LABORATORY PRACTICE-I

//SPOS codes

1.Pass I of two pass Assembler

```
import java.util.HashMap;
import java.util.Map;
public class Pass1Assembler{
public static void main(String[] args) {
// Print Intermediate Code
System.out.println("Intermediate Code:");
String[] intermediateCode = {
"(AD, 01) (C,100)",
"(IS, 04) (1) (S,0)",
"(IS, 01) (2) (L,0)",
"(IS, 05) (1) (S,1)",
"(IS, 02) (3) (L,1)",
"(AD, 05)",
"(IS, 01) (4) (L,2)",
"(DL, 01) (10)",
"(AD, 05)",
"(IS, 02) (1) (L,1)",
"(DL, 02) (1)",
"(DL, 02) (1)",
"(AD, 02)"
};
for (String code : intermediateCode) {
System.out.println(code);
}
```

```
// Print MOT Table
System.out.println("\nMOT Table:");
Map<String, String[]> motTable = new HashMap<>();
motTable.put("START", new String[] {"AD", "01", "0"});
motTable.put("END", new String[] {"AD", "02", "0"});
motTable.put("LTORG", new String[] {"AD", "05", "0"});
motTable.put("ADD", new String[] {"IS", "01", "1"});
motTable.put("SUB", new String[] {"IS", "02", "1"});
motTable.put("MULT", new String[] {"IS", "03", "1"});
motTable.put("MOVER", new String[] {"IS", "04", "1"});
motTable.put("MOVEM", new String[] {"IS", "05", "1"});
motTable.put("DS", new String[] {"DL", "01", "0"});
motTable.put("DC", new String[] {"DL", "02", "1"});
for (Map.Entry<String, String[]> entry : motTable.entrySet()) {
System.out.println(entry.getKey() + " " + entry.getValue()[0] + " " + entry.getValue()[1] + " " +
entry.getValue()[2]);
}
// Print Literal Table
System.out.println("\nLiteral Table:");
Map<String, Integer> literalTable = new HashMap<>();
literalTable.put("='6'", 102);
literalTable.put("='1'", 104);
literalTable.put("='5'", 105);
literalTable.put("='1'", 106);
for (Map.Entry<String, Integer> entry : literalTable.entrySet()) {
System.out.println(entry.getKey() + " " + entry.getValue());
}
```

```
// Print Symbol Table
System.out.println("\nSymbol Table:");
Map<String, Integer[]> symbolTable = new HashMap<>();
symbolTable.put("B", new Integer[] {101, 1});
symbolTable.put("A", new Integer[] {103, 1});
for (Map.Entry<String, Integer[]> entry : symbolTable.entrySet()) {
System.out.println(entry.getKey() + " " + entry.getValue()[0] + " " + entry.getValue()[1]);
}
}
}
```

2.Pass II of two pass Assembler

```
import java.util.*;
class Symbol {
  String name;
  int address;

  Symbol(String name, int address) {
    this.name = name;
    this.address = address;
  }
}

public class TwoPassAssemblerPass2 {
  private static final List<Symbol> symbolTable = Arrays.asList(
      new Symbol("START", 0),
      new Symbol("A", 1),
      new Symbol("B", 2),
      new Symbol("END", 3)
  );

  private static final Map<String, String> opcodeMap = Map.of(
      "LOAD", "0001", // 01 in hex
      "STORE", "0010", // 02 in hex
      "ADD", "0011", // 03 in hex
      "SUB", "0100", // 04 in hex
      "JUMP", "0101" // 05 in hex
  );

  public static void main(String[] args) {
    String[] instructions = {
        "LOAD A",
        "ADD B",
        "STORE A",
        "JUMP START"
    };

    System.out.println("Machine Code:");
    for (String instruction : instructions) {
      generateMachineCode(instruction);
    }
  }

  private static void generateMachineCode(String instruction) {
    String[] parts = instruction.split(" ");
    String opcode = opcodeMap.get(parts[0]); // Get the opcode in binary
    int address = getAddress(parts[1]); // Get the address from the symbol table
    String machineCode = opcode + String.format("%04d",
Integer.parseInt(Integer.toBinaryString(address))); // Concatenate opcode and address
    System.out.println(machineCode); // Print the machine code
  }

  private static int getAddress(String symbol) {
    for (Symbol s : symbolTable) {
      if (s.name.equals(symbol)) {
        return s.address;
      }
    }
    return -1; // Not found
  }
}
```

3.Pass-I of Two pass macro processor

```
import java.util.*;
class MacroDefinition {
  String name;
  List<String> parameters;
  List<String> body;

  MacroDefinition(String name, List<String> parameters, List<String> body) {
    this.name = name;
```

```java
        this.parameters = parameters;
        this.body = body;
    }
}

public class MacroProcessorPass1 {
    private static final List<MacroDefinition> macros = new ArrayList<>();
    private static final Set<String> macroNames = new HashSet<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the assembly code (type 'END' to finish):");

        String line;
        while (!(line = sc.nextLine()).equals("END")) {
            processLine(line.trim());
        }

        // Display the collected macro definitions
        System.out.println("\nMacro Definitions:");
        for (MacroDefinition macro : macros) {
            System.out.println("Macro Name: " + macro.name);
            System.out.println("Parameters: " + macro.parameters);
            System.out.println("Body:");
            for (String bodyLine : macro.body) {
                System.out.println("  " + bodyLine);
            }
            System.out.println();
        }

        sc.close();
    }

    private static void processLine(String line) {
        if (line.startsWith("MACRO")) {
            String macroName = line.split("\\s+")[1]; // Get the macro name
            List<String> parameters = new ArrayList<>();
            List<String> body = new ArrayList<>();

            // Read macro parameters
            while (!(line = readNextLine()).equals("ENDM")) {
                body.add(line);
            }

            macros.add(new MacroDefinition(macroName, parameters, body));
            macroNames.add(macroName);
        }
    }
    private static String readNextLine() {
        Scanner sc = new Scanner(System.in);
        return sc.nextLine().trim();
    }
}
```

4.Pass-II of two pass macro processor

```
import java.util.*;


class Macro {
    String name;
    List<String> body;


    Macro(String name, List<String> body) {
        this.name = name;
        this.body = body;
    }
}


public class MacroProcessorPass2 {
    private static final List<Macro> macros = new ArrayList<>();


    public static void main(String[] args) {
        // Example macro definitions (usually filled from Pass 1)
        defineMacros();


        Scanner sc = new Scanner(System.in);
        System.out.println("Enter code with macro calls (type 'END' to finish):");


        String line;
        while (!(line = sc.nextLine()).equals("END")) {
            processLine(line);
        }
        sc.close();
    }

    private static void defineMacros() {
        macros.add(new Macro("M1", Arrays.asList("MOVE A, B", "ADD A, C")));
        macros.add(new Macro("M2", Arrays.asList("SUB A, D")));
    }

    private static void processLine(String line) {
        String[] parts = line.split("\\s+");
        if (macros.stream().anyMatch(macro -> macro.name.equals(parts[0]))) {
            for (Macro macro : macros) {
                if (macro.name.equals(parts[0])) {
                    System.out.println(String.join("\n", macro.body));
                }
            }
        } else {
            System.out.println(line); // Print as-is if not a macro call
        }
    }
}
```

5.Scheduling Algorithms

1)FCFS

```
import java.util.Scanner;

public class FCFS {
```

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the number of processes: ");
    int n = sc.nextInt();
    int[] burstTime = new int[n];
    int[] waitingTime = new int[n];
    int[] turnaroundTime = new int[n];

    for (int i = 0; i < n; i++) {
        System.out.print("Enter burst time for process " + (i + 1) + ": ");
        burstTime[i] = sc.nextInt();
    }

    // Calculate waiting time
    waitingTime[0] = 0;
    for (int i = 1; i < n; i++) {
        waitingTime[i] = waitingTime[i - 1] + burstTime[i - 1];
    }

    // Calculate turnaround time
    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = waitingTime[i] + burstTime[i];
    }

    // Display results
    System.out.println("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
    for (int i = 0; i < n; i++) {
            System.out.println((i + 1) + "\t" + burstTime[i] + "\t\t" + waitingTime[i] + "\t\t" + turnaroundTime[i]);
        }
        sc.close();
    }
}
```

2)SJF(Preemptive)

```
import java.util.Scanner;
class SJF{
public static void main(String args[]){
int burst_time[],process[],waiting_time[],tat[],i,j,n,total=0,pos,temp;
float wait_avg,TAT_avg;
Scanner s = new Scanner(System.in);
System.out.print("Enter number of process: ");
n = s.nextInt();
process = new int[n];
burst_time = new int[n];
waiting_time = new int[n];
tat = new int[n];
System.out.println("\nEnter Burst time:");
for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)+"]: ");
burst_time[i] = s.nextInt();;
process[i]=i+1; //Process Number
}
//Sorting
```

```
for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(burst_time[j]<burst_time[pos])

pos=j;

}

temp=burst_time[i];

burst_time[i]=burst_time[pos];

burst_time[pos]=temp;

temp=process[i];

process[i]=process[pos];

process[pos]=temp;

}

//First process has 0 waiting time

waiting_time[0]=0;

//calculate waiting time

for(i=1;i<n;i++)

{

waiting_time[i]=0;

for(j=0;j<i;j++)

waiting_time[i]+=burst_time[j];

total+=waiting_time[i];

}

//Calculating Average waiting time

wait_avg=(float)total/n;
```

```
total=0;
System.out.println("\nProcess\t Burst Time \tWaiting
Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=burst_time[i]+waiting_time[i]; //Calculating Turnaround
Time
total+=tat[i];
System.out.println("\n p"+process[i]+"\t\t "+burst_time[i]+"\t\t
"+waiting_time[i]+"\t\t "+tat[i]);
}
//Calculation of Average Turnaround Time
TAT_avg=(float)total/n;
System.out.println("\n\nAverage Waiting Time: "+wait_avg);
System.out.println("\nAverage Turnaround Time: "+TAT_avg);
}}
```

3)Round Robin(Preemptive)

```
import java.util.Scanner;
public class Roundfinal1 {

 public static void main(String args[]) {
 Scanner s = new Scanner(System.in);
 int wtime[],btime[],rtime[],num,quantum,total;
 wtime = new int[10];
 btime = new int[10];
 rtime = new int[10];
System.out.print("Enter number of processes(MAX 10): ");
num = s.nextInt();
System.out.print("Enter burst time");
for(int i=0;i<num;i++) { System.out.print("\nP["+(i+1)+"]: ");
btime[i] = s.nextInt(); rtime[i] = btime[i]; wtime[i]=0; }
System.out.print("\n\nEnter quantum: "); quantum = s.nextInt();
int rp = num; int i=0; int time=0; System.out.print("0");
wtime[0]=0; while(rp!=0) { if(rtime[i]>quantum)
{
 rtime[i]=rtime[i]-quantum;
 System.out.print(" | P["+(i+1)+"] | ");
 time+=quantum;
 System.out.print(time);
 }
else if(rtime[i]<=quantum && rtime[i]>0)
{time+=rtime[i];
 rtime[i]=rtime[i]-rtime[i];
 System.out.print(" | P["+(i+1)+"] | ");
 rp--;
System.out.print(time);
}
i++;
if(i==num)
{
i=0;
}}
}}
```

4)Priority (Non-Preemptive)

```
import java.util.*;
class Process {
    int id, burstTime, priority;

    Process(int id, int burstTime, int priority) {
        this.id = id;
        this.burstTime = burstTime;
        this.priority = priority;
    }
}

public class PriorityScheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of processes: ");
        int n = sc.nextInt();
        Process[] processes = new Process[n];

        for (int i = 0; i < n; i++) {
            System.out.printf("Enter Burst Time and Priority for Process %d: ", i + 1);
            int burstTime = sc.nextInt();
            int priority = sc.nextInt();
            processes[i] = new Process(i + 1, burstTime, priority);
        }

        Arrays.sort(processes, Comparator.comparingInt(p -> p.priority)); // Sort by priority
    System.out.println("Order of execution:");
    for (Process p : processes) {
      System.out.printf("Process %d (Burst Time: %d, Priority: %d)%n", p.id, p.burstTime, p.priority);
    }
    sc.close();
  }
}
```

6.Memory placement techniques (BEST, WORST, FIRST, NEXT)

```
import java.util.Scanner;
class MemoryPlacement {
  private int[] blocks;

  MemoryPlacement(int[] sizes) {
    blocks = sizes.clone();
  }

  void allocateProcess(int size, String strategy) {
    int index = -1;
    for (int i = 0; i < blocks.length; i++) {
      if (blocks[i] >= size) {
        if (strategy.equals("First Fit")) { index = i; break; }
        if (strategy.equals("Best Fit") && (index == -1 || blocks[i] < blocks[index])) index = i;
        if (strategy.equals("Worst Fit") && (index == -1 || blocks[i] > blocks[index])) index = i;
      }
```

```java
        }
        if (index != -1) {
            blocks[index] -= size;
            System.out.println("Allocated " + size + " to block " + (index + 1) + " (" + strategy + ")");
        } else {
            System.out.println("Could not allocate " + size + " (" + strategy + ")");
        }
    }
}

public class MemoryPlacementStrategies {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of blocks: ");
        int[] blocks = new int[sc.nextInt()];
        System.out.println("Enter block sizes:");
        for (int i = 0; i < blocks.length; i++) blocks[i] = sc.nextInt();
        MemoryPlacement mp = new MemoryPlacement(blocks);

        System.out.print("Enter number of processes: ");
        int[] sizes = new int[sc.nextInt()];
        System.out.println("Enter process sizes:");
        for (int i = 0; i < sizes.length; i++) sizes[i] = sc.nextInt();

        String[] strategies = {"First Fit", "Best Fit", "Worst Fit"};
        for (String s : strategies) {
            System.out.println("\n" + s + " Strategy:");
        mp = new MemoryPlacement(blocks);
        for (int size : sizes) mp.allocateProcess(size, s);
      }
      sc.close();
    }}
```

7.Page Replacement Algo

1)FIFO

```
import java.util.*;
class FIFO {
    private final List<Integer> pages;
    private final int capacity;

    FIFO(List<Integer> pages, int capacity) {
        this.pages = pages;
        this.capacity = capacity;
    }

    void execute() {
        List<Integer> frames = new ArrayList<>();
        int faults = 0;

        for (int page : pages) {
            if (!frames.contains(page)) { // Page fault occurs
                if (frames.size() == capacity) frames.remove(0); // Remove the oldest page
                frames.add(page);
                faults++;
            }
        }
        System.out.println("FIFO Page Faults: " + faults);
    }
}
public class FIFOPageReplacement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of pages: ");
        int n = sc.nextInt();
        List<Integer> pages = new ArrayList<>();
        System.out.println("Enter page reference sequence:");
        for (int i = 0; i < n; i++) pages.add(sc.nextInt());
        System.out.print("Enter frame capacity: ");
        int capacity = sc.nextInt();

        new FIFO(pages, capacity).execute();
        sc.close();
    }
}
```

2)Optimal

```
import java.util.*;
class Optimal {
    private final List<Integer> pages;
    private final int capacity;

    Optimal(List<Integer> pages, int capacity) {
```

```
    this.pages = pages;
    this.capacity = capacity;
}

void execute() {
    Set<Integer> frames = new HashSet<>();
    int faults = 0;

    for (int i = 0; i < pages.size(); i++) {
        if (frames.add(pages.get(i))) { // Page fault occurs
            if (frames.size() > capacity) {
                frames.remove(findOptimal(frames, i));
            }
            faults++;
        }
    }
    System.out.println("Optimal Page Faults: " + faults);
}

private int findOptimal(Set<Integer> frames, int currentIndex) {
    int farthest = -1, toRemove = -1;
    for (int frame : frames) {
        int nextUse = pages.subList(currentIndex + 1, pages.size()).indexOf(frame);
        if (nextUse == -1) return frame; // Not used again
        if (nextUse > farthest) {
            farthest = nextUse;
            toRemove = frame;
            }
        }
        return toRemove;
    }
}

public class OptimalPageReplacement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of pages: ");
        int n = sc.nextInt();
        List<Integer> pages = new ArrayList<>();
        System.out.println("Enter page reference sequence:");
        for (int i = 0; i < n; i++) pages.add(sc.nextInt());
        System.out.print("Enter frame capacity: ");
        int capacity = sc.nextInt();

        new Optimal(pages, capacity).execute();
        sc.close();
    }
}
```

3)LRU

```
import java.util.*;
class LRU {
    private final List<Integer> pages;
    private final int capacity;
    LRU(List<Integer> pages, int capacity) {
        this.pages = pages;
        this.capacity = capacity;
    }

    void execute() {
        List<Integer> frames = new ArrayList<>();
        int faults = 0;

        for (int page : pages) {
            if (!frames.contains(page)) { // Page fault occurs
                if (frames.size() == capacity) frames.remove(0); // Remove LRU
                frames.add(page);
                faults++;
            } else {
                frames.remove((Integer) page); // Refresh LRU
                frames.add(page);
            }
        }
        System.out.println("LRU Page Faults: " + faults);
    }
}

public class LRUPageReplacement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of pages: ");
        int n = sc.nextInt();
        List<Integer> pages = new ArrayList<>();
        System.out.println("Enter page reference sequence:");
        for (int i = 0; i < n; i++) pages.add(sc.nextInt());
        System.out.print("Enter frame capacity: ");
        int capacity = sc.nextInt();

        new LRU(pages, capacity).execute();
        sc.close();
    }
}
```

8.Dynamic Link Library

(ArithmeticOperations.java)

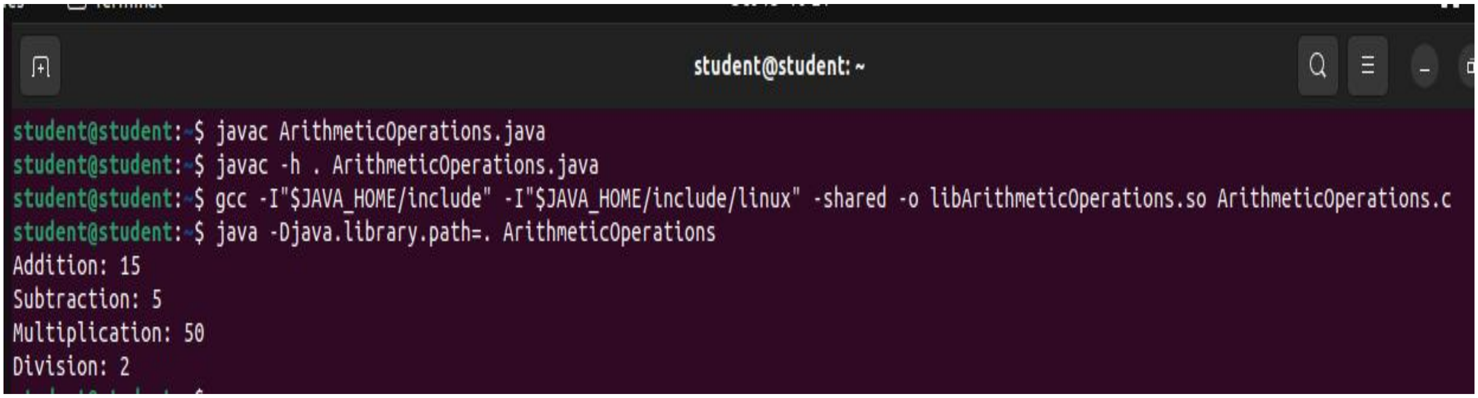
```
public class ArithmeticOperations {
    // Native methods for arithmetic operations
    public native int add(int a, int b);
    public native int subtract(int a, int b);
    public native int multiply(int a, int b);
    public native int divide(int a, int b);

    // Load the native library
    static {
        System.loadLibrary("ArithmeticOperations");
    }
    // Main method to test the arithmetic operations
    public static void main(String[] args) {
        ArithmeticOperations ops = new ArithmeticOperations()
        int a = 10;
        int b = 5;

        // Perform and display the results of the operations
        System.out.println("Addition: " + ops.add(a, b));
        System.out.println("Subtraction: " + ops.subtract(a, b));
        System.out.println("Multiplication: " + ops.multiply(a, b));
        System.out.println("Division: " + ops.divide(a, b));
    }
}
```

(ArithmeticOperations.c)

```
#include <jni.h>
#include "ArithmeticOperations.h"

// Function to add two integers
JNIEXPORT jint JNICALL Java_ArithmeticOperations_add(JNIEnv *env, jobject obj, jint a, jint b) {
    return a + b;  // Return the sum of a and b
}

// Function to subtract one integer from another
JNIEXPORT jint JNICALL Java_ArithmeticOperations_subtract(JNIEnv *env, jobject obj, jint a, jint b) {
    return a - b;  // Return the difference of a and b
}
```

```
// Function to multiply two integers

JNIEXPORT jint JNICALL Java_ArithmeticOperations_multiply(JNIEnv *env, jobject obj, jint a, jint b) {

    return a * b;  // Return the product of a and b

}

// Function to divide one integer by another

JNIEXPORT jint JNICALL Java_ArithmeticOperations_divide(JNIEnv *env, jobject obj, jint a, jint b) {

    if (b == 0) {

        return 0; // Handle division by zero (returns 0)

    }

    return a / b;  // Return the quotient of a and b

}
```

//Compilation steps



```
student@student:~$ javac ArithmeticOperations.java
student@student:~$ javac -h . ArithmeticOperations.java
student@student:~$ gcc -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -o libArithmeticOperations.so ArithmeticOperations.c
student@student:~$ java -Djava.library.path=. ArithmeticOperations
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
```

//IOT

1)Connectivity with IR sensor

```
//Define pin numbers

const int ledPin = 3;

const int irSensorPin = 2;


void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(irSensorPin, INPUT);
Serial.begin(9600);
}

void loop() {
  int sensorState = digitalRead(irSensorPin);

  // The sensor outputs LOW when it detects an obstacle
  if (sensorState == LOW) {
    digitalWrite(ledPin, HIGH);
    Serial.print("Object is detected\n");
  } else {
    digitalWrite(ledPin, LOW);
    Serial.print("Object is not detected\n");
  }

  delay(1000);  // Wait for 100 milliseconds
}
```

---

2. **Connectivity with temp sensor**

//DHT 11 IOT CODE

```
#include <DHT.h>
#define DHTPIN 2
#define DHTTYPE DHT11   // DHT 11 sensor
```

```
// Define LED pin
#define LED_PIN 13  // Use built-in LED on Arduino Uno (pin 13)

DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();

  pinMode(LED_PIN, OUTPUT);  // Initialize LED pin as an output
}

void loop() {
  delay(2000); // Wait for sensor to stabilize

  // Read humidity and temperature
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print humidity and temperature to Serial Monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");


  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");


  // Check if temperature exceeds 26°C
  if (temperature > 26.0) {
    digitalWrite(LED_PIN, HIGH);  // Turn on the LED
  } else {
    digitalWrite(LED_PIN, LOW);   // Turn off the LED
} }
```

3.Pi Camera

```
 from picamera import PiCamera
from PIL import Image,ImageDraw,ImageFont
from time import sleep
camera=Picamera()
filename=”/home/pi/Desktop/image1.png”
camera.start_preview()
sleep(5)
camera.capture(filename)
camera.stop_preview()
camera.close()
image=Image.open(filename)
draw=Image.Draw(image)
font=ImageFont.load_defaut()
image_width=100
image_height=image.size
x=700
```

```
y=700
text=” Hi Neha”
draw.text((x,y),text,font=font,fill=(255,255,255))
image.save(filename)
```

4.Dashboard to be deployed on cloud

************************ Publish DHT11 sensor data on thingspeak ******************

```
import RPi.GPIO as GPIO
import time
import requests
import Adafruit_DHT

# Sensor configuration
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 18  # GPIO pin where the DHT sensor is connected

# ThingSpeak configuration
API_KEY = "YOUR_WRITE_API_KEY"  # Replace with your Write API Key
THINGSPEAK_URL = "http://api.thingspeak.com/update"

# Setup GPIO
GPIO.setmode(GPIO.BCM)

try:
    while True:
        # Read humidity and temperature from DHT sensor
        humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)
        if humidity is not None and temperature is not None:
            print(f'Temperature: {temperature} °C, Humidity: {humidity} %')

            # Send data to ThingSpeak
            payload = {
                'api_key': API_KEY,
                'field1': temperature,
                'field2': humidity
            }
            response = requests.get(THINGSPEAK_URL, params=payload)
            print(f'ThingSpeak Response: {response.status_code}')
        else:
            print('Failed to retrieve data from temperature sensor')

        # Wait for 30 seconds before sending the next data
        time.sleep(30)

except KeyboardInterrupt:
    print("Program stopped by User")
finally:
    GPIO.cleanup()
```