UNIVERSITY
OF WOLLONGONG
IN DUBAI

Arrays, Structs and Strings in C

Muthana Al-Sirhan, #8785995

University of Wollongong in Dubai
CSCI291, Lab #2 Report

Table of Contents

1. Lab Objectives, Methodology

The goal of this lab is to design various programs that involve the usage of C-Arrays, Structs and Strings:-

- Arrays are groups of a fixed number of elements which are all of the same data type, useful for organizing grouping alike variables for ease of access.

- Structures (or "structs") provide a similar use case, however they are more suitable for records as variables in a struct can be of many different data types.

- Strings are a group of characters in the shape of a char array appended with a NULL terminator

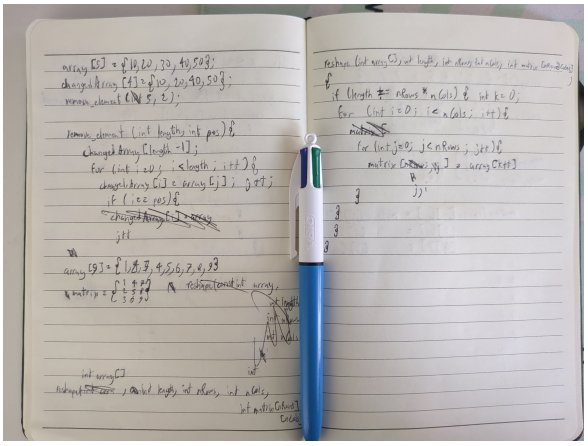
```c
int array[5] = {1, 2, 3, 4, 5};

char course[] = "CSCI291";

typedef struct {
    char name[25];
    char symptoms[50];
    int age;
    float height;
} patientDetails;
```

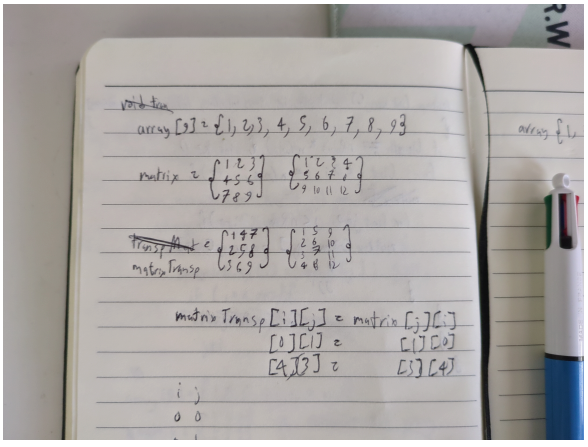
2. Array Utility Functions

Our first task is to program functions that can transform 1D arrays and matrices (2D arrays). They could be manipulated in any of the following ways:-

- **`[remove/insert]_element`** are functions that can remove or insert an element in a specificed position within a 1D array.
- **`found_duplicate`** returns true if there are duplicate values in a 1D array.
- **`reshape`** can copy the elements of a 1D array into a matrix, filling it column by column.
- **`trans_matrix`** can transpose a matrix by flipping it about its diagonal [1], a common phenomenon in linear algebra.

Paper prototyping was perfect for planning the design for this program as it is a simple implementation of arrays:-





[1] https://en.wikipedia.org/wiki/Transpose

2.1. Array Utility Functions [is_valid, insert/remove_element, found_duplicate]

Two functions are programmed to insert and remove an element of an array at a position specified by the user. **changedArray[]** is the modified version of the original array, which is also printed by the function. **is_valid** is called b y the aforementioned functions to check if the user-specified index is a valid position in the array, while **found_duplicate** reports if an inserted element is a duplicate of another.

```
Original array:-
1 2 3 4 5 6 7 8 9 10 11 12

How would you like to transform this array?
1. Insert an element
2. Remove an element
3. Reshape into matrix
4. Transpose matrix (execute 3 first)
Select your choice: 1
Input the desired value and position of the element (value position): 13 7
1 2 3 4 5 6 7 13 8 9 10 11 12
```

```
How would you like to transform this array?
1. Insert an element
2. Remove an element
3. Reshape into matrix
4. Transpose matrix (execute 3 first)
Select your choice: 2
Input the element you wish to remove (position): 15
Position isn't a valid index

How would you like to transform this array?
1. Insert an element
2. Remove an element
3. Reshape into matrix
4. Transpose matrix (execute 3 first)
Select your choice: 2
Input the element you wish to remove (position): 11
1 2 3 4 5 6 7 8 9 10 11
```

```
Input the desired value and position of the element (value position): 12 5
1 2 3 4 5 12 6 7 8 9 10 11 12
'12' duplicate found
```

2.2. Array Utility Functions [`reshape, found_duplicate `]

2D arrays (or matrices) can be created and manipulated by using elements of 1D arrays. Function **`reshape`** fills a matrix row by row with elements from a 1D array, that is if the length of the array is equal to the area of the matrix:-

```
length = rows * cols
```

`trans_matrix` can "transpose" the matrix by flipping it about its diagonal, elements transform similar to the following:-

```
matT[0][1] = mat[1][0]
```

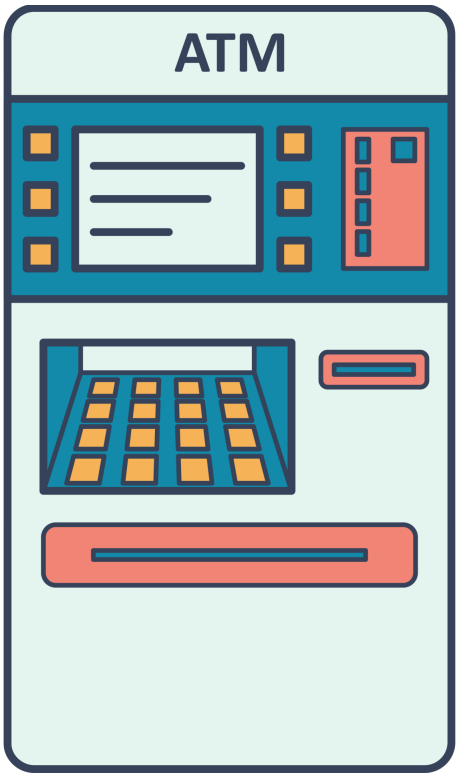
```
How would you like to transform this array?
1. Insert an element
2. Remove an element
3. Reshape into matrix
4. Transpose matrix (execute 3 first)
Select your choice: 3
Column 0 = 1 2 3
Column 1 = 4 5 6
Column 2 = 7 8 9
Column 3 = 10 11 12


How would you like to transform this array?
1. Insert an element
2. Remove an element
3. Reshape into matrix
4. Transpose matrix (execute 3 first)
Select your choice: 4
Transpose Column 0 = 1 4 7 10
Transpose Column 1 = 2 5 8 11
Transpose Column 2 = 3 6 9 12
```

3. Banking Transactions Processing (loop control with array, break, continue)

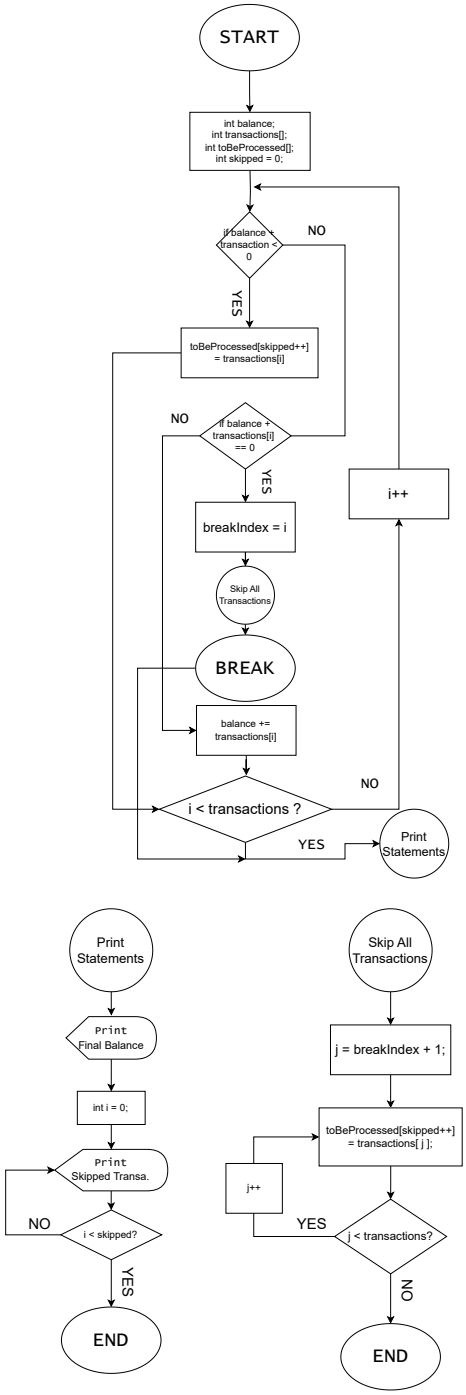
The flow of a loop in C can be controlled with "break" and "continue". This has various real world applications, one being the process of bank account transactions stored in an array. Positive values represent deposits whereas negative numbers represent withdrawals. The program could be simultaneously implemented and tested with the following steps:-

1. Start a variable **`balance`** of 1000 AED
2. Process the array of transactions in the following manner:-
    a. Update the balance for valid transactions
    b. If a withdrawal exceeds the current balance, skip it and print a message informing that it is invalid. Afterwards, it is stored in a separate **`toBeProcessed`** array.
    c. If the balance reaches 0 at any point, stop processing further transactions with an accompanying message that states so, then store the unprocessed transactions in the **`toBeProcessed`** array.
3. After cycling through all transactions, print the final balance and transactions that are yet to be processed (contents of **`toBeProcessed`** array).

The algorithm is compiled as a flowchart in the following page.



3.1. Banking Transactions Processing (flowchart)



3.2. Banking Transactions Processing (testing)

First of all, a balance of 1000 AED and an array of transactions to process are initialized. Another array **toBeProcessed** is initialized with the purpose of storing skipped transactions that either result in balance less than or equal to 0.

```
int main(void) {
    // initialize balance variable and transaction arrays
    int balance = 1000;
    int transactions[11] = {200, -150, -500, -400,
                            -50, -200, 300, -400,
                            500, 500, 500};
    int toBeProcessed[11];
```

Transactions -200 and -400 should be skipped as -200 results in a negative balance while -400 results in 0 balance. In the latter case, all subsequent transactions are also skipped, as shown below.

```
Balance = 1200
Balance = 1050
Balance = 550
Balance = 150
Balance = 100
WARNING: Negative balance
Skipping transaction...
Balance = 400
WARNING: Balance close to 0
Skipping subsequent transactions...
Final Balance = 400
Skipped Transactions:
[1]: -200
[2]: -400
[3]: 500
[4]: 500
[5]: 500
```

4. League Team Application (using arrays, structs)

As mentioned in the introduction, Structures can be great for data records. One real world scenario could be an application that holds and processes a squad of **`NUM_CLUBS`** enrolled in a football league. The attributes of each player and club can be grouped into struct definitions:-

Table 1: Player Attributes				
Name (string)	**Kit Number (int)**	**Club (string)**	**Age (struct)**	**Position (string)**
Full name of the player (first and last)	Players wear 2-digit numbered kits (1-99)	Name of club player is enrolled in	Age of the player through birthdate (day, month, year)	Describe a player's role within their club

Table 2: Club Attributes		
Name (string)	**Array of players**	**Active Size (int)**
Name of the team	To store details of each individual player	To store amount of enrolled players in club

Various user defined functions within this hypothetical program can enable core utility such as:-

1. **`enroll_club`** for enrolling a club by entering a club name. Cannot enroll more clubs if **`NUM_CLUBS`** is reached.
2. **`add_player`** for enrolling a player into the selected club. Should check for duplicates in names and kit numbers and discard them.
3. **`search_update`** for searching a player by name and kit number, then updating any player attributes. (see page 8 for psuedocode on this functions algorithm)
4. **`display_club_statistics`** for displaying a list of all enrolled clubs and details of each player.
5. Various utility functions for displaying the main interface and handling invalid input types without crashing the program.

4.1. League Team Application [enroll_club(), add_player()]

Clubs can be added to the database, then players can be enrolled into each club.

```
###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 1

Enter the name of the FC: Mild FC

Club 'Mild FC' has been enrolled successfully.

###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 1

Enter the name of the FC: Crazy FC

Club 'Crazy FC' has been enrolled successfully.

###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 2

Available clubs:-
1. Mild FC
2. Crazy FC

Enter club number to enroll player: 1
Enter player name: Tom Park
Enter kit number (1-99): 23
Enter position: Player
Enter birth date (day month year): 9 5 1992

Player 'Tom Park' has been added to Mild FC.
```

Players cannot be enrolled if there are no clubs added.

```
###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 2
No clubs available. Please enroll a club first.
```

4.2. League Team Application [`display_club_statistics()`]

Statistics of each club and each players attributes can be accessed with this function. Multiple "for" loops cycle through the entire database and print information.

```
###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 3

Club: Mild FC
Total Players: 2
Player 1:
  Name: Muthana Al-Sirhan
  Kit Number: 25
  Position: Goalie
  Birth Date: 22-08-2005
Player 2:
  Name: Andrew Guzikowski
  Kit Number: 93
  Position: Defense
  Birth Date: 19-06-1999
Average Age of Players: 22
```

4.3. League Team Application [search_update()]

Players can be searched by name. Multiple "for" loops cycle through the entire database of enrolled clubs and players to search for the player. Once found, the player's attributes can be updated.

```
###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 4

Enter the name of the player to search: Muthana Al-Sirhan

Player found in club 'Mild FC'
Enter updated birth date (day month year): 30 11 2005
Enter new kit number: 87
Enter new position: Super Goalie

Player details updated successfully.

###--- League Team Application ---###
1. Enroll Club
2. Add Player
3. Display Club Statistics
4. Search and Update Player
0. Exit

Enter your choice: 3

Club: Mild FC
Total Players: 2
Player 1:
  Name: Muthana Al-Sirhan
  Kit Number: 87
  Position: Super Goalie
  Birth Date: 30-11-2005
```

4.3.1. League Team Application [search_update() algorithm]

```
// search and update player details

void search_update() {
// temp char array for player search
char playerTemp

// bool "found" based on player found or not
Set "found" to false

Prompt "Enter the name of the player"
Read playerTemp
// cycles through all clubs
    For i=0 to clubCount do
        // cycles through players of each club
        For j=0 to clubs[i].activeSize step 1 do
            If clubs[i].Players.playerName is equal to playerTemp do
                Set "found" to true
                Print "Player found in club <clubs[i].name>"

                Prompt "Enter new birth date (day month year)
                Read clubs[i].clubPlayers[j].playerAge

                Prompt "Enter new kit number"
                Read clubs[i].clubPlayers[j].kitNumber

                Prompt "Enter new club position"
                Read clubs[i].clubPlayers[j].position

                Print "Player details updated successfully"
                return;
            endIf
        endFor
    endFor

    If "found" is false do
        Print "Player not found"
}


    If maximum clubs enrolled {
        Print "Cannot enroll more clubs"
        End
    }

    else {
        Print "Enter the name of the FC"
        Read <new club name>
    }

    // detect duplicate names
    For i=0 to <enrolled clubs amount> do
        If clubs[i].name == newClub.name then do {
            Print "This club has already been enrolled"
            End
        }

    // if not duplicate, add to clubs array
    Copy newClub.name to clubs[i]
    Set clubs[i] player size to 0
    Print "Club has been enrolled successfully"
}
```