I have created a shell based on the conditions assigned in Codio to execute commands from the command line.

Task 1 - Print the Shell Prompt and Adding Built-in Commands

a. cd: changes the current working directory

It takes the first string and compares it with “cd”. Using the inbuilt chdir function, it changes the current working directory to the requested location.

```
if (strcmp(arguments[0],"cd")==0){
        chdir(arguments[1]);
```

b. pwd: prints the current working directory

It takes the first string and compares it with “pwd” and using the inbuilt getcwd function, it prints the current working directory.

```
else if (strcmp(arguments[0],"pwd")==0){
        printf("%s\n",getcwd(wd, wd_size));
 }
```

c. echo: is used to print **a** message and the values of environment variables

There are two conditions for this command. If the variable is defined and the string is stored, it needs to find that through the “$” sign. Otherwise it prints what is inputted after echo. getenv inbuilt function is used, if not it prints the information written after echo.

```
else if (strcmp(arguments[0],"echo")==0){
        i = 1;
        while(arguments[i] != NULL){
          if (arguments[i][0] == '$'){
            printf("%s ", getenv(arguments[i]+1));
          }
          else{
            printf("%s ", arguments[i]);
          }
          i++;
        }
      }
```

d. setenv: it sets an environment variable and it also stores the values in it.

```
else if (strcmp(arguments[0],"setenv")==0) {
        char* temp_val[2];
        temp_val[0] = strtok(arguments[1], "=");
        i=0;
        while(temp_val[i] != NULL || i == 2) {
         i++;
         temp_val[i] = strtok(NULL, "=");
        }
        setenv(temp_val[0], temp_val[1], 1);
       }
```

e. exit: it terminates the shell and is used as an inbuilt exit to terminate the shell.

```
else if (strcmp(arguments[0],"exit")==0){
        exit(0);
       }
```

f. env: prints the current values of the environment variables

It displays the predefined environment if any exists or otherwise displays the built in environment variable like pwd, etc. It searches for the variable using the getenv inbuilt function of the argument is not Null.

```
else if (strcmp(arguments[0],"env")==0){
        if (arguments[1] != NULL){
         printf("%s\n",getenv(arguments[1]));
        } else {
         char** env= environ;
         for (; *env; env++)
         printf("%s\n", *env);
        }
       }
```

Task 2: Adding Processes

I executed the following lines of code using execvp which runs the input argument as the built in commands. For example if the inputted argument is ls it will execute the command as if it was the built in command in a normal computer terminal.

```
} else {
    pid = fork();
    if (pid==0){
        signal(SIGINT, SIG_DFL);
        execvp(arguments[0], arguments);
        exit(0);
    }
```

Task 3: Adding Background Processes

I created a boolean variable to keep track of adding background processes. If there is & at the end of the command line I set the boolean variable to true. If not the boolean is set to False.

```
char* last_arg = arguments[i-1];
bool background_process = false;
if (strcmp(last_arg,"&")==0){
    background_process = true;
    arguments[i-1] = NULL;
}
```

Task 4: Signal Handling

I also created signal_handler to handle the signal call for Ctrl C. It terminates a foreground process. It is useful as to prevent the shell from quitting,

```
void signal_handler(int signum)
{
    if (command_p != -1){
        kill(command_p, SIGINT);
    }
}
```

Task 5: Killing off long running processes

I created a function named terminate_pro whose abbreviation is to terminate the process. This terminates the foreground process after 10 seconds have elapsed and if the process hasn't been completed in given time.

```
void terminate_pro(int time, int pid) {
  sleep(time);
  printf("Time out foreground process.\n");
  kill(pid, SIGINT);
}
```