# 情報工学ゼミナール

*Information Engineering Seminar*

－　２００１年度版　－

福岡工業大学
情報工学部 情報工学科
柴田望洋

*BohYoh Shibata*
*Fukuoka Institute of Technology*

# 本資料について

◆　本資料は、2001年度・福岡工業大学 情報工学部 情報工学科３年生の講義

**『情報工学ゼミナール』**

の補助テキストとして、福岡工業大学 情報工学部 情報工学科 柴田望洋が編んだものである。

◆　参考文献・引用文献等は、資料の最後にまとめて示す。

◆　諸君が本資料をファイルに綴じやすいように、研究室の学生達（卒研生と大学院生）が時間を割いて、わざわざ穴を開けるという作業を行っている（一度のパンチで開けることのできる枚数は限られており、**気の遠くなるような**時間がかかっている）。
**必ずＢ５のバインダーを用意して、きちんと綴じていただきたい。**



本資料は、Microsoft 社のワープロソフトウェアである Microsoft Word 2000 を用いて作成した。

# 数当てゲームを作ろう！

C言語を使って、数当てゲームを作りましょう。

## ■ 第１版　変数に格納された値を当てさせる

```
/*
    数当てゲーム（その１）
*/

#include  <stdio.h>

int main(void)
{
    int  x;                /* 読み込んだ値 */
    int  no = 7;           /* この数を当てさせる  */

    printf("整数を入力せよ：");
    scanf("%d", &x);

    if (x > no)
        printf("¥a大きいです。¥n");
    else if (x < no)
        printf("¥a小さいです。¥n");
    else
        printf("正解です。¥n");

    return (0);
}
```

if 文を用いて判断を行います。読み込んだ値 x が、当てるべき値 no より大きいか、小さいか、等しいかを判断します。

この後でも説明しますが、¥a は警報を意味する拡張表記でしたね。通常は、ビープ音がなります。

※※※　このプログラムは、数を入力して判断を行うのが１回限りです。当たるまで繰り返そうとする場合は、毎回プログラムを起動しなければなりません。

## ■ 第２版　変数に格納された値を当てさせる（当たるまで繰り返す）

```
/*
    数当てゲーム（その２）
*/

#include  <stdio.h>

int main(void)
{
    int  x;                /* 読み込んだ値 */
    int  no = 7;           /* この数を当てさせる  */

    do {
        printf("整数を入力せよ：");
        scanf("%d", &x);

        if (x > no)
            printf("¥a大きいです。¥n");
        else if (x < no)
            printf("¥a小さいです。¥n");
    } while (x != no);     /* 当たっていない間繰り返す  */

    printf("正解です。¥n");

    return (0);
}
```

**do** 文を利用して、繰返しを行います。繰返しを続けるかどうかの判定は(x != no)です。すなわち、読み込んだ値x が、当てるべき値no と等しくない間、繰り返します。

**do** 文による繰返しが終了したときは、x と no は等しくなっていますので、『正解です。』と表示します。

※※※　ここまでのプログラムは、当てさせるべき数である 7 がプログラム中に埋め込まれています。もし、7 以外の数を当てさせようと思ったら、プログラムを書きかえて、コンパイル・実行し直さなければなりません。

これでは、ゲームとしては不完全です。そこで、当てさせるべき数を、ランダムなものとしましょう。

## ■ 第３版　当てさせる数を乱数にする

```
/*
    数当てゲーム（その３）
*/

#include  <time.h>
#include  <stdlib.h>
#include  <stdio.h>

int main(void)
{
    int  x;                    /* 読み込んだ値 */
    int  no;                   /* この数を当てさせる  */
    time_t   t;                /* 時刻 */

    srand(time(&t) % RAND_MAX);     /* 乱数の種を初期化 */
    no = rand() % 1000;             /* 0〜999の乱数を発生 */

    do {
        printf("整数を入力せよ：");
        scanf("%d", &x);

        if (x > no)
            printf("¥a大きいです。¥n");
        else if (x < no)
            printf("¥a小さいです。¥n");
    } while (x != no);     /* 当たっていない間繰り返す  */

    printf("正解です。¥n");

    return (0);
}
```

srand 関数および rand 関数は、乱数発生のための関数です。

■ srand 関数の呼出しを省略すると、プログラムを起動するたびに、毎回同じ乱数が発生されてしまいます。実際に確認してみましょう。

■ 発生した乱数を 1000 で割ったあまりが no に代入されます。したがって、0〜999 を当てさせることになります。この値を変更してみましょう。

★★★　関数の仕様や乱数発生のテクニックについては、
　柴田望洋後援会オフィシャルホームページ　http://www.BohYoh.com/
で学習できます。

※※※　何度も入力していれば、当たるのは当然です。入力できる回数に制限を付けると面白くなります。

## ■ 第４版　入力回数に制限を付ける

```
/*
    数当てゲーム（その４）
*/

#include  <time.h>
#include  <stdlib.h>
#include  <stdio.h>

int main(void)
{
    const int max = 10;         /* 入力制限回数 */
    int  x;                     /* 読み込んだ値 */
    int  no;                    /* この数を当てさせる  */
    int  cnt = max;             /* 残り何回入力できるか？ */
    time_t   t;                 /* 時刻 */

    srand(time(&t) % RAND_MAX);     /* 乱数の種を初期化 */
    no = rand() % 1000;             /* 0〜999の乱数を発生 */

    do {
        printf("残り%d回です。¥n", cnt);
        printf("整数を入力せよ：");
        scanf("%d", &x);
        cnt--;

        if (x > no)
            printf("¥a大きいです。¥n");
        else if (x < no)
            printf("¥a小さいです。¥n");
    } while (x != no && cnt > 0);

    if (x == no) {
        printf("正解です。¥n");
        printf("%d回で当たりましたね。¥n", max - cnt);
    }
    return (0);
}
```

max は最大で何回まで挑戦できるかを表す定数です。cnt は、あと何回挑戦できるかを表す変数です。最初は、cnt の値は max と同じ値です（このプログラムでは 10 回)。入力するたびに、cnt の値をデクリメントしていきます。

do 文の判定に、cnt > 0 が追加されていることに注意しましょう。残り挑戦回数を使い果たすと、do 文は終了です。

# 拡張表記を効果的に使おう！

拡張表記（*escape sequence*）を効果的に使った、画面表示のテクニックを覚えましょう。

## ■ 警報と改行

```
/*
    拡張表記（エスケープシーケンス）の利用例１
        ¥a  警報
        ¥n  改行
*/

#include  <stdio.h>

int main(void)
{
    printf("¥a警告します。¥n");

    return (0);
}
```

## 拡張表記（escape sequence）

| | | |
|---|---|---|
| ¥a | 警報（*alert*） | 聴覚的または視覚的な警報を発する。 |
| ¥b | 後退（*alert*） | 表示位置を直前の位置へ移動する。 |
| ¥t | 水平タブ（*horizontal tab*） | 次の水平タブ位置へ移動する。 |
| ¥n | 改行（*new line*） | 改行して、次の行の先頭へ移動する。 |
| ¥v | 垂直タブ（*vertical tab*） | 次の垂直タブ位置へ移動する。 |
| ¥f | 書式送り（*form feed*） | 改ページして、次のページの先頭へ移動する。 |
| ¥r | 復帰（*carriage return*） | 現在の行の先頭位置へ移動する。 |

## ■ 後退

```
/*
     拡張表記（エスケープシーケンス）の利用例２
          ¥b  後退（カーソルを一つ戻す）
*/

#include  <time.h>
#include  <stdio.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
     clock_t   s = clock();
     clock_t   c;
     do {
          if ((c = clock()) == (clock_t)-1)        /* エラー */
               return (0);
     } while (1000UL * (c - s) / CLOCKS_PER_SEC <= x);
     return (1);
}

int main(void)
{
     int   i;

     printf("ABCDEFG");

     for (i = 0; i < 7; i++) {       /* 後ろから１文字ずつ */
          sleep(1000);               /* １秒ごとに消す     */
          printf("¥b ¥b");
     }

     return (0);
}
```

このプログラムは、ABCDEFG と表示した後に、１秒ずつ後ろから１文字ずつ消していきます。

なお、関数 sleep は、x ミリ秒だけ待つための関数です。分からない人は、そのまま使いましょう。

## ■ 復帰

```
/*
     拡張表記（エスケープシーケンス）の利用例３
          ¥r  復帰（カーソルを行の先頭へ戻す）
*/

#include  <time.h>
#include  <stdio.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
     clock_t   s = clock();
     clock_t   c;
     do {
          if ((c = clock()) == (clock_t)-1)        /* エラー */
               return (0);
     } while (1000UL * (c - s) / CLOCKS_PER_SEC <= x);
     return (1);
}

int main(void)
{
     int   i;

     printf("私の名前は柴田望洋。");

     sleep(2000);
     printf("¥r福岡工業大学の助教授です。");

     sleep(2000);
     printf("¥r講義を楽しくやりましょう。");

     return (0);
}
```

このプログラムは、三つのメッセージを２秒ごとに書きかえます。

文字列を表示した後に、カーソルを行の先頭に戻して別の文字列を表示しますので、書きかえるように見えるのです。

## ■ 文字列への応用（その１）

```
/*
     自分の名前を１文字ずつ表示し後ろから１文字ずつ消去するのを繰り返す
*/

#include  <time.h>
#include  <stdio.h>
#include  <string.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    /*--- 省略 ---*/
}

int main(void)
{
     int    i, j;
     char  name[] = "BohYoh Shibata";          /* 名前（全角文字は不可） */
     int    name_len = strlen(name);           /* 文字列nameの文字数 */

     while (1) {                                    /* 無限に繰り返す */
          for (i = 0; i < name_len; i++) {
               putchar(name[i]);
               sleep(100);
          }
          for (i = 0; i < name_len; i++) {
               printf("¥b ¥b");
               sleep(100);
          }
     }
     return (0);
}
```

C言語では、文字列は char 型の配列として表わすことができます。このプログラムの場合、配列name の要素数は15 となります。

| B | o | h | Y | o | h |  | S | h | i | b | a | t | a | ¥0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

最後の¥0 は、文字列の終端を示す**ナル文字**（*null character*）です。

したがって、name[0], name[1], name[2], … が、'B', 'o', 'h', …となります。

なお、strlen 関数は、文字列の長さ（ナル文字は含みません）を求める関数です。

## ■ 文字列への応用（その２）

```
/*
    名前をテロップ表示
*/

#include  <time.h>
#include  <stdio.h>
#include  <string.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    /*--- 省略 ---*/
}

int main(void)
{
    int   i;
    int    cnt = 0;                          /* 何文字目が先頭か */
    char  name[] = "BohYoh Shibata ";        /* 名前（全角文字は不可） */
    int    name_len = strlen(name);          /* 文字列nameの文字数 */

    while (1) {
        putchar('¥r');          /*  カーソルを行の先頭へ */

        for (i = 0; i < name_len; i++) {
            if (cnt + i < name_len)
                putchar(name[cnt + i]);
            else
                putchar(name[cnt + i - name_len]);
        }

        sleep(300);

        if (cnt < name_len - 1)
            cnt++;
        else
            cnt = 0;
    }
    return (0);
}
```

空白文字を忘れないように

このプログラムの場合、配列 name の要素数は 16 となります（苗字の後ろに空白文字があることに注意しましょう)。

| B | o | h | Y | o | h |  | S | h | i | b | a | t | a |  | ¥0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

変数 cnt は、どの文字から書き始めるのかを表わす変数です。最初は 0 ですから、右に示す for 文は、

```
for (i = 0; i < 15; i++) {
    if (0 + i < 15)
        putchar(name[i]);
    else
        putchar(name[i - name_len]);
}
```

です。このとき if 文は必ず成立しますので、実質的には

```
for (i = 0; i < 15; i++) {
    putchar(name[cnt]);
}
```

であり、name[0], name[1], name[2], … 、すなわち、'B', 'o', 'h', … を順に表示します。つまり、"BohYoh Shibata "と表示します。

```
for (i = 0; i < name_len; i++) {
    if (cnt + i < name_len)
        putchar(name[cnt + i]);
    else
        putchar(name[cnt + i - name_len]);
}
```

その後、右に示す if 文によって、cnt の値はインクリメントされて、1 となります。

while 文による繰返しが行われ、拡張表記¥r を出力することによって、カーソルが行の先頭に移動します。

再び for 文が実行されるときは、

```
for (i = 0; i < 15; i++) {
    if (cnt + 1 < 15)
        putchar(name[1 + i]);
    else
        putchar(name[1 + i - name_len]);
}
```

によって、name[1], name[2], … name[15], name[0]が表示されます。したがって、このとき、"ohYoh Shibata B"と表示されます。

このように、1 文字ずつ、ずらしながら表示することによって、テロップのような表示効果を出させるのです。

```
if (cnt < name_len - 1)
    cnt++;
else
    cnt = 0;
```

毎回の名前の表示を name[cnt]から始めることが分かりましたね。名前の表示が終わるたびに cnt の値をインクリメントしますが、この値が文字列の長さを超えるといけませんから、そのときは 0 に戻します。

# 処理に要した時間の計算

## ■ ４桁の数を記憶するプログラム（その１）

```
/*
     ４桁の数を記憶する（その１）
*/

#include  <time.h>
#include  <stdio.h>
#include  <string.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x) { /*--- 省略 ---*/ }

int main(void)
{
     const int try_no = 10;              /* 挑戦回数 */
     int       i;
     int       success;                  /* 成功回数 */
     time_t    t;                        /* 時刻 */

     srand(time(&t) % RAND_MAX);         /* 乱数の種を初期化 */

     printf("４桁の数値を記憶しましょう。¥n");

     success = 0;
     for (i = 0; i < try_no; i++) {
          int   no;                      /* この数を記憶させる */
          int   x;                       /* 読み込んだ値 */

          no = rand() % 9000 + 1000;     /* 1000～9999の乱数を発生 */

          printf("%d", no);
          sleep(500);                    /* 0.5秒 */
          printf("¥r    ¥r数値は：");
          scanf("%d", &x);

          if (x == no) {
               printf("正解です。¥n");
               success++;
          } else {
               printf("¥a間違いです。¥n");
          }
     }
     printf("%d回中%d回成功しました。¥n", try_no, success);

     return (0);
}
```

## ■ ４桁の数を記憶するプログラム（その２）

```
/*
     ４桁の数を記憶する（その２）
*/

#include  <time.h>
#include  <stdio.h>
#include  <string.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
     /*--- 省略 ---*/
}

int main(void)
{
     const int try_no = 10;              /* 挑戦回数 */
     int       i;
     int       success;                  /* 成功回数 */
     time_t    t;                        /* 時刻 */
     clock_t   start, end;               /* 開始時刻・終了時刻 */

     srand(time(&t) % RAND_MAX);         /* 乱数の種を初期化 */

     printf("４桁の数値を記憶しましょう。¥n");

     success = 0;
     start = clock();
     for (i = 0; i < try_no; i++) {

          /*--- 省略 ---*/

     }
     end = clock();
     printf("%d回中%d回成功しました。¥n", try_no, success);
     printf("%.1f秒でした。¥n", (double)(end - start) / CLOCKS_PER_SEC);

     return (0);
}
```

※　網掛け部が追加部分で、それ以外は、前ページのプログラムと同じです。

関数 sleep 中でも利用している clock 関数によって、プログラム起動時からの時間が得られます。この関数をうまく利用すると、処理に要した時間が計算できます。

# ここまでのテクニックのまとめ

ここまで紹介したテクニックをまとめます（拡張表記は除く）。

## ■ 乱数　<stdlib.h>

以下の宣言をしておき、

```
time_t   t;                    /* 時刻 */
```

一度だけ

```
srand(time(&t) % RAND_MAX);    /* 乱数の種を初期化 */
```

を実行します。

その後は、rand 関数を呼び出せば乱数が得られます。なお、特定の範囲の乱数を得るには、以下のようにします。

```
例  rand() % A                 /* 0 以上     A - 1 以下の乱数を発生*/
例  rand() % A + B             /* B 以上 B + A - 1 以下の乱数を発生*/
```

## ■ 一定時間処理を停止　<time.h>

以下の関数を用意して呼び出します。引数の単位はミリ秒です。

```
/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    clock_t   s = clock();
    clock_t   c;
    do {
        if ((c = clock()) == (clock_t)-1)       /* エラー */
            return (0);
    } while (1000UL * (c - s) / CLOCKS_PER_SEC <= x);
    return (1);
}
```

## ■ 処理に要した時間を計算　<time.h>

以下のように変数を宣言しておき、

```
clock_t   start, end;          /* 開始時刻・終了時刻 */
```

処理の前後でclock関数を呼び出すとよいでしょう。

```
start = clock();
/*--- 処理 ---*/
end = clock();
```

ただし、clock関数が返す時間の単位は処理系に依存します。

処理に要した時間を秒単位で得るためには、それらの差を CLOCKS_PER_SEC で割ります。

```
(end - start) / CLOCKS_PER_SEC
```

なお、実数値で得るには、double型にキャストしなければなりません。

```
(double)(end - start) / CLOCKS_PER_SEC
```

# 脳力テストプログラム

## ■ ２桁の数を記憶して１を加えた値を入力するプログラム（その１）

```
/*
     ２桁の数に１を加えた値を当てる（その１）
*/

#include  <time.h>
#include  <stdlib.h>
#include  <stdio.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
     clock_t    s = clock();
     clock_t    c;
     do {
          if ((c = clock()) == (clock_t)-1)        /* エラー */
               return (0);
     } while (1000UL * (c - s) / CLOCKS_PER_SEC <= x);
     return (1);
}

int main(void)
{
     const int try_no = 10;              /* 挑戦回数 */
     int       i, stage;
     int       level;                    /* レベル */
     int       success;                  /* 成功回数 */
     time_t    t;                        /* 時刻 */
     clock_t    start, end;              /* 開始時刻・終了時刻 */

     srand(time(&t) % RAND_MAX);         /* 乱数の種を初期化 */

     printf("２桁の数値を記憶して１を加えた値を入力しましょう。¥n");

     do {
          printf("挑戦するレベル（2〜5）：");
          scanf("%d", &level);
     } while (level < 2  ||  level > 5);

     success = 0;
     start = clock();
     for (stage = 0; stage < try_no; stage++) {
          int   no[5];               /* この数を記憶させる */
          int   x[5];                /* 読み込んだ値 */
          int   seikai = 0;          /* このステージでの成功数 */
```

```
        for (i = 0; i < level; i++) {
            no[i] = rand() % 90 + 10;  /* 10～99の乱数を発生 */
            printf("%d ", no[i]);
        }
        sleep(500);                          /* 0.5秒 */
        printf("¥r１を加えた値を順に入力してください。¥n");

        for (i = 0; i < level; i++) {
            printf("%d番目の数：", i+1);
            scanf("%d", &x[i]);
        }

        for (i = 0; i < level; i++)
            if (x[i] == no[i] + 1)
                seikai++;
        printf("%d個正解です。¥n", seikai);
        success += seikai;
    }
    end = clock();
    printf("%d個中%d個成功しました。¥n", level * try_no, success);
    printf("%.1f秒でした。¥n", (double)(end - start) / CLOCKS_PER_SEC);

    return (0);
}
```

２桁の値が表示されて直ぐに（0.5 秒で）消えます。それに１を加えた値を入力します。なお、レベルは、表示される数の個数に対応します。たとえば、レベル３であれば、

```
11 95 72
```

と表示されますので、

```
1番目の数：12
2番目の数：96
2番目の数：73
```

と入力しなければなりません（網掛け部）。

さて、レベルを入力する箇所に着目しましょう。

```
do {
    printf("挑戦するレベル（2～5）：");
    scanf("%d", &level);
} while (level < 2  ||  level > 5);
```

変数 level に読み込んだ値が 2 以上 5 以下でなければ、do 文が繰り返され再入力が促されるますね。

すなわち、この do 文が完了したときは、必ず level の値は 2 以上 5 以下となります。

## ■ ２桁の数を記憶して１を加えた値を入力するプログラム（その２）

```
/*
     ２桁の数に１を加えた値を当てる（その２）
*/

#include  <time.h>
#include  <stdlib.h>
#include  <stdio.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
     /*--- 省略 ---*/
}

int main(void)
{
     const int try_no = 10;              /* 挑戦回数 */
     int       i, stage;
     int       level;                    /* レベル */
     int       success;                  /* 成功回数 */
     int       suc[10];                  /* 各ステージでの成功回数 */

     /*--- 途中省略 ---*/

     for (stage = 0; stage < try_no; stage++) {

          /*--- 途中省略 ---*/

          printf("%d個正解です。¥n", seikai);
          suc[stage] = seikai;
          success += seikai;
     }
     end = clock();

     printf("¥n■□成績□■¥n");
     for (stage = 0; stage < try_no; stage++)
          printf("%2dステージ：%d¥n", stage+1, suc[stage]);

     printf("%d個中%d個成功しました。¥n", level * try_no, success);
     printf("%.1f秒でした。¥n", (double)(end - start) / CLOCKS_PER_SEC);

     return (0);
}
```

※　網掛け部が追加部分で、それ以外は、前ページのプログラムと同じです。

このプログラムは、各ステージにおける成功回数を記憶しておき、最後に以下のような表示を行います。

```
■□成績□■
第 1ステージ：3
第 2ステージ：2
第 3ステージ：3
第 4ステージ：1
第 5ステージ：2
第 6ステージ：0
第 7ステージ：1
第 8ステージ：4
第 9ステージ：3
第10ステージ：4
```

各ステージにおける成功回数を記憶するための配列が suc です。C言語やC++での配列の添字は、1 ではなく 0 から始まりますから、第 1 ステージの成功回数は suc[1]ではなく suc[0]に格納されます。

すなわち、第n ステージの成功回数は suc[n-1]に格納されることに注意しましょう。

＊

さて、プログラムの［　　］部を以下のように変更してみましょう。

**■ 2桁の数を記憶して１を加えた値を入力するプログラム（その３）**

```
    printf("¥n■□成績□■¥n");
    for (stage = 0; stage < try_no; stage++) {
        printf("%02dステージ：", stage+1);
        for (i = 0; i < suc[stage]; i++)
            printf("★");
        putchar('¥n');
    }
```

各ステージにおける成功回数が、次に示すように、★によるグラフで表示されます。

```
■□成績□■
第 1ステージ：★★★
第 2ステージ：★★
第 3ステージ：★★★
第 4ステージ：★
第 5ステージ：★★
第 6ステージ：
第 7ステージ：★
第 8ステージ：★★★★
第 9ステージ：★★★
第10ステージ：★★★★
```

さらに、プログラムの［　　］部を以下のように変更してみましょう。

## ■ 2桁の数を記憶して1を加えた値を入力するプログラム（その4）

```
    printf("¥n■□成績□■¥n");
    for (i = level; i >= 1; i--) {
        for (stage = 0; stage < try_no; stage++)
            if (suc[stage] >= i)
                printf(" ★ ");
            else
                printf("    ");
        putchar('¥n');
    }
    printf("----------------------------------------¥n");
    for (stage = 0; stage <try_no; stage++)
        printf(" %02d ", stage);
    putchar('¥n');
```

こうすると、各ステージにおける成功回数が、次に示すように、★による**縦方向の**グラフで表示されます。


**課題**

1を加えた値を入力させるのではなく、1〜9のいずれかの値を加えた値を入力させるように変更せよ。加える値は、ステージのたびに乱数で発生されること。

※たとえば、レベル3の場合は、次のように実行できなければならない。

```
3を加えた値を順に入力してください。
31 97 22                              ← 表示されて消える
1番目の数：34
2番目の数：100
2番目の数：25
7を加えた値を順に入力してください。
15 31 20                              ← 表示されて消える
1番目の数：22
2番目の数：38
2番目の数：27
（以下省略）
```

# 脳力開発ソフトを作ろう

視覚から得られた情報を高速に処理できるよう、情報処理の能力を高めるためのソフトウェアを作成しましょう。

まずは、配列について復習します。

## ■ 配列の要素を 1, 2, …, 9 で初期化して表示

```
/*
    配列を{1, 2, 3, 4, 5, 6, 7, 8, 9}で初期化して表示
*/

#include  <stdio.h>

int main(void)
{
    int   i;
    int   num[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

    for (i = 0; i < 9; i++)
        printf("%d ", num[i]);

    return (0);
}
```

空白文字を忘れないように

num は、要素型が int 型であり、要素数が 9 の配列です。num[0]～num[8]の各要素は、下図に示すように、1, 2, 3, 4, 5, 6, 7, 8, 9 で初期化されます。

| num[0] | num[1] | num[2] | num[3] | num[4] | num[5] | num[6] | num[7] | num[8] |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

C言語では、配列の添字は1でなく、0から始まります。

for 文の働きによって、num[0]～num[8]の値が

**1 2 3 4 5 6 7 8 9**

と表示されます。

## ■ 配列をコピーして表示

```
/*
     配列をコピーして表示
*/

#include  <stdio.h>

int main(void)
{
     int   i;
     int   num[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
     int   a[9];

     for (i = 0; i < 9; i++)              /* 全ての要素をコピー */
          a[i] = num[i];

     for (i = 0; i < 9; i++)
          printf("%d ", a[i]);

     return (0);
}
```

空白文字を忘れないように

このプログラムは、配列num の全要素を配列a にコピーし、その後a[0]～a[8]の値を

**1 2 3 4 5 6 7 8 9**

と表示します。

最初の for 文で、num[0]～num[9]の値を、それぞれa[0]～a[8]にコピーします。

| num[0] | num[1] | num[2] | num[3] | num[4] | num[5] | num[6] | num[7] | num[8] |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

↓　全要素をコピー

| a[0] | a [1] | a [2] | a [3] | a [4] | a [5] | a [6] | a [7] | a [8] |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

なお、C言語では配列の代入はできません。すなわち、

```
a = num;          /* エラー：配列の代入は不可 */
```

といったことはできないので、このように繰返し文によってコピーしないといけないのです。

次に、1〜9 の中のどれか 1 文字を欠けさせ、その数を当てさせるプログラムを作りましょう。たとえば、

**1 2 3 4 6 7 8 9**

と表示した場合は、5 を入力させることにします。

## ■ 1〜9 の並びから欠けている数字を入力させる（試作版）

```
/*
    1〜9の並びから欠けている数字を入力させる（試作版）
*/

#include  <time.h>
#include  <stdio.h>
#include  <stdlib.h>

int main(void)
{
    int      i, j, x;
    int      num[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int      a[9];
    time_t   t;

    srand(time(&t) % RAND_MAX);     /* 乱数の種を初期化 */

    x = rand() % 9;                 /* 0〜8の乱数を発生 */

    i = 0;
    j = 0;
    while (i < 9) {                 /* num[x]を飛ばしてコピー */
        if (i != x)
            a[j++] = num[i];
        i++;
    }

    for (i = 0; i < 8; i++)
        printf("%d ", a[i]);

    return (0);
}
```

9 ではなく 8 になります

変数 x には、0〜8 の乱数を発生させて代入します。これは、欠けさせる数字が格納されている、配列 num の添字です。たとえば、乱数によって得られた値である x が 2 であれば、num[2]に格納されている 3 を欠けさせます。

配列 num から配列 a へのコピーの様子を示したのが下の図です。

i

| num[0] | num[1] | num[2] | num[3] | num[4] | num[5] | num[6] | num[7] | num[8] |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 5 | 6 | 7 | 8 | 9 | |

j

□ 最初は、i も j も 0 です。
□ 繰返しのたびに i の値はインクリメントされます。
□ i の値が x と等しいときはコピーしません。すなわち num[x]の値はスキップされます。
□ j の値は、コピーしたときだけインクリメントされます。
※ a[j++] = num[i];は、a[j] = num[i]; j++;と同じ意味です。

ここまでが理解できれば、これをゲーム感覚で実行できる能力開発ソフトウェアにするのは簡単です。

### ■ 1～9 の並びから欠けている数字を入力させる

```
/*
    1～9の並びから欠けている数字を入力させる
*/

#include  <time.h>
#include  <stdio.h>
#include  <stdlib.h>

int main(void)
{
    const int try_no = 10;          /* 挑戦回数 */
    int       i, j, x;
    int       num[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int       a[9];
    int       stage;
    int       no;                       /* 読み込んだ値 */
    double    jikan;                    /* 時間 */
    clock_t   start, end;           /* 開始時刻・終了時刻 */
```

```
    time_t   t;

    srand(time(&t) % RAND_MAX);     /* 乱数の種を初期化 */

    printf("欠けている数字を入力してください。¥n");

    start = clock();
    for (stage = 0; stage < try_no; stage++) {
        x = rand() % 9;                     /* 0〜8の乱数を発生 */

        i = 0;
        j = 0;
        while (i < 9) {                     /* num[x]を飛ばしてコピー */
            if (i != x)
                a[j++] = num[i];
            i++;
        }

        for (i = 0; i < 8; i++)
            printf("%d ", a[i]);
        printf("：");

        do {
            scanf("%d", &no);
        } while (no != x + 1);
    }
    end = clock();

    jikan = (end - start) / CLOCKS_PER_SEC;

    printf("%.1f秒かかりました。¥n", jikan);

    if (jikan > 25.0)
        printf("鈍すぎます。¥n");
    else if (jikan > 20.0)
        printf("少し鈍いですね。¥n");
    else if (jikan > 17.0)
        printf("まあまあですね。¥n");
    else
        printf("素早いですね。¥n");

    return (0);
}
```

# コンソール入出力

C言語の標準ライブラリであるscanf関数やgetchar関数による読み込みでは、バッファリングなどの関係上、↵キーが押されるまで入力された情報を得ることはできません。

したがって、先ほどのプログラムでは、

**1 2 3 4 6 7 8 9 : 5↵**

と、必ず↵キーをも押す必要があります。

ここでは、Visual C++やBorland C++に特有のgetch関数を利用して、操作性を改善することにします。そうすると、↵キーが押されなくても、押されたキーを読み込めるようになります。この関数を利用したテスト的なプログラム例を示します。

## ■ getch の利用例

```
/*
    getchの利用例（VisualC++以外での動作は保証されない）
*/

#include  <conio.h>
#include  <stdio.h>

int main(void)
{
    int   ch;
    int   retry;

    do {
        printf("¥nキーを押してください。¥n");

        ch = getch();

        printf("押されたキーは%cで値は%dです。¥n", ch, ch);

        printf("もう一度？（Y／N）：");
        retry = getch();
    } while (retry == 'Y'  || retry == 'y');

    return (0);
}
```

文字として表示

文字のコードを数値で表示

プログラムの実行例を示します。

```
キーを押してください。
押されたキーは1で値は49です。
もう一度？（Y／N）：Y
キーを押してください。
（以下省略）
```

この関数を利用して p.23〜p.24 のプログラムを書きかえると、プログラムは次のようになります。

### ■ 1〜9 の並びから欠けている数字を入力させる（getch 関数を利用）

```
/*
    1〜9の並びから欠けている数字を入力させる（getchを利用）
*/

#include  <time.h>
#include  <conio.h>
#include  <stdio.h>
#include  <stdlib.h>

int main(void)
{
    /*--- 中略 ---*/

        for (i = 0; i < 8; i++)
            printf("%d ", a[i]);
        printf("：");

        do {
            no = getch() - '0';
        } while (no != x + 1);

        putchar('¥r');
    }
    /*--- 中略 ---*/
}
```

追　加

変　更

getch 関数によって得られる値は、文字に対応するコードです。数字文字'0', '1', …の文字の値は、文字コード体系によって異なりますが、必ず順に一つずつ増えていく値となります。ちなみに、一般的なパソコンで利用されている ASCII コード体系では、それらは、48, 49, …です。

すなわち、5 キーが押されたら、getch 関数が返す値は 53 です。この値から'0'の値である 48 を引いたら、53 − 48 によって 5 が得られます。数字文字を、対応する数値に変換するには、その数字文字から'0'を引くのが定石です。

＊

このプログラムでは、↵ キーを押す必要がなくなって操作しやすくなるばかりではありません。新しい問題を提示するたびに、それまでの問題が消されますから、欠けている文字を探すのも難しくなります。

# キーボード練習ソフトを作ろう

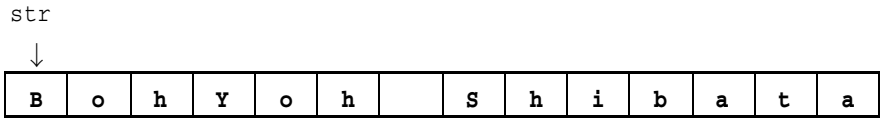
getch関数をうまく利用して、キーボード練習ソフトを作りましょう。

## ■ キーボード練習プログラム（その１）

```
/*
    キーボード練習プログラム（その１）
*/

#include  <time.h>
#include  <conio.h>
#include  <stdio.h>
#include  <string.h>

int main(void)
{
    char *str = "BohYoh Shibata";
    int      i;
    int      len = strlen(str);     /* 文字列strの文字数 */
    double   jikan;                 /* 時間 */
    clock_t  start, end;            /* 開始時刻・終了時刻 */

    printf("%s¥n", str);

    start = clock();                /* 開始時刻 */

    for (i = 0; i < len; i++) {
        int   ch;

        do {
            ch = getch();
            putch(ch);              /* 押されたキーを表示 */
            if (ch != str[i])       /* 違うキーが押されたら */
                putch('¥b');        /* カーソルを一つ戻す   */
        } while (ch != str[i]);
    }

    end = clock();                      /* 終了時刻 */

    jikan = (end - start) / CLOCKS_PER_SEC;

    printf("¥n%.1f秒かかりました。¥n", jikan);

    return (0);
}
```

練習する文字列（適当にかえてください）

下図に示すように、ポインタ str は、文字列リテラル"BohYoh Shibata"の先頭文字'B'を指しています。




このとき、各文字'B', 'o', 'h', …は、先頭から順に str[0], str[1], str[2], …と表わされます。

getch関数によって読み込んだ文字をchに代入すると、

putch(ch)

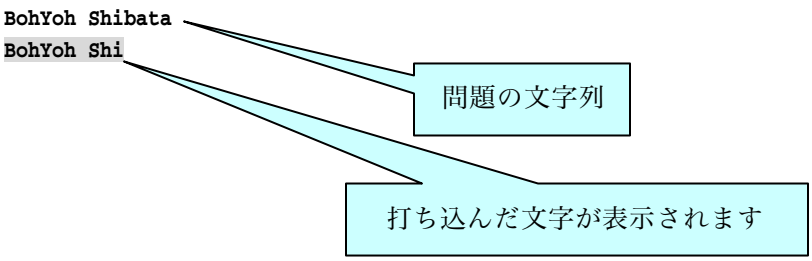
によって、その文字を表示します。

なお、その文字が str[i]と等しければ、正しいキーを打ったことになりますが、等しくなければ、後退文字'¥b'を出力することによって、カーソル位置を戻します。

また、do 文の働きによって、正しい文字が読み込まれるまで、次の文字には進めないことに注意しましょう。

```
do {
     ch = getch();
     putch(ch);              /* 押されたキーを表示 */
     if (ch != str[i])       /* 違うキーが押されたら */
          putch('¥b');       /* カーソルを一つ戻す   */
} while (ch != str[i]);
```

プログラムの実行例を示します。



今度は、打ち込んだ文字が消えていくようにしましょう。プログラムを以下に示します。

## ■ キーボード練習プログラム（その２）

```
/*
     キーボード練習プログラム（その２：打った文字が消えていく）
*/

#include  <time.h>
#include  <conio.h>
#include  <stdio.h>
#include  <string.h>

int main(void)
{
     char  *str = "BohYoh Shibata";
     int    i;
     int    len = strlen(str);        /* 文字列strの文字数 */
     double   jikan;                 /* 時間 */
     clock_t   start, end;           /* 開始時刻・終了時刻 */

     start = clock();                /* 開始時刻 */

     for (i = 0; i < len; i++) {
          int   ch;

          /* str[i]以降を表示してカーソルを先頭へ戻す */
          printf("%s ¥r", &str[i]);

          do {
               ch = getch();
          } while (ch != str[i]);
     }

     end = clock();                        /* 終了時刻 */

     jikan = (end - start) / CLOCKS_PER_SEC;

     printf("¥r%.1f秒かかりました。¥n", jikan);

     return (0);
}
```

スペース文字を忘れないように

```
BohYoh Shibata
      ↓
ohYoh Shibata
      ↓
hYoh Shbiata
      ↓
```

１文字タイプするたびに、
１文字ずつ消えていきます。

文字をタイプするたびに、その文字が消えていきます。先ほどのプログラムよりも高級なことを行っているにもかかわらず、プログラム自体は簡単になっています。

このプログラムを理解するためには、

```
printf("%s ¥r", &str[i]);
```

の部分をきちんと理解しなければなりません。

printf 関数に渡している２番目の引数である&str[i]は、str[i]のアドレスです。したがって、i が 0 であれば、以下のようになります。

```
printf("%s ¥r", &str[0]);     /* str[0]以降の文字を表示 */
     ↓
BohYoh Shibata と表示してカーソルを先頭に戻す
```

また、i が 1 であれば、以下のようになります。

```
printf("%s ¥r", &str[1]);     /* str[1]以降の文字を表示 */
     ↓
ohYoh Shibata と表示してカーソルを先頭に戻す
```

なお、表示する文字数が１文字ずつ減っていきますので、最後尾の文字が画面に残らないように、スペース文字を表示する必要があることに注意しましょう。

*

さて、タイプする文字列が一つだけでは、たいした練習になりません。複数の文字列を練習できるように拡張しましょう。そのプログラムを次ページに示します。

配列 str[0], str[1], str[2], …は、それぞれ文字列"book", "computer", "default"の先頭文字'b', 'c', 'd', …を指します。

## ■ キーボード練習プログラム（その３）

```
/*
    キーボード練習プログラム（その３：問題12問）
*/

#include  <time.h>
#include  <conio.h>
#include  <stdio.h>
#include  <string.h>

int main(void)
{
    char  *str[] = {"book",   "computer", "default",  "comfort",
                    "monday", "power",    "light",    "music",
                    "programming", "dog", "video",    "include" };
    int    i, stage;
    double   jikan;                 /* 時間 */
    clock_t   start, end;           /* 開始時刻・終了時刻 */

    start = clock();                /* 開始時刻 */

    for (stage = 0; stage < 12; stage++) {
        int   len = strlen(str[stage]);      /* 文字列str[stage]の文字数 */
        for (i = 0; i < len; i++) {
            int   ch;

            /* str[stage][i]以降を表示してカーソルを先頭へ戻す */
            printf("%s ¥r", &str[stage][i]);

            do {
                ch = getch();
            } while (ch != str[stage][i]);
        }
    }

    end = clock();                  /* 終了時刻 */

    jikan = (end - start) / CLOCKS_PER_SEC;

    printf("¥r%.1f秒かかりました。¥n", jikan);

    return (0);
}
```

# 単語学習ソフトを作ろう

文字列の配列を用いて、複数の文字列を練習するキーボード練習ソフトを作りました。今度は、英単語を学習するプログラムを作りましょう。

## ■ 単語学習プログラム（その１）

```
/*
    単語学習プログラム（その１：単語の一覧を表示）
*/

#include  <stdio.h>

#define  QNO    12          /* 単語の数 */

/*--- 日本語 ---*/
char *jptr[] = {
    "動物", "車", "花",    "家",    "机",    "本",
    "椅子", "父", "母",    "愛",    "平和",  "雑誌",
};

/*--- 英語 ---*/
char *eptr[] = {
    "animal", "car",  "flower", "house", "desk", "book",
    "chair",  "father",    "mother", "love",  "peace","magazine",
};

int main(void)
{
    int  i;

    for (i = 0; i < 12; i++)
        printf("%s %s¥n", jptr[i], eptr[i]);

    return (0);
}
```

このプログラムは、日本語の単語と、それに対応する英単語の一覧を表示します。

jptr が日本語の単語の配列であり、eptr が英単語の配列であることは分かりますね。たとえば、jptr[1]は〃車〃で、eptr[2]は〃flower〃です。

```
動物 animal
車 car
花 flower
家 house
机 desk
本 book
椅子 chair
父 father
母 mother
（以下省略）
```

それでは、日本語の単語をランダムに表示するように変更しましょう。プログラムを以下に示します。

## ■ 単語学習プログラム（その２）

```
/*
    単語学習プログラム（その２：英単語をランダムに表示）
*/

#include  <time.h>
#include  <stdio.h>
#include  <stdlib.h>

#define  QNO    12          /* 単語の数 */

/*--- 日本語 ---*/
char *jptr[] = {
    "動物", "車","花",    "家",    "机",    "本",
    "椅子", "父","母",    "愛",    "平和",  "雑誌",
};

/*--- 英語 ---*/
char *eptr[] = {
    "animal", "car",  "flower", "house", "desk", "book",
    "chair",  "father",    "mother", "love",  "peace","magazine",
};

int main(void)
{
    int   nq;               /* 問題の番号 */
    int   retry;            /* 再挑戦するか？ */
    time_t   t;             /* 現在の時刻 */

    srand(time(&t) % RAND_MAX);

    do {
        int   no;

        nq = rand() % QNO;

        printf("%s¥n", jptr[nq]);

        printf("もう一度？ 0-いいえ／1-はい：");
        scanf("%d", &retry);
    } while (retry == 1);

    return (0);
}
```

```
机
もう一度？ 0-いいえ／1-はい：1
動物
もう一度？ 0-いいえ／1-はい：1
母
もう一度？ 0-いいえ／1-はい：0
```

次に、日本語もしくは英語の単語をランダムに表示するように変更しましょう。プログラムを以下に示します。

## ■ 単語学習プログラム（その３）

```
/*
    単語学習プログラム（その３：英単語／日単語をランダムに表示）
*/

#include  <time.h>
#include  <stdio.h>
#include  <stdlib.h>

#define  QNO    12          /* 単語の数 */

/*--- 日本語 ---*/
char *jptr[] = { /*--- 省略（前のプログラムと同じ） ---*/ };

/*--- 英語 ---*/
char *eptr[] = { /*--- 省略（前のプログラムと同じ） ---*/ };

int main(void)
{
    int   nq;              /* 問題の番号 */
    int   sw;              /* 0：英語／1：日本語 */
    int   retry;           /* 再挑戦するか？
*/
    time_t   t;            /* 現在の時刻 */

    srand(time(&t) % RAND_MAX);

    do {
        nq = rand() % QNO;

        sw = rand() % 2;

        printf("%s¥n", sw ? eptr[nq] : jptr[nq]);

        printf("もう一度？ 0-いいえ／1-はい：");
        scanf("%d", &retry);
    } while (retry == 1);

    return (0);
}
```

```
動物
もう一度？ 0-いいえ／1-はい：1
車
もう一度？ 0-いいえ／1-はい：1
mother
もう一度？ 0-いいえ／1-はい：1
love
もう一度？ 0-いいえ／1-はい：0
```

変数 sw は、日本語か英語かを表します。

次に、一つの単語を問題として示すだけでなく、さらに、それに対する選択肢を四つ表示し、それから選択させるようにしましょう。たとえば、一つの英単語（日本語の単語）と、四つの日本語の単語（四つの英単語）を選択肢として表示し、それから選択してもらうようにします。

プログラムを示します。

## ■ 単語学習プログラム（その４）

```
/*
    単語学習プログラム（その４：選択肢を表示）
*/

#include  <time.h>
#include  <stdio.h>
#include  <stdlib.h>

#define  QNO    12          /* 単語の数 */
#define  CNO     4          /* 選択肢の数 */

/*--- 日本語 ---*/
char *jptr[] = { /*--- 省略（前のプログラムと同じ） ---*/ };

/*--- 英語 ---*/
char *eptr[] = { /*--- 省略（前のプログラムと同じ） ---*/ };

/*--- 選択肢を作成し正解の添字を返す ---*/
int make_cand(int c[], int n)
{
    int  i;

    c[0] = n;                       /* 解答 */
    for (i = 1; i < CNO; i++) {
        c[i] = rand() % QNO;
    }

    return (0);
}

/*--- 選択肢を表示 ---*/
void print_cand(int c[], int sw)
{
    int  i;
    for (i = 0; i < CNO; i++)
        printf("(%d) %s  ", i, sw ? jptr[c[i]] : eptr[c[i]]);
    printf("：");
}
```

```
int main(void)
{
    int   nq;              /* 問題の番号 */
    int   na;              /* 正解の番号 */
    int   sw;              /* 0：英語→日本語／1：日本語→英語 */
    int   retry;           /* 再挑戦するか？ */
    int   cand[CNO];       /* 選択肢の番号 */
    time_t   t;            /* 現在の時刻 */

    srand(time(&t) % RAND_MAX);

    do {
        int   no;

        nq = rand() % QNO;
        na = make_cand(cand, nq);
        sw = rand() % 2;

        printf("%sはどれですか？¥n", sw ? eptr[nq] : jptr[nq]);

        do {
            print_cand(cand, sw);
            scanf("%d", &no);
            if (no != na)
                puts("違います。");
        } while (no != na);
        puts("正解です。");
        printf("もう一度？ 0-いいえ／1-はい：");
        scanf("%d", &retry);
    } while (retry == 1);

    return (0);
}
```

配列 cand_no には、選択肢の番号を格納します。たとえば、問題として〝花〟を表示した場合は、問題の番号 nq は 2 であり、配列には、以下のように三つの値を乱数として発生して格納します。

| cand_no[0] | cand_no[0] | cand_no[0] | cand_no[0] |
|---|---|---|---|
| 2 | 乱数 | 乱数 | 乱数 |

↑
正解の番号

```
bookはどれですか？
(0) 本　(1) 花　(2) 愛　(3) 愛　：0
正解です。
もう一度？ 0-いいえ／1-はい：1
雑誌はどれですか？
(0) magazine　(1) house　(2) house (3) magazine　: 0
正解です。
もう一度？ 0-いいえ／1-はい：
```

したがって、このプログラムには、以下に示す問題があります。

- 正解が必ず選択肢の先頭にきてしまう。
- 正解以外の三つの選択肢は乱数で発生させるため、たまたま同じ値が得られた場合は、同じ選択肢が表示されることになる。

以上の問題を改良しましょう。以下のようにします。

- ◆ 発生させた乱数が、既に選択肢として格納されていれば、繰り返し乱数を発生させて、重複しないようにする。
- ◆ 四つの重複しない選択肢が決定したら、先頭の番号と、任意の選択肢の番号を交換します)。

関数 make_cand を以下のように変更しましょう。これで、一応完成します。

```
/*--- 選択肢を作成し正解の添字を返す ---*/
int make_cand(int c[], int n)
{
    int   i, j;

    c[0] = n;                                   ← 正解は先頭
    for (i = 1; i < CNO; i++) {
        int   x;
        do {
            x = rand() % QNO;
            for (j = 0; j < i; j++)             ← 既に選択肢となっていない
                if (c[j] == x)                    値が得られるまで、繰り返
                    break;                        し乱数を発生する。
        } while (i != j);
        c[i] = x;
    }
    j = rand() % CNO;
    swap(int, c[0], c[j]);                      ← 先頭と、選択肢のどれか一つを交換

    return (j);
}
```

演習

直前に出された問題と同じ問題を出さないように改良せよ。

# 参考文献

1）American National Standards Institute
“ANSI / ISO 9899-1990 American National Standard for Programming Languages - C”，1992

2）柴田望洋
“明解Ｃ言語入門編”，ソフトバンク，1996

3）柴田望洋
“明解Ｃ言語入門編例解演習”，ソフトバンク，1999

4）柴田望洋
“明解Ｃ言語実践編”，ソフトバンク，2001

5）柴田望洋
“秘伝Ｃ言語問答ポインタ編第２版”，2001