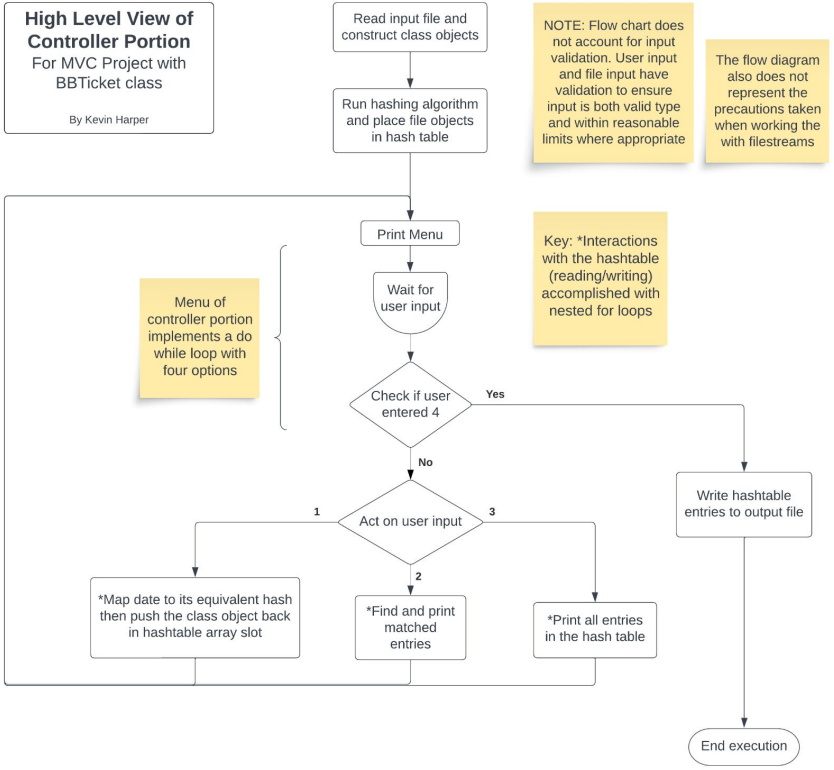
Intermediate C++: MVC Project

(User-Interactable Class Object Storage & Retrieval System)

Anthony Mauro and Kevin Harper

Implementation from a high level...



The project implements both the model and controller portion of the MVC framework

This is split amongst two compile modules:

- main.cpp
- customClass.cpp/.hpp

Makes use of:

- <iostream> // std streams
- <fstream> // ifstream and ofstream
- <vector> // hash table indeces
- “customClass.hpp”

Model class .hpp file

- Includes forward declarations of private and public data members
  - Private: Data members
    - string customerName;
    - int customerAge;
    - float ticketPrice;
    - bool fanStatus;
    - string purchaseDate; // MM/DD/YYYY
  - Public: Class methods
    - BBTicket() = default;
    - BBTicket(string initName, int initAge, float initPrice, bool initFan, string initDate);
    - bool isDate(string date);
    - string toString();
    - string toFile();
    - void updateFanStatus(int numVisits);
    - void print();
    - These are further defined in the associated model class source code file...

Model class .cpp file

- Includes definitions of class methods, constructors, etc.
  - Custom (parameterized) and default constructors
  - toString()
    - formats a return string for console printing, containing the private members of an instance of the BBTicket class
  - toFile()
    - Formats a return string for writing data members for a BBTicket object CSV style for writing to a text file, format: name.age.price.fan.date
  - isDate()
    - Determines whether or not the date of a certain BBTicket object matches the date provided as an argument. Date strings are MM/DD/YYYY with no leading zeroes.
  - updateFanStatus()
    - Used to update the fanStatus data member depending on the passed integer (a number of games attended)
  - print()
    - Used to automatically format a BBTicket object using toString and print to console with a newline

Controller Portion: Preprocessors

- Includes
  - <iostream> // std streams
  - <fstream> // ifstream and ofstream
  - <vector> // hashtable indeces contain vectors of the BBTicket class objects read in and entered
    // by the user
  - “BBTicket.hpp”
- Defines
  - File names
  - Input validation values
  - Hashtable array length
  - Debugging utility

Controller Portion: Menu

- Do {...} while (userInput != 4);
- Gather user inputs to output desired selection
  - Add New Ticket
  - Search Tickets
  - Print Out All Tickets
  - Quit
- Uses a loop to cycle through the options
- Grabbed data from file to print out all tickets including user entries
- Used a vector to gather inputs from user
- Option parameters; only allow certain inputs

Add input validation stuff

Controller Portion: Functions

- There are two important functions used in main
  - readFileLine(ifstream &inputFile, BBTicket &currentClassObject)
    - To be used in a for loop with a predeclared ifstream and default constructed class object
    - Each input data member is checked for validity by char type and reasonable limits are set for further validation where possible; an errCode is passed by reference which is a 5 bit binary number where a 0 represents an error for a data member
    - Chops string at comma delimiters from a file line grabbed with getline()
      - Uses calls to .substring(), which is passed delimiter positions via .find_first_of()
      - Implicit or explicit type casts to proper data member object type
      - Custom class constructor called to update the currentClassObject data members, passed by reference
    - Calculates hash value using hashIt() so as to take advantage of the private member being available then, and returns the hash index for use in the controller portion
  - hashIt(std::string date)
    - Computes hash for a particular class object (by its date member) for storing at a particular array index in the hash table.

Course Objectives Addressed

- Program construction and design
  - Create programs with more than one compile module
  - Collaborate with others in designing a larger program
  - Use of preprocessor defines to aid in debugging/testing
  - Adhere to course formatting requirements
  - Create robust test plans to test all corner cases
- OOP Principles
  - Adhered to OOP principles of design
    - Privacy, deferral to experts, etc.
    - Avoid use of getters and setters
- Use of complex aggregate data structures
  - Array of vectors containing custom class objects
  - Iterating through such a structure
- Apply bit-level operators
  - hashIt() operates on the bit-level
  - Custom errCode
- Use text files and streams for data I/O

Future Improvements

- Shorten main() by writing individual functions for the menu options
  - Lots of the input validation lines are repeated (begging to be implemented with a controller function)
- Some text printing can be combined into function calls
- Add more methods to the BBTicket class
- Implement the hashtable as a custom class object (pay homage to OOP)
  - The hashtable::hashIt() method will contain the equivalent controller logic implemented in yet another compile module

CODE DEMO