

「プログラミングによる問題解決」入門

教育実践総合センター 小山智史

授業のねらいと内容

パソコンは、「どこでどう活用するか」が決まっているわけではありません。その判断をも含めた柔軟な対応能力を養うことが重要です。単なる操作方法の修得では不十分です。

最近のアプリケーションソフトは高度な処理を簡単に行ってくれる一方、危惧もあります。それは、「コンピュータの中で行われていることはとても自分には理解できない高度なものだ」と初めから思い込み、主体的に責任を持って情報を取り扱うことをしなくなることです。それに拍車をかけるのが「適当に操作すると何か結果が得られる」という恵まれた(?)環境です。

この授業では Windows 以前にひとつの時代を築いた MS-DOS の環境に戻り、文字中心のインタフェース (CUI) でプログラミングの演習を行います。そこでは正確な指示を与え、結果を正しく読み取ることが求められます。プログラミング技術を習得することよりも、例題や練習問題を通じて柔軟な対応能力が身につくよう題材を吟味しました。半期の間、心地好い「頭のトレーニング」を楽しんでください。

授業は下記のように計画しています。

1～3	CUIによるパソコン操作
4～7	AWK プログラミング (入門編)
8～10	AWK プログラミング (応用編)
11～15	(課題)

出席と評価

試験は行いません。小課題とレポートで評価します。

第1章 CUIによるパソコン操作

1.1 はじめてのMS-DOS

[よく使うプログラム](一般には[スタート][プログラム])から[MS-DOS プロンプト]を起動すると、ウィンドウが開き

```
H>
```

と表示され、カーソルが現れる。これはプロンプトと呼ばれ、「どんな様何をしましょうか?」と利用者に仕事の指示を促しているのである。

ここで、「仕事を指示する」わけだが、試しにキーボードをでたらめに押して、最後に `Enter` を押してみなさい。

```
H>e9rakd7a34df9f Enter
```

コマンドまたはファイル名が違います。

```
H>
```

「そんな意味不明なことを指示されても困る」というメッセージが返ってくる。

このように、パソコンからの「文字の」プロンプトに促され、キーボードから「文字で」仕事の指示を与える。パソコンの仕事の進捗状況や結果はディスプレイに「文字で」表示される。これがCUI(Character User Interface)である。これに対し、WindowsやMacintoshのインタフェースはGUI(Graphical User Interface)と呼ばれる。

1.2 ファイルの操作

1.2.1 ドライブ

パソコンにはフロッピーディスクを回転させてデータを読み書きするドライブ¹が内蔵されている。他にはハードディスクドライブやCD-ROMドライブがパソコンに内蔵されていて、これらのドライブはABC...の名前をつけて区別される。ドライブ名とメディアの対応関係と各々の特徴を表1.1にまとめた。

ドライブの切り換え

フロッピーディスク(Aドライブ)で仕事をしたい時には

```
H>a: Enter
```

と操作する。すると

```
A>
```

となる(以後 `Enter` の表記は省略する)。

¹ドライブ(drive)はディスクやCD-ROMなどの記憶媒体を「駆動」して読み書きするための装置。

名前	ドライブ
A	フロッピーディスクドライブ
C	ハードディスクドライブ
D	CD-ROM ドライブ
H	ネットワークドライブ(各自のホーム)
P	ネットワークドライブ(全員共通)
W	ネットワークドライブ(Web 公開用)

表 1.1 パソコンで利用できるドライブ

1.2.2 ファイル

ハードディスクやフロッピーディスクにデータを記録する場合、データをファイルに格納し、ファイルには名前をつけて管理する。ファイルの名前の付け方には制約があり、sumoやkijiのように8文字以内の英数字列にする。大文字小文字は区別されない。いい加減な名前をつけてしまうと、ファイルの数が増えてきたときに中に何をしまったかわからなくなるので注意。

ファイル名にはsumo.csvやkiji.txtのように、「.(ピリオド)」と1~3文字の拡張子を付け加えることができる。kiji.txtはテキストファイルであることを表している。Windowsの場合は、拡張子でアプリケーションソフトと関連づけがされている。また、Windowsでは「8文字以内の英数字列」の制約は緩和されている。

どんなファイルがあるかを見る: dir コマンド

現在使っているディスクには、すでによくつかのファイルが入っていて、それらのファイルには名前が付いている。どのような名前のファイルがあるか見るにはdirコマンドを使う。

```
H>dir
```

と操作してみなさい。

```
...
SUMO    CSV          504  97-07-24  18:07 SUMO.CSV
...
```

のように、ファイルの名前、サイズ、ファイルを作った日時が表示される。サイズの単位はバイト(byte)で、英数字ならば504文字分のデータが中に格納されていることを示している。

ファイルの名前だけを見たいときは、

```
H>dir/w
```

とする。

ファイルの数も数十個になると、その中から自分の関心のあるファイルを見つけるのが大変になる。そんな時は、

```
H>dir *.csv
...
H>dir hello.*
```

のようにする。「*」は「そこはどんな文字列でもよい」ということを意味している。ファイル名のところでは、ワイ

ルドカードと呼ばれる 2 つの特殊文字を使うことができる。「?」は、その位置に「任意の 1 文字」があてはまり、「*」は、その位置に「任意の 0 文字以上の文字列」があてはまる。

(練習) H ドライブにどのようなファイルがあるかを、MS-DOS と Windows の両方の操作で調べなさい²。

ファイルの内容を見る : type コマンド

dir コマンドで確認したように、ディスクの中には kiji.txt という名前のファイルがある。type コマンドで内容をディスプレイに表示してみよう。

```
H>type kiji.txt
```

どんなファイルでもこのように見れるかということ、そうではない。実行形式のプログラムファイルなどは、見るためのファイルではないから、そのようなものを無理に見ようとするとどうなるか... いずれ試してみよう。

Windows の場合は、type コマンドに相当する機能はないので、メモ帳を起動してファイルの中身を表示する。

同じ操作を繰り返す

プロンプト (H>) に対して直前の操作を繰り返す場合は **f3** を使う (続けて **Enter**)。もしももっと前に行った操作を再度行いたいならば、 を使う。これにより、キー操作を大幅に軽減できる。

同じ内容のファイルを作る (複写) : copy コマンド

既にあるファイルと同じ内容のファイルをもうひとつ作る時には copy コマンドを使う。

```
H>copy kiji.txt test.txt
```

これで、kiji.txt と全く同じ内容の test.txt という名前のファイルが新しく作られる。dir コマンド、type コマンドおよび Windows の操作で確かめなさい。

ファイル名を変更する : ren コマンド

ファイル名を変更するには ren コマンドを使う。

```
H>ren test.txt new.txt
```

dir コマンド、type コマンドおよび Windows の操作で確かめなさい。

ファイルを消去する : del コマンド

ファイルを消去するには del コマンドを使う。

```
H>del new.txt
```

²Windows の操作で拡張子の文字が表示されないことがあるかもしれない。その場合は、[表示][オプション] の [表示] タブで、「登録されているファイルの拡張子は表示しない」のチェックをはずすと拡張子を含めたファイル名が表示されるようになる。

これで new.txt というファイルは消えてしまう。

ファイルの数は増える一方である。「保存しておく必要のないファイルはその場で消す」ことが肝心。

1.2.3 ディレクトリ

ファイルの数が多くなると、どれが何のファイルかわからなくなってくる。そこで、ディスクの中にディレクトリを作り、用途や種類によってファイルの置き場所を分ける。ディレクトリ (directory) は登録簿という意味で、ここでは、管理下にあるファイルとディレクトリの目録の意味で使われる。Windows や Macintosh ではフォルダと呼ばれる。

ディレクトリの中に更にディレクトリを作ることができ、ディレクトリは「階層構造」になっている。下図は H ドライブの中のファイルの様子 (一部) である。< > はディレクトリを表している。

```
H: ¥
  <PROG>
  .      lib.awk
  .      hello.awk
  .      ...
  .      sumo.csv
```

図を 90 度回転して「H: ¥」を下にすると「木」のように見える。「¥」は「木の根」に相当することから「ルート (根) ディレクトリ」と呼ばれる。ファイルは葉に相当する。

dir コマンドではカレントディレクトリ内のファイル一覧が表示される。

カレントディレクトリを移動する : cd コマンド

カレントディレクトリを移動するコマンドが cd コマンドである。cd コマンドで移動しながら、dir コマンドで中にどんなファイルがあるか見てみよう。まず、

```
H>cd ¥
```

でルートディレクトリに移動し、続いて dir コマンドでファイルの一覧を表示してみなさい。ここで、<DIR>の印がついた項目がディレクトリで、PROG というディレクトリがあることがわかる (図と照合のこと)。

```
H>cd prog
```

の操作で prog というディレクトリに移動できる。このように、図と照合しながら演習用ディスクの中を「探検」してみなさい。親ディレクトリに戻るには

```
H>cd ..
```

と操作する。dir コマンドで表示されるファイル一覧の中に、「.」や「..」という名前があるが、「.」はカレントディレクトリ、「..」は親ディレクトリを表している。

以上の操作では、カレントディレクトリからの相対パスで移動先を指定していることになる。

ひとつおり探検が終わったら、

```
H>cd ¥prog
```

と操作しなさい。この操作では目的地を絶対パス (ルートディレクトリからの経路) で直接指定している。

ディレクトリを作る：md コマンド

新しくディレクトリを作るには md コマンドを使う。

これからは A: ¥home ディレクトリで仕事をするが、ファイルの数が増えてきたら、ここに用途別や課題別にディレクトリを作成して整理すると良いかもしれない。

ディレクトリを消す：rd コマンド

ディレクトリを消すには、そのディレクトリの中のファイルをすべて消してから、rd コマンドを使う。

1.3 リダイレクションとパイプ

MS-DOS のコマンドの多くは、入力はキーボード (標準入力) から行い、結果はディスプレイ (標準出力) に表示されている。コマンド行で特別な指示をすることにより、入力をキーボードからファイルに切り換えたり、出力をディスプレイからファイルやプリンタに切り換えることができる。これを入出力のリダイレクトという。

また、あるコマンドの出力を別のコマンドの入力に直結して渡すパイプと呼ばれる機能がある。

出力のリダイレクト

dir をはじめ多くのコマンドは、結果をディスプレイに表示されている。ここで、ディスプレイに表示するかわりにファイルにしまうには次のようにする。

```
H>dir >test.txt
```

これで、本来表示されたはずのものが test.txt というファイルにしまわれる。dir コマンドと type コマンドで確かめなさい。「>」が出力のリダイレクトを指示する記号である。

既にあるファイルに追加したい場合は、

```
H>dir/w >>test.txt
```

のように「>>」を使う。test.txt の内容を確認しなさい。今度は、

```
H>dir >test.txt
```

と操作して、test.txt の内容を見なさい。さっきまでの test.txt の中身は失われ、新しい内容に置き替わっていることがわかる。

入力のリダイレクト

sort というプログラムがある³。このプログラムは、キーボードから何行か文字を入力すると、それを行単位でアルファベット順に並べ換えて表示してくれる。実行の様子は例えば次のようになる。

³MS-DOS にも sort が付属しているが、ここでは J.W.Rider 氏が作成したフリーソフトを使用する。カラムを指定したり、数値でソートすることができる。「sort /h」で使い方が表示される。

```
H>sort
iida
abe
ueno
~Z
abe
iida
ueno
H>
```

逆順に並べ換えたいときは、

```
H>sort /r
```

とする。でも、実はこのようにデータをキーボードから入力して使うことはない。ソートしたいデータファイルが既にあって (例えば test.txt)、

```
H>sort <test.txt
```

のようにする。test.txt というファイルの中のデータが、アルファベット順になって表示される。「<」が入力をリダイレクトすることを指示する記号である。

演習用ディスクには sumo.csv というファイルがあるので、dir コマンドや type コマンドで確認しなさい。中身は相撲の横綱のデータである。各行は「名前, 出身地, 身長, 体重」で構成されている。

```
H>sort <sumo.csv
```

とすると、各行の先頭文字からアルファベット順に比べて並べ替えられる (といっても日本語文字なのでおそらく望むような順にはならない)。身長の順に表示する時は

```
H>sort /f /t, /3 /n <sumo.csv
```

とする。「/f」と「/t,」と「/3」は、「,」を区切り文字とした 3 番目のフィールドについて比較する指示、「/n」は数値で比較する指示である。

入力のリダイレクト機能と出力のリダイレクト機能を組み合わせて

```
H>sort <sumo.csv >test.csv
```

のようにすれば、sumo.csv というファイルの中身が並べ換えられて test.csv というファイルが作られる。

(練習) 体重について逆順のソートをして taiju.csv というファイルにしなさい。

パイプ

あるプログラムの出力を、そのまま別のプログラムの入力にしたいことがある。前出の例では、一旦 test.txt というファイルを作ってから sort を実行したが、dir から sort に直接データを引き渡すことができるならば、途中の test.txt というファイルを作る必要は無かった。そのためには次のようにする。

```
H>dir | sort
```

このように、2 つのプログラムの間を「|」で区切ると、本来ディスプレイに表示されるところと本来キーボードから入力するところが、パイプで結ばれる。この結果をファイルにしまいたいならば、

```
H>dir | sort >test1.txt
```

のようにする。パイプは何段も連続して結ぶことができる。sort のようなコマンドをフィルタというが、フィルタには他に more や find などがある。

dir や type を実行すると表示量がたくさんあるために、はじめの部分がディスプレイから消えてしまうことは既に経験した。more を使えば、丁度ディスプレイにいっぱい表示されたところで一旦停止してくれる。続きを見るときは スペース を押す。

```
H>type kiji.txt | more
```

この他、ファイルの中から文字列を見つける find コマンドがある。

```
H>dir | find "2004"
```

とすると、「2004 の文字列を含む (2004 年に作成した) ファイルやディレクトリ」を表示してくれる。find はいかにも「フィルタ」と呼ぶにふさわしい。

(練習) sumo.csv から青森県出身者のデータを抽出し、aomori.csv というファイルにしよう。

1.4 プログラムの呼び出し

dir や type や del や cd, md, rdなどは内部コマンドと呼ばれ、OS が機能を提供している。これに対し、メモ帳のプログラムは notepad.exe というファイルに格納されていて、

```
H>notepad
```

という操作で呼び出すことができる。拡張子が exe, com, bat のファイル (実行ファイル) は、MS-DOS からファイル名を入力することで呼び出すことができる (外部コマンド)。

実行ファイルはカレントディレクトリまたはパスが設定してあるディレクトリに置く。notepad.exe はパスが設定されている c:\windows に格納されている。

どのディレクトリにパスが設定されているかは

```
H>path
```

の操作で確認できる。

Windows では、実行ファイルのアイコンをダブルクリックすることでプログラムを呼び出すことができる。また、拡張子とアプリケーションの関連づけにより、例えば、

.txt のファイルであれば、このファイルアイコンをダブルクリックすることにより、メモ帳 (notepad.exe) が起動する。どの拡張子がどのアプリケーションと関連づけられているかは、[ツール][フォルダオプション]の[ファイルのタイプ] タブで調べることができる。

カレントディレクトリに無くて、パスが設定してあるディレクトリにも無い実行ファイルは、ファイルパスを指定して呼び出すことができる。キー操作は少し長いが、

```
H>"c:\Program Files\Microsoft Office\Office10\excel.exe"
```

のように Excel を呼び出すことができる。

1.5 バッチプログラム

簡単なバッチプログラム

上記の

```
H>dir | find "2004"
```

の操作を「dir04」の操作で済むようにしてみよう。このために必要なことは、「dir | find "2004"」という内容のファイル dir04.bat を作るだけである (このファイルを実行ファイルと言う)。このファイルを作り、動作を確認しよう。

1度このようなバッチファイルを作っておけば、その後もいつでもコマンドとして利用できる。

(練習) Excel を呼び出す xls.bat というファイルを作り、動作を確認しよう。

パラメータの指定

上記の「dir04」はカレントディレクトリにしか利用できない。

```
H>dir04 %sample
```

のようにパラメータを指定できるようにするにはどうしたらいいだろうか。このためには、dir04.bat の内容を「dir %1 | find "2004"」と変更する。

```
H>dir
...
H>dir04
...
H>dir sample
...
H>dir04 sample
...
H>dir %html
...
H>dir04 %html
...
H>
```

1 番目のパラメータを %1 と書くが、必要な場合は、%2, %3, ... と複数のパラメータを与えることができる。

バッチファイル中に何行かを記載すれば、それらは順に処理される。

以下の内容で dirnote.bat というファイルを作成しておけば、dirnote.bat を呼び出すことにより、データを処理してその結果をファイルに格納し (1 行目)、メモ帳でそのファイルが開かれる (2 行目)。

```
dir | sort >test1.txt
notepad test1.txt
```

第2章 AWK プログラミング (入門編)

2.1 プログラムとプログラミング言語

プログラミング言語はコンピュータが理解できる特別な言葉である¹。プログラミング言語を使ってコンピュータにさせたい仕事を記述したものがプログラムである。プログラミング言語には多くの種類があり、目的によって使い分けられるが、ここでは、AWK というプログラミング言語を用いてプログラムを作成する²。

AWK 言語で書いたプログラムを実行するには、「AWK の解釈プログラム (インタプリタ)」を呼び出し、プログラムの 1 行ずつを解釈しながら実行させる (図 2.1(a) 下)。

他の多くの言語では、文字で記述されたプログラム (ソースプログラム) を「翻訳プログラム (コンパイラ)」を使って「コンピュータだけが理解できる機械語プログラム」に変換し、この機械語プログラムを実行するという方法をとる (図 2.1(a) 上)。市販のアプリケーションソフトでは機械語プログラムが販売され、通常はソースプログラムが販売されることはない。

次節で「データ処理」のプログラムを取り上げるが、この場合は図 2.1(b) のように考えることができる。

さて、演習用ディスクには4つの異なる言語で書かれたプログラム (hello.awk, hello.ub, hello.c, hello.jp) が入っている。どれも

「ディスプレイに『hello, world』と表示しなさい」

という意味の短いプログラムである。type コマンドで内容を確認し、実行してみなさい。教育用システムでは AWK インタプリタ、BASIC インタプリタ、C コンパイラがそれぞれ、jgawk、ubv、lcc で実行できる。

(a) AWK 言語で書かれたプログラム

```
H>type hello.awk
...
H>jgawk -f hello.awk ...AWK プログラムの実行
hello, world ... 実行結果
H>
```

「jgawk -f hello.awk」という操作は

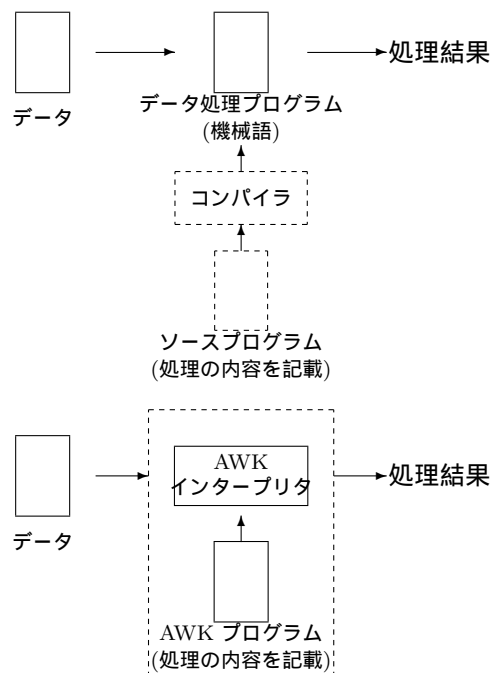
「jgawk さんをお願いします。私が作った hello.awk というプログラムを解釈して、そこに書かれた仕事をしてください」

¹プログラミング言語に対し、我々が用いる日本語や英語は自然言語と呼ばれる。

²AWK 言語は、広く使われている C 言語とよく似た記法のプログラミング言語で、開発者である Aho, Weinberger, Kernighan の頭文字をとって名前が付けられた。ここでは、米国 FSF から配布されている gawk を田中良知氏が日本語化した jgawk というフリーソフトを利用している。AWK 言語の詳細は「プログラミング言語 AWK(トッパン)」を参照。



(a) プログラムの実行



(b) データ処理

図 2.1 コンパイラとインタプリタ

と読む。

(b) BASIC 言語で書かれたプログラム

```
H>type hello.ub
...
H>ubv hello.ub ...BASIC プログラムの実行
hello, world ... 実行結果
H>
```

「ubv hello.ub」という操作は

「ubv さんをお願いします。私が作った hello.ub というプログラムを解釈して、そこに書かれた仕事をしてください」

と読む。

(c) C 言語で書かれたプログラム

```
H>type hello.c
...
H>lcc hello.c ...C プログラムのコンパイル
H>hello ... 機械語プログラムの実行
hello, world ... 実行結果
H>
```

「lcc hello.c」という操作は C プログラムのコンパイルを指示するもので、

「lcc さんをお願いします。私が作った hello.c というプログラムを機械語に直して hello.exe というファイルを作ってください」

と読む。次の「hello」は実行の指示で、

「hello.exe の機械語プログラムを実行してください」

と読む。1 度機械語プログラム (hello.exe) を作ってしまえば、プログラムを実行するのにソースプログラム (hello.c) は必要ない。

(d) 日本語で書かれたプログラム

```
H>type hello.jpn
...
H>jgawk -f hello.jpn ... 日本語プログラムの実行
...
H>
```

結果はどうなるだろうか。日本語で書いた指示を AWK インタープリタで処理させようとしてもうまくいかない。自然言語の処理はプログラミング言語の処理よりも遥かに難しく、日本語インタープリタや日本語コンパイラはない。

前掲の図と照らし合わせて、「言語」の違いや、プログラム記述言語と言語処理ソフト (コンパイラやインタープリタ) の関係を理解していただきたい。

(練習) 「jgawk -f hello.ub」「lcc hello.awk」など、対応しない言語処理ソフトを用いるとどうなるかを試しなさい。

(練習) hello.exe (機械語プログラム) を type コマンドで表示してみなさい。また、ren コマンドで hello.exe を hello.txt に変更してから、type コマンドで表示してみなさい (名前を変えても中身は一緒)。

2.2 AWK プログラムの実行

2.2.1 プログラムとデータ

big.awk というプログラムファイルと sumo.csv というデータファイルが入っている。

```
big.awk:
1: BEGIN{FS=","}
2: $4>=150
```

```
sumo.csv:
1: 栃錦, 東京都, 178, 127
2: 若乃花, 青森県, 177, 105
3: 朝潮, 鹿児島県, 188, 145
4: 柏戸, 山形県, 188, 146
5: 大鵬, 北海道, 187, 144
6: 栃ノ海, 青森県, 177, 107
7: 佐田の山, 長崎県, 182, 122
8: 玉の海, 愛知県, 177, 135
9: 北の富士, 北海道, 185, 135
10: 琴桜, 鳥取県, 181, 143
11: 輪島, 石川県, 185, 115
12: 北の湖, 北海道, 180, 150
13: 若乃花 II, 青森県, 187, 123
14: 三重ノ海, 三重県, 180, 135
15: 千代の富士, 北海道, 182, 116
16: 隆の里, 青森県, 182, 155
17: 双羽黒, 三重県, 199, 157
18: 大乃国, 北海道, 189, 203
19: 北勝海, 北海道, 181, 151
20: 旭富士, 青森県, 189, 143
21: 曙, ハワイ, 204, 225
22: 貴乃花, 東京都, 186, 145
```

ここで、「big.awk:」や「sumo.csv:」はファイル名、「1: ~」は行番号を便宜的に示したものである。dir コマンドと type コマンドでこれらのファイルを確認しなさい。

sumo.csv には、横綱のデータが入っていて、big.awk には「体重が 150kg 以上のデータを抽出するプログラム」が入っている。

ここで、以下のように操作してみなさい。

```
H>jgawk -f big.awk sumo.csv
...
H>
```

この操作は

「jgawk さんをお願いします。big.awk のプログラムを解釈して、そこに書かれたとおり sumo.csv のデータを処理してください」

と読む。より具体的には

「横綱のデータの中から体重が 150kg 以上のものを探しなさい」

ということを意味している。実行結果を確認のこと。

(練習) 上と同様の処理を Excel を用いて行いなさい。

2.2.2 データの作成と修正

データファイルはテキストエディタを使って作成・編集する。

sumo.csv には、各行に「名前 出身地 身長 体重」の各データが「,(カンマ)」で区切られて記録されている。

ここで、1 行分に相当する 1 件のデータをレコードという。1 番目のレコードは栃錦に関するデータである。名前、出身地、身長、体重の各項目のことをフィールドという。4 番目のフィールドは体重のデータである。データファイルは、レコード区切り文字が「改行」、フィールド区切り文字が「,(カンマ)」になっている。区切り文字を変えることもできる。

データファイルを作る場合は、次のことに注意する。

- (1) 数値や区切り文字の「,(カンマ)」を入力するときには、必ず日本語入力モードを解除し、半角文字とする。全角の数字は数値とみなされず、全角「,(カンマ)」は区切り文字とみなされない。
- (2) 1 行が 1 レコードに対応するので、行末は必ず改行すること。最終行に改行を忘れると、プログラム実行時にエラーとなる。空行 (改行だけの行) も 1 件のレコードとみなされ、余計な処理をしてしまうので注意。

さて、データファイルのどこを見ても「kg」とか「cm」などの単位は書かれていないし、どこが体重の項目であるかも書かれていない。また、プログラムを見ても書かない。データがどのように格納されているかを正確に知って、それに基づいてプログラムを作る。データとプログラムの整合性が重要だということに注意。

(練習) テキストエディタで `sumo.csv` の最後に、あなたのデータを付け加え、`big.awk` で処理してみなさい。また、追加したデータの体重の値を変えると結果がどう変わるか調べなさい。

2.2.3 プログラムの作成と修正

プログラムファイルはテキストエディタを使って作成・編集する。

`big.awk` の内容は「`$4>=150`」というものであった。これは、体重が 150kg 以上のデータを表示する AWK プログラムである。「`$4`」は 4 番目のフィールドを表している(詳しくは次節)。

プログラムは、

```
H>jgawk -f big.awk sumo.csv
```

のように実行するが、自分でプログラムを作っても、すぐにうまく動作するわけではない。「誤り箇所を修正して保存し、実行する」ということを、うまく動作するまで繰り返すが、この作業をデバグという。

プログラムが所定の規則(文法)に従って記述されていない場合は、AWK インタープリタが誤りを指摘してくれる。指摘は英語である上、適切な表現でないかもしれないが、「プログラムの何行目で問題が生じたか」は参考になることが多い。このようなエラーを文法エラーという。

文法的には正しいが不注意の入力ミスや考え方に間違いがある場合、例えば上記の例で「`$4>=1500`」としてしまったらどうだろうか。このようなエラーを論理エラーといい、プログラムは「指示どおりに正しく」動作する。従って、一応動作するようになって、それが妥当な結果かどうかを別の方法で確認することが重要である。

(練習) `big.awk` のプログラム中の「150」の値を変えると結果がどう変わるか調べなさい。プログラムが確かに自分が意図したように動作しているという確信を持ってほしい。

2.2.4 実行結果の保存

上の例では実行結果は画面に表示される。次のように、リダイレクト機能を使うと実行結果は画面に表示される代わりにファイルに格納される。

```
H>jgawk -f big.awk sumo.csv >big.log
...
H>
```

処理結果を再利用したい場合はこのようにする。レポート作成時にも利用できる。

(練習) `sumo.csv` に関して、身長が 190cm 以上のデータを表示する `tall.awk` というプログラムを作り、実行しなさい。データファイル、プログラムファイル、実行結果のそれぞれに簡単な説明を付けて、レポートを作成し、報告しなさい。

2.3 AWK 言語の文法

2.3.1 パターンとアクション

`big.awk` の「`$4>=150`」というプログラムは、実は

```
$4>=150{print $0}
```

の省略形である。`$1`, `$2`, ..., `$4` は、それぞれ 1 番目のフィールド、2 番目のフィールド、..., 4 番目のフィールドの値を示すフィールド変数で、`$0` でレコード(すべてのフィールド)が参照できる。

「`$4>=150`」の部分をパターン、「`print $0`」の部分をアクションという。AWK プログラムは

```
パターン{アクション} の並び
関数定義 の並び
```

である。「パターン{アクション}」は

「パターンに適合したレコードにアクションを実施しなさい」

と読み、データファイルのレコードをひとつ読んで「パターン{アクション}」を最後のレコードまで繰り返す³。

関数定義については 2.3.6 で説明する。

パターンとアクションのうちいずれかを省略することができる。パターンを省略すると、すべてのレコードに適合する。すなわち、

```
big.awk:
1: BEGIN{FS=","}
2: {print $1,$4}
```

はすべての横綱の名前と体重を表示する。また、初めに説明したとおり、

```
big.awk:
1: BEGIN{FS=","}
2: $4>=150
```

のようにアクションを省略すると「`print $0`」というアクションが仮定され、パターンに適合するレコードが表示される。

特殊なパターンとして「`BEGIN`」と「`END`」がある。「`BEGIN`」はレコードの入力をはじめる前、「`END`」はすべてのレコードの処理が終わった後に、それぞれ 1 度だけ適合する。これらは無くても良いし、複数あった場合は順に処理される。データファイルのデータを処理するのでなければ、「`BEGIN{ }`」の部分だけがあれば良い。2.1 節の `hello.awk` は、このような例である。

さて、上記のプログラムはアクションが 1 つの文(`print` 文)で構成されていたが、一般にアクションは「文の並び」である。文は「~しなさい」という指示の単位で文の末尾には「; (セミコロン)」を置くが省略しても良い。1 行にふたつ以上の文を書くこともでき、その場合は文と文の間に「;」が必要である。

文字列の中を除き、プログラム中に「`#`」が現れると、以後行末までコメントとなる。コメントはプログラムの実行に影響せず、プログラムの説明などを書く。

³ 「パターン{アクション}」が複数ある場合は、1 つレコードを読み込む毎に、すべての「パターン{アクション}」を順に適用する。

2.3.2 定数と変数

定数は、数値データまたは文字列データで、文字列データは、「"ABC"」のように前後を「"(二重引用符)"」で囲む。

変数はデータを入れる「箱」のことで、名前(変数名)が付けられる。宣言せずに突然変数を使い始めることができ、0 または空文字列("") に初期設定される。変数名は、アルファベットで始まる英数文字列で、大文字と小文字は区別される。大文字の変数名には、組み込み変数として使われているものもあるので、なるべく小文字を使う。

例を示そう。big.awk を次のようにして実行してみなさい⁴。

```
big.awk:
1: BEGIN{
2:   FS=","
3:   n=0
4: }
5: $4>=150{
6:   n=n+1
7:   print NR,$1,$4
8: }
9: END{print n "人います" }
```

3 行目で(レコードを読み込む前に)変数 n に定数 0 を代入している。

1 レコードずつ読み込み、「\$4>=150」が真の場合だけ「n=n+1」と「print NR,\$1,\$4」が実行される。「n=n+1」は変数 n の値に 1 を加える⁵。結果、全レコードの処理を終了した END{ } の中では、n の値が、「\$4>=150」を満たしたレコード数になっている。9 行目ではこれを表示している。

現在のレコードが何番目かが NR という変数に、いくつかのフィールドからなるかが NF という変数に自動的にセットされる。sumo.csv では、どのレコードも NF の値は 4 となるので、\$4 の箇所は \$NF としてもよい。また、フィールド区切り文字はスペースまたはタブであるが、「FS=","(カンマ)」とするとフィールド区切り文字を「,」に変更できる。このように予め名前と役割が決まっている変数を組み込み変数という。

(練習) big.awk のプログラムを次のようにわざと間違えて実行してみなさい。ただし、全部一度にやるとわからなくなるので、1ヶ所ずつ調べること。

- (1) BEGIN{ を「BIGIN{」と間違う
- (2) 字下げに全角スペースを使う
- (3) 「"」を全角の「”」にする
- (4) 「{」の相棒の「}」を忘れる

2.3.3 演算と式

```
big.awk:
6:   n=n+1
```

⁴プログラムの構造を見やすくするために、「{」と「}」の中を字下げ(インデント)して書くことが多い。プログラム途中の空白は無視され、実行結果に影響しない。ただし、空白には半角スペースやタブを用い、全角スペースを用いるとエラーになるので注意。

⁵「=」は左辺と右辺が等しいということではなく、左辺の変数に右辺の値を代入する意味なので注意。

の「n+1」は式、「+」は加算の演算を指示する演算子と呼ばれる。式は値を持つ。「=」は代入演算子で、右側の値を左側の変数に代入し、その値を式の値とする⁶。「n=n+1」は式であり、文である。算術演算子には+(加算)、-(減算)、*(乗算)、/(除算)、%(剰余)、^(べき乗)がある。

上記の「n=n+1」は、「n+=1」「n++」「++n」としてもよい。「+=」は左側の変数に右側の値を加えて代入する演算子で、同様のものに「-=」「*=」「/=」「%=」「^=」がある。「++」は変数に 1 を加えるインクリメント演算子で、同様のものに「--」(デクリメント演算子)がある⁷。

「3+4*2」のように演算が組み合わせあった場合に、どの演算が優先されるかは決まっている。それを明示したい場合は「3+(4*2)」のようにすれば良い。なお、このような場合の括弧記号は() を使い、{ } や [] は使わない。

「\$4>=150」は \$4 の値によって真(1) または偽(0) となる。「>」「>=」「==」「<=」「<」などは関係演算子と呼ばれる。「==」は左右の値が等しい時に真となるが、「=」の代入演算子と間違いやすいので注意。また、関係演算子は文字列にも適用され、「"ABC"=="ABC"」「"A"<"ABC"」「"A"<"B"」などが真となる(「"10"<"2"」も真となるので注意)。

論理演算は、1(真) と 0(偽) を値とする数値の演算として扱われる。「&&(AND: 論理積)」は左が真かつ右も真の時に式の値が真となる。「|| (OR: 論理和)」は左が真または右が真の時に式の値が真となる。「!(NOT: 否定)」は右が真の時に偽、偽の時に真となる。なお、このような論理演算はパターンの組み合わせにも用いることができる。

「+」のように、その左右の値に演算を施す演算子を 2 項演算子という。これに対し、「-1」の「-」や「!(否定)」を単項演算子という。

文字列の演算には接続演算がある。

```
y="ABC" "DE"
```

のようにふたつの文字列を並べると(半角スペースで区切るだけで演算子の記号はない)、接続演算(ふたつの文字列をくっつける演算)が行われ、y には演算結果である"ABCDE"という文字列が代入される。

```
big.awk:
9: print n "人います"
```

の箇所では、「n "人います"」で接続演算が行われ、その結果を print で表示している。ここで、n の値は数値であるが自動的に文字列に変換された後、接続演算が行われる。このように、文字列データと数値データは、演算の途中で変換する必要が生じると、文脈に応じて自動的に変換される。自動的な型変換は大変便利である反面、思わぬ落とし穴もあるので要注意である⁸。

⁶従って「x=y=0」のような書き方も可。

⁷前置は 1 を加えた後の値を式の値とし、後置は 1 を加える前の値を式の値とする。「x=2; y=x++; z=++x」とすると、y の値は 2、z の値は 4 となる。「--」についても同様。

⁸変数 a のデータを明示的に型変換させるには、「a+0」のように 0 を加算して数値データとしたり、「a ""」のように空文字列と接続演算を行って文字列とする。

(練習) (1) 体重が 150kg 以上かつ身長が 190cm 以上の横綱の名前を表示しなさい。(2) 体重が 150kg 未満または身長が 190cm 未満の横綱の名前を表示しなさい。(3)(2) の否定を表示するようにし、(1) と同じになるか確認しなさい。また、これらの関係をベン図で示しなさい。

(練習) big.awk を参考に、青森県出身の横綱の名前と出身地を表示し、最後に人数を表示するプログラム syussin.awk を作りなさい。

2.3.4 入出力文

これまで、「\$4>=150」のようにプログラム中で 150 の値を固定していた。この値を変える場合はプログラムを変更しなければならなかった。そこで、この値を実行時に指定できるようにしてみよう。

big.awk を次のようにして実行してみなさい。初めに「体重?」の表示が現れるので、何 kg 以上の横綱を表示するか数を入力する。他は前と同様である。

```
big.awk:
1: BEGIN{
2:   FS=","
3:   printf "体重?"
4:   getline t<"con"
5:   n=0
6: }
7: $4>=t{
8:   n=n+1
9:   print $1,$4
10: }
11: END{print n "人います" }
```

3 行目は「printf」であることに注意。print が行末で改行されるのに対し、printf では改行されない。

4 行目は、キーボード ("con") から入力した文字列を変数 t に代入する getline 文である⁹。「getline t」あるいは単に「getline」とすると、実行時に指定したデータファイル (ここでは sumo.csv) から 1 行 (1 レコード) 入力し、t あるいは \$0, \$1 ~ に代入する。9 行目は指定した文字列をディスプレイに表示する print 文である。「print \$1,\$4」のようにデータを「,(カンマ)」で区切ると、そこは半角スペースで埋められる¹⁰。ここで、9 行目を次のように変更して実行してみなさい。

```
big.awk:
9:   printf "%-10s%5d¥n",$1,$4
```

「%-10s%5d¥n」は表示の書式 (format) を示している。「%-10s」は、10 文字分の場所に左詰め (-) で、文字列 (s) として \$1 を表示せよという指示である。「%5d」は 5 文字分の場所に右詰めで、整数値 (d) として \$4 を表示せよという指示である。面倒ではあるが、このように細かな表示の指定が可能であり、表示の不揃いを解消できる。

なお、「¥n」は改行を表す特殊文字で、print 文や printf 文で用いることができる。他の特殊文字としては、タブを表す「¥t」などが用いられる。

以下に printf 文で用いられる書式変換の例を示す。

⁹getline 文では、1.3 節で示した「入力のリダイレクト機能」を利用できる。「con」はキーボードを示す特殊なファイル名である。

¹⁰print 文で表示する場合の区切り文字は OFS という組み込み変数で決まり、初めは半角スペースになっている。変更可能。

```
printf "%c",65      |A|
printf "%d",65     |65|
printf "%5d",65    | 65|
printf "%05d",65   |00065|
printf "%f",65     |65.000000|
printf "%5.1f",65  | 65.0|
printf "%e",65     |6.500000e+01|
printf "%5.1e",65  |6.5e+01|
printf "%s","hirosaki" |hirosaki|
printf "%10s","hirosaki" | hirosaki|
printf "%-10s","hirosaki" |hirosaki |
```

(練習) syussin.awk を変更し、初めに出身地を入力し、そこで指定した出身地の横綱の名前と人数を表示するようにしなさい。

(練習) 次のように、初めに 2 つの数を入力し、さまざまな演算結果を表示するプログラム calc.awk を作りなさい。

```
H>jgawk -f calc.awk
x? 3
y? 2
3+2=5
3-2=1
3*2=6
3/2=1.5
3%2=1
3^2=9
3 2=32
H>
```

データファイルからデータを読み込むことはないので、プログラムは BEGIN{ } の部分だけでよい。

(練習) calc.awk を参考に、別の題材のプログラム (例えば底辺と高さを入力して三角形の面積を表示するプログラム) を 3 つ作り、そのプログラムを誰か他の人に使ってもらい、感想を聞きなさい。

汎用機器と専用機器

パソコンは汎用機器であり、電源を入れた後、OS (MS-DOS や Windows などのシステムプログラム) にやりたい仕事を指示する。これに対し、ビデオデッキやエアコンや電気炊飯器の中にもコンピュータが内蔵されているが、ここでは特定の機能を実現するために専用のプログラムが動作している。パソコンでも、電源を入れるとすぐに決まったプログラム (例えば calc.awk) が動作するようにでき、こうすると、パソコンを含むシステム全体は専用機器となる。単一目的で利用する業務用パソコンの多くはこのようになっている。

2.3.5 組み込み関数の利用

ローレル指数は次式で計算できる。

$$\text{ローレル指数} = \frac{\text{体重}}{(\text{身長})^3} \times 10000000$$

この計算値は次のように評価される。

~ 100	100 ~ 115	115 ~ 145	145 ~ 160	160 ~
やせすぎ	やせぎみ	正常	太りぎみ	太りすぎ

次のプログラムは、データファイル (ここでは sumo.csv) の身長と体重の値からローレル指数を計算して表示する。

```
rohrer.awk:
1: BEGIN{FS=","}
```

```
2: {
3:   x=$4*10000000/$3^3
4:   print $1,x
5: }
```

3行目では、身長(\$3)と体重(\$4)からローレル指数を計算し、xに代入している。4行目で名前とローレル指数を表示している。実行すると下記ようになる。

```
H>jgawk -f rohrer.awk sumo.csv
柄錦 225.187
若乃花 189.352
...
```

ここで、小数点以下を表示させないようにしたい。

```
rohrer.awk:
4:   printf "%-10s%d¥n",$1,x
```

としても良いが、ここでは `int()` という組み込み関数を用い、

```
rohrer.awk:
4:   print $1,int(x)
```

としてみよう。`int()` は、数値データを引数(ひきすう)として渡すと、その小数点以下を切り捨てた値を返す関数である。関数を呼び出すときに渡す引数を実引数という。

関数が値を返さなくてもよい場合もある。次のプログラムを試してみよう。

```
osaka.awk:
1: {
2:   gsub("本当","ホンマ")
3:   print $0
4: }
```

`gsub()` という組み込み関数は、\$0の中の「本当」という文字列を「ホンマ」に置き換え、置き換えた数を返す(この値は利用していない)。実行すると下記ようになる¹¹。

```
H>jgawk -f osaka.awk
本当ですか      ... キーボードから入力
ホンマですか
本当にありがとう ... キーボードから入力
ホンマにありがとう
...^C
H>jgawk -f osaka.awk kiji.txt
...
H>
```

下表は関数呼び出しの具体例を示したものである。

関数呼び出し	関数の値
<code>sqrt(2)</code>	1.414
<code>int(3.14)</code>	3
<code>int(-3.14)</code>	-3
<code>jlength("弘前大学教育学部")</code>	8
<code>jsubstr("弘前大学教育学部",3,2)</code>	"大学"
<code>toupper("This")</code>	"THIS"
<code>tolower("This")</code>	"this"

(練習) 標準体重は

$$(\text{身長} - 100) \times 0.9$$

で計算できる。キーボードから身長を入力し、標準体重とローレル指数が「正常」となる体重の範囲を表示するプログラム `stdw.awk` を作りなさい。

¹¹実行時にデータファイルを指定しないと、キーボードからデータを入力するようになる。

(練習) `calc.awk` に、`sin(x)`(ただし x の単位は度)と `log10(x)` を表示するように付け加えなさい。なお、組み込み関数 `sin()` は引数をラジアンで与え、組み込み関数 `log()` は自然対数(底が e)なので注意。

```
H>jgawk -f calc.awk
x? 90
y? 2
90+2=92
...
sin 90=1
log 90=1.95424
H>
```

(練習) `gsub()` を使い、「しかし」と「そして」の後には必ず「、」が入るように文書を整形するプログラム `toten.awk` を作りなさい。「もしかしたら」など不都合が生じるとされる例をあげなさい。

(練習) `gsub()` を使い、「ですます体」の文書を「である体」に変換する整形プログラム `dearu.awk` を作りなさい。作成したプログラムでは不都合が生じるとされる例をあげなさい。

(練習) 次のように、`osaka.awk` を拡張すれば大阪弁翻訳プログラム(?)となる。

```
osaka.awk:
1: {
2:   gsub("ありがとうございました","おおきに")
3:   gsub("りますので","るさかいに")
4:   gsub("本当","ホンマ")
5:   gsub("大変な","エライ")
...
:   print $0
: }
```

標準語の日本語文章をあなたの出身地の方言に変換するプログラムを作りなさい。

2.3.6 関数の定義と呼び出し

ここではオリジナルの関数を定義してみよう。

`rohrer.awk` を次のように変更し、実行してみなさい。

```
rohrer.awk:
1: BEGIN{FS=","}
2: {
3:   x=rohrer($3,$4)
4:   print $1,int(x)
5: }
6: function rohrer(h,w){ return w*10000000/h^3 }
```

6行目が関数定義で、`rohrer` という関数は2つの値を変数 h と w に受け取り、計算結果を関数の戻り値として返す。 h と w を仮引数という。3行目では\$3(身長)と\$4(体重)の値が関数呼び出しの引数(実引数)となるが、例えば「`rohrer(178,127)`」のように具体的な値が関数に渡されることになる。これらの値は、6行目の関数定義の仮引数 h と w に代入される。その結果、関数は225.187の値を返し、3行目ではこれを受け取り x に代入している。仮引数の h や w は関数定義の中だけで有効な変数で、局所変数という。また、ある変数の有効範囲のことをスコープという。もしもこの関数定義の外で h や w などの変数が用いられたとしても、それは別の変数として扱われる。これに対し、変数 x はプログラム中のどこでも参照や代入ができ、広域変数という。

この例では関数を定義する効用は少ないが、この関数が何箇所かで利用されるような場合に有効である。

(練習) calc.awk で、「度」で引数を与える sin 関数 sindeg() と常用対数 log10() を関数定義するよう書き換えなさい。

(練習) 関数 int() を使い 1 の位を四捨五入する関数 int1(x) を定義し、名前と 1 の位を四捨五入した体重と身長を表示させなさい。

2.3.7 制御文 (if 文)

ここでは、条件により処理の内容を変えるプログラムを作ってみよう。

rohrer.awk を次のように変更し、実行してみなさい。

```
rohrer.awk:
1: BEGIN{FS=","}
2: {
3:   x=rohrer($3,$4)
4:   if(x>=160) print " ",$1,int(x)
5: }
6: function rohrer(h,w){ return w*10000000/h^3 }
```

4 行目は if 文で、

「x>=160 が真ならば print ... を実行しなさい」

と読み、「太りすぎ」の人だけが表示される。

更に rohrer.awk に次のように 1 行追加し実行してみなさい。

```
rohrer.awk:
1: BEGIN{FS=","}
2: {
3:   x=rohrer($3,$4)
4:   if(x>=160) print " ",$1,int(x)
5:   else print " ",$1,int(x)
6: }
7: ...
```

4 行目と 5 行目は

「x>=160 が真ならば print ... を実行しなさい。
そうでなければ print ... を実行しなさい。」

と読み、条件によって異なる文が実行される。結果、太りすぎの人には 印が付いた一覧表示が得られる。

if 下あるいは else 下の文が複数になる場合は { } でくくる。

if 文を用いた処理の流れをフローチャート (流れ図) で表したものが図 2.2 である。上記の rohrer.awk のフローチャートは図 2.3 となる。

上記の例では、if 文を使って処理を 2 つの場合に分けたが、細かく場合分けするにはどうしたらよいだろうか。ローレル指数の計算値によって (太りすぎ)、(太りぎみ)、(正常)、(やせぎみ)、(やせすぎ) のような印を付けてみよう。

印については既に正しく動作するので、「x>=160」が偽の場合 (else 下) について更に場合分けをすれば良い。

```
rohrer.awk:
3: ...
4:   if(x>=160) print " ",$1,int(x)
5:   else if(x>=145) print " ",$1,int(x)
6:   else print " ",$1,int(x)
7:   ...
```

5 行目の「if(x>=145) ...」は 4 行目の x>=160 が偽の場合に実行されるので、「print " ",...」は 145 ≤ x < 160 の時に実行される。6 行目の else 下は、「x>=160」が偽で更に「x>=145」が偽の時に実行される。次のように書くと理解しやすいかもしれない。

```
rohrer.awk:
3: ...
4:   if(x>=160){
5:     print " ",$1,int(x)
6:   }else{
7:     if(x>=145) print " ",$1,int(x)
8:     else print " ",$1,int(x)
9:   }...
```

以下、同様に考えれば良いので、結果、下記となる。

```
rohrer.awk:
1: BEGIN{FS=","}
2: {
3:   x=rohrer($3,$4)
4:   if(x>=160) print " ",$1,int(x)
5:   else if(x>=145) print " ",$1,int(x)
6:   else if(x>=115) print " ",$1,int(x)
7:   else if(x>=100) print " ",$1,int(x)
8:   else print " ",$1,int(x)
9: }
10: function rohrer(h,w){ return w*10000000/h^3 }
```

あるいは、印を求める関数 mark() を定義し、次のようにしてもよい。

```
rohrer.awk:
1: BEGIN{FS=","}
2: {
3:   x=rohrer($3,$4)
4:   print mark(x),$1,int(x)
5: }
6: function rohrer(h,w){ return w*10000000/h^3 }
7: function mark(x){
8:   if(x>=160) return " "
9:   else if(x>=145) return " "
10:  else if(x>=115) return " "
11:  else if(x>=100) return " "
12:  else return " "
13: }
```

このように、関数定義を使って機能を独立させることにより、プログラムの見通しが良くなり、将来プログラムを部分的に再利用できる可能性もある (例えばここで定義した mark() は次項の例題でも使用する)。

(練習) big.awk を変更して、全員の名前と体重を表示させ、ただしプログラム起動時に入力した値よりも体重が大きいデータには行頭に 印を付けなさい。

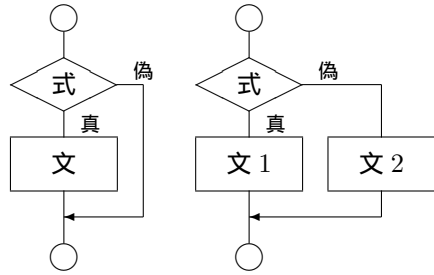
(練習) sumo.csv の体重の最大値を表示させるプログラム max.awk を作りなさい。ただし、2 つの引数の大きい値を返す関数 max(m,n) をプログラム中で定義し、下記のようにする。

```
max.awk:
1: BEGIN{FS=","}
2: { x=max(x,$4) }
3: END{print x}
4: function max(m,n){
5:   ...
```

また、名前も表示するにはどうしたらよいか考えなさい。

(練習) rohrer.awk を変更して、初めに身長と体重を入力し、ローレル指数と評価を表示するようにしなさい。

```
H>jgawk -f rohrer.awk
身長 (cm)? 154
```



(a) if(式) 文 (b) if(式) { 文 1 } else { 文 2 }

図 2.2 if 文のフローチャート

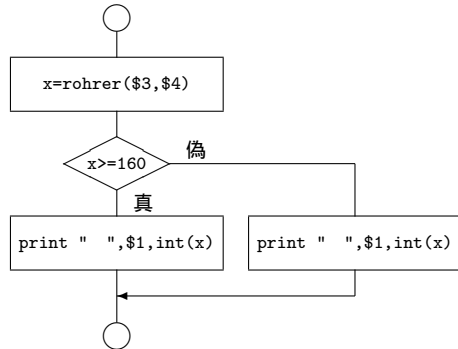


図 2.3 rohrer.awk のフローチャート

```

体重 (kg)? 56
ローレル指数は 153
太りすぎです
H>

```

(練習) calc.awk で x に 0 を入力すると exit 文で終了するようにしなさい。

2.3.8 制御文 (for 文と while 文)

for 文を用いて「繰り返し処理」を行ってみよう。
for 文は次のように書く。

```
for(式 1; 式 2; 式 3) 文
```

最初に「式 1」を評価 (実行) し、「式 2」が真である限り「文」を実行して「式 3」を評価 (実行) することを繰り返す (図 2.4(a))。

例で示そう。次のプログラムは初めに h に 150 を代入し、 $h \leq 185$ が真ならば「print h」を実行した後 h に 5 を加え、再び $h \leq 185$ を調べる。以下、 $h \leq 185$ が真である限りこれを繰り返す。この場合のフローチャートは図 2.5 となる。プログラムを入力し、実行してみなさい。

```

tab.awk:
1: BEGIN{
2:   for(h=150;h<=185;h+=5) print h
3: }

```

実行すると下記ようになる。

```

H>jgawk -f tab.awk
150
155
...
185
H>

```

while 文 (図 2.4(b)) を用いて次のようにしても結果は同じである。

```

tab.awk:
1: BEGIN{
2:   h=150
3:   while(h<=185){
4:     print h
5:     h+=5
6:   }
7: }

```

以下は、for 文を用いてローレル指数の表を作る例である (mark() の関数定義は 2.3.7 参照)。

```

tab.awk:
1: BEGIN{
2:   s="cm/kg"
3:   for(w=45;w<=70;w+=5) s=s " ¥t" w
4:   print s
5:   for(h=150;h<=185;h+=5){
6:     s=h
7:     for(w=45;w<=70;w+=5){
8:       x=rohrer(h,w)
9:       s=s " ¥t" mark(x) int(x)
10:    }
11:    print s
12:  }
13: }
14: function rohrer(h,w){ return w*1000000/h^3 }
15: function mark(...)
16: ...

```

実行すると下記ようになる。

```

H>jgawk -f tab.awk
cm/kg      45    50    55    60    65    70
150        133   148   162   177   192   207
155        120   134   147   161   174   187
160        109   122   134   146   158   170
...
H>

```

次のプログラムはデータファイルのデータを読んで表示する。

```

while.awk
1: {print $0}

```

これは、while 文と getline 文を用いて次のようにしてもよい。

```

while.awk
1: BEGIN{
2:   while(getline>0) print $0
3: }

```

getline 文は正しく 1 レコード読んだ時に正の値となり¹²、読んだデータは \$0, \$1, ... に代入される。もしもプログラム中でファイル名を特定したければ次のようにする。「<」は入力のリダイレクトの指示なので注意。

```

while.awk
1: BEGIN{
2:   while(getline <"sumo.csv" >0) print $0
3: }

```

while 文は、条件を満たさなければ 1 度も文を実行しないことがある。これに対し、do-while 文 (図 2.4(c)) は最初に文を実行し、その後に式を調べるので、少なくとも 1 度は文を実行する。例は省略。

for 文, while 文, do-while 文ではループの先頭または末尾で終了条件が調べられるが、ループの途中で条件を満たした場合に終了したい場合もある。以下はその例である。

¹² ファイルを読み終わると 0、エラーのときは -1 となる。

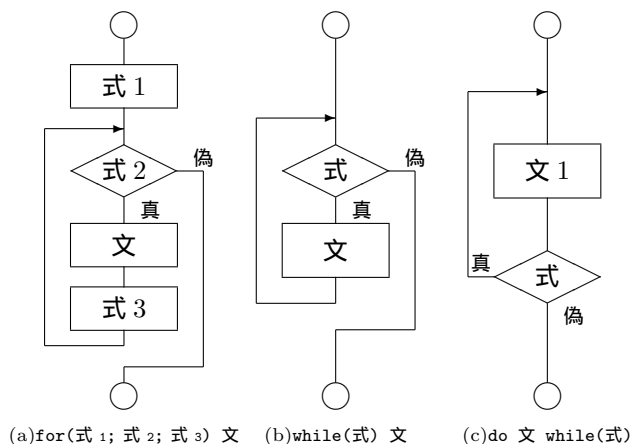


図 2.4 for 文と while 文のフローチャート

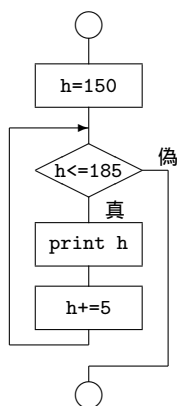


図 2.5 for(h=150;h<=185;h+=5) print h

```
BEGIN{
  while(1){
    printf "身長?"
    getline h
    if(h=="") break
  }
}
```

このようにすると「身長?」の表示に対して単に の操作をすると、ループから脱け出してプログラムを終了する。break は for や while や do のループからの脱出を指示する。

(練習) 身長と体重をキーボードから入力するようにした rohrer.awk(前項の練習) に上記の while(1){...if(...) break...}の仕組みを組み込みなさい。

(練習) calc.awk で、この動作を何度も繰り返すようにしなさい。また、x の入力を単に改行した場合にプログラムを終了するようにしなさい。

(練習) 半径に対する面積と体積の表を作りなさい。また、他の題材で数値表を作ってみなさい。

2.3.9 配列

同じ種類の大量のデータを変数に代入したい場合は、配列を用いる。

下記のプログラムは、全員の体重データを配列に格納した後に表示するものである。

```
array1.awk:
1: BEGIN{FS=","}
2: { w[NR]=$4 }
3: END{for(i=1;i<=NR;i++) print w[i]}
```

2 行目で w[1] に 127、w[2] に 105、...、w[22] に 145 がそれぞれ代入される。1~22 の添字 (index) で、配列 w のどの要素かを指定する。すべての体重データを配列 w に代入した後、3 行目で全データを表示している。実行すると下記ようになる。

```
H>jgawk -f array1.awk sumo.csv
127
105
...
H>
```

添字に文字列を使うこともできる (連想配列という)。下記のプログラムは、名前毎の体重データを配列に格納した後に、指定した名前の体重を表示するものである。

```
array2.awk:
1: BEGIN{FS=","}
2: { w[$1]=$4 }
3: END{
4:   while(1){
5:     printf "名前?"
6:     getline name <"con"
7:     if(name=="") break
8:     print w[name] "kg です"
9:   }
10:  print "お疲れさま"
11: }
```

2 行目で w["栃錦"] に 127、w["若乃花"] に 105、...、w["貴乃花"] に 145 がそれぞれ代入される。4~9 行目は while(1) となっているので、{ } 中の処理を何度でも繰り返す。5 行目で「名前?」と表示され、6 行目でキーボードから入力した値 (文字列) を変数 name に入れる。7 行目では、入力した文字列が空文字列の時 (改行のみ行った場合) に break で while(){ } から抜け出て、「お疲れさま」を表示して終了する。break の代わりに exit とすると、直ちにプログラムを終了する。実行すると下記ようになる。

```
H>jgawk -f array2.awk sumo.csv
名前?隆の里 ... キーボードから入力
155kg です
名前?北の湖 ... キーボードから入力
150kg です
名前? ... 改行のみで終了
お疲れさま
H>
```

今度は、出身地別に人数を表示するプログラムを示そう。

```
syussin.awk:
1: BEGIN{FS=","}
2: { p[$2]++ }
3: END{for(i in p) print i,p[i]}
```

2 行目で全レコードを読んだ時点で、p["鹿児島県"] に 1、p["東京都"] に 2、...、p["ハワイ"] に 1 が代入されている。3 行目の for 文では、i に配列 p の添字がひとつずつ代入され、「print ...」が繰り返される¹³。実行結果は下記。

¹³ただし、どういう順番で添字を取り出してくれるかはわからない。

```
H>jgawk -f syussin.awk sumo.csv
鹿児島県 1
東京都 2
...
ハワイ 1
H>
```

「どんな出身地が現れるかわからない」というような問題も簡単に解決できる。

多次元配列として使う場合は、`x[1,2]` のように添字を「,(カンマ)」で区切る。

(練習) キーボードから 10 個の数を入力し、それらを配列にしまい、最大値と最小値と平均値を表示させなさい。2.3.7 の練習で大きい方の値を返す関数 `max()` を作ったが、ここでは小さい方の値を返す関数 `min()` も定義しなさい。

2.3.10 再帰

上記で、出身地別に人数が表示されたが、これを棒グラフで表示させてみよう。AWK 言語にグラフィック機能は無いので、人数分だけ横に 印を並べることにする。プログラムは次のようにする。

```
syussin.awk:
1: BEGIN{FS=","}
2: { p[$2]++ }
3: END{
4:   for(i in p) printf "%-10s%s¥n",i,rep(" ",p[i])
5: }
6: function rep(a,n, i){
7:   while(n-- >0)   i=i a
8:   return i
9: }
```

ここで、関数 `rep(a,n)` は `a` という文字列を `n` 個繰り返した文字列を返すように定義している。`rep("A",4)` は "AAAA"、`rep("AB",3)` は "ABABAB" となる。ここで、関数定義の中で使用する変数 `i` を局所変数とするために、仮引数としている。この例では、変数 `i` は 4 行目でも用いているため、もしもこのように書いておかないと正常に動作しない(もちろん違う名前にすればよいが)。仮引数として記載した変数は関数定義の中だけで有効で、局所変数という。

7 行目で `while` 文を用いて `a` を `n` 個くっつけた文字列 `i` を作成し、8 行目で関数の値として返している。

```
H>jgawk -f syussin.awk sumo.csv
鹿児島県
東京都
長崎県
青森県
...
H>
```

さて、上記の関数定義でも良いのだが、次のように再帰的な関数定義に書き換えてみよう。

```
syussin.awk:
6: function rep(a,n){
7:   if(n==0) return ""
8:   else     return rep(a,n-1) a
9: }
```

これは、`rep("A",4)` として呼び出された場合、`rep("A",3)` と "A" をくっつけた文字列を返せばよいと考えるものであ

る。関数定義の中で定義中の自分自身を呼び出していて、このように考えることにより問題を単純にとらえることができる場合がある。このように、より簡単な問題に分割する方法を一般に分割統治という(3.4.1 参照)。

なお、`if` 文を用いる代わりに条件式を用いて次のようにしてもよい。

```
syussin.awk:
6: function rep(a,n){
7:   return (n==0)?"":(rep(a,n-1) a)
8: }
```

条件式は「式₁?式₂:式₃」のように書き、式₁ が真なら式₂ の値を返し、偽なら式₃ の値を返す。

2.3.11 正規表現

文字パターンによるデータ検索を行うには正規表現を用いる。

`sumo.csv` から青森県出身者のデータを抽出するには次のようにすればよい。

```
search.awk:
1: BEGIN{FS=","}
2: $2=="青森県"{print $0}
```

ここで、次のようにしても結果は同じである。

```
search.awk:
1: BEGIN{FS=","}
2: $2~/青森県/{print $0}
```

これは、「2 番目のフィールドに青森県という文字列が含まれているレコードを表示しなさい」という意味である。こうすることにより、出身地の記載が「青森県弘前市」や「日本国青森県中津軽郡」などとなっても「\$2~/青森県/」は真となる。

もっと複雑な文字パターンの表現もできる。

```
search.awk:
1: BEGIN{FS=","}
2: $1~/山/{print $0}
```

は「名前のどこかに山という文字が含まれていれば……」だが、

```
search.awk:
1: BEGIN{FS=","}
2: $1~/山$/
```

とすると「名前の最後の文字が『山』であれば……」になる。「/山\$/」の「\$」は「文字列の最後」を意味する。

```
search.awk:
1: BEGIN{FS=","}
2: $1~/^北/
```

とすると「名前の先頭文字が『北』であれば……」になる。「/^北/」の「^」は「文字列の先頭」を意味する。

「\$1~/ /」は「1 番目のフィールドが「/ /」で表されるような文字列であれば…」と読む。「/ /」で囲まれた部分は「正規表現」と呼ばれ、「欲しい情報」を的確にコンピュータに伝え、たくさんのデータの中からその情報を探し出すのに便利である。

少し複雑な例を示そう。

```
search.awk:
1: BEGIN{FS=","}
2: $1~/[のノ乃].$/ {print $0}
```

これは、「名前が『の』『ノ』『乃』のいずれかを含み、そのあとが1文字で終わるデータを表示しなさい」というものである。「[]」は、その中のどれか1文字、「.」は「任意の1文字」を表す。

下記は、名前が3文字で2文字目が「の」の人のデータを表示するプログラムである。

```
search.awk:
1: BEGIN{FS=","}
2: $1~/^.の.$/ {print $0}
```

このように、複雑な文字パターンでも簡潔な表示が可能である。

(練習) 名前が「海」で終わる人と「山」で終わる人がそれぞれ何人いるか調べなさい。

(練習) 北海道出身で名前のどこかに「北」がつく人のデータを表示しなさい。

(練習) world.csv というファイルには、世界の193ヶ国について

国名, 地域, 英語名, 首都, 人口(万人), 面積(万 km²), 総生産(億ドル), 備考

のデータが入っている。

- (1) エディタや表計算ソフトでこのファイルを開いて中を見てください。
- (2) 次のプログラムは、「人口密度を計算し、300人/km²以上の国の名前と人口密度を表示する」というものである。

```
world.awk:
1: BEGIN{FS=","}
2: {
3:     x=$5/$6
4:     if(x>=300) print $1,x
5: }
```

これを入力し、実行してみなさい。

- (3) その他、各自関心のある事項について world.awk の内容を変更し、実行しなさい。

(練習) hyaku.csv というファイルには

百人一首の上句, 下句, 作者, 作者略歴, 動物, 草木, 地名, 自然

のデータが入っている。

- (1) エディタや表計算ソフトでこのファイルを開いて中を見てください。
- (2) 次のプログラムは、作者のどこかに「大臣」という文字列が含まれていれば、上句と下句と作者を表示するというものである。

```
hyaku.awk:
1: BEGIN{FS=","}
2: {
3:     if($3~/大臣/) print $1,$2,$3
4: }
```

これを入力し、実行してみなさい。

- (3) プログラム中の「大臣」の部分、プログラム起動時に入力できるようにしなさい。
- (4) その他、各自関心のある事項について hyaku.awk の内容を変更し、実行しなさい。

2.4 ライブラリ

プログラム作りを楽しむのでない限り、プログラムを作る努力は最小限にすべきである。既に作ったプログラムが部分的にでも再利用できるならばそれにこしたことはない。何度も同じことに労力を費やすのは賢明とは言えない。プログラムを作るときには、なるべく再利用できるように作ることを心がけたい。

max(), min(), rep() などの関数や、文字の色を切り換える特殊な文字列や、 の値など、「また利用できそうなもの」は、ライブラリに登録しておくことと便利である。組み込み変数の定義は BEGIN{ } の中に書く。演習用ディスクでは付録 C に示したライブラリ (lib.awk) が利用でき、この中に rep() が定義されている。2.3.10 の syussin.awk の rep() の関数定義を削除し、下記とする。

```
syussin.awk:
1: BEGIN{FS=","}
2: { p[$2]++ }
3: END{
4:     for(i in p) printf "%-10s%$n",i,rep(" ",p[i])
5: }
```

実行時に次のようにライブラリファイルを指定する。

```
H>jgawk -f lib.awk -f syussin.awk sumo.csv
鹿児島県
東京都
...
H>
```

2.5 Windows での利用

AWK 言語は Windows 環境でも利用できるし、Mac 版の AWK もある。

次に、「しばしば再利用したい有用なプログラム」を作ったとしよう。Windows 環境ならば、Windows らしく利用できるようにしたい。それが、ある程度は可能である。以下に例を示す。

```
makehtm.awk:
1: BEGIN{
2:     FS=","
3:     print "<table border>"
4:     print "<tr><th>名前</th><th>出身地</th><th>身長</th><th>体重</th></tr>"
5: }
6: { print "<tr><td>" $1 "</td><td>" $2 "</td><td>" $3 "</td><td>" $4 "</td></tr>" }
7: END{ print "</table>" }
```

```
makehtm.bat:
1: jgawk -f makehtm.awk sumo.csv>sumo.htm
2: notepad sumo.htm
```

以上は、sumo.csv のデータを処理して、Web ページファイル sumo.htm を作成するプログラムである。実行はキーボードから

```
H>makehtm
```

とすると、sumo.htm が作られ、メモ帳で開かれる (メモ帳で開く部分は余分かもしれない)。

Windows で makehtm.bat のアイコンをダブルクリックしてもよい。新たに MS-DOS プロンプトのウィンド

ウが開き、これをマウス操作で閉じないといけないが、makehtm.bat のアイコンを右クリックし、プロパティを選び、[プログラム] タブの実行時の大きさを [最小化の状態]、[プログラム終了時にウィンドウを閉じる] をチェックしておく、MS-DOS ウィンドウが現れることはなく、まるで Windows 版プログラムと変わらない (残念ながら「対話が必要なプログラム」ではこうはいかない)。

同じプログラムで様々なデータファイル进行处理したい場合もあるだろう (例えば、クラス毎に上記のような Web ページを作るなど)。その場合は makehtm.bat の 1 行目のデータファイルの箇所 (sumo.csv) を「%1」にしておく、すると、

```
H>makehtm sumo.csv
```

のようにキーボードでデータファイル名を指定、またはマウスでデータファイルのアイコンを makehtm.bat のアイコンにドラッグ&ドロップ操作すればよい。

以上のように作られた sumo.htm のアイコンをダブルクリックすれば Web ブラウザが起動し、目的の表が表示される。もちろん Web サーバで公開するためであっても良いが、「文書清書」の目的で使用する価値もある。WYSIWIG 全盛の中で、マークアップ方式の特徴を生かした有効な方法である。

文書清書 (加工) の他の例として、「電子メールでの会合の参加申し込み」をあげておこう。

```
... 下記のとおり参加を申し込みます。
%%氏名%%:
%%所属%%:
%%メールアドレス%%:
...
```

のような「申し込み用紙」の書式を決める。「%%氏名%%:」などはキーワードで、キーワードに続けて必要事項を記載して返信してもらう。返信メールを受け取ったらこれから 1 件のレコード (1 行データ) を生成し、データファイルに追加する。

(練習) 身近な題材で「申し込み用紙」の書式を決め、電子メールで出す案内文書を作りなさい。返信メールで届いた複数の申し込み文書を加工し、CSV 形式のデータファイルを生成するプログラムを作りなさい。CSV ファイルができたらそのアイコンをダブルクリックし、Excel で開いてみなさい。

2.6 AWK 言語のまとめ

AWK プログラム	
パターン{アクション}	
...	
関数定義	
...	

パターン	
BEGIN	レコードの処理前に 1 度適合
END	レコードの処理終了後に 1 度適合
式	\$4>=150, \$1~/山\$/など
/正規表現/	\$0~/正規表現/の省略形。正規表現については 2.3.11 参照
パターン 1&&パターン 2	パターン 1 とパターン 2 の論理積 (AND)
パターン 1 パターン 2	パターン 1 とパターン 2 の論理和 (OR)
!パターン (パターン)	パターンの否定 (NOT)
パターン 1, パターン 2	グループ化
	パターン 1 に適合してからパターン 2 に適合するまで

アクション	
文の並び	
文	
式	x=0, 関数呼び出しなど
制御文	if 文、for 文など
入出力文	print 文、printf 文、getline 文など
{文の並び}	

入出力文	
getline	次のレコードを「\$0」にセット
getline <file	file の次のレコードを「\$0」にセット
getline 変数	次のレコードを変数にセット。NR と FNR もセット
getline 変数 <file	file の次のレコードを変数にセット
print	現在レコード\$0 を表示
print 式の並び	式の並びを表示
print 式の並び >file	式の並びを file に保存
printf 書式, 式の並び	式の並びを書式に従って表示
printf 書式, 式の並び >file	式の並びを書式に従って保存
system コマンド	MS-DOS のコマンドを実行 (エラーレベルを返す)
close 式	入出力を閉じる

制御文	
if(式) 文	式が真なら文を実行
if(式) 文 1 else 文 2	式が真なら文 1 を実行、偽なら文 2 を実行
while(式) 文	式が真である限り文を繰返し実行
do 文 while(式)	文を実行し、式が真である限り文の実行を繰返す
for(式 1; 式 2; 式 3) 文	初めに式 1 を評価し、式 2 が真である限り文を実行しては式 3 を評価することを繰返す
for(変数 in 配列) 文	配列の添字をひとつずつ取り出しては変数に代入することをすべての添字について繰返す
break	for や while のループから抜け出す
continue	for や while のループ中の以後の処理をやめ、次のループに進む
next	
exit	プログラムを終了する
exit 式	式をエラーコードとしてプログラムを終了する
return	関数から戻る
return 式	式を値として関数から戻る
delete 配列要素	配列要素を削除する

式	
定数	数値定数または文字列定数
変数	数値か文字列
フィールド変数	\$0, \$1, ..., \$NF
関数呼び出し	
配列要素 (式)	優先する演算の明示
以下の演算子で式を結合したもの	
代入演算子	/= += -= *= /= %= ^=
条件式演算子	式 1 ? 式 2 : 式 3
論理演算子	(論理和:OR) && (論理積:AND) !(否定:NOT)
照合演算子	~ !~
関係演算子	< <= == != > >=
接続演算子	文字列の接続演算 (演算子は明示されない)
算術演算子 (単項) 算術演算子	+ - * / % ^
インクリメント演算子	++(前置と後置)
デクリメント演算子	--(前置と後置)

組み込み変数	
ARGC	コマンド行の引数の数
ARGV []	コマンド行の引数の配列
FILENAME	現在の入力ファイル名
ENVIRON["..."]	環境変数の値
FS	入力フィールド区切り文字 (初めはスペースまたはタブ)
RS	入力のレコード区切り文字 (初めは改行)
NF	現在レコードのフィールド数
NR	現在の通算レコード
FNR	現在の入力ファイルの通算レコード
OFFS	表示のフィールド区切り文字 (初めはスペース)
ORS	表示のレコード区切り文字 (初めは改行)
OPMT	数の表示の書式 (初めは「%.6g」)
RSTART	match で適合した文字列の開始位置
RLENGTH	match で適合した文字列の長さ
\$0	現在の入力レコード

フィールド変数	
\$1, ..., \$NF	第 1 フィールド, ..., 第 NF フィールド

組み込み関数	
atan2(y,x)	atan(y/x) で - ~ の値 .
sin(x)	sin 関数
cos(x)	cos 関数
exp(x)	exp 関数
log(x)	自然対数
sqrt(x)	平方根
int(x)	小数点以下を切り捨て
rand()	疑似乱数 0 以上 1 未満の一樣分布
srand()	乱数の初期化 (srand() を呼び出すと以後 rand() は毎回異なる乱数系列を生成する)
gsub(r,s,t)	文字列 t の中に現れる文字列 r をすべて文字列 s で置換し、置換した数を返す。t を省略すると \$0 が使われる。
index(s,t)	文字列 s 中の文字列 t の位置。t が現れない場合は 0。
jindex(s,t)	日本語文字列 s 中の文字列 t の位置。t が現れない場合は 0。
length(s)	文字列 s の長さ
jlength(s)	日本語文字列 s の長さ
toupper(s)	文字列 s を大文字にした文字列
tolower(s)	文字列 s を小文字にした文字列
match(s,r)	文字列 s が文字列 r に適合する位置。適合しないときは 0。
split(s,a,fs)	fs をフィールド区切り文字として文字列 s を配列 a に分解し、フィールド数を返す。
sprintf(書式, 式)	書式で整えた式の並び
sub(r,s,t)	はじめの 1 回だけ置換する他、gsub() と同様。
substr(s,i,n)	文字列 s の i 番目から始まる n 文字
jsubstr(s,i,n)	日本語文字列 s の i 番目から始まる n 文字
system()	現在の時刻
strftime()	時刻情報からの書式変換

printf/sprintf 書式変換	
%c	ASCII 文字
%d	10 進数
%e	[-]d.dddddE[+-]
%f	[-]ddd.ddddd
%g	e 変換と f 変換の短い方で無意味な 0 を表示しない
%o	符号なし 8 進数
%s	文字列
%x	符号なし 16 進数
%%	%そのもの

正規表現	
A	A そのもの (以下の特殊文字を除く普通の文字)
¥ ¥	¥ そのもの
¥ "	" そのもの
¥ t	タブ
¥ n	改行
¥ f	フォームフィード
¥ b	バックスペース
¥ 033	8 進数 033
^ a	文字列の先頭が a
a \$	文字列の末尾が a
.	任意の 1 文字
[abc]	abc のどれか 1 文字
[^abc]	abc 以外の 1 文字
[a-e]	abcde のどれか 1 文字
[^a-e]	abcde 以外の 1 文字
a*	0 個以上の a の並び
a+	1 個以上の a の並び
a?	a が 1 個あるいは 0 個
abc de	abc または「de」

課題

各自関心のある事柄について、「名称と 2~3 の数値」で構成されるデータファイルを作りなさい。また、big.awk を参考に、このデータを処理する簡単なプログラムを作成し、実行しなさい。内容は問わないが、その処理が「自分にとって意味のあるもの」であること。レポートには、プログラムのねらい、データの構造、動作の概要などをまとめなさい。提出日、タイトル、学籍番号、名前を冒頭に記載し、プログラムとデータは付録としなさい。印刷して提出できるようなテキストファイル (学籍番号.txt) として作成し、これをメール本文に読み込んで送りなさい。

第3章 AWK プログラミング・応用編

入門編では例題を中心に AWK プログラミングの基礎を学んだ。本章では、さまざまな題材をとりあげ、プログラミングの応用例を示す。興味のある箇所を読み（あるいは試し）、各自オリジナルのプログラム（課題）にチャレンジしていただきたい。

3.1 対話型学習ソフト

3.1.1 記述式クイズ

以下の pair.awk は「対の関係」を記述式で問うプログラムである。データを変えれば、「日本語を示し、英単語を問う問題」、「英単語を示し、逆の意味の英単語を問う問題」、「漢字を示し、読みを問う問題」などとなる。

(プログラム)

```
pair.awk:
1: BEGIN{FS=","}
2: {
3:   printf "問" NR ": " $1 " は何ですか?"
4:   getline s<"con"
5:   if(s==$2) print " 正解です"
6:   else      print "× 違います 正解:" $2
7: }
```

(データ)

```
pair.csv:
1: 愛,love
2: 飛行機,airplane
3: 夢,dream
4: 猿,monkey
5: 休日,holiday
6: 学校,school
```

(実行例)

```
H>jgawk -f pair.awk pair.csv
問 1: 愛 は何ですか? love
      正解です
問 2: 飛行機 は何ですか? plane
      × 違います 正解: airplane
問 3: 猿 は何ですか? monkey
      正解です
...
```

(練習) pair.awk のプログラムを変更し、全問終了後に正答数を表示するようにしなさい。

3.1.2 穴埋め問題

(プログラム)

```
subst.awk:
1: BEGIN{FS=","}
2: {
3:   print "問" NR ": " $1
4:   for(i=2;i<=NF;i++){
5:     printf "(" i-1 " ) は何ですか?"
```

```
6:     getline a<"con"
7:     if($i==a) print " 正解です"
8:     else      print "× 違います 正解:" $i
9:   }
10: }
```

(データ)

```
subst.csv:
1: 夢 (1) みる, を
2: 猿 (1) 木 (2) 落ちる, も, から
3: 温 (1) 知 (2), 故, 新
4: (1)+(2)=12 (1)-(2)=6,9,3
5: (1) なぜなくの (1) は (2) に~, からず, 山
```

(実行例)

```
H>jgawk -f lib.awk -f subst.awk subst.csv
問 1: 夢 (1) みる
(1) は何ですか? を
      正解です
問 2: 猿 (1) 木 (2) 落ちる
(1) は何ですか? も
      正解です
(2) は何ですか? に
      × 違います 正解: から
...
```

(練習) pair.awk のプログラムを変更し、データファイルの構成を、「問題, 解答 1, 解答 2,...」のようにし、2 つ以上の解答を許すようにしなさい。

(練習) subst.awk を間違った箇所だけを赤で表示するようにしなさい。

(練習) 以下のように、各問複数の項目を問うプログラムを作りなさい。

```
問 1
1221 年は何がありましたか? 承久の乱
      正解です
その中心人物は誰ですか? 北条政子
      × 違います 正解: 北条義時
問 2
1637 年は何がありましたか? 島原の乱
      正解です
その中心人物は誰ですか? 天草四郎
      正解です
...
```

(練習) 選択式のクイズプログラムを作りなさい。

3.1.3 CAI プログラム

少し実用性を考慮した CAI プログラムを紹介しよう¹。問題ファイル (cai.csv) の内容は次のように扱われ、問題文を自由記述できる。

- 「A:」で始まる行は、\$2 を尋ねて答えを入力させ、正解 (\$3) との一致を調べて表示する (3~5 行目)。
- 空行 (改行だけの行) では「次の問題 (Enter キー)」と表示してキー操作を促す (8~9 行目)。
- 上記以外の行はそのまま表示される (10 行目)。

なお、多少拡張したプログラムが A: ¥sample にある²

¹三上修氏が作成した AWKCAI というフリーソフトを参考にして作成したもので、題材も一部引用している。

² H>cd ¥sample

```
H>jgawk -f lib.awk -f cai.awk qanda.csv
H>jgawk -f lib.awk -f cai.awk momo.csv
```

で試すことができる。誤答の場合にヒントを出したり、全問回答後に回答記録を表示したり (実際はファイルに記録して後から分析する)、機能を拡張している。

(プログラム)

```
cai.awk:
1: BEGIN{FS=","}
2: {
3:   if($1=="A:"){
4:     printf $2 "は何ですか? "
5:     getline a<"con"
6:     if(a=="3") print " 正解です"
7:     else print "x 違います 正解:" $3
8:   }else if($0==""){
9:     printf "次の問題 (Enter キー)"
10:    getline <"con"
11:   }else print $0
12: }
```

(データ)

```
cai.csv:
1: 第1問 次の日本語の意味を表す語を ( ) に書きなさい。
2: (1) 経済 ( ) (2) 困難 ( )
3: A:,(1),economy
4: A:,(2),difficulty
5:
6: 第2問 次の各語で最も強く読む部分を番号で答えなさい。
7: (1) ex-pe-ri-ence ( ) (2) en-vi-ron-ment ( )
8: 1 2 3 4 1 2 3 4
9: A:,(1),2
10: A:,(2),2
11:
12: 第3問 ( ) に入ると思われる英単語を答えなさい。
13: 桃太郎さん 桃太郎さん -MOMOTAROSAN, MOMOTAROSAN,
14: お腰につけた吉備団子 give me one of your KIBIDANGO,
15: 一つわたしにくださいな . which you carry with you
16: あげましょうあげましょう around your ( 1 ).
17: これから鬼の征伐について -I will give you one if you
18: くるならあげましょう . will come with me to get rid
19: of the ( 2 ).
20: A:,(1),waist
21: A:,(2),devils
```

(実行例)

```
H>jgawk -f cai.awk cai.csv
第1問 次の日本語の意味を表す語を ( ) に書きなさい。
(1) 経済 ( ) (2) 困難 ( )
(1)は何ですか? economy
正解です。
(2)は何ですか?
...
```

(練習) データの構成を「問題、解答、ヒント」とし、誤答の場合はヒントを出すようなプログラムを考えなさい。

(練習) 誤答の場合は補足問題を解かせるようなプログラムを考えなさい。

(練習) 解答の正誤で次に出す問題の難易度を変えるようなプログラムを考えなさい。

3.2 教材分析 (英語)

3.2.1 頻出単語

以下の count.awk は、大小文字の区別なしに、3回以上現れる単語を調べるプログラムである。2行目は tolower() で大文字を全部小文字に変換し、3行目は gsub() でアルファベット以外の数字や記号を除く処理をしている。4行目は単語のカウント。集計終了後の8行目で3回以上現れた単語を頻度順に表示している。

(プログラム)

```
count.awk:
1: {
2:   $0=tolower($0)
3:   gsub(/[^\a-z] */, " ")
```

```
4:   for(i=1;i<=NF;i++) n[$i]++
5: }
6: END{
7:   for(w in n){
8:     if(n[w]>=3) print n[w],w | "sort /r /n"
9:   }
10: }
```

(実行例)

```
H>jgawk -f count.awk eigo.txt
6 and
5 a
4 is
4 the
4 lines
4 of
3 awk
3 to
H>
```

3.2.2 文字パターン検索

以下の re.awk は、入力した正規表現にマッチする単語を探して表示するプログラムである。

(プログラム)

```
re.awk:
1: BEGIN{
2:   printf "string? "
3:   getline re<"con"
4:   while(getline>0){
5:     $0=tolower($0)
6:     gsub(/[^\a-z] */, " ")
7:     for(i=1;i<=NF;i++) if($i~re) n[$i]++
8:   }
9:   for(w in n) print n[w],w
10: }
```

(実行例)

```
H>jgawk -f re.awk eigo.txt
string? ^..t
1 set
2 csva
1 with
2 action
1 pattern
1 often
1 matching
1 matched
2 actions
1 but
2 patterns
```

「^..t」は「3文字目が t の単語を探しなさい」という意味である。「e\$」とすれば「最後が e で終わる単語」、「^t.*t\$」とすれば「t で始まって t で終わる単語」ということになる。いろいろ試してみなさい。

3.2.3 可読性 (読みやすさ)

ARI Readability Score は次式で与えられる³。

$$G = 4.71 \text{ lpw} + 0.5 \text{ wps} - 21.43$$

lpw: 1 単語の平均文字数
wps: 1 文の平均単語数

英語文書が与えられたとき、次のプログラムは、段落や文章や単語の数、1段落の平均文数、1文の平均単語数、1単語の平均文字数、そして ARI スコアを計算し、表示する。

³L.L.Cherry, W.Vesterman: "Writing Tools - The STYLE and DICTION Programs", UNIX Document, USD-32.

(プログラム)

```
readable.awk:
1: {
2:   gsub(/[^\A-Za-z.\-]*/, " ")
3:   ns+=gsub(/\./, " ")
4:   nw+=NF
5:   np++
6:   for(i=1;i<=NF;i++) nl+=length($i)
7: }
8: END{
9:   wps=nw/ns
10:  lpw=nl/nw
11:  spp=ns/np
12:  print "No. paragraphs: " np
13:  print "No. sentences: " ns
14:  print "No. words:" nw
15:  printf "Avg. paragraph length: " spp " sentences"
16:  printf "Avg. sentence length: " wps " words"
17:  printf "Avg. word length: " lpw " letters"
18:  printf "ARI Grade: " (4.71*lpw+.5*wps-21.43)
19: }
```

(実行例)

```
H>jgawk -f readable.awk eigo.txt
No. paragraphs: 1
No. sentences: 5
No. words: 125
Avg. paragraph length: 5.0 sentences
Avg. sentence length: 25.0 words
Avg. word length: 4.7 letters
ARI Grade: 13.3
```

(練習) 英語文書が与えられたときに、単語の長さによるヒストグラムを表示するプログラムを作りなさい。

(練習) 日本語文書が与えられたときに、40文字以上「、」が現れない文章および80文字以上「。」が現れない文章を表示し、警告を与えるプログラムを作りなさい。

(練習) テキスト中の指定した文字列を赤色で表示するプログラムを作りなさい。赤色で表示するには、

```
print RED 文字列 NORMAL
```

のように赤色にしたい文字列の前後に RED と NORMAL を付ける。これらは lib.awk で定義されている。

(練習) 疑問文や感嘆文には対応していません。これらに対応できるように拡張しなさい。

(練習) 日本語の可読性の尺度として、次式が提案されている⁴。例えば、小学3年レベルの文章ならば G が 3、中学3年レベルの文章ならば G が 9 と評価される。

$$G = -0.17h - 0.28k - 3.49e + 27.62$$

h: ひらがなの相対頻度 (%)
k: カタカナの相対頻度 (%)
e: 句点の相対頻度 (%)

これに従って、日本語文書の可読性を調べるプログラムを作りなさい。

3.3 シミュレーション

教育用としてのシミュレーションソフトは、実物を観察できないようなとき (例えば自動車のエンジンの動作や化学反応など) や、概念的なものを学習するとき (例えば筆算の桁上がりの概念など) に効果的で、アニメーションがしばしば用いられる。

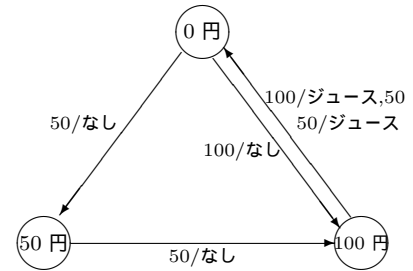


図 3.1 自動販売機の状態遷移図

ここでは、「有限状態機械」で自動販売機を作ってみよう (このプログラムは文献 [4] の 2.2 を参考にした)。以下の 3 つの状態からなる「50 円玉と 100 円玉が使える 150 円のジュースの自動販売機」を考える。入力 / 出力の関係と状態遷移を図 3.1 に示す。

0 円: 自動販売機にお金が入っていない状態
50 円: 自動販売機に 50 円入っている状態
100 円: 自動販売機に 100 円入っている状態

データファイル juice.csv は、この図を表現したもので、1 番目のフィールドが現在の状態、2 番目のフィールドが入力、3 番目のフィールドが次の状態、4 番目のフィールドが出力を意味している。このデータファイルを変更すれば、さまざまなシミュレーションが可能になる。いわゆるシミュレーションゲームのようなものも作ることができる。

(プログラム)

```
fsm.awk:
1: BEGIN{FS=","}
2: {
3:   trans[$1,$2]=$3
4:   out[$1,$2]=$4
5:   if(NR==1) s=$1
6: }
7: END{
8:   for(;;){
9:     print "状態: " s
10:    printf "入力: "
11:    getline x<"con"
12:    if(x=="") break
13:    if(trans[s,x]=="") print x "は使えません!"
14:    else{
15:      print "出力: " out[s,x]
16:      s=trans[s,x]
17:    }
18:  }
19: }
```

(データ)

```
juice.csv:
1: 0,50,50
2: 50,50,100
3: 100,50,0,Juice
4: 0,100,100
5: 50,100,0,Juice
6: 100,100,0,Juice+50
```

(実行例)

```
H>jgawk -f fsm.awk juice.csv
状態: 0
入力: 50      ... キーボードから入力
出力:
状態: 50
入力: 50      ... キーボードから入力
出力:
状態: 100
入力: 100     ... キーボードから入力
```

⁴浅野陽子, 小川克彦:「日本文の可読性を尺度とした読みスタイルの分析」, 情報処理学会ヒューマンインタフェース研究会資料, 37-7, 1991.

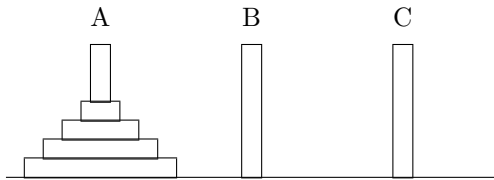


図 3.2 ハノイの塔 (A にある円盤をすべて B に移したい)

```
出力: Juice+50
状態: 0
入力:      ... 改行で終了
H>
```

このプログラムを核に、絵と音で脚色し、附属養護学校で使われた。

(練習) 10 円玉も使えるように、juice.csv を変更しなさい。

3.4 パズルとゲーム

パズルとゲームのプログラムを取り上げる。どちらもコンピュータが問題を解いているように見えるかもしれないが、解法はプログラムとして与えていることに注意。

3.4.1 ハノイの塔

3本の棒 A,B,C があり、Aの棒に n 枚 (図では4枚) の円盤が積み重ねられている (図 3.2)。これらの円盤を B の棒に移したい。どのような手順で移動したらよいか。ただし、円盤は1度に1枚ずつ移動し、小さな円盤の上に大きな円盤を重ねることはできない。

(プログラム)

```
hanoi.awk:
1: BEGIN{
2:   hanoi(4,"A","B","C")
3: }
4: function hanoi(n,from,to,buf){
5:   if(n>0){
6:     hanoi(n-1,from,buf,to)
7:     print from " " to
8:     hanoi(n-1,buf,to,from)
9:   }
10: }
```

(実行例)

```
H>jgawk -f hanoi.awk
A C
A B
C B
A C
B A
...
```

6行目で from(A) の $n-1$ 枚の円盤を buf(C) に移し、7行目で from(A) に残った1枚の円盤を to(B) に移し、8行目で buf(C) の $n-1$ 枚の円盤を to(B) に移す。6行目と8行目は再帰呼び出しになっている。このように「より簡単な問題に分解して問題を解決する方法」を分割統治という。

2行目の枚数の指定を初めは1枚にしてプログラムを実行し、続いて2枚、3枚、4枚,... のように増やして実行し、表示結果からプログラムの動作を考えなさい。

3.4.2 石取りゲーム

10個の石がある。交互に1~3個の石を取り、最後に石を取ったら負けというゲームである⁵。

(プログラム)

```
ishi.awk:
1: BEGIN{
2:   N=10
3:   M=3
4:   print "1~" M "個の石を取ることができます"
5:   print "最後に石を取った方が負けだよ"
6:   print rep(" ",N)
7:   while(1){
8:     printf "あなた: 何個取りますか? "
9:     getline x
10:    print rep(" ",N-x)
11:    if(N<=0){ print "あなたの負け"; break}
12:    x=(N-1)%(M+1)
13:    if(x==0) x=1
14:    print "コンピュータ: " x "個の石を取ります"
15:    print rep(" ",N-x)
16:    if(N<=0){ print "コンピュータの負け"; break}
17:  }
18: }
19: function rep(s,n){
20:   if(n==0) return ""
21:   else return s rep(s,n-1)
22: }
```

(実行例)

```
H>jgawk -f ishi.awk
1~3個の石を取ることができます
最後に石を取った方が負けだよ
```

```
あなた: 何個取りますか? 3
```

```
コンピュータ: 2個の石を取ります
```

```
あなた: 何個取りますか?
```

```
...
```

コンピュータの戦略は11~12行目にある。相手が1個取ったらこちらは3個... というように4個ずつ確実に減らすことができるので、4の倍数+1個残せば勝ちが決まる。それがだめなら1個取る。

(練習) 石の数 (N) や、取ることができる最大個数 (M) を変えても勝つことができるかどうか試みなさい。

課題

これまで学んだことを参考に、各自の専攻と関連して「誰かに使ってもらう応用プログラム」を作りなさい。作ったプログラムを誰かに使ってもらい、感想を聞きなさい。レポートには、プログラムのねらい、動作の概要、工夫したところ、使ってもらってどうだったかなどをまとめなさい。提出日、タイトル、学籍番号、名前を冒頭に記載し、プログラムリストは付録としなさい。印刷して提出できるようなテキストファイル (学籍番号.txt) として作成し、メールに添付しなさい。

⁵奥村晴彦「C言語による最新アルゴリズム事典 (技術評論社)」