Peter M Piper

An interactive and demonstrative program that implements a Hash Table and explores different hashing functions and collision resolution.

Data Structures 2

Hash

**Program Description and Design**

This program implements a user authentication system using a hash with two hashing functions. The hash supports linear and quadratic probing. Users are stored as a user data type that includes an email, a role and a hashed password. The email is used as the key for the users hash table with a simple ASCII hash of the sum of the characters. The password is hashed using the MD5 hash algorithm.

When the program is initially run, the user is prompted to enter an admin username and password. A password file is generated with the user information. This file should be stored in a secure directory to prevent tampering by unauthorized users. The format of the password file is as follows:

Peter@gmail.com:1:893b56e3cfe153fb770a120b83bac20c

The email, role and password are separated by a colon. The password is stored in hashed format. The role is 0 for regular users and 1 for admins.

If the password file exists when the program is started the users will be automatically imported.

Once logged in a user can change their password, run a performance test, or log off. Admin users are also able to delete, add and show user accounts.

The performance test creates two hashes, one each with linear and quadratic probing. It runs a series of tests for insert, search (existing accounts), search (non-existing accounts) and delete. It calculates the number of collisions for each operation and displays a comparison between the hashes.

**Future Improvements**

* Additional account roles, e.g. super admin
* Role changing
* Guest login
* Invalid login attempts
* Forget password reminder
* Graphical user interface

**Test Plan**

**Reason for Test Case \_\_\_\_\_\_\_\_ Input Values\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_Expected Output**

Create initial admin user address: Peter@gmail.com,

password: Japan1 Admin account created

Logging in – user does not exist address: foo@bar.com

password: baz Incorrect username and password

Logging in – admin user –valid password address: Peter@gmail.com,

password: japan1 Welcome back admin!

Logging in – admin user –invalid password address: Peter@gmail.com

password: baz Incorrect username and password

Add new user – non existing address: foo@bar.com

password: baz User account created

Add new user – existing address: Peter@gmail.com

password: Japan1 Account already exists

Delete existing user – non existing address: foo@bar.com

admin password: Japan1 Account not found

Delete existing user – existing address: foo@bar.com

admin password: Japan1 User account removed

**Reason for Test Case \_\_\_\_\_\_\_\_ Input Values\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_Expected Output**

Display all existing users in a readable format enter Show users U:Foo@bar.com

A:Peter@gmail.com

Logging in – non-admin user address: foo@bar.com

password: Baz Welcome back!

Change password – old password matches old password: Japan1

new password: Japan2

confirmation: Japan2 Password has been changed

Change password – old password does not match old password: Japan3

new password: Japan2

confirmation: Japan2

Old password incorrect. Password not changed.

Change password – confirmation matches old password: Japan1

new password: Japan2

confirmation: Japan2 Password has been changed

Change password – confirmation does not match old password: Japan1

new password: Japan2

confirmation: Japan2

New password and confirmation do not match. Password not changed.

**Reason for Test Case \_\_\_\_\_\_\_\_ Input Values\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_Expected Output**

Performance Test select Performance Test Displays results of performance test

Creates two hashes

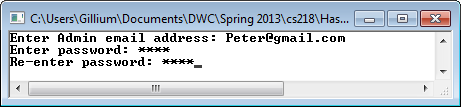
One with linear probing

Other with quadratic probing

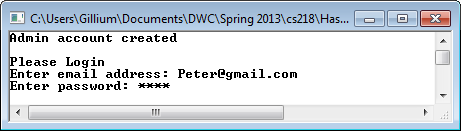
Runs tests for insert, search and delete

**Sample Outputs**

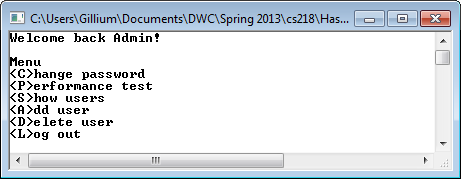
Initial Run



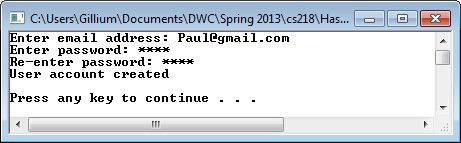
Log in



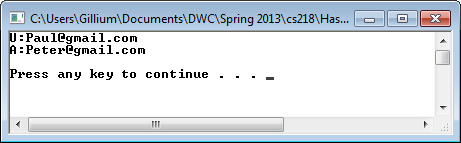
Menu



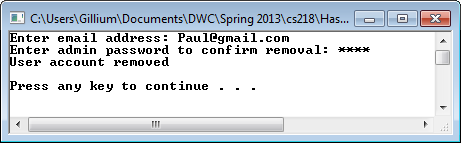
Add user



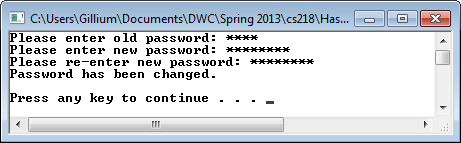
Show users



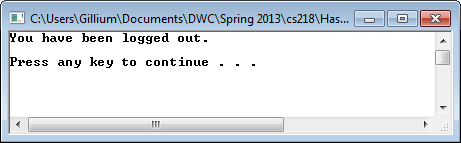
Delete user



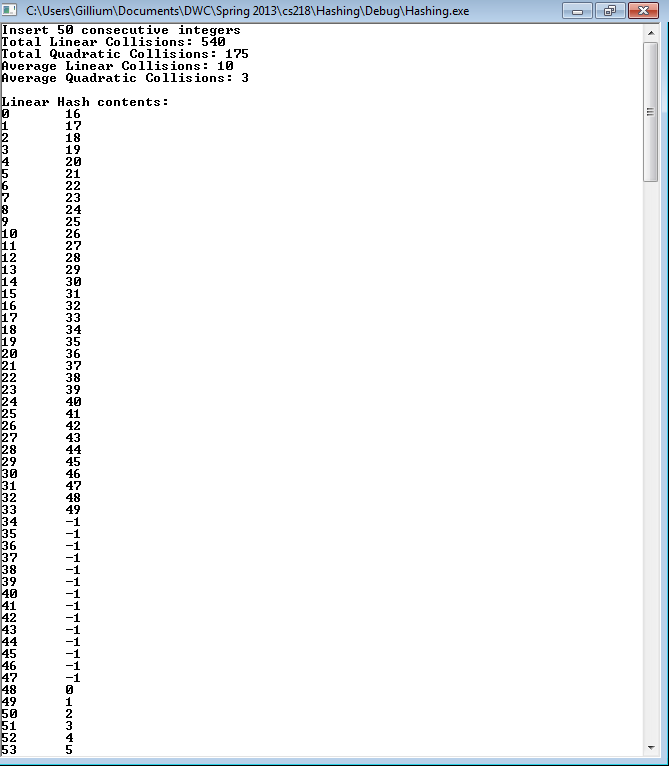
Change password



Log out



Performance test



**Analysis**

The results from the performance tests show the following:

Insert 50 consecutive integers

Total Linear Collisions: 540

Total Quadratic Collisions: 175

Average Linear Collisions: 10

Average Quadratic Collisions: 3

Search 50 consecutive integers

Linear Searches successful: 50

Quadratic Searches successful: 50

Total Linear Collisions: 540

Total Quadratic Collisions: 175

Average Linear Collisions: 10

Average Quadratic Collisions: 3

Search 50 consecutive integers

Linear Searches unsuccessful: 50

Quadratic Searches unsuccessful: 50

Total Linear Collisions: 1475

Total Quadratic Collisions: 416

Average Linear Collisions: 29

Average Quadratic Collisions: 8

Delete 50 consecutive integers

Total Linear Collisions: 540

Total Quadratic Collisions: 175

Average Linear Collisions: 10

Average Quadratic Collisions: 3

For these tests 50 consecutive integers were used. The results for insert, search and delete for existing items indicate that the number of collisions are the same for both operations. However, quadratic probing results in a 68% reduction in the number of collisions versus linear probing. Searching for non-existing items shows an even greater reduction of collisions for quadratic probing. This is probably due to the reduced clustering that occurs. These results are dependent on the inputs used and further testing could be done with randomized inputs, for example.