FAST NUCES Programming Olympics

PROGRAMMING FUNDAMENTAL CS1002 - Fall 2024

Release Date: 15th November 2024 Due Date: 29th november 2024

Total Marks 70

Congratulations, athletes! You've advanced to the next stage of the Programming Olympics for Fall 2024 by excelling in Assignment 2. Your journey in CS1002 continues with the third round of challenges.

- The rules remain simple: Your mission is to cross the finish line, but not just with speed.

  This time, we focus even more on elegance, creativity, and the dedication you put into your problem-solving journey.
- True champions face obstacles with integrity. Remember, the biggest challenge isn't just solving the problems—it's overcoming the temptations of using shortcuts like ChatGPT, Gemini, or relying on others' solutions. Such actions will lead to immediate disqualification.

Now, show us your best work, stay honest, and keep your eyes on the prize. Let the challenge begin!

How to submit

There are a total of 7 questions in the assignment and each question carries equal marks i.e 10. You must submit the .c files compressed in a zip folder with your student id on google classroom within the due date.

Question 01

Write a program that contains a structure named Employee which contains the following data members:

- employeeCode
- employeeName
- dateOfJoining

Perform the following operations:

1. **Define a function** that assigns user-defined values to these variables.
2. **Create an array of 4 Employee structs** and initialize them with user-defined values.
3. **Define a function** that asks the user to enter the current date, calculates the tenure of each employee, and checks if the tenure is more than three years. Display the details of employees with tenure of more than three years and the count of such employees.

Question 02

Write a program that organizes a digital cricket match, "Cricket Showdown," where two players, Player 1 and Player 2, compete over 12 balls. Each player takes turns to score runs on each ball. Players can enter scores between 0 and 6 for each ball, and if a score outside this range is entered, it will be disregarded, but the ball will still be marked.

1. **Define a structure Player** with the following members:
    a. ballScores[12]: An array to store the score for each ball.
    b. playerName: A string to hold the player's name.
    c. totalScore: An integer to store the total score for each player.
2. **Define functions**:
    a. playGame(struct Player player): This function prompts each player to enter their score for each of the 12 balls.
    b. validateScore(int score): This function checks if the score is between 0 and 6 (inclusive). If it's not, the score is disregarded, but the ball is still marked.
    c. findWinner(struct Player player1,struct Player player2): Determines the winner based on the total score.
    d. displayMatchScoreboard(struct Player player1,struct Player player2): Displays a summary of each player's performance, including each ball's score, the average score, and total score.

Question 03

Create a program to validate an email address based on a few basic criteria. The program will prompt the user to enter an email address and will dynamically allocate memory to store and process the input.

Define the following function:

1. *int validateEmail(char* email)*: This function validates the email based on the following criteria:
    - Contains exactly one @ symbol.
    - Contains at least one dot (.) after the @ symbol.

- Is non-empty.
- Returns 1 if the email is valid, and 0 if invalid.

Steps:

1. User Input: Prompt the user to enter an email address. Allocate memory dynamically for the email, ensuring the memory size is based on the input length.
2. Validation Process:
   - Call the validateEmail function to check if the email meets the criteria.
3. Display Results:
   - Print "Valid Email" if the email meets the criteria.
   - Print "Invalid Email" if the email does not meet the criteria.
4. Memory Cleanup:
   - Free the dynamically allocated memory after validation to prevent memory leaks.

Question 04

You are creating a system to track employee performance ratings over multiple evaluation periods. The system will dynamically allocate memory to store ratings and perform various analysis tasks, including finding the top-performing employee and the best-rated evaluation period.

Define the Employee Structure:

Create a structure called Employee with the following fields:

- ratings: A dynamically allocated array to store the ratings for each evaluation period.
- totalScore: An integer to store the employee's total score across all evaluation periods.

Define the following function:

1. Input Ratings Function:
   - Implement a function ***void inputEmployees(int** ratings, int numEmployees, int numPeriods)*** to allow the user to enter ratings for each employee across all evaluation periods.
   - Each rating should be between 1 and 10 (inclusive). Input validation should be implemented to enforce this.
2. Display Performance Function:
   - Implement a function ***void displayPerformance(int** ratings, int numEmployees, int numPeriods)*** that displays the performance ratings for each employee across all evaluation periods.
3. Find Employee of the Year Function:
   - Implement a function ***int findEmployeeOfTheYear(int** ratings, int numEmployees, int numPeriods)*** to calculate and return the index of the employee with the highest average rating.
4. Find Highest Rated Period Function:
   - Implement a function ***int findHighestRatedPeriod(int** ratings, int numEmployees, int numPeriods)*** to calculate and return the evaluation period with the highest average rating across all employees.
5. Find Worst Performing Employee Function:
   - Implement a function ***int findWorstPerformingEmployee(int** ratings, int numEmployees, int numPeriods)*** to calculate and return the index of the employee with the lowest average rating.

Memory Management:

- Dynamically allocate memory for each employee's ratings based on the number of evaluation periods.
- After completing all tasks, ensure that the dynamically allocated memory is properly freed to prevent memory leaks.

Question 05

You're building an inventory system for a pet shop called "**Pets in Heart**" that keeps track of different species of animals and their specific supplies (e.g., food, toys, bedding). The shop inventory system uses a 2D dynamic array char** speciesSupplies where:

- Rows are explicitly set for each species (e.g., "Dogs," "Cats," "Parrots"), and each row corresponds to a different species.
- Columns are dynamically allocated for each species to hold their specific supplies (e.g., "Food," "Leash," "Toys").

Task:

Write a C program that:

1. Initialize the Inventory: Allocate memory for a specified number of species, with each species having its own list of supplies (initially empty).
2. Add Supplies: For each species, dynamically allocate memory for a list of supplies and allow the user to add supplies for that species.
3. Update Supplies: Let users update the name of a supply item for a specific species.
4. Remove Species: Allow users to delete a species and free the associated memory (both for the species and its supplies).
5. Display Inventory: Show the current supplies for each species in the inventory.

Question 06

In a cutting-edge agritech system for precision farming, a dynamic pointer-based architecture is deployed to seamlessly manage interconnected data across fields, crops, weather, and smart equipment. Each **field** is represented as a structure containing GPS coordinates, soil health metrics, and moisture levels, alongside a pointer to a dynamically allocated array of **crop structures**. These **crop structures** store critical details like crop type, growth stage, and expected yield while maintaining pointers to **weather forecast structures** that provide hyper-local predictions for temperature, rainfall, and wind patterns.

The **field structure** also includes a pointer to an **equipment array**, representing key farming tools like autonomous tractors, irrigation systems, and drones. Each equipment structure tracks operational status, fuel levels, and activity schedules, enabling synchronized field operations. In addition, an array of **sensor structures** is linked to each field, capturing real-time data on soil nutrients, pH levels, and pest activity, empowering farmers to make informed decisions.

To scale operations, fields are grouped into **regional hubs**, each represented by a structure with pointers to arrays of fields. These hubs maintain aggregate data like yield predictions, resource distribution, and emergency response plans. All regional hubs are connected to a central analytics server through pointers, allowing AI algorithms to process massive datasets and generate real-time insights on crop health, irrigation efficiency, and equipment optimization.

This system’s dynamic design ensures that every byte of memory is utilized efficiently, enabling rapid scaling and adaptation to environmental conditions. By leveraging advanced pointer

structures, the agritech platform offers farmers a futuristic, data-driven farming experience that maximizes yield, minimizes waste, and supports sustainable agricultural practices.

Question 07

In a Netflix-like streaming platform, 2D pointers are used to dynamically manage and personalize the viewing experience for millions of users across diverse content categories and device types. The platform employs a 2D pointer structure, where each row represents a **user profile**, and each column corresponds to a **content category** (e.g., Action, Drama, Comedy). A double** engagementMatrix pointer points to this 2D array, where each element stores a numerical engagement score (e.g., average viewing time or like/dislike ratio) for a user's interaction with that category.

Each user profile structure includes a pointer to their respective row in the engagement matrix, allowing for quick retrieval and updates of personalized data. For example, engagementMatrix[userIndex][categoryIndex] can be updated whenever the user streams content from a specific category, dynamically recalibrating their preferences in real-time.

The system also uses a secondary 2D pointer structure to manage **device-specific preferences**. For instance, deviceMatrix[userIndex][deviceIndex] points to dynamically allocated arrays holding resolution preferences, playback history, and bandwidth usage for different devices (smart TVs, laptops, smartphones) associated with a user's profile. This enables seamless transitions between devices while maintaining personalized settings like resolution and playback position.

Additionally, another 2D pointer system tracks **content metadata**, where each row corresponds to a content category and each column represents a specific piece of content. Each element in this matrix contains a pointer to a structure with attributes like title, rating, runtime, and encoding formats, enabling quick access to metadata for streaming.

This multi-layered 2D pointer-based design allows the platform to efficiently store, retrieve, and update personalized recommendations, device preferences, and content metadata. By leveraging such dynamic data structures, the system delivers a highly tailored, device-optimized viewing experience for users, ensuring maximum engagement and satisfaction while handling the scalability needs of a global user base.