ソフトウェア工学 (第3回)

土村 展之
(関西学院大学 理工学部 教育技術職員)
http://ist.ksc.kwansei.ac.jp/~tutimura/

東京農工大学 工学部 情報工学科
2012 年 7 月 21 日

文字列の扱い

C言語の文字列

- char の配列で、最後にヌル文字 '\0'
- 文字列の長さに制限なし
- 作業領域は自前で確保する

- 正規表現マッチングは標準関数にない
- 作業領域をあふれないよう、細心の注意を
  sizeof() と strlen() は 1 ずれる

```
#define STR "hoge"
printf("%d\n", strlen(STR));  /* 4 */
printf("%d\n", sizeof(STR));  /* 5 */
```

よく使う文字列操作関数

- 標準関数
  - strlen(s)
  - strcat(d,s), strcpy(d,s), strncat(d,s,n)
  - strchr(s,c), strstr(s,t), strcmp(d,s)
  - fgets(s,n,fp)
  - sprintf(s,fmt,...), snprintf(s,n,fmt,...)
- 非標準関数
  - strdup(s)
- strncpy() は設計ミスなので使用を避ける
  （終端の '\0' を書かないことがある）
- gets() も設計ミス（バッファ溢れを防げない）

クイズ

- 文字列→数値変換の関数は?

- 数値→文字列変換の関数は?

クイズの答え

- 文字列→数値変換の関数は?
  - atoi(), atol(), sscanf() など

```
while ( fgets(buff, sizeof buff, stdin) != NULL ) {
    sscanf(buff, "%d", &i); ...
}
```

- 数値→文字列変換の関数は?
  - sprintf()

```
sprintf(buff, "%d", i);
```

文字列の初期化

```
char a[] = "this";
char b[] = { 't', 'h', 'i', 's', '\0' };

char *c  = "that";
char *d  = { 't', 'h', 'a', 't', '\0' }; /* NG */

a[0] = 'T';
b[0] = 'T';
c[0] = 'T';  /* NG */
```

文字型

- char, signed char, unsigned char は３つとも異なる
  （char の実体は残りのどちらかに等しい）
- `sizeof(char)` = 1 であるが、残りは１である保証はない
- `getchar()`, `getc(fp)`, `fgetc(fp)`
- `isalpha(c)`, `isdigit(c)`, `isxdigit(c)`
- `isspace(c)`, `iscntrl(c)`, `isprint(c)`
- `isupper(c)`, `islower(c)`
- `toupper(c)`, `tolower(c)`

これらの関数の引数は int であることに注意

例題 *(40) (41) (42)*

(40)　標準入力から文字を受け取り、
　　大文字に変換して標準出力に出力しなさい。

(41)　標準入力から文字を受け取り、文字数を数えなさい。

(42)　標準入力から文字を受け取り、数字の文字数を数えなさい。

解答例 *(40)*

```
#include <stdio.h>
#include <ctype.h>

int main() {
    int c;

    while ( (c=getchar()) != EOF ) {
        putchar(toupper(c));
    }
    return 0;
}
```

解答例 *(42)*

以下のプログラムは正しくは動きません。修正しなさい。

```
#include <stdio.h>
#include <ctype.h>
#define TRUE  1
#define FLASE 0

int main() {
    int c, num = 0;

    while ( (c=getchar()) != EOF ) {
        if ( isdigit(c) == TRUE ) num++;
    }
    printf("%d\n", num);
    return 0;
}
```

例題 *(43) (43')*

(43)　標準入力から文字を受け取り、
一行ごとに行末から順に表示しなさい。
（文字単位の入れ換え）

ex.
1234 → 4321
abcd → dcba

(43')　標準入力から文字を受け取り、
最後の行から順に表示しなさい。（行単位の入れ換え）

ex.
1234 → xyz
4567 → abcd
abcd → 5678
xyz → 1234

(43') は入力をすべて記憶する必要があるので極端に難しい

解答例 *(43)*

以下のプログラムの動作は少し変です。

```
#include <stdio.h>
#include <string.h>
#define BUFF_SIZE 1000

int main() {
    int i, len;
    char buff[BUFF_SIZE];

    while ( fgets(buff, sizeof buff, stdin) != NULL ) {
        len = strlen(buff);
        for (i=len-1; i>=0; i--) putchar(buff[i]);
    }
    return 0;
}
```

例題 *(43')* について

- C 言語で正しく書くのは大変
  - 入力行数に上限をつければ簡単になる
- Perl 言語なら簡単

```
perl -e 'print reverse(<STDIN>)'
```

レポートの講評

前回のレポート課題

ビンゴゲームのビンゴ判定をするプログラムを作りなさい。

- 5× 5のマスに1~ 75の数字が並んでいる。
- 中央のマスは初めから有効になっている。
- 数字がひとつずつ選ばれる。
  選ばれた数字が盤面にあれば、そのマスが有効になる。
- 縦横斜めいずれか１列が揃って有効になったらビンゴ。
- ビンゴになるまでに選ばれた数字の個数を出力せよ。

講評

- ２次元配列をいくつ持つか
  - 数字のテーブル以外に、出たかどうかのテーブルを持つ
  - 数字のテーブルを１つだけ持ち、破壊する
- ビンゴ判定のタイミング
  - 新たな数字が読み上げられるたび
  - 読み上げられた数字が盤面にあるときのみ

講評

- ビンゴ判定の方法
  - ループ
  - ハードコーディング
  - 行ごと列ごとの情報を配列に保存して更新
- 情報出力
  - どこがビンゴ（リーチ）になったか
  - 盤面

慣用句

ループ

```
for (i=0; i<n; i++)  { /* 0オリジン */ };
for (i=1; i<=n; i++) { /* 1オリジン */ };

for (i=0; i<=n; i++) { /* 奇妙 */ };
for (i=1; i<n; i++)  { /* 奇妙 */ };

for (i=0; i<n+1; i++)  { /* 0オリジン, 1回多い */ };
for (i=1; i<=n-1; i++) { /* 1オリジン, 1回少ない */ };

for (i=n-1; i>=0; i--) { /* 0オリジン, 逆順,
                            ループ変数は符号付きであること */ }
for (i=n; i>0; i--)    { /* 1オリジン, 逆順 */ }
```

ブール代数

```
if ( !(a && b) )    <==>  if ( !a || !b )
if ( !(a || b) )    <==>  if ( !a && !b )
```

基本姿勢

- 変数のスコープ（通用範囲）はなるべく狭く。
- キャスト（型変換）は最小限に。
- コメントは多く。... とは言うけれど...
  - 適切な変数名・関数名は値千金。
  - 不自然な処理にはコメントを。
  - コメントには「なぜ」を書く。
    （その実装を選んだ理由を説明する）

便利な外部コマンド

文字列検索・置換・比較

文字列検索

```
grep 'print' < infile
```

文字列置換

```
sed -e 's/print/PRINT/g' < infile > outfile
```

テキスト比較（強力！）

```
diff -u file1 file2 | less
```

乱数

標準関数の乱数

```
#include <stdlib.h>

int rand(void);
void srand(unsigned int seed);
```

- rand() で 0~ RAND_MAX の整数値が得られる。
- 再現可能なので、正確には疑似乱数。
- 乱数としての性質は、一般にあまり良くない。
- 乱数列は環境依存。
- srand() を用いると同一環境では再現性あり。

乱数の使い方 *(rand)*

- 実数値（$0 \leq r < 1$）
  - `r = rand() / (RAND_MAX+1.0);`
- 実数値（$0 \leq r \leq 1$）
  - `r = (double)rand() / RAND_MAX;`
- サイコロ（$r = \{1, 2, 3, 4, 5, 6\}$）
  - `r = 1 + rand() % 6;  /* 非推奨 */`
  - `r = 1 + (int)(rand() / (RAND_MAX+1.0) * 6);`

乱数の使い方 (srand)

- srand() で seed（乱数の種）を指定。
  - 同じ seed からは同じ乱数系列が得られる。
  - 通常は main() で一度だけ実行すれば十分。
  - seed をコマンドライン引数で指定できると便利。

- 再現性をわざとなくしたければ時刻を用いることが多い。

```
srand((unsigned)time(NULL));
```

```
int main() {
    srand(0); printf("%d\n", rand());
    srand(0); printf("%d\n", rand());  /* 同じ値 */
    return 0;
}
```

乱数を使う上での注意点

- 標準関数には問題が多い。
  - 環境に依存（違うコンパイラでは違う数列）。
  - 乱数としての性質。
- メルセンヌ・ツイスターを使うのがよい。
  - http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/
  - 環境依存しない。
  - 乱数としての性質も大きく改善。

例題 (44)

- ビンゴ判定のプログラムを改造して、
  読み上げられる数字を、
  ファイルから読み込む代わりに、
  乱数で生成しなさい。
  - 再現性に注意すること。(バグ取りに必須)
  - プログラムを２本に分割してもよい。
  - 重複した数字を出さないよう配慮する?

整列

標準関数のソート

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

- 配列を並び替える標準関数。
- アルゴリズムがクイックソートとは限らない。
  （実際にマージソートの場合も）
- 安定性は保証されない。
- 比較関数を用意するのが面倒だが、応用範囲は広い。
  - 逆順のソート
  - 安定にする
  - インデックスのみのソート（実体の並び順を保存）

比較関数の作り方

```
/* int[] の比較関数 */
int cmp(const void *p, const void *q) {
    /* p, q をすぐに代入しなおす */
    const int *a = (const int*)p;
    const int *b = (const int*)q;

    /* p, q は２度と使わない */
    if (*a > *b) return +1;
    if (*a < *b) return -1;
    return 0;
}
```

qsort()の使い方

```
int main() {
    int i, data[100];

    for (i=0; i<100; i++) data[i] = (i-50)*(i-50);

    qsort(data,                 /* 配列の先頭 */
          100,                  /* 要素数 */
          sizeof(data[0]),      /* １要素の大きさ */
          cmp);                 /* 比較関数 */

    for (i=0; i<100; i++) printf("%d\n", data[i]);
    return EXIT_SUCCESS;
}
```

例題 *(45)(45')*

(45)　乱数で 0~ 1000 の範囲の整数を 100 個生成し、
大きいもの順にソートしなさい

- テストプログラムも
- 外部コマンドを利用した方法も

(45')　乱数で x,y 座標（座標はそれぞれ 0~ 1000 の範囲の整数）を
100 組生成し、原点に近いもの順にソートしなさい

- かなりの難問
- 外部コマンドを利用した方法も

構造体

構造体とは

- 複数のデータをひとまとめにして扱うためのもの
- ユーザーが定義する「型」
- 関連の深いデータをまとめるのに用いる
- オブジェクト指向の基礎でもある
- 構造体内部の変数を「メンバ」「フィールド」と呼ぶ

```
struct point_st {  /* 型宣言のみ */
    int x, y;
};

struct point_st a, b;  /* 実体を伴う */
```

構造体のメンバへのアクセス

```
struct point_st {  /* 型宣言 */
    int x, y;
} a, b;              /* 実体を伴う */

a.x = 1;  a.y = 2;  /* 個々のメンバーに代入 */
printf("%d %d\n", a.x, a.y);  /* 1 2 */

b = a;                  /* 構造体ごとコピー */
printf("%d %d\n", b.x, b.y);  /* 1 2 */
```

構造体へのポインタと引数渡し

```
struct point_st { int x, y; };

void point_print(struct point_st *p) {
    printf("%d %d\n", p->x, p->y);
}

int main() {
    struct point_st a, *p;    /* p には実体がない */
    p = &a;
    p->x = 1;  (*p).y = 2;
    printf("%d %d\n", a.x, a.y);  /* 1 2 */
    point_print(&a);              /* 1 2 */
    point_print(p);               /* 1 2 */
....
```

構造体の意義

```
struct point3_st {
    int x, y, z;
} p[10];

void point3_print(
        struct point3_st *p) {
    printf("%d %d %d\n",
            p->x, p->y, p->z);
}

point3_print(&p[5]);
```

```
int x[10], y[10], z[10];


void xyz_print(int x,
                int y, int z) {
    printf("%d %d %d\n",
            x, y, z);
}

xyz_print(x[5], y[5], z[5]);
```

構造体に操作をする関数群を作る

- メンバーを表示する
- メンバーを初期化（代入）する
  （パラメータの範囲チェックも）

```
struct date_st {
    int year, month, day;
    int num;  /* なくてもよいが、状況次第で役立つ */
}

void date_init(struct date_st *p, int y, int m, int d) {
    if (!is_valid_date(y,m,d)) { exit(1); /*エラー*/}
    p->year = y;
    p->month = m;
    p->day = d;
    p->num = date_to_num(y,m,d);
}
```

構造体の設計方針と扱い方

- 何を構造体にまとめるべきかは、難しい問題
  - 「何を関数に分離するか」と類似
  - 「オブジェクト指向」という思想が手助け

- 構造体は専用の関数を通じて操作する
  （メンバーを直接アクセスすることは避ける）
  - パラメータの範囲チェックが意味をなす
  - メンバーを変更してもプログラムの修正が容易

例題 *(46)(46')*

(46)　乱数で x,y 座標（座標はそれぞれ 0~ 1000 の範囲の整数）を 100 組生成し、原点に近いもの順にソートしなさい

(46')　乱数で x,y 座標（座標はそれぞれ 0~ 1000 の範囲の整数）を 100 組生成し、偏角でソートしなさい。

状態遷移

例題 (50)

- 次のような自動販売機の真似をするプログラムを書け。
- 商品は１種類で、金額は 100 円。
- 受け取る硬貨は 10 円, 50 円, 100 円の３種類。
- 投入金額が 100 円以上になると、商品とお釣りが出る。
- キーボードから投入する硬貨の金額を受け付けて、お釣りの金額を表示して終了せよ。

```
投入する硬貨の金額を入力して下さい。→ 10
投入する硬貨の金額を入力して下さい。→ 50
投入する硬貨の金額を入力して下さい。→ 100
お釣りは 60 円です。
```

解答例 (50) その 1

```
int main(void) {
    int sum = 0, coin;

    while (sum < 100) {
        printf("投入する硬貨の金額を入力して下さい。→");
        scanf("%d", &coin);
        if (coin ==10 || coin == 50 || coin == 100) {
            sum += coin;
        }
    }
    printf("お釣りは%d円です。\n", sum-100);
    return 0;
}
```

解答例 (50) その 2

```
int main(void) {
    int coin;

    while (1) {
        printf("投入する硬貨の金額を入力して下さい。→");
        scanf("%d", &coin);
        if (coin == 100) { // 100円
            printf("お釣りは0円です。\n"); break;
        } else if (coin == 50) { // 50円
            printf("投入する硬貨の金額を入力して下さい。→");
            scanf("%d", &coin);
            if (coin == 100) { // 150円
                printf("お釣りは50円です。\n"); break;
            } else if (coin == 50) { // 100円
                printf("お釣りは0円です。\n"); break;
            } else if (coin == 10) { // 60円
                printf("投入する硬貨の金額を入力して下さい。→");
                scanf("%d", &coin);
            }
```

状態の記憶方法

- ここでは「投入金額」が状態だと思うことにする。
- 変数で覚える vs. 実行部分で覚える
- キー入力のエラー処理は 1 ヶ所にまとめたい。
- （この場合は）変数で状態を表現するのが妥当。
  - 10 円→ 50 円と 50 円→ 10 円を区別する必要がない。
  - 動作確認にすべての投入順序を試す必要は？

例題 *(51)*

- 次のようなローマ字かな変換をするプログラムを書け。
- 入力はキーボードから受け取る。
- 変換結果を画面に出力する。
- 入力には `getchar()` を 1 ヶ所のみで用い、while ループで囲え。

再帰呼び出し

再帰呼び出し

- ある関数から、その関数自身を呼び出すこと
- 再帰的な手続きを簡潔に表現できる
- 再帰関数には、再帰の末端の場合分けが必要

- 必ず等価なループ処理に書き換えることができる
- 再帰の深さが 1000 を越えるようだと設計ミス
  通常は 10~ 20 程度で収まる

```
int factorial(int n) {
    if (n==0) return 1;    /* 末端の処理 (0! = 1) */
    return n * factorial(n-1); /* 再帰呼び出し */
}
```

レポート課題

迷路を解くプログラムを書きなさい

- 77 × 21 のマスに空白か文字のどちらかが入っている
- 空白のマスは通れる
- 文字のマスは壁で通れない
- 右下と左上のマスがスタートとゴール
- スタートとゴールの間を結ぶ道を表示しなさい

- `http://ist.ksc.kwansei.ac.jp/~tutimura/tuat/`
  で迷路のデータを配布する
- 提出期限は 7/31(火)