

情報基礎 B
第 14 回 アカデミック・スキル II
C 言語プログラミング (6)

長江 剛志

(nagae@m.tohoku.ac.jp)

東北大学大学院工学研究科
技術社会システム専攻

2015 年 7 月 17 日 (金)

今日やること

今日やること

配列を用いた複数のデータの処理

バブルソート

レポート課題 III-4: 素数と合成数に分類

目次

ディレクトリとサンプル・コードの準備

配列を用いた複数のデータの処理

レポート課題

ディレクトリとサンプル・コードの準備 (1)

ディレクトリの準備

Terminal 上から `mkdir` コマンドを使って

`~/Documents/prog/04_array_func` というディレクトリを作る.

サンプル・コードのダウンロード

- ▶ ISTU (<http://www.istu.jp>) にアクセスし, 右下の 受講授業科目 から, 金曜の 3 時限 情報基礎 B を選択
- ▶ 科目共通教材 の中から 配布資料 第 14 回 C 言語サンプルプログラム (4) を選択

ディレクトリとサンプル・コードの準備 (2)

サンプル・コードの移動

- ▶ Home (メニューバーの「場所」でもよい) を開いて「ダウンロード」内にある上記のファイルを
~/Documents/prog/04_array へ移動.

現在の作業ディレクトリの変更

Terminal 上の `cd` コマンドを使って、現在の作業ディレクトリを
~/Documents/prog/04_array に変更する.

目次

ディレクトリとサンプル・コードの準備

配列を用いた複数のデータの処理

レポート課題

配列にデータを格納・表示する (nonarray_reverse.c, array_reverse.c) (1)

やりたいこと

4 個の整数を入力した後、それらを 入力されたのと逆の順 に表示させる。

配列を使わないサンプル・コード (nonarray_reverse.c)

```
1 #include <stdio.h> /* 標準入出力ライブラリ */
2 int main(void) /* main 関数の引数と戻り値の定義 */
3 { /* main 関数の始まり */
4     int a0, a1, a2, a3; /* 整数型の変数 a0~a3 を宣言 */
5
6     /* 4つの整数を読み込む */
7     printf("4つの整数? ");
8     scanf("%d %d %d %d", &a0, &a1, &a2, &a3); /* 4つの整数を a0, a1, a2, a3 の順に読み込む */
9
10    /* 入力されたのと逆の順に表示 */
11    printf("%d %d %d %d\n", a3, a2, a1, a0); /* a3, a2, a1, a0 の順に表示 */
12
13    return 0; /* 戻り値として 0 を返す */
14 }
```

配列にデータを格納・表示する (nonarray_reverse.c, array_reverse.c) (2)

配列を使ったサンプル・コード (array_reverse.c)

```
1 #include <stdio.h>           /* 標準入出力ライブラリ */
2 int main(void)               /* main 関数の引数と戻り値の定義 */
3 {                             /* main 関数の始まり */
4     int N = 4;               /* 入力する整数の数(この場合4)を N に格納 */
5     int a[N];                /* N個の要素を持つ整数型配列 a を定義 */
6
7     int i;                   /* 要素番号を格納するための変数 */
8
9     /* N個の整数を読み込む */
10    printf("%d個の整数? ", N);
11    for (i = 0; i < N; ++i)   /* iを0からN-1まで1つずつ増やしながら繰り返し */
12        { scanf("%d", &a[i]); /* 配列aのi番目要素 a[i] に値を読み込む */
13    }
14
15    /* 入力されたのと逆の順に表示 */
16    for (i = N-1; i >= 0; --i) /* iをN-1から0まで1つずつ減らしながら繰り返し */
17        { printf("%d ", a[i]); /* 配列aのi番目要素 a[i] を出力する */
18    }
19    printf("\n");             /* 最後に改行を出力 */
20
21    return 0;                 /* 戻り値として 0 を返す */
22 }
```


nonarray_reverse.c のコンパイルと実行

1. gcc コマンドを用いてソースファイルから **実行ファイル** を生成する. -o オプションを用いて, 実行ファイル名を nonarray_reverse とする.

```
$ gcc -o nonarray_reverse nonarray_reverse.c
```

2. 生成された実行ファイル (./nonarray_reverse) を呼び出す.

```
$ ./nonarray_reverse
```

```
4個の整数? 7 3 1 4
```

```
4 1 3 7
```

入力待ち状態になるので4つの数を適当に

3. array_reverse.c も同様にコンパイル・実行してみよう

array_reverse.c の解説 (1)

配列 は複数のデータをまとめて扱うのに必須の機能。

- ▶ nonarray_reverse.c では a0, a1, a2, a3 という 4 個の変数を個別に宣言し、それらに値を順に代入した後、a3, a2, a1, a0 の順に出力した。この方法だと、入力する値が増減した場合、

- ▶ 変数の宣言を行なう行 (第 4 行)

```
int a0, a1, a2, a3;           /* 整数型の変数 a0~a3 を宣言 */
```

- ▶ キーボードから入力された値を変数に取り込む行 (第 8 行)

```
scanf("%d %d %d %d", &a0, &a1, &a2, &a3); /* 4つの整数を a0, a1, a2, a3 の順に読
```

- ▶ 出力する行 (第 11 行)

```
printf("%d %d %d %d\n", a3, a2, a1, a0); /* a3, a2, a1, a0 の順に表示 */
```

を全て変更する必要がある。

array_reverse.c の解説 (2)

- ▶ 一方, array_reverse.c では, まず, a[0], a[1], a[2], a[3] という N=4 個の要素を持つ配列 a を宣言し, それぞれの要素に値を順に代入した後, a[3], a[2], a[1], a[0] の順に出力している. 具体的には,
 - ▶ 第 5 行目 (配列の宣言):

```
int a[N];                                /* N個の要素を持つ整数型配列 a を定義 */

int i;                                  /* 要素番号を格納するための変数 */

/* N個の整数を読み込む */
printf("%d個の整数? ", N);
for (i = 0; i < N; ++i)                 /* iを0からN-1まで1つずつ増やしながら繰り返し */
    { scanf("%d", &a[i]); }             /* 配列aのi番目要素 a[i] に値を読み込む */

/* 入力されたのと逆の順に表示 */
for (i = N-1; i >=0; --i)               /* iをN-1から0まで1つずつ減らしながら繰り返し */
    { printf("%d ", a[i]); }            /* 配列aのi番目要素 a[i] を出力する */
printf("\n");                           /* 最後に改行を出力 */

return 0;                               /* 戻り値として 0 を返す */
}
```

N 個の要素を持つ整数 (int) 型の配列 a を宣言している. 配列は以下の書式で宣言する:

array_reverse.c の解説 (3)

変数の型 配列名 [配列の要素数];

配列の各要素に値を代入したり, 値を参照したりするには,
a[0], a[1], ... のように,

配列名 [配列の要素番号]

とすることで, 通常の変数と同様にアクセスできる.

配列の要素番号は 0 から始まる. つまり, N 個の要素を持つ配列 a の先頭の要素は a[0], 最後の要素は a[N-1] で表される.

- ▶ 第 11~12 行目 (キーボードから入力された値を変数に取り込む):

```
for (i = 0; i < N; ++i)      /* iを0からN-1まで1ずつ増やしながら繰り返し */
{ scanf("%d", &a[i]); }      /* 配列aのi番目要素 a[i] に値を読み込む */
```

for 文を使って, i の値を 0 から N-1 まで 1 ずつ増やしながら配列の要素 a[i] にキーボードから入力された値を読み込んでいる. このように「配列の各要素 に対して 同じ処理 を繰り返す」ことが簡単に実装できるのが配列のメリットである.

array_reverse.c の解説 (4)

- ▶ 第 15～16 行目 (入力されたのと逆順に出力):

```
for (i = N-1; i >=0; --i)    /* iをN-1から0まで1ずつ減らしながら繰り返し */
{ printf("%d ", a[i]); }    /* 配列aのi番目要素 a[i] を出力する */
```

for 文を使って, i の値を $N-1$ から 0 まで 1 ずつ減らしながら配列の要素 $a[i]$ を出力している.

配列に任意の個数のデータを格納する (array_reverse_v2.c) (1)

やりたいこと

array_reverse.c を改良して, 10 個以下の任意の個数の正の整数を入力した後, それらを 入力されたのと逆の順 に表示させる. 負の値 が入力された場合, 10 個未満でも入力を打ち切り, そこまで入力された整数を逆順に表示させる.

サンプル・コード (array_reverser_v2.c)

配列に任意の個数のデータを格納する (array_reverse_v2.c) (2)

```
1  #include <stdio.h>          /* 標準入出力ライブラリ */
2  int main(void)              /* main 関数の引数と戻り値の定義 */
3  {                            /* main 関数の始まり */
4      int N_max = 10;          /* 入力できる数の上限(10)を N_max に可能 */
5      int a[N_max];           /* N_max個の要素を持つ整数型配列 a を定義 */
6      int N = 0;              /* 入力された整数の個数を格納する変数 */
7
8      int i;                  /* 要素番号を格納するための変数 */
9      int input;              /* キーボードから入力された値を一時的に格納する変数 */
10
11     /* 負の値が入力されるか N_max 個の整数が入力されるまで繰り返す */
12     for (i = 0; i < N_max; ++i) /* i を 0 から N_max-1 まで 1 つずつ増やしながら繰り返し */
13     {
14         scanf("%d", &input);    /* キーボードから入力された値を input に読み込む */
15         if (input > 0)          /* 入力値が正ならそれを a[N] に格納し, N を 1 増やす */
16         {
17             a[N] = input;
18             ++N;
19         }
20         else                    /* 入力値が負なら break でループを抜ける */
21             { break; }
22     }
23
24     /* 入力されたのと逆の順に表示 */
25     for (i = N-1; i >= 0; --i) /* i を N-1 から 0 まで 1 つずつ減らしながら num 期化で i の値を 1 つ減らし
26     { printf("%d ", a[i]); }    /* 配列 a の i 番目要素 a[i] を出力する */
27     printf("\n");              /* 最後に改行を出力 */
28
29     return 0;                  /* 戻り値として 0 を返す */
30 }
```

array_reverse_v2.c のコンパイルと実行

1. gcc コマンドを用いてソースファイルから **実行ファイル** を生成する. -o オプションを用いて, 実行ファイル名を array_reverse_v2 とする.

```
$ gcc -o array_reverse_v2 array_reverse_v2.c ↵
```

2. 生成された実行ファイル (./array_reverse_v2) を呼び出す.

```
$ ./array_reverse_v2 ↵  
7 3 1 4 5 -1          # いくつかの数の後 -1 を入れて ↵  
5 4 1 3 7  
1 2 3 4 5 6 7 8 9 10 11 # 10個以上の値を入れた場合は -1 は不要  
10 9 8 7 6 5 4 3 2 1
```


array_reverse_v2.c の解説 (1)

▶ 4～6 行目

```
int N_max = 10;          /* 入力できる数の上限(10)を N_max に可能 */
int a[N_max];            /* N_max個の要素を持つ整数型配列 a を定義 */
int N = 0;               /* 入力された整数の個数を格納する変数 */
```

実際に入力される整数の個数が実行されるまで判らないので、入力できる整数の個数の上限を `N_max` に格納しておき、`max_N` 個の要素を持つ配列 `a` を宣言.

実際に入力された整数の個数は、上限 `N_max` とは別の整数型変数 `N` (第 6 行目で宣言) に格納する.

▶ 12～22 行目

array_reverse_v2.c の解説 (2)

```
for (i = 0; i < N_max; ++i) /* iを0からN_max-1まで1つつ増やしなが  
{                               り繰り返す */  
    scanf("%d", &input);      /* キーボードから入力された値を input に読み込む */  
    if (input > 0)             /* 入力値が正ならそれを a[N] に格納し、N を1増やす */  
    {  
        a[N] = input;  
        ++N;  
    }  
    else                       /* 入力値が負なら break でループを抜ける */  
    { break; }  
}
```

for 文を使って i を 0 から N_max-1 まで 1 つずつ増やしなが
らキーボードから入力された値を `input` に格納している。

`array_reverse.c` とは違い、15～19 行目の `if` ブロックで入
力された値が正だった場合 に限り、その値を `a[N]` に格納
し、入力された整数の個数 N を 1 つ増やしている。

同時に、20～21 行目の `else` ブロックで、入力された値が負
だった場合 には、`break` を使って for ループから抜け出して

array_reverse_v2.c の解説 (3)

いる. これによって, `N_max` より少ない個数でも入力を打ち切ることができる.

どの時点においても, `N` には, **それまで入力された整数の個数**が格納され, `N > 0` ならば, それまで入力された値は `a[0]`, ..., `a[N-1]` に格納されているので, 最後に, 25~27 行目で, これを `a[N-1]`, ..., `a[0]` の順に表示させている.

入力された整数を小さい順に並べ替える (1)

やりたいこと

array_reverse.c を改良して, 7 個 の整数を入力した後, それらを 小さい順 に表示させる.

サンプル・コード (array_sort.c)

入力された整数を小さい順に並べ替える (2)

```
1  #include <stdio.h>          /* 標準入出力ライブラリ */
2  int main(void)              /* main 関数の引数と戻り値の定義 */
3  {                            /* main 関数の始まり */
4      int N = 7;              /* 入力する整数の数(この場合4)を N に格納 */
5      int a[N];               /* N個の要素を持つ整数型配列 a を定義 */
6
7      int i, j;               /* 要素番号を格納するための変数 */
8      int tmp;                /* 配列の要素を入れ替える際に値を一時的に格納する変数 */
9
10     /* N個の整数を読み込む */
11     printf("%d個の整数? ", N);
12     for (i = 0; i < N; ++i)  /* iを0からN-1まで1つずつ増やしながら繰り返し */
13     { scanf("%d", &a[i]); }  /* 配列aのi番目要素 a[i] に値を読み込む */
14
15     /* 小さい順にバブルソート */
16     for (i = 0; i < N; ++i) { /* ソートされていない一番左端の要素 a[i] を取り出す */
17         for (j = i+1; j < N; ++j) { /* a[i] が a[i+1]...a[N-1]の中で最小となるよう入れ替え */
18             if (a[i] > a[j]) { /* a[j] < a[i] なら a[i] と a[j] を入れ替える */
19                 tmp = a[j];
20                 a[j] = a[i];
21                 a[i] = tmp;
22             }
23         }
24     }
25
26     /* 配列を順に表示 */
27     for (i = 0; i < N; ++i)  /* iをN-1から0まで1つずつ減らしながら繰り返し */
28     { printf("%d ", a[i]); } /* 配列aのi番目要素 a[i] を出力する */
29     printf("\n");           /* 最後に改行を出力 */
30
31     return 0;               /* 戻り値として 0 を返す */
32 }
```

array_sort.c のコンパイルと実行

1. gcc コマンドを用いてソースファイルから **実行ファイル** を生成する. -o オプションを用いて, 実行ファイル名を array_sort とする.

```
$ gcc -o array_sort array_sort.c ↵
```

2. 生成された実行ファイル (./array_sort) を呼び出す.

```
$ ./array_sort ↵
7個の整数? 8 4 5 6 1 3 5 # 入力待ち状態になるので7つの数を適当に入力
1 3 4 5 5 6 8 # 小さい順に並べてくれる
$ ./array_sort ↵
7個の整数? 3 7 2 5 6 1 9 # 色々試してみよう
1 2 3 5 6 7 9
```

array_sort.c の解説 (1)

▶ 8 行目

```
int tmp; /* 配列の要素を入れ替える際に値を一時的に格納する変数 */
```

配列の要素を入れ替える際に、値を一時的に格納しておくための変数 tmp を定義しておく。

▶ 16～24 行目

```
for (i = 0; i < N; ++i) { /* ソートされていない一番左端の要素 a[i] を取り出す */
    for (j = i+1; j < N; ++j) { /* a[i] が a[i+1]...a[N-1]の中で最小となるよう入れ替え
        if (a[i] > a[j]) { /* a[j] < a[i] なら a[i] と a[j] を入れ替える */
            tmp = a[j];
            a[j] = a[i];
            a[i] = tmp;
        }
    }
}
```

小さい順に配列をバブルソートする。2重の for ループで構成されており、それぞれの処理は

array_sort.c の解説 (2)

- ▶ **外側のループ**: 配列 $a[0] \dots a[N-1]$ の左から順にソートしていく.
- ▶ **内側のループ**: ソートされていない部分 $a[i] \dots a[N-1]$ の中で最も小さい要素を $a[i]$ に代入する.

```
for (j = i+1; j < N; ++j) { /* a[i] が a[i+1]...a[N-1]の中で最小となるよう入れ替える */
    if (a[i] > a[j]) { /* a[j] < a[i] なら a[i] と a[j] を入れ替える */
        tmp = a[j];
        a[j] = a[i];
        a[i] = tmp;
    }
}
```

$a[i]$ と $a[i+1] \dots a[N-1]$ を順に比較し「 $a[i]$ の方が $a[j]$ より大きかったら $a[i]$ と $a[j]$ を入れ替える」という操作を繰り返す.

例えば, 外側のループにおいて $i=3$ で, $a[3]$, $a[4]$, $a[5]$, $a[6]$ にそれぞれ 7, 5, 3, 6 が格納されていたとしよう. 内側の for ループにおいては j に 4, 5, 6 が順に格納されながら以下の処理が行なわれる:

- ▶ $j=4$ の時, $a[3]=7$ と $a[4]=5$ が比較され, $a[3]>a[4]$ なので $a[3]=5$, $a[4]=7$ と入れ替えられる.

array_sort.c の解説 (3)

- ▶ $j=5$ の時, $a[3]=5$ と $a[5]=3$ が比較され, $a[3]>a[5]$ なので $a[3]=3$, $a[5]=5$ と入れ替えられる.
- ▶ $j=6$ の時, $a[3]=3$ と $a[6]=6$ が比較され, $a[4]>a[6]$ ではないので入れ替えはない

となり, $a[3]$, $a[4]$, $a[5]$, $a[6]$ の中で最小の値 (3) が $a[3]$ に格納される (残りの $a[4]$, $a[5]$, $a[6]$ は **どんな順になってもよく**, $i=3$ 以上の要素の中で一番左の **$a[3]$ がそれらより大きくない** ことだけが重要).

ここで紹介した **バブルソート** は, 理解も実装も容易だが, 効率が悪い (配列のサイズが大きくなるにつれ処理時間が著しく増大する) ことで知られている. より効率のよい方法として **ヒープソート**, **マージソート**, **クイックソート** などがある. 自分で色々調べてみよう.

目次

ディレクトリとサンプル・コードの準備

配列を用いた複数のデータの処理

レポート課題

レポート課題 III-4 (1)

レポート課題 III-4 (素数と合成数に分類)

「入力された 15 個以下の任意の個数の正の整数を入力した後、それらを 素数 とそれ以外 (合成数) とに分類し、それぞれを 入力されたのと逆の順 に表示させる」プログラムを作り、その ソースファイル と、以下の 9 個の整数:

83 32 21 19 97 76 63 3 35 51

に対する結果を提出せよ。ただし、下記を満足すること。

- ▶ 提出するファイル名は B5TB9999_prime_reverse.c (ソースファイル) および B5TB9999_prime_reverse.txt (実行結果) とせよ。
- ▶ ソースファイルには適宜 コメント を記入せよ。

(続く)

レポート課題 III-4 (2)

レポート課題 III-4 の仕様 (続き)

- ▶ 15 個未満 で入力を打ち切る場合は 負の値 を入力する.
- ▶ 出力は以下の形式で行なうこと.

素数 (3 個): 7 3 11

合成数 (2 個): 20 8

なお, 上記は, 5 つの整数 11 8 3 7 20 に対する出力である.

- ▶ 今回までの講義で紹介されていない C 言語の機能 (関数など) を使ってもよい. ただし, 該当する部分でどのような処理が行なわれるのかをコメント として記載すること

(さらに続く)

レポート課題 III-4 (3)

レポート課題 III-4 の仕様 (さらに続き)

- ▶ 「入力されたのと逆順」に出力するだけでは物足りない」人は、「素数と合成数のそれぞれを **小さい順** に出力するプログラム」も作成して提出せよ。その場合、ソースファイルと実行結果を、それぞれ、`B5TB9999_prime_sort.c` (ソースファイル) および `B5TB9999_prime_sort.txt` (実行結果) という名前で保存せよ。

提出期限：2015 年 7 月 24 日 (金) (ただし、演習でフォローできるのは 7 月 17 日まで)

レポート III-4 の評価基準 (1)

必須要素

守られていない場合は減点

- ▶ 提出ファイル名 は適切か
- ▶ C 言語ソースファイル と 出力結果 を提出しているか
- ▶ gcc でコンパイルでき、生成したファイルを実行できるか
- ▶ 83 32 21 19 97 76 63 3 35 51 の入力に対して 適切な出力 がされるか
- ▶ 式や処理について 十分なコメント が記載されているか

加点要素 (1)：技術の習得

- ▶ 講義で使っていない機能 の利用
 - ▶ 十分なコメント が付されている場合に限る
 - ▶ 関数, ファイル入出力などの利用
- ▶ 下記のような入力に対しても適切に動作する頑健性
 - ▶ 15 個以上の整数が入力された場合
 - ▶ 0 個の整数が入力された場合 (e.g. 最初に -1 が入力された場合)
 - ▶ 入力された整数が全て素数 (もしくは合成数) の場合

レポート III-4 の評価基準 (2)

加点要素 (2)：創意工夫

- ▶ 指定されていない数値 についての実行結果 (ただし多くても 10 パターン程度まで)
- ▶ 「素数と合成数のそれぞれを **小さい順** に出力するプログラム」を作成できた場合は **大幅加点** (レポート III-1, 2, 3 の出来が悪かった場合, そちらもカバーできるものとする).

レポート課題 III-4 の進め方 (1)

1. `mkdir` コマンドを用いて
~/Documents/report/Report-III/Report-III-4/ というディレクトリを作る (`-p` オプションを使うと便利).
2. Home でのマウス操作や `cp` コマンドなどで
`array_reverse_v2.c` をコピーし, 1. で作ったディレクトリの下に `B5TB9999_prime_reverse.c` という名前で保存する.
3. `cd` コマンドを用いて, 現在の作業ディレクトリを
~/Documents/report/Report-III/Report-III-4/ に変更する.
4. `gedit` を用いて, `B5TB9999_prime_reverse.c` を「入力された整数を素数と合成数に分解して配列に格納」し「それぞれを逆順に表示させる」ように変更する. その方法の **一例** を以下に示す:
 - ▶ まず, 入力できる整数の上限数 (15) を `N_max` に代入

レポート課題 III-4 の進め方 (2)

- ▶ 入力された素数と合成数のそれぞれを格納するための int 型配列 `prime[N_max]`, `composite[N_max]` を定義
 - ▶ 「素数の個数」と「合成数の個数」を格納するための変数 `N_p` と `N_c` を宣言し、それぞれ 0 で初期化しておく.
 - ▶ for ループ内で、キーボードから読み込まれた値を `input` に格納する. `input` が正ならば、それが **素数か合成数か** を判別し、以下の処理を行なう
 - ▶ 素数なら `prime[N_p]` に格納し `N_p` を 1 増やす
 - ▶ 合成数なら `composite[N_c]` に格納し `N_c` を 1 増やす
- `input` が負ならば, `break` を使って for ループを抜け出す
- ▶ for ループを抜け出した時点で、入力された正の整数のうち,
 - ▶ 素数は `prime[0]`, ..., `prime[N_p-1]` に格納
 - ▶ 合成数は `composite[0]`, ..., `composite[N_c-1]` に格納されているはずなので、それぞれを
 - ▶ 素数: `prime[N_p-1]`, ..., `prime[0]`
 - ▶ 合成数: `composite[N_c-1]`, ..., `composite[0]`の順に表示させる.

レポート課題 III-4 の進め方 (3)

5. 「素数の判定」, 「配列のソート」, 「配列の表示」などは **関数** を使うと **スッキリと実装** できる. 興味がある人は自分で調べたり, 教員/TA に相談してみよう.
6. コンパイル時には問題がないのに, プログラム実行時に Segmentation fault と表示されることがある. 本講義の範疇でこのエラーが表示された場合, **存在しない配列の要素** を参照したり, 代入していると思ってよいだろう. 例えば, 以下のプログラム (seg_fault.c)

```
1  #include <stdio.h>          /* 標準入出力ライブラリ */
2  int main(void)              /* main 関数の引数と戻り値の定義 */
3  {                            /* main 関数の始まり */
4      int a[3];               /* 3個の要素を持つ配列 */
5      int N=1000;
6      printf("%d\n", a[N]);    /* 存在しない要素 a[1000] を参照 */
7      a[N] = 10;              /* 存在しない要素 a[1000] に10を代入 */
8
9      return 0;               /* 戻り値として 0 を返す */
10 }
```

レポート課題 III-4 の進め方 (4)

では, 3 個の要素しか持たない配列 `a[3]` を宣言した後, 存在しない要素 `a[1000]` を参照したり, `a[1000]` を代入しようとしている. 配列を使う場合は「存在しない配列の要素を指定していないか」を **常にチェックする** ように心がけよう.