**Lab Session : DSA - Mark Zuckerberg**

**Date -** 21 January 2025

**Name -** Nivedita Tapan Biswas

**Roll No -** 150096724019

**Instruction:**

**Submission Protocol**

* Timeliness: Submit your assignments or code files within the given deadline i.e by 5.15 PM.
* Format: Ensure your submission follows the prescribed format, including file names and extensions.
* Please upload the same folder or file to Github as well.

**Format**  
**1. Folder Structure**

* **Folder Name:**
  + The folder should be named as follows:  
    [StudentName]\_[SubjectName]\_[Date]
  + Example:  
    If the student’s name is *JohnDoe*, the subject is *ProgrammingFundamentals*, and the date is *15-Jan-2025*, the folder name should be:  
    JohnDoe\_ProgrammingFundamentals\_15Jan2025

**2. File Naming Conventions**

* **Program Files:**
  + Each program file should have a meaningful name that reflects its purpose.
  + Example: If the task is to implement a sorting algorithm, name the file:  
    SortingAlgorithm.cpp

**Questions**

1. **Types of Data Structures**

* **What are the main types of data structures, and can you provide an example of each?**

**Ans:**

* A data structure is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.
* It enables programs to manage huge amounts of data easily and efficiently.
* Examples: array, stack, queue, linked list, tree, graph, etc,.

**The main types of data structures can be divided into two broad categories:**

1. Primitive Data Structure:

* The primitive data structures can hold a single value. The int, char, float, double are primitive data structures.
* These are basic data structures and directly operated upon by the machine instructions.
* In general, there are different representations for primitive data structures on different computers.

1. Non-Primitive Data Structure:

* There are more sophisticated data structures.
* These are derived from the primitive data structures.
* The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.
* Lists, Stack, Queue, Tree, Graph are examples of non-primitive data structures.
* The design of an efficient data structure must take operations to be performed on the data structure.

**Non-Primitive Data structures are divided into two sub categories they are follows:**

1. Linear Data Structures:

* In linear data structure, the data elements are organized in sequential manner.
* The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues.

**Array:** A collection of elements stored in contiguous memory locations.

Example: [1, 2, 3, 4, 5]

**Linked List**: A collection of nodes, where each node contains data and a reference to the next node.

Example: Head -> [10] -> [20] -> [30] -> None

**Stack**: A collection of elements that follows the **LIFO** (Last In, First Out) principle.

Example:

Push: 10 -> 20 -> 30

Pop: 30 (last pushed element is removed first.

**Queue**: A collection of elements that follows the **FIFO** (First In, First Out) principle.

Example:

Enqueue: 10 -> 20 -> 30

Dequeue: 10 (first enqueued element is removed first)

* In these data structures, one element is connected to another element in a linear form.

1. Non-Linear Data Structures:

* Data structure in which elements are organized in any arbitrary order.
* When one element is connected to the 'n' number of elements, it is known as a non-linear data structure.
* Example: trees and graphs.

**Tree:** A hierarchical structure where each node has a value and pointers to its children.

Example:

1

/ \

2 3

/ \

4 5

**Graph**: A set of vertices (nodes) connected by edges. It can be **directed** or **undirected**.

Example: (Undirected graphs)

A -- B

| |

C -- D

**2. Importance of Data Structures**

* **Why are data structures important in programming? Discuss how they affect algorithm efficiency.**

**Ans:**

* Efficiency: If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
* Reusability: The data structures provide reusability means that multiple client programs can use the data structure.
* Abstraction: The data structure also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

1. **Characteristics of Arrays**
   * **What are the key characteristics of an array in c++? How does it differ from other data structures like linked lists?**

**Ans:**

**The Key Characteristics of Arrays in C++ are as follows:**

1. Contiguous Memory Allocation:Arrays store elements in a continuous block of memory. This allows direct access to elements using their index.
2. Fixed Size:The size of an array is specified at the time of declaration and cannot be changed during runtime.
3. Random Access: Array elements can be accessed in constant time, O(1)O(1)O(1), using their index. For example, arr[3] directly accesses the 4th element.
4. Homogeneous Data:Arrays can store only elements of the same data type (e.g., int, float, char).
5. Static or Dynamic Allocation: Arrays can be declared with a fixed size at compile-time (static arrays) or allocated dynamically at runtime using new or malloc.
6. Sequential Indexing:Elements are stored sequentially, starting with an index of 0.
7. Overhead-Free Access:Arrays do not require extra memory for pointers or references, unlike linked lists.

### 

### 

### 

### 

### 

### 

### 

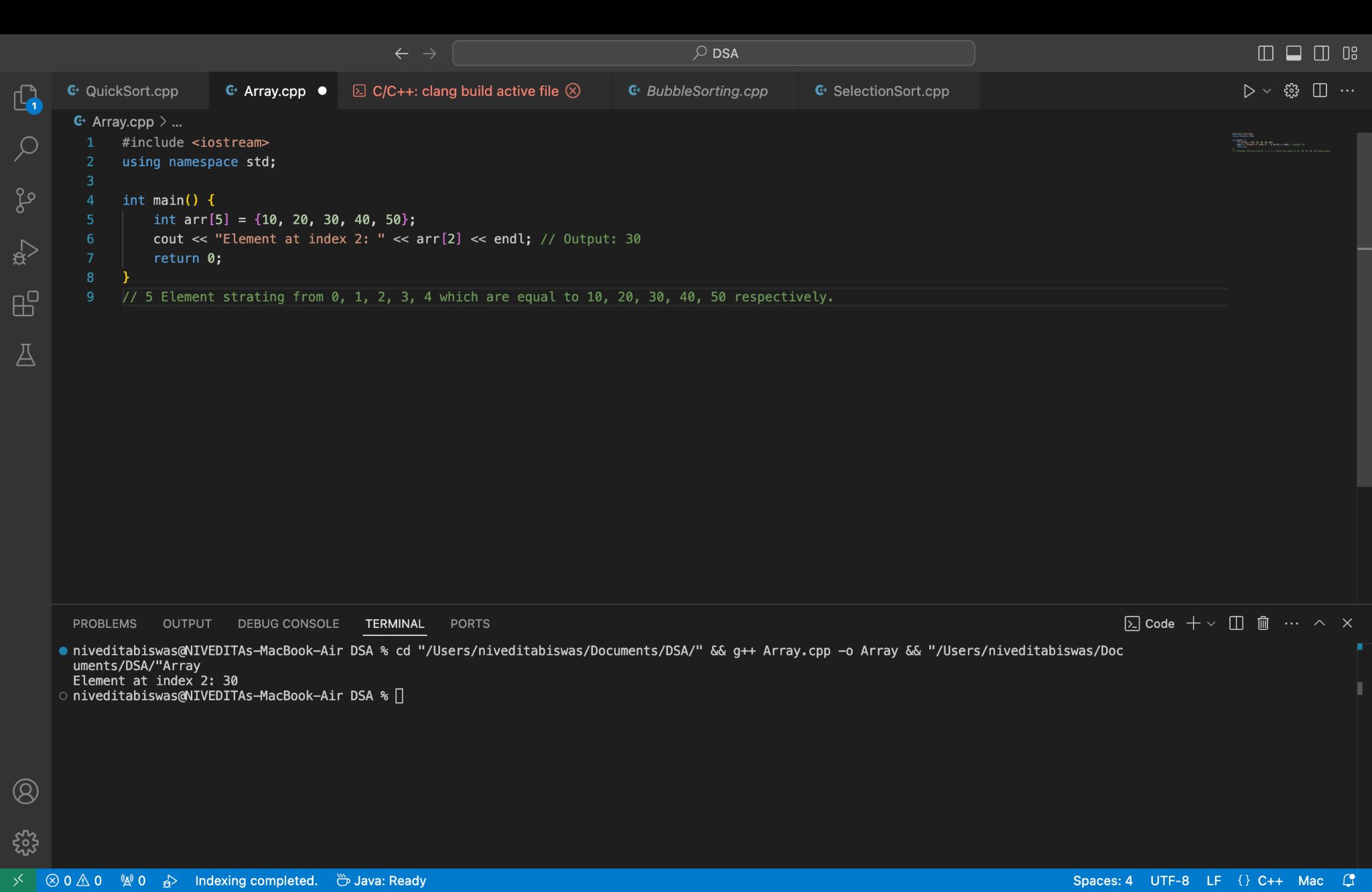
### 

### **Differences Between Arrays and Linked Lists**

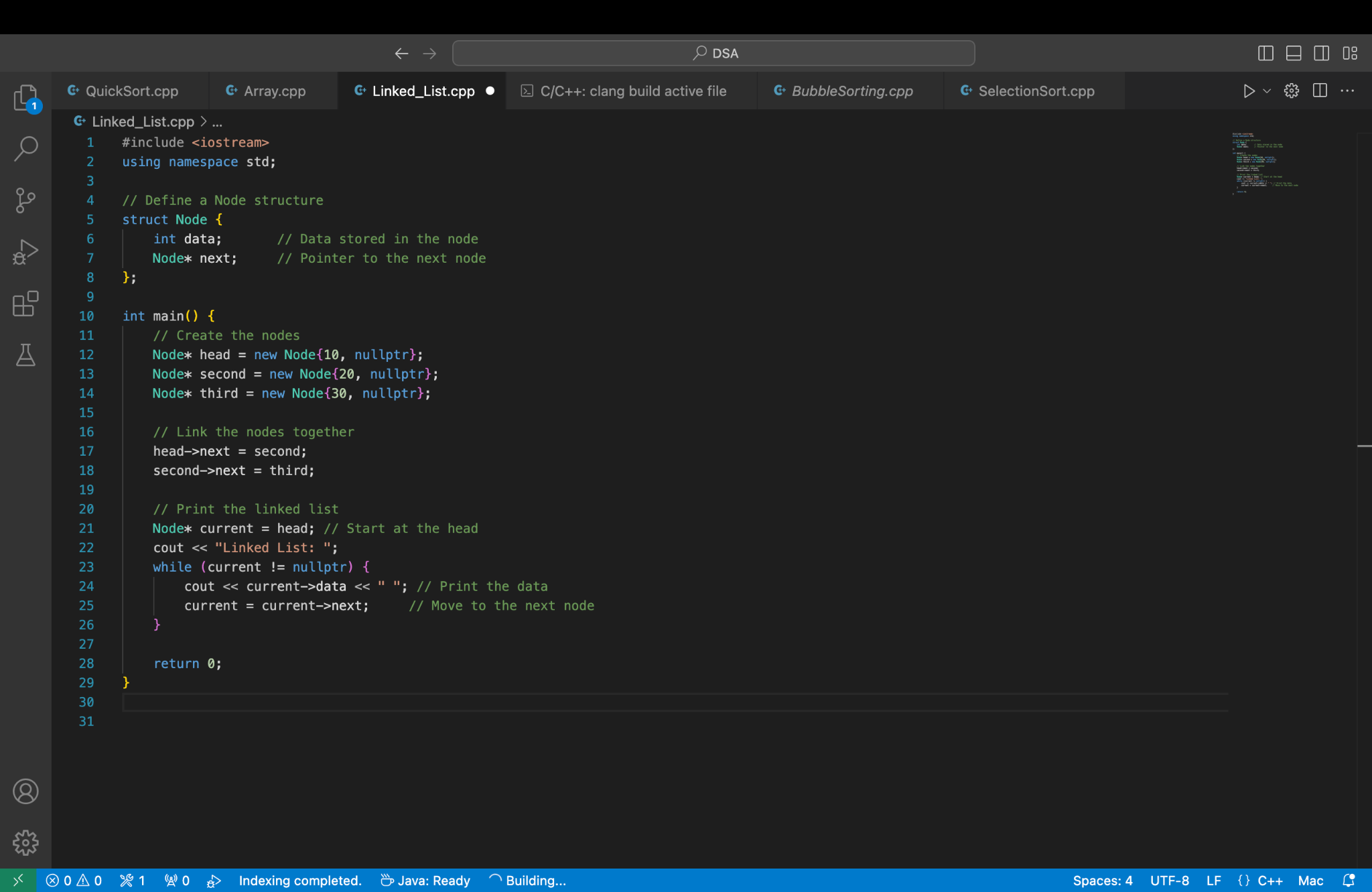
| **Feature** | **Array** | **Linked List** |
| --- | --- | --- |
| **Memory Allocation** | Contiguous memory allocation. | Non-contiguous memory allocation. |
| **Access Time** | O(1) for accessing elements via index. | O(n) for accessing elements (sequential traversal). |
| **Insertion/Deletion** | Costly for large arrays (O(n)O(n)O(n)). | Efficient, especially at the head or tail (O(1)O(1)O(1)). |
| **Dynamic Sizing** | Fixed size once declared. | Can grow or shrink dynamically during runtime. |
| **Memory Overhead** | No extra memory for pointers. | Requires extra memory for pointers in each node. |
| **Data Type** | Homogeneous data only. | Can store heterogeneous data using void pointers. |
| **Cache Friendliness** | High, due to contiguous memory. | Low, due to scattered memory locations. |
| **Use Cases** | Suitable for applications needing random access or fixed-size data**.** | Suitable for applications needing frequent insertion/deletion. |

### 

**Array:**

****

**Linked-list:**

****

**4. Fundamental Concepts of Algorithms**

* **Define an algorithm and explain its fundamental concepts. How do data structures play a role in algorithm design?**

**Ans:**

An algorithm is a finite set of instructions that, if followed,

accomplishes a particular task.

**In addition, all algorithms must satisfy the following criteria:**

1. Input: There are zero or more quantities that are externally supplied.

2. Output: At least one quantity is produced.

3. Definiteness: Each instruction is clear and unambiguous.

4. Finiteness: If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.

5. Effectiveness: Every instruction must be basic enough to be carried out, in

principle, by a person using only pencil and paper.

### **Role of Data Structures in Algorithm Design**

1. Efficient Data Organization:

Data structures determine how data is organized and accessed. An algorithm's performance often depends on choosing the appropriate data structure. **Example**: Using a heap in Dijkstra's algorithm for efficient priority queue operations.

1. Time and Space Optimization:

Data structures allow algorithms to minimize execution time and memory usage.**Example**: A hash table provides O(1)O(1)O(1) lookup time. A linked list uses dynamic memory but has O(n)O(n)O(n) lookup time.

1. Problem Modeling:

Data structures help model real-world problems effectively. **Example**: A graph represents networks like social media connections. A stack models undo operations in text editors.

1. Algorithm Design Patterns:

Algorithms often rely on specific data structures:

* **Divide and Conquer**: Arrays or recursion.
* **Greedy Algorithms**: Priority queues or heaps.
* **Dynamic Programming**: Tables or matrices.

1. Trade-Offs:

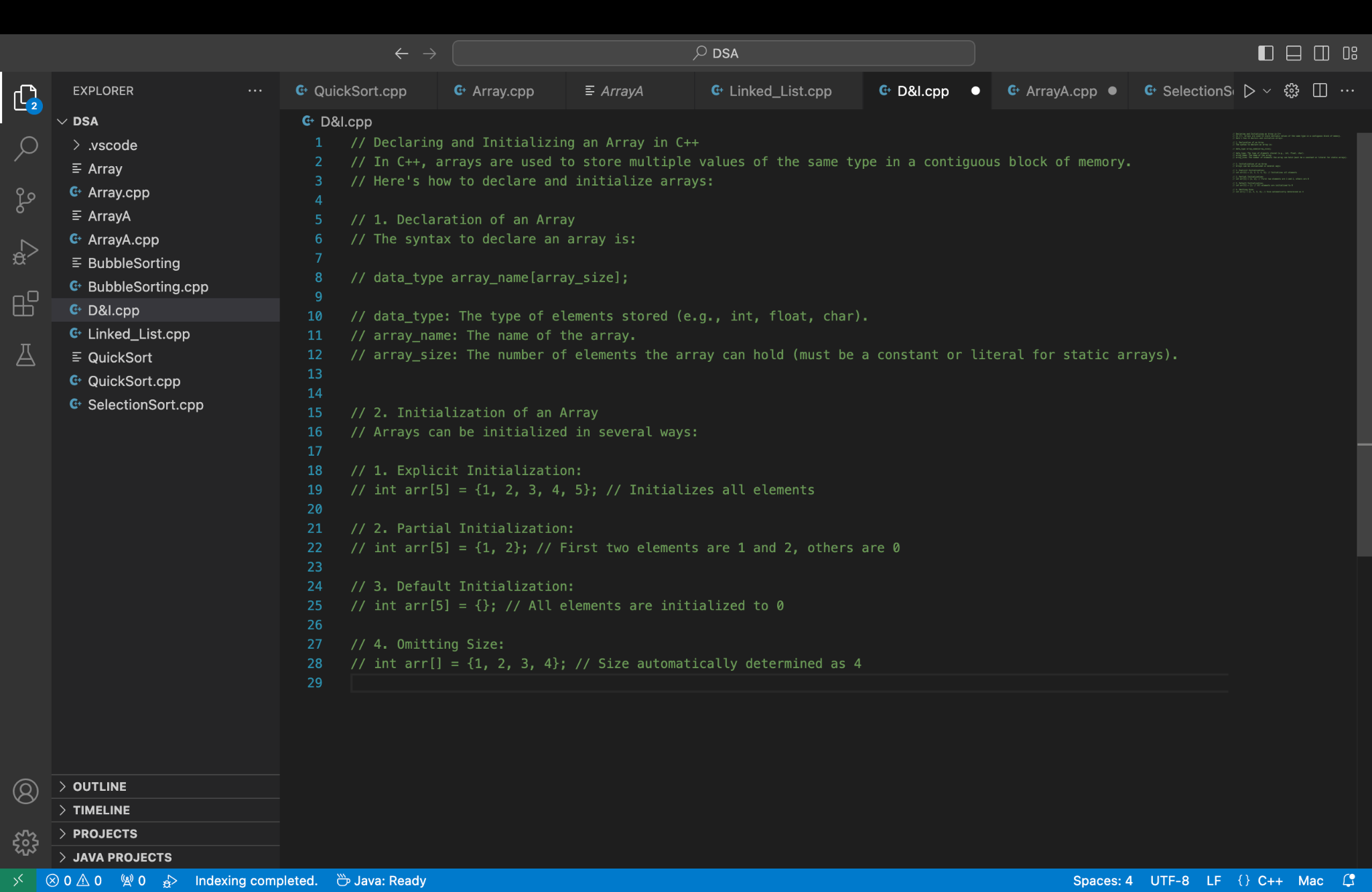
Data structures influence algorithmic trade-offs between time and space. **Example**: A binary search tree provides efficient search, insertion, and deletion (O(log⁡n)O(\log n)O(logn)), but requires balancing to maintain performance.

**5. Array Implementation**

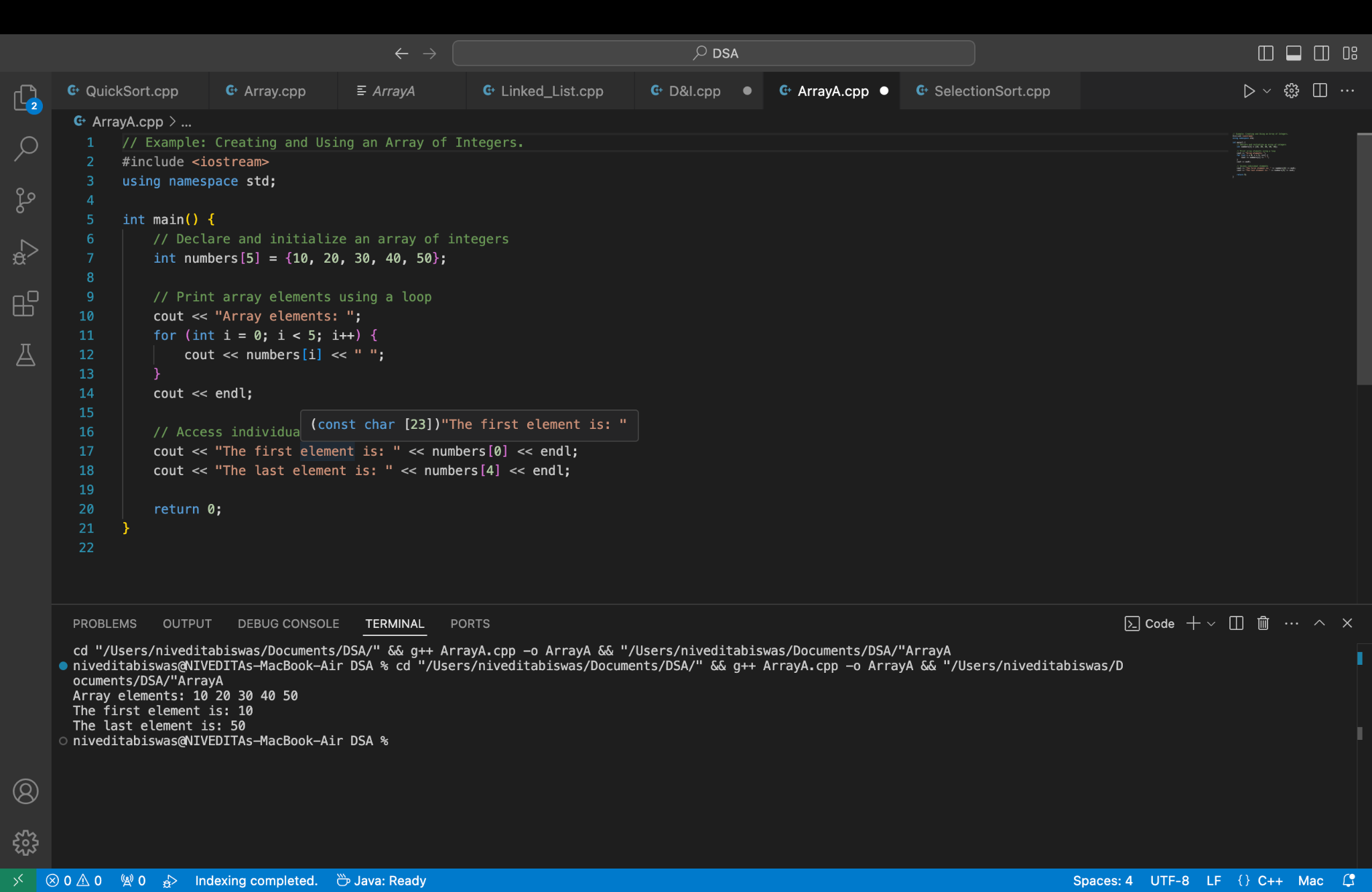
* **How do you declare and initialize an array in c++? Provide an example of creating an array of integers.**

**Ans:**

**Declare and Initialize an array in c++**

****

**Example:**

****