「情報処理」1年文I/IIクラス9-10　＃11

久野　靖*

1995.1.23

レポートおつかれ様でした。(まだ出していない人はなるべく早く出してください。繰り返しますが、自分の腕前相応であれば特に高度な絵を描く必要はありません。むしろ、レポートの体裁を整えてきちんと考察を書くことが重視されます。)

さて、先週のアンケートで「プログラムがもはやよく分からない」というご意見をいくつか頂きました。レポートも終わったことですし、今回は少しレベルを戻して、手続きのない (あまり長くない) プログラムをまたやろうと思います。それと、前回のプロセスの続きとして「シェル」の話をします。本日の内容は次の通り。

- シェルの機能とシェルスクリプトについて学ぶ。
- 配列と文字型を使って短いが役にたつプログラムが書けることを学ぶ。

1　シェルとスクリプト

1.1　プロセスは何の役に立つ?

「計算機の中で多数のプロセスが動いているのは分かったが、それで何が嬉しいのか分からない」というご意見を頂いた。そもそも皆様は、画面に複数の窓を開いて、ある窓でプログラムを実行しながら別の窓でファイルを編集したりしていますね? これはUnixで複数のプログラムが並行して動かせるからはじめて可能なわけでしょう?

さて、それぞれのプロセスでどんなことができるかは、そのプロセスでどんなプログラムが動いているかによって決まる。例えば代表的なプログラムとして次のものがある。

```
xclock  --- 画面に時計を表示する。
oclock  --- 画面に丸い時計を表示する。
xbiff   --- 画面に郵便箱を表示する。
twm     --- 画面の窓の位置を制御し、メニュー機能を提供する。
mule    --- 画面に編集用の窓を表示し、ファイルを入力・修正する。
kterm   --- 画面にコマンド入力用の窓を表示する。
a.out   --- あなたが作ったプログラム (機能はあなた次第)。
tcsh    --- コマンドを読み込み、実行する。
```

ところで、皆様は「3つの願い」の話を知っていますね? その手の話で主人公はだいたい「お金が欲しい」「ごちそうが欲しい」などの願いをかなえて、最後の願いをつまらないことに使ってしまうわけだが、私は子供心に「そんなもの頼むより、どんな願いでもかなえてくれる能力をお願いしたらいいではないか」と思っていた (ひねたガキだ)。上の場合でいうと、tcshがまさにそれに相当するわけですね? このように「〜して欲しい」というとその通りのことが起こるプログラムをUnixの世界では「シェル」と呼ぶ。今回はシェルの機能について整理してみる。

*筑波大学大学院経営システム科学専攻

1.2 コマンドとは?

シェルの一番基本になる機能は、様々なコマンドを実行することである。コマンドには引数を渡すことができる。以下のは典型的なコマンド実行の例である。

```
% ps ax
% xv test.ppm
```

ではコマンドとは何だろう? あらかじめシステムを作る時に登録しておいた名前がコマンドとして使える? 全然違う。実は、Unix のシェルではコマンドとは基本的にファイルである。コマンドが実はどのファイルか調べるには which コマンドを使う。

```
% which ps
/usr/bin/ps
% which xv
/usr/local/bin/xv
```

このようにコマンドは/usr/bin、/usr/local/bin などいくつかのコマンドディレクトリに分かれて格納されている。コマンドディレクトリの一覧は

```
% echo $path
...（すごく沢山）...
```

により表示させられる。

ところで、コマンドディレクトリの中に~/bin が含まれている。だから、ここに適当な名前でファイルを用意すると自分固有のコマンドを増やすことができる。

演習 1 まず、「How are you?」と表示する Pascal プログラムを作り、動かせ。次に「mv a.out how」のようにしてこのプログラムファイルの名前を変更せよ。

a. how というコマンドが実行できることを確認。
b. cd コマンドで現在位置を別の場所に移すと実行できないことを確認。
c. ファイル how を~/bin に移動し、「rehash」を実行。[1]
d. 今度は現在位置がどこでも how コマンドが使えることを確認。
e. ファイル how の名前を「ls」に変更せよ。ls というコマンドはファイル一覧と自分のプログラムのうちどちらになるか?

1.3 標準入出力、リダイレクション、パイプライン

Unix の多くのプログラムは、「標準入力」と呼ばれるチャネル (電気のソケットのようなもの) から入力を受け取り、加工して、結果を「標準出力」と呼ばれるチャネルに出す。特に指定しなければ標準入力はキーボード、標準出力は画面につながっている (図 1)。

```
% tr a-z A-Z    ←小文字を大文字に
This is a pen.
THIS IS A PEN.
^D              ←おわりの印
%
```

[1] コマンドを増やした時には rehash を実行しないと増やしたコマンドが利用可能にならない。

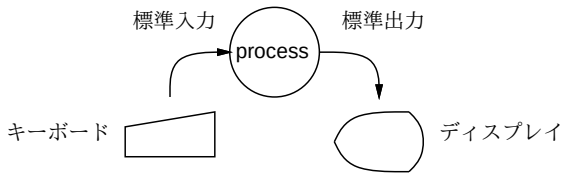

図 1: 標準入力と標準出力

ここで

% コマンド 引数 … <ファイル

のように指定すると、標準入力をキーボードの代わりに指定したファイルに接続させられる。これを入力リダイレクションと呼ぶ。[2] また、

% コマンド 引数 … >ファイル

により、標準出力を画面の代わりに指定したファイルに接続させられる。これを出力リダイレクションと呼ぶ。両方同時に使ってもかまわない。

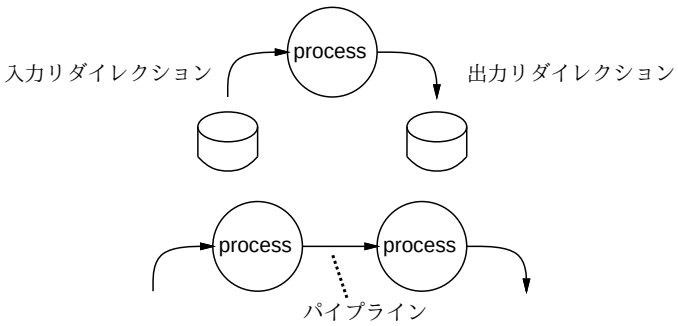

図 2: リダイレクションとパイプライン

さらに、あるコマンドの標準出力を別のコマンドの標準入力に接続することもできる (図 2 下)。

% コマンド 引数 … | コマンド 引数 …

これをパイプラインと呼ぶ。コマンドを 3 個以上つなげることもできる。

演習 2 入力がおしまいになるまで、[3]標準入力から 1 行読んではそれを 2 回書き出すというプログラムを作り、それを 3 個パイプラインでつないで

```
% a.out | a.out | a.out
```

のように動かし、行を打ち込んでみよ。どんなことが起きるか? また、なぜそうなるのか?

[2]実際には多くのコマンドは入力リダイレクションを使わなくてもファイル名を指定すればそのファイルから読み込むようにできている。tr は例外的にそうなっていない。

[3]入力がおしまいかどうかは `eof(input)` という関数で調べることができる。おしまいにならない限り繰り返すには「`while not eof(input) do` …」というループにする。

1.4 シェルの調整

シェルは頻繁に使うプログラムなので、その動き方を好みに合わせて調整できると何かと便利である。そのために、「シェル変数」というものがある。シェル変数は

```
% set 変数名 = 値
```

で値を設定できる。また値を見たい時には

```
% echo $変数名
```

とする (実はさっきコマンドディレクトリを見たのは変数 path の内容を見ていた)。例えばコマンドプロンプト (コマンドを打ち込んでいいよ、という記号) は変数 prompt に文字列を入れることで変更できる。

```
xss23{kuno}18: set prompt = 'kuno% '
kuno% set prompt = '>> '
>> set prompt = 'Baka '
Baka
```

演習 3 まず手で上のように打ち込み、コマンドプロンプトが自由に変更できることを確認せよ。気に入ったプロンプトが決まったらファイル.cshrc を Mule で編集し、「set prompt =」のところを適宜書き換えてから kterm の窓を新しく作り、確かにそのように変わったことを確認せよ。

なお、シェル変数には prompt のように特別な意味を持つものもあるが、普通の変数のようにただ値を覚えておくのに変数を使うこともできる。

1.5 シェルスクリプト

上で見たように、Unix の多くのプログラムは入力をキーボードから読み込むこともできるし、ファイルから読み込むこともできる。実は、シェルについても同じことが言える! つまり普段はコマンドをキーボードから打ち込んでいるが、代わりにファイルにコマンドを書いておいてシェルに読み込ませて実行させることができる。

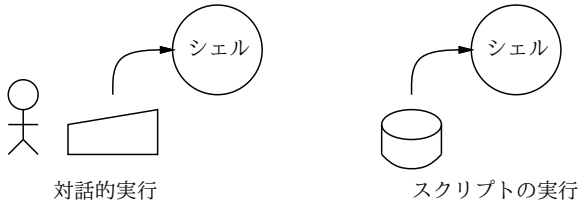

図 3: スクリプトの概念

```
% cat datewho  ←datewho というファイルの中身は…
date     ←日付を表示
who      ←現在このマシンを使っている人を表示
% tcsh datewho
1995 年 01 月 18 日 (水) 13 時 32 分 23 秒 JST
g430301  ttyr2   Nov 24 16:02   (xt204:0.0)
```

```
g440869  ttyp0   Jan 18 13:19   (xt206:0.0)
kuno     ttypa   Jan 18 11:45   (utogw.gssm.otsuk)
%
```

さらに面白いことに、このファイルの保護モードを「実行可能」にしておくとそのファイル名をいうだけで実行できるようになる。

```
% chmod ugo+x datewho  ←このファイルを実行可能に
% datewho
1995 年 01 月 18 日 (水) 13 時 33 分 18 秒 JST
g430301  ttyr2   Nov 24 16:02   (xt204:0.0)
g440869  ttyp0   Jan 18 13:19   (xt206:0.0)
kuno     ttypa   Jan 18 11:45   (utogw.gssm.otsuk)
%
```

つまり、テキストファイルを実行可能にしておけばそれはコマンドであるかのように使える。もちろん、~/bin に入れておけばどこからでも使える。

さて、このような機能 (スクリプト) は何の役に立つか?

- 決まり切ったコマンドの繰り返しをいちいち打ち込まずにすむ。
- 自分の好みのコマンドを簡単に追加できる。
- コマンド列をプログラムによって生成して実行させられる。
- 今すぐでなく後でコマンド列を実行させるようなことができる。

最後のをやりたい場合には、at というコマンドを利用する。

```
% at 1450 datewho
job 19062 at Wed Jan 18 14:50:00 1995
%
```

これは、datewho というスクリプトを 14:50 になったら実行することを指定している。これを利用すれば、(既に今年には遅いけれど)1 月 1 日の午前 0 時に「あけましておめでとう」のメールを友人に送信するなども簡単にできる。

演習 4 「echo Hello | mail 自分のユーザ名」とだけ書いたファイルを作って、これをスクリプトとして実行させ、自分にメールが送れることを確認せよ。OK だったら、at で数分後を指定してこのスクリプトを登録し、確かにその時間にメールが送られることを確認せよ。最後に、Pascal プログラムで

```
echo this is message no. 1 | mail 自分のユーザ名
echo this is message no. 2 | mail 自分のユーザ名
...
echo this is message no. 20 | mail 自分のユーザ名
```

と打ち出すものを作り、その出力をスクリプトとして実行し、自分あてに 20 個メールが来ることを確認せよ。

1.6 tcsh の対話機能

皆様が使っている tcsh の他にも Unix にはいくつかのシェルがある。代表的なのは

- `/bin/sh` — 作成した人の名前をつけて Bourne シェルとも呼ばれる。一番基本のシェル。
- `/bin/csh` — バークレー版 Unix で作られた改良版のシェル。

の 2 つである。これらは使い方がちょっと違う。tcsh は名前から想像がつくように、csh を改良したものである。どういう改良か? それについて説明しておこう。

まず、コマンドを打ち込んでいて打ち間違えることがありますね? 一番簡単なのは再度打ち込み直すことだが、かったるい。そこで、tcsh では`^P` を使って前に打ち込んだコマンド群 (履歴ともいう) を順に取り出してくることができる。行きすぎた時は`^N` で戻れる。

そして、`^F` と`^B` を使って取り戻したコマンド行の中でカーソルを動かすことができる。また、`[DEL]` や`^D` を使えばいらない文字を消すことができる。普通の字を打つとそれはその場所に入る。要するに、Mule とおんなじやり方で行を直すことができる。最後に `[RET]` を打つと直したコマンド行を実行させられる。便利でしょう?

また、`^P` で 1 個ずつ遡るのではなく、「〜という文字列で始まるコマンド行へ行きたい」ということもできる。それには、まずその文字列を打ち込んだ後で `[ESC]p` を打てばよい。探していたのよりもっと手前で同じものが見つかった場合には繰り返し `[ESC]p` を打てばさらに遡ることができるし、行きすぎた場合には `[ESC]n` を打てば戻る方向に探すことができる。

もう 1 つ覚えておくと便利な機能に「補完」(completion) というのがある。これは、シェルのコマンド行は

コマンド名　引数 …

で、コマンド名は (先に述べたように) コマンドディレクトリにあるファイルのどれかであり、また引数はファイル名であることが多い、ということを利用している。たとえば、oclock というコマンドを打ち込むのに全部打ち込まなくても、「oc」まで打ち込み、そこで `[TAB]` を打つと、他に「oc」で始まるコマンドはないので tcsh が残りを補ってくれる。つまり、

```
% oc[TAB]   →  % oclock_
```

のように変化する。同様に、ファイルやディレクトリの場合も「pa」で始まるディレクトリが「pascal」しかなければ

```
% ls pa[TAB]  →  % ls pascal/_
```

のように補ってくれる。

この `[TAB]` キーによる補完は選択肢が 2 つ以上あって一意に決まらないところで止まるので、そこで「どんな選択肢があるか見たい」という場合には`^D` を打つと選択肢を表示してくれる。例えば次のような具合である。

```
% ls pascal/sam14^D
sam14a.p  sam14b.p  sam14c.p  sam14d.p
% ls pascal/sam14_
```

つまり、`^D` を使えば ls を使わなくても「どんなファイルがあるか」をその場で見ることができる。同様に、コマンド名についても`^D` で一覧を表示することができる。

2　より実用性の高いプログラム

「文字列が読み書きできたからといってどうとも思わない」というご意見を頂いたが、前回やった文字列を利用すると、数値だけの時よりずっと色々なことができる。ここでは例題として「電卓」と「単語あてゲーム」を取り上げる。

まず、電卓を取り上げよう。普通の電卓は「窓」があってそこに数が表示されている。ここではそれに対応して v という変数を用意し、そこに現在計算中の値を入れておくとともに、演算するごとにその内容を表示させることにしよう。そして、演算は当然「+」「*」などの文字で現し、足したり掛けたりする数があればその後に続けて書く。つまり電卓プログラムの入力は

<指令><改行>
<指令><数値><改行>

のような形になる。<指令>は 1 文字、<数値>は Pascal の read で読めるようなものだとする。必ず<指令>がないといけないので、最初に数値を入れるのには=という指令を使うことにしよう。つまり、例えば「12+35」という計算をする場合は

```
=12
+35
```

のように打ち込む。演算が終わるごとに結果が表示されるので、普通の電卓のような「=」キーは要らない。PAD を図 4 に示す。つまり、ループの中でまず read(c) でコマンド文字を読み、コマン

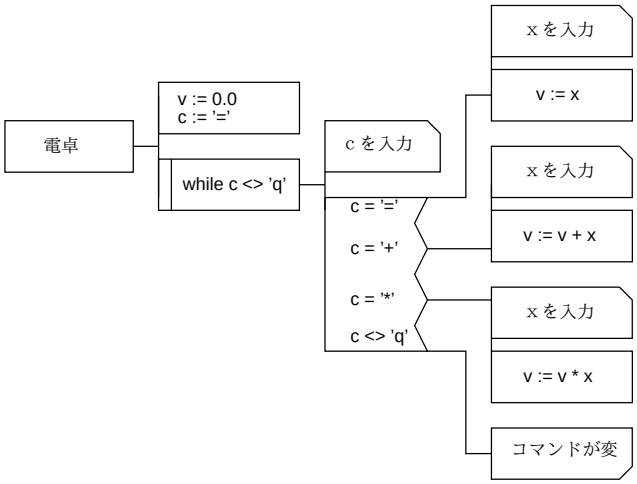

図 4: 電卓プログラムの PAD 図

ドが次に数値を必要とするものだった場合のみ read(x) でさらに数値を読む。コマンドが「q」の時は何もしないが、while の条件が成立しなくなってプログラムが終わる。プログラムを次に示す。

```
program sam17a(input, output);
var c:char;
    v, x:real;
begin
  v := 0.0; c := '=';
  while c <> 'q' do begin
    writeln(v:20:10);
    write('? '); read(c);
```

```
    if      c = '=' then begin read(x); v := x end
    else if c = '+' then begin read(x); v := v + x end
    else if c = '*' then begin read(x); v := v * x end
    else if c <> 'q' then writeln('???');
    readln  ←※
  end
end.
```

なお、※の readln は「改行まで読み飛ばし」を意味する。これがないと、次の read(c) で数値などの後にある改行文字が読めてしまいうまく動作しない。

演習 5 まず電卓プログラムを打ち込んで動かせ。その後、次のような改良をしてみよ。

a. 引き算や割り算もできるようにせよ。

b. sin、cos、平方根の計算もできるようにせよ。[4]

c. メモリ機能つき電卓にしてみよ。

どのような使い方にするかは自由に設計してよい。

さて、次は単語あてゲームである。これは、まず出題者が思いついた英単語を入力する。次に回答者が画面の前に座ると、まずすべての文字が「*」に置き代わったものが表示される。ここで解答者は 1 文字だけ好きな文字を打ち込む。すると、その文字が正解の中にあれば、その場所だけ「*」が正解の文字に戻って表示される。それを見て解答者は答を打ち込む。正解が出るまで繰り返し、「*」から元に戻す文字を聞いてくる。例えば次のような感じである。

```
% a.out
word  = these
                ←問題を画面から消すため 24 回改行する
word  = *****   ←問題
peek  = e       ←まず e がどこにあるか見る
hint  = **e*e
guess = there   ← there かな?
peek  = s       ←違った。じゃあ s は?
hint  = **ese
guess = these   ← these だろう!
Conglaturations!
%
```

プログラムの PAD を図 5 に示す。単語は配列 w に読み込み、空白以外をすべて「*」に置き換えたものを配列 o に用意する。そして、次の 1 文字 c を打ち込むごとに w の中に c と同じ文字があれば o の対応する位置を w と同じに戻して行く。配列 a は解答を読み込むために使う。Pascal プログラムは次の通り。

```
program sam17b(input, output);
var w, o, a: array[1..20] of char;
    c: char;
    i: integer;
begin
```

[4] 式 x の平方根は Pascal では sqrt(x) で求められる。

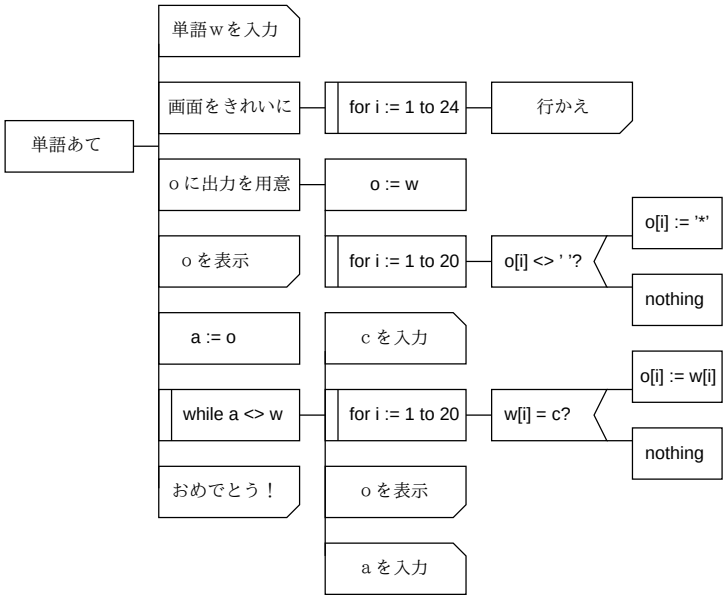

図 5: 単語あてプログラムの PAD 図

```
  write('word  = '); readln(w);
  for i := 1 to 24 do writeln;
  o := w;
  for i := 1 to 20 do if o[i] <> ' ' then o[i] := '*';
  a := o;
  writeln('word  = ', o);
  while a <> w do begin
    write('peek =  '); readln(c);
    for i := 1 to 20 do if w[i] = c then o[i] := w[i];
    write('hint  = '); writeln(o);
    write('guess = '); readln(a)
  end;
  writeln('Conglaturations!')
end.
```

演習 6 単語あてゲームのプログラムを次のように改良せよ。

a. 不正解なら「残念でした!」のようなメッセージを表示する。
b. 不正解の個数を数えて表示する。
c. 正解が出なくても N 回不正解だったら答を表示して終わる。
d. 2 人で交互に出題しながら遊べるようにする。
e. さらに、適当な方式で得点をつけ点数を競うようにする。[5]

N や表示形式や得点のつけ方などは自由に設計してよい。

[5]もちろん、少ない回数で当てた方が得点が高く、また難しい (長い?) 単語だと正解の得点も高くなるのが正しそうですね。

A　レポートの絵の収集について

せっかく皆様にプログラムで絵を作って頂いたのですから、互いに労作を鑑賞しましょう。作った絵を xv で表示した状態で、マウス右ボタンをつついて control パネルを出し、「Save」ボタンをつついて save パネルを出し、ファイル形式「GIF」を選んで保存してください。ファイル名は「`~/WWW/report1r.gif`」にすること。こちらで皆様の絵がまとめて見えるように WWW のページを作っておきます。出席点 1 点を差し上げますので、ぜひやっておいてください。

B　本日の課題 11A

本日の課題は、演習 1 と演習 2(いずれもやってみてどんなことが分ったか) を提出してください。(余裕があれば演習 4 も。) レポートのタイトルは 11A です。アンケートは次の通り。

Q1. コマンドを組み合わせたり、入出力を切り替えたり、新しいコマンドを作れたり、前のコマンドを探したり、手直しできたりすることについてどう思いましたか? 面倒すぎて使う気にならないですか?

Q2. 今日の 2 つの Pascal 例題プログラムについて、どう思いましたか? 役に立つと思いますか? 自分でもアイデア次第で役に立つプログラムは作れるという気がしますか?

Q3. その他、感想や要望があればどうぞ。

C　次回までの課題 11B

レポートが終ったので、次回までの課題が復活します (ブーイングが聞こえてくるようだなあ…)。演習 5 または 6 のどちらかを (a〜c、a〜e のどれか 1 つ以上やればよい) やってください。PAD はいいのでプログラムと実行例を提出のこと。レポート番号は 11B です。アンケートは次の通り。

Q1. プログラムを作るのは面白いですか? できたプログラムで遊ぶのはどうですか? 自分でプログラムを作るようになって、その辺の感じ方は変わりましたか?

Q2. 課題に対する感想と今後の要望をお書きください。

課題は、次回授業開始時刻までに、レポートボックスに提出してください。