プログラミングI
数理物理，総合理学等向け

2018年11月5日

海谷 治彦

目次

- 9章[レ] 配列 (ただし数値のみ)
- 配列の宣言，代入，利用
- 配列の初期化
- 多次元配列
- Ring buffer
- フラグによる制御 , の表示等
- キャスト (配列とは関係ない話)

- 文字の配列

- 今回の演習

配列への動機付け

p.109-
(明)

- 連続した同種のデータ群を扱う計算はそこらじゅうに出てくる.
  - 学生の成績リスト，株価の変動リスト等
- 加えて，このデータ群を繰り返し(while)で処理すると，すごく，すっきりしている匂いがする.

- 配列は，こういった同種のデータを扱うための，変数の一種である.

List 9-1

```
// list9-1a.c
#include <stdio.h>

int main(void){
  float kokugo, sugaku, eigo; // バラで定義するのめんどい
  float heikin;

  kokugo=65.0;
  sugaku=90.0;
  eigo=75.0;
  heikin = (kokugo+sugaku+eigo)/3.0;

  printf("Japanese %0.0f, Math %0.0f, English %0.0f¥n",
                    kokugo, sugaku, eigo);
  printf("Average %f¥n", heikin);
  return 0;
}
```

```
sh-3.1$ cc l9-1.c
sh-3.1$ ./a.exe
Japanese 65, Math 90, English 75
Average 76.666664
sh-3.1$
```

配列の基本

- 配列は変数の一種である.
  - 宣言する必要あり.
  - 値を読む(使う)ことができる.
  - 値を書く(更新する)ことができる.
- 通常の変数と異なり，同種の値を複数個扱うことができる.
  - 1個でもいいが，あんまり意味がない.
- 同じ変数名で扱う値群を区別するために，添字という0から始まる番号を用いる.
  - 値を読む場合，書く場合に，変数名に加えて，この添字を指定する必要がある.
- 具体例は，次頁の List 9-2 改を参照.

List 9-2 改

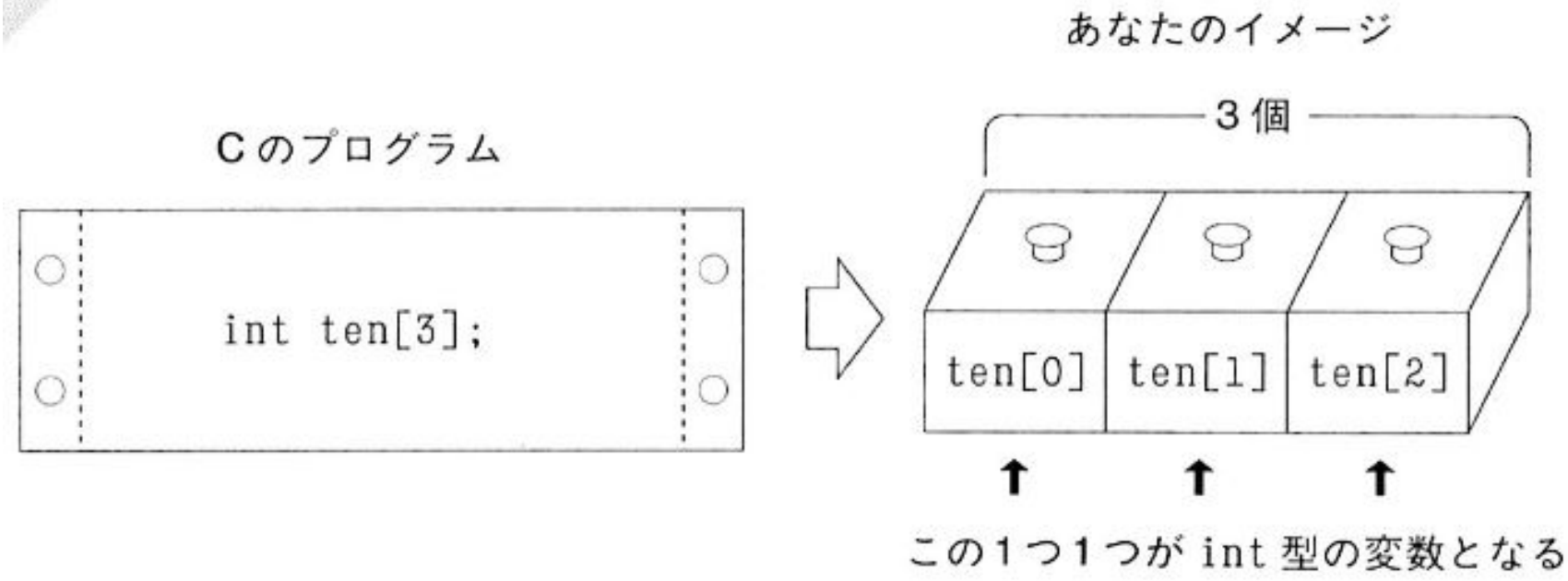
```
// list9-2b.c
#include <stdio.h>

int main(void){
  float ten[3]; // 配列の定義
  float heikin;

  ten[0]=65.0; // 配列のアクセス 添字の初期値はゼロ！
  ten[1]=90.0;
  ten[2]=75.0;
  heikin = (ten[0]+ten[1]+ten[2])/3.0;

  printf("Japanese %.0f, Math %.0f, English %.0f¥n",
                    ten[0], ten[1], ten[2]);
  printf("Average %f¥n", heikin);
  return 0;
}
```

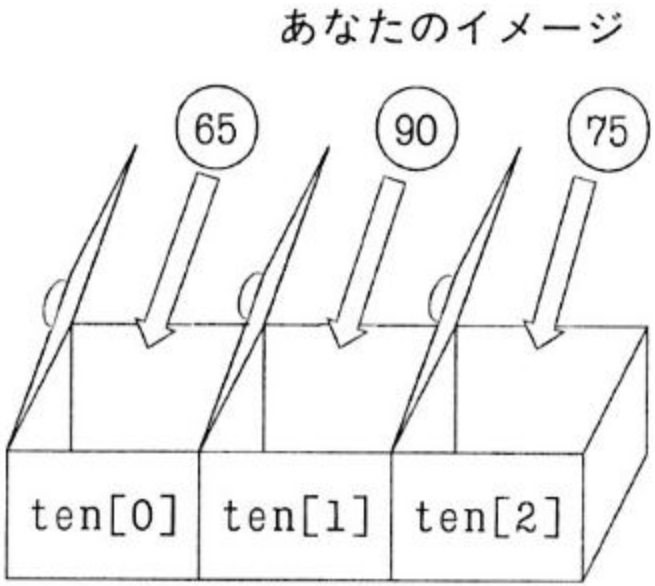
配列定義のイメージ Fig 9-2



配列要素への代入 Fig 9-3

Cのプログラム

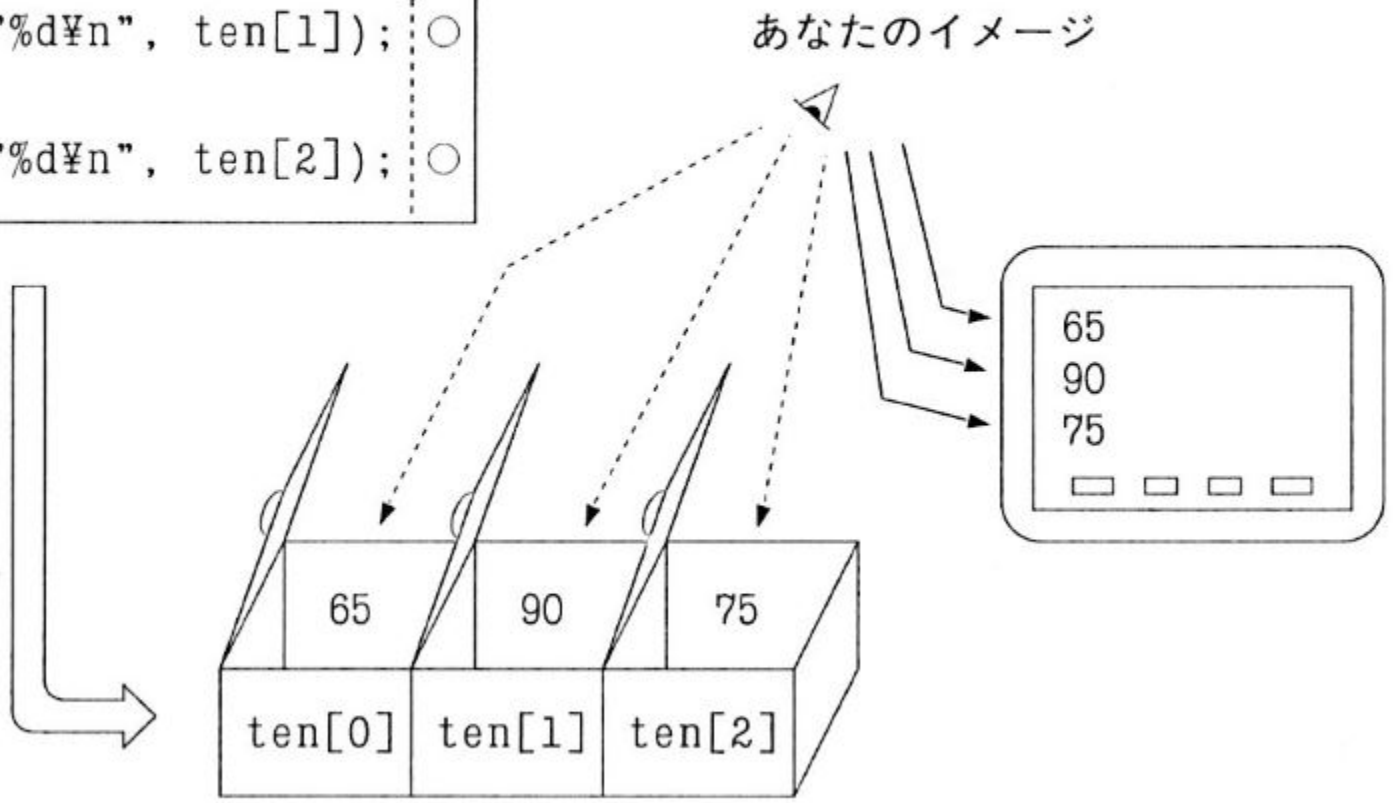
```
ten[0] = 65;
ten[1] = 90;
ten[2] = 75;
```



配列要素の参照 Fig 9-4

Cのプログラム

```
printf("%d¥n", ten[0]);
printf("%d¥n", ten[1]);
printf("%d¥n", ten[2]);
```



(箱の中身が消えるわけではない)

添字を変数で指定できる

- 配列の最大の利点は，添字を変数で指定できることである.
- これによって，配列への操作を繰り返し(while)と組み合わせて利用できる.

List 9-3

```
#include <stdio.h>

int main(void){ // list9-3a.c
  float ten[3]; // 配列の定義
  int i=0;
  float heikin, sum=0;

  ten[0]=65; // 配列のアクセス 添字の初期値はゼロ！
  ten[1]=90;
  ten[2]=75;
  while(i<3){
    sum += ten[i]; // 添字に変数が使える
    i++;
  }
  heikin = sum/3.0;

  printf("Japanese %0.f, Math %0.f, English %0.f¥n",
                    ten[0], ten[1], ten[2]);
  printf("Average %f¥n", heikin);
  return 0;
}
```

配列の初期値設定

- 配列も宣言(定義)の時点で，初期値を明示的に設定できる.
- 普通の変数と同様に，初期値を設定しない場合，最初に入っている値は不定である.
- 初期値の値列は{}で囲んで，，で区切る.
- 初期値を宣言時点で行なう場合，配列の要素数([]内の数)は省略できる.

List 9-3 配列の初期値

```
#include <stdio.h>

int main(void){ // list9-3c.c
  float ten[3]={65.0, 90.0, 75.0}; // 配列の定義と初期化
  int i=0;
  float heikin, sum=0;

  while(i<3){
    sum += ten[i]; // 添字に変数が使える
    i++;
  }
  heikin = sum/i;

  printf("Japanese %0.f, Math %0.f, English %0.f¥n",
                    ten[0], ten[1], ten[2]);
  printf("Average %f¥n", heikin);
  return 0;
}
```

List 9-3 配列要素数を略

```
#include <stdio.h>

int main(void){ // list9-3d.c
  float ten[]={65.0, 90.0, 75.0}; // 配列の定義と初期化
  int i=0;
  float heikin, sum=0;

  while(i<3){
    sum += ten[i]; // 添字に変数が使える
    i++;
  }
  heikin = sum/i;

  printf("Japanese %0.f, Math %0.f, English %0.f¥n",
                    ten[0], ten[1], ten[2]);
  printf("Average %f¥n", heikin);
  return 0;
}
```

二次元配列

p.124-
(明)

- 添え字を2個以上もつ二次元，もしくはそれ以上の次元の配列を扱うことができる.
- 通常の配列が直線上の格子にデータがあるのに対して，
- 二次元配列の場合，平面上の格子，
- 三次元配列の場合，空間上の格子

にデータがあるイメージとなる.

- 多次元にしたほうが便利な場合を除き，数回後で話す構造体を用いられるほうが多いと思われる.

ノーマルな例

- 次頁にもうちょっと一般的な例を示す.

- 簡単な会計プログラム
- 三種類の品目(食費 交通費 医療費)
- 5ヶ月分の実測値

- まぁ, エクセルでやったほうが早いけど, こーいうプログラムもかけるよ, という例.

シンプルな二次元配列の例

```
// account.c 簡単な家計簿的なもの
#include <stdio.h>
int main(void){
  // {Jan, Feb. Mar. Apr, May} × {食費 交通費 医療費}
  int account[5][3]={
    {30000, 10000, 1000},
    {25000, 20000, 10000},
    {35000,  1000, 0},
    {40000, 11000, 2000},
    {30000, 30000, 15000}
  };
  int total[3]={0, 0, 0}; // 合計とか
  int i=0, j=0;
  printf("%8s %9s %9s %9s¥n", "Month:", "food", "transport", "medical");
  while(i<5){
    printf("Month #%d ", i+1);
    j=0;
    while(j<3){
      printf("%9d ", account[i][j]);
      total[j] += account[i][j]; // 品目別の合計値を累積している
      j++;
    }
    i++;
    printf("¥n");
  }
  printf("total:   ");
  j=0;
  while(j<3) {printf("%9d ", total[j]); j++;}
  printf("¥n");
  return 0;
}
```

```
sh-3.1$ ./a.exe
  Month:      food transport   medical
Month #1     30000     10000      1000
Month #2     25000     20000     10000
Month #3     35000      1000         0
Month #4     40000     11000      2000
Month #5     30000     30000     15000
total:      160000     72000     28000
```

注意: 配列の代入と参照

- 配列同士の丸ごと全部の代入はできない.

```
int a[5]={3, 1, 4, 1, 5};
int b[5];
   b = a; // ダメ
```

- 配列の要素を丸ごと参照することはできない.

```
int a[5]={3, 1, 4, 1, 5};
  printf("%d", a); // ダメ というか意味が違う
```

配列を表示

- あまり効率的な表示方法は無い.
- 以下のようにwhileと組み合わせるのがお勧め.

```
// arrayprint.c
#include <stdio.h>

int main(void){
int a[]={3, 1, 4, 1, 5, 9, 2};
int i=0;
  while(i<7){
    printf("%d ", a[i]);
    i++;
  }
  printf("¥n"); // 最後に改行
  return 0;
}
```

```
bash-3.1$ cc arrayprint.c
bash-3.1$ ./a
3 1 4 1 5 9 2
bash-3.1$
```

配列に入力 1/2

- 効率的な入力方法は本授業の範囲外.
- 以下のように並べて書くのも手だが汎用性が無い.

```
// arrayscan1.c
#include <stdio.h>

int main(void){
int a[5];
int i=0;
  scanf(“%d %d %d %d %d”, &a[0], &a[1], &a[2], &a[3], &a[4]);

  while(i<5){
    printf("%d, ", a[i]);
    i++;
  }
  printf("¥n");
  return 0;
}
```

```
bash-3.1$ cc arrayscan1.c
bash-3.1$ ./a
3 1 3 1 5
3, 1, 3, 1, 5,
bash-3.1$
```

配列に入力 2/2

```
// arrayscan1.c
#include <stdio.h>

int main(void){
int a[5];
int i=0;
  while(i<5){
    scanf(“%d”, &a[i]);
    i++;
  }

  i=0;
  while(i<5){
    printf("%d, ", a[i]);
    i++;
  }
  printf("¥n");
  return 0;
}
```

- 左のように，whileで読むのもよいが，データ毎に改行しないといけない．
- この授業の範囲では，この方法がお勧め．

```
bash-3.1$ cc arrayscan2.c
bash-3.1$ ./a
3
1
4
1
5
3, 1, 4, 1, 5,
bash-3.1$
```

フラグによる制御

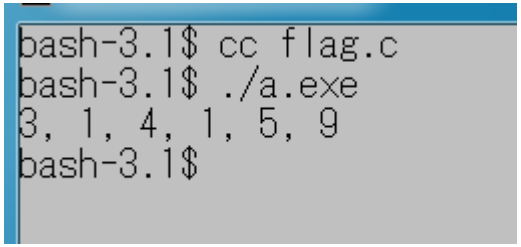
- 「1, 2, 3, 4, 5, 」みたいに，最後にも「,」がつくのが気に入らない人もいるかもしれない.
- これを防止して，「1, 2, 3, 4, 5」みたいにする一般的な方法としてフラグを使う方法がある.
- 入力の状態をフラグとしてもち，最初の入力か否かで表示を変更する.
- 最後の入力はいつかわからないので，最初を区別するのが一般的.

- 正直，プログラムはあまりエレガントでない.

サンプル

```
// flag.c
#include <stdio.h>

int main(void){
int i=0, data[]={3, 1, 4, 1, 5, 9};
int isBegin=1; // 最初の要素かどうかを記憶するフラグ
  while(i<6){
    if(isBegin){ // 次は最初じゃないからフラグをリセット
      isBegin=0;
    }else{ // 最初の要素じゃなかったら直前の要素があるから", "を書く
      printf(", ");
    }
    printf("%d", data[i]);
    i++;
  }
  printf("¥n");
  return 0;
}
```



履歴の記憶 ring buffer

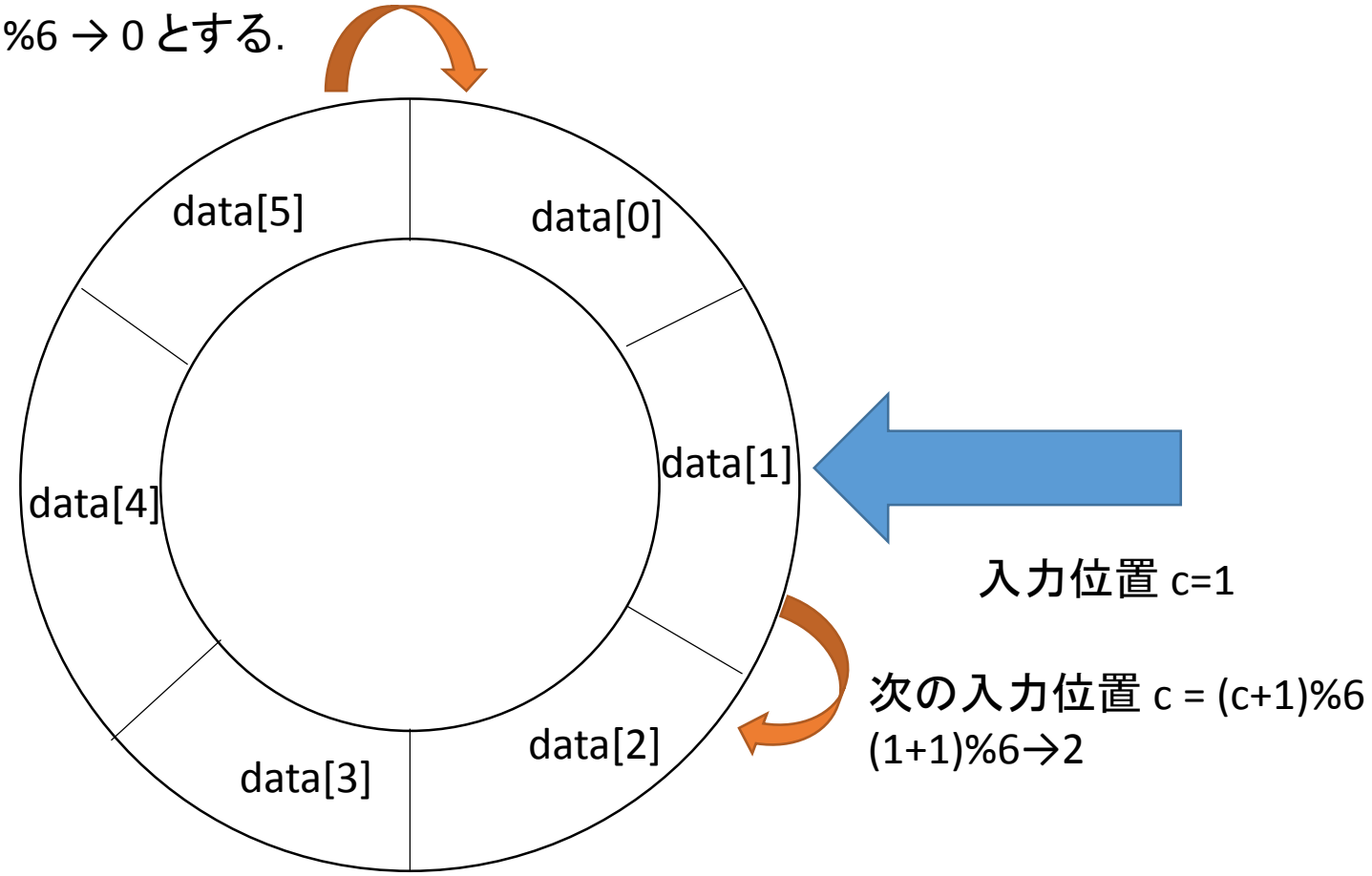
```
// ringbuf.c
// 直近の6個の入力値を記憶するプログラム
// 負の数を入力して終わる
#include <stdio.h>

int main(void){
int data[]={0, 0, 0, 0, 0, 0}, num=6;
int c=0;
  while(1){
  int v;
    scanf("%d", &v);
    if(v<0){
      break;
    }
    data[c]=v;
    {
    // dataの内容を表示
    int i=0;
      while(i<num){
        printf("%d", data[i]);
        if(i==c){printf("+ ");} else {printf(", ");}
        i++;
      }
      printf("¥n");
    }
    c=(c+1)%num;
  }
  return 0;
}
```

```
bash-3.1$ cc ringbuf.c
bash-3.1$ ./a
3
3+ 0, 0, 0, 0, 0,
1
3, 1+ 0, 0, 0, 0,
4
3, 1, 4+ 0, 0, 0,
1
3, 1, 4, 1+ 0, 0,
5
3, 1, 4, 1, 5+ 0,
9
3, 1, 4, 1, 5, 9+
2
2+ 1, 4, 1, 5, 9,
6
2, 6+ 4, 1, 5, 9,
5
2, 6, 5+ 1, 5, 9,
-1
bash-3.1$
```

Ring buffer のイメージ

5+1 → 6 だと，配列上限からあふれるので，
(5+1)%6 → 0 とする．



強制型変換 キャスト p.218[レ] p.34[明]

- 配列とは関係無い話です.
- 演算は基本，同じ型同士でないと成立しません.
- よって，整数値の平均を計算する場合も，小数点部分は切り捨てられてしまいます.
- C言語およびその他の言語では，型を強制的に変換する文法が備わっており，キャストとよばれています.

- 次頁には int を float に強制的に変換する文があります.

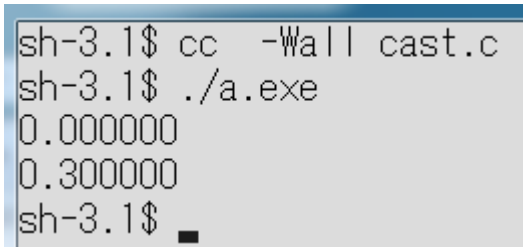
Castの例

```
// cast.c
#include <stdio.h>

int main(void){
int a=3, b=10;
float div;
  // a/bはintなので切り捨てでゼロに
  div = a/b;
  printf("%f¥n", div);

  // floatに変換するので切り捨てられない
  div = (float)a/b;
  printf("%f¥n", div);

  return 0;
}
```



文字の配列

文字列

文字型の配列 文字列

- 単語や文は文字型charの配列で扱うことができる.
- intやfloatの配列と違い，単語や文の終端を表す文字が設定する. (意図的にしないことも可能)

  '¥0'
  - データとしては，全てのビットがゼロのデータ.
- '¥0' の値自体が，while, if 等の条件判断における偽(false)の値となっている.
  - そもそもCでの条件判断は，判断対象の式が，ゼロかそれ以外かを判定しているに過ぎない.
    - ゼロの場合「成り立たない」，それ以外は「成り立つ」と判断.
- 「文字型の配列」のことを単に「文字列」と呼ぶことも多い.

List 9-5

```
// list9-5a.c
#include <stdio.h>

int main(void){
  int i=0;
  char a[7];
  a[0]='H';
  a[1]='e';
  a[2]='l';
  a[3]='l';
  a[4]='o';
  a[5]='¥n';
  a[6]='¥0';

  while(a[i]!='¥0') {printf("%c", a[i]); i++;}
  // printf("%s", a); // 上記と同じ意味
  return 0;
}
```

配列の初期値設定

- 配列の初期値設定を代入文でちまちまやると，結構めんどうくさい.
  - 前ページの例のとおり.
- 宣言時点で初期値を設定することができる.
- 特に文字型の配列(文字列)の場合，"aaa" 等の表現を使えるのでとても楽.
  - これは文字型 char だけの特例で，int や float では，このようには書けない.

左と下は同じ意味

```
// list9-5b.c
#include <stdio.h>

int main(void){
  int i=0;
  char a[7]={'H', 'e', 'l', 'l', 'o', '¥n', '¥0'};

  while(a[i]) {// a[i]!='¥0' と同じ
    printf("%c", a[i]); i++;
  }
  return 0;
}
```

```
// list9-5c.c
#include <stdio.h>

int main(void){
  char a[]="Hello¥n"; // こう書くほうが一般的

  printf("%s", a);
  return 0;
}
```

二次元配列と文字列

- 文字列自体が文字の一次元配列である.
- よって，文字列の配列は，文字の二次元配列で扱うことになる.
- 文字列の配列が扱えると，プログラムの見た目のバリエーションが増やせて嬉しい.

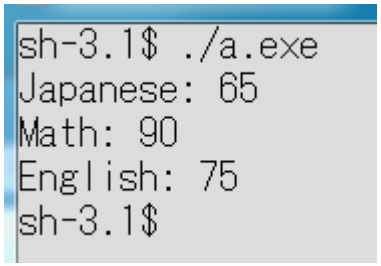
List 9-6 改

```
// 本講義では日本語処理は扱いません
// list9-6x.c
#include <stdio.h>

#define MAX_TEN 3

int main(void){
  int i=0;
  int ten[MAX_TEN]={65, 90, 75};
  char name[MAX_TEN][10]={"Japanese", "Math", "English"};
                              // 8, 4, 7文字

  while(i<MAX_TEN){
    printf("%s: %d¥n", name[i], ten[i]);
    i++;
  }
  return 0;
}
```



nameのイメージ

```
char name[MAX_TEN][10]={"Japanese", "Math", "English"};
                                    // 8, 4, 7文字
```

第二の
添え字

添字	0	1	2	3	4	5	6	7	8	9
0	**J**	**a**	**p**	**a**	**n**	**e**	**s**	**e**	**¥0**	**未使用**
1	**M**	**a**	**t**	**h**	**¥0**	**未使用**	**未使用**	**未使用**	**未使用**	**未使用**
2	**E**	**n**	**g**	**l**	**i**	**s**	**h**	**¥0**	**未使用**	**未使用**

第一の
添え字

特殊文字 (一部復習)

- C言語等では表示しにくい特殊文字を

  ¥n

等の形式で表現し，文字型の変数に格納できる.

- 内部的には 8bitのデータ列のどれかに対応する.

- ¥n 改行
- ¥r キャリッジリターン 行頭に戻る制御文字
- ¥0 全ての値がゼロの文字
- ¥t タブ

余談: ¥n と ¥r の歴史的違い

- 由来はタイプライターの時代までさかのぼる.
- タイプライターは,
  - 次の行にいく
  - 行の先頭に戻る

  が異なる命令（機械的な動作）だった.
- それぞれに対応するのが, ¥n ¥r
- PCのキーボードでは, あまり意味が無いのだが,
  - Linux/UNIX系は ¥n が改行
  - Winでは ¥r¥n の両方が必要
  - Macでは ¥r だけ？

  とテキストファイルでの改行の流儀が分かれてしまった.

普通の文字

- コンピュータで標準的に扱われる文字は，ASCII と呼ばれる英数文字と記号である.
  - AからZ aからz 0から9 %, # 等の一部記号
- 標準的: ここでは，変数名，関数名等に用いることができるものとしている.
- 日本語や韓国語の文字，ドイツ語等の拡張部分(ä 等，通常アルファベットに装飾があるもの, いわゆる latin1)は標準的ではない.

- 本授業では，プログラム内では，ASCIIコードの範囲の文字のみ使ってください.
  - コメント内は除く

ASCIIの表

USASCII code chart

b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits ↓ / Column → / Row ↓	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

例えば，100 0001 という7bitの値を文字 A と規定している．

101 1100 の ＼ が，日本のPCにおける ¥ になっている．

文字の連続性

- 前述の表をみると，文字のAの次はBとデータとして連続性があるように見える.
  - char c='A'; c++; で cの中身は 'B' かも等.
- よって，C言語では，A-Z, a-z, 0-9 等の直感的な文字の連続性が保証されていると思う人もいる.
- 残念ながら，C言語では，このような文字の連続性は保証されていない.
  - 実際に反例がある文字コード体系が存在する/した.
- しかし，多くのコンピュータはASCIIおよびその拡張文字コード(日本のJIS等)に基づいているので，実質，問題無いという人もいる.
  - ASCIIに基づいていないコンピュータもある.

連続性の例 今時は正しく動くことが多い

```
/* charcont.c ASCII に基づくOSで動かした場合のみに正常に動く */

#include <stdio.h>

int main(void){
char c=' ';
  while(1){
    printf("[%c] ", c);
    if(c=='~'){break;}
    c++;
  }
  printf("¥n");
  return 0;
}
```

```
bash-3.1$ ./a
[ ] [!] [˝] [#] [$] [%] [&] [’] [(] [)] [*] [+] [,] [-] [.] [/] [0] [1] [2] [3]
[4] [5] [6] [7] [8] [9] [:] [;] [<] [=] [>] [?] [@] [A] [B] [C] [D] [E] [F] [G]
[H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z] [[]
[¥] []] [^] [_] [`] [a] [b] [c] [d] [e] [f] [g] [h] [i] [j] [k] [l] [m] [n] [o]
[p] [q] [r] [s] [t] [u] [v] [w] [x] [y] [z] [{] [|] [}] [~]
bash-3.1$
```

注意: 配列の代入と参照

- 配列同士の丸ごと全部の代入はできない.

```
int a[5]={3, 1, 4, 1, 5};
int b[5];
   b = a; // ダメ
```

- 配列の要素を丸ごと参照することはできない.

```
int a[5]={3, 1, 4, 1, 5};
  printf("%d", a); // ダメ というか意味が違う
```

- ただし, 文字配列だけは例外的にprintf文で丸ごと参照できる. (代入とかは文字列でもダメ)

```
char a[7]={'j', 'i', 'n', 'd', 'a', 'i', '¥0'};
  printf("%s", a); // これはOK
```

本日の演習 (演習7)

- 2018年において，X日目の日は，何月かを表示するプログラムを作成せよ.
- X日目のXはユーザーが入力するものとせよ.
- 参考までに，2018年の1月からのそれぞれの月は，以下の日数ある. (合計 365日)

  31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

- 1月1日を1日目と考えよ.
- 結果は，10 gatsu 等，数値で月を示せばよい. 「October」や「神無月」などの表示は不要.
- 0以下もしくは366以上の数字が入力された場合，何も表示しないこととせよ.
- 提出ファイル名は cal.c を推奨します.

演習の主なテストケース

入力	出力
1	1
400	
360	12
-10	
200	7
60	3
59	2

入力	出力
31	1
365	12
150	5
0	
366	
90	3
91	4

ヒント

- 以下の順に機能を段階的に拡張しながら，プログラムを作成してみなさい．

1. まずはキーボードから数値Xを読み込み表示する．
2. 12回ループを回って，各月の日数を表示してみる．
3. 12回ループを回って，その月が終わったら，何日過ぎたかの累積数を表示してみる．
   - 例えば，2月なら，1月+2月=59日，3月なら，1月+2月+3月なので，31+28+31=90日となる．尚，1月の場合31日，12月は365日．
4. 累積数≧Xかどうかを判定する．
   - 例えば，X=70 なら，2月では，まだ X≧累積数 だが，3月だと累積数≧Xとなる．
5. はじめて 累積数≧X となる月が答え．

本日は以上