

課題 No.12 の解答例

myshell にリダイレクト機能を追加しなさい.

```
1 //
2 // myshell.c : 簡易UNIXシェル (リダイレクト機能完成版)
3 //
4 #include <stdio.h> // perror() のため
5 #include <stdlib.h> // exit() のため
6 #include <string.h> // strcmp(), strchr() のため
7 #include <unistd.h> // fork(), exec(), close() のため
8 #include <sys/wait.h> // wait() のため
9 #include <ctype.h> // ispace() のため
10 #include <fcntl.h> // open() のため
11 #define MAXLINE 1000 // コマンド行の最大文字数
12 #define MAXARGS 60 // コマンド行文字列の最大数
13
14 int parse(char *p, char *args[]) { // コマンド行を解析する
15     int i=0; // 解析後文字列の数
16     for (;;) {
17         while (isspace(*p)) *p++ = '\0'; // 空白を '\0' に書換える
18         if (*p=='\0' || i>=MAXARGS) break; // コマンド行の終端に到達で終了
19         args[i++] = p; // 文字列を文字列配列に記録
20         while (*p!='\0' && !isspace(*p)) p++; // 文字列の最後まで進む
21     }
22     args[i] = NULL; // 文字列配列の終端マーク
23     return *p=='\0'; // 解析完了なら 1 を返す
24 }
25
26 void cdCom(char *args[]) { // cd コマンドを実行する
27     if (args[1]==NULL || args[2]!=NULL) { // 引数を確認して
28         fprintf(stderr, "Usage: cd DIR\n"); // 過不足ありなら使い方表示
29     } else if (chdir(args[1])<0) { // 親プロセスが chdir する
30         perror(args[1]); // chdir に失敗したら perror
31     }
32 }
33
34 void setenvCom(char *args[]) { // setenv コマンドを実行する
35     if (args[1]==NULL || args[2]==NULL || args[3]!=NULL) { // 引数を確認して
36         fprintf(stderr, "Usage: setenv NAME VAL\n"); // 過不足ありなら使い方表示
37     } else if (setenv(args[1], args[2], 1)<0) { // 親プロセスが setenv する
38         perror(args[1]); // setenv に失敗したら perror
39     }
40 }
41
42 void unsetenvCom(char *args[]) { // unsetenv コマンドを実行する
43     if (args[1]==NULL || args[2]!=NULL) { // 引数を確認して
44         fprintf(stderr, "Usage: unsetenv NAME\n"); // 過不足ありなら使い方表示
45     } else if (unsetenv(args[1])<0) { // 親プロセスが unsetenv する
46         perror(args[1]); // unsetenv に失敗なら perror
47     }
48 }
49
50 char *ofile; // 出力リダイレクトファイル名
```

```
51 char *ifile; // 入力リダイレクトファイル名
52
53 void findRedirect(char *args[]) { // リダイレクトの指示を探す
54     int i, j;
55     ofile = ifile = NULL;
56     for (i=j=0; args[i]!=NULL; i++) { // コマンド行の全文字列について
57         if (strcmp(args[i], "<")==0) { // 入力リダイレクト発見
58             ifile = args[++i]; // ファイル名を記録する
59             if (ifile==NULL) break; // ファイル名が無かった
60         } else if (strcmp(args[i], ">")==0) { // 出力リダイレクト発見
61             ofile = args[++i]; // ファイル名を記録する
62             if (ofile==NULL) break; // ファイル名が無かった
63         } else { // どちらでもない
64             args[j++] = args[i]; // 文字列をargsに記録する
65         }
66     }
67     args[j] = NULL;
68 }
69
70 void redirect(int fd, char *path, int flag) { // リダイレクト処理をする
71     close(fd); // リダイレクトするfdをクローズ
72     int nfd = open(path, flag, 0644); // 新しくオープン
73     if (nfd<0) { // open がエラーなら
74         perror(path); // エラーメッセージを表示
75         exit(1); // 子プロセスを終了する
76     } // プログラムにバグが無ければ他のエラーの心配は無いハズ
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         if (ifile!=NULL) { // 入力リダイレクトがあれば
87             redirect(0, ifile, O_RDONLY); // リダイレクトする
88         }
89         if (ofile!=NULL) { // 出力リダイレクトがあれば
90             redirect(1, ofile, O_WRONLY|O_TRUNC|O_CREAT); // リダイレクトする
91         }
92         execvp(args[0], args); // コマンドを実行
93         perror(args[0]);
94         exit(1);
95     } else { // 親プロセスなら
96         while (wait(&status) != pid) // 子の終了を待つ
97             ;
98     }
99 }
100
101 void execute(char *args[]) { // コマンドを実行する
102     if (strcmp(args[0], "cd")==0) { // cd コマンド
103         cdCom(args);
```

```
104 } else if (strcmp(args[0], "setenv")==0) { // setenv コマンド
105     setenvCom(args);
106 } else if (strcmp(args[0], "unsetenv")==0) { // unsetenv コマンド
107     unsetenvCom(args);
108 } else { // その他は外部コマンド
109     externalCom(args);
110 }
111 }
112
113 int main() {
114     char buf[MAXLINE+2]; // コマンド行を格納する配列
115     char *args[MAXARGS+1]; // 解析結果を格納する文字列配列
116     for (;;) {
117         printf("Command: "); // プロンプトを表示する
118         if (fgets(buf,MAXLINE+2,stdin)==NULL) { // コマンド行を入力する
119             printf("\n"); // EOF なら
120             break; // 正常終了する
121         }
122         if (strchr(buf, '\n')==NULL) { // '\n'がバッファにない場合は
123             fprintf(stderr, "行が長すぎる\n"); // コマンド行が長すぎたので
124             return 1; // 異常終了する
125         }
126         if (!parse(buf,args)) { // コマンド行を解析する
127             fprintf(stderr, "引数が多すぎる\n"); // 文字列が多すぎる場合は
128             continue; // ループの先頭に戻る
129         }
130         findRedirect(args); // リダイレクトの指示を見つける
131         if (args[0]!=NULL) execute(args); // コマンドを実行する
132     }
133     return 0;
134 }
```

```
/* 実行結果
% make                                     ; コンパイルエラーなし
cc -D_GNU_SOURCE -Wall -std=c99 -o myshell myshell.c
% ./myshell                               ; myshellを起動
Command: ls
Makefile      README.pdf      myshell.c
README.md     myshell
Command: echo aaa > a.txt                ; ファイルの作成をテスト
Command: cat a.txt                        ; 内容を確認
aaa                                           ; ファイルができている
Command: ls > a.txt                       ; ファイルの上書きをテスト
Command: cat < a.txt                      ; 内容を確認
Makefile                                           ; 上書きされている
README.md
README.pdf
a.txt
myshell
myshell.c
Command: grep .pdf < a.txt                ; 入力リダイレクトをテスト
README.pdf                                ; 正しい出力がされた
Command: grep .pdf < a.txt > b.txt        ; 入出力の同時リダイレクト
Command: cat b.txt                        ; 正しく処理されている
README.pdf
Command: grep .pdf < c.txt                ; 入力リダイレクトのエラー処理
c.txt: No such file or directory          ; 正しいエラーメッセージ
Command: grep .pdf > /d.txt              ; 出力リダイレクトのエラー処理
/d.txt: Read-only file system             ; 正しいエラーメッセージ
Command: grep .pdf < c.txt > /d.txt      ; 同時リダイレクトのエラー処理
c.txt: No such file or directory          ; 片方のメッセージだけ表示
Command:
*/
```