C 言語 013 文字配列

【文字配列】

char 型変数は ASCII コード 1 文字しか記録できません。文字列（複数の文字）を記録するには char 型の配列を使用します。文字列を扱う char 型配列を「文字配列」と称します。

注意点として、例えば 5 文字の文字列を記録したければ、文字配列の要素数は 6 個必要です。

C 言語では文字列の最後に文字列の終わりを示す「¥0」（NULL 文字）が必要で、実際の文字数よりも 1 文字分多くなるからです。

```
char str[6] = "abcde";
```

文字列は値を"（ダブルクォーテーション）で括ります。

文字列の場合、末尾には NULL 文字が自動的に付きます。

文字配列 str の値

添字	0	1	2	3	4	5
値	a	b	c	d	e	¥0

```
char str[11] = "あいうえお";
```

日本語等の全角文字は 1 文字 2 バイトになります。

5 文字の日本語を記録したいなら 5×2＋1＝11 バイト以上、つまり要素数 11 個以上が必要になります。

＜文字列の代入＞

文字配列の宣言時であれば上記のように初期化ができますが、そうでなければ、下記のように 1 文字ずつ代入することになります。

```
char str[4];
str[0] = 'a';
str[1] = 'b';
str[2] = 'c';
str[3] = '¥0';    /* 最後に NULL 文字を代入する */
```

文字は値を'（シングルクォーテーション）で括ります。

文字配列 str の値

添字	0	1	2	3
値	a	b	c	¥0

もしくは「sprintf 関数」「strcpy 関数」を使用する方法があります。

＜scanf 関数を使用した文字列の入力＞

```
char str[10];        /* 文字配列 */
printf("文字列の入力¥n");
scanf("%s", str);    /* 型の指定は s、配列はポインタなので&は付かない */
printf("入力した文字列：%s¥n", str);
```

9 バイトまでの文字列を入力して、変数 str に記録します。

【文字列関連の良く使用される関数】

＜sprintf 関数＞

定　　義	int sprintf(char* str , const char* format , ...);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdio.h
説　　明	printf 関数の書式指定子を使用して str に文字列を代入する関数。 末尾に自動的に NULL 文字が入る。
戻 り 値	正常：代入した文字列のバイト数／異常：負数
例　　文	char str[10]; sprintf(str, "%s", "abc");　/* str に「abc」を代入する */

＜strcpy 関数＞

定　　義	char* strcpy(char* str1 , const char* str2);
ﾍｯﾀﾞｰﾌｧｲﾙ	string.h
説　　明	str1 に str2 の文字列（NULL 文字を含む）をコピーする関数。
戻 り 値	コピー先の文字列のアドレス。
例　　文	char str[10]; strcpy(str, "abc");　/* str に「abc」を代入する */

＜strcmp 関数＞

定　　義	int strcmp(const char* str1 , const char* str2);
ﾍｯﾀﾞｰﾌｧｲﾙ	string.h
説　　明	str1 と str2 の文字列を比べる関数。
戻 り 値	引数が同じ文字列であれば「0」、str1 の文字コードが str2 の文字コードより大きければ「1 以上」、小さければ「-1 以下」が返る。
例　　文	char str[] = "abc"; if(strcmp(str, "abc") == 0){　/* str は「abc」か？真になる */

＜strlen 関数＞

定　　義	size_t strlen(const char* str);
ﾍｯﾀﾞｰﾌｧｲﾙ	string.h
説　　明	引数 str のサイズ（バイト数）を取得する関数。 全角文字は 1 文字 2 バイトで計算され、NULL 文字はサイズに含まれない。
戻 り 値	文字列のバイト数。
例　　文	char str1[] = "abcde"; char str2[] = "あいうえお"; size_t size1 = strlen(str1);　　/* size1 は 5 バイト */ size_t size2 = strlen(str2);　　/* size2 は 10 バイト */

＜sscanf 関数＞

定　　義	int sscanf(const char* str, const char* format, ...);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdio.h
説　　明	scanf 関数の書式指定子を使用して str を分解し、各変数に代入する関数。
戻 り 値	正常に代入することができた値の個数を返す。

＜例文 1＞

```
char str[] = "2018/2/14";
int ret, year, mon, day;
ret = sscanf(str, "%d/%d/%d", &year, &mon, &day);
```

変数の値
ret→3
year→2018
mon→2
day→14

＜例文 2＞

```
char str[] = "10,10.5,abc";
int     ret;
int     var1;
double  var2;
char    var3[10];
ret = sscanf(str, "%d,%lf,%d", &var1, &var2 ,&var3);
```

変数の値
ret→2（1 個正常に取得できない）
var1→10
var2→10.5
var3→異常データ

var3 は文字列なので書式指定子は%s になります。「"%d,%lf,%d"」を「"%d,%lf,%s"」にすれば正常に文字列を分解して、全変数に正常値を代入することができます。

＜例文 3＞

```
char str[] = "A 太郎 開発部";
char name[10];
char post[10];
sscanf(str, "%s %s", name, post);
printf("%s¥n", name);
printf("%s¥n", post);
```

＜実行結果＞
A 太郎
開発部

```
char str[] = "A 太郎/開発部";
char name[10];
char post[10];
sscanf(str, "%s/%s", name, post);
printf("%s¥n", name);
printf("%s¥n", post);
```

＜実行結果＞
A 太郎/開発部
（不定値が出力）

書式指定子に%s（文字列）を指定した場合、スペースかタブ区切り以外では正しく値が取得できません。
これは「scanf 関数」「fscanf 関数」にも該当します。
文字列を,（カンマ）や/（スラッシュ）区切りで分解したい場合は下記の「strtok 関数」を使用します。

＜strtok 関数＞

定　　義	char* strtok(char* str1, const char* str2);
ﾍｯﾀﾞｰﾌｧｲﾙ	string.h
説　　明	str1 を str2 で区切って分解していく関数。
戻 り 値	分解した文字列のアドレス。

＜例文＞

```
char  str[] = "aaa,bbb,ccc";
char* cell;                  /* ポインタなのに注意 */
cell = strtok(str, ",");     /* ,（カンマ）で文字列を分解する */
printf("%s¥n", cell);
cell = strtok(NULL, ",");    /* 同じ変数 str を読む場合、2 回目以降は「NULL」にする */
printf("%s¥n", cell);
cell = strtok(NULL,",");
printf("%s¥n", cell);
```

＜実行結果＞

```
aaa
bbb
ccc
```

＜strtol 関数＞

定　　義	long strtol(const char* str, char** error, int base);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdlib.h
説　　明	str を long 型に変換する関数。 error には、変換できなかった文字列の格納先のアドレスを指定。NULL を指定した場合は変換できなかった文字列を取得できない。 base は基数を 0～36 で指定。2 であれば 2 進数、16 であれば 16 進数になる。 0 の場合は定数表記（str の形式通り）になり、先頭に 0 が付けば（例：010）なら 8 進数、0x が付けば（例：0x10）であれば 16 進数になる。
戻　り　値	変換に成功した場合は変換後の数値、失敗した場合は 0 が返る。

＜例文＞

```
char str[] = " *** ";    /* ここに下記の 4 パターンを実行してみる */
char* error;
long ret = strtol(str, &error, 0);    /* str を定数表記で変換 */
printf("%s¥n", error);
printf("%ld¥n", ret);
```

文字列	実行結果	解説
char str[] = "100";	100	変換成功 error の値は¥0 のみ。
char str[] = "str";	str 0	変換失敗
char str[] = "100str";	str 100	変換成功 最初の 100 だけが変換される。
char str[] = "str100";	str100 0	変換失敗

＜strtod 関数＞

定　　義	double strtod(const char* str, char** error);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdlib.h
説　　明	str を doubl 型に変換する関数。 error には、変換できなかった文字列の格納先のアドレスを指定。NULL を指定した場合は変換できなかった文字列を取得できない。
戻　り　値	変換に成功した場合は変換後の数値、失敗した場合は 0 が返る。