基礎プログラミング演習 II 教材 (#6)

■ 補足：配列の範囲

★教科書 p.188「配列の添字に注意する」にも若干の記述あり

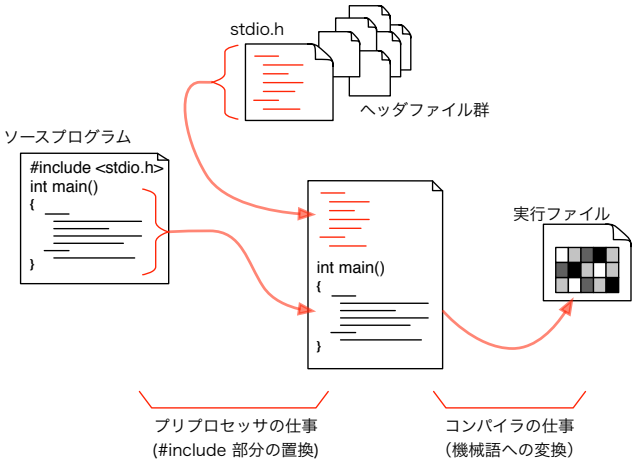
・添字の範囲には注意が必要
・用意した配列の要素数範囲に収める（ `int array[5];` なら添字は 0〜4 まで）
・範囲を超えてアクセスした場合でも実行時にエラーは出ない
・コンパイルエラーは出ない
・実行時エラーは出る場合もあるが、偶然正常に動作してしまうかも知れない

右のプログラムは正常に動作するはずです。しかし例えば a[5] に値を代入した場合、何が起きるか試してみると良いでしょう。添字を巨大な値にした場合はどうでしょう。マイナスにした場合何が起きるでしょう。
つまり C 言語ではこれはプログラマの責任範囲なのです。

```
int x1, x2, a[5], y1, y2;
x1=100; x2=101;
y1=200; y2=201;
a[0]=1;
a[1]=2;
a[2]=3;
a[3]=4;
a[4]=5;
printf("x1=%d, x2=%d, y1=%d y2=%d\n",
        x1, x2, y1, y2);
```

■ 文法的落ち穂拾い：include

C 言語の処理系では、cc コマンドによるコンパイル処理は本来の「等価な振る舞いをする機械語への変換処理」だけでなく、その前に #include や #define の処理が含まれています。この前処理を実行するプログラムをプリプロセッサと呼んでいます。
#include は指定されたヘッダファイルをソースプログラムに取り込むための指示です。ヘッダファイルには各種の定義が書かれており、それらはコンパイル作業に必要な情報です。
つまりプログラマが書いたコードはまずプリプロセッサによって#include や#define の処理が施されてコンパイルされます。



押さえて欲しいポイント：
・プログラムは多くの段階を経てコンパイルされる
・プリプロセッサと呼ばれる前処理機構がある
・#include はそこでヘッダファイルを差し込むものである
・ヘッダファイルには printf( ) などの関数定義があり、それらの利用のために include が必要
・ちなみに #define もプリプロセッサ命令

興味のある人へ：
ヘッダファイルはおよそ以下のディレクトリにあります。覗いてみると良いでしょう。
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk/usr/include

■ 文字・文字列

★教科書 p.49 「型」の表 3-1 に文字型の型名、サイズ、値の範囲について説明がある。
教科書にはほとんど char 型についての説明がない。右にサンプルのコードと実行結果を示す。

```
// 文字型変数の宣言
char a, b;
// 代入
a='Z'; // 引用符で囲む。一文字だけ。
b=a;
// 変換文字は %c
printf("a=%c, b=%c\n", a, b);
```

```
$ ./a.out
a=Z, b=Z
$
```

押さえて欲しいポイント：

- 文字とは「a」「1」など一文字のことを指す（数字の 1 を表現する文字もある）
- 文字を格納するために char 型が用意されている
- 文字定数は ‘a’ などシングルクォートで囲む
- エスケープ文字による特殊記号の表現ができる
  （ a='\n'; などとして改行文字を扱える）

□ 文字列の扱い

★教科書 p.203「文字列と配列の関係を知る」を参照
押さえて欲しいポイント：(p.203～208 まで)
・文字列とは「abc」「word sample」など複数の文字の集まりを指す
・文字列型はなく、文字型 char の配列として処理する
・終端文字（C 言語では NULL 文字）に注意
・変数として用意する文字型の配列は、必要な文字数+1 の長さを用意する（NULL 文字のため）

文字列定数の扱いには注意が必要です。
・文字列型の変数はないが、文字列定数はある
・文字列定数は NULL 文字のために一バイト長く用意される
・文字配列の初期化の方法に注目（p.205 冒頭）
・文字列を扱う機能は関数として実現されている（p.205 中央。代入はできない。連結演算子はない。）

□ 実験：文字数をカウントする

右のプログラムは scanf によって入力された文字列の文字数を出力するものです。内容を吟味して実行してください。

（自分の想像どおりの結果が出ますか？scanf の %s が一行の入力ではなく分かち書きされた単語だけを取ってくることに気がつきましたか？）

```
char string[100];
int length;

scanf("%s", string);

length=0;
while( string[length] != '\0' ) {
  length++;
}
printf("%d\n", length);
```

なお string は要素数 100 としてみました。これが溢れると実行時エラーとなる場合があるので宣言する要素数には注意を払うよう習慣づけましょう。文字配列に対する scanf では &string とはせず、単に string と書きます。理由は配列とポインタの関係を学ぶまで分からないでしょうから基礎プロ II では丸覚えして下さい。

□ 課題 1.

同様に scanf で入力された文字列のうち、英字（a-z, A-Z)、数字、それ以外を個別にカウントするプログラムを作ってください。

```
$
echo 'test=123' | ./a.out
Alpha=4 Number=3 Else=1
$
```

考え方：
・英字の判定は ‘a’ 以上 ‘z’ 以下あるいは ‘A’ 以上 ‘Z’ 以下で良いか？
（文字型変数は `if(str[0] >= 'a')` のようにして比較ができます。2 ページ先にコード表があります。）
・興味のある人は標準文字処理関数（教科書 p.466 ctype.h の欄）に英字や数字判定関数も参照してください。

□ 初期化・空文字列

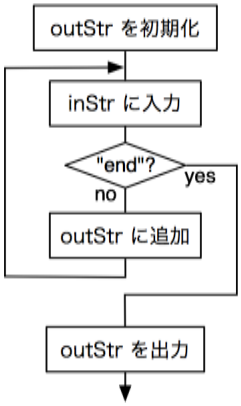
教科書 p.204 から文字列の配列を初期化する方法が示されています。
またそこには記述がありませんが、空（くう）の文字列を想像することができますか？つまり文字配列の中身が終端文字（NULL 文字）だけの状態です。
定数として表現する際には「""」と二重引用符を続けて二つ書きます。（"A"と書けば中身が A と終端文字だけの文字型配列（つまり要素数は 2）が出来上がるのですから、"" と書けば空文字列が出来上がります。）

□ 標準文字列処理関数

★教科書 pp.326-334 まで「文字列の操作」の「標準ライブラリ関数を使う」から
C 言語は文字列を処理する機能をほとんど言語自体にもっていません。その代わり一般的な文字列処理を実現する関数が幾つも用意されています。
これらのライブラリ関数を利用するためには string.h のインクルードが必要です。
他にも strncpy, strncat, strncmp などがあります。興味があれば調べると良いでしょう。

□ 課題 2.

入力した文字列を連結して、最後にまとめて連結した文字列として表示するプログラムを作ってください。end と入力されたら終了することにします。
例えば「hello」「dear」「my」「friend」「end」と入力すると「hellodearmyfriend」と表示します。



処理の流れを右図に示しておきます。
・蓄積・出力用文字列変数（右例 outStr）は ""（空文字）で初期化
・文字列の追加は strcat 関数が使えるでしょう

□ 課題 3.

課題 2. を改良して、結果が「hello dear my friend」と単語ごとに区切られて表示されるようにしてください。また、まだ end と入力されなくても、結果が 20 バイトに届くようになったらその時点で出力してしまってください。
（画面の端まで行かずに折り返して表示するプログラムを作っていると思えばいいでしょう。）

```
$ ./string6
string >> hello
string >> dear
string >> my
string >> friend,
# hello dear my#
string >> welcome
string >> back.
#friend, welcome#
string >> end
#back.#
$
```

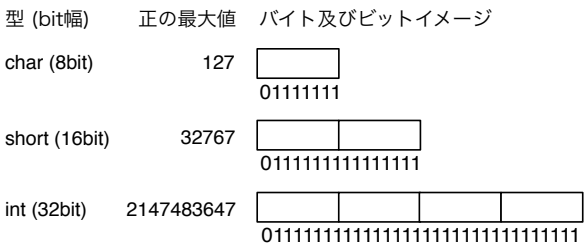
右図の出力（# で囲まれた行）がに注目してください。「hello」「deear」「my」まで連結されたところで、次の「friend」を加えると 20 バイトを超えてしまうので、まず「hello dear my」までを出力し、「friend」は次の出力の先頭になっています。

課題 3. でやったように、まずデータを処理する手順を検討し（必要であれば流れ図を描き）、それからプログラム修正に取りかかると良いでしょう。

■ データのサイズ

★ 教科書 p.49, 表 3-1 参照

変数にせよ定数にせよ、C 言語が扱う値には型があり、型によって利用するメモリの量が異なります。
メモリ量の違いはそのまま表現できる値の幅に直結します。
教科書の表 3-1 あるいは右図にあるように、例えば変数は割り当てられたデータの量によって表現できる値の限界が決まります。



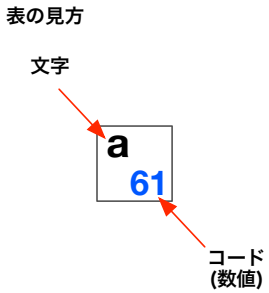
文字も内部的には数値で表現するため、数値同様の大小比較ができます。(次節を参照)

■ 文字コード

教科書には文字型について書かれていました。以下の点について注意しましょう。

- メモリには値が入る
  (値しか入らない)
- 文字は数値として入れる
  (そのメモリが文字型だとしたら、「a」はこの値としよう)
- これをコードと呼ぶ
- 文字と値との割り付け(マップ)表が必要(コード表)
- コードの体系は複数ある
  (参考:アスキー文字コード表)

ASCII 文字コード表



ASCII文字は 1 バイトで表現されているが、その実体は数値である。
つまり 'a' は番号 61 の文字。
61 は 16 進数表記なので、10進数で表記すると 97 番文字となる。

・20番(16進)の sp は空白文字。
・\t はタブ。
・\n は改行文字。
・\0 はいわゆるヌル文字。
(文字列の終端記号)

\0 00	10	sp 20	0 30	@ 40	P 50	` 60	p 70
01	11	! 21	1 31	A 41	Q 51	a 61	q 71
02	12	" 22	2 32	B 42	R 52	b 62	r 72
03	13	# 23	3 33	C 43	S 53	c 63	s 73
04	14	$ 24	4 34	D 44	T 54	d 64	t 74
05	15	% 25	5 35	E 45	U 55	e 65	u 75
06	16	& 26	6 36	F 46	V 56	f 66	v 76
07	17	' 27	7 37	G 47	W 57	g 67	w 77
08	18	(28	8 38	H 48	X 58	h 68	x 78
\t 09	19	) 29	9 39	I 49	Y 59	i 69	y 79
\n 0a	1a	* 2a	: 3a	J 4a	Z 5a	j 6a	z 7a
0b	1b	+ 2b	; 3b	K 4b	[5b	k 6b	{ 7b
0c	1c	, 2c	< 3c	L 4c	\ 5c	l 6c	| 7c
0d	1d	- 2d	= 3d	M 4d	] 5d	m 6d	} 7d
0e	1e	. 2e	> 3e	N 4e	^ 5e	n 6e	~ 7e
0f	1f	/ 2f	? 3f	O 4f	_ 5f	o 6f	7f

□ 参考:sizeof( ), %x

教科書 p.92 参照。
sizeof 関数は引数に与えた変数やデータのバイト数を答えます。
また、printf 関数の変換文字として %x があります。データの中身を 16 進数で出力します。
ともに変数の内部表現(コンピュータ内部でどのように実現されているか)を調べるのに便利です。

右のプログラムは、文字型(char)は 1 バイトで、A は 16 進数で 41 であること、int 型は 4 バイトで、そこに格納された整数値 100 は 16 進数で 64 (6*16 + 4 = 100) として表示されることを確認している。

プログラム:
```
char a='A';
int i=100;
printf("%ld - %x\n", sizeof(a), a);
printf("%ld - %x\n", sizeof(i), i);
```

結果:
```
1 - 41
4 - 64
```