基本的なデータ構造とその表現1
（配列とポインター）

No.2　2019年4月16,18日

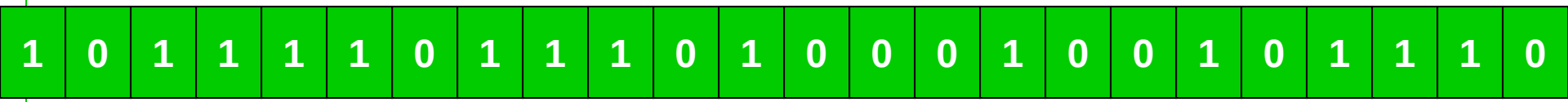
高橋信行

目的

- 基本データ型
  - C言語で予め準備されている情報の表現
- 型
  - 整数(int)，実数(float, double)，文字(char)
- データ構造：型を組み合わせて情報を表現
  - 配列，構造体
- データの操作に慣れる

情報(データ)の計算機中の表現

- 計算機中のデータ
  - メモリ，ディスクの内容:「1」,「0」のビット列
  - 各種情報:ビット列の解釈の仕方が異なる



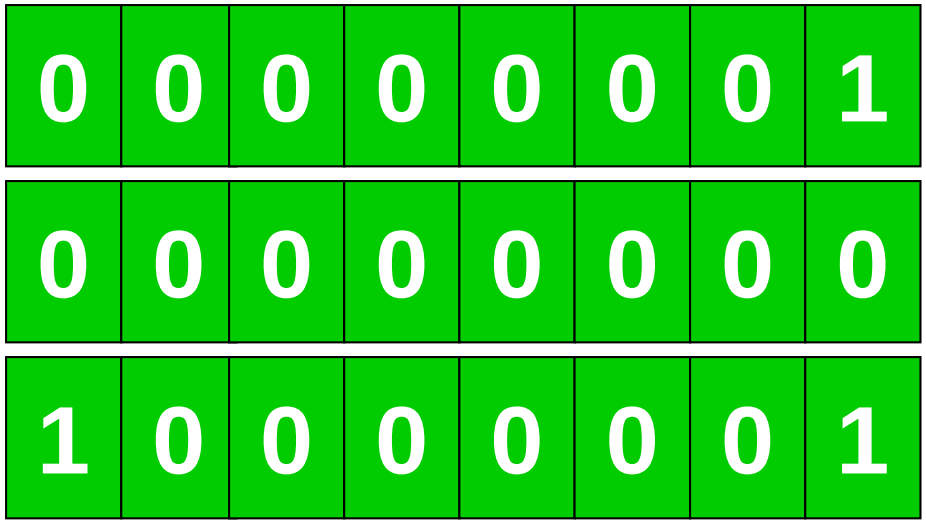
- 数，文字などの基本データは共通
  - 長さ(有限桁)と解釈方法が決まっている
  - 多くは，バイト単位の長さ

整数

- 1B，2B，4Bなどで表現範囲が異なる
  - 1B: 0～255
- 符号なし整数
  - unsigned (short int, int , long)
  - 各位が2のべき数：10進数と同様

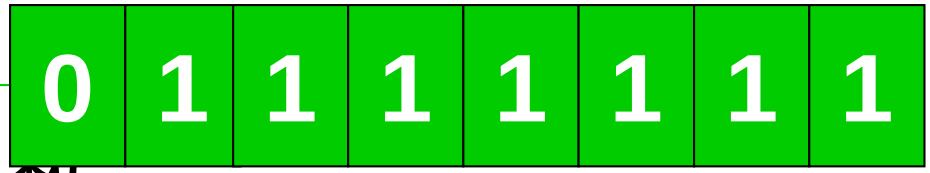
整数

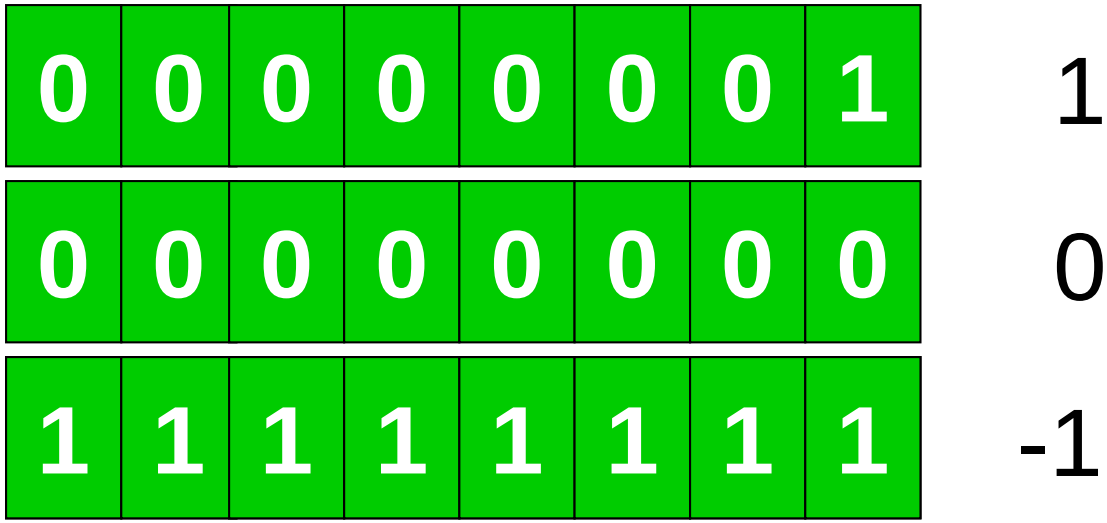
- 符号付き整数
  - short int, int , long
  - 2の補数表現（1B:-128から127）



整数

- 符号付き整数
  - short int, i
  - 2の補数表現（1B:-128から127）





実数(一定精度内の近似値)

- 浮動小数点型
  - ビット列を仮数部と指数部と解釈
- 仮数部:数の精度
- 指数部:数の位

1.0×10^{5}　　1.1011×2^{10111}

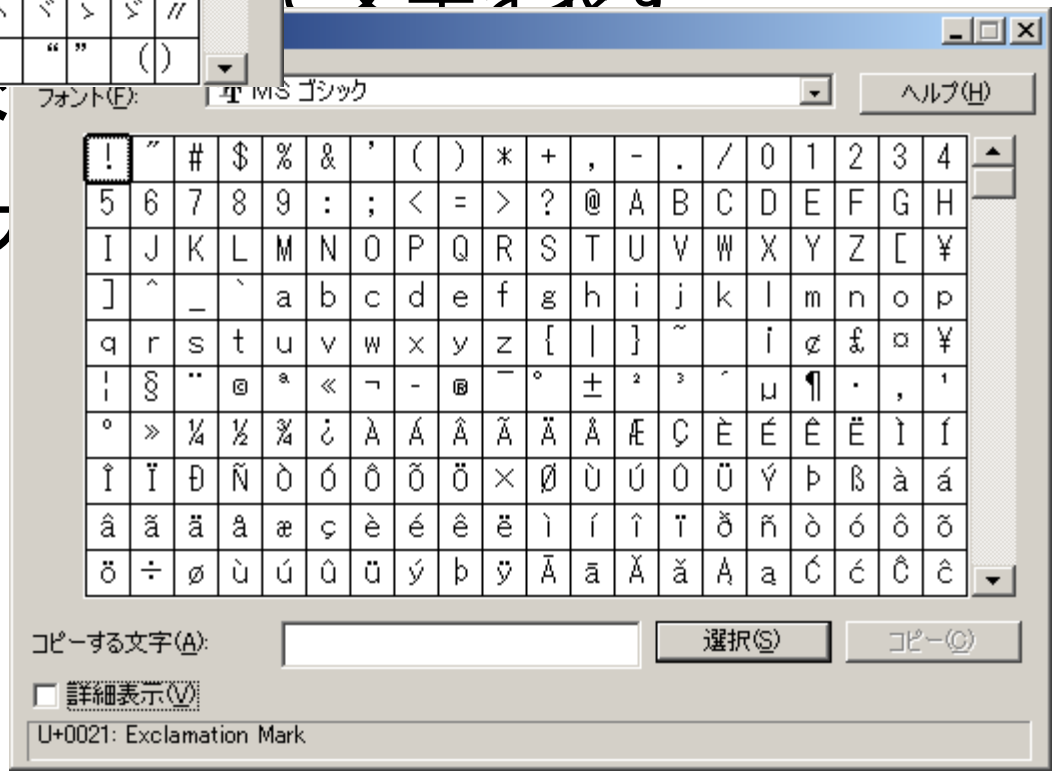
文字

- 文字の一覧表（文字コード）
  - 表の位置（符号なし整数）で文字を表す
- 複数の文字コード
  - 文字種類で表の大きさが変化
  - 英文字：1B
  - 日本語：2B

ド)

で文字を表す

- 複数の文字コード
  - 文字種類で表の
  - 英文字：1B
  - 日本語：2B



日本語コードにおける問題点

- 英文字（1B）が通信の前提
  - 表示以外の目的の文字コードがある
  - アラーム(¥a)を鳴らす，改行(¥n)，タブ(¥t)
- 日本語コード(2B)
  - 単純な表を用いれば，英文字2文字と認識されて，問題が生じる.
  - 文字の誤認識問題がある.

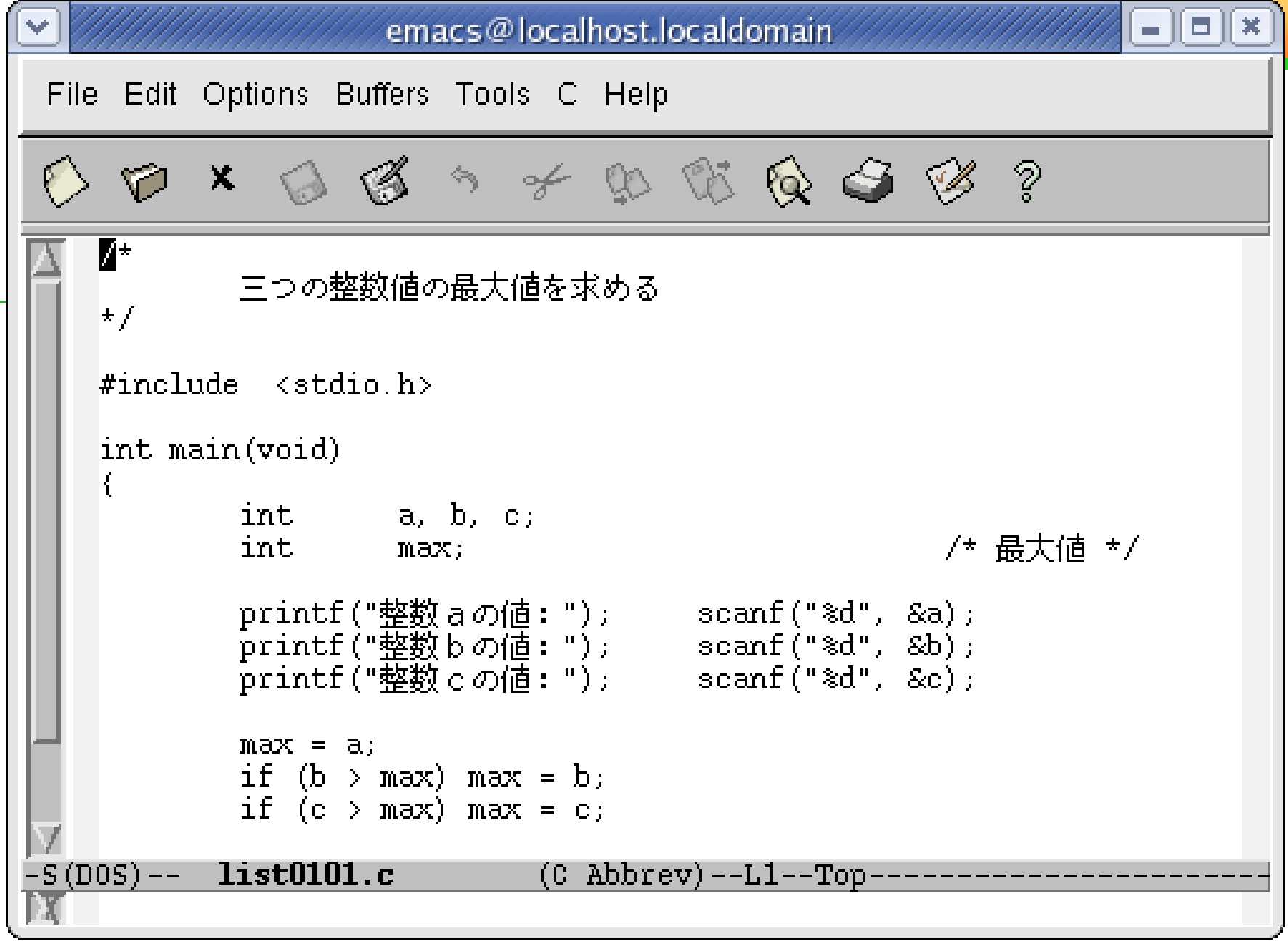
日本語の文字コード

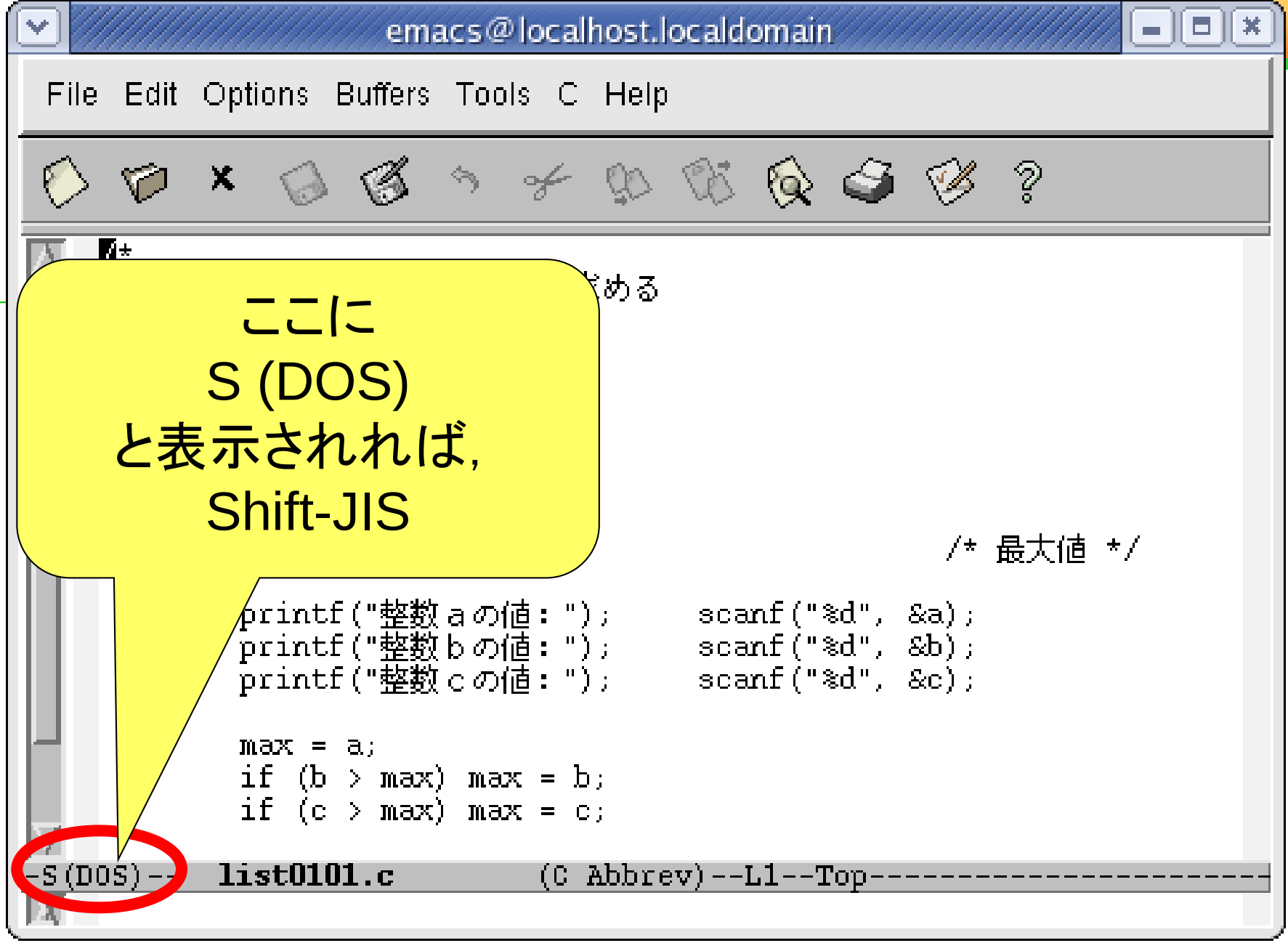
- 複数の文字コードがある
  - 昔は，コンピュータを作った企業ごとにコードがあった
- MS-Windows
  - Shift-JIS（MS漢字コード）
- Linux（多言語対応）
  - 日本語EUC (Extended UNIX Code)
  - 他に中国語EUC，韓国語EUCなどがある．
- UTF-8 (UTF-2, UTF-FSS)
  - 誤認識問題を解決，最近普及してきた．

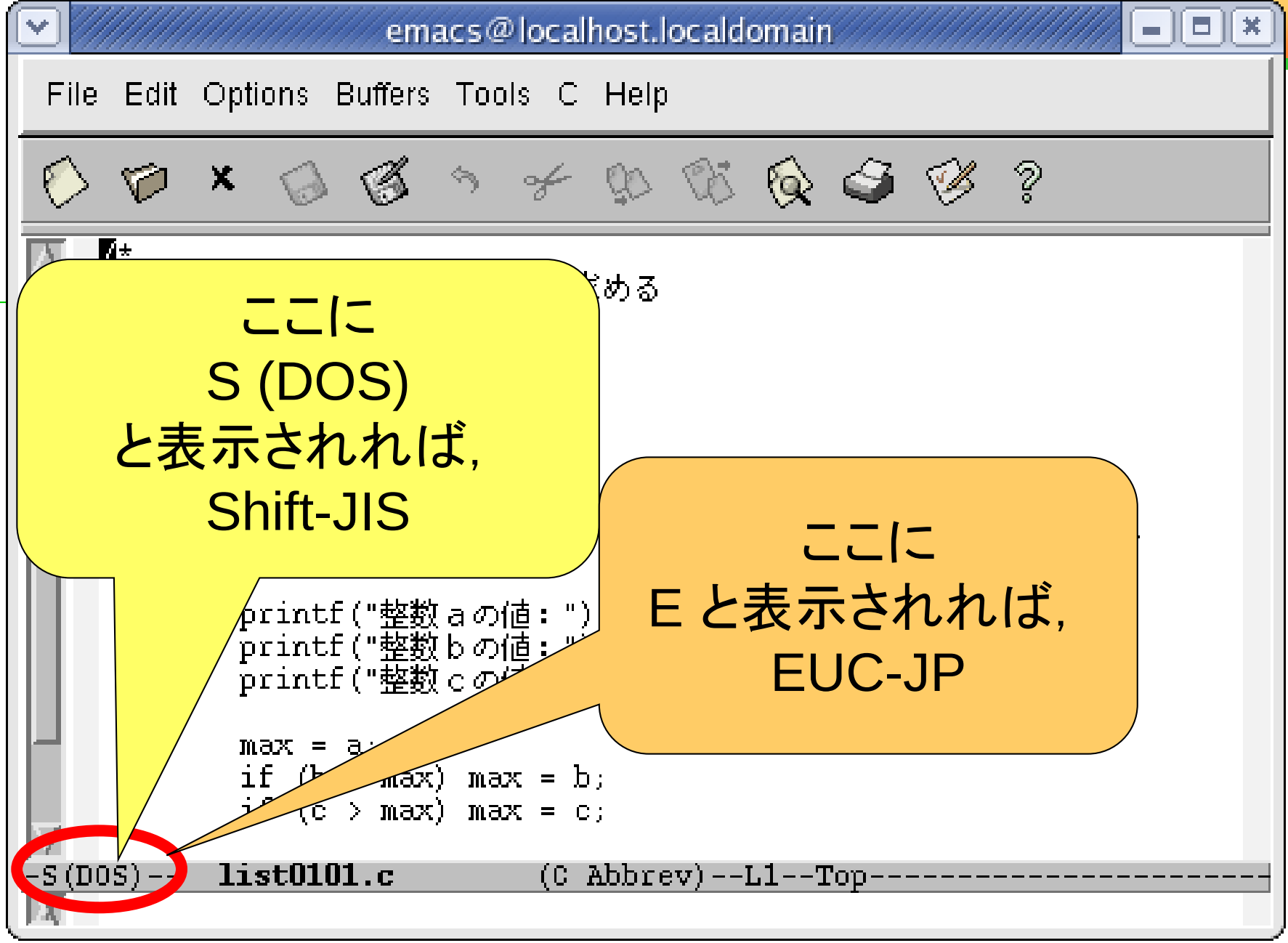
Gnomeの設定変更

- CentOSの標準日本語コード
  - UTF-8
- Gnome端末文字コードの変更
  - 端末→文字コード設定→日本語(EUC-JP)
- 環境変数の変更
  - 以下のコマンドを入力

```
export LC_CTYPE=ja_JP.EUC-JP
```

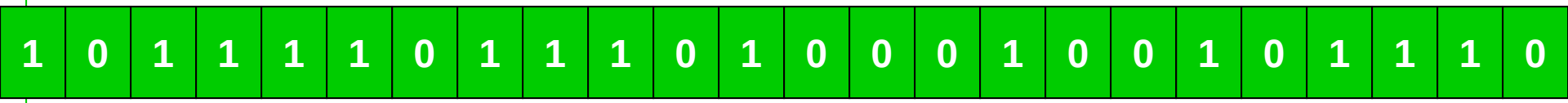

emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
/*
三つの整数値の最大値を求める
*/
#include <stdio.h>
int main(void)
{
int a, b, c;
int max; /* 最大値 */
printf("整数ａの値："); scanf("%d", &a);
printf("整数ｂの値："); scanf("%d", &b);
printf("整数ｃの値："); scanf("%d", &c);
max = a;
if (b > max) max = b;
if (c > max) max = c;
-S(DOS)-- list0101.c (C Abbrev)--L1--Top


emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
/*
/* 最大値 */
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;
if (b > max) max = b;
if (c > max) max = c;
-S(DOS)-- list0101.c (C Abbrev)--L1--Top
ここに
S (DOS)
と表示されれば，
Shift-JIS


emacs@localhost.localdomain
File Edit Options Buffers Tools C Help
printf("整数ａの値：")
printf("整数ｂの値：")
max = a;
if (b > max) max = b;
if (c > max) max = c;
-S(DOS)-- list0101.c (C Abbrev)--L1--Top----------
ここに
S (DOS)
と表示されれば，
Shift-JIS
ここに
E と表示されれば，
EUC-JP

メモリとC言語の一般変数

- 参照するメモリの位置が固定：宣言で決定
  - 計算機のメモリの位置をアドレス（番地）で指定
  - 指定メモリのビット列を解釈
- 解釈の方法が固定：型宣言で決定
  - メモリ中のビット列を型で指定された長さ，解釈方法で解釈



メモリとC言語の一般変数

- 参照するメモリの位置が固定：宣言で決定
  - 計算機のメモリの位置をアドレス（番地）で指定
  - 指定メモリのビット列を解釈
- 解釈の方法が固定：型宣言で決定
  - 中のビット列を型で指定された長さ，解釈方釈

変数の代入

- ＝　が代入を指示する命令

  変数名 ＝　式 ；

- ＝の右辺のみを考える
  1. ＝の左辺は無視，右辺の計算
  2. 計算結果を右辺の変数に代入
  - 結果に応じた内容を変数の値にする
  - 値を変数の型に応じたビット列としてメモリに書く

変

- ＝ が代入

  変

- ＝の右辺のみを考える
  1. ＝の左辺は無視，右辺の計算
  2. 計算結果を右辺の変数に代入
  - 結果に応じた内容を変数の値にする
  - 値を変数の型に応じたビット列としてメモリに書く

```
i = 1;
i = i + 1;
```

ポインター（教科書50ページ）

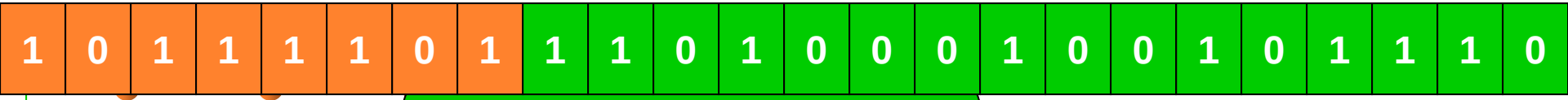
- C言語でアドレスを取り扱うための型
  - 実質的には，メモリ上の位置（アドレス）
- C言語での利便性のため
  - 位置（アドレス）に長さ，解釈方法の情報が付加
- 宣言
  - 型　＊
- ポインター情報の取り出し
  - &

ポインターの例

- ポインター変数
  - 参照するビット列のアドレスが変更できる
  - 参照するビット列の参照方法は固定

```
int    x ;     /*一般の整数変数*/
int   *y ;     /*整数のポインター変数*/
y = &x;
```

ポインターの例



変数 *y

&：変数からアドレスを取り出す

- ポインタ
  - 参照す 変更できる
  - は固定

```
int   x ;    /*  般の整数変数*/
int  *y ;    /*整数
y = &x;
```

*yとxの実態は同じになる

ポインターの演算

- 演算によりポインターの値が変化
  - ポインター（アドレス）には大小関係あり
  - 実際のメモリ上の移動距離は型に依存
  - アドレスの位置に関する計算が可能
- ＋＋
  - 現在のアドレスに1を加算（次の変数の内容を指す）
- ーー
  - 現在のアドレスから1を減算（前の変数の内容を指す）
- 四則演算
  - 計算結果に応じてアドレスの位置が変化

ポインターの演算

- 演算によりポインタ
  - ポインター（アドレ
  - 実際のメモリ上の
  - アドレスの位置に
- ＋＋
  - 現在のアドレスに
- ーー
  - 現在のアドレスから1を減算（前の変数の内容を指す）
- 四則演算
  - 計算結果に応じてアドレスの位置が変化

```
int *x, y ;
 x = &y + 1;
 x--;
```

変数のサイズ（教科書44ページ）

- 型によって必要となるメモリの大きさが異なる
- メモリの大きさを調べることが可能

sizeof(型名)，sizeof(変数名)

配列（教科書42ページ）

- 同種のデータの集まりを表現
- 集まり方で使い分ける
  - 変数（0次元）：スカラー
  - 1次元：ベクトル，時系列など
  - 2次元：行列，画像データ，表など
  - 3次元：テンソル，動画像，3次元物体など

１次元配列変数とポインター（教科書50，51ページ）

- 配列：配列名＋添え字
- 配列名
  - 配列の最初の要素のアドレス
  - 静的ポインター変数
  - 内容が変更できないポインター変数
- 配列の要素
  - 添え字：最初の要素からの相対位置で示す
  - 配列名[0]:　1番目（最初）の要素
  - 配列名[n]:　n+1番目の要素

１次元配列変数とポインター（教科書50、51ページ）

- 配列：配列
- 配列名
  - 配列の最初の
  - 静的ポインター
  - 内容が変更で
- 配列の要素
  - 添え字：最初の要素からの
  - 配列名[0]：　1番目（最初）の要素
  - 配列名[n]：　n+1番目の要素

＊（配列名）
＊（配列名+0）

＊（配列名+n）

最大値の検索

- 多数のデータ中からの最大値を探す
- 一般変数でプログラムを作成
  - データ数だけの変数が必要
  - IF文による判断が多数必要
  - プログラムが長くなり，理解に時間がかかる
- 配列によるプログラム作成
  - 繰り返し文が利用可能
  - データの長さに依存しないプログラム

```
#include <stdio.h>
int main(void){
int     a,b,c,d,e,f;
int    max;
        scanf("%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f);
        max = a;
        if (max < b)  max = b;
        if (max < c)  max = c;
        if (max < d)  max = d;
        if (max < e)  max = e;
        if (max < f)   max = f;
        printf("最大値は %d¥n", max);
}
```

- 配列によるプログラム作成
  - 繰り返し文が利用可能
  - データの長さに依存しないプログラム

```
#include <stdio.h>
int main(void){
int     a,b,c,d,e,f;
int    max;
       scanf("%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f);
       ma
       if (
       if (
       if (
       if (
       if (
       prin
}
```

```
#include <stdio.h>
int main(void)
{
int     a[6]={1,2,3,4,5,6};
int     i, max;

        max = a[0];
        for(i=1; i<6; i++){
           if (max < a[i])    max = a[i];
        }
        printf("最大値は %d¥n", max);
}
```

- 配

```
#include <stdio.h>
int main(void){
int
int
}
```

```
#include <stdio.h>
int main(voi
{
int    a[6]={1,2,3,4,5,6};
int    i, *max;

       max = a;
       for(i=1; i<6; i++){
          if (*max < *(a+i))    max = a + i ;
       }
       printf("最大値は %d¥n", *max);
}
```

a　　a+4

```
            }
            printf("最大値は %d¥n", max);
        }
```

2次元配列（教科書74ページ）

- 情報の内容が2つの指標に依存
  - 画像（幅，高さ）
  - 表，行列（行，列）
- C言語による宣言

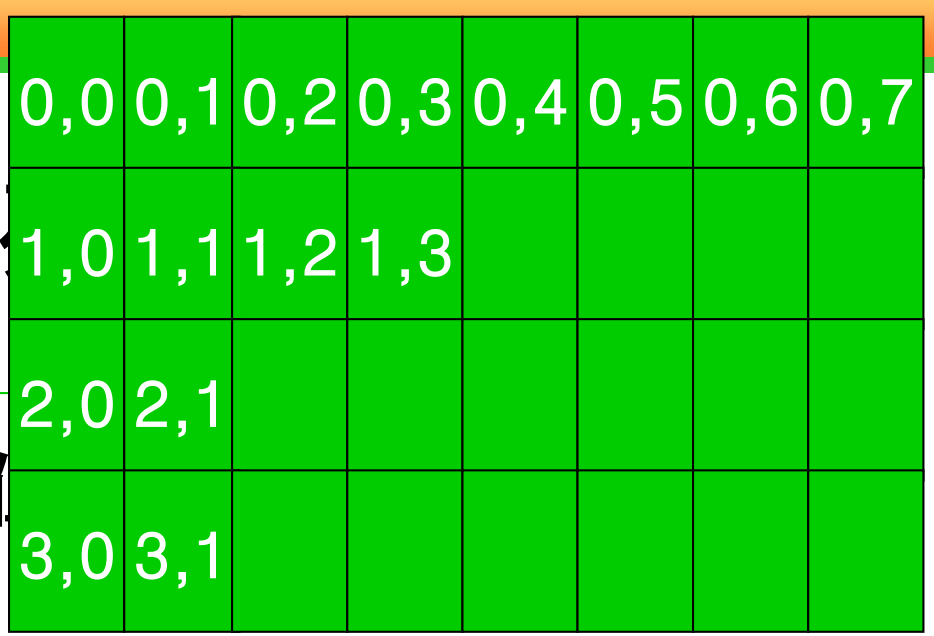
  int　a[3][6];
  - 型は表現する情報に依存
  - 画像：点の明るさ

多次元配列の実現

- 計算機中のメモリの位置
  - アドレスで指定
  - アドレスは正整数と思ってよい.
  - 指標(アドレス)は1つ
- 2次元指標⇒1次元指標への変換
  - 例:a[y][x], 0≦x≦7, 0≦y≦7

    address = 8*y + x

多次元配

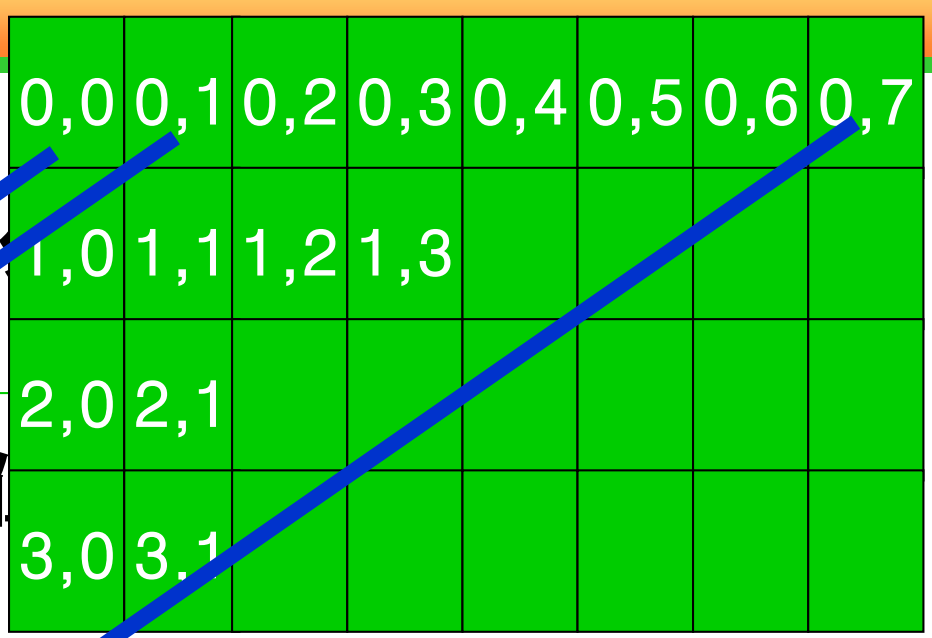


- 計算機中のメモリの
  - アドレスで指定
  - アドレスは正整数と思ってよい

- 2次元指標⇒1次元指標への変換
  - 例：a[y][x], 0≦x≦7, 0≦y≦7

$$address = 8*y + x$$

多次元配

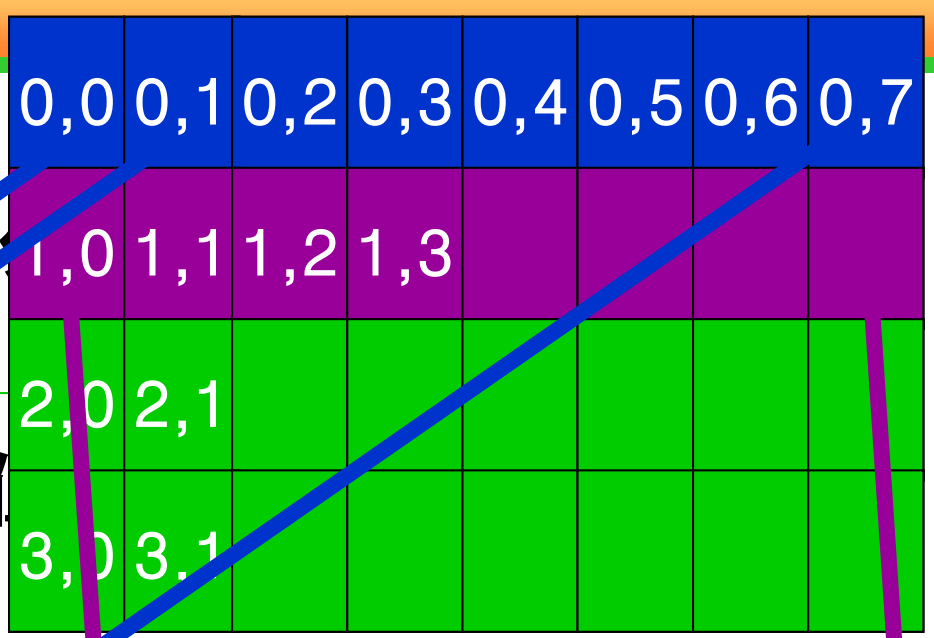


- 計算機中のメモリの
  - アドレスで指定
  - アドレスは正整数と思ってよい

- 2次元指標⇒1次元指標への変換
  - 例：a[y][x], 0≦x≦7, 0≦y≦7

$$address = 8*y + x$$

多次元配



- 計算機中のメモリの
  - アドレスで指定
  - アドレスは正整数と思ってよい

- 2次元指標⇒1次元指標への変換
  - 例：a[y][x], 0≦x≦7, 0≦y≦7

$$address = 8*y + x$$

優先的に変化する添え字

- 添え字の変化とアドレスの変化の関係
  - 後(右)側の変化⇒アドレスの変化に優先
- a[添え字n][添え字n-1]… [添え字1]
  - 添え字1が優先的
- 多次元を1次元に変換
  - 添え字を位のように考える

$$678=6\times100+7\times10+8\times1$$

2(多)次元配列の宣言

- 型　配列名[行大きさ][列大きさ];
  - 要素数:　行大きさ×列大きさ
- 参照:利用
  - 配列名[0][0]～配列名[行大きさ-1][列大きさ-1]
- 2次元以上のn次元配列
  - 型 配列名[大きさn]...[大きさ2][大きさ1];

```
int a[10][10];
int b[2][3];

if (a[i][j] > b[i][j]) b[i][j] = a[i][j] ;
```

- 2次元以上のn次元配列
  - 型 配列名[大きさn]...[大きさ2][大きさ1];

配列の初期化

- 予め配列の要素の値を指定する.
- 配列を表などのように利用する場合
  - 宣言と同時に指定する
- 例(2次元)

型 配列名[n][m] ={初期値の並び(n×m個の値をコンマで区切る)};

int a[2][3] = {1,2,3,4,5,6};
列変化優先で配列に値が記憶
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[1][0] = 4;
a[1][1] = 5;
a[1][2] = 6;

- 予め
- 配列
  - 宣
- 例(2

型 配列名[][] = {初期値の並び(n×m個の値をコンマで区切る)};

int a[2][3] = {1,2,3,4,5,6};
列変化優先で配列に値が記憶
a[0][0] =
a[0][1
a[0
3;

- 予め
- 配列

多次元では，{}を用いて初期化を分りやすくできる.

```
int a[2][3] = {
    {1,2,3},
    {4,5,6}
};
```

初期化で利用できる省略

- 配列の要素数指定の省略
  - 初期値の個数から自動判断
- 例(最も左の要素数が省略可能)
  - int a[] = {1,2,3,4,5,6};
  - int b[][3] = {{1,2,3},{4,5,6}};

初期化で利用できる省略

- 初期値を指定しない要素の初期値は0
  - どの要素の初期値も与えないと配列の初期値は不定
- 例
  - int b[2][3] = {{1,2},{4}};
  - int b[2][3] = {{1,2,0},{4,0,0}};

データの並びの逆転

- アルゴリズム：考え方
  - データを配列で表現

1. 用意した別配列へ逆順で代入，元の配列へ戻す．
   - データ数の2倍の代入とメモリ
2. 配列の両側の要素を交換
   - データ数の3/2倍の代入とデータ数＋1のメモリ
   - 教科書58～59ページ

配列要素の並びを逆転する

```
int main(void)
{
    int     i;
    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]);
    }

    ary_reverse(x, nx);/* 配列xの要素の並びを逆転 */

    printf("配列の要素の並びを逆転しました。¥n");
    for (i = 0; i < nx; i++)
        printf("x[%d] : %d¥n", i, x[i]);
    return (0);
}
```

逆転する

プログラムは関数mainから実行

配列xにデータを記憶

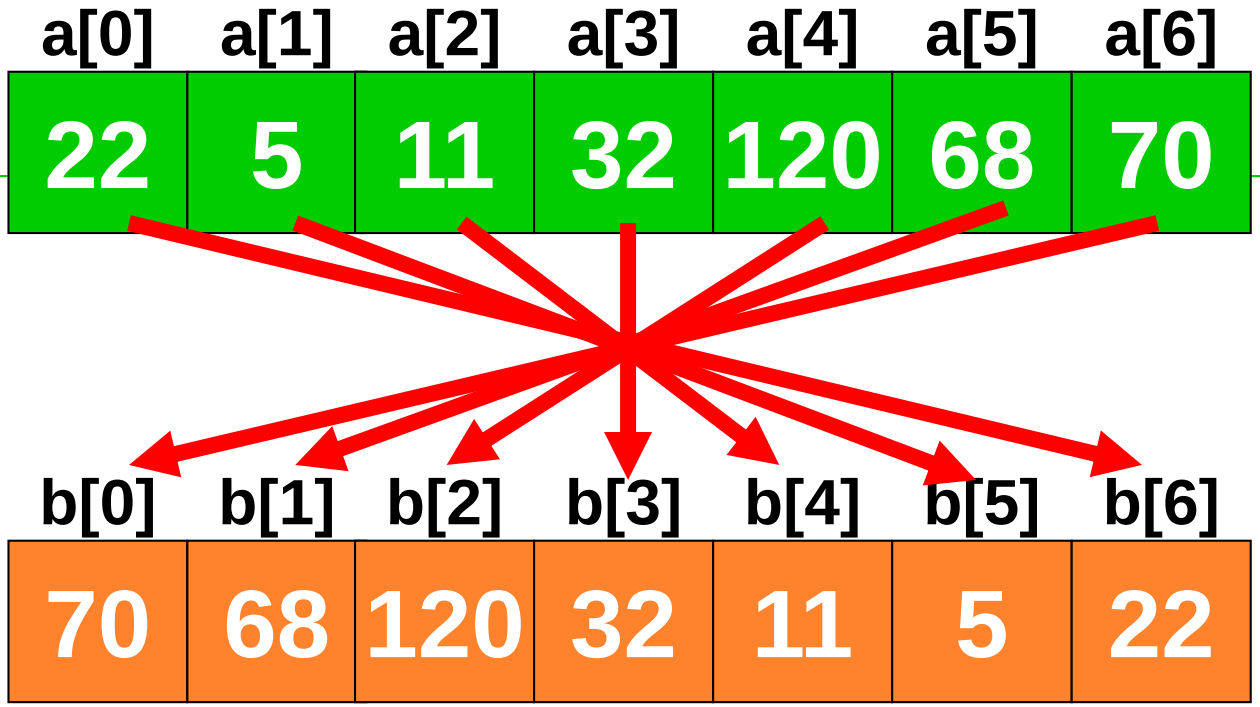
関数ary_reverseにデータの並びの逆転を依頼

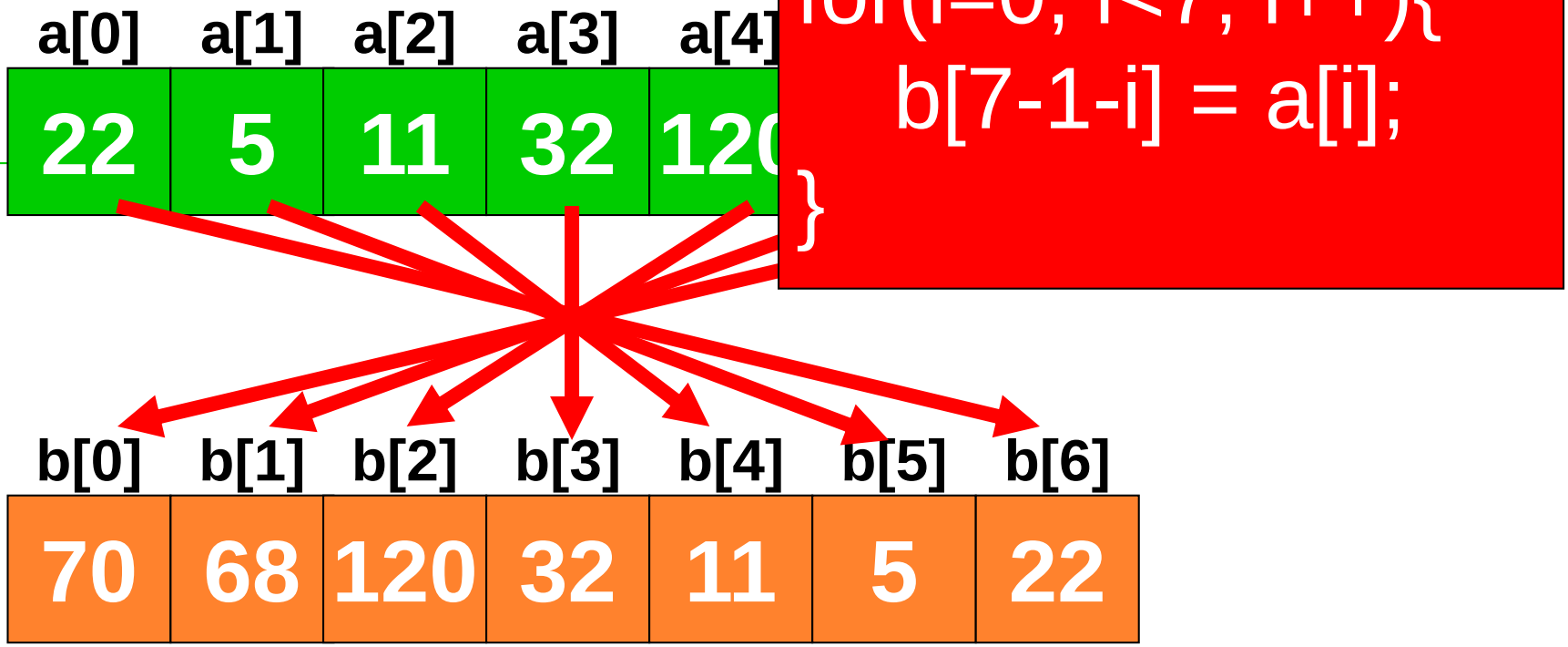
実行は上から下へ

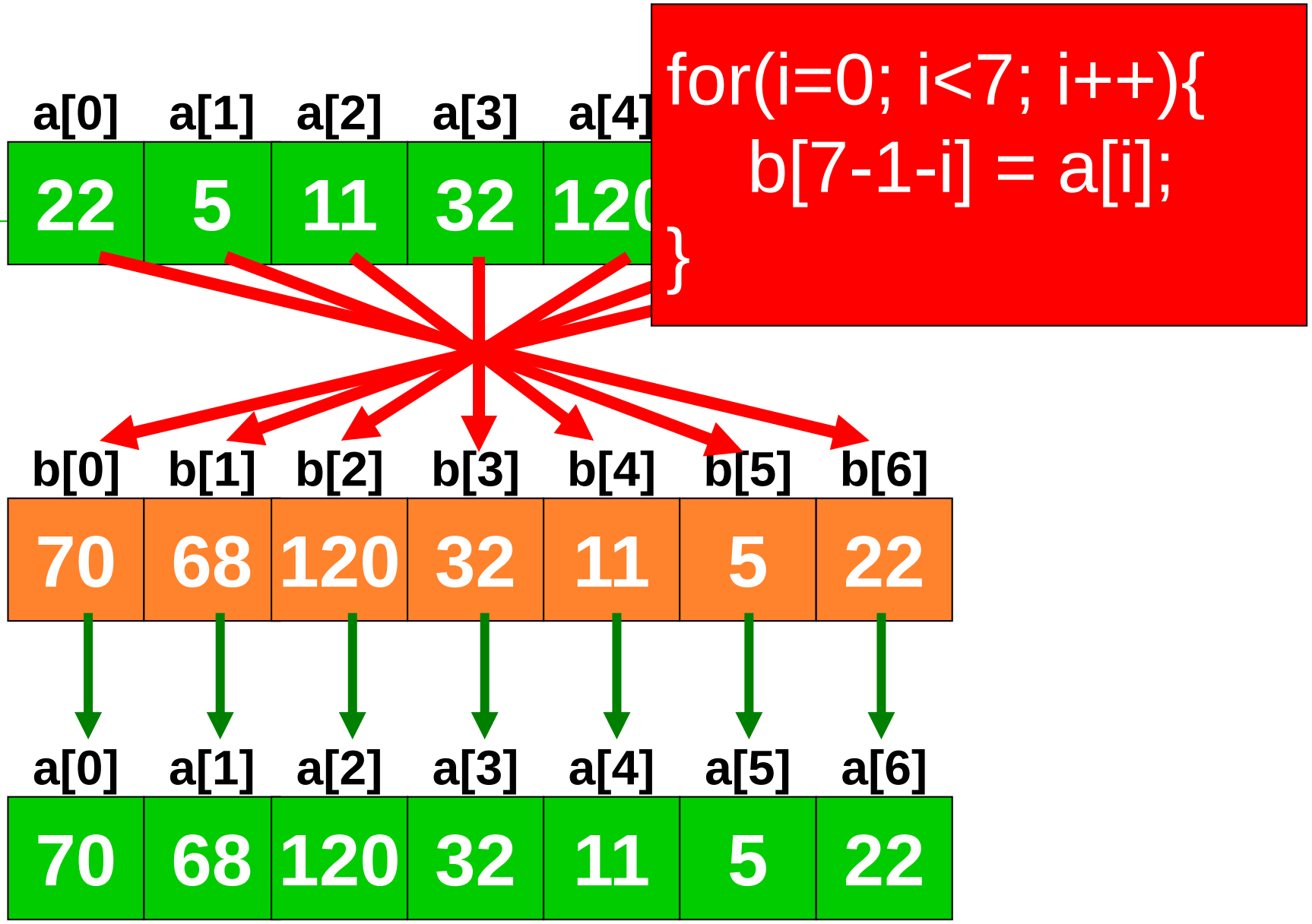
```
int main(void)
{
    int     i;
    int     x[7];
    int     nx =

    printf("%d
    for (i = 0;
            print
            scanf(
    }

    ary_reverse(x, nx);/* 配列xの要素の並びを逆転 */

                の要素
                < nx
                [%
}
```

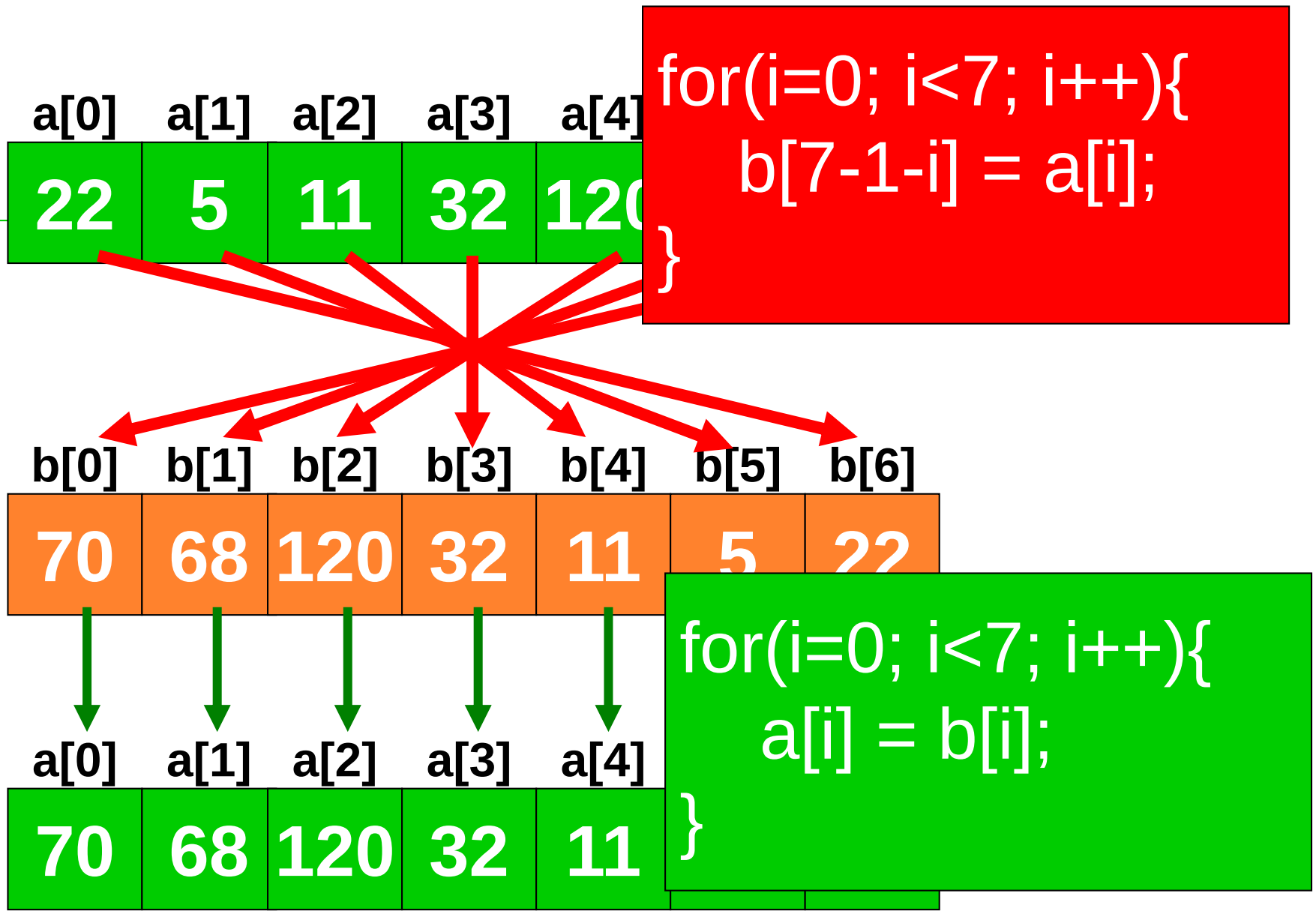
データが入っている最初の位置

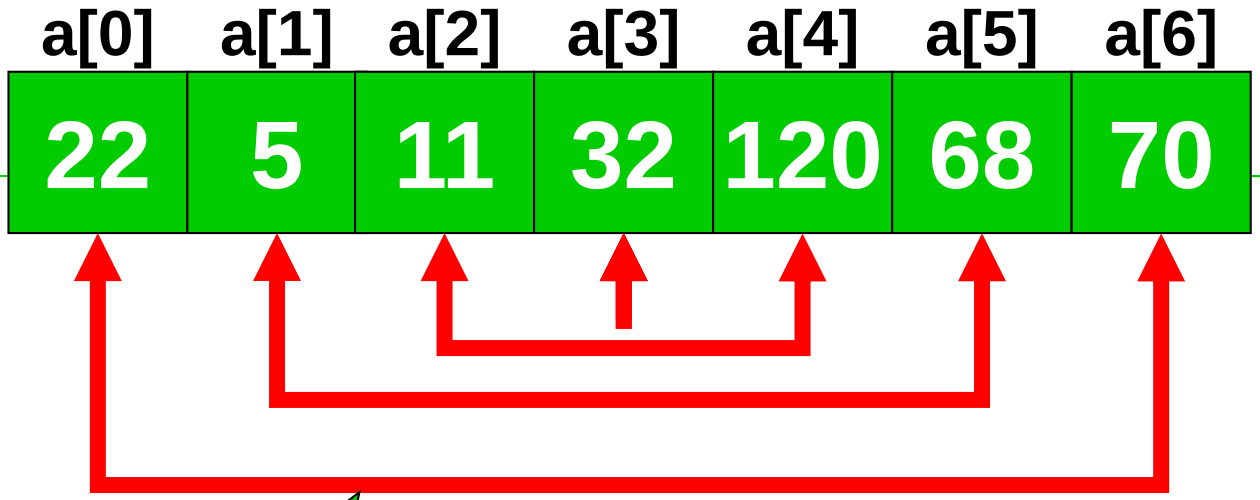
データの個数


a[0] a[1] a[2] a[3] a[4] a[5] a[6]
22 5 11 32 120 68 70
b[0] b[1] b[2] b[3] b[4] b[5] b[6]
70 68 120 32 11 5 22

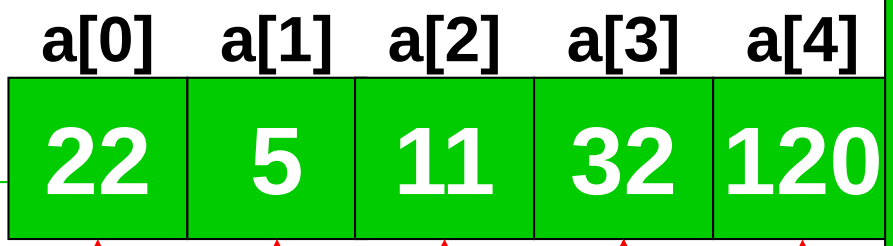

a[0] a[1] a[2] a[3] a[4]
22 5 11 32 120
for(i=0; i<7; i++){
b[7-1-i] = a[i];
}
b[0] b[1] b[2] b[3] b[4] b[5] b[6]
70 68 120 32 11 5 22


a[0] a[1] a[2] a[3] a[4]
22 5 11 32 120
for(i=0; i<7; i++){
b[7-1-i] = a[i];
}
b[0] b[1] b[2] b[3] b[4] b[5] b[6]
70 68 120 32 11 5 22
a[0] a[1] a[2] a[3] a[4] a[5] a[6]
70 68 120 32 11 5 22

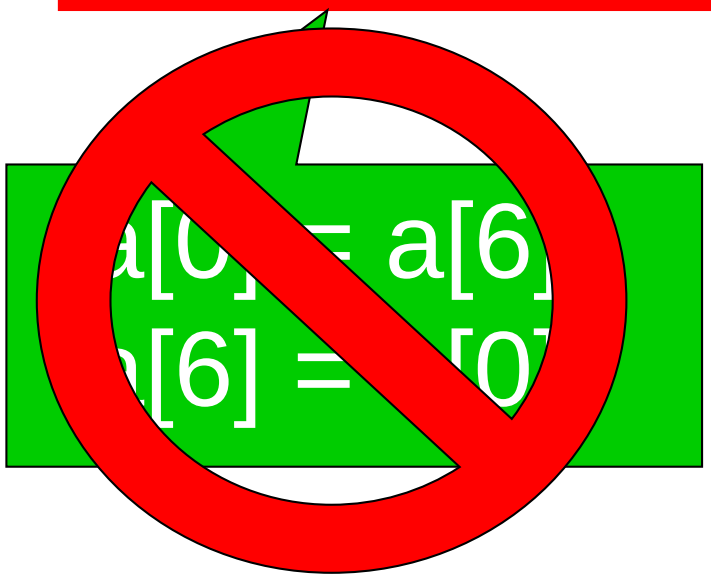

a[0] a[1] a[2] a[3] a[4]
22 5 11 32 120
for(i=0; i<7; i++){
b[7-1-i] = a[i];
}
b[0] b[1] b[2] b[3] b[4] b[5] b[6]
70 68 120 32 11 5 22
a[0] a[1] a[2] a[3] a[4]
70 68 120 32 11
for(i=0; i<7; i++){
a[i] = b[i];
}

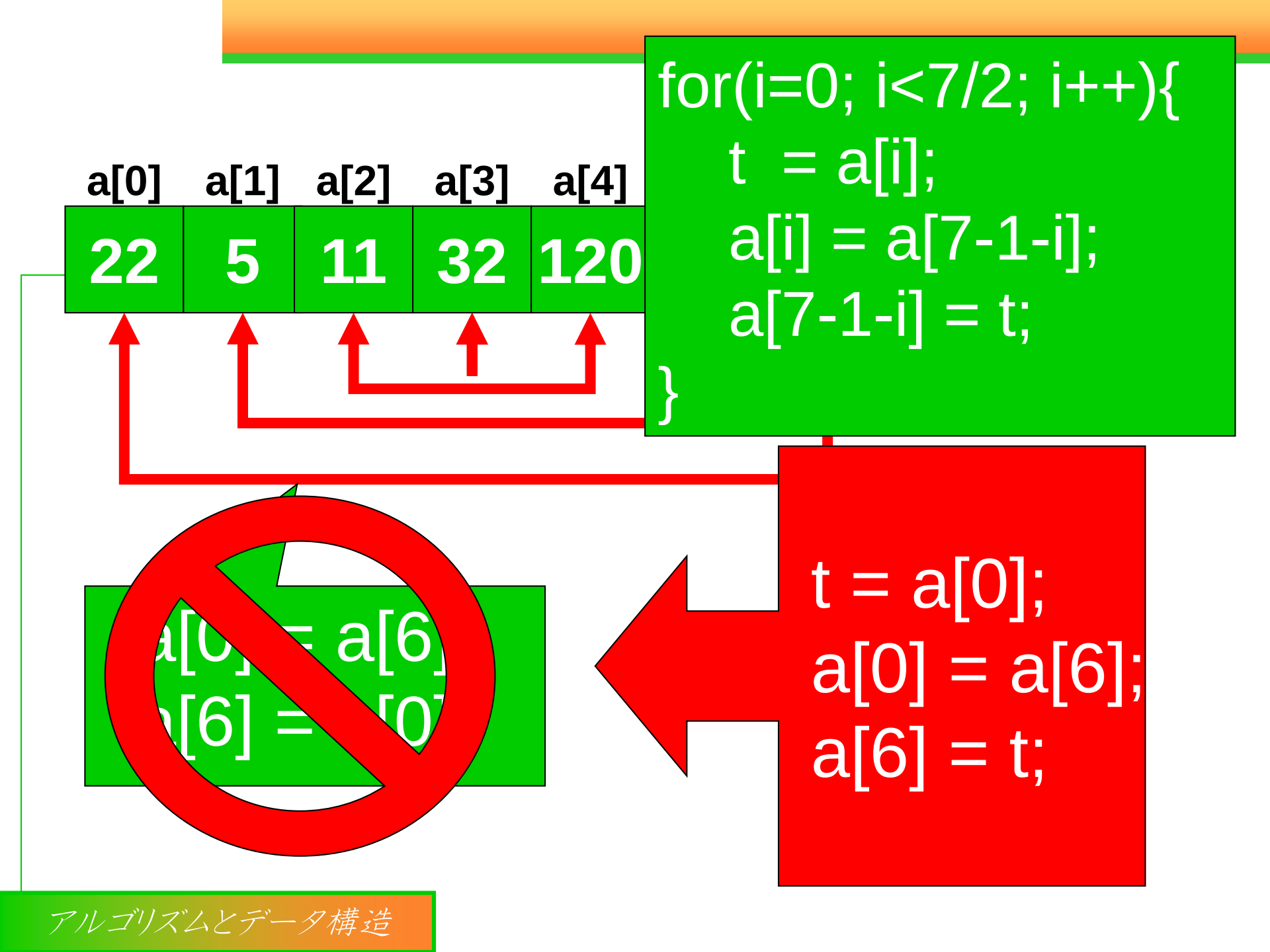



```
a[0] = a[6];
a[6] = a[0];
```




```
for(i=0; i<7/2; i++){
    t  = a[i];
    a[i] = a[7-1-i];
    a[7-1-i] = t;
}
```



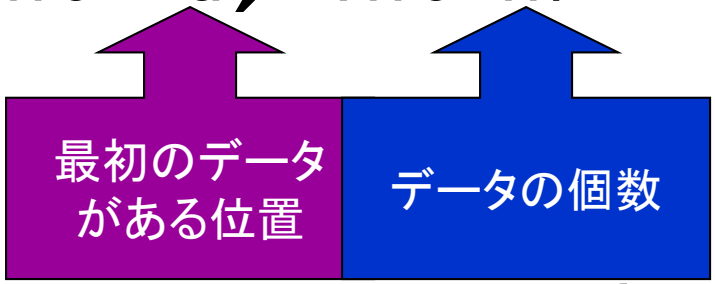

a[0] a[1] a[2] a[3] a[4]
22 5 11 32 120
for(i=0; i<7/2; i++){
t = a[i];
a[i] = a[7-1-i];
a[7-1-i] = t;
}
t = a[0];
a[0] = a[6];
a[6] = t;

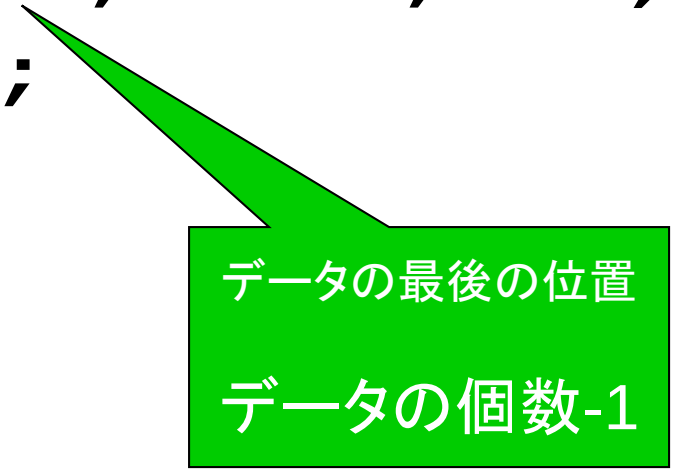
要素数nの配列aの要素の並びを逆転

```
void ary_reverse(int a[], int n)
{
 int i;
 for(i=0; i < n/2; i++){
  int t = a[i];
  a[i] = a[n-i-1]; /* swap(int,a[i],a[n-i-1]) */
  a[n-i-1] = t;
 }
}
```

ポインターによる表現

```
void ary_reverse(int *a, int n)
{
  int *b;
    for(b=a+n-1; a < b; a++,b--){
     int t=*a;
     *a=*b;
     *b=t;
    }
}
```





t=10

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]
10	17	11	19	20	11	7

a

b

最初のデータがある位置

データの個数

```
void ary_reverse(int *a, int n)
{
  int *b;
   for(b=a+n-1; a < b; a++,b--){
    int t=*a;
    *a=*b;
    *b=t;
   }
}
```

データの最後の位置

データの個数-1

t=17

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]
10	17	11	19	20	11	7

a

b

```
void ary_reverse(int *a, int n)
{
  int *b;
   for(b=a+n-1; a < b; a++,b--){
    int t=*a;
    *a=*b;
    *b=t;
   }
}
```

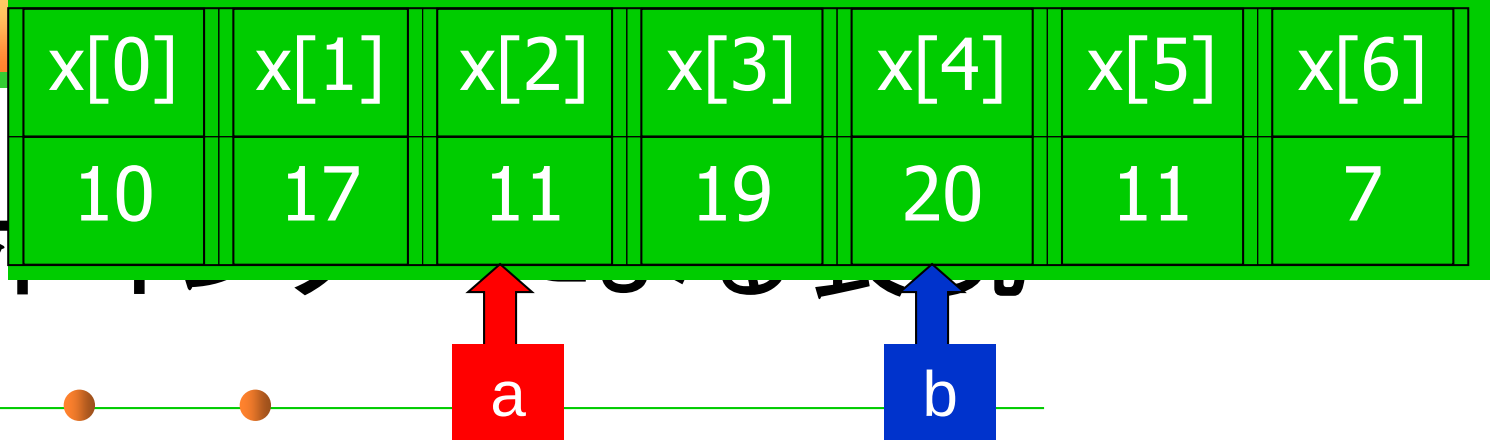
最初のデータがある位置

データの個数

データの最後の位置

データの個数-1

t=11




```
void ary_reverse(int *a, int n)
{
  int *b;
  for(b=a+n-1; a < b; a++,b--){
    int t=*a;
    *a=*b;
    *b=t;
  }
}
```

最初のデータがある位置

データの個数

データの最後の位置

データの個数-1

t=11

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]
10	17	11	19	20	11	7

a b

```
void ary_reverse(int *a, int n)
{
  int *b;
   for(b=a+n-1; a < b; a++,b--){
    int t=*a;
    *a=*b;
    *b=t;
   }
}
```

最初のデータがある位置

データの個数

データの最後の位置

データの個数-1

途中結果の表示

```
void ary_reverse(int *a, int n)
{
  int *b;
    for(b=a+n-1; a < b; a++,b--){
     int t=*a;
     *a=*b;
     *b=t;
    }
}
```

途中結果の表示

```
printf("t=%d, *a=%d,*b=%d¥n",t,*a,*b);
```

```
  int *b;
    for(b=a+n-1; a < b; a++,b--){
      int t=*a;
      *a=*b;
      *b=t;
    }
}
```

結果を表示したい場所に挿入

プログラムの実行

- main関数が最初に実行
  - 必ず，main関数は1つだけ存在する
  - プログラムの位置に無関係
- 関数内での実行順序
  - 命令(1命令は;まで)を1つづつ逐次実行
  - 関数の最初から最後へ（左→右，上↓下）
  - 繰り返しなどを含む複雑な命令でも，同じ

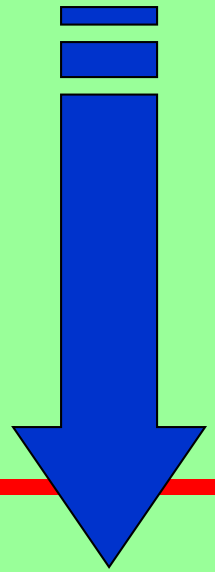
```
for(b=a+n-1; a < b; a++,b--){
    t=*a;
    *a=*b;
    *b=t;
}
```

- 必ず，main関数は1つだけ存在する
- プログラムの位置に無関係

- 関数内での実行順序
  - 命令(1命令は;まで)を1つづつ逐次実行
  - 関数の最初から最後へ（左→右，上↓下）
  - 繰り返しなどを含む複雑な命令でも，同じ

```
for(b=a+n-1; a < b; a++,b--){
    t=*a;
    *a=*b;
    *b=t;
}
```

```
  b=a+n-1;
Check_Loop:
  if (!(a < b )) goto End_of_Loop;
  t=*a; *a=*b; *b=t;

  a++;
  b--;
goto Check_Loop;
End_of_Loop:
```



- 関数

```
for(b=a+n-1; a < b; a++,b--){
    t=*a;
    *a=*b;
    *b=t;
}
```

```
b=a+n-1;
while(a < b){
  t=*a;
  *a=*b;
  *b=t;
  a++;
  b--;
}
```

```
f_Loop;
  b--;
goto Check_Loop;
End_of_Loop:
```

例題（教科書76ページ）

- 年内の経過日数を計算する
  - 西暦で表された年，月，日の三値を与える.
- その年の1月1日からの経過日数を計算
  - 1月1日を指定した場合を1日とする.
- 各月の日数が異なる
  - 表として各月の日数を記憶
  - 計算で表現できない情報では，一般的な方法

```
int mdays[12] ={
        31,28,31,30,31,30,
        31,31,30,31,30,31
};
```

- 年内
  - 西
- その年の1
- 各月の日数が異なる
  - 表として各月の日数を記憶
  - 計算で表現できない情報では，一般的な方法

閏年の問題

- 一般の年と閏年で2月の日数が異なる
  - 2種類の表を用意する
  - 閏年を判定して，表を使い分ける
- 閏年の判定（条件）：西暦表示
  - 4で割り切れ，かつ100で割り切れない年
  - または，400で割り切れる年

```
int mdays[][12] ={
 {31,28,31,30,31,30,
       31,31,30,31,30,31},
 {31,29,31,30,31,30,
       31,31,30,31,30,31}
};
```

- 一般の
  - 2
  - 閏年を
- 閏年の判定（条件）：西暦表示
  - 4で割り切れ，かつ100で割り切れない年
  - または，400で割り切れる年

List2-12の補足

- 論理式の値（ifの条件式）
  - 真： 1
  - 偽： 0
- isleap関数のreturnで使用（戻値は0か1）

```
return    year%4 == 0   &&
    year % 100 != 0 ||
    year % 400 == 0 ;
```