Assignment 1: Design

October 20, 2017
Fall 2017

Andrew Lvovsky
Ryan Villena

Introduction

We will be creating a command shell that reads in user input and parses that input based on commands, flags, pathnames, and strings (for tokens after the command “echo”). The main focus (from a design point of view) will be to incorporate a composite pattern for the token class, which will have children classes in charge of tokenizing certain phrases. We will also have a string tokenizer that takes a string of input and splits it into separate tokens, which then get funneled into their specific tokens. The manager class will be the main class that the user will interact with. We will also need a logic operator manager when the user enters multiple commands with AND and OR.

Classes and Class Groups

The user will interact directly with the Manager object. The Manager object must parse the user’s input, determine which operation to carry out (and with which parameters), and execute it.

To parse the user’s input, Manager will use a String Tokenizer (or StrTokenizer), which will take as input the complete command line and tokenize the input. (The logic that StrTokenizer uses is beyond the scope of this project, but it essentially utilizes an internal pre-initialized state machine to determine what kind of input becomes what kind of token.) When it does this, it will create tokens of either the Cmd_Token, Flag_Token, or Connector_Token classes. All of these are mere extensions of the Token class, which in turn only specifies two getter functions - one function for reading the category of the token and one for reading the actual contents of the token.

Manager will also find SC_Delimiter useful. SC_Delimiter (short for Semicolon Delimiter) is a solution for the case in which multiple commands are taken into the same command line delimited by semicolons. SC_Delimiter will produce a queue of strings, each of which can be parsed as if they are separate command-line instructions.

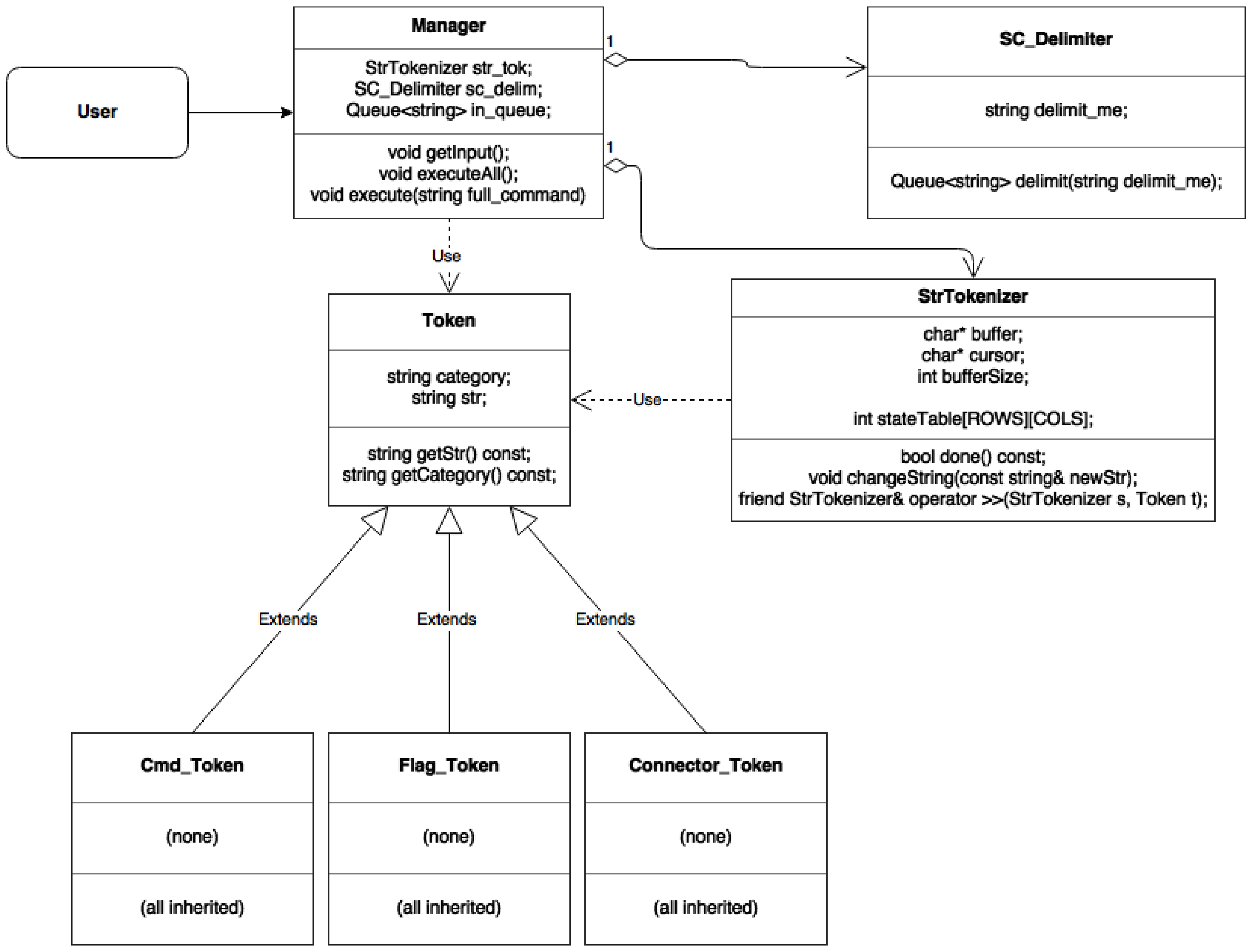
Coding Strategy

Since Ryan has designed functional StrTokenizer and Queue classes in a previous C++ data structures course, he will be in charge of the StrTokenizer for this project, as well as the SC_Delimiter class. Andrew will implement the Token class and all of its children. Both Ryan and Andrew will collaborate on the Manager class and the flow of control therein, as the flow of control seems to be one of the more daunting aspects of this project.

Roadblocks

We feel that system calls, such as *fork* and *execvp*, will take some time to get used to. We have seen there are resources and tutorials available for the aforementioned commands, so if any issues arise, we will refer to them. Another issue we might face would be how to properly use the composite token class, since this is our first big project that is implementing that design pattern. We just have to remember that the composite class would simplify managing tokens.

UML Design



User
Manager
StrTokenizer str_tok;
SC_Delimiter sc_delim;
Queue<string> in_queue;
void getInput();
void executeAll();
void execute(string full_command)
1
1
SC_Delimiter
string delimit_me;
Queue<string> delimit(string delimit_me);
Use
Token
string category;
string str;
string getStr() const;
string getCategory() const;
Use
StrTokenizer
char* buffer;
char* cursor;
int bufferSize;
int stateTable[ROWS][COLS];
bool done() const;
void changeString(const string& newStr);
friend StrTokenizer& operator >>(StrTokenizer s, Token t);
Extends
Extends
Extends
Cmd_Token
(none)
(all inherited)
Flag_Token
(none)
(all inherited)
Connector_Token
(none)
(all inherited)