ヒューマンインタフェース'99 ＃2

久野　靖*

1999.4.16

はじめに

□ 前回は「インタフェースを設計してみる」ところまでで時間切れになってしまいました。

□ 今回は「作って動かす」ところから始めて、その後で「計算機入力の人間学」ぽい話をしましょう。

1　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、画面端末までの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

1.1　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。

1.2　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

1.3　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…

*筑波大学大学院経営システム科学専攻

- 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)

2 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」

2.1 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
```

2.2 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

3 C コンパイラの使い方

□ ソースファイル: すべて「XXXX.c」のように、最後に「.c」をつける。

□ 今回の例題は termcap ライブラリを使うので次のようにする。

```
% gcc file1.c file2.c file3.c -ltermcap  ←コンパイル
% ./a.out                               ←実行
```

4 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。

4.1 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;
}
```

4.2 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);
  }
}
```

4.3 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。

5 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

5.1 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);
}
```

5.2 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();
}
```

6　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

6.1　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);
  }
}
```

6.2　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();
}
```

7　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)

7.1　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; }
  }
}
```

7.2　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);
}
```

7.3 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。

8 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

8.1 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;
}
```

8.2 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);
}
```

8.3 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。

- 入力データは別途用意してあります。

9 「計算機入力の人間学」

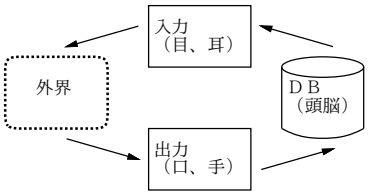
□ …ギルブ、ワインバーグ著、木村、米澤訳、共立、1986。

□ …「古い」ところもあるが (e.g. パンチ室…)、古くないところも。

10 ループを閉じる

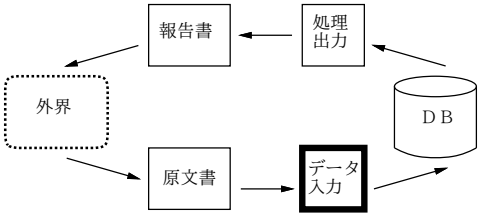
10.1 とある個人企業…

□ ループが閉じられている。間違いがあれば修正される。



10.2 計算機の導入

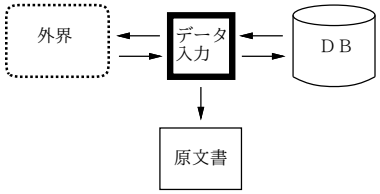
□ ループをまわって来るまで間違いは修正できない。



10.3　オンライン対話入力

□ ループを閉じることができる

□ 「原文書」は単なる「出力」になる



11　省略時の解釈

11.1　省略時の解釈とは?

□ 忘れた時は何が起きるか?→省略時の解釈

- 「やらないことはやりそこないようがない」

□ 次のような文を考えて見る。

- ジョンは車で食料品店に行き、ジョンはパンを買った。
- ジョンは車で店に行き、ジョンはパンを買った。
- ジョンは車で店に行き、パンを買った。
- ジョンは車で行き、パンを買った。
- ジョンはパンを買った。
- ジョン。
- (間)
- (まったく何もしない)

11.2　数量的評価

□ あるデータに対する自然な (頻度の高い値) は?　(例: 1、1 ダース、…)

- 頻度分布をもとに省略の効果を求めることができる。
- 頻度分布はあくまで仮定だということを忘れないこと。
- 頻度分布の仮定に対する感度も重要。

11.3　構造の利用

□ 伝票などの構造→顧客は同一→省略時の値

11.4　省略時の値の適用

□ 物品ごとの性質→省略時の値を変化

- 物品ごとに省略時の値を覚えるのは負担
- 省略時の値をフィードバック
- 省略時の値を使わなかったときに教える
- 省略時の値を使うことも使わないことも許容されるべき

12　出現順の指定

12.1　出現順の指定とは

□ 項目を決まった順序で入れる

- cf.　項目名と項目値のペアで入れる
- 「位置パラメタ」「キーワードパラメタ」(IBM 用語?)

□ 固定長は数えなければいけないからよくない

- 可変長が読みやすく自然、個数も自由

12.2　欄の区切り

□ 出現順→欄を区切る必要がある

- 例:　空白、カンマ、…
- IBM JCL ではカンマが区切り、空白はコメント開始 (!) だった

12.3　末尾以外での出現順指定

□ 出現順指定は末尾だけで使われることが多かった

- しかし範囲が分かれば真中で使うこともできる

```
AAA 3942 6869 Yasushi Kuno 1999.6.1
ABA 045 4011 2845 Yoshiko Kuno 1999.8.25
ACD 869 James Willam Galls 1999.9.1
```

□ 「/」などの特別な区切りを使って再同期させる、というのも

- 状態をきちんと考える

12.4　信頼性のための技法

□ 出現順指定→冗長度が小さい→短く打てるが、誤りに弱い

- 並べ間違いに弱い→自然な順で打たせる、自己識別
- 項目数を 4 程度に (例外:　順序が問題でない場合)
- 部分並びの利用 (名前、電話番号、…)

13　識別情報

13.1　識別情報とは?

□ 「何のあたいはいくつ」の「何」

- 積極指定:　キーワードパラメタ (e.g.　NUM=100)
- 自己識別:　形を見ただけで分かるもの

13.2　自己識別情報

□ どこにあってもそれと分かる→順序が問題でない、同期点として利用

□ さまざまな自己識別情報

- 国名など (USA、JPN、…)
- 形で分かる (\$20.0、25%、3942-6869、128kg、MR. JONES

□ 分かる解釈から順にふるって、最後まで分からないなら手作業も可

13.3　日付の書式

□ すごくいっぱいある

- 1978.11.1、11/1/78、11.1.78、Nov 1 1978、12.IV.78、12IV78
- 月だけ:　6/1978、6/78
- 期間:　1977/1978、1977/78、21-30AUG
- 曜日:　TUE4/12

□ 2000 年問題

13.4　積極指定

□ 「NATION=USA」のように種別を明示する

- 間違いが起きにくい、設計と処理が楽、読みやすい、順序が自由
- 自己識別の方が打鍵数は少ないが、思わぬ問題に出会うことも

□ うまく組み合わせれば…

- 必要なところだけ積極指定する
- 指定しなかった場合は前回の値を引き継ぐ

14　繰り返し

14.1　繰り返しの意味

□ 冗長度をあげてエラーを検出する

- しかし「まったく同じものを 2 度打つ」では…
- 必要なところで適宜繰り返しを (部分的に) 使うとよい

```
234 234 UI2425
```

14.2　例外的繰り返し

□ 特に注意すべきことだけ繰り返す

- たとえば特に高価、特に大量など

□ 計算機の方から確認のため繰り返しを求める

- 例外的な場合、繰り返し入力での不一致

14.3　繰り返しの有効な利用

□ どこから繰り返しを用いるか?

- 例:　伝票の 2 箇所に同じ数字を書かせる→役に立つか?　何の?
- 冗長度はできるだけ情報源に近いところに

14.4　独立性

□ 反復によるエラー検出→独立性がなければ意味がない (本当は負の従属性)

- 2 人で数えるのなら 1 人目は結果を教えてはいけない
- 例:　ベリをさぼろうとした話

□ 一人でも独立性を

- 反復の間に別のことをさせる
- 逆打ち、変換 (漢数字とか、切上げ/切捨て、符号化)
- 暗黙の繰り返し (別の場所に冗長なものがある)

14.5　三重の繰り返し

□ まず間違わないし、多数決で正しい値が分かる→ホント?

- 人間が犯しがちな誤りのパターン。「6623546」
- その他、短縮、抜け、穴埋め、中央への偏り、前の入力への同化

□ 結局、三重でも安全とは言えない

14.6 圧縮された繰り返し

□ 検査桁→一種の繰り返しによる検査

□ 単純な繰り返しとの相違点

□ 圧縮されていると…

- 打鍵数が少ない。手間は大きい。機械が必要。検査が複雑。訂正は難しい。ほぼ数字むき。

□ 単純な繰り返しでは…

- 簡単に導入でき検査できる。訂正にも多様な方法が可能。数字に限らず何にでも可能。

□ 検査桁の計算方法で、検出能力や訂正能力が変化

- 弱点のある組み合わせを使わないようにすることも可能
- しかしそのことが忘れ去られると…(理解が困難な方式の運命)

□ 総計による検査も圧縮された繰り返し

- グループが大きくなるほど、誤りの可能性が増し、誤りがあったときそれを見つけるのが大変になる
- コストには誤りを訂正するためのコストも含めなければならない

15 確認語

15.1 確認語とは?

□ 外部の情報と比較することで検出/訂正を可能にする余分な情報

□ 記憶している情報を参照する方法…

- ポインタ→計算機内の位置。てっとり速いが融通が効かない
- キー(見出し)→索引による検索が可能。だが見出しさえあればというのは迷信。
- 名前(識別子)→情報項目を特定する「傾向のある」情報(第 1 近似)

15.2 確認語の定義

□ 「記録内容と比較することで、見つけた記録が確かに求めていた通りのものである確率を見積もるのに使われるデータ」

- 名前で探す→近似だから間違うこともある→確認語が有効。
- 見出しやポインタで済むというのは迷信。だから確認語。

□ 例:

```
241240 GILB   ←姓の最初の部分
1410KOLBOTN   ← ZIP コードに地名をつける
```

15.3 確認語の出所

□ 原文書の文面や対話から拾い出せる

- 例: 電話番号、品名、値段、
- 例: 伝票の走り書き、日付、…

□ 入力データから作り出すこともできる→顧客の名前や電話番号等

15.4 確認語と検査桁

□ 検査桁→記憶装置を利用*できない*場合の検査法→名前から*導き出せるものでなければならない*。

□ 確認語→記憶装置を利用*できる*ことを利用した検査法→名前から*導き出してはならない*。

□ 例: 「82-3942-6869/836」(先頭桁を取り出した確認語)→駄目な設計。検査桁より発見能力が劣る。

- 数字 10 桁のうち 7 桁は検査されない
- 3 個の数字も早い段階で入り込んだ間違いには無力
- 数字の読み間違いが検査できない

□ ではどうしたら? 「82-3942-6869/kuno」(その電話番号の人の苗字) →この方法でエラーが発見され損なう条件は

- 間違った番号もデータベース中にあり、かつ
- その人も kuno という苗字のときだけ。

□ 名前と電話番号は独立している(データベースを通じてのみつながっている)→独立性があってこそ、確認語が有効に働く

15.5 確認語のさまざま

□ 1 種類の方法でうまく行かない場合も→複数方式(例: 企業→性名がない)

□ 確認語は一意的に決まるとは限らない→変化を許容

□ 長さは 2〜4 文字くらいが普通だが上限はない

□ 置き場所は名前に隣接が多いが離れていてもよい (独立性が高まる)

15.6 実現

□ 確認語を選ぶ→それほど重大に考えなくてよい (検索ではなく確認のため)

□ 複数キーに同じ確認語がつかないようにする

□ 確認語の索引をメモリか DB に置いて検索/照合

15.7 誤り訂正への利用

□ 確認語は必要なら誤り訂正にも使える

□ 確認語で候補を絞ってその中から選択

- 計算機がやる場合→入力と 1 文字抜け、訂正などの比較照合、他の DB データに基づく照合 (例:本の 10 進分類など)
- 人間がやる場合→ずっと賢くできる

□ しかしどちらでも失敗することはある (ゼロにはできないと思え)

□ 逆に予期せぬ利益もある

- 例: 住所の確認語→引っ越したため不一致エラー→それは失敗ではなく成功のチャンス (住所未変更を発見できる)

15.8 不変確認語と可変確認語

□ 不変確認語→設計時に割り当て、名前の一部に組み込む「373-467-TBLE」

- 利点: 誤り訂正が確実。偶然検出能力を低下させることを防止できる。
- 欠点: 独立性が失われている。前段階でのエラーが検出されない。

□ 可変確認語→オペレータが規則に従って作る「2345 TG」「2345 GIL」

- 運用が用意。訓練が少ない。拡張による問題がおきにくい。独立性が増す (名前そのものが間違っている場合も検出可能)
- 実装は面倒になる。検出/訂正能力がやや低い (複数つけられれば OK)

15.9 有効性の評価

□ 数学モデルによる解析

□ 同じ確認語や類似した確認語が現われる等→有効性が低下

□ 設計の代替案を考える

□ よいと思えるものができたら、シミュレーションで検証

- オフラインのシミュレーション
- ライブの (一部オペレータによる実地の) シミュレーション

16 入力検査と適応制御

16.1 範囲検査

□ 入力数値の間違い→範囲検査

□ 範囲はいくつに? 品目ごとに決めるのは難しい

- 過去のデータに基づいて決める
- ということは、数値が変動しても適応して行ける

16.2 範囲の調節

□ 入力数値による選択肢: 警告なし/警告+範囲調節/確認を求める+範囲調節/確認を求める+範囲調節なし/確認+管理者に報告

□ 限界値が誤って調節されてしまう→毎月自動的に限界値を 10%ずつ減らすことで元に復帰

- ということは、限界値を超えない注文の情報も利用していることになる

□ 自動調節では、限界値の調節のようすを無作為抽出で検討する必要

- 不正等で「じわじわと」変更されているのを見つける
- その他興味深い状況を発見する

□ さまざまな調節方式: 平均 (移動平均)、標準偏差、正規分布 (低すぎる値も監視)、ポアソン…

16.3 パターン検査

□ めったに起きない注文→エラーの可能性

□ 固定パターン検査、ビットマップで学習、

16.4 初期設定

□ 最初に初期データを入れるのでは大変

- 自動的に立ち上がるように設計するとよい (適当な値から出発)

□ 例: 誕生日を使った身分確認

□ 例: 題名の入っていない図書館データベース

17 変動の許容

17.1 変動を許容することの重要性

□ かっては変動を許容しないことがコストを節約した。

□ 今では変動を許容しないコストは他のいかなるコストより高い。

□ 次のような手順で…

- 要求をはっきりさせる
- 現実の運用データを見る
- すべての変動を許容しなくてよい。どこまで許容するか判断
- 最初はありふれたものだけ許容すればよい。徐々に (データを取りつつ) 改良

17.2 ニセモノの変動許容

□ 例: OS/360 JCL

```
//名前 命令 オペランド 注釈
```

「ジョブ制御文の各フィールドは、名前フィールドが欄 3 から始まらなければならないことを除いては、自由形式で書いてよい。ここで自由形式とは、各フィールドが一定の欄からはじまる必要はない、ということである。各フィールドは空白で区切る。空白はフィールドとフィールドを区切る区切り記号として使われる。」

□ 実際には:

- フィールド間にはほとんど何個の空白でも入れてよい
- フィールドの中には空白を入れてはいけない
- 欄 73〜80 は使ってはならない
- 欄 72 はある特別の意味にしか使えない
- 空白をたくさん置きすぎてフィールドがあまり右の方に来てはならない

□ 何がいけないか?

- 利用者が欲しいと思う変動許容ではなく、たまたま簡単に取り入れられる変動許容だけが実現されている→こんなのは許容とは言えない

17.3 順序の変動

□ 用紙記入症はいけない

□ 自然な順序があれば、それに従うのがよい (無理に変動を許容しない)

□ 処理の都合があれば、計算機の方で並べかえればよい

□ メッセージの集団で処理できるとよい (あとで個別修正が簡単であること)

17.4 内容の変動

□ 標準的な形→時間、金額、数量、範囲など

□ 同義語のリストを持つ、適応可能なリストも (Zipf の法則)

□ スペルの変動→スペル検査/訂正

- 発音の聴き間違いを許容するスペル検査 (BPFVW、RL、MN、DT、CKQSZX、GJ をそれぞれグループにしてマッチさせる)

17.5 手順の変動

□ 便利なサブルーチンを用意する

□ 親切にする

- 謙虚な言葉づかい
- メッセージをテストする
- メッセージの別形式を用意する
- 手順を間違えてもシステムが壊れない
- ありふれたことはたやすくできるようにする
- 監査用の記録をとる

□ 人をロボット扱いすれば、ロボットのようにふるまう。

17.6 性能の見積り

□ とんでもなく歪曲された見積りに到達しないこと。歪曲の原因としては:

- 原文書が入力に便利にできていない
- 原文書がいろいろあって混ざっている
- 実は原文書を利用していない

- 実はしろうとが入力している
- 1 回かぎりの変換作業

□ コストの見積もり

- 何がコストになるか (大学は労働がタダ?)
- 経費の割り振りの問題

□ パイロットテストをする

□ 直接入力 (例:　テレフォン通販) →順序が自然でないと…

17.7　変動の積極利用

□ 変動を利用して人間を見分けることができる

□ 一定時間待って応答がなかったら先へすすむ

□ 打鍵タイミングを見張っていると打鍵間違いの発見が可能に

□ あまり誤りが多いようなら、その人に何かの問題が…

□ 変動の監視を抑圧に使わない (きっと報復を受ける…)