



**K.R. MANGALAM UNIVERSITY**

# **DATA STRUCTURES**

## **LAB FILE**

Submitted by : Manan Gola

Submitted to : SWATI

Course : B.TECH CSE DATA SCIENCE

Semester : III

Roll No : 2401420043

# TASK-1

➤ Store Inventory: To manage items in inventory

{Add item, remove item, display inventory, search item}

- Code:

```
1 stock_items = {}
2 def insert_product():
3     prod_code = input("Enter product SKU: ").strip()
4     if prod_code in stock_items:
5         print("This SKU already exists in stock.")
6         return
7     prod_title = input("Enter product name: ").strip()
8     qty_str = input("Enter quantity: ")
9     if not qty_str.isdigit():
10        print("Quantity must be a positive number.")
11        return
12     qty_val = int(qty_str)
13     if qty_val <= 0:
14        print("Quantity must be greater than zero.")
15        return
16     stock_items[prod_code] = {"title": prod_title, "qty": qty_val}
17     print(f"Product {prod_title} with SKU {prod_code} added.")
18 def remove_product():
19     prod_code = input("Enter SKU to remove: ").strip()
20     if prod_code in stock_items:
21         deleted = stock_items.pop(prod_code)
25 def show_inventory():
26     if not stock_items:
27         print("Inventory is empty.")
28         return
29     print("\nCurrent Inventory:")
30     print("{:<10} {:<20} {:<10}".format("SKU", "Product", "Quantity"))
31     for code, info in stock_items.items():
32         print("{:<10} {:<20} {:<10}".format(code, info["title"],
33                                         info["qty"]))
33     print()
34 def find_product():
35     prod_code = input("Enter SKU to search: ").strip()
36     if prod_code in stock_items:
37         info = stock_items[prod_code]
38         print(f"Found: {prod_code} - {info['title']}, Qty: {info['qty']}")
39     else:
40         print("Item not present in inventory.")
41 def inventory_menu():
42     while True:
43         print("\n---- Inventory Stock Manager ----")
44         print("1. Add new product")
45         print("2. Remove a product")
46         print("3. Display inventory")
47         print("4. Search product")
```

```
48         print("5. Exit")
49         option = input("Enter your choice (1-5): ")
50     if option == "1":
51         insert_product()
52     elif option == "2":
53         remove_product()
54     elif option == "3":
55         show_inventory()
56     elif option == "4":
57         find_product()
58     elif option == "5":
59         print("Closing inventory manager.")
60         break
61     else:
62         print("Invalid choice, please try again.")
```

## \*Output:

```
---- Inventory Stock Manager ----
1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit
Enter your choice (1-5): 1
Enter product SKU: P101
Enter product name: Pencil
Enter quantity: 50
Product Pencil with SKU P101 added.

---- Inventory Stock Manager ----
1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit
Enter your choice (1-5): 1
Enter product SKU: P102
Enter product name: Pen
Enter quantity: 100
Product Pen with SKU P102 added.

---- Inventory Stock Manager ----
1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit
Enter your choice (1-5): 3

Current Inventory:
SKU      Product          Quantity
P101    Pencil            50
P102    Pen               100

---- Inventory Stock Manager ----
1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit
Enter your choice (1-5): 2
Enter SKU to remove: P102
Removed Pen from inventory.
```

---- Inventory Stock Manager ----

1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit

Enter your choice (1-5): 3

Current Inventory:

SKU	Product	Quantity
P101	Pencil	50
P102	Pen	100

---- Inventory Stock Manager ----

1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit

Enter your choice (1-5): 2

Enter SKU to remove: P102

Removed Pen from inventory.

---- Inventory Stock Manager ----

1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit

Enter your choice (1-5): 4

Enter SKU to search: P101

Found: P101 - Pencil, Qty: 50

---- Inventory Stock Manager ----

1. Add new product
2. Remove a product
3. Display inventory
4. Search product
5. Exit

Enter your choice (1-5): 5

Closing inventory manager.

== Code Execution Successful ==

## TASK-2

➤ Inventory Stock Manager:Reduce stock according to sales and identify zero stock.

- Code:

```
1 def process_sale(items_dict, sale_sku, sale_qty):  
2     if sale_sku not in items_dict:  
3         print(f"SKU {sale_sku} not present in inventory.")  
4         return  
5  
6     available = items_dict[sale_sku]  
7     if sale_qty > available:  
8         print(f"Insufficient stock for {sale_sku}. Available: {available}")  
9     else:  
10        items_dict[sale_sku] -= sale_qty  
11        print(f"Sale processed: {sale_qty} units of SKU {sale_sku}.")  
12  
13 def zero_stock_skus(items_dict):  
14     zeros = [code for code, q in items_dict.items() if q == 0]  
15     return zeros  
16  
17  
18 # sample usage  
19 item_levels = {  
20     101: 50,  
21     102: 20,  
22     103: 0,  
23     104: 10  
24 }  
25  
26 process_sale(item_levels, 101, 30)  
27 process_sale(item_levels, 102, 40)    # insufficient  
28 process_sale(item_levels, 999, 5)    # not found  
29  
30 print("Zero stock SKUs:", zero_stock_skus(item_levels))  
31 print("Updated Inventory:", item_levels)
```

## Output:

```
Sale processed: 30 units of SKU 101.  
Insufficient stock for 102. Available: 20  
SKU 999 not present in inventory.  
Zero stock SKUs: [103]  
Updated Inventory: {101: 20, 102: 20, 103: 0, 104: 10}  
  
==== Code Execution Successful ===
```

# TASK-3

Inventory system with two functionalities:

1. Calculate Total & Average Stock

2. Find Maximum Stock Item

\*CODE:

```
1 def calc_total_and_avg(stock_map):
2     if not stock_map:
3         return 0, 0
4     total_qty = sum(stock_map.values())
5     avg_qty = total_qty / len(stock_map)
6     return total_qty, avg_qty
7
8 def find_max_stock_item(stock_map):
9     if not stock_map:
10        return None, 0
11     max_sku = max(stock_map, key=lambda k: stock_map[k])
12     return max_sku, stock_map[max_sku]
13
14
15 def inventory_main_menu():
16     store_levels = {}
17     while True:
18         print("\n---- Inventory Menu ----")
19         print("1. Add Product to Inventory")
20         print("2. Calculate Total and Average Stock")
21         print("3. Find Maximum Stock Item")
22         print("4. Exit")
23         choice = input("Enter your choice (1-4): ")
24
25         if choice == "1":
26             code = input("Enter product SKU: ")
27             qty = int(input("Enter quantity: "))
28             store_levels[code] = qty
29             print(f"Product with SKU {code} and quantity {qty} added.")
30         elif choice == "2":
31             total, avg = calc_total_and_avg(store_levels)
32             print("Inventory:", store_levels)
33             print("Total Stock:", total)
34             print("Average Stock:", avg)
35         elif choice == "3":
36             sku, max_val = find_max_stock_item(store_levels)
37             if sku is None:
38                 print("Inventory empty.")
39             else:
40                 print(f"Item with max stock: SKU {sku}, Quantity {max_val}")
41         elif choice == "4":
42             print("Exiting program. Goodbye!")
43             break
44         else:
45             print("Invalid choice, try again.")
46
47 inventory_main_menu()
```

## OUTPUT:

```
---- Inventory Menu ----
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 1
Enter product SKU: 101
Enter quantity: 50
Product with SKU 101 and quantity 50 added.

---- Inventory Menu ----
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 1
Enter product SKU: 102
Enter quantity: 40
Product with SKU 102 and quantity 40 added.

---- Inventory Menu ----
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 2
Inventory: {'101': 50, '102': 40}
Total Stock: 90
Average Stock: 45.0

---- Inventory Menu ----
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 3
Item with max stock: SKU 101, Quantity 50

---- Inventory Menu ----
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 4
Exiting program. Goodbye!

==== Code Execution Successful ===
```

## TASK 4: STACK OPERATIONS: {push,pop,peek,display}

CODE:

```
1  stack_store = []
2  def push_item():
3      element = input("Enter item to push: ")
4      stack_store.append(element)
5      print("Stack now:", stack_store)
6  def pop_item():
7      if not stack_store:
8          print("Stack is empty, cannot pop.")
9      else:
10         removed = stack_store.pop()
11         print("Popped:", removed)
12         print("Stack now:", stack_store)
13 def peek_item():
14     if not stack_store:
15         print("Stack is empty.")
16     else:
17         print("Top element:", stack_store[-1])
18 def show_stack():
19     if not stack_store:
20         print("Stack is empty.")
21     else:
22         print("Stack contents:", stack_store)
23 def stack_menu():
24     while True:
25         print("\n1. Push")
26         print("2. Pop")
27         print("3. Peek")
28         print("4. Display")
29         print("5. Exit")
30         ch = input("Enter your choice: ")
31
32         if ch == "1":
33             push_item()
34         elif ch == "2":
35             pop_item()
36         elif ch == "3":
37             peek_item()
38         elif ch == "4":
39             show_stack()
40         elif ch == "5":
41             print("Exiting stack demo.")
42             break
43         else:
44             print("Invalid choice, please try again.")
45
46 stack_menu()
```

## Output:

```
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter item to push: Apple
Stack now: ['Apple']

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter item to push: Orange
Stack now: ['Apple', 'Orange']

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2
Enter your choice: 2
Popped: Orange
Stack now: ['Apple']

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element: Apple

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack contents: ['Apple']

1. Push
2. Pop
3. Peek
4. Display
```

## TASK-5: Implementation of stack browser:

Code:

```
1 history_pages = []
2 def visit_site(url):
3     history_pages.append(url)
4     print(f"Visited: {url}")
5 def go_back():
6     if not history_pages:
7         print("No pages in history.")
8         return
9     last_page = history_pages.pop()
10    print(f"Going back from: {last_page}")
11    if history_pages:
12        print("Current page:", history_pages[-1])
13    else:
14        print("No pages left in history.")
15
16 def display_history():
17     if not history_pages:
18         print("History is empty.")
19     else:
20         print("History:", " -> ".join(history_pages))
21
22
23 def browser_sim():
24     while True:
25         print("\n1. Visit Page")
26         print("2. Back")
27         print("3. Show History")
28         print("4. Exit")
29         choice = input("Enter choice: ")
30
31         if choice == "1":
32             page = input("Enter page URL/name: ")
33             visit_site(page)
34         elif choice == "2":
35             go_back()
36         elif choice == "3":
37             display_history()
38         elif choice == "4":
39             print("Exiting browser simulation.")
40             break
41         else:
42             print("Invalid choice.")
43
44 browser_sim()
```

## Output:

```
1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page URL/name: Google
Visited: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page URL/name: Youtube
Visited: Youtube

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 3
History: Google -> Youtube

1. Visit Page

2. Back
3. Show History
4. Exit
Enter choice: 2
Going back from: Youtube
Current page: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 3
History: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 4
Exiting browser simulation.

==== Code Execution Successful ===
```

## TASK-6: Insertion in circular linkedlist

CODE:

```
1 - class CNode:
2 -     def __init__(self, value):
3 -         self.data = value
4 -         self.next = None
5 - class CircularListInsert:
6 -     def __init__(self):
7 -         self.head = None
8 -     def insert_at_end(self, value):
9 -         new_node = CNode(value)
10 -        if self.head is None:
11 -            self.head = new_node
12 -            new_node.next = self.head
13 -        return
14 -        temp = self.head
15 -        while temp.next != self.head:
16 -            temp = temp.next
17 -            temp.next = new_node
18 -            new_node.next = self.head
19 -    def insert_at_begin(self, value):
20 -        new_node = CNode(value)
21 -        if self.head is None:
22 -            self.head = new_node
23 -            new_node.next = self.head
24 -        return
25 -        temp = self.head
26 -        while temp.next != self.head:
27 -            temp = temp.next
28 -            new_node.next = self.head
29 -            temp.next = new_node
30 -            self.head = new_node
31 -    def insert_at_pos(self, pos, value):
32 -        if self.head is None or pos <= 1:
33 -            self.insert_at_begin(value)
34 -            return
35 -        new_node = CNode(value)
36 -        temp = self.head
37 -        count = 1
38 -        while count < pos - 1 and temp.next != self.head:
39 -            temp = temp.next
40 -            count += 1
41 -        new_node.next = temp.next
42 -        temp.next = new_node
43 -    def show_list(self):
44 -        if self.head is None:
45 -            print("List is empty")
46 -            return
47 -        temp = self.head
48 -        while True:
```

```
49         print(temp.data, end=" -> ")
50         temp = temp.next
51     if temp == self.head:
52         break
53     print("(back to head)")
54 cl_ins = CircularListInsert()
55 cl_ins.insert_at_end(10)
56 cl_ins.insert_at_end(20)
57 cl_ins.insert_at_end(30)
58 print("Original list:")
59 cl_ins.show_list()
60 cl_ins.insert_at_begin(5)
61 print("\nAfter inserting at beginning:")
62 cl_ins.show_list()
63 cl_ins.insert_at_end(40)
64 print("\nAfter inserting at end:")
65 cl_ins.show_list()
66 cl_ins.insert_at_pos(3, 15)
67 print("\nAfter inserting 15 at position 3:")
68 cl_ins.show_list()
```

## OUTPUT:

```
Original list:
10 -> 20 -> 30 -> (back to head)

After inserting at beginning:
5 -> 10 -> 20 -> 30 -> (back to head)

After inserting at end:
5 -> 10 -> 20 -> 30 -> 40 -> (back to head)

After inserting 15 at position 3:
5 -> 10 -> 15 -> 20 -> 30 -> 40 -> (back to head)

==== Code Execution Successful ===
```

## TASK-7:

CODE:

```
1  class CNodeDel:
2      def __init__(self, value):
3          self.data = value
4          self.next = None
5  class CircularListDelete:
6      def __init__(self):
7          self.head = None
8      def append(self, value):
9          new_node = CNodeDel(value)
10     if self.head is None:
11         self.head = new_node
12         new_node.next = self.head
13         return
14     temp = self.head
15     while temp.next != self.head:
16         temp = temp.next
17     temp.next = new_node
18     new_node.next = self.head
19     def display(self):
20         if self.head is None:
21             print("List is empty")
22             return
23             print("Circular Linked List:", end=" ")
24             temp = self.head
25
26             while True:
27                 print(temp.data, end=" -> ")
28                 temp = temp.next
29                 if temp == self.head:
30                     break
31                     print("(back to head)")
32     def delete_begin(self):
33         if self.head is None:
34             print("List is empty, nothing to delete.")
35             return
36             if self.head.next == self.head:
37                 self.head = None
38                 print("Deleted the only node in the list.")
39                 return
40             last = self.head
41             while last.next != self.head:
42                 last = last.next
43                 self.head = self.head.next
44                 last.next = self.head
45                 print("Node deleted from beginning.")
46     def delete_end(self):
47         if self.head is None:
48             print("List is empty, nothing to delete.")
```

```
49         if self.head.next == self.head:
50             self.head = None
51             print("Deleted the only node in the list.")
52             return
53         prev = None
54         temp = self.head
55         while temp.next != self.head:
56             prev = temp
57             temp = temp.next
58         prev.next = self.head
59         print("Node deleted from end.")
60     def delete_at_pos(self, pos):
61         if self.head is None:
62             print("List is empty, nothing to delete.")
63             return
64         if pos == 1:
65             self.delete_begin()
66             return
67         prev = None
68         temp = self.head
69         count = 1
70         while count < pos and temp.next != self.head:
71             prev = temp
72             temp = temp.next
```

```
73             count += 1
74         if count != pos:
75             print("Position out of range.")
76             return
77         prev.next = temp.next
78         print(f"Node deleted from position {pos}.")
79 cl_del = CircularListDelete()
80 cl_del.append(10)
81 cl_del.append(20)
82 cl_del.append(30)
83 cl_del.append(40)
84 print("Initial List:")
85 cl_del.display()
86 cl_del.delete_begin()
87 cl_del.display()
88 cl_del.delete_end()
89 cl_del.display()
90 cl_del.delete_at_pos(2)
91 cl_del.display()
```

## OUTPUT:

```
Initial List:  
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (back to head)  
Node deleted from beginning.  
Circular Linked List: 20 -> 30 -> 40 -> (back to head)  
Node deleted from end.  
Circular Linked List: 20 -> 30 -> (back to head)  
Node deleted from position 2.  
Circular Linked List: 20 -> (back to head)  
  
==== Code Execution Successful ===
```

## TASK 8: Circular queue insertion and deletion

CODE:

```
1 MAX_SIZE = 100
2 class CircularQueue:
3     def __init__(self):
4         self.front = -1
5         self.rear = -1
6         self.buffer = [None] * MAX_SIZE
7     def is_full(self):
8         return (self.front == 0 and self.rear == MAX_SIZE - 1) or \
9             (self.rear + 1 == self.front)
10    def is_empty(self):
11        return self.front == -1
12    def enqueue(self, value):
13        if self.is_full():
14            print("Queue is full.")
15            return
16        if self.front == -1:
17            self.front = 0
18        self.rear = (self.rear + 1) % MAX_SIZE
19        self.buffer[self.rear] = value
20        print("Inserted element:", value)
21    def dequeue(self):
22        if self.is_empty():
23            print("Queue is empty.")
24            return
25        if self.front == self.rear:
26            self.front = self.rear = -1
27        else:
28            self.front = (self.front + 1) % MAX_SIZE
29        print("Deleted element:", value)
30    def display(self):
31        if self.is_empty():
32            print("Queue is empty.")
33            return
34        print("Elements in the queue:", end=" ")
35        i = self.front
36        while True:
37            print(self.buffer[i], end=" ")
38            if i == self.rear:
39                break
40            i = (i + 1) % MAX_SIZE
41        print()
42 cq = CircularQueue()
43 cq.enqueue(10)
44 cq.enqueue(20)
45 cq.enqueue(30)
46 cq.display()
47 cq.dequeue()
48 cq.display()
```

## OUTPUT:

```
Inserted element: 10
Inserted element: 20
Inserted element: 30
Elements in the queue: 10 20 30
Deleted element: 10
Elements in the queue: 20 30

==== Code Execution Successful ===
```

## TASK-9:

### LINEAR SEARCH

CODE:

```
1 def linear_search(arr, target):
2     for idx, val in enumerate(arr):
3         if val == target:
4             return idx
5     return -1
6
7 nums = [34, 7, 23, 32, 5, 62]
8 print("Array:", nums)
9 to_find = int(input("Enter element to search (linear): "))
10 pos = linear_search(nums, to_find)
11 if pos != -1:
12     print(f"Element found at index {pos}")
13 else:
14     print("Element not found")
```

OUTPUT:

```
Array: [34, 7, 23, 32, 5, 62]
Enter element to search (linear):
==== Session Ended. Please Run the code again ===
```

## TASK-10:

### BINARY SEARCH:

#### CODE:

```
1 import random
2
3 values = sorted(random.randint(1, 100) for _ in range(10))
4 print("Array:", values)
5
6 key = int(input("Enter element to search (binary): "))
7
8 low = 0
9 high = len(values) - 1
10 found_flag = False
11
12 while low <= high:
13     mid = (low + high) // 2
14     if values[mid] == key:
15         print("Element found at index", mid)
16         found_flag = True
17         break
18     elif values[mid] < key:
19         low = mid + 1
20     else:
21         high = mid - 1
22
23 if not found_flag:
24     print("Element not found")
```

#### OUTPUT:

```
Array: [3, 7, 13, 16, 22, 51, 62, 71, 92, 98]
Enter element to search (binary): |
```

## TASK 11:BUBBLE SORT

CODE:

```
1 def bubble_sort(data_list):
2     n = len(data_list)
3     for i in range(n - 1):
4         for j in range(0, n - 1 - i):
5             if data_list[j] > data_list[j + 1]:
6                 data_list[j], data_list[j + 1] = data_list[j + 1],
7                                         data_list[j]
8
9 numbers = [64, 25, 12, 22, 11]
10 print("Unsorted list:", numbers)
11 bubble_sort(numbers)
12 print("Sorted list:", numbers)
```

OUTPUT:

```
Unsorted list: [64, 25, 12, 22, 11]
Sorted list: [11, 12, 22, 25, 64]
```

```
==== Code Execution Successful ===
```