計算機プログラミング'95 ＃2

久野　靖*

1995.9.14

1　Cと文字列処理

今回の前半は、直接システムプログラミングというわけではないが、今後の内容に必要なので「文字列」を取り上げる。多くの言語と同様、C でも文字列は単なる「文字列の配列」として実現されるが、ただしここで多少問題がある。

前回やったファイルコピー問題では、

```
char buf[20];
```

として宣言した配列 buf には 20 バイトきっかりを詰め込んでいた。しかし、プログラムで文字列を扱う場合には、その文字列は普通可変長である (ユーザにきっかり 20 文字打ち込め、というのはいまいちだから)。その場合、言語によっていくつかの流儀がある。

- 後ろに空白などをつめてきっかり 20 バイトにしてしまう。
- 長さを別に保持する。
- 最後に「終わりの印」を入れる。

C では 3 番目の「終わりの印」を入れる方法をとる。だから、20 文字の文字列まで入れたければ「終わりの印」を含めて 21 バイトのバッファが必要になる。もっとも、ふつうは「十分大きい」バッファを使うのでこれが問題になることはあまりない。

ときに、「終わりの印」としては文字コード「0」を使う。従って C の文字列ルーチンでは途中に文字コード 0 が入った文字は正しく扱えない。これも 0 は NUL という制御コードだから普段は問題にならない。

ではさっそく、文字列の読み込み/書き出しの例を見てみよう。

```
/* t21.c -- simple scanf&printf example */
#include <stdio.h>

main() {
  char buf[100];
  while(scanf("%s", buf) > 0) {
    printf("%s\n", buf);
  }
}
```

*筑波大学大学院経営システム科学専攻

printf/scanf では「%s」というフォーマット指定で文字列を読み書きできる。ただし、scanf では空白で区切られた「語」を読み込むという仕様になっている。(scanf の戻り値は読み込めた要素数なので、0 が戻ったらファイルの終わり。)

練習 1: 打ち込んで動かせ。

練習 2: 文字列の長さを調べるサブルーチンを作り、文字列の長さも一緒に打つようにしてみよ。

練習 3: 1 行読み込みサブルーチン getline を作れ。中では getchar を使って 1 文字ずつ読めばよい。バッファの最大文字数を渡して、長い行が来たときバッファ領域外が壊されないようにすること。

2 コマンド引数

Unix では、コマンドに様々な情報を「引数」として渡すことができる。コマンド引数はいったんシェルが受け取り、「文字列ポインタの配列」という形にして、コマンド (=プログラム) が実行される時に渡してくれる。これを受け取るには、main の引数をちゃんと指定すればよい。

```
/* t22.c -- argc & argv example */
#include <stdio.h>

main(int argc, char *argv[]) {
  int i;
  for(i = 0; i < argc; ++i) {
    printf("%s\n", argv[i]);
  }
}
```

なお、argv の宣言は「文字へのポインタの配列」と読む。argc はこの配列の要素数。argv[0] には常に「コマンド文字列」が入る。

練習 4: 打ち込んで動かせ。

練習 5: 一番長い引数だけ打ち出すようにしてみよ。

練習 6: 2 つの文字列が等しいかどうか調べる関数を作れ。

練習 7: 標準入力から 1 行ずつ読み、第 1 引数と同じ行の個数を数えよ。

練習 8: 2 つの文字列を受け取り、第 2 引数に第 1 引数と同じ部分文字列があればその位置、なければ 0 を返す関数を作れ。

練習 9: 標準入力から 1 行ずつ読み、第 1 引数と同じ文字列を含む行を打ち出せ (fgrep と同じ)。

なお、文字列の長さ、比較、位置検出、コピーなどはいちいち作らなくても標準ライブラリがある。たとえば「文字列 a と b が等しいか」は strcmp を使う。詳しくは man string を参照。

3 open と close

文字列の話が済んだのでまた入出力システムコールの話題に戻る。前回はチャネル 0 とか 1 につながった (既に開かれている) ファイルを扱ったが、新たにファイルへのチャネルを接続するには open、接続を切るには close を利用する。呼び出し方は次の通り:

```
ディスクリプタ番号 = open("パス名", フラグ, モード)
close(デスクリプタ番号)
```

「フラグ」というのは、例えば「O_RDONLY」だとファイルを読み込みモードで開く。「O_WRONLY」だと書き出し専用モードで開く。書き出しモードの場合、さらに「O_CREAT」を指定するとファイルが存在しない場合にファイルを新規作成する (そのファイルの保護モードを「モード」引数で与える)。「O_TRUNC」を指定すると出力ファイルの長さを最初に 0 にする。「O_APPEND」だと追加モードになる。複数のフラグを指定する場合には「|」でつなげる。たとえば出力だと

```
fd = open(..., O_WRONLY|O_CREAT|O_TRUNC, 0644);
```

というのが普通だろう。これらのフラグや関数の定義を読み込むために

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

が必要である。

open でファイルが開けなかった場合には「-1」が返される。一般に Unix ではシステムコールが失敗すると errno という広域変数にその説明コードが格納される。それを調べてもいいが、perror を使うと適当なメッセージを出してくれる。

```
perror("メッセージの一部")
```

練習 10 以上の知識をもとに、「自前の cp」を作れ。つまり、「a.out ファイル 1 ファイル 2」とするとファイル 1 の内容をファイル 2 にコピーするプログラムを作れ。

練習 11 出力ファイルのフラグに O_CREAT や O_TRUNC がないとどうなるか、またさらに O_APPEND があるとどうなるか、試せ。

練習 12 プログラムを変更して、「自前の cat」にせよ。つまり、「a.out ファイル 1 ファイル 2 …」とすると全ファイルの内容を順に標準出力にコピーするプログラムを作れ。ファイル名として「-」が指定されると標準入力が読まれるようにできればなおよい。

練習 13 第 1 引数が「-a」の場合には「空白を*に置き換える」動作を併せて行うようにしてみよ。

4 オプション解析

ここまで見て来たように、コマンドの引数は自分でどうにでも処理できるが、Unix コマンドはだいたい統一した形式のオプションを指定方法を持つ。つまり:

- オプション指定は「-」ではじまる。
- 1 文字のオプション「-a」「-c」などと、
- 後に文字列がつくオプション「-o ファイル」などがある。

- 1 文字のオプションは「-ac」のように複数くっつけてよい。

このような規則に従ったオプションを効率よく解析するためのライブラリ関数 getopt を学んでおこう。呼び方は次の通り

```
値 = getopt(argc, argv, オプション指定)
```

つまり、argc と argv をそのまま渡して繰り返し呼ぶと、「どのオプション」を表す文字を返してくれる。オプションが終わった場合には EOF が返される。なお、文字列がくっつくものの場合に外部変数 optarg にその文字列の先頭が入り、EOF になった時には次の引数の番号が optind に入っている。テスト用の例を示そう。

```
/* t24.c --- getopt example */
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[]) {
  int c;
  extern char *optarg;
  extern int optind;
  while((c = getopt(argc, argv, "abo:s:")) != EOF) {
    printf("%d: %c : %s\n", optind, c, optarg?optarg:"x");
  }
}
```

なお、「optarg?optarg:"x"」というのは「optarg が 0 でなければ optarg、そうでなければ文字列"x"」という if-then-else 式を表す。

もう少しそれらしい例として、echo と同じだが、「-o ファイル」とすると出力ファイルが指定でき、「-a」とすると引数と引数の間で空白を空けないというオプション指定ができる。プログラムを見て見よう。

```
/* t24b.c --- echo with some additional options */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0

main(int argc, char *argv[]) {
  int c, i;
  extern char *optarg;
  extern int optind;
  int o_nonewline = FALSE;
  int o_append = FALSE;
  int o_outfile = FALSE;
  char *o_ofilename = 0;
  while((c = getopt(argc, argv, "ano:")) != EOF) {
```

```
    switch(c) {
case 'a': o_append = TRUE; break;
case 'n': o_nonewline = TRUE; break;
case 'o': o_outfile = TRUE; o_ofilename = optarg; break;
    }
  }
  if(o_outfile) {
    close(1);
    if(open(o_ofilename, O_WRONLY|O_CREAT|O_TRUNC, 0644) < 0) {
      perror("myecho: "); exit(1);
    }
  }
  for(i = optind; i < argc; ++i) {
    if(!o_append) putchar(' ');
    printf("%s", argv[i]);
  }
  if(!o_nonewline) putchar('\n');
}
```