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

| **Course Instructor: Usama Bin Shakeel** | **Dated: 30/10/2023** |
| --- | --- |
| **Teaching Assistant: Abeera Ashraf** | **Semester: Fall 2023** |
| **Teaching Assistant: Aasma Waheed** | **Batch: BSSE2023** |

# **Lab 11. Problem solving using structures in C++**

| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case of ambiguity seek help from the course instructor, lab engineers, assigned teaching assistants

**Lab Tasks**

**Task 1:** Accept the assignment posted in Google Classroom and after accepting clone the repository to your computer for this ensure you have logged into github app with your account.

**Task 2:** Solve the given problems written after task instructions, write code through IDE like CLion

**Task 3:** Ensure your code/solution is in the cloned folder.

**Task 4:** Commit and Push the changes through the Github App

**Task 5:** Run ‘make run’ to run C++ code

Write code in functions, after completing each part, verify through running code using **“make run”** on cygwin

***Q1. Write a struct named Frequency, this will have two attributes, one char array of size 30 and second an integer for count of the word in the document***

| #include <iostream>  const int MAX\_SIZE = 30;  struct Frequency {  char word[MAX\_SIZE];  int count;  };  int main() {  // Example usage of the Frequency struct  Frequency frequencyInstance;  // Assign values to the attributes  std::cout << "Enter a word: ";  std::cin >> frequencyInstance.word;  std::cout << "Enter the count for the word: ";  std::cin >> frequencyInstance.count;  // Display the values  std::cout << "Word: " << frequencyInstance.word << std::endl;  std::cout << "Count: " << frequencyInstance.count << std::endl;  return 0;  } |
| --- |

***Q2. Write a struct named “GrowingArray”, this will have three attributes in it. First is the pointer of Frequency created in Q1, second is the size of array and third is the number of occupied elements in the array.***

| #include <iostream>  const int MAX\_SIZE = 30;  // Frequency struct definition  struct Frequency {  char word[MAX\_SIZE];  int count;  };  // GrowingArray struct definition  struct GrowingArray {  Frequency\* frequencyArray;  int size;  int occupiedElements;  };  int main() {  // Example usage of the GrowingArray struct  GrowingArray growingArrayInstance;  // Initialize the size and allocate memory for the array  growingArrayInstance.size = 5; // You can change the size as needed  growingArrayInstance.frequencyArray = new Frequency[growingArrayInstance.size];  // Initialize the occupied elements  growingArrayInstance.occupiedElements = 0;  // Assign values to the first element in the array  std::cout << "Enter a word: ";  std::cin >> growingArrayInstance.frequencyArray[0].word;  std::cout << "Enter the count for the word: ";  std::cin >> growingArrayInstance.frequencyArray[0].count;  // Update the occupied elements  growingArrayInstance.occupiedElements++;  // Display the values  std::cout << "Word: " << growingArrayInstance.frequencyArray[0].word << std::endl;  std::cout << "Count: " << growingArrayInstance.frequencyArray[0].count << std::endl;  // Deallocate memory for the array  delete[] growingArrayInstance.frequencyArray;  return 0;  } |
| --- |

***Q3. Write a function named “push” in GrowingArray, which will increase the size of array if it is full, and in any case append the new element at the end***

| #include <iostream>  const int MAX\_SIZE = 30;  // Frequency struct definition  struct Frequency {  char word[MAX\_SIZE];  int count;  };  // GrowingArray struct definition  struct GrowingArray {  Frequency\* frequencyArray;  int size;  int occupiedElements;  };  // Function to push a new element into the GrowingArray  void push(GrowingArray& growingArray, const char newWord[MAX\_SIZE], int newCount) {  // Check if the array is full  if (growingArray.occupiedElements == growingArray.size) {  // Increase the size of the array (for simplicity, doubling the size)  int newSize = growingArray.size \* 2;  // Create a new array with the increased size  Frequency\* newFrequencyArray = new Frequency[newSize];  // Copy the elements from the old array to the new array  for (int i = 0; i < growingArray.size; ++i) {  newFrequencyArray[i] = growingArray.frequencyArray[i];  }  // Deallocate memory for the old array  delete[] growingArray.frequencyArray;  // Update the GrowingArray attributes  growingArray.frequencyArray = newFrequencyArray;  growingArray.size = newSize;  }  // Append the new element at the end  std::strcpy(growingArray.frequencyArray[growingArray.occupiedElements].word, newWord);  growingArray.frequencyArray[growingArray.occupiedElements].count = newCount;  // Update the occupied elements  growingArray.occupiedElements++;  }  int main() {  // Example usage of the GrowingArray struct  GrowingArray growingArrayInstance;  // Initialize the size and allocate memory for the array  growingArrayInstance.size = 2; // Initial size, you can change this as needed  growingArrayInstance.frequencyArray = new Frequency[growingArrayInstance.size];  // Initialize the occupied elements  growingArrayInstance.occupiedElements = 0;  // Example usage of the push function  push(growingArrayInstance, "apple", 3);  push(growingArrayInstance, "orange", 5);  // Display the values  for (int i = 0; i < growingArrayInstance.occupiedElements; ++i) {  std::cout << "Word: " << growingArrayInstance.frequencyArray[i].word << ", Count: "  << growingArrayInstance.frequencyArray[i].count << std::endl;  }  // Deallocate memory for the array  delete[] growingArrayInstance.frequencyArray;  return 0;  } |
| --- |

***Q4. Write a function named “uniqueWords” to determine unique words from a file, this will use the work done in Q1-Q3.***

| #include <iostream>  #include <fstream>  #include <sstream>  #include <cstring>  const int MAX\_SIZE = 30;  // Frequency struct definition  struct Frequency {  char word[MAX\_SIZE];  int count;  };  // GrowingArray struct definition  struct GrowingArray {  Frequency\* frequencyArray;  int size;  int occupiedElements;  };  // Function to push a new element into the GrowingArray  void push(GrowingArray& growingArray, const char newWord[MAX\_SIZE], int newCount) {  // Check if the array is full  if (growingArray.occupiedElements == growingArray.size) {  // Increase the size of the array (for simplicity, doubling the size)  int newSize = growingArray.size \* 2;  // Create a new array with the increased size  Frequency\* newFrequencyArray = new Frequency[newSize];  // Copy the elements from the old array to the new array  for (int i = 0; i < growingArray.size; ++i) {  newFrequencyArray[i] = growingArray.frequencyArray[i];  }  // Deallocate memory for the old array  delete[] growingArray.frequencyArray;  // Update the GrowingArray attributes  growingArray.frequencyArray = newFrequencyArray;  growingArray.size = newSize;  }  // Append the new element at the end  std::strcpy(growingArray.frequencyArray[growingArray.occupiedElements].word, newWord);  growingArray.frequencyArray[growingArray.occupiedElements].count = newCount;  // Update the occupied elements  growingArray.occupiedElements++;  }  // Function to determine unique words from a file  void uniqueWords(const std::string& filename, GrowingArray& uniqueWordsArray) {  // Open the file for reading  std::ifstream inFile(filename);  // Check if the file is opened successfully  if (!inFile.is\_open()) {  std::cerr << "Error opening the file." << std::endl;  return;  }  std::string line;  while (std::getline(inFile, line)) {  std::istringstream iss(line);  std::string word;  while (iss >> word) {  // Check if the word is already in the uniqueWordsArray  int existingIndex = -1;  for (int i = 0; i < uniqueWordsArray.occupiedElements; ++i) {  if (std::strcmp(uniqueWordsArray.frequencyArray[i].word, word.c\_str()) == 0) {  existingIndex = i;  break;  }  }  // If the word is not found, add it to the array  if (existingIndex == -1) {  push(uniqueWordsArray, word.c\_str(), 1);  } else {  // If the word is found, increment its count  uniqueWordsArray.frequencyArray[existingIndex].count++;  }  }  }  // Close the file  inFile.close();  }  int main() {  // Example usage of the GrowingArray struct  GrowingArray uniqueWordsArray;  // Initialize the size and allocate memory for the array  uniqueWordsArray.size = 5; // Initial size, you can change this as needed  uniqueWordsArray.frequencyArray = new Frequency[uniqueWordsArray.size];  // Initialize the occupied elements  uniqueWordsArray.occupiedElements = 0;  // Example usage of the uniqueWords function  uniqueWords("example.txt", uniqueWordsArray);  // Display the unique words and their frequencies  for (int i = 0; i < uniqueWordsArray.occupiedElements; ++i) {  std::cout << "Word: " << uniqueWordsArray.frequencyArray[i].word << ", Count: "  << uniqueWordsArray.frequencyArray[i].count << std::endl;  }  // Deallocate memory for the array  delete[] uniqueWordsArray.frequencyArray;  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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_