「情報処理」1年文I/IIクラス9-10　＃13

久野　靖*

1995.2.6

半年に渡っておつき合い頂いた「情報処理」も今回で最終回となりました (おつかれさま!)。前回予告した通り、今回は Pascal プログラムはなく、「計算機をいかに利用するか」というテーマでこれまでに取り上げられなかった話題を簡単に紹介して行こうと思います。本日の内容は次の通り。

- 基本ソフトとアプリケーションソフトについて学ぶ。
- いくつかの代表的な種類のアプリケーションソフトを見学する。
- スクリプト言語による GUI プログラミングを体験する。

1　様々なアプリケーションソフト

1.1　基本ソフトとアプリケーションソフト

計算機の内部では様々なプログラムが動いている、ということはこれまでに散々見てきた。ところで、プログラムは大きく分けて 2 つに分類できる。

- 基本ソフトウェア — 計算機を管理したり、より便利に使ったり、ソフトウェアを開発したりするためのプログラム。
- アプリケーションソフトウェア — それ自体を使って特定の仕事/作業をするためのプログラム。

これまでに学んできたうちで、「OS(Unix)」、「シェル (tcsh)」、「Kterm」、「Mule」、「X サーバ」、「ウィンドウマネージャ(twm)」、「Pascal コンパイラ (pc)」、「xv」、ファイル操作の各コマンドなどは基本ソフトである。つまり、これらのソフト自体で特定の仕事ができるわけではない。(Mule はちょっと微妙で、単に文字を打ち込んだファイルを作るという仕事をしたいのならアプリケーションに分類できるかも知れない。しかし大体は「プログラムを作るために打ち込む」とか「メールを出すために打ち込む」など、別の仕事の一環として Mule を使うことが多い。)

一方、アプリケーションソフトの例としては「xpaint」、「idraw」(絵を描く)、「xmosaic」、「mnews」、MH の各コマンド (情報をやりとりする)、「jlatex」(文書を整形する)、X の各種クライアント、そしてあなたの作った各種のプログラムなどが挙げられる。アプリケーションソフトは簡単にいえば「〜をするためのソフト」なので、基本ソフトに比べてその用途も明確だし使い方も学びやすい。

この科目では、計算機の原理という観点から話題を選んできたので、アプリケーションソフトについてはあまり沢山取り上げなかった。しかし世の中の多くの計算機ユーザは「計算機を使うこと」が目的ではなく、「何かやりたいことがあって、そのための手段として計算機を使う」わけであり、だからその「やりたいこと」をさせてくれるアプリケーションソフトを使っている時間

*筑波大学大学院経営システム科学専攻

が 9 割以上を占めるはずである。(例えばワープロ機というのはワープロというアプリケーションソフトしか動かない計算機だが、それで十分という人が沢山いるから存在するわけだ。) ただし、アプリケーションが動くためには基本ソフトの手助けは常に必要だ、ということは頭に置いておいて頂きたい。

さて、アプリケーションソフトもさらに詳しく見ると次の 2 種類に分かれる。

- パッケージソフト — 「できあい」のソフト。服で言えばレディメードに相当する。
- カスタムソフト — 目的に合わせて作る。オーダーメード。

皆様が「〜を計算するプログラムを作れ」という課題に応じて作っていたのはカスタムソフトに相当する。あの程度の簡単なものなら何ということはないが、一般にはカスタムソフトを作ってもらうのは人件費が高いためものすごくお金がかかる。(銀行のオンラインシステムなどお金を節約するためにライバルのはずの同業数社が共同で作ることさえある。)

一方、カスタムソフトを作ることが飯の種という会社も多数ある。富士通、日電、日立、東芝、三菱、沖、日本 IBM などの計算機メーカのソフト部門もそうだし、「ソフトウェアハウス」と呼ばれるもっと小さい企業は数え切れないほど多数ある。ただしこれらの会社では注文に応じたカスタムソフトだけでなくパッケージソフトも開発していることが多い。(パッケージソフトのみでやっている所も少数ながらある。)

さて、パッケージソフトの方はいわゆるパソコンショップに行くと棚に並んでいたりするのでおなじみである。これは 1 回作ればそれを多数コピーして販売できるので 1 個当たりの価格は安いし、そしてヒットすれば (沢山売れれば) 開発会社も沢山儲かる。ただし、沢山の人が 1 つの同じプログラムを使うわけで、ある程度汎用性 (つまり色々な人が自分の必要に合わせた使い方をできること) が必要である。

ワープロやお絵描きソフトはその典型例で、誰でもきれいに整った報告書や図は描きたいだろうから、パッケージソフトとして開発するのに適している。これらについては一応やったので、以下ではそれらに続く代表的なアプリケーションソフトとして、表計算、統計解析、数式処理の 3 種類を取り上げて見物しよう。なお、あくまで「見物」であるので、それがどんなものでるか大体分かればいいものとする。本当に使いこなしたい場合は参考書を求めて自主的に勉強するように。

1.2 表計算ソフト

xcalc を見て「便利だ」とコメントした人が数名いたが、本当に便利だろうか? 計算したいデータをマウス操作で入力して、計算して、結果を読み取って紙なり Mule の画面なりに控える…データはすべて計算機の中にあるのだから、いちいち人間が手で転記するなんておかしいと思いません?

世の中のデータは「表」の形で表現されることがとても多い。そこで、まず中身が空っぽでとても大きい表を計算機の中に用意する。これを「ワークシート」と呼ぶ。その表の各「箱」は「セル」といい、その中にデータや説明の文字を書き込んで行く。また、データだけでなく計算式を書き込むことで、計算を行わせることができる。(表計算という名前の由来。)

表計算ソフトの代表といえば MS-DOS 向けの Lotus 1-2-3(ワンツースリーと読むように)、Windows と Macintosh 向けの Microsoft Excel だが、ここでは「アシストカルク」を説明する。(基本部分はどれも似ている。) 自分で試す場合には設定のため「`~kuno/setup s2020`」を最初の 1 回だけ実行し、あとは「`s2020 &`」で起動する。

起動直後のアシストカルクの窓の様子を図 1 に示す。中央にワークシートが表示され、その周囲にコマンドメニューなどが表示されている。ワークシート中の各セルは「A0」「B12」など「列の名前」と「行の番号」を組み合わせた名前を持っている。ここでデータ (数値でも文字列でもいい)

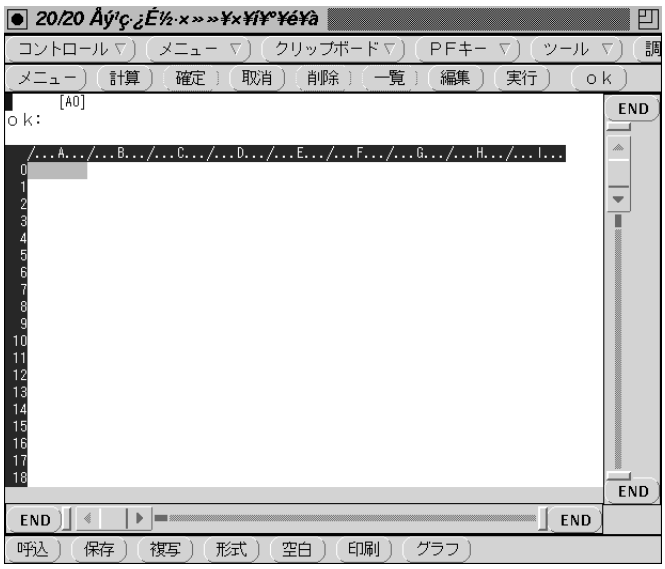

図 1: アシストカルクの起動直後の様子

をセルに入力するには、矢印キーかマウスクリックで反転表示の位置を入力したいセルまで持ってきて、そこで「数値 [RET]」(数値の場合) または「’ 文字列 [RET]」(文字列の場合) と打ち込む。[1]

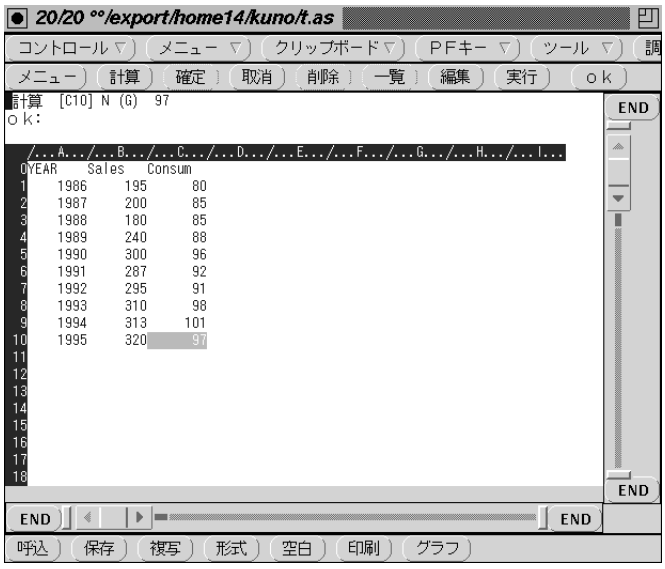

図 2: データを打ち込んだワークシート

図 2 にここ 10 年間のとある企業の売り上げと経費のデータを打ち込んだところを示す。さて、いよいよ計算をしよう。10 年間の経費と売り上げの合計をそれぞれ計算したいとする。B12 のセルに売り上げ合計を計算した結果を表示するには、このセルにデータではなく「sum(B1..B10)[RET]」と計算式を打ち込む。sum(...) は指定した範囲のセルの合計を求める組み込み関数である。打ち込み終わった瞬間に、セルに合計のデータが表示される。しかしこれは計算結果なので、例えば B1〜B10 のデータのどれかを変更するとただちにそれに応じて合計データも変化する。なかなか便利でしょう?

[1] 文字列としては漢字も入れられるはずなのですが、マニュアルが不備でどうしても漢字が入るように設定できませんでした。当面はアルファベットのみとしてください。

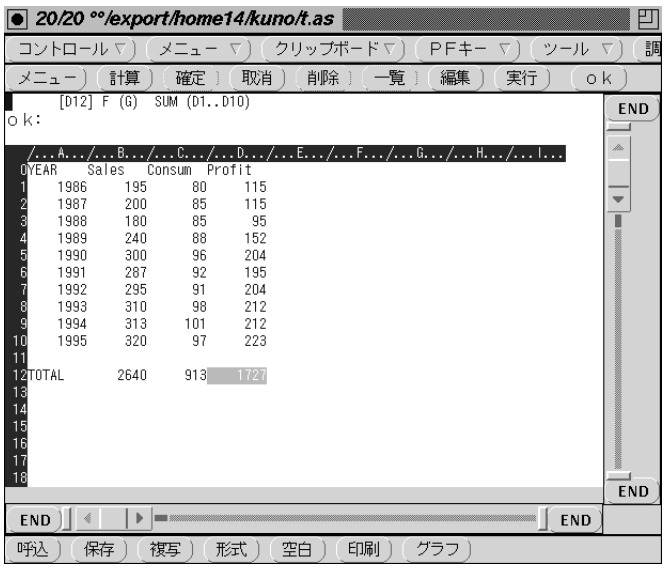

図 3: 合計集計のできたワークシート

同じように、経費合計を C12 のセルに出すには、先と同様に打ち込んでもいいのだが…B12 セルの内容をコマンド操作「/ccb12[RET]c12[RET]」で C12 にコピーしてもよい。この意味は次の通り。

- 「/」を打つと窓の中にメニューが表示される。
- 「c」で複写機能を選ぶ。
- さらに「c」で計算つき複写を選ぶ。
- するとコピーするセルを聞いてくるので「b12[RET]」。
- 次にコピー先を聞いてくるので「c12[RET]」。

単にそのままコピーしたら「sum(B1..B10)」だから意味がなさそうだが、表計算ソフトではコピーすると自動的にセル名も調整されて「sum(C1..C10)」になるのでこれでうまく行く。今度は各年の利益を列 D に出そう。まず D1 に行って「b1-c1[RET] とすると、最初の年の利益が表示される。これを各年にコピーするには先と同様に「/ccd1[RET]d2..d10[RET]」でよい。あとこれにタイトルと利益合計を追加した結果が図 3 になる。どうです、電卓とメモよりずっといいでしょう?

さて、このように表の形になったデータをグラフ化することもできる。ここでは/grb1..d10[RET]v により、売り上げ/経費/利益の 3 つを標準的な折れ線グラフとして表示した (図 x)。「q」を打つと元に戻る。ただし、表計算ソフトのグラフ機能は「ビジネスグラフ」などと呼ばれ、プレゼンテーションに向く程度のものに限られる。

終わる前に打ち込んだデータを保存したい時には/sw ファイル名 [RET] によりファイルに現在のワークシートをそっくり保存できる。そして、終わるには/q とすればよい。

1.3 統計解析ソフト

表計算ソフトの計算やグラフは分かりやすい分、比較的簡単なものに限られている。対象データについてもっと細密な分析をしたい場合には統計解析ソフトなどの方が向いている。ここでは S と呼ばれる Unix 定番のデータ解析ソフト (用途が統計に限られるわけではないのでこう名乗っている) を見てみる。S は、単に「S[RET]」で起動できる。起動すると S のプロンプトが出てコマンドが入力できる状態になる。

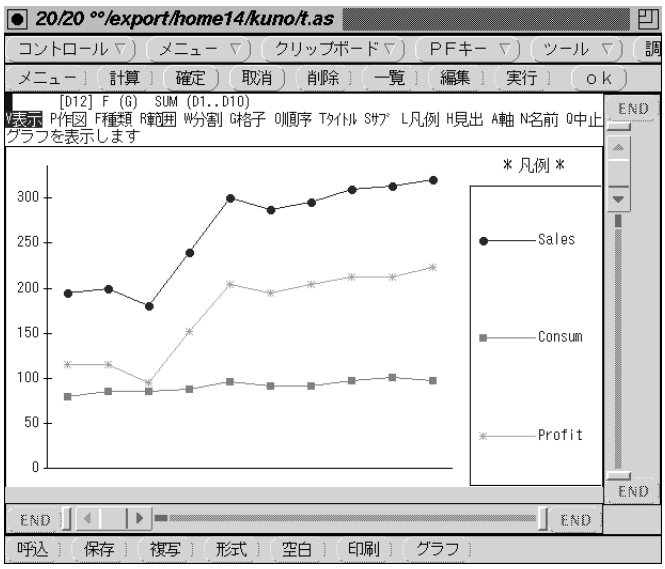

図 4: 折れ線グラフによる表示

```
% S
  S Language version 3.2.2 (S Wed Aug  4 1:11:18 JST 1993)
  Copyright(c) 1988-1993 ISAC,Inc.
  Copyright(c) 1988,1989 AT&T Co.,Ltd.
  SToolkit version 2.2. for OpenWindows
  Copyright(c) 199-1993 ISAC,Inc.
kanji code: EUC
>
```

ここでアシストカルクの時と同じデータを打ち込むには、次のようにする。(「1986:1995」は「1986,1987,...,1995」と打つのと同じ。)

```
> year <- c(1986:1995)
> sales <- 1 c(195,200,180,240,300,287,295,310,313,320)
> consum <- c(80,85,85,88,96,92,91,98,101,97)
```

このように、S では Pascal などと同様、変数に値を入れることによってデータを保持する。ただしいちいち配列などを宣言する必要はなく、c(...) という関数を使うと打ち込んだデータに対応するベクトルが作られる。また値を入れるのは:=ではなく<-で表す。変数に入れた値は、その変数名だけを打ち込むと表示できる。

```
> year
 [1] 1986 1987 1988 1989 199 1991 1992 1993 1994 1995
> sales
 [1] 195 200 180 240 300 287 295 310 313 320
> consum
 [1]  80  85  85  88  96  92  91  98 110  97
```

また、ベクトル同士の引き算や足し算も使えるので、利益を計算するには次のようにすればよい。

```
> profit <- sales - consum
> profit
 [1] 115 115  95 152 204 195 204 212 212 223
```

そして、平均とか最大などの関数も予め用意してある。

```
> mean(profit)
[1] 172.7
```

このように論理式 (条件式) にベクトルを書くと結果もベクトルになる。そして、ベクトルからその条件が「T」(真) であるような要素だけを取り出すこともできる。

```
> profit > 200
 [1] F F F F T F T T T T
> year[profit > 2]
[1] 1990 1992 1993 1994 1995
```

グラフィクスを使いたい場合には、まず出力装置を指定する。ここでは X-Window の画面を指定する。

```
> x11()
Warning messages:
  Color name "MediumForestGreen" was invalid, foreground color used
  Color name "MediumGoldenrod" was invalid, foreground color used
```

警告は無視してよい。画面にグラフィクス用の窓が現れる。そして、この上に年と利益の関係をプロットしてみる。

```
> plot(year, profit)
```

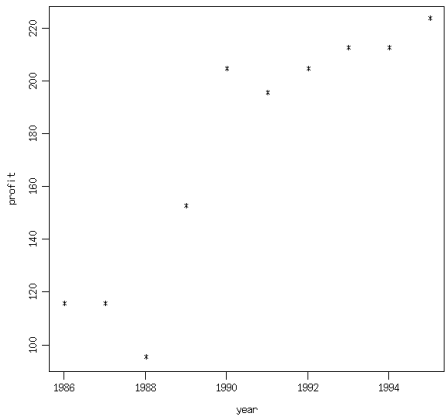

図 5: S のプロット図

結果は図 5 のようになる。ここで、最小自乗法により直線をあてはめて、平均して毎年どのくらい利益が増えているか見てみよう。

```
> reg1 <- lsfit(year, profit)
> abline(reg1)
```

これで先のグラフに直線が引かれる。(これ以上は私は統計には素人なので聞かないように!) 最後に S を終るには「q()」という関数を実行する。

```
> q()
%
```

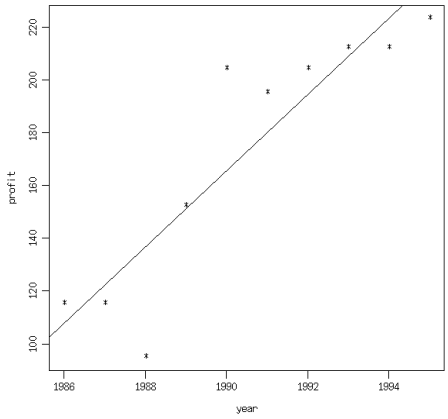

図 6: プロット図に回帰直線を引いたところ

データを保存してない? 実は S ではデータはユーザのホームディレクトリの下に.Data というディレクトリを作ってそこに保存されているので、次に S を起動するとそのままさっきの変数のデータが使える。

1.4 数式処理ソフト

ここまでの計算ではすべて、計算機では「具体的な値」のみを計算してきた。例えば「自乗すると 2 になる数」は「1.414216」という具合である。これを「数値計算」と呼ぶ。しかし、場合によっては一般式、つまり「自乗すると x になる数」は「$\sqrt{x}$」という答を欲しいこともある。そのような処理を「数式処理」と呼ぶ。ここでは数式処理機能を持つデータ解析ソフトである Mathematica を見てみよう。(Mathematica で統計解析をやることもちゃんとできるし、橋爪先生のクラスでは実際そうしているらしいが、ここでは数式処理のさわりだけやる。)

Mathematica を使う場合には、「`~kuno/setup math[RET]`」を 1 回だけ実行しておき、あとは「`math[RET]`」により起動する。[2]

```
% math
Mathematica (sun4) 1.2 (August 22, 1989) [With pre-loaded data]
by S. Wolfram, D. Grayson, R. Maeder, H. Cejtin,
   S. Omohundro, D. Ballman and J. Keiper
with I. Rivin and D. Withoff
Copyright 1988,1989 Wolfram Research Inc.
 -- X11 windows graphics initialized --
In[1]:=
```

ここで、In[番号] というのがプロンプトで、ここに入力を打ち込む。打ち込むものは「数式」である。

```
In[1]:= (x + 1) ^ 2

                    2
```

[2] これは S と同様のコマンドインタフェースを起動する。そのほかに、コマンドの編集が可能なインタフェースや、グラフィクスも一緒の窓に現れる notebook インタフェースも使える。より詳しくは橋爪先生 (ユーザ名 has) の WWW ページを参照のこと。

```
Out[1]= (1 + x)
```

出力は Out[番号]=という形で表示される。数式は数式のまま扱われることに注意。ここで、%番号というとその番号の出力が参照できる。ではこの式を「展開」しよう。

```
In[2]:= Expand[%1]

                     2
Out[2]= 1 + 2 x + x
```

このように、Mathematica では様々な「数式を処理する関数」が用意されている。展開したものを因数分解すると当然元に戻る。

```
In[3]:= Factor[%2]

                2
Out[3]= (1 + x)
```

こんどは積分しよう。

```
In[4]:= Integrate[%2, x]

                   3
              2   x
Out[4]= x + x  + --
                   3
```

さて、自分でも関数を定義できる。その時は (1) 名前は小文字で始めること、(2) 関数の引数 (パラメタ) は名前の後ろに「_」をつけて表す、ということに注意。

```
In[5]:= f[x_] = %4

                   3
              2   x
Out[5]= x + x  + --
                   3
```

こんどはこの関数を x^2 と一緒にグラフに表してみよう。

```
In[6]:= Plot[{f[x], x^2}, {x, -1, 1}]

Out[6]= -Graphics-
```

Plot の 1 番目の引数は式、2 番目の引数は「{変数，始点，終点}」という形のリストである。

今度は、「方程式の解」をやる。

```
In[7]:= Solve[f[x] == 0, x]

                          -3 + Sqrt[-3]          -3 - Sqrt[-3]
Out[7]= {{x -> 0}, {x -> -------------}, {x -> -------------}}
                               2                      2
```

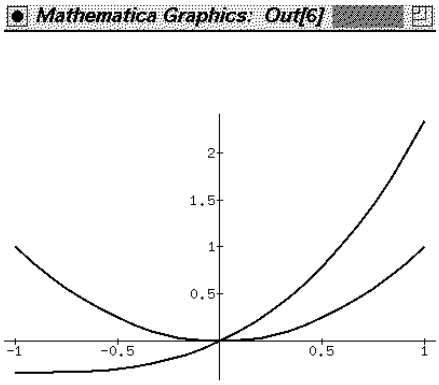

図 7: Mathematica のグラフ

虚数解もちゃんとでて来る。もっと普通ぽいのをやろう。

```
In[8]:= Solve[x^2 + y^2 + z^2 == 1, z]

                          2    2                      2    2
Out[8]= {{z -> Sqrt[1 - x  - y ]}, {z -> -Sqrt[1 - x  - y ]}}
```

この 8 番は「$x^2 + y^2 + z^2 = 1$ という式を z について解く」と読む。この式そのものは 3 次元空間内の球を表しますね? これを改めて g という関数にする。[3]

```
In[9]:= g[x_, y_] = Sqrt[1 - x^2 - y^2]

                   2    2
Out[9]= Sqrt[1 - x  - y ]
```

これを 3 次元プロットしたいのだが、そのままやると球の外のところで z が虚数になるので、絶対値を取ってから 3 次元プロットする。

```
In[11]:= Plot3D[Abs[g[x, y]], {x, -1.2, 1.2}, {y, -1.2, 1.2}]


Out[11]= -SurfaceGraphics-
```

この結果は図 8 のようになる。Mathematica を終りにする時は^D を打てばよい。なかなかすごかったでしょう?

2 Tcl/tk — X の GUI プログラミング用スクリプト言語

2.1 様々な言語と Tcl/tk

前回まで皆様には Pascal 言語によるプログラミングを勉強して頂いたわけだが、世の中にはまだまだ沢山の「計算機用言語」が存在している。Basic とか C とか Fortran とか Lisp とか Prolog とか… が、これらはいずれも「汎用プログラミング言語」であって、機能や性質や用途の点では Pascal によく似ている。

[3]打ち直さなくても取って来られるのだが説明が面倒なので略。

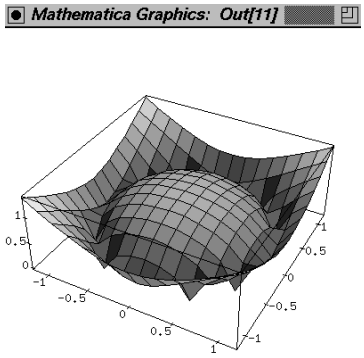

図 8: Mathematica のグラフ

計算機言語にはまだ他にも色々ある。例えば今回やった S や Mathematica やアシストカルクもその中には言語が組み込まれて、それを使って決まった手順を実行したりできる。これらはそれぞれのアプリケーションソフトに適した「専用言語」ということになる。また、シェルスクリプトも実は言語だといってよい。シェルには変数機能もあるし、(説明は省略したが) 判断や分岐といった制御構造もある。シェル語の場合には、コマンドを実行するなどの「高レベルの」動作が簡単に書けるという特徴があるが、反面実行速度はあまり早くない。その他の専用言語もこれに類似している。

さて、前回 X-Window の各種クライアントをやった時、「こんなプログラムを自分でも作れたらいいな」と思いませんでしたか? それは、ものすごく努力すればできるだろうけど、ちょっと皆様のレベルでは大変である。何が大変かというと、X の各種動作を制御するための「定石」が沢山あって、それを見につけるのに時間が掛かってしまう。

しかし、各種コマンドを実行するのに Pascal でなくシェルスクリプトにすると簡単なように、「ボタン」「ラベル」「入力欄」などいくつかの標準的な部品を組み合わせて GUI プログラムを作るだけなら、それ用の専用言語を使うとずっと簡単にできる。そのような言語である Tcl/tk について、簡単に眺めてみよう。

2.2 簡単な例題

まず、一番簡単な GUI プログラムとして、「ボタンがいくつかできて、それを押すとそれぞれ対応するプログラムが動く」というのを考えよう。ファイル launch に以下のような内容を入れてあるとする。

```
#!/usr/local/bin/wish -f
button .btn1 -text "時計"   -command "exec xclock -a -u 1 &"
button .btn2 -text "電卓"   -command "exec xcalc &"
button .btn3 -text "メール" -command "exec xmh &"
button .btn4 -text "終了"   -command "exit 0"
pack .btn1 .btn2 .btn3 .btn4 -fill x
```

まず 1 行目の「`#!/usr/local/bin/wish -f`」というおまじないは、これが普通のシェルスクリプトではなく Tcl/tk の標準実行系である wish によって実行されるべきであることを示している。(1 行目は Tcl/tk ではなく Unix への指示になっている。)

その後 4 行は、各種のボタンを作り出している。ボタンはそれぞれ`.btn1`〜`4`(GUI の部品はすべて「`.`」で始まる名前を持つ) という名前を持っていて、その上には「時計」「電卓」などと書い

てある。そして、そのボタンが押された時には`-command`の次に指定された Tcl/tk のコマンドが実行される。

最初の 3 つは、exec というコマンドが実行されるが、これは「Unix のコマンドを実行する」という Tcl/tk のコマンドである。だからこれらのボタンを押すと先週やったように時計や電卓などの窓が現われる。4 つ目のボタンを押した時は exit というコマンドが実行されるが、これはこの Tcl/tk スクリプトを終了させるというコマンドである。

以上でボタンはできたが、これをこのプログラムが動き出した時の窓の中に配置しなければならない。pack コマンドでボタン名を列挙して、これをたて 1 列に並べる。`-fill x`というのは「ボタンの幅に不揃いがあった場合には一番広いものに合わせてそろえる」という指定である。

このファイル launch はさらに実行可能にしておくものとする。すると、launch を指令として実行させると図 9 のような窓ができる。そして、この窓の中のボタンをつつくことで時計などを出

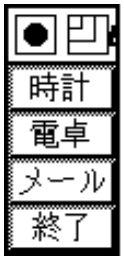

図 9: Tcl/tk の例題 (1)

す事ができる。また終了ボタンをつつくと launch コマンドは終了してこの窓は消える。

改めてこのプログラムを見ると、Pascal やシェルのプログラムとはだいぶ違っている。そもそも、このプログラムはひたすら「部品をならべている」だけであって、普通のプログラムのように入力したり出力したりする部分も制御の流れも実行もなさそうに見える。

実は入力はあるので、それはユーザが「ボタンを押す」ことで行われる。しかしボタンが押された時の細かい処理はボタンという部品の機能として予め用意されているので、わざわざ書かなくてもよい。そして、各ボタンが押されたときの機能の本体は「`-command ...`」の部分に「バラバラに」書かれていて、ボタンを押すという動作 (事象、イベント) があった時個別に実行される。このようなスタイルを「イベントドリブン」といい、GUI プログラミングに固有のスタイルである。

演習 1 launch の例題を打ち込んで動かしてみよ。また、好みに応じて起動するプログラムを変更したり増やしたりしてみよ。気に入ったら`~/bin`に置いてどこからでも使えるコマンドにしてみよ。

2.3 様々な GUI 部品

先の例題では「ボタン」しか使わなかったが、Tcl/tk には多数の GUI 部品が備わっている。「`widget`」というデモプログラムを動かすと、これらを一通り見ることができる。

ここでは次の例題として、図 10 のような窓ができて、oclock のオプションを色々調整した上で起動できる、というのをやってみる。ここには、先に出てきたボタンの他に label(ただの表題)、scale(スライドレバー)、checkbutton(チェックボックス)、radiobutton(選択ボックス) が使われている。

```
#!/usr/local/bin/wish -f
label .lab1 -text "丸い時計の調整"
scale .scl1 -label "幅"   -from 100 -to 600 -orient horizontal -length 200
```

図 10: Tcl/tk の例題 (2)

```
scale .scl2 -label "高さ" -from 100 -to 600 -orient horizontal -length 200
label .lab2 -text "オプション"
checkbutton .chk1 -text "透明" -variable trans -anchor w
label .lab3 -text "内部の色"
radiobutton .rad1 -text Yellow -value yellow -variable color1 -anchor w
radiobutton .rad2 -text SkyBlue -value SkyBlue -variable color1 -anchor w
radiobutton .rad3 -text MistyRose -value MistyRose -variable color1 \
  -anchor w
set color1 yellow
button .btn1 -text "時計の起動"   -command {
  set geom "[.scl1 get]x[.scl2 get]"
  if { $trans } {
    exec oclock -transp -bg $color1 -geom $geom &
  } else {
    exec oclock -bg $color1 -geom $geom &
  }
}
button .btn2 -text "終了"   -command "exit 0"
pack .lab1 .scl1 .scl2 .lab2 .chk1 .lab3 .rad1 .rad2 .rad3 \
  .btn1 .btn2 -fill x
```

まず.lab1 は label で、単に「丸い時計の調整」という表題を表示するだけのものである。次に時計の幅と高さに対応する.scl1、. scl2 という scale を用意する。それぞれ、表題は「幅」「高さ」とし、値の調整範囲は 100〜600 として、水平方向に動かし長さ 200 ドットとする。

次に「オプション」という表題の後に.chk1 という checkbutton を作るが、これは「透明」というラベルを持つ。チェックボタンはその上を 1 回クリックするごとに on/off が切り替わる。現在 on か off かは変数 trans に入るようにする。美観上、ボックスが左 (画面の上を北とすると West 方向) に揃うようにする。

次に「内部の色」という表題の後に、.rad1〜3 という checkbutton を用意する。これらは Yellow/SkyBlue/MistyRose という名前を持ち、押されるとそれぞれ yellow/SkiBlue/MistyRose という値を変数 color1 に入れる。これらのボタンも左に揃える。最初は color1 を yellow に設定しておく。

次に時計の起動ボタンだが、今度はボタンが押された時の処理が長いので処理を「{ }」で囲んで複数行に分けて書く。まず変数 geom に「幅 x 高さ」の形の文字列を入れるが、幅や高さの値は .scl1 と 2 に対して get 命令を実行して埋め込む。([...] は命令を実行してその値をそこに埋め込む働きがある。) その後、if 文で trans が on か off かにより枝分かれし、oclock を exec で起動するが、その際ここまでに設定したオプションをそれぞれ指定しておく。

終了ボタンはさっきと同じ。そして、これまでの部品を全部上から順番に詰めていく。どうですか、ちょっと長いけれどこれくらいなら読めるでしょう? そして、起動ボタンの処理で分かるように tcl/tk というのもちゃんと言語らしく判断の枝分かれができる。(今回は説明しないが手続きやループもちゃんとある。)

2.4 部品の配置

GUI プログラムでは部品を美しく配置することも重要である。ここまでては全部の部品をたて 1 列に並べていたが、それだけではいまいちですね? もっと美しく配置するために、frame(枠) というものを使う。枠の中にはこれまでの窓全体と同様に部品が詰められ、その枠全体をさらに窓の中に詰めることができる。例題として、メールを送信するツール (図 11) を示す。

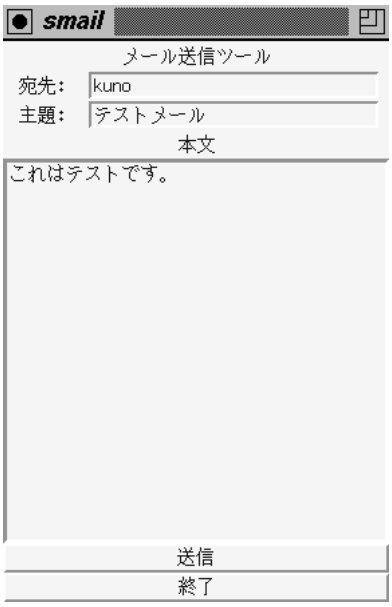

図 11: Tcl/tk の例題 (3)

ここでは、entry(1 行だけ入る入力欄) と text(複数行が入る入力欄) を利用している。そこで、宛先と主題の入力欄の左側にラベルがついているのに注目されたい。つまり、ラベルと入力欄を左右に並べて、それをこれまで同様上から順に詰めているわけである。プログラムは次の通り。

```
#!/usr/local/bin/wish -f
label .lab0 -text "メール送信ツール"
frame .f1
label .lab1 -text "宛先:"
entry .ent1 -relief sunken -textvariable dest -width 30
pack  .lab1 .ent1 -in .f1 -side left -expand 1
frame .f2
label .lab2 -text "主題:"
```

```
entry .ent2 -relief sunken -textvariable subj  -width 30
pack  .lab2 .ent2 -in .f2 -side left -expand 1
label .lab3 -text "本文"
text  .txt1 -width 40 -height 20 -relief sunken -bd 3
button .btn1 -text "送信" -command {
  exec mail -s $subj $dest <<[.txt1 get 1.0 end]
}
button .btn2 -text "終了" -command "exit 0"
pack .lab0 .f1 .f2 .lab3 .txt1 .btn1 .btn2 -fill x
```

つまり、.f1 や.f2 という枠を予め用意しておき、そこの中に label と entry を横につめている。-side left は左からつめること、-expand 1 は幅が足りない場合に引き延ばしを行なうことを示す。次に entry と text の使い方だが、entry では単に「へこんだ形」「値はこの変数」「幅は何文字」のみ指定している。text では加えて「高さは何文字」「ふちの幅は何ドット」も指定している。これらの上でマウスボタンをクリックして入力するとテキストが打ち込める。(Control-\を使えば漢字も打ち込める。ただしローマ字モードから出るには Shift-Space。) 最後に送信ボタンだが、ここでは mail コマンドを exec で実行してメールを送信するが、それに対してテキスト入力欄 .txt1 の内容を最初 (1.0) から最後までまとめて取り出したものを標準入力から与えている。これにより入力欄の内容がメールとして送れることになる。

ほんのさわりだけだったが、tcl/tk を使うと「部品を組み合わせて」それらしい GUI プログラムの外観が作れること、またそのプログラム固有の動作はごく簡単ながらプログラミング言語と同様にして記述できることを理解いただけたかと思う。

A 最後の課題 13A

実は、先週山口 (泰) 先生より依頼があり、最終回には「情報処理」に関する無記名アンケートを実施して欲しいとのことでした。なので、時間内にはアンケート記入の方をお願いし、13A はメールとニュースの復習を兼ねて次のようにします。

まず、メールで kuno あて本日のアンケート回答を送付してください。Subject: は「Report 13A」としてください。アンケート内容は次の通り。期限は今週中とします。

Q1. 本日見聞したアシストカルク、S、Mathematica についてどのような印象を持ちましたか。

Q2. Tcl/tk について、どんなものか大体分かりましたか。また、利用できそうですか。

Q3. この科目全体を通じた感想を述べてください。

次に、このクラスのニュースグループ komaba.lectures.jousho.kuno-1 にニュース記事を 3 個以上投稿していただきます。そのうち 1 個ないし 0 個が通常の記事、残りは既にある記事を参照したフォローアップとしてください。テーマとしては、この科目で学んだ (というか取り上げた内容) ○○を任意に撰び、「○○について学ぶことは我々にとって意味がある」「○○について勉強するのは我々にとって意味がない」のいずれかの主張を行ってください。フォローはそれに対する賛成または反対の議論となります。Subject:も内容を現したものをつけるようにお願いします。「なんとなく私はこう思う」「そうだそうだ」「そんなことはない」というだけの内容の無い記事は却下して数に入れませんので、きちんと理由ないし根拠を述べるようにしてください。期限は来週一杯としますので、試験の空き時間にでも来て気軽に投稿してください。

なお、私は今後駒場に来ることはまずありませんので、紙のレポートの提出は本日以降はできません。メールとニュースは自分の職場から見られますので、皆様の意見をうかがうのを楽しみにしています。では、半年間ありがとうございました。