第 3 章

テキストファイルとその扱い方

前章まで扱ってきたファイルはすべて「テキストファイル」であった。ファイルの実態は単なるデータの塊であるから，コンピュータにとってはどんなファイルも区別がない。しかし，人間が読んで理解できる文字 (に割り当てられたデータ）だけで構成されたファイルは「文字の集合」として解釈できる。このようなファイルを「テキストファイル」，それ以外のファイル (画像，動画ファイル等）を「バイナリファイル」(binary file) と呼ぶ。本章ではこのテキストファイルをあれこれ操作して UNIX の CUI に慣れるための訓練を行う。

3.1　文字コード = 文字集合 + 符号化方式

文字コードとは，文字集合（使用可能な文字の集まり）に対してどのような整数（2 進数を bit 列で表現) の対応付けを行うか（符号化方法）を定めた方式のことを言う。文字コードが異なると「文字化け」が生じる (図 3.1) が，これは文字コード同士で同じもでも異なる整数を割り当ててているために起きる。異なる文字コード体系では，bit 列との対応付けが異なり，同じ bit 列でも違う文字が割り当てられるからである。

現在使われている主な文字コードは下記のものがある。1 Byte=8 bit を単位として文字を割り当てる方式が主流である。

■1 Byte コード・・・ASCII, JIS X 0201　ASCII(American Standard Code for Information Interchange) コードは 7bit で英語で使用されるアルファベット，数字，その他の記号（スペース，タブ，カンマ，セミコロン (;)，コロン (:)，シングルコーテーション (’)，ダブルコーテーション (") 等) といった文字集合に 7bit の符号化方式を定めたもの。日本ではかつて「半角英数字」とも呼ばれていたが，「半角」「全角」というのはフォント (文字の形態) の横幅を指す言葉であるので，現在のように様々なフォントを混ぜ込んで使うよう

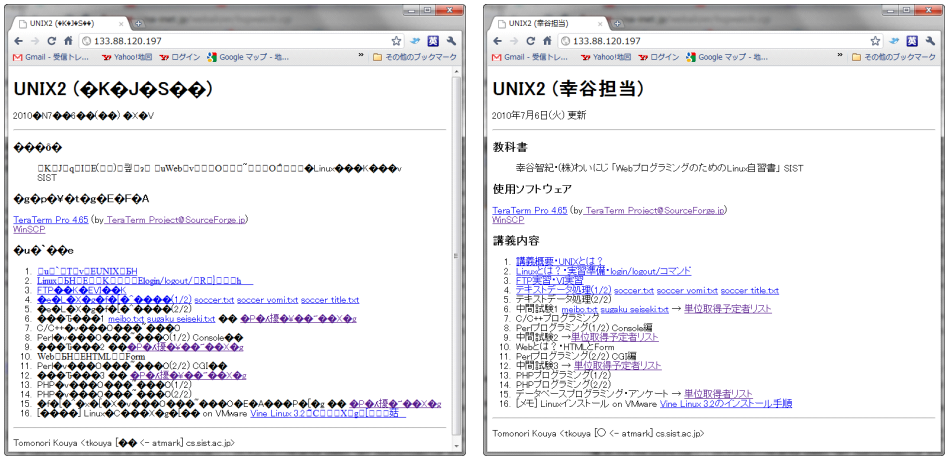

図 3.1　文字化けの例 (左図)：Shift JIS 指定で正しく表示 (右図)

になると，必ずしも「半角英数字」が ASCII コードの文字表現とは限らない。

この ASCII 文字集合に日本語で用いられるカタカナと記号 (¥ マークなど) を付加して 8bit = 1 Byte で表現したものが JIS(Japan Industrial Standard) X 0201 である (X は情報関係の規格を意味する)。

■2 Byte コード・・・JIS コード，ISO-2022-JP, Shift JIS, 日本語 EUC　日本人が日常的に使用する感じをすべて表現しようとすると，どうしても 1 Byte では足りず，2 Byte 分の文字集合が必要となる。これを JIS X 0208 と呼び，区と点という 2 次元座標に展開して文字を並べた表が用いられていることから，区点コードとも呼ぶ。古語や特殊な異体字を除けば，現在の日本語表記を表現するにはこれで十分であることが多い。この区点コードをそのまま符号化方式として採用したものを JIS コードと呼ぶ。

この JIS X 0208 という文字集合に対して，現在使用されている符号化方式は 3 つある。

一つは E-mail に日本語表記を用いる際に使用される ISO[3] 2022-JP。古いメーラー (MTA, Message Transfer Agent) を使っている人とメールのやり取りをすると，これ以外の符号化方式に対応しておらず，下手に Unicode なぞを使うと「文字化けして読めない！」というお叱りを受けたりする。

二つ目は Shift JIS という，Microsoft 社が提案して自社の MS-DOS(Windows 以前の CUI OS) で活用された符号化方式である。そのため，MS 漢字という言い方もされる。その名の通り，JIS X 0208 をシフト (shift,「ずらす」の意) して，JIS X 0201 と干渉しないように 2 byte 文字の割り当てを行い，最初の 1 Byte 目をチェックするだけで JIS X 0201 との区別ができるようにしたものである。現在の Windows は後述する Unicode を OS 内部で使用しているが，エディタ等では未だに Shift JIS がデフォルトの文字コードとして使用される。

三つ目は日本語 EUC (Extended UNIX Code) である。その名の通り，UNIX 用の日本語表記のための符号化方式である。中国語でも EUC が定められているが，日本語 EUC との互換性はないようである。Linux でも Vine Linux のように日本人がサポートしているディストリビューションではまだ日本語 EUC が使用されているが，主流は Unicode 系に移りつつある。

■4 Byte コード・・・Unicode　Unicode(ユニコード)[4] は，世界中で使用されている文字集合を可能な限り集めて 21 bit < 4 Byte で表現した文字集合である。中国，台湾，韓国，日本で使用されている漢字は，それぞれ字体が異なっているものが多いが，それを一つにまとめている (CJK, China-Japan-Korea) ため，制定当初はナショナリスティックな小反発が日本では起きた。現在，Windows や UNIX といった主要 OS では Unicode をサポートしているものが増えてきている。これは国際化が容易であること，4 Byte コードを用いても何ら支障がないほどコンピュータの性能が上がった (CPU の処理速度が上がり，メインメモリも大容量化した）ことで Unicode の負担が気にならなくなった，という事情も寄与しているようである。

この Unicode 文字集合を，1 Byte 単位，2 Byte 単位, 4 Byte 単位で符号化する方式を，それぞれ UTF-8, UTF-16, UTF-32 と呼ぶ。ASCII コードとの互換性が高いせいもあって，多くの OS，アプリケーションでは UTF-8 を使うことが多いようである。本書で使用している CentOS 5 の環境は，特に断らない限りこの UTF-8 を使っている。

3.2 リダイレクトとパイプ

ここでは入出力系統を適宜変更する**リダイレクト** (redirect) と，コマンド間の入出力を接続する**パイプ** (pipe) の使い方を簡単に解説する。

3.2.1 リダイレクト

まず本講義用の資料 (P.vii 参照) からつぎの三つのファイルをダウンロードし，自分の UNIX マシンへ cat コマンドを利用して転送してみよう。

soccer.txt　・・・サッカー選手のデータ（27 人分)
soccer_yomi.txt　・・・サッカー選手の氏名（漢字表記）とよみがな（27 人分）
soccer_title.txt　・・・サッカー選手データのタイトル部分（3 行）

まず，Web ブラウザで Web ページを開き，テキストファイルを表示する。文字化けしていないことを確認したら，TeraTerm 上で cat コマンドを使って

```
$ cat > soccer.txt
```

とする。この「cat > **ファイル名**」内の不等号>がリダイレクトを意味する。この場合は，cat コマンドの出力を指定ファイルに流し込む，という意味になる。不等号の向きを逆にすると (<)，ファイルの内容を入力としてコマンドに流し込む，という意味になる。

これを実行すると，cat コマンドは標準入力からの入力待ち状態になるので，Web ブラウザで表示したテキストファイルの内容をコピーし，TeraTerm の上にペーストする。するとペーストしたテキストファイルの内容が標準入力から流し込まれるので，終了したらファイルの終了を意味する [CTRL] + [z] を入力してリダイレクト処理を完了させる。

```
$ cat > soccer.txt
(略)
FW      柳沢敦   メッシーナ      177     73      1977/5/27
^Z ←[CTRL]キーを押しながら[z]キーを押す。
[1]+  Stopped                 cat >soccer.txt
$ ←プロンプトが戻ってくれば処理完了
```

最後に，ファイルがきちんと指定ファイル名になっているか，内容がコピーされているかを確認する。

```
$ ls
soccer.txt
$ cat soccer.txt
FIFA ワールドカップTM アジア最終予選
選手名簿
ポジション      氏　名   所属チーム      身長     体重     生年月日
GK      土肥洋一          ＦＣ東京         184     80      1973/7/25
(略)
FW      柳沢敦   メッシーナ      177     73      1977/5/27
```

soccer_yomi.txt, soccer_title.txt も同様にして転送する。転送後は上記のように必ずなず cat コマンドで文字化けせずに保存されていることを確認しよう。

以降ではこれらのテキストファイルを使用していく。

3.2.2　パイプ

パイプは，二つ以上のコマンドの出力と入力を接続する結果を繋ぐために使用される。例えば wc というコマンドはテキストファイルの行数，文字数を数えるために使用されるが

```
$ wc soccer.txt
```

としても

```
$ cat soccer.txt | wc
```

としても結果は同じである。下記の縦棒 (|) がパイプを意味し，cat soccer.txt の出力内容，つまり soccer.txt の内容を wc コマンドの入力として流し込んでいるので，上記と同じ処理を行ったことになる。

パイプはしばらくテキストファイル処理のためのコマンドを連結して使用する時に必要になるので，頭の片隅に入れておいて頂きたい。

3.2.3 コマンド出力の保存

リダイレクトはコマンドの出力結果をファイルに保存する時によく利用される。例えば，wc の出力結果を保存したいときには

```
$ wc soccer.txt > wc_soccer.txt
```

とリダイレクトを使うことによって wc_soccer.txt に書き込まれる。

3.3 grep, sort, join コマンド

さて，保存した 3 つのテキストファイルを使って，以下のテキストファイル処理のためのコマンドを使っていくことにしよう。

grep ・・・「正規表現」に合致する文字列を探索するコマンド (プログラム)
sort ・・・ファイル内のテキストを，ユーザの指定に従って並べ替えるコマンド
join ・・・ファイル内のテキストを，ユーザの指定に従ってくっつけるコマンド

3.3.1 grep コマンド

grep は，**正規表現**を用いてユーザが探しだしたい文字列を検索して表示するコマンドである。正規表現をきちんと説明するとそれだけで一冊の本ができてしまうので，ここではごく簡単な使い方だけを紹介する。

一番単純な使い方としては

```
$ grep ジュビロ soccer.txt
DF      田中誠  ジュビロ磐田    178     74      1975/8/8
DF      茶野隆行        ジュビロ磐田    177     74      1976/11/23
MF      福西崇史        ジュビロ磐田    181     77      1976/9/1
```

とする。「ジュビロ」という文字列を含む行を表示する。シングルコーテーションで検索

したい文字列を括って

```
$ grep 'ジュビロ' soccer.txt
DF      田中誠　ジュビロ磐田     178     74      1975/8/8
DF      茶野隆行        ジュビロ磐田    177     74      1976/11/23
MF      福西崇史        ジュビロ磐田    181     77      1976/9/1
```

と指定してもよい。スペースを含む文字列を指定する時には必ずこのようにすること。

正規表現は特別な意味を持つ ASCII 文字を利用する。例えば

```
$ grep 2$ soccer.txt
GK      曽ヶ端準        鹿島アントラーズ        187     80      1979/8/2
MF      中田英寿        フィオレンティーナ      175     72      1977/1/22
```

とすると，行末に 2 という文字を含む行を表示する。$は行末を意味する。

```
$ grep /*ビロ* soccer.txt
DF      田中誠　ジュビロ磐田     178     74      1975/8/8
DF      茶野隆行        ジュビロ磐田    177     74      1976/11/23
MF      福西崇史        ジュビロ磐田    181     77      1976/9/1
```

とすると，アスタリスク*部分は任意の文字 (なくても良い) を意味するので，「ジュ ビロ」「ジュ ビロ 磐田」「ビロ ビロ バー」などが検索される。このアスタリスクを**ワイルドカード**と呼ぶ。

[文字 1-文字 2] とすると，文字 1 から文字 2 までの文字コードの範囲の文字すべてを意味する。例えば

```
$ grep /*[3-5]$ soccer.txt
GK      土肥洋一        ＦＣ東京        184     80      1973/7/25
GK      楢崎正剛        名古屋グランパスエイト  185     76      1976/4/15
DF      三浦淳宏        ヴィッセル神戸  176     69      1974/7/24
DF      茶野隆行        ジュビロ磐田    177     74      1976/11/23
DF      松田直樹        横浜Ｆ・マリノス        183     78      1977/3/14
DF      中沢佑二        横浜Ｆ・マリノス        187     78      1978/2/25
DF      加地亮　ＦＣ東京        175     67      1980/1/13
MF      中村俊輔        レッジーナ      178     73      1978/6/24
MF      小笠原満男      鹿島アントラーズ        173     72      1979/4/5
FW      鈴木隆行        鹿島アントラーズ        182     75      1976/6/5
FW      高原直泰        ハンブルガーＳＶ        181     75      1979/6/4
FW      大黒将志        ガンバ大阪      177     74      1980/5/4
```

とすると，行末に ASCII 数字の 3〜5 までの文字がある行すべてが表示される。

一文字以上の任意の文字が存在することをチェックしたいときにはピリオド (.) を使用する。ワイルドカード (アスタリスク) とどのように異なるか，自分で考えてみよう。

```
$ grep /.[1-2]$ soccer.txt
MF      中田英寿        フィオレンティーナ      175     72      1977/1/22
FW      玉田圭司        柏レイソル      173     67      1980/4/11
$ grep /*[1-2]$ soccer.txt
GK      曽ヶ端準        鹿島アントラーズ        187     80      1979/8/2
MF      福西崇史        ジュビロ磐田    181     77      1976/9/1
MF      中田英寿        フィオレンティーナ      175     72      1977/1/22
FW      玉田圭司        柏レイソル      173     67      1980/4/11
```

3.3.2 sort コマンド

sort コマンドはその名の通り，テキストファイルの行を指定文字列の文字コード順に並び替えて表示するコマンドである。

例えば，soccer_yomi.txt の２列目 (-k2 オプション) のよみがな順に並び替えを行うと

```
$ sort -k2 soccer_yomi.txt
稲本潤一        いなもとじゅんいち
遠藤保仁        えんどうやすひと
(略)
本山雅志        もとやままさし
柳沢敦  やなぎさわあつし
```

と表示される。日本語の場合，漢字の並び替えはあまり意味を成さないので，カタカナ，ひらがなの文字列を並び替えの基準として用いることがほとんどであろう。

-r オプションを用いると逆順に表示される。

```
$ sort -r -k2 soccer_yomi.txt
```

前述の結果と比較して確認してみよう。

3.3.3 join コマンド

join コマンドはその名の通り，テキストファイルを指定文字列をキーとして結合するためのコマンドである。例えば次のようにして用いる。

```
$ join -j1 2 -j2 1 -o 1.1 1.2 2.2 1.3 1.4 1.5 1.6 soccer.txt socc
er_yomi.txt
GK 土肥洋一 どいよういち ＦＣ東京 184 80 1973/7/25

(略)

FW 柳沢敦 やなぎさわあつし メッシーナ 177 73 1977/5/27
```

soccer.txt の２列目と soccer_yomi.txt の１列目に同じ並びで同じ文字列があるということを前提に，これらをキーとして結合処理を行っている。処理結果は-o 以下に指定した順

に表示される。上記の場合は「soccer.txt」を 1,「soccer_yomi.txt」を 2 として表現し，1.1 が「soccer.txt の 1 列目」，2.2 が「soccer_yomi.txt の 2 列目」(=「soccer.txt の 1 列目」と同じ文字列)，1.3 が「soccer.txt の 3 列目」・・・という並びで結合結果を出力している。

課題 A

soccer.txt, soccer_yomi.txt を結合して, よみがな順にソートし, タイトル部 soccer_title.txt を先頭に付加して soccer_meibo.txt を作成せよ。但し，選手名簿は

よみがな　ポジション　氏名　所属チーム　身長　体重　生年月日

という順に並べること (図 3.2 参照)。きちんとした手順を得るまで試行錯誤し，手順が明確になった時点でログを取って印刷し，「学籍番号」「氏名」を明記の上，提出せよ。

```
FIFA ワールドカップTM アジア最終予選
選手名簿
よみがな　　ポジション　　氏　名　所属チーム　　身長　体重　生年月日
いなもとじゅんいち MF 稲本潤一 カーディフ 181 75 1979/9/18
えんどうやすひと MF 遠藤保仁 ガンバ大阪 177 70 1980/1/28
おおぐろまさし FW 大黒将志 ガンバ大阪 177 74 1980/5/4

・・・・・・・・・・・・・・・・・・・・・・・・・・・・(中略)・・・・・・・・・・・・・・・・・・・・・・・・・・・・

みやもとつねやす DF 宮本恒靖 ガンバ大阪 176 72 1977/2/7
もとやままさし MF 本山雅志 鹿島アントラーズ 175 64 1979/6/20
やなぎさわあつし FW 柳沢敦 メッシーナ 177 73 1977/5/27
```

図 3.2　soccer_meibo.txt の作成例

3.4　相対パス指定と準備作業

カレントディレクトリから，別のディレクトリを表現する方法として，絶対パス指定 (表記) と相対パス指定がある。UNIX では概ね深い階層がホームディレクトリになっているので，カレントディレクトリを基準とした相対パス指定が使えるようになると便利である。相対パス指定は，カレントディレクトリを基準とする。カレントディレクトリはドットスラッシュ (./) で表記する。カレントディレクトリよりひとつ上のディレクトリはドットドットスラッシュ (../) で表記する。

例として，本章で行う作業のため，先週作成した soccer_meibo.txt を unix05 ディレクト

リにコピーする作業を示す。

以下のように，unix05 ディレクトリを掘り，カレントディレクトリを unix05 に移動した後，相対パス指定を使って次のように行えばよい。

```
$ ls ../unix04 ← unix05のひとつ上のディレクトリにあるunix04にアクセスしてファイルリストを確認する
(略) ・・・  soccer_meibo.txt  ・・・(略)
$ cp ../unix04/soccer_meibo.txt ./ ←unix04からsuccer_meibo.txtをカレントディレクトリにコピー
$ cat soccer_meibo.txt  ← コピーしたsoccer_meibo.txtの内容を確認
```

3.5　sed・・・正規表現による文字列の置換

sed コマンドは，一種の言語とも言えるほど複雑な文字列の置換を可能にする。ここでは以降の作業に必要な最低限の機能だけ体験しておく。

sed コマンドの使い方は次のようになる。

```
$ sed 置換処理 処理対象テキストファイル名
```

置換処理には grep 同様，正規表現が用いられる。例えば”MF”という文字列を”ミッドフィルダー”に置き換えるときには

```
$ sed 's/MF/ミッドフィルダー/' soccer_meibo.txt
FIFA ワールドカップTM アジア最終予選
選手名簿
よみがな　　ポジション　　氏　名　所属チーム　　身長　　体重　　生年月日
いなもとじゅんいち ミッドフィルダー 稲本潤一 カーディフ 181 75 1979/9/18
(略)
やなぎさわあつし FW 柳沢敦 メッシーナ 177 73 1977/5/27
```

と指定する。

soccer_meibo.txt には３行のタイトル文があるので，その３行文は無視し，以降の行だけを置換の対象としたいときには，置換の前に 1,3d を指定し，セミコロン (;) で置換処理とつなげておく。

```
$ sed '1,3d;s/MF/ミッドフィルダー/' soccer_meibo.txt
いなもとじゅんいち ミッドフィルダー 稲本潤一 カーディフ 181 75 1979/9/18
(略)
やなぎさわあつし FW 柳沢敦 メッシーナ 177 73 1977/5/27

$ sed '1,3d;s/ /¥t/' soccer_meibo.txt
$ sed '1,3d;s/ /¥t/g' soccer_meibo.txt
```

3.6　コンパイラとインタプリタ

次に，インタプリタ言語である awk スクリプトを作成する。その前に，**コンパイラ**と**インタプリタ**という二つの言語処理系について，簡単に復習しておこう。

コンピュータ上で実行されるプログラムは，元となるテキストファイルにコンピュータ言語で処理内容を記述したものを実行可能な形式に変換してできたものである。この元になったテキストファイルを**ソースコード** (ソースプログラム) と呼ぶ。このソースコードをどのように「解釈」して「実行」するのかは，言語や処理系によって異なる。

■コンパイラ　ソースコードから，様々な**ライブラリ**に記述されている機能を追加した上で，CPU が直接処理できる実行可能ファイル[*1]を生成するソフトウェアをコンパイラと呼ぶ。実行速度重視のプログラムに適しており，C, C++, Fortran 等のコンピュータ言語ではもっぱらこのコンパイラを使って開発を行う。

■インタプリタ　ソースコードを一行ずつ解釈してその都度実行するソフトウェアをインタプリタと呼ぶ。インタプリタで解釈されるソースコードは**スクリプト**とも呼ばれる。テキストファイル処理・CGI(Common Gateway Interface) などに適している。後述する awk, Web アプリケーション作成のために使う PHP, Ruby，Perl といった言語はインタプリタを使用することが多い。

3.7　awk・・・古典的スクリプト言語

C 言語/UNIX の開発者が提案し実装した，テキストファイル処理用言語である。文法構造は C/C++ とよく似ている。ただし Perl が普及した現在はあまり使用されていない。Perl については後述する。

awk スクリプトは”ファイル名.awk”のように拡張子に”.awk”を使うのが普通である。スクリプトの構造は大きく 3 つに分割される。

```
BEGIN { データ入力前に実行 }
{データ入力時に実行}
END{ 終了前に実行 }
```

BEGIN{・・・}, END{・・・}の部分は，実行時の最初と最後に一度だけ実行される部分である。その間の{・・・}は，テキストファイルが一行読み込まれるごとに繰り返し実

[*1] Java や.NET 上の処理系のように，仮想的な (ソフトウェア上の) 処理系を包含したプログラムを生成する場合もある。

行される部分である。

例えば，サッカー選手名簿に登録されている人数を数える awk スクリプトは次のようになる。左の「行番号:」はスクリプトとは関係ないので打ち込まないこと！

```
1: BEGIN{print "---サッカー選手名簿---¥n";}
2: {print $1; print $5; print $6; count++;}
3: END{print "¥n合計", count, "人"; print "---サッカー選手名簿終了---¥n";}
```

これを実行すると次のような結果が得られる。

```
$ sed '1,3d' soccer_meibo.txt | awk -f sample.awk
--- サッカー選手名簿 ---

いなもとじゅんいち
181
75
・・・・
やなぎさわあつし
177
73

合計 26 人
---サッカー選手名簿終了---
```

課題 B

1. データをカンマ (,) で区切って表現したテキストファイルを CSV ファイル (拡張子は.csv) と呼ぶ。sed を用いて，soccer_meibo.txt から soccer_meibo.csv ファイルを作成せよ。例えば次のように sed とリダイレクトを併用すればよい。

```
$ sed '1,3d; (文字列置換の指定);' soccer_meibo.txt

いなもとじゅんいち，MF，稲本潤一，・・・
・・・
やなぎさわあつし，FW，柳沢敦，・・・

$ sed '1,3d; (文字列を置換する正規表現をここに書く);' soccer_meibo.txt > soccer_meibo.csv
```

2. awk を用いて，soccer_meibo.txt に掲載されている全ての選手の身長・体重の平均をそれぞれ計算する awk スクリプト ave.awk を作成して実行せよ。例えば次のように sed と ave.awk をパイプでつないで用いればよい。

```
$ sed "1,3d" soccer_meibo.txt | awk -f ave.awk

いなもとじゅんいち MF 稲本潤一 ・・・
・・・
やなぎさわあつし FW 柳沢敦 ・・・
平均身長: 178.923 平均体重: 73.0385
```

第 3 章の学習チェックリスト

- □ テキストファイルとは何かを第三者に説明することができる。
- □ リダイレクトとパイプを活用して実行結果をファイルに保存したり，複数のコマンドを組み合わせて実行することができる。
- □ テキストファイルに対して文字列の検索や置換，結合処理を実行することができる。
- □ 簡単な awk スクリプトを用いてテキストファイルからデータを取り出して加工することができる。