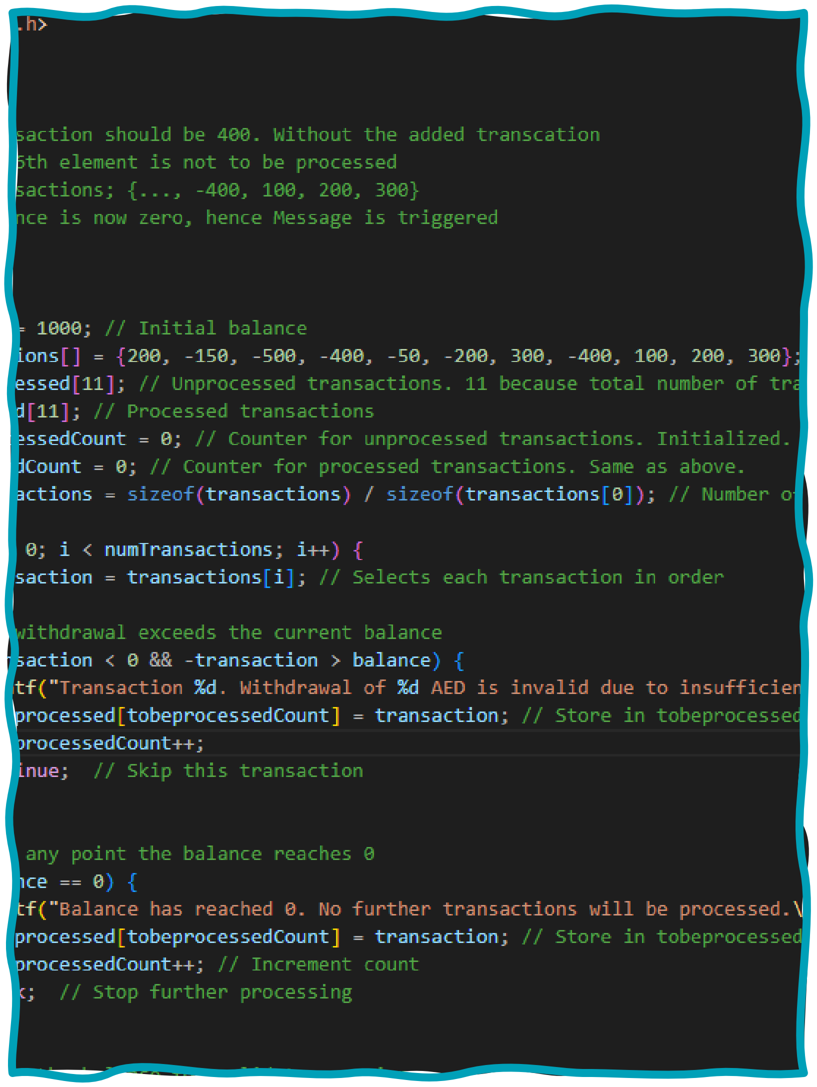


```
.h>



saction should be 400. Without the added transcation
6th element is not to be processed
sactions; {..., -400, 100, 200, 300}
nce is now zero, hence Message is triggered



= 1000; // Initial balance
ions[] = {200, -150, -500, -400, -50, -200, 300, -400, 100, 200, 300};
essed[11]; // Unprocessed transactions. 11 because total number of tra
d[11]; // Processed transactions
essedCount = 0; // Counter for unprocessed transactions. Initialized.
dCount = 0; // Counter for processed transactions. Same as above.
actions = sizeof(transactions) / sizeof(transactions[0]); // Number o

 0; i < numTransactions; i++) {
saction = transactions[i]; // Selects each transaction in order

withdrawal exceeds the current balance
saction < 0 && -transaction > balance) {
tf("Transaction %d. Withdrawal of %d AED is invalid due to insufficien
processed[tobeprocessedCount] = transaction; // Store in tobeprocessed
processedCount++;
inue;  // Skip this transaction


 any point the balance reaches 0
nce == 0) {
tf("Balance has reached 0. No further transactions will be processed.\
processed[tobeprocessedCount] = transaction; // Store in tobeprocessed
processedCount++; // Increment count
k;  // Stop further processing
```

Lab 3: Arrays, C-struct, and C-string

PROGRAMMING FOR ENGINEERS LAB

Name: Ahmed Ismail | ID: 8541747 | Course Title: CSCI291 | Date: 27/10/2024

UNIVERSITY OF WOLLONGONG IN DUBAI

Table of Contents

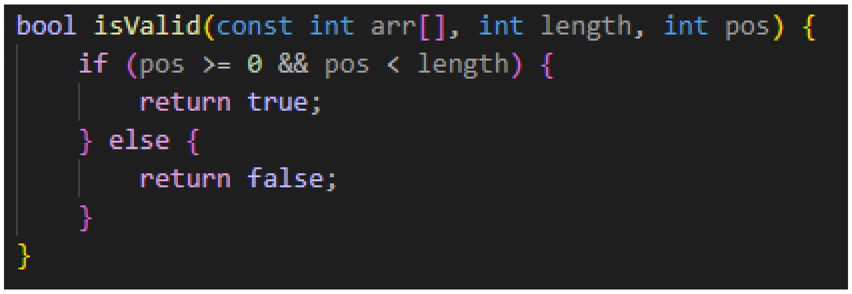
1. Source Code Documentation 2
Question 1: Array Utility Functions 2
Question 2: Banking Transactions Processing 4
Question 3: League Team Application 5
2. Design Solutions 7
Question 1: Array Utility Functions 7
Question 2: Banking Transaction Processing Pseudocode 7
Question 2: Banking Transaction Processing Flow chart: 9
Question 3: search_update Function Design 9
Question 3: search_update Function FlowChart 11
3. Testing Evidence and Discussion 12
Question 1 Testing Results 12
Question 2 Testing Results 14
Question 3 Testing Results 14
GitHub Page: 18

1. Source Code Documentation

QUESTION 1: ARRAY UTILITY FUNCTIONS

Part 1:

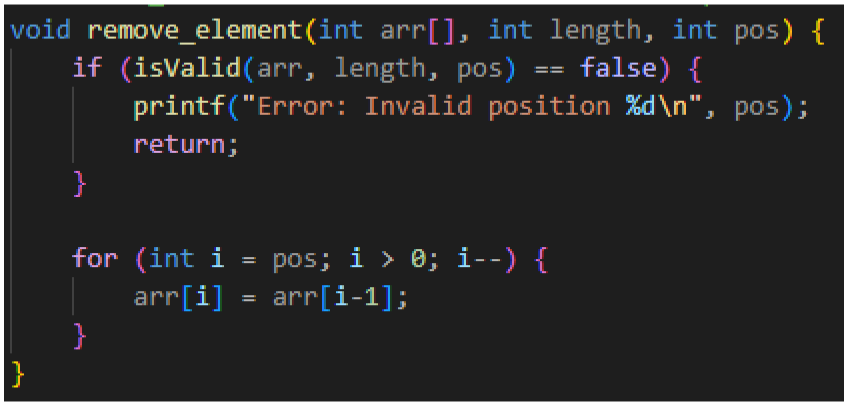
- isValid: Validates array positions. If pos is valid, return true. Else return false.



```c
bool isValid(const int arr[], int length, int pos) {
    if (pos >= 0 && pos < length) {
        return true;
    } else {
        return false;
    }
}
```

Figure 1 isValid Function code

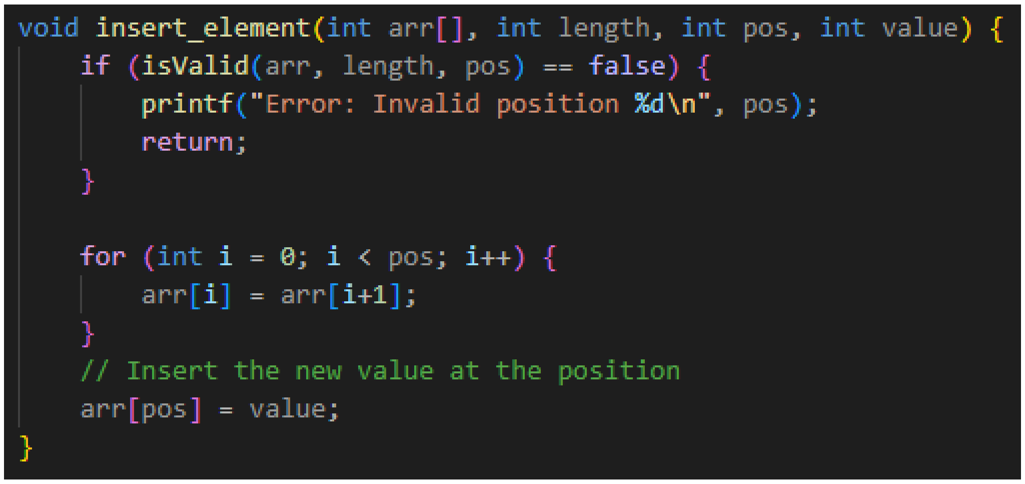
- remove_element: Removes elements with upward shifting. Preserves first element as required. If pos is valid, it performs a **leftward shift** starting from the specified position. It uses a for loop to assign each element to the position to its right (arr[i] = arr[i-1]). This "removes" the element at pos by overwriting it.



```c
void remove_element(int arr[], int length, int pos) {
    if (isValid(arr, length, pos) == false) {
        printf("Error: Invalid position %d\n", pos);
        return;
    }

    for (int i = pos; i > 0; i--) {
        arr[i] = arr[i-1];
    }
}
```

Figure 2 remove_element Function code

- insert_element: Inserts elements with downward shifting. If valid, it performs a **rightward shift** to make room for the new value. It uses a for loop to shift each element to the left (arr[i] = arr[i+1]) up to the specified position. This shift causes the first element in the array to be overwritten by the second, and so on, until the new value is inserted at pos. After the loop, the function inserts value into arr[pos].

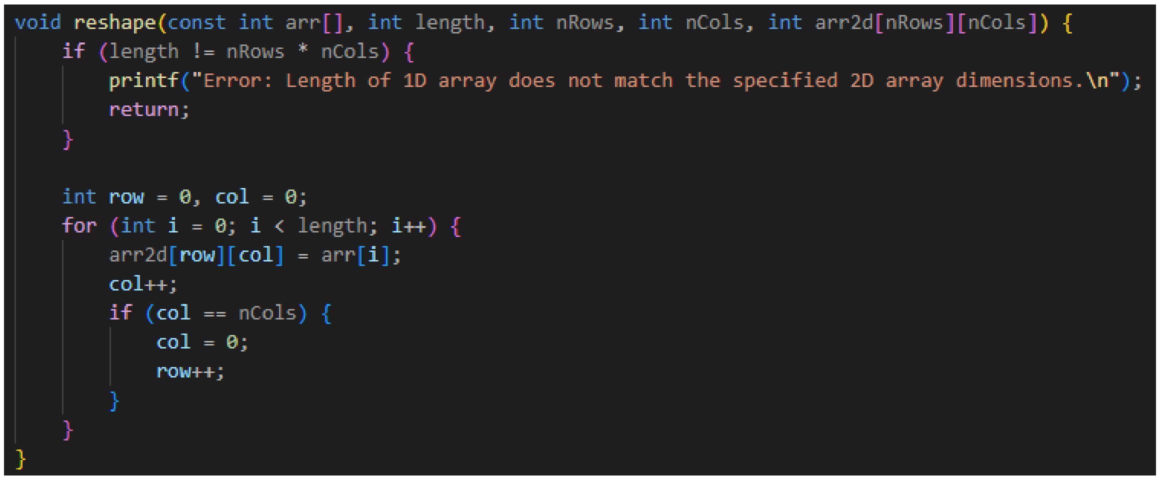


```
void insert_element(int arr[], int length, int pos, int value) {
    if (isValid(arr, length, pos) == false) {
        printf("Error: Invalid position %d\n", pos);
        return;
    }

    for (int i = 0; i < pos; i++) {
        arr[i] = arr[i+1];
    }
    // Insert the new value at the position
    arr[pos] = value;
}
```

Figure 3 insert_element Function code

Part 2:

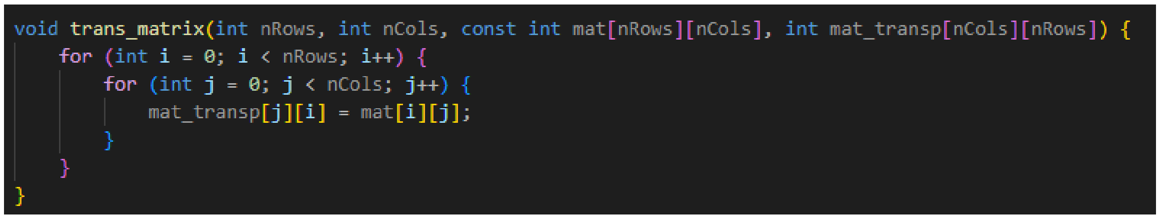
- reshape: Converts 1D array to 2D array column-wise. row and col keep track of the current position in the 2D array, with col incrementing until it reaches the number of columns (nCols). When col reaches nCols, it resets to 0, and row increments to move to the next row.



```
void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]) {
    if (length != nRows * nCols) {
        printf("Error: Length of 1D array does not match the specified 2D array dimensions.\n");
        return;
    }

    int row = 0, col = 0;
    for (int i = 0; i < length; i++) {
        arr2d[row][col] = arr[i];
        col++;
        if (col == nCols) {
            col = 0;
            row++;
        }
    }
}
```

Figure 4 reshape Function code

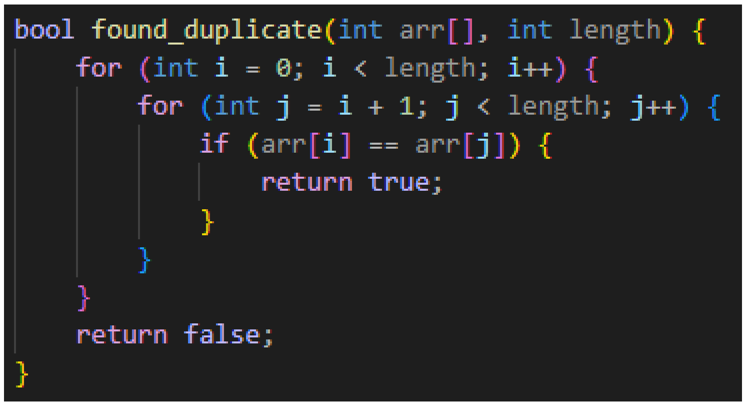
- trans_matrix: Creates transpose of input matrix. The function iterates through each element in the input matrix (mat), switching the row and column indices when assigning values to the transposed matrix (mat_transp[j][i] = mat[i][j]).



```c
void trans_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat_transp[nCols][nRows]) {
    for (int i = 0; i < nRows; i++) {
        for (int j = 0; j < nCols; j++) {
            mat_transp[j][i] = mat[i][j];
        }
    }
}
```

Figure 5 trans_matrix Function code

- found_duplicate: Checks for duplicate values in array. It uses a nested loop to compare each element with all subsequent elements in the array. The outer loop iterates through each element, and the inner loop compares it with all elements that come after it. If a match is found (arr[i] == arr[j]), the function returns true, indicating a duplicate was found.



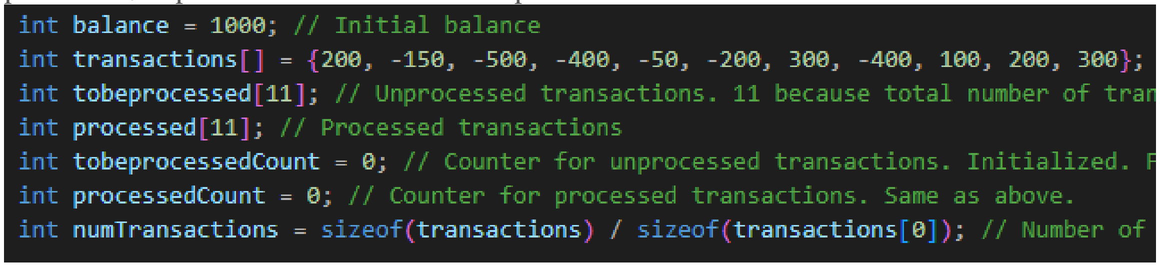
```c
bool found_duplicate(int arr[], int length) {
    for (int i = 0; i < length; i++) {
        for (int j = i + 1; j < length; j++) {
            if (arr[i] == arr[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Figure 6 found_duplicate Function code

QUESTION 2: BANKING TRANSACTIONS PROCESSING

Code components:

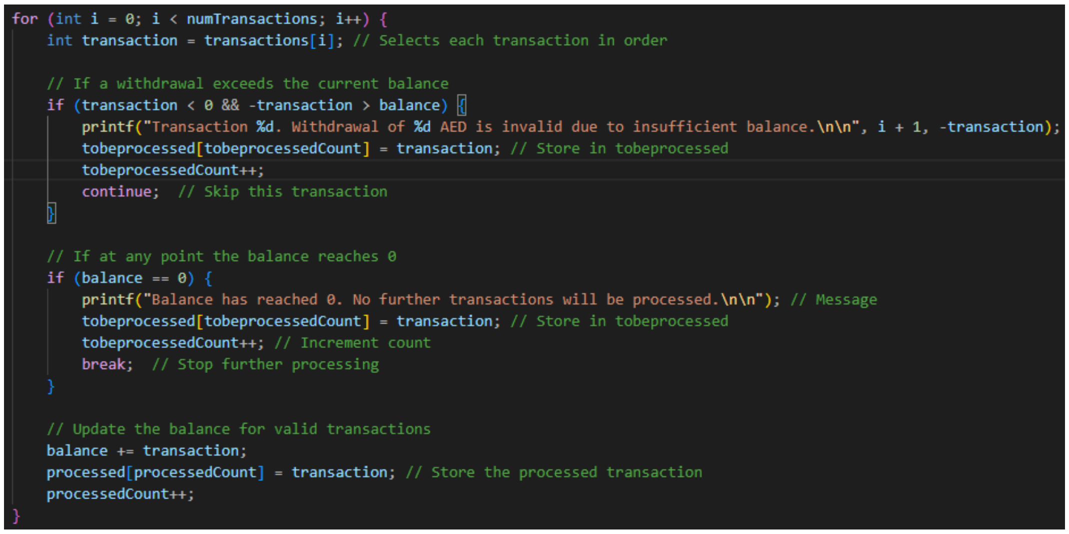
- Begins with initial balance of 1000 AED. Uses parallel arrays for tracking processed/unprocessed transactions. Implements transaction counters.



```c
int balance = 1000; // Initial balance
int transactions[] = {200, -150, -500, -400, -50, -200, 300, -400, 100, 200, 300};
int tobeprocessed[11]; // Unprocessed transactions. 11 because total number of tran
int processed[11]; // Processed transactions
int tobeprocessedCount = 0; // Counter for unprocessed transactions. Initialized. F
int processedCount = 0; // Counter for processed transactions. Same as above.
int numTransactions = sizeof(transactions) / sizeof(transactions[0]); // Number of
```

Figure 7 Initialization of balance, transaction and arrays code

- Implements validation for withdrawals, Handles zero balance condition. Uses separate arrays for tracking transaction status.

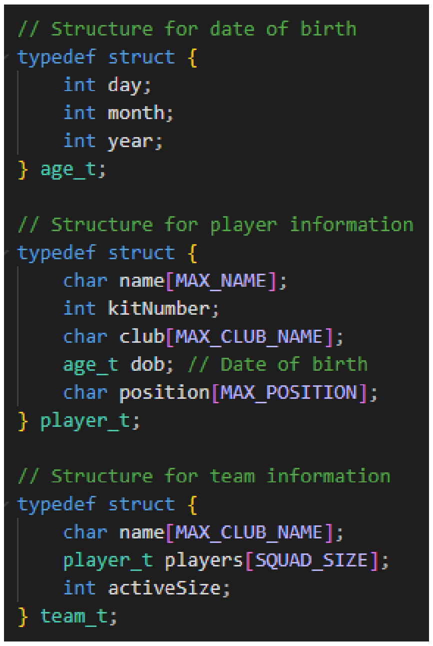


```c
for (int i = 0; i < numTransactions; i++) {
    int transaction = transactions[i]; // Selects each transaction in order

    // If a withdrawal exceeds the current balance
    if (transaction < 0 && -transaction > balance) {
        printf("Transaction %d. Withdrawal of %d AED is invalid due to insufficient balance.\n\n", i + 1, -transaction);
        tobeprocessed[tobeprocessedCount] = transaction; // Store in tobeprocessed
        tobeprocessedCount++;
        continue;  // Skip this transaction
    }

    // If at any point the balance reaches 0
    if (balance == 0) {
        printf("Balance has reached 0. No further transactions will be processed.\n\n"); // Message
        tobeprocessed[tobeprocessedCount] = transaction; // Store in tobeprocessed
        tobeprocessedCount++; // Increment count
        break;  // Stop further processing
    }

    // Update the balance for valid transactions
    balance += transaction;
    processed[processedCount] = transaction; // Store the processed transaction
    processedCount++;
}
```

Figure 8 For loop of transaction processing

QUESTION 3: LEAGUE TEAM APPLICATION

Code components:



```c
// Structure for date of birth
typedef struct {
    int day;
    int month;
    int year;
} age_t;

// Structure for player information
typedef struct {
    char name[MAX_NAME];
    int kitNumber;
    char club[MAX_CLUB_NAME];
    age_t dob; // Date of birth
    char position[MAX_POSITION];
} player_t;

// Structure for team information
typedef struct {
    char name[MAX_CLUB_NAME];
    player_t players[SQUAD_SIZE];
    int activeSize;
} team_t;
```

Figure 9 Structs declared for Question 3

Nested Structure Design:

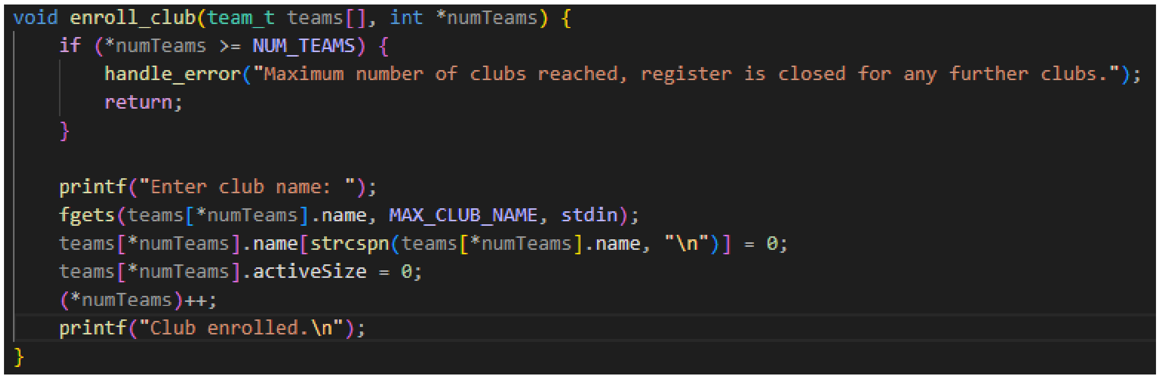
- Uses typedef for clean syntax
- Implements hierarchical data organization
- Proper size constraints for strings

Memory Management:

- Fixed-size arrays for predictable memory usage
- No dynamic allocation required
- Efficient structure packing

Core Functions:

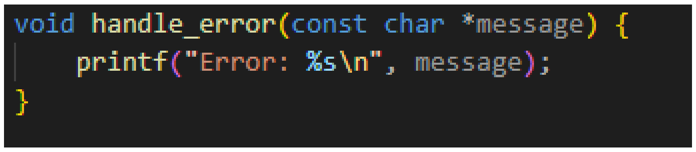
- enroll_club:



```c
void enroll_club(team_t teams[], int *numTeams) {
    if (*numTeams >= NUM_TEAMS) {
        handle_error("Maximum number of clubs reached, register is closed for any further clubs.");
        return;
    }

    printf("Enter club name: ");
    fgets(teams[*numTeams].name, MAX_CLUB_NAME, stdin);
    teams[*numTeams].name[strcspn(teams[*numTeams].name, "\n")] = 0;
    teams[*numTeams].activeSize = 0;
    (*numTeams)++;
    printf("Club enrolled.\n");
}
```

Figure 10 enroll_club code

  - Maintains team count
  - Validates against maximum teams
  - Uses pointer for count update
- add_player:
  - Club Selection
  - Play Information Noting
- search_update:
  - Search Options:
    - Name-based (case-insensitive)
    - Kit number-based
  - Update Options:
    - All player attributes modifiable
    - Maintains data integrity
  - Validation:
    - Duplicate checking
    - Range validation
    - Date validation
- Error Handling:



```c
void handle_error(const char *message) {
    printf("Error: %s\n", message);
}
```

Figure 11 Error Handling message maker code

- Validation Functions:
  - Kit Number Validation
  - Date Validation
  - Duplicate Checking

2. Design Solutions

QUESTION 1: ARRAY UTILITY FUNCTIONS

Main functionalities:

1. isValid: Checks if a given position in an array is valid based on the array's length.
2. remove_element: Removes an element from an array by shifting all subsequent elements up.
3. insert_element: Inserts an element into an array by shifting all subsequent elements down.
4. reshape: Converts a 1D array into a 2D matrix with the specified number of rows and columns.
5. trans_matrix: Transposes a 2D matrix.
6. found_duplicate: Checks if an array contains any duplicate elements.

QUESTION 2: BANKING TRANSACTION PROCESSING PSEUDOCODE

Initialize:

Set balance = 1000

Create arrays: transactions[], tobeprocessed[], processed[]

Set counters: tobeprocessedCount = 0, processedCount = 0

Process Transactions:

FOR each transaction in transactions array:

```
IF transaction is withdrawal AND exceeds balance:
Print invalid transaction message
Add to tobeprocessed array
Continue to next transaction
IF balance is 0:
Print zero balance message
Add remaining transactions to tobeprocessed
Break loop
Update balance with transaction
Add to processed array
Increment processed counter
Output Results:
Print final balance
Print all processed transactions
Print all unprocessed transactions
```

QUESTION 2: BANKING TRANSACTION PROCESSING FLOW CHART:

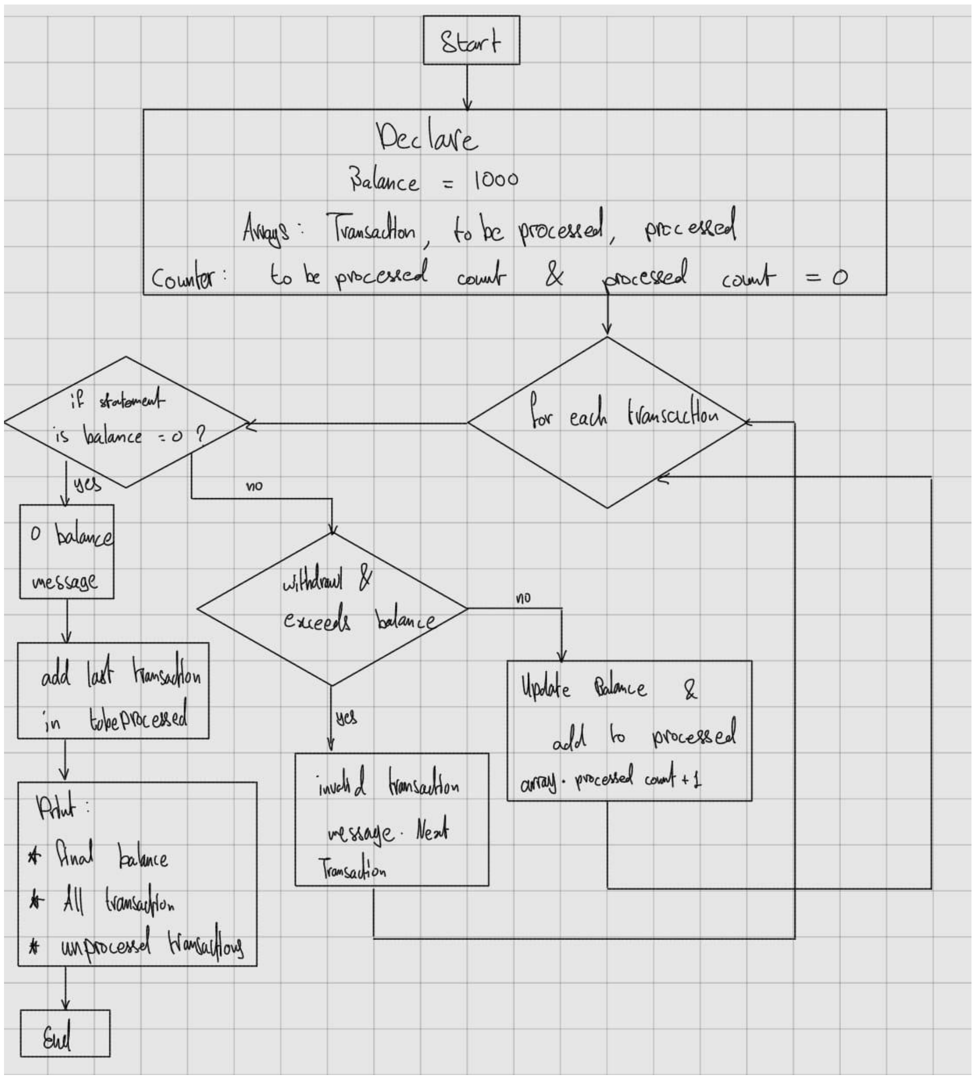

Start
Declare
Balance = 1000
Arrays: Transaction, to be processed, processed
Counter: to be processed count & processed count = 0
for each transaction
if statement is balance = 0 ?
yes
no
0 balance message
withdrawl & exceeds balance
no
yes
add last transaction in tobeProcessed
Update Balance & add to processed array. processed count + 1
invalid transaction message. Next Transaction
Print:
* final balance
* All transaction
* unprocessed transactions
End

Figure 12 Question 2 Code Flow Chart

QUESTION 3: SEARCH_UPDATE FUNCTION DESIGN

Structure Access Explanation:

1. The function accesses struct data through nested structures:

   team_t teams[] -> player_t players[] -> Individual fields

Data access patterns:

- Direct access: teams[i].name
- Nested access: teams[i].players[j].name

- Pointer access: player_t *player = &teams[foundTeamIndex].players[foundPlayerIndex]

The search_update function follows this workflow:

1. Takes user input for search method (name/kit number)
2. Traverses team and player arrays using nested loops
3. Uses pointer to found player for efficient updates
4. Implements validation before updates
5. Maintains data integrity through duplicate checks

QUESTION 3: SEARCH_UPDATE FUNCTION FLOWCHART

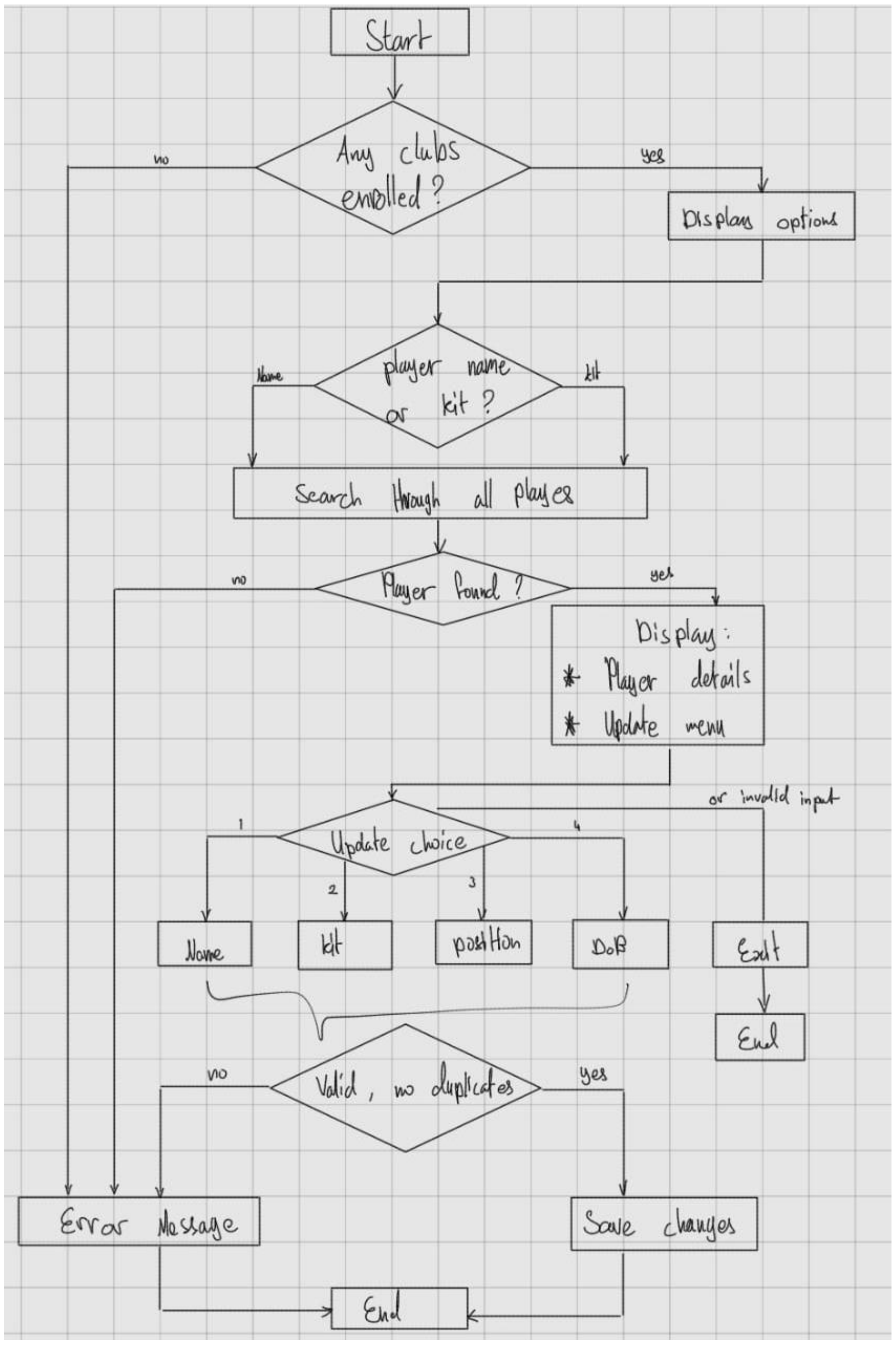
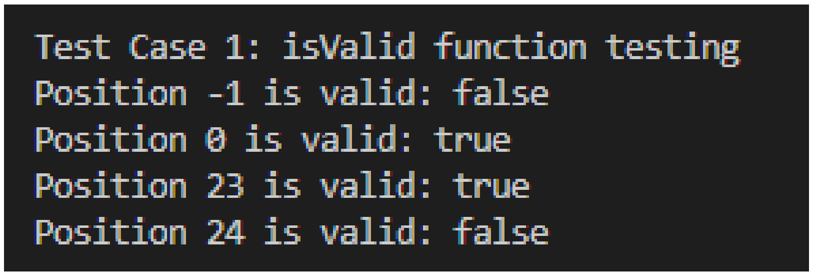


Figure 13 Question 3 Code Flow Chart

3. Testing Evidence and Discussion

QUESTION 1 TESTING RESULTS

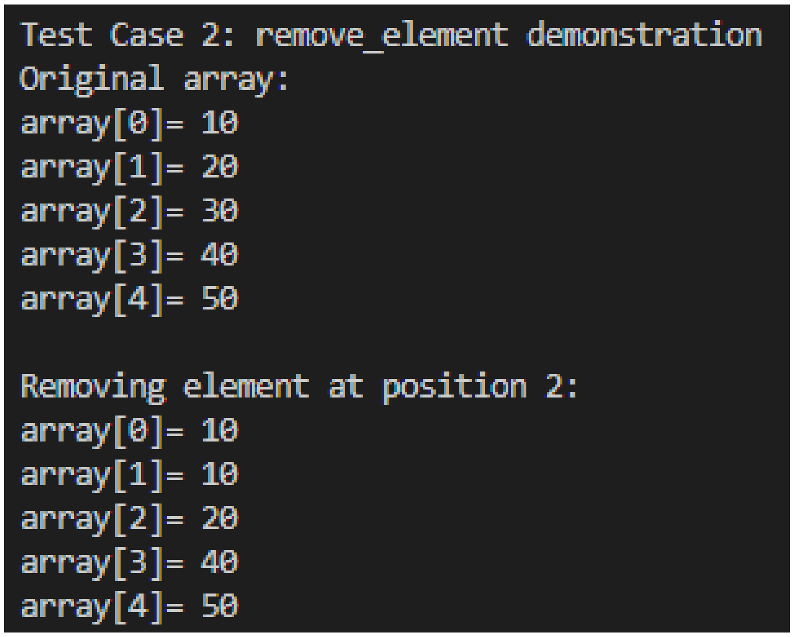
Test case 1: isValid function testing



```
Test Case 1: isValid function testing
Position -1 is valid: false
Position 0 is valid: true
Position 23 is valid: true
Position 24 is valid: false
```

Figure 14 isValid Test Evidence

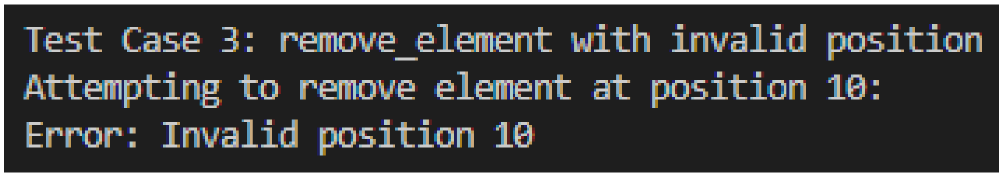
Test Case 2: remove_element demonstration



```
Test Case 2: remove_element demonstration
Original array:
array[0]= 10
array[1]= 20
array[2]= 30
array[3]= 40
array[4]= 50

Removing element at position 2:
array[0]= 10
array[1]= 10
array[2]= 20
array[3]= 40
array[4]= 50
```

Figure 15 remove_element Test Evidence

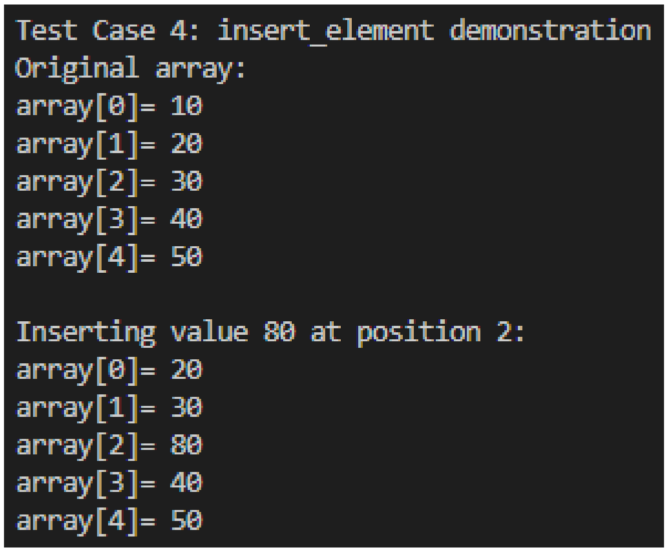
Test Case 3: remove_element with invalid position



```
Test Case 3: remove_element with invalid position
Attempting to remove element at position 10:
Error: Invalid position 10
```

Figure 16 remove_element invalid Test Evidence

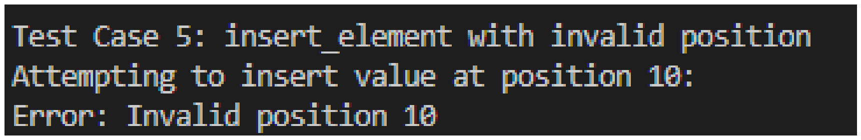
Test Case 4: insert_element demonstration



```
Test Case 4: insert_element demonstration
Original array:
array[0]= 10
array[1]= 20
array[2]= 30
array[3]= 40
array[4]= 50

Inserting value 80 at position 2:
array[0]= 20
array[1]= 30
array[2]= 80
array[3]= 40
array[4]= 50
```

Figure 17 insert_element Test Evidence

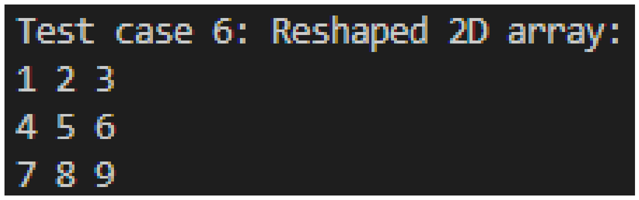
Test Case 5: insert_element with invalid position



```
Test Case 5: insert_element with invalid position
Attempting to insert value at position 10:
Error: Invalid position 10
```

Figure 18 insert_element invalid Test Evidence

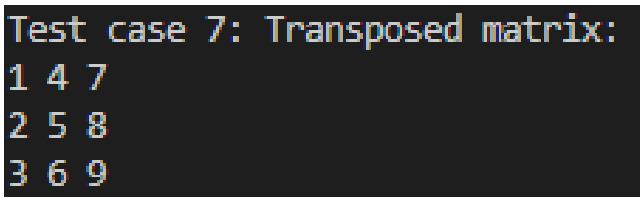
Test case 6: Reshaped 2D array



```
Test case 6: Reshaped 2D array:
1 2 3
4 5 6
7 8 9
```

Figure 19 reshape Test Evidence

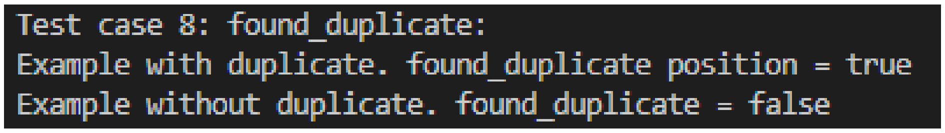
Test case 7: Transposed matrix



```
Test case 7: Transposed matrix:
1 4 7
2 5 8
3 6 9
```

Figure 20 trans_matrix Test Evidence

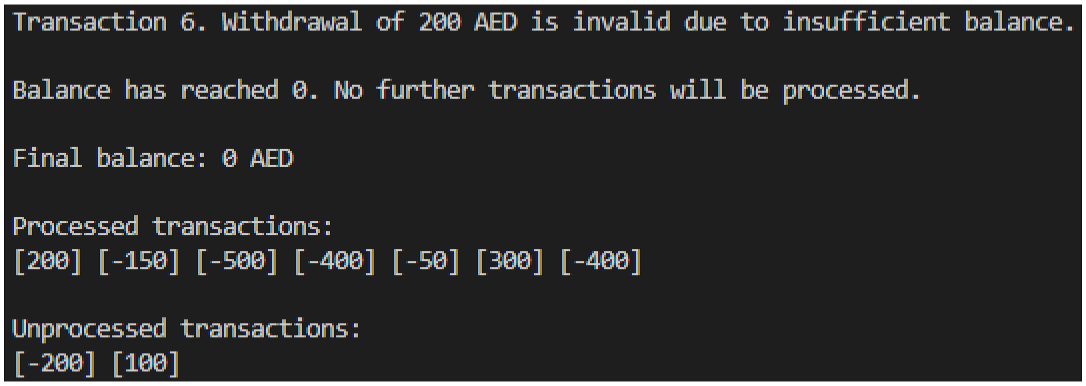
Test case 8: found_duplicate



```
Test case 8: found_duplicate:
Example with duplicate. found_duplicate position = true
Example without duplicate. found_duplicate = false
```

Figure 21 found_duplicate Test Evidence

QUESTION 2 TESTING RESULTS

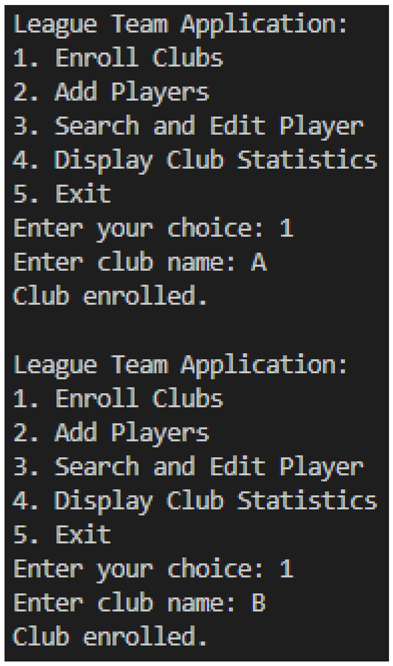


```
Transaction 6. Withdrawal of 200 AED is invalid due to insufficient balance.

Balance has reached 0. No further transactions will be processed.

Final balance: 0 AED

Processed transactions:
[200] [-150] [-500] [-400] [-50] [300] [-400]

Unprocessed transactions:
[-200] [100]
```

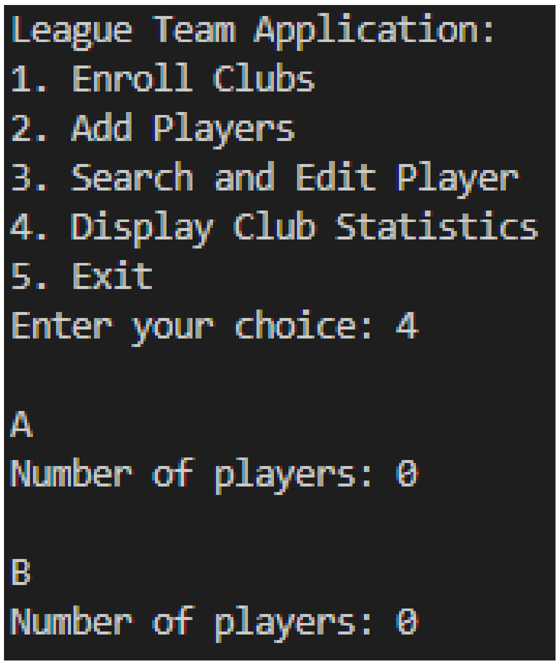
Figure 22 Question 2 Test Evidence

QUESTION 3 TESTING RESULTS

Club enrolment:



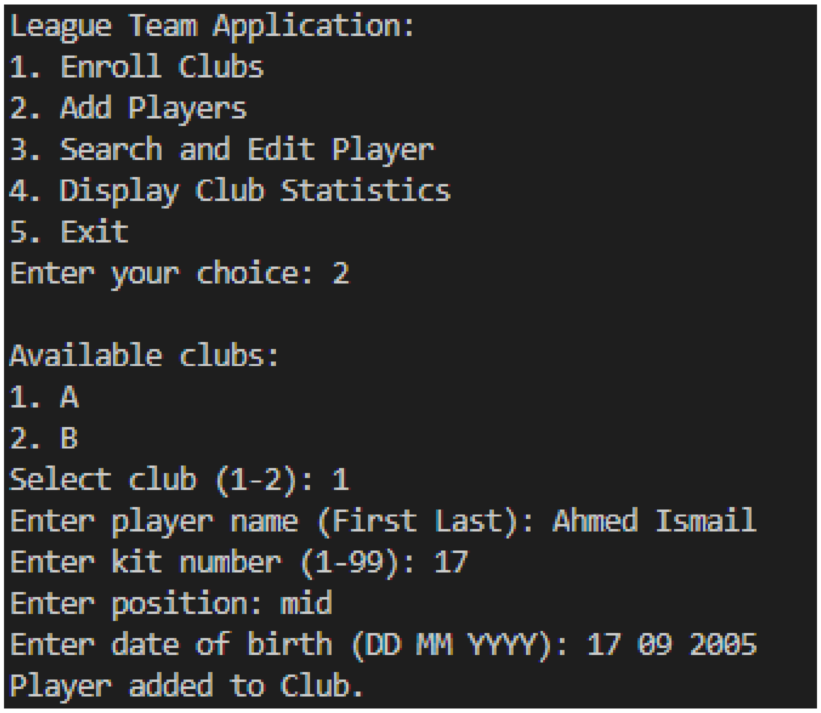
```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 1
Enter club name: A
Club enrolled.

League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 1
Enter club name: B
Club enrolled.
```



```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 4

A
Number of players: 0

B
Number of players: 0
```

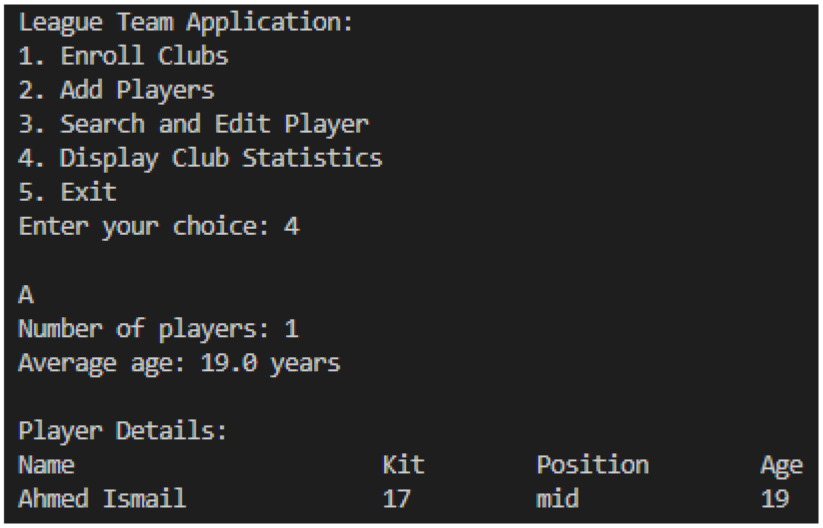
Figure 23 Enrolment Test Evidence

Player addition:



```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 2

Available clubs:
1. A
2. B
Select club (1-2): 1
Enter player name (First Last): Ahmed Ismail
Enter kit number (1-99): 17
Enter position: mid
Enter date of birth (DD MM YYYY): 17 09 2005
Player added to Club.
```

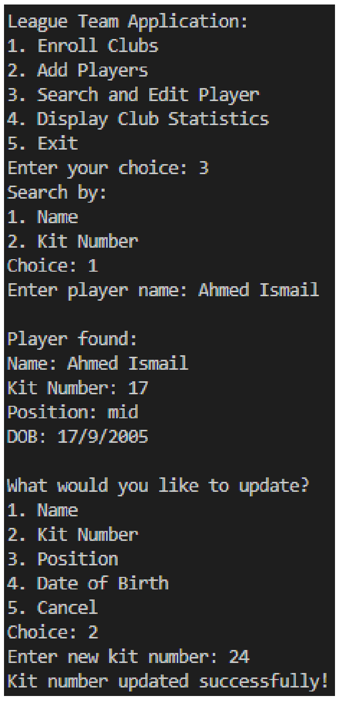
Figure 24 Player addition Test Evidence



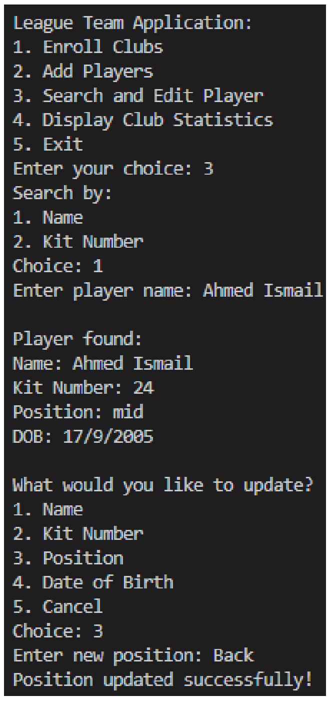
```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 4

A
Number of players: 1
Average age: 19.0 years

Player Details:
Name                        Kit        Position        Age
Ahmed Ismail                17         mid             19
```

Figure 25 Player addition proof in Club Statistics

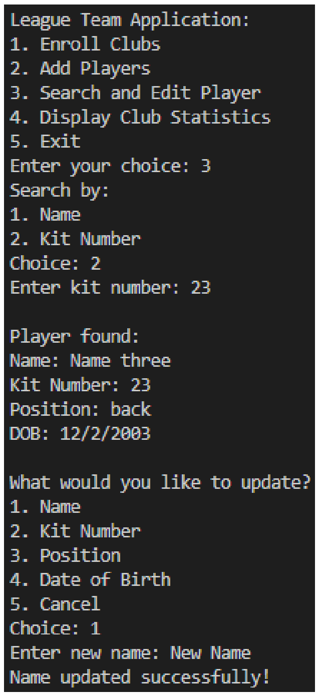
Search and update functionality verified:



```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 3
Search by:
1. Name
2. Kit Number
Choice: 1
Enter player name: Ahmed Ismail

Player found:
Name: Ahmed Ismail
Kit Number: 17
Position: mid
DOB: 17/9/2005

What would you like to update?
1. Name
2. Kit Number
3. Position
4. Date of Birth
5. Cancel
Choice: 2
Enter new kit number: 24
Kit number updated successfully!
```

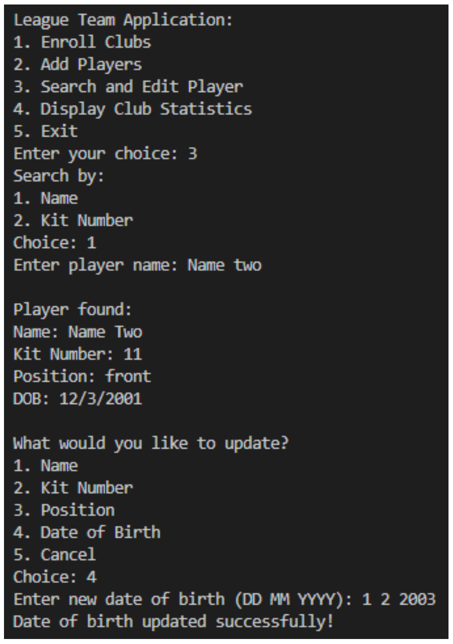


```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 3
Search by:
1. Name
2. Kit Number
Choice: 1
Enter player name: Ahmed Ismail

Player found:
Name: Ahmed Ismail
Kit Number: 24
Position: mid
DOB: 17/9/2005

What would you like to update?
1. Name
2. Kit Number
3. Position
4. Date of Birth
5. Cancel
Choice: 3
Enter new position: Back
Position updated successfully!
```

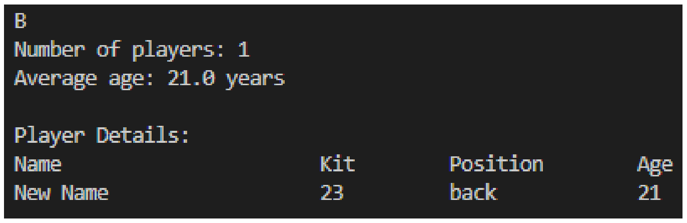


```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 3
Search by:
1. Name
2. Kit Number
Choice: 2
Enter kit number: 23

Player found:
Name: Name three
Kit Number: 23
Position: back
DOB: 12/2/2003

What would you like to update?
1. Name
2. Kit Number
3. Position
4. Date of Birth
5. Cancel
Choice: 1
Enter new name: New Name
Name updated successfully!
```

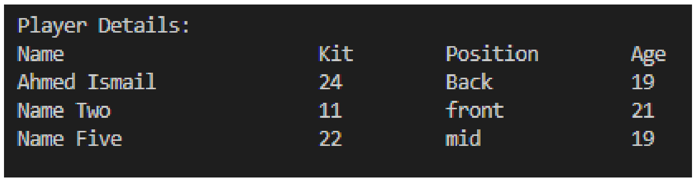
Figure 26 Update Functionality for several cases Test Evidences



```
League Team Application:
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 3
Search by:
1. Name
2. Kit Number
Choice: 1
Enter player name: Name two

Player found:
Name: Name Two
Kit Number: 11
Position: front
DOB: 12/3/2001

What would you like to update?
1. Name
2. Kit Number
3. Position
4. Date of Birth
5. Cancel
Choice: 4
Enter new date of birth (DD MM YYYY): 1 2 2003
Date of birth updated successfully!
```



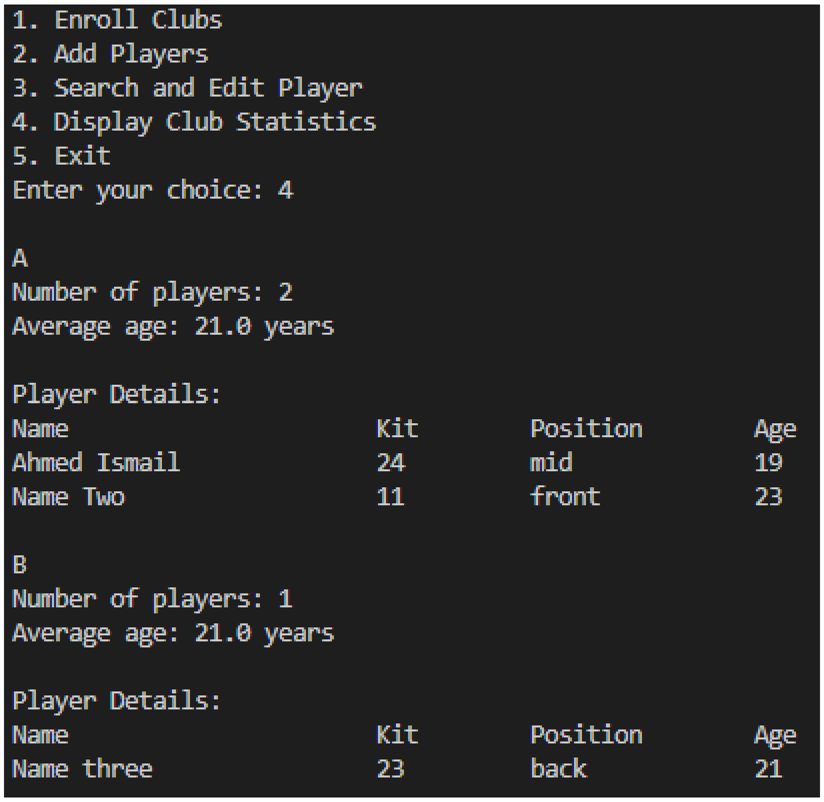
```
B
Number of players: 1
Average age: 21.0 years

Player Details:
Name                Kit       Position       Age
New Name            23        back           21
```



```
Player Details:
Name                Kit       Position       Age
Ahmed Ismail        24        Back           19
Name Two            11        front          21
Name Five           22        mid            19
```

Figure 27 More Update Functionality Test Evidences

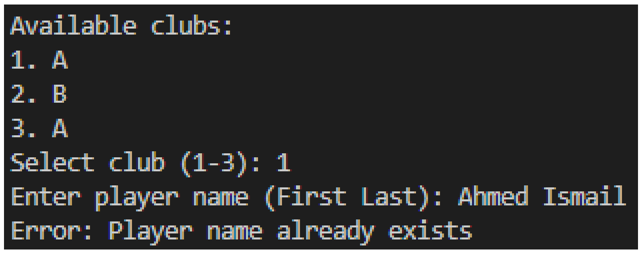
Statistics display correctly formatted:



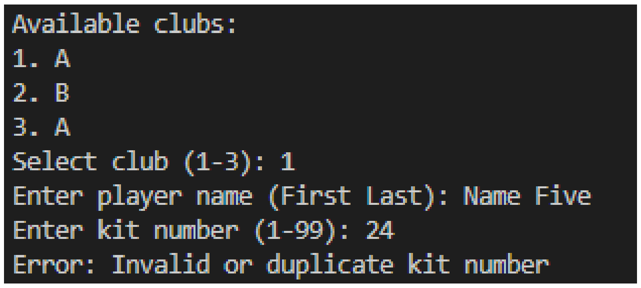
```
1. Enroll Clubs
2. Add Players
3. Search and Edit Player
4. Display Club Statistics
5. Exit
Enter your choice: 4

A
Number of players: 2
Average age: 21.0 years

Player Details:
Name                      Kit        Position        Age
Ahmed Ismail              24         mid             19
Name Two                  11         front           23

B
Number of players: 1
Average age: 21.0 years

Player Details:
Name                      Kit        Position        Age
Name three                23         back            21
```

Figure 28 Statistics Display Test Evidence

Invalid Entries Tests:



```
Available clubs:
1. A
2. B
3. A
Select club (1-3): 1
Enter player name (First Last): Ahmed Ismail
Error: Player name already exists
```



```
Available clubs:
1. A
2. B
3. A
Select club (1-3): 1
Enter player name (First Last): Name Five
Enter kit number (1-99): 24
Error: Invalid or duplicate kit number
```

Figure 29 Invalid Entries, Duplicates and such Test Evidence

GitHub Page:

https://github.com/AIsmail17/CSCI291-Lab-3-Evidence.git