Operating Systems and Systems Programming I Assignment

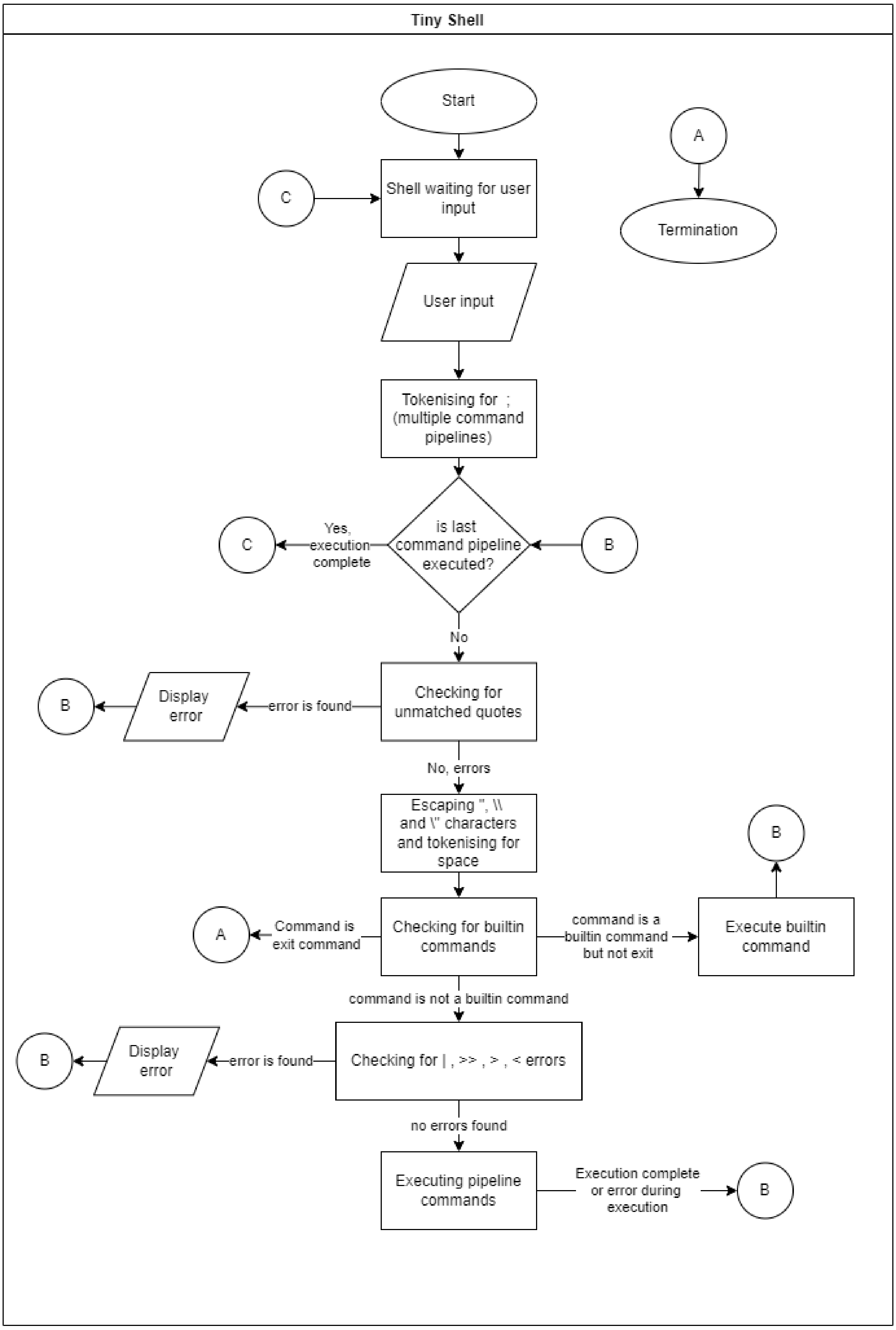
Overview:

The purpose of the assignment involved the creation of a tiny shell (tish), which would have the functionality of executing multiple builtin commands and pipeline commands. The shell included various error checking capabilities, the functionality of executing multiple command pipelines, and accepting text and meta characters which were enclosed in quotes. **All tasks of this assignment were completed, documented, and tested, and no bugs were found.**

The implementation of the shell included the following structure:

1. On start-up the shell will wait for user input
2. The user inputs the input to the terminal
3. The user input is checked for multiple command pipelines in a single input, and the shell will separate the multiple pipelines into individual pipelines (delimited by the ; metacharacter).
4. The individual pipeline is then checked for Unmatched quotes, if the individual pipeline has an odd number of quotes, then an error is presented, and the shell will resort to executing the next individual pipeline, else it will continue to execute the current individual pipeline.
5. The current individual pipeline is escaped for quotes and backslash characters, preceding by a backslash. The shell also tokenises the individual pipeline into individual arguments which are delimited by a single or multiple spaces. Text which is enclosed inside the quotes scope is taken as a single argument, such text may include meta characters and spaces.
6. The shell then checks if the individual arguments, match any builtin commands, if so, it will resort to exit said commands and continue execution of the next individual pipeline. Else it will continue executing the current individual pipeline. Note that if the exit builtin is executed, the shell will terminate.
7. Subsequently, the shell checks for errors pertaining to the |, >>, >, < metacharacters, if an error is found, the relevant error is displayed to the user, and the shell will proceed to execute the next individual pipeline. Else it will continue executing the current individual pipeline.
8. The shell will proceed to execute an external pipeline command, and will either execute successfully, or an error occurs during execution, either way, the shell will resort to execute the next individual pipeline.
9. The shell continues to iterate over the individual pipelines, until last individual pipeline is executed, then it will repeat from step 1.
10. The shell will terminate if the user inputs the exit command.

This can all be seen pictorially in the next page…

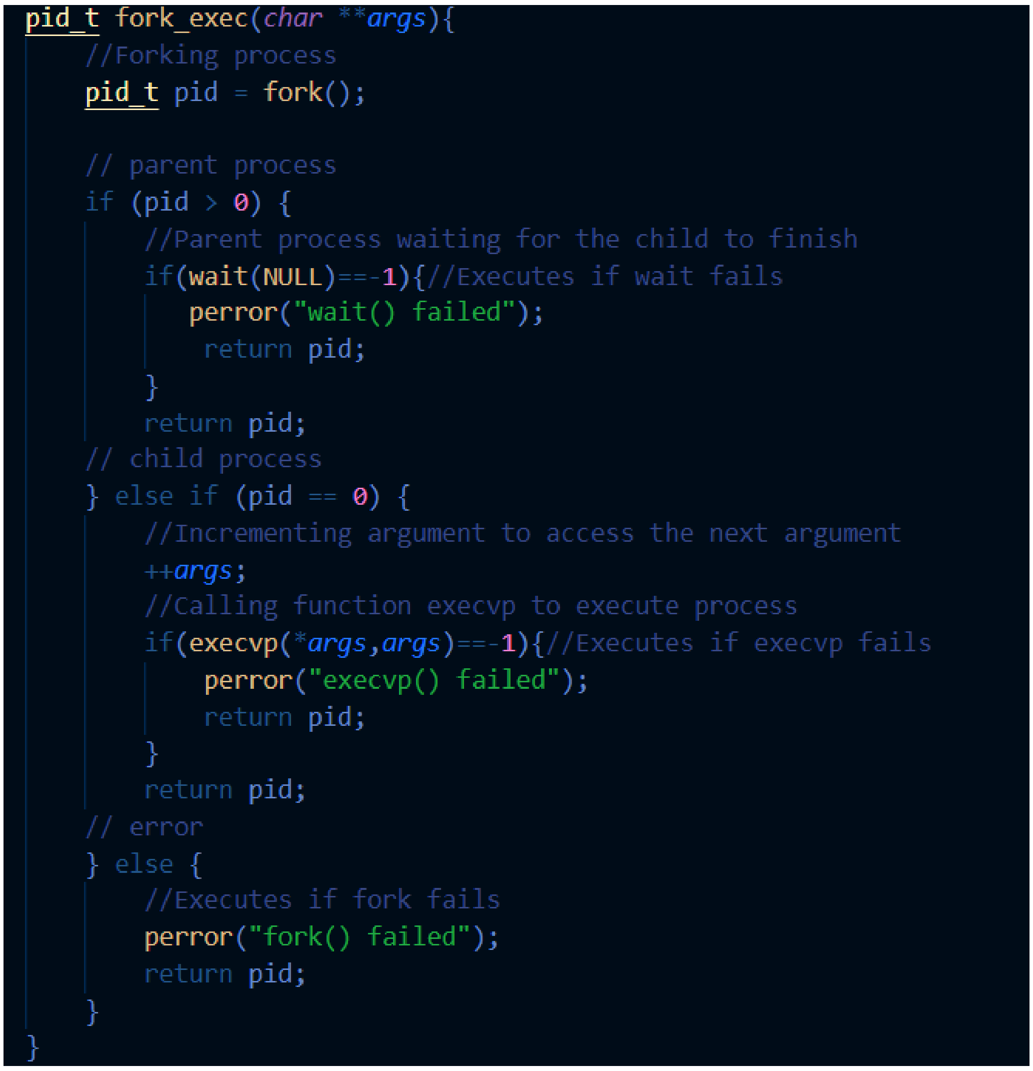

Tiny Shell
Start
A
C
Shell waiting for user input
Termination
User input
Tokenising for ; (multiple command pipelines)
C
Yes, execution complete
is last command pipeline executed?
B
No
B
Display error
error is found
Checking for unmatched quotes
No, errors
Escaping ", \\ and \" characters and tokenising for space
B
A
Command is exit command
Checking for builtin commands
command is a builtin command but not exit
Execute builtin command
command is not a builtin command
B
Display error
error is found
Checking for | , >> , > , < errors
no errors found
Executing pipeline commands
Execution complete or error during execution
B

Task 1: Process Control

Question a)

Explanation:

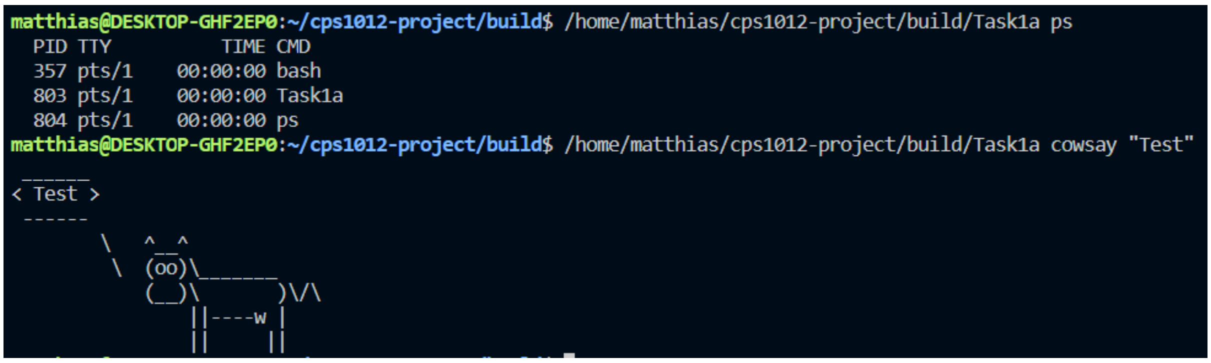
The program launches an executable by calling the fork-exec pattern. The program calls the fork_exec function which takes a char** argument holding the executable name and other arguments. The function executes the fork function to fork a child which will use the execvp function to overwrite the process image with the executable from the passed argument. The function code can be seen below:



```c
pid_t fork_exec(char **args){
    //Forking process
    pid_t pid = fork();

    // parent process
    if (pid > 0) {
        //Parent process waiting for the child to finish
        if(wait(NULL)==-1){//Executes if wait fails
           perror("wait() failed");
            return pid;
        }
        return pid;
    // child process
    } else if (pid == 0) {
        //Incrementing argument to access the next argument
        ++args;
        //Calling function execvp to execute process
        if(execvp(*args,args)==-1){//Executes if execvp fails
            perror("execvp() failed");
            return pid;
        }
        return pid;
    // error
    } else {
        //Executes if fork fails
        perror("fork() failed");
        return pid;
    }
}
```

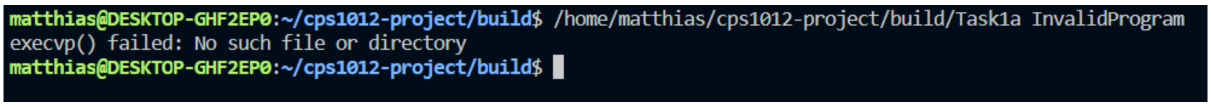
Testing:

Testing for valid input given:



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1a ps
  PID TTY          TIME CMD
  357 pts/1    00:00:00 bash
  803 pts/1    00:00:00 Task1a
  804 pts/1    00:00:00 ps
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1a cowsay "Test"
 ______
< Test >
 ------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Testing for invalid input given:



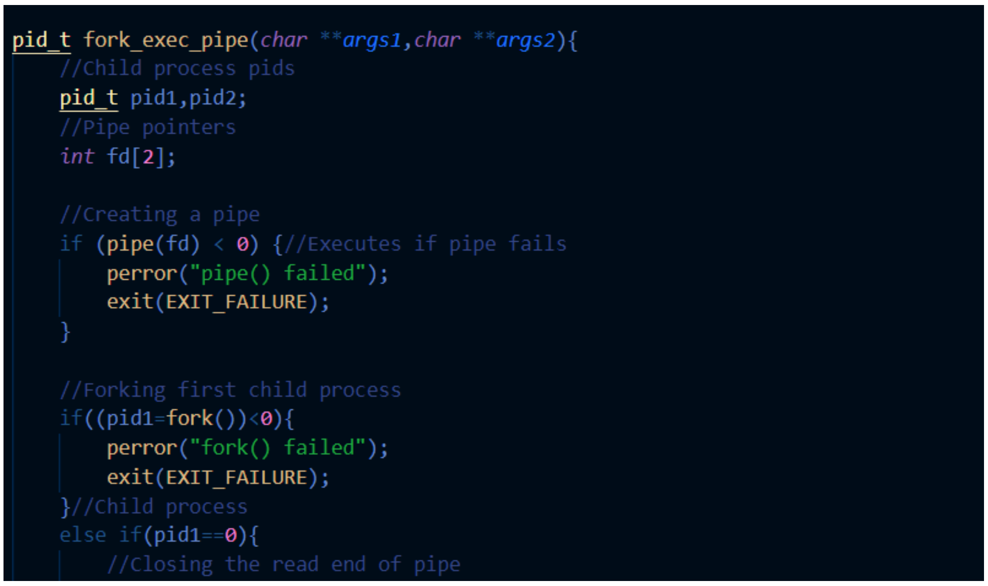
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1a InvalidProgram
execvp() failed: No such file or directory
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$
```

Question b)

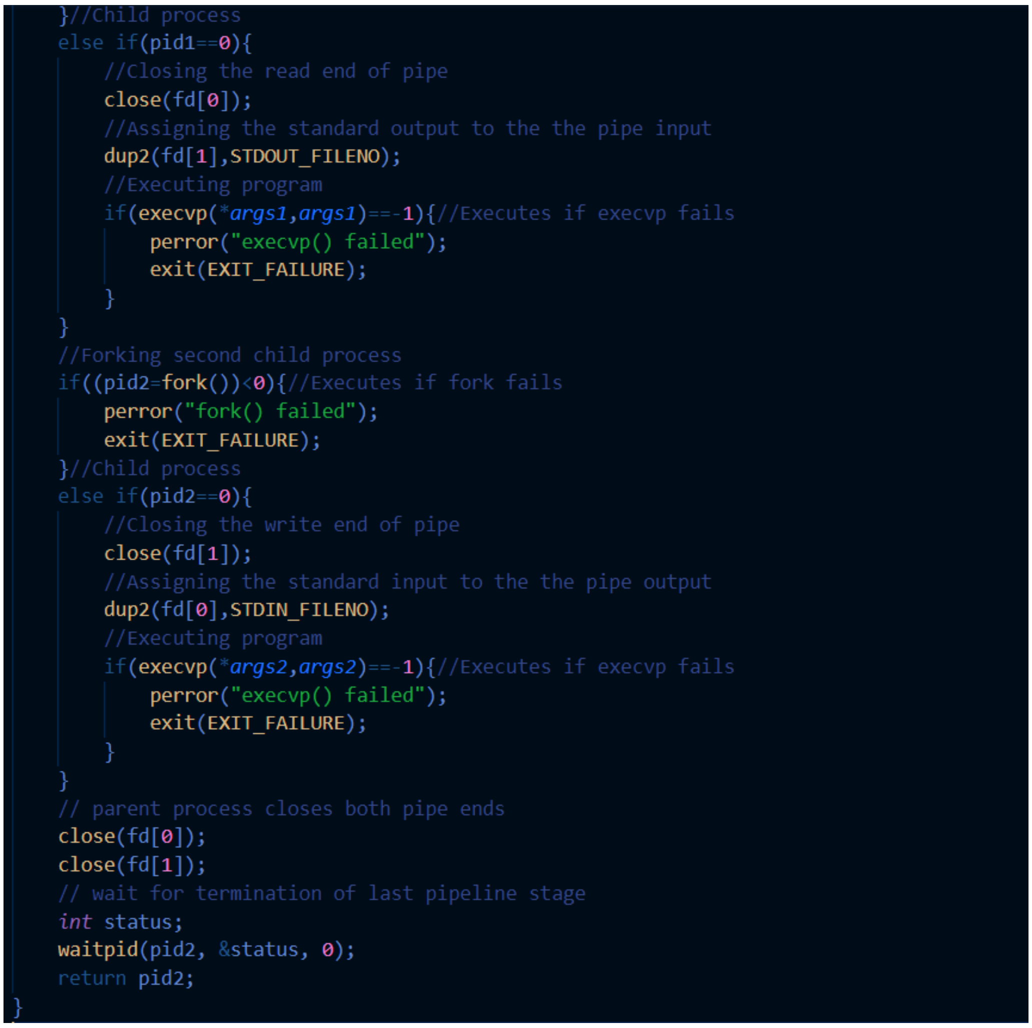
Explanation:

The program launches two executables by calling the fork-exec pattern, conjoining the processes using a pipe, whereby the output of the first process is piped into the input of the second. The program calls the fork_exec_pipe function which takes two char** arguments holding the two processes names and arguments (these are hard coded in the main method). The function first creates a pipe and proceeds to fork 2 children. The output of the first child process is piped to the input of the second child process, this is facilitated through the dup2 function. The pipe ends which are not required are closed. To avoid orphan processes the parent process waits for the termination of the second child process. Respective error messages are displayed for pipe(), fork() and execvp().

The function code was modified as can be seen below:



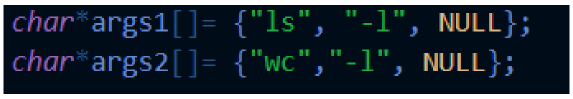
```
pid_t fork_exec_pipe(char **args1,char **args2){
    //Child process pids
    pid_t pid1,pid2;
    //Pipe pointers
    int fd[2];

    //Creating a pipe
    if (pipe(fd) < 0) {//Executes if pipe fails
        perror("pipe() failed");
        exit(EXIT_FAILURE);
    }

    //Forking first child process
    if((pid1=fork())<0){
        perror("fork() failed");
        exit(EXIT_FAILURE);
    }//Child process
    else if(pid1==0){
        //Closing the read end of pipe
```



```
    }//Child process
    else if(pid1==0){
        //Closing the read end of pipe
        close(fd[0]);
        //Assigning the standard output to the the pipe input
        dup2(fd[1],STDOUT_FILENO);
        //Executing program
        if(execvp(*args1,args1)==-1){//Executes if execvp fails
            perror("execvp() failed");
            exit(EXIT_FAILURE);
        }
    }
    //Forking second child process
    if((pid2=fork())<0){//Executes if fork fails
        perror("fork() failed");
        exit(EXIT_FAILURE);
    }//Child process
    else if(pid2==0){
        //Closing the write end of pipe
        close(fd[1]);
        //Assigning the standard input to the the pipe output
        dup2(fd[0],STDIN_FILENO);
        //Executing program
        if(execvp(*args2,args2)==-1){//Executes if execvp fails
            perror("execvp() failed");
            exit(EXIT_FAILURE);
        }
    }
    // parent process closes both pipe ends
    close(fd[0]);
    close(fd[1]);
    // wait for termination of last pipeline stage
    int status;
    waitpid(pid2, &status, 0);
    return pid2;
}
```

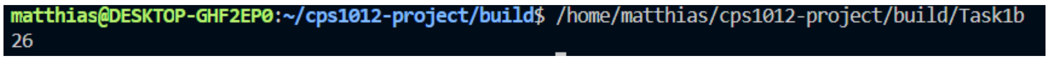
Testing:

Testing for valid input given:



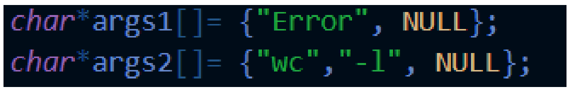
```
char*args1[]= {"ls", "-l", NULL};
char*args2[]= {"wc","-l", NULL};
```

Output received:



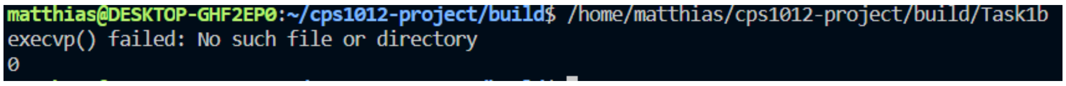
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1b
26
```

Testing for invalid input given:



```
char*args1[]= {"Error", NULL};
char*args2[]= {"wc","-l", NULL};
```

Output received:



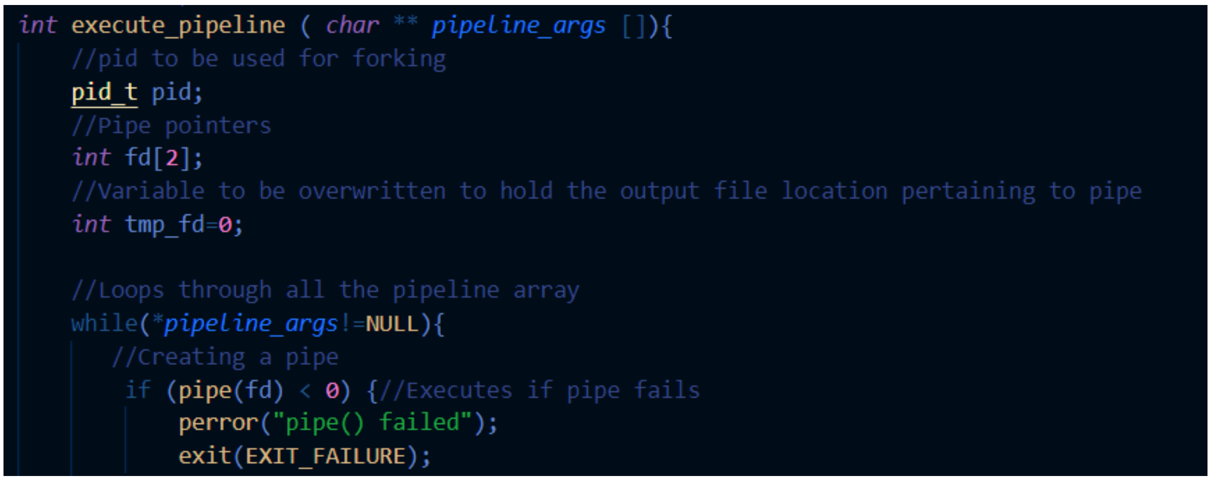
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1b
execvp() failed: No such file or directory
0
```

Question c)

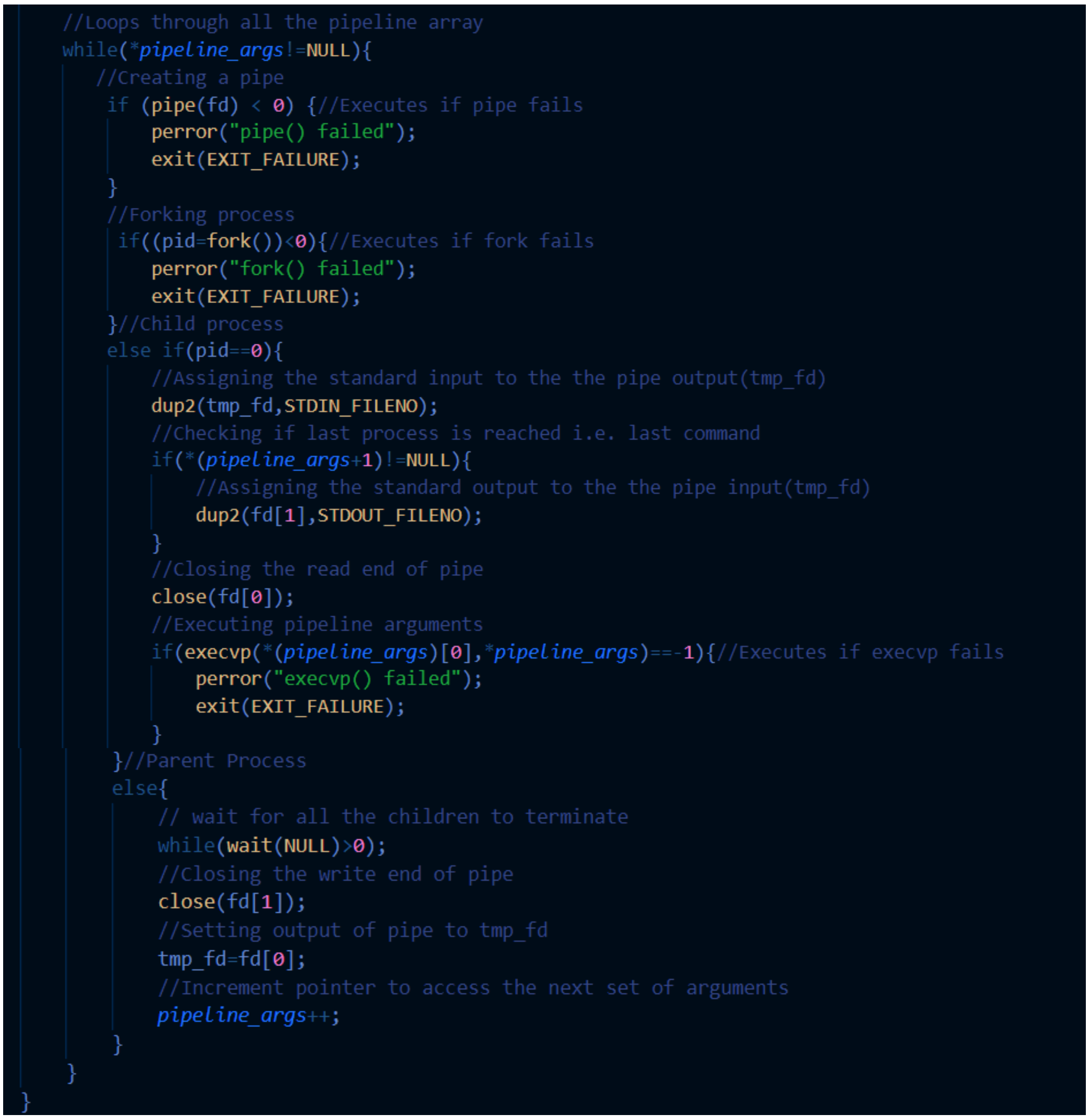
Explanation:

The program launches multiple executables by calling the fork-exec pattern, conjoining the processes using a pipe, whereby the output of the first process is piped into the next process, until the last process is reached. The program calls the execute_pipeline function which takes a char** pipeline [] argument holding all the processes names and arguments (these are hard coded in the main method). The function first creates a pipe and proceeds to forks the required number of children. Each child output will be fed to the pipe, which will be piped as an input to the next child process, a temporary file descriptor is used to facilitate this process. The pipe ends which are not required are closed. To avoid orphan processes the parent process waits for the termination of all the child processes. Respective error messages are displayed for pipe(), fork() and execvp().

The function code was modified significantly as can be seen below:



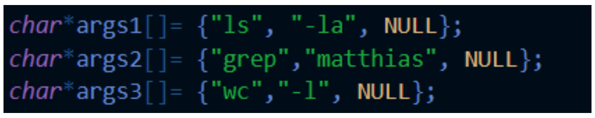
```
int execute_pipeline ( char ** pipeline_args []){
    //pid to be used for forking
    pid_t pid;
    //Pipe pointers
    int fd[2];
    //Variable to be overwritten to hold the output file location pertaining to pipe
    int tmp_fd=0;

    //Loops through all the pipeline array
    while(*pipeline_args!=NULL){
       //Creating a pipe
        if (pipe(fd) < 0) {//Executes if pipe fails
            perror("pipe() failed");
            exit(EXIT_FAILURE);
```



```
    //Loops through all the pipeline array
    while(*pipeline_args!=NULL){
       //Creating a pipe
        if (pipe(fd) < 0) {//Executes if pipe fails
            perror("pipe() failed");
            exit(EXIT_FAILURE);
        }
        //Forking process
         if((pid=fork())<0){//Executes if fork fails
            perror("fork() failed");
            exit(EXIT_FAILURE);
        }//Child process
        else if(pid==0){
            //Assigning the standard input to the the pipe output(tmp_fd)
            dup2(tmp_fd,STDIN_FILENO);
            //Checking if last process is reached i.e. last command
            if(*(pipeline_args+1)!=NULL){
                //Assigning the standard output to the the pipe input(tmp_fd)
                dup2(fd[1],STDOUT_FILENO);
            }
            //Closing the read end of pipe
            close(fd[0]);
            //Executing pipeline arguments
            if(execvp(*(pipeline_args)[0],*pipeline_args)==-1){//Executes if execvp fails
                perror("execvp() failed");
                exit(EXIT_FAILURE);
            }
        }//Parent Process
        else{
            // wait for all the children to terminate
            while(wait(NULL)>0);
            //Closing the write end of pipe
            close(fd[1]);
            //Setting output of pipe to tmp_fd
            tmp_fd=fd[0];
            //Increment pointer to access the next set of arguments
            pipeline_args++;
        }
    }
}
```

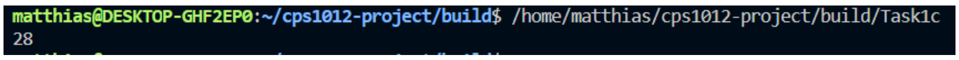
Testing:

Testing for valid input given:



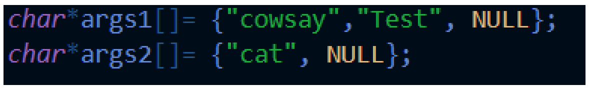
```
char*args1[]= {"ls", "-la", NULL};
char*args2[]= {"grep","matthias", NULL};
char*args3[]= {"wc","-l", NULL};
```

Output received:



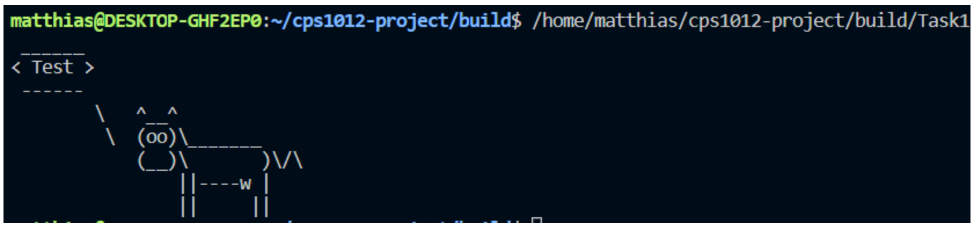
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1c
28
```

Testing with different number of arguments:



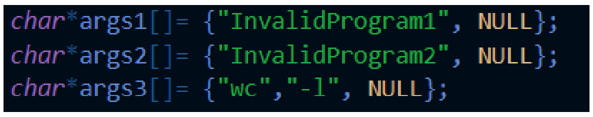
```
char*args1[]= {"cowsay","Test", NULL};
char*args2[]= {"cat", NULL};
```

Output in terminal:



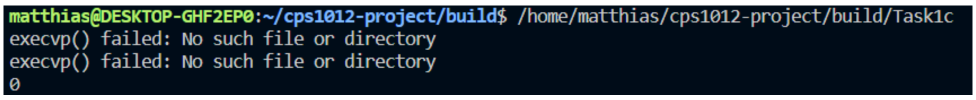
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1
 ______
< Test >
 ------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Testing for invalid input given:



```
char*args1[]= {"InvalidProgram1", NULL};
char*args2[]= {"InvalidProgram2", NULL};
char*args3[]= {"wc","-l", NULL};
```

Output received:

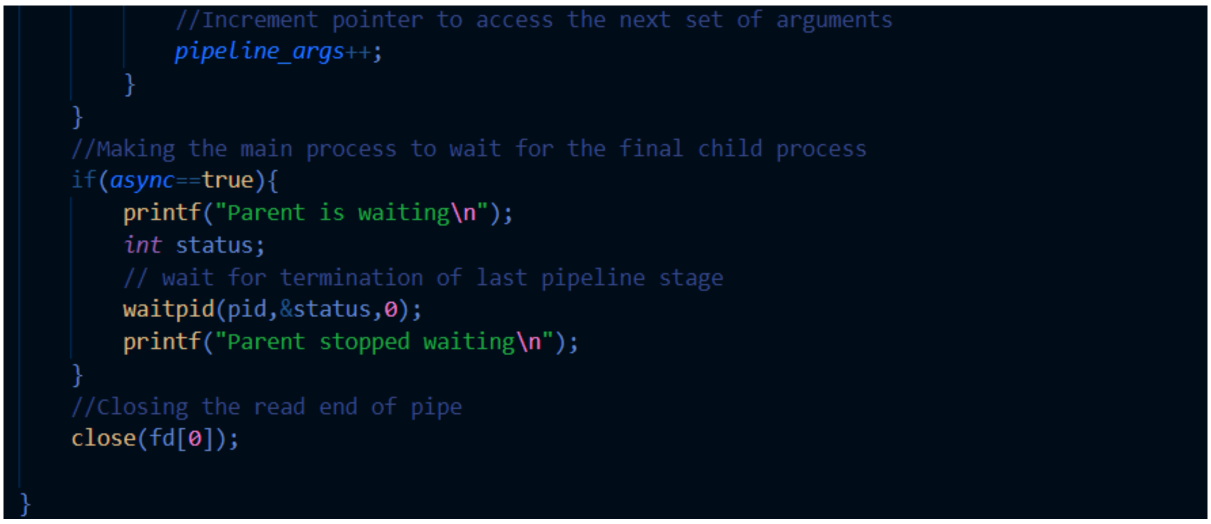


```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1c
execvp() failed: No such file or directory
execvp() failed: No such file or directory
0
```

Question d)

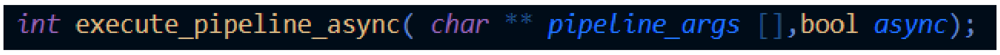
Explanation:

The program modifies Task 1c) with the addition of the async flag to execute_pipeline function, which is taken as an extra argument. This Boolean argument serves as a flag which forces the parent to wait for all children to be terminated if the flag is set to true. The wait(NULL) function was removed, and was replaced with the following code outside the while loop:



```
            //Increment pointer to access the next set of arguments
            pipeline_args++;
        }
    }
    //Making the main process to wait for the final child process
    if(async==true){
        printf("Parent is waiting\n");
        int status;
        // wait for termination of last pipeline stage
        waitpid(pid,&status,0);
        printf("Parent stopped waiting\n");
    }
    //Closing the read end of pipe
    close(fd[0]);

}
```

Also note that the function name was modified to be execute_pipeline_async as can be seen below:

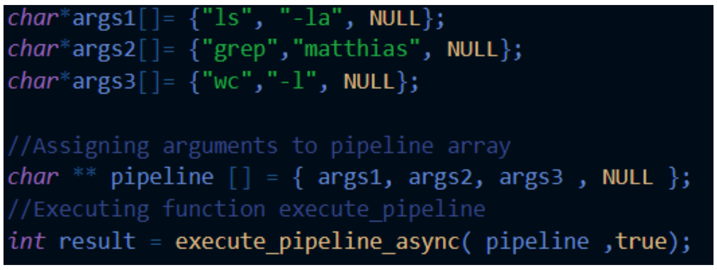


```
int execute_pipeline_async( char ** pipeline_args [],bool async);
```

Testing:

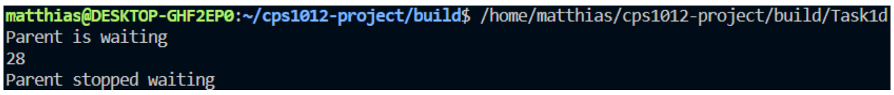
Testing with valid input:

Testing with async flag set to true:



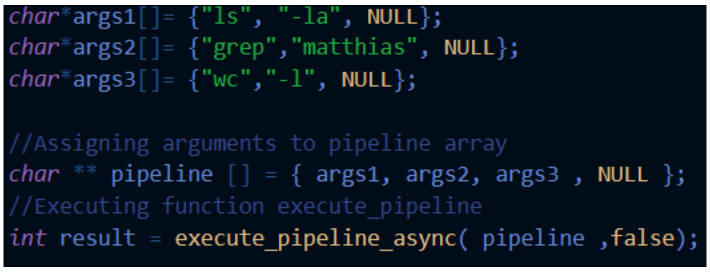
```
char*args1[]= {"ls", "-la", NULL};
char*args2[]= {"grep","matthias", NULL};
char*args3[]= {"wc","-l", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1, args2, args3 , NULL };
//Executing function execute_pipeline
int result = execute_pipeline_async( pipeline ,true);
```

Presents the following output:



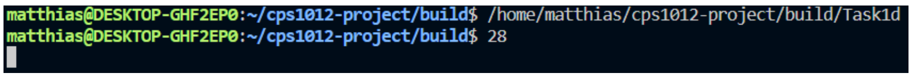
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1d
Parent is waiting
28
Parent stopped waiting
```

Testing with async flag set to false:



```
char*args1[]= {"ls", "-la", NULL};
char*args2[]= {"grep","matthias", NULL};
char*args3[]= {"wc","-l", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1, args2, args3 , NULL };
//Executing function execute_pipeline
int result = execute_pipeline_async( pipeline ,false);
```

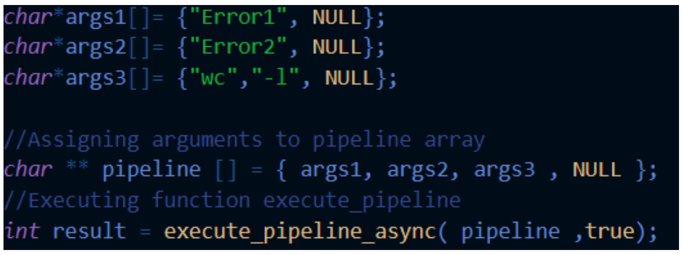
Presents the following output:



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1d
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ 28
```

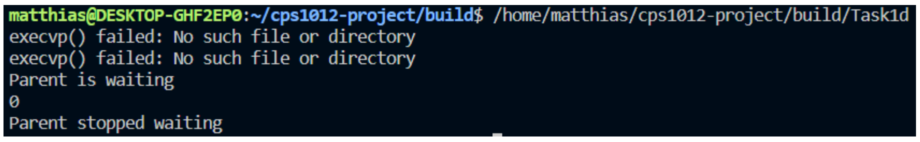
Testing with invalid input:

Testing with async flag set to true:



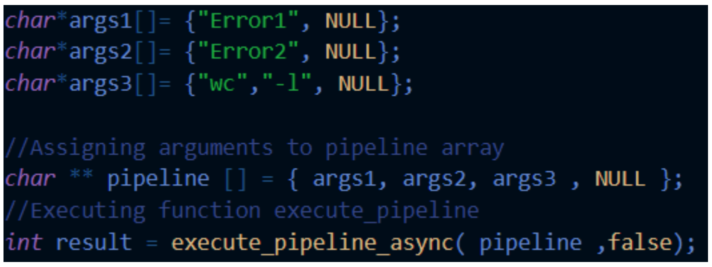
```
char*args1[]= {"Error1", NULL};
char*args2[]= {"Error2", NULL};
char*args3[]= {"wc","-l", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1, args2, args3 , NULL };
//Executing function execute_pipeline
int result = execute_pipeline_async( pipeline ,true);
```

Presents the following output:



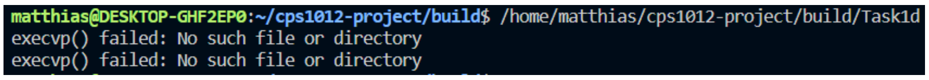
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1d
execvp() failed: No such file or directory
execvp() failed: No such file or directory
Parent is waiting
0
Parent stopped waiting
```

Testing with async flag set to false:



```
char*args1[]= {"Error1", NULL};
char*args2[]= {"Error2", NULL};
char*args3[]= {"wc","-l", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1, args2, args3 , NULL };
//Executing function execute_pipeline
int result = execute_pipeline_async( pipeline ,false);
```

Presents the following output:



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1d
execvp() failed: No such file or directory
execvp() failed: No such file or directory
```

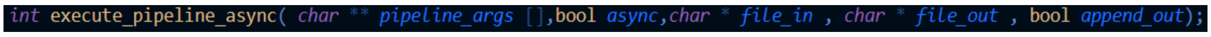
Question e)

Explanation:

The program modifies Task 1d) with the addition of the file_in, file_out and append_out variables to execute_pipeline_async function, which are taken as extra arguments. The file_in holds the file name which will be used for input redirection for the first stage of the pipeline. The file_out holds the file name which will be used for output redirection for the last stage of the pipeline. The append_out flag is used to determine whether to truncate to the output file or append to it. If the append_out flag is set to true, it will attempt to append to file, else it will truncate. The program will perform output or input redirection if the file_out and file_in are not null respectively.

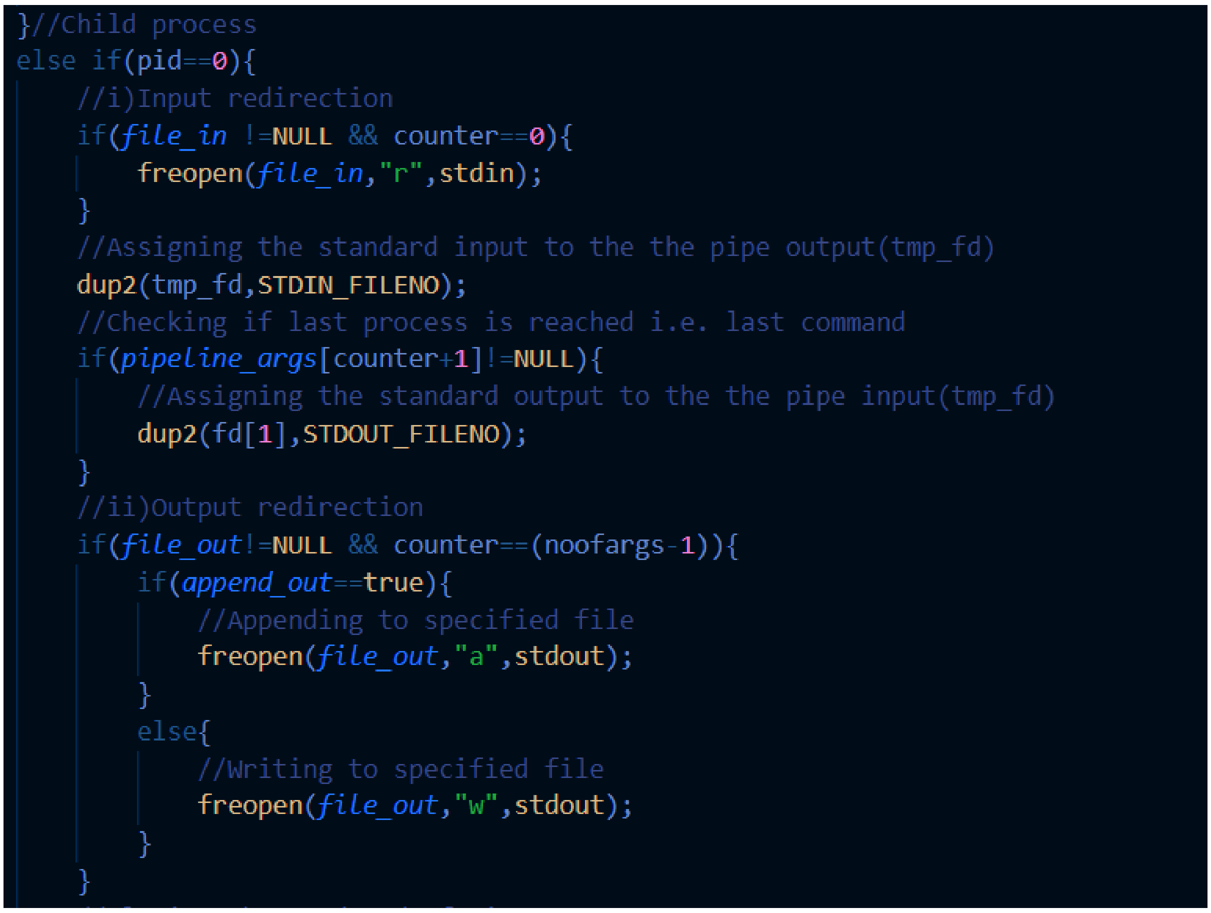
The modifications can be seen below:

The function with the new parameters:



```c
int execute_pipeline_async( char ** pipeline_args [],bool async,char * file_in , char * file_out , bool append_out);
```

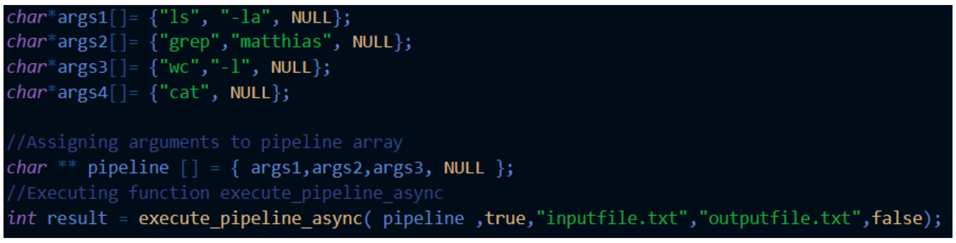
Modification to the execute_pipeline_async method:



```c
}//Child process
else if(pid==0){
    //i)Input redirection
    if(file_in !=NULL && counter==0){
        freopen(file_in,"r",stdin);
    }
    //Assigning the standard input to the the pipe output(tmp_fd)
    dup2(tmp_fd,STDIN_FILENO);
    //Checking if last process is reached i.e. last command
    if(pipeline_args[counter+1]!=NULL){
        //Assigning the standard output to the the pipe input(tmp_fd)
        dup2(fd[1],STDOUT_FILENO);
    }
    //ii)Output redirection
    if(file_out!=NULL && counter==(noofargs-1)){
        if(append_out==true){
            //Appending to specified file
            freopen(file_out,"a",stdout);
        }
        else{
            //Writing to specified file
            freopen(file_out,"w",stdout);
        }
    }
```

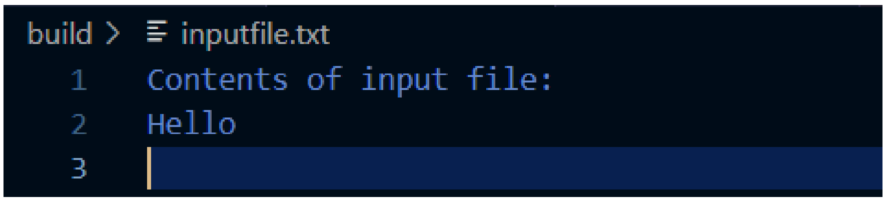
Testing:

Testing with file_in as inputfile.txt, file_out as outputfile.txt and append_out as false:



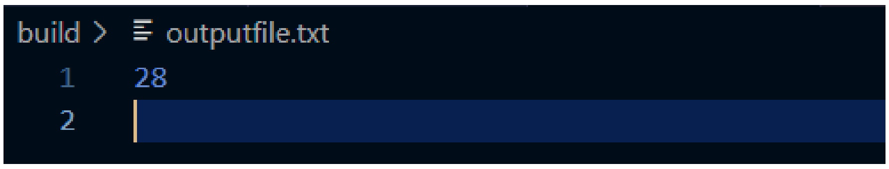
```c
char*args1[]= {"ls", "-la", NULL};
char*args2[]= {"grep","matthias", NULL};
char*args3[]= {"wc","-l", NULL};
char*args4[]= {"cat", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1,args2,args3, NULL };
//Executing function execute_pipeline_async
int result = execute_pipeline_async( pipeline ,true,"inputfile.txt","outputfile.txt",false);
```

Input file:



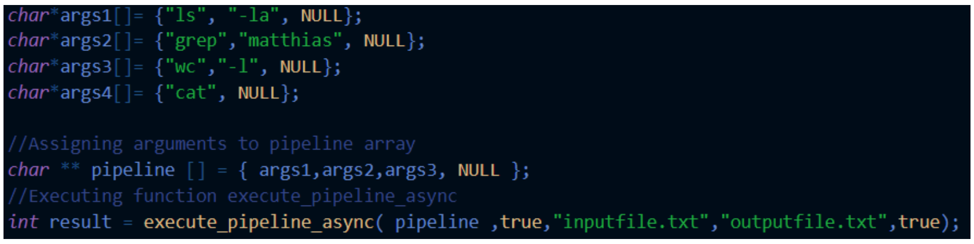
```
build > inputfile.txt
Contents of input file:
Hello
```

Output file:



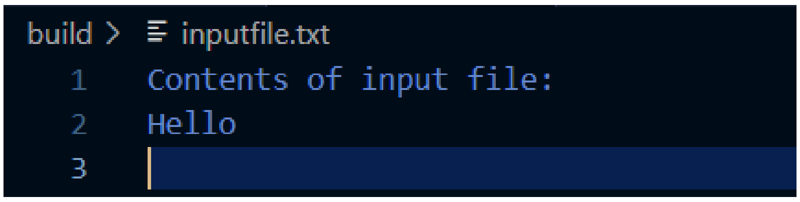
```
build > outputfile.txt
28
```

Testing with file_in as inputfile.txt, file_out as outputfile.txt and append_out as true:



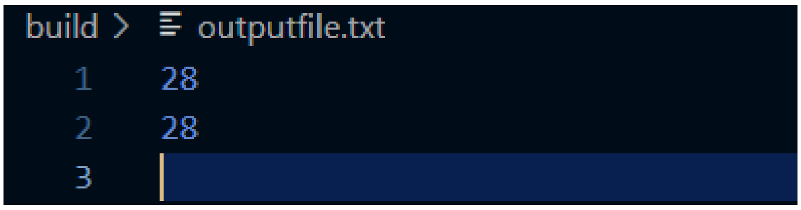
```c
char*args1[]= {"ls", "-la", NULL};
char*args2[]= {"grep","matthias", NULL};
char*args3[]= {"wc","-l", NULL};
char*args4[]= {"cat", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1,args2,args3, NULL };
//Executing function execute_pipeline_async
int result = execute_pipeline_async( pipeline ,true,"inputfile.txt","outputfile.txt",true);
```

Input file:



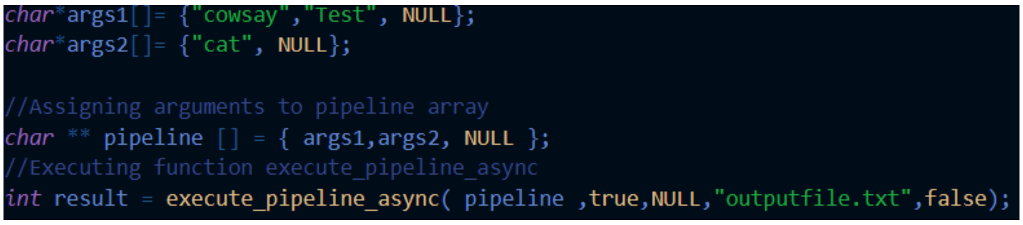
```
build > inputfile.txt
Contents of input file:
Hello
```

Output file:



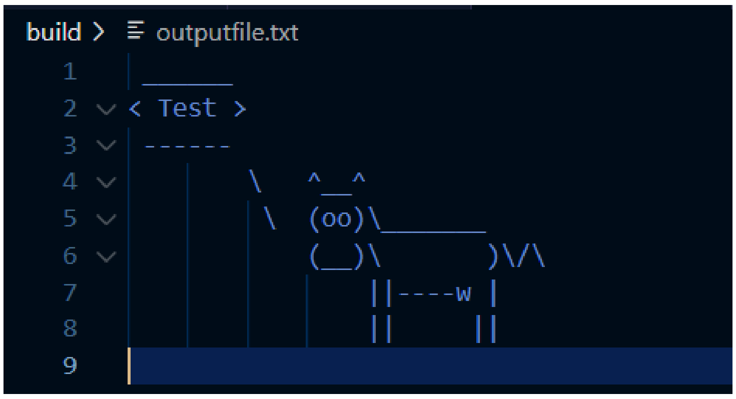
```
build > outputfile.txt
28
28
```

Testing with file_in as NULL, file_out as outputfile.txt and append_out as false:



```c
char*args1[]= {"cowsay","Test", NULL};
char*args2[]= {"cat", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1,args2, NULL };
//Executing function execute_pipeline_async
int result = execute_pipeline_async( pipeline ,true,NULL,"outputfile.txt",false);
```

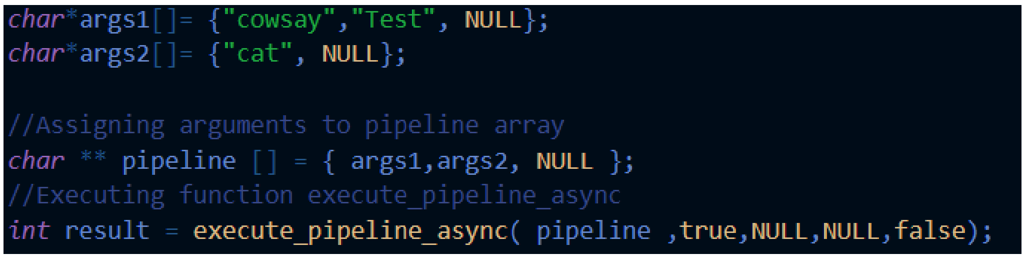
Output file:



build > outputfile.txt

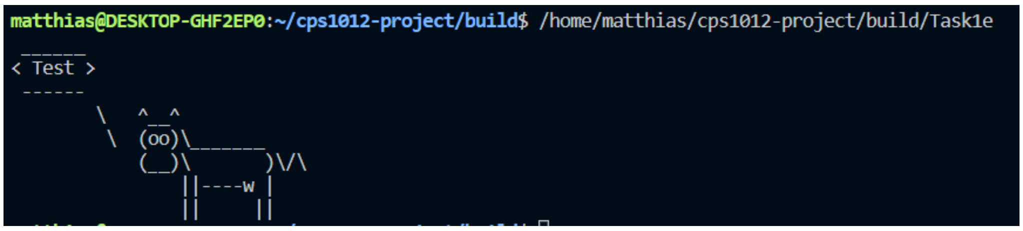
```
 ______
< Test >
 ------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

```

Testing with file_in as NULL, file_out as NULL and append_out as false:



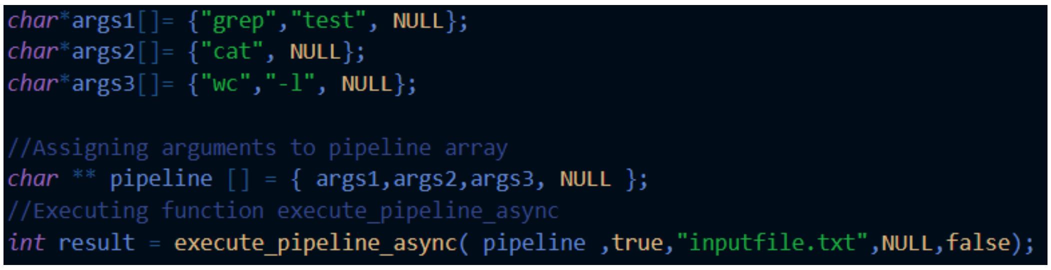
```c
char*args1[]= {"cowsay","Test", NULL};
char*args2[]= {"cat", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1,args2, NULL };
//Executing function execute_pipeline_async
int result = execute_pipeline_async( pipeline ,true,NULL,NULL,false);
```

Output in terminal:



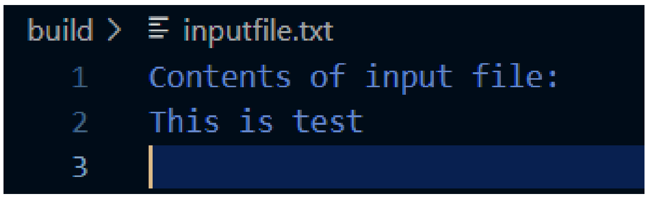
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1e
 ______
< Test >
 ------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Testing with file_in as inputfile.txt, file_out as NULL and append_out as false:



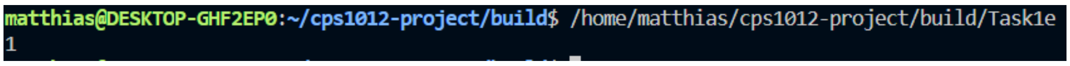
```
char*args1[]= {"grep","test", NULL};
char*args2[]= {"cat", NULL};
char*args3[]= {"wc","-l", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1,args2,args3, NULL };
//Executing function execute_pipeline_async
int result = execute_pipeline_async( pipeline ,true,"inputfile.txt",NULL,false);
```

Input file:



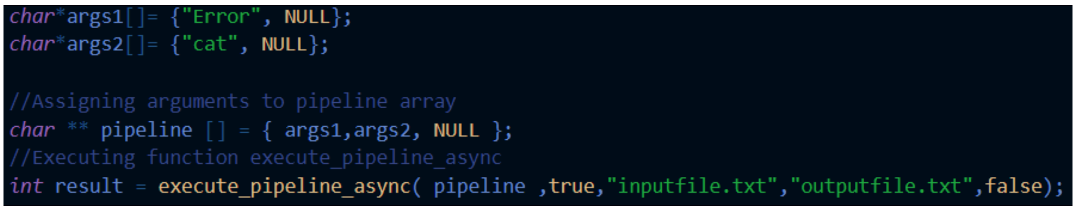
```
build > inputfile.txt
Contents of input file:
This is test

```

Output in terminal:



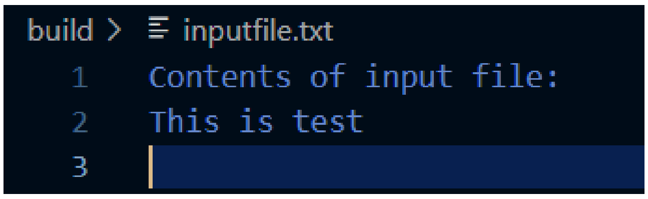
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1e
1
```

Testing for invalid input with file_in as inputfile.txt, file_out as NULL and append_out as false:



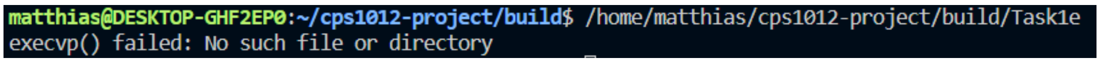
```
char*args1[]= {"Error", NULL};
char*args2[]= {"cat", NULL};

//Assigning arguments to pipeline array
char ** pipeline [] = { args1,args2, NULL };
//Executing function execute_pipeline_async
int result = execute_pipeline_async( pipeline ,true,"inputfile.txt","outputfile.txt",false);
```

Input file:



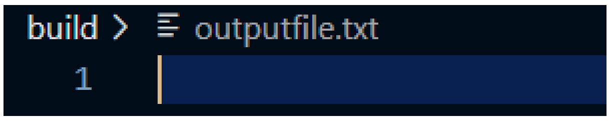
```
build > inputfile.txt
Contents of input file:
This is test

```

Output in terminal:



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task1e
execvp() failed: No such file or directory
```

Output file:



```
build > outputfile.txt

```

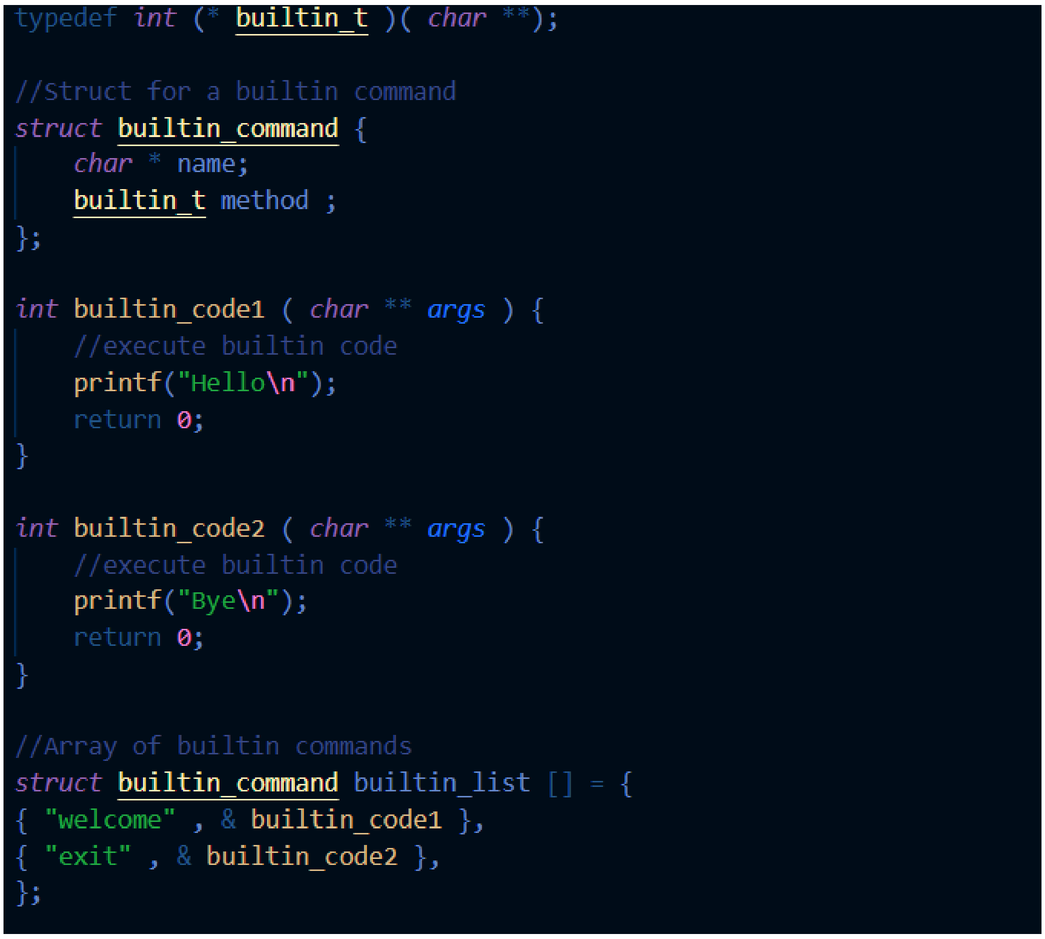
Task 2: Shell Builtin Commands

Question a)

Explanation:

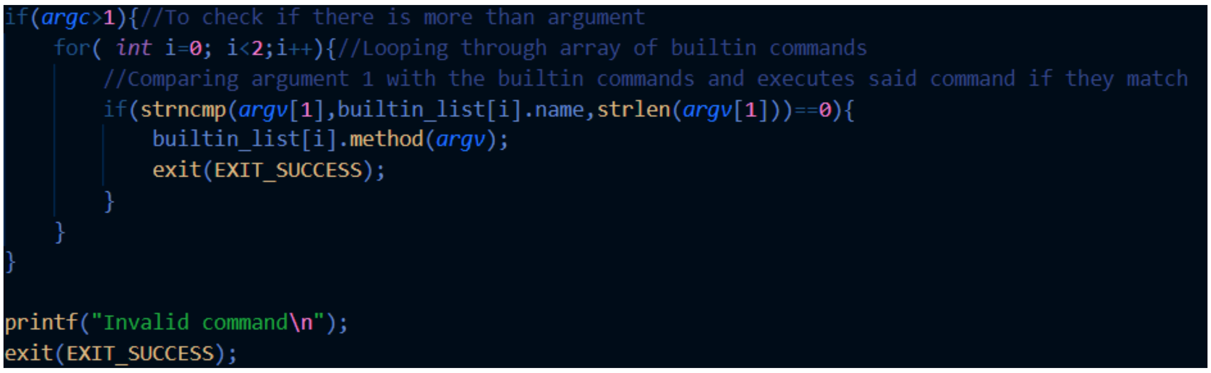
The program utilises the following declaration to map a builtin name to a corresponding function, this is enabled using structs. The struct builtin_command contains the function name and a method, the latter pertaining to the function to be performed. Moreover, builtin_code1 and builtin_code2 are both functions, which perform specific tasks. Builtin_list is an array of builtin commands.

This can all be seen below:



```
typedef int (* builtin_t )( char **);

//Struct for a builtin command
struct builtin_command {
    char * name;
    builtin_t method ;
};

int builtin_code1 ( char ** args ) {
    //execute builtin code
    printf("Hello\n");
    return 0;
}

int builtin_code2 ( char ** args ) {
    //execute builtin code
    printf("Bye\n");
    return 0;
}

//Array of builtin commands
struct builtin_command builtin_list [] = {
{ "welcome" , & builtin_code1 },
{ "exit" , & builtin_code2 },
};
```

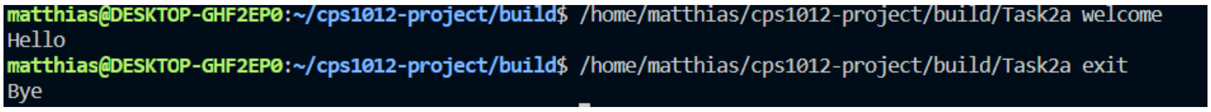
In the code segment seen below, the program checks if the input matches with any builtin command names, if so it will execute the respective builtin function:



```
if(argc>1){//To check if there is more than argument
    for( int i=0; i<2;i++){//Looping through array of builtin commands
        //Comparing argument 1 with the builtin commands and executes said command if they match
        if(strncmp(argv[1],builtin_list[i].name,strlen(argv[1]))==0){
            builtin_list[i].method(argv);
            exit(EXIT_SUCCESS);
        }
    }
}

printf("Invalid command\n");
exit(EXIT_SUCCESS);
```

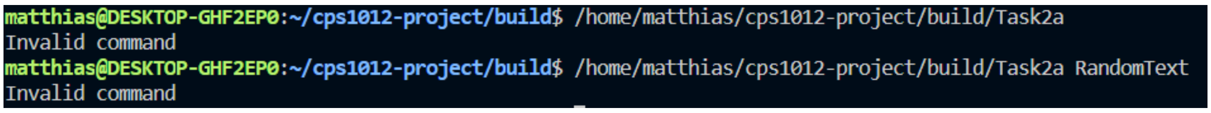
<u>Testing:</u>

<u>Testing for valid input:</u>



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2a welcome
Hello
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2a exit
Bye
```

<u>Testing for invalid input:</u>



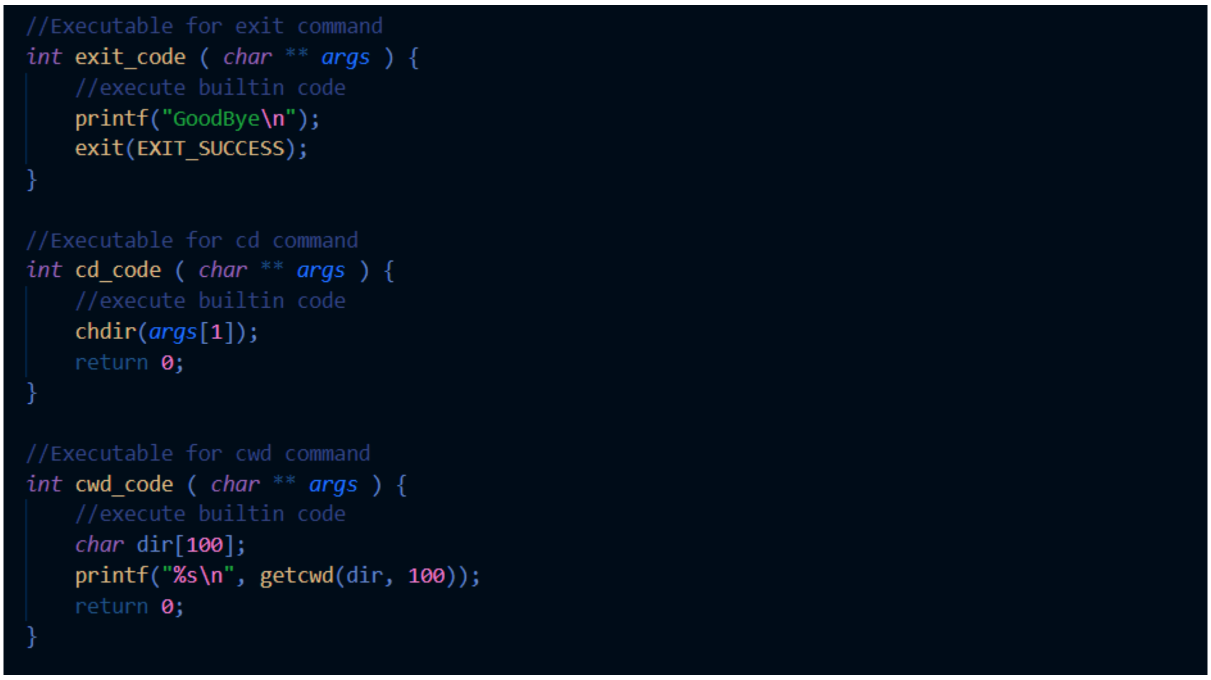
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2a
Invalid command
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2a RandomText
Invalid command
```

Question b)

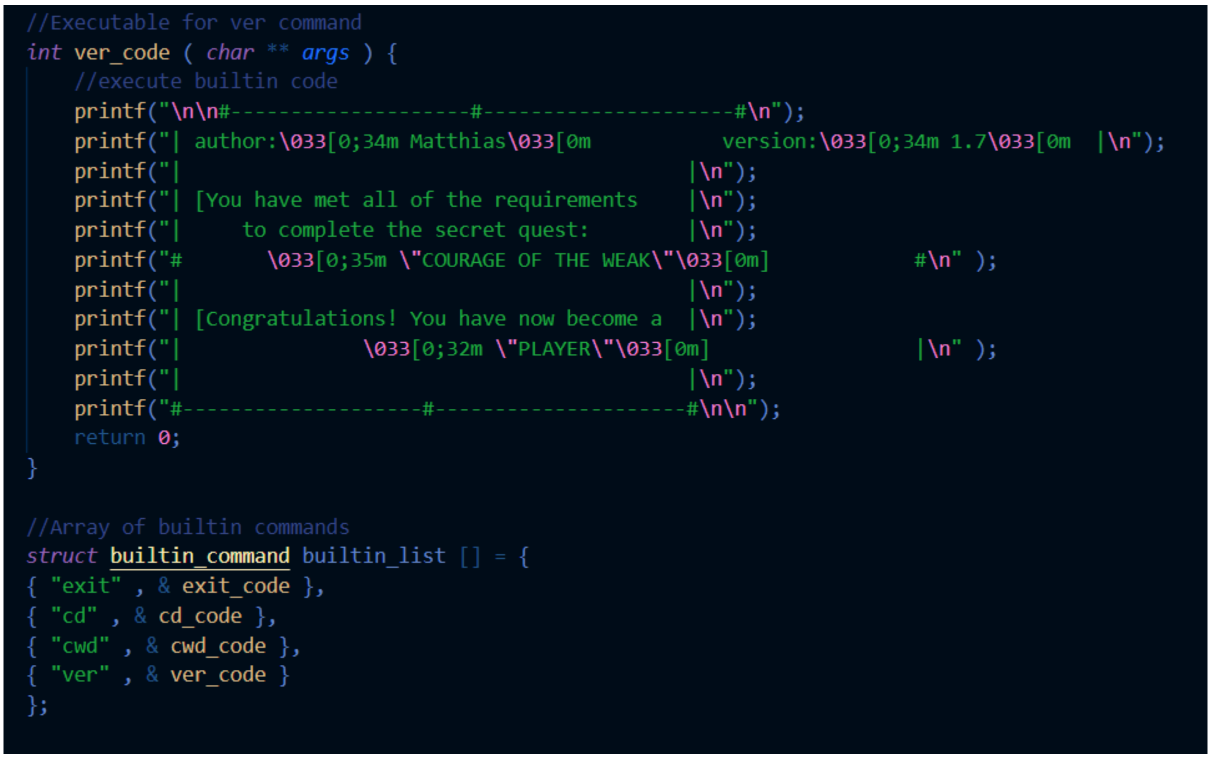
Explanation:

The program modifies Task 2a), by adding the exit, cd, cwd, ver commands and their functionality respectively. The contents of builtin_code1, builtin_code2 and builtin_list is modified.

This can all be seen below:



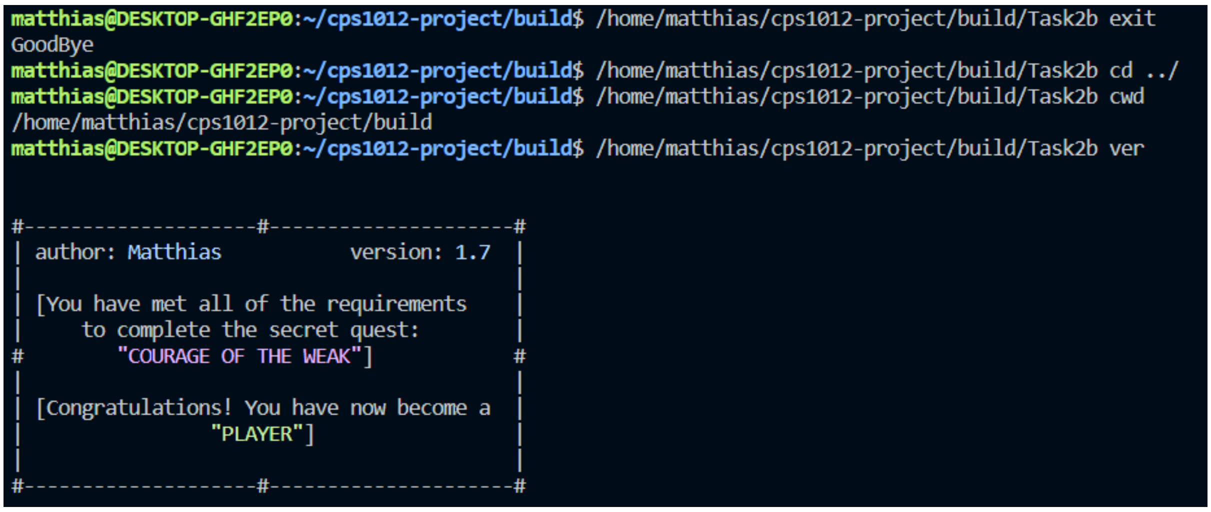
```c
//Executable for exit command
int exit_code ( char ** args ) {
    //execute builtin code
    printf("GoodBye\n");
    exit(EXIT_SUCCESS);
}

//Executable for cd command
int cd_code ( char ** args ) {
    //execute builtin code
    chdir(args[1]);
    return 0;
}

//Executable for cwd command
int cwd_code ( char ** args ) {
    //execute builtin code
    char dir[100];
    printf("%s\n", getcwd(dir, 100));
    return 0;
}
```



```c
//Executable for ver command
int ver_code ( char ** args ) {
    //execute builtin code
    printf("\n\n#--------------------#---------------------#\n");
    printf("| author:\033[0;34m Matthias\033[0m           version:\033[0;34m 1.7\033[0m  |\n");
    printf("|                                         |\n");
    printf("| [You have met all of the requirements    |\n");
    printf("|     to complete the secret quest:        |\n");
    printf("#      \033[0;35m \"COURAGE OF THE WEAK\"\033[0m]            #\n" );
    printf("|                                         |\n");
    printf("| [Congratulations! You have now become a  |\n");
    printf("|              \033[0;32m \"PLAYER\"\033[0m]                 |\n" );
    printf("|                                         |\n");
    printf("#--------------------#---------------------#\n\n");
    return 0;
}

//Array of builtin commands
struct builtin_command builtin_list [] = {
{ "exit" , & exit_code },
{ "cd" , & cd_code },
{ "cwd" , & cwd_code },
{ "ver" , & ver_code }
};
```

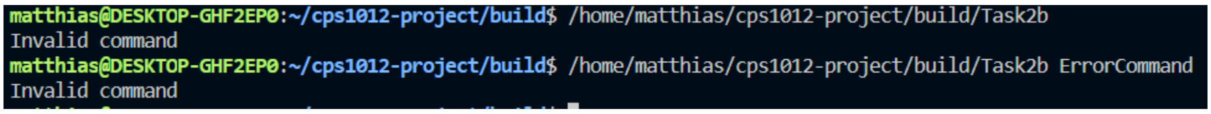
Testing:

Testing for valid input:



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2b exit
GoodBye
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2b cd ../
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2b cwd
/home/matthias/cps1012-project/build
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2b ver


#--------------------#---------------------#
| author: Matthias           version: 1.7  |
|                                          |
| [You have met all of the requirements    |
|     to complete the secret quest:        |
#        "COURAGE OF THE WEAK"]            #
|                                          |
| [Congratulations! You have now become a  |
|                "PLAYER"]                 |
|                                          |
#--------------------#---------------------#
```

Testing for invalid input:



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2b
Invalid command
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task2b ErrorCommand
Invalid command
```

Task 3: Terminal Input and Output

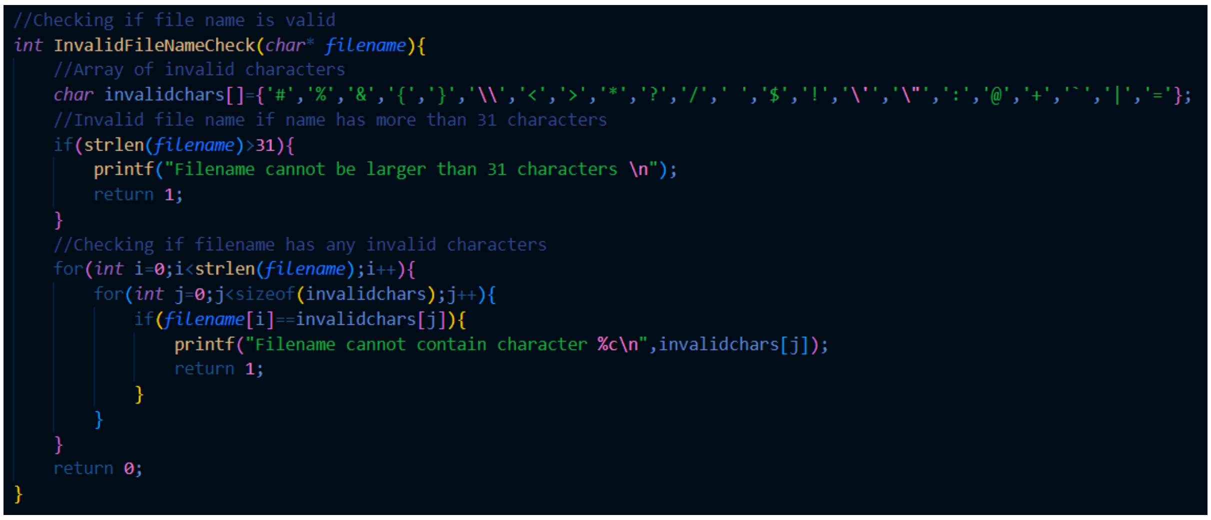
Question a)

Explanation:

The program is required to read a single line of text from standard input, tokenising the input string using a space as a delimiter. This was achieved using **linenoise** for reading a single line of text from standard input and tokenizing said text through the strtok() function.

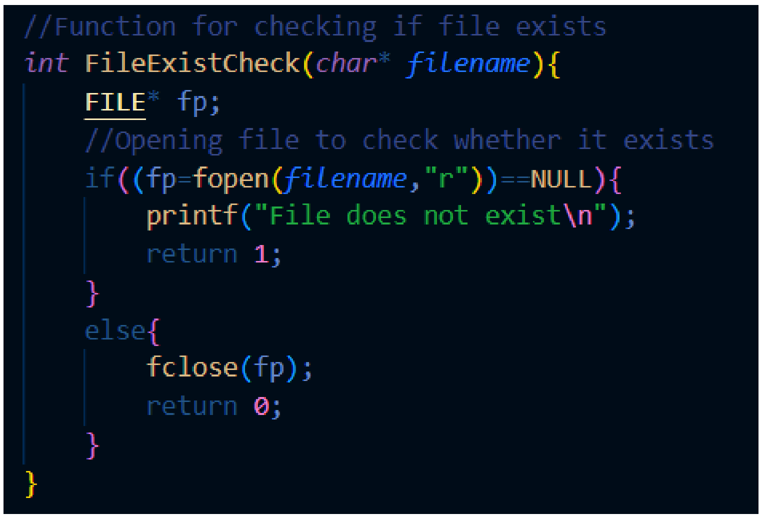
Moreover, error checking was also established for tokens which contain metacharacters.

Two functions were used to facilitate this operation, the function which can be seen below, takes a filename as input and checks if the filename contains invalid characters, and that the file name is smaller than 32 characters. The function loops through all the characters in the filename and checks each character separately, if said characters match with an invalid character then 1 is returned, else 0 is returned. If the name is larger than 31 characters, then 1 is returned.



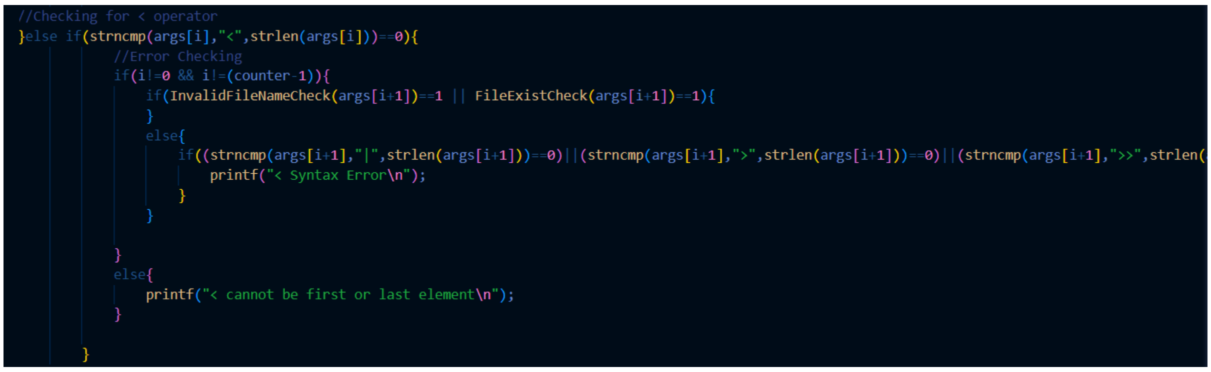
```c
//Checking if file name is valid
int InvalidFileNameCheck(char* filename){
    //Array of invalid characters
    char invalidchars[]={'#','%','&','{','}','\\','<','>','*','?','/',' ','$','!','\'','\"',':','@','+','`','|','='};
    //Invalid file name if name has more than 31 characters
    if(strlen(filename)>31){
        printf("Filename cannot be larger than 31 characters \n");
        return 1;
    }
    //Checking if filename has any invalid characters
    for(int i=0;i<strlen(filename);i++){
        for(int j=0;j<sizeof(invalidchars);j++){
            if(filename[i]==invalidchars[j]){
                printf("Filename cannot contain character %c\n",invalidchars[j]);
                return 1;
            }
        }
    }
    return 0;
}
```

The function seen here, checks whether the file exists; by utilising the function fopen(). If the result of fopen() is NULL, thus implies that file does not exist, and function would return 1. Else, it would imply that the file exists, and the function would return 0.



```c
//Function for checking if file exists
int FileExistCheck(char* filename){
    FILE* fp;
    //Opening file to check whether it exists
    if((fp=fopen(filename,"r"))==NULL){
        printf("File does not exist\n");
        return 1;
    }
    else{
        fclose(fp);
        return 0;
    }
}
```

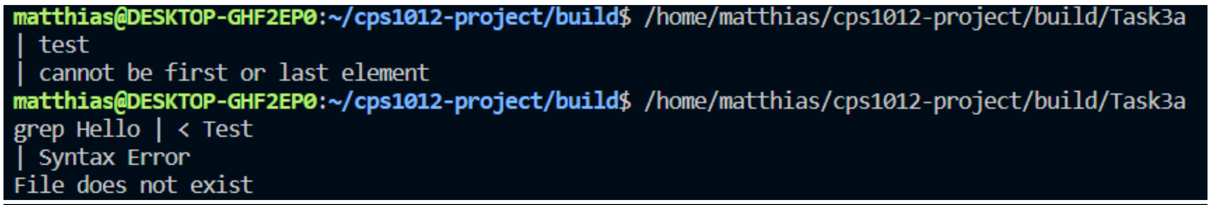
The main method includes if statements that first check whether the token match the metacharacter, if so will proceed to check for errors. Error checking includes checking that the metacharacter is not the first or last token (first token only for pipe operator), as well as checking that the subsequent token is not also a metacharacter. Furthermore, the 2 testing functions listed in the previous page are used for subsequent error checking. The if statement for checking the errors of a metacharacter:



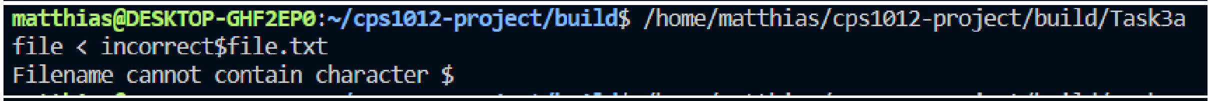
```
//Checking for < operator
}else if(strncmp(args[i],"<",strlen(args[i]))==0){
        //Error Checking
        if(i!=0 && i!=(counter-1)){
            if(InvalidFileNameCheck(args[i+1])==1 || FileExistCheck(args[i+1])==1){
            }
            else{
                if((strncmp(args[i+1],"|",strlen(args[i+1]))==0)||(strncmp(args[i+1],">",strlen(args[i+1]))==0)||(strncmp(args[i+1],">>",strlen(
                    printf("< Syntax Error\n");
                }
            }

        }
        else{
            printf("< cannot be first or last element\n");
        }

    }
```

Testing:

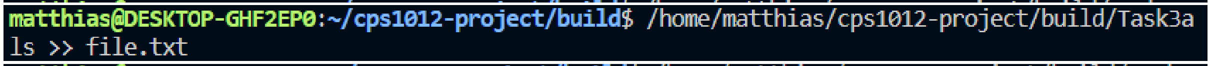
Testing for error checking (The first line shows the input):



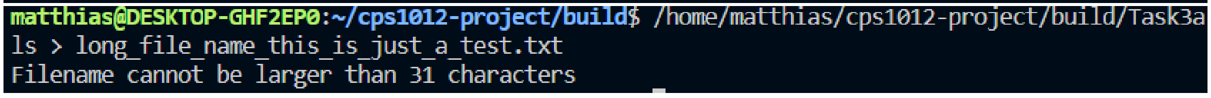
```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3a
| test
| cannot be first or last element
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3a
grep Hello | < Test
| Syntax Error
File does not exist
```



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3a
file < incorrect$file.txt
Filename cannot contain character $
```



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3a
ls >> file.txt
```

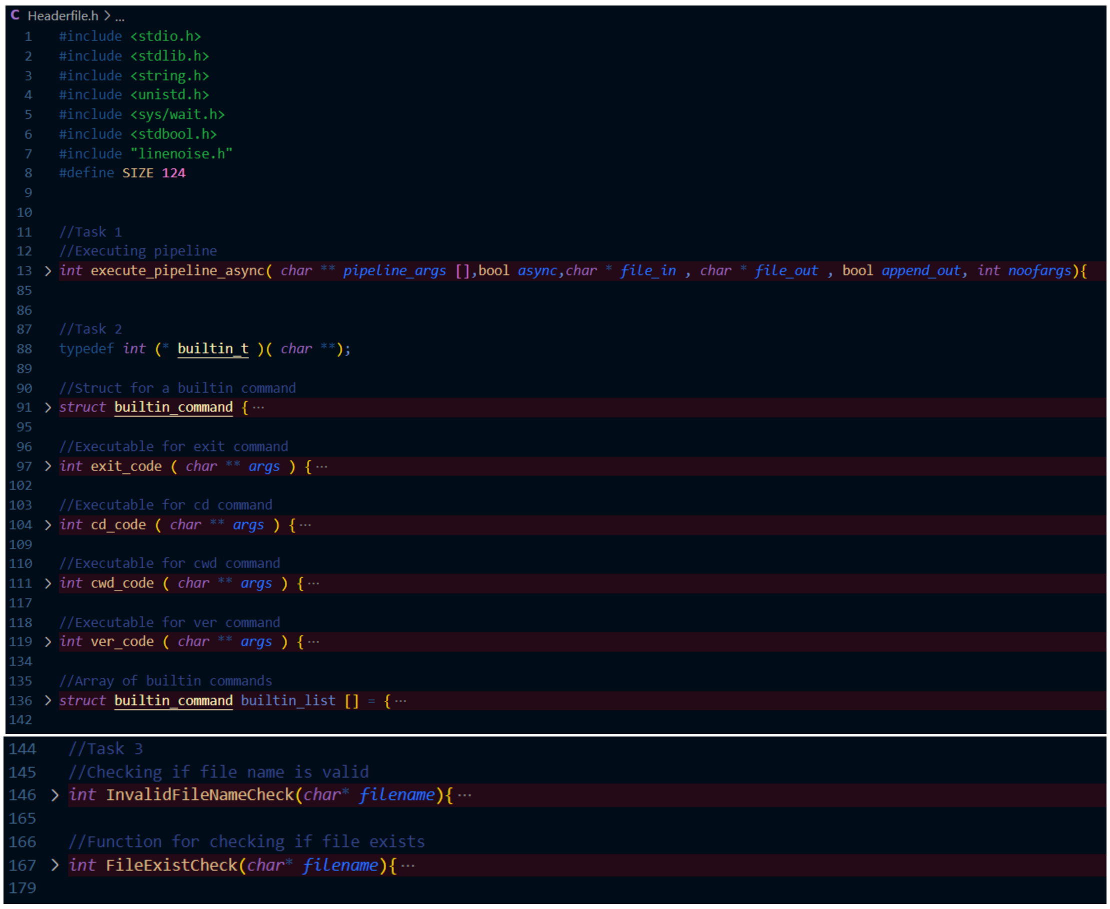


```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3a
ls > long_file_name_this_is_just_a_test.txt
Filename cannot be larger than 31 characters
```

Question b)

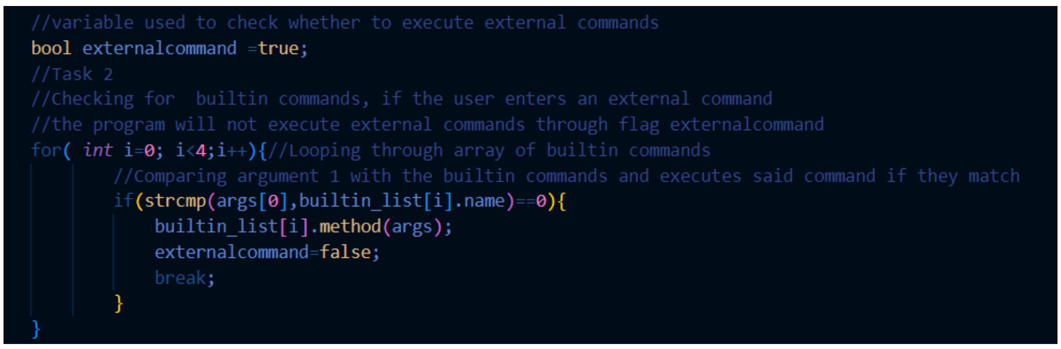
Explanation:

The program is required to execute builtin commands developed in task 2 and command pipeline commands developed in task 1, this was achieved by placing the methods and builtins specified in task 1e), task 2b) and task3a) in a header file called Headerfile.h. Contents of Headerfile.h can be seen below:



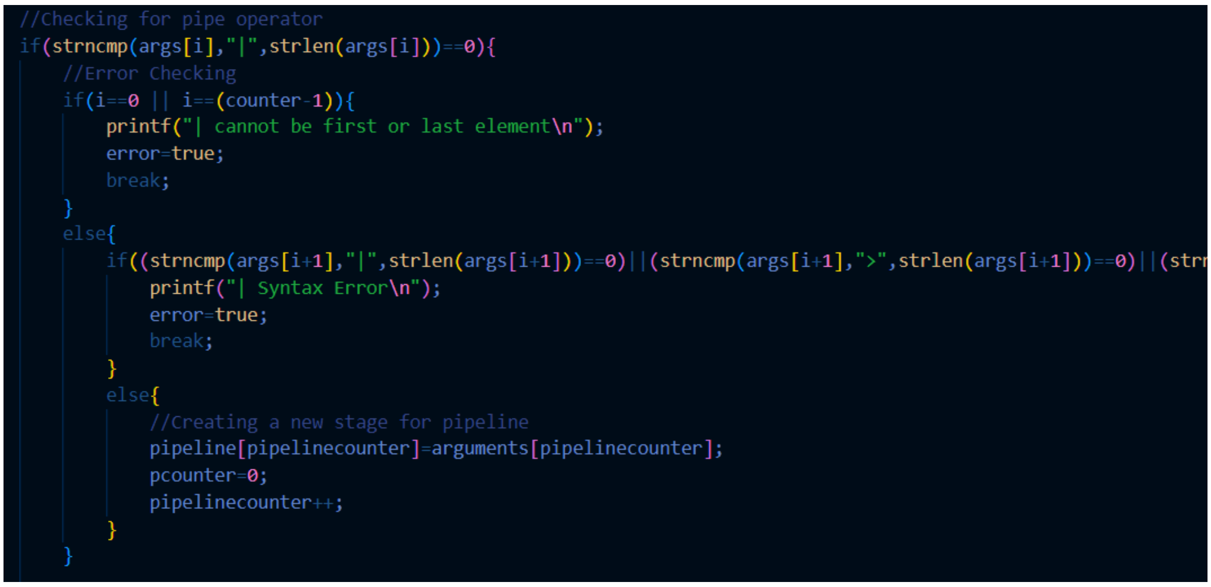
```
C Headerfile.h > ...
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include "linenoise.h"
#define SIZE 124


//Task 1
//Executing pipeline
int execute_pipeline_async( char ** pipeline_args [],bool async,char * file_in , char * file_out , bool append_out, int noofargs){


//Task 2
typedef int (* builtin_t )( char **);

//Struct for a builtin command
struct builtin_command {···

//Executable for exit command
int exit_code ( char ** args ) {···

//Executable for cd command
int cd_code ( char ** args ) {···

//Executable for cwd command
int cwd_code ( char ** args ) {···

//Executable for ver command
int ver_code ( char ** args ) {···

//Array of builtin commands
struct builtin_command builtin_list [] = {···

//Task 3
//Checking if file name is valid
int InvalidFileNameCheck(char* filename){···

//Function for checking if file exists
int FileExistCheck(char* filename){···
```

The program first checks for builtin commands, and if a match is found, it will resort to execute said command, and would terminate. This can be seen below:



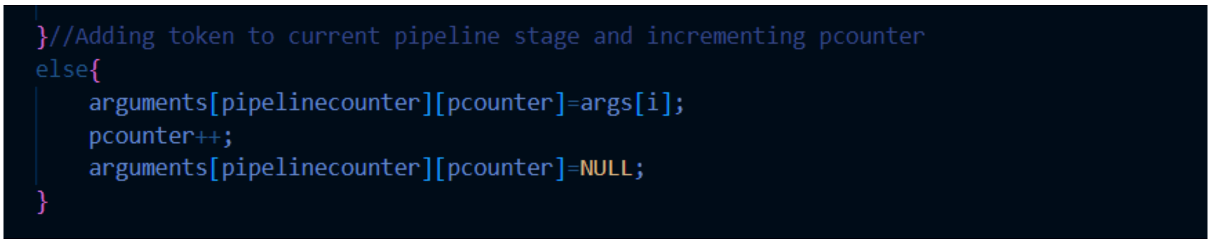
```
//variable used to check whether to execute external commands
bool externalcommand =true;
//Task 2
//Checking for  builtin commands, if the user enters an external command
//the program will not execute external commands through flag externalcommand
for( int i=0; i<4;i++){//Looping through array of builtin commands
        //Comparing argument 1 with the builtin commands and executes said command if they match
        if(strcmp(args[0],builtin_list[i].name)==0){
            builtin_list[i].method(args);
            externalcommand=false;
            break;
        }
}
```

If the user input is not a builtin command, the program will resort to execute an external command pipeline. The program will add the relevant arguments to the pipeline array which will then be fed to the execute_pipeline_async function if no errors occured. This can be seen below:

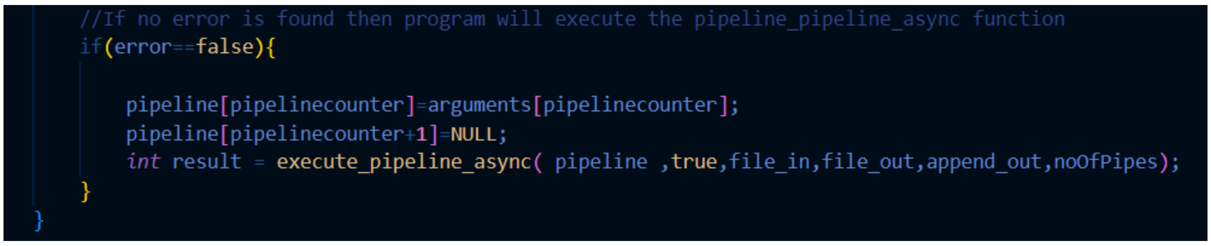


```
//Checking for pipe operator
if(strncmp(args[i],"|",strlen(args[i]))==0){
    //Error Checking
    if(i==0 || i==(counter-1)){
        printf("| cannot be first or last element\n");
        error=true;
        break;
    }
    else{
        if((strncmp(args[i+1],"|",strlen(args[i+1]))==0)||(strncmp(args[i+1],">",strlen(args[i+1]))==0)||(str
            printf("| Syntax Error\n");
            error=true;
            break;
        }
        else{
            //Creating a new stage for pipeline
            pipeline[pipelinecounter]=arguments[pipelinecounter];
            pcounter=0;
            pipelinecounter++;
        }
    }
```

Adding to the pipeline array:



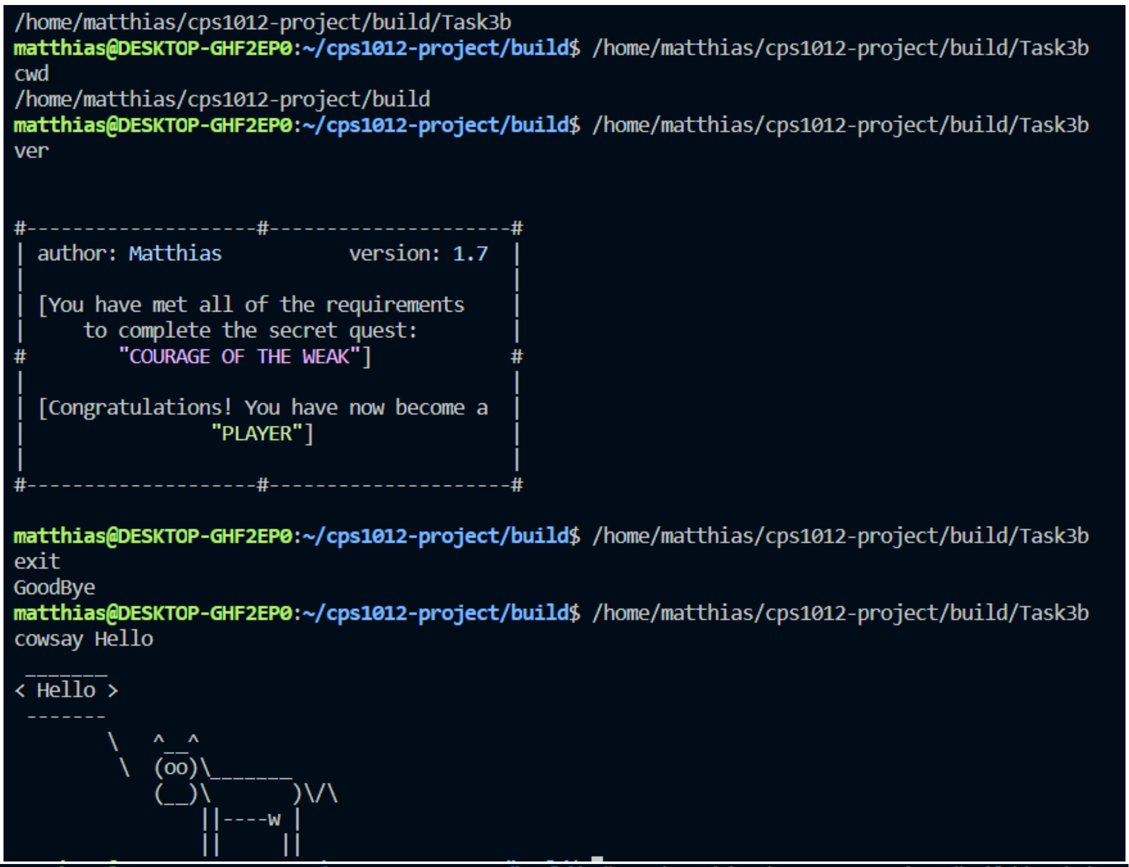
```
}//Adding token to current pipeline stage and incrementing pcounter
else{
    arguments[pipelinecounter][pcounter]=args[i];
    pcounter++;
    arguments[pipelinecounter][pcounter]=NULL;
}
```



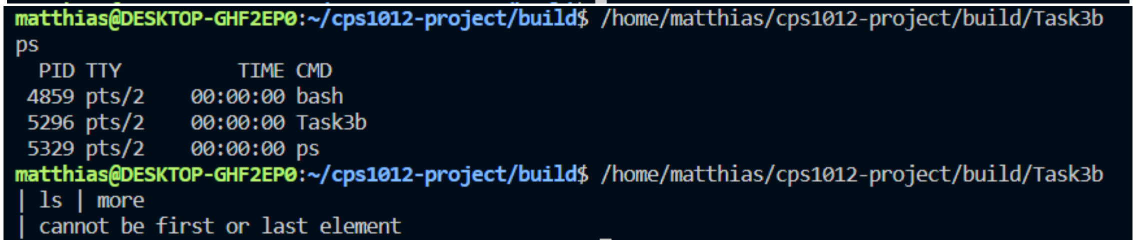
```
    //If no error is found then program will execute the pipeline_pipeline_async function
    if(error==false){

        pipeline[pipelinecounter]=arguments[pipelinecounter];
        pipeline[pipelinecounter+1]=NULL;
        int result = execute_pipeline_async( pipeline ,true,file_in,file_out,append_out,noOfPipes);
    }
}
```

Testing:

Testing for builtin and pipeline commands and errors (The first line shows the input):



```
/home/matthias/cps1012-project/build/Task3b
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3b
cwd
/home/matthias/cps1012-project/build
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3b
ver


#--------------------#---------------------#
| author: Matthias           version: 1.7  |
|                                          |
| [You have met all of the requirements    |
|     to complete the secret quest:        |
#        "COURAGE OF THE WEAK"]            #
|                                          |
| [Congratulations! You have now become a  |
|                "PLAYER"]                 |
|                                          |
#--------------------#---------------------#

matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3b
exit
GoodBye
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3b
cowsay Hello
 _______
< Hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

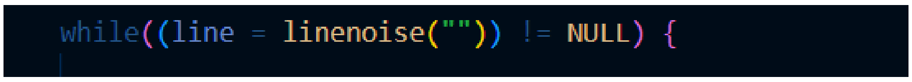


```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3b
ps
  PID TTY          TIME CMD
 4859 pts/2    00:00:00 bash
 5296 pts/2    00:00:00 Task3b
 5329 pts/2    00:00:00 ps
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3b
| ls | more
| cannot be first or last element
```

Question c)

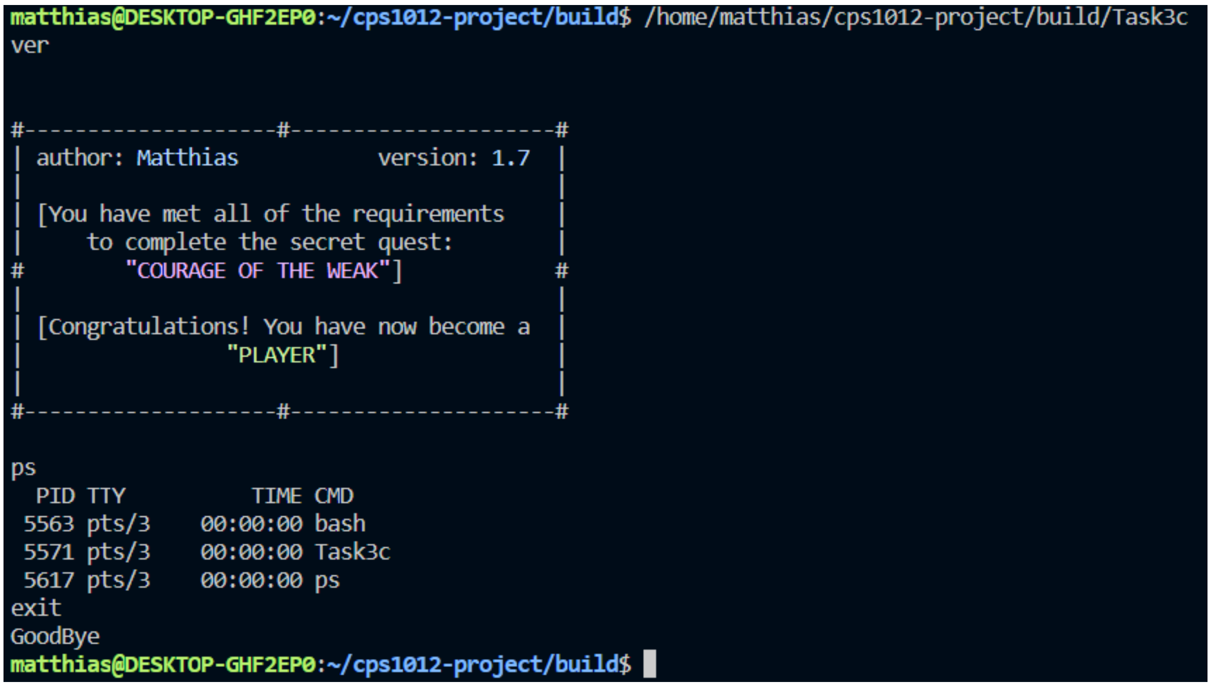
Explanation:

The program was modified to continue execution until the user inputs the exit builtin. This was achieved by enclosing the program developed in task 3b) in an infinite loop, which will loop until the user enters the exit command. This can be seen below:



```
while((line = linenoise("")) != NULL) {
```

Testing (The first line shows the input):

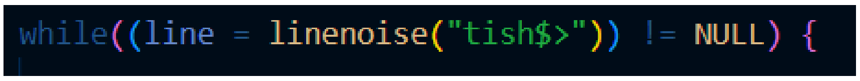


```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3c
ver


#-------------------#--------------------#
| author: Matthias          version: 1.7 |
|                                        |
| [You have met all of the requirements  |
|     to complete the secret quest:      |
#        "COURAGE OF THE WEAK"]          #
|                                        |
| [Congratulations! You have now become a |
|                "PLAYER"]               |
|                                        |
#-------------------#--------------------#

ps
  PID TTY          TIME CMD
 5563 pts/3    00:00:00 bash
 5571 pts/3    00:00:00 Task3c
 5617 pts/3    00:00:00 ps
exit
GoodBye
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$
```

Question d)

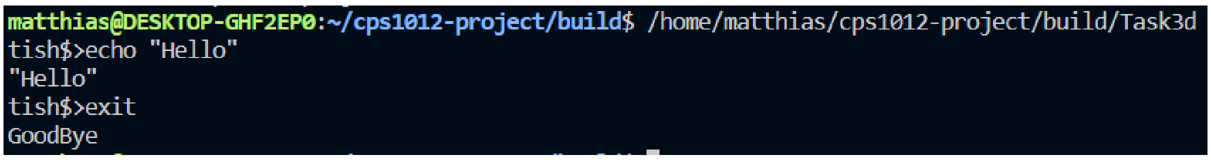
Explanation:

The program was modified to display a prompt (tish>$) when the Shell is requesting for user input. This was achieved by modifying the linenoise command, as can be seen below:



```
while((line = linenoise("tish$>")) != NULL) {
```

Testing (The first line shows the input):



```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task3d
tish$>echo "Hello"
"Hello"
tish$>exit
GoodBye
```

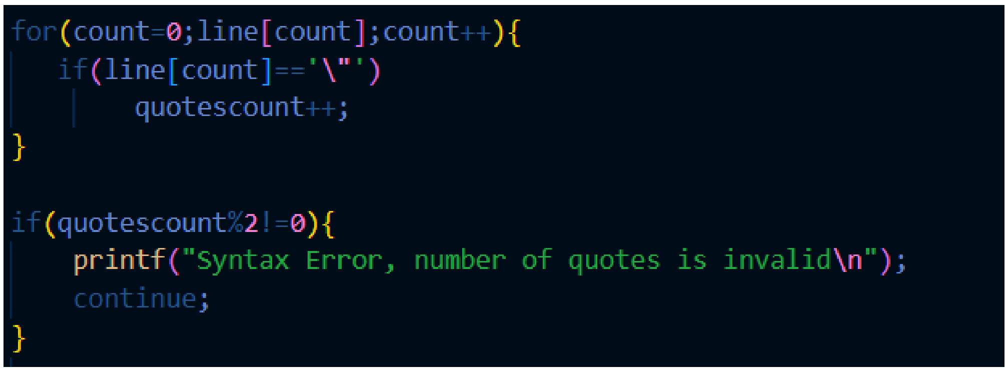
Task 4: Advanced Scanning

Question a)

Explanation:

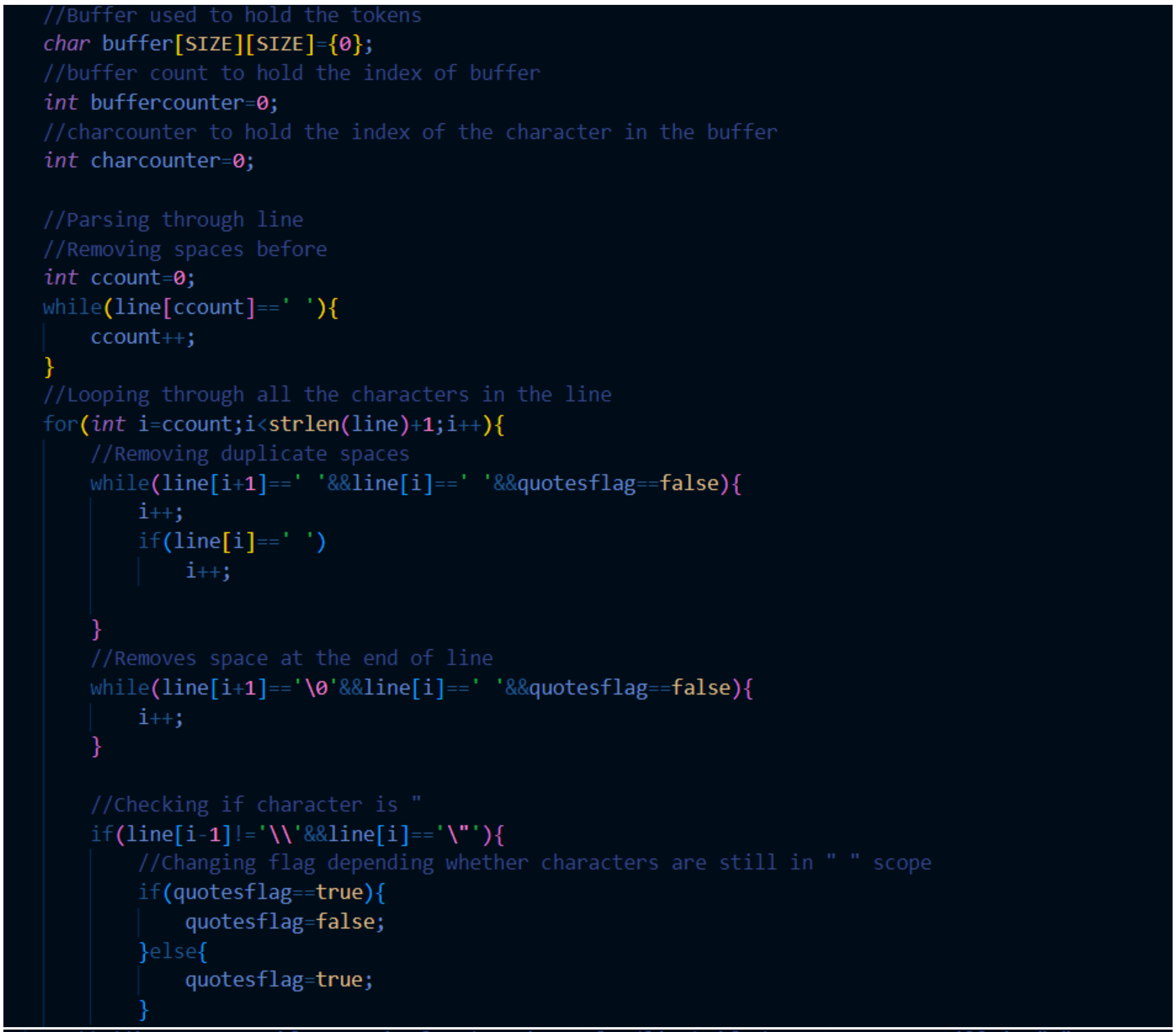
The program is required to read a string of quotation marks which change the meaning of whitespace and the other metacharacters specified inside the quotes. This program continues to build upon the solution presented in task 3d).

The program first checks whether there were any unmatched quotes, this was achieved by looping through all the characters in the line, and every time a quote appears, the quotes counter would be incremented. Finally, the program checks if the quotes counter is odd or even, if it is odd then it will resort to prompt the user with an error, as there would be an unmatched quote. If the number of quotes is even, the program will commence with the execution. This functionality, can be seen below:

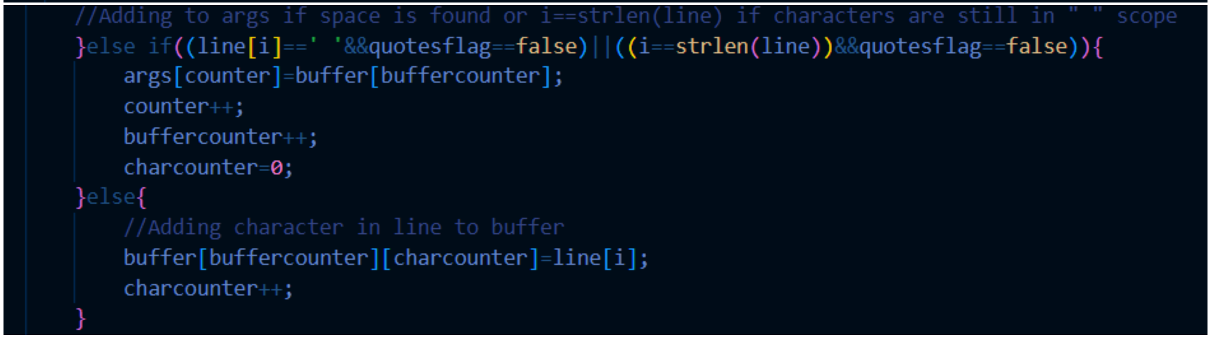


```
for(count=0;line[count];count++){
    if(line[count]=='\"')
        quotescount++;
}

if(quotescount%2!=0){
    printf("Syntax Error, number of quotes is invalid\n");
    continue;
}
```

The parsing method specified in the previous tasks had to be changed, by utilising a for loop and looping through all the characters in the line, rather than utilising the strtok() function. The program required the creation of a buffer which was used to hold the tokens. A while loop is used to count the number of spaces in the beginning, so that if there are any spaces, they are ignored. Inside the for loop, the program checks whether the current character is a " and the preceding character is not a \, if so it will set the quotesflag to true, implying that the next characters are in the " " scope. In the subsequent iterations, if the quotesflag is still set to true, the program will resort to add the characters to the buffer, until another " is found, which's preceding character is not a \, which will trigger the quotesflag to become false. If the quotesflag is false, and a space is found or the end of the line is reached, the program will resort to add the buffer to the arguments array, and the buffer counter is incremented to store the next token in a new position. The arguments counter is also incremented. In addition, there are 2 while loops inside the for loop, whereby one removes the space at the end of the line, and the other would remove the duplicate space. The following procedure can be seen in the following page:



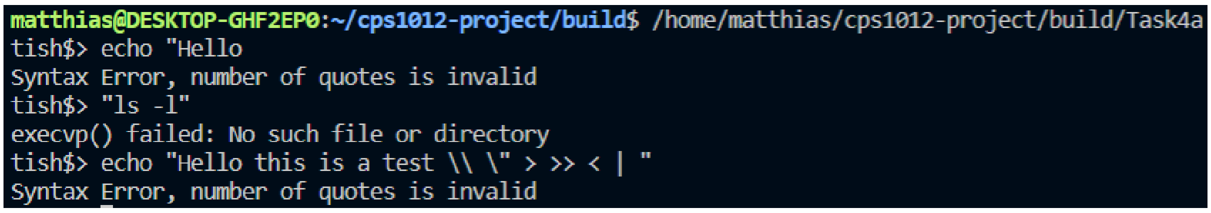
```
//Buffer used to hold the tokens
char buffer[SIZE][SIZE]={0};
//buffer count to hold the index of buffer
int buffercounter=0;
//charcounter to hold the index of the character in the buffer
int charcounter=0;

//Parsing through line
//Removing spaces before
int ccount=0;
while(line[ccount]==' '){
    ccount++;
}
//Looping through all the characters in the line
for(int i=ccount;i<strlen(line)+1;i++){
    //Removing duplicate spaces
    while(line[i+1]==' '&&line[i]==' '&&quotesflag==false){
        i++;
        if(line[i]==' ')
            i++;

    }
    //Removes space at the end of line
    while(line[i+1]=='\0'&&line[i]==' '&&quotesflag==false){
        i++;
    }

    //Checking if character is "
    if(line[i-1]!='\\'&&line[i]=='\"'){
        //Changing flag depending whether characters are still in " " scope
        if(quotesflag==true){
            quotesflag=false;
        }else{
            quotesflag=true;
        }
```



```
    //Adding to args if space is found or i==strlen(line) if characters are still in " " scope
    }else if((line[i]==' '&&quotesflag==false)||((i==strlen(line))&&quotesflag==false)){
        args[counter]=buffer[buffercounter];
        counter++;
        buffercounter++;
        charcounter=0;
    }else{
        //Adding character in line to buffer
        buffer[buffercounter][charcounter]=line[i];
        charcounter++;
    }
```

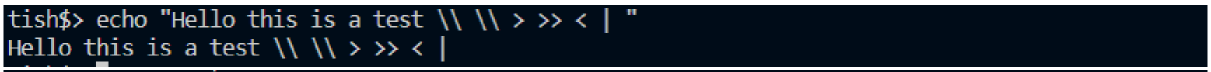
Testing (The line with the user prompt (tish$>) shows the input):

Testing for invalid input:

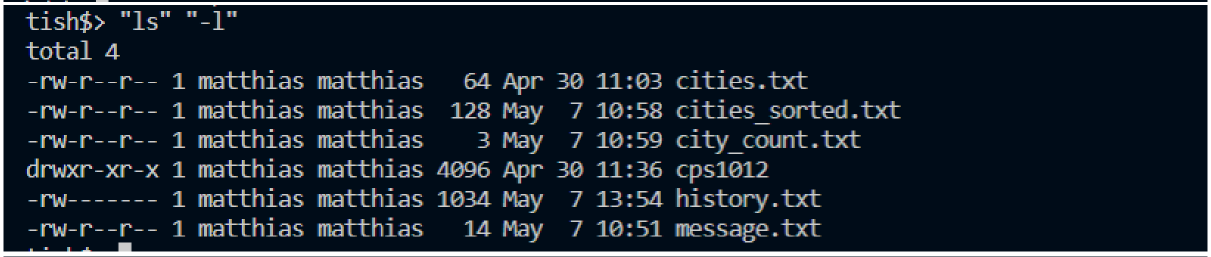


```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task4a
tish$> echo "Hello
Syntax Error, number of quotes is invalid
tish$> "ls -l"
execvp() failed: No such file or directory
tish$> echo "Hello this is a test \\ \" > >> < | "
Syntax Error, number of quotes is invalid
```

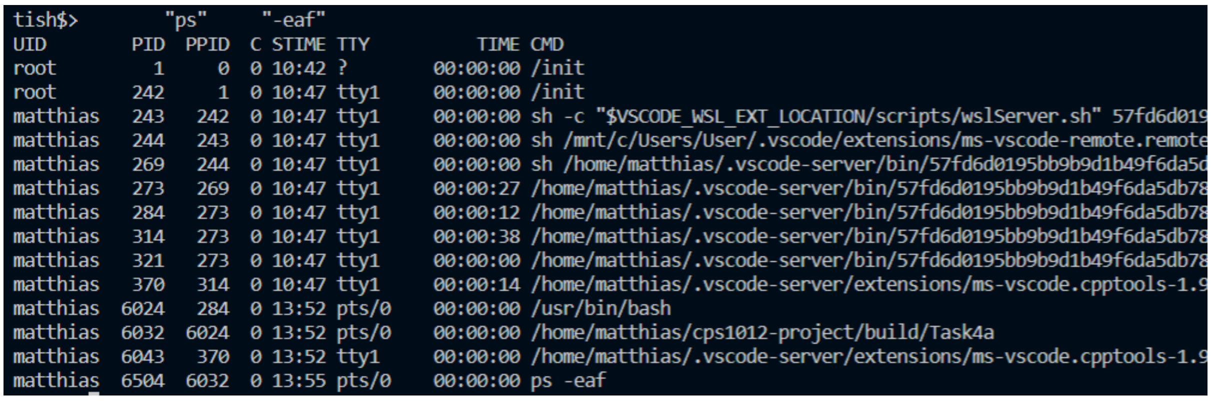
Testing for valid input:



```
tish$> echo "Hello this is a test \\ \\ > >> < | "
Hello this is a test \\ \\ > >> < |
```



```
tish$> "ls" "-l"
total 4
-rw-r--r-- 1 matthias matthias   64 Apr 30 11:03 cities.txt
-rw-r--r-- 1 matthias matthias  128 May  7 10:58 cities_sorted.txt
-rw-r--r-- 1 matthias matthias    3 May  7 10:59 city_count.txt
drwxr-xr-x 1 matthias matthias 4096 Apr 30 11:36 cps1012
-rw------- 1 matthias matthias 1034 May  7 13:54 history.txt
-rw-r--r-- 1 matthias matthias   14 May  7 10:51 message.txt
```

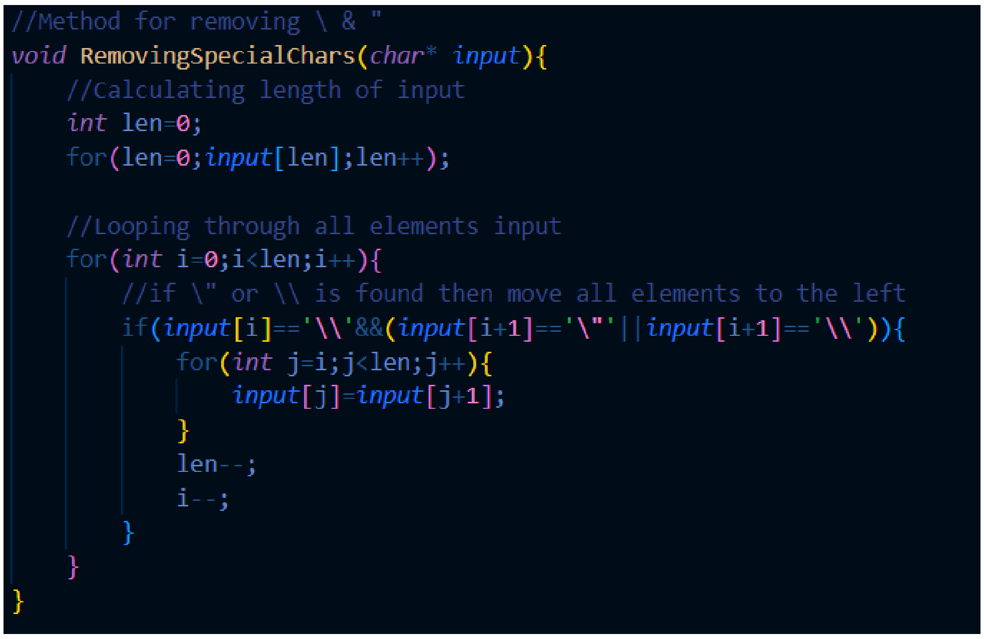


```
tish$>       "ps"     "-eaf"
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 10:42 ?        00:00:00 /init
root       242     1  0 10:47 tty1     00:00:00 /init
matthias   243   242  0 10:47 tty1     00:00:00 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" 57fd6d019
matthias   244   243  0 10:47 tty1     00:00:00 sh /mnt/c/Users/User/.vscode/extensions/ms-vscode-remote.remote
matthias   269   244  0 10:47 tty1     00:00:00 sh /home/matthias/.vscode-server/bin/57fd6d0195bb9b9d1b49f6da5d
matthias   273   269  0 10:47 tty1     00:00:27 /home/matthias/.vscode-server/bin/57fd6d0195bb9b9d1b49f6da5db78
matthias   284   273  0 10:47 tty1     00:00:12 /home/matthias/.vscode-server/bin/57fd6d0195bb9b9d1b49f6da5db78
matthias   314   273  0 10:47 tty1     00:00:38 /home/matthias/.vscode-server/bin/57fd6d0195bb9b9d1b49f6da5db78
matthias   321   273  0 10:47 tty1     00:00:00 /home/matthias/.vscode-server/bin/57fd6d0195bb9b9d1b49f6da5db78
matthias   370   314  0 10:47 tty1     00:00:14 /home/matthias/.vscode-server/extensions/ms-vscode.cpptools-1.9
matthias  6024   284  0 13:52 pts/0    00:00:00 /usr/bin/bash
matthias  6032  6024  0 13:52 pts/0    00:00:00 /home/matthias/cps1012-project/build/Task4a
matthias  6043   370  0 13:52 tty1     00:00:00 /home/matthias/.vscode-server/extensions/ms-vscode.cpptools-1.9
matthias  6504  6032  0 13:55 pts/0    00:00:00 ps -eaf
```

Question b)

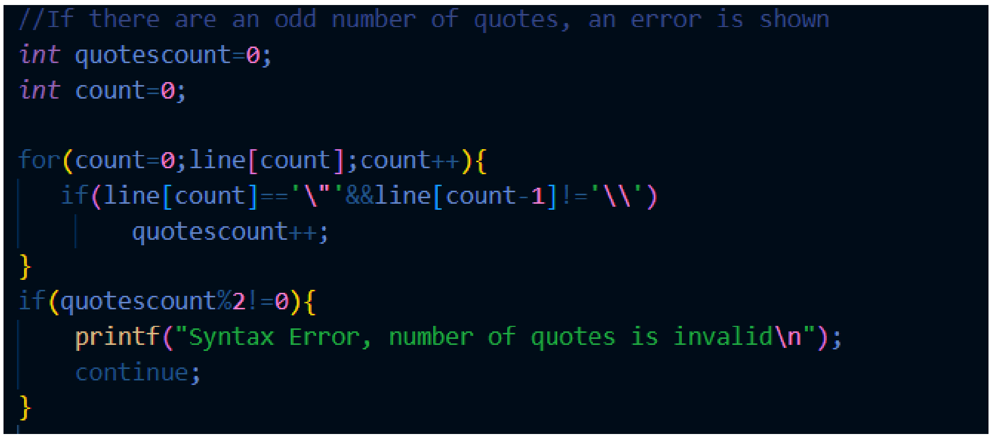
Explanation:

The program builds upon the rule specified in task 4a), whereby backslash characters strip the meaning of the \ and “ characters. To implement said functionality, a function was added to the Headerfile.h file, which loops through a string character by character, and if a \\ or \” is encountered, it will remove the first backslash from the string. The function which can be seen below was called, right before the contents of the buffer are copied to the arguments array.



```
//Method for removing \ & "
void RemovingSpecialChars(char* input){
    //Calculating length of input
    int len=0;
    for(len=0;input[len];len++);

    //Looping through all elements input
    for(int i=0;i<len;i++){
        //if \" or \\ is found then move all elements to the left
        if(input[i]=='\\'&&(input[i+1]=='\"'||input[i+1]=='\\')){
            for(int j=i;j<len;j++){
                input[j]=input[j+1];
            }
            len--;
            i--;
        }
    }
}
```

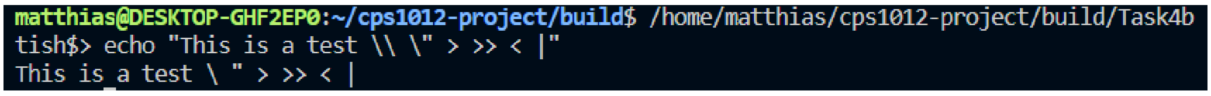
The method for checking for unmatched quotes was modified to adapt to the new requirements. Modifications can be seen below:



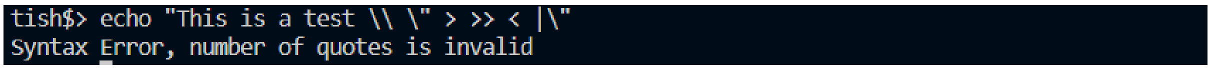
```
//If there are an odd number of quotes, an error is shown
int quotescount=0;
int count=0;

for(count=0;line[count];count++){
    if(line[count]=='\"'&&line[count-1]!='\\')
        quotescount++;
}
if(quotescount%2!=0){
    printf("Syntax Error, number of quotes is invalid\n");
    continue;
}
```

Testing (The line with the user prompt (tish$>) shows the input):

Testing for valid input:


```
matthias@DESKTOP-GHF2EP0:~/cps1012-project/build$ /home/matthias/cps1012-project/build/Task4b
tish$> echo "This is a test \\ \" > >> < |"
This is a test \ " > >> < |
```

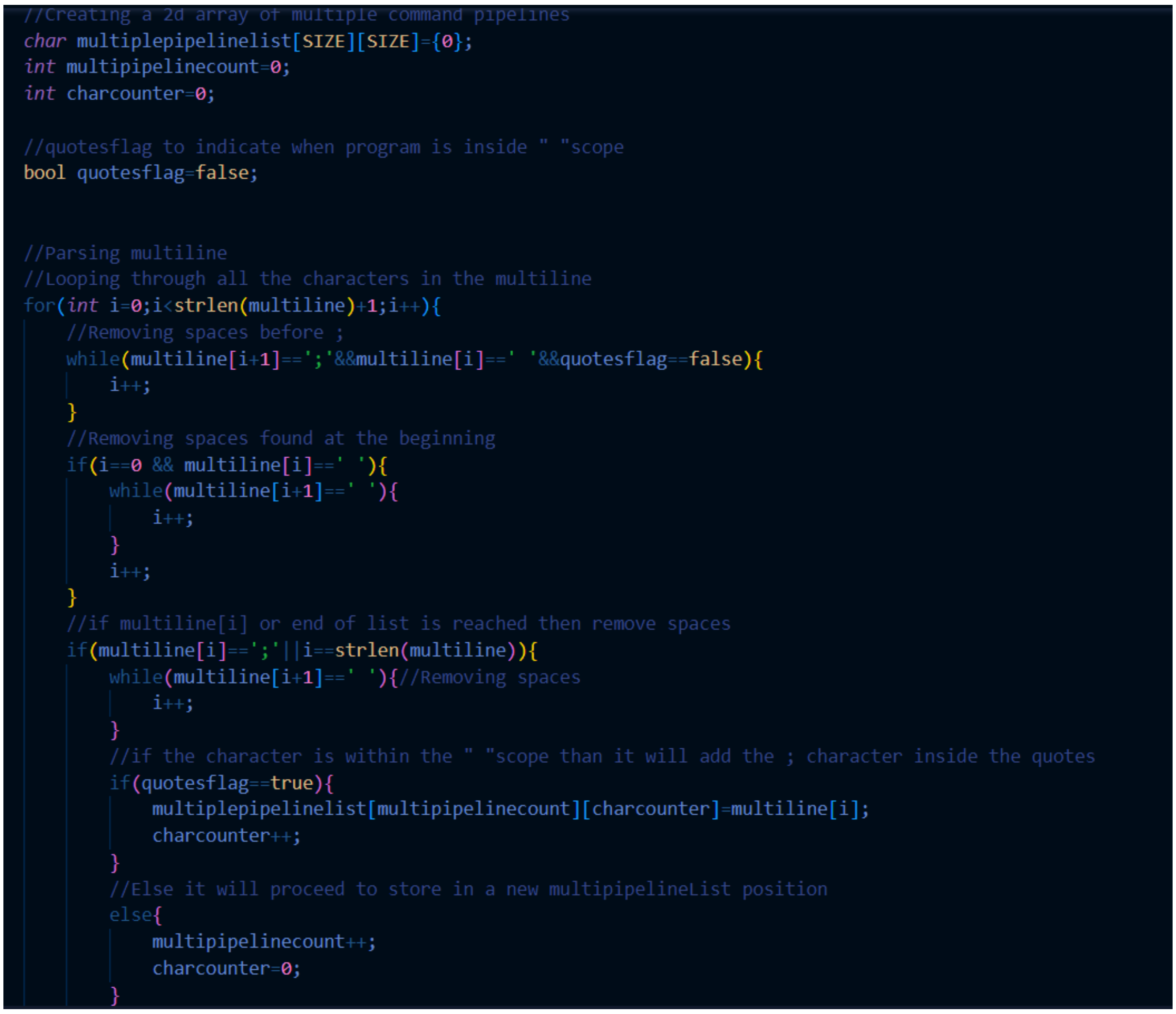
Testing for invalid input:


```
tish$> echo "This is a test \\ \" > >> < |\"
Syntax Error, number of quotes is invalid
```

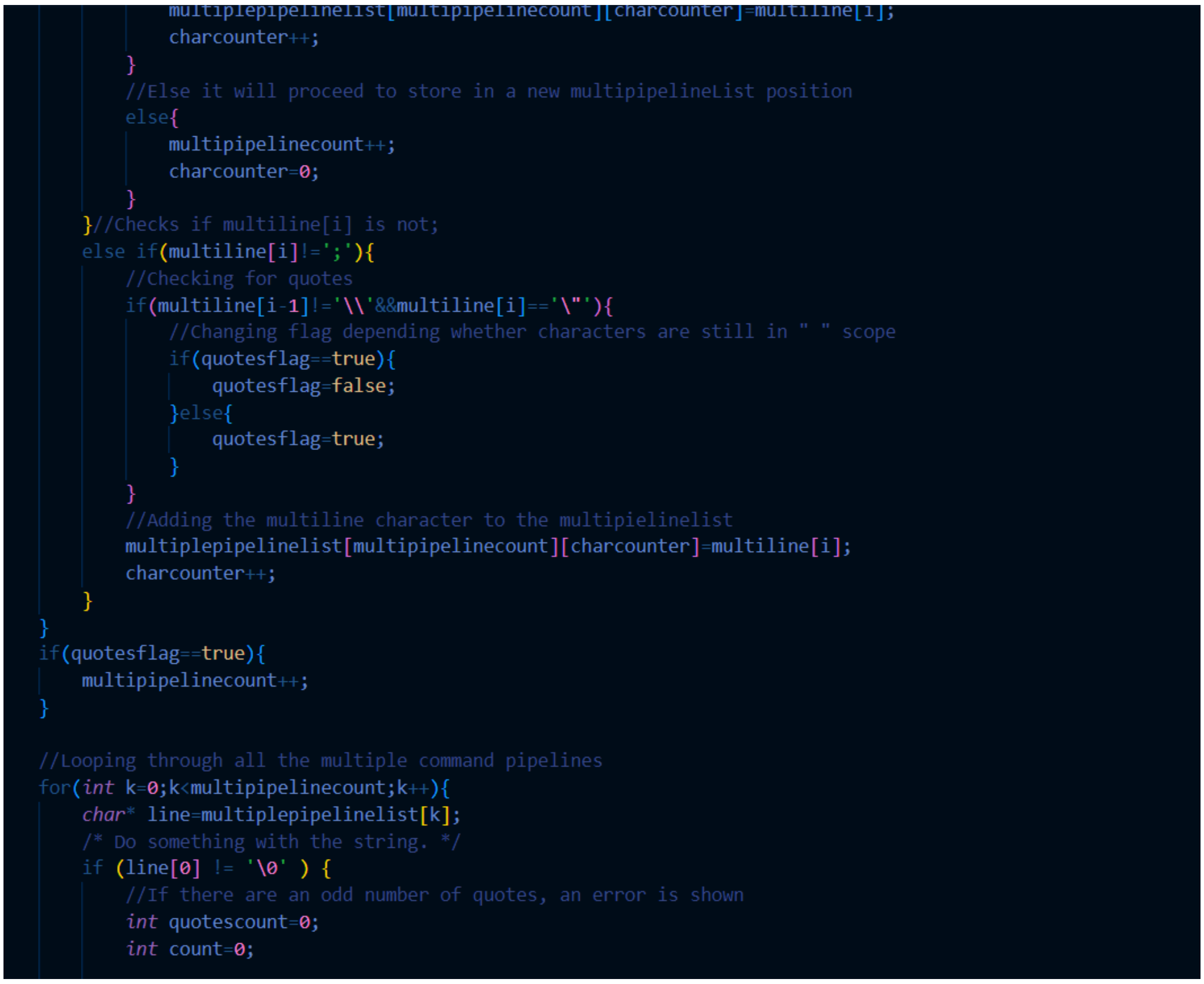
Question c)

Explanation:

The program modifies task 4b), to include a new metacharacter, which may be used to separate multiple command pipelines. The program utilises another parsing method, to parse the ; metacharacter. This parsing method works very similar to the parsing method specified in task 4a), and occurs before parsing for “ (parsing specified in task 4a)). If the ; is inside the “ “ scope, the program will treat the character as a normal character, following the rule specified in task 4a). The program will add the separate multiple command pipelines in the array multiplepipelinelist, as each element in the array is a command pipeline. The program will loop through the contents of the multiplepipelinelist, and each time will execute task 4b for each pipeline command. The modified section can be seen below:



```
//Creating a 2d array of multiple command pipelines
char multiplepipelinelist[SIZE][SIZE]={0};
int multipipelinecount=0;
int charcounter=0;

//quotesflag to indicate when program is inside " "scope
bool quotesflag=false;


//Parsing multiline
//Looping through all the characters in the multiline
for(int i=0;i<strlen(multiline)+1;i++){
    //Removing spaces before ;
    while(multiline[i+1]==';'&&multiline[i]==' '&&quotesflag==false){
        i++;
    }
    //Removing spaces found at the beginning
    if(i==0 && multiline[i]==' '){
        while(multiline[i+1]==' '){
            i++;
        }
        i++;
    }
    //if multiline[i] or end of list is reached then remove spaces
    if(multiline[i]==';'||i==strlen(multiline)){
        while(multiline[i+1]==' '){//Removing spaces
            i++;
        }
        //if the character is within the " "scope than it will add the ; character inside the quotes
        if(quotesflag==true){
            multiplepipelinelist[multipipelinecount][charcounter]=multiline[i];
            charcounter++;
        }
        //Else it will proceed to store in a new multipipelineList position
        else{
            multipipelinecount++;
            charcounter=0;
        }
```



```
            multiplepipelinelist[multipipelinecount][charcounter]=multiline[i];
            charcounter++;
        }
        //Else it will proceed to store in a new multipipelineList position
        else{
            multipipelinecount++;
            charcounter=0;
        }
    }//Checks if multiline[i] is not;
    else if(multiline[i]!=';'){
        //Checking for quotes
        if(multiline[i-1]!='\\'&&multiline[i]=='\"'){
            //Changing flag depending whether characters are still in " " scope
            if(quotesflag==true){
                quotesflag=false;
            }else{
                quotesflag=true;
            }
        }
        //Adding the multiline character to the multipielinelist
        multiplepipelinelist[multipipelinecount][charcounter]=multiline[i];
        charcounter++;
    }
}
if(quotesflag==true){
    multipipelinecount++;
}

//Looping through all the multiple command pipelines
for(int k=0;k<multipipelinecount;k++){
    char* line=multiplepipelinelist[k];
    /* Do something with the string. */
    if (line[0] != '\0' ) {
        //If there are an odd number of quotes, an error is shown
        int quotescount=0;
        int count=0;
```

<u>Testing for task 4c) can be seen in the next section.</u>

Final testing on the Tiny Shell:

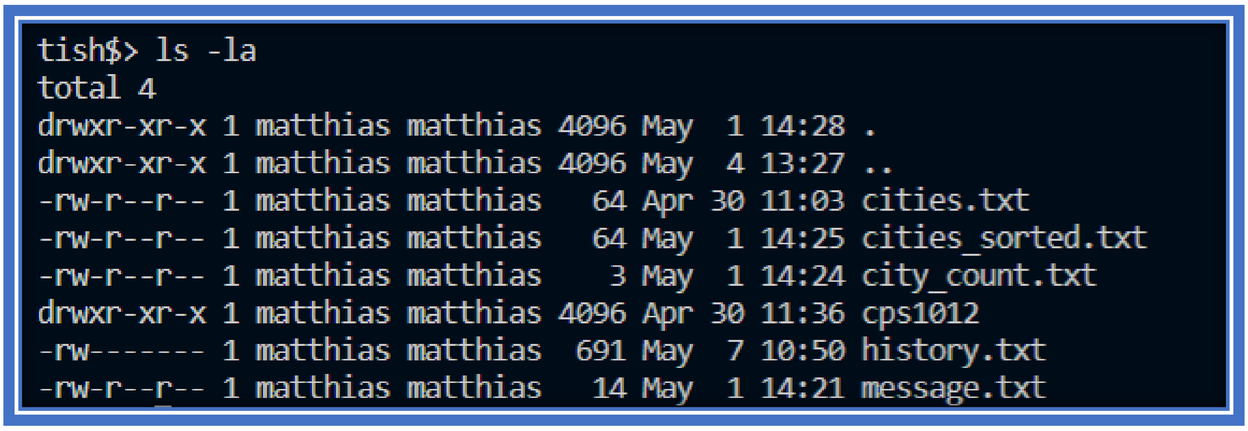
The following are tests which were performed on the Tiny Shell implementation of Task 4c):

Note:

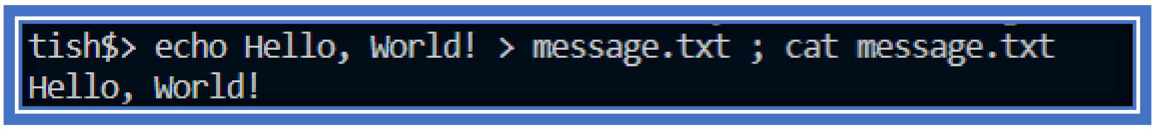
- The input text is prefixed by the Shell command prompt (tish$>), and prompt is not part of input string. The output of input follows in the subsequent lines.

- The shell was tested with testing examples found in the **tiny_shell_examples.pdf** file on VLE, and all produce the output which is specified in the sheet.

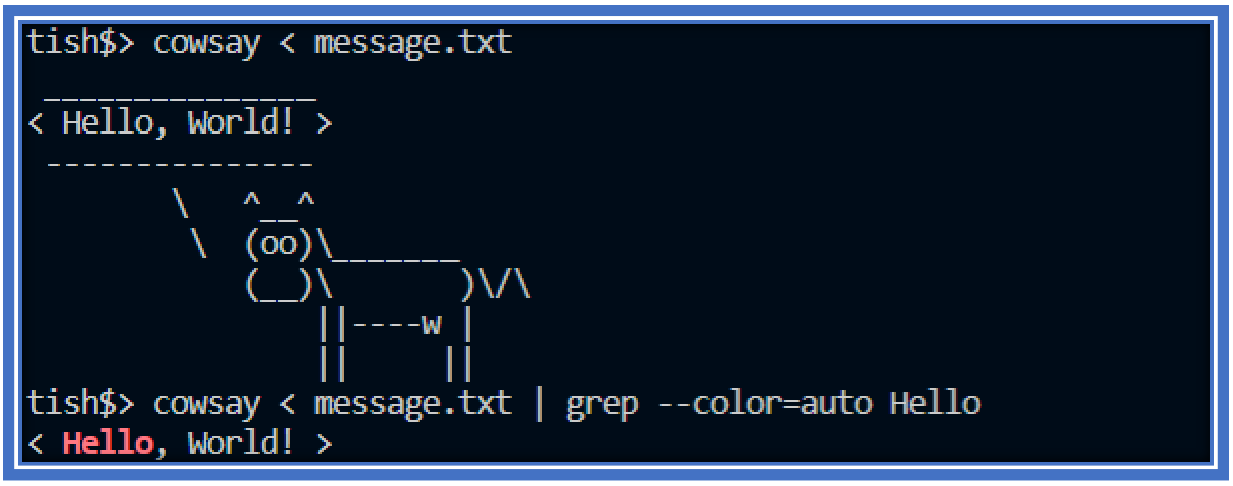
Command Pipelines:



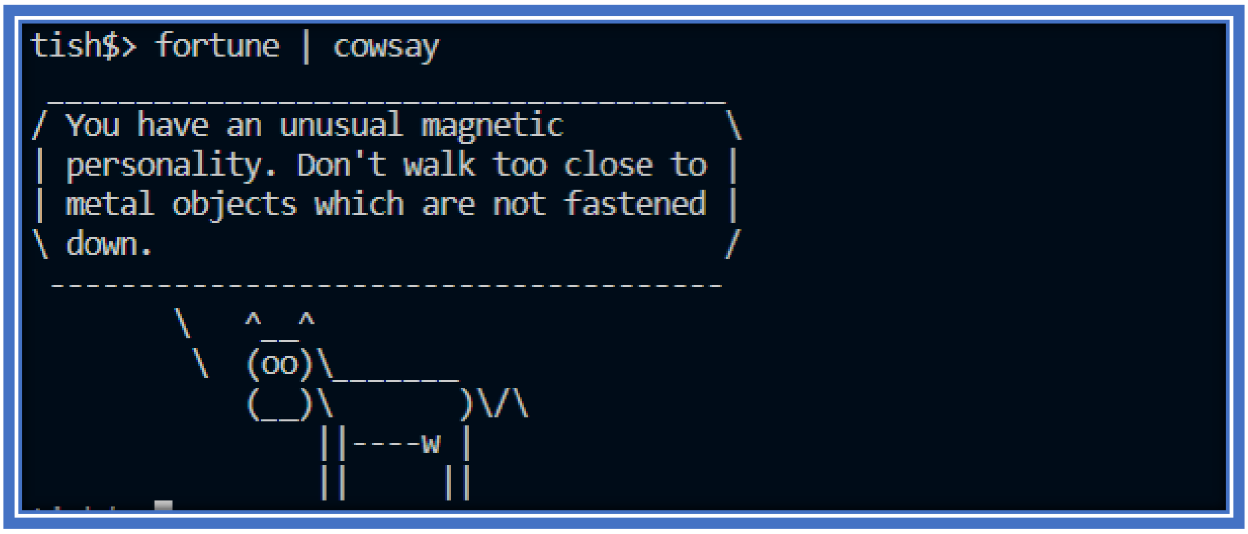
```
tish$> ls -la
total 4
drwxr-xr-x 1 matthias matthias 4096 May  1 14:28 .
drwxr-xr-x 1 matthias matthias 4096 May  4 13:27 ..
-rw-r--r-- 1 matthias matthias   64 Apr 30 11:03 cities.txt
-rw-r--r-- 1 matthias matthias   64 May  1 14:25 cities_sorted.txt
-rw-r--r-- 1 matthias matthias    3 May  1 14:24 city_count.txt
drwxr-xr-x 1 matthias matthias 4096 Apr 30 11:36 cps1012
-rw------- 1 matthias matthias  691 May  7 10:50 history.txt
-rw-r--r-- 1 matthias matthias   14 May  1 14:21 message.txt
```



```
tish$> echo Hello, World! > message.txt ; cat message.txt
Hello, World!
```

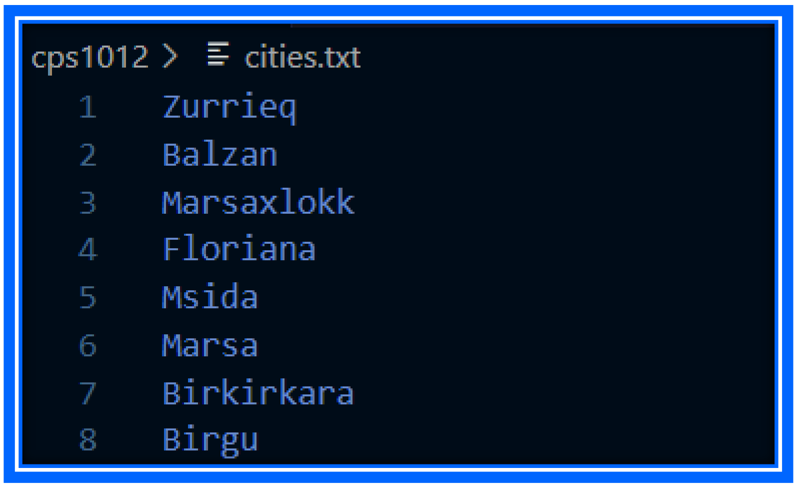


```
tish$> cowsay < message.txt
 _______________
< Hello, World! >
 ---------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
tish$> cowsay < message.txt | grep --color=auto Hello
< Hello, World! >
```

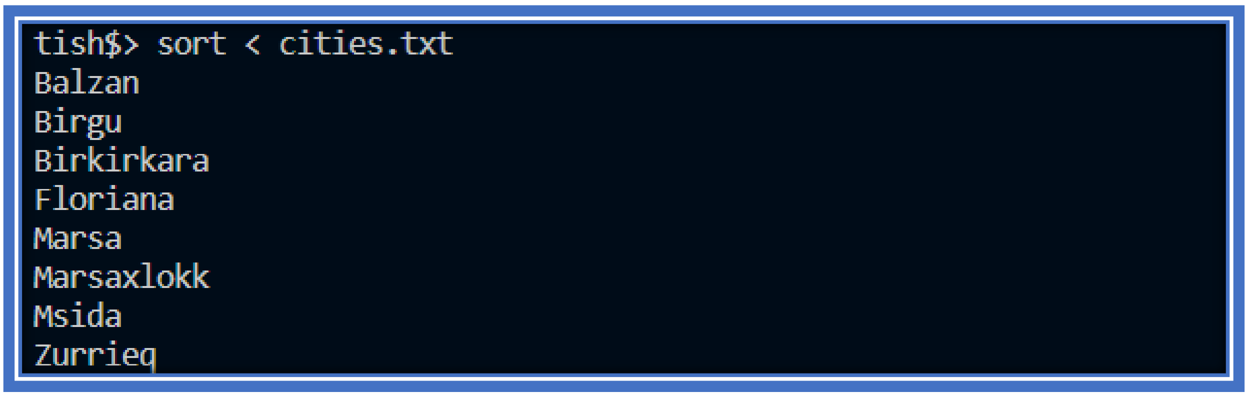


```
tish$> fortune | cowsay
 ______________________________________
/ You have an unusual magnetic         \
| personality. Don't walk too close to |
| metal objects which are not fastened |
\ down.                                /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

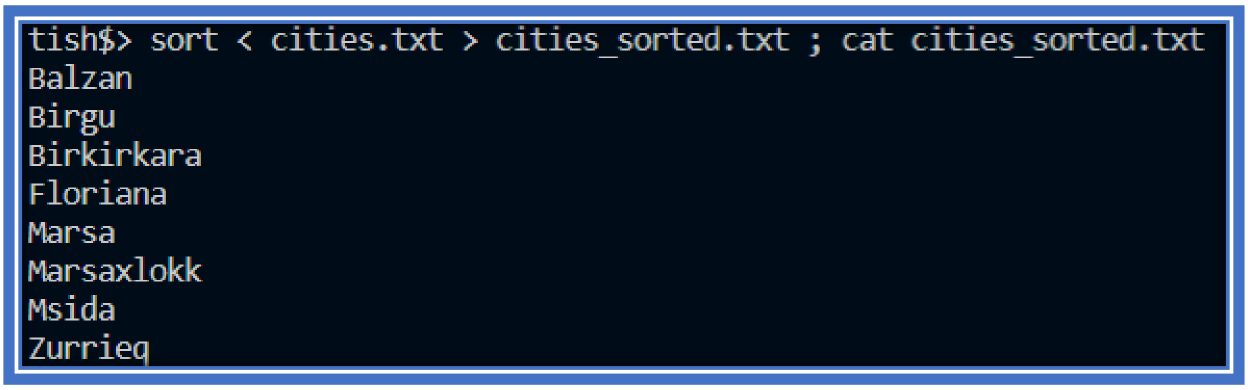
Note that the cities file contains the following elements, this file will be used in the subsequent tests:



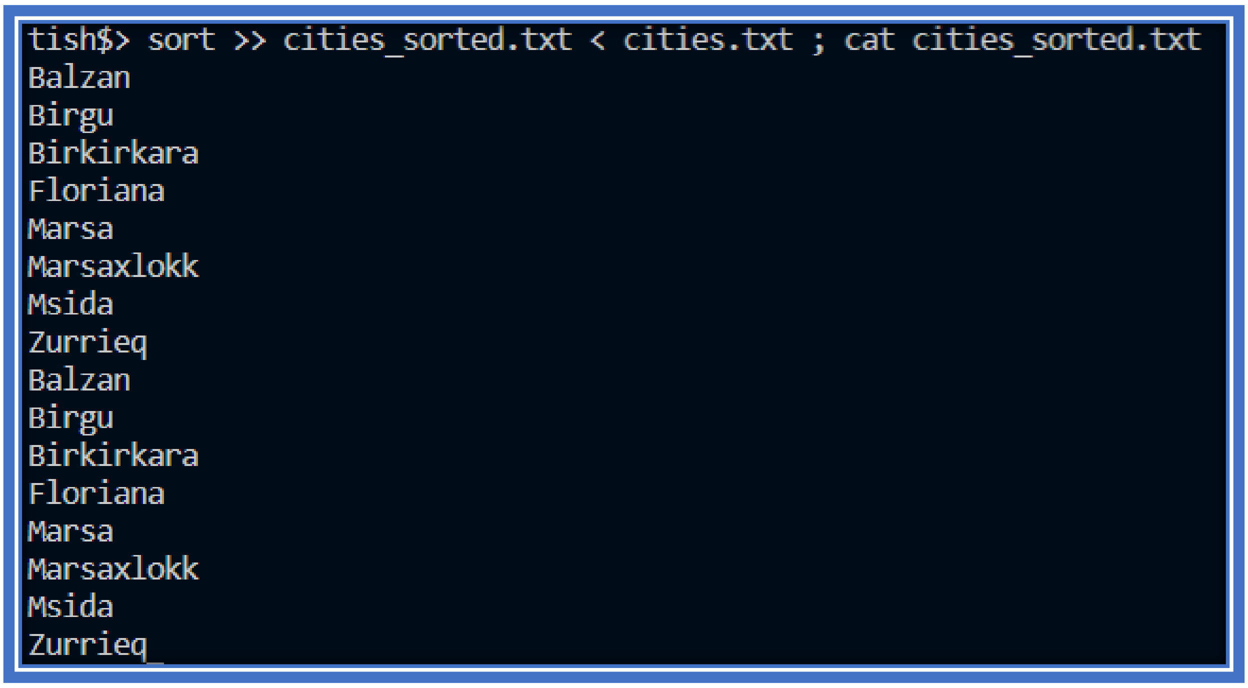
```
cps1012 > cities.txt
Zurrieq
Balzan
Marsaxlokk
Floriana
Msida
Marsa
Birkirkara
Birgu
```



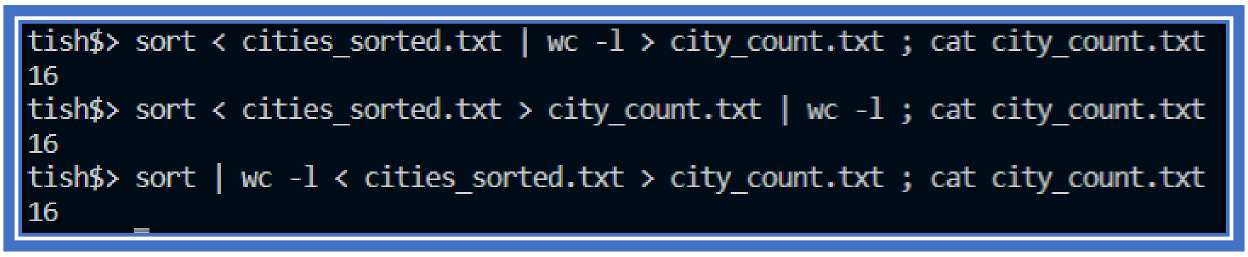
```
tish$> sort < cities.txt
Balzan
Birgu
Birkirkara
Floriana
Marsa
Marsaxlokk
Msida
Zurrieq
```



```
tish$> sort < cities.txt > cities_sorted.txt ; cat cities_sorted.txt
Balzan
Birgu
Birkirkara
Floriana
Marsa
Marsaxlokk
Msida
Zurrieq
```

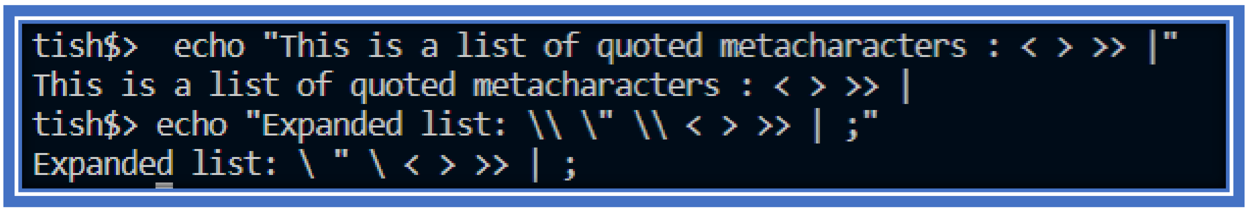


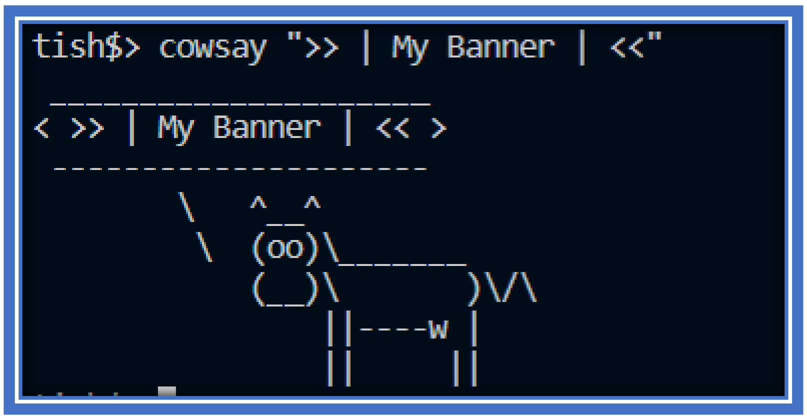
```
tish$> sort >> cities_sorted.txt < cities.txt ; cat cities_sorted.txt
Balzan
Birgu
Birkirkara
Floriana
Marsa
Marsaxlokk
Msida
Zurrieq
Balzan
Birgu
Birkirkara
Floriana
Marsa
Marsaxlokk
Msida
Zurrieq_
```

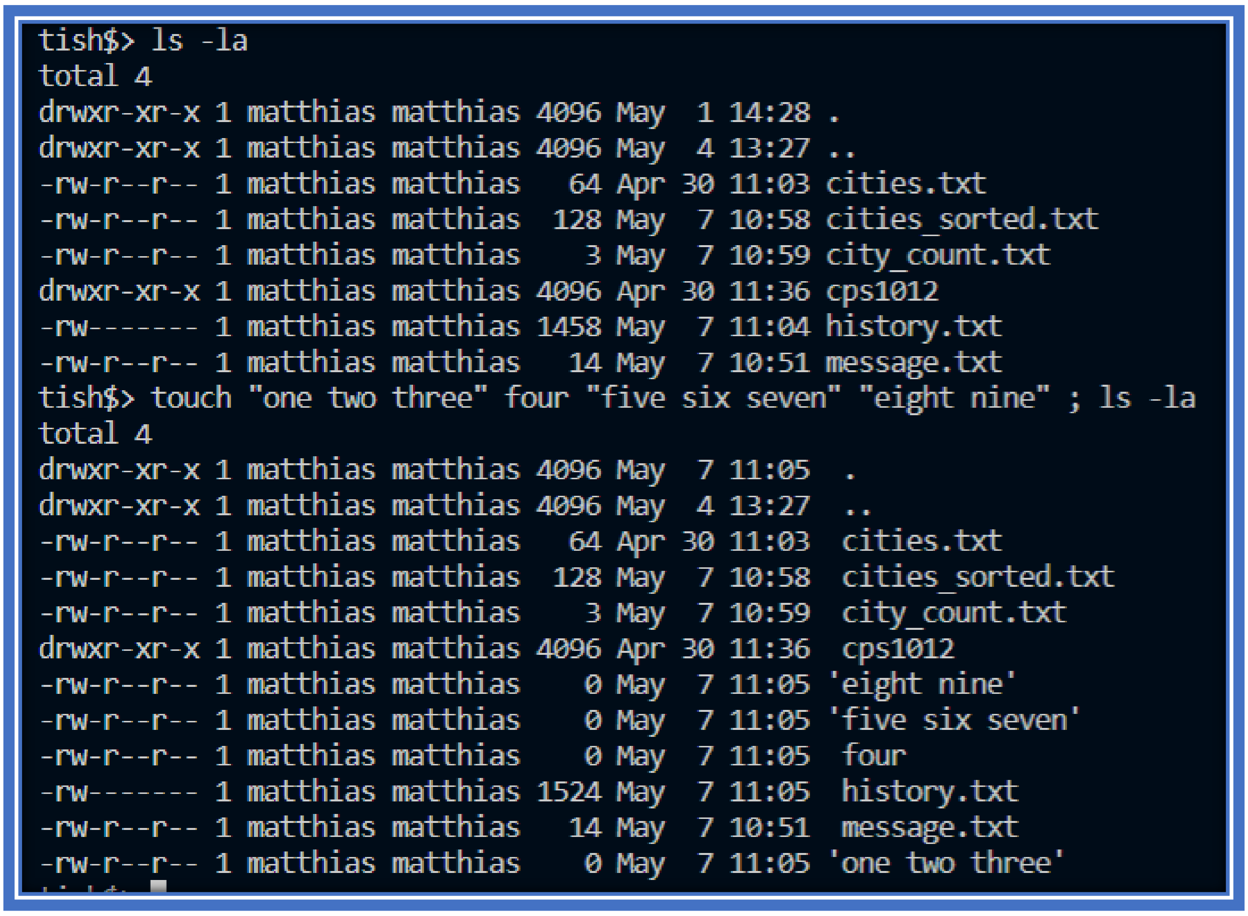


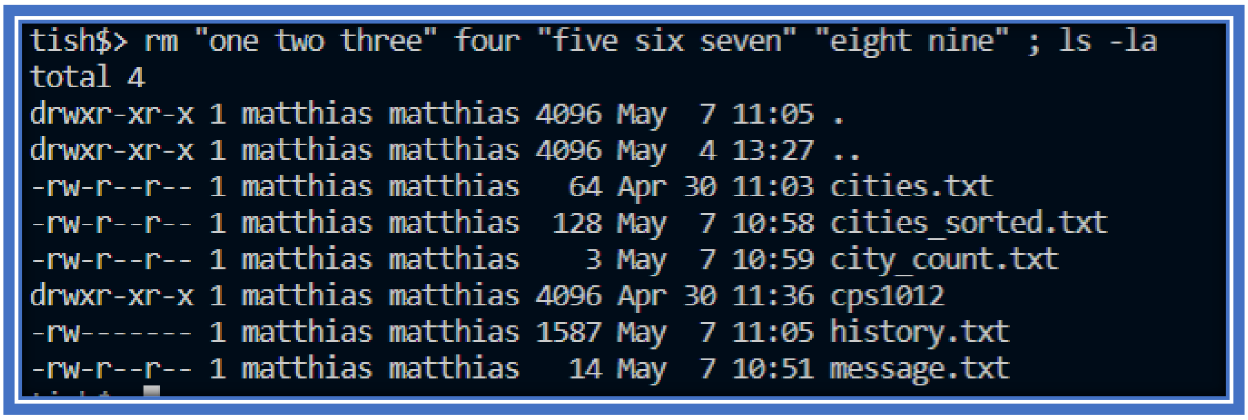
```
tish$> sort < cities_sorted.txt | wc -l > city_count.txt ; cat city_count.txt
16
tish$> sort < cities_sorted.txt > city_count.txt | wc -l ; cat city_count.txt
16
tish$> sort | wc -l < cities_sorted.txt > city_count.txt ; cat city_count.txt
16
```

Advanced Scanning:

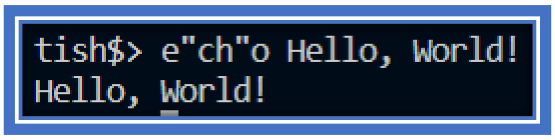

```
tish$>  echo "This is a list of quoted metacharacters : < > >> |"
This is a list of quoted metacharacters : < > >> |
tish$> echo "Expanded list: \\ \" \\ < > >> | ;"
Expanded list: \ " \ < > >> | ;
```


```
tish$> cowsay ">> | My Banner | <<"

 ______________________
< >> | My Banner | << >
 ----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```


```
tish$> ls -la
total 4
drwxr-xr-x 1 matthias matthias 4096 May  1 14:28 .
drwxr-xr-x 1 matthias matthias 4096 May  4 13:27 ..
-rw-r--r-- 1 matthias matthias   64 Apr 30 11:03 cities.txt
-rw-r--r-- 1 matthias matthias  128 May  7 10:58 cities_sorted.txt
-rw-r--r-- 1 matthias matthias    3 May  7 10:59 city_count.txt
drwxr-xr-x 1 matthias matthias 4096 Apr 30 11:36 cps1012
-rw------- 1 matthias matthias 1458 May  7 11:04 history.txt
-rw-r--r-- 1 matthias matthias   14 May  7 10:51 message.txt
tish$> touch "one two three" four "five six seven" "eight nine" ; ls -la
total 4
drwxr-xr-x 1 matthias matthias 4096 May  7 11:05  .
drwxr-xr-x 1 matthias matthias 4096 May  4 13:27  ..
-rw-r--r-- 1 matthias matthias   64 Apr 30 11:03  cities.txt
-rw-r--r-- 1 matthias matthias  128 May  7 10:58  cities_sorted.txt
-rw-r--r-- 1 matthias matthias    3 May  7 10:59  city_count.txt
drwxr-xr-x 1 matthias matthias 4096 Apr 30 11:36  cps1012
-rw-r--r-- 1 matthias matthias    0 May  7 11:05 'eight nine'
-rw-r--r-- 1 matthias matthias    0 May  7 11:05 'five six seven'
-rw-r--r-- 1 matthias matthias    0 May  7 11:05  four
-rw------- 1 matthias matthias 1524 May  7 11:05  history.txt
-rw-r--r-- 1 matthias matthias   14 May  7 10:51  message.txt
-rw-r--r-- 1 matthias matthias    0 May  7 11:05 'one two three'
```

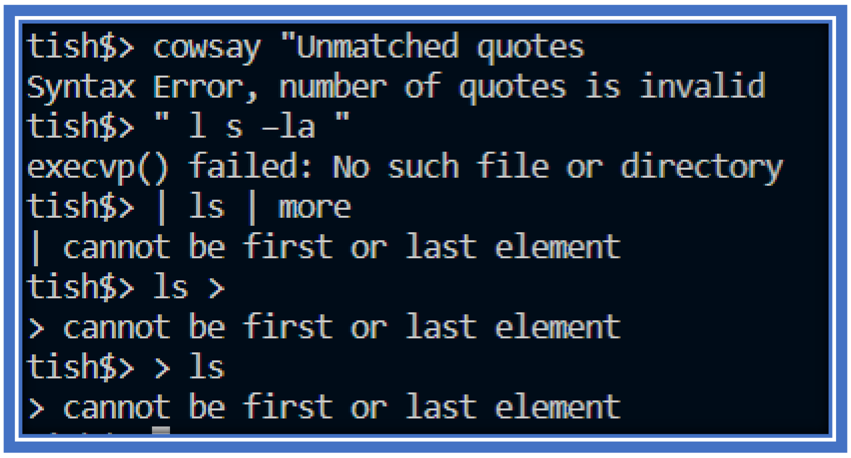


```
tish$> rm "one two three" four "five six seven" "eight nine" ; ls -la
total 4
drwxr-xr-x 1 matthias matthias 4096 May  7 11:05 .
drwxr-xr-x 1 matthias matthias 4096 May  4 13:27 ..
-rw-r--r-- 1 matthias matthias   64 Apr 30 11:03 cities.txt
-rw-r--r-- 1 matthias matthias  128 May  7 10:58 cities_sorted.txt
-rw-r--r-- 1 matthias matthias    3 May  7 10:59 city_count.txt
drwxr-xr-x 1 matthias matthias 4096 Apr 30 11:36 cps1012
-rw------- 1 matthias matthias 1587 May  7 11:05 history.txt
-rw-r--r-- 1 matthias matthias   14 May  7 10:51 message.txt
```



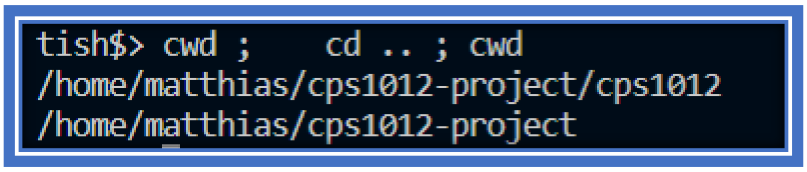
```
tish$> e"ch"o Hello, World!
Hello, World!
```

Error Handling:

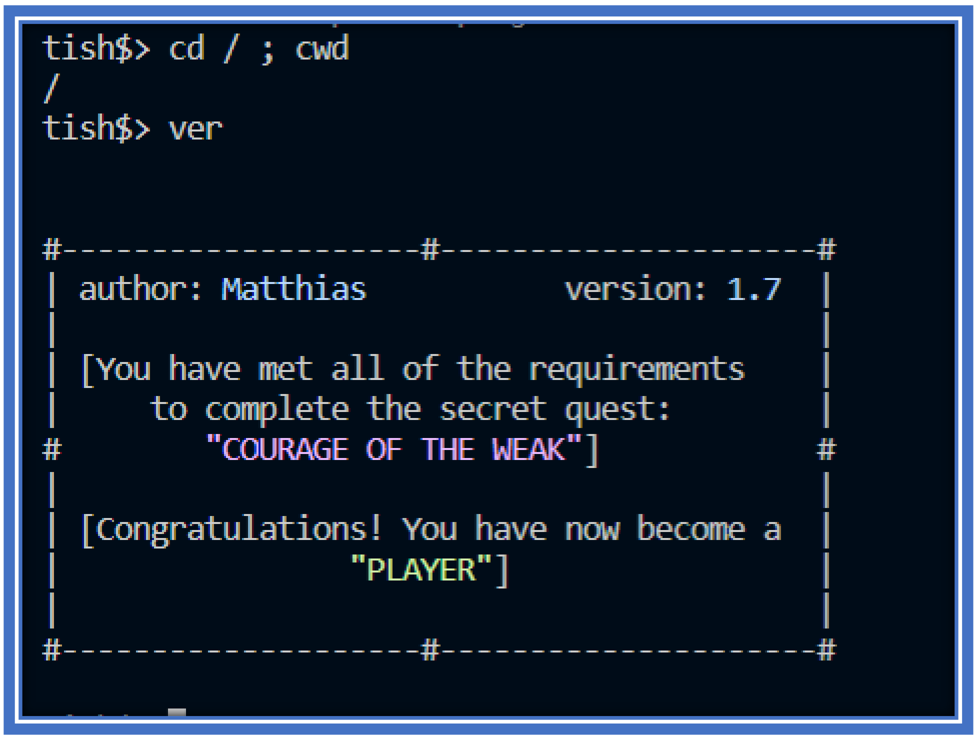


```
tish$> cowsay "Unmatched quotes
Syntax Error, number of quotes is invalid
tish$> " l s -la "
execvp() failed: No such file or directory
tish$> | ls | more
| cannot be first or last element
tish$> ls >
> cannot be first or last element
tish$> > ls
> cannot be first or last element
```

Builtin commands:



```
tish$> cwd ;     cd .. ; cwd
/home/matthias/cps1012-project/cps1012
/home/matthias/cps1012-project
```



```
tish$> cd / ; cwd
/
tish$> ver


#--------------------#--------------------#
| author: Matthias           version: 1.7 |
|                                         |
| [You have met all of the requirements   |
|     to complete the secret quest:       |
#        "COURAGE OF THE WEAK"]           #
|                                         |
| [Congratulations! You have now become a |
|               "PLAYER"]                 |
|                                         |
#--------------------#--------------------#
```