#06

文字列の操作

2022 年度 / プログラミング及び実習 III

角川裕次

龍谷大学　先端理工学部

もくじ

今回 (#06) の内容：シラバスでの該当部分

小テーマ: **文字列の操作**

第 9 回：**文字列のプログラミング**

重要概念リスト

- **文字列の長さ**
- **文字列を表示する関数**：puts, fputs
- **書式付で表示する関数**：printf, fprintf
- **文字を表示する関数**：putc, fputc, putchar
- **大文字/小文字の変換**：toupper, tolower

今回の実習・課題 (manaba へ提出)

実習内容と課題内容は講義途中に提示します

(**作成したファイル類は** manaba **に提出**)

第 9-2 節　文字列の配列

□ 文字列の配列 p.262

List 9-6：**文字列の配列とその表示** (2 **次元配列**)

```
#include <stdio.h>
int main(void)
{
  char cs[][6] = {"Turbo", "NA", "DOHC"};
  for (int i = 0; i < 3; i++)
    printf("cs[%d] = \"%s\"\n", i, cs[i]);
  return 0;
}
```

配列の各要素の値

- cs[0] = "Turbo"
- cs[1] = "NA"
- cs[2] = "DOHC"

	[0]	[1]	[2]	[3]	[4]	[5]
cs[0]	T	u	r	b	o	\0
cs[1]	N	A	\0	\0	\0	\0
cs[2]	D	O	H	C	\0	\0

初期化子が不足している要素：\0 **で初期化**

文字列の配列 (つづき)

```
char cs[][6] = {"Turbo", "NA", "DOHC"};
```

2 次元配列なのでいつものようにアクセスできる

- cs[0][0] = 'T'
- cs[0][1] = 'u'
- cs[0][2] = 'r'
- cs[2][3] = 'C'
- cs[2][5] = '\0'

	[0]	[1]	[2]	[3]	[4]	[5]
cs[0]	T	u	r	b	o	\0
cs[1]	N	A	\0	\0	\0	\0
cs[2]	D	O	H	C	\0	\0

cs : 要素数 3 の配列

- 各要素 : 要素数 6 の文字配列

cs						
[0]	T	u	r	b	o	\0
[1]	N	A	\0	\0	\0	\0
[2]	D	O	H	C	\0	\0

□ 文字列の配列への文字列の読み込み p.263

List 9-7：**文字列の配列の各要素に文字列を読み込んで表示**

```
#include <stdio.h>
int main(void)
{
  char s[3][128];
  for (int i = 0; i < 3; i++) {
    printf("s[%d] : ", i);
    scanf("%s", s[i]);
  }
  for (int i = 0; i < 3; i++)
  printf("s[%d] = \"%s\"\n", i, s[i]);
  return 0;
}
```

scanf **を使わず** fgets **を使うように書き換えましょう…**

各文字列は 127 **文字まで**

- **配列の要素数は** 128：127 **文字** + **ナル文字** 1 **つ**

注意：scanf("%s", s[i]); **の** s **の前に**&**がない**

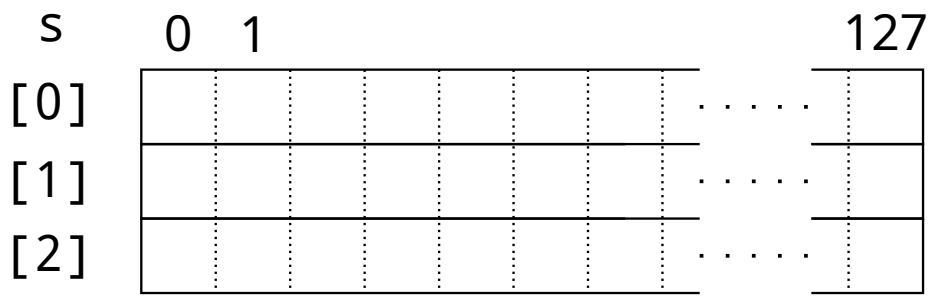
List 9-7 : 文字の 2 次元配列の理解

(再掲; 部分) List 9-7

```
char s[3][128];
```

s : 要素数 3 の配列

- 各要素 : 要素数 128 の文字配列



(再掲; 部分) List 9-7

```
scanf("%s", s[i]);
```

- scanf は要素数 128 の文字配列へ読み込む
- s[i] : 要素数 128 の文字配列 (s のひとつの要素)
- なので s[i] に & は不要ですよね

第 9-3 節　文字列の操作

□ 文字列の長さ p.264

最初のナル文字までの文字数と定義

- **ナル文字は数えない**

例 1 : **文字列** `char str[6] = "ABC";` **の長さは** 3

例 2 : **文字列** `char str[6] = "";` **の長さは** 0 (**空文字列**)

文字列の長さのプログラム例

List 9-8：**文字列の長さを調べる**

```
#include <stdio.h>
int str_length(const char s[])
{
  int len = 0;
  while (s[len])
    len++;
  return len;
}

int main(void)
{
  char str[128];
  printf("文字列を入力せよ：");
  scanf("%s", str);
  printf("文字列\"%s\"の長さは%dです。\n",
         str, str_length(str));
  return 0;
}
```

文字列の長さ，プログラム (List 9-8) 実行例

実行例 1

```
$ ./list-9-8
文字列を入力せよ：ABC
文字列"ABC"の長さは3です。
```

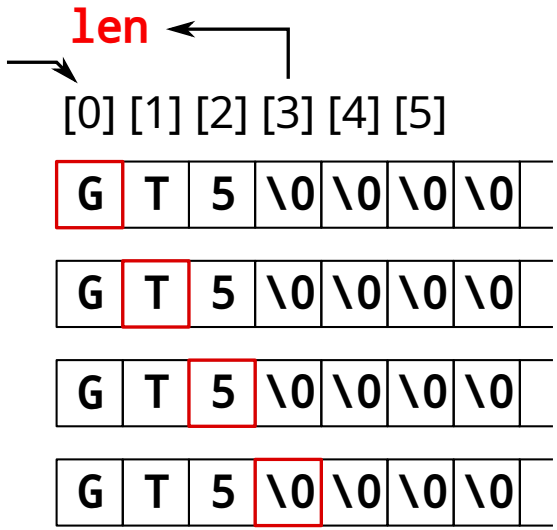
実行例 2

```
$ ./list-9-8
文字列を入力せよ：JugemuJugemu
文字列"JugemuJugemu"の長さは12です。
```

関数 str_length() の観察

```
int str_length(const char s[])
{
  int len = 0;
  while (s[len])
    len++;
  return len;
}
```

`while (s[len])` **は** `while (s[len] != '\0')` **と同じ**



scanf の問題点がわかる実行例

```
$ ./list-9-8
文字列を入力せよ：ABC ABC
文字列"ABC"の長さは3です。
```

- scanf：空白を区切りとみなしている
- 空白を含む文字列を取り扱うには不向き
- これが scanf の使用をおすすめしない理由のひとつ

1 行読み込み (scanf 不使用版) の実装

scanf の代わりに fgets (1 行読み込み) を使うのが良い

- EOF を読むと fgets() は NULL を返す
- fgets() は '\n' も含めて読み込む仕様: 除去が必要

List 9-8 (改; 主要部) : 1 行の文字列を読み込む

```
char *read_string(int n, char *s)
{
  char *p = fgets(s, n, stdin);
  if (p != NULL) {
    char *q = index(s, '\n');
    if (q != NULL) {
      /*文字列最後の改行(Enterキー)は除去*/
      *q = '\0';
    }
  }
  return p;
}
```

1 行読み込み (scanf 不使用版) の使用の main 関数

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char str[8];
  printf("文字列を入力せよ：");
  if (read_string(sizeof(str), str) != NULL) {
    printf("文字列\"%s\"の長さは%dです。\n",
           str, str_length(str));
  } else {
    printf("END-OF-FILE\n");
  }
  return 0;
}
```

注意：**文字配列の要素数を** 8 **にしている**

- **わざと短くしている (問題発生を分かりやすくしている)**

1 行読み込み (scanf 不使用版) : 実行例

バッファオーバーフローを回避できている

- **要素数 8 の文字配列へ長い入力を与えても OK**

```
$ ./list-9-8-fixed
文字列を入力せよ：Jugemu
文字列"Jugemu"の長さは6です。
```

```
$ ./list-9-8-fixed
文字列を入力せよ：JugemuJugemu
文字列"JugemuJ"の長さは7です。
```

文字列に空白が含まれていても期待通りに動作

```
$ ./list-9-8-fixed
文字列を入力せよ：X Y Z
文字列"X Y Z"の長さは5です。
```

□ 文字列の表示 p.266

表示のための関数： putchar, fputc, putc, puts, fputs, printf, fprintf

putchar **関数**

- **(標準出力に) 文字を出力**

fputc **関数**, putc **関数**

- **(指定ストリームに) 文字を出力**

puts **関数**

- **(標準出力に) 文字列を出力**

fputs **関数**

- **(指定ストリームに) 文字列を出力**

printf **関数**

- **(標準出力に) 文字列, 文字, 整数, 実数等を書式に従って出力**

fprintf **関数**

- **(指定ストリームに) 文字列, 文字, 整数, 実数等を書式に従って出力**

putchar 関数を使って文字列を表示

List 9-9 (部分)：putchar 関数を使って文字列を表示

```
void put_string(const char s[])
{
  int i = 0;
  while (s[i])
    putchar(s[i++]);
}
```

注：`while (s[i])` は
`while (s[i] != '\0')` と同じ

動作

- 文字列の先頭から 1 文字づつ putchar 関数で表示してゆく
- ただしナル文字に出会うと終了
- ナル文字は表示しない

□ 数字文字の出現回数 p.267

0 から 9 までの各数字が文字列中に何回出現するかを勘定

実行例

```
文字列を入力せよ：pi=3.1415926535,e=2.71828
数字文字の出現回数
'0':0
'1':3
'2':3
'3':2
'4':1
'5':3
'6':1
'7':1
'8':2
'9':1
```

文字列中の各数字の出現回数のカウント方：main 関数

List 9-10 (部分 1/2)：文字列の入力と結果表示

```
int main(void)
{
  int  dcnt[10] = {0};   /* 分布 */
  char str[128];         /* 文字列 */
  printf("文字列を入力せよ：");
  scanf("%s", str);
  str_dcount(str, dcnt);
  puts("数字文字の出現回数");
  for (int i = 0; i < 10; i++)
    printf("'%d'：%d\n", i, dcnt[i]);
  return 0;
}
```

数字文字の出現回数, 本体部分

List 9-10 (**部分** 2/2) : **実際のカウント**

```
void str_dcount(const char s[], int cnt[])
{
  int i = 0;
  while (s[i]) {
    if (s[i] >= '0' && s[i] <= '9')
      cnt[s[i] - '0']++;
    i++;
  }
}
```

動作

- **文字列を先頭から** 1 **文字づつ眺めてゆく**
- **数字** ('0', '1', ..., '9') **なら**
  配列 cnt **の対応する要素の値を** 1 **増加**

□ 大文字・小文字の変換 p.268

関数 toupper(int c) : **小文字を大文字に変換する**

- **小文字以外はそのまま**
- **例** : a → A
- **例** : A → A
- **例** : 0 → 0
- **ヘッダ** ctype.h **のインクルードが必要**
- **いわゆる半角文字だけが変換対象 (全角文字は対象外)**

List 9-11 (**部分**): **文字列中の小文字をすべて大文字に変換**

```
void str_toupper(char s[])
{
  int i = 0;
  while (s[i]) {
    s[i] = toupper(s[i]);
    i++;
  }
}
```

大文字・小文字の変換 (つづき)

関数 tolower(int c) : **大文字を小文字に変換する**

- **大文字以外はそのまま**
- **例** : `A` → `a`
- **例** : `a` → `a`
- **例** : `0` → `0`
- **ヘッダ** `ctype.h` **のインクルードが必要**
- **いわゆる半角文字だけが変換対象 (全角文字は対象外)**

List 9-11 (**部分**): **文字列中の大文字をすべて小文字に変換**

```
void str_tolower(char s[])
{
  int i = 0;
  while (s[i]) {
    s[i] = tolower(s[i]);
    i++;
  }
}
```

□ 文字列の配列の受け渡し p.270

List 9-12：**文字列の配列の内容を表示する関数 (仮引数 s[][6] に注目)**

```
#include <stdio.h>
void put_strary(const char s[][6], int n)
{
  for (int i = 0; i < n; i++)
    printf("s[%d] = \"%s\"\n", i, s[i]);
}
int main(void)
{
  char cs[][6] = {"Turbo", "NA", "DOHC"};
  put_strary(cs, 3);
  return 0;
}
```

出力 (表示)

```
s[0] = "Turbo"
s[1] = "NA"
s[2] = "DOHC"
```

仮引数 `char s[ ][6]` の 6 は省略できない

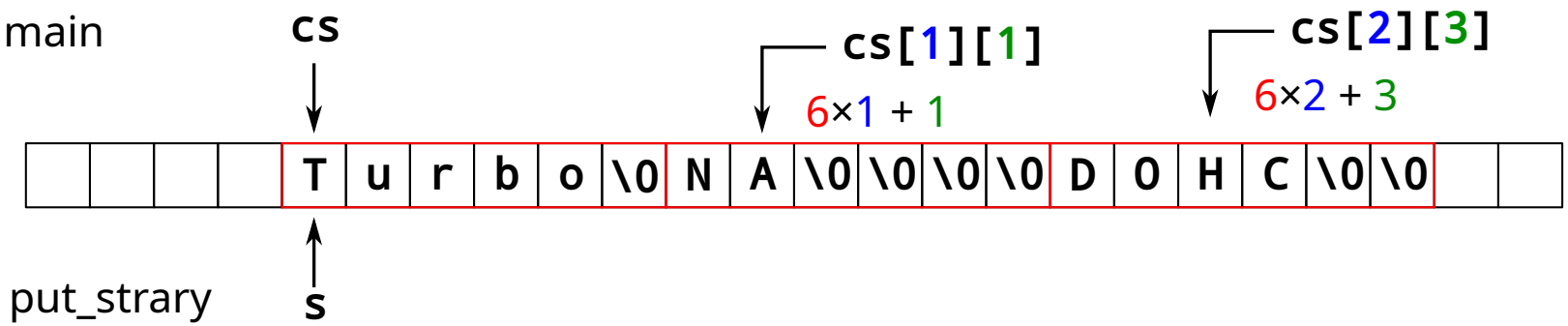
```
void put_strary(const char s[][6], int n) {  ...  }
```

Q: **関数内で要素 `s[i][j]` の配置アドレスはどうやって求める?**

A: 6 **という情報があれば要素の配置アドレスを (コンパイラは) 計算可能**

要素 `s[i][j]` の配置アドレス

$$= \quad \texttt{s} \text{ の先頭のアドレス} + \mathrm{sizeof(char)} \times (6 \times i + j)$$



6 **という情報がないと要素の配置アドレスを計算する手立てがない**
(変数宣言を正しくすれば配置アドレス計算は気にする必要なし)

Q. でも他で仮引数を char p[ ] とかやってましたよ?

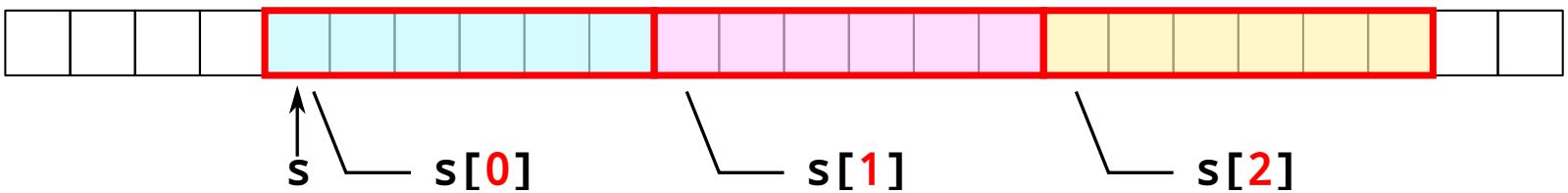
A. それができるのは 1 要素のサイズが char で分かるからです

- なので p[10] の配置アドレスを計算できます

Q. char s[][] でも 1 要素のサイズが分かりますよね?

A. それだと char s[] のサイズが分からないです

- s[2][3] の配置アドレスを計算できません



s[2][3]：要素数 2 の配列

- 各要素：要素数 3 の文字配列
- この 1 要素のサイズが分かる必要あり

putchar 関数を使ってみる

List 9-13 (部分) : 文字列の配列を表示

```
void put_strary2(const char s[][6], int n)
{
  for (int i = 0; i < n; i++) {
    int j = 0;
    printf("s[%d] = \"", i);
    while (s[i][j])
      putchar(s[i][j++]);
    puts("\"");
  }
}
```

注 : `while (s[i][j])` は
`while (s[i][j] != '\0')` と同じ

(独自) 文字に関する標準ライブラリ関数

文字に関する標準ライブラリ関数：概要

インクルードすべきヘッダ

```
#include <ctype.h>
```

わりとよく使う：

`isspace(c)` — **スペース文字か**?
`isprint(c)` — **表示可能文字** (**スペース以外**) **か**?
`isupper(c)` — **大文字アルファベットか**?
`islower(c)` — **小文字アルファベットか**?
`isalpha(c)` — **アルファベットか**?
`isalnum(c)` — **アルファベットまたは数字** (10 **進数**) **か**?
`isdigit(c)` — **数字** (10 **進数**) **か**?
`isxdigit(c)` — **数字** (16 **進数**) **か**?

文字種別の判定 (1)

`int isspace(int c)`

- `c` はスペースか否か
- 具体的にスペースとは：空白, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

`int isprint(int c)`

- `c` は表示可能文字 (含むスペース) か否か

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

文字種別の判定 (2)

`int isupper(int c)`

- `c` は大文字アルファベットか否か

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

`int islower(int c)`

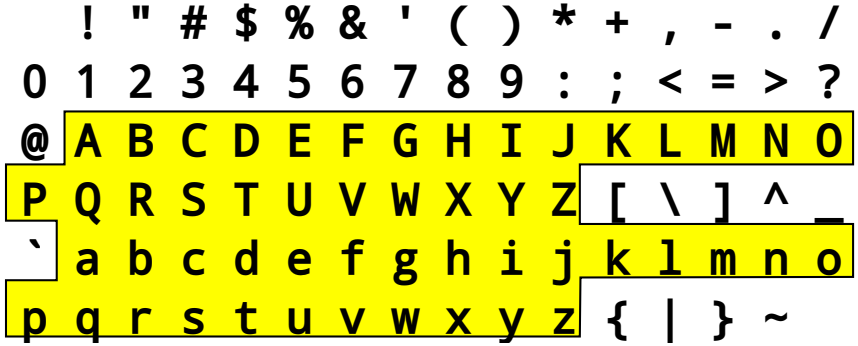
- `c` は小文字アルファベットか否か

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

文字種別の判定 (3)

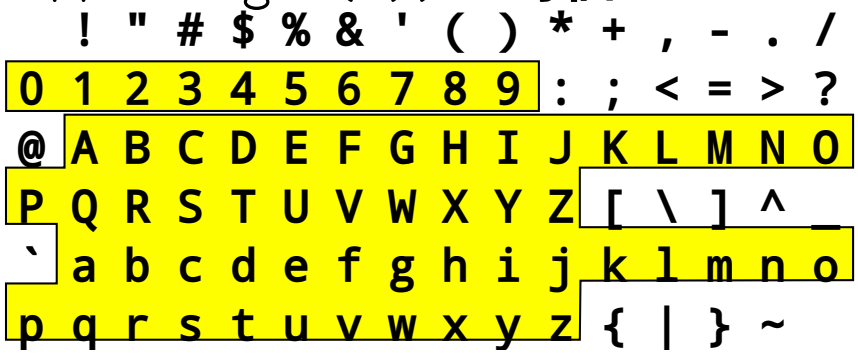
`int isalpha(int c)`

- c はアルファベットか否か
- `(isupper(c) || islower(c))` と等価



`int isalnum(int c)`

- c は英数文字 (アルファベット又は数字) か否か
- `(isalpha(c) || isdigit(c))` と等価



文字種別の判定 (4)

`int isdigit(int c)`

- `c` は数字か否か

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

`int isxdigit(int c)`

- `c` は 16 進数表示用の文字か否か

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

文字種別の判定 (5)

`int ispunct(int c)`

- `c` はスペース・英数文字以外か否か (記号か否か)

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

`int isgraph(int c)`

- `c` は (スペース以外の) 表示可能な文字か否か

```
  ! " # $ % & ' ( ) * + , - . /
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 { | } ~
```

文字種別の判定 (そのほか)

`int isascii(int c)`

- `c` は符号無し 7 ビットで表現可能な文字か否か

`int isblank(int c)`

- `c` は空白文字・タブ文字か否か

`int iscntrl(int c)`

- `c` は制御文字か否か

大文字/小文字変換

`int toupper(int c)`

- 小文字は大文字に変換（それ以外の文字はそのまま）
- `touppder('a')` → `'A'`
- `touppder('X')` → `'X'`

`int tolower(int c)`

- 大文字は小文字に変換（それ以外の文字はそのまま）
- `tolower('A')` → `'a'`
- `tolower('y')` → `'y'`

(独自) 文字列に関する標準ライブラリ関数

文字列に関する標準ライブラリ関数：概要

インクルードすべきヘッダ

```
#include <string.h>
```

`strlen(s)` — **長さ**
`strcmp(s1,s2)` — **大小比較**
`strncmp(s1,s2,n)` — **大小比較 (文字数限定)**
`strchr(s,c)` — **文字** `c` **の検索 (先頭から)**
`strrchr(s,c)` — **文字** `c` **の検索 (末尾から)**
`strstr(s,t)` — **文字中の文字列の検索**
`strncat(s,t,n)` — `s` **の後に** `t` **を連結**
`strncpy(s,t,n)` — `s` **に** `t` **を上書きコピー**
`strdup(s)` — `s` **のコピーを生成**

文字列の長さ

```
size_t strlen(const char *s)
```

- **文字列 `s` の長さを返す**

文字列の比較

`int strcmp(const char *s1, const char *s2)`

- **文字列** `s1` **と** `s2` **を比較**
- **返り値**: 0 (`s1` = `s2` **の場合**)
- **返り値**: < 0 (`s1` < `s2` **の場合**)
- **返り値**: > 0 (`s1` > `s2` **の場合**)

`int strncmp(const char *s1, const char *s2, size_t n)`

- **文字列** `s1` **と** `s2` **を比較 (先頭から** `n` **文字まで)**
- **返り値**: 0 (`s1` = `s2` **の場合**)
- **返り値**: < 0 (`s1` < `s2` **の場合**)
- **返り値**: > 0 (`s1` > `s2` **の場合**)

文字の検索

`char *strchr(const char *s, int c)`

- 文字列 `s` の中の文字 `c` を検索
- 検索は文字列の先頭から
- 見つかればその文字へのポインタを返す
- 見つからなければ NULL ポインタを返す

`char *strrchr(const char *s, int c)`

- 文字列 `s` の中の文字 `c` を検索
- 検索は文字列の末尾から
- 見つかればその文字へのポインタを返す
- 見つからなければ NULL ポインタを返す

文字列の連結・コピー

```
char *strncat(char *dest, const char *src,
              size_t n)
```

- 文字列 dest の末尾に文字列 src の内容を連結
- dest の長さの制限は n

```
char *strncpy(char *dest, const char *src,
              size_t n)
```

- 文字列 dest に文字列 src をコピー
- コピーするのは src の n 文字まで
- 要注意:
  n 文字ちょうどのコピーの場合 dest は\0 で終端されない

```
char *strdup(const char *s)
```

- 新たな文字配列 (メモリ) を割り当て文字列 s をコピー
- 新たな文字配列へのポインタを返す
- (その文字列が不要になれば free() でメモリ解放が必要)

おわり

番外編の課題 1

任意に与えられた文字列に対し文字列中の 1 単語を 1 行ごとに表示する関数の作成

- `void pstrword(const char *s);`
- **ただし単語を構成する文字はアルファベットと数字に限定**

例: `pstrword("He was born in 2001.");` **の出力**

```
He
was
born
in
2001
```

例: `pstrword("  Hello, world.  You have mail. ");` **の出力**

```
Hello
world
You
have
mail
```