ヒューマンインタフェース'99 ＃1

久野　靖*

1999.4.9

はじめに

□ 本科目の目的→人間と計算機のインタフェースを担うソフトウェアの基本的な考え方/枠組み/実現技術について学ぶこと。

□ 各回の進め方

- 前半: レクチャー (各回ごとに別テーマ)
- 後半: 実習 (各回のテーマを題材にする)

□ 成績評価→出席、各回の課題内容

1　ユーザインタフェースの基礎概念

□ ユーザインタフェースとは?

□ ユーザインタフェースの歴史

□ ユーザインタフェースで何が問題か?

□ ユーザインタフェースの評価方法

□ 古典的なユーザインタフェース

- コマンド入力
- フォーム入力
- メニュー入力

□ 実験:　古典的なユーザインタフェースの実装と比較

2　ユーザインタフェースとは?

□ インタフェース (界面) →「接するところ」

- ユーザインタフェース?　ヒューマンインタフェース? ユーザ (人間) と「何の」インタフェース?

□ 「計算機と人間がやりとりする部分」

- 「対話的入出力」(cf.　ファイル入出力、プリンタ、…)
- Human Computer Interaction (HCI) という言い方が一般的に
- CHI (Computer Human Interaction) --- SIGCHI, CHI'99 (カンファレンス) → 語呂がいいから (人間より計算機が高価だったなごり?)

□ HCI とかいうと「認知科学的に」とか「実験」とかが欠かせない気がして…→比較的「古典的な」UI という用語を使う (久野)

3　ユーザインタフェースで何が問題?

□ 計算機の…

- 計算機の処理速度?
- 計算機の所要記憶容量?
- 計算機のプログラミングの複雑さ?

□ 人間の…

- 人間の操作所要時間?
- 人間の記憶負荷?
- 人間の学習/習熟負荷?
- 人間の疲労?
- 人間のたのしさ?

4　ユーザインタフェースの歴史

□ 計算機科学基礎でやりましたが…

□ 計算機のはじまり→コンソールパネルのランプとスイッチ

- 現在でも「マシン語やハードウェアを教えるにはこれがベスト」と言っている人が沢山いる
- それはなぜ?　何が重要なの?
- 直接操作感 vs 計算機内部のイメージ→プログラミングの得手不得手にも通じる?

□ バッチシステムの時代→パンチカードで「ジョブ」を作成→提出→オペレータが実行→出力を返却 (数時間のターンアラウンドタイム)

*筑波大学大学院経営システム科学専攻

- これは対話性という点では最悪だが…
- 「プログラムをじっくり見直す」という点ではよかったという節も。
- 結局、何がどうであれば「よりよい」と考えるか?
- とはいえ、バッチシステムは「計算機の利用効率」に偏した方式。

□ 対話的に計算機を使う===「TSS」「ミニコン」の時代

- 入出力機器→テレタイプ端末
- 圧倒的に出力がのろい!(毎秒 10 文字)…でも「いいこと」も。
- 対話的な計算機の使用→画期的
- 対話型エディタ (テレタイプエディタ) → QED、teco、…→「めくらでの編集」しかしそれでも以前のパンチカード打ち直しや紙テープ切り貼りよりずっと嬉しい。

□ その後→画面端末、回線の高速化

- 「画面エディタ」→「見ながら」編集できる→画期的

□ 画面端末…「決まった位置に決まった大きさの文字」しか表示できない

□ これに対する変革→ビットマップディスプレイ。ALTO(最初のワークステーション) → XEROX の WS、PERQ、Sun、Apollo、…(ワークステーション文明)、LISA → Mac → Windows(パソコン文明)

- この辺に関する話題は # 2 で。

□ ともかく、ユーザインタフェースに関するトレンドは「計算機の資源は豊富に使ってよいから、ユーザにサービスする」方向。昔の「計算機が貴重だった時代」とは対照的。

- だからとってむやみに飾ればいいわけではない…

5 ユーザインタフェースの課題

□ 一般に目標とされること…

- 単純さ/簡潔さ
- 表現能力 (コマンドのバリエーション、…)
- ユーザの満足 (制御感、親切、…)
- コスト (実現コスト、学習コスト、…)

6 モデルとメタファ

□ ユーザに一貫したモデルを提示→ユーザの認知負荷軽減、学習しやすさ、記憶しやすさ、予測しやすさに貢献

- 物理法則に従う、等…
- しかし、計算機で扱うものすべてにユーザと共有できるモデルが存在するわけではない

□ メタファ→ユーザがよく知っているものに「比喩」

- デスクトップメタファ、等…
- 完全に「もの」と同じにはできない (机の上を整理するのと同じくらい労力が掛かるのでは無意味) → 計算機を使うなりの何らかの「magic」が必要 (でなければ計算機を使う意味はない)。

7 ユーザインタフェースの評価方法

□ 前述の目標を多く満足するほどよい

- どうやって達成するか考えるだけでなく、そのことをどうやって評価/検証するかが問題。

□ 明らかな答えの 1 つ: 主観的評価

- 侮るべきでないが、やはりこれだけでは困る。

□ 明らかな答えの 1 つ: 実験をする

- どうやって実験をするか?
- 例: 決まった動作をさせて、時間を計測する。
- 実験で測っているのは何を測っているのか?
- 測っているものは本当に測りたいものか?

□ この先は次回の「おたのしみ」です (宿題あり)。

8 GUI 以前の代表的なユーザインタフェース方式

□ とりあえず、画面端末までででデータ入力などに使われていたものでいうと:

- コマンド方式: コマンドやオペランドを 1 行ないし複数行で打ち込んで行く
- 書式 (フォーム) 方式: 記入欄を持つ画面に必要項目を記入し行く
- メニュー方式: メニューが表示され、その中から項目を選択して行く

□ それぞれの利点/欠点は何だと思います?

8.1 コマンド方式

□ コマンド、オペランドを順に打ち込んで行く

```
Q> ship prod=QS190241 cust=15141 qty=50 date=4/30/99
```

- パラメタ、書き方などすべて覚えている必要がある
- その代り高速にできる (設計しだい)
- エラー検出のワザが多数使用できる

□ その他の特性:

- 「位置パラメタ」と「キーワードパラメタ」がある。
- 現在ではコンプリーション (補完) のようなワザも使える。

8.2 フォーム方式

□ 穴埋め書式が画面に表示され、それを埋めて行く

```
---------------------------------

Command: _____

Prod-ID: _____    QTY:  _______

Cust-ID: _____    Date: _______

---------------------------------
```

- 「何を打ち込むか」は覚えてなくてよい
- 「どのように打ち込むか」は覚える必要がある (しかし大抵は自明)
- 必要なら「どのように」の説明もフォームに含めることができる

□ その他の特性:

- 欄によっては「あけたまま」にしておくこともある
- 打ち込む位置の移動はタブなどのキー操作で行うのが普通

8.3 メニュー方式

□ 画面がメニューになっていて、そこから項目を選択すると次の画面へ進む

```
---------------------------------

   Command:
   1: Ship
   2: Order
   3: Query Stock
   9: EXIT

---------------------------------
```

- 「どれか 1 つを選択するだけ」だから極めて簡単、何も覚えておく必要がない→初心者むけ
- その時点で選択可能でない選択肢は始めから選べない
- その代りのろい! 見る→考える→選択操作→次の画面→見る…
- 簡単に迷子になってしまう
- 最後まで全部メニューには普通できない (データ入力とか…)
- なのでフォームと組み合わせることが多い

□ その他の特性:

- GUI の (マウスで選択する) メニューとはちょっと違う (マウスだとさらに遅い)

9 設計演習

□ あるシステムのユーザインタフェースを上記の 3 方式で設計する

□ シナリオ: 専門のオペレータが使うインタフェース

- 備わっている機能: 「発送処理」「補充注文」「在庫確認」「終了」
- どの処理かの選択は行うが、実際には「発送処理」の部分だけ実装する
- 「発送処理」に必要な項目: 「商品番号」「数量」「顧客番号」「発送日」

9.1 データ例

□ データの例を示す (設計に当たって本物のデータを見る事は重要!)

- 商品番号の例: (入力伝票には番号と名前が両方書かれている)

```
0151 Steel Desk (Large)
0152 Steel Desk (Small)
1044 Pine Table
1241 Pine Bench (Long)
2415 Pine Chair
5514 Old Fashoned Swing
4156 Color Pencil Set
0843 Convertible Sofa
5141 Magazine Rack
```

- 顧客番号の例: (入力伝票には番号と名前が両方書かれている)

```
A0012 Jack Lennon
A1021 Loan Channon
A1581 Susan McDonald
B0141 Julian Garland
```

```
B1042 Suna Alford
A4015 Canon Hufford
B1411 Judy Barman
C4105 Lenox Bullis
B0015 Alenn Gummon
```

9.2 演習

□ 上記のような条件で、3 方式 (コマンド、フォーム、メニュー) の入力アプリケーションを設計せよ。

- 画面例を描くこと (画面をスケッチすることはよい設計の手段)

□ 自分/他人の設計を次の点から検討せよ

- その設計だとどのような入力間違いが発生するか
- その設計だとどの部分で時間が掛かると思うか
- その設計だと 1 項目あたりの入力時間はどれくらいだと思うか

□ 上記の検討で問題となった事柄を解消する手段を検討せよ

10 コマンド入力

□ コマンド入力のためのライブラリルーチンを作ってみた。

- cmdinput: プロンプトを出し、1 行入力。入力した行の内容を空白で区切り、各かたまりを 1 つの文字列にして渡された配列に入れる。
- cmdprefix: ある文字列が別の文字列のプレフィクスになっているかどうかを調べる。
- cmdrecog: ある文字列が指定した複数の語のどれのプレフィクスになっているかを調べる。

10.1 コード: command.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAXBUF 4096

cmdinput(char *msg, char *words[], int n) {
  static char buf[MAXBUF];
  int c, i, j = 0, p = 0, l = 0;
  char *s;
  printf("%s ", msg); fflush(stdout);
  while((c = getchar()) != EOF && c != '\n') {
    if(l+2 < MAXBUF) buf[l++] = c;
  }
  buf[l++] = ' ';
  for(i = 0; i < l; ++i) {
    if(isspace(buf[i]) && p < i) {
      buf[i] = 0; c = strlen(buf+p); s = malloc(c+1);
      strcpy(s, buf+p); if(j < n) words[j++] = s;
      p = i+1;
    } else if(isspace(buf[i])) {
      p = i+1;
    }
  }
  return j;
}

cmdprefix(char *p, char *s) {
  int i;
  for(i = 0; ; ++i) {
    if(p[i] == 0 || p[i] == '=') return i;
    if(p[i] != s[i]) return -1;
  }
}

cmdrecog(char *cmd, char *words[], int n) {
  int p = 0, len = 0, i, k;
  for(i = 0; i < n; ++i) {
    int k = cmdprefix(cmd, words[i]);
    printf("(%s,%s,%d)\n", words[i],cmd,k);
    if(k < 0) {
      ; /* do nothing */
    } else if(k > len) {
      len = k; p = i+1;
    } else if(k == len) {
      p = -1;
    }
  }
  return p;
}
```

10.2 コード: t30.c — テストドライバ

```
static char *cmds[] = {"exit", "edit", "edu", "bye", "good"};

main() {
  char *w[100];
  int i, l;
  while(1) {
    while((l = cmdinput("Q>", w, 100)) == 0) ;
    for(i = 0; i < l; ++i) printf("%s\n", w[i]);
    i = cmdrecog(w[0], cmds, sizeof(cmds)/sizeof(char*));
    printf("%d\n", i);
    if(i == 1) exit(0);
  }
}
```

10.3 演習

□ 上記のライブラリを利用して先のデータ入力プログラムを作成してみよ。

11 画面端末の制御

□ 画面端末: エスケープシーケンスで制御

- あくまで最後に描いたものが残っている
- プログラム内部: 「これこれの位置に文字を描く」と言いたい

□ 使いやすい界面と効率よい描き直しを実現するライブラリを作ってみた

- 内部バッファに画面の写しを保持し、変更箇所のみを端末に送信
- scrinit, scrend: 初期設定、後しまつ (端末の設定を変更)
- scrgetc: 1 文字入力
- scrlines, scrcols: 画面の行数、カラム数を取得
- scrclear: 画面を全クリア
- scrputc, scrputs: 指定位置に文字/文字列を書き込む
- scrputl: scrputs と同様だが行末までクリア
- scrupdate: 変更内容を画面に反映
- scrpos: カーソルを指定位置に置く

11.1 コード: screen.c

```
/* screen.c --- screen package, input handling */
#include <stdio.h>
#include <termios.h>
static char tibuf[1024];
static int **line;
static int **lineb;
static int co, li;
static struct termios ttb1, ttb2;
#define HL 0x80000000

/* scrinit -- initialize screen */
scrinit() {
  int i;
  if(tcgetattr(0, &ttb1) != 0) {
    fprintf(stderr, "cannot get termnal controls.\n");
    exit(1); }
  ttb2 = ttb1;
  ttb2.c_iflag |= IGNBRK;
  ttb2.c_iflag &= ~ICRNL&~ISTRIP&~IXOFF;
  ttb2.c_oflag &= ~ONLCR;
  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) {
    fprintf(stderr, "cannot find terminfo entry.\n"); exit(1);
  li = tgetnum("li");
  co = tgetnum("co");
  line = (int**)malloc(sizeof(char*)*li+1);
  lineb = (int**)malloc(sizeof(char*)*li+1);
  for(i = 1; i <= li; ++i) {
    line[i] = (int*)malloc(sizeof(int)*(co+1));
    lineb[i] = (int*)malloc(sizeof(int)*(co+1));
  }
  scrclear();
  tcsetattr(0, TCSANOW, &ttb2);
}

/* scrend -- finalize screen */
scrend() {
  tcsetattr(0, TCSADRAIN, &ttb1);
}

/* scrgetc -- read 1 char from keyboard */
scrgetc() {
  char buf[1]; read(0, buf, 1); return buf[0];
}

/* 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);
}

/* scrputc -- put a charcter on the screen */
scrputc(int l, int c, int x, int h) {
  x &= 0x7f; if(h) x |= HL;
  if(1 <= l && l <= li && 1 <= c && c <= co) line[l][c] = x;
}

/* scrputs -- put a string on the screen */
scrputs(int l, int c, char *s, int h) {
  while(c <= co && *s) {
    scrputc(l, c, *s, h); ++c; ++s; }
}

/* scrputl -- put a string on the screen */
scrputl(int l, int c, char *s, int h) {
  while(c <= co && *s) {
    scrputc(l, c, *s, h); ++c; ++s; }
  while(c <= co) {
    scrputc(l, c, ' ', 0); ++c; }
}

/* scrupdate -- update screen actually */
scrupdate() {
  int i, j, l, r, h = 0;
  for(i = 1; i <= li; ++i) {
```

```
    for(l = 1; l <= co; ++l)
      if(line[i][l] != lineb[i][l]) break;
    for(r = co; r >= l; --r)
      if(line[i][r] != lineb[i][r]) break;
    if(l <= r) {
      printf("\033[%d;%dH", i, l);
      for(j = l; j <= r; ++j) {
        if(line[i][j]&HL) {
          if(!h) { printf("\033[7m"); h = 1; } }
        else {
          if(h) { printf("\033[0m"); h = 0; } }
        putchar(line[i][j]&0x7f);
        lineb[i][j] = line[i][j];
      }
    }
  }
  if(h) printf("\033[0m");
}

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

11.2 コード: t47c.c — 画面お絵描き

```
/* t47c.c --- screen demo */
#include <stdio.h>

main() {
  int i, j, c, co, li;
  int mode = 1;  /* 1:draw, 2:erase, 0:pass */
  scrinit();
  co = scrcols();
  li = scrlines();
  i = j = 10; scrpos(i, j);
  while((c = scrgetc()) != 'q') {
    if(c == 'd') mode = 1;
    if(c == 'e') mode = 2;
    if(c == 'p') mode = 0;
    if(c == 'h') --j;
    if(c == 'j') ++i;
    if(c == 'k') --i;
    if(c == 'l') ++j;
    if(mode == 1)
      scrputc(i, j, '*', 0);
    else if(mode == 2)
      scrputc(i, j, ' ', 0);
    scrupdate(); scrpos(i, j);
  }
  scrpos(li, 1); scrend();
}
```

12 大きな単位での配置

□ 任意の文字列を画面上の任意の場所に置くようにした。

- layout_create: N 個の「文字列」が置けるように用意
- layout_sets: i 番目の「文字列」を設定/変更
- layout_toggle: i 番目の「文字列」の反転/非反転をトグル
- layout_update: 変更を画面上に反映

12.1 コード: layout.c

```
typedef struct unit {
  int l, c, h; char *s; } unit_t;

typedef struct layout {
  int size;
  unit_t *body; } layout_t;

layout_t *layout_create(int size) {
  layout_t *lay = (layout_t*)malloc(sizeof(layout_t));
  lay->size = size;
  lay->body = (unit_t*)malloc(sizeof(unit_t)*size);
  return lay;
}

layout_sets(layout_t *lay, int n, int l, int c, char *s) {
  if(n >= 0 && n < lay->size) {
    lay->body[n].s = s;
    lay->body[n].l = l;
    lay->body[n].c = c;
    lay->body[n].h = 0;
  }
}

layout_toggle(layout_t *lay, int n) {
  if(n >= 0 && n < lay->size)
    lay->body[n].h = !lay->body[n].h;
}

layout_update(layout_t *lay) {
  int i;
  for(i = 0; i < lay->size; ++i) {
    unit_t *u = &lay->body[i];
    scrputl(u->l, u->c, u->s, u->h);
  }
}
```

12.2 コード: t51.c — 配置のデモ

```
/* t51.c -- layout interface test */

typedef void *layout_t;
layout_t *layout_create();

main() {
  int c, li;
  layout_t *lay = layout_create(4);
  layout_sets(lay, 0, 2, 5, "aaaa");
  layout_sets(lay, 1, 4, 5, "bbbb");
  layout_sets(lay, 2, 6, 5, "cccc");
  layout_sets(lay, 3, 8, 5, "dddd");
  scrinit();
  li = scrlines();
```

```
  layout_update(lay);
  scrupdate(); scrpos(li, 1);
  while((c = scrgetc()) != 'q') {
    layout_toggle(lay, 2); layout_update(lay);
    scrupdate(); scrpos(li, 1);
  }
  scrend();
}
```

13 フォーム機能

□ レイアウト機能を利用すればフォームは簡単

- formfill: フォームを表示し記入。渡す配列の奇数番目はタイトル、偶数番目は入力欄になっている (呼ぶ側で配列をそのように設定して呼ぶ)

13.1 コード: form.c

```
formfill(char *mes, char *form[], int n) {
  int sel = 0;
  int li = scrlines();
  int i, c;
  char *s;
  scrclear();
  scrputs(1, 1, "[TAB]: next, [RET]: end", 0);
  scrputs(li, 1, mes);
  while(1) {
    for(i = 0; i < n; ++i) {
      int h = (sel == i);
      scrputs(i*2+3, 15-strlen(form[2*i]), form[2*i], h);
      scrputl(i*2+3, 17, form[2*i+1], h);
    }
    scrupdate(); scrpos(li, 1); c = scrgetc();
    s = form[2*sel+1];
    if(isprint(c)) {
      i = strlen(s); s[i] = c; s[i+1] = 0; }
    else if(c == '\b' || c == '\177') {
      i = strlen(s)-1; if(i >= 0) s[i] = 0; }
    else if(c == '\t') {
      ++sel; if(sel >= n) sel = 0; }
    else if(c == '\r') {
      break; }
  }
}
```

13.2 コード: t52.c — フォームのデモ

```
/* t53.c -- form filling test. */

char f_name[100] = "", f_age[100] = "", f_occ[100] = "";
char *f1[] = {
  "Name:", f_name,
  "Age:", f_age,
  "Occupation:", f_occ };

main() {
  int i;
  scrinit();
  i = formfill("Fill your info.", f1, 3);
  scrend();
  printf("your name:%s, age:%s, occupation:%s\n",
         f_name, f_age, f_occ);
}
```

13.3 演習

□ 上記のフォームライブラリを使ってデータ入力プログラムを作れ。

14 メニュー

□ メニューも同様にできる (フォームより簡単)

- menuselect: 配列にメニューの項目を用意

14.1 コード: menu.c

```
menuselect(char *mes, char *menu[], int n) {
  int sel = 0;
  int li = scrlines();
  int i, c;
  scrclear();
  scrputs(1, 1, "^N: down, ^P: up, [RET]: select", 0);
  scrputs(li, 1, mes);
  while(1) {
    for(i = 0; i < n; ++i) {
      int h = (sel == i);
      scrputs(i*2+3, 4, menu[i], h);
    }
    scrupdate(); scrpos(li, 1); c = scrgetc();
    if(c == '\r') {
      break; }
    else if(c == ('P' - '@')) {
      --sel; if(sel < 0) sel = n-1; }
    else if(c == ('N' - '@')) {
      ++sel; if(sel >= n) sel = 0; }
  }
  return sel;
}
```

14.2 コード: t52.c — メニューのデモ

```
/* t52.c --- menuselect test. */

char *m1[] = {
  "0. Male.",
  "1. Female.",
  "2. Neutral" };

main() {
  int i;
  scrinit();
  i = menuselect("Choose your sex.", m1, 3);
  scrend();
```

```
  printf("your selection was: %d\n", i);
}
```

14.3 演習

□ メニューを使ったデータ入力プログラムを作れ。

□ 3 つのバージョンでデータ入力を行って結果を比較せよ。