## 課題 No.11 の解答例 1

myshell に環境変数の管理機能 (setenv, unsetenv) を追加しなさい.

```
// perror() のため
#include <stdio.h>
#include <stdlib.h>
                                         // eixt() のため
#include <string.h>
                                         // strcmp(), strchr() のため
#include <unistd.h>
                                         // fork(), exec(), close()のため
#include <sys/wait.h>
                                         // wait() のため
                                         // ispace() のため
#include <ctype.h>
                                         // コマンド行の最大文字数
#define MAXLINE 1000
#define MAXARGS 60
                                         // コマンド行文字列の最大数
int parse(char *p, char *args[]) {
                                         // コマンド行を解析する
 int i=0;
                                         // 解析後文字列の数
 for (;;) {
                                        // 空白を '\o'に書換える
   while (isspace(*p)) *p++ = '\0';
                                        // コマンド行の終端に到達で終了
   if (*p=='\0' || i>=MAXARGS) break;
                                        // 文字列を文字列配列に記録
  args[i++] = p;
  while (*p!='\0' && !isspace(*p)) p++;
                                        // 文字列の最後まで進む
                                         // 文字列配列の終端マーク
 args[i] = NULL;
 return *p=='\0';
                                         // 解析完了なら 1 を返す
                                         // コマンドを実行する
void execute(char *args[]) {
 if (strcmp(args[0], "cd")==0) {
                                         // cd (内部コマンド)
   if (args[1]==NULL || args[2]!=NULL) {
                                         // 引数を確認して
                                        //
     fprintf(stderr, "Usage: cd DIR\n");
                                              過不足ありなら使い方表示
                                         //
                                             親プロセスが chdir する
   } else if (chdir(args[1])<0) {</pre>
                                               chdirに失敗したらperror
    perror(args[1]);
 } else if (strcmp(args[0], "setenv")==0) { // setenv (内部コマンド)
   if (args[1]==NULL || args[2]==NULL || args[3]!=NULL) { // 引数を確認して
    fprintf(stderr, "Usage: setenv NAME VAL\n"); // 過不足ありなら使い方表示
   } else if (setenv(args[1], args[2], 1)<0) { // 親プロセスがsetenvする
                                              setenvに失敗したらperror
    perror(args[1]);
 } else if (strcmp(args[0], "unsetenv")==0) { // unsetenv (内部コマンド)
                                         // 引数を確認して
   if (args[1] == NULL || args[2]! = NULL) {
     fprintf(stderr, "Usage: unsetenv NAME\n"); //
                                              過不足ありなら使い方表示
                                             親プロセスがunsetenvする
   } else if (unsetenv(args[1])<0) {</pre>
                                         //
                                               unsetenvに失敗ならperror
     perror(args[1]);
   }
                                         // 外部コマンドなら
 } else {
   int pid, status;
   if ((pid = fork()) < 0) {</pre>
                                         // 新しいプロセスを作る
    perror("fork");
     exit(1);
   if (pid==0) {
                                             子プロセスなら
                                              コマンドを実行
    execvp(args[0], args);
     perror(args[0]);
     exit(1);
   } else {
                                         // 親プロセスなら
```

```
while (wait(&status) != pid)
                                        //
                                             子の終了を待つ
   }
 }
}
int main() {
                                        // コマンド行を格納する配列
 char buf[MAXLINE+2];
 char *args[MAXARGS+1];
                                        // 解析結果を格納する文字列配列
 for (;;) {
                                        // プロンプトを表示する
   printf("Command: ");
                                        // コマンド行を入力する
   if (fgets(buf,MAXLINE+2,stdin)==NULL) {
    printf("\n");
                                        // EOF なら
                                        // 正常終了する
    break;
   if (strchr(buf, '\n')==NULL) {
                                        // '\n'がバッファにない場合は
                                       // コマンド行が長すぎたので
    fprintf(stderr, "行が長すぎる\n");
                                        // 異常終了する
    return 1;
                                        // コマンド行を解析する
   if (!parse(buf,args)) {
    fprintf(stderr, "引数が多すぎる\n"); // 文字列が多すぎる場合は
                                       // ループの先頭に戻る
     continue;
   if (args[0]!=NULL) execute(args);
                                // コマンドを実行する
 return 0;
}
/* 実行例(動作テスト結果)
                                <--- myshell を起動
% ./myshell
                                <--- 環境変数を作ることができるかテスト
Command: printenv A
Command: setenv A B
Command: printenv A
                                <--- 環境変数を上書きできるかテスト
Command: setenv A C
Command: printenv A
C
                                <--- 環境変数を削除できるかテスト
Command: unsetenv A
Command: printenv A
Command: setenv
                                <--- setenuの引数の過不足テスト
Usage: setenv NAME VAL
Command: setenv A
Usage: setenv NAME VAL
Command: setenv A B C
Usage: setenv NAME VAL
Command: unsetenv
                                 <--- unsetenvの引数の過不足テスト
Usage: unsetenv NAME
Command: unsetenv A B
Usage: unsetenv NAME
                                <--- setenv()がエラーを起こす場合
Command: setenv A= B
A=: Invalid argument
                                <--- unsetenv()がエラーを起こす場合
Command: unsetenv A=
A=: Invalid argument
                                 <--- ^D を入力すると EOF になる
Command: ^D
*/
```

## 課題 No.11 の解答例2

機能拡張しやすいように関数の構成を工夫した。

```
#include <stdio.h>
                                         // perror() のため
#include <stdlib.h>
                                         // eixt() のため
#include <string.h>
                                         // strcmp(), strchr() のため
#include <unistd.h>
                                         // fork(), exec()のため
                                         // wait() のため
#include <sys/wait.h>
#include <ctype.h>
                                         // ispace() のため
                                         // コマンド行の最大文字数
#define MAXLINE 1000
                                        // コマンド行文字列の最大数
#define MAXARGS 60
int parse(char *p, char *args[]) {
                                        // コマンド行を解析する
                                        // 解析後文字列の数
 int i=0;
 for (;;) {
   while (isspace(*p)) *p++ = '\0';
                                         // 空白を '\o'に書換える
   if (*p=='\0' || i>=MAXARGS) break;
                                        // コマンド行の終端に到達で終了
                                        // 文字列を文字列配列に記録
   args[i++] = p;
   while (*p!='\0' && !isspace(*p)) p++;
                                        // 文字列の最後まで進む
                                         // 文字列配列の終端マーク
 args[i] = NULL;
 return *p=='\0';
                                         // 解析完了なら 1 を返す
                                         // cd コマンドを実行する
void cdCom(char *args[]) {
                                       // 引数を確認して
 if (args[1]==NULL || args[2]!=NULL) {
                                             過不足ありなら使い方表示
  fprintf(stderr, "Usage: cd DIR\n");
                                       //
                                        // 親プロセスが chdir する
 } else if (chdir(args[1])<0) {</pre>
   perror(args[1]);
                                        // chdirに失敗したらperror
 }
}
void setenvCom(char *args[]) {
                                        // setenu コマンドを実行する
 if (args[1]==NULL || args[2]==NULL || args[3]!=NULL) { // 引数を確認して
   fprintf(stderr, "Usage: setenv NAME VAL\n"); // 過不足ありなら使い方表示
 } else if (setenv(args[1], args[2], 1)<0) { // 親プロセスがsetenvする
   perror(args[1]);
                                         //
                                              setenvに失敗したらperror
 }
}
void unsetenvCom(char *args[]) {
                                         // unsetenv コマンドを実行する
 if (args[1]==NULL || args[2]!=NULL) {
                                        // 引数を確認して
                                              過不足ありなら使い方表示
   fprintf(stderr, "Usage: unsetenv NAME\n");
                                       //
 } else if (unsetenv(args[1])<0) {</pre>
                                        // 親プロセスがunsetenvする
                                         //
   perror(args[1]);
                                              unsetenvに失敗ならperror
 }
}
                                        // 外部コマンドを実行する
void externalCom(char *args[]) {
 int pid, status;
                                        // 新しいプロセスを作る
 if ((pid = fork()) < 0) {</pre>
   perror("fork");
   exit(1);
```

```
// 子プロセスなら
 if (pid==0) {
  execvp(args[0], args);
                                    //
                                          コマンドを実行
  perror(args[0]);
  exit(1);
                                    // 親プロセスなら
 } else {
                                         子の終了を待つ
   while (wait(&status) != pid)
 }
                                    // コマンドを実行する
void execute(char *args[]) {
if (strcmp(args[0], "cd")==0) {
                                    // cd (内部コマンド)
 cdCom(args);
} else if (strcmp(args[0], "setenv")==0) {
                                    // setenv (内部コマンド)
 setenvCom(args);
} else if (strcmp(args[0], "unsetenv")==0) { // unsetenv (内部コマンド)
 unsetenvCom(args);
                                     // 外部コマンドなら
} else {
 externalCom(args);
}
int main() {
                                   // コマンド行を格納する配列
 char buf[MAXLINE+2];
 char *args[MAXARGS+1];
                                    // 解析結果を格納する文字列配列
 for (;;) {
  printf("Command: ");
                                    // プロンプトを表示する
                                    // コマンド行を入力する
   if (fgets(buf,MAXLINE+2,stdin)==NULL) {
                                    // EOF なら
   printf("\n");
                                    // 正常終了する
    break;
                                   // '\n'がバッファにない場合は
   if (strchr(buf, '\n')==NULL) {
   // 異常終了する
   return 1;
                                    // コマンド行を解析する
   if (!parse(buf,args)) {
    fprintf(stderr, "引数が多すぎる\n"); // 文字列が多すぎる場合は
                                   // ループの先頭に戻る
    continue;
  if (args[0]!=NULL) execute(args); // コマンドを実行する
 return 0;
```