PARSING A COMMAND LINE

OVERVIEW

This assignment has two purposes. The first is to provide a gentle introduction to programming using the ANSI C language, a UNIX environment, and the compiler and debugger demonstrated in class. The second is to produce the first component of a shell program. Specifically, your program will prompt the user for input, accept that input, parse the input into tokens, and then repeat the process until the user indicates that the program should terminate.

THE PROGRAM

You are to create a program in ANSI C named *parse.c* that performs the following steps until the string *exit* is entered. When the string is entered, the program will terminate.

1. Display a prompt on *stdout*.
2. Accept a string of input from the user (Input string will terminate with a newline character).
3. Parse the input string into tokens (Use a space, a tab, or a newline as token delimiters).
4. Store the tokens in a provided structure.
5. Print the contents of the structure using the provided function.

The structure used to store the parsed input is shown below. I have included a defined value to indicate the maximum number of tokens you will find on the command line.

```
/* don't test program with more than this many tokens for input */
#define MAXARGS 32
/* structure to hold input data */
struct PARAM
{
  char *inputRedirect;           /* file name or NULL */
  char *outputRedirect;          /* file name or NULL */
  int  background;               /* either 0 (false) or 1 (true) */
  int  argumentCount;            /* same as argc in main() */
  char *argumentVector[MAXARGS]; /* array of strings */
};

/* a typedef so we don't need to use "struct PARAM" all the time */
typedef struct PARAM Param_t;
```

Notice that the first three components of the struct are special. The first two indicate that either input redirection or output redirection is desired. The third indicates whether or not the background operator has been found on the command-line. Consider this line of input shown with the prompt *$$$*.

```
$$$ one two >three <four five &
```

When the line is parsed, the first two strings are not special, so they should be placed in `argumentVector[0]` and `argumentVector[1]`. When the third string is extracted, it is identified as an output redirection because of the beginning character ('>'). The characters following the redirection indicator are the name of the file to which output should be redirected. The name of the output file ("three") should be stored in `outputRedirect`.

Similarly, the beginning character ('<') of the forth token identifies input redirection and the characters following the redirect character specifies the name of the file from which input should be read. You may or may not allow for spaces following the input and output redirection indicator. If you allow for spaces, the token following the redirection indicator must identify the filename. If you disallow spaces and no filename is specified, an error message must be generated. The name of the input file ("four") should be stored in `inputRedirect`. The fifth token is stored in `argumentVector[2]`. Backgrounding is indicated by the ampersand (&) and is recorded by setting the value of `background` to 1. If it is included in an input string, it must always appear as the last character in the string.

Once the input line is parsed and the structure elements are properly set, the results should be printed. The following function must be used to display the results in a consistent manner.

```
void printParams(Param_t * param)
{
       int i;
       printf ("InputRedirect: [%s]\n",
         (param->InputRedirect != NULL) ? param->inputRedirect:"NULL");
       printf ("OutputRedirect: [%s]\n",
         (param->outputRedirect != NULL) ? param->outputRedirect:"NULL");
       printf ("Background: [%d]\n", param->background);
       printf ("ArgumentCount: [%d]\n", param->argumentCount);
       for (i = 0; i < param->argumentCount; i++)
              printf("ArgumentVector[%2d]: [%s]\n", i, param->argumentVector[i]);
}
```

SUGGESTED SYSTEM CALLS

I suggest reading the man pages for the following system calls. These calls may be used in your program or may lead you to calls that will be used in your program.

- `fgets(3)` – input characters and strings
- `strtok(3)` – extract tokens from strings
- `printf(3)` – formatted output conversion

DELIVERABLES

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of your instructor as published on *eLearning* under the Content area.
2. You should have at a minimum the following files for this assignment:
    a. parse.c
    b. Makefile
    c. README

The README file describes purpose and usage of your program and the names of your team members; the Makefile allows the grader to compile your code correctly. Keep in mind that documentation of source code is an essential part of programming. If you do not include comments in your source code, points will be deducted.

GRADING

This project is worth 100 points total. The points will be given based on the following criteria

- 60 points for correct implementation of code (runs and all functionalities are implemented),
- 10 points for README and Makefile,
- 10 points for appropriate documentation of source code,
- 15 points for program runs.
- 5 points for correct submission format.

DUE DATE

The project is due on, Sept. 11th by 3:00 pm to the Dropbox for project 1 in *eLearning*. I will not accept submissions emailed to me or the grader. Upload ahead of time, as last minute uploads may fail.

HINTS

It would be wise to create a function to accept a string (the input string typed by the user) as input and return a filled-in structure as output. Your next program will be easier if you create a function this time. Avoid using global variables in the function in preparation for the next project.

Notice that warnings and errors are not permitted and will afford me to grad your work quickly. Tests to be completed on your program include:

1. Compile the program using *–g* and *–Wall*. You must include these options in your Makefile!
    - If errors or warnings occur, no points will be given for the assignment.
2. Run your code with input of grader's choosing.
    - Are arguments tokenized correctly?
    - Is input redirection correctly recorded?
    - Is output redirection correctly recorded?
    - Is backgrounding correctly recorded?
    - Does the program terminate when exit is entered?

COMMENTS

1. Projects will be graded on whether they correctly solve the problem, and whether they adhere to good programming practices.
2. Projects must be submitted by the time specified on the due date. Projects submitted after that time will get a grade of zero.
3. Please review UWF's academic conduct policy. Note that viewing another student's solution, whether in whole or in part is considered academic dishonesty. Also note that submitting code obtained through the Internet or other sources, whether in whole or in part, is considered academic dishonesty. All programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.