #8: 動的データ構造/表と探索

□ 動的 (再帰的?) データ構造について説明し、単リストとその操作を説明

```
class Buffer
  Cell = Struct.new(:data, :next)
  def initialize
    @tail = @cur = Cell.new("EOF", nil)
    @head = @prev = Cell.new("", @cur)
  end
  def atend
    return @cur == @tail
  end
  def top
    @prev = @head; @cur = @head.next
  end
  def forward
    if atend then return end
    @prev = @cur; @cur = @cur.next
  end
  def insert(s)
    @prev.next = Cell.new(s, @cur); @prev = @prev.next
  end
  def print
    puts("  " + @cur.data)
  end
```



□ 演習として「現在行削除」「現在行と次の行の交換」「1 つ戻る」「全体の順序逆転」から 1 つ作ってもらう

- おまけとしてファイル読み書き、置換コマンドの実装を説明 (本当にエディタとして使えるようにするために不可欠)

□ 表と探索の概念を説明し、線形探索、2 分探索木の実装例を時間計測例とともに示す。時間計算量の検討を課題として提示

- Ruby の Hash について表の一種として説明し、具体的な記法や使い方を示したあと、これの時間計測も課題として提示

3.9 #9: 抽象構文木/動的分配/字句解析/再帰下降解析

□ 前回課題の解説として単リストの操作を解説。またハッシュ法の解説と実装例を示し、線形探索、2 分探索木、Ruby の Hash も含めて時間計測と計算量の検討例を示す

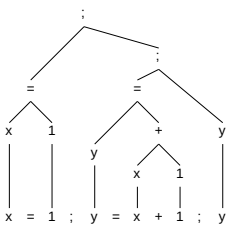
□ 式木/抽象構文木の概念を説明し、簡単な式木を組み立てて評価する例題を提示した後、演算の追加や制御構造の追加を演習

```
$vars = {}
class Add
  def initialize(l, r) @left = l; @right = r  end
  def exec() return @left.exec + @right.exec end
  def to_s()
    return '(' + @left.to_s + ' + ' + @right.to_s + ')'
  end
end
class Mul
  def initialize(l, r) @left = l; @right = r end
  def exec() return @left.exec * @right.exec end
  def to_s()
    return '(' + @left.to_s + ' * ' + @right.to_s + ')'
  end
end
class Lit
  def initialize(v) @left = v end
  def exec() return @left end
  def to_s() return @left.to_s end
end
class Var
  def initialize(v) @left = v end
  def exec() return $vars[@left] end
  def to_s() return @left.to_s end
end
def test
  $vars['x'] = 5
  e = Add.new(Var.new('x'), Lit.new(1))
  puts(e); puts(e.exec)
  e = Add.new(Lit.new(3),
              Mul.new(Var.new('x'), Lit.new(2)))
  puts(e); puts(e.exec)
  e = Mul.new(Add.new(Var.new('x'), Lit.new(3)),
              Lit.new(2))
  puts(e); puts(e.exec)
end
```

```
  irb(main):004:0> test
  (x + 1)
  6
  (3 + (x * 2))
  13
  ((x + 3) * 2)
  16
  => nil
```

□ 類似したノードクラスを継承により短く定義できることを示す

□ ループ文なども同様に定義できることを示す




```
class Assign < Node
  def initialize(l, r) super; @op = '=' end
  def exec()
    v = @right.exec; $vars[@left.getleft] = v; return v
  end
end
class Seq < Node
  def initialize(l, r) super; @op = ';' end
  def exec() @left.exec; return @right.exec end
end
class Loop < Node
  def initialize(l, r) super; @op = 'L' end
  def exec()
    v=0; @left.exec.times do v=@right.exec end; return v
  end
end

def test1
 e =
  Seq.new(
   Assign.new(Var.new('n'), Lit.new(5)),
   Seq.new(
    Assign.new(Var.new('x'), Lit.new(1)),
    Seq.new(
     Loop.new(
      Var.new('n'),
      Seq.new(
       Assign.new(Var.new('x'),
        Mul.new(Var.new('x'), Var.new('n'))),
       Assign.new(Var.new('n'),
        Sub.new(Var.new('n'), Lit.new(1))))),
      Var.new('x'))))
 puts(e)
 return e.exec
end
```

```
  irb(main):006:0> test1
  (((n$)=(5#));(((x$)=(1#));(((n$)L(((x$)=
  ((x$)*(n$)));((n$)=((n$)-(1#)))));(x$))))
  => 120
```

□ 字句解析器の考え方とコード例を示す

```
class Scanner
  def initialize(s) @str = s + '$'; @pos = 0 end
  def peek() return @str[@pos..@pos] end
  def next()
    if @pos < @str.length-1 then @pos = @pos + 1 end
  end
  def to_s()
    return @str[0..@pos-1] + '!' + @str[@pos..-1]
  end
end
```

□ BNF、文法、構文解析木の解説。構文解析木を描かせる演習

□ 再帰下降解析器の考え方とコード例を示す

```
def prog(sc)
  s = stat(sc); c = sc.peek
  if c == '$' || c == '}' then return s
  elsif c == ';' then sc.next; return Seq.new(s, prog(sc))
  else puts('STAT:' + sc.to_s); return Noop.new
  end
end
def stat(sc)
  c = sc.peek
  if c == '{' then
    sc.next; p = prog(sc)
```

```
    if sc.peek != '}' then
       puts('NO_}:' + sc.to_s); return Noop.new
    end
    sc.next; return p
  elsif c == 'L' then
    sc.next; e = expr(sc); return Loop.new(e, stat(sc))
  else return expr(sc)
  end
end
def expr(sc)
  e = factor(sc); c = sc.peek
  if c == '+' then sc.next; return Add.new(e, expr(sc))
  elsif c == '-' then sc.next; return Sub.new(e, expr(sc))
  elsif c == '*' then sc.next; return Mul.new(e, expr(sc))
  elsif c == '/' then sc.next; return Div.new(e, expr(sc))
  elsif c == '=' then sc.next; return Assign.new(e, expr(sc))
  else return e
  end
end
def factor(sc)
  c = sc.peek; sc.next
  if c >= 'a' && c <= 'z' then return Var.new(c)
  elsif c >= '0' && c <= '9' then return Lit.new(c.to_i)
  elsif c == '(' then
    e = expr(sc)
    if sc.peek != ')' then
      puts('NO_):' + sc.to_s); return Noop.new
    end
    sc.next; return e
  else puts('FACTOR:' + sc.to_s); return Noop.new
  end
end
def test2
  e = prog(Scanner.new('n=5;x=1;L5{x=x*n;n=n-1};x'))
  puts(e)
  puts(e.exec)
end
```

□ 昨年よりはかなり譲歩したが、まだ「難しい」との意見多数…

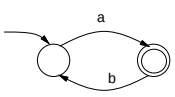
- 簡単のため式が演算子右結合にしてあるのはよかったのか…

3.10 #10: スタックとキュー/状態空間の探索

□ 抽象データ型としてのスタックとその実現 (配列、リスト)。抽象データ型としてのキューとその実現 (配列、リスト)。いずれも実例は配列版のみ示し、連結リスト版を実装する演習課題

- 式木を深さ優先 (スタック)、幅優先 (キュー) でたどってみせる
- JR 路線図をたどる課題、エディタバッファを 2 つのスタックで実装する課題

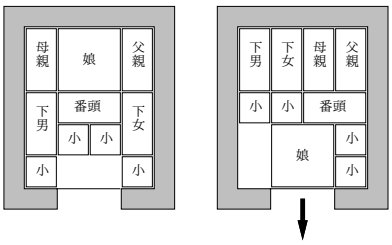
□ 有限オートマトンの復習/オートマトンを Ruby で実装




```
$atm = [{'a' => 1},
        {'b' => 0, :final => true}]
def accept(s)
  cur = 0
  s.length.times do |i|
    k = $atm[cur][s[i..i]];
    if k == nil then return false else cur = k end
  end
  return $atm[cur][:final] == true
end
```

- Hash のリテラルを使うことで「まんま」書ける感じ→ Ruby の利点
- いくつかオートマトンを示して受理するものは何か考えさせる。また Ruby で動かして確認させる

□ パズルやゲームが状態空間探索として定式化できることを説明し、例として箱入り娘を解くコードを提示 (64 行)



- これは昨年の内容になかったものだが、奥さんに「状態探索がなかったら CS じゃない」と言われてその気になったので…
- 課題として好きなゲームやパズルを解くものを提示

3.11 #11: 動的計画法/パターン認識

□ 動的計画法→パターン認識の前フリとして標準スライドで指定

- 動的計画法の考え方を「釣り銭問題」「ナップサック問題」で説明し、「釣り銭問題」の硬貨群の変更、「ナップサック問題」のトレースバック表示、「合宿の部屋わり」問題のプログラムのいずれかを作ってもらう
- 多くの学生にとって (ほぼ全員)「考え方は分かっても自分で作れるようになる気がしない」。課題も一番易しいものの提出ばかりだった

□ 文字列のアラインメント→できればこのプログラムを自力で作れるようになって欲しいところだが、授業 1 回では難しいかも

□ パターン認識、forward algorithm、Viterbi アルゴリズム→一応説明してコードも掲載しているが、これも 1 回では時間がない感じ

4 まとめ

□ まだ「さまざまなプログラミング言語」1 回+Java2 回を残していますが…

□ Java から Ruby へ→思ったより変わらなかった。

- Ruby だからいい点…短く書ける、irb で配列リテラル等を打ち込める (文字からデータ構造を作る部分のコードが要らない)
- Ruby だと困る点…グラフィクスがない、勝手に多倍長など仕掛けが埋もれている
- 強い型検査、コンパイラのチェックはちゃんと通す習慣をつけて、それからスクリプト言語に行って欲しいと思う

□ 昨年よりハードさは和らいだがまだハードだと思う

□ 学生さんの感想 (下記) を見ていると、やはりプログラミングを学んでいるという意識が強い。それはプログラミングが魅力的だからと思うことにして… :-)

□ 各回の題材についてはそれなりに興味を持ってもらえていると思う (いかにも情報科学なので学生もそう思ってくれている?)

□ これからも、「情報科学を学んでもらいながら」「プログラミングも楽しんでもらう」ことを目指して頑張ります

5 学生アンケートから

□ すべてのレポートで 3 問ずつアンケートに回答してもらっている

- だいたいは、2 個の内容についての質問+感想・要望
- #11 だけはプ会向けの質問を 2 つ入れた

□ Q. この講義は「情報科学」の学習を目的とし、その確認手段として「Ruby によるプログラミング」を演習して頂いていますが、本クラスにおけるその両者の位置づけやバランス、難易度についてどう思いますか。

- 一部非常に難しい演習もありましたが、それを除けばバランスは取れていると思います。
- 情報科学としてはバランスはちょうど良かったと思います。難易度は途中で急に上がったような…
- こういうのはよくわかりませんが、個人的には小難しい話を延々とするよりはこの講義のように実習を多めに取り入れるほうがいいと思います。ただ難易度はもうちょっと低くてもいい気がします。
- 講義の難易度が途中から著しくあがった気がします …
- 情報科学についても、Ruby によるプログラミングも、そこそこ理解できた気がします。比較的易しい内容や、プログラミングに関してはほとんど理解できなかった内容もありました。Ruby を通して情報科学の学習を行うというのは出来たと思います。
- 情報科学の考え方が理解できても、いざそれを Ruby で表現するとなると大きな困難が伴い、大変苦戦しました。個人的な意見としては情報科学の学習が主目的であれば、プログラミングに関してはもうちょっと易しくしてもよかったような気がします。
- バランスはちょうど良いと思いました。難易度は、後半が結構高かったです。
- この講義では演習の方に重点を置いていたような気がします。原理的なものについては理解できても、実際プログラムを作ろうとすると難しいことが多かったです。
- 内容は理解できても、ルビー化できないときが多々ありました。難しいです。
- 少し課題が大変なときもありましたが、全体としてちょうどいい難度だったと思います。 はじめてのプログラミングでしたが、楽しくなんとかついていくことができました。
- この授業はプログラムの実習が中心だったので、もう少し, 解説に力を入れてほしかったと思います。
- この講義は、位置付け、バランス、難易度のいずれも程良い感じの良い講義だと思います。効率良く情報科学の内容が身についたように思います。
- Ruby は簡素な言語だなあと思いました。内容は時間の指数関数的に難しくなっていった気がします。
- Ruby プログラミングの難易度が高すぎる気がします。初学者にはきついです。
- どちらかというと、Ruby プログラミングがメインだったような感じがしないでもないです。ただ、プログラミングができる=情報科学が学べている、というきもするし、なんともいえないです。
- 後半はプログラミング未経験者には無理でしょう。継承や演算子オーバーロード (実質) がいきなり出てくるのは……(苦笑
- 情報科学を学んでいるというより、どっちかというとほとんど「Ruby でのプログラミング」を学んでいるという感じでした。ただ、そっちのほうがよかったので、たくさん学べてうれしかったですが。難易度はクラスぐらいまではよかったのですが、動的データ構造あたりからはかなりきつく、おもちゃ言語では力尽きました。
- 情報科学を学ぶために Ruby を利用するのは最初疑問でしたが、まぁありかなって思います。

- ちょうど良かったです。時々与えられる自由な課題が大変でしたが、やりがいがありました。

□ Q. 講義開始時の自分を振り返り、今回までの講義で自分がどんなことを学んだか簡単にまとめて見てください (全体的な感想も含めてどうぞ)。

- ･自分はプログラミングの経験はありましたが、Ruby を新たに習得することができました･実用的なプログラミングができるぐらいのアルゴリズムは学べたと思います。･でも結局完全には理解できていない理論はあります。そして、あやふやなアルゴリズムもあります。試験までに復習しないと…
- Ruby が書けるようになったのがよかったですね。課題も楽しかったです。
- Ruby でちょっとしたプログラムをつくれるようになってくらいであまり進歩を感じませんが、プログラミングに取り組む中で「こういうモノがあって、こういう使い方が出来る」というようなことをたくさん見た気がします。
- エディタバッファとかは今後使いそうです。後半の授業は難しかったですが、得るものも多かったです。
- とりあえず、プログラミング自体初めて扱うものだったので、新鮮でした。なかなか複雑なもので、自分で新しいプログラムを作るのはかなり難しそうですが、この講義で基礎を学べたと思います。
- 講義開始時に比べて、パソコンがどのようなロジックで動いているのかがよく理解できるようになった。またごく簡単なものに関してだけであるが、プログラムを書いたり、書いてあるものを理解できるようになった。あと、記号とか打つのが速くなりました。
- 講義開始時は再帰的なものがとても苦手だったのですが、現在ではある程度扱えるようになりました。
- プログラムの計算量について結構理解できまるようになりました。同じことをやるのでも、プログラムの種類によってかかる時間がかなり違っているということには驚かざるを得ません。
- 数学に似てても全く異なる分野でした。使いこなせるのは html どまり…
- まずプログラミングという、色々な手法を組み合わせて、一つの動作をおこすものが作れるんだとわかりました。そういうものを自分で作ってうごかすのは、楽しかったです。
- この情報科学の授業で学んだ事で最も興味深かったのは誤差と計算量でした。 浮動小数点、桁落ち、情報落ちの仕組みを理解する事で，これまで原因が分からなかった乗除計算の時に微小な誤差が生じる仕組みが理解できました。また、同じ目的のプログラムでも計算方法が違う事で計算時間に非常に大きな差が出る事は実際にプログラムを動かす事で実感する事ができたと思います。
- ソートの辺りからは (OO はなんとなく分かっていましたが) 知らないことばかりで、いろいろ新しく勉強させてもらいました。
- アルゴリズムの組み立て方など
- データ型やクラスなどの概念は分かったと思いますが、プログラミング言語としての Ruby はまだまだ分からないことだらけです。あと、/の逆向きの記号がまだ打てません。なので、上のプログラムも/(の逆)n がありません。
- プログラミングというものを習得し、コンピュータの動くしくみの一部を垣間見た。というのが、「学んだこと」に対して真っ先に思い浮かんだものです。
- アルゴリズム関連弱かったので、経験できてよかったです。
- 講義開始時はプログラムなんて全く分からず、もちろんソースなんか見てもただの意味不明な羅列にしかすぎなかったのですが、現在ではソースを見て何をしているのかなんとなくは理解できるようになりました。数学的な話題 (連立方程式とか微分方程式とか) はほとんど忘れてしまっていますが、基本的な構文、クラスの使い方、配列、ハッシュ等は大方使えるようになりました。興味深い内容も多くあり、楽しくまなぶことができました。(字句解析器はきつかったですが)
- 内容が量的に結構大目で大変でしたが、プログラムがどんな風に組み立てられているのがが少しわかって面白かったです。
- ものごとの表現方法はいろいろあるのだと思いました。