PURPOSE: The purpose of this lab is to allow you to learn a user interface aspect of a UNIX shell. Student will work with some basic system calls.

UNIX Shell

Your shell is basically an interactive loop:

- it repeatedly prints a prompt "**csc60msh >** ",
- parses the input, executes the command specified on that line of input,
- and waits for the command to finish.

We will start with 3 built-in commands:

- **exit** (exit shell)
- **pwd** (print working directory)
- **cd** (change directory)

The built-in functions are **not** executed by forking and executing an executable. Instead, the shell process executes them itself.

FILES TO COPY:

To get the file you need, first move to your class folder by typing: **cd csc60**
Type: **cp -R /home/college/bielr/files_csc60/lab9 .**
Spaces needed: (1) After the **cp** and after the **-R** ↑ Don't miss the space & dot.
(2) After the directory name at the end & before the dot.
After the file is in your account and you are still in **csc60**, you need to type: **chmod 755 lab9**
This will give permissions to the directory.
Next move into lab9 directory by typing: **cd lab9**
Type: **chmod 644 *.***
This will give permissions to the file.
Your new lab9 directory should now contain: *lab9.c, ParseCmd.c, lab9_10.h*

Please follow these steps:

- Review the source codes, compile, and execute the programs.
- I have provided you with a shell template (**lab9.c**) that you can work from. You build your program into it. **Read** the existing code closely to identify the key components and understand its execution flow.
- In the makefile, on each line that starts with ***gcc -c,*** add at the end ***-Wall*** which will generate *WarningsALL*, not errors, and can be helpful.
- The main job in lab9 is to write code for **built-in Commands**: There are four special cases where your shell should execute a command directly itself instead of running a separate process.
    - *First*, we must deal with the fact that a user might just type an Enter Key with no command (or SpaceBar EnterKey).
    - *Second*, if the user enters **"exit"** as a command, the shell should terminate.

- *Third*, if the user enters **"pwd"**, print the current working directory. This can be obtained with *getcwd()* function.
- *Fourth,* if the user enters **"cd dir"**, you should change the current directory to "dir" by using the *chdir* system call. If the user simply types **"cd"** (no dir specified), change to the user's home directory. The $HOME environment stores the desired path; use *getenv("HOME")* to obtain this.

Pseudo Code (Highlight indicates Provided Code.)

```
#include “lab9_10.h”
/*-------------------------------------------------------*/
int main (void)
{
    char *cmdLine;
    char * argv[MAXARGS];
    int argc;

    while (TRUE)
   {
        print the prompt();     /* i.e. csc60msh > , Use printf  */

        fgets(cmdline, MAXLINE, stdin);

        /* You must write the call. The function itself is provided: function ParseCmd */
        Call the function ParseCmd, sending in cmdline & argv, and returning argc

        /* Required. Code to print out the argc and the argv list, to make sure it all came in. */
        Print a line. Ex: “Argc = %i”
        loop starting at zero, thru less than argc, increment by one.
        {
            print each argv[loop counter]   [ ("Argv %i = %s \n", i, argv[i]); ]
        }

        /* Process the empty commands */
        if ( argc  compare-equal-to  zero)
        {
            /* a command was not entered, might have been just Enter or a Space&Enter */
            continue    to end of while(TRUE)-loop
        }

        // Next deal with the built-in commands
        //   Use strcmp to do the test
        //   if(strcmp(argv[0], "exit") == 0)      // Sample of complete line.
```

```
        //   After each command except for "exit', do a continue to end of while(TRUE)-loop
        if ("exit")
        {
           issue an exit call
        }
        else if ("pwd")
        {
            declare a char variable array "path" of size MAX_PATH_LENGTH to hold the path
                  (MAX_PATH_LENGTH is in a #define…. look for it in lab9_10.h)
            do a getcwd          (May skip error checking here)
            print the path
            continue
        }
        else if ("cd")
        {
             declare a char variable dir as a pointer (with an *)
             if the argc is 1
             {
                 Variable dir gets assigned the value from the call to getenv with "HOME"
             }
             else
             {
                 Variable dir gets assigned the value of argv[1]
             }
            execute a call to chdir(dir) with error checking. Message = "error changing directory"
            continue
        }  // end of the CD section
//.....IGNORE – Just leave this as is until lab10.....
//            /* Else, fork off a process */
        else
        {           // This function will be uncommented and provided to you with lab10.
            //  RunExternalCommand(int argc, char **argv);
        }           // end of the if-else-if that started with "if (exit)"
    }               // end of the while(TRUE)
}                   // end of main
```

Partnership

- Students may form a group of 2 students (maximum) to work on this lab. As usual, please always contact your instructor for questions or clarification. Your partner does not have to attend your section but must attend one of Biel's sections.

- All code files should include **both names**.

- Using **vim**, create a small name file with both of your names in it. When you start your script file, *cat* that name file so both names show up in the script file.

- **You must BOTH submit the same code and script. Each of these files will include both of your names**.

- As both of your names occur on everything, when I or another grader find the first submission, we will then give the same comments and grade to the second student.

Resource Section:

Useful Unix System Calls (Also see PowerPoint Slides named Lab9 Slides):

getenv: get the value of an environment variable
path = **getenv**("PATH");
cwd = **getenv**("PWD");
getcwd: get current working directory.
chdir: change the current working directory (use this to implement **cd**)

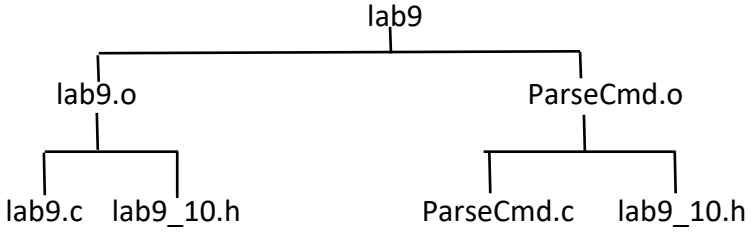
C Library functions:

```
#include <string.h>
String compare:
    int strcmp(const char *s1, const char *s2);
        This line is in #include <string.h> & does not need to be duplicated in your code.

    if(strcmp(argv[0], "exit") == 0)      // Sample of a complete line.
       strcmp(argv[0], "cd")
       strcmp(argv[0], "pwd")

Print a system error message:
    perror("Shell Program error\n");

Input of characters and strings:
    fgets(cmdline, MAXLINE, stdin);
```

Compilation & Building your program

You will need to write a makefile, including **lab9.c**, **lab9_10.h**, and **ParseCmd.c**
On the second line of each Rule, you will have a **gcc -c** command. End the line with **-Wall** which stands for WarningsAll. At this stage of programming, warnings can be very helpful.

Dependency Chart for the makefile



Hints

- Writing your shell in a simple manner is a matter of finding the relevant library routines and calling them properly. Please see the resources section above.

- Keep versions of your code. This is in case you need to go back to your older version due to an unforeseen bug/issue.

- Hint**: char *argv[ ]** does the same thing as **char **argv**

- *Our compiler does not like:* **for (int i = 0; …..)**

  You will receive the following errors:
  test_loopcounter.c:6: error: 'for' loop initial declarations are only allowed in C99 mode
  test_loopcounter.c:6: note: use option -std=c99 or -std=gnu99 to compile your code
  These errors imply that on every "gcc" line, you must add: -std=c99 OR -std=gnu99.
  This can be down on each **gcc -c** line in your make file if you want.
  C does like it on two lines if you want to avoid the -std *options:*

  int i;
  for (i = 0; ……)

→ **More on next page!**

Preparing your script file: Lab 9 with the built-in commands is worth 40 points.

You should be in your lab9 directory when you start the script.

When all is well and correct, type:

1. **script lab9.txt**
2. *If you are on a team, cat your name file here.*

At the prompt, type:

3. **touch lab9_10.h** *Required for the script, but not every time you run make.*
4. **make** *to compile the code*
5. **lab9** *to run the program, or **./lab9** or whatever you named it*

Enter in sequence:

6. the Enter Key,	*You should get the prompt back.*
7. a Space, and then the Enter Key,	*You should get the prompt back.*
8. pwd	*Check your directory* (lab9).
9. cd **..**	*Go up one level to another directory.*
10. pwd	*Check that you moved up one level to* CSC60*.*
11. cd lab9	*Go back down a level.*
12. pwd	*Check that you moved down one level to* lab9*.*
13. cd	*Go to your home directory.*
14. pwd	*Check that you got to your home directory.*

15. exit *(exit from the shell)*
16. exit *(exit from the script)*

If all worked, submit your files to Canvas.

Dates

Assign: 4/24 & 4/25
Due: 5/9
Lock: 5/16

Deliverables

Submit **four** files to Canvas. No zip files please.

1. lab9.c
2. lab9_10.h
3. makefile
4. lab9.txt (the script file)