Tyler Jackson
5/5/2015
Software Security
Final Project

Final Project Report

My project can be run in Visual Studio. It takes 5 different phonebook commands LIST ADD DELETE, EXIT, and phonebook - -help. The ADD commands can be called with ADD <name> <phonenumber>. This will add this person and their phone number to the phonebook. It converts all name input to hold the last name first, so that the numbers will be stored alphabetically in the phonebook. You can delete an entry from the phonebook with the DEL <name> command or the DEL <number> command. The names get structured so that if the same name is input in different formats it will be recognized as the same. For example, Tyler Jackson and Jackson, Tyler would be registered as the same person and the phonebook would not let you add both. Numbers however are not unique, so 2 names could have the same number. Therefore, when you delete an entry by a phone number it will delete the first entry in the phone book with that number. Also, if the same number is inputted with different formats this won't get reformatted. For example (303) 841-1825 and 303 841-1825 will be registered as different numbers. Therefore if you want to delete by phone number you need to provide it in the exact way it was input. This can be done by using the LIST command which lists all the entries in the phonebook to see how it is being stored. The EXIT command ends the program, and the phonebook --help command will print the list of commands for reference.

My program starts by initiating a while loop that only ends when the user types EXIT. This will end the program. It prompts the user for a command, and before doing anything it checks to see if this command follows the command syntax (i.e. begins with ADD, DEL, LIST, etc.). If the user provided an invalid command it will notify the user and await the next attempt at a valid command. Assuming the user issues a valid command the program will make a call to a PhoneBookManager class that will handle the bulk of the logic for the application. This class will hold a Dictionary of name-number key-value pairs that it reads in from a file upon construction. The file holds all entries in the current phonebook. This file will persist through multiple runs of the program. It also has the power to ADD an entry and DEL an entry as mentioned above. If asked to add an entry it makes a call to a utility class that I wrote to validate the input. This class, InputValidator, will match the input to a regex string using .NETs Regex Class and the IsMatch function inside a function GetNameAndNumber. If the input matches against the regex string I created then it will be deemed valid input. Then the PhoneBookManager will make a call to a GetName function that will format the name correctly to be stored in the phonebook. The number is taken as is. An entry will be added and the Phonebook text file will be updated. Upon a DEL call the PhoneBookManager will make a call to the InputValidator GetNameOrNumber function. This can accept either a name string, or a number string, and using Regex will validate that it is proper input, and return

either the formatted name or the number as is. This will then be used to look up the entry by name (key) or the first entry to match the number (by value) from our dictionary of entries. Assuming this entry exists it will delete it. If it doesn't exist then it will just notify the user that the entry does not exist and await the next command.

The regex string is separated into 2 main parts. The first part handles the name input validation and the second part handles the number. The name string has a few different rules that enable it to take a bunch of different first name, middle name, last name, and suffix (Jr.,Sr.,1st,2nd, etc.) combinations. The number regex string takes any combination of U.S. number and any of the international numbers from the preview list. I tried to name each logically separate rule and separate them with OR operators in the string to make it easier to debug. To make it easier to test I wrote up a bunch of potential inputs and made unit tests so that I could mess with the regex till all the tests passed.

Storing the entire phonebook in one text file isn't scalable, but for the sake of this project it was fine. The way the regex is broken up into different rules connected by OR operators makes it easily maintainable. To add a rule for a new input you can just add another clause to the OR chain. It is also easy to add new commands to the app making it scalable in that sense. Simply add a new if else statement in the program file and then add its functionality to the PhonebookManager and its done. Also, by separating the regex out into a utility class it could be reused on other projects. If the application was bigger we would probably separate this out into a utilities project that could be referenced when needed.