プログラミング演習
ソーティング（1）

阿萬裕久
aman@cs.ehime-u.ac.jp

ソーティング（sorting）

- データを何らかの順序で並べ替えること
  - 昇順（小さい順），降順（大きい順），50音順．．．

例えば，値段の安い順

例えば，売れている順

データ処理の基本となる操作

詳しくは，後期の「データ構造とアルゴリズム」でやります．
本演習ではその初歩についてプログラミングを行ってみましょう．

例題1

※ソーティングの前に，ひとまず基本を！

- 5 個の整数を順に読み込み，その中の最大値を表示しなさい．
  ただし，各整数は入力順に配列 a に格納すること．

例題1：解説

Web ページのコーディング例を参照

- まずはデータを順に読み込み，配列へ格納

12～14行目

```
for ( i = 0; i < 5; i++ ){
  scanf("%d", &a[i]);
}
```

入力例　8　-2　91　0　1

	0	1	2	3	4
a	8	-2	91	0	1

例題1：解説

Web ページのコーディング例を参照

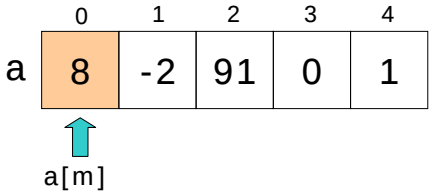
- 最大値を a[m] で表す
  - 実際には m の値は 0 ～ 4 のいずれかになる
- そして，ひとまず a[0] を最大と仮定する

16行目

```
m = 0;
```


例題1：解説

- 最初は m = 0



例題1：解説

Web ページのコーディング例を参照

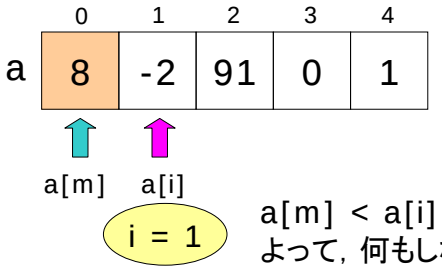
- その後，a[1] ～ a[4] と a[m] を比較していき，a[m] より大きいものと遭遇すれば，そちらを改めて a[m] とする

17～21行目

```
for ( i = 1; i < 5; i++ ){
  if ( a[m] < a[i] ){
    m = i;
  }
}
```


例題1：解説

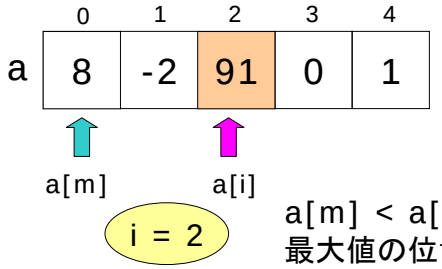
- i = 1 ～ 4 として，a[m] と a[i] の比較



a[m] < a[i] は不成立
よって，何もしない

例題1：解説

- i = 1 ～ 4 として，a[m] と a[i] の比較



i = 3, 4 の場合も同様なので説明は省略

a[m] < a[i] は成立
最大値の位置を変更

```
if ( a[m] < a[i] ){
   m = i;
}
```


例題1：解説

Web ページのコーディング例を参照

- 最後に a[m] を出力して終わり

23行目

```
printf("最大値 = %d¥n", a[m]);
```

※説明するまでもないが，同様にして
最小値を見つけることもできる

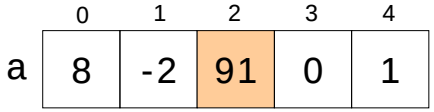
例題2

- 5 個の整数を順に読み込み，配列 a に格納しなさい．
  そして，それらを昇順（小さい順）に並び替えて表示しなさい．

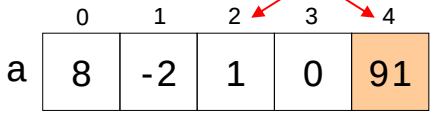
例題2：解説

- 大まかな手順

①最大値を見つけ出す

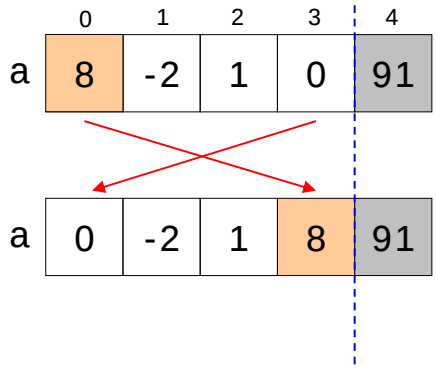


②右端と交換する



例題2：解説

③残りについても同様に繰り返す



例題2：解説

- このように
  - 特定の1個（最大値等）を見つけ出し
  - 適切な位置へ移動させる

という手順（アルゴリズム）で行うソーティングを**選択ソート**（selection sort）という

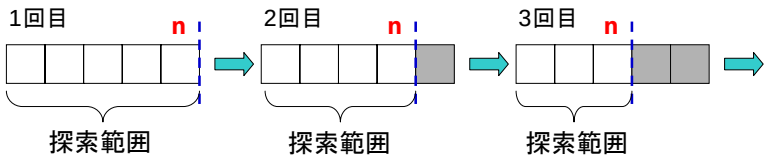
例題2：解説

Web ページのコーディング例を参照

- 右端の位置（これを **n** とおく）は繰り返しのたびに左へ1マスずつずれていく

16～28行目

```
for ( n = 4; n >= 0; n-- ){
   . . . . . .
}
```



例題2：解説

Web ページのコーディング例を参照

- n を使った繰り返しの中で a[0] ～ a[n] から最大値を探す

「最大値を a[m] で表す」という前提に注意

17～22行目

```
m = 0;
for ( i = 1; i <= n; i++ ){
    if ( a[m] < a[i] ){
        m = i;
    }
}
```

最大値は a[0] と仮定

もっと大きい値があればそちらを a[m] とする

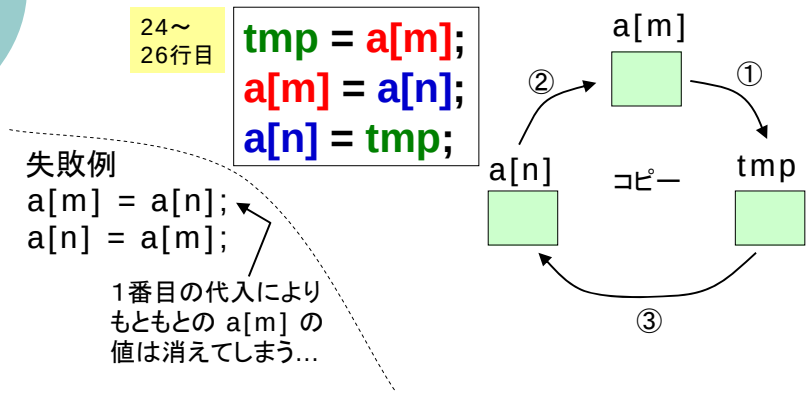
例題2：解説

Web ページのコーディング例を参照

- 最大値（a[m]）と右端（a[n]）の値を交換

24～26行目

```
tmp = a[m];
a[m] = a[n];
a[n] = tmp;
```



例題3

- 任意個の整数を読み込み，入力順に配列 a へ格納しなさい．ただし，最初に個数を読み込むものとする．

  なお，所定の個数を格納できるだけの配列をメモリ上に確保できない場合は「メモリが足りません」というエラーメッセージを出力して終了せよ．

例題3：解説

- 配列を用意する基本的な方法

```
型名  配列名[大きさ];
```

（例1）

```
int  a[10];
```

（例2）

```
#define  N  256
.....
int  a[N];
```

いずれにしても**大きさ**は**事前に指定**

例題3：解説

- データの個数が**実行するまで分からない**場合はどうする？
  - 1つの対処法

    予想される最大個数の分だけ配列を用意

しかし！
- こちらの予想を超えたらそこまでである
- 予想よりはるかに個数が少ない場合，無駄にメモリを食うソフトになる（重くなる）

例題3：解説

- 必要な時に必要な分だけ配列を確保する方法がある： 以下は定番のコード

```
型名* 配列名;
配列名 = (型名*)calloc(サイズ, sizeof(型名));
```

（例）

```
int* a;
a = (int*)calloc(n, sizeof(int));
```

scanf で実行時に読み込む

例題3：解説

- 関数 calloc

```
calloc(n, b);
```

変数 n 個分の配列をメモリ上に確保する

変数1個分には b バイトが必要という意味

この関数の戻り値は，確保した配列の先頭番地（ポインタ）

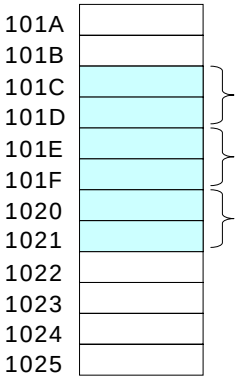
例題3：解説

- 例えば，

```
calloc(3, 2);
```

自動的に空き領域から
2バイト×3マス
が確保される

calloc(3,2) の戻り値は
先頭番地 101C となる



例題3：解説

- calloc 関数の戻り値（前の例でいえば 101C）は**番地なので，ポインタ変数**に代入すれば利用できる

```
int* a;
a = (int*)calloc(3, 2);
```

ポインタの**参照先の型が何なのか**を明記しないと処理に困ってしまう．仮に int 型ならば，a はそのためのポインタ（int*）となる．

例題3：解説

- 常に int 型が 2 バイトとは限らないので，その実行環境で int 型が何バイトなのかを調べる
  - これが sizeof の役割

```
a = (int*)calloc(3, 2);
```

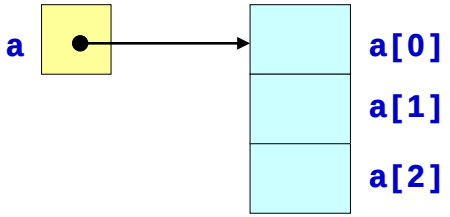
↓

```
a = (int*)calloc(n, sizeof(int));
```


例題3：解説

- 前回説明したように，配列はポインタ変数を使った構成になっている

```
a = (int*)calloc(3, sizeof(int));
```



例題3：解説

- メモリ確保に失敗した場合
  - メモリが足りないこともあり得る
  - その場合，calloc 関数の戻り値は NULL

```
a = (int*)calloc(3, sizeof(int));
if ( a == NULL ){
    メモリが足りない場合の処理
}
```


例題3：解説

- 不要になったらメモリを解放する
  - calloc 関数を使って確保したメモリ（配列）は自分で解放を指示しなければならない
  - 解放には free 関数を用いる

```
free( 配列名 );
```

誤って2回実行すると実行時にエラーとなるので注意！

（例）

```
free( a );
```

free を使う習慣を身に付けること！
メモリリーク（使用可能なメモリがだんだん減っていく問題）を未然に防ぐ