Nguyen Minh Chien BH01527
DSA SE06304

A stack ADT, a concrete data structure for a First In First out (FIFO) queue.

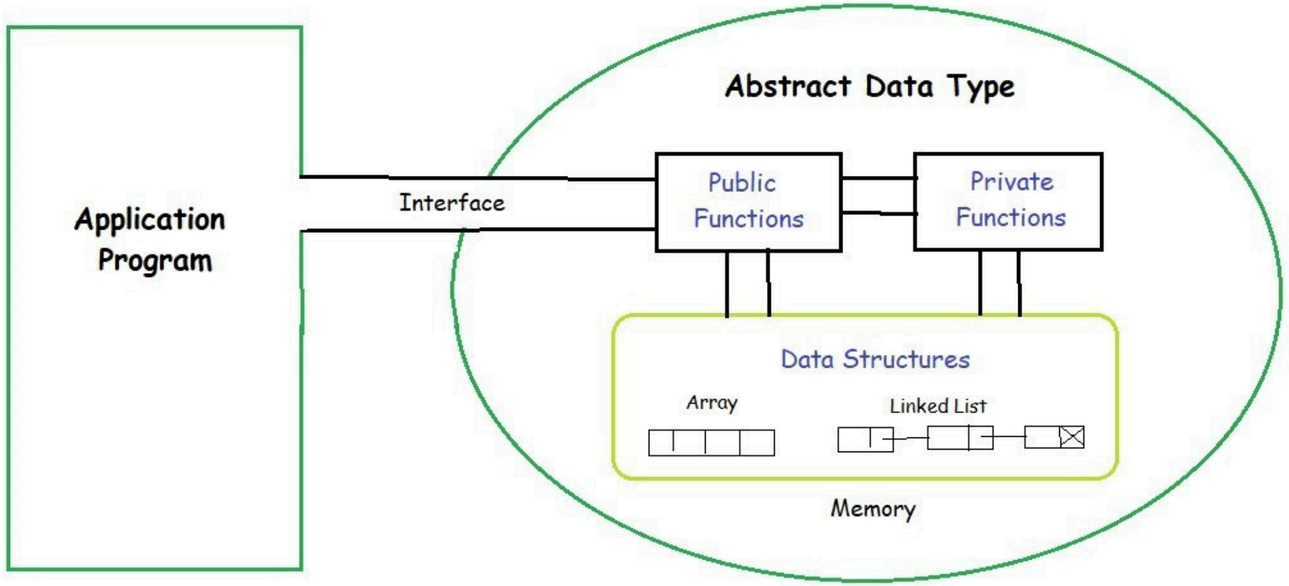
1. What is DSA?

Data Structures and Algorithms (DSA) is a fundamental part of Computer Science that teaches you how to think and solve complex problems systematically.
Using the right data structure and algorithm makes your program run faster, especially when working with lots of data.
Knowing DSA can help you perform better in job interviews and land great jobs in tech companies.
The word 'algorithm' comes from 'al-Khwarizmi', named after a Persian scholar who lived around year 800.
The concept of algorithmic problem-solving can be traced back to ancient times, long before the invention of computers.
The study of Data Structures and Algorithms really took off with the invention of computers in the 1940s, to efficiently manage and process data.
Today, DSA is a key part of Computer Science education and professional programming, helping us to create faster and more powerful software.

2. What is ADT?

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view.

The process of providing only the essentials and hiding the details is known as abstraction.



The user of data type does not need to know how that data type is implemented, for example, we have been using Primitive values like int, float, char data types only with the knowledge that these data type can operate and be performed on without any idea of how they are implemented.

3. Compare different between Stack and Queue

Stacks

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed. It can be visualized as a pile of plates where you can only add or remove the top plate.

Operations on Stack:

The primary operations associated with a stack are:

- **Push:** Adds an element to the top of the stack.
- **Pop:** Removes and returns the top element of the stack.
- **Peek (or Top):** Returns the top element of the stack without removing it.
- **IsEmpty:** Checks if the stack is empty.
- **Size:** Returns the number of elements in the stack.

Use Cases of Stack:

Stacks are used in various applications, including:

- **Function Call Management:** The call stack in programming languages keeps track of function calls and returns.
- **Expression Evaluation:** Used in parsing expressions and evaluating postfix or prefix notations.
- **Backtracking:** Helps in algorithms that require exploring all possibilities, such as maze solving and depth-first search.

Queues

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. This means that the first element added to the queue is the first one to be removed. It can be visualized as a line of people waiting for a service, where the first person in line is the first to be served.

Operations on Queue:

The primary operations associated with a queue are:

- **Enqueue:** Adds an element to the end (rear) of the queue.
- **Dequeue:** Removes and returns the front element of the queue.
- **Front (or Peek):** Returns the front element of the queue without removing it.
- **IsEmpty:** Checks if the queue is empty.
- **Size:** Returns the number of elements in the queue.

Use Cases of Queue:

Queues are used in various applications, including:

- **Task Scheduling:** Operating systems use queues to manage tasks and processes.
- **Breadth-First Search (BFS):** In graph traversal algorithms, queues help in exploring nodes level by level.
- **Buffering:** Used in situations where data is transferred asynchronously, such as IO buffers and print spooling.

Feature	Stack	Queue
Definition	A linear data structure that follows the Last In First Out (LIFO) principle.	A linear data structure that follows the First In First Out (FIFO) principle.
Primary Operations	Push (add an item), Pop (remove an item), Peek (view the top item)	Enqueue (add an item), Dequeue (remove an item), Front (view the first item), Rear (view the last item)
Insertion/Removal	Elements are added and removed from the same end (the top).	Elements are added at the rear and removed from the front.
Examples	Browser history (back button), reversing a word.	Customer service lines, CPU task scheduling.
Real-World Analogy	A stack of plates: you add and remove plates from the top.	A queue at a ticket counter: the first person in line is the first to be served.
Complexity (Amortized)	Push: O(1), Pop: O(1), Peek: O(1)	Enqueue: O(1), Dequeue: O(1), Front: O(1), Rear: O(1)
Memory Structure	Typically uses a contiguous block of memory or linked list.	Typically uses a circular buffer or linked list.

4. Different Ways to implement Stack and Queue

A queue can be implemented using an array, a linked list, or a dynamic array (such as a vector in C++ or ArrayList in Java). In programming, queues are used to implement algorithms like breadth-first search, round-robin scheduling, and more.

A stack can be implemented using an array, a linked list, or a dynamic array (such as a vector in C++ or ArrayList in Java). In programming, stacks are used to implement recursive algorithms, undo/redo functionality, expression evaluation, and more.