文字処理 (その 1)

山本昌志*

2004 年 12 月 10 日

1　復習と本日の学習内容

コンピューターは、プログラムに従いデータを処理する。CPU がデータを処理するためには、データをメモリーの中に格納する必要がある。このメモリーの中に格納されたデータを取り扱う方法がいろいろあり、それらを称してデータ構造という。c 言語では、プログラムが便利にできるようにいろいろなデータ構造が用意されている。今までは、単純型と配列型のデータ構造を学習した。

本日は、文字の取り扱いの学習をする。文字と言えども、メモリーに格納しなくてはならないのは数値と同じである。これを学習するためには、以前学習したデータ構造を理解している必要がある。以前学習したデータ構造は、

単純型	int a	1 個のデータを格納
配列型	int b[100]	複数の同じ型のデータを格納

である。

これまでに、整数型 (`int`) と倍精度実数型 (`double`) を学習して、プログラムに作成してきた。本日は、文字型のデータの取り扱い方法を学習する。

2　コンピューター内部での文字の表現

2.1　英数字

以前も述べたように、コンピューター内部では、全てのデータは数字とし取り扱われている。文字を扱う場合、それと整数との対応関係が決められている。決められた関係は、コード表にまとめられている。例えば、英数字の場合、アスキーコード表というものがあり、世界中で使われている。参考資料の表 2 にアスキーコードを示しておく。この表から分かるように、英数字の各々には、0〜127 の整数が割り当てられているのである。

コンピュータープログラムで、アスキーコードの値を確かめたければ、以下のようにすればよい。

*国立秋田工業高等専門学校　電気情報工学科

```
#include <stdio.h>
  int main(void){
  char c;

  c='L';

  printf("L = %d\n",c);

  return 0;
}
```

この例で分かるように、文字型変数 (char) を用いて、文字を格納することができる。しかし、この文字型の変数は、-128〜127 まで[1]の変数しか格納できない[2]ことになっている。従って、文字型の変数で表現できるのは 256 種類の文字に限られる。アルファベットを使う文化圏では、それで問題ない。多くのコンピューターでは、英数字はこのアスキーコードが使われる。0〜255 と言うのは、16 進数で 2 桁である。

ここで、理解して欲しいのは、文字は整数に置き換えられると言うことである。コンピューター内部では、この整数を扱うことにより文字の処理をしている。そして、文字と整数の対応を示したものがコード表である。

ここは少し難しいが、とりあえず説明しておく。文字型の単純データ構造では、1 バイトのデータが格納できる。1 バイトは 8 ビットで、0〜255 の整数が格納できる。255 は、$2^8 - 1$ である。

2.2 日本語

中国語や日本語のように多くの文字を使う文化圏では、256 種類しか表現できないコードは使い物にならない。

その解決のために、日本語はアスキーコードとは異なる文字コードが使われる。代表的なのが、

日本語 euc	extend unix code の略で、主に UNIX で使われる。
shift-jis	主にパソコン (DOS や Windows, machintosh) で使われている。
JIS コード	日本工業規格 JIS(Japanse Industrial Standards) が決めたコード。
ユニコード	漢字を含む世界のすべての文字を全部表現できるコード。

である。いろいろあるが、すべて文字と整数との対応が決められている。ただし、コード毎にその対応は異なる。例えば、「秋」という漢字は、表 1 のようになっている。

[1] 処理系によっては、0〜255 のものもある。
[2] これを 8 ビット、1 バイトと言う。詳しくは、2 年生以降に学習。

表 1: ”秋”をそれぞれの日本コードで表す。c 言語の場合、先頭に 0x とつく数字は 16 進数を表す。

コード	16 進数	10 進数
日本語 EUC	`0xbda9`	48553
shift-jis	`0x8f48`	36680
jis コード	???????	????????
ユニコード	???????	????????

英数字と日本語では、1 文字を表現するために必要な情報量が異なる。

- 英数字は、1 バイトで十分である。
- 日本語 (ユニコードを除く) は、2 バイト必要である。

3 文字の記憶方法

文字をコンピューターのメモリーに格納させるためには、その文字が持つ情報量を考えなくてはならない。先に述べたように、1 文字記憶するためには英数字では 1 バイト、日本語では 2 バイトが必要である。これは、記憶させるための箱 (変数) の大きさが、日本語では英語の 2 倍必要と言うことである。

通常の文字の処理では、1 文字では余り役に立たない。普通の文章は、文字が連なって、一つの情報の固まりとなっている。このように文字が連なったものを、文字列と言う。この様な場合、文字型の配列を使うことになる。

このようなことから、文字の状態によって、記憶方法を変える必要がある。これは、プログラマーに対して、厳しいことを要求しているが、仕様がそうなっているので仕方ない。少しだけ、ハードウェアーの知識が必要と言うことである。

ここでは、このような文字をメモリーへの格納方法を示す。

3.1 英数字 1 文字の場合 (文字型変数)

英数字 1 文字の場合、これは単純で文字型変数を用いる。次のように宣言をすれば、変数名 `hoge` の文字を入れる入れ物が用意される。この入れ物の容量は、1 バイトである。

```
char hoge;
```

型名の `char` と言うのは、character(文字) の略である。

この変数に、文字を格納するは簡単で、

```
hoge = 'A';
```

のよにすればよい。要するに、格納したい文字をシングルクォーテーションで囲んで、代入演算子 (=) を使えばよいのである。

文字型変数のイメージで表を、図 1 に示す。

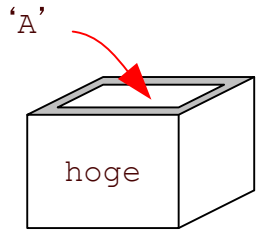

図 1: 文字型変数 hoge を用意して、それに'A' を格納。この箱の大きさは、1 バイトなので、日本語を入れることはできない。

表示まで含めた文字型変数を使ったプログラムは、次のようになる。文字型変数を表示させるためには、変換指定子%c を用いる。c は、character の略であろう。

```
#include <stdio.h>
int main(void){
  char hoge;              /* 文字型の変数 */
  hoge='A';               /* 代入 */
  printf("%c\n",hoge);    /* 表示 */
  return 0;
}
```

3.2 英数字の文字列の場合 (文字型配列)

3.2.1 文字型配列の使い方

次に、いくつかの英数字で構成される文字列の場合である。このようなときは、文字型の配列のデータ構造を使う。以前、同じ型の数値データが複数ある場合、int 型あるいは double 型の配列を使ったのと同じである。

文字型の配列を使うときには、次のように宣言する。

```
char hoge[10];
```

これで、10 個の文字を格納できるメモリーの領域が確保できる。しかし、実際に入れることができる文字数は、9 文字である。文字列の最後の文字の後に、文字列終了の記号を入れるため、1 文字分少なくなる。文字列終了の記号は、'\0' という文字が使われる。これは、アスキーコードの 0 番の NUL[3]のことである。

いろいろな方法で、この文字型の配列に文字を格納することができる。

[3] ヌルと読む

- まずは、教科書の p.190 のように、配列の要素毎に文字を入れる方法がある。

```
hoge[0]='A';
hoge[1]='k';
hoge[2]='i';
hoge[3]='t';
hoge[4]='a';
hoge[5]='\0';
```

  実際、この方法で文字を代入することはまれである。文字列のある特定の要素を操作するには有効である。

- 文字列を操作するライブラリーを用いて、配列に文字を代入することができる。これは、教科書の p.217 に書いてある方法である。具体的には、次のようにする。

```
strcpy(hoge, "Akita");
```

  これで、配列に文字が格納できる。ただ、この方法を使う場合は、プログラムの最初に`#include <string.h>`とライブラリーをインクルードする必要がある。

- 最後は、私が好んで使う方法である。多くのプログラマーもこれを使っているだろう。これは簡単で、次のようにする。

```
sprintf(hoge, "Akita");
```

  `printf` や `fprintf` 関数とよく似ている。

教科書に載っている方法と、私が使う方法を示したが、これ以外の方法も存在するであろう。3 番目の方法がお勧めであるので、これはしっかり憶えるように。

文字型配列のイメージは、図 2 のようになる。

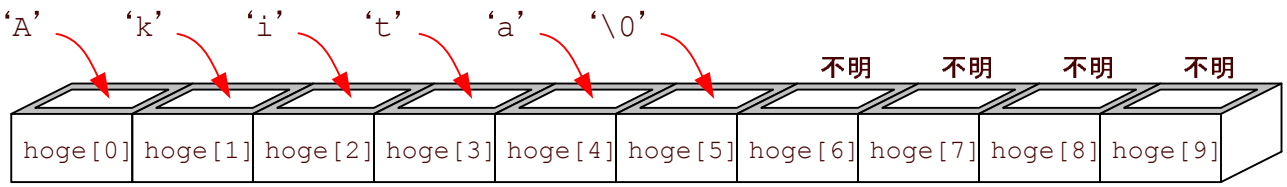

図 2: 文字型の配列 `hoge[10]` を用意して、それに"Akita"を格納。一つの箱には、一つの英数字しか入れられない。そして、最後に `\0` が入る。

表示まで含めた文字型変数を使ったプログラムは、次のようになる。配列に格納された文字列を表示させるためには、変換指定子`%s` を用いる。`s` は、string(ひも、一続き) の略であろう。

```
#include <stdio.h>
int main(void){
  char hoge[10];              /* 文字型の変数 */
  sprintf(hoge, "Akita");     /* 代入 */
```

```
  printf("%s\n",hoge);       /* 表示 */
return 0;
}
```

3.2.2 注意

配列を使い場合、そのサイズはプログラマーが決めなくてはならない。そのサイズを超えて、代入をするような操作をすると、エラーメッセージを出して止まる。あるいは、コンピューターがクラッシュすることもあり得る。従って、通常プログラムを作成するときには、十分大きい配列を用意するのが普通である。

配列のサイズを超えた場合、実際の動作は処理系に依存する。通常は、「segmentation fault」あるいは「セグメンテーション違反です」とかのメッセージを出して、プログラムは止まる。

3.2.3 ポインターを使う方法

最後に余談であるが、ポインターというものを使えば、次のようなこともできる。

```
#include <stdio.h>
int main(void){
  char *hoge;            /* 文字型のポインターを用意 */
  hoge="Akita";          /* 代入 */
  printf("%s\n",hoge);   /* 表示 */
  return 0;
}
```

ポインターについては、2 年生で学習するので、ここでは、こんな方法もあるのかという程度のことで良い。

3.3 日本語の場合

3.3.1 文字型配列の使い方

文字型の場合、それに格納できるデータは 1 バイトである。一方、日本語の場合、文字は 2 バイトで表現する。文字型が 2 バイトであれば問題ないのであるが、コンピューターを発展させたのが米国であるため、仕様は 1 バイトになってしまった。そこで、日本語を扱う場合、少しばかり考えなくてはならない。

この問題を解決するために、日本語では 2 個の文字型のデータ領域に 1 個の文字を格納しているのである。なんか、「泥縄式」の解決法に思えるが、実際そうなっているのである。結構、コンピューターの世界もいい加減である。

実際に、文字型の配列に日本語の文字を格納する方法は、アルファベットとほとんど同じである。先に示したアルファベットの 2 番目と 3 目の方法である次の方法が使える。

```
strcpy(hoge,"秋田");
sprintf(fuga,"秋田");
```

しかし、hoge[0]=’秋’; のような代入はできないことには気を付けなくてはならない。hoge[0] は 1 バイトで、日本語の「秋」は 2 バイトだらかである。文字型配列のイメージは、図 3 のようになる。

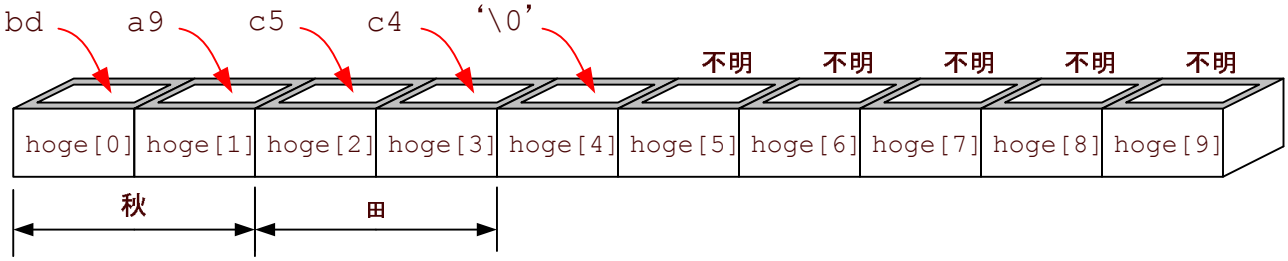

図 3: 文字型の配列 hoge[10] を用意して、それに”秋田”を格納。日本語の場合、2 つの箱で 1 つの文字が入る。そして、最後に \0 が入る。

表示方法もアルファベットと同じである。ただし、変換指定子の%c が使えないことには注意が必要である。

```
#include <stdio.h>
int main(void){
  char hoge[10];
  sprintf(hoge, "秋田");
  printf("%s\n",hoge);
  return 0;
}
```

3.3.2 注意

ここでも、配列を使うため、そのサイズに気を付ける必要がある。アルファベット同様、余裕を持って、配列を確保しなくてはならない。ただ、アルファベットと異なり、文字数の 2 倍と、文字列の終わりを示す \0 が格納できるサイズが必要である。

4 来週の授業

来週も、教室で講義を行う。講義の内容は、以下の通りである。

1. 文字列のファイルの取り扱い方法
2. 文字列処理のためのライブラリー
3. 練習問題

講義を受ける前に、本日配布したプリントを、教科書の第 6 章を 2〜3 回程度、読み返すこと。

5　参考資料

5.1　アスキーコード

教科書にアスキーコード表が載っていないので、表 2 に載せておく。この表から数字 (10 進数) を計算する方法は、次のようにする。

1. 対応する文字の上位 3 ビットの値を探す。
2. 次に下位 4 ビットを探す。
3. 対応する整数の計算を行う。計算方法は 16×(上位 3 ビット)+(下位 4 ビット) である。ただし、A=10, B=11,$\cdots$,F=15 である[4]。

具体的な例で表すと、大文字の'L' は、4C なので

$$16 \times 4 + 12 = 76 \tag{1}$$

となる。

表 2: アスキーコード表。表の行 (0〜7) は上位 3 ビットで、列 (0〜F) は下位 4 ビットを表す。表中の 2 文字以上のものは文字ではなく、制御コードと呼ばれる特殊文字である。

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	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	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

[4] 0〜F まで用いて数値を表す方法を 16 進数と言う。2 年生で詳細を学習する。

5.2 バイトとビット

ここでは、位取り記数法の話を少ししておく。詳細は、2 年生で学習するので、雰囲気が分かれば良い。

人間は 10 進数を使うが、コンピューター内部では 2 進数が使われている。また、実際のプログラマーは、2 進数との対応がつきやすい 16 進数を好んで使う。それぞれの対応は、表 3 のようになる。

ここでは、以下のことを憶えて欲しい。

- 2 進数の 1 桁を 1 ビット (bit) と言う。これで、0 と 1 の整数を表すことができる。
- 2 進数の 8 桁、即ち 8 ビットで、1 バイト (byte) になる。これは、10 進数だと、0〜255 までの整数を表すことができる。16 進数だと、0〜`ff` までの整数を表せる。
- 2 バイトの場合、10 進数だと、0〜65535 までの整数を表すことができる。16 進数だと、0〜`ffff` までの整数を表せる。

実際のコンピューターの世界では、バイトにキロバイトとかメガバイトとかの補助単位がつく。これは、自然科学で使われる補助単位と少し異なるので、注意が必要である。

キロ	KB	$2^{10} = 1024$ バイト
メガ	MB	$2^{20} = 1048576$ バイト
ギガ	GB	$2^{30} = 1073741824$ バイト

表 3: 基数を変えた場合の整数の表し方

10 進数	2 進数	16 進数
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c
13	1101	d
14	1110	e
15	1111	f
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
⋮	⋮	⋮
255	11111111	ff
⋮	⋮	⋮
65535	1111111111111111	ffff
⋮	⋮	⋮