アスキーコード表
(ASCII code)

日本語は通常2バイトの文字コード．JISコード，シフトJISコード，Unicode (UTF-8)等の様々な文字コードがある．

漢字変換無しでキーボードから直接入力できる半角文字

復習

文字コード

1文字＝1バイト
⇒ char型変数

アスキーコード(値)											
32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	”	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	¥	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

この授業では1バイトのアスキーコードのみを学習する

この授業では日本語文字のプログラムは扱わない

復習

C言語における文字の取り扱い(2)

文字の出力

```
char moji1, moji2, xx;
moji1 = 65;
moji2 = 'A';
xx = '0';
printf("moji1は%d, moji2は%d, xxは%dである. ¥n", moji1, moji2, xx);
printf("moji1は%c, moji2は%c, xxは%cである. ¥n", moji1, moji2, xx);
```

```
moji1は65, moji2は65, xxは48である.
moji1はA, moji2はA, xxは0である.
```

変換文字%cは文字コードを文字に変換する

文字の入力

```
char aa;
printf("文字を一つ入れてください:");
scanf("%c", &aa);
printf("ASCIIコードは%d, 文字は%cである. ¥n", aa, aa);
```

変換文字%cは入力された文字の文字コードを変数に代入する

```
文字を一つ入れてください:1
ASCIIコードは49, 文字は1である.
```

復習

C言語における文字列の取り扱い(1)

文字

a　A　b　B　z　Z　0　1　9　=　+　?　/　!　など

文字列

文字列は文字の集合

Hello　Kandai　programming　など

理由：文字数より要素数の多い配列を用いた時に，文字列の最後を示すため

原則：C言語では文字列をchar型配列で扱う

文字列Helloを文字配列s[6]に入れる場合

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
↑	↑	↑	↑	↑	↑
'H'	'e'	'l'	'l'	'o'	
72	101	108	108	111	0

必ず最後にはコード0が入る

文字数プラス1の要素数が必要

復習

文字列(文字配列)の初期化

s[6]～s[9]は現在使っていないという目印

文字列の設定

必要な文字数より多めに宣言する

最後に0を入れる

```
char s[10];
s[0] = 'H'; s[1] = 'e'; s[2] = 'l'; s[3] = 'l'; s[4] = 'o'; s[5] = 0;
```

配列 s[10] ⇒

H	e	l	l	o	0	%	&	*	#

ゴミ！　初期化されてない不定な値

```
char s[10] = {'H', 'e', 'l', 'l', 'o', 0};
```

配列初期化の応用

It's New!

```
char s[10] = "Hello";
```

配列宣言と同時に文字列を設定する方法
（最後の0も自動的に入る）

```
char s[] = "Hello";
```

char s[6] = "Hello" と同じ

```
char s[10];
s[10] = "Hello";
s[0] = "Hello";
s[ ] = "Hello";
```

配列宣言と同時でなくてはダメ！
（宣言文と代入文に分かれてはダメ）

文字列の入出力

復習

aa[0], aa[1], ・・・
⇒インデックスを付けると一つ一つの文字を表す
aa
⇒ インデックスを付けないときはかたまりとしての文字列を表す

文字列の出力

```
char aa[] = "KANDAI";
printf("私は%s生です. ¥n", aa);
```

変換文字は%s

注意！ [ ]を付けない

```
私はKANDAI生です.
```

文字列の入力

必要な文字数より多めに宣言

注意！ &も[ ]も付けない

最後の0も自動的に入る

```
char aa[100];
printf("99文字以下で文字列を入力してください:");
scanf("%s", aa);
printf("文字列は%sである. ¥n", aa);
```

変換文字は%s

文字配列aaに入力した文字列が代入される

```
99文字以下で文字列を入力してください:Kandai
文字列はKandaiである.
```

本来はループを回す必要がある

文字列を操作するライブラリ関数の例

ライブラリ関数	意味
int strlen(char s[])	文字列sの文字数を返す
char* _strset(char s[], int c)	文字列sをアスキーコードcの文字で埋める
char* strcpy(char s1[], char s2[])	文字列s2を文字列s1にコピーする
char* strcat(char s1[], char s2[])	文字列s1の末尾に文字列s2を付加する
int strcmp(char s1[], char s2[])	文字列s1と文字列s2を比較する. 同じ文字列なら0を返す. 辞書の順序で s1がs2より前なら, 正の値を返す. s1がs2より後なら, 負の値を返す.

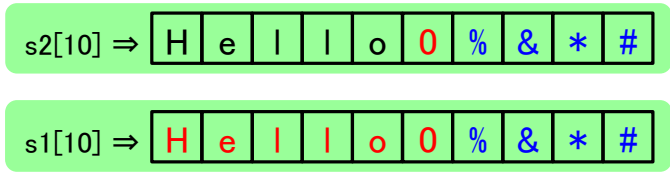
char*はポインタ. 2年生以降で学習

本来はループ処理をする必要がある

```
if (s1 == s2)
    ・・・・・・
```

文字列を==演算子で比較することはできない

これらの文字列操作用ライブラリ関数を利用するには
#include <string.h>
が必要.



注1） 関数の表記は, 現在の学習レベルに合わせて変更してある.
注2） 関数については第10回以降で詳しく学習.

文字列操作ライブラリを用いたソースプログラムの例

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char ss1[] = "Kandai-sei", ss2[] = "Computer Science";
    char ss3[100] = ""; //初めは空の文字列．十分な文字数を確保する．

    printf("ss1の文字数は%dですが，ss3の文字数は%dです．¥n",
                                                strlen(ss1), strlen(ss3));
    strcpy(ss3, ss1);       //ss1の内容をss3にコピー
    printf("ss3の内容は%sになり，文字数は%dになりました．¥n",
                                                ss3, strlen(ss3));
    strcat(ss3, " / ");     //ss3の最後に" / "を付加
    strcat(ss3, ss2);       //ss3の最後にss2を付加
    printf("今度のss3は%sで，文字数は%dです．¥n", ss3, strlen(ss3));
}
```

配列 ss3[100] ⇒ | 0 | $ | x | # | ¥ | Y | % | & | * | # | ・・・ |

ゴミ

空の文字列として初期化することは重要！

配列なので，ss3=ss1という代入はできない

ss3 | K | a | n | d | a | i | - | s | e | i | 0 | ＋ | | / | | 0 | ＋ ss2 | C | o | m | p | u | t | e | r | | S | c | i | e | n | c | e | 0 |

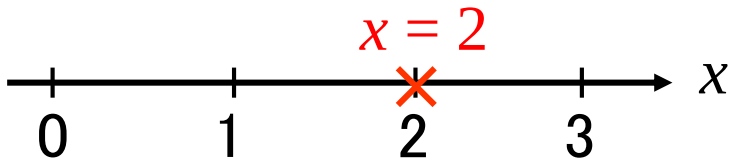
ss3 | K | a | n | d | a | i | - | s | e | i | | / | | C | o | m | p | u | t | e | r | | S | c | i | e | n | c | e | 0 |

```
ss1の文字数は10ですが，ss3の文字数は0です．
ss3の内容はKandai-seiになり，文字数は10になりました．
今度のss3はKandai-sei / Computer Scienceで，文字数は29です．
続行するには何かキーを押してください．．．
```

1次元から2次元へ

数学では

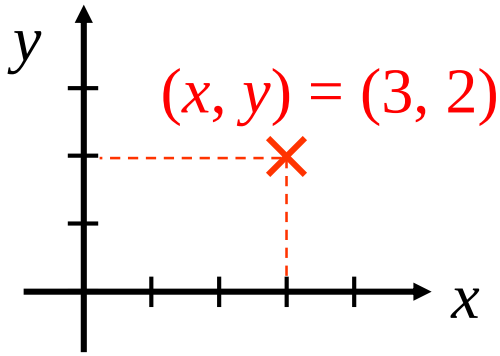
1次元座標



直線 x

1つの変数xで位置が定まる

2次元座標



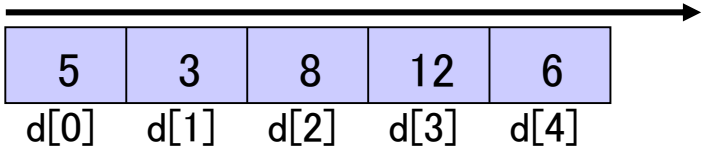
平面 (x, y)

2つの変数xとyで位置が定まる

プログラミングでは

1次元

```
int d[5] = { 5, 3, 8, 12, 6 };
```

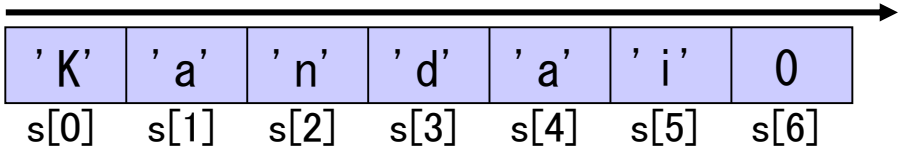


1次元配列

配列 d[i], s[i]

1つのインデックスiでデータが定まる

```
char s[7] = "Kandai";
```



2つのインデックスでデータが定まる配列
⇒ 2次元配列

2次元配列

```
float d[3][4];
```

2次元配列 d[i][j]
⇒ 2つのインデックスiとjでデータが指定される

縦が3行　横が4列　のfloat型の表

横のインデックスは3まで

4列

	j = 0	j = 1	j = 2	j = 3
i = 0	d[0][0]	d[0][1]	d[0][2]	d[0][3]
i = 1	d[1][0]	d[1][1]	d[1][2]	d[1][3]
i = 2	d[2][0]	d[2][1]	d[2][2]	d[2][3]

3行

縦のインデックスは2まで

d[0][4],
d[1][4],
...
d[3][0],
...
d[3][4]
等々は
存在しない

2次元配列の初期化(1)

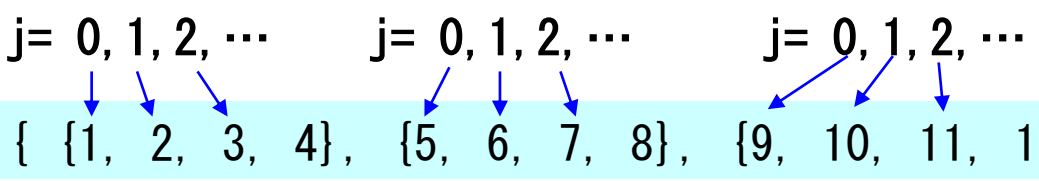
1次元

```
int b[5] = {10, 20, 25, 35, 40};
```

これと同じ意味

```
b[0] = 10;  b[1] = 20;  b[2] = 25;  b[3] = 35;  b[4] = 40;
```

2次元



これと同じ意味

	j = 0	j = 1	j = 2	j = 3
i = 0	d[0][0] = 1	d[0][1] = 2	d[0][2] = 3	d[0][3] = 4
i = 1	d[1][0] = 5	d[1][1] = 6	d[1][2] = 7	d[1][3] = 8
i = 2	d[2][0] = 9	d[2][1] = 10	d[2][2] = 11	d[2][3] = 12

2次元配列の初期化(2)

```
int aa[][] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

```
int aa[][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

縦の行数は省略可

横の列数は省略不可

⇒ 使用予定の最大の列数を指定する

```
int aa[][4] = { {1, 2, 3, 4}, {5, 6}, {9, 10, 11} };
```

	j = 0	j = 1	j = 2	j = 3
i = 0	d[0][0] = 1	d[0][1] = 2	d[0][2] = 3	d[0][3] = 4
i = 1	d[1][0] = 5	d[1][1] = 6	d[1][2] = 0	d[1][3] = 0
i = 2	d[2][0] = 9	d[2][1] = 10	d[2][2] = 11	d[2][3] = 0

この部分は0に初期化される

注意
```
int aa[3][4];
```
のように初期化しない配列の値は未定義

初期化は宣言と同時に！

```
int aa[3][4];
aa[][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

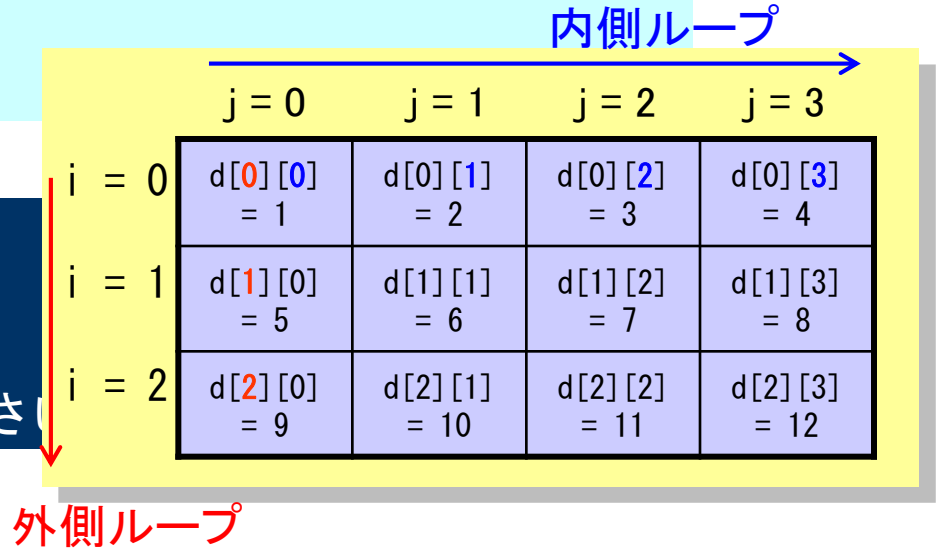
2次元配列を用いたソースプログラムの例(1)

```
#include    <stdio.h>
main()
{
    int aa[][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            printf("%d  ", aa[i][j]);
        }
        printf("¥n");
    }
}
```

内側ループ

外側ループ

```
1   2   3   4
5   6   7   8
9   10   11   12
続行するには何かキーを押してくださ
```



2次元配列を用いたソースプログラムの例(2)

改良型

```
#include    <stdio.h>
main()
{
    int aa[][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            printf("%2d  ", aa[i][j]);
        }
        printf("¥n");
    }
}
```

修正点
%d ⇒ %2d

縦にきれいにそろって出力！

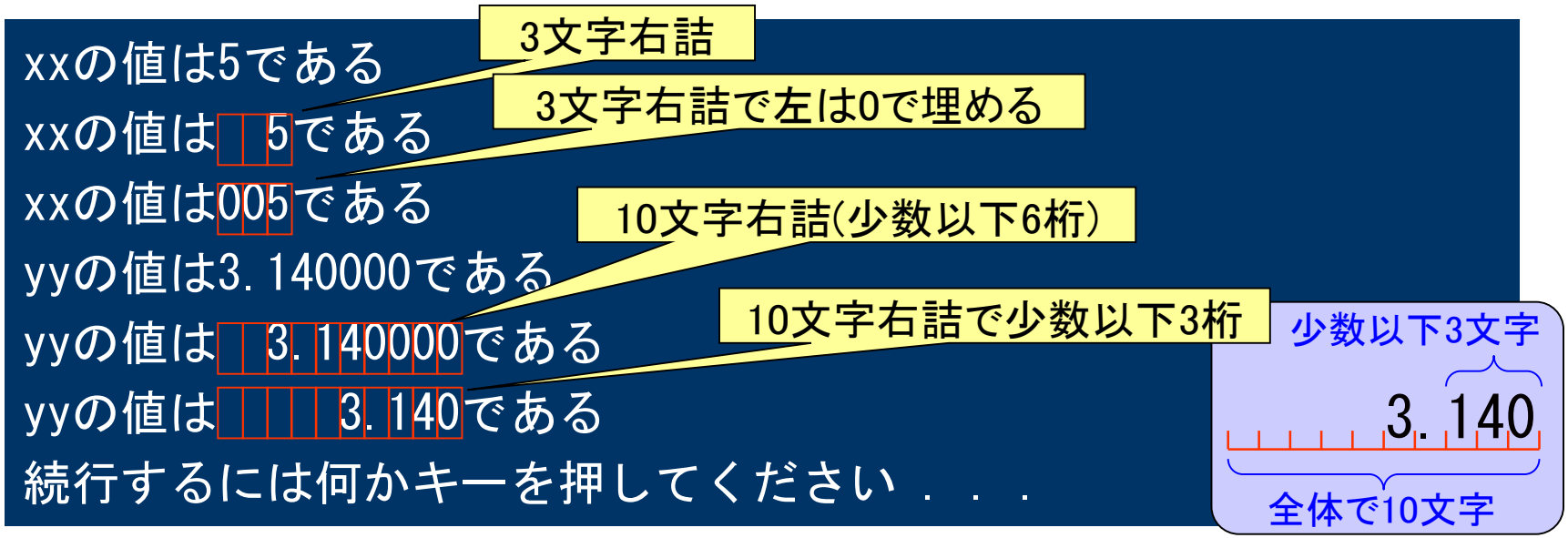
```
 1   2   3   4
 5   6   7   8
 9  10  11  12
続行するには何かキーを押してくださ
```

	j = 0	j = 1	j = 2	j = 3
i = 0	d[0][0] = 1	d[0][1] = 2	d[0][2] = 3	d[0][3] = 4
i = 1	d[1][0] = 5	d[1][1] = 6	d[1][2] = 7	d[1][3] = 8
i = 2	d[2][0] = 9	d[2][1] = 10	d[2][2] = 11	d[2][3] = 12

printf()関数における書式指定

```
#include    <stdio.h>
main()
{
    int     xx = 5;
    float   yy = 3.14;
    printf("xxの値は%dである¥n", xx);
    printf("xxの値は%3dである¥n", xx);        //全体で3文字の範囲に右詰
    printf("xxの値は%03dである¥n", xx);       //同上に加えて，0を詰める
    printf("yyの値は%fである¥n", yy);
    printf("yyの値は%10fである¥n", yy);       //全体で10文字の範囲に右詰
    printf("yyの値は%10.3fである¥n", yy);     //同上に加えて，小数以下3桁
}
```

```
xxの値は5である
xxの値は  5である
xxの値は005である
yyの値は3.140000である
yyの値は  3.140000である
yyの値は     3.140である
続行するには何かキーを押してください . . .
```



2次元配列を用いたソースプログラムの例(3)

改良型:縦の合計を出力

```
#include    <stdio.h>
main()
{
    int aa[][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
    int i, j, sum = 0;
    for (i = 0; i < 3; i++)      // 配列内容を出力
    {
        for (j = 0; j < 4; j++)
        {
            printf("%2d  ", aa[i][j]);
        }
        printf("¥n");
    }
    printf("==============¥n");
    for (j = 0; j < 4; j++)      // 縦の合計を出力
    {
        sum = 0;
        for (i = 0; i < 3; i++)
        {
            sum = sum + aa[i][j];
        }
        printf("%2d  ", sum);
    }
    printf("¥n");
}
```

d[0][0] = 1	d[0][1] = 2	d[0][2] = 3	d[0][3] = 4
d[1][0] = 5	d[1][1] = 6	d[1][2] = 7	d[1][3] = 8
d[2][0] = 9	d[2][1] = 10	d[2][2] = 11	d[2][3] = 12

合計を求める
内側ループ

合計を求める
外側ループ

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
==============
15 18 21 24
続行するには何かキーを. . .
```

2次元配列を用いたソースプログラムの例(4)

データの入力

```
#include    <stdio.h>
main()
{
    int aa[3][4];
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            printf("aa[%d][%d]はいくら?", i, j);
            scanf("%d", &aa[i][j]);
        }
    }
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            printf("%3d ", aa[i][j]);
        }
        printf("¥n");
    }
}
```

&が必要

！注意！
&が不要なのは文字列を%sで入力する場合だけ！

d[0][0] = 1	d[0][1] = 2	d[0][2] = 3	d[0][3] = 4
d[1][0] = 5	d[1][1] = 6	d[1][2] = 7	d[1][3] = 8
d[2][0] = 9	d[2][1] = 10	d[2][2] = 11	d[2][3] = 12

```
aa[0][0]はいくら?1
aa[0][1]はいくら?2
aa[0][2]はいくら?3
aa[0][3]はいくら?4
aa[1][0]はいくら?5
aa[1][1]はいくら?6
aa[1][2]はいくら?7
aa[1][3]はいくら?8
aa[2][0]はいくら?9
aa[2][1]はいくら?10
aa[2][2]はいくら?11
aa[2][3]はいくら?12
  1   2   3   4
  5   6   7   8
  9  10  11  12
続行するには何か. . .
```

バグの種類

コンパイルエラー

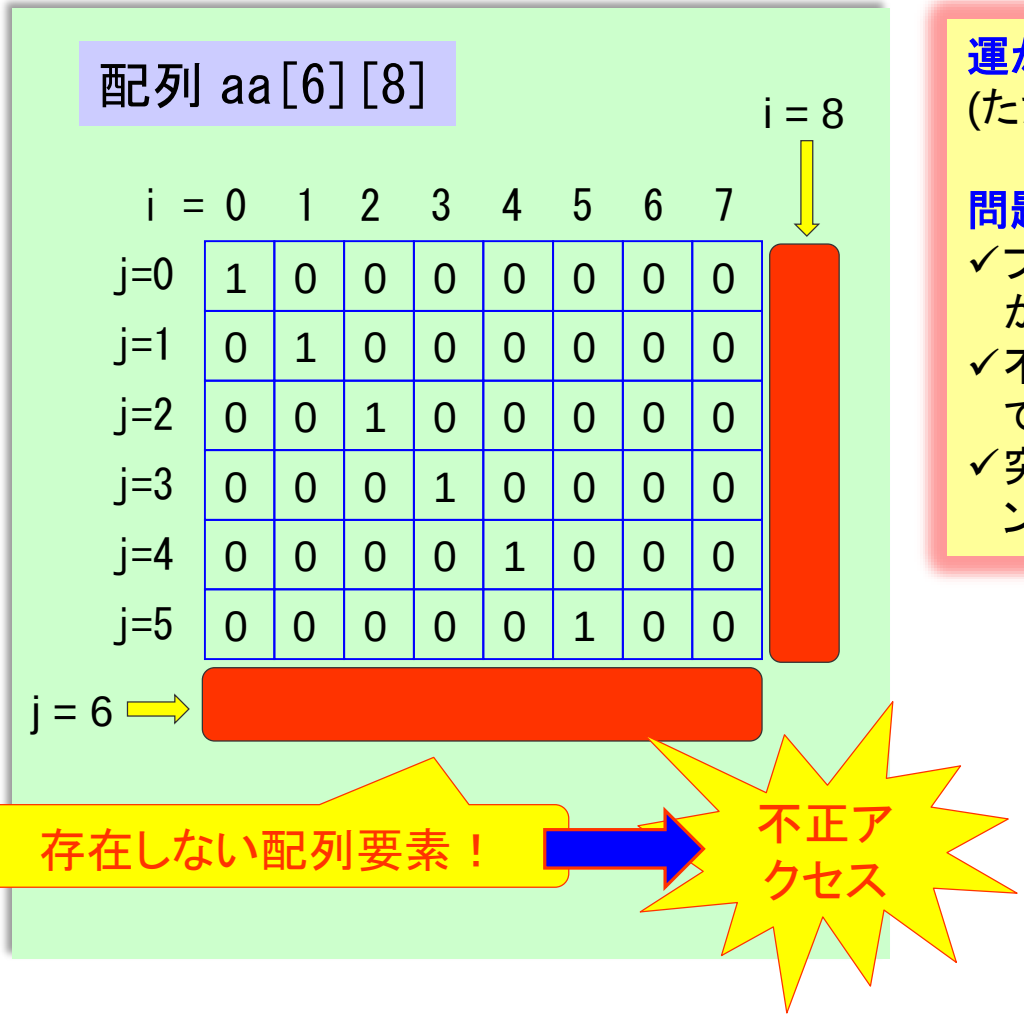
- プログラミング言語の文法上の間違い
- コンパイラが指摘してくれる
- プログラムを実行できない
- デバッガは使えない
- プログラミング言語の文法を正しく理解していればデバッグは簡単

コンパイラが指摘するエラーの場所がずれていることもあり

実行時エラー

- プログラムの論理的な間違い
- コンパイラは指摘できない
- プログラムを実行できる
  ⇒ただし正常動作しない
- デバッガを使える
- デバッグが難しい

典型的な実行時エラー：メモリの不正アクセス



運が良い時 ⇒ 何も問題は起きない
(ただし正解と一致したのは偶然)

問題が起きるときの症状

- ✓プログラムが途中で停止しているが，一見正常終了
- ✓不正アクセス等のエラーメッセージで終了
- ✓突然，コンソールウィンドウ(黒ウィンドウ)が閉じる

VS2019情報
不正アクセスを起こして終了した場合は，コード0以外のコードが示される