Hour 2 – Arrays Summary

Note: First read the chapter from the book then refer this thoroughly.

Introduction to Arrays: An array is like a list where you can store a bunch of similar things together. It's commonly used in programming to organize data. In this discussion, we'll talk about how arrays are used to keep track of baseball players attending a practice.

Basic Operations on Arrays: Imagine you're a coach, and you want to use your computer to keep track of which players are at practice. You need a program that can do three main things:

1. **Insert:** Add a new player when they arrive.
2. **Search:** Check if a specific player is present.
3. **Delete:** Remove a player when they leave.

These three actions are essential for many different ways we organize information in computer programs.

Array Workshop Applet: There's a computer program called an "applet" (like a small application) that visually shows how these operations work on an array. It's like a simulation to help us understand.

Visual Representation: Picture an array like a row of boxes where each box represents a player. The applet shows 20 boxes, but only 10 have players in them. Each player has a number and a colored shirt.

Operations in the Applet:

- **Insert (Ins):** You can add a new player by pressing a button. Imagine a kid arriving, and you assign them a number (like 678). The applet then shows that player in the first empty box.
- **Search (Find):** You can check if a specific player is present.
- **Delete (Del):** If a player leaves, you can remove them from the array.

Adding and Removing Players:

- When you press the "Insert" button, you tell the computer the player's number, and it adds them to the array.
- The "Search" button helps you find a player's number in the array.
- The "Delete" button removes a player when they go home.

Understanding the Applet:

- The applet gives you options to create a new array or fill it with players.
- There's a choice about allowing or not allowing duplicate player numbers.
- If you try to add a player with the same number as an existing player, the applet warns you but still lets you do it (assuming you won't make this mistake).

Summary: The applet is like a game where you can practice managing a list of baseball players using an array on your computer. It helps you understand how computers handle information and perform basic actions like adding, finding, and removing players.

To Do: Search for a Given Item:

1. Click the "Find" button. It will ask you to enter the number of the person you're looking for.
2. Choose a number from the middle of the array.
3. Type that number and press the "Find" button repeatedly. The applet will show a red arrow moving down the cells, checking each one at a time.
    - You'll see messages like "Checking next cell, index = 2."
4. Once you reach the specified item, a message will confirm the find, like "Have found item with key 505."
    - If duplicates are not allowed, the search stops as soon as it finds the item.
    - If the chosen number is not in the array, it will search through all cells before saying it can't find the item.

Understanding the Search:

- The average number of steps needed to find an item is $N/2$ (half of the items).
- The algorithm looks through an average of half the data items to find a specific item.
- If there are N items, it takes N/2 steps on average to find an item.
- In the worst-case scenario, if the item is at the end, it takes N steps.

Comparison with Insertion:

- Searching takes longer on average (N/2 steps) than insertion (one step).
- Time is proportional to the number of steps, so searching is generally more time-consuming.

Deletion:

- To delete an item, you first need to find it.
- Pressing the "Del" button moves the arrow down the array until the item is found, then deletes it.
- The assumption is that there are no holes in the array (empty spaces with filled cells above them). Holes would complicate and slow down the process.
- After deletion, the applet shifts the contents of each subsequent cell down one space to fill the hole.

Understanding Deletion Process:

- Assume no duplicates are allowed.
- Deletion involves searching an average of N/2 elements.
- After locating and deleting the item, it requires an average of N/2 moves to shift remaining elements and fill the gap.
- In total, deletion takes N steps on average.

These steps ensure that the array remains organized, and the process adapts depending on whether duplicates are allowed and if holes in the array are permitted.

Understanding the Duplicates Problem:

Decision on Duplicates:

- When designing a data storage system, you need to decide whether duplicate keys (like having the same identification number) are allowed.
- For certain scenarios (e.g., employee numbers), duplicates may not make sense. In contrast, for other scenarios (e.g., last names), duplicates might be expected.
- The Array Workshop applet lets you choose whether duplicates are allowed when creating a new array.

Implications of Allowing Duplicates:

- Allowing duplicates affects how searching, insertion, and deletion operations work.

Searching with Duplicates:

- If duplicates are allowed, the search algorithm must find all items matching the search key, requiring N steps.
- The applet, when duplicates are allowed, continues searching for possible additional matches even after finding the first match.

Insertion with Duplicates:

- Insertion remains the same whether duplicates are allowed or not; a single step inserts the new item.
- If duplicates are not allowed, there might be a need to check every existing item before insertion.

Deletion with Duplicates:

- Deletion can be more complicated with duplicates. If deletion means removing only the first item with a specified value, it requires $N/2$ comparisons and $N/2$ moves on average, similar to when duplicates are not allowed.
- If deletion means removing every item with a specified key value, it might require multiple deletions, checking N cells, and moving more than $N/2$ cells.

Array Workshop Applet:

- The applet assumes the second meaning of deletion, deleting multiple items with the same key.
- The deletion process involves shifting subsequent items each time an item is deleted.

Comparison of Operations with and Without Duplicates:

- Table 2.1 provides a summary of the average number of comparisons and moves for searching, insertion, and deletion, considering whether duplicates are allowed or not.

Efficiency Considerations:

- The difference between N and $N/2$ is generally not significant, but it matters when fine-tuning a program.
- More critical considerations for operations are whether they take one step, N steps, log(N) steps, or N^2 steps.

Observations on Array Algorithms:

- When using the Array applet, one can observe that the algorithms (excluding insertion) involve a slow and methodical process.
- While arrays are simple, they might not be the most efficient for certain operations, and more complex data structures can offer faster algorithms.

Array Example - Old-fashioned Procedural Version:

- The provided C++ program (array.cpp) demonstrates how an array is used.
- It initializes an array, displays items, searches for a specific item, deletes an item, and displays the remaining items.

Insertion, Searching, and Deletion in the Example:

- Insertion is straightforward and adds a new item to the array.
- Searching involves stepping through the array to find a match for a specific value.
- Deletion requires searching for the item, then shifting subsequent items down to fill the gap.

Limitation of Fixed-size Array:

- The example uses a fixed-size array of 100 items. This leads to memory wastage if fewer items are stored or a program failure if more than 100 items are inserted.

Next Step: Introduction to Vector for a More Flexible Approach.

This explanation provides a high-level understanding of how duplicates impact array operations and how a simple array example works in a procedural programming style.

Displaying the Array Contents:

Overview:

- Displaying all elements in the array is done by stepping through the array and accessing each element with `arr.getElem(j)` in the `main()` function.

Program Organization:

- The original `array.cpp` program lacked organization and classes.
- In the first step towards an object-oriented approach, the program is divided into two parts:
    - Data storage structure (array) becomes a separate class called `LowArray`.
    - The remaining part of the program (main functionality) remains in the `main()` function.
- This division enhances program clarity, design, understanding, and future modifications.

Introduction of LowArray Class:

- A new class named `LowArray` is introduced.
- `LowArray` encapsulates an array (now represented using a `vector<double>`) and provides member functions for communication with `main()`.

LowArray Class Implementation:

- The `LowArray` class contains:
  - A private `vector<double> v` to hold doubles (array-like).
  - A constructor `LowArray(int max)` that sizes the vector to a specified maximum.
  - `setElem(int index, double value)` function to put an element into the array at a given index.
  - `getElem(int index)` function to get an element from the array at a given index.

Main Function using LowArray Class:

- The `main()` function creates an instance of `LowArray` named `arr`.
- It inserts 10 items into the array, displays the items, searches for an item, deletes an item, and displays the remaining items.

Program Output:

- The output of the program is similar to the previous version (`array.cpp`), including attempts to find non-existent key values and deleting an existing item.

Limitations of Current Design:

- The current design of `LowArray` is not entirely successful, highlighting the need for a better approach.

Next Steps:

- In the next step, the communication between the storage structure (LowArray) and its user (main()) will be improved, addressing the limitations of the current design.

This explanation provides an overview of how the program has been organized into a class-based structure (LowArray) and how the main functionality (main()) interacts with this data storage structure.

LowArray Class and main():

Overview:

- In `lowArray.cpp`, the `LowArray` class is introduced to encapsulate an ordinary C++ array.
- The array is private, hidden within the class, and accessed only through class member functions (`setElem()`, `getElem()`, and a constructor).
- A vector is used instead of an array, allowing dynamic sizing through `resize()` in the class constructor.
- The main function acts as a user of the `LowArray` class, creating an object (`arr`) and manipulating data.

Class Interfaces:

- **Class Interface Definition:** It refers to how a class user interacts with the class, typically involving member functions. In this context, the class user is the `main()` function.
- **Advantage of Object-Oriented Programming (OOP):** OOP allows designing class interfaces to be convenient and efficient, enhancing usability.

Improving Interface:

- The current `LowArray` interface is not convenient for the `main()` function.
- Issues:
    - Low-level operations like `setElem()` and `getElem()` are reminiscent of direct array access.
    - No convenient way to display the array contents.

Responsibility Division:

- The `main()` function in `lowArray.cpp` must keep track of array indices, requiring a more direct, hands-on approach.
- This may make sense for users needing random access but isn't helpful for typical program users.

HighArray Example:

- An improved interface is demonstrated in the `HighArray` class in the next sample program.
- The `HighArray` class interface frees the user from handling index numbers, providing `insert()`, `find()`, and `delete()` member functions.
- Users can concentrate on the what (insert, delete, access) rather than the how.

Next Steps:

- The `highArray.cpp` program will be explored to understand the improved interface and how it simplifies the `main()` function's role.

This explanation covers the transition from `LowArray` to the concept of class interfaces and the motivation for improving the interface for better usability.

Listing 2.3 - highArray.cpp Program:

Overview:

- In `highArray.cpp`, the `HighArray` class offers a high-level interface, addressing the usability issues of the previous version.
- The class is wrapped around a vector (`v`), and the class user (`main()`) interacts with the class through `insert()`, `find()`, `remove()`, and `display()` member functions.
- The `main()` function creates an object of the `HighArray` class, performs various operations, and displays the array contents.

Improved Interface:

- **User's Life Made Easier:** The `main()` function is simplified. It no longer needs to manage index numbers or worry about the array's internal details.
- **Member Functions:**
    1. **find():** Searches for a specified value and returns true or false.
    2. **insert():** Inserts a new element into the array.
    3. **remove():** Removes an element from the array.
    4. **display():** Displays all values stored in the array.

Abstraction:

- **Abstraction Defined:** Abstraction is the process of separating the how (implementation details) from the what (visible functionality) inside a class.
- **Benefits of Abstraction:** It makes designing a program easier as designers can focus on high-level functionality without dealing with low-level implementation details.

Comparison:

- **Code Reduction:** The search operation code in `main()` is reduced from eight lines in `lowArray.cpp` to one line in `highArray.cpp`.
- **Ease of Use:** The `main()` function is relieved from managing index numbers, and the class user does not need to know the specific data structure used by `HighArray`.

Summary:

- Unordered arrays provide fast insertion but slower searching and deletion.
- Wrapping arrays in a class protects them and introduces an interface for interaction.
- A class interface comprises member functions accessible to the class user.
- A well-designed interface simplifies tasks for the class user without revealing internal details.
- Abstraction involves separating how an operation is performed from what is visible to the class user.

Workshop:

- **Quiz Answers:**
    1. N/2.
    2. N/2.
    3. N/2.
    4. A class interface is the set of member functions (and occasionally data members) accessible to the class user.
    5. Simplifying things for the class user is important for usability and ease of implementation.
    6. Wrapping an array in a class provides encapsulation, protecting the array from unintended alterations.
    7. Sorting an array is an example of an operation easier to perform on an array in a class.
    8. Abstraction is the process of separating how an operation is performed inside a class from what is visible to the class user.

Exercise:

- **Improving Interface:**
  - Introduce new member functions for the `HighArray` class, such as:
    - A function displaying all data members with keys greater than a certain value.
    - The specific number and type of functions depend on the general purpose and requirements of the array class.
  - Evaluate whether adding too many such functions would lead to a cumbersome or overloaded interface, considering the principle of simplicity and usability.