プログラミングおよび実習 I・第 14 回

2003 年 7 月 16 日

1 模擬試験

問題 1 次のプログラム reverse.c は、標準入力 (キーボード) から入力された 1 行分の文字列を反転して出力するプログラムである。reverse.c のコンパイルと実行例を参考にして、このプログラムで省略されている部分に補うべき関数 revstr の定義 (プログラム) を書きなさい。ただし、実行例の中で「This is a pen.」とそれに続く改行文字はキーボードから入力したものであり、「.nep a si sihT」とそれに続く改行文字はこのプログラムが出力したものである。入力される 1 行の文字列の内、100 バイト目以降の文字は無視されて構わないものとする。

reverse.c

```
#include <stdio.h>

int getLine(char buf[], int size)
{
    int ch, n = 0;

    while ((ch = getchar()) != EOF && ch != '\n') {
        if (n < size - 1)
            buf[n++] = ch;
    }
    buf[n] = '\0';
    return (n == 0 && ch == EOF) ? -1 : n;
}

⋮

int main()
{
    char buf[100];

    getLine(buf, sizeof(buf));
    revstr(buf);
    printf("%s\n", buf);
    return 0;
}
```

reverse.c のコンパイルと実行例

```
s1609h017% cc -o reverse reverse.c
s1609h017% ./reverse
This is a pen.
.nep a si sihT
s1609h017%
```

問題 2 2 人のプレーヤーで楽しむ次のようなゲームを考える。10 個の石が一列に並べてあり、ここから 2 人のプレーヤーは交互に石を取り去っていく。ただし、1 回に取ることのできる石は多くても 2 個であり、複数の石を取り去る場合は、隣り合って並んでいる石でないといけない。最後の石を取り去った方のプレーヤーが負けとなる。

次は、このゲームを行うためのプログラム stones.c のコンパイルと実行例である。10 個の石には、左端から順に a から j までの英文字の名前が付けてあり、たとえば、c の石 1 個を取り去る場合は

「c1」のようにキーボードから入力し、c と d の 2 個の石を取り去る場合は「c2」のように入力するものとしている。また、ルールに反する取り方が入力されたときには再度入力を促す。石の数が 1 個以下になったらこのプログラムは終了する。

stones.c のコンパイルと実行例

```
s1609h017% cc -o stones stones.c
s1609h017% ./stones
(a)(b)(c)(d)(e)(f)(g)(h)(i)(j)
? a2
      (c)(d)(e)(f)(g)(h)(i)(j)
? c1
         (d)(e)(f)(g)(h)(i)(j)
? f1
         (d)(e)   (g)(h)(i)(j)
? e2
? h1
         (d)(e)   (g)   (i)(j)
? i2
         (d)(e)   (g)
? e2
? e1
         (d)      (g)
? g1
         (d)
s1609h017%
```

次のプログラムでは、関数 main で定義されている struct State 型の変数 st に現在の石の状況 (どの石が残っているか) を保持している。st のメンバ配列 line はそれぞれの石が残っているかどうかを表しており、st.line[0] が左端の a の石に、st.line[9] が右端の j の石に対応している。対応する要素の値が 1 であればその石がまだ残っているし、0 であればすでに取り去られている。また、st.rest には、残っている石の総数を保持している。

省略されてる部分を補って、このプログラムを完成しなさい。そのとき、最初の石の数 LEN や、1 度に取り去ることのできる石の最大数 MAX を少々変えても、プログラムが正しく動作するようにしなさい。ただし、1 度に取り去ることのできる石は、MAX が大きくなった場合でも連続していなければならないものとする。

stones.c

```
#include <stdio.h>

#define LEN     (10)
#define MAX     (2)
#define FOR_EACH(x)     for ((x) = 0; (x) < LEN; (x)++)
#define IN_LINE(x)      (0 <= (x) && (x) < LEN)

struct State {
    int line[LEN];
    int rest;
};

void show(struct State *s)
{
    ⋮
}

int main ()
{
```

```
    struct State st;
    char ch;
    int i, j, n;

      ⋮

    show(&st);
    do {
        printf ("? ");
        if (scanf(" %c%d", &ch, &n) != 2)
            continue;
        i = ch - 'a';

        ⋮

        for (j = i; j < i + n; j++)
            st.line[j] = 0;
        st.rest -= n;
        show(&st);
    } while (st.rest > 1);
    return 0;
}
```

問題 3　次のプログラム linefreq.c は、つぎつぎと 1 行分の文字列を入力していき、最後に空行を入力すると、(行を単位として) それぞれの文字列が入力された回数を表示するものである。関数 main で定義されている Stat 型の変数 st のメンバ num には、これまでに何通りの文字列が入力されたが保持されており、メンバ table には、それぞれの文字列がこれまでに現れた回数が保持されている。

main から呼び出されている関数 add は、入力された 1 行分の文字列が st のメンバ配列 table 中に存在するかどうかを調べ、すでに存在している場合は、その要素の count を 1 増やす。もし、存在していない場合は、配列 table の新しい要素を使って、この文字列が 1 回現れたことを記録する。また、関数 print は、すべての文字列の読み込みが完了したときに呼ばれ、変数 st の内容に従って、それぞれの文字列が現れた回数を表示する。

linefreq.c のコンパイルと実行例を参考にして、このプログラムで省略されている部分に補うべき関数 add と関数 print の定義 (プログラム) を書きなさい。ただし、それぞれの行の文字列の 100 バイト目以降は無視されて構わないものとし、入力された文字列が 100 通りを越えた場合は、それ以降に新しく登場した文字列は無視されるものとする。

linefreq.c

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

#define LEN     (100)
#define NUM     (1000)

typedef struct {
    char str[LEN];
    int count;
} Freq;

typedef struct {
    int num;
    Freq table[NUM];
} Stat;
```

```
int getLine(char buf[], int size)
{
    int ch, n = 0;

    while ((ch = getchar()) != EOF && ch != '\n') {
        if (n < size - 1)
            buf[n++] = ch;
    }
    buf[n] = '\0';
    return (n == 0 && ch == EOF) ? -1 : n;
}

⋮

int main()
{
    char buf[LEN];
    Stat st;

    st.num = 0;
    while (getLine(buf, sizeof(buf)) > 0)
        add(&st, buf);
    print(&st);
    return 0;
}
```

linefreq.c のコンパイルと実行例

```
s1609h017% cc -o linefreq linefreq.c
s1609h017% ./linefreq
a white cat
a white dog
a black horse
a black cat
a brown dog
a red horse
a white dog
a red horse
a white dog
a black cat

回数 : 文字列
   1 : a white cat
   3 : a white dog
   1 : a black horse
   2 : a black cat
   1 : a brown dog
   2 : a red horse
s1609h017%
```

ヒント　2 つ文字列の比較には標準ライブラリ関数 strcmp を使うことができる。また、文字列のコピーには

```
extern char *strcpy(char *dst, char *src);
```

と宣言できる標準ライブラリ関数 strcpy を使ってもよい。strcpy は、引数 src が指す文字列を (終端の 0 を含めて) 引数 dst が指す領域へコピーする (返り値としては dst をそのまま戻す)。

プログラミングおよび実習 I・第 14 回・終り

2　演習問題 (追加)

2.1　オセロプログラムをもう一度チェックする

プログラム othello27.c を othello28.c にコピーしてコンパイルし、「mprog1 othello28.c」のようにして mprog1 コマンドを実行し、第 12 回の演習問題が正しくできているかどうか、もう一度チェックしてください。othello28.c のチェックは othello27.c と比べてより厳しくなっています。

授業アンケート

「プログラミングおよび実習 I」の授業の進め方に関するアンケート

http://www602.math.ryukoku.ac.jp/%7Enakano/UnivOnly/prog1enq.html

にご協力ください。「プログラミングおよび実習 I」のホームページに、アンケートページへのリンクがあります。

付録 1: 第 12 回の演習問題 othello27.c のプログラム例

othello27.c

```
  1 #include <stdio.h>
  2 #include <ctype.h>
  3
  4 typedef int Boolean;
  5
  6 #define TRUE            (1)
  7 #define FALSE           (0)
  8
  9 #define BLACK           (1)         /* 黒石を表す定数 */
 10 #define WHITE           (-BLACK)    /* 白石を表す定数 */
 11 #define NONE            (0)         /* 石がないことを表す */
 12 #define OPPONENT(t)     (-(t))      /* t の相手 */
 13 #define BOARD_SIZE      (4)         /* オセロボードの縦横のマス数 */
 14
 15 #define IS_VALID(x)             (0 <= (x) && (x) < BOARD_SIZE)
 16 #define IS_VALID_POS(x, y)      (IS_VALID(x) && IS_VALID(y))
 17 #define FOR_EACH(x)             for ((x) = 0; (x) < BOARD_SIZE; (x)++)
 18 #define FOR_EACH_POS(x, y)      FOR_EACH(x) FOR_EACH(y)
 19
 20 typedef struct {
 21     int board[BOARD_SIZE][BOARD_SIZE];  /* 盤面の状況 */
 22     int turn;                           /* 次の手番 */
 23     int blackCount;                     /* 黒石の数 */
 24     int whiteCount;                     /* 白石の数 */
 25     Boolean passed;                     /* パスした直後か? */
 26     int lastX, lastY;                   /* 最後の手 */
 27 } OState;
 28
* 29 #define MAX_UNDO        (2*BOARD_SIZE*BOARD_SIZE)
 30
* 31 typedef struct {
* 32     int         mode;                   /* 自動対局の状態 */
* 33     OState      state;                  /* 現在局面 */
* 34     OState      undoStack[MAX_UNDO];    /* 局面のスタック */
* 35     int         sp;                     /* スタックポインタ */
```

```
* 36 } GameState;
  37
  38 /* 現在局面をスタックへプッシュする */
* 39 Boolean push(GameState *game)
* 40 {
* 41     if (game->sp == MAX_UNDO)
* 42         return FALSE;
* 43     game->undoStack[game->sp++] = game->state;
* 44     return TRUE;
* 45 }
  46
  47 /* スタックから局面をポップして、それを現在局面とする */
* 48 Boolean pop(GameState *game)
* 49 {
* 50     if (game->sp == 0)
* 51         return FALSE;
* 52     game->state = game->undoStack[--game->sp];
* 53     return TRUE;
* 54 }
  55
  56 /* キーボードからの文字を改行文字まで読み込み、大きさ size の char 型
  57    配列 buf へ格納する。 改行文字や size 番目以降の文字は無視される。
  58    buf は必ず NUL 文字で終端される。 ただし、size は 1 以上の整数でな
  59    ければならない。関数 getLine の戻り値は、buf へ格納した文字(byte)
  60    数となる。*/
  61 int getLine(char buf[], int size)
  62 {
  63     int ch;
  64     int n = 0;
  65
  66     /* 文字が読み込めなかったり、改行文字の場合は繰り返しをやめる */
  67     while ((ch = getchar()) != EOF && ch != '\n') {
  68         /* 配列 buf にまだ余裕があれば、読み込んだ文字を格納する */
  69         if (n < size - 1)
  70             buf[n++] = ch;
  71     }
  72     /* buf を NUL 文字で終端する */
  73     buf[n] = '\0';
  74     return n;
  75 }
  76
  77 /* 盤面を初期化する */
  78 void reset(OState *state)
  79 {
  80     int x, y;
  81
  82     /* 先手は黒 */
  83     state->turn = BLACK;
  84     /* すべてのマスをクリアする */
  85     FOR_EACH_POS (x, y)
  86         state->board[x][y] = NONE;
  87     /* 中央に4つの石を置く */
  88     state->board[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
  89     state->board[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
  90     state->board[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
  91     state->board[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
  92     /* blackCount と whiteCount を初期化する */
  93     state->blackCount = state->whiteCount = 2;
  94     /* パスフラグをクリア */
  95     state->passed = FALSE;
  96     /* 最後の手を初期化する */
  97     state->lastX = state->lastY = -1;
```

```
 98 }
 99
100 /* 勝負けを表示する */
101 void showResult(OState *state)
102 {
103     if (state->blackCount > state->whiteCount)
104         printf("黒の勝ちです");
105     else if (state->whiteCount > state->blackCount)
106         printf("白の勝ちです");
107     else
108         printf("引き分けです");
109     printf(" (黒%d/白%d)\n", state->blackCount, state->whiteCount);
110 }
111
112 /* 盤面を表示する */
113 void showBoard(OState *state)
114 {
115     Boolean last;
116     int x, y;
117
118     /* 列番号を表示する */
119     FOR_EACH (x)
120         printf(" %c", x + 'a');
121     printf("\n");
122     /* 各行を表示する */
123     FOR_EACH (y) {
124         printf("%c", y + '1');
125         FOR_EACH(x) {
126             last = (x == state->lastX && y == state->lastY);
127             if (state->board[x][y] == BLACK)
128                 printf("%s", last ? "★" : "●");
129             else if (state->board[x][y] == WHITE)
130                 printf("%s", last ? "☆" : "○");
131             else
132                 printf("□");
133         }
134         printf("\n");
135     }
136 }
137
138 /* (x, y) に石を置いた時に、(dx, dy) 方向で挟まれる石の数を返す。
139    update が真なら、挟まれた石を反転する */
140 int checkDir(OState *state, int x, int y, int dx, int dy, Boolean update)
141 {
142     int count = 0;      /* 挟まれた石の数 */
143     int i;
144
145     /* (dx, dy) 方向に石を調べていく */
146     while (1) {
147         /* 1つ進む */
148         x += dx;
149         y += dy;
150         /* 盤の外や空きマスに至ったら終り */
151         if (!IS_VALID_POS(x, y) || state->board[x][y] == NONE)
152             return 0;
153         /* 自分の石なら繰り返しを終了 */
154         if (state->board[x][y] == state->turn)
155             break;
156         count++;
157     }
158     if (update) {
159         /* 1マスずつ戻りながら石を反転させていく */
```

```
160         for (i = 0; i < count; i++) {
161             /* 1つ戻る */
162             x -= dx;
163             y -= dy;
164             /* 石を自分のものにする */
165             state->board[x][y] = state->turn;
166         }
167     }
168     /* 反転した石の数を返す */
169     return count;
170 }
171
172 /* (x, y) の位置に手番の石を置いたとして、挟まれる相手の石の数を返り
173    値として返す。update が真で、かつ挟まれた石があれば実際にそこに石を
174    置き手番を交代する。 */
175 int putDisk(OState *state, int x, int y, Boolean update)
176 {
177     int count = 0;      /* 挟まれた石の数 */
178     int dx, dy;
179
180     /* 盤の外や、すでに石のあるマスなら石を置かずに 0 を返す */
181     if (!IS_VALID_POS(x, y) || state->board[x][y] != NONE)
182         return 0;
183     for (dx = -1; dx <= 1; dx++)
184         for (dy = -1; dy <= 1; dy++)
185             if (dx != 0 || dy != 0)
186                 count += checkDir(state, x, y, dx, dy, update);
187     /* update が真で、石が挟まれたのなら石を置き手番を変える */
188     if (update && count > 0) {
189         if (state->turn == BLACK) {
190             state->blackCount += count + 1;
191             state->whiteCount -= count;
192         }
193         else {
194             state->whiteCount += count + 1;
195             state->blackCount -= count;
196         }
197         state->board[x][y] = state->turn;
198         /* パスフラグを下ろす */
199         state->passed = FALSE;
200         /* 最後の手を記録する */
201         state->lastX = x;
202         state->lastY = y;
203         /* マスがすべて埋っていたら終局する */
204         if (state->blackCount + state->whiteCount == BOARD_SIZE*BOARD_SIZE)
205             state->turn = NONE;
206         else
207             state->turn = OPPONENT(state->turn);
208     }
209     /* 反転した石の数を返す */
210     return count;
211 }
212
213 /* 手番をパスして、返り値として 1 を返す。 相手を挟める手がある場合
214    にはパスをせずに 0 を返す */
215 Boolean pass(OState *state)
216 {
217     int x, y;
218
219     FOR_EACH_POS(x, y) {
220         if (putDisk(state, x, y, FALSE) > 0)
221             return FALSE;
```

```
 222     }
 223     if (state->passed)
 224         state->turn = NONE;
 225     else
 226         state->turn = OPPONENT(state->turn);
 227     state->passed = TRUE;
 228     return TRUE;
 229 }
 230
 231 /* 自動的に選んだ手を差して TRUE を返す。 パスしたら FALSE を返す */
 232 Boolean autoMove(OState *state)
 233 {
 234     int c, maxCount = 0;
 235     int x, y, maxX = -1, maxY = -1;
 236
 237     FOR_EACH_POS(x, y) {
 238         c = putDisk(state, x, y, FALSE);
 239         if (c > maxCount) {
 240             maxX = x;
 241             maxY = y;
 242             maxCount = c;
 243         }
 244     }
 245     /* 相手の石を最も多く反転させるような手を選んで差す。
 246        手がなければパスする */
 247     if (maxCount > 0) {
 248         putDisk(state, maxX, maxY, TRUE);
 249         return TRUE;
 250     }
 251     pass(state);
 252     return FALSE;
 253 }
 254
 255 /* 待ったする */
*256 void undo(GameState *game)
*257 {
*258     int t = game->state.turn;
*259
*260     /* まず、push されている現在局面を pop する */
*261     pop(game);
*262     /* 自分の手番になるまで過去の局面を pop する */
*263     do {
*264         if (!pop(game))
*265             return;
*266     } while(game->state.turn != t);
*267 }
 268
 269 /* ユーザーが指定した位置に石を置く */
*270 Boolean inputMove(GameState *game)
 271 {
*272     OState *state = &game->state;
 273     char buf[100], ch1, ch2, ch3;
 274     int n;
 275
 276     /* 位置を入力する */
 277     while (TRUE) {
 278         /* 終局していたら勝ち負けを表示してプログラムを終了する */
 279         if (state->turn == NONE) {
 280             showResult(state);
 281             return TRUE;
 282         }
 283         /* 現在の局面をプッシュして、局面が変化する準備をする */
```

```
*284            if (!push(game)) {
*285                printf("局面のスタックが溢れました\n");
*286                return TRUE;
*287            }
 288            printf("%s番 (黒%d/白%d) = ", (state->turn == BLACK) ? "黒" : "白",
 289                   state->blackCount, state->whiteCount);
 290            /* 自動対局モードなら autoMove を呼び出す */
*291            if (state->turn == game->mode) {
*292                if (autoMove(state)) {
*293                    printf("%c%c\n", state->lastX + 'a', state->lastY + '1');
*294                    break;
*295                }
*296                printf("p\n");
*297                continue;
*298            }
 299            /* キーボードからの1行分の入力を buf に格納する */
 300            if (getLine(buf, sizeof buf) < 1) {
*301                pop(game);
 302                continue;
 303            }
 304            /* buf から空白類を無視して、最大3文字を抜き出す */
 305            n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
 306            ch1 = tolower(ch1);
 307            ch2 = tolower(ch2);
 308            if (n == 1) {
 309                /* q の1文字なら 1 を返り値として return する */
 310                if (ch1 == 'q')
 311                    return TRUE;
 312                /* t の1文字ならコンピュータに手を考えさせる */
 313                if (ch1 == 't') {
 314                    if (autoMove(state))
 315                        break;
 316                    continue;
 317                }
 318                /* u の1文字なら待ったをする */
*319                if (ch1 == 'u') {
*320                    undo(game);
*321                    break;
*322                }
 323                /* p の1文字なら手番をパスする */
 324                if (ch1 == 'p') {
 325                    if (!pass(state)) {
*326                        pop(game);
 327                        printf("パスはできません\n");
 328                    }
 329                    continue;
 330                }
 331            }
 332            /* 2文字でないなら再入力を促す */
 333            if (n != 2) {
*334                pop(game);
 335                continue;
 336            }
 337            /* とりあえず c5 のような形式の入力と仮定してみる */
 338            if (putDisk(state, ch1 - 'a', ch2 - '1', TRUE) > 0)
 339                break;
 340            /* 逆に 5c のような形式の入力と仮定してみる */
 341            if (putDisk(state, ch2 - 'a', ch1 - '1', TRUE) > 0)
 342                break;
*343            pop(game);
 344            printf("そこには置けません\n");
 345        }
```

```
 346     return FALSE;
 347 }
 348
*349 int main(int argc, char *argv[])
 350 {
*351     GameState game;
 352
*353     game.mode = NONE;
*354     game.sp = 0;
 355
*356     if (argc > 1) {
*357         if (argc != 2) {
*358             printf("引数の数が多すぎます\n");
*359             return 1;
*360         }
*361         if (strcmp(argv[1], "-b") == 0)
*362             game.mode = BLACK;
*363         else if (strcmp(argv[1], "-w") == 0)
*364             game.mode = WHITE;
*365         else {
*366             printf("「%s」は認識できないオプションです\n", argv[1]);
*367             return 1;
*368         }
*369     }
 370
*371     reset(&game.state);
 372     do {
*373         showBoard(&game.state);
*374     } while (!inputMove(&game));
 375     return 0;
 376 }
```

関数 inputMove のもう 1 つの例

```
  /* 現在の局面を保存する */
* void save(GameState *game)
* {
*     if (!push(game)) {
*         printf("局面のスタックが溢れました\n");
*         exit(1);
*     }
* }

  /* ユーザーが指定した位置に石を置く */
* Boolean inputMove(GameState *game)
  {
*     OState *state = &game->state;
      char buf[100], ch1, ch2, ch3;
      int n;

*     save(game);

      /* 位置を入力する */
      while (TRUE) {
          /* 終局していたら勝ち負けを表示してプログラムを終了する */
          if (state->turn == NONE) {
              showResult(state);
              return TRUE;
          }
          printf("%s番 (黒%d/白%d) = ", (state->turn == BLACK) ? "黒" : "白",
                  state->blackCount, state->whiteCount);
*         /* 自動対局モードなら autoMove を呼び出す */
```

```
*         if (state->turn == game->mode) {
*             if (autoMove(state)) {
*                 printf("%c%c\n", state->lastX + 'a', state->lastY + '1');
*                 break;
*             }
*             printf("p\n");
*             save(game);
*             continue;
*         }
          /* キーボードからの1行分の入力を buf に格納する */
          if (getLine(buf, sizeof buf) < 1)
              continue;
          /* buf から空白類を無視して、最大3文字を抜き出す */
          n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
          ch1 = tolower(ch1);
          ch2 = tolower(ch2);
          if (n == 1) {
              /* q の1文字なら 1 を返り値として return する */
              if (ch1 == 'q')
                  return TRUE;
              /* t の1文字ならコンピュータに手を考えさせる */
              if (ch1 == 't') {
                  if (autoMove(state))
                      break;
*                 save(game);
                  continue;
              }
              /* u の1文字なら待ったをする */
*             if (ch1 == 'u') {
*                 undo(game);
*                 break;
*             }
              /* p の1文字なら手番をパスする */
              if (ch1 == 'p') {
                  if (!pass(state))
                      printf("パスはできません\n");
*                 else
*                     save(game);
                  continue;
              }
          }
          /* 2文字でないなら再入力を促す */
          if (n != 2)
              continue;
          /* とりあえず c5 のような形式の入力と仮定してみる */
          if (putDisk(state, ch1 - 'a', ch2 - '1', TRUE) > 0)
              break;
          /* 逆に 5c のような形式の入力と仮定してみる */
          if (putDisk(state, ch2 - 'a', ch1 - '1', TRUE) > 0)
              break;
          printf("そこには置けません\n");
      }
      return FALSE;
  }
```

関数 inputMove のさらにもう 1 つの例

```
  /* ユーザーが指定した位置に石を置く */
* Boolean inputMove(GameState *game)
  {
*     OState *state = &game->state;
      char buf[100], ch1, ch2, ch3;
```

```
    int n;
    /* 現在局面をスタックに保存する必要があるか */
*   Boolean saveCurrent = TRUE;

    /* 位置を入力する */
    while (TRUE) {
        /* 終局していたら勝ち負けを表示してプログラムを終了する */
        if (state->turn == NONE) {
            showResult(state);
            return TRUE;
        }
        /* 必要なら、現在の局面をプッシュして、局面が変化する準備をする */
*       if (saveCurrent) {
*           if (!push(game)) {
*               printf("局面のスタックが溢れました\n");
*               return TRUE;
*           }
*           saveCurrent = FALSE;
*       }
        printf("%s番 (黒%d/白%d) = ", (state->turn == BLACK) ? "黒" : "白",
               state->blackCount, state->whiteCount);
        /* 自動対局モードなら autoMove を呼び出す */
*       if (state->turn == game->mode) {
*           if (autoMove(state)) {
*               printf("%c%c\n", state->lastX + 'a', state->lastY + '1');
*               break;
*           }
*           printf("p\n");
*           saveCurrent = TRUE;
*           continue;
*       }
        /* キーボードからの1行分の入力を buf に格納する */
        if (getLine(buf, sizeof buf) < 1)
            continue;
        /* buf から空白類を無視して、最大3文字を抜き出す */
        n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
        ch1 = tolower(ch1);
        ch2 = tolower(ch2);
        if (n == 1) {
            /* q の1文字なら 1 を返り値として return する */
            if (ch1 == 'q')
                return TRUE;
            /* t の1文字ならコンピュータに手を考えさせる */
            if (ch1 == 't') {
                if (autoMove(state))
                    break;
*               saveCurrent = TRUE;
                continue;
            }
            /* u の1文字なら待ったをする */
*           if (ch1 == 'u') {
*               undo(game);
*               break;
*           }
            /* p の1文字なら手番をパスする */
            if (ch1 == 'p') {
                if (!pass(state))
                    printf("パスはできません\n");
                else
*                   saveCurrent = TRUE;
                continue;
            }
```

```
        }
        /* 2文字でないなら再入力を促す */
        if (n != 2)
            continue;
        /* とりあえず c5 のような形式の入力と仮定してみる */
        if (putDisk(state, ch1 - 'a', ch2 - '1', TRUE) > 0)
            break;
        /* 逆に 5c のような形式の入力と仮定してみる */
        if (putDisk(state, ch2 - 'a', ch1 - '1', TRUE) > 0)
            break;
        printf("そこには置けません\n");
    }
    return FALSE;
}
```

付録 2: 第 14 回の模擬試験の解答例

問題 1 の関数 revstr

```
void revstr(char str[])
{
    char ch;
    int i, j;

    i = j = 0;
    while (str[j] != '\0')
        j++;

    while (i < --j) {
        ch = str[i];
        str[i++] = str[j];
        str[j] = ch;
    }
}
```

問題 2 の関数 show と main

```
void show(struct State *s)
{
    int i;

    FOR_EACH(i) {
        if (s->line[i] == 1)
            printf("(%c)", 'a' + i);
        else
            printf("   ");
    }
    printf("\n");
}

int main ()
{
    struct State st;
    char ch;
    int i, j, n;

    FOR_EACH(i)
        st.line[i] = 1;
    st.rest = LEN;
```

```
    show(&st);
    do {
        printf ("? ");
        if (scanf(" %c%d", &ch, &n) != 2)
            continue;
        i = ch - 'a';
        if (n <= 0 || MAX < n || !IN_LINE(i) || !IN_LINE(i+n-1))
            continue;
        for (j = i; j < i + n; j++) {
            if (st.line[j] == 0)
                break;
        }
        if (j < i + n)
            continue;
        for (j = i; j < i + n; j++)
            st.line[j] = 0;
        st.rest -= n;
        show(&st);
    } while (st.rest > 1);
    return 0;
}
```

問題 3 の関数 add と print

```
void add(Stat *st, char *line)
{
    int i;

    for (i = 0; i < st->num; i++) {
        if (strcmp(st->table[i].str, line) == 0) {
            st->table[i].count++;
            return;
        }
    }

    if (st->num >= NUM)
        return;

    for (i = 0; line[i]; i++)
        st->table[st->num].str[i] = line[i];
    st->table[st->num].str[i] = '\0';
    st->table[st->num].count = 1;
    st->num++;
}

void print(Stat *st)
{
    int i;

    printf("回数 : 文字列\n");
    for (i = 0; i < st->num; i++)
        printf("%4d : %s\n", st->table[i].count, st->table[i].str);
}
```