データ構造とアルゴリズムII

Data Structure and Algorithms II

－　２００８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、2008 年度・福岡工業大学 情報工学部 情報工学科２年生の講義

『データ構造とアルゴリズムⅡ』

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。

◆　本資料のプログラムを含むすべての内容は、著作権法上の保護を受けており、著作権者である柴田望洋の許諾を得ることなく、無断で複写・複製をすることは禁じられている。

本資料は、Microsoft 社のワープロソフトウェアである Microsoft Word 2007 を用いて作成した。

Copyright 2008 BohYoh Shibata

１ 基本的なアルゴリズム

■ 三値の最大値

テキスト List1-1 と List1-2 は、三値の最大値を求めるプログラムです。以下のようになっています。

```
int max = a;            /* 最大値 */
if (b > max) max = b;
if (c > max) max = c;
```

もっとも、現実のプログラムで３値の最大値のみが分かればいいということは、ほとんどありません。最大値だけでなく、『どれが最大値であるのか』も同時に求めることが、一般的な要求です。
※たとえば、ゲームにおいて、キャラクタＡ・キャラクタＢ・キャラクタＣの得点の最大値さえ分かればいい、というのではなく、どのキャラクタが最大値であるかを知る必要があるのが、一般的です。
以下に、プログラムを示します。

■ 第１版　三つの整数値の最大値を求める（どれが最大値かも求める）

```
/*
    三つの整数値の最大値を求める(改:どれが最大値であるかも求める)
*/

#include <stdio.h>

int main(void)
{
    int  a, b, c;
    int  max;              /* 最大値 */
    int  flag;             /* どれが最大値であるか(下位３ビットで表す) */
    int  count;            /* 最大値は何個か */

    printf("整数ａの値：");   scanf("%d", &a);
    printf("整数ｂの値：");   scanf("%d", &b);
    printf("整数ｃの値：");   scanf("%d", &c);

    max = a;
    flag = 1;
    count = 1;

    if (b > max) {             /* bだけが最大値 */
        max = b;
```

Copyright 2008 BohYoh Shibata

```
        flag = 2;
        count = 1;
    } else if (b == max) {      /* bも最大値 */
        max = b;
        flag += 2;
        count++;
    }

    if (c > max) {              /* cだけが最大値 */
        max = c;
        flag = 4;
        count = 1;
    } else if (c == max) {      /* cも最大値 */
        max = c;
        flag += 4;
        count++;
    }

    printf("最大値は%dです。¥n", max);
    printf("最大値は以下の変数です:");
    if (flag & 1) { printf(" A "); count--; if (count > 0) printf("と"); }
    if (flag & 2) { printf(" B "); count--; if (count > 0) printf("と"); }
    if (flag & 4) printf(" C ");
    printf("¥n");

    return (0);
}
```

flag は、「最大値であるか」どうかをビットに対応させています。最大値であれば 1、最大値でなければ 0 とします。最下位ビットがａ、下から２番目のビットがｂ、下から３番目のビットがｃに対応します。

いくつかの例を示します。

				10 進数
ａのみが最大値	0	0	1	1
ａとｂが最大値	0	1	1	3
ｂとｃが最大値	1	1	0	4
ａとｂとｃが最大値	1	1	1	7

flag の値は、全部で、1～7 までの 7 パターンがあります。

Copyright 2008 BohYoh Shibata

■ １から n までの和

テキスト List1-3 は、1 から n までの和を求めるプログラムです。和を求める部分は、以下のようになっています。

```
sum = 0;
i = 1;
while (i <= n) {          /* iがn以下であれば繰り返す */
    sum += i;             /* sumにiを加える */
    i++;                  /* iの値をインクリメント */
}
```

```
sum i
 0  1
 1  2
 3  3
 6  4
10  5
15  6
```

右に示しているのは、n が 5 のときの、変数 sum と i の変化です。この値の変化を表示するように、プログラムを修正してみます。以下に、二つのバージョンを示します。

Version 1　加算を行う前に表示

```
sum = 0;
i = 1;
while (i <= n) {
    printf("sum=%3d i=%3d¥n", sum, i);
    sum += i;
    i++;
}
```

```
sum=  0 i=  1
sum=  1 i=  2
sum=  3 i=  3
sum=  6 i=  4
sum= 10 i=  5
1から5までの和は15です。
```

Version 2　加算を行った後に表示

```
sum = 0;
i = 1;
while (i <= n) {
    sum += i;
    i++;
    printf("sum=%3d i=%3d¥n", sum, i);
}
```

```
sum=  1 i=  2
sum=  3 i=  3
sum=  6 i=  4
sum= 10 i=  5
sum= 15 i=  6
1から5までの和は15です。
```

二つのバージョンは、表示のタイミングが異なります。Version1 は、加算を行う前、Version2 は加算を行った後に表示を行っています。

いずれにせよ、n が５であるときに、表示されるのは、5 回だけです。そのため、以下のようになっています。

Version1 … 5 を加算した後の値が表示されない。

Version2 … 1 を加算する前の値が表示されない。

n が 5 のときに、while 文の制御式 i <= n を通過するのは 6 回です。表示を 6 回行うためには、以下のように書きかえる必要があります。

Copyright 2008 BohYoh Shibata

Version 1[改]　while 文の後に表示を追加

```
sum = 0;
i = 1;
while (i <= n) {
    printf("sum=%3d i=%3d¥n", sum, i);
    sum += i;
    i++;
}
printf("sum=%3d i=%3d¥n", sum, i);
```

```
sum=  0 i=  1
sum=  1 i=  2
sum=  3 i=  3
sum=  6 i=  4
sum= 10 i=  5
sum= 15 i=  6
1から5までの和は15です。
```

Version 2[改]　while 文の前に表示を追加

```
sum = 0;
i = 1;
printf("sum=%3d i=%3d¥n", sum, i);
while (i <= n) {
    sum += i;
    i++;
    printf("sum=%3d i=%3d¥n", sum, i);
}
```

```
sum=  0 i=  1
sum=  1 i=  2
sum=  3 i=  3
sum=  6 i=  4
sum= 10 i=  5
sum= 15 i=  6
1から5までの和は15です。
```

いずれも、sum と i を表示する printf 関数の呼出しを追加しています。追加位置は、Version1 では while 文の後、Version2 では while 文の前です。

これで、問題は一応解決しました。しかし、同じプログラム部分（printf 関数の呼出し）が２カ所に埋め込んだことによって、プログラムの保守性が低下しています。

たとえば、sum や i の表示を 3 桁ではなく、4 桁に変更するのであれば、２カ所を両方とも書きかえなければなりません。

while 文の条件判定部を通過する際の sum と i の値を表示すればいいわけですから、表示は、その部分に埋め込むべきです。そのように改良したプログラムを、以下に示します。

```
sum = 0;
i = 1;
while (printf("sum=%3d i=%3d¥n", sum, i), i <= n) {
    sum += i;
    i++;
}
```

ここで利用しているのが、コンマ演算子です。式 a, b は、次のように評価されます。

(1) a を評価・実行する。

(2) b を評価・実行する。

(3) 式 a, b 全体の値としては、(2)の評価によって得られた値とする。

※左オペランドの a は評価・実行されるものの、その値は切り捨てられます。

Copyright 2008 BohYoh Shibata

――

■ 演習 1-3

List 1-4 のプログラムを、たとえば n が 7 であれば『1 から 7 までの和は 28 です。』と表示するのではなく、『1 + 2 + 3 + 4 + 5 + 6 + 7 = 28』と表示するように変更せよ

――

先ほどは、sum と i の値を n + 1 回表示するプログラムを考えました。本問は、逆といえます。”+”の表示回数が n – 1 回であることに注意しましょう（n が 7 のときは、+を表示するのは６回です)。

プログラム例を以下に示します。

Version 0 … i が n 未満のとき／i が n のときで処理を変える

```
sum = 0;

for (i = 1; i <= n; i++) {      /* i = 1, 2, …, n */
    sum += i;                   /* sumにiを加える */
    if (i != n)
        printf("%d + ", i);     ← [A]
    else
        printf("%d = ", i);     ← [B]
}

printf("%d¥n", sum);
```

```
1 + 2 + 3 + 4 + 5 = 15
```

プログラムは、正しく動きます。n が 5 のときは、for 文内での表示は、以下のように行われます。

・i が１のとき　　"1 + " を表示
・i が２のとき　　"2 + " を表示
・i が３のとき　　"3 + " を表示
・i が４のとき　　"4 + " を表示
・i が５のとき　　"5 = " を表示

for 文の繰返しのたびに if 文が実行されます。i が n より小さければ[Ａ]で"i + "と表示し、i が n と等しければ[Ｂ]で"i = "と表示します。

if 文が成立せず、else 以降の[Ｂ]が実行されるのは１回だけです。最後の１回を除いた 9,999 回は、if 文は成立して[Ａ]が実行されます。もし n が 10,000 であれば、最後に１回だけ実行すべき[Ｂ]のために、10,000 回もの判定を行っているわけです。

Copyright 2008 BohYoh Shibata

最後の 1 回だけに成立する（あるいは成立しない）ということが既知であるのに、毎回条件判定を行うのは、明らかに無駄です。

もしも、if 文の判定が、単なる「整数の比較」でなく、「データベースファイルからの検索」といった重たい作業であれば、コストは非常に高くなります。

したがって、i が n のときだけは「特別扱い」すべきです。そのように書き直したプログラムを示します。

Version 1 … まず合計を求めて、それから表示する。

```
sum = 0;

for (i = 1; i <= n; i++) {      /* i = 1, 2, …, n */
    sum += i;                   /* sumにiを加える */
}

for (i = 1; i < n; i++)
    printf("%d + ", i);
printf("%d = %d¥n", n, sum);
```

Version 2 …合計を求めながら表示する。

```
sum = 0;

for (i = 1; i < n; i++) {       /* i = 1, 2, …, n */
    sum += i;                   /* sumにiを加える */
    printf("%d + ", i);
}

sum += i;
printf("%d = %d¥n", i, sum);
```

※for 文の制御式 i < n と i <= n の違いに注意しましょう。

Copyright 2008 BohYoh Shibata

■ 記号文字の表示

以下に示すのは、記号文字★を n 個連続表示するプログラムです。

```
/*
    星をn個表示
*/

#include <stdio.h>

int main(void)
{
    int i, n, w;

    printf("何個表示しますか：");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        printf("★");
    }

    printf("¥n");

    return (0);
}
```

w 個ごとに改行するプログラムは、以下のようになります。

```
/*
    星をn個表示（その２：w個ごとに改行）
*/

#include <stdio.h>

int main(void)
{
    int i, n, w;

    printf("何個表示しますか：");
    scanf("%d", &n);

    printf("１行は何個ですか：");
    scanf("%d", &w);

    for (i = 1; i <= n; i++) {
```

Copyright 2008 BohYoh Shibata

```
        printf("★");
        if (i % w == 0)
            printf("¥n");
    }

    if (n % w != 0) printf("¥n");

    return (0);
}
```

n 回の繰返しのたびに、w で割り切れるかどうかを判定します。その判定のコストは小さくありません。以下のように、２重ループによって実現すると、判定が不要となります。

```
/*
    星をn個表示（その３：w個ごとに改行／２重ループ）

*/

#include <stdio.h>

int main(void)
{
    int i, j, n, w;
    int g, h;

    printf("何個表示しますか：");
    scanf("%d", &n);

    printf("１行は何個ですか：");
    scanf("%d", &w);

    g = n / w;
    for (i = 1; i <= g; i++) {
        for (j = 1; j <= w; j++)
            printf("★");
        printf("¥n");
    }

    h = n % w;
    if (h != 0) {
        for (i = 1; i <= h; i++)
            printf("★");
        printf("¥n");
    }

    return (0);
}
```

Copyright 2008 BohYoh Shibata

※“その２”の考え方（n が 22 で w が 6 であるとする）

```
 1  2  3  4  5  6
★ ★ ★ ★ ★ ★↵
 7  8  9 10 11 12
★ ★ ★ ★ ★ ★↵
13 14 15 16 17 18
★ ★ ★ ★ ★ ★↵
19 20 21 22
★ ★ ★ ★↵
```

6, 12, 18 のように、i が６で割り切れるときに改行する。

※“その３”の考え方（n が 22 で w が 6 であるとする）

```
★ ★ ★ ★ ★ ★↵
★ ★ ★ ★ ★ ★↵
★ ★ ★ ★ ★ ★↵
★ ★ ★ ★↵
```

22 を６で割った商ｇは３。
３行×６個（ｇ行×ｗ個）を表示。

22 を６で割った剰余ｈは４。
残りの４個（ｈ個）を表示。

さらに、『何行ごとに空行を挿入しますか』と y に読み込むようにしましょう。たとえば、n が 49 で w が 6 で y が 3 であれば、以下のように表示します。

```
★★★★★★↵
★★★★★★↵
★★★★★★↵
↵
★★★★★★↵
★★★★★★↵
★★★★★★↵
↵
★★★★★★↵
★★★★★★↵
★↵      ← ここまで。
↵       ← これは駄目!!
```

実用例:
プリンタにｎ人の一覧を出力する。
１行にｗ人の名前を出力する。
１頁に出力できるのはｙ行である。
※最後に ２ 回以上改行すると、プリンタ用紙を １ 枚無駄にしてしまうことになります。

Copyright 2008 BohYoh Shibata

```
/*
    ★をn個表示（その３：w個ごとに改行・y行ごとに空行）
*/

#include <stdio.h>

int main(void)
{
    int  i, j, n, w, y;
    int g, h;

    printf("何個表示しますか：");
    scanf("%d", &n);

    printf("1行は何個ですか：");
    scanf("%d", &w);

    printf("何行ごとに空行を挿入しますか：");
    scanf("%d", &y);

    g = n / w;
    h = n % w;

    for (i = 1; i <= g; i++) {
        for (j = 1; j <= w; j++)
            printf("★");
        printf("¥n");
        if (i % y == 0 && (i < g || h != 0))
            printf("¥n");
    }

    if (h != 0) {
        for (i = 1; i <= h; i++)
            printf("★");
        printf("¥n");
    }

    return (0);
}
```

Copyright 2008 BohYoh Shibata

■ C++のテンプレート

C++では、型に依存しないアルゴリズムを、関数テンプレートによって実現することができます。

```
/*
    三つの整数値の最大値を求めるmax3.cpp
*/

#include  <stdio.h>

template<class Type>Type max(Type a, Type b, Type c)
{
    Type max = a;
    if (b > max) max = b;
    if (c > max) max = c;
    return max;
}

int main(void)
{
    int  a, b, c;
    double x, y, z;

    printf("整数ａの値：");    scanf("%d", &a);
    printf("整数ｂの値：");    scanf("%d", &b);
    printf("整数ｃの値：");    scanf("%d", &c);

    printf("実数xの値：");     scanf("%lf", &x);
    printf("実数yの値：");     scanf("%lf", &y);
    printf("実数zの値：");     scanf("%lf", &z);

    printf("a, b, cの最大値は%dです。¥n", max(a, b, c));
    printf("x, y, zの最大値は%fです。¥n", max(x, y, z));

    return (0);
}
```

ここに示す関数テンプレート max は、三値の最大値を求めるアルゴリズムを実現したものです。=による初期化・代入が可能であり、>による大小関係の判定が可能である型であれば、どんな型でも（int でも double でも、自作のクラス型でも）適用することができます。

Copyright 2008 BohYoh Shibata

■ Java による三値の最大値

三値の値を求めるプログラムの Java 版を示します。

```
// 三つの整数値を読み込んで最大値を求めて表示

import java.util.Scanner;

class Max3 {

    public static void main(String[] args) {
        Scanner stdIn = new Scanner(System.in);

        System.out.println("三つの整数の最大値を求めます。");
        System.out.print("aの値：");int a = stdIn.nextInt();
        System.out.print("bの値：");int b = stdIn.nextInt();
        System.out.print("cの値：");int c = stdIn.nextInt();

        int max = a;
        if (b > max) max = b;
        if (c > max) max = c;

        System.out.println("最大値は" + max + "です。");
    }
}
```

■演習

三値の最大値を求める部分をメソッド（関数）として独立させてみよう。

Copyright 2008 BohYoh Shibata

――

■ 演習 1-5

正の整数値を読み込んで、その値の桁数を表示するプログラムを作成せよ。たとえば、135 を読み込んだら『その数は3 桁です。』と表示し、1314 を読み込んだら『その数は4 桁です。』と表示すること。

――

解答を以下に示します。

```
/*
    演習1-5
    正の整数値の桁数を求める
*/

#include <stdio.h>

int main(void)
{
    int  x;
    int  no = 0;        /* 桁数 */

    printf("正の整数値：");
    scanf("%d", &x);

    while (x > 0) {
        x /= 10;    /* xを10で割る */
        no++;
    }

    printf("その数は%d桁です。¥n", no);

    return (0);
}
```

整数値を 10 で割ると、最下位の桁が弾き出されます。たとえば、1314 を 10 で割ったら、最下位の 4 が弾き出されて、131 になります。すべての桁が弾き出されるまで 10 で割る作業を繰り返します。割った回数が、桁数と一致します。

n が 1314 であれば、x と弾き出される値と no は、以下のように変化します。

x	弾き出される値	no
1314	4	1
131	1	2
13	3	3
1	1	4
0		

Copyright 2008 BohYoh Shibata

弾き出される値を表示すると、4131 となります。すなわち、もともとの x を逆順に表示することができます。そのように書きかえたプログラムを以下に示します。

```
/*
    演習1-5 [改造]
    正の整数値を逆順に表示する
*/

#include <stdio.h>

int main(void)
{
    int  x;

    printf("正の整数値：");
    scanf("%d", &x);

    while (x > 0) {
        printf("%d", x % 10);   /* 10で割った余りを表示 */
        x /= 10;     /* xを10で割る */
    }

    return (0);
}
```

前のプログラムと基本的な構造は同じです。

弾き出されることになる値を表示するだけです。そのために、x を 10 で割る直前に、x を 10 で割った余りを表示します。

Copyright 2008 BohYoh Shibata

２ 基本的なデータ構造

■ 配列の最大値を求める

テキスト List2-4 は、配列の最大値を求めるプログラムです。以下のようになっています。

```
int  i;
int  max = a[0];            /* 最大値 */
for (i = 1; i < n; i++)
    if (a[i] > max) max = a[i];
```

「３値の最大値」と同様に、最大値のみが分かればいいということは、ほとんどありません。『どれが最大値であるのか』も同時に求めることが、一般的な要求です。

「３値の最大値」では、対象データが 3 個でしたので、「どれが最大値であるのか」のデータを、int 型のビットで表せました。しかし、配列の要素は 100 個とか 10,000 個とかになるかもしれません。ビットで表すのは、不可能ではありませんが困難です。

「どれが最大値であるのか」も、配列で表すことにすると、プログラムは次のようになります。

■ 配列要素の最大値を求める（どれが最大値かも求める）・第１版

```
/*
    配列の要素の最大値を求める（第１版）
*/

#include  <stdio.h>

/*--- 要素数nの配列aの要素の最大値を求める ---*/
int maxof(const int a[], int n, int idx[], int *count)
{
    int  i;
    int  max = a[0];            /* 最大値 */
    *count = 1;                 /* 最大値はa[0]の1個だけ */
    idx[0] = 0;                 /* その《添字》0を登録 */

    for (i = 1; i < n; i++)
        if (a[i] == max) {      /* 暫定最大値と同じ値に出会ったら */
            ++*count;           /* カウンタを1増やす */
            idx[*count -1] = i; /* 《添字i》を追加登録 */

        } else if (a[i] > max) {/* 暫定最大値よりも大きな値に出会ったら */
            max = a[i];
            *count = 1;     /* これまでの登録をクリアしてカウンタを1にリセット*/
            idx[0] = i;     /* 《添字i》を登録 */
        }
```

Copyright 2008 BohYoh Shibata

```
    return (max);
}

int main(void)
{
    int  i;
    int  x[10];
    int  idx[10];                          /* 最大値の添字を格納する配列 */
    int  count;                            /* 最大値の個数 */
    int  nx = sizeof(x) / sizeof(x[0]);        /* 配列xの要素数 */

    printf("%d個の整数を入力してください。¥n", nx);
    for (i = 0; i < nx; i++) {
        printf("x[%d] : ", i);
        scanf("%d", &x[i]);
    }

    printf("最大値は%dです。¥n", maxof(x, nx, idx, &count));
    printf("以下の%d個です。{", count);
    for (i = 0; i < count; i++)
        printf("a[%d],", idx[i]);
    printf("¥n", count);

    return (0);
}
```

配列 idx には、最大値である要素の「添字」を格納します。たとえば、以下の例では、配列 a のインデックスが 7 の要素まで走査したときの a と idx の様子です。a[1]と a[3]と a[7]が最大値であり、配列 idx の先頭には 1, 3, 7 を入れています。

	0	1	2	3	4	5	6	7	8	9	10	11
a	32	55	10	55	42	39	18	55				

idx	1	3	7									

ここで、a[8]が 55 よりも大きい値であれば、a[8]だけが最大値となります。そのため、せっかく idx に入れていた「添字」を全部クリアすることになり、次のようになります。

	0	1	2	3	4	5	6	7	8	9	10	11
a	32	55	10	55	42	39	18	55	90			

idx	8											

Copyright 2008 BohYoh Shibata

要素が 10,000 個であり、最後の要素のみが最大値であれば、そこまで求めた最大値の「添字」の集合は無駄となます。それまで得られた結果が、一気にひっくり返されることになるのですから。

＊

要素が多いときは、以下のように、２段階に分けて処理を行ったほうが、遙かに効率よく求めることができます。

・最大値を求める

・最大値と同じ値の添字をすべて抽出する。

このように書きかえたプログラムを以下に示します。

■ 配列要素の最大値を求める（どれが最大値かも求める）・第２版

```
/*
    配列の要素の最大値を求める（第２）
*/

#include <stdio.h>

/*--- 要素数nの配列aの要素の最大値を求める ---*/
int maxof(const int a[], int n, int idx[], int *count)
{
    int i;
    int max = a[0];             /* 最大値 */
    for (i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];

    *count = 0;                 /* 最大値の個数 */
    for (i = 0; i < n; i++)
        if (a[i] == max)
            idx[(*count)++] = i;

    return (max);
}

int main(void)
{
    int i;
    int x[10];
    int idx[10];                            /* 最大値の添字を格納する配列 */
    int count;                              /* 最大値の個数 */
    int nx = sizeof(x) / sizeof(x[0]);          /* 配列xの要素数 */

    printf("%d個の整数を入力してください。¥n", nx);
    for (i = 0; i < nx; i++) {
        printf("x[%d] : ", i);
```

Copyright 2008 BohYoh Shibata

```
        scanf("%d", &x[i]);
    }

    printf("最大値は%dです。¥n", maxof(x, nx, idx, &count));
    printf("以下の%d個です。{", count);
    for (i = 0; i < count; i++)
        printf("a[%d],", idx[i]);
    printf("¥n", count);
    return (0);
}
```

プログラムが短くスッキリしているだけでなく、明らかに実行効率もよくなります。
※計算量は n です。

Copyright 2008 BohYoh Shibata

■ 基数変換

テキスト List2-8 は、10 進数で読み込んだ整数を、2 進数～36 進数の基数に変換して表示するプログラムです。

変換の過程を表示するように書きかえたプログラムを以下に示します。

■ 基数変換

```
/*
    整数を2進数～36進数へと基数変換を行う[改]
*/

#include <stdio.h>

/*--- 整数値xをn進数に変換して配列dに下位桁から格納 ---*/
int card_convr(unsigned x, int n, char d[])
{
    char  dchar[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int   digits = 0;                   /* 変換後の桁数 */

    if (x == 0)                         /* 0であれば */
        d[digits++] = dchar[0];         /* 変換後も0 */
    else
        while (x) {
            printf("%2d  | %5d  … %c¥n", n, x, dchar[x % n]);
            printf("    └────────¥n");
            d[digits++] = dchar[x % n];     /* nで割った剰余を格納 */
            x /= n;
        }
    return (digits);
}

int main(void)
{
    int       i;
    unsigned  no;           /* 変換する整数 */
    int       cd;           /* 基数 */
    int       dno;          /* 変換後の桁数 */
    char  cno[100];     /* 変換後の各桁を格納する文字の配列 */
    int       retry;    /* もう一度？ */

    do {
        printf("正の整数値を入力してください：");
        scanf("%u", &no);

        do {
            printf("何進数に変換しますか (2-36) ：");
            scanf("%d", &cd);
```

Copyright 2008 BohYoh Shibata

```
        } while (cd < 2  ||  cd > 36);

        dno = card_convr(no, cd, cno);          /* noをcd進数に変換 */

        printf("%d進数では", cd);
        for (i = dno - 1; i >= 0; i--)          /* 上位桁から順に表示 */
            printf("%c", cno[i]);
        printf("です。¥n");

        printf("もう一度しますか (1…はい／0…いいえ) :");
        scanf("%d", &retry);
    } while (retry == 1);

    return (0);
}
```

2 進数は 0 と 1 の 2 個の数字を使い、10 進数は 0～9 までの 10 個の数字を使います。36 進数では、0～Z までの 36 個の数字を使って表現します。

配列 dchar は、0～9 と A～Z の各文字が並んだ配列です。以下のようになっています。

dchar[0]	文字'0'
dchar[1]	文字'1'
…	…
dchar[8]	文字'8'
dchar[9]	文字'9'
dchar[10]	文字'A'
dchar[11]	文字'B'
…	…
dchar[34]	文字'Y'
dchar[35]	文字'Z'

実行結果の一例を示します。

```
正の整数値を入力してください：123
何進数に変換しますか（2-36）：16
16 |   123  … B
   └──────
16 |     7  … 7
   └──────
         0
```

Copyright 2008 BohYoh Shibata

■ 素数を求める

テキスト List2-9～List2-11 は、素数を求めるプログラムです。アルゴリズムによって、効率が違う（この場合、乗除算の回数）がまったく異なることに注意しましょう。

■ 素数を求める各アルゴリズムを視覚的に比較

```
/*
    1,000以下の素数を列挙（第１版～第３版）
*/

#include  <stdio.h>
#include  "display.h"

/*--- 第１版 ---*/
void display_v1(int n, int prime[], int ptr)
{
    int i, flag = 0;
    static long int counter = 0;

    locate(1, 6); color(LIGHT_GREEN);
    printf("■第１版----------------------------------------------【乗除算の回数＝
】¥n");

    color(LIGHT_CYAN);
    for (i = 2; i < n; i++) {
        counter++;
        if (n % i == 0) {
            flag = i;
            color(LIGHT_YELLOW);
            printf("%4d", i++);
            break;
        }
        printf("%4d", i);
    }
    color(GRAY);
    for ( ; i < n; i++)
        printf("%4d", i);

    locate(72, 6); color(LIGHT_CYAN); printf("%4ld", counter);

    locate(1, 13); color(WHITE);
    if (flag != 0)
        printf("%dで割り切れたので素数ではないと判断します。¥n", flag);
    else
        printf("上記の全整数で割りきれないので素数と判断します。¥n", n);
    getch();
}
```

Copyright 2008 BohYoh Shibata

```
/*--- 第２版 ---*/
void display_v2(int n, int prime[], int ptr)
{
    int i, flag = 0;
    static long int counter = 0;

    locate(1, 15); color(LIGHT_GREEN);
    printf("■第２版----------------------------------------【乗除算の回数＝
】¥n");

    if (n % 2 == 0) {
        locate(72, 15); color(LIGHT_CYAN); printf("%4ld", counter);
        locate(1, 18); color(WHITE);
        printf("偶数ですから素数でないことは自明です。除算は行いません。¥n");
        getch();
        return;
    }

    color(LIGHT_CYAN);
    for (i = 1; i < ptr; i++) {
        counter++;
        if (n % prime[i] == 0) {
            flag = i;
            color(LIGHT_YELLOW);
            printf("%4d", prime[i++]);
            break;
        }
        printf("%4d", prime[i]);
    }
    color(GRAY);
    for ( ; i < ptr; i++)
        printf("%4d", prime[i]);

    locate(72, 15); color(LIGHT_CYAN); printf("%4ld", counter);

    locate(1, 18); color(WHITE);
    if (flag != 0)
        printf("%dで割り切れたので素数ではないと判断します。¥n", prime[flag]);
    else
        printf("上記の全整数で割りきれないので素数と判断します。¥n", n);
    getch();
}

/*--- 第３版 ---*/
void display_v3(int n, int prime[], int ptr)
{
    int i, flag = 0;
    static long int counter = 0;
```

Copyright 2008 BohYoh Shibata

```
    locate(1, 20); color(LIGHT_GREEN);
    printf("■第３版----------------------------------------------【乗除算の回数＝
】¥n");

    if (n % 2 == 0) {
        locate(72, 20); color(LIGHT_CYAN); printf("%4ld", counter);
        locate(1, 23); color(WHITE);
        printf("偶数ですから素数でないことは自明です。除算は行いません。¥n");
        getch();
        return;
    }

    color(LIGHT_CYAN);
    for (i = 1; counter++, prime[i]*prime[i] <= n; i++) {
        counter++;
        if (n % prime[i] == 0) {
            flag = i;
            color(LIGHT_YELLOW);
            printf("%4d", prime[i++]);
            break;
        }
        printf("%4d", prime[i]);
    }
    color(GRAY);
    for ( ; prime[i]*prime[i] <= n; i++)
        printf("%4d", prime[i]);

    locate(72, 20); color(LIGHT_CYAN); printf("%4ld", counter);

    locate(1, 23); color(WHITE);
    if (flag != 0)
        printf("%dで割り切れたので素数ではないと判断します。¥n", prime[flag]);
    else
        printf("上記の全整数で割りきれないので素数と判断します。¥n", n);
    getch();
}

void display_information(int n, int prime[], int ptr)
{
    int i;

    cls();
    locate( 1,  1);  color(LIGHT_CYAN);   printf("現在判定中の整数");
    locate(21,  1);  color(LIGHT_GREEN);  printf("%4d", n);

    locate(  1,  2);  color(LIGHT_CYAN);   printf("現在までに求められた素数は%d個で
す。¥n", ptr);
```

Copyright 2008 BohYoh Shibata

```
    color(LIGHT_MAGENTA);
    for (i = 0; i < ptr; i++)
        printf("%4d", prime[i]);

    display_v1(n, prime, ptr);
    display_v2(n, prime, ptr);
    display_v3(n, prime, ptr);
}

int main(void)
{
    int  i, n;
    int  prime[500];                /* 素数を格納する配列 */
    int  ptr = 0;                       /* 既に得られた素数の個数 */
    unsigned long  counter = 0;         /* 除算の回数 */

    prime[ptr++] = 2;       /* ２は素数である */
    prime[ptr++] = 3;       /* ３も素数である */

    for (n = 5 ; n <= 100; n ++) {
        display_information(n, prime, ptr);
        if (n % 2) {
            for (i = 1; i < ptr; i++) {         /* 既に得られた素数で割ってみる */
                if (n % prime[i] == 0)          /* 割り切れると素数ではない */
                    break;                  /* それ以上の繰返しは不要 */
            }
            if (ptr == i) {                 /* 最後まで割り切れなかった */
                prime[ptr++] = n;               /* 配列に登録 */
            }
        }
    }

    return (0);
}
```

```
C:¥WINDOWS¥system32¥cmd.exe
現在判定中の整数       67
現在までに求められた素数は18個です。
   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61

■第１版----------------------------------------------【乗除算の回数＝   602 】
   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21
  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41
  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61
  62  63  64  65  66

上記の全整数で割りきれないので素数と判断します。

■第２版----------------------------------------------【乗除算の回数＝   174 】
   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61

上記の全整数で割りきれないので素数と判断します。

■第３版----------------------------------------------【乗除算の回数＝   117 】
   3   5   7

上記の全整数で割りきれないので素数と判断します。
```

Copyright 2008 BohYoh Shibata

■ 式と評価

《式》と《評価》についてきちんと理解しましょう。

◆ 式

式（expression）は、以下のものの総称です。

・変数
・定数
・変数や定数を演算子で結合したもの

以下の式を例に考えましょう。

abc + 32

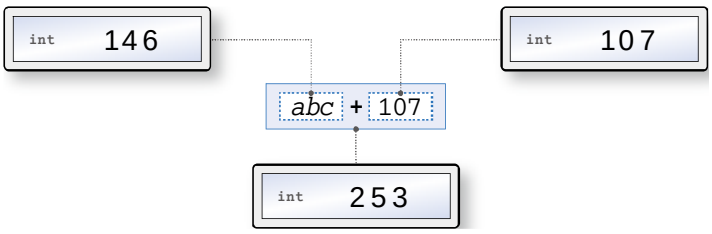
変数 abc、整数定数 32、それらを+演算子で結んだ abc + 32 のいずれもが式です。

◆ 評価

式には、基本的に値があります。その値は、プログラム実行時に調べられます。式の値を調べることを評価（evaluation）といいます。

以下の図を見てください（変数 abc は int 型で、値が 146 であるとします)。もちろん、abc, 107, abc + 107 のいずれもが式です。

変数 abc の値が 146 ですから、それぞれの式を評価した値は 146, 107, 253 となります。もちろん、三つの値の型はいずれも int 型です。



ここでは、ディジタル温度計のような図で評価値を示すことにします。左側の小さな文字が≪型≫で、右側の大きな文字が≪値≫です。

Copyright 2008 BohYoh Shibata

■ 条件演算子

以下に示すのは、変数 a と b の小さい方の値を max に代入する if 文です。

```
if (a < b)
    min = a;
else
    min = b;
```

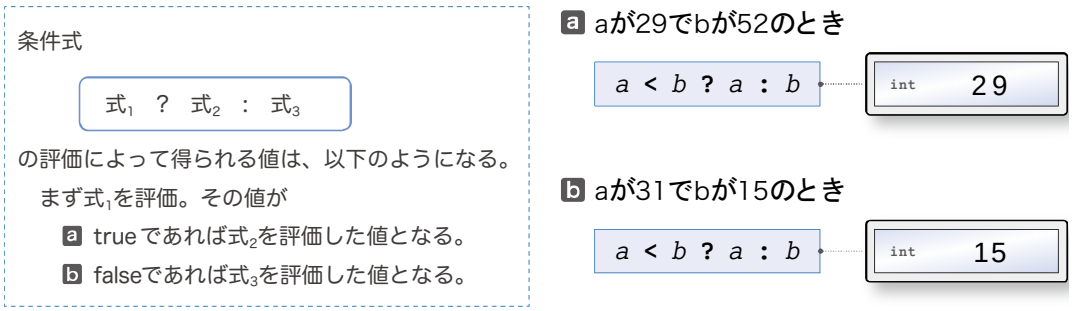
これは、if 文を用いずに、以下のように簡潔に実現できます。

```
min = a < b ? a : b;
```

ここで利用している?: は、条件演算子（conditional operator）です。条件演算子を用いた式のことを、条件式（conditional expression）と呼びます。

条件演算子を用いた条件式 x ? y : z は、x が非ゼロであれば y を評価した値を、そうでなければ z を評価した値を生成します。

したがって、上に示した式で行われる評価は、下図のように行われることになります。変数 min に入れられるのは、a が b より小さければ a の値、そうでなければ b の値となります。



例

a == 5 ? 3 : 4

　　a が 5 であれば　　… 3

　　そうでなければ　　… 4。

a < 5 ? (b < 7 ? 1 : 2) : 3

　　a が 5 未満であれば　… b が 7 未満であれば 1、そうでなければ 2。

　　そうでなければ　　… 3。

Copyright 2008 BohYoh Shibata

3　探索

■ 線形探索

探索の様子を視覚的に表示するプログラムを以下に示します。

■ 線形探索

```
/*
    線形探索[改]
*/

#include  <stdio.h>
#include  "display.h"

/*--- 要素数nの配列aからkeyと一致する要素を線形探索（第２版） ---*/
int search(const int a[], int n, int key)
{
    int  i, j;

    cls();
    printf("                                  i          a[i]");
    for (i = 0; i < n; i++) {
        locate(1, i + 2);
        for (j = 0; j < n; j++) {
            color((i == j) ? (a[j] == key) ? LIGHT_YELLOW : LIGHT_GREEN :
                  (i < j) ? GRAY : LIGHT_BLUE);
            printf("%4d", a[j]);
        }
        color(LIGHT_CYAN);  printf("%5d", i);
        color(WHITE);       printf("  着目要素");
        color(LIGHT_GREEN); printf("%3d", a[i]);
        color(WHITE);
        if (a[i] == key)
            printf("は目的とする値と一致します。¥n");
        else
            printf("は目的とする値と一致しません。¥n");
        getch();
        if (a[i] == key)
            return (i);         /* 探索成功 */
    }
    return (-1);            /* 探索失敗 */
}

int main(void)
{
    int  i, ky, idx;
```

Copyright 2008 BohYoh Shibata

```
    int  x[7];
    int  nx = sizeof(x) / sizeof(x[0]);

    printf("%d個の整数を入力してください。¥n", nx);
    for (i = 0; i < nx; i++) {
        printf("x[%d]：", i);
        scanf("%d", &x[i]);
    }
    printf("探す値：");
    scanf("%d", &ky);

    idx = search(x, nx, ky);    /* 配列xから値がkyである要素を線形探索 */

    if (idx == -1) {
        color(LIGHT_RED);
        printf("%dと一致する要素はありませんでした。¥n", ky);
        puts("探索に失敗しました。");
    } else {
        color(LIGHT_CYAN);
        printf("%dは%d番目にあります。¥n", ky, idx + 1);
    }
    color(WHITE);

    return (0);
}
```

```
cmd.exe
                                    i          a[i]
  1   3   5   6   2   4   7    0  着目要素  1は目的とする値と一致しません。
  1   3   5   6   2   4   7    1  着目要素  3は目的とする値と一致しません。
  1   3   5   6   2   4   7    2  着目要素  5は目的とする値と一致しません。
  1   3   5   6   2   4   7    3  着目要素  6は目的とする値と一致しません。
  1   3   5   6   2   4   7    4  着目要素  2は目的とする値と一致します。
2は5番目にあります。

D:¥C¥講義¥2008¥データ構造とアルゴリズムⅡ¥chap03>
```

Copyright 2008 BohYoh Shibata

■ ２分探索

探索の様子を視覚的に表示するプログラムを以下に示します。

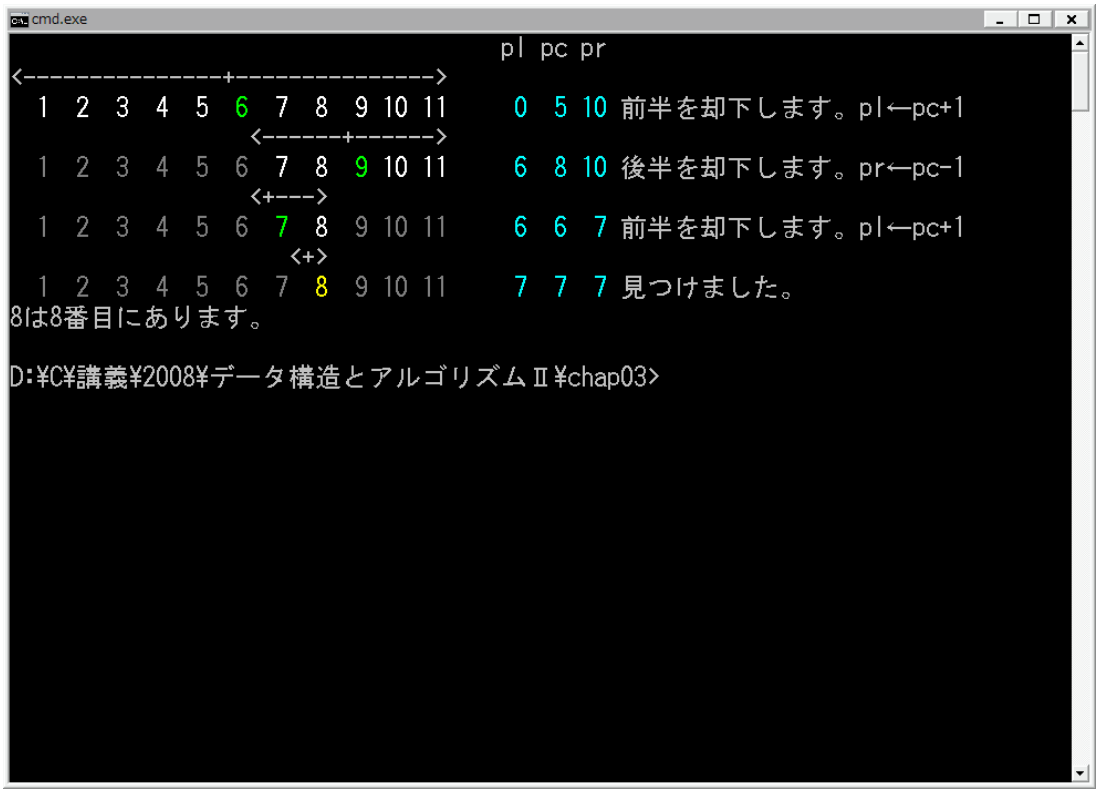
■ ２分探索

```
/*
    ２分探索[改]
*/

#include <stdio.h>
#include "display.h"

/*--- 要素数nの配列aからkeyと一致する要素を２分探索 ---*/
int bin_search(const int a[], int n, int key)
{
    int pl = 0;     /* 探索範囲先頭の添字 */
    int pr = n - 1;/*    〃    末尾の添字 */
    int pc;         /*    〃    中央の添字 */
    int count = 1;      /* 何回目か */

    cls();
    printf("                                        pl pc pr¥n");
    do {
        int j;
        pc = (pl + pr) / 2;

        for (j = 0; j < (pl)*3; j++) printf(" ");
        printf("<");
        for (j = 0; j < (pc - pl)*3; j++) printf("-");
        printf("+");
        for (j = 0; j < (pr - pc)*3; j++) printf("-");
        printf(">¥n");
        for (j = 0; j < n; j++) {
            color((j < pl || j > pr) ? GRAY
                                     : (j == pc) ? (a[pc] == key) ? LIGHT_YELLOW
                                                                  : LIGHT_GREEN
                                                 : BLIGHT_WHITE);
            printf("%3d", a[j]);
        }
        color(LIGHT_CYAN);
        printf("   %3d%3d%3d", pl, pc, pr);
        color(WHITE);
        if (a[pc] == key) {     /* 探索成功 */
            printf(" 見つけました。¥n");
            return (pc);
        } else if (a[pc] < key) {
            pl = pc + 1;
```

Copyright 2008 BohYoh Shibata

```
            printf(" 前半を却下します。pl←pc+1¥n");
        } else {
            pr = pc - 1;
            printf(" 後半を却下します。pr←pc-1¥n");
        }
        count++;
        getch();
    } while (pl <= pr);

    return (-1);            /* 探索失敗 */
}

int main(void)
{
    int  i, ky, idx;
    int  x[11];
    int  nx = sizeof(x) / sizeof(x[0]);

    printf("%d個の整数を昇順に入力してください。¥n", nx);

    printf("x[0]：");
    scanf("%d", &x[0]);

    for (i = 1; i < nx; i++) {
        do {
            printf("x[%d]：", i);
            scanf("%d", &x[i]);
        } while (x[i] < x[i - 1]);  /* 一つ前の値よりも小さければ再入力 */
    }
    printf("探す値：");
    scanf("%d", &ky);

    idx = bin_search(x, nx, ky);/* 配列xから値がkyである要素を２分探索 */

    if (idx == -1)
        puts("探索に失敗しました。");
    else
        printf("%dは%d番目にあります。¥n", ky, idx + 1);

    return (0);
}
```

Copyright 2008 BohYoh Shibata



２分探索は、探索速度が速い反面、『探索すべき値と同じ値の要素が複数存在する場合、最も先頭の要素を見つけるとは限らない。』という欠点があります。

最も先頭の要素を見つけるように改良したプログラムを示します。

■ ２分探索 … 一致する先頭要素の添字を返却

```
/*
    ２分探索［改］一致する先頭要素の添字を返却
*/

#include <stdio.h>

/*--- 要素数nの配列aからkeyと一致する要素を２分探索 ---*/
int bin_search(const int a[], int n, int key)
{
    int pl = 0;     /* 探索範囲先頭の添字 */
    int pr = n - 1;/*    〃    末尾の添字 */
    int pc;         /*    〃    中央の添字 */

    do {
```

Copyright 2008 BohYoh Shibata

```
        pc = (pl + pr) / 2;
        if (a[pc] == key) {     /* 探索成功 */
            while (a[pc - 1] == key && pc > 0)
                pc--;
            return (pc);
        } else if (a[pc] < key)
            pl = pc + 1;
        else
            pr = pc - 1;
    } while (pl <= pr);

    return (-1);            /* 探索失敗 */
}

int main(void)
{
    int  i, ky, idx;
    int  x[7];
    int  nx = sizeof(x) / sizeof(x[0]);

    printf("%d個の整数を昇順に入力してください。¥n", nx);

    printf("x[0] : ");
    scanf("%d", &x[0]);

    for (i = 1; i < nx; i++) {
        do {
            printf("x[%d] : ", i);
            scanf("%d", &x[i]);
        } while (x[i] < x[i - 1]);  /* 一つ前の値よりも小さければ再入力 */
    }
    printf("探す値 : ");
    scanf("%d", &ky);

    idx = bin_search(x, nx, ky);/* 配列xから値がkyである要素を２分探索 */

    if (idx == -1)
        puts("探索に失敗しました。");
    else
        printf("%dは%d番目にあります。¥n", ky, idx + 1);

    return (0);
}
```

Copyright 2008 BohYoh Shibata

6　ソート

■ 単純交換ソート

テキストでは、3種類の単純交換ソートのプログラムを示しています。それらの交換の様子を視覚化して表示するプログラムを以下に示します。

■第1版は、いわゆる単純交換ソートのアルゴリズムをそのまま実現したものです。二要素の比較と交換を行うパスを n－1 回実行します。

■ 単純交換ソート（第1版）

```
/*--- 単純交換ソート ---*/
void bubble(int a[], int n)
{
    int  i, j, m;
    int  ccnt = 0;      /* 比較回数 */
    int  scnt = 0;      /* 交換回数 */

    for (i = 0; i < n - 1; i++) {
        printf("パス%d：¥n", i + 1);
        for (j = n - 1; j > i; j--) {
            for (m = 0; m < n - 1; m++)
                printf("%3d %c", a[m], (m != j-1) ? ' ' :
                                    (a[j - 1] > a[j]) ? '+' : '-');
            printf("%3d¥n", a[n - 1]);

            ccnt++;
            if (a[j - 1] > a[j]) {
                scnt++;
                swap(int, a[j - 1], a[j]);
            }
        }
        for (m = 0; m < n; m++)
            printf("%3d  ", a[m]);
        putchar('¥n');
    }
    printf("比較は%d回でした。¥n", ccnt);
    printf("交換は%d回でした。¥n", scnt);
}
```

Copyright 2008 BohYoh Shibata

■各パスにおける交換回数をカウントしておきます。あるパスで交換が１回もなければ、ソートが完了しているということです。交換回数が０回であれば、それ以降のパスを実行しないようにします。

■ 単純交換ソート（第２版）

```
/*--- 単純交換ソート（第２版：交換回数による打ち切り） ---*/
void bubble(int a[], int n)
{
    int  i, j, m;
    int  ccnt = 0;      /* 比較回数 */
    int  scnt = 0;      /* 交換回数 */

    for (i = 0; i < n - 1; i++) {
        int  exchg = 0;                 /* パスにおける交換回数 */
        printf("パス%d：¥n", i + 1);

        for (j = n - 1; j > i; j--) {
            for (m = 0; m < n - 1; m++)
                printf("%3d %c", a[m], (m != j-1) ? ' ' :
                                    (a[j - 1] > a[j]) ? '+' : '-');
            printf("%3d¥n", a[n - 1]);

            ccnt++;
            if (a[j - 1] > a[j]) {
                swap(int, a[j - 1], a[j]);
                exchg++;
                scnt++;
            }
        }
        for (m = 0; m < n; m++)
            printf("%3d  ", a[m]);
        putchar('¥n');
        if (exchg == 0) break;          /* 交換が行われなかったら終了 */
    }
    printf("比較は%d回でした。¥n", ccnt);
    printf("交換は%d回でした。¥n", scnt);
}
```

Copyright 2008 BohYoh Shibata

■各パスにおいて、最後に交換した位置を記憶しておきます。次のパスでは、最後の交換位置までを走査することによって、交換範囲を縮めます。

■ 単純交換ソート（第３版）

```
/*--- 単純交換ソート（第３版：走査範囲を限定） ---*/
void bubble(int a[], int n)
{
    int  i = 0, j, m;
    int  ccnt = 0;      /* 比較回数 */
    int  scnt = 0;      /* 交換回数 */
    int  k = 0;                         /* a[k]より前はソート済み */

    while (k < n - 1) {
        int  j;
        int  last = n - 1;              /* 最後に交換した位置 */

        printf("パス%d：¥n", ++i);
        for (j = n - 1; j > k; j--) {
            for (m = 0; m < n - 1; m++)
                printf("%3d %c", a[m], (m != j-1) ? ' ' :
                                    (a[j - 1] > a[j]) ? '+' : '-');
            printf("%3d¥n", a[n - 1]);

            ccnt++;
            if (a[j - 1] > a[j]) {
                swap(int, a[j - 1], a[j]);
                last = j;
                scnt++;
            }
        }
        k = last;
        for (m = 0; m < n; m++)
            printf("%3d  ", a[m]);
        putchar('¥n');
    }
    printf("比較は%d回でした。¥n", ccnt);
    printf("交換は%d回でした。¥n", scnt);
}
```

Copyright 2008 BohYoh Shibata

三つのプログラムの実行結果を比較したものを以下に示します。

+　比較して交換
-　比較したが交換していない（交換の必要がない)。

```
パス1：
  1   3   4   9   7   8 + 6
  1   3   4   9   7 + 6   8
  1   3   4   9 + 6   7   8
  1   3   4 - 6   9   7   8
  1   3 - 4   6   9   7   8
  1 - 3   4   6   9   7   8
  1   3   4   6   9   7   8
パス2：
  1   3   4   6   9   7 - 8
  1   3   4   6   9 + 7   8
  1   3   4   6 - 7   9   8
  1   3   4 - 6   7   9   8
  1   3 - 4   6   7   9   8
  1   3   4   6   7   9   8
パス3：
  1   3   4   6   7   9 + 8
  1   3   4   6   7 - 8   9
  1   3   4   6 - 7   8   9
  1   3   4 - 6   7   8   9
  1   3   4   6   7   8   9
パス4：
  1   3   4   6   7   8 - 9
  1   3   4   6   7 - 8   9
  1   3   4   6 - 7   8   9
  1   3   4   6   7   8   9
パス5：
  1   3   4   6   7   8 - 9
  1   3   4   6   7 - 8   9
  1   3   4   6   7   8   9
パス6：
  1   3   4   6   7   8 - 9
  1   3   4   6   7   8   9
比較は21回でした。
交換は5回でした。
```

```
パス1：
  1   3   4   9   7   8 + 6
  1   3   4   9   7 + 6   8
  1   3   4   9 + 6   7   8
  1   3   4 - 6   9   7   8
  1   3 - 4   6   9   7   8
  1 - 3   4   6   9   7   8
  1   3   4   6   9   7   8
パス2：
  1   3   4   6   9   7 - 8
  1   3   4   6   9 + 7   8
  1   3   4   6 - 7   9   8
  1   3   4 - 6   7   9   8
  1   3 - 4   6   7   9   8
  1   3   4   6   7   9   8
パス3：
  1   3   4   6   7   9 + 8
  1   3   4   6   7 - 8   9
  1   3   4   6 - 7   8   9
  1   3   4 - 6   7   8   9
  1   3   4   6   7   8   9
パス4：
  1   3   4   6   7   8 - 9
  1   3   4   6   7 - 8   9
  1   3   4   6 - 7   8   9
  1   3   4   6   7   8   9
比較は18回でした。
交換は5回でした。
```

```
パス1：
  1   3   4   9   7   8 + 6
  1   3   4   9   7 + 6   8
  1   3   4   9 + 6   7   8
  1   3   4 - 6   9   7   8
  1   3 - 4   6   9   7   8
  1 - 3   4   6   9   7   8
  1   3   4   6   9   7   8
パス2：
  1   3   4   6   9   7 - 8
  1   3   4   6   9 + 7   8
  1   3   4   6   7   9   8
パス3：
  1   3   4   6   7   9 + 8
  1   3   4   6   7   8   9
比較は9回でした。
交換は5回でした。
```

Copyright 2008 BohYoh Shibata

前ページの実行例では、第１版→第２版→第３版と比較回数が少なくなり、改良に伴って計算効率が向上しています。

別の実行例を示します。

```
パス1：
  7   1   2   3   4   5 - 6
  7   1   2   3   4 - 5   6
  7   1   2   3 - 4   5   6
  7   1   2 - 3   4   5   6
  7   1 - 2   3   4   5   6
  7 + 1   2   3   4   5   6
  1   7   2   3   4   5   6
パス2：
  1   7   2   3   4   5 - 6
  1   7   2   3   4 - 5   6
  1   7   2   3 - 4   5   6
  1   7   2 - 3   4   5   6
  1   7 + 2   3   4   5   6
  1   2   7   3   4   5   6
パス3：
  1   2   7   3   4   5 - 6
  1   2   7   3   4 - 5   6
  1   2   7   3 - 4   5   6
  1   2   7 + 3   4   5   6
  1   2   3   7   4   5   6
パス4：
  1   2   3   7   4   5 - 6
  1   2   3   7   4 - 5   6
  1   2   3   7 + 4   5   6
  1   2   3   4   7   5   6
パス5：
  1   2   3   4   7   5 - 6
  1   2   3   4   7 + 5   6
  1   2   3   4   5   7   6
パス6：
  1   2   3   4   5   7 + 6
  1   2   3   4   5   6   7
比較は21回でした。
交換は6回でした。
```

```
パス1：
  7   1   2   3   4   5 - 6
  7   1   2   3   4 - 5   6
  7   1   2   3 - 4   5   6
  7   1   2 - 3   4   5   6
  7   1 - 2   3   4   5   6
  7 + 1   2   3   4   5   6
  1   7   2   3   4   5   6
パス2：
  1   7   2   3   4   5 - 6
  1   7   2   3   4 - 5   6
  1   7   2   3 - 4   5   6
  1   7   2 - 3   4   5   6
  1   7 + 2   3   4   5   6
  1   2   7   3   4   5   6
パス3：
  1   2   7   3   4   5 - 6
  1   2   7   3   4 - 5   6
  1   2   7   3 - 4   5   6
  1   2   7 + 3   4   5   6
  1   2   3   7   4   5   6
パス4：
  1   2   3   7   4   5 - 6
  1   2   3   7   4 - 5   6
  1   2   3   7 + 4   5   6
  1   2   3   4   7   5   6
パス5：
  1   2   3   4   7   5 - 6
  1   2   3   4   7 + 5   6
  1   2   3   4   5   7   6
パス6：
  1   2   3   4   5   7 + 6
  1   2   3   4   5   6   7
比較は21回でした。
交換は6回でした。
```

```
パス1：
  7   1   2   3   4   5 - 6
  7   1   2   3   4 - 5   6
  7   1   2   3 - 4   5   6
  7   1   2 - 3   4   5   6
  7   1 - 2   3   4   5   6
  7 + 1   2   3   4   5   6
  1   7   2   3   4   5   6
パス2：
  1   7   2   3   4   5 - 6
  1   7   2   3   4 - 5   6
  1   7   2   3 - 4   5   6
  1   7   2 - 3   4   5   6
  1   7 + 2   3   4   5   6
  1   2   7   3   4   5   6
パス3：
  1   2   7   3   4   5 - 6
  1   2   7   3   4 - 5   6
  1   2   7   3 - 4   5   6
  1   2   7 + 3   4   5   6
  1   2   3   7   4   5   6
パス4：
  1   2   3   7   4   5 - 6
  1   2   3   7   4 - 5   6
  1   2   3   7 + 4   5   6
  1   2   3   4   7   5   6
パス5：
  1   2   3   4   7   5 - 6
  1   2   3   4   7 + 5   6
  1   2   3   4   5   7   6
パス6：
  1   2   3   4   5   7 + 6
  1   2   3   4   5   6   7
比較は21回でした。
交換は6回でした。
```

第１版・第２版・第３版で、まったく同じ結果が得られます。

Copyright 2008 BohYoh Shibata

■ シェーカーソート

前ページの実行例は、データが以下のように並んでいます。

7 1 2 3 4 5 6

ほぼソート済みであるにも関わらず、最大の要素 7 が先頭に位置しているために、第３版のアルゴリズムでも、ソート作業を早期に打ち切ることはできません。というのも、最大の要素は、各パスにおいて、一つずつしか後ろに移動しないからです。

そこで、奇数パスでは最小要素を先頭側に移動させ、偶数パスでは最大要素を末尾側に移動するというように、パスを交互に行えば、このような並びを高速にソートできるはずです。バブルソートを改良したこのアルゴリズムは、双方向バブルソート（bidirection bubble sort）あるいはシェーカーソート（shaker sort）という名称で知られています。

シェーカーソートのプログラムを示します。

■ シェーカーソート

```
void shakersort(int a[],int n)
{
    int  left = 0;
    int  right = n - 1;
    int  last = right;

    while (left < right){
        int  j;
        for (j = right; j > left; j--){
            if (a[j - 1] > a[j]){
                swap(int, a[j - 1], a[j]);
                last = j;
            }
        }
        left = last;

        for (j = left; j < right; j++){
            if (a[j] > a[j + 1]){
                swap(int, a[j], a[j + 1]);
                last = j;
            }
        }
        right = last;
    }
}
```

Copyright 2008 BohYoh Shibata

バブルソートとシェーカーソートの実行例を比較します。

左側がバブルソート（第１版・第２版・第３版）の実行例で、右側がシェーカーソートの実行例です。

```
パス1：
  7   1   2   3   4   5 - 6
  7   1   2   3   4 - 5   6
  7   1   2   3 - 4   5   6
  7   1   2 - 3   4   5   6
  7   1 - 2   3   4   5   6
  7 + 1   2   3   4   5   6
  1   7   2   3   4   5   6
パス2：
  1   7   2   3   4   5 - 6
  1   7   2   3   4 - 5   6
  1   7   2   3 - 4   5   6
  1   7   2 - 3   4   5   6
  1   7 + 2   3   4   5   6
  1   2   7   3   4   5   6
パス3：
  1   2   7   3   4   5 - 6
  1   2   7   3   4 - 5   6
  1   2   7   3 - 4   5   6
  1   2   7 + 3   4   5   6
  1   2   3   7   4   5   6
パス4：
  1   2   3   7   4   5 - 6
  1   2   3   7   4 - 5   6
  1   2   3   7 + 4   5   6
  1   2   3   4   7   5   6
パス5：
  1   2   3   4   7   5 - 6
  1   2   3   4   7 + 5   6
  1   2   3   4   5   7   6
パス6：
  1   2   3   4   5   7 + 6
  1   2   3   4   5   6   7
比較は21回でした。
交換は6回でした。
```

```
パス1：
  7   1   2   3   4   5 - 6
  7   1   2   3   4 - 5   6
  7   1   2   3 - 4   5   6
  7   1   2 - 3   4   5   6
  7   1 - 2   3   4   5   6
  7 + 1   2   3   4   5   6
パス2：
  1   7 + 2   3   4   5   6
  1   2   7 + 3   4   5   6
  1   2   3   7 + 4   5   6
  1   2   3   4   7 + 5   6
  1   2   3   4   5   7 + 6
パス3：
  1   2   3   4   5 - 6   7
  1   2   3   4 - 5   6   7
  1   2   3 - 4   5   6   7
  1   2 - 3   4   5   6   7
比較は15回でした。
交換は6回でした。
```

Copyright 2008 BohYoh Shibata

■ シェーカーソート（交換の過程を表示）

```
void shakersort(int a[],int n)
{
    int  i = 0, m;
    int  left = 0;
    int  right = n - 1;
    int  last = right;
    int  ccnt = 0;      /* 比較回数 */
    int  scnt = 0;      /* 交換回数 */

    while (left < right) {
        int  j;
        printf("パス%d：¥n", ++i);
        for (j = right; j > left; j--){
            for (m = 0; m < n - 1; m++)
                printf("%3d %c" , a[m], (m != j-1) ? ' ' :
                                    (a[j - 1] > a[j]) ? '+' : '-');
            printf("%3d¥n", a[n - 1]);
            ccnt++;
            if (a[j - 1] > a[j]){
                swap(int, a[j - 1], a[j]);
                last = j;
                scnt++;
            }
        }
        left = last;

        if (left < right) printf("パス%d：¥n", ++i);
        for (j = left; j < right; j++){
            for (m = 0; m < n - 1; m++)
                printf("%3d %c" , a[m], (m != j) ? ' ' :
                                    (a[j] > a[j + 1]) ? '+' : '-');
            printf("%3d¥n", a[n - 1]);
            ccnt++;
            if (a[j] > a[j + 1]){
                swap(int, a[j], a[j + 1]);
                last = j;
                scnt++;
            }
        }
        right = last;
    }
    printf("比較は%d回でした。¥n", ccnt);
    printf("交換は%d回でした。¥n", scnt);
}
```

Copyright 2008 BohYoh Shibata

配列を使いこなす

Ｃ言語は配列をコピーするライブラリすら提供しません。配列を扱う便利なライブラリを作っていきましょう。

まずは、配列要素の削除です。配列中の不要な要素の削除は、それ以降の要素を前方にシフトする（ずらす）ことによって実現できます。

以下の二つの関数を作りましょう。aryrmv は 1 個の要素を削除するライブラリで、aryrmvs は連続する複数個の要素を削除するライブラリです。

	aryrmv
形　式	void aryrmv(int a[], int n, int idx)
解　説	要素型が int で要素数 n の配列 a から、要素 a[idx]を削除する。 すなわち、a[idx + 1], a[idx + 2], …, a[n - 1]の要素を a[idx], a[idx + 1], …, a[n - 2]へコピーする。末尾要素 a[n - 1]の値は変化しない。

	aryrmvs
形　式	void aryrmvs(int a[], int n, int idx1, int idx2)
解　説	要素型が int で要素数 n の配列 a から、要素 a[idx1]から a[idx2]までの要素を削除する。すなわち、a[idx2 + 1], a[idx2 + 2], …, a[n - 1]の要素を a[idx1], a[idx1 + 1], …, a[idx1 + n - idx2 - 1]へコピーする。a[idx1 + n - idx2]以降の要素の値は変化しない。

『実験データの中にノイズが混入したデータが入っていた。それを削除したい。』『ゲームのキャラクタが配列に入っている。撃沈したキャラクタを配列から除きたい。』…など、配列から要素を削除するといったことは、あらゆる分野で要求されるものです。

配列要素の削除や挿入は、入門書で扱うには少し難しいためか、扱っているテキストは稀なようです。

その一方で、「アルゴリズムとデータ構造」といった類のテキストで取り扱うには、簡単すぎるものです。

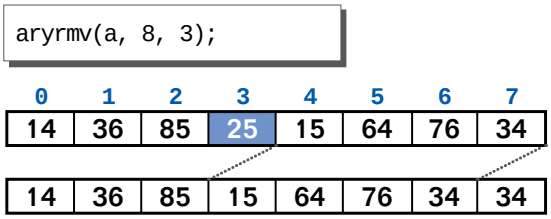
○○ソートとか△△サーチといった名称を与えるほどでもない《名もなきアルゴリズム》であり、入門書と、その次のステップのテキストとのスキマに存在します。

とはいっても、現実のプログラム開発では、挿入や削除・その応用は必須です。

Copyright 2008 BohYoh Shibata

これらの関数の動作イメージを下図に示します。シフトの対象とならない末尾側の要素の値は変化しません。なお、処理を行うのは、aryrmv では、0≦idx＜n のときのみで、aryrmvs では、0≦idx1≦idx2＜n のときのみです。

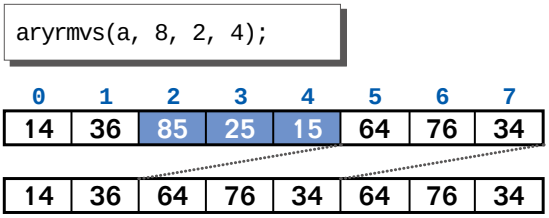
□aryrmv … 要素を 1 個だけ削除



要素数8の配列aからa[3]を削除。

```
/*--- 要素数nの配列aからa[idx]を削除（以降の要素を前方へずらす）---*/
void aryrmv(int a[], int n, int idx)
{
    if (idx >= 0 && idx < n) {
        int i;
        for (i = idx; i < n - 1; i++)
            a[i] = a[i + 1];
    }
}
```

□aryrmvs … 連続する要素を削除



要素数8の配列aからa[2]～a[4]を削除。

```
/*--- 要素数nの配列aからa[idx1]～a[idx2]を削除（以降の要素を前方へずらす）---*/
void aryrmvs(int a[], int n, int idx1, int idx2)
{
    if (idx1 >= 0 && idx2 >= idx1 && idx2 < n) {
        int i;
        for (i = idx1; i < idx1 + n - idx2 - 1; i++)
            a[i] = a[i + idx2 - idx1 + 1];
    }
}
```

Copyright 2008 BohYoh Shibata

番外編 … キャラクタ移動ゲームを作ろう

キャラクタ移動プログラム（第１版～第４版）を作りました。を作成しました。第４版のプログラムを示します。

■ キャラクタ移動ゲーム

```
/*
    キャラクタ移動プログラム（その４）
*/

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

int xp = 1;          /* キャラクタのX座標 */
int yp = 1;          /* キャラクタのY座標 */
int xx = 1;          /* キャラクタのX座標（移動前の座標） */
int yy = 1;          /* キャラクタのY座標（移動前の座標） */
int counter = 0;     /* 移動回数 */
int wcounter = 0;    /* 壁にぶつかった回数 */
int wx;              /* 壁のX座標 */
int wy;              /* 壁のY座標 */


/*--- キャラクタ表示 ---*/
void display_character()
{
    locate(xp, yp);
    color(LIGHT_YELLOW);
    printf("■");
}

/*--- キャラクタ消去 ---*/
void erase_character()
{
    locate(xx, yy);
    color(LIGHT_YELLOW);
    printf("  ");
}

/*--- 情報表示 ---*/
void display_information()
{
```

Copyright 2008 BohYoh Shibata

```
    color(WHITE);
    locate(1, 22);
    printf("(%02d,%02d) COUNTER:%8d WCOUNTER:%8d", xp, yp, counter, wcounter);
}

int main(void)
{
    srand(time(NULL));
    cls();
    display_character();
    display_information();
    wx = 1 + (2 * rand() % 40); /* 壁のX座標 */
    wy = 1 + (rand() % 20);     /* 壁のY座標 */
    locate(wx, wy);
    color(RED);
    printf("■");

    while (1) {
        int ch = getch();
        switch (ch) {
         case '6' : if (xp < 79) xp += 2; break;
         case '4' : if (xp >  1) xp -= 2; break;
         case '2' : if (yp < 20) yp++;    break;
         case '8' : if (yp >  1) yp--;    break;
        }
        if (xx != xp || yy != yp) {
            if (xp == wx && yp == wy) { /* 壁とぶつかった */
                putchar('¥a');
                wcounter++;
                xp = xx;
                yp = yy;
            } else {
                erase_character();
                display_character();
                counter++;
            }
            display_information();
        }
        xx = xp;
        yy = yp;
    }
    return (0);
}
```

Copyright 2008 BohYoh Shibata