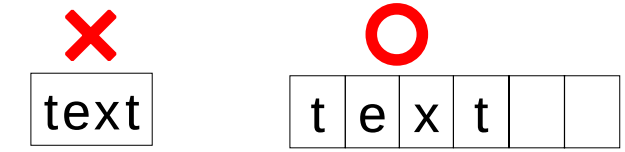
プログラミング演習
文字列操作(1)

阿萬裕久
aman@cs.ehime-u.ac.jp

文字列の取り扱い

- C 言語のデータ型として**文字列**という型は**存在しない**
- **文字(char)の配列**として処理される



1つの文字列　　文字の配列

例題1

- 大きさ 10 の文字(char)型配列 str を用意し，その中に "Ehime" という文字列を格納せよ．ただし，ダブルクォーテーション(")は文字列に含まないものとする．

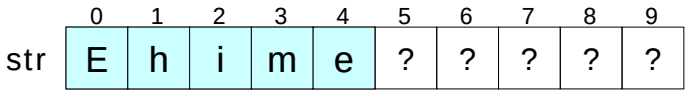
例題1：解説

Web ページのコーディング例を参照

- 順番に文字を代入していけばOK

```
str[0] = 'E';
str[1] = 'h';
・・・・・・
str[4] = 'e';
```

注意！
シングルクォーテーション(')
で文字を囲むこと



例題1：解説

Web ページのコーディング例を参照

- 文字列の**終端を明示**する必要がある
- 特殊文字 '¥0' を使用する（※0はゼロ）

```
str[0] = 'E';
・・・・・・
str[4] = 'e';
str[5] = '¥0';
```

文字列は「**ここまで**」という印になる

	0	1	2	3	4	5	6	7	8	9
str	E	h	i	m	e	¥0	?	?	?	?

例題1：解説

- 終端が無いと printf("%s¥n", str); では

	0	1	2	3	4	5	6	7	8	9
str	E	h	i	m	e	?	?	?	?	?

文字列は**まだまだ続く**ものだと勘違いされる

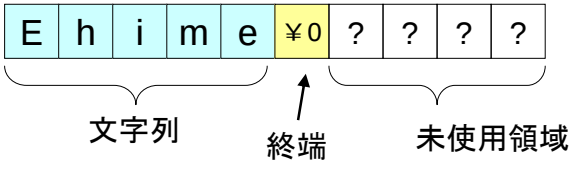
例

```
$ ./a.out
Ehime・・・(・・植 ・hY◎@・・t ・・・hY◎@ ・D◁@t ・・・H ・・・◎L▷@◁
```

表示がここで終わってくれない！
残りは何が出てくるか分からない

例題1：解説

- つまり，***n* 文字の文字列**を表現するには少なくとも ***n*+1 文字分の領域**が必要



例題1：解説

- 文字列をコピーするための関数があるのでこれを利用してもよい

```
strcpy(コピー先, コピー元);
```

例

```
char  str[10];
strcpy(str, "Ehime");
```

プログラムの冒頭に
#include <string.h>
を書く必要あり

この場合はダブルクォーテーション(")を使う．str には最後の ¥0 までが自動で格納される．

例題2

- getchar 関数を使って文字を 1 つずつ順番に読み込み，それらを文字列として生成しなさい．ただし，文字列の長さは 64 未満とする．

例題2：解説

- **getchar** 関数
  - これを1回呼び出すと**キーボードから1文字**だけ読み込む
  - 使用例

```
int  x;
x = getchar();
```

char ではない点に注意

x には読み込んだ文字の**文字コード(整数)**が入る

文字コードとは

（※情報リテラシーの復習ですが…）

- 各文字に割り当てられている整理番号
  - アルファベットや記号に対して一般に使われている割り当て方式は ASCII（アスキー）コード

（例）

```
a ⇔ 97
b ⇔ 98
  ・・・
z ⇔ 122
```


getchar を使った例

- 簡単なプログラム

```
int  x;
x = getchar();
printf("%d¥n", x);
printf("%c¥n", x);
```

「a」を入力

「97」を出力

「a」を出力

データとしては整数であるが，
これを **char 型だと思って**出力（**%c 指定**）すれば
対応する**文字が表示**される！

int → char 変換

- このように実際には文字データは整数値
- では，その整数値を char として扱うには？

気にすることなく char 型変数に代入したらよい

```
int x;
char ch;
x = getchar();
ch = x;
```


なぜ getchar() は int を返すのか

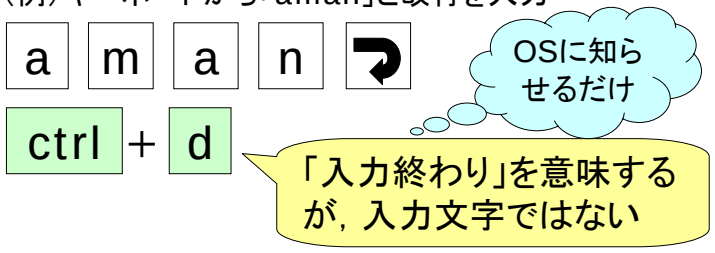
- getchar 関数の実行によって文字コードが得られるのなら，最初から戻り値は int ではなく char でよいのでは？

実は，キーボードからの
「入力終わり」を検出
するための**からくり**がある

入力の終わり

- 「入力の終わり」は文字でない！
- getchar 関数の戻り値が char だと「入力の終わり」を表現できない

（例）キーボードから「aman」と改行を入力



入力の終わりは -1

- getchar() は入力の終わりを検知すると，**文字コードの代わりに -1 を答える**.
  - それゆえ戻り値は char ではなく int
  - -1を文字コードとする文字は無いので char に代入すると都合が悪い
  - ただし，-1 だと意味が分かりにくいので **EOF** という別名が使える．これは（入力）ファイルの終端「End Of File」の意味．

例題2：解説

Web ページのコーディング例を参照

- ポイントは次の6行

②それが EOF なら終わり

12～17行目

```
i = 0;
while ( (c = getchar()) != EOF ){
    str[i] = c;
    i++;
}
str[i] = '¥0';
```

①結果を c に代入

③さもなくば配列 str に入れていく

④最後に文字列の終端を確定

例題3

- fgets 関数を使って文字列を 1 行まとめて読み込み，その長さを出力するプログラムを作りなさい．ただし，読み込む文字列の長さは 64 未満とする．

例題3：解説

プログラムの冒頭に
#include <string.h>
を書く必要あり

- **fgets** 関数
  - 1回呼び出すと，**1行(改行文字まで)**読み込む
  - 使用例

```
char  str[64];
fgets(str, 64, stdin);
```

キーボードから文字列を1行だけ読み込む
文字列は str へ格納される('¥0'も自動追加)
ただし，最大でも 64－1 文字しか読まない

¥0 の分

fgets 関数

- 関数の引数の意味

```
fgets(str, 64, stdin);
```

文字列の格納先　格納先の許容容量　文字列の読み込み元

一般的には読み込み元ファイルへのポインタ
（例）ファイルポインタ fp
FILE＊ fp = fopen("foo.dat", "r");

stdin とは？

- stdin とは**標準入力**(standard input)を意味する
  - まさに標準的な入力元であり，**キーボード**がこれに該当するのが通常
  - 別途にファイルを用意しておいて，キーボードの代わりにそれを読み込ませることもできる

```
./a.out  <  data.dat
```

※標準入力の**リダイレクション**
(redirection＝変更，付け替え)

もしも許容量以上の入力があると

- 仮に 64 文字以上続く行があると
  - str には**最初の 63 文字**しか格納されない
  - その後でもう一度 fgets を呼び出すと読み出しが**64 文字目から再開**される

```
fgets(str, 64, stdin);
```

この数字が読み込みのリミッターである．
これを誤るとメモリ違反を犯してしまうので注意！

例題3：解説

Web ページのコーディング例を参照

- 問題で求められているのは「文字列の長さ」

15～18行目

```
n = 0;
while ( str[n] != '¥0' ){
   n++;
}
```

文字列の終端 ¥0 に遭遇するまで n をカウントアップしていく

例題3：解説

- 別解として **strlen** という関数も使える

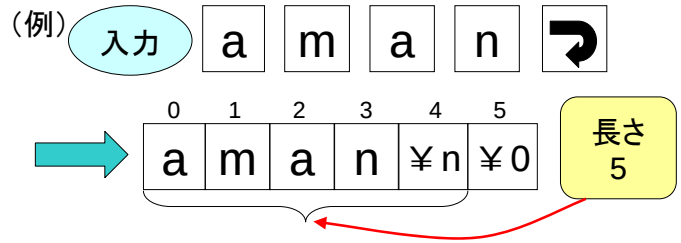
```
int  n = strlen(str);
```

n には文字列 str の長さ
(文字数，¥0 は含まない)が入る

プログラムの冒頭に
#include <string.h>
を書く必要あり

fgets を使う場合の注意

- 文字列を入力する場合，行の終端には改行文字（￥n）がある
- fgets は**改行文字も文字列の一部**として扱うことになっている



なぜ scanf 関数ではダメなのか？

- 「**データ入力は scanf**」という固定観念があるのではないだろうか
  - たしかに scanf は有用な関数
  - しかし，これにも**苦手**はある

それが文字列の読み込み

なぜ scanf 関数ではダメなのか？

- 簡単な例

```
char  str[16];
scanf(“%s”, str);
printf(“%s￥n”, str);
```

実行例
入力）Aman　Hirohisa
出力）Aman

これが
scanf の
抱える問題

途中の空白で読み込みが
勝手に終わってしまう！