「情報科学」2009 年度版演習問題解答例 ver1.1

解答例作成者 1-7 章 平野, 8-9 章 大筒

まえがき

増原英彦氏「情報科学　2009 年度版」に収録されている練習問題（1〜9 章）の解答例を収録してある。
多々、増原氏のソースを転用したところがあり、この場を借りて増原氏に感謝する。

ソースはあくまで解説のために書いたなので、以前に定義した関数をロードせずに用いているところがあります。なので、そのまま通そうとするとエラーが出ることがあります。
分からないところがあったり、間違えてるってところがあれば、それぞれの解答例作成者に訊けばどういう意図でソースを書いたのかとかを含めていい回答ができると思います。
解説を当てて行こうとも思ったけど、コーディングに難しさがある奴はどうせテストに出ない（出せない）んで、アルゴリズムだけ一見して「ふーｎ」程度にでも頭にとどめとけば OK.
ただ、大事なのは**コードが読める**こと。試験でもこの点が割と重視されてます。特に、CT スキャンのやつなんかは、この中ではかなり難しいほうに入るんでどういう仕組みなのかも含めて考えてみると面白いし、力つくかも。

1　1 章

練習　1.1 略

練習　1.2 (関数の定義) 次のような関数を定義せよ。さらに、それを使った計算の例を示せ。

a) 平面上の 2 点 (x, y) と (u, v) の距離を求める `distance(x,y,u,v)` .
b) f フィート i インチをセンチメートルに変換する `feet_to_cm(f,i)` . ただし、1 フィート= 12 インチ= 30.48 cm である。
c) p ポンド o オンスをキログラムに変換する `pound_to_kg(p,o)`. 1 ポンド= 16 オンス= 0.4536 kg である。

解答例　1.2

```
include Math
#a)
def distance (x,y,u,v)
    return sqrt((x-u)**2+(y-v)**2)
end

#b)
def feet_to_cm(f,i)
  return (12*f+i)*30.48/12.0
end

#c)
```

```
def pound_to_kg(p,o)
  return (p*16+o)*0.4536/16.0
end
```

練習 1.3 略

練習 1.4 略

練習 1.5 略

練習 1.6 略

練習 1.7 略

練習 1.8 (もっと関数の定義) 次のような関数を定義せよ。さらに、それを使った計算の例を示せ。

a) セ氏温度 c を華氏温度に変換する `celsius_to_fahrenheit(c)` .
b) 華氏温度 f をセ氏温度に変換する `fahrenheit_to_celsius(f)`.
c) 毎秒メートル (m/s) で表わされた速度 v を毎時マイルに変換する `ms_to_mph(v)` .
d) 上と逆の変換をする `mph_to_ms(v)` .
e) 米国では気温 (華氏 t 度) 風速 (v 毎時マイル) を加味した体感温度 (Wind Chill Index) を次のような式で定めている。

$$35.74 + 0.6215t - 35.75(v^{0.16}) + 0.4275t(v^{0.16})$$

この変換をする関数 `wind_chill_index(t,v)` . (例えば 2007 年の 10 月 20 日のニューヨークは華氏 20 度、風速 20 毎時マイルであった。)
f) 上で定義されている体感温度を、気温がセ氏、風速が m/s で与えて、結果もセ氏で求めたい。そのような関数 `wind_chill_index_celsius(t,v)` .
g) 二次方程式 $ax^2 + bx + c = 0$ に関して
   (a) 判別式 $b^2 - 4ac$ を求める det(a,b,c).
   (b) 解の 1 つ $\dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$ を求める `solution1(a,b,c)` . (det を使って定義せよ。)
   (c) もう 1 つの解 $\dfrac{-b - \sqrt{b^2 - 4ac}}{2a}$ を求める `solution2(a,b,c)` . (solution1 と solution2 の共通部分を 1 つの補助関数にできるか?)
   (d) 二次関数 $f(x) = ax^2 + bx + c$ の値を求める quadratic(a,b,c,x).

解答例 1.8

```
#a)
def celsius_to_fahrenheit(c)
  return 9*c/5.0+32
end

#b)
def fahrenheit_to_celsius(f)
  return 5*(f-32)/9.0
end

#c)
def ms_to_mph(v)
  return v*3600/1609.344
end

#d)
def mph_to_ms(v)
  return v*1609.344/3600.0
end

#e)
include Math

def wind_chill_index(t,v)
  return 35.74+0.6215*t-35.75*(v**0.16)+0.4275*t*(v**0.16)
end

#f)
include Math

def wind_chill_index_celsius(t,v)
  ft=9*t/5.0+32
  mph=v*3600/1609.344
  f=35.74+0.6215*ft-35.75*(mph**0.16)+0.4275*ft*(mph**0.16)
  return 5*(f-32)/9.0
```

```
end

#g)
include Math
def det(a,b,c)
  return b**2-4*a*c
end

def solution1(a,b,c)
  return (-b+sqrt(det(a,b,c)))/2*a
end

def solution2(a,b,c)
  return (-b-sqrt(det(a,b,c)))/2*a
end

def quadratic(a,b,c,x)
  return a*(x**2)+b*x+c
end
```

練習 1.9 略

練習 1.10 (局所変数の有効範囲) 第 1.4.3 節で定義した関数 heron の外で変数 s が使われていた場合、関数呼び出しの後の s の値はどうなっているだろうか。次の命令と式を順に実行・計算し、関数定義の中と外で使われている変数 s の関係について説明を試みよ。

解答例 1.10 メソッド heron 内で定義された変数 s はローカル変数（局所変数）であるので、スコープ外の変数にアクセスすることはできない。よって、関数呼び出し後の s の値はメソッドの実行にかかわらず不変である。

練習 1.11 (局所変数と補助関数) ヒラメキの良い市谷くんは局所変数を定義せずに heron と同じ計算をする関数が定義できたと主張している。青山さん:「疑わしいわね。s の定義を展開したんじゃないの?」市谷くん:「そんなことはないさ。$0.5 * (a + b + c)$ という計算は一度しかやらないよ。補助的な関数を定義するのが鍵なんだ。」市谷くんが定義した heron はどのようなものか、作ってみよ。

解答例 1.11

```
include Math

def s(a,b,c)
  0.5*(a+b+c)
end

def calc(s,a,b,c)
  sqrt(s * (s-a) * (s-b) * (s-c))
end

def heron (a,b,c)
  calc(s(a,b,c),a,b,c)
end
```

練習 1.12 (わざと間違ってみる) 第 1.5.3 節で紹介した他にも、誤りの種類は数多くある。実際にエラーメッセージを見てから誤りを探すことは時として簡単でない。練習のために、意図的に誤りを混入させたファイルを作り、どのようなエラーメッセージが表示されるかを観察しておけ。後日、同じようなエラーメッセージが表示されたときの原因究明に役立つはずである。例えば、次のような誤りを含んだファイルを作り、読み込ませてみよ。

a) ファイル 1.1 の 1 行目の# 記号を取り除いたもの。
b) 空白の挿入。定義の中の適当な位置に空白文字を入れてみよ。場所によっては誤りになるし、場所よってはならない。
c) 改行の挿入。定義の中の適当な位置で改行してみよ。場所によっては誤りになるし、場所よってはならない。
d) end の不足。定義の終わりを示す end を取り除いてみよ。できれば、複数の関数定義があるファイルで、最初の定義の end を取り除いてみよ。
e) いわゆる全角文字、つまり、日本語文字コードが混ざったもの。例えば空白を全角空白に置きかえたものや、数字を全角の数字に置きかえたものを試みよ。

解答例 1.12

a) syntax error
b) 空白の場所によってさまざま。undefined method だのいわれるときもあるし、unexpected tIDENTIFIER とか言われるときもある。要は、どこに空白を入れてもいいか、どこに入れてはいけないかをきっちり理解していることが大事。
c) 上とほぼ同じ
d) unexpected `$end`
e) invalid multibyte char とか。

2 2章

練習 2.1 略

練習 2.2 (画像データの操作) 画像データの座標 (x, y) の点と、その周囲の 8 点の合計 9 点の明度の平均値を計算する関数 `image_average9(image,x,y)` を作れ。定義を簡単にするために image は画像データ、座標 (x, y) は常に画像の内側の点になっていると仮定してよい。つまり image の幅と高さが h, w だったとき、$1 \leq x \leq w-2, 1 \leq y \leq h-2$ だと仮定してよい。(整数を整数で割ると値が切り捨てられてしまうので、そうならないように工夫せよ。)

解答例 2.2

```
def image_average9(image,x,y)
  return (image[x-1][y-1]+image[x][y-1]+image[x+1][y-1]+
  image[x-1][y]  +image[x][y]  +image[x+1][y]+
  image[x-1][y+1]+image[x][y+1]+image[x+1][y+1])/9.0
end
```

練習 2.3 (配列)

a) a は数値が並んだ 1 次元の配列とする。a の x 番目の値と、その前後の値の合計値を求める sum3(a,x) を作れ。ただし x の値は 1 以上かつ (a の大きさ-1) 未満であり、必ず前後に値があるものと仮定せよ。例えば a が [1,2,3,4] のとき sum3(a,1) は 6 に、sum3(a,2) は 9 になる。
b) a の x 番目の値と、その前後の値の平均値を求める `array_average3(a,x)` を作れ。ただし x の値は 1 以上かつ (a の大きさ- 1) 未満であると仮定せよ。例えば a が [1,2,3,4] のとき `array_average3(a,1)` は 2.0, `array_average3(a,2)` は 3.0 になる。

解答例 2.3

```
#a)
def sum3(a,x)
  a[x-1]+a[x]+a[x+1]
end

#b)
def array_average3(a,x)
  (a[x-1]+a[x]+a[x+1])/3.0
end
```

練習 2.4 略

3 3章

練習 3.1 (絶対値) 数値 x の絶対値を求める関数 abs(x) を定義せよ。

解答例 3.1

```
def abs(x)
  if x>=0
    x
  else
    -x
  end
end
```

練習 3.2 (0 ばかりの配列を作る) 大きさ n で中身がすべて 0 であるような 1 次元配列を作る関数 make1d(n) を定義せよ。(ヒント: 3 行目から 8 行目までがほぼそのまま関数定義の中身になる。)

解答例 3.2

```
def make1d(n)
  a = Array.new(n)
  for i in 0..(n-1)
    a[i] = 0
  end
  a
end
```

練習 3.3 (2 次元配列) h 行 w 列の配列を作る make2d(h,w) を定義せよ。ただし、作られる配列の中身はすべて 0 にせよ。

解答例 3.3

```
def make1d(n)
  a = Array.new(n)
  for i in 0..(n-1)
    a[i] = 0
  end
  a
end
```

#同じディレクトリに `make1d` **が定義されてある** `rb` **ファイルがあるならそれを** `load` **する形でもよい。**

```
#load("./make1d.rb")

def make2d(height,width)
  a = Array.new(height)
  for i in 0..(height-1)
    a[i] = make1d(width)
  end
```

```
  a
end
```

練習 3.4 式 3.2 の計算をする関数 b(r,x,y) を定義せよ (x, y だけでなく r も引数となっていることに注意せよ)。show(sphere(20)) を実行して図 3.1 のような画像が表示されることを確めよ。

解答例 3.4

```
include Math

def d(r,x,y)
    return sqrt((r-x)**2+(r-y)**2)
end
```

#distance.rb をロードして `distance(x,y,r,r)` **という形で呼び出してもよい。**

```
def b(r,x,y)
  if r >= d(r,x,y)
(r-d(r,x,y))/r
  else
    1
  end
end
```

練習 3.5 (条件分岐) 次のような計算を行う関数を定義せよ。

a) 2 次方程式 $ax^2 + bx + c = 0$ の実数解の個数を求める solutions(a,b,c). 判別式の値だけでなく、1 次方程式になっている場合も考えよ。

b) 2 次方程式 $ax^2 + bx + c = 0$ の実数解を 1 つ求める solve1(a,b,c). 判別式の値だけでなく、1 次方程式になっている場合も考えよ。

c) 3 つの異なる値 x, y, z が与えられたときの中央値を求める median(x,y,z). 中央値とは大きさ順に並べたときに真ん中に来る値のことである。

d) 年間所得が income 円だったときの所得税の税額を求める `income_tax(income)` . ただし 2008 年の時点での日本の所得税率は次のようになっている。
   (a) 195 万円以下の所得に対して 5% .
   (b) 195 万円より多く 330 万円以下の所得に対して 10% .
   (c) 330 万円より多く 695 万円以下の所得に対して 20% .
   (d) 695 万円より多く 900 万円以下の所得に対して 23% .
   (e) 900 万円より多く 1,800 万円以下の所得に対して 33% .
   (f) 1,800 万円より多い所得に対して 40% .
   ただし、上の税率は超過分についてのみ適用される。つまり、所得が 250 万円だった場合は、そのうちの 195 万円に対して 5% , 残りの 55 万円に対して 10% の税が課されるので、$195\times0.05+(250-195)\times0.10 = 15.25$(万円) が税額となる。東大生の家庭のうち約 19% は、950 万円から 1050 万円の年収を得ているという。収入と所得が同じだったとして、年間所得 1000 万円の場合の税額を計算してみよ。

e) 西暦 year 年 2 月の日数を求める `days_of_february(year)` . ただし、year は次の条件を満たすときは閏年であることに注意せよ。
   (a) year が 4 の倍数である場合は閏年である
   (b) 条件 (a) を満たしていても、year が 100 の倍数の場合は閏年でない
   (c) 条件 (b) を満たしていても、year が 400 の倍数の場合は閏年である。

f) 西暦 year 年 month 月の日数を求める `days_of_month(year,month)` .

解答例 3.5

```
#a)
def solutions(a,b,c)
  if a==0
    1
  end
  if det(a,b,c)>0
    2
  elsif det(a,b,c)==0
1
  else
0
  end
end

#b)
def solution1(a,b,c)
  return (-b+sqrt(det(a,b,c)))/2*a
end

def solve1(a,b,c)
  if a==0
-c/b
  elsif det(a,b,c)>=0
solution1(a,b,c)
  else
p "No answer"
  end
end
#c)
def median(x,y,z) #sort というインスタンスメソッドを使えばもっと楽にできます
  if x >= max(y,z)
    max(y,z)
  elsif y >= max(x,z)
    max(x,z)
  else
max(x,y)
  end
end

#d)
def min(x,y)
  if x>=y then return y
  else return x
  end
end

def income_tax(income)
  tax195 = min(income*0.05,195*0.05)
  tax330 = min((income-195)*0.1,(330-195)*0.1)
  tax695 = min((income-330)*0.2,(695-330)*0.2)
  tax900 = min((income-695)*0.23,(900-695)*0.23)
  tax1800 = min((income-900)*0.33,(1800-900)*0.33)
  tax1800o = (income-1800)*0.4

  if income<=195  then return tax195
  elsif income<=330 then return tax195+tax330
  elsif income<=695 then return tax195+tax330+tax695
  elsif income<=900 then return tax195+tax330+tax695+tax900
  elsif income<=1800 then return tax195+tax330+tax695+tax900+tax1800
  else return tax195+tax330+tax695+tax900+tax1800+tax1800o
  end
end
```

```
#e)
def days_of_february(year)
  if year%400 == 0 then return 29
  elsif year%100 == 0 then return 28
  elsif year%4 ==0 then return 29
  else return 28
  end
end

#f)
def days_of_month(year,month)
  case month
  when 1 then return 31
  when 2 then days_of_february(year)
  when 3 then return 31
  when 4 then return 30
  when 5 then return 31
  when 6 then return 30
  when 7 then return 31
  when 8 then return 31
  when 9 then return 30
  when 10 then return 31
  when 11 then return 30
  when 12 then return 31
  end
end
```

練習 3.6 (条件式の組み合わせ) 練習 3.5c で定義した関数 median を定義し直して、(a)if y < x という形の条件式だけ使った median1 と、(b) `&&` などを使って条件式を組み合て条件分岐の数を最小にした median2 を作れ。どちらの定義が分かりやすいだろうか。

解答例 3.6

`#median1` は先ほどの練習 `3.5c` で用いたものでよい

```
def median2(x,y,z)
  if (x>=z&&z>=y)||(y>=z&&z>=x) then return z
  elsif (y>=x&&x>=z)||(z>=x&&x>=y) then return x
  else return y
  end
end
```

練習 3.7 (真偽値を求める関数) 次のような計算を行う関数を定義せよ。

a) x が y で割り切れることを判定する divisible(x,y).
b) x < y < z であることを判定する ascending(x,y,z)
c) 西暦 y 年が閏年であるときに真となる `leap_year(y)`. ただし現行の閏年の条件は「西暦年が 4 で割り切れる年を閏年とする。但し西暦年が 100 で割り切れるものの中でさらに 4 で (100 で割った) 商が割り切れない年は平年とする」というものである。従って 1900 年は平年、2000 年は閏年、2008 年も閏年、2100 年は平年である。
d) 添字 i が配列 a の範囲内かどうかを判定する `within_range(a,i)`. ただし配列 a の添字は 0 からはじまり、a.length() の値未満である。
e) 座標 (x, y) が濃淡画像 img の中にあるかを判定する `within_image(img,x,y)`. (ヒント: 上で定義した `within_range` が使える。)

解答例 3.7

```
#a)
def divisible(x,y)
  if x%y == 0
    true
  else
    false
  end
end

#b)
def ascending(x,y,z)
  if y == median(x,y,z)
    if x<y then
      true
    else
      false
    end
  else
    false
  end
end

#c)
def leap_year(y)
  if days_of_february(y) == 29
    true
  else
    false
  end
end

#d), e)
def within_range(a,i)
  if 0<=i && i<a.length()
    true
  else
    false
  end
end

def within_image(img,x,y) #Ruby 自体デフォルトで多次元配列を実装してないので変な二次元配列を
image に格納するとパァです。
  if within_range(img,y)&&within_range(img[0],x)== true
    true
  else
    false
  end
end
```

練習 3.8 (論理関数) 真偽値を与えて真偽値を答えるような関数は論理関数という。次のような真理値表の通りに答えを返す論理関数 xor(x,y), implies(x,y) を定義せよ。

x	y	$xor(x,y)$	$implies(x,y)$
$false$	$false$	$false$	$true$
$false$	$true$	$true$	$true$
$true$	$false$	$true$	$false$
$true$	$true$	$false$	$true$

解答例 3.8

```
def implies(x,y)
  if x==true && y == false
    false
  else
    true
  end
end

def xor(x,y)
  if x==y
    false
  else
    true
  end
end
```

練習 3.9(文字列) 次のような関数を定義せよ。

a) 2 つの文字列 s, t のうち、長い方の文字列を求める longer(s,t).

b) 長さ 2 以上の文字列 s の先頭と最後の文字を取り除いた文字列を作る trim(s). 例えば trim(”abc”))”b” となる。

c) 文字列 s の前半と後半を入れ替えた文字列を作る upsidedown(s). 例えば upsidedown(”takeover”)⇒ ”overtake” となる。余力があれば奇数文字のときに upsidedown(”least”)⇒ ”stale”となるように定義してみよ。

解答例 3.9

```
def longer(s,t)
  if s.length() > t.length() then return s
  else return t
  end
end

def trim(s)#s.strip という便利なメソッドもあります。
  s = s[1..s.length()-2]
  return s
end

def upsidedown(s)
  if s.length()%2 == 0
    s1 = s[0..s.length()/2-1]
    s2 = s[s.length()/2..s.length()-1]
    s = s2 + s1
    return s
  else s.length()%2 ==1
    s1 = s[0..(s.length()-1)/2-1]
    s2 = s[(s.length()-1)/2+1..s.length()-1]
    s = s2+s[(s.length()-1)/2]+s1
  end
end
```

練習 3.10(条件分岐を使った配列の操作)

a) a は 1 次元の配列とする。a の x 番目とその前後の要素数を数える関数 length3(a,x) を作れ。ただし、l は配列 a の長さである。

$$length3(a,x) = \begin{cases} 3 & (0 < x < l-1) \\ 1 & (x = 0 and l = 1) \\ 2 & (l > 1 and (x = 0 or x = l-1)) \end{cases}$$

注: この関数は次に定義する sum3(a,x) で合計する要素の個数を求めている。

b) 練習 2.2 で定義した `image_average9(image,x,y)` および練習 2.3 で定義した `sum3(a,x)`, `array_average3(a,x)` を、x(や y) の値が配列の端だった場合でも正しく計算できるように定義を修正した関数 `image_average(image,x,y),sum,array_average(a,x)` を定義せよ。例えば a が [1,2,3,4] のとき sum(a,3) は 3 と 4 の和である 7 と答える。

c) 配列 a の i 番目と j 番目の要素を入れ替える関数 `swap(a,i,j)` を作れ。

d) 配列 a の i 番目と i+1 番目の数値の大きさを比べ、前者が後者より大きいときに両者を入れ替える関数 `swap_ascending(a,i)` を作れ。

解答例 3.10

```
#)a
def length3(a,x)
  l = a.length()
  if l==1 && x==0 then return 1
  elsif 0<x && x<l-1 then return 3
  elsif l>1 && (x==0 || x==l-1) then return 2
  else return 0
  end
end

#b)
#for を使って回したり、(1,1) 配列をずらして周りに０を埋めてまた元に戻すとかいろいろ工夫の仕様はありますが、ややこしくなるので素直に書いてます。
def image_average(image,x,y)
  scaleX = image[0].length()-1
  scaleY = image.length()-1
  if scaleX==0&&scaleY==0
    image[0][0]
  elsif scaleX==0
    a = Array.new(image.length())
    for i in 0..scaleY
      a[i]=image[i][0]
    end
    array_average(a,y)
  elsif scaleY==0
    array_average(image[0],x)
  elsif x==0&&y==0
    (image[0][0]+image[0][1]+image[1][0]+image[1][1])/4.0
  elsif x==0&&y==scaleY
    (image[scaleY][0]+image[scaleY][1]+image[scaleY-1][0]+image[scaleY-1][1])/4.0
  elsif x==scaleX&&y==0
    (image[0][scaleX]+image[0][scaleX-1]+image[1][scaleX]+image[1][scaleX-1])/4.0
  elsif x==scaleX&&y==scaleY
    (image[scaleY][scaleX]+image[scaleY][scaleX-1]
     +image[scaleY-1][scaleX]+image[scaleY-1][scaleX-1])/4.0
  elsif y==0
    (image[0][x-1]+image[0][x]+image[0][x+1]+image[1][x-1]+image[1][x]+image[1][x+1])/6.0
  elsif y==scaleY
    (image[scaleY][x-1]+image[scaleY][x]+image[scaleY][x+1]
     +image[scaleY-1][x-1]+image[scaleY-1][x]+image[scaleY-1][x+1])/6.0
  elsif x==0
    (image[y-1][0]+image[y][0]+image[y+1][0]+image[y-1][1]+image[y][1]+image[y+1][1])/6.0
```

```
  elsif x==scaleX
    (image[y-1][scaleX]+image[y][scaleX]+image[y+1][scaleX]
     +image[y-1][scaleX-1]+image[y][scaleX-1]+image[y+1][scaleX-1])/6.0
  else
    (image[x-1][y-1]+image[x][y-1]+image[x+1][y-1]+image[x-1][y]
     +image[x][y]+image[x+1][y]+image[x-1][y+1]+image[x][y+1]+image[x+1][y+1])/9.0
  end
end

def sum(a,x)
  array_average(a,x)*length3(a,x)
end

def array_average(a,x)
  if length3(a,x) == 3
    (a[x-1] + a[x] + a[x+1])/3.0
  elsif length3(a,x) == 1
    a[x]
  elsif x == 0
    (a[0]+a[1])/2.0
  elsif x == a.length()-1
    (a[a.length()-1] + a[a.length()-2])/2.0
  else
    0
  end
end
```

解答例 3.10 c)

```
def swap(a,i,j)
  ai = a[i]
  aj = a[j]
  a[i] = aj
  a[j] = ai
  a
end
```

解答例 3.10 d)

```
def swap_ascending(a,i)
  if a[i] > a[i+1]
    swap(a,i,i+1)
  else
    a
  end
end
```

練習 3.11 (数列を作る) 大きさ n で i 番目の値が i/n になっているような 1 次元配列を作る関数 gradation(n) を定義せよ。(ヒント: make1d で 0 を代入している所を i を使った式に変える。整数どうしの割り算は整数商を求めてしまうことに注意せよ。)

解答例 3.11

```
def gradation(n)
  a = Array.new(n)
  for i in 0..(n-1)
    a[i] = i/(n/1.0)
  end
  a
end
```

練習 3.12 略

練習 3.13 略

4　4章

練習　4.1 (素数の判定) 素数とは、1 と自分自身しか約数がないような数である。上で定義した関数 sod を使って 2 以上の整数 n が素数のときにのみ true, そうでないときに false となるような関数 `prime(n)` を定義せよ。(n が 1 の場合は考えなくてよい。つまり、prime(1) の答は true でも false でもよいとする。)

解答例　4.1

```
def sod(k,n)
  if n >= 2
    if k%n == 0
      sod(k,n-1) + n
    else
      sod(k,n-1)
    end
  else
    1
  end
end

def prime(n)
  if n >= 2
    if sod(n,n) == 1 + n
      true
    else
      false
    end
  else
    false
  end
end
```

練習　4.2 (組み合わせ数) n 個から k 個を選ぶ組み合わせ数 ${}_nC_k$ を求める `combination(n,k)` を定義せよ。ただし ${}_nC_k$ は、次のような関係を満たしている。

$$ {}_nC_k = \begin{cases} 0 & (if k > n) \\ 1 & (if k = 0) \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & otherwise \end{cases} $$

この関係は「n 個から k 個を選ぶ」方法は、n 個の中の最初の 1 個に注目すると「それを選び、残りの n - 1 個から k - 1 個を選ぶ」か「それを選ばず、残りの n - 1 個から k 個を選ぶ」に限られるからだと理解できる。

解答例　4.2

```
def combination(n,k)
  if n < k then return 0
  elsif n == k then return 1
  elsif k == 0 then return 1
  elsif k == 1 then return n
  else return combination(n-1,k-1)+combination(n-1,k)
  end
end
```

練習　4.3 (素数の判定の改良) 素数とは、自身を除く最大の約数が 1 であるような数だとも言える。関数 gd を使って素数の判定をする関数 `prime2(n)` を定義せよ。練習 4.1 で定義した `prime` と計算回数の違いがあるかを考えよ。

解答例 4.3

```
#the greatest divisor of k in between 1 to n
load("./divisible.rb")

def gd(k,n)
  if divisible(k,n)
    n
  else
    gd(k,n -1)
  end
end

def prime2(n)
  if gd(n,n-1) == 1
    true
  else
    false
  end
end
```

練習 4.4 (再帰と繰り返しの比較) 練習 4.2 で定義した `combination` とここで定義した `combination_loop` が同じ結果を返すことを、いくつかの n, k について確めよ。また、n, k を大きくした場合にはどちらが速いか、実際に計算させて比べてみよ。

解答例 4.4
再帰と繰り返しでは、繰り返しの方が早く処理されることが知られている。

練習 4.5 略

練習 4.6 (繰り返しによって値を集める) 繰り返しによって以下のような値を求める Ruby の関数を定義せよ。

a) n の階乗、即ち 1 から n までの積 n! を求める `factorial_loop(n)` .
b) 2^n を求める `power2_loop(n)` . 記号**を使わずに定義せよ。
c) x^n を求める `power_loop(x,n)` . 同じく記号**を使わずに定義せよ。
d) 次の式に示されるような級数を求める `taylor_e_loop(x,n)` .

$$\sum_{k=0}^{n} \frac{x^k}{k!}$$

($n \to \infty$ のとき、この式は e^x の Taylor 級数になっている。)

解答例 4.6

```
def factorial_loop(n)
  var = 1
  for i in 1..n
    var = var*i
  end
  var
end

def power2_loop(n)
  var = 1
  for i in 1 .. n
    var = var*2
  end
  var
end

def power_loop(x,n)
```

```
  var = 1
  if n == 0
    1
  else
    for i in 1..n
      var = var*x
    end
  end
  var
end

def taylor_e_loop(x,n)
  var = 0
  for i in 0..n
    var = var + (x**i*1.0)/factorial_loop(i)
  end
  var
end
```

練習 4.7(繰り返しによって条件を満たす値を集める) 繰り返しによって以下のような値を求める Ruby の関数を定義せよ。

a) 1 から n までの数のうち、k の約数になっているものの個数 (number of divisors) を求める `nod_loop(k,n)`. (ヒント: 式 4.5 の足す数を変える。)
b) 1 から n までの数のうち、素数になっているものの個数 (number of primes) を求める `nop_loop(n)`.
c) 1 から n までの数について、それぞれ自身を含むような約数の和を求めたとき、最大のもの (maximum sum of divisors) を求める `msod_loop(n)`.

解答例 4.7

```
def nod_loop(k,n)
  ount = 0
  for i in 1..n
    if k%i == 0
      count = count + 1
    end
  end
  count
end

def nop_loop(n)
  count = 0
  for i in 1..n
    if nod_loop(i,i) == 2
      count = count + 1
    end
  end
  count
end

def msod_loop(n)
  var = 0
  for i in 1..n
    if sod(i,i) >= var
      var = sod(i,i)
    end
  end
  var
end
```

練習 4.8 (繰り返しによって条件を満たす値を探す) 繰り返しによって以下のような値を求める Ruby の関数を定義せよ。

a) n より大きくかつ最小の素数、つまり次の素数 (next prime number) を求める `np_loop(n)`.
b) 素数 p よりも大きな素数のうち n 番目の素数を求める `nth_prime_loop(p,n)`.
c) 第 3.2 節の式 3.1(p. 35) で紹介した関数 tnpo(n) は n が偶数なら 1/2, 奇数なら 3 倍して 1 加えた数を求めるものだった。数学者 Collatz はどんな整数 n が与えられたときでも、この関数を使って数を変化させてゆくと、いずれ 1 になると予想した。例えば 3 から始めた場合はこの予想は、反例は見つかっていないが証明もされていない、数学の未解決問題の 1 つである。$3 \Rightarrow 10 \Rightarrow 5 \Rightarrow 16 \Rightarrow 8 \Rightarrow 4 \Rightarrow 2 \Rightarrow 1$ といった具合に予想通りになっていることが確められる。そこで n から上の手順で数を変化させて 1 になるまでの回数を collatz(n) とする。例えば $\mathrm{collatz}(5) = 5$, $\mathrm{collatz}(16) = 4$ である。collatz(n) を求める Ruby の関数 `collatz_loop(n)` を定義せよ。
d) n 以上の整数で最小の完全数を見つける `next_perfect_loop(n)`. ただし完全数 k とは k を除く約数の和が k と等しいような数のことである。例えば 6 の約数は (6 を除くと) 3, 2, 1 だけであり、その和が 6 になるので 6 は完全数である。最初に n が完全数であるかを判定する関数 `perfect_loop(n)` を定義しておくとよい。

解答例 4.8

```
def np_loop(n)
  var = n
  while prime(var) == false
    var = var + 1
  end
  var
end

def nth_prime_loop(p,n)
  count = n
  var = p
  while count != 0
    var = var+1
    if prime(var) == true
      count = count-1
    end
  end
  var
end

def tnpo(n)
  if n%2 == 0
    n/2
  else
    3*n + 1
  end
end

def collatz_loop(n)
  count = 0
  var = n
  while var != 1
    var = tnpo(var)
    count = count+1
  end
  count
end

def perfect_loop(n)
  if sod(n,n) == 2*n
    true
```

```
  else
    false
  end
end

def next_perfect_loop(n)
  var = n
  while perfect_loop(var) == false
    var = var + 1
  end
  var
end
```

練習 4.9 (再帰的に値を集める) 以下のような値を求める再帰的な Ruby の関数を定義せよ。繰り返しは使わないこと。

a) n の階乗、即ち 1 から n までの積 n! を求める `factorial(n)`.
b) 2^n を求める `power2(n)`. 記号**を使わずに定義せよ。
c) x^n を求める `power(x,n)`. 同じく記号**を使わずに定義せよ。
d) 次の式に示されるような級数を求める `taylor_e(x,n)`.

$$\sum_{k=0}^{n} \frac{x^k}{k!}$$

($n \to \infty$ のとき、この式は e^x の Taylor 級数になっている。)

解答例 4.9

```
def factorial(n)
  if n == 0 then return 1
  elsif n == 1 then return 1
  else return factorial(n-1)*n
  end
end

def power2(n)
  if n == 0 then return 1
  elsif n == 1 then return 2
  else return power2(n-1)*2
  end
end

def power(x,n)
  if n == 0
    1
  else
    power(x,n-1)*x
  end
end

def taylor_e(x,n)
  if x == 0 then return 1
  elsif n == 0 then return 1
  else return taylor_e(x,n-1) + (x**n)*1.0/factorial(n)
  end
end
```

練習 4.10 (文字列を作る再帰) ファイル 4.3 の 6 行目の数値 1 を文字列"1" に置きかえた場合、sum(n) はどのような答えを返すか?

解答例 4.10

```
# sum of the numbers from 1 to n
def sum(n)
  if n >= 2
    sum(n -1) + n
  else
    "1" #character'1'
  end
end
```

数字と文字列の足し算は型が違うためできない。厳密にいえば、型が違っていてもスーパークラスが同じであれば足し算ができるが、この場合数字は Numeric クラスであり、文字列は String クラスであるため演算不可能である。例えば、次のようなエラーが考えられる。`can't convert Fixnum into String (TypeError)`

練習 4.11 (再帰的に条件を満たす値を集める) 以下のような値を求める再帰的な Ruby の関数を定義せよ。繰り返しは使わないこと。

a) 1 から n までの数のうち、k の約数になっているものの個数 (number of divisors) を求める `nod(k,n)`. (ヒント: 式 4.5 の足す数を変える。)
b) 1 から n までの数のうち、素数になっているものの個数 (number of primes) を求める `nop(n)`.
c) 1 から n までの数について、それぞれ自身を含むような約数の和を求めたとき、最大のもの (maximum sum of divisors) を求める `msod(n)`. sod(k, k) $= s_k$ と書くことにすると、この問題は s_1 から s_n の最大値を求めることに等しい。「s_1 から s_n の最大値」は「s_1 から s_{n-1} の最大値」より s_n が大きい場合とそうでない場合に分けて考えればよいので、次のような関係を満たしている。

$$msod(n) = \begin{cases} msod(n-1) & (msod(n-1) \geq s_n) \\ s_n & (msod(n-1) < s_n) \\ s_1 & (n = 1) \end{cases}$$

解答例 4.11

```
def nod(k,n)
  if n >= 2
    if k%n == 0
      nod(k,n-1) + 1
    else
      nod(k,n-1)
    end
  else
    1
  end
end

def nop(n)
  if n >= 2
    if prime(n) == true
      nop(n-1)+1
#prime とは素数のとき true、合成数である時 false を返す関数
    else
      nop(n-1)
    end
  else
    1
  end
end

def msod(n)
```

```
  if msod(n-1) >= sod(n,n)
    msod(n-1)
  elsif msod(n-1) < sod(n,n)
    sod(n,n)
  elsif n == 1
    1
  end
end
```

練習 4.12 (再帰的に条件を満たす値を探す) 以下のような値を求める再帰的な Ruby の関数を定義せよ。繰り返しは使わないこと。

a) n より大きくかつ最小の素数、つまり次の素数 (next prime number) を求める `np(n)`. まず、n が素数だった場合は np(n)=n である。そうでない場合は、np(n) は「n+1 より大きくかつ最小の素数」に一致する。

b) 素数 p よりも大きな素数のうち n 番目の素数を求める `nth_prime(p,n)`. 例えば `nth_prime(5,3)` は 5 よりも大きな素数のうち 3 番目であり、5 の次の素数は 7 なので 7 よりも大きな素数のうち 2 番目つまり `nth_prime(7,2)` と等しい。これはさらに 7 の次の素数 11 よりも大きな素数のうち 1 番目と等しいので 11 の次の素数 13 が答になるはずである。

c) 第 3.2 節の式 3.1(p. 35) で紹介した関数 tnpo(n) は n が偶数なら 1/2, 奇数なら 3 倍して 1 加えた数を求めるものだった。数学者 Collatz はどんな整数 n が与えられたときでも、この関数を使って数を変化させてゆくと、いずれ 1 になると予想した。例えば 3 から始めた場合はこの予想は、反例は見つかっていないが証明もされていない、数学の未解決問題の 1 つである。$3 \Rightarrow 10 \Rightarrow 5 \Rightarrow 16 \Rightarrow 8 \Rightarrow 4 \Rightarrow 2 \Rightarrow 1$ といった具合に予想通りになっていることが確められる。そこで n から上の手順で数を変化させて 1 になるまでの回数を collatz(n) とする。例えば collatz(5) = 5, collatz(16) = 4 である。
   (a) collatz(n) と collatz(tnpo(n)) の関係を書け。
   (b) collatz(n) を求める Ruby の関数 collatz(n) を定義せよ。

d) n 以上の整数で最小の完全数を見つける `next_perfect(n)`. ただし完全数 k とは k を除く約数の和が k と等しいような数のことである。例えば 6 の約数は (6 を除くと) 3, 2, 1 だけであり、その和が 6 になるので 6 は完全数である。最初に n が完全数であるかを判定する関数 `perfect(n)` を定義しておくとよい。

解答例 4.12

```
def np(n)
  var = n
  if prime(n) == true
    n
  else
    np(n+1)
  end
end

def nth_prime(p,n)
  if count != 0
    prime(np(p),n-1)
  else
    p
  end
end

def tnpo(n)
  if n%2 == 0
    n/2
  else
    3*n+1
  end
end
```

```
def collatz(n)
  count = 0
  var = n
  if n == 1
    0
  elsif n%2 == 0
    var = n/2
    count = 1 + collatz(var)
  else
    var = 3*var + 1
    count = 1 + collatz(3*n+1)
  end
  count
end

def perfect(n)
  if sod(n,n) == 2*n
    true
  else
    false
  end
end


def next_perfect(n)
  if perfect(n) == true
    n
  else
    next_perfect(n+1)
  end
end
```

練習 4.13 (Eratosthenes の篩) 素数を求める方法の 1 つに Eratosthenes の篩というものがある。考え方はとしては「整数が素数かどうか」を記録する大きな表を用意して、素数を発見するたびにその倍数を表から消してゆくものである。この考え方をもとに、2 から n までの数が素数かどうかを示す配列を作る `primes(n)` を定義せよ。ただし、作られる配列の i 番目は、i が素数のとき 0, 素数でないとき 1 だとする。

解答例 4.13

```
load("./prime_loop.rb")
include Math

def primes(n)
  era = Array.new(n+1)
  var = 2
  for i in 0..n
    era[i] = 0
  end
  era[0],era[1] = 1,1
  while var <= n
    for i in (var+1)..n
      if i%var == 0
        era[i] = 1
      end
    end
    var = np_loop(var+1)
  end
  era
end
```

練習 4.14 (Sierpinski の三角形) 大きさ $n \times n$ で、i 行 j 列目が (${}_iC_j$ を 2 で割った余り) になっているような配列を作る関数 `sierpinski(n)` を定義せよ。

解答例 4.14

```
load("./make2d.rb")
load("./combination.rb")

def sierpinski(n)
  array = make2d(n,n)
  for i in 0..n-1
    for j in 0..n-1
      array[j][i] = combination(j,i)%2
    end
  end
  array
end
```

練習 4.15 (見つからない場合) ファイル 4.8 の match の定義では s 中に p が必ず現われることを仮定していた。p が現われない場合に -1 と答える `match_safe(s,p)` を定義せよ。

解答例 4.15

```
def submatch (s,i,p,w)
  j = 0
  while j < w && s[(i+j)..(i+j)] == p[j..j]
    j = j + 1
  end
  j
end

def match_safe(s,p)
  i = 0
  w = p.length()
  var = 0
  for k in 0..s.length()
    if submatch(s,k,p,w) >= var
      var = submatch(s,k,p,w)
    end
  end
  if var != w
    -1
  else
    while submatch(s,i,p,w) < w
      i = i + 1
end
    i
  end
end
```

5 5 章

練習 5.1 略

練習 5.2 略

練習 5.3 略

練習 5.4 (代入の順序) ファイル 5.2 の、12 行目と 13 行目を入れ替えた場合、どのような値が計算されるかその理由とともに説明せよ。

解答例 5.4 ファイル 5.2 の、１２行目と１３行目を入れ替えると次のようなプログラムになる。

```
def fibl (k)
  f=1
  p1 =1
  for i in 2..k
    p1 = f #fib(i -1)
    p2 = p1 #fib(i -2)
    f = p1 + p2 #fib(i)
  end
  f #fib(k)
end
```

この場合、2 の冪が計算されることになる。

練習 5.5 略

練習 5.6 略

練習 5.7 (行列の冪乗) 行列 A の冪乗 A^n を計算する関数 `matpower(a,n)` を次のようにして定義せよ。行列は 2×2 のものだけを扱うことにして、2 次元配列で表わすことにする。

a) 行列 a, b の積を求める `matmul(a,b)` を定義せよ。ヒント: 行列を 2×2 に限っているので次のような計算をするだけである。

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \end{pmatrix}$$

b) 行列 a の自乗を求める `matsquare(a)` を `matmul` を用いて定義せよ。

c) 式 5.11 に従って行列 a の n 乗を求める `matpower(a,n)` を定義せよ。

解答例 5.7

```
#a)
def matmul(a,b)
ans = make2d(2,2)
ans[0][0] = a[0][0]*b[0][0]+a[0][1]*b[1][0]
ans[0][1] = a[0][0]*b[0][1]+a[0][1]*b[1][1]
ans[1][0] = a[1][0]*b[0][0]+a[1][1]*b[1][0]
ans[1][1] = a[1][0]*b[0][1]+a[1][1]*b[1][1]
return ans
end

#b)
def matsquare(a)
return matmul(a,a)
end

#c)
def matpower(a,n)
if n == 0 then return [[1,0],[0,1]]
elsif n%2 == 1 then return matmul(a,matpower(a,n-1))
elsif n%2 == 0 then return matsquare(matpower(a,n/2))
end
end
```

練習 5.8 (行列を用いた Fibonacci 数の計算) 練習 5.7 で定義した `matpower(a,n)` を用いて `fib(k)` を求める関数 `fibm(k)` を定義せよ。また、`fibm(k)` と `fibl(k)` が同じ答を出すことをいくつかの k について確認せよ。

解答例 5.8

```
def fibm(k)
  q = make2d(2,2)
  a=[[1,1],[1,0]]
  q = matpower(a,k)
  q[1][0]+q[1][1]
end
```

練習 5.9 略

練習 5.10 (単純整列法) 配列 a の (先頭を 0 番目としたときの)i 番目以降の値の中で、最小値が出現する番号を答える `min_index(a,i)` をファイル 5.3 に追加して、単純整列法を完成させよ。

解答例 5.10

```
def min_index(a,i)
  min = a[i]
  min_num = i
  for k in (i+1) .. (a.length-1)
    if a[k] <= min
      min = a[k]
      min_num = k
    end
  end
  min_num
end

def simplesort (a)
  for i in 0..( a. length() -1)
    k = min_index(a,i)
    v = a[i]
    a[i] = a[k]
    a[k] = v
    end
  a
end
```

練習 5.11 (併合) ファイル 5.4 の省略された部分を補って関数 `merge` を完成させよ。

解答例 5.11

```
def merge(a,b)
  c = Array.new(a.length()+b.length())
  ia = 0
  ib = 0
  ic = 0
  while ia < a.length() && ib < b.length()
    if a[ia] < b[ib]
      c[ic] = a[ia]
      ia = ia + 1
      ic = ic + 1
    else
      c[ic] = b[ib]
      ib = ib + 1
      ic = ic + 1
    end
  end
  if a.length()-ia >= 1
    for k in (ia+ib) .. (c.length()-1)
      c[k] = a[k-ib]
    end
  else
```

```
    for k in (ia+ib) .. (c.length()-1)
      c[k] = b[k-ia]
    end
  end
  c
end
```

練習 5.12 略

練習 5.13 略

練習 5.14 (組み合わせ数の計算量)

1. 組み合わせ数の計算を練習 4.2(p. 53) の定義に従って再帰的に行う場合の計算量を求めよ。
2. 同じ計算を第 4.3.1 節のように繰り返しによって行う場合の計算量を求めよ。

練習 5.15 (言葉探しの計算量) 文字列 s 中に文字列 p が現われる位置を探す方法として、第 4.3.2 節 (p. 56) で紹介したものがある。s と p の長さ n,m を入力の大きさだとして、これの計算量を n,m で表わせ。

ヒント: 最悪の場合を考えればよい。s が「$\overbrace{aa\cdots a}^{n-1}$ b」、p が「$\overbrace{aa\cdots a}^{m-1}$ b」という文字列だったときの submatch と match の繰り返し回数を考えよ。

練習 5.16 (同じ誕生日の人達) 同じ年に生まれた n の学生がいる。この中には、同じ誕生日を持つ学生と、そうでない学生がいる。前者が何人いるか数えるアルゴリズムをいくつか考え、その計算量を考えよ。簡単のために、誕生日は 1 月 1 日からの日数 (つまり 0 から 364 の整数) で表わすことにする。また、学生の誕生日は大きさ n の配列 b に入っているが、その順序はデタラメだとする。

アルゴリズム 1: 同じ誕生日を持つ学生の人数を変数 m で表わすことにして、最初は 0 にしておく。学生の誕生日を 1 人ずつとり出し、配列 b のすべての中身と比較して同じ誕生日の学生がいるかどうかを調べ、1 人でも同じ誕生日の者がいた場合は m を 1 増やす。これをすべての学生について行う。

アルゴリズム 2: 同じ誕生日を持つ学生の人数を変数 m で表わすことにして、最初は 0 にしておく。配列 b を併合整列法によって小さい順に並べる。並べ換えた配列を先頭から順に調べてゆき、同じ値が連続する区間の長さ k を数える。k が 2 以上の場合の区間があった場合は、m を k だけ増やす。これを配列の最後まで行う。

アルゴリズム 3: 日付ごとの人数を数えるために、大きさ 365 の配列 c を作り、中身をすべて 0 にしておく。学生の誕生日 d を 1 人ずつとり出し、その日付の人数 c[d] を 1 増やす。これをすべての学生について行う。最後に、c を先頭から調べて 2 以上になっている値の和を求める。

解答例 5.16

```
def birthday1(b)
  n = b.length()
  m = 0
  for i in 0 .. (n-1)
    temp = 0
    for k in 0 .. (n-1)
      if i==k
      elsif b[i]==b[k]
        temp = temp + 1
      end
    end
    if temp >=1 then m = m+1 end
  end
  m
end
```

```
def birthday2(b)
  m=0
  b = mergesort(b)
  index = 0
  for i in 0 .. (b.length()-1)
    if b[i] == b[i+1] || b[i] == b[i-1]
      m = m+1
    end
  end
  m
end

def birthday3(b)
  n = b.length()
  c = Array.new(365)
  for i in 0 .. 364
    c[i]=0
  end
  for i in 0 .. (n-1)
    c[b[i]] = c[b[i]] + 1
  end
  sum = 0
  for i in 0 .. 364
    if c[i]>=2 then sum = sum + c[i] end
  end
  sum
end
```

練習 5.17 (再帰的な併合整列) 併合整列を再帰的に行う方法を紹介する。これに従って配列 a を整列する `mergesort_r(a)` をファイル `mergesort_r.rb` に定義せよ。まず、配列 a を併合整列する最後の段階を考える。この段階では、配列 a の前半部分を整列した配列 p と後半部分を整列した配列 q が得られていて、その 2 つを関数 merge によって併合すればよい。配列 a の前半部分や、後半部分だけを整列した配列を作るには、再帰的に併合整列を行えばよい。ただし、配列の一部だけを整列する必要があるため「配列 a の l 番目から r 番目までを整列する」ような関数を定義する必要がある。そこで、配列 a の l 番目から r 番目までを整列した配列を作る関数を `merge_rec(a,l,r)` として、この関数をどのように定義すればよいか考えてみよう。

- l = r のときは、整列の必要はないので a[l] の同じ値を持った大きさ 1 の配列を作り、それを答える。
- l < r のときは、l から r までを 2 つに分割して再帰的に併合整列を行い、その結果を merge する。つまり、
  1. m = (l + r)/2 として、
  2. l から m までと、m+1 から r までをそれぞれ `merge_rec` を使って併合整列する (再帰)。(それらの結果を b, c とする。)
  3. b と c を merge によって併合した結果が併合整列の結果となる。

`merge_rec` が完成すれば、`mergesort_r(a)` は `merge_rec(a,0,a.length()-1)` を呼び出すだけでよい。

解答例 5.17

```
def merge_rec(a,l,r)
  if l==r
    ary = Array.new(1)
    ary[0] = a[l]
    ary
  else
    m = (l+r)/2
    b = Array.new(m-l+1)
    c = Array.new(r-m)
    b = merge_rec(a,l,m)
    c = merge_rec(a,m+1,r)
    merge(b,c)
```

```
  end
end

def mergesort_r(a)
  if a == nil then return nil
  else
    merge_rec(a,0,a.length()-1)
  end
end
```

練習 5.18 (ビン配列) 新たな整列のアルゴリズムとして、ビン整列法を紹介する。これに従って値の範囲が 0 以上 max 未満の整数からなる配列 a を整列する `binsort(a,max)` をファイル binsort.rb に定義せよ。ビン整列は、値の表われる頻度を求めることで整列を行う。例えば [3, 1, 4, 1, 5] という配列が与えられたとき、0 から 5 までの数が表われた回数は [0, 2, 0, 1, 1, 1] となる。値の頻度が分かれば、小さい値から順に、その頻度だけ数を並べてゆけば整列された配列が得られる。

1. max 未満の値からなる配列 a の頻度を求める関数 `count(a,max)` を定義せよ。この関数は大きさ max の配列を作り、配列の (先頭を 0 とした)i 番目は数 i が a の中に現われた回数となる。
2. 頻度を表わす配列 c から整列された配列を作る関数 `rebuild(c,n)` を定義せよ。ただし n は頻度の合計値だとする。(ヒント: まず大きさ n の配列を作り、0 を c[0] 個、1 を c[1] 個、2 を c[2] 個、のように配列の先頭から埋めてゆく。)
3. count と rebuild を使ってビン整列を行う関数 `binsort(a,max)` を定義せよ。ただし a は 0 以上 max 未満の整数からなる配列である。

解答例 5.18

```
def count(a,max)
  ary = Array.new(max)
  for i in 0 .. (max-1)
    count = 0
    for k in 0 .. (a.length()-1)
      if a[k]==i
        count = count+1
      end
    end
    ary[i]=count
  end
  ary
end

def rebuild(c,n)
  ary = Array.new(n)
  index = 0
  for i in 0 .. (c.length()-1)
    if 0 < c[i]
      for k in 0 .. (c[i]-1)
        ary[index] = i
        index = index+1
      end
    end
  end
  ary
end

def binsort(a,max)
  ary = Array.new(a.length()-1)
  ary1 = Array.new(max)
  ary1 = count(a,max)
  rebuild(ary1,a.length())
end
```

6　6章

練習　6.1 略

練習　6.2 (Simpson 公式による積分)

a)

$$\int_{x_s}^{x_e} f(x)dx \simeq \sum_{i=0}^{n-1} \frac{1}{3}\left\{f(x_s+2i\Delta x)+4f(x_s+(2i+1)\Delta x)+f(x_s+(2i+2)\Delta x)\right\}\Delta x$$
$$= \frac{\Delta x}{3}\left\{f(x_s)+4f(x_s+\Delta x)+f(x_e)+\sum_{i=1}^{n-1}(2f(x_s+2i\Delta x)+4f(x_s+(2i+1)\Delta x))\right\}$$

に従って積分を行う関数 `simpson(xs,xe,n)` を定義せよ。(注意: 台形公式の場合と違って Δx が積分範囲の 1/2n になっている。)

b) 練習 6.1 と同様に `simpson(xs,xe,n)` の解析的な積分値との違いを調べよ。

解答例　6.2

```
#6.2 Function f has already defined.
def f(x)
  return x/((x+1)*(x+2)*1.0)
end

def simpson(xs,xe,n)
  sum = 0
  dx = (xe-xs)/(2*n*1.0)
  for i in 1..(n-1)
    sum = sum + 2*f(xs+2*i*dx) + 4*f(xs+(2*i+1)*dx)
  end
  return dx*(f(xs)+4*f(xs+dx)+f(xe)+sum)/3.0
end
```

練習　6.3 略

練習　6.4 (乱数列となる条件) 次のような数列は乱数列と言えるかどうか考えよ。

a) ある地点の毎日の最高気温を日付順に並べたもの

b) 毎年の年末ジャンボ宝くじの当籤番号を年ごとに並べたもの (抽籤方法は数字が書かれた回転する円盤を矢で射て決めており、数字の範囲と当籤本数は毎年同じだとする)

c) 52 枚のカードを充分にシャッフルした後で一列に並べ、各カードの数字を取り出したもの

解答例　6.4

a)　ならない。　暑いときは暑い。　寒い時は寒い。

b)　なる。　予測できれば今頃億万長者。

c)　ならない。　一見充分な乱数列に見えるが回を追うごとに確率分布が偏るため、乱数列とは言えない。具体的にいえば、51 枚見た後に 52 枚目を予測できる。

練習　6.5　(浮動小数点数の表現)

a) 次のビット列は IEEE 754 規格の単精度浮動小数点数である。どのような値を表わしているかを求めて 10 進数で書け。$\overset{s}{1}\overbrace{11111110}^{d'_1,\ldots d'_8}\overbrace{10000000000000000000000}^{d_1\ldots d_{23}}$

b) 次の数を IEEE 754 規格の単精度浮動小数点数で表わすときの $s, d'_1, \ldots d'_s, d_1 \ldots d_{23}$ をビット列で書け。

(a)　1.0

(b) $3145728_{(10)}$

(c) $5/65536_{(10)} (\simeq 7.62939453125 \times 10^{-5})$

解答例 6.5

a) 符号、指数、仮数に注意して書き下すと

$$(-1)^1 \times 1.11000000000000000000000_{(2)} \times 2^{11111110_{(2)}-127}_{(10)}$$

となる。すなわち、$1.75 \times 2^{127} = 2.894802 \times 10^{38}$

b) (a) $1.0 = +1.00000000000000000000000 \times 2^{01111111-127}$より 00111111100000000000000000000000

(b) $3145728 = +1.10000000000000000000000 \times 2^{10010100-127}$より 01001010010000000000000000000000

(c) $5/65536 = 5 * 2^{-16} = +1.01000000000000000000000 \times 2^{01110001-127}$より 00111000101000000000000000000000

なんか期末に出そうな気がしますねぇ。詳しい導出は解説にて。

練習 6.6 略

補足：大小関係保存は広義ですね。

練習 6.7 略

練習＆解答例 6.8 (倍精度表現の精度) IEEE 754 規格の単精度表現の有効桁数と表現できる値の範囲は下のようにまとめられる。これにならって倍精度の場合の有効数と表現できる値の範囲を求めよ。

	単精度	倍精度
仮数部のビット数	23	52
10 進有効桁数 (約)	$log_{10}(2^{23+1}) \simeq 7$	$log_{10}(2^{52+1}) \simeq 16$
指数部のビット数	8	11
最小の指標	$2^{-126} \simeq 1.2 \times 10^{-38}$	$2^{-1022} \simeq 2.2 \times 10^{-308}$
最大の指標	$2^{127} \simeq 1.7 \times 10^{38}$	$2^{1023} \simeq 9.0 \times 10^{307}$
最大値	$2 \times 1.7 \times 10^{38} \simeq 3.4 \times 10^{38}$	$2 \times 9.0 \times 10^{307} \simeq 1.8 \times 10^{308}$

練習 6.9 (Pivoting の完成) 関数 `maxrow(a,k)` の定義を追加し、pivoting 付き Gauss-Jordan 法を完成させよ。さらに前出の連立 1 次方程式が、より小さな誤差で解けることを確認せよ。

解答例 6.9

```
load("./abs.rb")
#load("./swap.rb") #exercise 3.10, 配列 a の i 番目と j 番目の要素を入れ替える関数
def swap(a,i,j)
  ai = a[i]
  aj = a[j]
  a[i] = aj
  a[j] = ai
  a
end

def maxrow(a,k)
  maxValue=0
  max=0
  for i in k..a.length()-1
    if abs(a[i][k]) >= maxValue
      maxValue = abs(a[i][k])
      max = i
    end
  end
  max
end

def gjp(a) # Gauss-Jordan method WITH pivoting
  row = a.length()
  col = a[0].length()
```

```
  for k in 0..(col-2)
    max=maxrow(a,k) # find absolute-maximal coeff.
    swap(a,k,max)   # swap rows
    akk = a[k][k]
    for i in 0..(col-1)       # normalize row k
      a[k][i]=a[k][i]*1.0/akk
    end
    for i in 0..(row-1)       # eliminate column k
      if i != k       # of all rows but k
        aik = a[i][k]
        for j in k..(col-1)
          a[i][j] = a[i][j] - aik * a[k][j]
        end
      end
    end
  end
  a
end
```

練習 6.10 (部分の通過距離)

$$p(\theta,d,u,v)=\begin{cases}\sqrt{1-4q^2} & (q<\frac{1}{2})\\ 0 & (q\geq\frac{1}{2})\end{cases}$$

$$\text{ただし } q=\mid x\cos\theta+y\sin\theta-d\mid$$

に従って $p(\theta,d,x,y)$ を求める関数 `penetration(theta,d,x,y)` を定義せよ。

解答例 6.10

```
include Math
def penetration(theta,d,x,y)
  q = abs(x*cos(theta)+y*sin(theta)-d)
  if q < 1/2
    sqrt(1-4*(q**2))
  else
    0
  end
end
```

練習 6.11 (係数行列の作成) 画像の 1 辺の長さが l, スキャンの回数が n のときの係数行列 M を作成する関数 `coefficients(l,n)` を定義せよ。

解答例 6.11 これはむづいww　てかもっと式の定義書けよwww

```
class Num
  attr_accessor("quotient","remainder")

  def initialize(var,l)
    self.quotient = var/l
    self.remainder = var%l
  end

end

def coefficients(l,n)
  m = make2d(l*n,l**2)
  for i in 0..m.length-1
    for j in 0..m[i].length-1
      numI = Num.new(i,l)
      numJ = Num.new(j,l)
      m[i][j]=penetration(numI.quotient*PI*1.0/n,numI.remainder,numJ.remainder,numJ.quotient) #penet
    end
```

```
  end
  m
end
```

練習 6.12 (係数行列と定数ベクトルの合成) 係数行列 M と、測定値ベクトルを表わす配列 s から連立 1 次方程式を表わす配列を作る `make_equations(l,n,m,s)` を定義せよ。ただし l は 1 辺の長さ、n はスキャンの回数、m は $l \times n$ 行 l^2列 の配列、s は n 行 l 列の配列である。作られる配列は $l \times n$ 行 l^2+1 列 の配列で、右端の列以外は m と同じ内容で、右端の列に s の内容を縦に並べた値が入る。

解答例 6.12

```
def make_equations(l,n,m,s)
  equations = make2d(l*n,l**2+1)
  for i in 0..m.length()-1
    for j in 0..m[i].length()
      if j == m[i].length()
        numI = Num.new(i,l)
        equations[i][j] = s[numI.quotient][numI.remainder]
      else
        equations[i][j] = m[i][j]
      end
    end
  end
  equations
end
```

練習 6.13 (画像の復元) 測定値を表わす配列 s から 2 次元の画像を作成する関数 `reconstruct(s)` は次のように定義できる。この中で使われている `equations_to_image(l,solved)` は、配列 solved の (左端を 0 列目として)l^2 列目の上から l^2 個を $l \times l$ に並べ直した 2 次元配列を作るような関数である。この関数を定義して、`reconstruct` を完成させよ。配布プログラム scans.rb に用意された関数 `scan1(),scan2(),scan3(),scan4()` が与える測定値から画像を作成せよ。(`show(reconstruct(scan1()))` のように実行すればよい。)

解答例 6.13

```
def equations_to_image(l,solved)
  m = make2d(l,l)
  for i in 0..m.length()-1
    for j in 0..m[i].length()-1
      m[i][j] = solved[l*i+j][l**2]
    end
  end
  m
end

def reconstruct (s)
  n = s.length()
  l = s[0].length()
  m = coefficients(l,n)
  equations = make_equations(l,n,m,s)
  solved = gjp(equations)
  equations_to_image(l,solved)
end
```

練習 6.14 (解析的な積分が難しい関数の数値積分) 関数 $g(x) = \dfrac{sinx}{logx}$ についての積分 $\displaystyle\int_{x_s}^{x_e} g(x)dx$(ただし $x_e < 1$) について

a) 台形公式によって数値積分を求める関数 `trapezoid_sinlog(xs,xe,n)` を定義せよ。ただし n は分割の数とする。

b) Simpson 公式によって数値積分を求める関数 `simpson_sinlog(xs,xe,n)` を定義せよ。ただし n は分

割の数とする。

解答例 6.14

```
#分母が小さくなる場合があるので、情報落ち誤差が起こる可能性がありますね。
include Math
def g(x)
  sin(x)/log(x)
end

def trapezoid_sinlog(xs,xe,n)
  deltax = (xe-xs)*1.0/n
  sum = (g(xs)+g(xe))/2.0
  for i in 1..n-1
    sum = sum + g(xs+i*deltax)
  end
  deltax * sum
end


def simpson_sinlog(xs,xe,n)
  sum = 0
  dx = (xe-xs)/(2*n*1.0)
  for i in 1..(n-1)
    sum = sum + 2*g(xs+2*i*dx) + 4*g(xs+(2*i+1)*dx)
  end
  return dx*(g(xs)+4*g(xs+dx)+g(xe)+sum)/3.0
end
```

練習 6.15 (Monte Carlo 法による 3 次元の積分) 半径 1 の球の体積を Monte Carlo 法によって求める関数 `montecarlo3d(n)` を定義せよ。ただし n は点の個数だとする。半径 r の球の体積は $\frac{4}{3}\pi r^3$ であることが知られているので、これとの誤差を調べよ。

解答例 6.15 これ、チェックプログラムが間違えてますね。直径 1 と勘違いしてるっぽいです。

```
def montecarlo3d(n)
  counter=0
  for i in 0..n
    x=rand()*2
    y=rand()*2
    z=rand()*2
    if sqrt((x-1)**2+(y-1)**2+(z-1)**2) < 1
      counter = counter + 1
    end
  end
  counter*8.0/n
end
```

練習 6.16 (Monte Carlo 法による積分) 関数 $f(x) = \dfrac{x}{(x+1)(x+2)}$ を Monte Carlo 法によって数値積分する関数 `montecarlo_f(xs,xe,n)` を定義せよ。ただし、(xs, xe) は $0 < xs$ であるような積分区間、n は点の個数だとする。ヒント: 区間 (xs, xe) において $f(x)$ が内側に収まるような長方形の領域を求め、その内側に点を打てばよい。$x > 0$ の範囲で $0 < f(x) < 3 - 2\sqrt{2}$ である。

解答例 6.16

```
def f(x)
  x*1.0/((x+1)*(x+2))
end

def montecarlo_f(xs,xe,n)
  counter=0
  for i in 0..n-1
```

```
    x = xs+(xe-xs)*rand()
    y = rand()*(3-2*sqrt(2))
    if y < f(x)
      counter = counter +1
    end
  end
  counter*(xe-xs)*(3-2*sqrt(2))/n
end
```

7　7章

練習　7.1 略

練習　7.2 略

練習　7.3 略

練習　7.4 (アラインメント表の完成) ファイル 7.2 の 14 行目には、a[i][j-1], a[i-1][j-1], a[i-1][j] を用いて a[i][j] を求める命令が入る。ここを埋めて関数 align を完成させよ。(ヒント:ファイル 7.1 の 19 行目から 22 行目)

解答例　7.4

```
def align(s,t)
  m = s.length()
  n = t.length()
  a = make2d(m+1,n+1)
  a[0][0] = 0
  for j in 1..n
    a[0][j] = a [0][j-1] + g()
  end
  for i in 1..m
    a[i][0] = a[i-1][0] + g()
  end
  for i in 1..m
    for j in 1..n
      #a[i][j] を計算する
      v0 = a[i-1][j-1] + q(s[i-1],t[j-1])
      v1 = a[i-1][j] + g()
      v2 = a[i][j-1] + g()
      a[i][j] = max3(v0,v1,v2)
    end
  end
  a
end
```

練習　7.5 (アラインメントを求めるプログラムの完成) ファイル 7.3 の 32 行目、35 行目を埋めてアラインメントを求めるプログラムを完成させよ。

```
def traceback (a,s,t)
  u = ""
  v = ""
  i = s.length()
  j = t.length()
  while i>0 || j>0
    if j>0 && a[i][j] == a[i][j-1] + g()
      u = "-" + u
      v = t[j-1 .. j-1] + v
```

```
      j = j - 1 # go left
    else
      if i >0 && j >0 && a[i][j] == a[i-1][j-1] + q(s[i-1] , t[j-1])
      #左上から求められた場合
      u = s[i-1 .. i-1] + u
      v = t[j-1 .. j-1] + v
      j = j-1
      i = i-1
  else
    if i >0 && a[i][j] == a[i -1][ j] + g()
          #上から求められた場合
          v = "-" + v
          u = s[i-1 .. i-1] + u
          i = i-1
        end
      end
    end
  end
  [u,v]
end
```

練習 7.6 略

練習 7.7 (スペルチェッカ) 辞書にある単語を並べた辞列 dict と、誤りのある単語 word が与えられたときに、word に最も似ている単語を答える `spell(dict,word)` を定義せよ。ただしここでの「最も似ている」とは、アラインメントの得点が最も高いものとする。

```
load ("./align.rb")

def spell(dict,word)
#ここでは採用しませんでしたが、一個当たりの align_rec の計算量が半端ない時は、一旦すべての dict で
align_rec をもとめてからその最大値をとる、といった方が早く計算が終わりますね。
  alignRec = align_rec(dict[0],word)
  counter = 0
  for i in 1..dict.length()-1
    if align_rec(dict[i],word) > alignRec
      counter = i
      alignRec = align_rec(dict[i],word)
    end
  end
  dict[counter]
end
```

8 8章

メソッドを追加せよ、という問題が多いので、あらかじめソースを置いておきます。適宜参照しながら進んでください。

```
include Math
class Point
  attr_accessor("x","y")

  def initialize(u,v)
    self.x = u
    self.y = v
  end

  def scale(s)
    Point.new(self.x * s, self.y * s)
  end
```

```
  def add(q)
    Point.new(self.x + q.x, self.y + q.y)
  end

  def sub(q)
    Point.new(self.x - q.x, self.y - q.y)
  end

  def draw(a)
    if 0 <= self.y+0.5 && self.y+0.5 < a.length() && 0 <= self.x+0.5 && self.x+0.5 < a[0].length
      a[self.y+0.5][self.x+0.5] = 1
    end
  end

  def interpolate(q,t)
     b = self.scale(t)
     c = q.scale(1-t)
     b.add(c)
  end

  def rotate(theta)
    Point.new(self.x * cos(theta) + self.y * sin(theta), self.x * sin(theta) + self.y * cos(theta))
  end

  def move(p)
    self.add(p)
  end

  def turn_at(p,theta)
    q = self.move(p.scale(-1))
    c = q.rotate(theta)
    c.move(p)
  end
end

class Line
  attr_accessor("p0","p1")

  def initialize(q,r)
    self.p0 = q
    self.p1 = r
  end

  def draw(a)
    n = max(abs(self.p1.x - self.p0.x),abs(self.p1.y - self.p0.y))
    for i in 0..n
      p = self.p0.interpolate(self.p1, i * 1.0 / n)
      p.draw(a)
    end
  end

  def turn(theta)
    self.p0 = self.p0.rotate(theta)
    self.p1 = self.p1.rotate(theta)
  end

  def move(p)
    self.p0 = self.p0.move(p)
    self.p1 = self.p1.move(p)
  end

  def zoom(s)
    self.p0.scale(s)
```

```
    self.p1.scale(s)
  end
end

class Bezier < Line
  attr_accessor("p0", "c", "p1")

  def initialize(q,r,s)
    super(q,s)
    self.c = r
  end

  def draw(a)
    n = 20
    prev = self.p1
    for i in 1..n
      t = i * 1.0 / n
      q0 = self.p0.interpolate(self.c, t)
      q1 = self.c.interpolate(self.p1, t)
      r  = q0.interpolate(q1,t)
      Line.new(prev, r).draw(a)
      prev = Point.new(r.x,r.y)
    end
  end

  def turn(theta)
    super(theta)
    self.c = self.c.rotate(theta)
  end

  def move(p)
    super(p)
    self.c = self.c.move(p)
  end

  def zoom(s)
    super(s)
    self.c.scale(s)
  end
end

class Circle
  attr_accessor("center","r")

  def initialize(p,r)
    self.center = p
    self.r = r
  end

  def turn(theta)
    self.center = self.center.rotate(theta)
  end

  def draw(a)
    n = 20
    q = Point.new(self.center.x + self.r, self.center.y)
    prev = q
    for i in 1..n
      r = q.turn_at(self.center,PI * i * 2.0 / n)
      Line.new(prev,r).draw(a)
      prev = Point.new(r.x,r.y)
    end
  end
```

```
  def move(p)
    self.center = self.center.move(p)
  end

  def zoom(s)
    self.center.scale(s)
    self.r = self.r * s
  end
end
```

練習 8.1 略

練習 8.2 (3 次曲線) 3 次の $Bezier$ 曲線は、4 点 $p0, c0, c1, p1$ が与えられたときに以下のような式によって定められる $s(t)$ の軌跡である。

$$q_0(t) = (1-t)p_0 + tc_0,\ q_1(t) = (1-t)c_0 + tc_1,\ q_2(t) = (1-t)c_1 + tp_1$$
$$r_0(t) = (1-t)q_0 + tq_1,\ r_1(t) = (1-t)q_1 + tq_2$$
$$s(t) = (1-t)r_0 + tr_1$$

a) この式をもとに、3 次の $Bezier$ 曲線を描く関数 `bezier_draw3(p0,c0,c1,p1,a)` を定義せよ。
b) 次の点列を使って一続きの曲線を描く関数 `kana3()` を定義せよ。

(323, 25), (210, 553), (-1, 361), (0, 242), (-1,-71), (242,-24),
(269, 234), (307, 448), (390, 347), (399, 302)

(練習 8.1 の図形は 5 本の曲線から成っていたのに対して、この図形は 3 本の曲線である。) ただし先頭を 0 番目として、$3i, 3i+1, 3i+2, 3i+3$ 番目の点を使って 1 つの $Bezier$ 曲線を描くものとする。

c) この関数を参考にして、自分でデザインした図形を描く関数 `mypicture3()` を定義せよ。

解答例 8.2 a), b)

```
#a)
def bezier_draw3(p0,c0,c1,p1,a)
  n = 10
  prev = p0
  for i in 1..n
    t = i * 1.0 / n
    q0 = point_interpolate(p0,c0,t)
    q1 = point_interpolate(c0,c1,t)
    q2 = point_interpolate(c1,p1,t)
    r0 = point_interpolate(q0,q1,t)
    r1 = point_interpolate(q1,q2,t)
    s  = point_interpolate(r0,r1,t)
    line_draw(prev,s,a)
    prev = s
  end
end

#b)
def kana3()
  a = make_points([323,25, 210,553, -1,361, 0,242, -1,-71, 242,-24, 269,234, 307,448, 390,347, 399,3
  b = make2d(400,400)
  for i in 0..2
    bezier_draw3(a[i*3],a[i*3 + 1],a[i*3 + 2],a[i*3 + 3],b)
  end
  show(b)
end
```

#c) **省略**

練習 8.3 (レコードを使わない曲線) $p_0 = (x0, y0), c = (xc, yc), p_1 = (x1, y1)$ によって定義される 2 次の *Bezier* 曲線を描く関数 `bezier_nr(a,x0,y0,xc,yc,x1,y1)` をレコードを使わずに定義せよ。ファイル 8.4 と比べて定義の分かりやすさは違うだろうか? (注意: 大きさ 2 の配列に X 座標と Y 座標をしまうことで 1 つの点を表わす方法もあるが、このようなやり方もしないものとする。これは配列によってレコードを表わしていることに相当する。)

解答例 8.3

```
def bezier_nr(a,x0,y0,xc,yc,x1,y1)
  n = 10
  prev_x = x0
  prev_y = y0
  for i in 1..n
    t = i * 1.0 / n
    q0_x = interpolate(x0,xc,t)
    q0_y = interpolate(y0,yc,t)
    q1_x = interpolate(xc,x1,t)
    q1_y = interpolate(yc,y1,t)
    r_x = interpolate(q0_x,q1_x,t)
    r_y = interpolate(q0_y,q1_y,t)
    line_draw_nr(prev_x,prev_y,r_x,r_y,a)
    prev_x = r_x
    prev_y = r_y
  end
end
```

練習 8.4 (円) 次の手順に従って円を描く関数を定義せよ。

a) 点 p を原点中心に角度 theta だけ回転させた座標を求める関数 `rotate(p,theta)` を定義せよ。ただし座標 (x, y) を原点まわりに θ だけ回転した座標は以下のように求められる。

$$(xcos\theta - ysin\theta, xsin\theta + ycos\theta)$$

b) 点 p を中心に半径 r の円を、2 次元配列 a に描くような関数 `circle_draw(p,r,a)` を定義せよ。(ヒント: 曲線を描くときと同じように、何本かの直線で近似すればよい。例えば、N 本の直線で近似させるのであれば、i 番目の点は点 (r, 0) を原点中心に $2\pi i/N$ だけ回転させ、それに p を加えることで決められる。)

解答例 8.4

```
#(a)
def rotate(p,theta)
point_make(p.x*cos(theta) - p.y*sin(theta),p.x*sin(theta) + p.y*cos(theta))
end

#(b)
def circle_draw(p,r,a)
n = 20
q = Point.new(p.x+r,p.y)
prev = q
for i in 1..n
s = rotate_at(prev,p,PI * 2.0 / n)
line_draw(prev,s,a)
prev = s
end
end

def rotate_at(p,q,theta)
r = Point.new(p.x,p.y)
r=r.add(q.scale(-1))
r=rotate(r,theta)
r=r.add(q)
r
```

```
end
```

練習 8.5 (点オブジェクトのメソッド) Point クラスの定義 (ファイル 8.6) に、以下の操作を行うメソッド定義を追加せよ。

a) 点 q との差を求める `sub(q)`
b) 2 次配列 a 上に点を打つ `draw(a)`
c) 点 q と (1 - t) : t の内分点を求める `interpolate(q,t)`
d) 原点中心に角度 theta だけ回転した位置を求める `rotate(theta)` ただし座標 (x, y) を原点まわりに θ だけ回転した座標は以下のように求められる。

$$(xcos\theta - ysin\theta, xsin\theta + ycos\theta)$$

練習 8.6 (円クラス) 円を表すクラス Circle を定義し、次のようなメソッドを定義せよ。

- 中心を p に、半径を r とする初期化メソッド `initialize(p,r)`
- 2 次配列 a に円を描く `draw(a)`
- 原点中心に theta 回転した位置に移動する `turn`

練習 8.7 (点のまわりの回転) 点 p のまわりに図形を theta だけ回転させるメソッド `turn_at(p,theta)` を定義したい。点 p のまわりで回転させるには (1) 図形を!p だけ平行移動させ (2) 原点中心に回転させ、(3)p だけ平行移動すればよい。もし `move(p)` というメソッドで図形を p だけ平行移動できたとすると、次のようなメソッドが定義されていればよいことになる。

```
def turn_at(p, theta)
  self.move(p.scale(-1)) # -p だけ平行移動
  self.turn(theta) # 原点中心に回転
  self.move(p) # p だけ平行移動
end
```

この定義を `Line` ,`Bezier` および練習 8.6 で定義した `Circle` の 3 つのクラスに共通して使いたいので、次のように定義せよ。

a) 図形を点 p だけ平行移動させるメソッド `move(p)` を `Line`, `Bezier` および練習 8.6 で定義した `Circle` の 3 つのクラスにそれぞれ定義せよ。
b) `Figure` というクラスを定義してその中に上に示したメソッド `turn_at` の定義を書け。
c) クラス `Line` と `Circle` が `Figure` の子クラスとなるようにそれぞれのクラス定義を変更せよ。これで例えばファイル 8.9 の `turning_figure` の e ように 1 つのメソッド定義で異なる種類の図形を回転させることができるようになる。

解答例 8.7 keep

練習 8.8 (3 次曲線のクラス) 3 次の *Bezier* 曲線を表わすクラス `Bezier3` を、`Bezier` を継承して定義せよ。(3 次の *Bezier* 曲線の描き方は練習 8.2 を参照せよ。)

解答例 8.8

```
class Bezier3 < Bezier
  attr_accessor("p0","c","c0","p1")

  def initialize(p0,c,c0,p1)
    super(p0,c,p1)
    self.c0 = c0
  end

  def draw(a)
    n = 10
    prev = self.p1
    for i in 0..n
      t = i * 1.0 / n
      q0 = self.p0.interpolate(self.c,t)
```

```
      q1 = self.c.interpolate(self.c0,t)
      q2 = self.c0.interpolate(self.p1,t)
      r0 = q0.interpolate(q1,t)
      r1 = q1.interpolate(q2,t)
      s = r0.interpolate(r1,t)
      Line.new(prev,s).draw(a)
      prev = s
    end
  end

  def turn(theta)
    super(theta)
    self.c0 = self.c.rotate(theta)
  end

  def move(p)
    super(p)
    self.c0.move(p)
  end
end
```

練習 8.9 略

練習 8.10 (図形の回転) 配列 f にしまわれた全ての図形をそれぞれ原点のまわりに theta だけ回転させる `turnall(f,theta)` を定義せよ。

解答例 8.10

```
def turnall(f,theta)
  for i in 0..(f.length-1)
    f[i] = f[i].turn(theta)
  end
end
```

練習 8.11 (図形の変形) 配布プログラム oo-face.rb に定義されている関数 `face` は、円、曲線、直線がしまわれた配列を作る。その配列を `drawall` を使って 400×400 の画像に表示すると以下の左の図のようになる。関数 `face` によって作られる図形を変形して、上の中央と右のような図を以下の手順に従って作成せよ。

a) 図形を原点を中心として s 倍に拡大・縮小させるメソッド `zoom(s)` ををクラス `Line`, `Circle`, `Bezier` に定義せよ。ただし拡大・縮小の中心は原点だとする。

b) ファイル 8.12 に定義された関数 `faces`, `whirl` を使うとそれぞれ上の中央と右の図が表示されるはずである。確めよ。

c) ここに示したものを参考に、回転・移転・拡大縮小を使って面白いパターンを描く関数 `art()` を定義せよ。

9 9章

練習 9.1 略

練習 9.2 (木構造の用語) 図 9.2(テキスト参照) に関して答えよ。

1. 根である連絡先の名前
2. 葉である全ての連絡先の名前
3. 節である全ての連絡先の名前
4. hamada の親にあたる連絡先の名前
5. hamada の子にあたる全ての連絡先の名前

解答例 9.2

a) Contact
b) Contact2,Contact3,Contact6,Contact7
c) Contact,Contact1,Contact4,Contact5
d) Contact
e) Contact2,Contact3

練習 9.3 略

練習 9.4 (最初の連絡先) 連絡先 p を根とする電話帳で、名前順で一番最初の連絡先を見つける関数 `first_contact(p)` をファイル `first_contact.rb` に定義せよ。(ヒント: 一番最初とは p から左 (left) へ進み続けた端にある連絡先のことである。)

解答例 9.4

```
#再帰バージョン
def first_contact(p)
  if p.left == nil
    p
  else
    first_contact(p.left)
  end
end

#ループバージョン
def first_contact_w(p)
  while p.left != nil
    p = p.left
  end
  p
end
```

練習 9.5 (予約リストの長さ) 予約情報 r から始まる予約リストの長さを求める関数 `request_length(r)` をファイル `request_length.rb` に定義せよ。(ヒント: 予約情報 r から始まるリストの長さと r.next から始まるリストの長さの再帰的な関係を考えよ。)

解答例 9.5

```
def request_length(r)
  if r == nil
    0
  else
    1+request_length(r.next)
  end
end
```

練習 9.6 (複数の予約の削除)

```
def delete_request(r,t)
  if r.title == t
    r.next
    else
    r.next = delete_request(r.next,t)
    r
  end
end
```

このファイルはリストの先頭から見て最初に見つけた予約を 1 つだけ削除する。また、リスト中に必ず削除されるべき予約があることも仮定している。これを修正して、同じ曲名の予約が複数あった場合にその全てを削除し、また、削除されるべき曲がない場合でもエラーとならないようにせよ。修正した関数はファイル `delete_request.rb` に定義せよ。

解答例 9.6

```
def delete_request(r,t)
  if r.title == t
    if r.next != nil
      r = r.next
      r = delete_request(r,t)
    else
      nil
    end
  else
    if r.next != nil
      r.next = delete_request(r.next,t)
      r
    end
  end
end
```

練習 9.7 (名前順の追加) 予約情報 r から始まる予約リストが曲名順に並んでいたときに、新しい予約情報 s を適切な場所に追加し、追加後の予約リストを答える `add_alphabetically(r,s)` を定義せよ。ただし曲名順とは、予約リストにある予約情報 t が以下の関係を満たしていることだとする。

$$\text{t.title} < \text{u.title} \text{ (ただし } \text{u} = \text{t.next)}$$

なお、第 9.2.2 節でも説明するように、Ruby では演算子 $<, >$ を使って文字列を比較できる。(ヒント: 予約の削除と同様に考えよ。`add_alphabetically(r,s)` が s を r の手前に追加するのはどのような場合か?)

解答例 9.7

```
def add_alphabetically(r,s)
  if r.title > s.title
    s.next = r
    s
  elsif r.next == nil
    r.next = s
  else
    r.next=add_alphabetically(r.next,s)
    r
  end
end
```

練習 9.8 (循環リスト) 下のような関数が作るリストを循環リストという。

```
def make_cycle(list)
  last = last_request(list)
  last.next = list
  list
end
```

a) `view(make_cycle(ut_songs()))` を実行して、どのような構造になっているかを確認せよ。
b) カラオケ演奏機の予約リストにこのようなデータが用いられた場合、どのようなことが起きるか?
c) これまでに定義した、予約リストを扱う関数のうち、予約リストが循環していると計算が止まらない (あるいはエラーになる) ものはどれか?

解答例 9.8

a) 省略
b) 演奏が終わらなくなる。最後の予約の次に最初の予約曲が入る。
c) `last_request` ,`add_request` ,`request_length`,`delete_request` など

練習 9.9 (カラオケ演奏機クラス) リスト構造 Request の操作を組み合わせて、カラオケ演奏機を模したクラス Karaoke をファイル karaoke.rb に定義せよ。いま `k = Karaoke.new()` として変数 k に Karaoke オ

ブジェクトをしまっているとする。クラス Karaoke は、以下のようなメソッドを持つものとする。

- `k.add("Tadahitotsu")` によって、リストの最後に予約を追加する
- `k.add_top("Tadahitotsu")` によって、リストの先頭に予約を追加する
- `k.play_next()` によって、次の曲を演奏する
- `k.cancel("Tadahitotsu")` によって、指定された曲名の予約を取り消す

解答例 9.9

練習 9.10 (末尾への追加の効率化) 予約リストの末尾に予約を追加する `add_request` は、リストの先頭から順にたどって最後の予約を見つけていた。この方法では n 個の予約があるときには n に比例した時間がかかるため、(カラオケでは考えにくいが) 非常に長いリストに何回も予約を追加する場合には適していない。そこで「予約リスト中の最後の予約」を常に変数 last に覚えておき、予約の追加は last のインスタンス変数 next に新しい予約を代入するようにすれば、予約リストの長さによらずに数回の代入だけで予約の追加ができるようになる。(もちろん追加した後は変数 last は、新たに追加された予約に変更しておかなければいけない。) 練習 9.9 で定義した Request をこのような方法で改良してみよ。具体的にはクラス Request にインスタンス変数 last を追加して、メソッド add の定義を変更すればよい。改良された Karaoke オブジェクトを図示すると下のようになるだろう。ただし、予約が 1 件もない場合には「最後の予約」自体が存在しないことに注意せよ。

解答例 9.10

```
class Karaoke
  attr_accessor("list")

  def initialize()
    self.list = Request.new(nil)
  end

  def add(title)
    if self.list.title == nil
      self.list.title =title
    else
      s = Request.new(title)
      last = last_request(self.list)
      add_request(last,s)
    end
  end

  def add_top(title)
    if self.list.title == nil
      self.list.title = title
    else
      s = Request.new(title)
      s.next = self.list
    end
  end

  def play_next()
    self.list = self.list.next
  end

  def cancel(title)
    delete_request(self.list,title)
  end
end


class Request
  attr_accessor("title","next","last")
```

```
  def initialize(t)
    self.title = t
    self.next = nil
    self.last = self.last_request
  end

  def last_request()
    if self.next == nil
      self
    else
      self.next.last_request
    end
  end
end
```

練習 9.11 (n 人目の連絡先) p を根とする電話帳の連絡先をアルファベット順に見たときに、先頭から n 目の連絡先をとり出す関数 `nth(p,n)` をファイル nth.rb に定義せよ。ただし、一番最初の連絡先は 0 番目だとする。また size(p)≤ n の場合は考えなくてよい。(ヒント: n < size(p.left) のとき、n 番目のは p の左右どちらにあるか?)

解答例 9.11

```
def size(p)
  if p == nil
    0
  else
    1 + size(p.left) + size(p.right)
  end
end


def nth(p,n)
  if n == size(p.left)
    p
  elsif n < size(p.left)
    nth(p.left,n)
  else
    nth(p.right,n-size(p.left)-1)
  end
end
```

練習 9.12 (深さ) 電話帳の中である連絡先に到達するまでには、インスタンス変数 left や right を何回かたどる必要がある。その回数を根から連絡先までの深さと言う。電話帳の深さとは、その電話帳に含まれてる連絡先の深さの最大値である。p を根とする電話帳の深さを求める関数 `depth(p)` をファイル depth.rb に定義せよ。

解答例 9.12

```
def depth(p)
  if p == nil
    0
  elsif p.left != nil || p.right != nil
    1 + max(depth(p.left),depth(p.right))
  else
    1
  end
end


def max(x,y)
  if y < x
```

```
    x
  else
    y
  end
end
```

練習 9.13 (二分木を用いた整列) 次のようにして配列を整列する関数 `binarysort(a)` をファイル binarysort.rb に定義せよ。

a) 配列 a から電話帳を作る関数 `array_to_tree(a)` を定義せよ。具体的には、名前 (name) が a[0], 番号 (number) が nil であるような連絡先を作りそれを根とする。さらに 1 番目以降の a[i] を名前として番号が nil であるような連絡先を順に作って根に加える (`add_contact`)。最後に根を答えとする。
b) 連絡先 p からたどれる全ての連絡先を名前順にたどり、その名前 (name) を配列 a の i 番目以降に代入する `tree_to_array(p,a,i)` を定義せよ。またこの関数は、最後に代入した番号を答とする。
c) 関数 `array_to_tree`, `tree_to_array` を使って binarysort(a) を定義せよ。

解答例 9.13

```
def array_to_tree(a)
  p = make1d(a.length-1)
  p[0] = Contact.new(a[0],nil)
  for i in 1..(a.length-1)
    p[i] = Contact.new(a[i],nil)
    p[0] = add_contact(p[0],p[i])
  end
  p[0]
end

def tree_to_array(p,a,i)
  for n in i..(i+size(p)-1)
    a[n] = nth(p,(n-i)).name
  end
  a
end

def binarysort(a)
  p = array_to_tree(a)
  tree_to_array(p,a,0)
end
```

練習 9.14 (二分木を用いた整列の計算量) 練習 9.13 で定義した整列アルゴリズムの計算量を考えてみよう。

a) p を根とする木の深さが n のとき、`add_contact(p,q)` の計算量を O 記法で表わせ。
b) m 個の連絡先を持つ電話帳の深さが最も小さい場合、その深さを m を使った式で表わせ。以降では、このようになっているときに「理想的な場合」と呼ぶことにする。
c) `array_to_tree(a)` の結果としてできる電話帳が理想的だったときの `array_to_tree` の計算量を O 記法で表わせ。ただし a の大きさを m とする。
d) 関数 `tree_to_array(p,a,i)` の計算量を O 記法で表わせ。ただし p からたどれる連絡先の個数は m だとする。
e) 関数 `binarysort(a)` の計算量を O 記法で表わせ。ただし a の大きさは m, 途中でできる電話帳は理想的だったとする。
f) `array_to_tree(a)` が作る電話帳が最も深くなるのは a に並んでいる値がどのような場合か。またそのときの深さはいくらになるか。

解答例 9.14

練習 9.15 (深さを考えた削除) 関数 `choose_root(p)` は、p の左右に連絡先があった場合、新しい根として名前順で p の直後に来る連絡先を選んでいる。しかしこれは、p の直前の連絡先としても構わない。そ

こで、p の左右に連絡先があった場合に、そこからたどれる連絡先の個数が多い側から次の根を選ぶように choose_root(p) をファイル contact_balance.rb に定義せよ。

解答例 9.15

```
def choose_root(p)
  if size(p.right) >= size(p.left)
    q = first_contact(p.right)
    r = delete_contact(p.right,q.name)
    q.left = p.left
    q.right = r
    q
  else
    q = last_contact(p.left)
    l = delete_contact(p.right,q.name)
    q.left = l
    q.right = p.right
  end
end

def last_contact(p)
  if p.right == nil
    p
  else
    last_contact(p.right)
  end
end
```

練習 9.16 (図形グループ) 第 8.2.4 節では、複数の図形から成る図案を表示・変形するために、配列に図形要素をしまって drawall, turnall などの関数を定義していた。この方法のかわりに「図形をグループ化した図形」のクラス Group をファイル oo-group.rb に定義してみよ。クラス Group のオブジェクトは、下のように図形要素配列を引数として作成され、draw や turn などのメソッドを持つものとする。

```
load("./oo-group.rb")
load("./oo-face.rb")
g = Group.new(face())# face() は練習 8.11 を参照
a = make2d(400,400)
g.draw (a)
show (a)
```

このクラス Group は、draw や turn ができるので、Line や Circle と同じく図形要素の 1 種である。しかも Group の内部に入るデータはまた図形要素であるので、再帰データ構造になっている。上の例では、Line や Circle などの図形要素から成るグループを作っているが、それをさらにグループ化して「複数の図形グループから成る図形グループ」といったような構造を作ることもできる。

解答例 9.16 略