Contestant Number: _____

Time: _____

Rank: _____

C# Programming

(330)

REGIONAL 2025

PRODUCTION:

Regional C# _____**(550 points)**

Test Time: 90 minutes

GENERAL GUIDELINES:

Failure to adhere to any of the following rules will result in disqualification:

1. Member must hand in this test booklet and all printouts if any. Failure to do so will result in disqualification.
2. No equipment, supplies, or materials other than those specified for this event are allowed in the testing area. No previous BPA tests and/or sample tests (handwritten, photocopied, or keyed) are allowed in the testing area.
3. Electronic devices will be monitored according to ACT standards.

Scenario Prompt: Magic Potion Lab GUI Application Challenge

Youve been hired as the lead developer for the Magic Potion Lab application, where you'll design and implement a graphical program to manage a collection of magical potions. Your goal is to create a user-friendly, Windows Forms (WinForms) application that allows users to add, remove, update, and search for potions, as well as generate a random "mystery mix" potion. You'll use specific naming conventions and follow all requirements listed below.

This prompt will guide you step-by-step through the challenge, detailing the components and features needed to build this application successfully.

Project Overview

Create a WinForms C# application named **MagicPotions**. The program will have a single form (called **MainForm**), and it will display potion details and allow users to interact with a collection of potions through various controls (buttons, text fields, and lists). All user interactions will be handled on this single form.

Step 1: Setting Up Data Structure

Potion Storage: Store each potion as a dictionary within a list of dictionaries. Each dictionary will represent a potion with the following keys:

- PotionName: A string that stores the name of the potion (e.g., "Potion of Strength").
- Ingredients: A List<string> that stores each ingredient of the potion (e.g., ["Unicorn Hair", "Goblin Dust"]).
- Effect: A string that describes the magical effect of the potion (e.g., "Increases speed by 50%").
- Potency: An int between 1 and 10 that describes the potency level of the potion.

Step 2: Designing the User Interface (UI)

TextBox Controls (for entering details of a new or existing potion):

- txtPotionNameInput: A TextBox to enter the potions name.
- txtIngredientsInput: A TextBox to enter the potions ingredients, separated by commas.
- txtEffectInput: A TextBox to enter a description of the potions effect.
- txtPotencyInput: A TextBox to enter the potency level (as an integer between 1 and 10).

Button Controls:

- btnAddPotion: Adds a new potion to the potion list using the details entered in the input fields.
- btnRemovePotion: Removes a potion from the list by its name, using the value entered in txtPotionNameInput.
- btnExperimentPotion: Updates an existing potions details, including ingredients and potency.
- btnSearchPotion: Searches the potion list for potions by name or by ingredients, using txtPotionNameInput for the search term.

- btnDisplayPotions: Displays all potions in the list, showing their names, ingredients, effects, and potency.
- btnMysteryMix: Generates a random potion with a name, ingredients, effect, and potency, then adds it to the potion list.

Display Control:

- lstPotionDisplayList: A ListBox that shows details of each potion currently stored in the list. When btnDisplayPotions is clicked, all potions in the list should appear here, formatted as:
    - Name: [PotionName], Ingredients: [Ingredients], Effect: [Effect], Potency: [Potency]

Status Label:

- lblStatusMessage: A Label at the bottom of the form that displays success messages, errors, and other status updates to guide the user.

Step 3: Implementing Core Functions

Adding a Potion

- When btnAddPotion is clicked:
    - Retrieve values from txtPotionNameInput, txtIngredientsInput, txtEffectInput, and txtPotencyInput.
    - Validate that txtPotionNameInput is non-empty, txtIngredientsInput contains at least one ingredient, txtEffectInput is non-empty, and txtPotencyInput contains an integer between 1 and 10.
    - If validation passes, create a new potion dictionary with "PotionName", "Ingredients", "Effect", and "Potency".
    - Add the new potion to the potions list.
    - Display a success message in lblStatusMessage and clear the input fields.

Removing a Potion

- When btnRemovePotion is clicked:
    - Use the value from txtPotionNameInput to search the potions list.
    - If a potion with the matching "PotionName" exists, remove it from the list, display a success message in lblStatusMessage, and refresh lstPotionDisplayList.
    - If the potion doesnt exist, display an error message in lblStatusMessage.

Updating a Potion (Experimenting)

- When btnExperimentPotion is clicked:
    - Search for a potion in potions by "PotionName" using txtPotionNameInput.
    - If found, update the potions "Ingredients" with values from txtIngredientsInput and "Potency" from txtPotencyInput (validate potency between 1 and 10).
    - Display a success message in lblStatusMessage and refresh lstPotionDisplayList.

Searching for Potions

- When btnSearchPotion is clicked:

- Search the potions list by name or ingredient using the value from txtPotionNameInput.
- If matches are found, display them in lstPotionDisplayList with full details.
- If no matches are found, display an error message in lblStatusMessage.

Displaying All Potions

- When btnDisplayPotions is clicked, display all potions in lstPotionDisplayList with each potions name, ingredients, effect, and potency level.

Creating a Mystery Mix

- When btnMysteryMix is clicked:
  - Generate a random potion with randomized values for "PotionName", "Ingredients", "Effect", and "Potency" and add it to the potions list.
  - Display the new potion in lstPotionDisplayList and show a message in lblStatusMessage.

Additional Requirements and Naming Conventions

Naming Conventions:

- All UI components must use the specified names (txtPotionNameInput, txtIngredientsInput, etc.) to ensure clarity and organization.
- Each method should be named logically based on its function (btnAddPotion_Click, btnRemovePotion_Click, etc.).

Error Handling:

- Ensure that invalid inputs (like non-integer potency levels or empty potion names) result in an error message in lblStatusMessage.
- Prevent duplicate potion names from being added to the list.

Helper Methods:

- Create a ClearInputs() method to clear txtPotionNameInput, txtIngredientsInput, txtEffectInput, and txtPotencyInput after a successful action.
- Use a DisplayPotionDetails method to format potion details for easy display in lstPotionDisplayList.

Step 5: Testing

- Test each button to confirm it performs the expected actions.
- Verify that invalid inputs (empty fields, invalid potency) result in clear error messages.
- Confirm that adding, updating, and removing potions update lstPotionDisplayList and lblStatusMessage correctly.
- Test the btnMysteryMix to ensure it generates random potions with valid attributes.

Sample Data for Potion TextBox Inputs

Potion 1: Speed Elixir

- txtPotionNameInput: Speed Elixir
- txtIngredientsInput: Phoenix Feather, Ginseng Root, Lightning Leaf
- txtEffectInput: Increases movement speed by 50%
- txtPotencyInput: 8

Potion 2: Elixir of Fire Resistance

- txtPotionNameInput: Elixir of Fire Resistance
- txtIngredientsInput: Fireweed, Salamander Scale, Ash Bark
- txtEffectInput: Protects the user from fire damage for 10 minutes
- txtPotencyInput: 7

Potion 3: Invisibility Potion

- txtPotionNameInput: Invisibility Potion
- txtIngredientsInput: Ghost Mushroom, Shadow Dust, Cloak Flower
- txtEffectInput: Grants invisibility for 5 minutes
- txtPotencyInput: 9

Potion 4: Elixir of Eternal Youth

- txtPotionNameInput: Elixir of Eternal Youth
- txtIngredientsInput: Phoenix Feather, Mermaid Tears, Pearl Powder
- txtEffectInput: Slows aging for 24 hours
- txtPotencyInput: 10

Potion 5: Night Vision Potion

- txtPotionNameInput: Night Vision Potion
- txtIngredientsInput: Owl Eye, Moonflower Petal, Bat Wing
- txtEffectInput: Grants night vision for 1 hour
- txtPotencyInput: 6

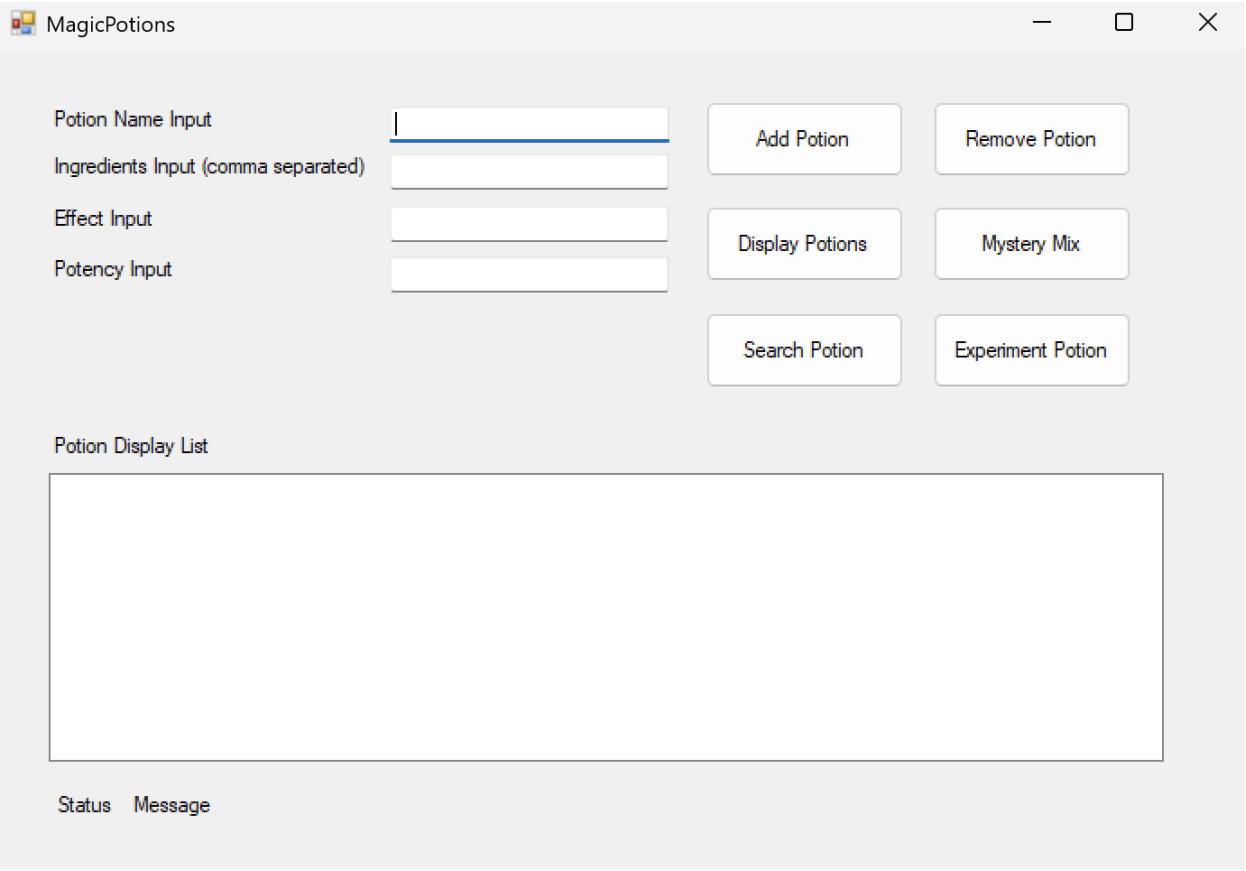

Figure 1: Application Image

Category	Criteria	Points Possible	Points Received
Data Structure (50 Points)	**Potion List & Dictionaries**: Properly stores potions as dictionaries within a list, with correct keys ("PotionName", "Ingredients", "Effect", "Potency") and data types for each key.	30	
	Randomization Setup: Initializes a Random object correctly to generate random values for mystery potions.	10	
	Potency Validation: Ensures txtPotencyInput only accepts integers between 1 and 10, rejecting other inputs.	10	
User Interface (120 Points)	**TextBox Components**: Correctly includes TextBox controls for txtPotionNameInput, txtIngredientsInput, txtEffectInput, and txtPotencyInput.	20	
	Button Components: Includes buttons with correct functionality and names (btnAddPotion, btnRemovePotion, btnExperimentPotion, btnSearchPotion, btnDisplayPotions, btnMysteryBrew).	20	
	PotionDisplayList Component: Utilizes a ListBox for potion display, displaying full potion details named lstPotionDisplayList	20	
	Status Label Component: Uses Label effectively to communicate success and error messages to the user named lblStatusMessage	20	
	Naming Conventions: Follows specified naming conventions for all controls, methods, and variables.	20	
	Clear Inputs: Correctly implements and utilizes a ClearInputs() method after each action.	20	
Core Functions (250 Points)	**Add Potion Functionality**: Adds a new potion using input values, validates all fields (ensuring no empty or invalid entries), and adds the potion to potions.	40	
	Add Potion Success Message: Updates lblStatusMessage with a success message upon adding a potion.	10	
	Remove Potion Functionality: Searches for and removes a potion by name; if the potion is not found, displays an error message in lblStatusMessage.	40	
	Remove Potion Success Message: Updates lblStatusMessage with a success message upon removing a potion.	10	

Category	Criteria	Points Possible	Points Received
	Experiment Potion Functionality: Updates an existing potion's ingredients and potency (validates 1–10 range), then refreshes the display list.	40	
	Experiment Potion Success Message: Updates lblStatusMessage with a success message when a potion is updated.	10	
	Search Potion Functionality: Searches by potion name or ingredients, displaying matching potions in lstPotionDisplayList and updating lblStatusMessage.	40	
	Display All Potions: Displays all potions in lstPotionDisplayList in the correct format with all potion details.	20	
	Mystery Brew Functionality: Generates a random potion with randomized values for "PotionName", "Ingredients", "Effect", and "Potency" and adds it to the potion list.	30	
	Mystery Brew Success Message: Displays a message in lblStatusMessage when a mystery potion is created.	10	
Helper Methods & Code Quality (80 Points)	**DisplayPotionDetails**: Correctly formats potion details into a single string for display in lstPotionDisplayList.	20	
	ClearInputs Method: Clears input fields (txtPotionNameInput, txtIngredientsInput, txtEffectInput, txtPotencyInput) after an action is completed.	20	
	Error Handling: Implements error messages for invalid inputs, such as empty fields, invalid potency, or non-existent potion names for removal or update.	20	
	Code Readability: Code is well-organized, with comments that clarify the purpose of each function and major code sections.	20	
Testing & Functionality (50 Points)	**Function Testing**: All core functions (btnAddPotion, btnRemovePotion, btnExperimentPotion, btnSearchPotion, btnDisplayPotions, btnMysteryBrew) perform as expected without errors.	20	
	User Feedback: Provides clear and correct messages in lblStatusMessage for both successful actions and errors.	10	

Category	Criteria	Points Possible	Points Received
	Form Resetting: After each action, the form resets properly (fields clear, display list updates, status label updates).	10	
	Overall User Experience: Application is intuitive and user-friendly, with smooth navigation, feedback, and performance.	10	

Total Points Possible 550