Description

The purpose of this lab is to check your understanding of the information presented in lectures 5a

# Part 1: Questions

Complete the assignment in Topic 5: Lab 5a named, "Lab 5a Questions", in Canvas.

# Part 2: Activity

## Section a: Working with arrays

## Review

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| An array allows you to store a group of items of the same data type together in memory.  A variable stores just a single value, and oftentimes can be cumbersome to work with when your program has similar values.  Values stored in an array are called elements. Each element has a subscript that makes it unique.  An array is defined as follows:  Declare Integer numbers[10]  Integer defines the type of numbers that can be stored, numbers is the name of the array, and [10] is how many numbers can be stored.  In most languages, the first element is assigned the subscript of 0, so the above array actually runs from 0 to 9.  Constant variables can also be used to declare the size of an array.  Constant Integer SIZE = 5  Declare Integer numbers[SIZE] = 847, 1238, 48, 123, 840  Elements in the array   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 847 | 1238 | 48 | 123 | 840 | | 0 | 1 | 2 | 3 | 4 |   Subscript or Index starting a 0  Loops are generally used to step through an array. This can be done using any type of loop and for any process such as filling, calculating, searching, sorting, or outputting elements of the array. |

### Problem i

Read the following programming problem prior to completing the lab.

The American Red Cross wants you to write a program that will calculate the average pints of blood donated during a blood drive. The program should take in the number of pints donated during the drive, based on a seven-hour drive period. The average pints donated during that period should be calculated and displayed. Additionally, the highest and the lowest number of pints donated should be determined and displayed. Write a loop around the program to run multiple times.

**Step 1: Declare the following variables**:

* An array named pints of the data type Real of size 7
* A variable named totalPints of the data type Real
* A variable named averagePints of the data type Real initialized to 0
* A variable named highPints of the data type Real initialized to 0
* A variable named lowPints of the data type Real initialized to 0

Module main()

//Declare local variables

Declare Constant SIZE

Set Constant SIZE = 7

Declare Real pints[SIZE]

Declare Real totalPints

Declare Real averagePints {0}

Declare Real highPints {0}

Declare Real lowPints {0}

Declare String again = “no”

While again == “no”

//module calls below

Call getPints(pints, SIZE)

Display “Do you want to run again: yes or no”

Input again

End While

End Module

Module getPints (Real pints[],Constant SIZE)

**Step 2:**  In the “main” module above, write a module call to a module named getPints that passes the pints array. Additionally, write a module header named getPints that accepts the pints array. (**Reference**: Passing an Array as an Argument to a Function, page 295).

**Step 3:** Write the “getPints” module with a for-loop that runs 7 times using the counter variable. Inside the for-loop, get the user’s input (Reals) for the 7 pints and store the values into the array. (Reference: Using a Loop to Step Through an Array, page 273).

// write getPints module here. Array of 7 Reals by reference

Declare integer counter

Set counter = 0

For counter = 0 to 6

Display “Enter the pints collected: “

Input userPints[counter]

End For

**Step 4:** Write a function call to a module named getTotal that passes the pints array and the totalPints variable. Additionally, write a function header named getTotal that accepts the pints array and the totalPints variable.

//Function call

Function Real getTotal(Real pints[SIZE], Real totalPints)

Set totalPints = \_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_)

//Function header

Function \_\_\_getTotal\_\_\_\_\_\_(Real \_\_\_\_pints[SIZE]\_\_\_\_\_\_\_\_\_\_, Real \_\_\_\_totalPints\_\_\_\_\_\_)

**Step 5:** Write a for loop that runs 7 times using the counter variable. Inside the for loop, total up the values of the array and store in the variable totalPints. Also, return the correct variable from the function. (Reference: Totaling the Values in an Array, page 289).

Declare Integer counter = 0

Set totalPints = 0

For \_\_\_\_\_\_counter\_\_\_\_\_\_\_\_\_\_\_\_ = 0 to \_\_\_\_\_\_\_6\_\_\_\_\_\_\_\_

Set \_\_\_\_\_totalPints\_\_\_\_ = \_\_\_\_totalPints\_\_\_\_ + \_\_\_\_\_\_pints\_\_\_\_\_\_\_\_[\_\_\_counter++\_\_\_\_]

End For

Return \_\_\_\_\_\_\_totalPints\_\_\_\_\_\_\_\_\_\_

**Step 6:** Write a function call to a module named getAverage that passes the totalPints variable and the averagePints variable. Additionally, write a function header named getAverage that accepts the totalPints variable and the averagePints variable.

//Function call

Function Real getAverage (Real totalPints, Real averagePints)

averagePints = \_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_Real totalPints\_\_\_\_\_, \_\_\_\_Real averagePints\_\_\_\_\_\_\_)

//Function header

Function \_\_\_Real getAverage\_\_\_\_\_\_(Real \_\_\_\_\_\_\_totalPints\_\_\_\_\_\_\_, Real \_\_\_\_\_averagePints\_\_\_\_\_\_)

**Step 7:** Write a statement that will calculate the average pints donated over the drive period. Also, return the correct variable from the function. (Reference: Averaging the Values in an Array, page 290).

averagePints = \_\_\_\_\_\_\_\_totalPints\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_SIZE\_\_\_\_\_\_\_\_\_\_

Return \_\_\_\_\_\_\_\_\_averagePints\_\_\_\_\_\_\_\_\_\_\_\_\_

**Step 8:** Write a function call to a module named getHigh that passes the highPints variable and the pints array. Additionally, write a function header named getHigh that accepts the highPints variable and the pints array.

//Function call

Function Real getHigh (Real highPints, array[SIZE])

highPints = \_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_Real highPints\_\_\_\_\_\_\_\_, \_\_\_\_\_array[SIZE]\_\_\_\_\_\_)

//Function header

Function \_Real\_\_\_\_getHigh\_\_\_\_(Real \_\_\_\_\_highPints\_\_\_\_\_\_\_\_, Real \_\_\_\_\_\_array\_\_\_\_\_[SIZE ])

**Step 9:** Write the code that will determine the highest value in an array. Also, return the correct variable from the function. (Reference: Finding the Highest Value in an Array, page 291).

Set highPints = pints[\_\_\_0\_\_\_\_\_]

Set index = 1

For index = 1 to 6

If \_\_\_\_\_\_pints\_\_\_\_\_\_\_\_\_[\_\_\_index\_\_\_\_] > highPints Then

Set \_\_\_\_\_highPints\_\_\_\_\_\_\_ = \_\_\_\_\_\_pints\_\_\_\_[\_\_\_index\_\_\_\_]

End If

End For

Return \_\_\_\_\_\_\_highPints\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Step 10:** Write a function call to a module named getLow that passes the lowPints variable and the pints array. Additionally, write a function header named getLow that accepts the lowPints variable and the pints array.

//Function call

Function Real getLow ( Real lowPints, pints[SIZE])

lowPints = \_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_Real lowPints\_\_\_\_\_\_\_, \_\_\_array[SIZE]\_\_\_\_\_\_\_\_)

//Function header

Function \_\_\_Real getLow\_\_\_\_\_\_(Real \_\_\_\_lowPints\_\_\_\_\_\_\_\_\_, Real \_\_\_\_array\_\_\_\_\_\_\_[SIZE ])

**Step 11:** Write the code that will determine the highest value in an array. Also, return the correct variable from the function. (Reference: Finding the Lowest Value in an Array, page 293).

Set lowPints = pints[\_\_\_0\_\_\_\_\_]

Set index = 1

For index = 1 to 6

If \_\_\_\_\_\_pints\_\_\_\_\_\_\_\_\_[\_\_\_index\_\_\_\_] < lowPints Then

Set \_\_\_\_lowPints\_\_\_\_\_\_\_\_ = \_\_\_pints\_\_\_\_\_\_\_[\_\_index\_\_\_\_\_]

End If

End For

Return \_\_\_\_\_\_\_\_lowPints\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Step 12:** Write a module call to a module named displayInfo. Pass the necessary variable to the functions that are needed to display the averagePints, the highPints, and the lowPints. Also, write the module header that accepts the same variables.

//Module call

Call \_\_\_\_\_displayInfo\_\_\_\_\_\_\_(\_\_\_\_\_\_\_Real averagePints\_\_\_\_\_\_\_, \_\_\_\_\_Real highPints\_\_\_\_\_\_, \_\_\_\_\_real lowPints\_\_\_\_\_\_)

//Module header

Module \_\_\_\_displayInfo\_\_\_\_(Real \_\_\_\_averagePints\_\_\_\_, Real \_\_\_highPints\_\_\_\_\_, Real \_\_lowPints\_\_\_\_\_)

### Problem ii

Using the program from [problem i](#_Problem_i), complete the following checks for a better understanding of your work.

**Step 1:** Imagine the following number of pints were entered into the array.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Element | 34 | 39 | 25 | 18 | 43 | 31 | 12 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

**Step 2:** Recall Problem i, Step 5 accumulates the pints collected.

Declare Integer counter = 0

Set totalPints = 0

For counter = 0 to 6

Set totalPints = totalPints + pints[counter]

End For

**Step 3:** Complete the following chart by writing what the counter and the totalPints value stores on each iteration of the loop.

|  |  |
| --- | --- |
| **Counter** | **totalPints** |
| 0 | 34 |
| 1 | 73 |
| 2 | 98 |
| 3 | 116 |
| 4 | 159 |
| 5 | 190 |
| 6 | 202 |

**Step 4:** Recall Problem i, Step 9 that determines the high value.

Set highPints = pints[0]

Set index = 1

For index = 1 to 6

If pints[index] > highPints Then

Set highPints = pints[index]

End If

End For

**Step 5:** Complete the following chart by writing what the highPints and the pints array value stores on each iteration of the loop. Also conclude whether it will be True or False.

|  |  |  |
| --- | --- | --- |
| **Pints** | **highPints** | **True or False** |
| 39 | 34 | TRUE |
| 25 | 39 | FALSE |
| 18 | 39 | FALSE |
| 43 | 39 | TRUE |
| 31 | 43 | FALSE |
| 12 | 43 | FALSE |

**Step 6:** Recall Problem i, Step 11 that determines the low value.

Set lowPints = pints[0]

Set index = 1

For index = 1 to 6

If pints[index] < lowPints Then

Set lowPints = pints[index]

End If

End For

**Step 7:** Complete the following chart by writing what the lowPints and the pints array value stores on each iteration of the loop. Also conclude whether it will be True or False.

|  |  |  |
| --- | --- | --- |
| **Pints** | **lowPints** | **True or False** |
| 39 | 34 | FALSE |
| 25 | 34 | TRUE |
| 18 | 25 | TRUE |
| 43 | 18 | FALSE |
| 31 | 18 | FALSE |
| 12 | 18 | TRUE |

Add an assignment statement that sets counter to 2. This refers to the second location in the array.

* Add an assignment statement that sets lowPints to the 1 index of the pints array.
* Add a loop that iterates 7 times.
* Inside the loop, add a selection statement that determines if pints in the counter location is less than lowPints.
* If that is true, then set lowPints to pints in the counter location.
* Increment counter by 1.

**Step 9:** Add the displayInfo( ) module in main. Go to the display() module and add the following inside the module:

* Display the averagePints variable
* Display the highPints variable
* Display the lowPints variable

**Step 10:** Using the following input values, check your results. If there are errors, verify steps 1 through 10.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Element | 34 | 39 | 25 | 18 | 43 | 31 | 12 |

Output should be as follows:

The average pints collected: 28.8571

The highest amount was: 43

The lowest amount was: 12

### Problem iii

### Review

The goal of this lab is to write the blood drive program in C++ based on the pseudocode. The source code file should be called, "blood\_drive.cpp".

**Note: write the pseudocode first the write you C++ code by following the pseudocode.**

Your code should produce the following output (user input in bold).

**Sample output:**

Enter pints collected: **43**

Enter pints collected: **25**

Enter pints collected: **64**

Enter pints collected: **35**

Enter pints collected: **19**

Enter pints collected: **37**

Enter pints collected: **46**

The average number of pints donated is 38.4285714286

The highest pints donated is 64

The lowest pints donated is 19

Do you want to end program? (y/n): **y**

Problem iv

Write the pseudocode then the C++ program for the problem description below. The pseudo code should be written in this document and the C++ program in a file called, "going\_green.cpp".

Last year, a dorm at a local college implemented rooftop gardens to promote energy efficiency and save money. Write a program that will allow the user to enter the energy bills from January to December for the year prior to going green (e.g. 2019). Next, allow the user to enter the energy bills from January to December of the year after going green (e.g. 2020). The program should calculate the energy difference from the two years and display the two years' worth of data, along with the savings.

**Be sure to modularize your code.**

**Resist the urge to program first. Write the pseudocode first.**

Hints: Create three arrays of size 12 each. The first array will store the non-green year of energy costs, the second array will store the year after going green. **These two arrays could be consolidated to a 2d array**. The third array will store the month names. The savings is the difference between the two. You can use integers for the money numbers. You will need to take as input the year before going green and the year after.

Your sample output might look as follows:

Welcome to the GREEN Savings Calculator

Enter year before going green (yyyy): **2019**

Enter energy costs for…

January $: **789**

February $: **790**

March $: **890**

April $: **773**

May $: **723**

June $: **759**

July $: **690**

August $: **681**

September $: **782**

October $: **791**

November $: **898**

December $: **923**

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

Enter year after going green (yyyy): **2020**

Enter energy costs for…

January $: **546**

February $: **536**

March $: **519**

April $: **493**

May $: **472**

June $: **432**

July $: **347**

August $: **318**

September $: **453**

October $: **489**

November $: **439**

December $: **516**

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

SAVINGS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SAVINGS 2019 2020 MONTH

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

$ 243 $ 789 $ 546 January

$ 254 $ 790 $ 536 February

$ 371 $ 890 $ 519 March

$ 280 $ 773 $ 493 April

$ 251 $ 723 $ 472 May

$ 327 $ 759 $ 432 June

$ 343 $ 690 $ 347 July

$ 363 $ 681 $ 318 August

$ 329 $ 782 $ 453 September

$ 302 $ 791 $ 489 October

$ 459 $ 898 $ 439 November

$ 407 $ 923 $ 516 December

Do you want to end program? (y/n): **y**

**Note, the years in the output table header are the years taken from user input.**

**Enter pseudocode in this box. It will grow as you enter information.**

Module main()

// ToDo

Declare Constant SIZE

Set Constant SIZE = 12

Declare Real beforeGreen[SIZE]

Declare Real afterGreen[SIZE]

Declare Real savings[SIZE]

Declare String months[SIZE] – January, February, March, April, May, June, July, August, September, October, November, December

Call greeting

Declare String response

Set response = “n”

While response == “n” Then

Declare Real firstYear

Set firstYear = Function Real getYear( String “Enter year before going green (yyyy) : “)

Function Real getSpendings(Real beforeGreen[],Real months[], Constant SIZE)

Declare Real secondYear

Set secondYear = Function Real getYear( String “Enter year after going green (yyyy) : “)

Function Real getSpendings(Real afterGreen[],Real months[], Constant SIZE)

Function Real calcSavings (Real beforeGreen[], Real afterGreen[], Constant SIZE)

Call displayInfo(Real beforeGreen[], Real afterGreen[], String months[], Real savings [], Constant SIZE, Real firstYear, Real secondYear)

Display “Do you want to end the program? (y/n) : “

Input response

End While

End Module

Module greeting()

Display “Welcome to the GREEN Saving Calculator”

End Module

Function Real getYear(String prompt)

Declare Real year

Display prompt

Input year

Return year

End Function

Function Real getSpendings( Real spendings[], Constant String months[], Constant SIZE)

Display “Enter the energy costs for …”

Declare counter

Set counter = 0

For counter to SIZE-1

Display months[counter] “$: “

Input spendings[counter]

End For

End Function

Function Real calcSavings(Real beforeGreen[], Real afterGreen[], Real savings[], Constant SIZE)

Declare counter = 0

For counter to SIZE-1

Set savings[counter] = beforeGreen[counter] – afterGreen[counter]

End For

End Function

Module displayInfo(Real beforeGreen[], Real afterGreen[], Constant String months[], Real savings[], Constant SIZE, Real firstYear, Real secondYear)

Display “ SAVINGS “

Display “\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”

Display “SAVINGS “, firstYear “ “, secondYear “ MONTH“

Display “\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Declare counter = 0

For counter to SIZE-1

Display “$”, savings[counter] “ $“, beforeGreen[counter] “ $”, afterGreen[counter] “ “, months[counter]

End For

End Module

# What to Submit for the Lab 5a Activity

* Complete the problems in this file and submit it.
* Your C++ source code files
  + blood\_drive.cpp
  + going\_green.cpp
* A screenshot of your program's output for the first test values listed above.
  + This screenshot should not include your desktop or any other information, just the editor that you wrote the program in and the output after running it.