ABBOTTABAD UNIVERSITY OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Final Project

Submitted By:

Name	**:**	**Abdul Hanan**
Roll No	**:**	**14681**
Discipline	**:**	**CS 3rd (A)**
Subject	**:**	**DSA**
Date	**:**	**2/1/2025**

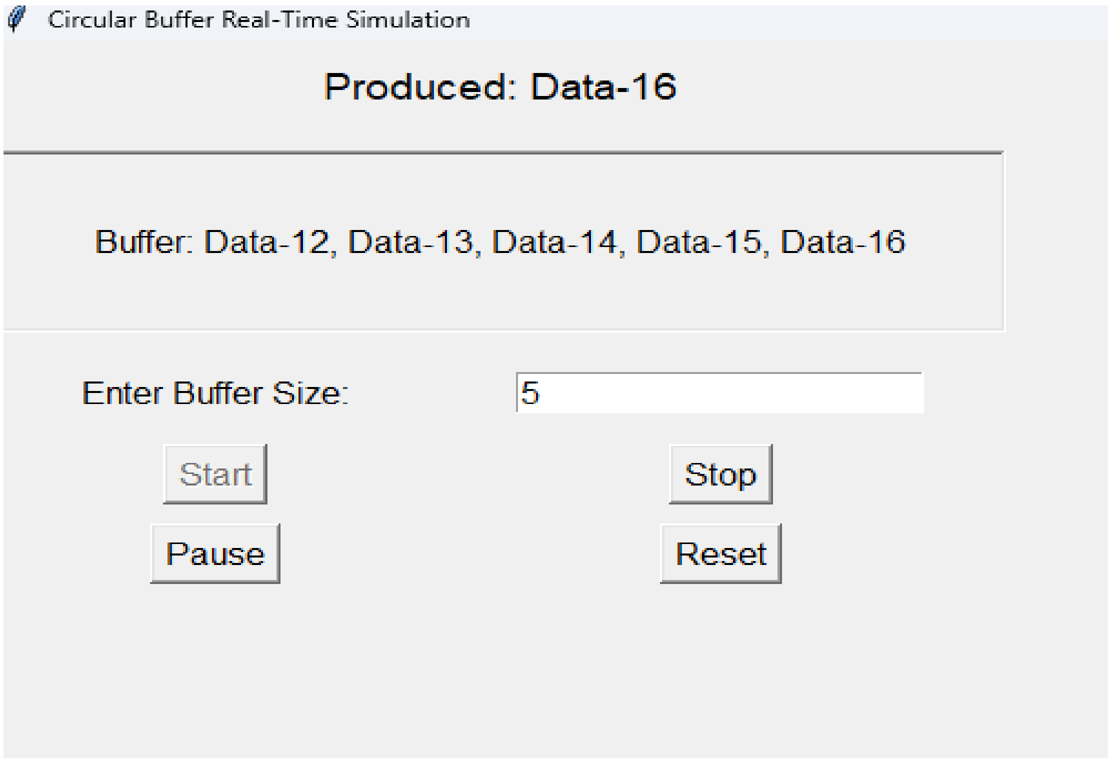
Submitted to:

SIR JAMAL ABDUL AHAD

```
import time
import threading
import tkinter as tk
from collections import deque


# Circular Buffer Class
class CircularBuffer:
    def __init__(self, size):
        self.size = size
        self.buffer = deque(maxlen=size)
        self.lock = threading.Lock()

    def write(self, data):
        with self.lock:
            self.buffer.append(data)

    def read(self):
        with self.lock:
            if len(self.buffer) > 0:
                return self.buffer.popleft()
            else:
                return None

    def is_full(self):
        with self.lock:
            return len(self.buffer) == self.size

    def is_empty(self):
        with self.lock:
            return len(self.buffer) == 0

    def get_buffer(self):
        with self.lock:
            return list(self.buffer)

    def reset(self):
        with self.lock:
            self.buffer.clear()
```

```
# Producer Thread to simulate data generation
def producer(buffer, stop_event, update_callback):
   counter = 0
   while not stop_event.is_set():
      if not buffer.is_full():
         data = f"Data-{counter}"
         buffer.write(data)
         update_callback(f"Produced: {data}")
         counter += 1
      time.sleep(0.5)  # Simulate time delay in data production


# Consumer Thread to simulate data processing
def consumer(buffer, stop_event, update_callback):
   while not stop_event.is_set():
      data = buffer.read()
      if data:
         update_callback(f"Consumed: {data}")
      else:
         update_callback("Buffer is empty, waiting for data...")
      time.sleep(1)  # Simulate time delay in data processing


# GUI to visualize the Circular Buffer processing
class CircularBufferApp:
   def __init__(self, root, buffer_size=5):
      self.root = root
      self.root.title("Circular Buffer Real-Time Simulation")
      self.buffer_size = buffer_size
      self.buffer = CircularBuffer(buffer_size)
      self.stop_event = threading.Event()

      # Create GUI elements
      self.info_label = tk.Label(self.root, text="Real-Time Data
Processing", font=("Helvetica", 14))
      self.info_label.grid(row=0, column=0, columnspan=2, pady=10)

      self.buffer_display = tk.Label(self.root, text="Buffer: Empty",
font=("Helvetica", 12), relief="sunken",
                            height=5, width=50)
      self.buffer_display.grid(row=1, column=0, columnspan=2,
pady=10)

      self.buffer_size_label = tk.Label(self.root, text="Enter Buffer
Size:", font=("Helvetica", 12))
      self.buffer_size_label.grid(row=2, column=0, pady=10)

      self.buffer_size_entry = tk.Entry(self.root, font=("Helvetica", 12))
      self.buffer_size_entry.grid(row=2, column=1, pady=10)
      self.buffer_size_entry.insert(0, str(buffer_size))

      self.start_button = tk.Button(self.root, text="Start",
command=self.start_simulation, font=("Helvetica", 12))
      self.start_button.grid(row=3, column=0, pady=5)

      self.stop_button = tk.Button(self.root, text="Stop",
command=self.stop_simulation, state=tk.DISABLED,
                            font=("Helvetica", 12))
      self.stop_button.grid(row=3, column=1, pady=5)

      self.pause_button = tk.Button(self.root, text="Pause",
command=self.pause_simulation, state=tk.DISABLED,
                            font=("Helvetica", 12))
      self.pause_button.grid(row=4, column=0, pady=5)

      self.reset_button = tk.Button(self.root, text="Reset",
command=self.reset_simulation, font=("Helvetica", 12))
      self.reset_button.grid(row=4, column=1, pady=5)

      # Update function for the producer and consumer
      self.update_gui_callback = self.update_buffer_display

   def start_simulation(self):
      try:
         self.buffer_size = int(self.buffer_size_entry.get())
         self.buffer = CircularBuffer(self.buffer_size)
         self.stop_event.clear()
         self.start_button.config(state=tk.DISABLED)
         self.stop_button.config(state=tk.NORMAL)
         self.pause_button.config(state=tk.NORMAL)

         # Start producer and consumer threads
         self.producer_thread = threading.Thread(target=producer,
                                                 args=(self.buffer, self.stop_event,
self.update_gui_callback))
         self.consumer_thread = threading.Thread(target=consumer,
                                                 args=(self.buffer, self.stop_event,
self.update_gui_callback))

         self.producer_thread.start()
         self.consumer_thread.start()
      except ValueError:
         self.update_buffer_display("Invalid buffer size input. Please
enter a valid number.")

   def stop_simulation(self):
      self.stop_event.set()
      self.start_button.config(state=tk.NORMAL)
      self.stop_button.config(state=tk.DISABLED)
      self.pause_button.config(state=tk.DISABLED)

      self.producer_thread.join()
      self.consumer_thread.join()
      self.update_buffer_display("Simulation stopped.")

   def pause_simulation(self):
      if self.stop_event.is_set():
         self.stop_event.clear()
         self.pause_button.config(text="Pause", state=tk.NORMAL)
         self.update_buffer_display("Simulation resumed.")
      else:
         self.stop_event.set()
         self.pause_button.config(text="Resume", state=tk.NORMAL)
         self.update_buffer_display("Simulation paused.")

   def reset_simulation(self):
      self.buffer.reset()
      self.update_buffer_display("Simulation reset.")
      self.start_button.config(state=tk.NORMAL)
      self.stop_button.config(state=tk.DISABLED)
      self.pause_button.config(state=tk.DISABLED)

   def update_buffer_display(self, message):
      # Update the buffer content and message
      current_buffer = self.buffer.get_buffer()
      buffer_text = f"Buffer: {', '.join(current_buffer) if current_buffer
else 'Empty'}"

      # Update GUI components in the main thread
      self.buffer_display.config(text=buffer_text)
      self.info_label.config(text=message)


# Run the application
if __name__ == "__main__":
   root = tk.Tk()
   app = CircularBufferApp(root)
   root.mainloop()
```

OUTPUT:



```
Circular Buffer Real-Time Simulation

Produced: Data-16

Buffer: Data-12, Data-13, Data-14, Data-15, Data-16

Enter Buffer Size:      5
```