Assignment 4: Writing a Basic Command Shell

December 13

Fall 2018

Brandon Tran

Christian Campos

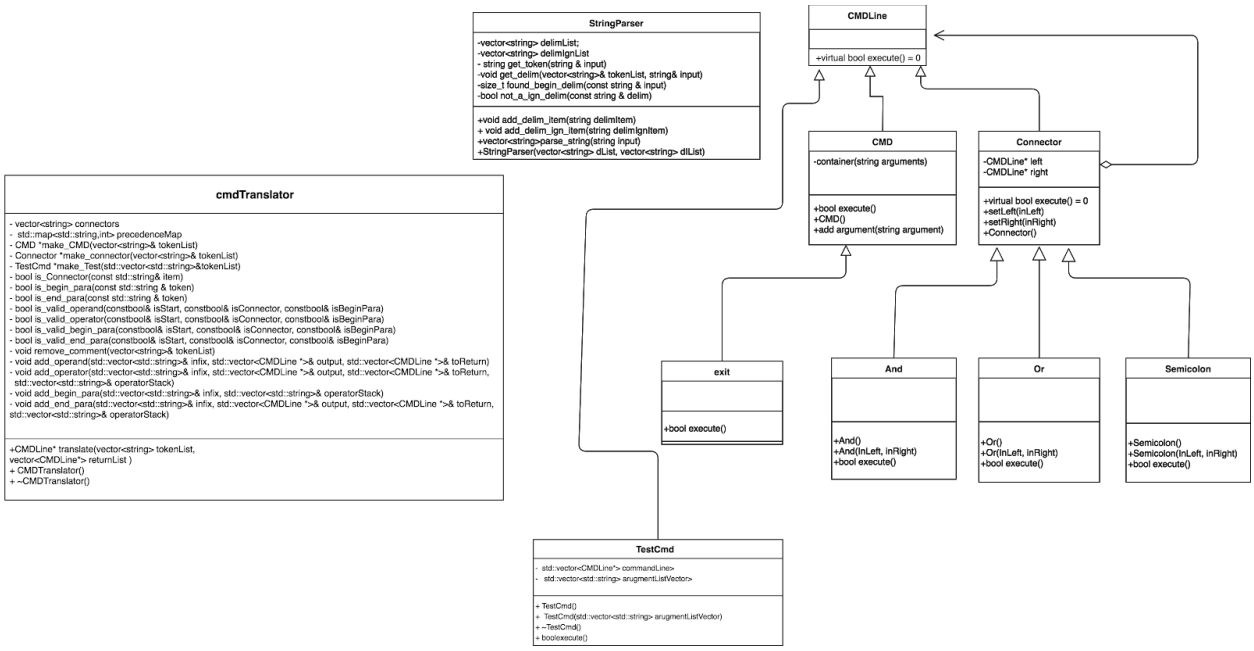
- Introduction:

  The main task of the assignment is to create a shell for the user to enter commands , arguments, and connectors in one line. Further, we are mainly using a composite pattern to implement the user interface. The commands and connectors will be implemented as hierarchy tree structure. The top of the hierarchy is CMDLine abstract class. The CMDLine base class will have virtual bool executed function for all the subclasses to inherit. The next layers of the trees are CMD and Connectors and then children of the CMD and Connectors are Exit,And, Or, and Semicolon. The CMD class will execute the appropriate commands commands using fork, execvp, and waitpid.

  Lastly, a Parser class was created to parse the user inputs from invalid inputs. During the assignment 2 sprint, we made new adjustments on the UML diagram. We added an another class called cmdTranslator. The main goal for the cmdTranslator is to convert string tokens into objects. With this conversion, we can use the token objects to call the connector's children (And, Or, Semicolon) to execute the boolean condition. In assignment 3 sprint, we modified our CmdTranslator to handle parenthese and created a testCmd class to check if a flag or directory or both exists. If the user wants to add parentheses with valid commands then our R shell will execute the commands based its precedence. Lastly, in the last sprint (assignment 4) , we extended the functionality of rshell by including the input redirection , output redirection , and double output redirection. The rshell properly handles multiple piping, redirections and can be combined with other connectors(And , Or, Semicolon,and etc).

  As the following the diagram demonstrates:

- Diagram:



- Classes/Class Groups:

➢ CMDLine (base)

The command line is the abstract base class. It will be the interface of the sub classes. A pure virtual function execute() will require all subclasses that inherit from the interface to execute a task, whether is is a composite or a leaf.

➢ CMD

The CMD class is a leaf to our composite structure. The CMD's purpose is to encapsulates one command and its arguments into an object. Example: If we have the command ls -a; pwd, then ls -a will be made into one CMD object and pwd will be different. We are treating a command input and its arguments as a single object.

➢ Connector

The connector class is composite class that contains two CMDLine base objects. It is a parent class which contains only valid connectors that separating two command line objects. The children are And , Or , and Semicolon by the

following assignment specifications. The commands should obey to the connector rules to decide what it should be executed.

- Exit

  The Exit class sole purpose is to exit the program when the user enters the exit command. It is a subclass that inherits from CMD class.

- And(&&)

  The And class is a child of connector. The And operation holds two base pointers which are data pointers. The left data pointer will executed first and it was able to execute properly, then the other data pointer will executed. The right data pointer will executed if and only if the left data pointer was able to complete its instruction(s).

- Or (||)

  The Or class is a child of connector. The Or operation inherits two base pointers from the connector. The Or operation will executed only if one of the base pointers has a valid command input. The right hand data will executed if and only if the left hand data failed.

- Semicolon (;)

  The Semicolon class is child of connector. The Semicolon separates two CMD objects from the input. The semicolon operation will always executed the next command.

- Parser

  The Parser class is to parse the user's input and convert to CMDLine objects. It's a class that doesn't inherits or composite the base class. Further, the Parser class will contain c string library strtok function to parse the spaces , connectors, commands from the terminal.

- cmdTranslator

CmdTranslator's main task is to convert string tokens into objects. With this conversion, we can use the token objects to call the connector's children (And, Or, Semicolon) to execute the boolean condition. Further, CmdTranslator is capable of handling the parentheses and test command in the user's input.

- TestCmd

The TestCmd's task is to add test command and symbolic equivalent bracket [ ] in our rshell. Our TestCmd class will use the stat command syscall to test if a file or directory exist. With the help of the stat function macros (S_ISDIR and S_ISREG) , we could determine the file or directory existence. Further, the test command and symbolic equivalent bracket [ ] can be combined with multiple connectors to write a complex bash command structure.

- Pipe

The Pipe class task is create a pipe object to use the pipe syscall function to pipe the input file descriptor and output file descriptor. In the execute function , it will receive two parameters that contained input file and output file. A local array of maximum of 2 were created to used for reading and writing the input and output files. If it's successful , the array will contain two new file descriptors to be used for the pipeline.

- Input Redirection

The Input Redirection class task is create Input redirection object to create to use the Input redirection (<) syntax. Input redirection execute function assigns the input file descriptor to open the file to read the flags only.The right child will get the filename from the argument list index 0.Then the left child will execute the input file descriptor and
output file descriptor

- Output Redirection

The Input Redirection class task is create Input redirection object to create to use the Output redirection (>) syntax. Output redirection execute function assigns the output file descriptor to open the file with flags to write only, check if the file exist , and if the previous file exists. A few modes were passed in the open function to restrict the user to do certain tasks in the file only.Then the left child will execute the input and Output file descriptor

Double Output Redirection

The Input Redirection class task is create Input redirection object to create to use the double output redirection (>>) syntax.Double output Redirection execute function assigns the double file descriptor to open the file to write to. The first parameter ; the right child will get the file name from
the argument list index 0.The second parameter restrict the user to append and write only. The third parameter will restrict the user from writing crazy stuff in the file descriptor.

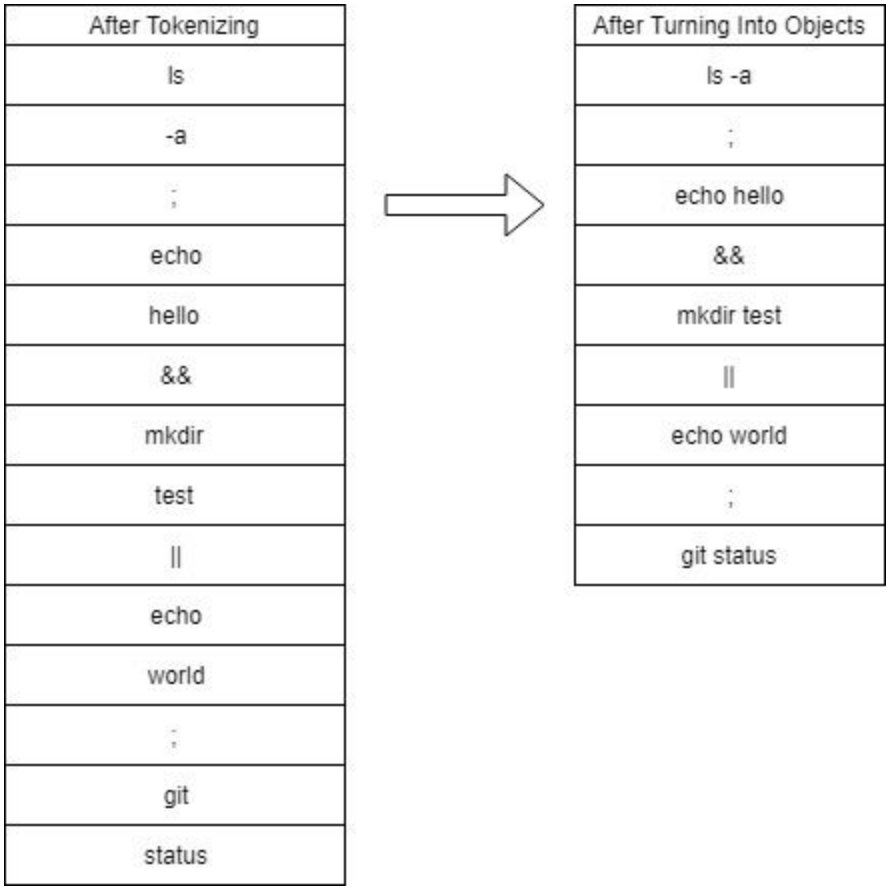
- Coding Strategy:

Brandon Tran - *Parsing the user's input*

Brandon's task will parse the command string the user enters. We are going to used the strtok function from the C standard libraries to parse the input. We are going to use strtok function to split the input from the client/user into strings tokens which will be turned into CMDLine objects. This will be implemented by the Parser class. For example: the client enters input,

```
$ ls -a; echo hello && mkdir test || echo world; git status
```

The input will then be parsed using spaces and connectors as the separators. After the input is tokenized, the tokens will then be created into CMDLine objects as the diagram shows below.

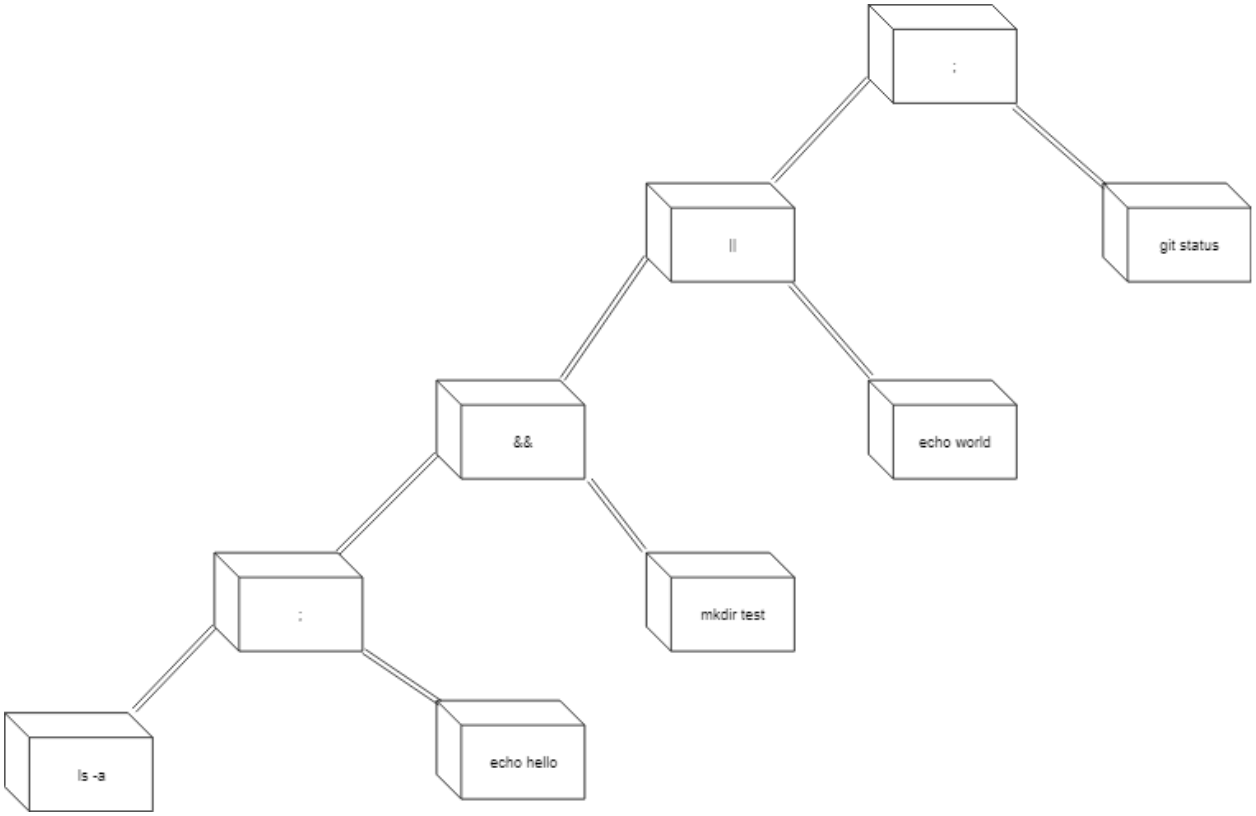


The CMDLine objects will be added from left to right ordering for its execution:

$ ((((ls -a; echo hello) && mkdir test) || echo world); git status)

Christian Campos - *Commands and Connector Tree*

Since the basic command shell is based on a hierarchy system, the first command and its arguments from the input will be the lowest level of the tree structure. For example: let's use the command input used in the parsing. The top node of the tree will be the last command or connectors that will be executed from the user's input. Once the execute() function is called, the root node of the class will execute() and traveled down to the lowest level of the tree which is "ls -a". The command "ls -a" will be executed and returns a boolean value to transerve one level up towards the connectors , ";".Then a boolean value will returned based on what connectors command interacts. If the bool value returns true from "ls -a" , then execute() will then travel to the right side of the tree to perform execute on the command " echo hello". As a result, it will recursively transverse to the top node of the tree until all the nodes in the tree are executed with the true boolean value. The following recursive and tree structure can be shown in the diagram below:



- Roadblocks:

  A roadblock that can hinder our project structures is the system calls. We don't fully understand how fork, execvp, and waitpid because we never used system calls before. However, we will thoroughly read multiple articles and watch youtube videos to help us comprehend the unix functions.

  An another potential roadblock that can occur is parsing the user input with strtok function. Since our previous projects we were mostly creating our own parse for the user input, strtok is new to us. Nevertheless, we will read c++ documentation and articles, and watch youtube videos on strtok function. "Zombie" children can be a potential roadblock because our program runs on a continuous loop and we create multiple children process

id for every new command input from the user , but our program has perror for every sys call. However, we will keep an eye on any “Zombie” children lurking in our program.