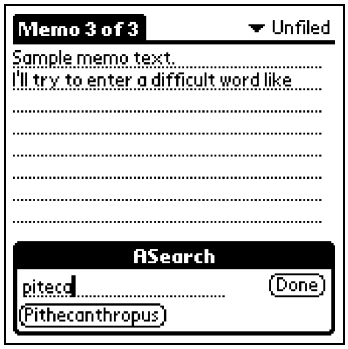
インターフェイスの街角 (3)
Pilot の動的曖昧検索 FEP

増井俊之

今回は、Pilot の Hack 機構[1]を用いて、前号で紹介した動的曖昧検索による英単語入力プログラム **ASearch** を作成する例を紹介します。

ASearch は Pilot のテキスト入力で使える FEP プログラムで、図 1 のようにメモ帳などのアプリケーションから呼び出して使います。この例では、〝piteca〟という誤った綴りを入力しています。しかし、動的曖昧検索により〝Pithecanthropus〟という正しい単語が候補として表示されるため、これを選択して正しい単語をメモ帳に入力できます。動的曖昧検索システムでは、検索条件を指定すると即座に検索が実行され、条件にもっとも近い（と思われる）結果が表示されます。

図 1　Pilot のメモ帳で ASearch を使用



動的曖昧検索システムの実装

曖昧度を自動的に変える動的曖昧検索システムは、おおよそ図 2 のようなプログラムとして実現できます。for ループを除去して曖昧度をつねに〝0〟とすると、普通の grep コマンドと同じになります。

一口に曖昧度といっても、さまざまな定義が考えられます。曖昧度 m を指定してパターン p とのマッチングをおこなうアルゴリズムは数多く提案されていますが、ここでは文献 [2] で用いられている比較的単純なものを使います[2]。この手法では、曖昧検索を実行する非決定性状態遷移機械の状態をビットパターンで表現し、入力文字に対応したシフト演算によってパターンマッチの成否を判断します。

図 2　動的曖昧検索プログラムの構造

```
int m; // 曖昧度
char *p = 検索パターン;
for(m=0;;m++){
    検索パターンp、曖昧度mでパターンマッチの準備;
    for(すべての検索対象){
        検索パターンp、曖昧度mでパターンマッチ実行;
    }
    if(マッチするエントリがある){
        結果を表示;
        break;
    }
}
検索結果を表示;
```

パターン〝ab∗ca〟を認識する状態遷移機械は図 3 のように表せますが、状態数を増やすことによって、ミスマッチ（誤字、脱字、誤挿入）を許す形式に拡張できます（図 4）。A0 はミスマッチを許さない受理状態で、A1 と A2 はそれぞれ 1 文字、2 文字の誤りを許す受理状態です。

たとえば図 4 の状態遷移機械に対して〝abracadabra〟と入力すると、状態は図 5 のように遷移します。〝ab〟まで読み込んだ状態で 2 文字の誤りを検出し、〝abra〟まで

1 Pilot のシステム・ライブラリのトラップテーブルを書き換えて実現するプログラムです。詳細は 1997 年 12 月号を参照してください。

2 前回紹介した agrep では、もっと複雑なアルゴリズムが使われています。

図 3　パターン `ab∗ca´を受理する状態遷移機械

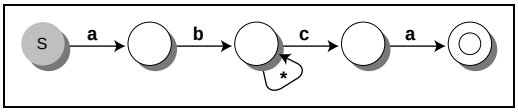


図 4　誤字/脱字を許容する状態遷移機械

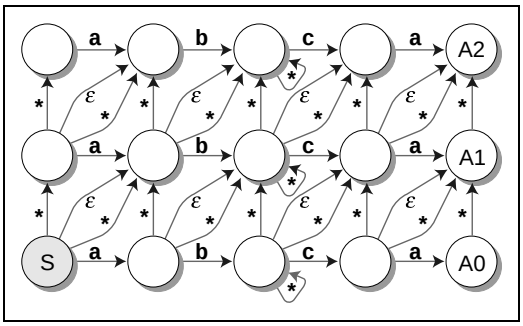
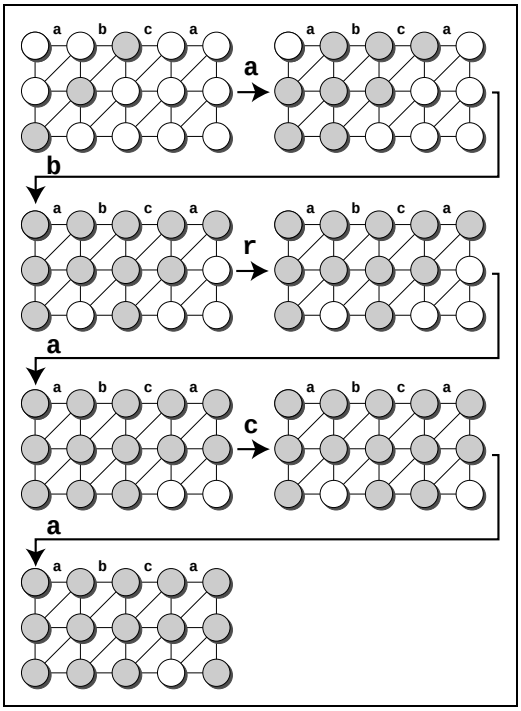


図 5　`abracadabra´に対する状態遷移



読み込んだ状態で 1 文字の誤りを検出していることが分かります。

曖昧なパターンマッチを許すこのような状態遷移機械は、図 4 の各行の状態を 2 進数で表現し、シフト演算によって遷移を計算することにより実現できます。各ノードのアクティブ状態を `1´と `0´で表現し、もっとも右のノードを LSB として各行を整数とみなすとしましょう。この場合、たとえば図 5 で最初の `a´を認識した状態は i2 = 01110_2、i1 = 11100_2、i0 = 11000_2という 3 個の変数で表現できます。リスト 1 のようなプログラムでこれらの値の変化を計算すれば、曖昧さを許容する状態遷移機械を表現できます。

まず MakePat() で、状態遷移の計算のための準備としてワイルドカードによる遷移のパターン `wildpat` および各文字に対応する遷移パターン `shiftpat` を計算します。図 3 では `wildpat` は00100_2となり、最初の状態と 4 番目の状態において入力文字 `a´によって状態遷移が起こるため、`shiftpat['a'] = 10010`$_2$とします。たとえば i0 の初期状態$10000_2$で `a´が入力されたときは、この値と `shiftpat['a']`（10010_2）の論理積をとり右に 1 つシフトした値（01000_2）が新しい i0 の値となります。

このように `wildpat` と `shiftpat` を用いて、リスト 1 の Match() 内のシフト演算を実行し、各入力文字に対する遷移状態を計算していくことができます。

この MakePat() と Match() を使って、動的曖昧検索をおこなうプログラムが作れます（リスト 2）。

Hack の作成例

ASearch は、リスト 1 に示した動的曖昧検索プログラム search.c を Pilot の FEP として動かすことで実現します。

ASearch について説明する前に、簡単な Hack プログラムの例として、Pilot の画面イメージをシリアルポートに出力するプログラム **ScreenDump** を作ってみましょう。アプリケーションの実行中に ScreenDump を呼び出すと、Pilot の画面イメージを PBM 形式で別の計算機に転送できます。

Hack プログラムを HackMaster（Hack プログラムを管理するためのアプリケーション・プログラム）に登録できるようにするには、HackMaster の API[3]に従ってプログラムやリソースを作成する必要があります。Pilot の各ライブラリ・ルーチンには `System/SysTraps.h` で定義されるトラップコードが対応しています。HackMaster

3 http://www.daggerware.com/hackapi.htm

リスト 1　曖昧検索プログラム

● メインルーチン search.c（UNIX/Pilot 共通）

```
#include "search.h"

void MakePat(Global *g, unsigned char *pat, int mismatch)
{
    int i;
    unsigned int mask = INITPAT;

    g->mismatch = mismatch;
    g->wildpat = 0;

    for(i=0;i<MAXCHAR;i++) g->shiftpat[i] = 0;
    for(;*pat;pat++){
        if(*pat == ' '){ // ワイルド カード 文字
            g->wildpat |= mask;
        }
        else {
            g->shiftpat[*pat] |= mask;
            mask >>= 1;
        }
    }
    g->acceptpat = mask;
}

int Match(Global *g, register unsigned char *text)
{
    register unsigned int i0 = INITPAT, i1=0, i2=0, i3=0;
    register unsigned int mask;
    register unsigned int e = g->wildpat;

    for(;*text;text++){ // シフトにより遷移計算
        mask = g->shiftpat[*text];
        i3 = (i3 & e) | ((i3 & mask) >> 1) | (i2 >> 1) | i2;
        i2 = (i2 & e) | ((i2 & mask) >> 1) | (i1 >> 1) | i1;
        i1 = (i1 & e) | ((i1 & mask) >> 1) | (i0 >> 1) | i0;
        i0 = (i0 & e) | ((i0 & mask) >> 1);
        i1 |= (i0 >> 1);
        i2 |= (i1 >> 1);
        i3 |= (i2 >> 1);
    }
    switch(g->mismatch){
        case 0: return (i0 & g->acceptpat);
        case 1: return (i1 & g->acceptpat);
        case 2: return (i2 & g->acceptpat);
        case 3: return (i3 & g->acceptpat);
        default: return 0;
    }
}
```

● ヘッダファイル search.h（UNIX 用）

```
#ifndef _SEARCH_H_
#define _SEARCH_H_

#define MAXCHAR 0x100
#define INITPAT 0x4000
#define MAXMISMATCH 3

typedef struct {
   int mismatch;
   unsigned int wildpat;
   unsigned int acceptpat;
   unsigned int shiftpat[MAXCHAR];
} Global;
#endif
```

の API では、まず置き換えたいライブラリ・ルーチンに 1000 番から順に ID を与えます。そして、この ID をもつリソースを使って置き換えたいライブラリ・ルーチンのトラップコードと置き換えるコード本体を指定します。したがって、すくなくとも下記の 2 つのリソースを定義する必要があります。

- リソースタイプが `code`、ID が 1000 のコード リソース（Hack プログラム本体）
- リソースタイプが `TRAP`、IDが 1000 のトラップリソース（2 バイトのトラップコードを指定）

このほかに、Hack プログラムの名前を示す文字列をリソースタイプが `tAIN` (Application Icon Name)、IDが

リスト 2　動的曖昧検索 grep

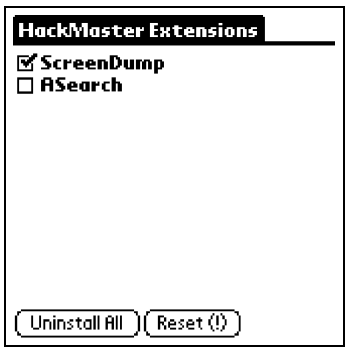
```
#include <stdio.h>
#include "search.h"

main(int argc, char **argv)
{
    Global g;
    char line[1000],pat[100];
    int m;
    FILE *f;

    if(argc < 3) exit(0);
    sprintf(pat," %s ",argv[1]);
           // 第1引数=パターン
    f = fopen(argv[2],"r");
          // 第2引数=ファイル
    if(f == NULL) exit(0);

    for(m=0;m<3;m++){       // 曖昧度を徐々に上げる
        int found = 0;
        MakePat(&g,pat,m); // 曖昧マッチの準備
        fseek(f,0,0);
        while(fgets(line,1000,f)){
            if(Match(&g,line)){ // 曖昧マッチ実行
                fputs(line,stdout);
                found = 1;
            }
        }
        if(found) break;
    }
}
```

図 6　HackMaster への登録



3000 のリソースとしておくと、HackMaster の表示画面内に Hack の名前が表示されるようになります（図 6）。

プログラム本体

リスト 3 がスクリーンダンプを出力するプログラムです。Pilot では画面イメージに直接アクセスできるので、シリアル通信ライブラリを用いて PBM 形式のスクリーン・イメージを簡単に送出できます。

ScreenDump() を実行すると、次のような PBM ファイルがシリアルポートに出力されます。

```
P1
160 160
01111111111111111111111111111111111111111
11111111111111111111111110000000000000000
00000000000000000000000000000000000000000
                ......
00000000000000000000000000000000000000000
```

エントリポイント

Hack プログラムは、コードリソースの先頭から実行されます。したがって、リスト 3 の ScreenDump() を含むファイルを GCC でコンパイルすると、オブジェクトの先頭に文字列定数などが含まれてしまうため Hack が正しく動きません。

これを避けるために、文字列定数などをまったく含まない関数のみからなる次のようなファイル (hackentry.c) を最初にリンクします。

```
void HackEntry()
{
    void ScreenDump();
    ScreenDump();
}
```

これらのプログラムをコンパイル、リンクしたあとのオブジェクト・ファイルを、**obj-res** を用いて Pilot のコードリソース形式に変更します。

Hack 用リソースの作成

HackMaster への登録時に必要な、リソースタイプが `TRAP` のリソースと Hack の名前を示す `tAIN` リソースは、それぞれ **hmtrap**、**hmname** というプログラムで作ることにします。これらのプログラムは私の Web ページ[4]で入手できますが、リソースファイルは次のように簡単なものなので、手作業で編集してもたいした手間はかかりません。

```
% hmname ABC
% od -xc tAIN0bb8.bin
0000000    4142    4300
           A   B   C  \0
0000004
% hmtrap sysTrapSysKeyboardDialog
% od -xc TRAP03e8.bin
```

4 http://www.csl.sony.co.jp/person/masui/UnixMagazine/

リスト 3　screendump.c

```
#include <System/SysAll.h>
#include <SerialMgr.h>

void ScreenDump()
{
    EventType event;
    int i,j;
    UInt serref;
    unsigned char *s = (unsigned char*) // 画面アドレスの取得
        (WinGetWindowPointer
             (WinGetDisplayWindow())->displayAddr);

    SysLibFind("Serial Library",&serref);
    SerOpen(serref,0,9600L);            // ポートと速度の設定
    SerSend(serref,"P1\n160 160\n",11); // PBMヘッダの送出

    for(i=0;i<160;i++){                 // 画面データの送出
        for(j=0;j<160;j++){
            int k = j % 8;
            SerSend(serref,*s & (0x80 >> k) ? "1":"0",1);
            if(k == 7) s++;
            if(j % 40 == 39) SerSend(serref,"\n",1);
        }
    }

    EvtGetEvent(&event, evtWaitForever);
    SerClose(serref);
}
```

```
0000000    a2e2
         242 342
0000002
%
```

Makefile

以上のようにして作ったリソースをまとめて、タイプが HACK の PRC ファイルを作成します。これを Pilot に転送すると、プログラムを HackMaster に登録できるようになります。

リスト 4 は、PRC ファイルを作るための Makefile です。UNIX 上の Pilot プログラム開発環境の 1 つである prc-tools[5]に付属する build-prc では、タイプが HACK の PRC ファイルは作れません。そこで、今回は Perl で書かれた **build-prc.perl**[6]を使います。prc-tools に付属の obj-res はリソース ID が 0001 のコードリソースを生成するので、HackMaster の API に適合させるために ID を 1000 (0x3e8) に変更します。

5 ftp://ryeham.ee.ryerson.ca/pub/PalmOS/prc-tools.0.5.0.tar.gz

6 電気通信大学の青柳龍也さんが作成された Perl スクリプトです。http://juju.cs.uec.ac.jp/pilotapp/syssnd/から入手できます。

これで作成した PRC ファイル scrdump.prc を pilot-xfer プログラムで Pilot に転送し、HackMaster に登録すればスクリーンダンプの機能が使えるようになります。この Makefile では SysKeyboardDialog() を ScreenDump() で置き換えるように指定しているため、キーボード呼出し操作をおこなうと、（ソフトキーボードが表示される代わりに）スクリーンダンプが Pilot のシリアルポートに出力されます。

ASearch の実装

以上に説明した Hack プログラムの作成手法とリスト 1 を使って ASearch を作ります。末尾にソースファイルを掲載しますが、注意が必要な点について順に説明していきます。

リソース定義

ASearch ではテキスト入力フィールドやボタンといったツールキット部品を使用しますが、これらの位置や大きさ、属性などはプログラム内で設定するのではなく、プログラムとは別にリソースとして定義します。たとえば、イ

リスト 4　ScreenDump の Makefile

```
CC=m68k-palmos-coff-gcc
CFLAGS = -O2 -g -Wall
OBJRES = m68k-palmos-coff-obj-res
BUILDPRC = build-prc.perl

# リソース名（任意）
NAME = ScreenDump
# タイプ（'HACK'でなければならない）
TYPE = HACK
# クリエータID（4文字のユニークな識別名）
ID = SDMP
# PalmOSのバージョン（旧型Pilotの場合は1にする）
OSVERSION = 2

scrdump.prc: screendump
        -/bin/rm -f *.bin
        $(OBJRES) screendump
        hmtrap -$(OSVERSION) sysTrapSysKeyboardDialog
        hmname $(NAME)
        /bin/mv code0001.screendump.grc \
                code03e8.screendump.grc
        $(BUILDPRC) scrdump.prc $(NAME) $(TYPE) $(ID) *.bin code03e8*.grc
screendump: screendump.o hackentry.o
        $(CC) -o screendump -nostartfiles hackentry.o screendump.o
```

リスト 5　asearch.rcp（ASearch のリソース定義）

```
#include "id.h"

// アプリケーション名定義（HackMasterで参照される）
APPLICATIONICONNAME ID 3000 "ASearch"

// ASearchウィンドウ定義
FORM ID ID_MAINFORM AT (2 115 156 43)
MODAL
USABLE
BEGIN
    TITLE   "ASearch"
    FIELD ID_PATFIELD 3 15 100 12 UNDERLINED EDITABLE
            MAXCHARS MAXFIELDSIZE
    BUTTON "Done" ID_BUTTON_DONE 126 15 26 10
    BUTTON " "    ID_BUTTON1     0 30 0 10
    BUTTON " "    ID_BUTTON2     0 30 0 10
    BUTTON " "    ID_BUTTON3     0 30 0 10
    BUTTON " "    ID_BUTTON4     0 30 0 10
    BUTTON " "    ID_BUTTON5     0 30 0 10
    BUTTON " "    ID_BUTTON6     0 30 0 10
    BUTTON " "    ID_BUTTON7     0 30 0 10
    BUTTON " "    ID_BUTTON8     0 30 0 10
    BUTTON " "    ID_BUTTON9     0 30 0 10
    BUTTON " "    ID_BUTTON10    0 30 0 10
END
```

リスト 6　id.h

```
#define ID_MAINFORM     1000
#define ID_PATFIELD     1100
#define ID_BUTTON_DONE  1200
#define ID_BUTTON       1300
#define ID_BUTTON1      1300
#define ID_BUTTON2      1301
#define ID_BUTTON3      1302
#define ID_BUTTON4      1303
#define ID_BUTTON5      1304
#define ID_BUTTON6      1305
#define ID_BUTTON7      1306
#define ID_BUTTON8      1307
#define ID_BUTTON9      1308
#define ID_BUTTON10     1309
#define MAXFIELDSIZE    30
```

ンターフェイスの基本となるウィンドウ（フォーム）はリソースタイプが tFRM のリソースとして定義します。

prc-tools に付属のリソース・コンパイラ pilrc (Pilot Resource Compiler) を利用すれば、テキストによるリソース記述ファイルから、各インターフェイス・ツールに対応するリソースファイルが作れます。

図 7　asearch.c の構造

```
void ASearch() // Hackの先頭
{
    初期化処理;
    do {
        入力イベント取得;
        AppHandleEvent();//ASearch用イベント処理
        システム用イベント処理;
    }while(! 終了イベント)
    終了処理;
}

AppHandleEvent()
{
    switch(イベント種別){
    case 文字入力:
          入力パターンにもとづき動的曖昧検索を実行し
            候補をボタンとして表示;
    case ボタン選択:
          候補単語をテキストに入力;
          パターンをクリア;
    }
}
```

pilrc による ASearch のリソース定義をリスト 5 に示します。ここでは、プログラム名を示す `tAIN` リソースと ASearch ウィンドウ枠を示す `tFRM` リソースを定義しています。`ID_`で始まるものについては `id.h` 内で数字を定義しています（リスト 6）。この値は、プログラム内からボタンなどを参照するときにも使用します。

メインプログラムの構造

ASearch のメインプログラムは、図 7 のような構造になります。

グローバル変数

Hack では、グローバル変数は使えません。そこで、関数間で共用される変数は ``Global''構造体のメンバーとして実現しています。このため、各関数では Global へのポインタを持ち回っています。Global 構造体の領域は MemPtrNew() で確保します。

メインルーチンの呼出し

メイン関数 ASearch() は、ScreenDump と同様の方法で呼び出します。ここでは、二重呼出しを避けるために Pilot の Feature 機能を使っています。

Feature とは、アプリケーションごとにデータを一時的に保存しておける機能です。ASearch の実行中は `FEATURE_LOCK` で示される Feature の値を ``1''としておくことにより、ASearch が実行されていないときにかぎり、Lock() が true を返して処理が続行されるようになっています。

リスト 7　edic2bin プログラム

```
#!/usr/local/bin/perl
while(<>){
    chop;
    next if /^#/ || /^\s*$/;
    ($word) = split;
    $_ = $word;
    s/^[^a-zA-Z]+//;
    /^./;
    $c = "\L$&";
    if($prevc ne $c){
        close(bin);
        $filename = sprintf("EDIC%04x.bin",
            $n = ord($c) - ord("a") + 1);
        open(bin,"> $filename");
    }
    $prevc = $c;
    print bin "$word\0";
}
```

辞書データベース

Pilot にはファイルシステムがありません。したがって、辞書データはメモリ上のデータ・データベースの形式にする必要があります。

Pilot では 64KB 以上の連続したメモリ領域は使えないので、辞書を分割して単語の先頭文字ごとに異なるリソースとして作成しておきます。辞書データベースのタイプは `DICt`、クリエータ ID は ASearch プログラム本体と同じ `ASRC` としておきます。Perl スクリプト **edic2bin**（リスト 7）を用いて、ソートされた英単語のテキストファイルから各先頭文字に対応したリソースファイルを生成し、build-prc を使って 1 つの PRC ファイルにまとめて転送します。

イベントループ

ASearch ではテキスト入力枠やボタンなどのインターフェイス・ライブラリを利用するため、Pilot の普通のアプリケーションを作成する際に使われる PilotMain() と同様のイベントループをもつ必要があります。イベントループでは、ASearch 関連のイベント処理をおこなうととも

リスト 8　ASearch の Makefile

```
CC=m68k-palmos-coff-gcc
CFLAGS = -O2 -g -Wall
PILRC = pilrc
OBJRES = m68k-palmos-coff-obj-res
BUILDPRC = build-prc.perl

NAME = ASearch
TYPE = HACK
ID = ASRC

OSVERSION = 2

all: asearch.prc edic.prc

asearch.prc: asearch asearch.rcp
        -/bin/rm -f *.bin
        $(OBJRES) asearch
        hmtrap -$(OSVERSION) sysTrapSysKeyboardDialog
        /bin/mv code0001.asearch.grc code03e8.asearch.grc
        $(PILRC) asearch.rcp .
        $(BUILDPRC) asearch.prc $(NAME) $(TYPE) $(ID) *.bin code03e8*.grc
asearch: asearch.o hackentry.o search.o
        $(CC) -o asearch -nostartfiles hackentry.o asearch.o search.o
edic.prc:
        edic2bin < /usr/dict/words
        $(BUILDPRC) edic.prc edic DICt $(ID) EDIC*.bin
```

に、文字認識や電源ボタンなどのシステムイベントを認識するために SysHandleEvent() を呼び出します。

ASearch ウィンドウのイベント処理

ASearch ウィンドウへの入力は AppHandleEvent() で処理します。候補単語ボタンが押された場合は、InsertToMainField() で ASearch を呼び出したアプリケーションに候補単語をペーストします。パターン文字が入力されたときは、SearchCand() で動的曖昧検索を実行し、検索結果を候補単語ボタンのリストとして表示します。

次候補/前候補ボタン

ASearch の候補表示領域は小さいので、Pilot のスクロールボタン（本体下部中央の 2 つのボタン）を使って次候補、前候補を出せるようにしてあります。

Makefile

ASearch の Makefile はリスト 8 のようになります。

おわりに

UNIX 上の開発環境を使うことにより、Pilot 上で動的曖昧検索をおこなう FEP をたかだか 500 行程度で作成することができました。こういったことが手軽にできる Pilot と UNIX の組合せは、携帯端末やペン・インターフェイスの実験にうってつけといえるでしょう。

なお、ScreenDump と ASearch のソースファイルは、前述の私の Web ページで公開しています。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] 増井俊之、水口充、George Borden、柏木宏一「なめらかなユーザインタフェース」、第 37 回冬のプログラミングシンポジウム予稿集、pp.13–23、情報処理学会、1996 年 1 月

[2] 山田八郎、高橋恒介、平田雅規、永井 肇「あいまい検索が可能な文字列検索 LSI」、日経エレクトロニクス No.422、pp.165–181、1987 年 6 月

ASearch のソースファイル

● asearch.h

```
#ifndef _ASEARCH_H_
#define _ASEARCH_H_

#include <System/SysAll.h>
#include <UI/UIAll.h>

#define NEWCAND 1
#define NEXTCAND 2
#define PREVCAND 3

#define MAXCANDS 10
#define MAXCANDPAGES 20
#define MAXCHAR 0x100

typedef struct {
    DmOpenRef edic;        // 英単語DB
    DmOpenRef db;          // ASearch DB
    FormPtr asfrm;         // ASearchフォーム
    FieldPtr patfield;     // パターンフィールド

    int ncands;                 // 表示候補数
    char candstr[MAXCANDS][20];// 候補単語
    int candpos;                // 候補表示の右端
    int candindex;              // 候補画面番号
    int candoffset[MAXCANDPAGES];

    unsigned long wildpat;
    unsigned long acceptpat;
    unsigned long shiftpat[MAXCHAR];
    int mismatch;

    Boolean finish;
} Global;

#define FEATURE_LOCK 10
#define APPTYPE 'ASRC'
#endif
```

● asearch.c

```
#pragma pack(2)

#include <stdio.h>
#include <System/SysAll.h>
#include <UI/UIAll.h>
#include <System/FeatureMgr.h>

#include "asearch.h"
#include "search.h"
#include "id.h"

static Boolean Lock(void);
static void Unlock(void);
static void OpenAsearch(Global *g);
static void CloseAsearch(Global *g);
static Boolean AppHandleEvent(Global *g,
                              EventPtr e);
static void InsertToMainField(Global *g,
                              char *s);
static void EraseCand(Global *g);
static void SearchCand(Global *g,
                       int kind);

////////////////////////////////////////
//          メインルーチン
////////////////////////////////////////

void ASearch()
{
    FormPtr frm;
    void (*oldtrap)(int);
    EventType event;
    Global *g;

    // Hackを呼び出すアプリケーションが
    // 文字入力状態かどうかをチェック
    if((frm = FrmGetActiveForm()) == NULL)
        return;
    if(FrmGetFocus(frm) == -1) return;

    // 重ねて呼び出されたときは元のライブラリを呼び出す
    if(! Lock()){
        DWord ftrvalue;
        FtrGet(APPTYPE,1000,&ftrvalue);
        oldtrap = (void (*)(int))ftrvalue;
        (*oldtrap)(0);
        return;
    }

    // グローバルデータ領域確保
    g = (Global*)MemPtrNew(sizeof(Global));
    ErrFatalDisplayIf(g==NULL,
        "Can't alloc global data");

    // 辞書データベースをオープン
    g->edic = DmOpenDatabaseByTypeCreator
        ('DICt', APPTYPE, dmModeReadOnly);
    ErrFatalDisplayIf(g->edic==0,
        "Can't open edic");

    OpenAsearch(g); // ASearch画面初期化

    // ASearchのイベント処理ループ
    g->finish = false;
    do {
        Word error;
        EvtGetEvent(&event, evtWaitForever);
        if(AppHandleEvent(g,&event)) continue;
        if(SysHandleEvent(&event)) continue;
        if(MenuHandleEvent(NULL, &event,
```

```
                            &error)) continue;
        FrmDispatchEvent(&event);
    }
    while (! g->finish &&
           event.eType != appStopEvent);
    if(event.eType == appStopEvent){
        EvtAddEventToQueue(&event);
    }

    CloseAsearch(g); // ASearch画面終了

    DmCloseDatabase(g->edic);
    MemPtrFree(g);
    Unlock();
}

////////////////////////////////////////////
//        ASearch画面開始/終了
////////////////////////////////////////////

static Boolean MainFormEvent(EventPtr e)
{
    return false;
}

static void OpenAsearch(Global *g)
{
    Word idx;

    // Formを呼び出せるようにするために
    // 共通データベースをオープン
    g->db = DmOpenDatabaseByTypeCreator('HACK',
            APPTYPE, dmModeReadWrite);
    ErrFatalDisplayIf(g->db==NULL,
                      "Can't open ASearch DB");

    g->asfrm = FrmInitForm(ID_MAINFORM);
    ErrFatalDisplayIf(g->asfrm==NULL,
                      "Can't init Form");

    FrmSetEventHandler(g->asfrm, MainFormEvent);
    FrmSetActiveForm(g->asfrm);

    EraseCand(g);

    FrmDrawForm(g->asfrm);

    idx = FrmGetObjectIndex(g->asfrm,
                            ID_PATFIELD);
    FrmSetFocus(g->asfrm,idx);
    g->patfield = FrmGetObjectPtr(g->asfrm,idx);
}

static void CloseAsearch(Global *g)
{
    //アプリケーション画面に復帰
    FrmReturnToForm(NULL);
    DmCloseDatabase(g->db);
}

////////////////////////////////////////////
//        ASearchのイベント処理
////////////////////////////////////////////

static Boolean AppHandleEvent(Global *g,
                              EventPtr e)
{
    Boolean handled = true;
    Word c;
    int button;

    switch(e->eType){
    case ctlSelectEvent:
        c = e->data.ctlSelect.controlID;
        switch(c){
        case ID_BUTTON_DONE:
            g->finish = true;
            break;
        case ID_BUTTON1:    case ID_BUTTON2:
        case ID_BUTTON3:    case ID_BUTTON4:
        case ID_BUTTON5:    case ID_BUTTON6:
        case ID_BUTTON7:    case ID_BUTTON8:
        case ID_BUTTON9:    case ID_BUTTON10:
            button = c - ID_BUTTON;
            InsertToMainField(g,
                g->candstr[button]);
            break;
        default:
            handled = false;
            break;
        }
    case keyDownEvent:
        c = e->data.keyDown.chr;
        // 上スクロールボタン
        if(c == pageUpChr){
            SearchCand(g,PREVCAND);//前候補表示
        }
        // 下スクロールボタン
        else if(c == pageDownChr){
            SearchCand(g,NEXTCAND);//次候補表示
        }
        else if(c == '\n'){
            char *s = FldGetTextPtr
                      (g->patfield);
            if(s){
                char buf[100];
                StrCopy(buf,s);
                InsertToMainField(g,buf);
            }
            else
                InsertToMainField(g,"\n");
        }
        else if(c >= 0x20 && c < 0x80){
            char cc = (char)c;
```

```
            FldInsert(g->patfield,&cc,1);
            SearchCand(g,NEWCAND);
        }
        else
            handled = false;
        break;
    default:
        handled = false;
        break;
    }
    return handled;
}

////////////////////////////////////////
//        Lock/Unlock
////////////////////////////////////////

// 二重呼出しを避けるロック
static Boolean Lock(void)
{
    DWord locked;
    Err err = FtrGet(APPTYPE,FEATURE_LOCK,
                     &locked);
    if(err // ASearchが1度も実行されていない
      || locked == 0 // ASearchが実行中でない
       ){
        FtrSet(APPTYPE,FEATURE_LOCK,1);//ロック
        return true;
    }
    return false;
}

static void Unlock(void)
{
    FtrSet(APPTYPE,FEATURE_LOCK,0); // UnLock
}

////////////////////////////////////////
//        候補単語表示
////////////////////////////////////////

static void EraseCand(Global *g) // 候補クリア
{
    int i;
    g->ncands = 0;
    g->candpos = 0;
    for(i=0;i<MAXCANDS;i++){
        Word idx = FrmGetObjectIndex
                   (g->asfrm, ID_BUTTON+i);
        CtlHideControl(FrmGetObjectPtr
                       (g->asfrm,idx));
        g->candstr[i][0] = '\0';
    }
}

static Boolean AddCand(Global *g,
                       unsigned char *cstr)
//
// 候補をリストに加えて表示。余裕がなければfalseを返す
//
{
    int i, width;
    ControlPtr cp;
    Word idx;

    if(g->ncands >= MAXCANDS) return false;

    // すでにその候補が候補リストにあれば何もしない
    for(i=0;i<g->ncands;i++)
        if(StrCompare(cstr,
           g->candstr[i]) == 0) return true;

    // 新しい候補文字列をボタンに登録/表示
    width = FntCharsWidth(cstr,
                          StrLen(cstr))+4;
    if(g->candpos + 2 + width >= 156)
       return false;
    StrCopy(g->candstr[g->ncands],cstr);
    idx = FrmGetObjectIndex(g->asfrm,
                      ID_BUTTON+g->ncands);
    cp = FrmGetObjectPtr(g->asfrm,idx);
    cp->bounds.topLeft.x = g->candpos + 2;
    cp->bounds.extent.x = width;
    cp->text = g->candstr[g->ncands];
    CtlShowControl(cp);

    g->candpos += width+3;
    g->ncands++;
    return true;
}

static void SearchCand(Global *g, int kind)
//
//    動的曖昧検索を実行して候補単語リストを表示
//
{
    Char pat[100],*s,*p,*s0;
    int resid;
    VoidHand h;
    Boolean found;
    ULong size;
    int mismatch; // 曖昧度

    switch(kind){
    case NEWCAND:
        g->candindex = 0;
        g->candoffset[0] = 0;
        break;
    case NEXTCAND:
        if(g->candindex >= MAXCANDPAGES-1)
           return;
        g->candindex++;
        break;
    case PREVCAND:
```

```
        if(g->candindex <= 0) return;
        g->candindex--;
        break;
    }

    if((s = FldGetTextPtr(g->patfield))==NULL)
      return;

    for(p=pat;*s;s++){
        if((*s >= 'a' && *s <= 'z') ||
             *s == ' ') *p++ = *s;
        if(*s >= 'A' && *s <= 'Z')
            *p++ = (*s + 'a' - 'A');
    }
    StrCopy(p," ");

    resid = (int)(pat[0] - 'a' + 1);
    if(resid < 1 || resid > 'z'-'a'+1) return;

    h = DmGetResource ('EDIC',resid);
    ErrFatalDisplayIf(h==0,
        "Can't get EDIC resource");
    s0 = MemHandleLock(h);
    ErrFatalDisplayIf(s==0,
        "Can't lock EDIC resource");
    size = MemPtrSize(s0);

    // 動的曖昧検索実行
    for(mismatch=0;mismatch<3;mismatch++){
        EraseCand(g);
        MakePat(g,pat,mismatch);
        found = false;
        s = s0 + g->candoffset[g->candindex];
        while(s-s0 < size){
            if(Match(g,s)){
                found = true;
                if(! AddCand(g,s)) break;
            }
            for(;*s;s++);
            s++;
        }
        if(found) break;
    }

    if(g->candindex+1 < MAXCANDPAGES)
        g->candoffset[g->candindex+1] = s - s0;

    MemHandleUnlock(h);
    DmReleaseResource(h);
}

//////////////////////////////////////////
//        選択単語の貼付け
//////////////////////////////////////////
static void InsertToMainField(Global *g,
                              char *s)
//
//  ASearchのフォームを完全に閉じて
//  元のフィールドに単語挿入
//
{
    LocalID id;
    VoidPtr v;
    FormPtr frm;
    FieldPtr field;
    FormObjectKind kind;

    CloseAsearch(g);

    frm = FrmGetActiveForm();
    id = FrmGetFocus(frm);
    v = FrmGetObjectPtr(frm,id);
    kind = FrmGetObjectType(frm,id);
    field = (kind == frmFieldObj ? v :
      kind == frmTableObj ?
              TblGetCurrentField(v) :
             NULL);
    if(field) FldInsert(field,s,StrLen(s));

    OpenAsearch(g);
}
```

- **hackentry.c（ASearch用）**

```
void HackEntry()
{
    void ASearch();
    ASearch();
}
```

- **search.h**

```
// 曖昧検索ヘッダ (Pilot用)
#ifndef _SEARCH_H_
#define _SEARCH_H_

#include "asearch.h"

#define INITPAT 0x4000

void MakePat(Global *g,
             unsigned char *pat, int m);
int Match(Global *g,
             unsigned char *text);

#endif
```