計算機プログラミング'95 ＃4

久野　靖*

1995.9.21

1　端末入出力

1.1　画面端末

現在の計算機システムではX-Window等に見られるGUIの使用が増えてはいるが、画面端末を前提としたプログラムもまだまだ多数ある。普段お世話になっている「kterm」というのはこの画面端末の真似をするプログラムで、その上で動くソフト(muleの-nwモード、nn、mnews、vi、などなど)はすべて画面端末モードプログラムである。

そこで今回は、画面端末の勉強をするが、格好の題材でもあるので「パッケージ」のようなプログラムの構造化についても併せて扱うことにする。

1.2　画面の大きさを知る — termcapデータベース

画面端末では画面が「何行何文字」かを知ることが基本である(あたりまえ)。そのためには、Unixではtermcapライブラリ(BSD系)またはtermioライブラリ(SystemV系)を利用する。ここではtermcap系の呼び出しについて学ぶが、Solaris(smb)でもこの呼び出しは利用可能である。

```
/* t41.c --- get termial size */
#include <stdio.h>
char tibuf[1024];

main() {
  int co, li;
  if(tgetent(tibuf, "ansi") <= 0) {
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1); }
  li = tgetnum("li");
  co = tgetnum("co");
  printf("%d x %d\n", co, li);
}
```

なお、新しく出てきたライブラリコールの意味は次の通り。

- tgetent --- 指定した種別の端末の属性値データベース項目をバッファに取り込んでくる。ここではANSI端末を前提として端末種別は固定にした。

*筑波大学大学院経営システム科学専攻

- tgetnum --- その端末の数値属性を検索する。「li」が行数、「co」がカラム数 (文字数) を表す。

練習 1 打ち込んで動かせ。窓の大きさを変えて動かすとどうか? なお、ライブラリの指定が必要なので「gcc t41.c -lcurses」のようにしてコンパイルすること。smb を使用すること。

実は、窓の大きさを変えたら「大きさを変えたのでその状況をデータベースに反映してね」というコマンドをその窓において実行しなければならない。具体的には

```
% set noglob; eval ‘resize‘
```

を実行すること。

1.3 画面制御シーケンス

さて、画面端末の (タイプライタ端末と比べた) 特徴は、画面上の任意の位置に文字を書けることだが、これはどうやって実現されているのだろう? 実は、ESC などの制御文字を組み合わせた画面制御シーケンスを送ることで様々な操作が行える。xterm(kterm) のシーケンスに関する資料を別添するので参照のこと。今回使うのは次の 2 つ (!) だけ。

- ESC [L ; C H --- L 行 C 列にカーソルを移動。
- ESC 2 J --- 画面クリア

さっそくこれらを使ったプログラムを見てみる。

```
/* t42.c --- ansi control sequence */
#include <stdio.h>
char tibuf[1024];

main() {
  int co, li, i, j;
  if(tgetent(tibuf, "ansi") <= 0) {
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1); }
  li = tgetnum("li");
  co = tgetnum("co");
  printf("\033[2J");
  for(i = 1; i <= li; ++i) {
    printf("\033[%d;%dH*", i, 1);
    printf("\033[%d;%dH*", i, co);
  }
  printf("\033[%d;%dH*", 1, 1);
}
```

練習 2 打ち込んで動かせ。窓の大きさを変えて (上述の操作を行った上で) 動かすとどうか?

練習 3 このプログラムを直して、任意の適当な模様を描いてみよ。

1.4 バッファリング

さて、このようにして画面のどこででも自由に文字が配置できるようになった。しかしここで、直接任意の位置に文字を書いてしまう代わりに、まず画面全体をメモリ内の配列に入れ、その後で書き出すという方法が使われることも多い。そのようになおしたプログラムを見てみよう。

```
/* t43.c --- buffering and update */
#include <stdio.h>
char tibuf[1024];
char **line;        ←行の並びをいれる。各行は文字のならび。

main() {
  int co, li, i, j;
  if(tgetent(tibuf, "ansi") <= 0) {
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1); }
  li = tgetnum("li");
  co = tgetnum("co");
  line = (char**)malloc(sizeof(char*)*li+1);  ←行の並び配列の用意
  for(i = 1; i <= li; ++i) {
    line[i] = (char*)malloc(co+1);  ←各行の配列の用意
    for(j = 1; j <= co; ++j) line[i][j] = ' ';  ←まず空白に
  }
  printf("\033[2J");                              ←画面もクリア
  for(i = 1; i <= li; ++i)
    if(i*2 <= co) line[li-i+1][i*2] = '*';       ←斜めに「*」
  for(i = 1; i <= li; ++i) {
    printf("\033[%d;%dH", i, 1);                  ←各行の先頭に行き
    for(j = 1; j <= co; ++j) putchar(line[i][j]); ←内容を出力
  }
  printf("\033[%d;%dH", 1, 1);                    ←カーソル左上
}
```

練習 4 このプログラムを動かせ (打ち込まなくてよい)。

練習 5 別の形を追加して描くようにしてみよ。

質問 1 このようにバッファを介在させることの利点は?

1.5 パッケージ/モジュールについて

閑話休題。ここでちょっと時間を取っていただいて、「パッケージ」とか「モジュール」と呼ばれるものについて考えてみよう。こういう言葉について、聞いたことはありますね?

質問 2 パッケージとかモジュールとは、何か?

質問 3 なぜ、そのような概念があるといいのか?

質問 4 今回の画面端末プログラムにそれを適用するとどうなるか?

1.6 画面端末パッケージの仕様設計

では、次のような画面パッケージがとりあえずあるものと考える。

```
scrinit() --- 画面パッケージの初期設定
screnｄ()  --- 画面パッケージの後しまつ
scrlines() --- 画面の行数を返す
scrcols()  --- 画面の文字数を返す
scrclear() --- 画面をクリアする
scrput(int l, int c, int x) --- l 行目の c 文字目に文字 x を入れる
scrupdate() --- 画面を更新する
scrpos(int l, int c) --- カーソルを l 行目の c 文字目に
```

こういうものがあるとすると、さっきのプログラムはこう書ける。

```
/* t44.c --- making it a package */
#include <stdio.h>

main() {
  int i, j, co, li;
  scrinit();
  co = scrcols();
  li = scrlines();
  for(i = 1; i <= li; ++i) scrput(li-i+1, i*2, '*');
  scrupdate(); scrpos(1, 1); screnｄ();
}
```

すっごく簡単でしょう? もう 1 つ、繰り返し画面を書き足していくプログラムも作ってみた。

```
/* t45.c --- multiple updates */
#include <stdio.h>

main() {
  int i, j, co, li;
  scrinit();
  co = scrcols();
  li = scrlines();
  for(j = 0; j < 20; j += 2) {
    for(i = 1; i <= li; ++i) scrput(li-i+1, j+i*2, '*');
    for(i = 1; i <= li; ++i) scrput(li-i+1, j+i*2+1, '*');
    scrupdate(); scrpos(1, 1);
    i = getchar();  ←ここで [RET] を打つと次の画面へ進む
  }
  screnｄ();
}
```

練習 6 上のメインプログラムを打ち込んで動かせ。ただし、パッケージのファイルはコピーしてきて、「gcc t44.c t4s0.c -lcurses」のようにして一緒にコンパイルすればよい。

練習 7 どちらでもいいから、別の図形が出るように直してみよ。

質問 5 パッケージを使うことの利点をさらに思いつきましたか。

1.7 画面パッケージの実現

では、パッケージの実現を示そう。といって、さっきのバッファ版のプログラムを「バラバラに」分解しただけのようなもの。C ではファイルを分けると static と指定された広域変数はそのファイル内だけで参照可能になる。

```
/* t4s0.c --- screen package, initial version */
#include <stdio.h>
static char tibuf[1024];
static char **line;
static int co, li;

/* scrinit -- initialize screen */
scrinit() {
  int i;
  if(tgetent(tibuf, "ansi") <= 0) {
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1); }
  li = tgetnum("li");
  co = tgetnum("co");
  line = (char**)malloc(sizeof(char*)*li+1);
  for(i = 1; i <= li; ++i) {
    line[i] = (char*)malloc(co+1);
  }
  scrclear();
}

/* scrend -- finalize screen */
scrend() { }

/* scrlines -- return no. of lines */
scrlines() { return li; }

/* scrcols -- return no. of columns */
scrcols() { return co; }

/* scrclear -- clear screen */
scrclear() {
  int i, j;
  for(i = 1; i <= li; ++i) {
    for(j = 1; j <= co; ++j) line[i][j] = ' ';
  }
  printf("\033[2J"); fflush(stdout);
}

/* scrput -- put a charcter on the screen */
scrput(int l, int c, int x) {
  if(1 <= l && l <= li && 1 <= c && c <= co) line[l][c] = x;
```

```
}

/* scrupdate -- update screen actually */
scrupdate() {
  int i, j;
  for(i = 1; i <= li; ++i) {
    printf("\033[%d;%dH", i, 1);
    for(j = 1; j <= co; ++j) putchar(line[i][j]);
  }
}

/* scrpos -- position cursor */
scrpos(int l, int c) {
  printf("\033[%d;%dH", l, c); fflush(stdout);
}
```

質問 6 パッケージを作るのは大変ですか? 簡単ですか?

1.8 パッケージの改良

ところで、上のパッケージは `scrupdate()` を呼ぶたびに全画面の内容をすべて書き直す。しかし、実際には 2 つの `update` 呼び出しの間で画面内容は一部しか変化していないのがふつう。だから、変化した近辺だけを書き直すように直せば、端末への転送バイト数がずっと少なくて済む。`X-Window` と `kterm` の時にはどっちでも違わないが、本当の端末を使っている場合には結構違う。

実現であるが、「さっき描いた内容」と「今度描く内容」の両方を知らないといけないから、バッファが 2 つになる。

```
/* t4s1.c --- screen package, efficient update */
#include <stdio.h>
static char tibuf[1024];
static char **line;
static char **lineb;
static int co, li;

/* scrinit -- initialize screen */
scrinit() {
  int i;
  if(tgetent(tibuf, "ansi") <= 0) {
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1); }
  li = tgetnum("li");
  co = tgetnum("co");
  line = (char**)malloc(sizeof(char*)*li+1);
  lineb = (char**)malloc(sizeof(char*)*li+1);
  for(i = 1; i <= li; ++i) {
    line[i] = (char*)malloc(co+1);
    lineb[i] = (char*)malloc(co+1);
  }
```

```
  scrclear();
}

/* scrend -- finalize screen */
scrend() { }

/* scrlines -- return no. of lines */
scrlines() { return li; }

/* scrcols -- return no. of columns */
scrcols() { return co; }

/* scrclear -- clear screen */
scrclear() {
  int i, j;
  for(i = 1; i <= li; ++i) {
    for(j = 1; j <= co; ++j) line[i][j] = ' ';
    for(j = 1; j <= co; ++j) lineb[i][j] = ' ';
  }
  printf("\033[2J"); fflush(stdout);
}

/* scrput -- put a charcter on the screen */
scrput(int l, int c, int x) {
  if(1 <= l && l <= li && 1 <= c && c <= co) line[l][c] = x;
}

/* scrupdate -- update screen actually */
scrupdate() {
  int i, j, l, r;
  for(i = 1; i <= li; ++i) {  ←各行について
    for(l = 1; l <= co; ++l)  ←変更された左端と
      if(line[i][l] != lineb[i][l]) break;
    for(r = co; r >= l; --r)  ←右端とを求める
      if(line[i][r] != lineb[i][r]) break;
    if(l <= r) {              ←行に変更があった時のみ
      printf("\033[%d;%dH", i, l);  ←左端位置から書く
      for(j = l; j <= r; ++j) {
        putchar(line[i][j]); lineb[i][j] = line[i][j];
      }
    }
  }
}

/* scrpos -- position cursor */
scrpos(int l, int c) {
```

```
  printf("\033[%d;%dH", l, c); fflush(stdout);
}
```

練習 8 最後のメインプログラムで、パッケージの最初の版と今度の版でどれくらい転送量が違うか計測してみよ。

質問 7 パッケージを使うことの利点をさらに思いつきましたか。

1.9 端末の入力処理の変更

さて、先のメインプログラムでは入力はあったが [RET] しか入れられなかった。なぜか? 通常モードでは、端末入力は 1 行ぶんが OS によってバッファされ、[RET] が打たれるとはじめてプログラムに渡されるようになる。なぜか?

しかし、画面制御を自分で行うプログラムでは、OS のこのような処理はじゃまである。で、OS に「このような処理はしないで」と頼むシステムコールがあるのだが…これがものすごく大変である。(なんで? というくらい。) この辺は Unix のオーバーデザインだと思うのだけど…

とりあえず、この機能を使って画面ライブラリ使用中は OS が余計な処理をしないように切り替えるようにして、あと 1 文字入力用の関数を追加した版を示す。

```
/* t4s2.c --- screen package, input handling */
#include <stdio.h>
#include <termios.h>
static char tibuf[1024];
static char **line;
static char **lineb;
static int co, li;
static struct termios ttb1, ttb2;

/* scrinit -- initialize screen */
scrinit() {
  int i;
  if(tcgetattr(0, &ttb1) != 0) {  ←現在の設定を取得
    fprintf(stderr, "cannot get termnal controls.\n"); exit(1); }
  ttb2 = ttb1;
  ttb2.c_iflag |= IGNBRK;          ←「何もしない設定」を用意
  ttb2.c_iflag &= ~ICRNL&~ISTRIP&~IXOFF;
  ttb2.c_oflag &= ~ONLCR&~OCRNL&~ONLRET;
  ttb2.c_cflag |= CS8;
  ttb2.c_cflag &= ~PARENB;
  ttb2.c_lflag |= TOSTOP;
  ttb2.c_lflag &=
   ~ISIG&~ICANON&~ECHO&~ECHOE&~ECHOK&~ECHONL&~ECHOCTL&~ECHOPRT;
  for(i = 0; i < NCCS; ++i) ttb2.c_cc[i] = -1;
  ttb2.c_cc[VMIN] = 1;
  ttb2.c_cc[VTIME] = 0;
  if(tgetent(tibuf, "ansi") <= 0) {  ←こここから前と同じ
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1); }
```

```
  li = tgetnum("li");
  co = tgetnum("co");
  line = (char**)malloc(sizeof(char*)*li+1);
  lineb = (char**)malloc(sizeof(char*)*li+1);
  for(i = 1; i <= li; ++i) {
    line[i] = (char*)malloc(co+1);
    lineb[i] = (char*)malloc(co+1);
  }
  scrclear();                          ←ここまで前と同じ
  tcsetattr(0, TCSANOW, &ttb2);        ←何もしない設定に
}

/* scrend -- finalize screen */
scrend() {
  tcsetattr(0, TCSADRAIN, &ttb1);  ←最初の設定に戻す
}

/* scrgetc -- read 1 char from keyboard */
scrgetc() {                            ← read で 1 文字読む
  char buf[1]; read(0, buf, 1); return buf[0];
}
ここから後ろは前と同じ
```

これを使って前のプログラムを動かせば、「任意のキーを 10 回押す」ようにできる。vi の「hjkl」キーで絵を描くというプログラムも作ってみました。

```
/* t47.c --- non-echo input */

main() {
  int i, j, c, co, li;
  scrinit();
  co = scrcols();
  li = scrlines();
  i = j = 10; scrpos(i, j);
  while((c = scrgetc()) != 'q') {
    if(c == 'h') --j;
    if(c == 'j') ++i;
    if(c == 'k') --i;
    if(c == 'l') ++j;
    scrput(i, j, '*'); scrupdate(); scrpos(i, j);
  }
  scrpos(li, 1); scrend();
}
```

練習 9 打ち込んで動かせ。次に、連続していない絵も描けるように、絵を描くモードと描かずにカーソルだけ動かすモードと切り替えられるように直してみよ。

課題 1 このパッケージを利用して何か役に立つプログラムを作ってみよ。たとえばエディタとか

ページャとか。