K.R. Mangalam University

ENCS203 – Data Structures

Lab File

Name: Abhishek Thakur

Roll number: 2401420020

Course: B.Tech CSE (Data Science)

Submitted to: Ms. Swati Gupta

Index

1. Browser History Navigation System using Stack concept

Code:

```python
# Browsing History navigation using two stacks

def visit(url, back_stack, forward_stack, current):
    if current is not None:
        back_stack.append(current)
    current = url
    forward_stack.clear()
    return current

def go_back(back_stack, forward_stack, current):
    if not back_stack:
        return current
    forward_stack.append(current)
    current = back_stack.pop()
    return current

def go_forward(back_stack, forward_stack, current):
    if not forward_stack:
        return current
    back_stack.append(current)
    current = forward_stack.pop()
    return current

def show_state(back_stack, forward_stack, current):
    print("Back stack:", back_stack)
    print("Current:", current)
    print("Forward stack:", list(reversed(forward_stack)))

if __name__ == '__main__':
    back = []
    forward = []
    current = None
    while True:
        print("\n1) Visit URL  2) Back  3) Forward  4) Show  5) Quit")
        choice = input("> ").strip()
        if choice == '1':
            url = input("URL: ").strip()
            if url:
                current = visit(url, back, forward, current)
        elif choice == '2':
            current = go_back(back, forward, current)
        elif choice == '3':
            current = go_forward(back, forward, current)
        elif choice == '4':
            show_state(back, forward, current)
        elif choice == '5':
            break
        else:
            print("invalid choice")
```

Output:

```
1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 1
URL: www.google.com

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 1
URL: www.youtube.com

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 1
URL: www.wikipedia.org

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 4
Back stack: ['www.google.com', 'www.youtube.com']
Current: www.wikipedia.org
Forward stack: []

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 2

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 4
Back stack: ['www.google.com']
Current: www.youtube.com
Forward stack: ['www.wikipedia.org']

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 3

1) Visit URL  2) Back  3) Forward  4) Show  5) Quit
> 4
Back stack: ['www.google.com', 'www.youtube.com']
Current: www.wikipedia.org
Forward stack: []
```

2. TICKETING SYSTEM USING QUEUE (LINEAR QUEUE IMPLEMENTATION)

Code:

```python
# Ticketing System using linear queue
class Queue:
    def __init__(self):
        self.items = []

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

    def dequeue(self):
        if not self.items:
            return None
        return self.items.pop(0)

    def show(self):
        return self.items


if __name__ == '__main__':
    q = Queue()
    counter = 1

    while True:
        print("\n1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit")
        choice = input("> ").strip()

        if choice == '1':
            ticket = f"T{counter}"
            q.enqueue(ticket)
            print("Generated:", ticket)
            counter += 1

        elif choice == '2':
            served = q.dequeue()
            if served:
                print("Served:", served)
            else:
                print("No tickets to serve")

        elif choice == '3':
            print("Queue:", q.show())

        elif choice == '4':
            break

        else:
            print("invalid choice")
```

Output:

```
1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 3
Queue: []

1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 1
Generated: T1

1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 1
Generated: T2

1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 1
Generated: T3

1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 3
Queue: ['T1', 'T2', 'T3']

1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 2
Served: T1

1) Generate Ticket  2) Serve Ticket  3) Show Queue  4) Quit
> 3
Queue: ['T2', 'T3']
```

3. Singly Linked List Operations: Insert, Delete, Search, Display

Code:

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

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

    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def delete_at_beginning(self):
        if self.head is None:
            print("List is empty.")
            return
        print("Deleting:", self.head.data)
        self.head = self.head.next

    def delete_at_end(self):
        if self.head is None:
            print("List is empty.")
            return
        if self.head.next is None:
            print("Deleting:", self.head.data)
            self.head = None
            return
        temp = self.head
        while temp.next.next:
            temp = temp.next
        print("Deleting:", temp.next.data)
        temp.next = None
```

```
47
48      def insert_after(self, value, new_value):
49          temp = self.head
50          while temp:
51              if temp.data == value:
52                  new_node = Node(new_value)
53                  new_node.next = temp.next
54                  temp.next = new_node
55                  print(f"Inserted {new_value} after {value}")
56                  return
57              temp = temp.next
58          print("Value not found.")
59
60      def delete_after(self, value):
61          temp = self.head
62          while temp:
63              if temp.data == value:
64                  if temp.next:
65                      print(f"Deleting: {temp.next.data}")
66                      temp.next = temp.next.next
67                  else:
68                      print("No node exists after the given value.")
69                  return
70              temp = temp.next
71          print("Value not found.")
72
73      def display(self):
74          temp = self.head
75          print("List:", end=" ")
76          while temp:
77              print(temp.data, end=" -> ")
78              temp = temp.next
79          print("None")
80
81      def search(self, key):
82          temp = self.head
83          while temp:
84              if temp.data == key:
85                  return True
86              temp = temp.next
87          return False
88
```

```python
if __name__ == "__main__":
    ll = LinkedList()
    ll.insert_at_beginning(5)
    ll.insert_at_beginning(10)
    ll.insert_at_beginning(15)

    while True:
        print("\n--- Linked List Menu ---")
        print("1. Insert at beginning")
        print("2. Insert at end")
        print("3. Insert after value")
        print("4. Delete at beginning")
        print("5. Delete at end")
        print("6. Delete after value")
        print("7. Display list")
        print("8. Exit")

        ch = input("> ").strip()

        if ch == '1':
            x = input("Enter value: ")
            ll.insert_at_beginning(x)

        elif ch == '2':
            x = input("Enter value: ")
            ll.insert_at_end(x)

        elif ch == '3':
            v = input("Insert after which value? ")
            x = input("Enter new value: ")
            ll.insert_after(v, x)

        elif ch == '4':
            ll.delete_at_beginning()

        elif ch == '5':
            ll.delete_at_end()

        elif ch == '6':
            v = input("Delete after which value? ")
            ll.delete_after(v)

        elif ch == '7':
            ll.display()

        elif ch == '8':
            break

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

Output:

```
--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 7
List: 15 -> 10 -> 5 -> None

--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 1
Enter value: 1

--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 5
Deleting: 5

--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 2
Enter value: 2
```

```
--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 7
List: 1 -> 15 -> 10 -> 2 -> None

--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 3
Insert after which value? 1
Enter new value: 99
Inserted 99 after 1
```

```
--- Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert after value
4. Delete at beginning
5. Delete at end
6. Delete after value
7. Display list
8. Exit
> 7
List: 1 -> 99 -> 15 -> 10 -> 2 -> None
```

4. Circular Singly Linked List Operations: Insert, Delete, Search, Display

Code:

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

# Circular Linked List class
class CircularLinkedList:
    def __init__(self):
        self.head = None

    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
            return
        temp = self.head
        while temp.next != self.head:
            temp = temp.next
        temp.next = new_node
        new_node.next = self.head

    def insert_at_beginning(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
            return
        temp = self.head
        while temp.next != self.head:
            temp = temp.next
        new_node.next = self.head
        temp.next = new_node
        self.head = new_node

```

```
37      def insert_in_middle(self, data, position):
38          if position <= 1 or self.head is None:
39              self.insert_at_beginning(data)
40              return
41
42          new_node = Node(data)
43          temp = self.head
44          count = 1
45
46          while count < position - 1 and temp.next != self.head:
47              temp = temp.next
48              count += 1
49
50          new_node.next = temp.next
51          temp.next = new_node
52
53      def search(self, key):
54          if self.head is None:
55              return False
56          temp = self.head
57          while True:
58              if temp.data == key:
59                  return True
60              temp = temp.next
61              if temp == self.head:
62                  break
63          return False
64
```

```
65      def delete(self, key):
66          if self.head is None:
67              print("List is empty.")
68              return
69
70          temp = self.head
71          prev = None
72
73          while True:
74              if temp.data == key:
75                  if prev is None:
76                      # Deleting head
77                      if temp.next == self.head:
78                          self.head = None
79                          print("Deleted:", key)
80                          return
81                      last = self.head
82                      while last.next != self.head:
83                          last = last.next
84                      self.head = temp.next
85                      last.next = self.head
86                  else:
87                      prev.next = temp.next
88                  print("Deleted:", key)
89                  return
90
91              prev = temp
92              temp = temp.next
93              if temp == self.head:
94                  break
95
96          print("Value not found.")
```

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

        temp = self.head
        prev = None

        while True:
            if temp.data == key:
                if prev is None:
                    # Deleting head
                    if temp.next == self.head:
                        self.head = None
                        print("Deleted:", key)
                        return
                    last = self.head
                    while last.next != self.head:
                        last = last.next
                    self.head = temp.next
                    last.next = self.head
                else:
                    prev.next = temp.next
                print("Deleted:", key)
                return

            prev = temp
            temp = temp.next
            if temp == self.head:
                break

        print("Value not found.")

    def display(self):
        if self.head is None:
            print("List is empty")
            return
        temp = self.head
        print("List:", end=" ")
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(back to head)")

```

```python
if __name__ == "__main__":
    ll = CircularLinkedList()
    ll.insert_at_end(5)
    ll.insert_at_end(10)
    ll.insert_at_end(15)
    ll.insert_at_end(20)

    while True:
        print("\n--- Circular Linked List Menu ---")
        print("1. Insert at beginning")
        print("2. Insert at end")
        print("3. Insert at a position")
        print("4. Search")
        print("5. Delete by value")
        print("6. Display list")
        print("7. Exit")
        ch = input("> ").strip()
        if ch == '1':
            x = input("Enter value: ")
            ll.insert_at_beginning(x)
        elif ch == '2':
            x = input("Enter value: ")
            ll.insert_at_end(x)
        elif ch == '3':
            x = input("Enter value: ")
            p = int(input("Enter position: "))
            ll.insert_in_middle(x, p)
        elif ch == '4':
            key = input("Search value: ")
            print("Found" if ll.search(key) else "Not found")
        elif ch == '5':
            key = input("Delete which value? ")
            ll.delete(key)
        elif ch == '6':
            ll.display()
        elif ch == '7':
            break
        else:
            print("Invalid choice.")
```

Output:

```
--- Circular Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Search
5. Delete by value
6. Display list
7. Exit
> 6
List: 5 -> 10 -> 15 -> 20 -> (back to head)

--- Circular Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Search
5. Delete by value
6. Display list
7. Exit
> 1
Enter value: 1

--- Circular Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Search
5. Delete by value
6. Display list
7. Exit
> 2
Enter value: 25

--- Circular Linked List Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at a position
4. Search
5. Delete by value
6. Display list
7. Exit
> 6
List: 1 -> 5 -> 10 -> 15 -> 20 -> 25 -> (back to head)
```

5. Reverse a String Using Stack

Code:

```python
# Reverse input string, stack implementation
def reverse_string(s):
    stack = []
    for ch in s:
        stack.append(ch)
    result = ""
    while stack:
        result += stack.pop()
    return result


if __name__ == "__main__":
    while True:
        s = input("Enter string to reverse: ")
        print("Reversed:", reverse_string(s))
```

Output:

```
Enter string to reverse: Glory to the republic
Reversed: cilbuper eht ot yrolG
Enter string to reverse: I am the king of the world!
Reversed: !dlrow eht fo gnik eht ma I
```

6. Check Balanced Parantheses Using Stack

Code:

```python
# Check if parentheses are balanced, stack implementation
def is_balanced(expr):
    stack = []
    pairs = {')': '(', '}': '{', ']': '['}

    for ch in expr:
        if ch in "({[":
            stack.append(ch)
        elif ch in ")}]":
            if not stack or stack[-1] != pairs[ch]:
                return False
            stack.pop()

    return not stack


if __name__ == "__main__":
    while True:
        expr = input("Enter any expression to see if the parentheses are balanced: ")
        print("Balanced" if is_balanced(expr) else "Not Balanced")
```

Output:

```
Enter any expression to see if the parentheses are balanced: hello()world()
Balanced
Enter any expression to see if the parentheses are balanced: hello(world))
Not Balanced
Enter any expression to see if the parentheses are balanced: [{{hi}}]]
Not Balanced
Enter any expression to see if the parentheses are balanced: ({[]})
Balanced
```

7. LAB PROJECT: INVENTORY STOCK MANAGEMENT SYSTEM

Code:

```python
# Inventory list to store product records
inventory = []

# Function to insert one or more products
def insert_product():
    SKUs = input("Enter SKU(s): ").replace(',', ' ').split()
    names = input("Enter product name(s): ").replace(',', ' ').split()
    quantities = input("Enter quantity/quantities: ").replace(',', ' ').split()

    # Validation checks
    if len(names) != len(SKUs):
        print("Error: The number of SKUs and product names must match.")
        return
    if len(quantities) != len(SKUs):
        print("Error: The number of SKUs and quantities must match.")
        return

    for i in range(len(SKUs)):
        sku = SKUs[i].strip()
        name = names[i].strip()
        qty_str = quantities[i].strip()

        if not sku:
            print("Error: SKU cannot be empty.")
            continue
        if sku in [item['sku'] for item in inventory]:
            print(f"Error: SKU {sku} already exists in the inventory.")
            continue
        try:
            qty = int(qty_str)
            inventory.append({'sku': sku, 'name': name, 'quantity': qty})
            print(f"Product '{name}' with SKU {sku} inserted successfully.")
        except ValueError:
            print(f"Error: Quantity '{qty_str}' for SKU {sku} is not a valid number.")

# Function to display inventory
def display_inventory():
    if not inventory:
        print("Inventory is empty.")
        return
    print("\nCurrent Inventory:")
    print("SKU\t\tProduct Name\t\tQuantity")
    print("-------------------------------------------------")
    for item in inventory:
        print(f"{item['sku']}\t\t{item['name']}\t\t{item['quantity']}")
    print()

```

```python
# Function to process sale
def process_sale():
    sku = input("Enter SKU for sale: ")
    try:
        qty_sold = int(input("Enter quantity sold: "))
    except ValueError:
        print("Invalid input. Quantity must be a number.")
        return

    for item in inventory:
        if item['sku'] == sku:
            if item['quantity'] >= qty_sold:
                item['quantity'] -= qty_sold
                print(f"Sale processed: {qty_sold} units of SKU {sku}.")
            else:
                print(f"Insufficient stock for SKU {sku}. Available: {item['quantity']}")
            return
    print(f"SKU {sku} not found in inventory.")

# Function to identify zero stock
def identify_zero_stock():
    zero_stock_list = [item['sku'] for item in inventory if item['quantity'] == 0]
    if zero_stock_list:
        print("Zero stock SKUs:")
        for sku in zero_stock_list:
            print(sku)
    else:
        print("No zero stock items found.")
    return zero_stock_list

# Function to calculate total and average stock (operates on dict-based inventory)
def total_and_avg():
    if not inventory:
        return 0, 0  # handle empty inventory
    total = sum(item['quantity'] for item in inventory)
    avg = total / len(inventory)
    return total, avg

# Function to find the item with maximum stock (returns SKU and Quantity)
def max_stock_item():
    if not inventory:
        return None, None
    max_item = max(inventory, key=lambda x: x['quantity'])
    return max_item['sku'], max_item['quantity']

```

```python
# Main program loop
def menu():
    while True:
        print("\nInventory Stock Manager")
        print("1. Insert New Product(s)")
        print("2. Display Inventory")
        print("3. Process Sale")
        print("4. Identify Zero Stock Items")
        print("5. Calculate Total and Average Stock")
        print("6. Find Maximum Stock Item")
        print("7. Exit")
        choice = input("Enter your choice (1-7): ")

        if choice == '1':
            insert_product()
        elif choice == '2':
            display_inventory()
        elif choice == '3':
            process_sale()
        elif choice == '4':
            identify_zero_stock()
        elif choice == '5':
            total, avg = total_and_avg()
            print(f"Inventory: {inventory}")
            print(f"Total Stock: {total}")
            print(f"Average Stock: {avg:.2f}")
        elif choice == '6':
            sku, qty = max_stock_item()
            if sku is None:
                print("Inventory is empty! Add products first.")
            else:
                # attempt to display name if present
                name = next((it['name'] for it in inventory if it['sku'] == sku), None)
                if name:
                    print(f"Item with Maximum Stock: SKU {sku}, Name: {name}, Quantity {qty}")
                else:
                    print(f"Item with Maximum Stock: SKU {sku}, Quantity {qty}")
        elif choice == '7':
            print("Exiting Inventory Manager.")
            break
        else:
            print("Invalid choice. Please select from 1 to 5.")

# Start the program when executed directly
if __name__ == "__main__":
    menu()
```

Output:

```
Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 2
Inventory is empty.

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 1
Enter SKU(s): 101, 102, 103, 104
Enter product name(s): pen, pencil, eraser, scale
Enter quantity/quantities: 10, 15, 5, 8
Product 'pen' with SKU 101 inserted successfully.
Product 'pencil' with SKU 102 inserted successfully.
Product 'eraser' with SKU 103 inserted successfully.
Product 'scale' with SKU 104 inserted successfully.

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 2

Current Inventory:
SKU             Product Name            Quantity
-------------------------------------------------
101             pen             10
102             pencil          15
103             eraser          5
104             scale           8
```

```
Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 3
Enter SKU for sale: 103
Enter quantity sold: 5
Sale processed: 5 units of SKU 103.

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 4
Zero stock SKUs:
103

Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 6
Item with Maximum Stock: SKU 102, Name: pencil, Quantity 15
```

```
Inventory Stock Manager
1. Insert New Product(s)
2. Display Inventory
3. Process Sale
4. Identify Zero Stock Items
5. Calculate Total and Average Stock
6. Find Maximum Stock Item
7. Exit
Enter your choice (1-7): 5
Inventory: [{'sku': '101', 'name': 'pen', 'quantity': 10}, {'sku': '102',
 'name': 'pencil', 'quantity': 15}, {'sku': '103', 'name': 'eraser', 'qua
ntity': 0}, {'sku': '104', 'name': 'scale', 'quantity': 8}]
Total Stock: 33
Average Stock: 8.25
```