プログラミング入門　　第1.1版　2004年8月24日

國分　尚

■■■　目次

■■■　１．アルゴリズムとプログラム

アルゴリズムとはある問題や物事に対して、それを解くあるいは実現するための手順のことで、算法などとも言う。もちろん、一つの問題について解法のアルゴリズムは一つとは限らない。例えば、２次方程式の解を求めるのに解の公式を使えばすべての場合に答えが出るが、簡単な因数分解でできることもある。コンピュータの場合はニュートン法を使うことが多い。

アルゴリズムは一見複雑な問題をうまく解くための手順であり、これをコンピュータが実行できるようにしたものがプログラムである。

例えば、かけ算は習ったが、分数はまだ知らない小学生にいきなり分数のかけ算の

問題を出しても答えがわかるはずはない。しかし、分数のかけ算はまず分子同士を掛けてその答えを書き、その下に横線を引いてさらにその下に分母同士のかけ算の答を書く、という手順を教えれば一応の解答は得られる。（約分の手順、つまり最大公約数を求めることは大変良いプログラムの例題なので、一度紙に書いてみるとよい。）

コンピュータに何かをさせる場合もこのように手順を細かく書いてやれば良い。特に解き方はわかっているが、手間がかかりすぎて人の手では計算できないことはコンピュータにおあつらえ向きの仕事である。例えば、1000番目の素数は何か、などプログラムは簡単に書けるが、手ではやりたくない。

プログラムはあるプログラミング言語で書かなければならないが、アルゴリズムそのものは普遍的であり、すべてのプログラムに共通である。とは言っても、プログラミング言語に機能が欠けている時はアルゴリズムを変更する必要もある。例えば、再帰的アルゴリズムはBASICやFORTRANでは実現できない。

あとで実際にプログラムに書き換えてみるが、今の1000番目の素数を求めるアルゴリズムは次のように書ける。これはしらみつぶし法と言う最も単純だが効率の悪いアルゴリズムであり、アルゴリズム事典のような参考書にはもっと賢いアルゴリズムが紹介されている。

アルゴリズム１．1000番目の素数を求める
2以外の偶数は素数ではないことに注目し、3から順にすべての奇数について素数であるかどうかを調べ、素数が１つ見つかるごとにカウンターを増やしていって、カウンターが1000になったらその素数を打ち出して終了する。

これではプログラムに書きにくいのでこれを土井・筧（1987）の記法で書き直すと、次のようになる。素数の判断部分は省略した。

アルゴリズム２．（概要）
●注目する奇数を3とする
●カウンターを2とする（これから見つけようとしている奇数、３は２番目）
■カウンターが1000以下である限り繰り返す
　　▼注目する奇数が素数であるなら
　　　　●注目する奇数を最後に見つかった素数として覚えておく
　　　　●カウンターを1増やす
　　●注目する奇数を次のものにする

●最後に見つかった素数を出力して終了する

ここで、●で始まる行は何かの操作を示す。■で始まる行は次の１段下げてある部分を繰り返すことを示す。▼は条件に当てはまるかどうかによって次の１段下げてある部分を実行する。

それでは、上の「素数であるなら」というのは具体的にどうすればよいだろうか。もう少し詳細に考えてみると、次のように書ける。

素数であるかどうかの判断は、調べたい数をまず３で割る。割り切れれば素数でないと判断する。割り切れなければ次の奇数である５で割ってみる。割り切れれば素数でないと判断する。割り切れなければ次の奇数である７で割ってみる。割り切れれば素数でないと判断する。割り切れなければ次の奇数である９で割ってみる。
・・・と繰り返していき、除数が調べたい数の1/3より大きくなったら素数であると判断する。

以上をまとめてもう少しプログラムらしく書くとこのようになる。

アルゴリズム３．（詳細）
●nを3とする（いま注目している奇数、3から始める）
●cを2とする（今見つけようとしている素数はc番目である）
●lastを2とする（最後に見つかった素数、3の１つ前は2）
●targetを1000とする（これを変えると任意の位置の素数が求められる）
■cがtarget以下である限り繰り返す
　　▼nが素数ならば
　　　●lastをnとする
　　　●cを1増やす
　　●nを2増やす（次の奇数を対象とする）
●lastを出力する

素数の判定（対象の数はnに入っている）
●iを3とする（除数、3からの奇数で割っていく）
■i =< n/3 であり（n/3より大きい数が約数になることはあり得ないので）、
　なおかつnをiで割った余りが0でない限り
　　●iを2増やす（次の奇数）
▼i > n/3ならば
　　●素数であると判断する

そうでなければ
●素数ではないと判断する

ここまで細かくアルゴリズムを分解すればたいていのプログラミング言語にすぐに書き出すことができる。

■■■　2．プログラミング言語

プログラミング言語は大きく分けてコンパイラ言語とインタープリタ言語に分類される。コンパイラはソースコード（人間が書くプログラムのこと）から機械語に翻訳した実行ファイルをあらかじめ作るため（これをコンパイルと言う）、プログラムの修正の度にコンパイルの時間がかかる（大規模なプログラムの時は数時間に及ぶ）が、出来上がったプログラムの実行は速い。インタープリタは実行するときにソースコードを１行ずつ解釈していくためプログラムの修正後直ちに実行できるが、プログラムの実行速度は遅い。

本講義で扱うRubyはインタープリタに分類されるが、実際には一旦ソースコードから内部コードに変換しているので純粋なインタープリタではない。また、BASICのように一つの言語でコンパイラとインタープリタの両方が存在しているものもある。

インタープリタの中でも特にスクリプト言語と呼ばれるものがある。スクリプトとは比較的短くて簡単に書ける割に複雑なことができるプログラムの総称で、Rubyはこのスクリプト言語の一種である。

世の中でよく使われる（ていた）プログラミング言語にはBASIC、C、C++、COBOL、FORTRAN、Java、Lisp、Pascal、Perl、Smalltalkなどがある。（書店のコンピュータ関係の本のコーナーに行けば並べられている本の数によってそのとき最もトレンディな言語がわかる。今ならC++とJava、Visual BASICが筆頭か。）その他にも一般にはあまり使われないが熱狂的なファンを持つ言語や、自分でプログラミング言語そのものを作ってしまう人などもあり、プログラミング言語の数は数百はあるだろう。（この世にはプログラミング言語おたくという人たちも存在する。）

同じアルゴリズムをプログラムにするのにそれぞれの言語で違った書き方をするのは厄介だが、それは慣れれば何とかなる。むしろ、与えられた問題に対して適切なアルゴリズムを考えつくかどうかがプログラミングで最も大切なことである。ま

た、よく使われるアルゴリズムを習って自分のものにしていく勉強も大切である。

■■■　３．Rubyプログラミング入門

上にも書いたように大事なのはアルゴリズムであって、プログラミング言語はそのときの条件にあった適当なものを選べば良い。本講義ではRubyという言語を選んだ。その理由は、studentsドメインのホストで実行できる、開発者が日本人であるため本家本元のテキストが日本語で書かれている、無料である、スクリプト言語なのでプログラムを実行するまでの道のりが簡単である、オブジェクト指向と言う比較的新しい考え方を実践している、はやりのスクリプト言語であるPerlに似ている、多くのパソコンで動く（Windows, Mac OS, Linux他）などがあるが、最大の理由は國分の趣味である。

Rubyはオブジェクト指向言語であるが、オブジェクト指向プログラミングだけを習うと他の言語への応用が難しいので、ここでは残念ながらRubyを手続き型の言語のように使った例を示す。このように応用が利くのがRubyの特徴とも言える。オブジェクト指向に興味がある人はがんばって自分で勉強して欲しい。

それでは、まず例に示したプログラムを使って解説する。

●　３-０．Hello World!

プログラミングを習う時は必ずこれをやることに決められているので．．．

プログラム０．画面に"Hello World!"と表示する

```
keyaki% ruby -e 'print "Hello World!¥n" '
```

これだけである。これはRubyインタープリタに-eのあとの' '（シングルクォーテーション）で囲まれた文字列をプログラムとして実行させている。eはexecuteまたはevaluateの頭文字だろう。このプログラムの部分だけを改めて書いてみると、

```
print "Hello World!¥n"
```

このプログラムはHello World!と言う文字列を画面に表示し、改行して終了する。

> **練習１．**最後の\nを書かずに実行したり、\n\nとするとどうなるか試してみる。また、Hello Worldの代わりに「こんにちは」とするとどうなるか。なお、\はキーボードではとなっていて、環境によっては実際にと表示される。

● 3-1．もう少し役に立つRubyプログラムの例-台形の面積を求める

Rubyプログラムの１例として、台形の面積を求めるプログラムを以下に示した。前のプログラムよりは実用性がある。例えば、将来子供が小学生になったとき、計算ドリルの答え合わせをしてあげられる。

プログラム1．daikei.rb

```
print "jouhen ="                #画面にjouhen =と表示する
jouhen = STDIN.gets             #変数jouhenにキーボードから数値を入力する
print "teihen ="                #画面にteihen =と表示する
teihen = STDIN.gets             #変数teihenに数値を入力する
print "takasa ="                #画面にtakasa =と表示する
takasa = STDIN.gets             #変数takasaに数値を入力する

a = jouhen.to_f      #jouhenに入力された文字を数値に変換して変数aに代入
b = teihen.to_f                 #teihenに入力された文字を数値に変換する
h = takasa.to_f                 #takasaに入力された文字を数値に変換する
area = (a+b)*h/2                #面積を計算し変数areaに代入する
print "menseki = ", area, "¥n"   #結果を画面に表示する
```

上辺の長さa、底辺b、および高さhの値を入力し、面積areaを計算し、その結果をディスプレイ画面上に出力している。用いた公式はもちろん

面積=(上辺+底辺)x高さ÷2

である。各行の説明を右側の#のあとに記してある（コメント文）。

● 3-2．Rubyプログラムの実行

Rubyはインタープリタ言語であるので、Cのようにコンパイルする手間がいらない。エディタでプログラムを書けばすぐに実行できる。

（１）エディタを用いてRubyプログラム（ソースプログラム）のファイルを作成する。これから複数のRubyプログラムファイルを作成していくので、専用のディレクトリを作成しておくとよいだろう。ディレクトリ名は例えばrubyとしておく。端末エミュレータで次のようにする。

```
keyaki% mkdir ruby
keyaki% cd ruby
```

続いて画面下の左から４番目のアイコンをクリックしてエディタを起動し、ファイルを作成する。保存する時は「ファイル」から「新規保存」を選んで保存ダイアログボックスを開き、左側のディレクトリ一覧からrubyディレクトリを選び、ファイル名を****.rbとする（例 daikei.rb）。実際には拡張子（.rb）は何でもいいのだが、こういう決まりは守っておく方がよい。

（２）Rubyインタープリタを用いてRubyプログラムを実行する。

ファイルを保存したら端末エミュレータで

```
keyaki% ruby daikei.rb
```

としてプログラムを実行する。プログラムに誤りがなければ画面にjouhen =と表示されてキーボードからの入力を受け付ける状態になるので適当な数値を入力する。入力後にエラーが表示されることもある。うまく行けば底辺、高さの数値を入力すると面積が表示されてプログラムが終了する。

エラーがあった場合はプログラムが停止する原因となったエラーとそれがある行の番号が表示されているので、エディタでプログラムファイルを開いたらまずその行を見てみる。

> **練習２．**プログラムファイルdaikei.rbを作成し実行してみること。ただし、コメント（#から右の文字）は入力する必要はない。

■■■　４．Rubyプログラムの構造

●　4-1．文、数値、文字列、変数、代入

Rubyプログラムは文の集まりである。他の多くのプログラミング言語と違い、Rubyの文は最後に何かの記号を付ける必要はない。ただし、複数の文を１行に書く時はセミコロンで区切る。

```
i = 1
a = 0; b = "text"
print "Hello world¥n"
print 'Hello world¥n'
c = 5 / 6
```

```
print c,"¥n"
c = 5.0 / 6.0
print c,"¥n"
```

この例でもわかるように数値はそのままの数字を書き、文字列は " " （ダブルクォーテーション）、または ' ' （シングルクォーテーション）でくくる。" " と ' ' には違った意味があるが、主に使うのは " " である。数値で気をつける必要があるのは整数と実数の違いで、小数点のない数値は整数として判断される。

> **練習３．**上の例を実行して""と''のちがい、小数点の有無に対する挙動に慣れること。また、上の割り算で、片方だけに小数点を付けるとどうなるか。

Rubyでは = は代入の記号であり、数学の等号とは異なる。

```
a = 5
```

は a という変数に 5 を代入している。二つの値が同じかどうかを調べるには==を使う。

```
a == 5
```

はaに代入されているのが5であれば真 (true)、そうでなければ偽 (false)である。

プログラミング言語で変数と呼ぶものは数学の変数とは異なる。簡単に言ってしまうとある値が入っている入れ物となる。（Rubyの場合、厳密には変数にはオブジェクトへの参照が入っている。）

RubyはCやPascal等と違って変数の型と言うものがなく、宣言をする必要もない。従ってプログラムに変数宣言部という特別な構造はない。しかし、次に出てくるローカル変数は初めに必ず代入をしないと使えないし（代入が暗示的な宣言になっている）、プログラムの見通しをよくするためにもあとで使う変数にプログラムの先頭であらかじめ初期値を代入しておくのは悪いことではない。

変数にはローカル変数、グローバル変数、インスタンス変数、定数の４種類があり、次のように名前の付け方で区別する。

ローカル変数	アルファベットの小文字で始まる
グローバル変数	$で始まる
インスタンス変数	@で始まる
定数	アルファベットの大文字で始まる

このうちグローバル変数は一般には使われない。また、インスタンス変数はオブジェクト指向プログラミング以外には使わない。定数は最初に一度だけ代入することができるが、その後は値を変えることはできない。

● 4-2．データの型、あるいはオブジェクトの種類

一般に、プログラミング言語では扱うデータが「型」と呼ばれる種類に厳密に分かれている。整数型、実数（浮動小数点）型、文字列型、論理型などがあり、変数も型が決まっている。例えば a という変数が整数型と指定されると整数以外のデータを入れようとするとエラーになる。

Rubyではこれらはオブジェクトの種類（クラス）と呼ぶ。例えば123という数字は他の言語では整数型のデータだが、Rubyでは整数クラスのオブジェクトである。すべてのオブジェクトはObjectという種類のオブジェクトでもあるので、Rubyの変数にはどんな種類のオブジェクトを代入してもよい。

```
a = 25
print a, "¥n"
a = "twenty-five"
print a, "¥n"
```

Rubyの特徴の１つに多倍長整数がある。普通のプログラミング言語では扱える整数の大きさには制限があり、±32767または0から65535であることが多い。しかし、Rubyでは整数の大きさはメモリの容量のみに制限され、例えば400の階乗のような大きな数も問題なく扱える。

練習４．次のプログラムfact.rbを作成し、実行してみる。

プログラム２．fact.rb（まつもと・石塚, 1999）

```
def fact(n)
    return 1 if n ==0    # 0の階乗は1である
    f = 1
    while n>0
        f *= n  # f = f * nと同じ
        n -= 1  # n = n - 1と同じ
    end
    return f
end

print fact(ARGV[0].to_i), "¥n"
```

ファイルを作成後、

```
keyaki% ruby fact.rb 10
3628800
```

と10の階乗が正しく計算されるのを確認したら

```
keyaki% ruby fact.rb 400
```

のようにすると400の階乗が出力される。この値は宇宙の直径と電子の直径の比よりも遥かに大きい。

● 4-3．配列とハッシュ

プログラミング言語で配列と呼ぶものも数学の配列とは異なる。簡単に言ってしまうと番号がついた変数のことで、この番号を別の変数を使って指定することで読み書きできるようになっている。

配列は　a[0]　とか　youbi[d]　の様な形をしている。
最初の例はaと言う名前の配列の0番目の要素、次の例はyoubiという名前の配列のd番目の要素と言うことで、dの値によって違う要素を指定できる。配列は下に出てくる繰り返しと組み合わせて使うことが多い。

プログラム３．dayofweek.rb

```
youbi = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday"]
print "Today is ",youbi[Time.now.wday],".¥n"
    #Time.now.wdayは今日が日曜なら0、金曜なら5という整数である
```

ハッシュ（連想配列）は配列と似ているが、要素を番号で指定するのでなく名前で指定するようになっている。使い方がわかると大変有効なものだが、ここでは紹介だけにとどめる。例えば　p_rec["name"]　というのはp_recという名前のハッシュのnameという要素を示す。

プログラム４．meibo.rb

```
def printPerson(p_rec)
    print "学籍番号: ", p_rec["id"],"¥n"
    print "名前: ", p_rec["name"],"¥n"
    print "出身地: ", p_rec["pob"],"¥n"
    print "性別: "
```

```
    if p_rec["sex"] = "M"
        print "男¥n"
    else
        print "女¥n"
    end
end

meibo=[]          # 空の配列を作る
meibo[0] = {"id" => "04H1111X", "name" => "田中一郎", "pob" =>
"新潟県", "sex" => "M"}  # 配列の0番目のデータとして{}内のハッシュを指定
meibo[1] = {"id" => "04H1112P", "name" => "新井素子", "pob" =>
"埼玉県", "sex" => "F"}
for person in meibo
    printPerson(person)
end
```

などという使い方が考えられる。いちいちこんなことをしなくてもid、name、age、sexという変数を使ったらいいのではないかと考える人は鋭い。ぜひともRubyプログラミングの参考書や一般的なプログラミングの本を調べて理由を考えてください。

● 4-4．制御構造

A. 文の連続

連続した文は上から順番に実行する。上のdaikei.rb等を参照。

B. 条件分岐

ある条件によって処理を変えたい時に使う。最も簡単には

if 条件
　条件に当てはまる時の処理
end

のように書く。もっと一般的には

if 条件１
　条件１に該当する時の処理
elsif 条件２
　条件２に該当する時の処理

...
elsif 条件n
　条件nに該当する時の処理
else
　どの条件にも当てはまらない時の処理
end

と言う形になる。この時に注意が必要なのは、条件１が成立した場合、その時の処理が終わったあとにはendの次の文を実行し、elsif以下の条件に当てはまるかどうかの判断は行わない、ということである。条件１に関わりなく条件２の判断をしたい場合は

if 条件１
　条件１に該当する時の処理
end
if 条件２
　条件２に該当する時の処理
end

と書く必要がある。

プログラム５．greeting.rb

```
hh = Time.now.hour            #Time.now.hourは現在時刻の時の部分
if hh >= 4 and hh <= 11
        print "おはようございます¥n"
elsif hh > 11 and hh <= 18
        print "こんにちは¥n"
elsif hh > 18
        print "こんばんは¥n"
else
        print "こんな時間に起きていると体を壊しますよ¥n"
end
```

夜中の１時頃このプログラムを実行して「大きなお世話だ」と言いたくなる人は適当に改造すること。

C. 繰り返し

ある条件が成立している間、同じ処理を繰り返す必要がしばしばある。これ

を行うのが繰り返しで、ループとも呼ぶ。Rubyは繰り返しの書き方が何通りもあるが、基本は次のようになる。

while 条件
　条件が成立している時の処理
　条件の成立に影響を与える処理
end

　よくあるプログラム上の間違い（bug,バグと呼ぶ）は繰り返しの処理の中で条件の成立に影響を与える処理を書き忘れることである。この場合運が悪いと繰り返しの中から抜け出せなくなる。これを無限ループと呼び、最も頻繁に出くわすバグの一つである。

　文章ではよくわからないので例を挙げると

プログラム６．sum10.rb

```
s = 0; i = 1
while i <= 10
        s = s + i
        i = i + 1    #カウンタを増やす
end
print s,"¥n"
```

　これは1から10までの和を求めるごくつまらないプログラムである。ところが、ここでカウンタを増やす文を書き忘れてしまうとプログラムはいつまでたっても終了しない。（このような事態に陥った場合はC-cをタイプするとプログラムを中止できる。）繰り返しの部分がこのように単純な場合はまず問題ないが、繰り返しの処理が複雑になってくると間違いが入り込む確率も上がることに注意する。

D. サブルーチン（メソッド）

　まとまりのある処理を値だけ変えて何度も行う必要があるとき、またはその処理を状況に応じて実行したりしなかったりする場合、一般にサブルーチン（または手続き、関数）と言うものを使う。Rubyの場合これをメソッドとよび、単なるサブルーチンとは性質が異なるが、オブジェクト指向プログラミングを意識しない限り普通のプログラミング言語のサブルーチンと違いはない。これまでの例題の中にも既に出て来ていて、プログラムX，fact.rbの中の

```
def fact(n)
```

```
        ....
  end
```

の部分がメソッドである。10の階乗でも400の階乗でもアルゴリズムは同じで、与える値を変える仕組みさえあればよい。これを行うのがメソッドである。

メソッドはdef ... で定義した部分では実行されず、他の部分から呼び出された時に初めて実行される。

● 4-5．アルゴリズムからプログラムへ

ここまでの情報をふまえてアルゴリズム３をRubyプログラムに書き直してみる。プログラミングではまず部品を完成させてから全体を作る方法と、全体の流れを作っておいてから細かい部分を作っていく方法があるが、今回の場合は後者は成立しないので（素数かどうかの判断ができないとプログラムが永久に終了しない）、まず素数の判定部分をメソッドとして作成し、それをテストしてみる。

プロブラム７．test_sosuu.rb

```
def sosuu?(x)
# 素数の判定を行う
# 入力条件: xは3以上の奇数
# 出力条件: 素数ならtrue、素数でなければfalseを返す
    i = 3
    while (i <= x/3) and (x % i != 0)    #x % i はxをiで割った余り
        i = i + 2   # i += 2 とも書ける
    end
    if i > x/3
        # 成立すればxは素数（割り切れた場合にはiは必ずx/3以下である）
        return true
    else
        return false
    end
end

print "n="
n=STDIN.gets.to_i
if sosuu?(n)
    print n," is a prime number.¥n"
else
    print n," is not a prime number.¥n"
```

```
end
```

練習５．test_sosuu.rbを実行して11が素数であり、9はそうでないことを確認し、そのほかのいくつかの数について素数の判定をする。このアルゴリズムは3以上の奇数について正しく働くが、偶数に対して判断するとどうなるか。

素数の判断に間違いがなければ、全体のプログラムに進む。これは次のようになる。ただし、全員が一斉に1000番目の素数を求めようとするとホストに負荷がかかりすぎる可能性があるので、100番目に変更してある。（一人だけで実行した時は1000番目で約4秒かかった。単純計算で100人同時だと400秒かかることになる。）このような場合、この1000とか100とかの数を後から簡単に変更できるようにプログラムを書くことは非常に重要である。下の例ではたった１か所を変更するだけでいいことに注意する。

プログラム８．sosuu.rb

```
def sosuu?(x)
    i = 3
    while (i <= x/3) and (x % i != 0)    #x % i はxをiで割った余り
        i = i + 2   # i += 2 とも書ける
    end
    if i > x/3      # 成立すればxは素数である
        return true
    else
        return false
    end
end

n = 3               # まずは3について調べる
c = 2               # もし、3が素数なら2番目の素数になる
last = 2            # 最初の素数は2であった（調べていないけど）
target = 100
while c <= target
    if sosuu?(n)    # nは素数か?
        last = n
        c = c + 1   # c += 1 とも書ける
    end
    n = n + 2       # n += 2 とも書ける
end
print target, "th prime is ", last, ".¥n"
```

練習６．sosuu.rbを実行して100番目の素数が541になることを確認する。また、プ

ログラムを変更して何通りかのn番目の素数を求める。

課題１． sosuu.rbを変更してキーボードから任意の自然数を入力し（プログラム１．daikei.rb、プログラム７．test_sosuu.rb参照）、n番目の素数を求めることができるようにする。または、プログラム２．fact.rbのやり方を参考にしてコマンドラインからnを指定するようにしてもよい。その時に、1番目の素数からn番目まですべての値をc: lastの形で出力すること。作成したプログラムと変更した部分についての説明をeメールで國分（hkokubun@faculty.chiba-u.jp）に提出すること。どうしてもプログラムができない時はアルゴリズム３を変更してアルゴリズムを提出すること。期日は6月10日（木）とする。
出力例
1: 2
1th prime is 2.

1: 2
2: 3
3: 5
4: 7
5: 11
6: 13
6th prime is 13.

応用１． 上の出力例で"1th prime is 2."となっているが、これでは大学生の作ったプログラムの出力としては恥ずかしいのでここも変更したい。targetを10で割った余りが1、2、3、それ以外の場合分けをすればよいだろう。（少し考えればこれでも十分でないことはわかるだろう。外国語の数字の扱いは大変である。ヒント：11はどうなる？）

● 4-6．オブジェクトとメソッド

オブジェクト指向プログラミングにおいてオブジェクトとは簡単に言うとデータとそれを操作するアルゴリズム（メソッド）をまとめたものである。オブジェクトは自分自身にどんなデータが格納されていて、それをどう扱うかを「知って」いる。例えばプログラム1の中にSTDIN.getsという表現が出てくるが、これはSTDINというオブジェクトのgetsというメソッドを使うことを意味し、「STDINにgetsメッセージを送る」と表現する。STDINは標準入力といわれるIOクラスの特別なオブジェクトであり、プログラムが使われた状況によってキーボードであったりファイルであったりするが、STDINオブジェクトはこれを知っていて、getsというメッ

セージをもらうと自分自身のおかれている状況に従って適切な処理を行い、呼び出したプログラムに１行分のデータを答えとして送り返す。
また、jouhen.to_fはjouhenという変数が示すオブジェクトにto_fというメッセージを送っている。プログラム1の場合はjouhenにはキーボードから入力した文字列がオブジェクトとして入っているので、文字列に対するto_fメッセージということになり、文字列オブジェクトは自分自身を数値に変換した値を答える。

■■■　５．ファイル処理と正規表現

コンピュータが電子計算機と呼ばれた頃は数値計算をさせるのが最も一般的な使い方であった。しかし、今日では文章を扱ったり、画像の編集をしたり、音楽を聴いたりすることの方が圧倒的に多く、「情報処理機」と呼ぶほうが的確であろう。

●　５-１．Genbankの出力を加工する

ここでは文字情報のファイルを扱う実用例としてGenBank (http://www.ncbi.nlm.nih.gov/)という遺伝子情報データベースの出力を使いやすい形に書き換えるプログラムを作ってみる。

GenBankからの出力は次のような形をしている。
genbank.txt（抜粋）

```
472, X12989. Petunia (Mitchell...[gi:20573]  Links

LOCUS       PHRBCSD5                 440 bp    DNA     linear   PLN
28-FEB-1992
DEFINITION  Petunia (Mitchell) 5' region of SSU911 gene for rbcS small subunit
            ribulose bisphosphate carboxylase (51-gene family).
ACCESSION   X12989
VERSION     X12989.1  GI:20573
KEYWORDS    rbcs gene; rbcS multigene family; ribulose bisphosphate
            carboxylase.
SOURCE      Petunia x hybrida
  ORGANISM  Petunia x hybrida
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; core eudicots;
            asterids; lamiids; Solanales; Solanaceae; Petunia.
REFERENCE   1  (bases 1 to 440)
  AUTHORS   Dean,C., van den Elzen,P., Tamaki,S., Dunsmuir,P. and Bedbrook,J.
  TITLE     Differential expression of the eight genes of the petunia ribulose
            bisphosphate carboxylase small subunit multi-gene family
  JOURNAL   EMBO J. 4, 3055-3061 (1985)
COMMENT     Paper compares members of multi-gene family encoding small subunit
            for rbcS. See accession numbers x12986-x12997 for all published 5'
            and 3' nucleotide sequences.
FEATURES             Location/Qualifiers
     source          1..440
```

```
                     /organism="Petunia x hybrida"
                     /mol_type="genomic DNA"
                     /strain="Mitchell"
                     /db_xref="taxon:4102"
                     /clone_lib="lambda EMBL3"
     misc_feature    1..437
                     /note="5' untranslated region SSU911"
     misc_feature    438..440
                     /note="translation initiation codon"
ORIGIN
        1 atatttggat atatacagta aagttttcaa gttatataca gaatttaaac taaagctaac
       61 atttccaaat ttattgtgca tctgaccttg gctatatgaa tccttaaata tcttattaaa
      121 gttctaaaat tatcataaaa aatgaaaaac ttgtccgtcg atctaatttt agacataata
      181 aaaacatatt ccattcatta atttggaaat gttaagataa ggactgagtg tagtgcgagg
      241 ggttaaattc atgtgggcct tcaatttagc aattcaagaa ccagtgaacc acaacaccac
      301 ataatccaaa tgttaccgtt cctctaagat gaggtttgct cgatttgtgt ccgttagatg
      361 agaaaaggat gtacaacctt atcactataa atagatattg caaatgtcaa gggaagcaat
      421 agcaattata tttagcaatg
//

473, X12994. Petunia (Mitchell...[gi:20572]  Links

LOCUS       PHRBCSD3                142 bp    DNA     linear   PLN
28-FEB-1992
DEFINITION  Petunia (Mitchell) 3' region of SSU491 gene for rbcS small subunit
            ribulose bisphosphate carboxylase (51-gene family).
ACCESSION   X12994
VERSION     X12994.1  GI:20572
KEYWORDS    rbcs gene; rbcS multigene family; ribulose bisphosphate
            carboxylase.
SOURCE      Petunia x hybrida
  ORGANISM  Petunia x hybrida
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; core eudicots;
            asterids; lamiids; Solanales; Solanaceae; Petunia.
REFERENCE   1  (bases 1 to 142)
  AUTHORS   Dean,C., van den Elzen,P., Tamaki,S., Dunsmuir,P. and Bedbrook,J.
  TITLE     Differential expression of the eight genes of the petunia ribulose
            bisphosphate carboxylase small subunit multi-gene family
  JOURNAL   EMBO J. 4, 3055-3061 (1985)
COMMENT     Paper compares members of multi-gene family encoding small subunit
            for rbcS. See accession numbers x12986-x12997 for all published 5'
            and 3' nucleotide sequences.
FEATURES             Location/Qualifiers
     source          1..142
                     /organism="Petunia x hybrida"
                     /mol_type="genomic DNA"
                     /strain="Mitchell"
                     /db_xref="taxon:4102"
                     /clone_lib="lambda EMBL3"
     misc_feature    1..3
                     /note="translation stop codon"
     misc_feature    4..142
                     /note="3' untranslated region SSU491"
ORIGIN
        1 tagatttcat tttaagacaa cttaccctgt ctctttaggg gatgtttgtt tgaatcagtt
       61 tttttcctgg aaaaattgct tttgttcctc tttttttaat tctttctatt cagtctatgt
      121 tgtcggatca gtttatgcgt ac
//
```

482, Z13997. P.Hybrida myb.Ph2...[gi:20560] Links

```
LOCUS       PHMYBPH22               1057 bp    mRNA    linear   PLN
12-SEP-1993
DEFINITION  P.Hybrida myb.Ph2 gene encoding protein 2.
ACCESSION   Z13997
VERSION     Z13997.1  GI:20560
KEYWORDS    myb homologue; myb.Ph2 gene; Petunia protein 2.
SOURCE      Petunia x hybrida
  ORGANISM  Petunia x hybrida
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; core eudicots;
            asterids; lamiids; Solanales; Solanaceae; Petunia.
REFERENCE   1  (bases 1 to 1057)
  AUTHORS   Avila,J., Nieto,C., Canas,L., Benito,M.J. and Paz-Ares,J.
  TITLE     Petunia hybrida genes related to the maize regulatory C1 gene and
            to animal myb proto-oncogenes
  JOURNAL   Plant J. 3 (4), 553-562 (1993)
  MEDLINE   94035159
   PUBMED   8220462
REFERENCE   2  (bases 1 to 1056)
  AUTHORS   Avila,J., Nieto,C., Canas,L., Benito,M. and Paz-Ares,J.
  TITLE     Petunia hybrida genes related to the maize regulatory C1 gene and
            to animal myb proto-oncogenes
  JOURNAL   Unpublished
REFERENCE   3  (bases 1 to 1057)
  AUTHORS   Paz-Ares,J.
  TITLE     Direct Submission
  JOURNAL   Submitted (01-JUL-1992) Paz-Ares J., Centro de Investigaciones
            Biologicas (C.S.I.C.), Ingenieria Genetica, C\Velazquez 144,
            Madrid, Madrid, Spain, 28006
FEATURES             Location/Qualifiers
     source          1..1057
                     /organism="Petunia x hybrida"
                     /mol_type="mRNA"
                     /strain="v26"
                     /db_xref="taxon:4102"
                     /clone="cPF2"
                     /tissue_type="floral"
                     /clone_lib="cDNA library in gt10"
                     /dev_stage="developing flowers"
                     /germline
     gene            1..1057
                     /gene="myb.Ph2"
     5'UTR           <1..131
                     /gene="myb.Ph2"
     CDS             132..974
                     /gene="myb.Ph2"
                     /function="DNA-binding protein, transcriptional activator"
                     /standard_name="MYB.Ph2"
                     /note="related to animal myb proto-oncoproteins"
                     /citation=[2]
                     /codon_start=1
                     /product="protein 2"
                     /protein_id="CAA78387.1"
                     /db_xref="GI:20561"
                     /db_xref="GOA:Q02993"
                     /db_xref="TrEMBL:Q02993"
```

```
                     /translation="MGRAPCCEKMGLKKGPWTPEEDQILVSYIEKNGHGNWRALPKLA
                                   GLLRCGKSCRLRWTNYLRPDIKRGNFTREEEDTIIQLHEMLGNR
                                   WSAIAARLPGRTDNEIKNVWHTHLKKRLKNYQPPQNSKRHSKNN
                                   HDSKAPSTSKMLDNSESFSTIQENINEPMTGPNSPQRSSSESST
                                   VTADSLAATDVTNDDQTFIKHEEMDSYENFPEIDESFWTEDLSM
                                   GDNLDLDMEVAGEKSQVQFPYSHDMKEQSVDMVGAKLEDDMDFW
                                   YNVFIKAEDLLDLPEF"
     3'UTR           974..1056
                     /gene="myb.Ph2"
                     /evidence=experimental
ORIGIN
        1 ccttcccatt tctttcattt tcttattgga acacaacaca caaccaaaga gacagtactt
       61 gtgaagaaca aaccaattga ttagagtcat tcaaagaagc aagatcagca agtaaacaaa
      121 gcaaaggaaa aatgggaaga gctccttgtt gtgaaaagat ggggctgaag aaagggccat
      181 ggactcctga agaagatcaa attcttgtct cttatattga aaagaatggt catggcaact
      241 ggagagccct ccctaagcta gctggacttt tgagatgtgg gaagagttgc agactccgtt
      301 ggactaacta tttgcgtcca gatattaaga ggggcaactt cacaagagaa gaagaagata
      361 ccattatcca gttacatgaa atgcttggca atagatggtc tgcaatagca gcgagattac
      421 cgggacgaac ggacaacgaa attaaaaatg tatggcacac tcacttgaag aaaaggctta
      481 aaaattacca gcctcctcag aactccaaaa gacactccaa gaacaaccat gattccaaag
      541 cacctagtac ttctaaaatg ttagacaatt cagaaagttt tagcaccatt caagaaaaca
      601 ttaatgagcc catgaccggt ccgaactcgc cacaacggtc atctagcgag tcatcaactg
      661 tcacggccga ttcattggcc gcgacagatg ttacaaacga tgatcaaacg tttattaagc
      721 acgaggaaat ggactcgtat gaaaattttc cagagattga tgagagcttt tggacggagg
      781 atttatccat gggagataat ttggatcttg atatggaggt tgctggtgaa aaatcacaag
      841 ttcaatttcc atattctcat gacatgaagg aacaaagtgt cgacatggtt ggagcaaaat
      901 tagaggacga catggacttt tggtacaatg ttttcataaa ggctgaggac ttattagact
      961 taccagaatt ttgaggggtc aaattagctg taatataaaa cttgaagtag tggaatgcca
     1021 gctaactaaa ctggtgctgg gtattttgag ggaattc
//
```

さて、この形式のファイルが与えられたが、あなたが必要なのはそのうちのわずかな情報で、しかも塩基配列はすべて１行につながっていて数字や空白がない方が都合がいいとしたらどうするだろう。

エディタやワープロで編集することもできるが、せいぜい数十レコードまでであろう。それ以上は人間がすべき仕事ではない。このような単純作業こそコンピュータが最も得意とする分野なのだから。

そこで、このようなプログラムを作ってみる。必要な情報は塩基配列の定義[DEFINITION]、レコード番号[ACCESSION]、核酸が取られた生物の名前[SOURCE]、この配列に関する文献[REFERENCE]（複数ある場合は最初の１つだけ）、配列そのもの[ORIGIN]の数字と改行、空白を除いたもの、とする。

プログラム９．genbank.rb

```
#    GenBank output formatting program
#         H. Kokubun
#         ver 0.1   30 Apr 2004 The first rendation
#         ver 0.2    1 May 2004 Selected some fields to display.
```

```
def usage
print <<"EOT"
genbank.rb usage:
     genbank.rb filename
          Format the GenBank output file given by filename.
          Output is to STDOUT, use redirection to write to a file.
     genbank.rb -help | -h | -?
          Get this help.
EOT
end

#if the first argument is related to "help!", display the usage and
# end the program
if /-help|-h|-¥?/ =~ ARGV[0]
     usage()
     exit()
end

while line = ARGF.gets          #コマンドラインで指定したファイルから1行読み込み、
                                #まだファイルの終わりでない間繰り返す
     if /LOCUS/ =~ line
          #読んだ行の中にLOCUSという文字列があれば、レコードの始まりと見なす
          #データをよく見れば正確にはこの行がレコードの始まりではないことがわかるだろう
          print "¥n----------¥n",line  #レコードの始まりなので区切り線を書く
     elsif /DEFINITION|ACCESSION|SOURCE/ =~ line
          #これらの文字列を含む行が必要なので、出力する
          print line
     elsif /REFERENCE¥s+1¥s/ =~ line
          #REFERENCEの場合は、情報が複数の行にわたっている上、最初の一つだけを取り出す
          #ために特別な処理になる
          print line
          line = ARGF.gets
          until /REFERENCE|FEATURES|COMMENT/ =~ line
               print line
               line = ARGF.gets
          end
     elsif /ORIGIN/ =~ line
          #塩基配列の場合の処理
          print line
          line = ARGF.gets
          seq = ""
          begin
               line.scan /[A-Za-z]+/ do |aChunk|
                    seq << aChunk
               end
               line = ARGF.gets
          end until /¥/¥// =~ line
          print seq, "¥n"
     end
end
```

● 5-2．ファイルからデータを読み込む

```
while line = ARGF.gets
    あれこれ
end
```

この形がRubyでファイルからデータを読み込む基本となる。ARGFとは特殊なファイルで、

```
keyaki% ruby test.rb a.txt b.txt c.txt
```

という形でコマンドが与えられたとき、a.txt、b.txt、c.txtの３つのファイルから１行づつデータを読み出す。最後のファイルの最後に達するとnilという特別な値が返り、whileの条件を満たさなくなるのでループから脱出する。

例えば、UNIXのcatコマンドをRubyで書くと

プログラム１０．cat.rb

```
while line = ARGF.gets
    print line
end
```

と、たった３行で済んでしまう。（１行で書くやり方もある。）

● 5-3．ファイルにデータを書き込む

Rubyでファイルにデータを書き込む方法も当然あるが（IOクラスのputsメソッドを使う）、今回はOSのリダイレクションを使うことにする。コマンドラインで

```
keyaki% ruby fact.rb 100 > output.txt
```

とすればprint文の出力は画面ではなくoutput.txtというファイルに保存される。リダイレクションについては「キャンパス情報リテラシー」p. 90を参照のこと。今回の例題ではこれで十分なのでRubyのファイルの扱いに関しては参考書で勉強してほしい。

● 5-4．正規表現

genbank.rbのプログラム中には　/あれこれ/ =~ 変数　という部分が多くあるが、/あれこれ/が正規表現と呼ばれるもので、文字列中にあるパターンが含まれる

（マッチする）かどうかを見つける手段である。文字列の検索や加工には欠かすことができない。正規表現はそれ自体がミニプログラミング言語であると言われるほど複雑なことができ、その説明だけで１冊の本が書けるほどである。ここではプログラム９．の例について最低限のものにしぼって説明する。

ア．特殊文字以外はその文字列と一致するものとマッチする

```
/ORIGIN/ =~ line
```

この表現はlineの中に“ORIGIN"という文字列があればマッチし、式の値が真になる。マッチした部分は$~という変数に格納されている。

イ．｜で区切ると区切りの両側の表現のどちらかにマッチする

```
/REFERENCE|FEATURES|COMMENT/ =~ line
```

この表現はlineの中に“REFERENCE"、“FEATURES"、“COMMENT"という文字列のいずれかがあればマッチする。複数ある時は最も左側のものにマッチする。

ウ．[ ]の中のどれか１文字にマッチする

```
/[A-Za-z]/ =~ line
```

この表現はlineの中の最初の英文字にマッチし、その文字を返す。A-Zは ABCDEF...XYZを省略した表記法である。

エ．+は直前の表現の１回以上の繰り返し、＊は０回以上の繰り返し、?は０回または１回、{m,n}はm回以上n回以下に相当する

オ．\sは空白文字（スペース、タブ、改行）、\nは改行、\tはタブ、^は行頭、$は行末、．は任意の１文字にマッチする

```
/^REFERENCE¥s+1¥s/ =~ line
```

この表現はlineの先頭に“REFERENCE"があり、最低１文字のスペースのあとに“1"があり、その直後にスペースがあるような文字列にマッチする。

REFERENCE　　1 abcde	○
a REFERENCE　1　12345	×（REFERENCEが先頭にない）
REFERENCE　12	×（1の直後がスペースでない）

カ．正規表現の中で特別な意味がある文字（メタ文字）を本来の文字として使う時は前に\を付ける

```
/¥/¥// =~ line
```

この表現は//にマッチする。

キ. ( )で表現をグループ化できる

`/abc+/` はabc, abcc, abcccなどにマッチするのに対し
`/(abc)+/` はabc, abcabc, abcabcabcなどにマッチする。

練習７． genbank.rbを書き換えて塩基配列の中に特定な配列、例えばtcga（制限酵素*TaqI*の認識配列）があったらその配列の前後20塩基までを出力するようにせよ。次の正規表現を使うと簡単にできる。

```
if /.{0,20}tcga.{0,20}/ =~ seq
    print "Taq I recognition site: ",$~,"¥n"
end
```

.はどんな1文字にもマッチする。{,20}は直前の表現を0から20回繰り返したものにマッチする（この書き方はgrepやawkでは使えない）。Rubyの正規表現は最長一致を原則としているのでtcgaの前に20文字以上あれば20文字がマッチする。後ろも同様である。$~に直前にマッチした文字列が格納されている。

課題２． genbankの出力はこのままではExcelなどの表計算ソフトに取り込むことはできない。そのためには一般に１レコードを１行とし、各データをタブという特殊文字（ \t で表す）で区切って書き出すのが便利である。これをタブ区切りテキストと言う。

そこで、上のgenbank.txtを読んで各レコードを
`LOCUS ¥t ACCESSION ¥t 長さ ¥t SOURCE ¥n`
のような形式で出力するプログラムを作成する。
例えば

```
LOCUS       PHMYBPH22       1057 bp    mRNA    linear   PLN
12-SEP-1993
DEFINITION  P.Hybrida myb.Ph2 gene encoding protein 2.
ACCESSION   Z13997
SOURCE      Petunia x hybrida
```

というレコードなら
`PHMYBPH22¥tZ13997¥t1057¥tPetunia x hybrida¥n`
のようにする。

LOCUS名と長さはLOCUSの行の2番目と3番目の単語を取り出すようにすればよいだろう。１行のデータを単語に分けるには文字列のsplitメソッドを使う。

words = line.split()

とするとwordsという配列にlineの内容を空白で分割した文字列が入る。上のLOCUSの行であればwords[0]はLOCUS, words[2]は1057となる。

作成したプログラムとその説明をeメールで國分（hkokubun@faculty.chiba-u.jp）に提出すること。どうしてもプログラムができない場合はアルゴリズムを考えて提出すること。期日は７月末とする。

■■■　６．その他

●　よく使うUNIXコマンド

cat ファイル名
ファイル名のファイルを画面に出力する。

cd ディレクトリ名
カレントディレクトリを移動する。

ls ディレクトリ名
ディレクトリの内容を表示する。ディレクトリ名を省略するとカレントディレクトリを表示する。ls -F とするとファイルとディレクトリの区別がつく。ls -l とするとファイルの属性やサイズがわかる。

mkdir ディレクトリ名
ディレクトリを作る。

pwd
カレントディレクトリのパス名を表示する。

●　Rubyプログラムファイルに実行権限を与える。

プログラムの先頭に次の行を書き加え、ファイルの属性を変えればファイル名を与えただけでプログラムを実行できる。

```
#!/usr/local/bin/ruby
```

これはプログラムを翻訳して実行するRubyインタープリタの場所を示すもので、利用するシステムによって異なることがある。千葉大学の教育用システムや多くのLinuxはこの場所だがMacOS Xでは/usr/bin/rubyとなる。

```
keyaki% chmod a+x aProgram.rb
```

上記のコマンドだとすべてのユーザーがこのプログラムを実行できるので、自分以外のものが実行するのを防止するにはchmod u+xとする。「キャンパス情報リテラシー」p. 83参照。上記のコマンドでファイルの属性を変更したらls -lで実際に属性が変わっていることを確認する。

以上の操作のあとは

```
keyaki% aProgram.rb
```

とするとプログラムを実行することができる。

Linuxなど一般のUNIXシステムではPATH環境変数にカレントディレクトリが入っていないためにこのままでは実行できない。その場合はファイル名の前に ./ を付けることが必要となる。セキュリティ上の理由からこのようになっているのだが、もし毎回./をつけるのが面倒であれば環境変数のPATHに./を付け足せば省略できるようになる。

■■■　7．参考文献

千葉大学情報処理教育研究会．2002．キャンパス情報リテラシー，第４版．昭晃堂.

土井範久・筧捷彦．1987．コンピュータ入門１　プログラミングの考え方．岩波書店．（アルゴリズムの考え方が良く分かり、プログラミングを初めてやる人や将来プログラマーになりたい人にはぜひとも読んでほしい本だが、既に絶版である）

まつもとゆきひろ・石塚圭樹．1999．オブジェクト指向スクリプト言語Ruby．アスキー出版局．（Rubyの開発者による標準テキストだが、プログラミング初心者向きではない）

● プログラミング一般の入門書

伊藤華子．2002．コンピュータプログラミング入門以前，第２版．毎日コミュニケーションズ．（土井ら，1987の代わりになる本だと思うのだが、実物を見ていないので断定できない）

● Ruby入門書

高橋征義・後藤裕蔵．2002．たのしいRuby　Rubyで始める気軽なプログラミング．ソフトバンク　パブリッシング．（プログラム例が無味乾燥だが、Rubyの基本を習うにはよい。アルゴリズムの勉強にはならない。）

● 言語リファレンス

まつもとゆきひろ．2000．Rubyデスクトップリファレンス．オライリー・ジャパン．（文法、キーワード、基本クラスなど、Rubyで本格的にプログラミングするために不可欠）