

文字列処理

三池 克明

char 型配列である文字列の操作について解説します。
覚えるというよりは必要に応じて調べられるようにしておけば良いでしょう。

目次

1.	入力した文字列を得る.....	1
1.1.	関数 scanf() を使う.....	1
1.2.	関数 gets() を使う.....	2
2.	文字列を数値に変換.....	4
2.1.	関数 atoi().....	4
2.2.	関数 atof().....	7
3.	文字列操作.....	9
3.1.	strlen()...文字列の長さを得る.....	9
3.2.	strcpy()...文字列の代入.....	11
3.3.	strcmp()...文字列の比較.....	13
3.4.	strcat()...文字列の連結.....	15
3.5.	sprintf()...書式指定された文字列の生成.....	17
3.6.	sscanf()...文字列から書式に従ってデータを取り出す.....	19
3.7.	strncpy()...最大長さを指定した strcpy() 関数.....	21
4.	文字列処理関数一覧.....	23
5.	演習問題.....	24

1. 入力した文字列を得る

1.1. 関数 scanf() を使う

まずは関数 scanf() を使ったプログラムを作ってみましょう。

scanf5.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: main()
6: {
7:     char    s[SIZE];
8:
9:     printf("文字列を入力して下さい。¥n");
10:    scanf("%s", s);
11:
12:    printf("あなたが入力した文字列は「%s」です。¥n", s);
13: }
```

実行例 1

文字列を入力して下さい。

hogehoge

あなたが入力した文字列は「hogehoge」です。

実行例 2

文字列を入力して下さい。

hoge hoge

あなたが入力した文字列は「hoge」です。

おや？ 実行例 2 を見ると文字列が途中で切れてますね。

実は関数 scanf() で文字列を得る場合、半角スペースの手前で入力を止めてしまいます。これは scanf() の仕様です。

1.2. 関数 gets() を使う

関数 scanf() では半角スペースを含む文字列を得ることができません。
そこで半角スペースも文字列として得るには関数 gets() を使います。

以下は関数 gets() を使ったプログラムです。
入力、コンパイル、実行してみましょう。

gets1.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: main()
6: {
7:     char    s[SIZE];
8:
9:     printf("文字列を入力して下さい。¥n");
10:    gets(s);
11:
12:    printf("あなたが入力した文字列は「%s」です。¥n", s);
    }
```

実行例 1

文字列を入力して下さい。

hogehoge

あなたが入力した文字列は「hogehoge」です。

実行例 2

文字列を入力して下さい。

hoge hoge

あなたが入力した文字列は「hoge hoge」です。

半角スペースを含めた文字列を得ることができるようになりました。

なお、関数 `gets()` の構文は以下の通りです。

構文：

```
gets(文字列)
```

ヘッダファイル：

```
stdio.h
```

関数 `gets()` はキーボードで入力した文字列を `char` 型配列に格納します。

また格納した文字列の最後にヌル文字を追加します。

よって `char` 型配列のサイズは最低でも[入力した文字数+1]でなければなりません。

2. 文字列を数値に変換

関数 `gets()` など得た文字列の内容を数値に変換する方法について解説します。

2.1. 関数 `atoi()`

関数 `atoi()` は文字列を `int` 型の値に変換します。

以下はその例です。

`atoi.cpp`

```
1: #include <stdio.h>
2: #include <stdlib.h>
3:
4: #define SIZE    128
5:
6: main()
7: {
8:     char    s[SIZE];
9:     int     v;
10:
11:     printf("数字を入力して下さい。¥n");
12:     gets(s);
13:     v = atoi(s);
14:
15:     printf("文字列「%s」は整数%d と変換されました。¥n", s, v);
16: }
```

実行例 1

数字を入力して下さい。

-234

文字列「 -234 」は整数 -234 と変換されました。

実行例 2

数字を入力して下さい。

hoge123

文字列「hoge123」は整数 0 と変換されました。

実行例 3

数字を入力して下さい。

123qw

文字列「123qw」は整数 123 と変換されました。

実行例 4

数字を入力して下さい。

3.14

文字列「3.14」は整数 3 と変換されました。

1 文字目が符号や数字でない文字だと「0」となるようですね。また小数点以降の文字も無視されるようです。

またソースコードの 2 行目をみてください。

```
1: #include <stdio.h>
2: #include <stdlib.h>
3:
```

このように関数 `atoi()` を使用するには「`#include <stdlib.h>`」と `stdlib.h` をインクルードしなければなりません。

また関数 `atoi()` の構文は以下のとおりです。

構文：

```
atoi(文字列)
```

ヘッダファイル：

```
stdlib.h
```

例：

```
v = atoi("-123");
```

関数 `atoi()` は文字列を `int` 型に変換した値を返します。

なお、この関数を使用するには `stdlib.h` をインクルードしなければなりません。

2.2. 関数 atof()

atoi.cpp の実行例 4 を見ると関数 atoi() では実数の変換が無理なことが分かります。

そこで文字列を実数として変換するには関数 atof() を使います。

atof.cpp

```
1: #include <stdio.h>
2: #include <stdlib.h>
3:
4: #define SIZE    128
5:
6: main()
7: {
8:     char    s[SIZE];
9:     double  v;
10:
11:     printf("数字を入力して下さい。¥n");
12:     gets(s);
13:     v = atof(s);
14:
15:     printf("文字列「%s」は実数%f と変換されました。¥n", s, v);
16: }
```

実行例 1

数字を入力して下さい。

3.14

文字列「3.14」は実数 3.140000 と変換されました。

実行例 2

数字を入力して下さい。

e = 2.72

文字列「e = 2.72」は実数 0.000000 と変換されました。

余計な文字が混じると「0.0」とするのは atoi() と同じですね。

なお、関数 `atof()` の構文は以下の通りです。

構文：

```
atof(文字列)
```

ヘッダファイル：

```
stdlib.h
```

例：

```
v = atof("3.14");
```

関数 `atof()` は文字列を `double` 型に変換した値を返します。

なお、この関数を使用するには `stdlib.h` をインクルードしなければなりません。

3. 文字列操作

3.1. strlen()...文字列の長さを得る

構文：

```
strlen(文字列)
```

ヘッダファイル：

```
string.h
```

例：

```
len = strlen("moji");
```

文字列の長さ(¥0 を含まない)を int 型の数値で返します。

なお、この関数を使用するには string.h をインクルードしなければなりません。

以下は strlen() を使ったプログラムです。

入力し、コンパイル、実行してみましょう。

strlen.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3:
4: #define SIZE    128
5:
6: main()
7: {
8:     char    s[SIZE];
9:     int    len;
10:
11:     printf("文字列を入力して下さい。¥n");
12:     gets(s);
```

```
13:         len = strlen(s);  
14:  
15:         printf("文字列の長さは%d文字です。¥n", len);  
16:     }
```

実行例 1

文字列を入力して下さい。

hogehoge

文字列の長さは 8 文字です。

実行例 2

文字列を入力して下さい。

あいうえお

文字列の長さは 10 文字です。

このように文字列の長さを得ることができます。

ただし全角文字は 1 文字 × 2 とカウントします。

3.2. strcpy()...文字列の代入

構文 :

```
strcpy(文字列 A, 文字列 B)
```

ヘッダファイル :

```
string.h
```

例 :

```
strlen(str1, str2);
```

文字列 B を文字列 A にコピーします。
文字列の代入文といってもよいでしょう。

以下は strcpy() を使ったプログラムです。
入力し、コンパイル、実行してみましよう。

strcpy.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3:
4: #define SIZE    128
5:
6: main()
7: {
8:     char    a[SIZE], b[SIZE];
9:
10:    printf("文字列を入力して下さい。 %n");
11:    gets(a);
12:
13:    strcpy(b, a);
14:    printf("b[] = %s %n", b);
15: }
```

実行例 1

文字列を入力して下さい。

```
foo!
```

```
b[] = foo!
```

このように文字列をコピーすることができます。

3.3. strcmp()...文字列の比較

構文 :

```
strcmp(文字列A, 文字列B)
```

ヘッダファイル :

```
string.h
```

例 :

```
if(strlen(str1, str2) == 0){  
    ...  
}
```

文字列 A と文字列 B の内容を比較し、同じなら 0、異なれば 0 以外の値を返しません。

以下は strcmp() を使ったプログラムです。
入力し、コンパイル、実行してみましょう。

strcmp.cpp

```
1: #include <stdio.h>  
2: #include <string.h>  
3:  
4: #define SIZE    128  
5:  
6: main()  
7: {  
8:     char    a[SIZE], b[SIZE];  
9:     int     n;  
10:  
11:     printf("メールアドレスを入力して下さい。¥n");  
12:     gets(a);  
13:     printf("再度メールアドレスを入力して下さい。¥n");
```

```
14:         gets(b);
15:
16:         if(strcmp(a, b) == 0){
17:             printf("メールアドレスを確認しました。¥n");
18:         } else {
19:             printf("メールアドレスが異なります。¥n");
20:         }
21:     }
```

実行例 1

```
メールアドレスを入力して下さい。
k miike@members15.tsukaeru.net
再度メールアドレスを入力して下さい。
k miike@members15.tsukaeru.net
メールアドレスを確認しました。
```

実行例 2

```
メールアドレスを入力して下さい。
aaa@bbb.ac.jp
再度メールアドレスを入力して下さい。
aaa@bbb.com
メールアドレスが異なります。
```

このように文字列の内容を比較することができます。
パスワードの確認などに使えそうですね。

3.4. strcat()...文字列の連結

構文：

```
strcat(文字列 A, 文字列 B)
```

ヘッダファイル：

```
string.h
```

例：

```
strcat(str1, str2);
```

文字列 A に文字列 B の内容を追加します。
そのため文字列 A のサイズは十分に大きくしておく必要があります。

以下は strcat() を使ったプログラムです。
入力し、コンパイル、実行してみましょう。

strcat.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3:
4: #define SIZE    128
5:
6: main()
7: {
8:     char    a[SIZE], b[SIZE];
9:     int     n;
10:
11:     printf("文字列を入力して下さい。 %n");
12:     gets(a);
13:     printf("追加する文字列を入力して下さい。 %n");
```

```
14:         gets(b);
15:
16:         strcat(a, b);
17:         printf("%s¥n", a);
18:     }
```

実行例

文字列を入力して下さい。

hogehoge

追加する文字列を入力して下さい。

foo!

hogehogefoo!

このように文字列の追加ができることがわかります。

3.5. sprintf()...書式指定された文字列の生成

構文 :

```
printf(文字列, 生成する文字列の書式, 引数...)
```

ヘッダファイル :

```
stdio.h
```

例 :

```
printf(str, "x = %d, y = %d\n", x, y);
```

関数 `printf()` は生成した文字列を画面に表示しますが、今回解説する関数 `sprintf()` は文字列に格納します。

以下は `sprintf()` を使ったプログラムです。
入力し、コンパイル、実行してみましょう。

sprintf.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: main()
6: {
7:     char    s[SIZE];
8:     int     a, b;
9:
10:    printf("a = ");
11:    scanf("%d", &a);
12:    printf("b = ");
13:    scanf("%d", &b);
14:
```

```
15:         sprintf(s, "a + b = %d + %d = %d", a, b, a + b);
16:
17:         printf("%s\n", s);
18:     }
```

実行例

```
a = 5
b = 6
a + b = 5 + 6 = 11
```

このように指定された書式で文字列を生成することがわかります。

3.6. sscanf()...文字列から書式に従ってデータを取り出す

構文：

```
sscanf(文字列, データを取り出す書式, 引数...)
```

ヘッダファイル：

```
stdio.h
```

例：

```
sscanf(str, "%d,%d", &a, &b);
```

関数 `scanf()` はキーボードで入力した文字列を指定された書式に従ってデータを取り出しますが、今回解説する関数 `sscanf()` は文字列から取り出します。

以下は `sscanf()` を使ったプログラムです。
入力し、コンパイル、実行してみましょう。

sscanf.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: main()
6: {
7:     char    str[SIZE];
8:     double a, b;
9:
10:    printf("文字列を入力して下さい。¥n");
11:    gets(str);
12:
13:    sscanf(str, "%lf %lf", &a, &b);
14:    printf("取り出した値は %f と %f です。¥n", a, b);
```

15: }

実行例

文字列を入力して下さい。

3.14 2.718

取り出した値は 3.140000 と 2.718000 です。

このように指定された書式にしたがって、文字列からデータを取り出せることがわかります。

3.7. strncpy()...最大長さを指定した strcpy()関数

構文：

```
strncpy(文字列A, 文字列B, 最大長さ)
```

ヘッダファイル：

```
string.h
```

例：

```
strncpy(str1, str2, len);
```

文字列Bを文字列Aにコピーします。ただし文字数は「最大長さ」までです。また最大長さが文字列Bの長さより小さい場合、最後に'¥0'は付加しません。

以下はstrncpy()を使ったプログラムです。
入力し、コンパイル、実行してみましょう。

strncpy.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3:
4: #define SIZE    128
5:
6: main()
7: {
8:     char    a[SIZE], b[SIZE];
9:     int     n;
10:
11:     printf("文字列を入力して下さい。¥n");
12:     gets(a);
13:     printf("複写する文字数を入力して下さい。¥n");
14:     scanf("%d", &n);
```

```
15:
16:     strncpy(b, a, n);
17:     printf("b[] = %s¥n", b);
18: }
```

実行例

文字列を入力して下さい。

hogehoge

複写する文字数を入力して下さい。

3

b[] = hog

このように文字数を指定して複写できることが分かります。

同様に文字数を指定できる関数 `strncmp()` や関数 `strncat()` もあります。

4. 文字列処理関数一覧

以下の表はよく使われる文字列処理関数の一覧です。

また、これらの関数から派生した関数もあるので必要に応じて調べてみましょう。

関数名	機能	必要なヘッダファイル
gets()	キーボードから入力された文字列を得る	stdio.h
atoi()	文字列を数値に変換する	stdlib.h
strlen()	文字列の長さを得る	string.h
strcpy()	文字列を代入する	
strcmp()	2つの文字列の内容を比較する	
strcat()	2つの文字列を連結する	
sprintf()	書式指定された文字列の生成	stdio.h
scanf()	指定された書式にしたがって文字列からデータを取り出す	

5. 演習問題

9-1. scanf()関数を使用しないで「for2.cpp」と同じ処理を行うプログラム「ensyu9-1.cpp」を作りなさい。

ヒント:

scanf()関数の代わりに gets()関数と atoi()関数を使えば大丈夫です。

また、atoi()関数を使うには「#include <stdlib.h>」が必要です。

9-2. scanf()関数を使用しないで「ensyu7-1.cpp」と同じ処理を行うプログラム「ensyu9-2.cpp」を作りなさい。

9-3. scanf()関数を使用しないで「ensyu7-4.cpp」と同じ処理を行うプログラム「ensyu9-3.cpp」を作りなさい。

9-4. scanf()関数を使用しないで「ensyu8-1.cpp」と同じ処理を行うプログラム「ensyu9-4.cpp」を作りなさい。