「情報処理」1年文I/IIクラス11-12　＃6

久野　靖*

1995.11.20

0　本日の目標

かねて予告の通り、次回は駒場際翌日なので補講日とし、出席自由とします。ただし、プログラミングが「わからん」という人は (わざわざそのためにやるのですから) ぜひ出席してください。当日レポート 7A のぶんだけ点数を差し上げます。(7B はありません。) なお、レポート 6B の〆切は 2 週間後 (12/4) となります。とりあえず、本日の目標は次の通り。

- 変数と値の推移について理解を深める。
- どこに「;」を入れるべきか分かるようになる。
- 反復 (繰り返し) についてちょっとだけ学ぶ。
- ドロー系ソフトによるお絵描きについて学ぶ。

1　プログラムはとは結局何か/☆

プログラムはむずかしい、よく分らない、という人は依然としているけれど、それらの人が「なぜわからない」かを究明するのは結構むずかしい。(ここでインタビューもしてみたいですね!) とまれ、同じことを色々な角度から説明するという方法を取ってみよう。

まず、プログラムというのは計算機が実行する「ビット列の加工」のやり方を指示したものだ、というのは前からさんざんやりましたね? そして、計算機というのはおもに CPU(加工を行うところ) とメモリ (ビット列を保持しておくところ) から成るということも。これに対応して、プログラムの「材料」というのは結局次の 2 つしかない。

(a) ビット列を入れる「箱」(変数)…メモリに対応。

*筑波大学大学院経営システム科学専攻

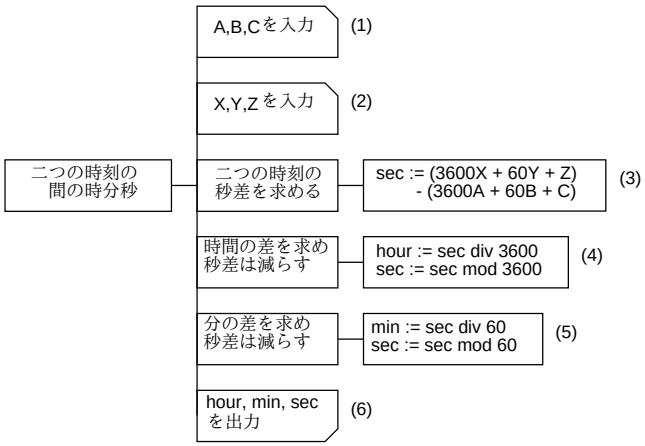


図 1: 枝分れのない時間差問題の PAD 図

(b) ビット列を加工する動作の指示…CPU の動作に対応。

これらはそれぞれ、Pascal プログラムの場合「変数宣言」(var 部) と「文」に対応していた。しかし「文」は沢山あって目立つけれど、変数の方は最初にちょっとあるだけだからあまり重要な気がしない。ところが、プログラムの各「文」を実行している時それぞれの変数がどんな値を保持しているか理解しなければ、プログラムは書けない。その辺を理解して頂くために、変数の「箱」を描いた絵を使ってみよう。プログラムそのものは枝分れのない時間差問題のプログラムを例に用いる (図 1)。

3 時 22 分 40 秒と 5 時 18 分 25 秒を入力とした場合に、各段階ごとに変数の「箱」の値が変化していく様子を図 2 に示した。このように、PAD なり Pascal なりで記述された「動作」が実行されるごとに、変数の値がどう変化していくかをイメージすることがプログラム理解のポイントとなる。もしよく分らなくなったら、自分でこれと同じような図を描いて考えることはよい方針である。

演習 1 ☆ 図 3 の PAD 図およびそれを変換してできた以下の Pascal プログラムについて、適当な入力を仮定して変数の箱の値の変化の様子を描き、これは要するに「何をしている」のか述べよ。

```
program sam6a(input, output);
var a, b, c: integer;
begin
  write('a = '); readln(a);
  write('b = '); readln(b);
```

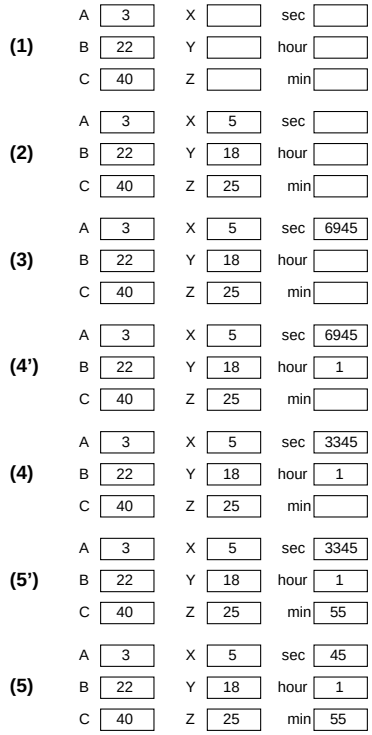


図 2: 図 1 のプログラムにおける変数の値の変化

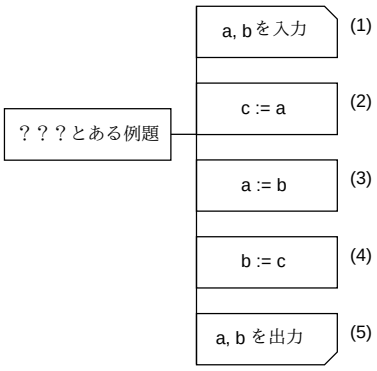


図 3: とある例題プログラムの PAD 図

```
    c := a; a := b; b := c;
    writeln('a = ', a:2, '  b = ', b:2)
end.
```

2　どこに「;」を入れるべきか?/☆

これまで Pascal のプログラムを書いて頂いて、いくらか書けるようになった人を一番悩ませるのは「どこに; を入れるべきで、どこには入れなくてもいいか」という問題のようです。ここではちょっと詳しくその説明をしてみませう。

まず、資料#2 でも説明したように、Pascal では一番外側の `begin` から `end.` までのところは次のようになっている。

`begin` 文 ; 文 ; … ; 文 `end.`

つまり、「;」は「文と文の区切りに入れる」ということである。それをはっきりさせるように、先の sam6a のプログラムについて、それぞれの文を四角で囲んだものを図 4 に示した。確かに文と文の切れ目に「;」が入っている。

```
program sam6a(input, output);
var a, b, c: integer;
begin
  [write('a = ')]; [readln(a)];
  [write('b = ')]; [readln(b)];
  [c := a]; [a := b]; [b := c];
  [writeln('a = ', a:2, '  b = ', b:2)]
end.
```

図 4: とある例題プログラムの文を箱で囲む

では、if による枝分かれがあったらどうだろう? if 自体も 1 つの「文」であり、なおかつこの中に複数の「文」が含まれている、ということが問題をやっかいにしている。つまり、こうなっているわけだ。

`if` 条件 `then` 文 `else` 文

演習 2 ☆ 前回出てきた次のプログラムについて、それぞれの文を四角で囲め (書き込み用に印刷したものを配ります)。if 文は全体として文であり、なおかつ中に文が入っていることに注意。

```
program sam6b(input, output);
var a, b, c: real;
begin
  write('a = '); readln(a);
  write('b = '); readln(b);
  write('c = '); readln(c);
  if a > b then
    if a > c then
      writeln('larger value = ', a:8:4)
    else
      writeln('larger value = ', c:8:4)
  else
    if b > c then
      writeln('larger value = ', b:8:4)
    else
      writeln('larger value = ', c:8:4)
end.
```

ところで、if-then-else if-then の連鎖は本当は多数の if 文が組み合わさっているのだが、ここでは簡単のため図 5 左のように各動作として文を複数含んだ 1 個の文であると考えておこう。そうしてみると、符号の問題のプログラムについて文を箱で囲んだものは図**??**右のようになり、確かに「文と文の切れ目」にのみ「;」があることが分かる。

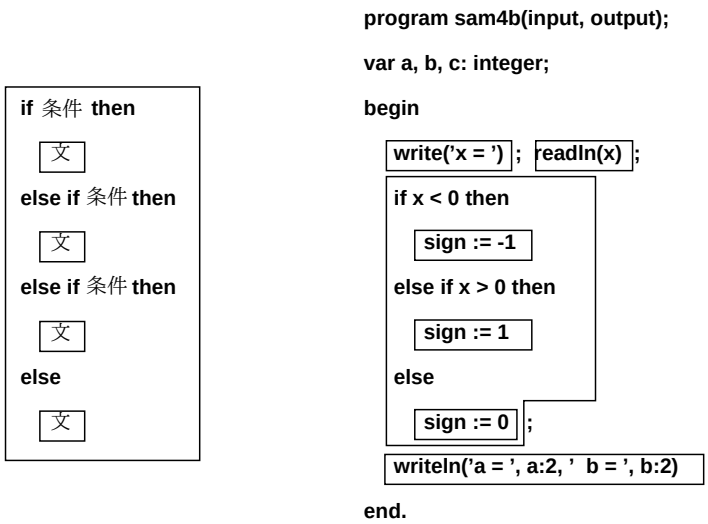
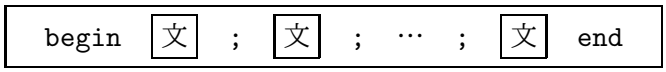


図 5: 符合問題プログラムの文を箱で囲む

最後の難関として、「if の枝に複数の文を書きたい時には begin と end で囲む」という問題を考える。begin と end で囲んだものは「複合文」といい、それ全体として 1 つの文だが、中に複数の文を含んでいる。つまり



のようになる。

演習 3 △ 次の例題プログラムについて、文を箱で囲み、確かに文と文の切れ目にのみ「;」があることを確認せよ。ついでに、何をするプログラムか考えよ。

```
program sam6c(input, output);
var a, b, c: integer;
begin
  write('a = '); readln(a);
  write('b = '); readln(b);
  if a < b then begin
    c := a; a := b; b := c
  end;
  writeln('a = ', a:2, '  b = ', b:2)
end.
```

というわけで、これからも「;」の入れ方が分からなくなったら文を箱で囲んで見ることをお勧めする。

3 前回の練習問題について/*

a. 2 つの数値 (整数) を読み込み、小さくない順になるように、並べかえて出力せよ。

この問題は何通りかの PAD 図が書ける。1 つは並べ替えた後の大きい方、小さい方に相当する変数を 2 つ用意しておくというもの。

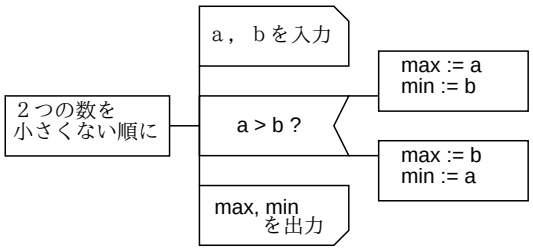


図 6: 大小順に打ち出す (1)

```
program sam6d1(input, output);
var a, b, max, min: integer;
begin
```

```
  write('a = '); readln(a);
  write('b = '); readln(b);
  if a > b then begin max := a; min := b end
           else begin max := b; min := a end;
  writeln('大: ', max:5);
  writeln('小: ', min:5)
end.
```

もう 1 つは、枝分かれした中で書き出すというもの。

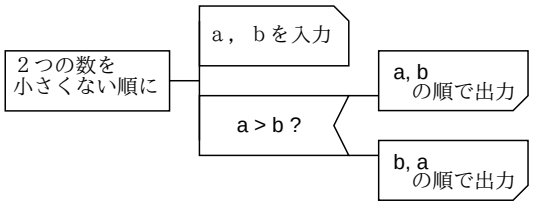


図 7: 大小順に打ち出す (2)

```
program sam6d2(input, output);
var a, b, max, min: integer;
begin
  write('a = '); readln(a);
  write('b = '); readln(b);
  if a > b then begin
    writeln('大: ', a:5);
    writeln('小: ', b:5)
  end
  else begin
    writeln('大: ', b:5);
    writeln('小: ', a:5)
  end
end.
```

なお、1 行に書いてしまう場合にはこっちのやり方だと begin-end がいらなくなるのでちょっと簡単。(しかし、プログラムの簡単さより見やすさを優先しよう!) もちろん、これら以外の方法もあってよい。それにしても、begin-end があると「どういう書き方をしたら美しいか」が難しい!

さて、あとは問題 b はちょっと難しいので次回回しにさせていただき、年号の 2 つを先にやる。まず西暦→元号だが、これは if-then-else if の連

鎖が上から順に条件を調べていくことを利用するとすっきりできる。PAD図は図 8 に示す。ところで、「`year - 1868 + 1`」は「`year - 1867`」で

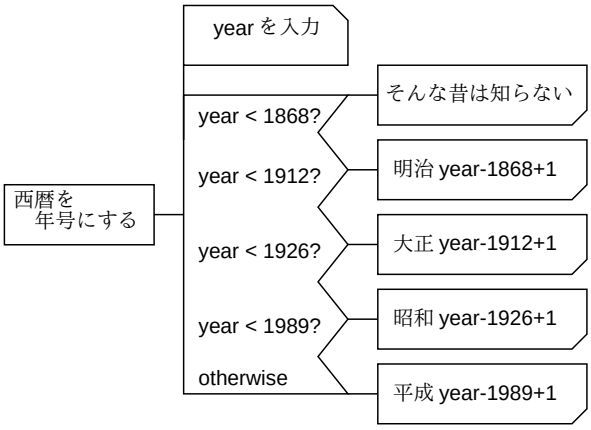


図 8: 西暦を元号に直す

いいと思いましたか? 別に間違いではないのだが、条件判定に使う数値と引き算に使う数値を一致させておいた方が「私は気持ちがいい」からそうやっている。なお、このために計算時間が損ということは全然ない。ではプログラムを示す。

```
program sam7c(input, output);
var year: integer;
begin
  write('西暦 = '); readln(year);
  if year < 1868 then
    writeln('too old, I do not know.')
  else if year < 1912 then
    writeln('明治', year-1868+1:1)
  else if year < 1926 then
    writeln('大正', year-1912+1:1)
  else if year < 1989 then
    writeln('昭和', year-1926+1:1)
  else
    writeln('平成', year-1989+1:1)
end.
```

ところで、「年がどこからどこまでの間」という条件を使いたいと思った人もいるでしょうね? その説明をしないですいません。それには、

```
if (year < 1989) and (year >= 1926) then ...
```

のように、条件を「and」という演算子で結合すれば書ける。同様に、「〜または〜」を表す「or」という演算子と、「〜でない」を表す「not」という演算子も用意されている。ただ、この問題に限っては複合条件を使うよりも if-then-else if の連鎖の性質を使う方がスマートだと思ったので、つい説明しそこなったわけなのでした。

逆の、元号→西暦もやっておこう。これも if-then-else if ですね。なお、

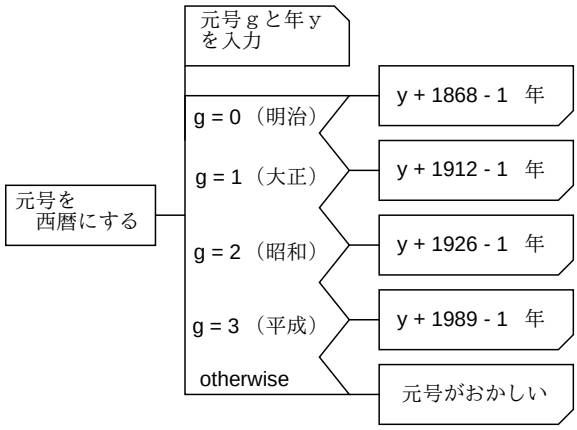


図 9: 元号を西暦に直す

予期しない (おかしい) 入力についてもなるべくちゃんと「おかしい」と教えてあげるのが親切というか、常識。(自分が使う人だったらどうか考えるように。) で、プログラムはもはや簡単ですね?

```
program sam7d(input, output);
var g, y: integer;
begin
  write('gengou (0=明治, 1=大正, 2=昭和, 3=平成) : '); readln(g);
  write('y = '); readln(y);
  if g = 0 then
    writeln(y + 1868 - 1:1)
  else if g = 1 then
    writeln(y + 1912 - 1:1)
  else if g = 2 then
    writeln(y + 1926 - 1:1)
  else if g = 3 then
    writeln(y + 1989 - 1:1)
```

```
  else
    writeln('g should be 0, 1, 2, or 3.')
end.
```

4 繰り返し/☆

さて、ここまでに説明してきた PAD や Pascal の機能では、上から順番に実行していって (枝分かれはあっても) 下まで来たらおしまい、だから大したことはできない。そこでいよいよ、繰り返し (ループとも呼ぶ) について学ぼう。まず一番基本である、while 型の繰り返しを表す PAD を見て頂く (図 10)。たて線の入った箱は「繰り返しの箱」であり、その右側

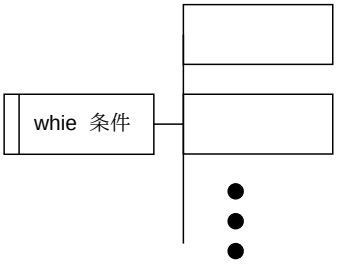


図 10: 繰り返しを表す PAD

にくっついている部分 (ループ本体とも呼ぶ) は繰り返し何回も実行される。ここがこれまでの 1 回しか実行されない PAD とは大幅に違っている。

では、どれくらいの回数実行されるのか? while 型の繰り返しでは、繰り返しは「条件」が成り立つ限り何回でも実行される。だから、ループ本体には必ず条件を変化させていく部分が含まれていなければならない。具体的に見てみよう。例題として、ずっと前に考えて頂いた例題を使う。まず次のものから (数はすべて正の整数)。

a. かけ算と割り算を使わず、数 x が奇数か偶数かを求める。

ここで、次の事実を使う。

- 0 は偶数、1 は奇数である。
- ある数から 2 を引いても奇数/偶数の別は変化しない。
- 1 より大きい正の整数から 2 を引いても、0 より小さくはならない。

これらに基づいて、図 11 を考えた。確かに、ループ本体「`x := x - 2`」は条件「`x > 1`」を変化させるような操作になっている。より具体的に考

えると、x がだんだん減っていくのだから、いつかは条件が成り立たなくなってループが終わるはずである。また、ループ本体を何回実行しても、x の奇数/偶数の別は変化しない。そして最後に、ループが終わった時は x は 1 以下であり、また 0 より小さくはないはずだから、結局 0 か 1 かのどっちかである。というわけで、0 か 1 かを調べれば偶数か奇数か分かることになるでしょう?

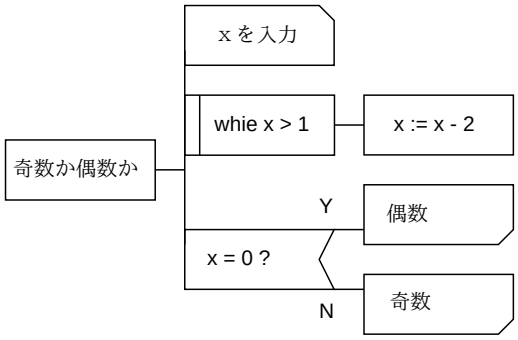
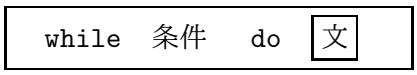


図 11: 奇数偶数判定の PAD

さて、このようなループを含む PAD を Pascal に直すのには、次の while 文を使う。



while 文はこれまで見て来た「文」が置ける場所ならどこにでも置ける。また構文 (箱の形) 的には if より簡単、というか else 部のない if のようなものである。では上の PAD をプログラムに直す。

```
program sam8a(input, output);
var x: integer;
begin
  write('x = '); readln(x);
  while x > 1 do x := x - 2;
  if x = 0 then writeln('guusuu') else writeln('kisuu')
end.
```

では次の問題へ行こう。

b. 正の正数 x と y を入力し、その積を求めよ (かけ算演算は使わずに)。

これを PAD にしたものを図 12 に示す。ここではまずループに入る前に n を 0 にしている。そして、ループの中では n を y 増やし、x を 1 減らしている。これはどういう意味があるだろう?

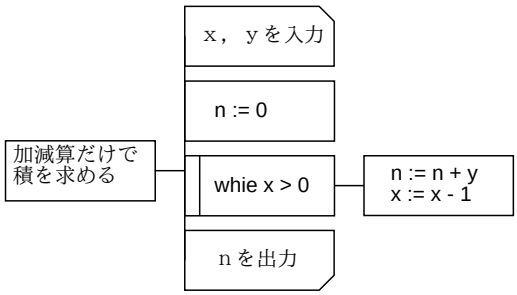


図 12: 加減算だけで積を求める PAD

実は、このプログラムでは求める答を M とすると、常に $M = n + xy$ が成り立っている。なぜなら、最初に n を 0 にするから、確かに M は積 xy に等しい。その後、ループの中では n を y 増やす変わりに x は 1 減らすのだから、$(n+y)+(x-1)y = n+xy = M$ で確かにこの状態は変化しない。次に、ループの中で x を減らしていくから、確かに条件「`x > 0`」に影響を与え、なおかついつかは x が 0 になってループは終わる。終わった時は x が 0 なのだから、$M = n$ が成り立っている。つまり n に答が求まっている。

なんて面倒くさい! と思いましたか? そんな理屈を考えないでも、単に y を x 回足していることは見ればわかるじゃないか! 確かにこれくらいの例題ならそうなのだが、これから次第に複雑なプログラムになった時、それを正しく作るコツは、次の事柄をきちんと確認することである。

- ループを通じて変化しない条件は何か?
- ループ本体に、条件の成否を変化させる部分があるか?
- そして、その変化は最後にはループが終わるような変化か?
- ループが終わった時、欲しい値が確かに求まっているか?

演習 4 ☆ 図 12 の PAD を Pascal に直して動かせ。

では次の問題へ進もう。

c. 正の整数 x、y を読み込み、その最大公約数を求めよ。

今度は考え方をまず説明するので、PAD からやって頂く。まず、x と y が等しければ、最大公約数は x(そして y) そのものですね? 次に、もし x と y が等しくなければ、そのうちより小さい方を x'、大きい方を y' とする (例によって、逆順なら交換すればいいですね)。そして、$y'' = y' - x'$ とすると、x と y の最大公約数は x' と y'' の最大公約数と同じである。(な

ぜか? x も y も最大公約数 g の整数倍なのだから、その大きい方から小さい方を引いた数も、やっぱり g の整数倍に決まっているでしょう?) 従ってこれを新たな x、y と考えてよい。これを繰り返していくと、x も y も段々小さくなっていくが、g より小さくなることはできないから (なぜ?)、最後はともに g に等しくならざるを得ない (おわかりかな?)。

演習 5 △ 最大公約数の問題を PAD にしてみよ。さらに、Pascal にして動かせ。

なお、Pascal では「x と y が等しくない」という条件は「`x <> y`」のように書く (単に≠という字がなかったから)。では、ループに慣れるための練習問題もいくつか挙げておこう。

演習 6 △ 次のプログラムの PAD 図を書け。さらに、Pascal にして動かせ。

a. 正の整数 x と m を読み込み、x を m で割った商と余りを求めよ。ただし加減算のみを使うこと。
b. 正の整数 n を読み込み、その階乗 $(n \times (n-1) \times \ldots \times 1)$ を求めよ。
c. 正の整数 n を読み込み、$x^2 \geq n$ を満たすできるだけ小さい正の整数 x を求めよ。

5 お絵描きについて (2)/＊

さて、前回は「ペイント系」のお絵描きソフトである xpaint を使っていただいた。感想の中に「ぎざぎざでなめらかな線が描けない」というのがあったが、それは絵を点の集まりで表す以上あたりまえの制約である。

これに対して、本日やるのは「ドロー系」のお絵描きソフトである。その基本的な考え方は、いわば「無限に伸縮性のある針金でできた、直線や長方形や円や楕円の形をした枠」を紙の上に置いていくものと思えば良い (なお、それに無限に伸縮性のあるスクリーントーンを貼ることもできる)。

それで何が嬉しいかと言うと、紙の上に置いてみてちょっと位置が悪いなと思ったら (何しろ針金の枠だから) ちょっとずらして見たりでき、いくらでも直しが効くということである。またもちろん、針金は斜めに置いてもぎざぎざしたりはしない。

5.1 idraw 入門/＊

ものは試し、ドロー系ソフトをとにかく使ってみよう。ここでは idraw という、上述の MacDraw の「そっくりさん」のフリーソフト (無料で入

手できるもの) を使う。無料だから多少使いづらくても文句を言わないように。idraw を起動するには、Kterm の窓で

```
% idraw &
```

という。しばらく待つと図 13 のような新しい窓が現れる。

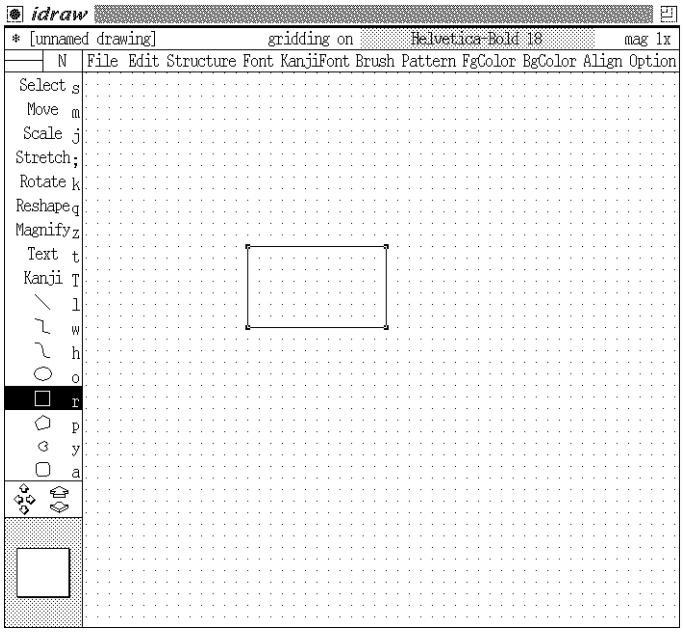


図 13: idraw の窓の様子

ここで、窓の左側に「Select」などと並んでいるのは「パレット」と呼び、各種操作や描く図形の種類を切替えるのに使う。パレットの上にマウスカーソルを持っていき、マウスボタンをどれでもいいから押してパレットの押した項目が反転することを確認してみよ。

一方、窓の上の方に「File」などと並んでいるのは「プルダウンメニュー」と呼び、パレットにないような各種の操作を起動するのに使う。メニューの使い方は既によくご存じの背景メニューに似ているが、少しだけ違う。まず「File」などと書かれた文字の上でボタンを押し下げ、そのまま保持する。するとメニューが現れるので、そうしたらボタンを押したままマウスを移動して選択したい項目を反転させ、その状態でボタンを離すとその項目が起動される。気が変わってやめたい場合にはマウスカーソルをメニューの外に出して (どれも反転してない状態で) ボタンを離せば何も起こらない。さっそく、各メニューにどんな項目があるかを一通り見物してみよ。

さて、ドロー系でお絵描きを行なう時美しくするコツは (あくまでも私の考えだが)、きちんと方眼紙を用いて、その目の上に合わせて線を引くことである。皆様の idraw の窓には方眼が現れていませんね? そこで、「Option」メニューの中の「Grid Visible/Invisible」という項目を選んで目盛りを表示させる。ついで、同じメニューの「Grid On/Off」を選んで窓の一番上に「gridding on」と表示された状態にする。これで、あなたが描く直線や四角はすべて目盛りの上に乗るように idraw が面倒を見てくれる。

5.2 idraw による図形描き / *

ではいよいよ絵 (ドロー系の場合には「図」という方が正確かも知れない) を描こう。以下では特に指定しない限り、マウスボタンはすべて左ボタンを使うこと!

まず、パレットで四角を選び、その後窓の中で左ボタンを押し、押したままマウスを移動する。すると、移動につれて点線で四角が表示される。(その四角が必ず方眼の目盛りに揃っていることに注意。) 適当なところでボタンを離すと、その位置に四角が置かれる。さっそく、いくつか四角を描いてみよ。また、丸や直線も同様のやり方で描ける。描いてみよ。

残りの図形は、やり方がちょっと違う。例えば折れ線のところを選んでから、窓の中で左ボタンをクリックする (押して離す)。そのままマウスを移動すると、さっきと同様に直線が描かれ始める。次に適当な位置で再度クリックすると、そこまでの直線は固定され、そこから新しい直線が描かれ始める。これだといつまでたっても終らないが、終りにしたい時はマウスの中ボタンを押せばそこまでで折れ線がおしまいになる。さっそく折れ線を描いてみよ。また、曲線 (折れ線をなめらかにしたようなもの)、多角形、閉曲線も (終りにすると最初の点と結ばれて閉じた図形になるだけで) ほぼ同様である。描いてみよ。

最後に、文字列が残った。文字列については、まずパレットで「Text」を選び、次に窓の中で左ボタンをクリックすると縦棒が現れる。その状態でキーボードから打ち込むと文字が入る。漢字を入れたい場合には「Kanji」を選び、Mule と同様「`Contorol-\`」を打てばローマ字モードになり、あとは同様に使えるが、ただし! ローマ字モードから抜ける方法はなぜか「`Shift-Space`」である。文字列を入れ終ったら、次に入れたい場所をクリックするか、パレットで別の図形を選べばそれで固定される。

5.3 選択と移動/＊

ところで、常に一番最後に描いたものの周囲に「小さい四角」が表示されていることに気がつきました? これは「ハンドル」といい、その図形が「選択されている」ことを意味する。

実は、マウスの右ボタンで図形の上をクリックすると、その図形が選択図形になる (やってみよ)。また、同時に複数の図形を選択することもできる。そのためには、次のどちらかを行なう。

1. 四角を描く要領で、選択したい図形を囲む領域の右上隅で右ボタンを押し、そのままマウスを移動して (長方形が表示される) 選択したい図形を囲む四角ができたところでボタンを離す。
2. シフトボタンを押しながら右ボタンで図形の上をクリックすると、その図形がこれまでの選択に追加される。選択されているものの上でシフトボタンを押しながらクリックすると、選択から外される。

だいたいは 1 をやった後 2 で微調整をする。さっそくやってみよ。

では、図形の選択は何の役に立つのか? まず、図形を移動するとき役に立つ。マウスの中ボタンは図形の移動機能に対応していて、選択された図形群のどれかの上で中ボタンを押し、そのままの状態でマウスを移動すると図形群がマウスに呼応して移動する。(やってみよ。) 実は、移動したい図形が 1 個だけの場合にはいちいちまず右ボタンで選択しなくても、いきなり中ボタンを押すことで選択しかつ移動開始できる。

5.4 様々な属性調整/＊

さて、これまでのところ図形はいろいろできたが、線の太さも文字の形も 1 種類なのであまりすばらしくない。ところが! ここからがドロー系ソフトのすばらしいところで、後から色々属性を変化させられる。まず文字列をどれか 1 個右ボタンで選択し、その状態で「Font」(漢字のは「KanjiFont」メニューを下ろしてどれか適当なフォントを選んでみよ。すると、選択した文字列はそのフォントに変わってしまう。

次に、直線または折れ線または曲線を右ボタンで選択し、「Brush」メニューで線の種類を選ぶと線の形が変わる。下 4 項目は矢印のつけ方 (なし、右矢印、左矢印、両矢印) で、線の太さなどとは独立に変えられる。実は四角や丸や多角形や閉曲線などでも線の太さは変えられるが、矢印については意味を持たない。

次に、四角形や丸や多角形や閉曲線を選び、パターンメニューから適当なパターンを選んでみよ。実はこれまでは「白」が選ばれていたが、一番上の「None」(透明) を選ぶと図形が枠だけになって重なった向う側の

図形も透けて見えるようにある。なお、図形の重なり順を変えたければ(窓とは違って移動しただけではだめで)、変えたい図形を選択した上で「Structure」メニューの「Bring to Front」「Send to Back」を使う。

次は色で、パターンを縞々などにしたままの状態で「FgColor」「BgColor」をそれぞれ選んでみる。パターンの「模様」と「地」の色がそれぞれ選べることがわかる。なお線や文字は「FgColor」だけが意味を持つ。

5.5 その他の操作/*

パレットの残りの操作はいちいち説明しない。必要に応じて試してみればわかる (分からなければ質問メールをください)。また、細かいところを拡大して見たり全体像のため小さくして見たりしたい場合は窓の左下の矢印群や四角を使う。これもやってみればわかる。最初の見え方に戻したい場合には「Optoin」メニューの「Normal Size」と「Center Page」を使えばだいたい大丈夫。詳しく知りたい場合にはマニュアルを読む (Kterm の窓で「`man idraw`」を実行すると画面に表示される)。

そして最後に! できた絵を保存しなければならない。そのためには、「File」メニューの「Save」と「Save As」を使う。これらはちょうど Mule での`^X^S`と`^X^W`に対応している。だから最初に保存する時は「Save As」を使う。すると、新しい窓が現れてファイル名が入力できるようになるので、そこで「`test.ps`」のようにファイル名を入れて `[RET]` を打つ。

なお、名前の最後が「`.ps`」で終わるのは PostScript(PS) 形式として扱えるファイルであることを示している。(厳密には、idraw が読み書きすのは idraw 形式のファイルなのだが、idraw 形式は PS 形式としても使うことができるので便宜上「`.ps`」をつけている。) 書き出した後 idraw を終了するには「File」メニューの「Quit」を選ぶ。

一度書いた絵を編集するには idraw を動かした状態で「File」メニューの「Open」を使う。するとさっきと同様の窓ができてファイル名が打ち込めるので、そこで打ち込んだ後 `[RET]` を打つ。絵が現れたら適宜修正し、最後に「Save」で元のファイルに書き出すか「Save As」で別のファイルに書く。これらの意味をまとめておこう。なお、前回やった xpaint もまったく同様である。

Mule	idraw/xpaint	機能
`^X^W`	「`Save As`」	ファイル名を指定して書く
`^X^F`	「`Open`」	ファイル名を指定して修正を開始
`^X^S`	「`Save`」	もとのファイルに書き戻す
`^X^C`	「`Quit`」	`Mule` や `idraw` や `xpaint` を終わる

idraw 形式を含め、PS 形式のファイルをプリンタに打ち出す時には `lwp` ではなく `lpr` というコマンドを使用すること。例えば次の通り。

```
lpr -Plw17 test.ps
```

演習 7 △ これまでに自分が描いた PAD 図または資料に載っていたのどれでもいいから、idraw で清書してみよ。なお、同じ形を沢山使う時には 1 つ描いた後それを選択した上「Edit」メニューの「duplicate」を使って複製を作り、適当な位置に移動して使うのがよい。完成したらファイルに保存し、プリンタに打ち出してみよ。PS ファイルを lpr で打ち出すのと、画面ハードコピーを使うのでは出力されたものはどう違うか?

A　本日の課題 6A

本日の課題は演習 1(描いたもの)、演習 2(囲んだもの)、演習 4(Pascal プログラム) を提出してください。あと演習 7 はできた人のみプリントアウトを提出してください。レポート番号は 6A です。アンケートは次の通り。

- Q1. 変数の値の移り変わりについて、前から理解していましたか? 今日ようやく理解しましたか? まだだめですか?
- Q2. idraw と xpaint についてそれぞれどう思いました? また、両者の比較も述べてみてください。
- Q3. その他、感想、要望、質問があればどうぞ。

B　次回までの宿題 6B

次回 (12/4) までの宿題は、演習 3(囲んだもの)、演習 5(難しいので、できたところまで。PAD の途中までででもよい)、演習 6 のどれか 1 つ以上、演習 7(出してない人) とします。ちょっと多いですが、期間が 2 週間ありますから。レポート番号は 6B、アンケートは次の通り。

- Q1. ループについて納得しましたか? どのような感想を持ちましたか?
- Q2. 文を四角で囲むのはできるようになりました?「;」を入れる場所についてはどうですか?
- Q3. その他、感想、要望、質問があればどうぞ。