|  |
| --- |
| **Software Engineering Department - ITU** |
| **SE101T: Programming Fundamentals Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 9/10/2023** |
| **Teaching Assistant: Abeera Ashraf** | **Semester: Fall 2023** |
| **Teaching Assistant: Aasma Waheed** | **Batch: BSSE2023** |

# **Lab 7. Arrays in C++ Solution**

***Q1. Write a program in a function named q1, to read five values into an array and print them out in reverse order.***

|  |
| --- |
| #include <iostream>  void q1() {  // Initialize an array to store five values  int values[5];  // Read five values into the array  for (int i = 0; i < 5; ++i) {  std::cout << "Enter value " << i + 1 << ": ";  std::cin >> values[i];  }  // Print the array in reverse order  std::cout << "Values in reverse order:" << std::endl;  for (int i = 4; i >= 0; --i) {  std::cout << values[i] << std::endl;  }  }  int main() {  // Call the function  q1();  return 0;  } |

***Q2. Write a program in a function named q2, to linearly search a number in a 1-d array, take array, size of array, & value to be searched as argument.***

|  |
| --- |
| #include <iostream>  // Function to perform linear search  int q2(int arr[], int size, int value) {  for (int i = 0; i < size; ++i) {  if (arr[i] == value) {  // Return the index if the value is found  return i;  }  }  // Return -1 if the value is not found in the array  return -1;  }  int main() {  // Declare an array  int array[] = {10, 20, 30, 40, 50};  // Calculate the size of the array  int size = sizeof(array) / sizeof(array[0]);  // Get the value to be searched  int searchValue;  std::cout << "Enter the value to be searched: ";  std::cin >> searchValue;  // Call the q2 function and store the result  int result = q2(array, size, searchValue);  // Check the result and print the appropriate message  if (result != -1) {  std::cout << "Value found at index " << result << std::endl;  } else {  std::cout << "Value not found in the array" << std::endl;  }  return 0;  } |

***Q3. Write a program in a function named q3, to create a recursive function to find the maximum element in an array.***

|  |
| --- |
| #include <iostream>  // Recursive function to find the maximum element in an array  int findMaxRecursive(int arr[], int start, int end) {  // Base case: If the array has only one element, return that element  if (start == end) {  return arr[start];  }  // Recursive case: Divide the array into two halves and find the maximum  int mid = (start + end) / 2;  int maxLeft = findMaxRecursive(arr, start, mid);  int maxRight = findMaxRecursive(arr, mid + 1, end);  // Compare and return the maximum of the two halves  return (maxLeft > maxRight) ? maxLeft : maxRight;  }  // Function to initialize the recursive function  int q3(int arr[], int size) {  // Check for the base case of an empty array  if (size == 0) {  std::cerr << "Error: Cannot find maximum element in an empty array." << std::endl;  return -1; // Return an error value  }  // Call the recursive function with the appropriate arguments  return findMaxRecursive(arr, 0, size - 1);  }  int main() {  // Declare an array  int array[] = {5, 8, 3, 1, 9, 6, 7};  // Calculate the size of the array  int size = sizeof(array) / sizeof(array[0]);  // Call the q3 function and store the result  int maxElement = q3(array, size);  // Print the maximum element  std::cout << "Maximum element in the array: " << maxElement << std::endl;  return 0;  } |

***Q4. Write a program in a function named q4, to add two arrays A and B of size m x n.***

|  |
| --- |
| #include <iostream>  // Recursive function to find the maximum element in an array  int findMaxRecursive(int arr[], int start, int end) {  // Base case: If the array has only one element, return that element  if (start == end) {  return arr[start];  }  // Recursive case: Divide the array into two halves and find the maximum  int mid = (start + end) / 2;  int maxLeft = findMaxRecursive(arr, start, mid);  int maxRight = findMaxRecursive(arr, mid + 1, end);  // Compare and return the maximum of the two halves  return (maxLeft > maxRight) ? maxLeft : maxRight;  }  // Function to initialize the recursive function  int q3(int arr[], int size) {  // Check for the base case of an empty array  if (size == 0) {  std::cerr << "Error: Cannot find maximum element in an empty array." << std::endl;  return -1; // Return an error value  }  // Call the recursive function with the appropriate arguments  return findMaxRecursive(arr, 0, size - 1);  }  int main() {  // Declare an array  int array[] = {5, 8, 3, 1, 9, 6, 7};  // Calculate the size of the array  int size = sizeof(array) / sizeof(array[0]);  // Call the q3 function and store the result  int maxElement = q3(array, size);  // Print the maximum element  std::cout << "Maximum element in the array: " << maxElement << std::endl;  return 0;  } |

### Assessment Rubric for Lab

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & Github Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_