Uzair390-Del /
Data-Structure-and-Algorithms-Lab

Code Issues Pull requests Actions Projects Wiki Security In

Data-Structure-and-Algorithms-Lab / DS_Lab2 / lab2.md

Uzair390-Del upda 9dfefbb · now

519 lines (377 loc) · 13.4 KB

Preview Code Blame Raw

Task 1: Simple C++ Program that Uses Loops, Conditionals, and Functions

```cpp
#include <iostream>  // Include the input-output stream library
using namespace std;  // Use the standard namespace to avoid prefixing standar

// Function to calculate the sum and average of an array of integers
void calculateSumAndAverage(int arr[], int size) {
    int sum = 0; // Initialize sum to 0

    // Loop through each element in the array
    for (int i = 0; i < size; i++) {
        sum += arr[i];  // Add each number to the sum
    }

    // Calculate average by casting sum to double to avoid integer division
    double average = static_cast<double>(sum) / size;

    // Output the sum and average to the console
    cout << "Sum: " << sum << endl;
    cout << "Average: " << average << endl;
}

int main() {
    const int size = 5; // Define the size of the array
    int numbers[size]; // Declare an array to hold the integers

    cout << "Enter 5 integers: "; // Prompt the user for input
    // Loop to read integers from user input
    for (int i = 0; i < size; i++) {
        cin >> numbers[i];  // Input each number into the array
    }

    // Call the function to calculate sum and average, passing the array and i
    calculateSumAndAverage(numbers, size);

    return 0; // Indicate that the program ended successfully
}
```

Task 2: Find Largest and Smallest in an Array

```
#include <iostream>
using namespace std;

int main() {
    int n;

    // Ask the user for the number of elements
    cout << "Enter the number of elements: ";
    cin >> n;

    // Create an array to store the elements
    int arr[n];

    // Input elements
    cout << "Enter the elements: " << endl;
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Initialize smallest and largest with the first element of the array
    int largest = arr[0];
    int smallest = arr[0];

    // Loop through the array to find the largest and smallest values
    for(int i = 1; i < n; i++) {
        if(arr[i] > largest) {
            largest = arr[i];
        }
        if(arr[i] < smallest) {
            smallest = arr[i];
        }
    }
    // Output the largest and smallest values
    cout << "Largest value: " << largest << endl;
    cout << "Smallest value: " << smallest << endl;

    return 0;
}
```

Dry Run: Finding the Largest and Smallest in an Array

Given Code:

The program asks the user to input a number of elements (n) and then stores the values in an array. It proceeds to find the largest and smallest elements in the array.

Input:

Suppose the input array has 5 elements: `{5, 2, 9, 3, 7}` .

Step-by-Step Dry Run:

Initialization:

The user is asked to input the number of elements, `n = 5` . The array `arr[]` is declared to hold 5 integers.

Input Phase:

The user enters the 5 elements: 5, 2, 9, 3, 7. The array now contains: `arr[] = {5, 2, 9, 3, 7}` .

Initialize largest and smallest:

The variables largest and smallest are both initialized to the first element of the array (`arr[0] = 5`). State:

- largest = 5
- smallest = 5

Iteration 1 (i = 1):

The second element of the array, `arr[1] = 2` , is compared with largest and smallest.

- Since 2 is smaller than smallest, smallest is updated to 2.

- No change is made to largest because 2 is not greater than 5.

State:

- largest = 5
- smallest = 2

Iteration 2 (i = 2):

The third element of the array, `arr[2] = 9`, is compared.

- Since 9 is greater than largest, largest is updated to 9.
- No change is made to smallest because 9 is not smaller than 2.

State:

- largest = 9
- smallest = 2

Iteration 3 (i = 3):

The fourth element of the array, `arr[3] = 3`, is compared.

- No change is made to largest because 3 is not greater than 9.
- No change is made to smallest because 3 is not smaller than 2.

State:

- largest = 9
- smallest = 2

Iteration 4 (i = 4):

The fifth and final element of the array, `arr[4] = 7`, is compared.

- No change is made to largest because 7 is not greater than 9.
- No change is made to smallest because 7 is not smaller than 2.

State:

- largest = 9
- smallest = 2

Final Output:

The loop completes, and the program outputs:

- Largest value: 9
- Smallest value: 2

Summary:

- The largest number in the array `{5, 2, 9, 3, 7}` is 9.
- The smallest number in the array is 2.

Task 3: Print the Index of the Smallest and Largest Value

Code Explanation:

This program takes an array of integers from the user, finds the largest and smallest values, and prints their values and respective indices.

C++ Code:

```cpp
// Print the index of the smallest and largest value
#include <iostream>
using namespace std;

int main() {
    int n;

    // Ask the user for the number of elements
    cout << "Enter the number of elements: ";
    cin >> n;

    // Create an array to store the elements
    int arr[n];

    // Input elements
    cout << "Enter the elements: " << endl;
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Initialize smallest and largest with the first element of the array
    int largest = arr[0], smallest = arr[0];
    int largestIndex = 0, smallestIndex = 0;

    // Loop through the array to find the largest and smallest values with the
    for(int i = 1; i < n; i++) {
        if(arr[i] > largest) {
            largest = arr[i];
            largestIndex = i; // Update the index of the largest element
        }
        if(arr[i] < smallest) {
            smallest = arr[i];
            smallestIndex = i; // Update the index of the smallest element
        }
    }

    // Output the largest and smallest values along with their indices
    cout << "Largest value: " << largest << " at index " << largestIndex << en
    cout << "Smallest value: " << smallest << " at index " << smallestIndex <<

    return 0;
}
```

Dry Run: Finding the Largest and Smallest Values with their Indices

Problem:

The program will find the largest and smallest values in an array of integers and print their values along with their respective indices.

Input:

- Suppose the user inputs an array of 5 integers: **{12, 5, 23, 3, 15}**.

Dry Run:

Step 1: Initialization

- The program prompts the user to enter the number of elements, say `n = 5` .
- An array `arr[]` of size 5 is created to store the integers.

Step 2: Input Phase

- The user enters the elements: **12, 5, 23, 3, 15**.
- Now the array looks like this: `arr[] = {12, 5, 23, 3, 15}` .

Step 3: Initialize `largest` , `smallest` , and their indices

- Both `largest` and `smallest` are initialized to the first element of the array (`arr[0] = 12`).
- The indices `largestIndex` and `smallestIndex` are set to `0` .

State after initialization:

- `largest = 12`
- `smallest = 12`
- `largestIndex = 0`
- `smallestIndex = 0`

Step 4: Iteration 1 (i = 1)

- Compare `arr[1] = 5` with `largest` and `smallest` .
- Since `5` is smaller than `smallest` , update `smallest = 5` and `smallestIndex = 1` .
- No change to `largest` .

State after iteration 1:

- `largest = 12`
- `smallest = 5`
- `largestIndex = 0`
- `smallestIndex = 1`

Step 5: Iteration 2 (i = 2)

- Compare `arr[2] = 23` with `largest` .
- Since `23` is greater than `largest` , update `largest = 23` and `largestIndex = 2` .
- No change to `smallest` .

State after iteration 2:

- `largest = 23`
- `smallest = 5`
- `largestIndex = 2`

- `smallestIndex = 1`

Step 6: Iteration 3 (i = 3)

- Compare `arr[3] = 3` with `smallest`.
- Since `3` is smaller than `smallest`, update `smallest = 3` and `smallestIndex = 3`.
- No change to `largest`.

State after iteration 3:

- `largest = 23`
- `smallest = 3`
- `largestIndex = 2`
- `smallestIndex = 3`

Step 7: Iteration 4 (i = 4)

- Compare `arr[4] = 15` with `largest` and `smallest`.
- No changes to `largest` or `smallest`.

Final State:

- `largest = 23`
- `smallest = 3`
- `largestIndex = 2`
- `smallestIndex = 3`

Final Output:

- **Largest value: 23 at index 2**
- **Smallest value: 3 at index 3**

Task 4: Linear Search

Linear Search in C++

```
#include <iostream>
using namespace std;
```

```
int linearSearch(int arr[], int n, int target) {
    // Traverse the array
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i; // Return the index if the element is found
        }
    }
    return -1; // Return -1 if the element is not found
}

int main() {
    int n, target;

    // Input the number of elements in the array
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];

    // Input the array elements
    cout << "Enter the elements: " << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Input the target value to search for
    cout << "Enter the value to search: ";
    cin >> target;

    // Perform linear search
    int result = linearSearch(arr, n, target);

    // Output the result
    if (result != -1) {
        cout << "Element found at index " << result << endl;
    } else {
        cout << "Element not found in the array" << endl;
    }

    return 0;
}
```

Dry Run: Linear Search in C++

Problem:

We are performing a linear search to find a target element in an array and return its index if found. If the target element is not found, the program will return `-1`.

Input:

- Suppose the user inputs an array of 5 integers: **{10, 25, 30, 45, 50}**.
- The target element to search is: **30**.

Dry Run:

Step 1: Initialization

- The program prompts the user to enter the number of elements, say `n = 5`.
- An array `arr[]` of size 5 is created.

Step 2: Input Phase

- The user enters the elements: **10, 25, 30, 45, 50**.
- Now the array looks like this: `arr[] = {10, 25, 30, 45, 50}`.
- The target value to search for is `30`.

Step 3: Perform Linear Search

- The program starts iterating through the array to find the target value.

Iteration 1 (i = 0):

- **arr[0] = 10**, which is not equal to `30`.
- Move to the next iteration.

Iteration 2 (i = 1):

- **arr[1] = 25**, which is not equal to `30`.
- Move to the next iteration.

Iteration 3 (i = 2):

- **arr[2] = 30**, which is equal to the target value `30`.
- The program returns the index `2`.

Final Output:

- The program outputs: **"Element found at index 2"**.

Example 2: Target not found

- **Input Array:** `{5, 8, 12, 16, 20}`
- **Target:** `18`

Step 1: Initialization

- The user inputs `n = 5` and the array `arr[] = {5, 8, 12, 16, 20}` .
- The target is `18` .

Step 2: Perform Linear Search

- **Iteration 1 (i = 0):** `arr[0] = 5` (not equal to 18).
- **Iteration 2 (i = 1):** `arr[1] = 8` (not equal to 18).
- **Iteration 3 (i = 2):** `arr[2] = 12` (not equal to 18).
- **Iteration 4 (i = 3):** `arr[3] = 16` (not equal to 18).
- **Iteration 5 (i = 4):** `arr[4] = 20` (not equal to 18).

Since the element is not found in the array, the program returns `-1` .

Final Output:

- The program outputs: **"Element not found in the array"**.

Task 5: Reverse and array

```
#include <iostream>
using namespace std;

int main() {
    int n;

    // Input the number of elements in the array
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];

    // Input the array elements
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Initialize start and end pointers
    int start = 0;
    int end = n - 1;

    // Reverse the array using start and end variables
    while (start < end) {
        // Swap the elements at start and end
        swap(arr[start], arr[end]);
        start++;  // Move the start pointer forward
        end--;    // Move the end pointer backward
    }

    // Output the reversed array
    cout << "Reversed array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Dry Run: Array Reversal in C++

Problem:

We are reversing an array in place using two pointers, `start` and `end`. The program will swap the elements until the `start` pointer is no longer less than the `end` pointer.

Input:

- Suppose the user inputs an array of 5 integers: **{1, 2, 3, 4, 5}**.

Dry Run:

Step 1: Initialization

- The program prompts the user to enter the number of elements, say `n = 5`.
- An array `arr[]` of size 5 is created.

Step 2: Input Phase

- The user enters the elements: **1, 2, 3, 4, 5**.
- Now the array looks like this: `arr[] = {1, 2, 3, 4, 5}`.

Step 3: Initialize Pointers

- Initialize `start = 0` and `end = 4` (since `n - 1 = 4`).

Step 4: Reverse the Array Using While Loop

- The program starts the while loop, checking if `start < end`.

Iteration 1:

- Condition: `start < end` (0 < 4) is **true**.
- Swap `arr[start]` (1) with `arr[end]` (5).
- After swap: `arr[] = {5, 2, 3, 4, 1}`.
- Update pointers: `start = 1`, `end = 3`.

Iteration 2:

- Condition: `start < end` (1 < 3) is **true**.
- Swap `arr[start]` (2) with `arr[end]` (4).
- After swap: `arr[] = {5, 4, 3, 2, 1}`.
- Update pointers: `start = 2`, `end = 2`.

Iteration 3:

- Condition: `start < end` (2 < 2) is **false**.
- Exit the loop.

Final Output:

- The program outputs: **"Reversed array: 5 4 3 2 1"**.

Summary:

The array has been successfully reversed in place from `{1, 2, 3, 4, 5}` to `{5, 4, 3, 2, 1}` using two pointers (`start` and `end`).

Task 6: insertion, deletion

Task 7:Binary search