K.R Mangalam University

Name- Ashwani Kumar thakur

Roll no – 2401420060

Course- B.tech DS

Sub – Data Structure

LAB file

Topic:

1. Inventry Stock Management System
2. singly Linked List
3. Circular singly Linked list
4. Ticketing System using Linear queue
5. Reverse string using Stack
6. Browser History Navigation
7. Check balanced Paranthesis using Stack

1.Inventry Stock Management System

InventoryManagement.py Transaction.class Update.py

code1.py > InventoryManagement.py

```python
##  Inventory Management

MAX_CAPACITY = 100

inventory = []

def insert_product():
    if len(inventory) >= MAX_CAPACITY:
        print("Error: Inventory capacity reached, cannot add more products.")
        return

    sku = input("Enter SKU: ").strip()
    if not sku:
        print("SKU cannot be empty.")
        return

    # Check if SKU exists
    for item in inventory:
        if item['sku'] == sku:

            print(f"Product with SKU {sku} already exists.")
            choice = input("Do you want to update the quantity? (y/n): ").lower()
            if choice == 'y':
                try:
                    new_qty = int(input("Enter new quantity: "))
                    if new_qty < 0:
                        print("Error: Quantity must be positive.")
                        return
                except ValueError:
                    print("Invalid input. Quantity must be a number.")
                    return
                item['quantity'] = new_qty
                print(f"Product SKU {sku} quantity updated to {new_qty}.")
            else:
                print("No changes made.")
            return

    name = input("Enter Product Name: ").strip()
    if not name:
        print("Error: Product name cannot be empty.")
        return

```

```
    for item in inventory:
        if item['sku'] == sku:

    name = input("Enter Product Name: ").strip()
    if not name:
        print("Error: Product name cannot be empty.")
        return

    try:
        quantity = int(input("Enter Quantity: "))
        if quantity < 0:
            print("Error: Quantity must be positive.")
            return
    except ValueError:
        print("Invalid input. Quantity must be a number.")
        return

    product = {'sku': sku, 'name': name, 'quantity': quantity}
    inventory.append(product)
    print("Product inserted successfully.")

def insert_multiple_products():
    while True:
        if len(inventory) >= MAX_CAPACITY:
            print("Error: Inventory capacity reached, cannot add more products.")
            break

        insert_product()

        cont = input("Do you want to insert another product? (y/n): ").lower()
        if cont != 'y':
            break

def display_inventory():
    if not inventory:
        print("Inventory is empty.")
        return
    print("\nCurrent Inventory:")
    print(f"{'SKU':<15}{'Product Name':<25}{'Quantity':<10}")
    print("-" * 55)
    for item in inventory:
        print(f"{item['sku']:<15}{item['name']:<25}{item['quantity']:<10}")
```

```
        print(f"{item['sku']:<15}{item['name']:<25}{item['quantity']:<10}")
    print()

def search_by_sku():
    sku = input("Enter SKU to search: ").strip()
    for item in inventory:
        if item['sku'] == sku:
            print(f"Product found:\nSKU: {item['sku']}\nName: {item['name']}\nQuantity: {item['quantity']}")
            return
    print("Product not found.")

def search_by_name():
    name = input("Enter Product Name to search: ").strip().lower()
    found = False
    for item in inventory:
        if item['name'].lower() == name:
            print(f"Product found:\nSKU: {item['sku']}\nName: {item['name']}\nQuantity: {item['quantity']}")
            found = True
    if not found:
        print("Product not found.")

def delete_product():
    sku = input("Enter SKU to delete: ").strip()
    for i, item in enumerate(inventory):
        if item['sku'] == sku:
            removed_product = inventory.pop(i)
            print(f"Product {removed_product['name']} with SKU {sku} removed from inventory.")
            return
    print("Product not found.")

def main():
    while True:
        print("\nInventory Stock Manager")
        print("1. Insert New Product(s)")
        print("2. Display Inventory")
        print("3. Search Product by SKU")
        print("4. Search Product by Name")
        print("5. Delete Product by SKU")
        print("6. Exit")
        choice = input("Enter your choice (1-6): ")
```

```
        print("6. Exit")
        choice = input("Enter your choice (1-6): ")

        if choice == '1':
            insert_multiple_products()
        elif choice == '2':
            display_inventory()
        elif choice == '3':
            search_by_sku()
        elif choice == '4':
            search_by_name()
        elif choice == '5':
            delete_product()
        elif choice == '6':
            print("Exiting Inventory Manager.")
            break
        else:
            print("Invalid choice. Please select from 1 to 6.")

if __name__ == "__main__":
    main()
```

```
Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Search Product by SKU
4. Search Product by Name
5. Delete Product by SKU
6. Exit
Enter your choice (1-6): 1
Enter SKU: A101
Enter Product Name: Glasses
Enter Quantity: 150
Product inserted successfully.
Do you want to insert another product? (y/n): y
Enter SKU: A102
Enter Product Name: shirt
Enter Quantity: 500
Product inserted successfully.
Do you want to insert another product? (y/n): n

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Search Product by SKU
4. Search Product by Name
5. Delete Product by SKU
6. Exit
Enter your choice (1-6): 2
```

```
Current Inventory:
SKU             Product Name                Quantity
-----------------------------------------------------
A101            Glasses                     150
A102            shirt                       500


Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Search Product by SKU
4. Search Product by Name
5. Delete Product by SKU
6. Exit
Enter your choice (1-6): 3
Enter SKU to search: A102
Product found:
SKU: A102
Name: shirt
Quantity: 500

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Search Product by SKU
4. Search Product by Name
```

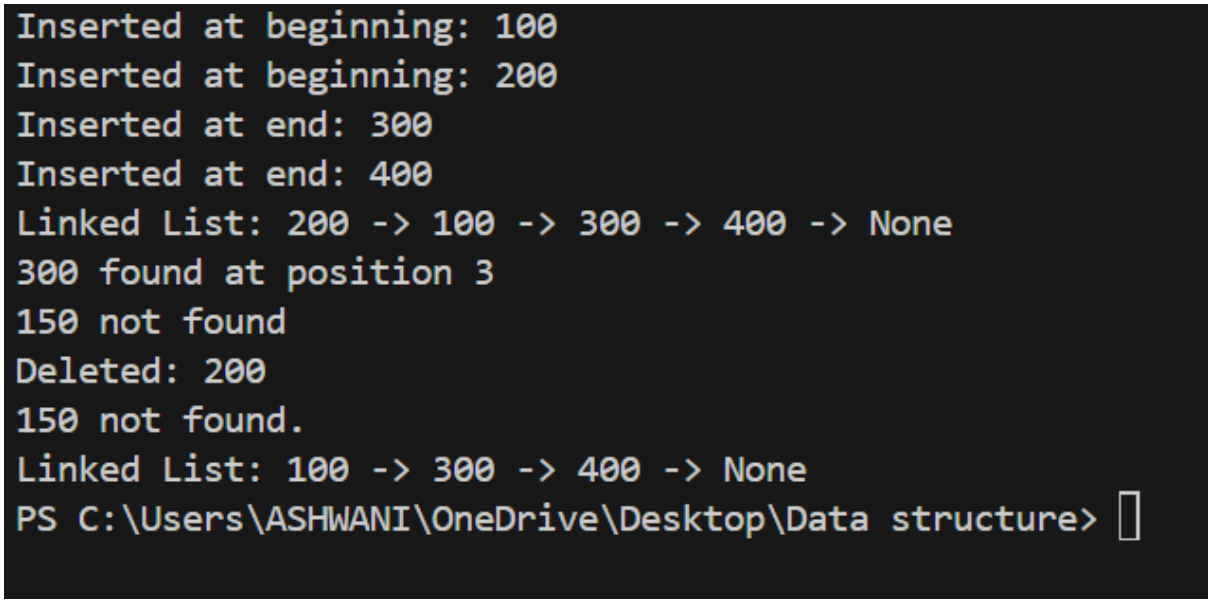
```
Enter your choice (1-6): 4
Enter Product Name to search: shirt
Product found:
SKU: A102
Name: shirt
Quantity: 500

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Search Product by SKU
4. Search Product by Name
5. Delete Product by SKU
6. Exit
Enter your choice (1-6): 5
Enter SKU to delete: A101
Product Glasses with SKU A101 removed from inventory.

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Search Product by SKU
4. Search Product by Name
5. Delete Product by SKU
6. Exit
Enter your choice (1-6): 6
Exiting Inventory Manager.
```

2.singly Linked List

```
    # Deleting middle/end
    prev = None
    while temp and temp.data != key:
        prev = temp
        temp = temp.next

    if temp is None:
        print(key, "not found.")
    else:
        prev.next = temp.next
        print("Deleted:", key)

# Searching a value
def search(self, key):
    temp = self.head
    pos = 1
    while temp:
        if temp.data == key:
            print(key, "found at position", pos)
            return True
        temp = temp.next
        pos += 1
    print(key, "not found")
    return False

# Displaying list
def display(self):
    if self.head is None:
        print("List is empty.")
        return
    temp = self.head
    print("Linked List:", end=" ")
    while temp:
        print(temp.data, end=" -> ")
        temp = temp.next
    print("None")
```

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


# Singly Linked List
class SinglyLinkedList:
    def __init__(self):
        self.head = None

    # Inserting at beginning
    def insert_begin(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node
        print("Inserted at beginning:", data)

    # Inserting at end
    def insert_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
        print("Inserted at end:", data)

    # Deleting a value
    def delete(self, key):
        temp = self.head

        # Emptying list
        if temp is None:
            print("List is empty.")
            return

        # Deleting head
        if temp.data == key:
            self.head = temp.next
            print("Deleted:", key)
```

Output:



3. Circular singly Linked list

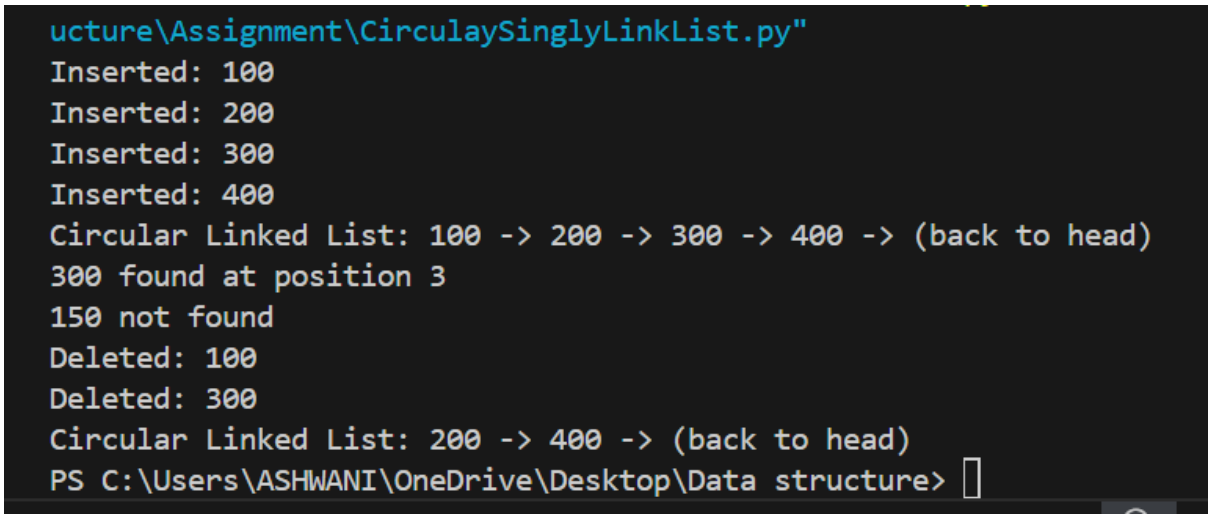
```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


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

    def insert(self, data):
        new = Node(data)

        if self.head is None:
            self.head = new
            new.next = new
            print("Inserted:", data)
            return

        temp = self.head
        while temp.next != self.head:
            temp = temp.next

        temp.next = new
        new.next = self.head
        print("Inserted:", data)

    def search(self, key):
        if self.head is None:
            print("List is empty.")
            return False

        temp = self.head
```

```
        pos = 1
        while True:
            if temp.data == key:
                print(key, "found at position", pos)
                return True
            temp = temp.next
            pos += 1
            if temp == self.head:
                break

        print(key, "not found")
        return False

    def delete(self, key):
        if self.head is None:
            print("List is empty.")
            return

        curr = self.head
        prev = None

        # Only one node
        if curr.data == key and curr.next == self.head:
            self.head = None
            print("Deleted:", key)
            return

        # Deleting head
        if curr.data == key:
            while curr.next != self.head:
                curr = curr.next
```

```
            curr.next = self.head.next
            self.head = self.head.next
            print("Deleted:", key)
            return

        # Middle / end node
        prev = self.head
        curr = self.head.next
        while curr != self.head:
            if curr.data == key:
                prev.next = curr.next
                print("Deleted:", key)
                return
            prev = curr
            curr = curr.next

        print(key, "not found")

    def display(self):
        if self.head is None:
            print("List is empty.")
            return

        temp = self.head
        print("Circular Linked List:", end=" ")
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
```

```
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(back to head)")


cll = CircularLinkedList()

cll.insert(100)
cll.insert(200)
cll.insert(300)
cll.insert(400)

cll.display()

cll.search(300)
cll.search(150)

cll.delete(100)
cll.delete(300)

cll.display()
```

Output:



4.Ticketing System using Linear queue

ssignment > TicketingSystem.py

```python
class TicketQueue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = -1
        self.rear = -1

    def isFull(self):
        return self.rear == self.size - 1

    def isEmpty(self):
        return self.front == -1 or self.front > self.rear

    def enqueue(self, ticket):
        if self.isFull():
            print("Queue is Full!")
            return
        if self.front == -1:
            self.front = 0
        self.rear += 1
        self.queue[self.rear] = ticket
        print("Ticket Added:", ticket)

    def dequeue(self):
        if self.isEmpty():
            print("Queue is Empty!")
            return
        print("Ticket Processed:", self.queue[self.front])
        self.front += 1

    def display(self):
        if self.isEmpty():
            print("No pending tickets.")
```

```
    def display(self):
        if self.isEmpty():
            print("No pending tickets.")
        else:
            print("Pending Tickets:", *self.queue[self.front : self.rear + 1])


def main():
    q = TicketQueue(5)

    while True:
        print("\n1. Add Ticket")
        print("2. Process Ticket")
        print("3. Show Tickets")
        print("4. Exit")

        ch = input("Enter choice: ")

        if ch == "1":
            q.enqueue(input("Enter Ticket ID: "))
        elif ch == "2":
            q.dequeue()
        elif ch == "3":
            q.display()
        elif ch == "4":
            print("Exiting...")
            break
        else:
            print("Invalid choice!")


if __name__ == "__main__":
```

Output:

```
1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: python -u "c:\Users\ASHWANI\OneDrive\Desktop\Da
Invalid choice!

1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: 1
Enter Ticket ID: T1
Ticket Added: T1

1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: 1
Enter Ticket ID: T2
Ticket Added: T2

1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: 3
```

```
Enter choice: 3
Pending Tickets: T1 T2

1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: 2
Ticket Processed: T1

1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: 2
Ticket Processed: T2

1. Add Ticket
2. Process Ticket
3. Show Tickets
4. Exit
Enter choice: 4
Exiting...
PS C:\Users\ASHWANI\OneDrive\Desktop\Data structure>
```

5.Reverse string using Stack

```python
# Reverse a string using stack (simple version)

def reverse_using_stack(s):
    stack = []

    for ch in s:
        stack.append(ch)

    rev = ""
    while stack:
        rev += stack.pop()

    return rev

text = input("Enter a string: ")
print("Original String:", text)

reversed_text = reverse_using_stack(text)
print("Reversed String:", reversed_text)

```

Output:

```
Enter a string: Ashwani
Original String: Ashwani
Reversed String: inawhsA
PS C:\Users\ASHWANI\OneDrive\Desktop\Data structure>
```

6.Browser History Navigation

```
back_stack = []
forward_stack = []
current_page = None

while True:
    print("\n1. Visit New Page")
    print("2. Go Back")
    print("3. Go Forward")
    print("4. Show History")
    print("5. Exit")

    ch = input("Enter choice: ")

    # Visit a new page
    if ch == "1":
        page = input("Enter page name/URL: ")

        if current_page is not None:
            back_stack.append(current_page)

        current_page = page
        forward_stack.clear()

        print("Visited:", current_page)

    # Go Back
    elif ch == "2":
        if not back_stack:
            print("No pages to go back to.")
        else:
            forward_stack.append(current_page)
            current_page = back_stack.pop()
            print("Back to:", current_page)
```

```
    # Go Forward
    elif ch == "3":
        if not forward_stack:
            print("No pages to go forward to.")
        else:
            back_stack.append(current_page)
            current_page = forward_stack.pop()
            print("Forward to:", current_page)

    # Show History
    elif ch == "4":
        print("\n--- Browser History ---")
        print("Back Stack:", back_stack)
        print("Current Page:", current_page)
        print("Forward Stack:", forward_stack)
        print("-------------------------")

    # Exit
    elif ch == "5":
        print("Exiting...")
        break

    else:
        print("Invalid choice!")
```

Output:

```
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name/URL: google
Visited: google

1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 3
No pages to go forward to.

1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name/URL: youtube
Visited: youtube

1. Visit New Page
2. Go Back
```

```
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 4

--- Browser History ---
Back Stack: ['google']
Current Page: youtube
Forward Stack: []
-----------------------

1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 2
Back to: google

1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 5
Exiting...
```

7.Check balanced Paranthesis using Stack

```python
def is_balanced(expr):
    stack = []
    pairs = {')': '(', '}': '{', ']': '['}

    for ch in expr:
        if ch in "([{":
            stack.append(ch)
        elif ch in ")]}":
            if not stack or stack.pop() != pairs[ch]:
                return False

    return stack == []

expression = input("Enter an expression: ")

if is_balanced(expression):
    print("Parentheses are balanced.")
else:
    print("Parentheses are NOT balanced.")
```

Output:

```
ucture\Assignment\CheckBalancedParanthes
Enter an expression: (a+b)
Parentheses are balanced.
```