Assignment 2: Milestone 2

CS346: Software Engineering Lab

Group 8B

Technical Documentation

Task:

Develop a Software to teach students fundamental concepts of "Data Structures" using step by step explanation. Basic topics must be covered like Searching, Sorting, Arrays, Stacks, Queues, Linked Lists, etc. Learning modules must be made with proper step by step procedures while making use of some graphics to make learning more interesting. Some small quizzes could also be included.

Instructor:

Prof. Pradip K. Das

Head - Centre for Drone Technology, IITG

Contents

1 Project Description

The provided code represents a Windows Forms application of a Data Structures Learning Software implemented in VB.NET. The application uses MySQL for data storage and retrieval.

The Data Structures Learning Software is a comprehensive tool designed to teach students fundamental concepts of data structures such as searching, sorting, arrays, stacks, queues, linked lists, etc. It provides step-by-step explanations, interactive learning *modules*, graphics, and quizzes to enhance the learning experience.

1.1 Brief Overview

Key features of the Data Structures Learning Software include:

- Step-by-step explanations of fundamental data structure concepts.
- Interactive learning modules with visualizations and demonstrations.
- Quizzes to test comprehension and reinforce learning.
- Progress tracking to monitor learning achievements over time.

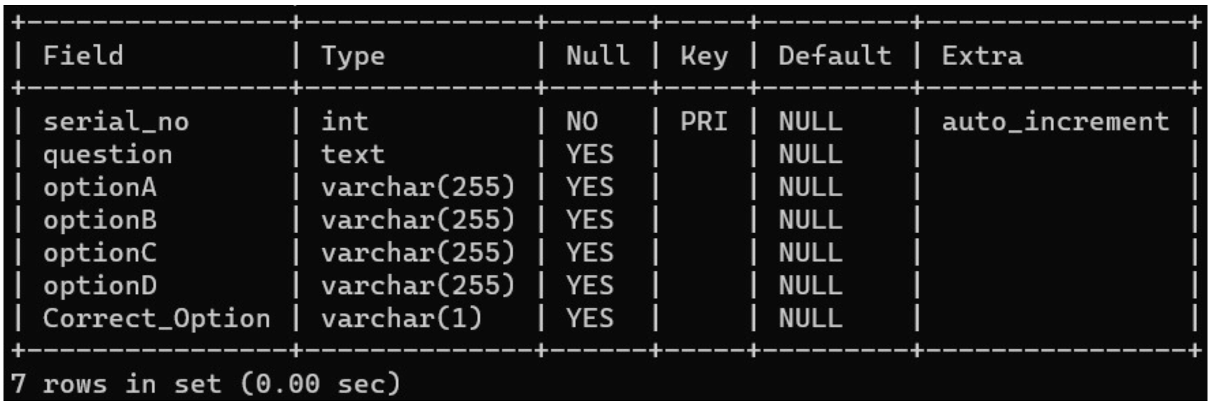
The Data Structures Learning Software covers the following topics:

1. Searching Algorithms: Linear Search, Binary Search
2. Sorting Algorithms: Bubble Sort, Insertion Sort, Merge Sort
3. Arrays
4. Stacks
5. Queues
6. Linked Lists

Each learning module includes instructional content, interactive demonstrations, and quizzes tailored to the specific topic.

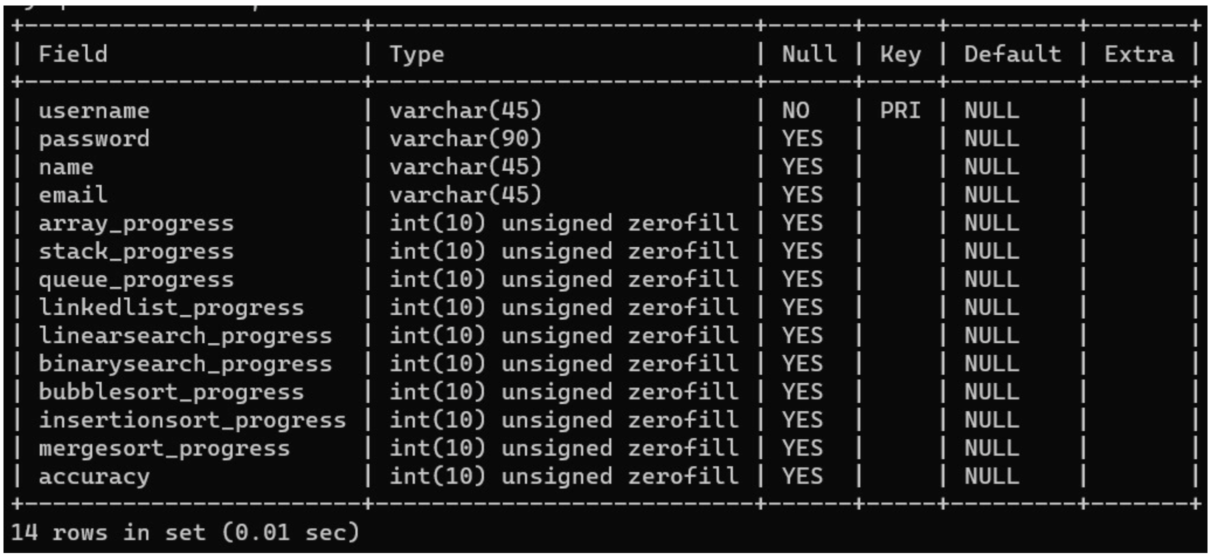
2 The Database Schema

- The database schema includes a table named queue containing quiz questions and options.



```
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| serial_no      | int          | NO   | PRI | NULL    | auto_increment |
| question       | text         | YES  |     | NULL    |                |
| optionA        | varchar(255) | YES  |     | NULL    |                |
| optionB        | varchar(255) | YES  |     | NULL    |                |
| optionC        | varchar(255) | YES  |     | NULL    |                |
| optionD        | varchar(255) | YES  |     | NULL    |                |
| Correct_Option | varchar(1)   | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

- It also includes a table named users to store user information and quiz progress.



```
+-----------------------+---------------------------+------+-----+---------+-------+
| Field                 | Type                      | Null | Key | Default | Extra |
+-----------------------+---------------------------+------+-----+---------+-------+
| username              | varchar(45)               | NO   | PRI | NULL    |       |
| password              | varchar(90)               | YES  |     | NULL    |       |
| name                  | varchar(45)               | YES  |     | NULL    |       |
| email                 | varchar(45)               | YES  |     | NULL    |       |
| array_progress        | int(10) unsigned zerofill | YES  |     | NULL    |       |
| stack_progress        | int(10) unsigned zerofill | YES  |     | NULL    |       |
| queue_progress        | int(10) unsigned zerofill | YES  |     | NULL    |       |
| linkedlist_progress   | int(10) unsigned zerofill | YES  |     | NULL    |       |
| linearsearch_progress | int(10) unsigned zerofill | YES  |     | NULL    |       |
| binarysearch_progress | int(10) unsigned zerofill | YES  |     | NULL    |       |
| bubblesort_progress   | int(10) unsigned zerofill | YES  |     | NULL    |       |
| insertionsort_progress| int(10) unsigned zerofill | YES  |     | NULL    |       |
| mergesort_progress    | int(10) unsigned zerofill | YES  |     | NULL    |       |
| accuracy              | int(10) unsigned zerofill | YES  |     | NULL    |       |
+-----------------------+---------------------------+------+-----+---------+-------+
14 rows in set (0.01 sec)
```

3 Code Structure

The code for modularity purposes is divided into the following classes.

3.1 Arrays

Arrays is one of the basic data structures taught in Computer Science. Here, we will be focusing on the basic features of array, irrespective of the language dependent features. There are three input variables, Size, Value and Index. Size represents the size of the array declared. Index and Value represent the index of the array and the value to be initialized or modified with. When the user enters the Size and clicks on the Declare button, an array of size Size is declared, and it is shown on the panel with the help of labels. The user can further initialize and modify the value of any index within the bounds by entering the values of Index and Value and clicking on the Initialize button. We have added an extra feature which is, Insertion. This is a language-dependent feature. In this, a Value can be inserted at a given Index. This inserts the value at the index and then shifts all the others by one spot. Error handling is done by taking care of empty input and input other than integers.

3.2 Stack

Here, for the visualization part of stack, buttons for push, peek, pop functionalities are present. User inputs text in the txtinput box and presses the push button, on which the value gets entered into the stack and a text box containing the number will get shown in the panel .For the next input the new text box comes in place of the old one and the old text box moves downward and for the pop the whole process will occur in reverse so the new text box value will get shown as popped value. For peek, we just show the new text box value without making any changes to the stack and finally the clear stack button clears all text boxes shown and deletes the stack and clears all textboxes.

3.3 Queue

Here, the queue has enqueue ,dequeue operations and it has two pointers for front and rear part of the queue. To show the working of queue buttons for enqueue ,dequeue, front ,rear and clear queue buttons, if the enqueue button is pressed, the txt input which is given will get added to queue and a text box containing the number will get shown in the panel. For the next input, the new text box comes in place of the old one and the old text box moves downward and for the dequeue the old text box will get deleted and get shown as dequeued value and the front button gives the value the front pointer is showing and the rear button shows the value of rear pointer and clear button will just delete everything and sets back.

3.4 Linked List

The entire program is divided into three parts. The first part explains what linked list is. It entails details about the types of linked list and the basic functions associated with the linked list. It also provides a graphical visualization of how the linked list is stored. The second part of the code provides dynamic visualization of the linked list with functions such as insert, delete, append and search in the linked list. Insert adds given value at given index. Delete finds the first occurrence of the value in linked list and deletes it with corresponding message. Appropriate message is displayed if value is not present. Append adds value at end of linked list while searching finds the index of an element of linked list by linear search algorithm. Also, a clear functionality is provided to erase the entire linked list. The third section contains the quiz where the knowledge of the user regarding linked list is checked.

3.5 Linear Search

This leaf form consists of 3 sections, namely, the textual section, the visualization section, and the quiz section, wrap in a parent panel which is scrollable. The Text section contains information on linear search algorithm, its pseudocode, advantages, disadvantages, etc. Next Section contains a visualization panel which simulates the working of the linear search algorithm, with buttons for initializing array, search, stop and reset. It takes both the list of elements (comma-separated integers) and the element (integer) to be searched as inputs from the user. The Stop button is used to stop the search process or resume it and the Reset button is used to clear the entire visualization. The Quiz section consists of 10 questions selected randomly from the database, with Previous and Next buttons, and finally Reset, Submit, and Back buttons. The Reset button fetches another random set of questions from the database.

3.6 Binary Search

This leaf form is very similar to the Linear Search form with the textual section containing information about Binary Search algorithm. The Visualization simulates the binary search algorithm, highlighting the current subarray on which it is searching. It also handles unsorted arrays as input, in which case the array is sorted before proceeding. Questions on binary search are also present at the bottom. To maintain consistency in the UI, at most 40 elements are allowed for initializing array for both linear and binary searches.

3.7 Bubble Sort

This leaf form consists of 3 sections, namely, the textual section, the visualization section, and the quiz section, wrap in a parent panel which is scrollable. The Text section contains information on bubble sort algorithm and its pseudocode . Next Section contains a visualization panel which simulates the working of the bubble sort algorithm, with buttons for creating array, step, reset and fill random. It takes an array to be sorted from the user. The user can fill array manually in text boxes or fill it randomly after creating the array. The Reset button is used to clear the entire visualization. The Quiz section consists of 10 questions selected randomly from the database, with Previous and Next buttons, and finally Reset, Submit, and Back buttons. The Reset button fetches another random set of questions from the database.

3.8 Insertion Sort

This leaf form consists of 3 sections, namely, the textual section, the visualization section, and the quiz section, wrap in a parent panel which is scrollable. The Text section contains information on insertion sort algorithm and its pseudocode . Next Section contains a visualization panel which simulates the working of the bubble sort algorithm, with buttons for creating array, step, reset and fill random. It takes an array to be sorted from the user. The user can fill array manually in text boxes or fill it randomly after creating the array. The Reset button is used to clear the entire visualization. The Quiz section consists of 10 questions selected randomly from the database, with Previous and Next buttons, and finally Reset, Submit, and Back buttons. The Reset button fetches another random set of questions from the database.

3.9 Merge Sort

This leaf form consists of 3 sections, namely, the textual section, the visualization section, and the quiz section, wrap in a parent panel which is scrollable. The Text section contains information on merge sort algorithm and its pseudocode . Next Section contains a visualization panel which simulates the working of the bubble sort algorithm, with buttons for creating array, step, reset and fill random. It takes an array to be sorted from the user. The user can fill array manually in text boxes or fill it randomly after creating the array. The Reset button is used to clear the entire visualization. The Quiz section consists of 10 questions selected randomly from the database, with Previous and Next buttons, and finally Reset, Submit, and Back buttons. The Reset button fetches another random set of questions from the database.

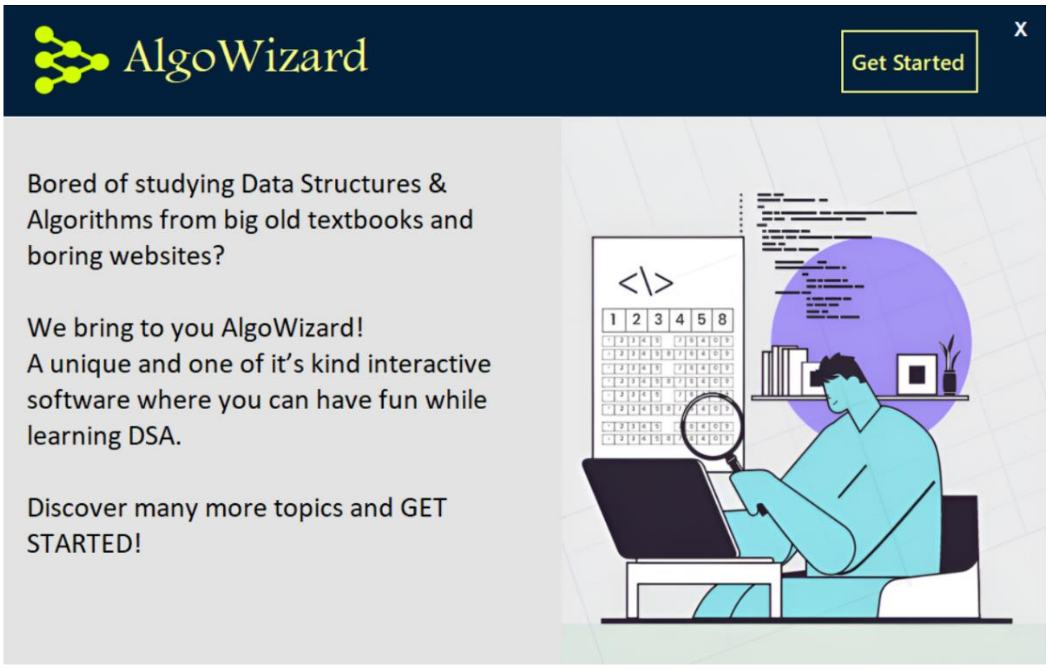
4 Testing & Error Handling

The program should handle input errors when the user tries to give invalid or empty inputs

- The user enters a non-valid input (including but not limited to alphabets, special characters, etc) instead of integers as the input of the textbox, a message indicating error should pop up.

- Empty or invalid strings in the sign up or log in pages are efficiently handled.
- Quiz results are displayed only after attempting all the given questions, and the corresponding database is updated.
- If the user attempts and submits the quizzes without logging in, then appropriate error messages are displayed.
- Out of bound indices are handled in the visualizations where needed.

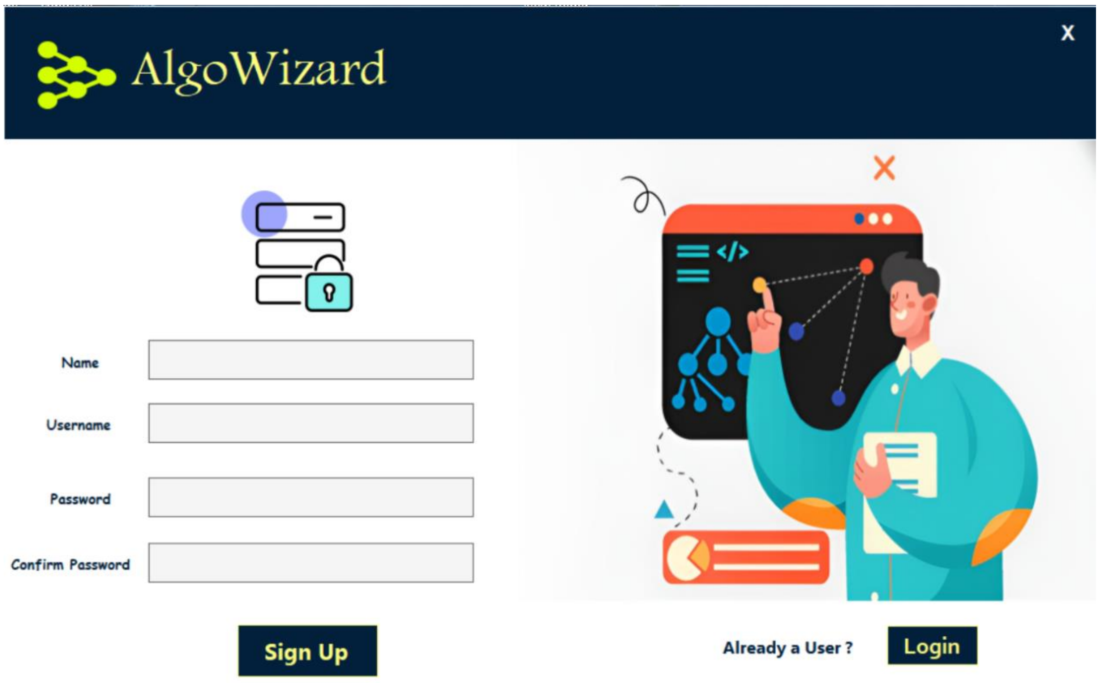
5 Interface

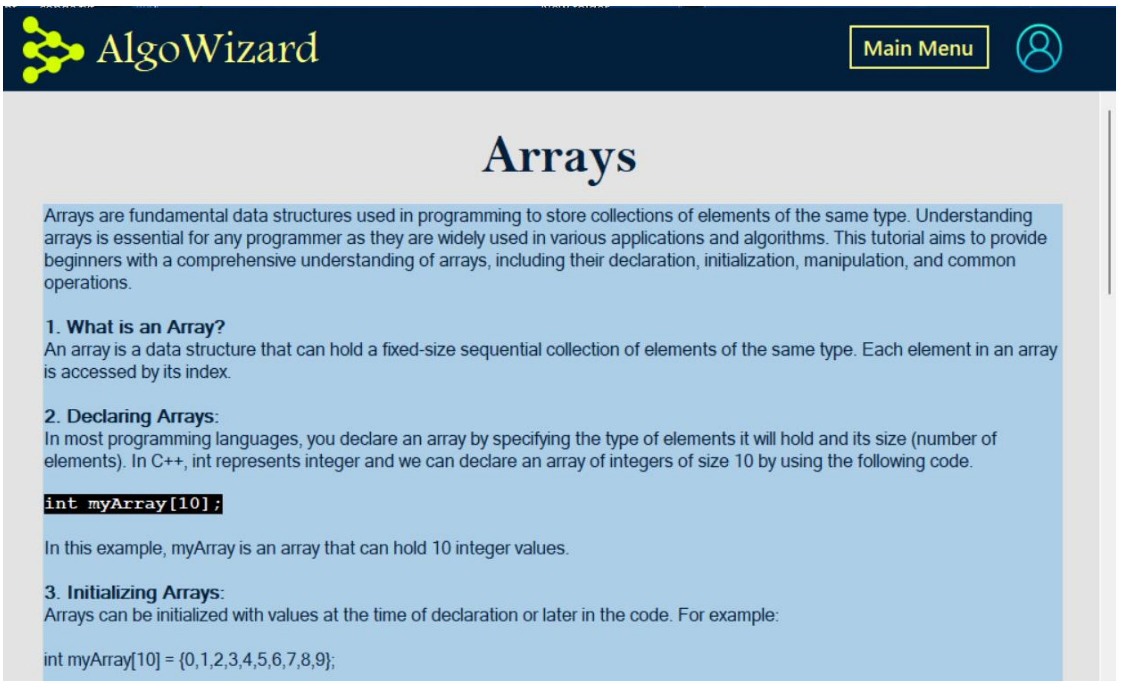


Bored of studying Data Structures & Algorithms from big old textbooks and boring websites?

We bring to you AlgoWizard!
A unique and one of it's kind interactive software where you can have fun while learning DSA.

Discover many more topics and GET STARTED!





Arrays

Arrays are fundamental data structures used in programming to store collections of elements of the same type. Understanding arrays is essential for any programmer as they are widely used in various applications and algorithms. This tutorial aims to provide beginners with a comprehensive understanding of arrays, including their declaration, initialization, manipulation, and common operations.

1. What is an Array?
An array is a data structure that can hold a fixed-size sequential collection of elements of the same type. Each element in an array is accessed by its index.

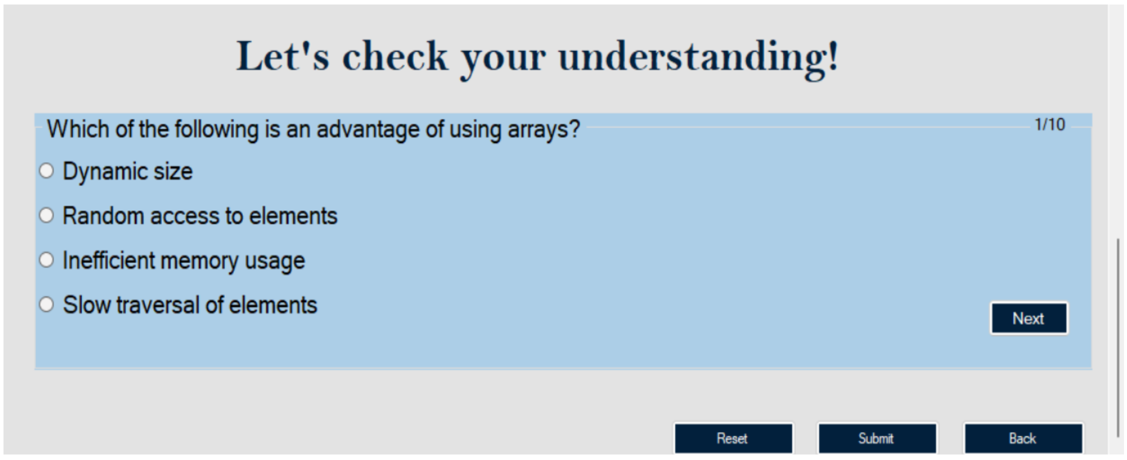
2. Declaring Arrays:
In most programming languages, you declare an array by specifying the type of elements it will hold and its size (number of elements). In C++, int represents integer and we can declare an array of integers of size 10 by using the following code.

```
int myArray[10];
```

In this example, myArray is an array that can hold 10 integer values.

3. Initializing Arrays:
Arrays can be initialized with values at the time of declaration or later in the code. For example:

int myArray[10] = {0,1,2,3,4,5,6,7,8,9};



Let's check your understanding!

Which of the following is an advantage of using arrays? 1/10

- ○ Dynamic size
- ○ Random access to elements
- ○ Inefficient memory usage
- ○ Slow traversal of elements