CSCI291, Task 2: Arrays, C-struct, and C-string (100 marks)
Due date: 15/11/2024 @ 11:55 pm

Constraint: C pointer variables must NOT be used to solve any of the questions in this assessment.

Question 1: Array Utility Functions [19 marks]

Consider the attached C program `Lab3_Q1.c` where *arr* is a 1D integer array of length SIZE, and *arr2d* is a 2D integer array of *nRows* rows and *nCols* columns.

After coding in **C** the below functions, complete the `main()` function to test the correctness of the function implementations. Your program should not use a global variable or additional define symbolic constant. Note that you **might** need to add further parameters to the below function signatures/headers

- **bool isValid(const int arr[],int length, int pos,…**): returns true if pos is within the valid range of array indices, false otherwise. **This function must be called in every function that checks the validity of the indices.**

[2 marks]

- **void remove_element(int arr[],int length, int pos):** removes the array element at index *pos* by shifting up all preceding array elements by one position, see Fig 1.1. The value of the array at index 0 remains unchanged. If "pos" is not a valid array index, print a relevant message and exit the function.

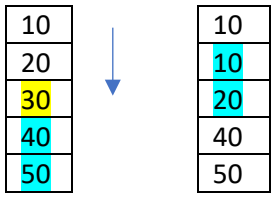

Fig 1.1: remove_element of arr[2]

[4 marks]

- **void insert_element(int arr[],int length, int pos, int value):** inserts the parameter `value` at the specified index *pos*, shifting the original value at pos and all its preceding elements one position down, see Fig 1.2. If the value of *pos* is not a valid array index, print a relevant message and exit the function.

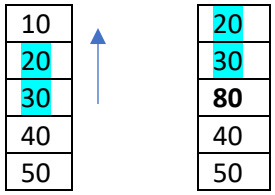

Fig 1.2: Insert the value 80 at array index 2

[3 marks]

- **void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols**])
  implements the two requirements:
  - If the length of *arr* is not equal "nRows*nCols", print a relevant error message and exit the function.
  - Otherwise, copy the elements of the 1-D array *arr* into *arr2d*, filling it **column by column.**

[4 marks]

- **void trans_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat[nCols][nRows]):** generates mat_transp, the transpose of the input matrix mat.

[2.5 marks]

- **bool found_duplicate(int arr[],int length, ….):** returns *true* if there is at least a duplicate values in *arr*; otherwise returns *false*.

[3.5 marks]

Question 2: Banking Transactions Processing (array, break, continue) [9 marks]

You are tasked with creating a program to process bank account transactions. These transactions are stored in an array, where positive numbers represent deposits and negative numbers represent withdrawals. Your program should perform the following steps:

1. Start with an initial balance of 1000 AED.
2. Sequentially process the array of transactions as follows:
   a. If a withdrawal exceeds the current balance, skip that transaction and print a message indicating that it is invalid. Store the unprocessed transaction in a separate array called *tobeprocessed*.
   b. Update the balance for valid transactions.
   c. If at any point the balance reaches 0, stop processing further transactions, log a message indicating that no further transactions will be processed, and store the unprocessed transaction in the *tobeprocessed* array.
3. After checking all transactions, print the final balance and the contents of the *tobeprocessed* array.

For testing purposes, create your own transactions array to demonstrate all the functionalities described above. You can start with the following data: transactions = {200, -150, -500, -400, -50, -200, 300}.

[9 marks]

Question 3: League Team Application (array, struct) [42 marks]

You have been hired as a developer in a popular sports channel. Your task is to design and implement a C application that holds and processes the squad of NUM_TEAMS teams participating in a competition league. Table 1 gives the attributes of each player, which should be grouped into the C struct definition **player_t** as follows:

Table 1: Player Attributes

Name (String)	Kit Number (int)	Club (string)	Age (struct)	Position (String)
Full name of the player (first and last name separated by a space, max total size: 25)	Players wear two-digit numbered kits ranging between 1-99.	Name of the club participating in the league. The name can include a space.	Age of the player consisting of day, month, and year of birth.	Describe a player's role within the team, for example, in football, a role can be that of a midfielder.

The participating teams should be represented in an array with a length defined by SQUAD_SIZE. Each element of the array is of **struct** type, having the attributes given in Table 2.

Table 2: Team Attributes

Name (String)	Array of players (player_t)	ActiveSize (int)
Name of the team (can include a space, max total size: 20)	To store the details of the team (size of the array is SQUAD_SIZE)	To store the current number of enrolled players in the squad.

Both NUM_TEAMS and SQUAD_SIZE should be defined constants in your program. Feel free to google any information about any sport discipline, player details, or make up your own players and team names.

Specification

The application should implement the following user-defined functions You may also add other utility functions to avoid unnecessary code duplication.

1. **void display_menu(…)**: Called from the main() function to *continually* display a text menu to the user. The menu specifies the integer value to input in order to call any of the below user-defined functions. Any other input value should terminate the program after displaying a relevant message.
   [3 marks]

2. **void enroll_club(…)**: Allows the user to enroll a club by entering a club name. The function should ensure that the maximum number of clubs does not exceed **NUM_TEAMS**.
   [3 marks]

3. **void add_player(…)**: Prompts the user to select a club from a list of enrolled clubs and to subsequently enter player details (name, unique kit number, date of birth, position). The function should check for duplicates in player names and kit numbers. If duplicates are found, the enrolled player names and kit numbers should be displayed, and the user should be prompted to enter a new valid input. The input values should be stored as a player_t struct.
   [11 marks]

4. **void search_update(…)**: Allows the user to search for players by either player name (case-insensitiv[1]e) or kit number. If a match is found, the details of the player are displayed and the user

[1] use strcasecmp:https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rtref/strcasecmp.htm).

is prompted to update any matching player attributes. Any changes input by the user should be updated in the application.

[12 marks]

5. **void display_club_statistics(…)**: Displays a list of all enrolled clubs along with the number of players in each team, the average age of the players, and each player's details.

[5 marks]

6. **void handle_error(…) functions or statements**: Provides appropriate messages for invalid inputs, such as out-of-range selections, invalid kit numbers, and duplicate entries.

[4 marks]

Struct definitions, function prototypes, defined constants, overall structure of the program.

[4 marks]

Recommendation:
First, study the requirements carefully before writing your design solution, and then begin the coding process. At the coding stage, first write the skeleton of your program in terms of headers, defined symbols, function prototypes, and function headers before starting to implement and test each function **in turn.**

What to submit:

Your submission must include:

a. Source files with concisely commented code.
b. A design solution description (Flowchart/pseudocode) for Question 2 and the function **search_update** from Question 3. For the latter, explain how the struct data is accessed in your code **(7 marks).**
c. Testing Evidence with discussion for ALL questions **(7 marks).**
d Weekly progress evidence submission to github and in-lab assessment via observations and Q&A.

(16 marks)