Store

CIS-17B

May 1, 2023

Danielle F

# Table of Contents

[Introduction](#_Introduction) .………………………………………........3

[Development Summary](#_Development_Summary) .……………………………......3

[GitHub](#_Reference) ….……….……………………………………...5

[Sample Inputs/Outputs](#_Sample_Outputs:) .…………………………….......5

UML Chart ……………………………………………..8

Flow Chart .……………………………………………..9

Version Summaries ….………………………………….9

[Pseudo Code](#_Pseudo_Code) …………………………………………...11

[Reference](#_Reference) .……….…………………………………......13

## 

# **[Introduction](#_Table_of_Contents)**

This program simulates a Store with three multiple choice options to choose from.

# **Objective**

Login as a user and place an online order from a sushi restaurant.

# **Rules**

* A user must have an account to access the Store.
* To sign-up for an account:
  + Name must be at least 2 characters long.
  + Email must be at least eight characters long and must contain an ‘@’ and ‘.’
  + Password must be at least 7 character long, contain a capital letter, one special character (!, $, #), and at least one digit.
  + Must confirm password by retyping it in again.
* User must sign-in to their account correctly to take the Store.
* Every question must be answered once.

## **[Development Summary](#_Table_of_Contents)**

|  |  |
| --- | --- |
| Lines of code  (including h files) | 2,226 |
| Comment lines | 264 |
| Blank lines | 137 |
| Total lines of source file | 1835 |

This project covers chapters one through thirteen in the [textbook](#_Reference). The primary requirement for this project is that everything needs to be in classes.

My initial challenge was figuring out how the class hierarchy. The User class needed to write the records to binary files and text files. Admin is the only class with permission to read binary files. How does the Admin access the User class members? Aggregate? Inheritance? Polymorphism? My initial solution was to make the Admin class be the base class and have the User Class inherit the Admin. I made the Admin’s read permissions private, so the User class couldn’t access them, but something was off. In version 13, I made Admin inherit the User class. This flow seemed more natural. However my group felt inheritance over complicated the program, so in version 18 I changed Admin’s relationship with the User class. Admin aggregated an instance of User instead of inheriting. I personally think aggregation is less efficient because it takes more code for it to work with the User. Also, I think inheritance gives classes a more streamlined and direct relationship.

My second dilemma involved writing and reading a User class object to a binary file and a text file. User class initially had three strings, but I soon realized I needed to save the size of each string as well. I struggled with rewriting a specific record in both file types. Rewriting a record in its same location requires you to know the beginning bit location of that file. I had to read each record to find its location in the file. For the text file, I had to take into account how many extra characters I was adding when I wrote a label for each variable in the record. In a binary file, if the variable is an integer, then it will always occupy the same number of bits. However, when rewriting a text file if you change the number of digits or letters, then it throws everything off. I had to rewrite hiScore as three individual digits for it to maintain the same number of bits it previous had. I decided not to allow a user to change their name or password.

My third dilemma was trying to reduce the repetitive code in Admin. Admin was opening and reading the User’s binary file in three different functions, findByIndx, findByEmail, and getAllUsrs. My solution was to create a double pointer array of Users in Admin’s private members. Allocate memory for the array of User objects in Admin’s default constructor and make getAllUsrs to be the only function who read the User’s binary file and initialize the dynamic array of Users. I made findByEmail function compare the user’s input against an array object value.

My fourth dilemma came when I protected by repository. I struggled with getting my pull request to go through and one of my pushes to GitHub got messed up and my latest version was overwritten by a previous version. I had to zip my programs and move it out of the Git repo folder before I pushed an update. Once I stopped protecting my repo I didn’t have any more problems with old versions overwriting new code.

My fifth hurdle came when I saved each User’s answer to the store in the binary file. It was a nightmare. When reading the integer array of votes, I was looping the .read() and it was throwing everything off. I fixed the bug by removing the loop and reading each store answer individually.

My final hurdle was anytime I updated the Admin and User classes in either my group project, Yahtzee or in my store project, I had to update it the other two projects.

## [**GitHub Repository**](#_Table_of_Contents)

#### <https://github.com/koa2019/e-store>

#### **Sample Inputs:** 3, homer@simp.com, Homer!23, 1, 2, 3

#### [**Sample Outputs**](#_Table_of_Contents)**:**

Figure 2 Store Results Chart

Figure 1 main menu

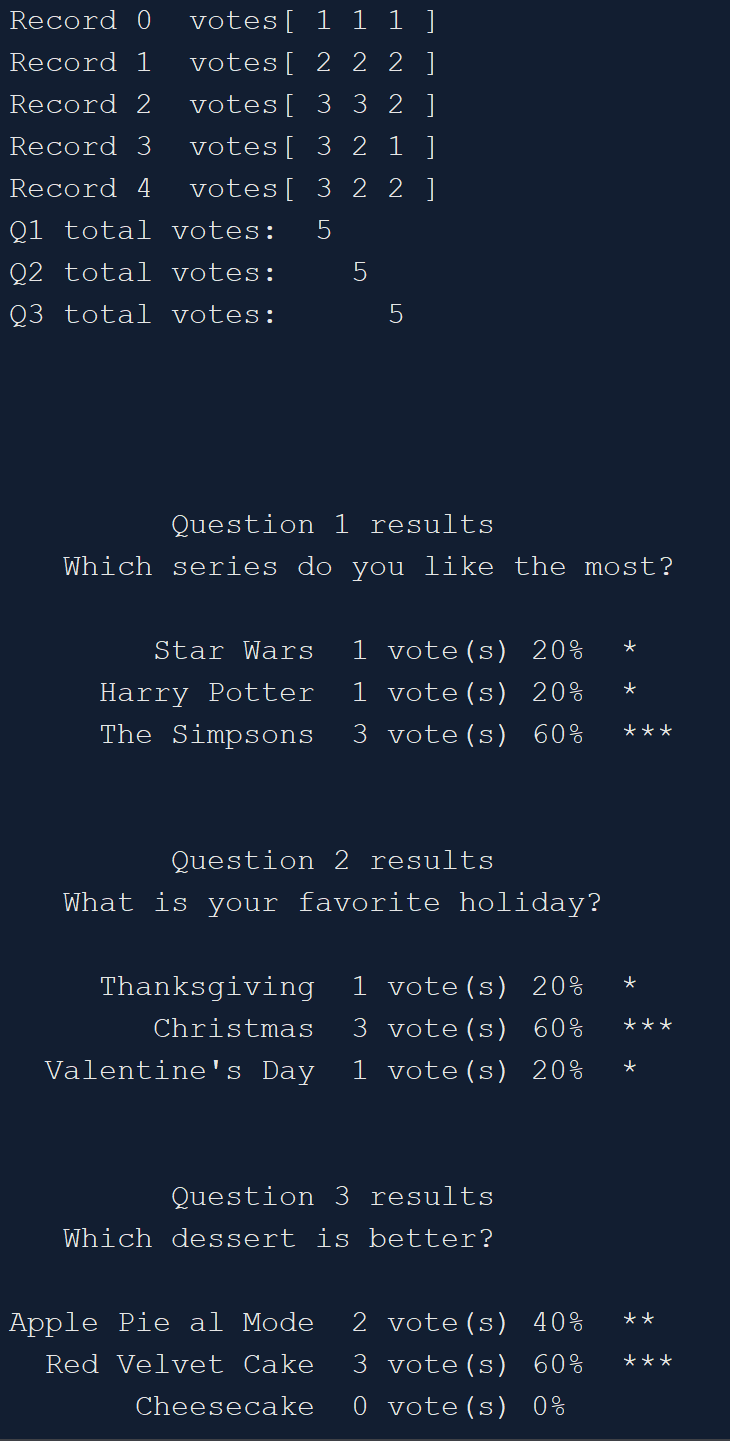
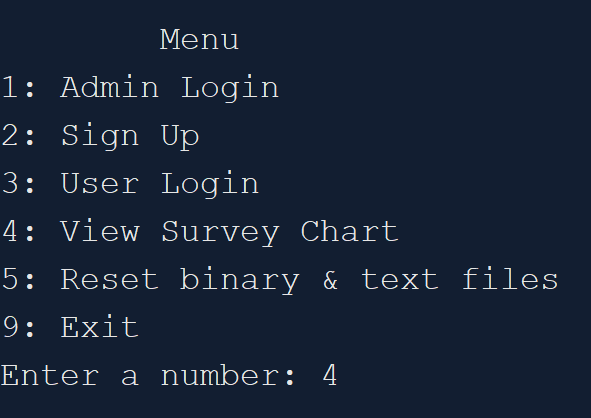


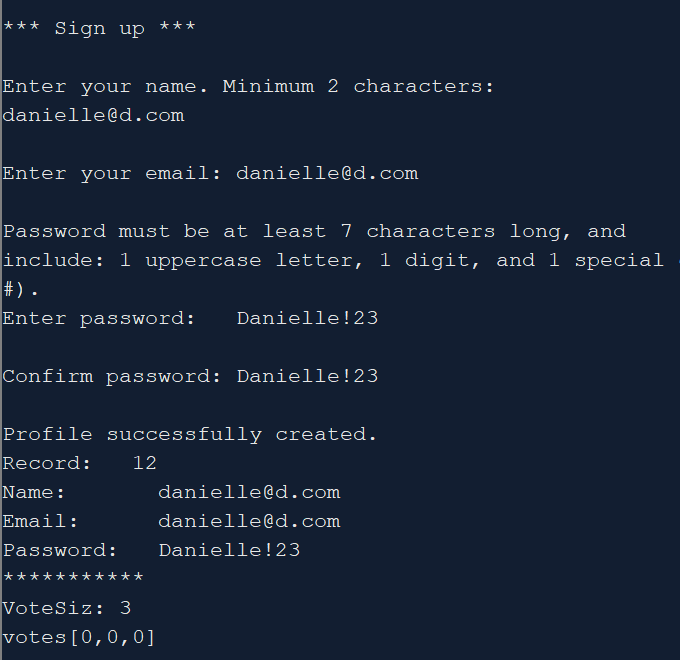
Figure 3 User Sign Up Demo 

Figure 4 User Login Demo

Text

Description automatically generated

Figure 5 Logged in User taking Store.

**Text

Description automatically generated**

Figure 6 Update User object in binary file

Text

Description automatically generated

Figure 7 Updated Store Results

A screenshot of a computer

Description automatically generated with medium confidence

## **[UML Chart](#_Table_of_Contents)**

The full version of this chart is in the store>docs>charts folder.

Diagram

Description automatically generated

## [**Flow Chart**](#_Table_of_Contents)

The full version of this flowchart is in the store>docs>charts folder.

Diagram, shape, polygon

Description automatically generated

### **[Version](#_Table_of_Contents) Summaries**

#### **Version 1**

* Based off of yahtzee\_v14\_adminClass.
* Admin inherits User.
* Added recSize & begnFile variables to Admin so it could hold their info when rewriting a specific record in the binary and text files.
* Admin::editVotes() rewrites the values for votes array in User’s binary file.
* Copied this folder to yahtzee\_v15.
* Rewrites 1 record in User’s text file after it rewrites User’s binary file.

#### **Version 2**

* Admin class aggregates User Class instead of Admin inheriting User.
* I copied my Admin & User classes from yahtzee\_v21 to here.
* Created string array with 1 question and 3 possible answers.
* Created a loop to mimic 5 people answering this 1 question.
* Created counters for each question's possible answer and reassigned int votes[] with the counters. i.e. votes[ opt1PickedNTimes, opt2PickedNTimes, opt3PickedNTimes].
* Passed the questions and answers string arrays to a print function.
* Created prntChart() that displays the voting results as well as a histogram with stars to represent how many votes each option received.

#### **Version 3**

* Moved string qusAns[] and int votes[] to a structure above main(). It won't work in Question.h
* Created 1 instance of Question structure and made sure it still ran correctly.
* Changed prntChart() to accept a structure instead of 2 difference types of arrays.

#### **Version 4**

* Question: getStore() reads a text file and saves it to a static array of Question objects.
* Added Votes structure and aggregated an instance of it in Questions class.
* getStore() prints one question and its three possible answers, gets the user’s input, and saves their answers to an instance of a Votes object.

**Version 5**

* Update this version with store\_v1.4's User and Admin Classes.
* Had to create Store class and aggregate Question class because I skipped a step. I needed to create an array of Question objects in the Store class’s constructor instead of in Question.cpp.
* Store class creates array of Question and aggregates instance of Votes, then rewrites hiScore in start().
* Moved Votes class to User class public members. This way I can access votes functions without writing extra get() in User class.
* Printed Votes object in User::wrtVotes( ).
* Wrote Votes[] to User’s text file. Added 14 chars to charCount in reWrtTxt().
* Wrote Votes object to the User binary file and read it from binary correctly. It writes and rewrites to text file as well.
* You can't loop through votes while reading from binary cause it messes it up.
* I realized i wasn't calling setPwrd() inside of readBin\_setArray(). idk how it still worked? lol
* Case 4 in adminPortal() calls editVotes();
* Inside of editVotes() and delete() I added usrArr[ind]-> to readBin\_setArray(), so it reset usrArr[]
* Aggregated Votes QueSums[3] in Admin's variables to represent the total number of votes each question received.
* Changed hiScore to voteSiz
* Created 3 int array to hold the Store results in Admin.
* Added setQueSums(), printQueSums(), getChart()

**Version 6**

* Changed Que1Sum, Que2Sum, Que3Sum to Votes QueSums[3].
* Admin::printQueSums() aggregates Store, so I can print questions & answers.
* Prints each answer’s percentage in printQueSums().

### [**Pseudo Code**](#_Table_of_Contents)

1. **Parent: Admin class**
   1. Dilemma. Option 1: User class will be the parent and then the Admin and Player classes will inherit User?
      1. Admin & Player both use User class’s variables & functions, but how will the Player ask Admin to test their hiScore variable?
   2. Dilemma. Option 2: Admin is the parent, list wrtBinary() as private and then User inherits Admin?
      1. Admin doesn’t need hiScore when it creates a new object for itself, but it needs to be able to reassign hiScore.
   3. Admin object:
      1. Private members:
         1. Static int num records / Doesn’t hold value between runs /
         2. unique ID
         3. name
         4. password
         5. email
         6. hiScore ? Should this be in User or Admin?
         7. readBinary()
         8. checkHiScore()
         9. setHiScore()
   4. Read inputs for 1 Admin record:
      1. Confirm inputs
      2. Save 1 admin record to object
      3. Write the object to text and binary files.
      4. Print message if successful or not
   5. ONLY Admin is allowed to read binary files
   6. Admin login()
      1. Read first record in Admin’s binary file
      2. Save it to an object
      3. Test login
      4. Print message if login was correct or not
      5. If login is correct, then allow them access to admin only functions
   7. Read User’s binary file:
      1. User login()
         1. Find a record by email
         2. Save it to an object
         3. Test login
         4. Print message if login was correct or not
         5. If login is correct, then allow them to view their profile and play game
      2. Be able to delete/edit a record or member’s of record
         1. Accept an ID or record as an object AND score from User, test if it’s bigger than their current hiSCore and update it accordingly
2. **Child: User class**
   1. Create User Class. User and game classes will have to be combined at some point.
   2. User object:
      1. Private members:
         1. Record #
         2. name
         3. password
         4. email
         5. voteSiz
         6. votes[ ]
         7. dice array or vector
   3. Sign up for a new account. Read in name, email, password
      1. Confirm user inputs before saving their info to object
   4. Write and append each User to a binary file
      1. ONLY has permission to write to binary file
   5. Write and append each User’s to text file
      1. ONLY has permission to write to text file
   6. Play game as guest or as logged in user
   7. When they win they should send their ID or their record as an object AND their current score to Admin, so then Admin can test if it’s bigger than the hiScore saved to their record
3. **Store**
   1. Loop through each question.
      1. Print 1 question.
      2. Print 3 possible answers.
      3. Save each answer in Votes array.
      4. Rewrite votes array in binary file and text file.
      5. Confirm user’s profile is updated by reprinting their profile.
      6. Print a chart of total votes for each question.

## [**Reference**](#_Table_of_Contents)

1. Gaddis, Tony. *Starting out with C++: From Control Structures through Objects*. 9th ed., Pearson, 2018.
2. Lehr, Mark. “2023\_Spring\_CSC\_CIS\_17B · ml1150258/2023\_spring\_csc\_cis\_17b.” GitHub, 2023,<https://github.com/ml1150258/2023_Spring_CIS_CSC_17B>.