CS 100 Assignment 1: Design

Daniel Stinson-Diess and Kennen DeRenard

October 20th, Fall 2017

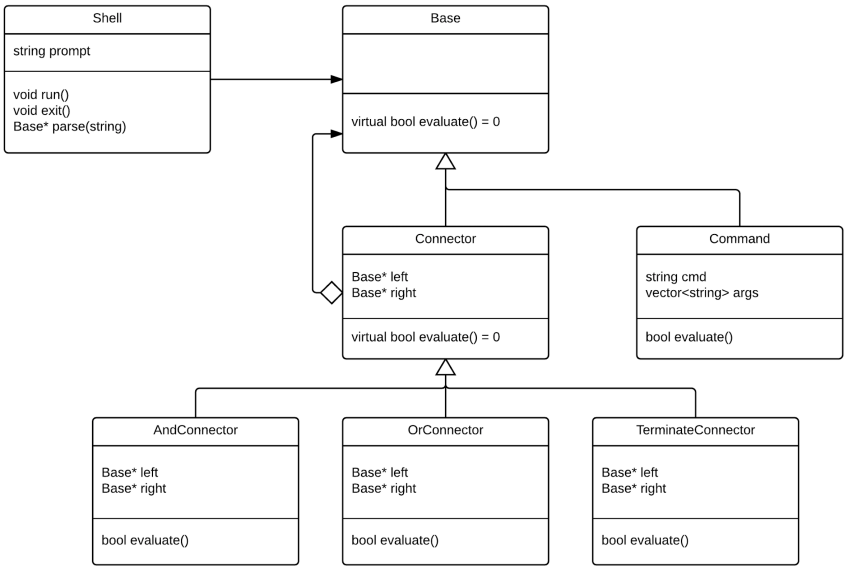
1 Introduction:

Our project will be to create an R'Shell, which is a C++ program that can take in Bash commands and execute them in a similar way as done in the Bash Shell. Some discrepancies between R'Shell and Bash are such that directory based commands such as "cd" cannot be executed.

Several assumptions that we shall make throughout the assignment are listed below:

- There are no limits to the amount of commands that may be chained.
- All chaining commands or comments should be preceded by a space, other than the semicolon (;), which MUST be immediately after a command, and a space will be after it.
- Anything that appears following the # character should be considered a comment. This may or may not have a space following the # character.

2 Diagram:



3 Classes/Class Groups:

We plan to implement our shell starting with the Composite Pattern. We are choosing this pattern because it allows us to break up commands passed in

into various connector components, where each of the components are composed of other conectors or single commands. The composite class group will consist of: Base, Command, Connector, AndConnector, OrConnector, and TerminateConnector classes.

- Base: the abstract class for the composite class group. It will consist of one pure virtual method, execute, which returns a bool, depending on if the execution succeeded or not.
- Command: the leaf class for the composite class group. It's data members will contain the main command executable, and a vector of the arguments. The vector includes the arguments for that single command. The implementation for its inherited method (execute) will involve executing the arguments contained in the vector using syscalls.
- Connector: the component used for inheritance with the connectors. This class will consist of 2 protected Base pointers, left and right. These data members will be protected because the connector classes will inherit them, and they will be Base pointers so that the children can be components or leaves (commands or operators). The execute method will not be implemented.
- AndConnector: the logical connector && (AND) used in shell commands. This class will implement the inherited method execute by performing execute on its left data member. If this operation returns true, then it will perform execute on its right and return that result, otherwise, it will return false (result of left).
- OrConnector: the logical connector || (OR) used in shell commands. This class will implement the inherited method execute by performing execute on its left data member, If this operation fails, then it will perform execute on its right and return that result, otherwise, it will return true (result of right).
- TerminateConnector: the connector ';' used in shell commands. This class will implement the inherited method execute by performing execute on its left data member and then performing execute on its right.

One more class, the Shell class, will be associated with our Base class. Shell will be in charge of starting and exiting the prompt, parsing the commands, and running the whole program in general. It will consist of string data member containing the prompt; a run method, which starts the shell and runs commands; an exit function, which is called if the user inputs a special command (exit); and a parse method, which parses an inputted command and returns a Base pointer to the component of the composite tree generated, consisting of the individual commands.

4 Coding Strategy:

We plan to evenly distribute the User Stories we create on the Kan-ban Board so that each of us will have an even workload and output towards this project. Another feature of Github we plan to utilize is effective branching for development and testing to ensure that the master branch is always a stable release of R'Shell.

Daniel will be in charge of implementing the argument / string parsing feature, and Kennen will design the command execution feature using syscalls (*fork, execvp,* and *waitpid*). We will share the workload for designing the class structure / hierarchy, as well as sharing the work for testing so that each of our segments are thoroughly evaluated and bug-free. All of our testing will also be executed on the UCR Hammer Server to verify that the code is ran on the same environment as the grader.

When it comes time to integrate our segments together, we will define our interface and combine our sections to create that interface. Our separate implementations will come together so that the results of the command parsing in the Shell class will be turned into a composite class structure, and the parsed command will be executed.

5 Roadblocks:

The initial roadblocks will be becoming familiar with the Boost library and C++ syscalls, but once implementation is overcome, the roadblocks should be less significant. To prevent glaring bugs within our project, we will be thoroughly testing each others code. Any other bugs that occur will be listed on Github as issues.