プログラミング及び演習
第5回 配列2・文字列・演習
(2017/05/19)

講義担当
大学院情報学研究科知能システム学専攻
教授　森健策

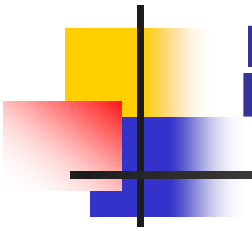
演習担当
大学院情報学研究科知能システム学専攻
小田昌宏

本日の講義・演習の内容

- 配列2
  - ソーティング
- 文字列
- 演習
- 講義・演習ホームページ
  - http://www.newves.org/~mori/17Programming

再度　メモを取る習慣をつけましょう

- 講義で補足された点のメモをとる習慣をつけてください。
- 重要なところを赤ペン・イエローマーカー等でマークするのも有効です。
- 折角の講義資料です。いろいろと書き込んでください。
- 後で資料を見直す際、メモした部分は意外に目に留まります。
  - 重要なポイントであったりします。



出席

- NUCTにて回答
  - 回答1
    - 今日の合言葉
  - 回答2
    - 配列は理解できましたか?
  - 回答3
    - 前回の演習について一言

質問メール

- 件名は
  - [質問] 必須課題235について
  - としてください。
- “[質問]”を忘れずに！！
  - メールの見落としの防止のためです。
- メールは以下のアドレスまで
  - 17Programming@mori.m.is.nagoya-u.ac.jp

配列とは？

- 「同じ」データ型の要素を記憶域上に連続して並べたもの
- 配列の要素は添字をつけてアクセス

int data[5];	
data[0]	1
data[1]	2
data[2]	4
data[3]	5
data[4]	20

float array[4];	
array[0]	0.1
array[1]	2.4
array[2]	4.2
array[3]	5.6

配列の宣言法 (1次元配列)

- <要素の型> <配列名>[<要素数>];
  - int array[5];
  - float data[10];
  - int array[4]={5,3,2,1};
  - int array[]={5,3,2,1};
  - unsigned char strings[200];
- 注意事項
  - 添字は0から始まる (1からではない!!)
  - 要素数を省略して初期化した場合には初期値の個数が要素数となる

配列要素の取り扱い方

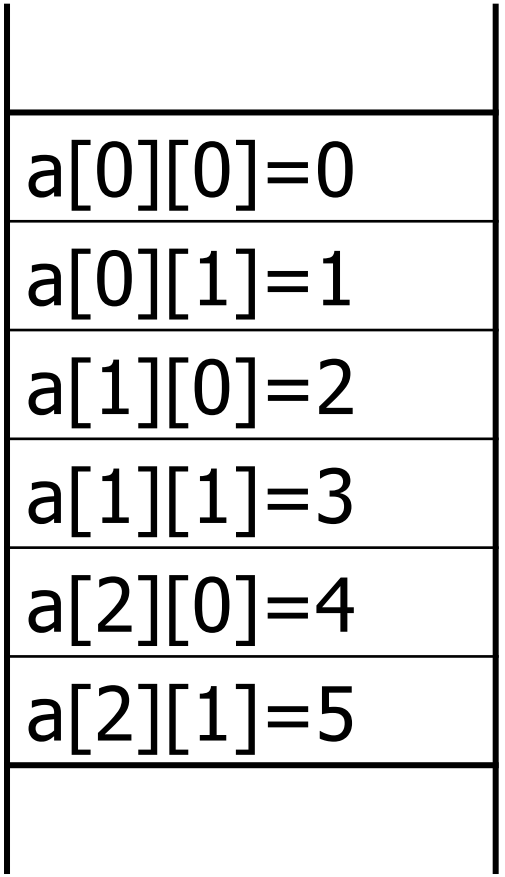
- 要素番号を[]括弧で囲んで配列名に続ける
- 要素は0から始まるのでdata[2]は3番目の要素
- 例
  - int a[5], x, i=1;
  - a[0]=1;a[1]=2;a[3]=5;a[4]=6;
  - x=a[1];
  - x=a[i+1];

多次元配列とメモリ

```
int a[3][2]={{0,1},{2,3},{4,5}};
```

a[0][0]=0	a[0][1]=1
a[1][0]=2	a[1][1]=3
a[2][0]=4	a[2][1]=5

構造



メモリにおける配置

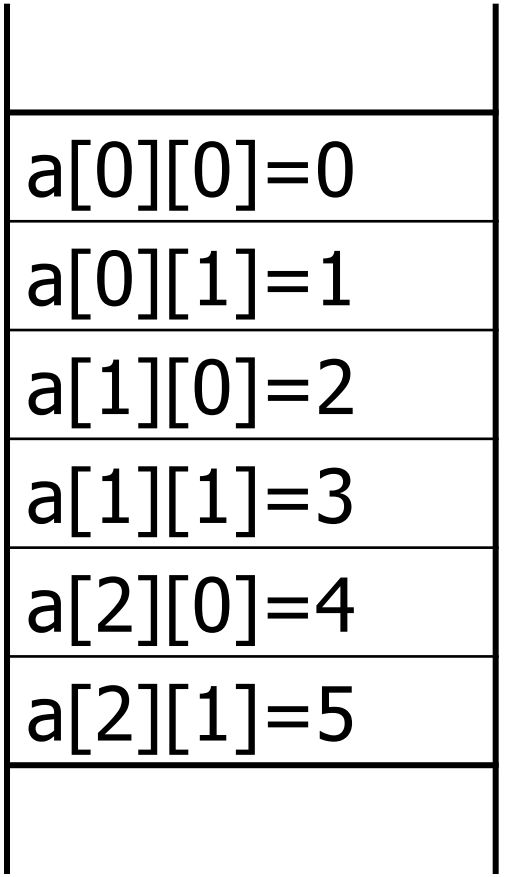
多次元配列とメモリ

- 最後の添字から順にメモリ上に配置される

int a[3][2]={{0,1},{2,3},{4,5}};

a[0][0]=0	a[0][1]=1
a[1][0]=2	a[1][1]=3
a[2][0]=4	a[2][1]=5

構造



メモリにおける配置

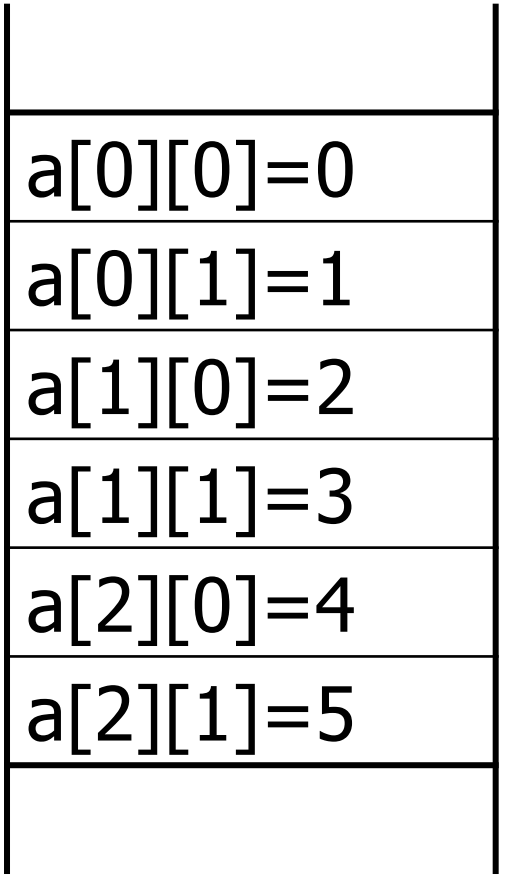
多次元配列とメモリ

```
int i,j,sum=0;
int a[3][2]={{0,1},{2,3},{4,5}};
for(i=0;i<3;i++){
    for(j=0;j<2;j++){
        sum+=a[i][j];
    }
}
```

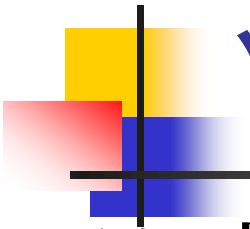
```
int i,j,sum=0;
int a[3][2]={{0,1},{2,3},{4,5}};
for(j=0;j<2;j++){
    for(i=0;i<3;i++){
        sum+=a[i][j];
    }
}
```

アクセス順序を比較

最後の添字から順にメモリ上に配置される



メモリにおける配置



ソート

- ソート
  - データ要素をある順序で並べ替えるアルゴリズム
  - アルゴリズム
    - 挿入ソート
    - 選択ソート
    - バブルソート
    - クイックソート
    - マージソート
    - Shellソート

    他多数

バブルソート

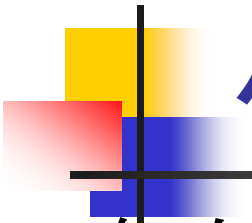
- 配列を最初から見てゆき隣同士を比較．逆順なら交換．
- 配列の最後まで行ったら，また最初に戻り同じことを続ける
- 交換するところがなくなるまで続行

3	2	5	6	2	5
2	3	5	6	2	5
2	3	5	6	2	5
2	3	5	6	2	5
2	3	5	2	6	5

2	3	5	2	5	6
2	3	5	2	5	6
2	3	5	2	5	6
2	3	2	5	5	6
2	3	2	5	5	6

2	3	2	5	5	6
2	3	2	5	5	6
2	2	3	5	5	6
2	2	3	5	5	6
2	2	3	5	5	6

2	2	3	5	5	6
2	2	3	5	5	6
2	2	3	5	5	6
2	2	3	5	5	6
2	2	3	5	5	6



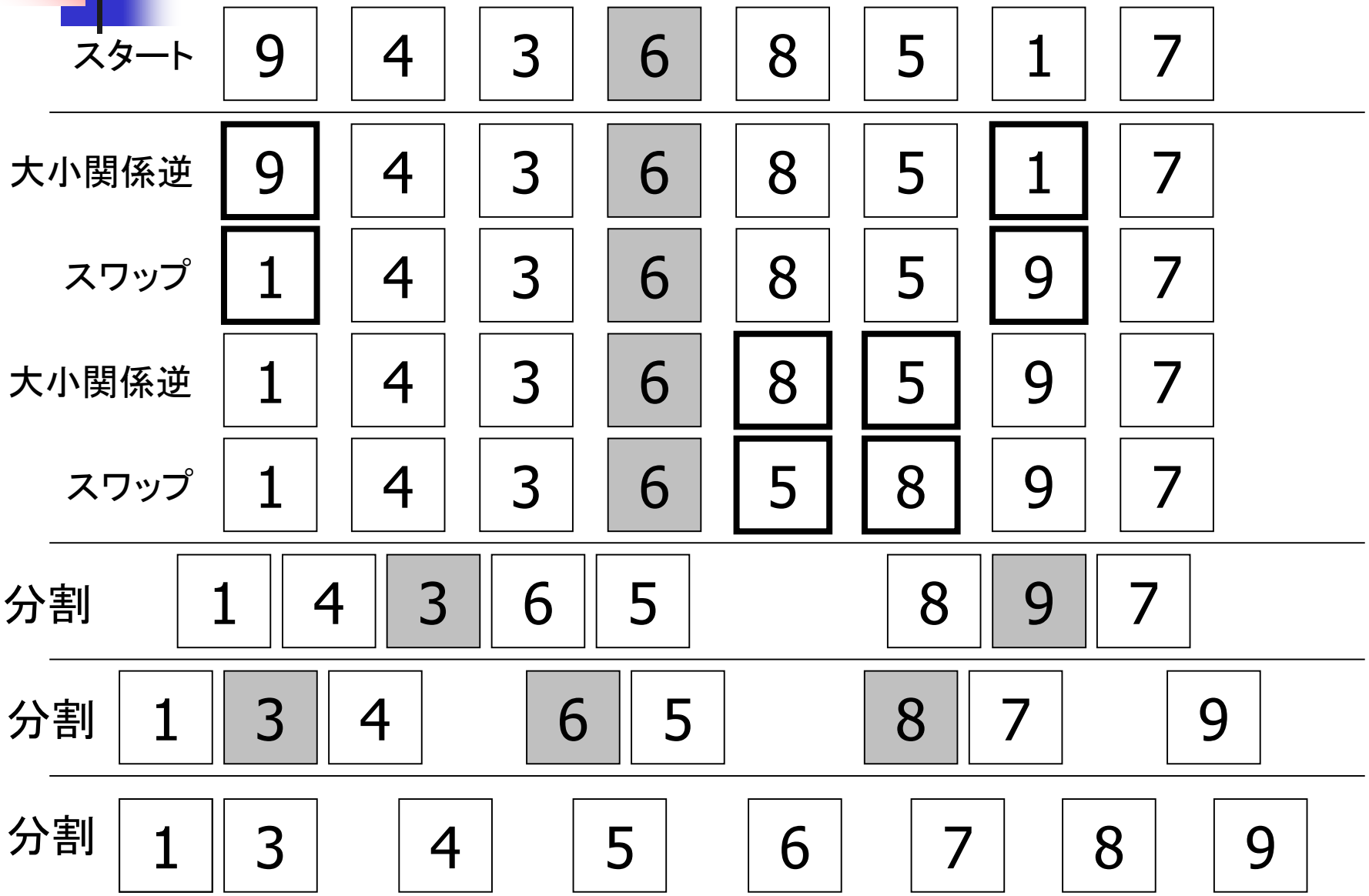
バブルソート

- 各自で作成

クイックソート

- 配列の大きさを次々に約半分にして自分自身を再帰的に呼び出す
- 手法
  - 適当な値xを選ぶ．その際，配列のx以下の要素の数とx以上の要素の数がなるべく同程度になるようにする
  - x以下の要素を配列の前半にx以上の要素を後半に集める
  - 配列の前半が長さ2以上なら配列の前半をクイックソート
  - 配列の後半が長さ2以上なら配列の前半をクイックソート

クイックソート


スタート
9 4 3 6 8 5 1 7
大小関係逆
9 4 3 6 8 5 1 7
スワップ
1 4 3 6 8 5 9 7
大小関係逆
1 4 3 6 8 5 9 7
スワップ
1 4 3 6 5 8 9 7
分割
1 4 3 6 5 8 9 7
分割
1 3 4 6 5 8 7 9
分割
1 3 4 5 6 7 8 9

クイックソート

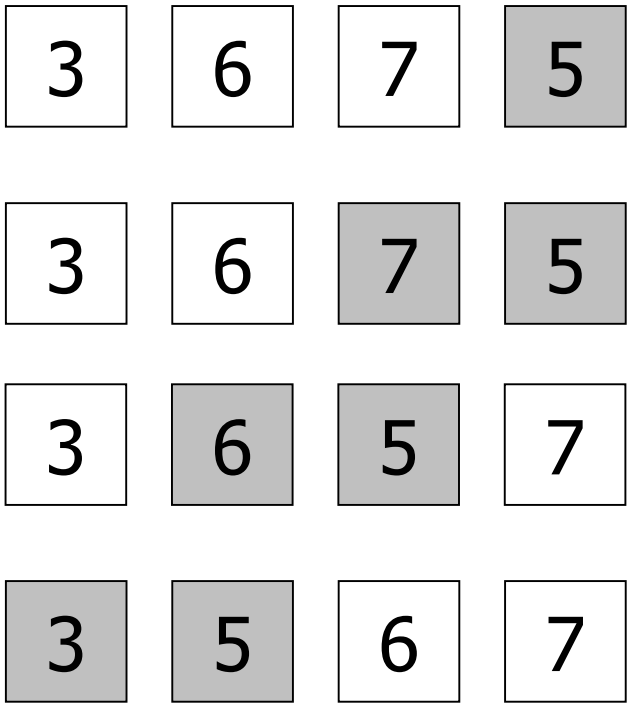
```
void quicksort(int data[], int first, int last)
{
  int i,j;
  int x,t;
  x = data[(first+last)/2];
  i=first; j=last;
  for(;;){
    while(data[i]<x) i++;
    while(x<data[j]) j--;
    if(i>=j) break;
    t=data[i]; data[i]=data[j]; data[j]=t;
    i++; j--;
  }
  if(first<i-1) quicksort(data,first,i-1);
  if(j+1<last) quicksort(data,j+1,last);
}
```

クイックソートの注意点

- 再帰を用いるのでスタックオーバーフローに注意
- ある程度整列されたデータを整列する場合には効率が悪い
  - for(i=0;i<n/2;i++) a[i]=i;
    for(i=n/2;i<n;i++) a[i]=n-i;
- 分割時にはそれぞれ同程度の要素数となることが理想

インサートソート

- ソート済み配列に新たにデータ追加することを考える



インサートソート

- 一番左端をソート済みと考え順次要素を追加

インサートソートのプログラム

- 各自考える

コンピュータでの文字の取り扱い
ASCIIコード (教科書 p.68)

- 各文字に符号(番号)を割り当て
  - 'A' = 0x41
  - 'B' = 0x42
- C言語では
  - char型
    - 1文字を記憶
    - 文字コードを保持
    - char a = 0x46;
      printf("%c¥n", a);

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

特殊文字

拡張表記	コード	意　味
¥a	0x07	ベルを鳴らす
¥b	0x08	1文字戻る
¥f	0x0c	改ページする
¥n	0x0a	改行し行の左端に戻る
¥r	0x0d	行の左端に戻る
¥t	0x09	水平タブ
¥v	0x0b	垂直タブ
¥0	0x00	空文字．終端記号
¥'	0x27	シングルquotation
¥"	0x22	ダブルquotation
¥?	0x3f	疑問符
¥¥	0x5c	円記号
¥NNN	0NNN	8進表記
¥xHH	0xHH	16進表記

文字の入出力

- int getchar(void);
  - 1文字読み込んでその文字コードを返す
- int putchar(int c);
  - 値cを文字コードとして書き出す
- 読み書きが成功したときは文字コードを関数値として返し，失敗したときはEOF (stdio.hでマクロ定義．ファイルの終了を表す)を返す
- 例
  - char chr;
    chr = getchar();
    putchar(chr); putchar('¥n');

小文字を大文字に変換する (lec6-cap.c)

```
#include <stdio.h>
main()
{
  char c;
  while(1){
    c = getchar();
    if( c>='a' && c<='z' ){
      printf( "%c¥n", c-'a'+'A' );
    }else if( c>='A' && c<='Z' ){
      printf( "%c¥n",c );
    }
  }
}
```

文字列と配列

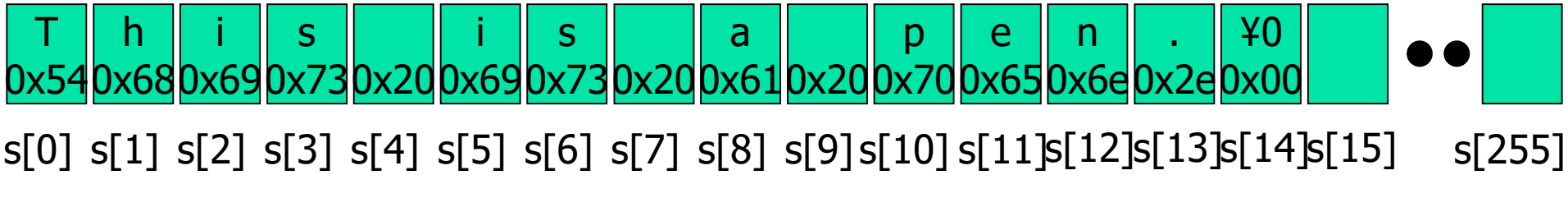
- 文字列
  - 文字の配列
  - 文字列の最後は¥0(終端記号)
  - unsigned char text[]="this is a pen."; として初期化可能
  - 配列の各要素は文字列をなす
  - text[0], text[1], text[2]…とすれば文字列の先頭から順次アクセス可能

Cにおける文字列の取り扱い

- 文字列　＝ 文字の配列
  - 文字列を成す文字を入れるだけの器(配列)を用意
  - 先頭から順に文字コードが格納される
  - 文字列の最後には'¥0'
  - 例

```
char s[256];
gets(string);
```

```
taka{mori}58: ./a.out
This
54(T) 68(h) 69(i) 73(s)
taka{mori}59:
```



文字列と配列

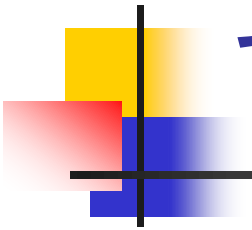
- 文字列
  - 文字の配列
  - 文字列の最後は¥0(終端記号)
  - unsigned char text[]="this is a pen."; として初期化可能
  - 配列の各要素は文字列をなす
  - text[0], text[1], text[2]…とすれば文字列の先頭から順次アクセス可能

文字列の入出力

- printf, scanfで%s指定を用いる
  - char tmp[12];
scanf("%s",tmp);
printf("tmp = %s¥n", tmp);
  - 12文字以上の入力では配列の大きさオーバー
- char *gets(char str[]);
  - 文字列を読み込みstr[]に格納
  - 読み出しに成功した場合にはstr[]へのポインタを返す
  - 失敗したときはNULLを返す
- int puts(char str[]);
  - str[]内の文字列を書き出して改行
  - 書き出しに成功したときは非負の値を返す
  - 失敗したときはEOFを返す

getsは絶対使わない

- gets()は絶対に使用してはならない。
- データを知ることなしにgets() が何文字読むかを先に知る事はできず、gets()がバッファの終りを越えて書き込み続けるため、使用は大変危険である。この事はコンピュータのセキュリティを破るのに使われてきた。代わりにfgets()を使うこと



では，どうするのか?

- fgetsを使う
- char *fgets(char *s, int size, FILE *stream);
  - char tmp[12];
    fgets(tmp, 12, stdin);
  - size よりも1文字以上少ない文字を stream から読み込み、s で示されるバッファに書き込む。読み込みは EOF または改行文字を読み込んだ後終わる。改行文字は読まれるとバッファに書き込まれる。'¥0' 文字がバッファの中の最後の文字の後に1文字書き込まれる。
  - 成功すると s を返し、ファイルの終りあるいはエラーの場合 NULL を返す。
  - streamにstdinと記述すると標準入力(大抵はキーボード)を表す

文字列長の計算 (lec5-length.c)

```
#include <stdio.h>
main()
{
  char tmp[256];
  int i;
  int lineno = 1;
  while(fgets(tmp,256,stdin)!=NULL){
    i=0;
    while(tmp[i]!='¥0' && i<256){
      i++;
    }
    printf( "No %d = %d¥n", lineno,i);
    lineno++;
  }
}
```

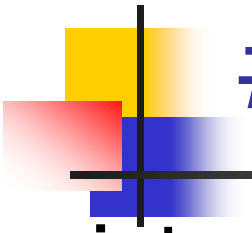
注意点！！
getsは改行文字を取り込まない
fgetsは改行文字を取り込む

文字列関数

- 文字列を処理するための関数
- string.hを必ずincludeすること
  - #include <string.h>
- 場合によってはstrings.hをinclude
- 個々の詳細はmanページ
  (例 man strlen)
  もしくはKR p.314を参照

#include <string.h>

- char *strcat(char s1[], const char s2[]);
- char *strncat(char s1[], const char s2[], size_t n);
- size_t strlcat(char dst[], const char src[], size_t dstsize);
- char *strchr(const char s[], int c);
- char *strrchr(const char s[], int c);
- int strcmp(const char s1[], const char s2[]);
- int strncmp(const char s1[], const char s2[], size_t n);
- char *strcpy(char s1[], const char s2[]);
- char *strncpy(char s1[], const char s2[], size_t n);
- size_t strlcpy(char dst[], const char src[], size_t dstsize);
- size_t strcspn(const char s1[], const char s2[]);
- size_t strspn(const char s1[], const char s2[]);
- char *strdup(const char s1[]);
- size_t strlen(const char s[]);
- char *strpbrk(const char s1[], const char s2[]);
- char *strstr(const char s1[], const char s2[]);
- char *strtok(char s1[], const char s2[]);
- char *strtok_r(char s1[], const char s2[], char **lasts);



#include <strings.h>

- int strcasecmp(const char *s1, const char *s2);
- int strncasecmp(const char *s1, const char *s2, size_t n);

各種文字列関数

- size_t strlen(const char s[]);
  - 文字列sの長さを返す
  - char s[]="HELLO";
printf("%d¥n", strlen(s));
- char *strcpy(char s1[], const char s2[]);
  - '¥0'を含めて文字列s2をs1にコピーしs1を返す
  - char tmp[256];
……
……
strcpy(s, "Hello”);
- char *strncpy(char s1[], const char s2[], size_t n);
  - 文字列s2のうち最大n文字をs1にコピーしs1を返す．s2がn文字より少ないときは¥0をつめる
  - s2がn文字より大きな場合にはs1の最後には¥0がない

各種文字列関数

- char *strcat(char s1[], const char s2[]);
  - 文字列s2を文字列s1に連結しs1を返す
  - char tmp[256];
    strcpy(tmp, "This is ");
    strcat(tmp , "a pen.");
- char *strncat(char s1[], const char s2[], size_t n);
  - 文字列s2の最大n文字を文字列s1に連結し，終わりに¥0を付け，s1を返す

各種文字列関数

- int strcmp(const char s1[], const char s2[]);
  - 文字列s1と文字列s2を先頭から順に比較し，最初に異なった文字の文字コード s1[i]<s2[i]なら時は負，s1[i]>s2[i]なら正,完全一致なら0を返す
  - strcmp("the", "that") はどうなる?
  - strcmp("fog", "foggy") はどうなる?
- int strncmp(const char s1[], const char s2[], size_t n);
  - 先頭からn文字のみ比較を行うことを除いてstrcmpと同様

各種文字列関数

- char *strchr(const char s[], int c);
  - sの中で最初に見つかった文字cへのポインタ
  - 見つからなければNULLを返す
  - char tmp[256];
strcpy(tmp, "This is");
printf("Result %s¥n", strchr(tmp, 'h'));
- char *strrchr(const char s[], int c);
  - sの中で最後に見つかった文字cへのポインタ
  - 見つからなければNULLを返す

各種文字列関数

- char *strstr(const char s1[], const char s2[]);
  - s1の中にs2が最初に現れる文字へのポインタ
  - 見つからなければNULLが返る
  - char tmp[256];
    strcpy(tmp, "This is a pen");
    printf("Result %s¥n", strstr(tmp, "is"));
  - s1の中にs2が含まれるかどうかをチェックするのに頻繁に使われる

文字列から数値へ

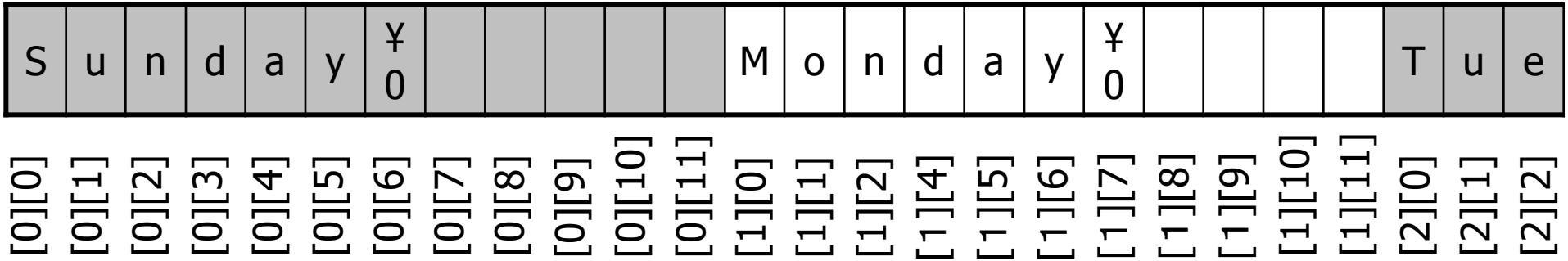
- int atoi(const char s[]);
- long atol(const char s[]);
  - 文字列sを数字の列と解釈しint, longに変換
- double atof(const char s[]);
  - 文字列sを数字の列と解釈しdoubleに変換
- #include <stdlib.h>を忘れずに

配列の中の文字列

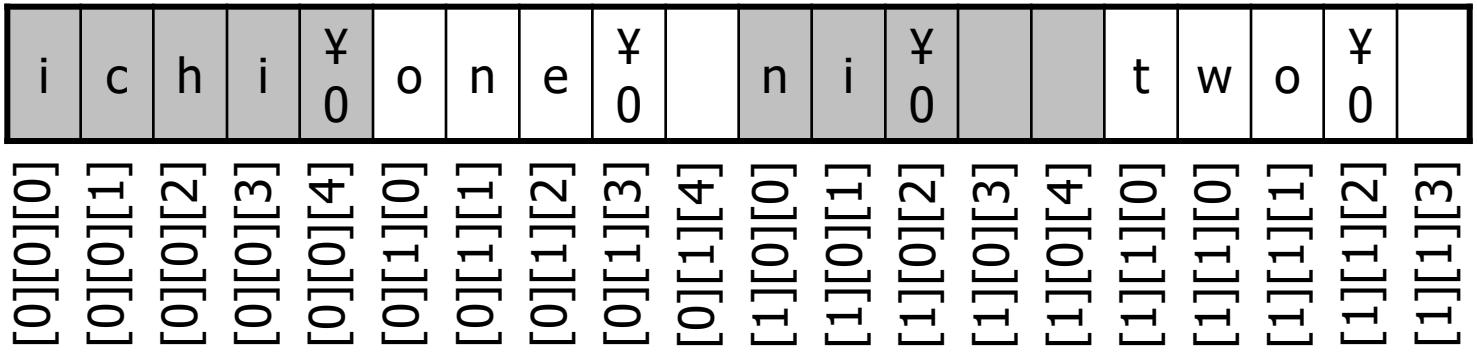
- 配列の要素として文字列を取り扱うことが可能
- 例)
  - unsigned char table[7][64]=
    {"Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"};
    printf("%s", table[4]);
  - unsigned char matrix[2][2][128]=
    {{"ichi", "one"}, {"ni","two}};
    for(j=0;j<2;j++){
    for(i=0;i<2;i++){
    printf("%d %d= %s¥n", i,j, matrix[j][i]);
    }
    }

文字列配列とメモリ内容

```
unsigned char table[7][12]=
        {"Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"};
```




```
unsigned char matrix[2][2][5]=
        {{"ichi","one"}, {"ni","two}};
```



プログラミング時の心がけ

- 他人が理解しやすいプログラムを書く
  - 必要十分なコメントを
- まずは理解しやすいプログラムを書く
  - トリッキーなプログラムを最初から追及しない
- ドキュメントをしっかり書く
  - 仕様書，マニュアル，リファレンス，アルゴリズムなど
- 他人のプログラムをコピーしない
  - 盗作はやめましょう
  - 採点時にわかります