Unique Karanjit

Department of Computer Science, University of the Cumberlands

MSCS-532-A01: Algorithms and Data Structure

Dr. Satish Penmatsa

Feb 9, 2025

Table of Contents

Implementation of Data Structures **3**

Performance Analysis **4**

Practical Applications **5**

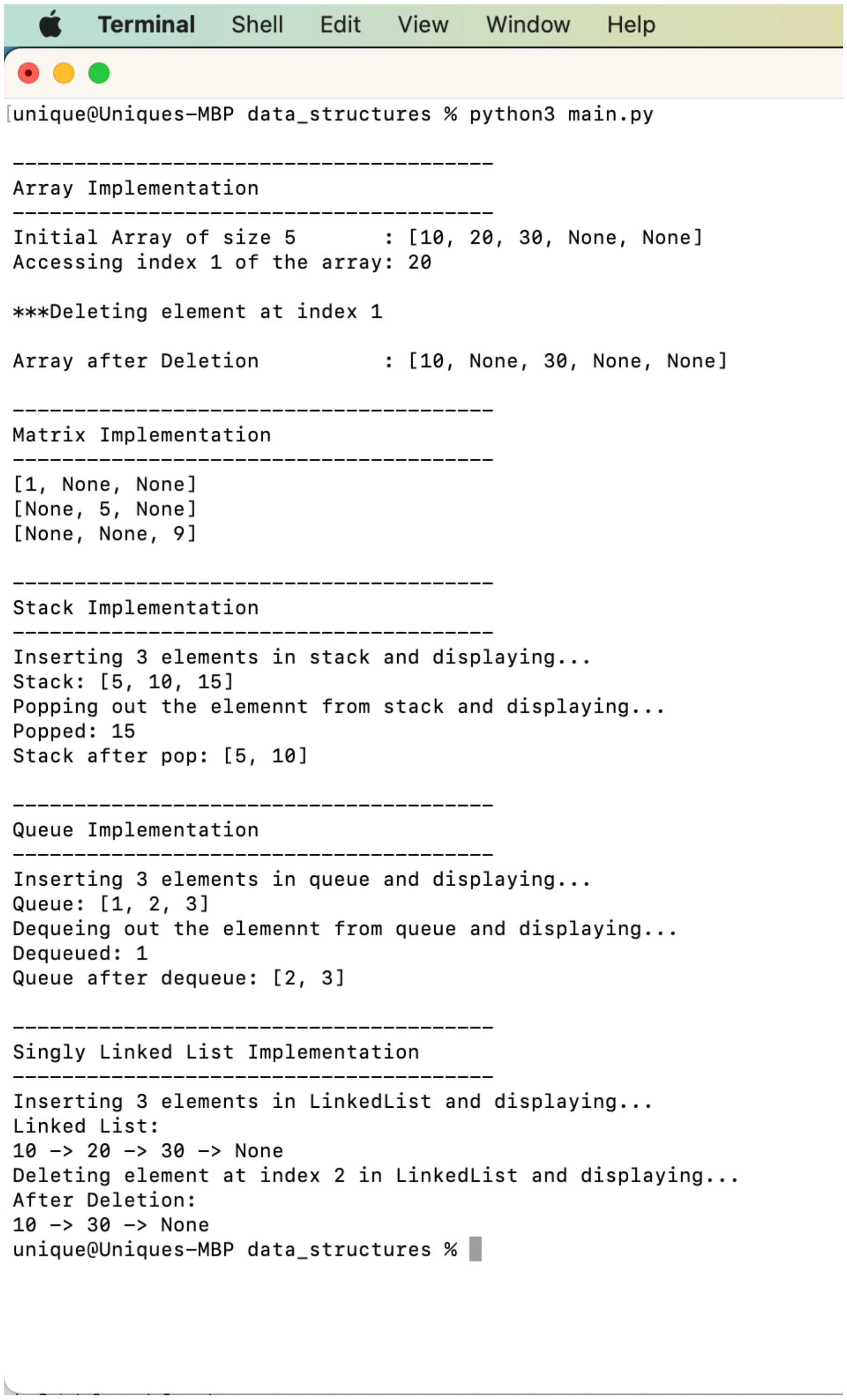
- Arrays: 5
- Stacks: 5
- Queues: 5
- Linked Lists: 5

Why Choose One Over Another? 6

References **6**

Implementation of Data Structures

Arrays, Stacks, Queues, Matrices, and Linked List are implemented in this project and the output of the running code is shown below.



```
unique@Uniques-MBP data_structures % python3 main.py

---------------------------------------
Array Implementation
---------------------------------------
Initial Array of size 5       : [10, 20, 30, None, None]
Accessing index 1 of the array: 20

***Deleting element at index 1

Array after Deletion          : [10, None, 30, None, None]

---------------------------------------
Matrix Implementation
---------------------------------------
[1, None, None]
[None, 5, None]
[None, None, 9]

---------------------------------------
Stack Implementation
---------------------------------------
Inserting 3 elements in stack and displaying...
Stack: [5, 10, 15]
Popping out the elemennt from stack and displaying...
Popped: 15
Stack after pop: [5, 10]

---------------------------------------
Queue Implementation
---------------------------------------
Inserting 3 elements in queue and displaying...
Queue: [1, 2, 3]
Dequeing out the elemennt from queue and displaying...
Dequeued: 1
Queue after dequeue: [2, 3]

---------------------------------------
Singly Linked List Implementation
---------------------------------------
Inserting 3 elements in LinkedList and displaying...
Linked List:
10 -> 20 -> 30 -> None
Deleting element at index 2 in LinkedList and displaying...
After Deletion:
10 -> 30 -> None
unique@Uniques-MBP data_structures %
```

Performance Analysis

The time complexity for operation on all data structures can be shown in the table as shown below.

Operation	Array	Stack	Queue	Singly Linked List
Insert – End	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Insert – Middle	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Delete – End	$O(1)$	$O(1)$	$O(n)$	$O(n)$
Delete – Middle	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Access	$O(1)$	$O(1)$	$O(1)$	$O(n)$

Trade Off-s Between Data Structures

- **Memory Usage**: Arrays use contiguous memory, while linked lists use extra memory for pointers.
- **Speed:** Arrays provide O(1) access but O(n) insert/delete (except at the end). Linked lists offer efficient insertions/deletions but slow lookups.
- **Use Cases:**
  - **Array-based Stacks/Queues:** Good when memory is predictable.
  - **Linked List-based Stacks/Queues:** Preferred when dynamic resizing is needed.

Practical Applications

• Arrays:

- Used in databases for storing records.
- Used in graphics processing where fast access is required.
- Used in scheduling problems and caching.

• Stacks:

- The function call stack in programming languages.
- Undo/redo functionality in text editors.
- Expression evaluation (e.g., parsing mathematical expressions).

• Queues:

- CPU scheduling and process scheduling.
- Printer queue management.
- Breadth-first search (BFS) in graphs.

• Linked Lists:

- Used in dynamic memory allocation.

- Implementation of hash tables (chaining for collision resolution).
- Navigation systems where elements need to be inserted/deleted dynamically.

Why Choose One Over Another?

- **When fast access is needed:** Array
- **When dynamic resizing is required:** a linked list
- **When LIFO structure is needed:** stack
- **When FIFO structure is needed:** queue

References

Weiss, M. A. (2006). *Data Structures and Algorithm Analysis in C++*. Pearson.