**LAB 6**

**SEMISTER 3rd**

fall-2024

**SUBJECT**

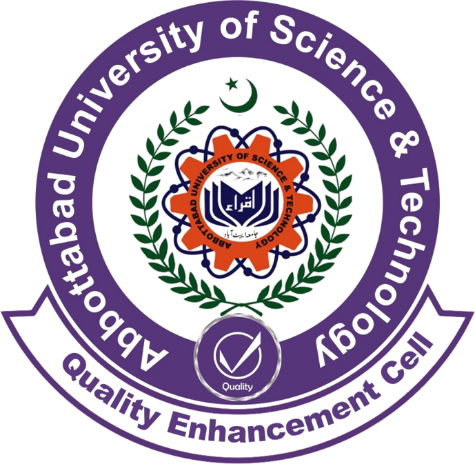
Database Structures

**COURSE CODE**

CC210

**PROGRAMME**

BS (4 year)



**ABBOTTABAD UNIVERSITY OF SCIENCE AND TECHNOLOGY ISLAMABAD**

**SUBMITTED TO**

**Mr. Jamal Abdul Ahad**

**SUBMITTED BY**

**Haroon Imran**

**STUDENT ROLL NO**

14723

**SUBMISSION DATE**

17 Dec 2024

**Activity 1: Array Initialization**

**1. Write a program to declare and initialize an array.**

**Example**

arr = [10, 20, 30, 40, 50]

print("Array elements:", arr)

**Activity 2: Traversing an Array**

**1. Write a program to print all elements of an array using a loop.**

**Example (C++):**

int arr[] = {1, 2, 3, 4, 5};

for (int i = 0; i < 5; i++) {

cout << arr[i] << " ";

}

**Section 2: Operations on Arrays**

**Activity 3: Insertion in an Array**

**1. Write a program to insert an element at a specific index.**

**Example (Python):**

arr = [10, 20, 30, 40, 50]

index = 2

value = 25

arr.insert(index, value)

print("Array after insertion:", arr)

**Activity 4: Deletion from an Array**

**1. Write a program to delete an element from a specific index.**

**Example (Java):**

int[] arr = {10, 20, 30, 40, 50};

int indexToRemove = 2;

for (int i = indexToRemove; i < arr.length - 1; i++) {

arr[i] = arr[i + 1];

}

System.out.println(Arrays.toString(Arrays.copyOf(arr,

arr.length - 1)));

**Activity 5: Searching in an Array**

**1. Write a program to search for an element in an array.**

**Example (Python):**

arr = [10, 20, 30, 40, 50]

value = 30

if value in arr:

print(f"{value} found at index {arr.index(value)}")

else:

print(f"{value} not found")

**Activity 6: Sorting an Array**

**1. Write a program to sort an array in ascending and descending order.**

**Example (C++):**

#include <algorithm>

int arr[] = {50, 30, 20, 40, 10};

sort(arr, arr + 5); // Ascending

reverse(arr, arr + 5); // Descending

**Section 3: Advanced Exercises**

**Activity 7: Merging Two Arrays**

**1. Write a program to merge two arrays into one.**

**Example (Python):**

arr1 = [1, 3, 5]

arr2 = [2, 4, 6]

merged = arr1 + arr2

print("Merged array:", merged)

**Activity 8: Rotating an Array**

**1. Write a program to rotate an array by a given number of positions.**

**Example (Python):**

arr = [1, 2, 3, 4, 5]

k = 2

rotated = arr[k:] + arr[:k]

print("Rotated array:", rotated)

**Activity 9: Two-Dimensional Arrays**

**1. Write a program to perform operations on a 2D array.**

**Example (C++):**

int matrix[2][2] = {{1, 2}, {3, 4}};

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

**Lab Outline:**

**Section 1: Introduction to Stacks**

**Activity 1: Basic Stack Operations Using Lists**

**1. Task: Write a Python program to implement basic stack operations using lists.**

2. stack = []

3.

4. # Push operation

5. stack.append(10)

6. stack.append(20)

7. stack.append(30)

8. print("Stack after pushes:", stack)

9.

10. # Pop operation

11. print("Popped element:", stack.pop())

12. print("Stack after pop:", stack)

13.

14. # Peek operation

15. if stack:

16. print("Top element:", stack[-1])

17. else:

18. print("Stack is empty")

**Section 2: Stack Implementation**

**Activity 2: Implementing a Stack Using a Class**

**1. Task: Define a custom Stack class with methods for common stack operations.**

2. class Stack:

3. def \_\_init\_\_(self):

4. self.items = []

5.

6. def push(self, item):

7. self.items.append(item)

8.

9. def pop(self):

10. if not self.is\_empty():

11. return self.items.pop()

12. else:

13. return "Stack is empty"

14.

15. def peek(self):

16. if not self.is\_empty():

17. return self.items[-1]

18. else:

19. return "Stack is empty"

20.

21. def is\_empty(self):

22. return len(self.items) == 0

23.

24. def size(self):

25. return len(self.items)

26.

27. # Example usage

28. stack = Stack()

29. stack.push(10)

30. stack.push(20)

31. print("Top element:", stack.peek())

32. print("Popped element:", stack.pop())

33. print("Stack size:", stack.size())

**Activity 3: Using collections.deque for Efficient Stacks**

**1. Task: Implement a stack using the deque class for better performance.**

2. from collections import deque

3.

4. stack = deque()

5.

6. # Push

7. stack.append(10)

8. stack.append(20)

9. print("Stack after pushes:", stack)

10.

11. # Pop

12. print("Popped element:", stack.pop())

13. print("Stack after pop:", stack)

14.

15. # Peek

16. if stack:

17. print("Top element:", stack[-1])

18. else:

19. print("Stack is empty")

**Section 3: Stack Applications**

**Activity 4: Balancing Parentheses**

**1. Task: Write a program to check if parentheses in an expression are balanced.**

2. def is\_balanced(expression):

3. stack = []

4. pairs = {')': '(', '}': '{', ']': '['}

5.

6. for char in expression:

7. if char in '({[':

8. stack.append(char)

9. elif char in ')}]':

10. if not stack or stack[-1] != pairs[char]:

11. return False

12. stack.pop()

13.

14. return len(stack) == 0

15.

16. # Example usage

17. expr = "{[()()]}"

18. print("Is balanced:", is\_balanced(expr))

**Activity 5: Reverse a String**

**1. Task: Use a stack to reverse a string.**

2. def reverse\_string(s):

3. stack = []

4. for char in s:

5. stack.append(char)

6.

7. reversed\_str = ''

8. while stack:

9. reversed\_str += stack.pop()

10.

11. return reversed\_str

12.

13. # Example usage

14. string = "hello"

15. print("Original:", string)

16. print("Reversed:", reverse\_string(string))

**Activity 6: Evaluate Postfix Expression**

**1. Task: Write a program to evaluate a postfix (Reverse Polish Notation) expression.**

2. def evaluate\_postfix(expression):

3. stack = []

4.

5. for char in expression:

6. if char.isdigit():

7. stack.append(int(char))

8. else:

9. b = stack.pop()

10. a = stack.pop()

11. if char == '+':

12. stack.append(a + b)

13. elif char == '-':

14. stack.append(a - b)

15. elif char == '\*':

16. stack.append(a \* b)

17. elif char == '/':

18. stack.append(a / b)

19.

20. return stack.pop()

21.

22. # Example usage

23. postfix\_expr = "231\*+9-"

24. print("Result of postfix evaluation:",

evaluate\_postfix(postfix\_expr))

**Activity 7: Implementing Undo-Redo Functionality**

**1. Task: Simulate undo and redo functionality using two stacks.**

2. class UndoRedo:

3. def \_\_init\_\_(self):

4. self.undo\_stack = []

5. self.redo\_stack = []

6.

7. def perform\_action(self, action):

8. self.undo\_stack.append(action)

9. self.redo\_stack.clear()

10.

11. def undo(self):

12. if self.undo\_stack:

13. action = self.undo\_stack.pop()

14. self.redo\_stack.append(action)

15. return action

16. return "Nothing to undo"

17.

18. def redo(self):

19. if self.redo\_stack:

20. action = self.redo\_stack.pop()

21. self.undo\_stack.append(action)

22. return action

23. return "Nothing to redo"

24.

25. # Example usage

26. ur = UndoRedo()

27. ur.perform\_action("Write code")

28. ur.perform\_action("Debug code")

29. print("Undo:", ur.undo())

30. print("Redo:", ur.redo())

**Activity 1: Basic Queue Operations Using Lists**

**1. Task: Write a Python program to implement basic queue operations using lists.**

2. queue = []

3.

4. # Enqueue operation

5. queue.append(10)

6. queue.append(20)

7. queue.append(30)

8. print("Queue after enqueues:", queue)

9.

10. # Dequeue operation

11. if queue:

12. print("Dequeued element:", queue.pop(0))

13. else:

14. print("Queue is empty")

15.

16. # Peek operation

17. if queue:

18. print("Front element:", queue[0])

19. else:

20. print("Queue is empty")

21. Experiment:

Enqueue elements of different types (e.g., integers, strings).

Handle edge cases such as dequeuing from an empty queue.

**Section 2: Queue Implementation**

**Activity 2: Implementing a Queue Using a Class**

**1. Task: Define a custom Queue class with methods for common queue operations.**

2. class Queue:

3. def \_\_init\_\_(self):

4. self.items = []

5.

6. def enqueue(self, item):

7. self.items.append(item)

8.

9. def dequeue(self):

10. if not self.is\_empty():

11. return self.items.pop(0)

12. else:

13. return "Queue is empty"

14.

15. def peek(self):

16. if not self.is\_empty():

17. return self.items[0]

18. else:

19. return "Queue is empty"

20.

21. def is\_empty(self):

22. return len(self.items) == 0

23.

24. def size(self):

25. return len(self.items)

26.

27. # Example usage

28. queue = Queue()

29. queue.enqueue(10)

30. queue.enqueue(20)

31. print("Front element:", queue.peek())

32. print("Dequeued element:", queue.dequeue())

33. print("Queue size:", queue.size())

Activity 3: Using collections.deque for Efficient Queues

1. Task: Implement a queue using the deque class for better performance.

2. from collections import deque

3.

4. queue = deque()

5.

6. # Enqueue

7. queue.append(10)

8. queue.append(20)

9. print("Queue after enqueues:", queue)

10.

11. # Dequeue

12. if queue:

13. print("Dequeued element:", queue.popleft())

14. else:

15. print("Queue is empty")

16.

17. # Peek

18. if queue:

19. print("Front element:", queue[0])

20. else:

21. print("Queue is empty")

**Section 3: Advanced Queue Operations**

**Activity 4: Circular Queue**

**1. Task: Implement a circular queue using a list.**

2. class CircularQueue:

3. def \_\_init\_\_(self, capacity):

4. self.capacity = capacity

5. self.queue = [None] \* capacity

6. self.front = self.rear = -1

7.

8. def is\_full(self):

9. return (self.rear + 1) % self.capacity == self.front

10.

11. def is\_empty(self):

12. return self.front == -1

13.

14. def enqueue(self, item):

15. if self.is\_full():

16. return "Queue is full"

17. elif self.is\_empty():

18. self.front = self.rear = 0

19. else:

20. self.rear = (self.rear + 1) % self.capacity

21. self.queue[self.rear] = item

22.

23. def dequeue(self):

24. if self.is\_empty():

25. return "Queue is empty"

26. item = self.queue[self.front]

27. if self.front == self.rear:

28. self.front = self.rear = -1

29. else:

30. self.front = (self.front + 1) % self.capacity

31. return item

32.

33. def display(self):

34. if self.is\_empty():

35. return "Queue is empty"

36. i = self.front

37. result = []

38. while True:

39. result.append(self.queue[i])

40. if i == self.rear:

41. break

42. i = (i + 1) % self.capacity

43. return result

44.

45. # Example usage

46. cq = CircularQueue(5)

47. cq.enqueue(10)

48. cq.enqueue(20)

49. print("Queue:", cq.display())

50. print("Dequeued:", cq.dequeue())

51. print("Queue after dequeue:", cq.display())

Activity 5: Priority Queue

1. Task: Implement a priority queue using heapq. 2. import heapq

3.

4. class PriorityQueue:

5. def \_\_init\_\_(self):

6. self.heap = []

7.

8. def enqueue(self, priority, item):

9. heapq.heappush(self.heap, (priority, item))

10.

11. def dequeue(self):

12. if not self.is\_empty():

13. return heapq.heappop(self.heap)[1]

14. else:

15. return "Queue is empty"

16.

17. def is\_empty(self):

18. return len(self.heap) == 0

19.

20. # Example usage

21. pq = PriorityQueue()

22. pq.enqueue(2, "Task B")

23. pq.enqueue(1, "Task A")

24. pq.enqueue(3, "Task C")

25. print("Dequeued:", pq.dequeue())

26. print("Dequeued:", pq.dequeue())

Section 4: Queue Applications

**Activity 6: Task Scheduling**

**1. Task: Simulate task scheduling using a queue**.

2. from collections import deque

3.

4. def task\_scheduler(tasks):

5. queue = deque(tasks)

6. while queue:

7. task = queue.popleft()

8. print(f"Processing task: {task}")

9. # Simulate task completion or rescheduling

10. if task.startswith("Reschedule"):

11. queue.append(task)

 Example usage

14. tasks = ["Task1", "Task2", "Reschedule Task3", "Task4"]

15. task\_scheduler(tasks)

Activity 7: Hot Potato Game

1. Task: Implement the "Hot Potato" game using a queue.

2. from collections import deque

3.

4. def hot\_potato(players, num):

5. queue = deque(players)

6. while len(queue) > 1:

7. for \_ in range(num):

8. queue.append(queue.popleft())

9. print(f"Eliminated: {queue.popleft()}")

10. return queue[0]

11.

12. # Example usage

13. players = ["Alice", "Bob", "Charlie", "David"]

14. print("Winner:", hot\_potato(players, 3))