Semester Project Part 2: Solving the SET Daily Puzzle

Data Structures and Analysis of Algorithms, akk5

Objectives

- To strengthen student's knowledge of C++ programming
- To give student experience reading and parsing strings of commands
- To give student experience in writing Data Structures for data types
- To give student experience designing and implementing a solution to a problem

Instructions

A classic example of an object is a playing card. There are numerous games that utilize a deck of 52 playing cards; consider the game called Set instead. The object of the game is to identify "sets" of 3 cards from an array of 12 cards.

Each card has four features:

1. Symbols - oval, squiggle, or diamond
2. Color - Red, green, or purple
3. Number - one, two or three
4. Shading - filled, open, or striped.

A "set" consists of 3 cards in which each of the card's features, looked at one-by-one, are the same on each card or different on each card. All features must satisfy this rule.

As an example, consider the following 3 cards --

- 2 red filled squiggles
- 2 red striped squiggles
- 2 red open squiggles

These cards form a set because:

1. They all feature squiggles
2. They are all red
3. They all have 2 symbols on them
4. One of them is filled, one is striped, and one is open.

Now consider the following 3 cards --

- 2 red filled squiggles
- 2 red striped squiggles
- 2 green open squiggles

These cards do not form a set because two of the cards are red and one is green.

Now consider the following 3 cards --

- 2 red filled squiggles
- 1 green striped oval
- 3 purple open diamonds

Is this a set? Yes

Why? Because all of the features on the cards are different.

For this assignment, you will need to create a program that will solve the Daily Set puzzle (see https://www.setgame.com/set/puzzle for more details).

Your program should implement a command line (text-based interface) capable of handling the following commands:

exit – exits the program

load *<file>* as *<puzzle>* - reads a file containing a list of cards and stores them in the puzzle named ***puzzle***. This command will create a new puzzle if necessary.

load *<file>* - parses the contents of the file as if they were entered from the command line

display – displays the list of created puzzles

display *<puzzle>* – displays the cards stored in the puzzle named ***puzzle***

new *<puzzle>* - creates an empty puzzle named ***puzzle***.

remove *<puzzle>* - Removes the specified puzzle from the list.

remove *<card>* from *<puzzle>* - Removes the specified card from the puzzle named ***puzzle***.

add *<card>* to *<puzzle>* - Inserts the specified card into the puzzle name ***puzzle***.

solve *<puzzle>* - solves the specified puzzle and displays its solution.

solve *<card1>* *<card2>* - displays the card required to form a set with ***card1*** and ***card2***.

test *<card1>* *<card2>* <card3>- displays whether ***card1***, ***card2***, and ***card3*** form a set.

Note: ***<file>*** and ***<puzzle>*** are strings but will not contain spaces.

All ***<card>*** entries are 4 characters long and have the following format:

[number][color][shading][shape]

Where ***[number]*** = {1,2,3}, ***[color]*** = {G,P,R}, ***[shading]*** = {F,O,S}, ***[shape]*** = {D,O,S}.

This means that the card showing 2 red filled ovals would be represented as 2RFO and the card showing 1 green striped squiggle would be represented as 1GSS.

Guidance

I am not providing you with much guidance on this project. Its main goal is for you to analyze a basic system (in this case a card game and its rules) and devise a set of approaches for solving the problems the system raises. I recommend visiting the SET daily puzzle site and working through the how to play material and playing the puzzle a couple of times to understand how to solve the puzzle.

I recommend playing close attention to how they present any set you find; if you understand what you are seeing, it actually tells you a lot about one method for solving the puzzle.

Also take advantage of code reuse; use the parsing code you wrote for Program 1 as the basis for your parsing in this program. If you had trouble with parsing in Program 1, a parsing code base has been provided you can reuse.

This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

Grading Breakdown

Point Breakdown	
Structure	12 pts
The program has a header comment with the required information.	3 pts
The overall readability of the program.	3 pts
Program uses separate files for main and class definitions	3 pts
Program includes meaningful comments	3 pts
Syntax	16 pts
Implements Class Card correctly	16 pts
Behavior	72 pts
Program handles all command inputs properly	
• Exit the program	6 pts
• Load a valid puzzle file	6 pts
• Load a valid command file	6 pts
• Display the list of puzzles	6 pts
• Display the cards of a given puzzle	6 pts
• Create an empty puzzle	6 pts
• Remove a puzzle from the list	6 pts
• Remove a card from a given puzzle	6 pts
• Add a card to a given puzzle	6 pts
• Display the solution for a give puzzle	6 pts
• Display the card that completes the set for the two cards provided	6 pts
• Determine if the three cards form a set	6 pts
Total Possible Points	**100pts**
Penalties	
Program does NOT compile	-100
Late up to 72 hrs	-10 / day
Late more than 72hrs	-100

Header Comment

At the top of each program, type in the following comment:

/*

Student Name: <student name>

Student NetID: <student NetID>

Compiler Used: <Visual Studio, GCC, etc.>

Program Description:

<Write a short description of the program.>

*/

Example:

/*

Student Name: John Smith

Student NetID: jjjs123

Compiler Used: Eclipse using MinGW

Program Description:

This program prints lots and lots of strings!!

*/

Assignment Information

Files Expected:

This project should require the implementation of one or more classes as well as the function necessary for parsing the various inputs. If you only need to implement one class, then I would expect at least 3 files: Main.cpp, Class.h and Class.cpp.