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.

```
#include <stdio.h>                                  // perror() のため
#include <stdlib.h>                                 // exit() 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
#include <ctype.h>                                  // ispace() のため
#define MAXLINE 1000                                // コマンド行の最大文字数
#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 (;;) {
    while (isspace(*p)) *p++ = '\0';                // 空白を'\0'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
    args[i++] = p;                                  // 文字列を文字列配列に記録
    while (*p!='\0' && !isspace(*p)) p++;           // 文字列の最後まで進む
  }
  args[i] = NULL;                                   // 文字列配列の終端マーク
  return *p=='\0';                                  // 解析完了なら 1 を返す
}
void execute(char *args[]) {                        // コマンドを実行する
  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
    } else if (chdir(args[1])<0) {                  //   親プロセスが chdir する
      perror(args[1]);                              //     chdirに失敗したらperror
    }
  } 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
      perror(args[1]);                              //     setenvに失敗したらperror
    }
  } else if (strcmp(args[0], "unsetenv")==0) {      // unsetenv (内部コマンド)
    if (args[1]==NULL || args[2]!=NULL) {           //   引数を確認して
      fprintf(stderr,"Usage: unsetenv NAME\n");     //     過不足ありなら使い方表示
    } else if (unsetenv(args[1])<0) {               //   親プロセスがunsetenvする
      perror(args[1]);                              //     unsetenvに失敗ならperror
    }
  } else {                                          // 外部コマンドなら
    int pid, status;
    if ((pid = fork()) < 0) {                       //   新しいプロセスを作る
      perror("fork");
      exit(1);
    }
    if (pid==0) {                                   //   子プロセスなら
      execvp(args[0], args);                        //     コマンドを実行
      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() {
  char buf[MAXLINE+2];                          // コマンド行を格納する配列
  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
    if (fgets(buf,MAXLINE+2,stdin)==NULL) {     // コマンド行を入力する
      printf("\n");                             // EOF なら
      break;                                    //   正常終了する
    }
    if (strchr(buf, '\n')==NULL) {              // '\n'がバッファにない場合は
      fprintf(stderr, "行が長すぎる\n");        //   コマンド行が長すぎたので
      return 1;                                 //   異常終了する
    }
    if (!parse(buf,args)) {                     // コマンド行を解析する
      fprintf(stderr, "引数が多すぎる\n");      //   文字列が多すぎる場合は
      continue;                                 //   ループの先頭に戻る
    }
    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                          <--- myshell を起動
Command: printenv A                 <--- 環境変数を作ることができるかテスト
Command: setenv A B
Command: printenv A
B
Command: setenv A C                 <--- 環境変数を上書きできるかテスト
Command: printenv A
C
Command: unsetenv A                 <--- 環境変数を削除できるかテスト
Command: printenv A
Command: setenv                     <--- setenv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
Command: setenv A= B                <--- setenv()がエラーを起こす場合
A=: Invalid argument
Command: unsetenv A=                <--- unsetenv()がエラーを起こす場合
A=: Invalid argument
Command: ^D                         <--- ^D を入力すると EOF になる
*/
```

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>                                 // exit() のため
#include <string.h>                                 // strcmp(), strchr() のため
#include <unistd.h>                                 // fork(),exec()のため
#include <sys/wait.h>                               // wait() のため
#include <ctype.h>                                  // ispace() のため
#define MAXLINE 1000                                // コマンド行の最大文字数
#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 (;;) {
    while (isspace(*p)) *p++ = '\0';                // 空白を'\0'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
    args[i++] = p;                                  // 文字列を文字列配列に記録
    while (*p!='\0' && !isspace(*p)) p++;           // 文字列の最後まで進む
  }
  args[i] = NULL;                                   // 文字列配列の終端マーク
  return *p=='\0';                                  // 解析完了なら 1 を返す
}

void cdCom(char *args[]) {                          // cd コマンドを実行する
  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
  } else if (chdir(args[1])<0) {                    //   親プロセスが chdir する
    perror(args[1]);                                //     chdirに失敗したらperror
  }
}

void setenvCom(char *args[]) {                      // setenv 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
    fprintf(stderr,"Usage: unsetenv NAME\n");       //     過不足ありなら使い方表示
  } else if (unsetenv(args[1])<0) {                 //   親プロセスがunsetenvする
    perror(args[1]);                                //     unsetenvに失敗ならperror
  }
}

void externalCom(char *args[]) {                    // 外部コマンドを実行する
  int pid, status;
  if ((pid = fork()) < 0) {                         //   新しいプロセスを作る
    perror("fork");
    exit(1);
```

```
  }
  if (pid==0) {                                    //   子プロセスなら
    execvp(args[0], args);                         //     コマンドを実行
    perror(args[0]);
    exit(1);
  } else {                                         //   親プロセスなら
    while (wait(&status) != pid)                   //     子の終了を待つ
      ;
  }
}

void execute(char *args[]) {                       // コマンドを実行する
 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);
 } else {                                          // 外部コマンドなら
   externalCom(args);
 }
}

int main() {
  char buf[MAXLINE+2];                             // コマンド行を格納する配列
  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
    if (fgets(buf,MAXLINE+2,stdin)==NULL) {        // コマンド行を入力する
      printf("\n");                                // EOF なら
      break;                                       //   正常終了する
    }
    if (strchr(buf, '\n')==NULL) {                 // '\n'がバッファにない場合は
      fprintf(stderr, "行が長すぎる\n");           //   コマンド行が長すぎたので
      return 1;                                    //   異常終了する
    }
    if (!parse(buf,args)) {                        // コマンド行を解析する
      fprintf(stderr, "引数が多すぎる\n");         //   文字列が多すぎる場合は
      continue;                                    //   ループの先頭に戻る
    }
    if (args[0]!=NULL) execute(args);              //   コマンドを実行する
  }
  return 0;
}
```