AI ASSISTED CODING ASS-11.1

2303A51678
B-28

Task Description #1 (Stack Implementation)

Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

CODE:

```python
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if self.is_empty():
            return "Stack is empty"
        return self.items.pop()

    def peek(self):
        if self.is_empty():
            return "Stack is empty"
        return self.items[-1]

    def is_empty(self):
        return len(self.items) == 0
```

Output:

20
20
False

Task Description #2 (Queue Implementation)

Task: Use AI to implement a Queue using Python lists.

CODE:

```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.items:
            return "Queue is empty"
        return self.items.pop(0)

    def peek(self):
        if not self.items:
            return "Queue is empty"
        return self.items[0]

    def size(self):
        return len(self.items)

```

Output:

1

1

1

Task Description #3 (Singly Linked List)

Task: Use AI to generate a Singly Linked List with insert and display methods.

CODE:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

Output:

5 -> 10 -> 15 -> None

Task Description #4 (Binary Search Tree – BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

CODE:

```python
class BST:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

    def insert(self, value):
        if value < self.value:
            if self.left is None:
                self.left = BST(value)
            else:
                self.left.insert(value)
        else:
            if self.right is None:
                self.right = BST(value)
            else:
                self.right.insert(value)

    def inorder(self):
        if self.left:
            self.left.inorder()
        print(self.value, end=" ")
        if self.right:
            self.right.inorder()
```

Output:

20 30 40 50 70

Task Description #5 (Hash Table)

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

CODE:

```python
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]

    def hash_function(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value
                return
        self.table[index].append([key, value])

    def search(self, key):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return "Key not found"

    def delete(self, key):
        index = self.hash_function(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                self.table[index].pop(i)
                return "Deleted"
        return "Key not found"
```

Output:

100
Deleted
Key not found

Task Description #6 (Graph Representation)

Task: Use AI to implement a graph using an adjacency list.

CODE:

```python
class Graph:
    def __init__(self):
        self.graph = {}

    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, v1, v2):
        if v1 not in self.graph:
            self.add_vertex(v1)
        if v2 not in self.graph:
            self.add_vertex(v2)
        self.graph[v1].append(v2)
        self.graph[v2].append(v1)

    def display(self):
        for vertex in self.graph:
            print(vertex, "->", self.graph[vertex])
```

Output:

A -> ['B', 'C']
B -> ['A']
C -> ['A']

Task Description #7 (Priority Queue)

Task: Use AI to implement a priority queue using Python's heapq module.

CODE:

```python
import heapq

class PriorityQueue:
    def __init__(self):
        self.queue = []

    def enqueue(self, priority, item):
        heapq.heappush(self.queue, (priority, item))

    def dequeue(self):
        if not self.queue:
            return "Queue is empty"
        return heapq.heappop(self.queue)

    def display(self):
        return self.queue
```

Output:

(1, 'High')

Task Description #8 (Deque)

Task: Use AI to implement a double-ended queue using collections.deque.

CODE:

```
from collections import deque

class DequeDS:
    def __init__(self):
        self.dq = deque()

    def add_front(self, item):
        self.dq.appendleft(item)

    def add_rear(self, item):
        self.dq.append(item)

    def remove_front(self):
        if not self.dq:
            return "Deque is empty"
        return self.dq.popleft()

    def remove_rear(self):
        if not self.dq:
            return "Deque is empty"
        return self.dq.pop()
```

Output:

10
20

Task Description #9 Real-Time Application Challenge – Campus Resource Management System.

CODE:

```
class CafeteriaQueue:
    def __init__(self):
        self.orders = []

    def place_order(self, student_name):
        self.orders.append(student_name)
        print("Order placed by", student_name)

    def serve_order(self):
        if not self.orders:
            print("No orders")
            return
        print("Serving", self.orders.pop(0))

    def display_orders(self):
        print("Current Orders:", self.orders)


cq = CafeteriaQueue()
cq.place_order("Ravi")
cq.place_order("Anjali")
cq.display_orders()
cq.serve_order()
cq.display_orders()
```

Output:

Order placed by Ravi
Order placed by Anjali
Current Orders: ['Ravi', 'Anjali']
Serving Ravi
Current Orders: ['Anjali']

Task Description #10 Smart E-Commerce Platform – Data Structure Challenge.

CODE:

```
class ProductSearchEngine:
    def __init__(self):
        self.products = {}

    def add_product(self, product_id, name):
        self.products[product_id] = name
        print("Product added:", name)

    def search_product(self, product_id):
        if product_id in self.products:
            print("Product Found:", self.products[product_id])
        else:
            print("Product not found")

    def remove_product(self, product_id):
        if product_id in self.products:
            print(self.products.pop(product_id), "removed")
        else:
            print("Product not found")


ps = ProductSearchEngine()
ps.add_product(101, "Laptop")
ps.add_product(102, "Mobile")
ps.search_product(101)
ps.remove_product(101)
ps.search_product(101)
```

Output:

Product added: Laptop
Product added: Mobile
Product Found: Laptop
Laptop removed
Product not found