```
01: #include <stdio.h>                                  // perror() のため
02: #include <stdlib.h>                                 // eixt() のため
03: #include <string.h>                                 // strcmp(), strchr() のため
04: #include <unistd.h>                                 // fork(),exec(),close()のため
05: #include <sys/wait.h>                               // wait() のため
06: #include <ctype.h>                                  // ispace() のため
07: #include <fcntl.h>                                  // open() のため
08: #define MAXLINE 1000                               // コマンド行の最大文字数
09: #define MAXARGS 60                                 // コマンド行文字列の最大数
10:
11: int parse(char *p, char *args[]) {                  // コマンド行を解析する
12:   int i=0;                                          // 解析後文字列の数
13:   for (;;) {
14:     while (isspace(*p)) *p++ = '\0';                // 空白を'\0'に書換える
15:     if (*p=='\0' || i>=MAXARGS) break;              // コマンド行の終端に到達で終了
16:     args[i++] = p;                                  // 文字列を文字列配列に記録
17:     while (*p!='\0' && !isspace(*p)) p++;           // 文字列の最後まで進む
18:   }
19:   args[i] = NULL;                                   // 文字列配列の終端マーク
20:   return *p=='\0';                                  // 解析完了なら 1 を返す
21: }
22:
23: void cdCom(char *args[]) {                          // cd コマンドを実行する
24:   if (args[1]==NULL || args[2]!=NULL) {             //   引数を確認して
25:     fprintf(stderr,"Usage: cd DIR\n");              //     過不足ありなら使い方表示
26:   } else if (chdir(args[1])<0) {                    //   親プロセスが chdir する
27:     perror(args[1]);                                //     chdirに失敗したらperror
28:   }
29: }
30:
31: void setenvCom(char *args[]) {                      // setenv コマンドを実行する
32:   if (args[1]==NULL || args[2]==NULL || args[3]!=NULL) {   // 引数を確認して
33:     fprintf(stderr,"Usage: setenv NAME VAL\n");     //   過不足ありなら使い方表示
34:   } else if (setenv(args[1], args[2], 1)<0) {       //   親プロセスがsetenvする
35:     perror(args[1]);                                //     setenvに失敗したらperror
36:   }
37: }
38:
39: void unsetenvCom(char *args[]) {                    // unsetenv コマンドを実行する
40:   if (args[1]==NULL || args[2]!=NULL) {             //   引数を確認して
41:     fprintf(stderr,"Usage: unsetenv NAME\n");       //     過不足ありなら使い方表示
42:   } else if (unsetenv(args[1])<0) {                 //   親プロセスがunsetenvする
43:     perror(args[1]);                                //     unsetenvに失敗ならperror
44:   }
45: }
46:
47: char *ofile;                                        // 出力リダイレクトファイル名
48: char *ifile;                                        // 入力リダイレクトファイル名
49:
50: void findRedirect(char *args[]) {                   // リダイレクトの指示を探す
51:   int i, j;
52:   ofile = ifile = NULL;
53:   for (i=j=0; args[i]!=NULL; i++) {                 // コマンド行の全文字列について
54:     if (strcmp(args[i], "<")==0) {                  // 入力リダイレクト発見
55:       ifile = args[++i];                            //   ファイル名を記録する
56:       if (ifile==NULL) break;                       //   ファイル名が無かった
57:     } else if (strcmp(args[i], ">")==0) {           // 出力リダイレクト発見
58:       ofile = args[++i];                            //   ファイル名を記録する
59:       if (ofile==NULL) break;                       //   ファイル名が無かった
60:     } else {                                        // どちらでもない
61:       args[j++] = args[i];                          //   文字列をargsに記録する
62:     }
63:   }
64:   args[j] = NULL;
```

```
65: }
66:
67: void redirect(int fd, char *path, int flag) {   // リダイレクト処理をする
68:   //
69:   // externalCom 関数のどこかから呼び出される
70:   //
71:   // fd   : リダイレクトするファイルディスクリプタ
72:   // path : リダイレクト先ファイル
73:   // flag : open システムコール渡すフラグ
74:   //        入力の場合 O_RDONLY
75:   //        出力の場合 O_WRONLY|O_TRUNC|O_CREAT
76:   //
77: }
78:
79: void externalCom(char *args[]) {                // 外部コマンドを実行する
80:   int pid, status;
81:   if ((pid = fork()) < 0) {                     //   新しいプロセスを作る
82:     perror("fork");                             //     fork 失敗
83:     exit(1);                                    //     非常事態，親を終了
84:   }
85:   if (pid==0) {                                 //   子プロセスなら
86:     execvp(args[0], args);                      //     コマンドを実行
87:     perror(args[0]);                            //     exec 失敗
88:     exit(1);                                    //     よくある，子を終了
89:   } else {                                      //   親プロセスなら
90:     while (wait(&status) != pid)                //     子の終了を待つ
91:       ;
92:   }
93: }
94:
95: void execute(char *args[]) {                   // コマンドを実行する
96:   if (strcmp(args[0], "cd")==0) {               // cd コマンド
97:     cdCom(args);
98:   } else if (strcmp(args[0], "setenv")==0) {    // setenv コマンド
99:     setenvCom(args);
100:   } else if (strcmp(args[0], "unsetenv")==0) {  // unsetenv コマンド
101:     unsetenvCom(args);
102:   } else {                                      // その他は外部コマンド
103:     externalCom(args);
104:   }
105: }
106:
107: int main() {
108:   char buf[MAXLINE+2];                          // コマンド行を格納する配列
109:   char *args[MAXARGS+1];                        // 解析結果を格納する文字列配列
110:   for (;;) {
111:     printf("Command: ");                        // プロンプトを表示する
112:     if (fgets(buf,MAXLINE+2,stdin)==NULL) {     // コマンド行を入力する
113:       printf("\n");                             // EOF なら
114:       break;                                    //   正常終了する
115:     }
116:     if (strchr(buf, '\n')==NULL) {              // '\n'がバッファにない場合は
117:       fprintf(stderr, "行が長すぎる\n");        //   コマンド行が長すぎたので
118:       return 1;                                 //   異常終了する
119:     }
120:     if (!parse(buf,args)) {                     // コマンド行を解析する
121:       fprintf(stderr, "引数が多すぎる\n");      //   文字列が多すぎる場合は
122:       continue;                                 //   ループの先頭に戻る
123:     }
124:     findRedirect(args);                         // リダイレクトの指示を見つける
125:     if (args[0]!=NULL) execute(args);           //   コマンドを実行する
126:   }
127:   return 0;
128: }
```