文字列 2

前回の授業ではコンピュータ内部での文字の取り扱い、文字型の変数、文字型変数への代入方法などを学習した。今回は、前回に引き続き、文字処理を学習する。内容は、標準入出力(キーボード、ディスプレイ)での文字処理、文字のファイル処理、文字を取り扱うライブラリ関数である。

標準入出力

Linux では、一般的に、標準入力装置としてキーボードが、標準出力装置としてディスプレイが割り当てられている。文字をキーボードから入力して、ディスプレイに書き出す方法を示す。

標準(キーボード)入力

getchar について

機能	キーボードから 1 文字入力する。
ヘッダー	stdio.h
形式	int getchar(void)
返却値	読み込んだ文字を返す。ファイル終了時は EOF を返す。
使用例	char a; /* 文字型の変数 */ a = getchar(); /* 代入 */

形式では返却値が int 型であり、使用例では char 型に代入しているが問題ない。C 言語では char 型は 1 バイトの整数として扱われる。この関数は日本語のような 2 バイト文字は代入できない。この関数と対になっている関数として、「putchar」がある。ここで注意点がある。以下のプログラムで確認しよう。

プログラム例 1

```
#include <stdio.h>

int main(void)
{
        char a, b;

        a = getchar();
        printf("1つ目のgetcharの結果：%c¥n", a);

        b = getchar();
        printf("2つ目のgetcharの結果：%c¥n", b);

        return 0;
}
```

「getchar」で 1 文字ずつ入力することを考え、上記のようなプログラムを作成したとする。1 文字目として、'a'を入力し、「Enter」を押した。その場合、2 文字目を入力することもできず、以下のような出力を行い、プログラムが終了してしまう。なぜ、このような結果になってしまったのだろうか。

```
a
1 つ目の getchar の結果：a
2 つ目の getchar の結果：
```

実は 2 文字目はすでに入力されてしまっている。キーボードからの入力を一区切りにするには「Enter」を押さなくてはならない。しかし、「Enter」を押したということは'¥n'という特殊記号が文字として入力されたことになる。出力結果を見ると「2 つ目の getchar の結果：」の後は確か

に改行が行われている。「Enter」は文字扱いになることに注意しなくてはならない。例題のプログラムではすぐ2つ目の入力処理を書いているが、実際のプログラムでは間に他の多くの処理が入ってくるため、この現象が起きてしまうと、混乱の元となり、デバックが難しくなる。対応策としては、以下のように'¥n'を格納する処理と1セットにしてしまえばよい。

プログラム例２

```
#include <stdio.h>

int main(void)
{
        char a, b, c;

        a = getchar();
        c = getchar();

        printf("1つ目のgetcharの結果：%c¥n", a);

        b = getchar();
        c = getchar();
        printf("2つ目のgetcharの結果：%c¥n", b);

        return 0;
}
```

gets について

機能	キーボードから 1 行読み込み、a(配列)に格納。1 行というのは、「Enter」キーが押されるまで。
ヘッダー	stdio.h
形式	char *gets(char *a)
返却値	入力文字をそのまま返す。ファイル終了時、あるいはエラーのときは NULL を返す。
使用例	char a[10]; /* 文字型の配列 */ gets(a); /* 代入 */

「getchar」では、「Enter」の入力は'¥n'という特殊記号の入力であると説明した。この関数では入力された文字列の最後の'¥n'を自動的に'¥0'に置き換えて、配列に格納してくれる。

ただし、こちらも使用には注意が必要で、配列で確保された文字数以上を入力すると、問題が起きる可能性がある。そのため、コンパイラによっては警告を出すものもある。この後紹介する「fgets」は入力文字数を制限するのでこちらを使用した方が良いだろう。「gets」と対になっている関数として、「puts」がある。

scanf について

機能	キーボードから文字列を読み込み、a(配列)に格納。
ヘッダー	stdio.h
形式	int scanf("%s", a)
返却値	通常、入力項目を返す。異常時は EOF を返す。
使用例	char a[10]; /* 文字型の配列 */ scanf("%s", a); /* 代入 */

この使用例では空白を入力できない。空白は文字列の区切りとして扱われ、そこで文字列が終わったと判断されてしまうからである。そのため、「gets」や「fgets」を用いた方が良い。この関数と対になっている関数はおなじみの「printf」である。

※入力される空白の数があらかじめ分かっている場合には以下のように利用することもできる。

プログラム例

```
#include <stdio.h>
int main(void)
{
        char a[10], b[10], c[10], d[10];

        scanf("%s %s %s %s", a, b, c, d);
        printf("%s %s %s %s¥n", a, b, c, d);

        return 0;
}
```

出力例(斜体は入力)

This is a pen
This is a pen

fgetsについて

どの関数にも問題があるが、文字列をキーボードから入力する場合のもっとも良い方法と思われるのは「fgets」を使う方法である。この関数自体はファイルからの読み込み処理を行う関数である。

機能	ファイルから1行読み込み、配列に格納。
ヘッダー	stdio.h
形式	char *fgets(char *a, int n, FILE *fi)
返却値	入力文字をそのまま返す。ファイル終了時、あるいはエラー のときは NULL を返す。
使用例	char a[10]; /* 文字型の配列 */ fgets(a, 10, stdin); /* 代入 */

この関数は本来、ファイルから1行読み込み、配列aに格納する。nバイト越えると読み込みを止める。読み込むバイト数を指定しているため、「gets」のように、許可されていないメモリ領域の内容を書き換えるようなことはない。

この関数を使用例に示したように書くと、キーボードからの入力に利用できる。これで、空白も入力できるし、指定した文字数を越える入力があっても、その文字数以降は無視され、メモリの内容を破壊することは無い。stdinと言うのは、standard inputの略であり、標準入力(キーボード)を示す。

この使用例での具体的な入力結果は以下である。

入力時に10文字以上	9文字目までが入力され、10文字目は’¥0’に置き換えられる。
入力時に9文字+「Enter」	9文字目までが入力され、10文字目は’¥0’に置き換えられる。
入力時に8文字以下+「Enter」	入力した文字列＋’¥n’＋’¥0’

「gets」では入力された文字列の最後の’¥n’は’¥0’に置き換えられたが、こちらでは基本的に’¥n’は取り除かれず、’¥0’が追加された形となる。つまり、入力された文字列＋’¥n’＋ ’¥0’が配列に格納される。ただし、バイト数（文字数）での制限があり、文字列の最後には’¥0’を必ず付ける仕様なので、例えば10バイトで制限した場合、9文字目までが入力され、10文字目は「Enter」、数字、アルファベット等、文字の種類に関わらず、’¥0’に置き換えられる結果となる。作りたいプログラムによっては、この改行の’¥n’がじゃまになるので、自分で’¥0’に置き換える処理を行わなくてはならない場合もある。

標準(ディスプレイ)出力

以下に標準出力に関する関数を説明していく。

putchar について

機能	ディスプレイに 1 文字出力する。
ヘッダー	stdio.h
形式	int putchar(int a)
返却値	出力した文字を返す。異常時は EOF を返す。
使用例	char a; /* 整数型の変数 */ a = getchar(); /* 代入 */ putchar(a); /* 1 文字出力 */

「putchar」でディスプレイに文字を書いた場合に改行は行われない。改行したい場合は

```
putchar(a);
putchar('¥n');
```

のようにして'¥n'を個別に出力する。

puts について

機能	ディスプレイに文字列を出力後、改行する。
ヘッダー	stdio.h
形式	int puts(char *a)
返却値	正常時は非負、異常時は EOF を返す。
使用例	char a[10]; /* 文字型の配列 */ gets(a); /* 代入 */ puts(a); /* 1 行出力 */

この関数では、自動的に最後の文字列の後に改行'¥n'が出力される。また、変換処理がないので、printf 関数よりも処理が早い。

printf について

機能	変換書式に従いディスプレイに書き出す。
ヘッダー	stdio.h
形式	int printf("%s", a)
返却値	正常時は出力されたバイト数を返す。異常時は負の値を返す。
使用例	char a[10]; /* 文字型の配列 */ scanf("%s", a); /* 代入 */ printf("%s", a); /* ディスプレイ出力 */

変換指定子のところに%c を使えば、文字列ではなく、1 文字を扱うことができる。

文字処理のための標準ライブラリ関数

文字処理

文字処理のためのライブラリ関数がＣ言語には用意されている。以下の表に示す。

表 1：1 文字処理関数。#include <ctype.h>が必要。変数は、int c;。

関数名	動作
isalnum(c)	英数字なら真
isalpha(c)	英文字なら真
iscntrl(c)	制御文字なら真
isdigit(c)	数字なら真
isgraph(c)	印字可能文字なら真
islower(c)	小文字なら真
isprint(c)	空白以外の印字可能文字なら真

ispunct(c)	区切り文字なら真
isspace(c)	空白類文字なら真
isupper(c)	大文字なら真
isxdigit(c)	16 進表示文字なら真
tolower(c)	文字 c を小文字に変換
toupper(c)	文字 c を大文字に変換

文字列処理

文字列処理のためのライブラリ関数がＣ言語には用意されている。以下の表に示す。

表 2: 文字列処理関数。#include <string.h>が必要。変数は、char s1[256],s2[256];のように文字型の配列である。配列のサイズは処理に必要なサイズよりも大きい必要がある(256 とは限らない)。c は文字型の変数 char c;である。

関数名	動作
strlen(s1)	文字列 s1 の長さ、すなわち文字数を整数値返す。
strcpy(s1, s2)	s1 に、文字列 s2 をコピーする。
strcat(s1, s2)	文字列 s1 の後に、文字列 s2 をコピーする。
strcmp(s1, s2)	文字列 s1 と s2 を比較する。 s1 > s2 の場合、戻り値は正 s1 == s2 の場合、戻り値は 0 s1 < s2 の場合、戻り値は負
strncpy(s1, s2, n)	s1 に文字列 s2 の先頭から n 文字をコピーする。
strncat(s1, s2, n)	文字列 s1 の後にと文字列 s2 の先頭から n 文字を連結する。
strncmp(s1, s2, n)	文字列 s1 と文字列 s2 の先頭から n 文字を比較する。比較の結果は、srcmp と同じ。
strchr(s1, c)	文字列 s1 の中の文字 c の位置を整数で返す。文字がないときは、NULL を返す。
strstr(s1, s2)	文字列 S1 の中にある文字列 S2 の位置を整数で返す。もし、文字列がない場合、NULL を返す。

課題 2 の続き

Kadai2-3 : キーボードから名前を読み込んで、挨拶をするプログラムを作成しなさい。以下に実行例を示す。

```
お名前は：Takeshita
はじめまして、Takeshita さん！
```

Kadai2-4 : キーボードから数字を含む文字列を入力し、その文字列から数字を取り除いた文字列を出力するプログラムを作成しなさい。以下に実行例を示す。

```
文字列を入力してください：abc91de2f
結果：abcdef
```

Kadai2-5 : キーボードから文字列を入力し、入力した文字列を右から左へと、一定時間で、一文字ずつ、テロップのように流しながら表示するプログラムを作成しなさい。これは以下のような結果となる。ただし、同じ行で出力されること。そのためには「復帰」の特殊文字'¥r'を使う。

```
Takeshita
↓
akeshitaT
↓
keshitaTa
↓
```

そのまま表示すると処理が速すぎて、確認できないと思う。以下のように、時間稼ぎ用の処理を使うこと。

```
for(k=0; k<1000; k++) // 繰り返しの回数は処理の速さを見て変える
{
        for(l=0; l<1000; l++)
        {
                c = sin((double)k) + cos((double)l); // 時間稼ぎ用の計算なのでなんでも良い
        }
}
```

文字の表示が正しく行われない場合には以下のようにして'¥r'の出力の後に fflush(stdout);を使うこと。表示処理はバッファというところに文字データが格納され、これをディスプレイに出力することになる。fflush(stdout);は実行時のタイミングでディスプレイへの出力を完了する。

```
        printf("¥r");
        fflush(stdout);
```

課題の締切は後で提示する。