**Labsheet 10 - Programs using 1D arrays**

You need to write the **pseudocode algorithms** first and then code and test each program. You should be getting **3 programs** **done per lab session at a minimum**. If you don’t you should work on them outside of lab time and have them done before the next lab session.

**Remember to design your programs with pseudo-code and/or flow diagrams first. Name the programs appropriately.**

# It is absolutely prohibited to copy anyone else’s design or code. You can ask for help with a particular problem from friends, colleagues, the lecturers in the lab etc but you must write your own design and code the fix to the problem yourself.

**If you get help from someone or take design/code from the web or elsewhere, you have to comment in your code to state what help you got and from where.**

**Saving Your Projects**

You can create a new project for each program or create them as separate files within the one project.

You should save the programs to a folder called **Labsheet 10** within **Programming\Labsheets** folder on your **M drive.**

**Commenting your Code**

1. Every program should have a comment at the top stating your name, student ID number, date created, approximate number of hours worked on, overall brief description of the program and any known bugs in it.
2. A brief function comment at the very start of each function you write.
3. Pieces of code could be commented briefly to explain what is being done.

**Global variables**

Global variables should be kept to a **minimum**. They should only be used when necessary.

**Important: Use constants where suitable in your programs**

**Use functions where suitable in your programs**

**Question 1.**

1. Write the pseudocode algorithm for this program. The program declares an array of 10 integers. It then fills the array with 10 random numbers in the range 8-22. It then

* displays on the screen the numbers within the array from index position 0 – 9.
* displays on the screen the numbers in reverse order from index position 9 - 0.
* displays every second number on the screen i.e. it displays the numbers at index 0, 2, 4, 6, 8.

1. Next code the program.
2. Test your code.

**Question 2.**

1. Write the pseudocode algorithm for this program. The program declares an array of 9 **floats**. It reads from the user the float values and fills the array. It then calculates and displays the number of odd and even numbers within the array.

**For example:** There are 3 even numbers and 6 odd numbers in the array.

1. Next code the program.
2. Test your code.

**Question 3.**

1. Write the pseudocode algorithm for this function. The function is called **arrayCalulations()** and declares an array of 8 integers. It should then read the 8 numbers from the user. The user must enter the numbers in the range from 1-10. An error message is displayed if the user enters a number outside this range. Your function should then display the list of numbers in the array on the screen and calculate the following information:
2. The largest number in the array.
3. The smallest number in the array.
4. The sum of numbers in the array.
5. The average number in the array.
6. Next code the program.
7. Test your code.

**Question 4.**

Modify Q3 above so that it tells you how many times the highest and lowest number appeared in the array.

**Question 5.**

The following array is declared in a function:

std::string temporaryItems[6];

How many strings can you store in the array *temporaryItems*?

What are the cells of the array initialized to?

------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------

**Question 6.**

In this question we are using two arrays. The ***namesArray*** is used to store the name of the players. The ***healthArray*** is used to store the health of the players.

The ***MAX\_PLAYERS*** variable is used to store the maximum number of players allowed in any game. The variable ***noOfPlayers*** stores the actual number of players playing in the game currently.

These variables (including the arrays) are global variables. What does this mean?

------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------

Given the following piece of code:

**// The following variables are GLOBAL variables**

const int MAX\_PLAYERS = 8; // maximum number of players

int noOfPlayers = 8; // the number of active players

**std::string namesArray[MAX\_PLAYERS];**

**int healthArray[MAX\_PLAYERS];**

void initializeArrays()

// Initialize the two arrays

{

namesArray[0] = "Tom";

namesArray[1] = "Sam";

namesArray[2] = "Ann";

namesArray[3] = "Ki";

namesArray[4] = "Joe";

namesArray[5] = "Ray";

namesArray[6] = "Dave";

namesArray[7] = "Sue";

healthArray[0] = 3;

healthArray[1] = 6;

healthArray[2] = 2;

healthArray[3] = 1;

healthArray[4] = 4;

healthArray[5] = 6;

healthArray[6] = 1;

healthArray[7] = 6;

}

**Write the pseudocode algorithms for each of the following 6 functions, then code and test them.**

**Important:** Write a ***separate*** ***function*** for each of the 6 parts.

1. Write a function called **calculateSum()** which calculates the **sum** of players health in the array and returns the sum from the function. The main() function should then display the sum of players health to the screen.
2. Write a function called **calculateAvg()** which calculates the **average** of the players health in the array and return the average from the function. The main() function should then display the average of players health to the screen.
3. Write a function called **displayPlayers( )** which displays the players names and their associated health values to the screen in sequential order within the function.

For example it would display: Player: Tom has health of 3, Player Sam has a health of 6, Player Ann has a health of 2 and so on.

1. Write a function called **displayReversePlayers( )** which displays the names and their associated health values in reverse within the function. For example it would display: Player Sue has a health of 6, Player Dave has a health of 1, Player Ray has a health of 6 and so on.
2. Write a function called **findHighestHealth( )** which finds the player with the **highest health** in the array and outputs the name of the player and their highest health within the function. **Note:** If there is more than one player with the highest health, output only the **last** player’s name.
3. Write a function called **findLowestHealth( )** which finds the player with the **lowest** health in the array and outputs the name of the player and their lowest health within the function. **Note:** If there is more than one player with the lowest health, output only the **first** player’s name.

**Question 7.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

Write a function called **findFirstHealthValue(int aValue)** which accepts a health value argument and then searches for the ***first occurrence*** of that particular health value in the array. If it finds it, it should return its index position within the array. If it is not in the array the function should return -1.

Appropriate messages should displayed.

Please note the main() function displays a message “Please enter a health value” and reads in a health value from the user. It then calls the function **findFirstHealthValue()** passing the health value to the function.

**For example** the program should execute as follows with the following input:

Please enter a health value: 6

Health value 6 found at index position 1 in the array.

OR

Please enter a health value: 7

Health value 7 is not found in the array.

**Question 8.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

Write a function called **findAllHealthValue(int aValue)** which accepts a health value argument and then searches for ***all occurrences*** of that particular health value in the array. If it finds it, it should output to the screen the list of players with that health value. If it is not in the array the function should output to the screen “No player found with that health value”.

Please note the main() function displays a message “Please enter a health value” and reads in a health value from the user. It then calls the function **findAllHealthValue()** passing the health value to the function.

For example if the health value 1 is input by the user, your program should output the player names Ki and Dave.

**Question 9.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

Write a function called **findLastName(std::string aName)** which accepts a name value argument and then searches for the ***last occurrence*** of that particular name in the array. If it finds it, it should return its index position within the array. If it is not in the array the function should return -1.

Appropriate messages should displayed.

Please note the **main()** function displays a message “Please enter a name” and reads in a name from the user. It then calls the function **findLastName** passing the name to the function.

For example the program should execute as follows with the following input:

Please enter a name: Ki

Name Ki found at index position 3 in the array.

OR

Please enter a name: Luke

Name Luke is not found in the array.

**Note:** To test this function properly you should add **duplicate** names to the array.

**Question 10.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

Write a function called **changePlayerHealth(t\_aName)** which accepts one string argument t\_aName. t\_aName is a player name. This function should ask the user what the players new health is. It should then find that player in the array and update their health to the new health inputted by the user.

If there are more than one player with the same name in the array, it should update the ***last*** player in the array with that name. The function should return the index position of the updated player. If the player is not in the array your function should return -1.

**Question 11.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

std::string namesArray[MAX\_PLAYERS];

int healthArray[MAX\_PLAYERS];

Write a function called **insertPlayerAtEnd(std::string t\_aName, int t\_newHealth)** which will insert a new player to the **end** of the global arrays namesArray and healthArray. The new player name is passed as an argument t\_aName and the new player health is passed as an argument t\_newHealth to the function.

The function should return the index position of where the new player was inserted within the array. If the array is full and the new player cannot be inserted into the array, the function should return -1.

**Question 12.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

Write a function called **deleteLastPlayer()** which will delete/remove the last player in the global arrays namesArray and healthArray. For example taking the example data in question 6, Sue would be deleted.

The function should return the index position of the deleted player within the array. If the array is empty and there are no players that can be deleted, the function should return -1.

**Question 13.**

**Modify question 6 to add the following function:**

The function **findPlayerHealth(std::string aName)** shouldfind the ***first*** player in the array which has the same name as aName. If there are more than one player with the same name in the array, it should only return details about the first player. The function should return a string stating what health that player has.

std::string **findPlayerHealth(std::string aName)**

// Exercise question

{

std::string aMessage = "";

bool found = false;

for (int index = 1; index <= noOfPlayers; index++)

{

if (namesArray[index] == aName)

{

found = true;

} // end if

} // end for

if (found == true)

{

aMessage = "Player " + aName + " has a health of " + std::to\_string(healthArray[index]);

}

else

{

aMessage = "There is no player called " + aName + " in the game.";

}

return aMessage;

}

* Does the code in the function above do what it is suppose to do?
* If it does, explain how the code works.
* If it does not, explain why it does not. Modify and write the code to make it work correctly.

**Question 14.**

Explain what the below algorithm does. Do you think that it will work as you would expect it to work? Can you see any problem with the way the algorithm is written?

const int MAX\_ITEMS = 10;

int arrayNum[] = { 4, 5, 1, 9, 20, 3, 4, 2, 28, 18 };

int main()

{

int index = 0;

int itemToBeFound = 2;

bool found = false;

std::string displayMessage = "";

while (!found && index < MAX\_ITEMS)

{ // start while

if (arrayNum[index] = itemToBeFound)

{

displayMessage = "Item " + std::to\_string(itemToBeFound) + " found at index " + std::to\_string(index);

found = true;

}// end if

index++;

} // end while

std::cout << displayMessage << std::endl;

system("Pause");

return 0;

}

**Question 15.**

Explain what the function **sumNumbers( )** does. Do you think that it will work as you would expect it to work? Can you see any problem with the way the function is written?

const int MAX\_ITEMS = 10;

int arrayNum[] = { 4, 5, 1, 9, 20, 3, 4, 2, 28, 18 };

void sumNumbers()

{

int sum = 0;

int index = 0;

for (index = MAX\_ITEMS; index >= 0; index = index-2)

{

sum = sum + arrayNum[index];

}

std::cout << "The sum is " + std::to\_string(sum) << std::endl;

}

**Question 16.**

**Modify question 6 to add the following function:**

**Write the pseudocode algorithm first, then code and test the function.**

Write a function called **findSecondLowest()** which will find the player with the ***second lowest health*** in the array and return a string with the name of the player and what his/her lowest health is. If there is more than one player in the array with the second lowest health, return the details of all players found with the second lowest health in the array.

For example the player with the second lowest health in the array is Ann with a health value of 2.

**Some Games using arrays**

**Question 1.**

**Nim Game:** The human plays against the computer. At the start of the game there are three piles of matches. In each pile there is a random number of matches in the range 1 to 20. The three piles are displayed throughout the game. A random choice determines who goes first. Players take it in turns to remove as many matches as they like from any pile, but only from one pile. A player must remove at least one match. The winner is the player who makes the other player take the last match. Make the computer play randomly, that is it chooses a pile randomly and then a number of matches randomly from those available.

**Question 2.**

**Safe Cracking:** A computer game which stores a 6 digit code in an array. The player is asked to crack the 6 digit code. The player inputs digits one by one. The game then checks whether they are correct. When a digit is entered, tell the player whether it is correct or not. If the player enters an incorrect digit, give the player 3 more tries before making him/her start from the beginning again. Hints could be given to the player to tell him/her that they are guessing too high or too low for each digit entered.

**Question 3.**

**BlackJack:** The game of casino blackjack or 21 is by far the most popular table game offered in gambling establishments. The basic premise of the game is that you want to have a hand value that is closer to 21 than that of the dealer, without going over 21. Your hand is strictly played out against the hand of the dealer. It is only necessary to write this game as a single user game.

The game starts by dealing the player and the dealer two cards each. The dealers’ cards are also shown on the screen. The dealers cards are dealt automatically. The player is then asked if they want to *hit* or *stand*. If either the dealer or the player goes bust the game is over. Your game should show all cards at the end of the game and tell the player whether they have won, drew or lost.