**Exercise 4: Employee Management System**

Explain how arrays are represented in memory and their advantages.

Memory Representation of Arrays

Arrays are stored in contiguous memory locations, meaning that each element is stored next to the other in memory. The memory layout of an array can be visualized as follows:

index 0 | index 1 | index 2 | ... | index n

Each element in the array is stored in a separate memory location, and the memory address of each element is calculated using the base address of the array and the index of the element.

Advantages of Arrays

1. Fast Access: Arrays provide fast access to elements using their index, making them suitable for applications where fast access is critical.

2. Cache Efficiency: Arrays can be cache-efficient due to their contiguous memory allocation, which can improve performance in certain scenarios.

3. Simple Implementation: Arrays are simple to implement and understand, making them a good choice for beginners.

4. Fixed Size: Arrays have a fixed size, which can make memory management easier and more predictable.

5. Multi-Dimensional: Arrays can be multi-dimensional, allowing for complex data structures to be represented.

6. Low Overhead: Arrays have low overhead in terms of memory usage, making them suitable for large datasets.

However, arrays also have some limitations, such as:

- Fixed size, which can lead to wasted space or overflow errors

- Slow search and delete operations for large datasets

- No dynamic resizing, which can lead to performance issues

Overall, arrays are a fundamental data structure in programming and are suitable for many applications, especially those requiring fast access and cache efficiency.

//Java

public class Employee {

private int employeeId;

private String name;

private String position;

private double salary;

public Employee(int employeeId, String name, String position, double salary) {

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

// Getters and setters

}

public class EmployeeManagementSystem {

private Employee[] employees;

private int size;

public EmployeeManagementSystem(int capacity) {

employees = new Employee[capacity];

size = 0;

}

public void addEmployee(Employee employee) {

if (size < employees.length) {

employees[size] = employee;

size++;

}

}

public Employee searchEmployee(int employeeId) {

for (int i = 0; i < size; i++) {

if (employees[i].getEmployeeId() == employeeId) {

return employees[i];

}

}

return null;

}

public void traverseEmployees() {

for (int i = 0; i < size; i++) {

System.out.println(employees[i]);

}

}

public void deleteEmployee(int employeeId) {

for (int i = 0; i < size; i++) {

if (employees[i].getEmployeeId() == employeeId) {

// Shift elements to fill the gap

for (int j = i; j < size - 1; j++) {

employees[j] = employees[j + 1];

}

size--;

break;

}

}

}

}

Analysis

Analyze the time complexity of each operation (add, search, traverse, delete).

- Add Employee: O(1) - constant time complexity

- Search Employee: O(n) - linear time complexity

- Traverse Employees: O(n) - linear time complexity

- Delete Employee: O(n) - linear time complexity

Discuss the limitations of arrays and when to use them.

Limitations of Arrays

- Fixed size: Arrays have a fixed capacity, which can lead to wasted space or overflow errors.

- Slow search and delete: Searching and deleting elements in an array can be slow for large datasets.

- No dynamic resizing: Arrays cannot be resized dynamically, which can lead to performance issues.

When to use Arrays

- Small datasets: Arrays are suitable for small datasets where the size is known in advance.

- Fast access: Arrays provide fast access to elements using their index.

- Cache efficiency: Arrays can be cache-efficient due to their contiguous memory allocation.