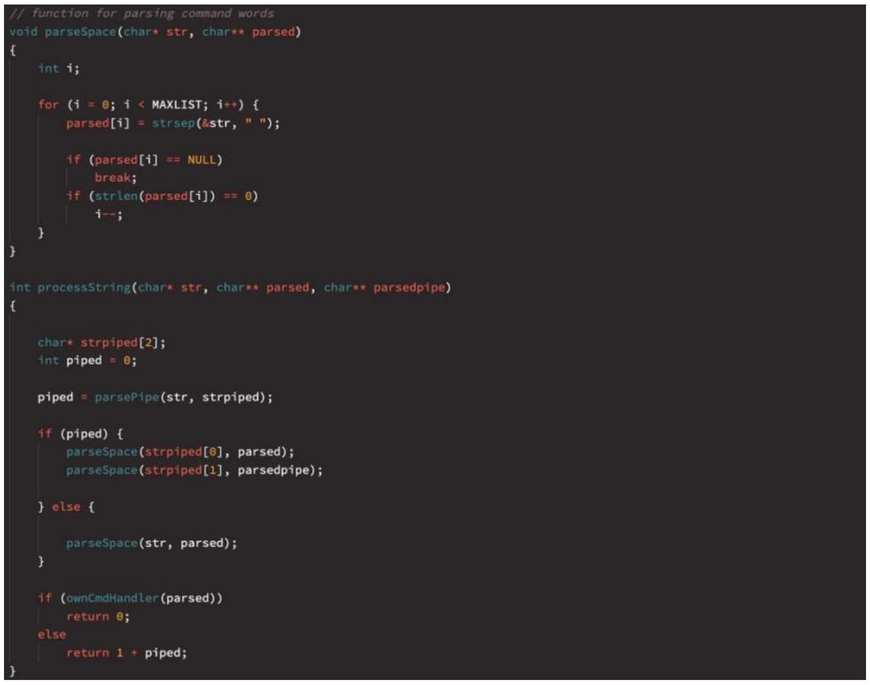
Simple Shell in C

This is an implementation of a simple custom shell in C. The shell performs some basic sets of instructions assigned in the tasks on codio to execute commands from the command line.

The program is divided into two categories: (1). The Functions (CD: Change Directory, LS : List) where the piped system command is excecuted and (2). The other functions demanded by the tasks

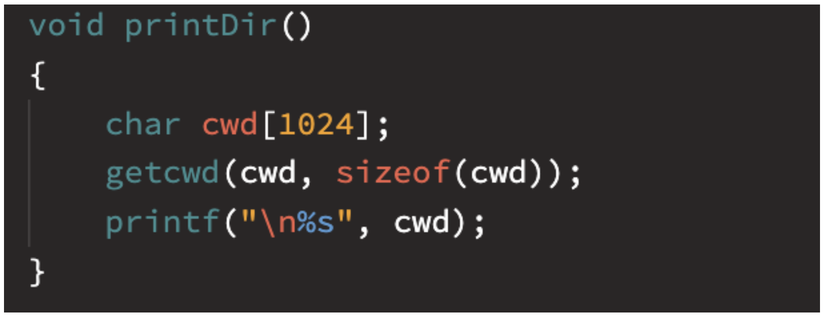
Task 1: Piped System Command and Adding Built-In Commands.

This function handles most of the input command in a parent and child process. The execArgs is a void function that takes a character pointer and an argument to parse all the inbuilt function to run the code and returns an error message if directory location doesn't exist. The implementation is below.



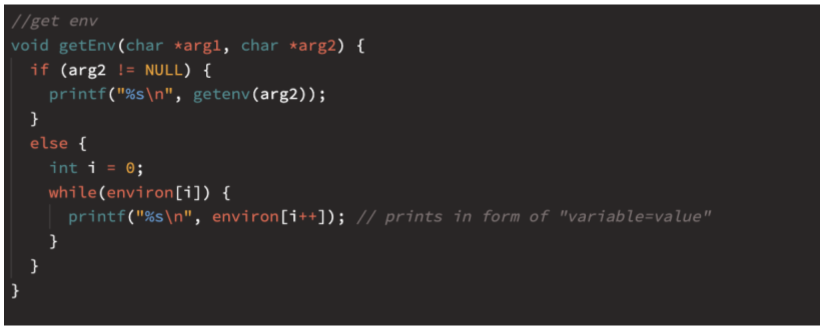
```c
// function for parsing command words
void parseSpace(char* str, char** parsed)
{
    int i;

    for (i = 0; i < MAXLIST; i++) {
        parsed[i] = strsep(&str, " ");

        if (parsed[i] == NULL)
            break;
        if (strlen(parsed[i]) == 0)
            i--;
    }
}

int processString(char* str, char** parsed, char** parsedpipe)
{

    char* strpiped[2];
    int piped = 0;

    piped = parsePipe(str, strpiped);

    if (piped) {
        parseSpace(strpiped[0], parsed);
        parseSpace(strpiped[1], parsedpipe);

    } else {

        parseSpace(str, parsed);
    }

    if (ownCmdHandler(parsed))
        return 0;
    else
        return 1 + piped;
}
```

Pwd : Prints the working directory. The printDir is a void function. It uses the inbuilt function getcwd function to print the working directory.



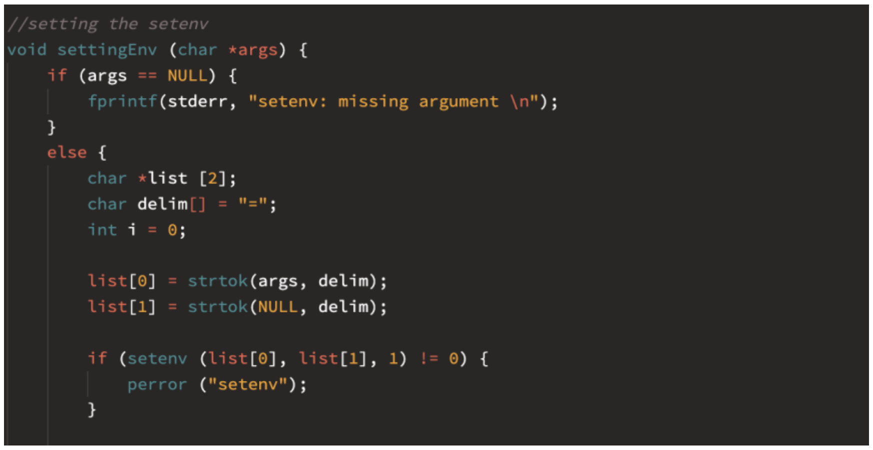
```c
void printDir()
{
    char cwd[1024];
    getcwd(cwd, sizeof(cwd));
    printf("\n%s", cwd);
}
```

Env: prints the current values of the environment variables. getEnv is a void function with a char pointer array of command line arguments. b. In the command line: If there is an argument after 'env' in the command line, the inbuilt function getenv is used to retrieve the value of the argument from the environment variables. Else, all the environment variable are printed to the command line.



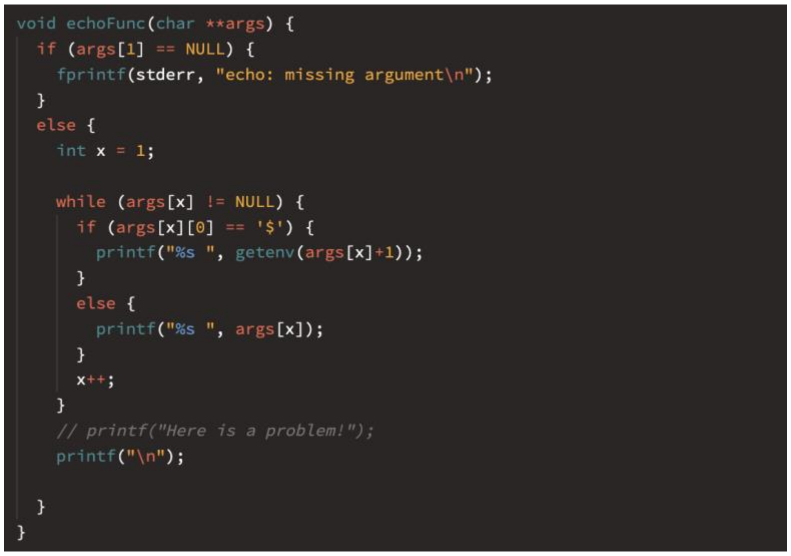
```c
//get env
void getEnv(char *arg1, char *arg2) {
  if (arg2 != NULL) {
    printf("%s\n", getenv(arg2));
  }
  else {
    int i = 0;
    while(environ[i]) {
      printf("%s\n", environ[i++]); // prints in form of "variable=value"
    }
  }
}
```

Setenv: sets an environment variable. setEnv is a void function with a char pointer array of command line arguments. It uses the inbuilt setenv function to set environment variables and edit them if they exist. The characters after "=" in the arguments after setenv in the command line is set as the value to the characters before "=" (the key). Returns an error message if inbuilt setenv fails.



```c
//setting the setenv
void settingEnv (char *args) {
    if (args == NULL) {
        fprintf(stderr, "setenv: missing argument \n");
    }
    else {
        char *list [2];
        char delim[] = "=";
        int i = 0;

        list[0] = strtok(args, delim);
        list[1] = strtok(NULL, delim);

        if (setenv (list[0], list[1], 1) != 0) {
            perror ("setenv");
        }
```

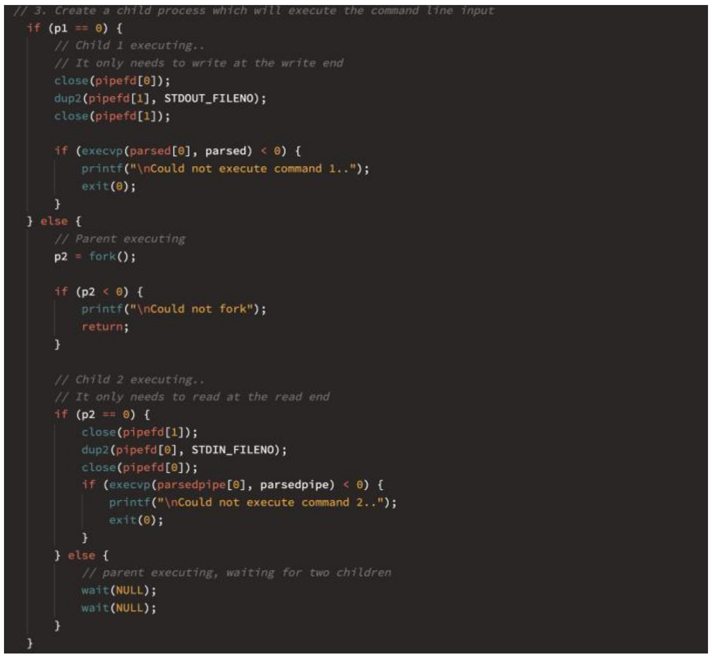
Echo: prints message and values of environment variables to the command line. a. echoFunc is a void function with a char pointer array of command line arguments. There are two conditions for this command: If the variable is defined and stored as an environment variable, on the command line, the variable is started with "$" and echo prints the value of the environment variable to the command line. The inbuilt function getenv is used to get the values for the environment variables. ii. Otherwise, the arguments after echo are printed to the command line. Returns an error message if getenv fails.



```
void echoFunc(char **args) {
  if (args[1] == NULL) {
    fprintf(stderr, "echo: missing argument\n");
  }
  else {
    int x = 1;

    while (args[x] != NULL) {
      if (args[x][0] == '$') {
        printf("%s ", getenv(args[x]+1));
      }
      else {
        printf("%s ", args[x]);
      }
      x++;
    }
    // printf("Here is a problem!");
    printf("\n");

  }
}
```

Task 2 : Adding Processes

The following lines of code create a fork to add a process. The function excvp is used to run the built-in commands implemented above.



```c
// 3. Create a child process which will execute the command line input
  if (p1 == 0) {
      // Child 1 executing..
      // It only needs to write at the write end
      close(pipefd[0]);
      dup2(pipefd[1], STDOUT_FILENO);
      close(pipefd[1]);

      if (execvp(parsed[0], parsed) < 0) {
          printf("\nCould not execute command 1..");
          exit(0);
      }
  } else {
      // Parent executing
      p2 = fork();

      if (p2 < 0) {
          printf("\nCould not fork");
          return;
      }

      // Child 2 executing..
      // It only needs to read at the read end
      if (p2 == 0) {
          close(pipefd[1]);
          dup2(pipefd[0], STDIN_FILENO);
          close(pipefd[0]);
          if (execvp(parsedpipe[0], parsedpipe) < 0) {
              printf("\nCould not execute command 2..");
              exit(0);
          }
      } else {
          // parent executing, waiting for two children
          wait(NULL);
          wait(NULL);
      }
  }
```