Assignment 1: Design
Fall Quarter 2018
October 28, 2018

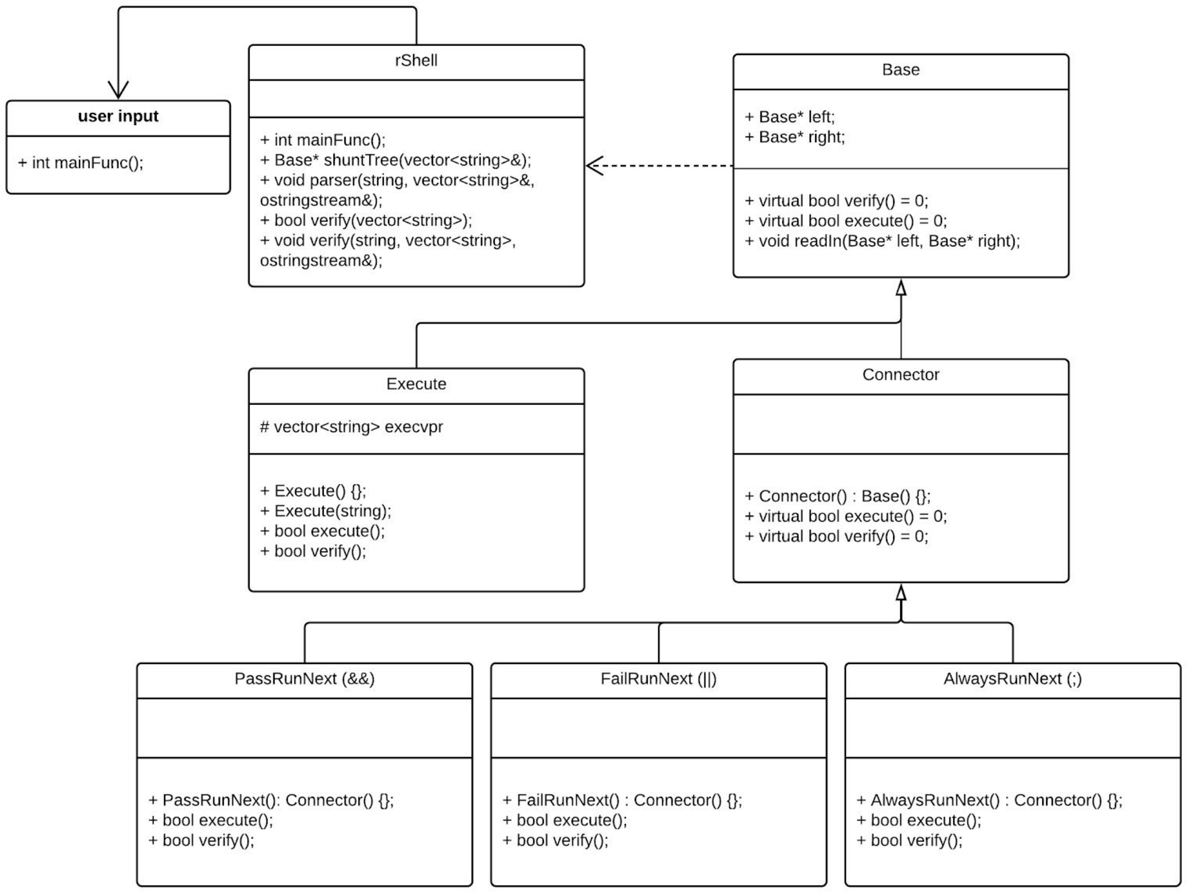
Authors:
Steven Tran,
Ted Kim

Introduction

The purpose of this design is to create a command shell called “rshell”, structured and coded completely in C++. This command should be able to read in commands inputted by the user, and if they are appropriate, will be executed. In order to separate the commands when multiple commands are being entered, the user should be able to use ||, &&, or ;, the only appropriate connectors that would allow the user to do so.
An example of how it would look in the command shell would be :

```
$ executable [argumentList] [connector] [cmd] . . .
```

Diagram


user input
+ int mainFunc();
rShell
+ int mainFunc();
+ Base* shuntTree(vector<string>&);
+ void parser(string, vector<string>&, ostringstream&);
+ bool verify(vector<string>);
+ void verify(string, vector<string>, ostringstream&);
Base
+ Base* left;
+ Base* right;
+ virtual bool verify() = 0;
+ virtual bool execute() = 0;
+ void readIn(Base* left, Base* right);
Execute
# vector<string> execvpr
+ Execute() {};
+ Execute(string);
+ bool execute();
+ bool verify();
Connector
+ Connector() : Base() {};
+ virtual bool execute() = 0;
+ virtual bool verify() = 0;
PassRunNext (&&)
+ PassRunNext(): Connector() {};
+ bool execute();
+ bool verify();
FailRunNext (||)
+ FailRunNext() : Connector() {};
+ bool execute();
+ bool verify();
AlwaysRunNext (;)
+ AlwaysRunNext() : Connector() {};
+ bool execute();
+ bool verify();

Classes / Class Groups

rShell Class : The rShell class contains the mainFunc() function where it will initialize the command line and take in arguments. We will store the commands in a tree format using the shuntTree function after the entire command is parsed through the parser function and verified to be correct through the verify functions. Both the parser and verify functions will also have a seperate function with identical names to test the specific functions of every possible type of input the command line can get.

Base Class : The base class holds all the members and virtual functions used throughout all of its inherited classes.

Op Class : The op class is a base class deriving from the Base Class and is a framework for the PassRunNext, FailRunNext, and AlwaysRunNext classes. As it has a pure virtual function determine(), it will not directly be used to create objects.

Execute Class: The execute class is a class used to execute the commands stored in a vector using the fork(), execvp(), and waitpid() commands.

PassRunNext(&&) Class: A PassRunNext object derives from the Op class and will be created when the parser() function reads in the "&&" connector.

FailRunNext(||) Class: A FailRunNext object derives from the Op class and will be created when the parser() function reads in the "||" connector.

AlwaysRunNext(;) Class: An AlwaysRunNext object derives from the Op class and will be created when the parser() function reads in the ";" connector.

Coding Strategy
Our strategy for development and division of work will be utilizing the Kanban strategy. We will have a list of tasks in our "product backlog". Each task will be given a timeframe that will be reasonably calculated, and each task will be pulled from the product backlog into a section that indicates that the task is currently "in progress." The progress section will only take in a limited number of tasks as to avoid an unmanageable amount of work to do in a given timeframe. The timeframes and tasks will also be accommodating the schedule of each individual partner, making sure that each partner is given their share of work to do, as well as making sure it is aligned with the rest of their tasks that they need to accomplish, whether it be for this project or other scheduled tasks.

Roadblocks
With a limited number of tasks that would be able to be put in the "in progress" section of the Kanban strategy, if either partner fails to follow the schedule organized through the Kanban strategy, then the "in progress" section will either overflow, or cause a jam in the flow of work being done as tasks are being pulled from the product backlog. As methods to avoid such traffic in the flow of work, a good way to avoid such conflicts at the project's inception would be to make sure that each task is given as accurate of a "time limit" as possible, to ensure that the work can indeed be done in the timeframe given. Also, as a means to keep the number of tasks in progress at a minimum, one would help the other if other roadblocks are being encountered in the process of development.