Lab Report 1: Distributed Lag Model **Date:** May 05, 2025

Qn: WAP to find the growth in national consumption for five years using Distributed Lag Model provided.

$\boldsymbol{I = 2 + 0.1\, Y_{-1}}$
$\boldsymbol{Y = 45.45 + 2.27\,(I + G)}$
$\boldsymbol{T = 0.2\, Y}$
$\boldsymbol{C = 20 + 0.7\,(Y - T)}$

Assume the initial value of Y_{-1} is 80 and take the governmental expenditure in 5 years to be as follows:

Year	G
1	20
2	25
3	30
4	35
5	40

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    float Y_1, Y, I, T;
    float C[5];
    float G[5] = {20, 25, 30, 35, 40};

    cout << "Enter initial value of lagged variable Y_1: ";
    cin >> Y_1;

    cout << "\nThe growth in consumption is given following table:\n";
    cout << "\nYear \t\tConsumption\n";
    cout << "---------------------------\n";

    for (int i = 0; i < 5; ++i)
    {
        I = 2 + 0.1f * Y_1;
        Y = 45.45f + 2.27f * (I + G[i]);
        T = 0.2f * Y;
        C[i] = 20 + 0.7f * (Y - T);

        cout << setw(4) << i + 1 << "\t\t" << fixed << setprecision(2) << C[i] << endl;

        Y_1 = Y;
    }

    return 0;
}
```

Output:

```
  ...  Simulation  main  ?5 ~8 -5  09:45  0.047s  cd "c:\Users\firoj\OneDrive\
?) { g++ lab_work1.cpp -o lab_work1 } ; if ($?) { .\lab_work1 }
Enter initial value of lagged variable Y_1: 8

The growth in consumption is given following table:

Year            Consumption
---------------------------
   1            74.44
   2            92.13
   3            102.50
   4            111.21
   5            119.55
```

Qn: Customers arrive in a bank according to Poisson's process with mean inter arrival time of 10 minutes. Customers spend an average of 5 minutes on a single available counter, and leave.

WAP to find:

i. The probability that the customer will not have to wait at the counter.
ii. Expected number of customers in the bank.
iii. Time can a customer expect the spend in the bank.

Source Code:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

void printLine(int width) {
    cout << string(width, '=') << endl;
}

int main() {
    float inter_arrival_time, service_time;

    // Input section with header
    printLine(50);
    cout << "|          M/M/1 Queue Calculator                 |" << endl;
    printLine(50);
    cout << "Enter inter-arrival time (minutes): ";
    cin >> inter_arrival_time;
    cout << "Enter average service time (minutes): ";
    cin >> service_time;

    // Input validation
    if (inter_arrival_time <= 0 || service_time <= 0) {
        printLine(50);
        cout << "| Error: Times must be positive.              |" << endl;
        printLine(50);
        return 1;
    }

    // Calculate rates (customers per minute)
    float lambda = 1.0 / inter_arrival_time;
    float mu = 1.0 / service_time;

    // Check for queue stability
    if (mu <= lambda) {
        printLine(50);
        cout << " Error: Service rate must exceed arrival rate. " << endl;
        printLine(50);
        return 1;
    }

    // M/M/1 queue calculations
    float p0 = 1.0 - (lambda / mu);              // Probability of no waiting
    float L = lambda / (mu - lambda);            // Expected number in system
```

```
    float W = 1.0 / (mu - lambda);               // Expected time in system (minutes)

    // Output section
    printLine(50);
    cout << "|           Queue Analysis Results                  |" << endl;
    printLine(50);
    cout << fixed << setprecision(4);
    cout << "  Arrival rate (lambda): " << setw(16) << lambda << " cust/min  " << endl;
    cout << "  Service rate (mu): " << setw(19) << mu << " cust/min  " << endl;
    printLine(50);
    cout << " I.  Probability of no waiting: " << setw(12) << p0 << endl;
    cout << " II. Expected customers in bank: " << setw(11) << L << endl;
    cout << " III. Expected time in bank: " << setw(11) << W << " min " << endl;
    printLine(50);

    return 0;
}
```

Output:

```
 ... Simulation  main ?5 ~8 -5  10:04  3.883s  cd "c:\Users\firoj\OneDrive\
?) { g++ lab_work2.cpp -o lab_work2 } ; if ($?) { .\lab_work2 }
==================================================
|          M/M/1 Queue Calculator                |
==================================================
Enter inter-arrival time (minutes): 10
Enter average service time (minutes): 5
==================================================
|          Queue Analysis Results                |
==================================================
  Arrival rate (lambda):          0.1000 cust/min
  Service rate (mu):             0.2000 cust/min
==================================================
 I.  Probability of no waiting:       0.5000
 II. Expected customers in bank:      1.0000
 III. Expected time in bank:     10.0000 min
==================================================
```

Qn: At the ticket counter of football stadium, people come in queue and purchase tickets. Arrival rate of customers is 1/min. It takes an average of 20 seconds to purchase the ticket.

WAP to calculate the total time spent by sports fan to be seated on his seat if it takes 1.5 minutes to reach the correct seat after purchasing the ticket. if a fan comes exactly 2 minutes before the game starts, can sports fan expect to be seated for the kick-off?

Source Code:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>

using namespace std;

void printLine(int width) {
    cout << string(width, '=') << endl;
}

int main() {
    float arrival_rate, service_time, time_to_seat, time_before_game;

    // Input section with header
    printLine(50);
    cout << "|          Stadium Ticket Queue Calculator        |" << endl;
    printLine(50);
    cout << "Enter arrival rate (customers per minute): ";
    cin >> arrival_rate;
    cout << "Enter average service time (seconds): ";
    cin >> service_time;
    cout << "Enter time to reach seat (minutes): ";
    cin >> time_to_seat;
    cout << "Enter time before game starts (minutes): ";
    cin >> time_before_game;

    // Input validation
    if (arrival_rate <= 0 || service_time <= 0 || time_to_seat < 0 || time_before_game < 0)
{
        printLine(50);
        cout << "| Error: Inputs must be positive (or non-negative for times). |" << endl;
        printLine(50);
        return 1;
    }

    // Convert service time to minutes and calculate service rate
    float service_time_min = service_time / 60.0;
    float mu = 1.0 / service_time_min;

    // Check for queue stability
    if (mu <= arrival_rate) {
        printLine(50);
        cout << "| Error: Service rate must exceed arrival rate. |" << endl;
        printLine(50);
        return 1;
    }

    // M/M/1 queue calculations
```

```cpp
    float W = 1.0 / (mu - arrival_rate);              // Expected time in system (minutes)
    float total_time = W + time_to_seat;              // Total time to be seated
    float prob_on_time = 1.0 - exp(-(mu - arrival_rate) * (time_before_game -
time_to_seat)); // P(T <= time_before_game)

    // Output section
    printLine(50);
    cout << "|          Queue Analysis Results                |" << endl;
    printLine(50);
    cout << fixed << setprecision(4);
    cout << " Arrival rate (lambda): " << setw(16) << arrival_rate << " cust/min  " << endl;
    cout << " Service rate (mu): " << setw(19) << mu << " cust/min  " << endl;
    cout << " Expected time in queue system: " << setw(11) << W << " min " << endl;
    cout << " Total time to be seated: " << setw(13) << total_time << " min " << endl;
    printLine(50);
    cout << " Probability of being seated by kick-off: " << setw(7) << prob_on_time * 100 <<
"% " << endl;
    cout << " Can fan expect to be seated? " << setw(15) << (total_time <= time_before_game
? "Yes" : "No") << endl;
    printLine(50);

    return 0;
}
```

Output:

```
 ... Simulation  main ?5 ~8 -5  10:16  41.506s  cd "c:\Users\firoj\OneDrive\
$?) { g++ lab_work3.cpp -o lab_work3 } ; if ($?) { .\lab_work3 }
==================================================
|          Stadium Ticket Queue Calculator          |
==================================================
Enter arrival rate (customers per minute): 1
Enter average service time (seconds): 20
Enter time to reach seat (minutes): 1.5
Enter time before game starts (minutes): 2
==================================================
|          Queue Analysis Results                |
==================================================
 Arrival rate (lambda):            1.0000 cust/min
 Service rate (mu):               3.0000 cust/min
 Expected time in queue system:       0.5000 min
 Total time to be seated:         2.0000 min
==================================================
 Probability of being seated by kick-off: 63.2121%
 Can fan expect to be seated?              Yes
==================================================
```

Qn: In a single pump service station, vehicles arrive for fueling with an average of 5 minutes between arrivals. If an hour is taken as a unit of time, cars arrive according to Poison's process with an average of $\lambda = 12$ cars/hr.

WAP to generate Poisson distribution for: $x = 0, 1, 2, \dots, 15$.

$$f(x) = \Pr(X = x) = \frac{e^{-\lambda}\,\lambda^x}{x!} = \frac{e^{-12}\,12^x}{x!}, \begin{cases} x = 0, 1, 2, \dots \\ \lambda > 0 \end{cases}$$

Source Code:

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>

using namespace std;

void printLine(int width) {
    cout << string(width, '=') << endl;
}

// Function to calculate factorial
double factorial(int n) {
    double fact = 1.0;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

int main() {
    float lambda = 12.0; // This is the arrival rate of cars per hr
    double pr; // Probability for each x
    // Header
    printLine(50);
    cout << "|      Poisson Distribution (lambda = " << fixed << setprecision(1) << lambda << ")|\n";
    printLine(50);
    // Validate lambda
    if (lambda <= 0) {
        printLine(50);
        cout << "| Error: Arrival rate must be positive.          |\n";
        printLine(50);
        return 1;
    }
    // Calculate and print Poisson probabilities for x = 0 to 15
    cout << "| " << setw(5) << "x" << " | " << setw(15) << "P(X = x)" << setw(27)<< " |\n";
    printLine(50);
    for (int x = 0; x <= 15; x++) {
        pr = (exp(-lambda) * pow(lambda, x)) / factorial(x);
        cout << "| " << setw(5) << x << " | " << fixed << setprecision(6) << setw(15) << pr <<
setw(27)<< " |\n";
    }
    printLine(50);

    return 0;
}
```

Output:

```
 ⊞ 〉⌂ 〉 ... 〉 Simulation 〉 main ?5 ~8 -5 〉 10:33 〉 0.684s 〉 cd "c:\Users\firoj\OneDrive\
?) { g++ lab_work4.cpp -o lab_work4 } ; if ($?) { .\lab_work4 }
==================================================
|     Poisson Distribution (lambda = 12.0)       |
==================================================
|      x |         P(X = x)                      |
==================================================
|      0 |         0.000006                      |
|      1 |         0.000074                      |
|      2 |         0.000442                      |
|      3 |         0.001770                      |
|      4 |         0.005309                      |
|      5 |         0.012741                      |
|      6 |         0.025481                      |
|      7 |         0.043682                      |
|      8 |         0.065523                      |
|      9 |         0.087364                      |
|     10 |         0.104837                      |
|     11 |         0.114368                      |
|     12 |         0.114368                      |
|     13 |         0.105570                      |
|     14 |         0.090489                      |
|     15 |         0.072391                      |
==================================================
```

Qn: The probabilities of weather conditions (modeled as either rainy or sunny), given the weather on the preceding day, can be represented by a transition matrix:

$$\begin{bmatrix} \mathbf{0.9} & \mathbf{0.1} \\ \mathbf{0.5} & \mathbf{0.5} \end{bmatrix}$$

The weather on day 0 is known to be sunny. This is represented by a vector in which the "sunny" entry is 100%, and the "rainy" entry is 0%: **(1 0)**.

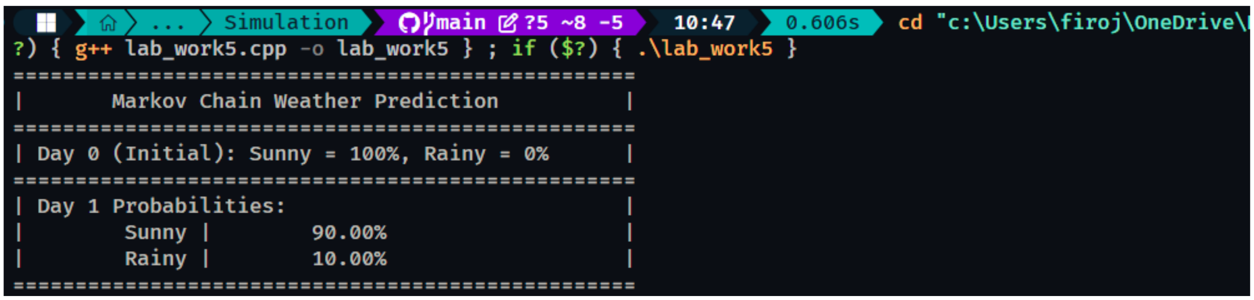
Write a program to find the weather of the next day by using Markov Chain Method.

Source Code:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>

using namespace std;

void printLine(int width) {
    cout << string(width, '=') << endl;
}

// Function to validate transition matrix (rows sum to 1)
bool isValidTransitionMatrix(float mat[2][2]) {
    const float epsilon = 1e-6; // Tolerance for floating-point comparison
    for (int i = 0; i < 2; i++) {
        float rowSum = mat[i][0] + mat[i][1];
        if (abs(rowSum - 1.0) > epsilon) {
            return false;
        }
    }
    return true;
}

int main() {
    // Initialize transition matrix and initial state vector
    float transMat[2][2] = {{0.9, 0.1}, {0.5, 0.5}}; // [sunny->sunny, sunny->rainy; rainy-
>sunny, rainy->rainy]
    float initialState[2] = {1.0, 0.0}; // 100% sunny, 0% rainy
    float result[2] = {0.0, 0.0}; // Result for Day 1

    // Validate transition matrix
    if (!isValidTransitionMatrix(transMat)) {
        printLine(50);
        cout << "| Error: Transition matrix rows must sum to 1. |\n";
        printLine(50);
        return 1;
    }

    // Matrix multiplication: result = initialState * transMat
    for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 2; k++) {
            result[j] += initialState[k] * transMat[k][j];
        }
    }
```

```
    // Output
    printLine(50);
    cout << "|        Markov Chain Weather Prediction          |\n";
    printLine(50);
    cout << "| Day 0 (Initial): Sunny = 100%, Rainy = 0%       |\n";
    printLine(50);
    cout << "| Day 1 Probabilities:                            |\n";
    cout << "| " << setw(12) << "Sunny" << " | " << setw(12) << fixed << setprecision(2) <<
result[0] * 100 << "%"<< setw(21)<< " |\n";
    cout << "| " << setw(12) << "Rainy" << " | " << setw(12) << result[1] * 100 << "%"<<
setw(21)<< " |\n";
    printLine(50);

    return 0;
}
```

Output:



```
 ... Simulation  main ?5 ~8 -5  10:47  0.606s  cd "c:\Users\firoj\OneDrive\
?) { g++ lab_work5.cpp -o lab_work5 } ; if ($?) { .\lab_work5 }
==================================================
|       Markov Chain Weather Prediction          |
==================================================
| Day 0 (Initial): Sunny = 100%, Rainy = 0%      |
==================================================
| Day 1 Probabilities:                           |
|        Sunny |        90.00%                   |
|        Rainy |        10.00%                   |
==================================================
```

Lab Report 6: (A) **Date:** May 05, 2025

Qn: What is a random number and what are its properties. WAP in C to generate 100 random numbers using Linear Congruential Method where X_0**=11, m=100, a = 5 and c = 13**.

Initial Question Answer:

A random number is a number generated in such a way that it is unpredictable and does not follow a pattern. It is typically used in simulations, cryptography, games, statistics, and randomized algorithms.

Properties:

1. **Unpredictability**
   - The next number in the sequence cannot be predicted from previous numbers.
2. **Uniform Distribution (for ideal random numbers)**
   - Each number within a given range has an **equal probability** of occurring.
3. **Independence**
   - The occurrence of one number **does not influence** the occurrence of another.
4. **Scalability**
   - Can be generated over any range or scale (e.g., 0–1, 1–100, etc.).
5. **Reproducibility (for pseudo-random numbers)**
   - Using the same **seed** will produce the same sequence.
6. **Statistical Randomness**
   - Over large samples, the numbers pass various statistical tests (e.g., mean, variance).
7. **Non-deterministic Nature (for true random numbers)**
   - Derived from unpredictable physical phenomena like radioactive decay or atmospheric noise.

Source Code:

```
#include <iostream>
#include <iomanip>
#include <array>
#include <string>

using namespace std;

void printLine(int width) {
    cout << string(width, '=') << endl;
}

// Validate LCM parameters
bool isValidLCMParameters(int m, int a, int c, int x0) {
    if (m <= 0) return false;
    if (a <= 0) return false;
    if (c < 0) return false;  // Increment must be non-negative
    if (x0 < 0 || x0 >= m) return false; // Seed must be in [0, m-1]
    return true;
}
```

```cpp
int main() {
    const int SIZE = 100;
    array<int, SIZE> x;
    int m = 100, a = 5, c = 13, x0 = 11;

    // Validate parameters before generation
    if (!isValidLCMParameters(m, a, c, x0)) {
        printLine(50);
        cout << "| Error: Invalid LCM parameters.                |\n";
        printLine(50);
        return 1;
    }

    // Initialize the first number with the seed
    x[0] = x0;

    // Generate the sequence using the LCM formula
    for (int i = 0; i < SIZE - 1; i++) {
        x[i + 1] = (a * x[i] + c) % m;
    }

    printLine(50);
    cout << "| Linear Congruential Method Random Numbers   |\n";
    cout << "| Parameters: X0=" << x0 << ", m=" << m << ", a=" << a << ", c=" << c << "
|\n";
    printLine(50);
    cout << "| Generated Numbers (100):                  |\n";
    printLine(50);
    for (int i = 0; i < SIZE; i++) {
        cout << setw(4) << x[i];
        if (i % 10 == 9) cout << "\n";
        else cout << " ";
    }
    if (SIZE % 10 != 0) cout << "\n";
    printLine(50);

    return 0;
}
```

Output:

```
 ... Simulation  main ?5 ~8 -5  11:02  0.627s  cd "c:\Users\firoj\OneDrive\
?) { g++ lab_work6_a.cpp -o lab_work6_a } ; if ($?) { .\lab_work6_a }
==================================================
| Linear Congruential Method Random Numbers      |
| Parameters: X0=11, m=100, a=5, c=13            |
==================================================
| Generated Numbers (100):                       |
==================================================
  11   68   53   78    3   28   53   78    3   28
  53   78    3   28   53   78    3   28   53   78
   3   28   53   78    3   28   53   78    3   28
  53   78    3   28   53   78    3   28   53   78
   3   28   53   78    3   28   53   78    3   28
  53   78    3   28   53   78    3   28   53   78
   3   28   53   78    3   28   53   78    3   28
  53   78    3   28   53   78    3   28   53   78
   3   28   53   78    3   28   53   78    3   28
  53   78    3   28   53   78    3   28   53   78
==================================================
```

Qn: WAP to generate 100 random numbers using Multiplicative Congruential Method where:

$$X_0=13, m=1000, a=15 \text{ and } c=7$$

Source Code:

```
#include <iostream>
#include <iomanip>
#include <array>
#include <string>

using namespace std;

void printLine(int width) {
    cout << string(width, '=') << endl;
}

// Function to validate LCM parameters
bool isValidLCMParameters(int m, int a, int c, int x0) {
    if (m <= 0) return false;
    if (a <= 0) return false;
    if (c < 0) return false;
    if (x0 < 0 || x0 >= m) return false; // Seed must be in [0, m-1]
    return true;
}

int main() {
    const int SIZE = 100;
    array<int, SIZE> x; // Fixed-size array for random numbers
    int m = 1000, a = 15, c = 7, x0 = 13; // LCM parameters

    // Validate parameters
    if (!isValidLCMParameters(m, a, c, x0)) {
        printLine(50);
        cout << "| Error: Invalid LCM parameters.               |\n";
        printLine(50);
        return 1;
    }

    // Initialize the first number with the seed
    x[0] = x0;

    // Generate random numbers using the Linear Congruential Method (LCM)
    for (int i = 0; i < SIZE - 1; i++) {
        x[i + 1] = (a * x[i] + c) % m;
    }

    printLine(50);
    cout << "| Multiplicative Congruential Random Numbers     |\n";
    cout << "| Parameters: X0=" << x0 << ", m=" << m << ", a=" << a << ", c=" << c <<
setw(13)<<" |\n";
    printLine(50);
    cout << "| Generated Numbers (100):"<<setw(25)<<"|\n";
    printLine(50);
    for (int i = 0; i < SIZE; i++) {
        cout << setw(5) << x[i];
        if (i % 10 == 9) cout << "\n";
        else cout << " ";
    }
    if (SIZE % 10 != 0) cout << "\n";
```

```
    printLine(50);

    return 0;
}
```

Output:

```
 ⌂  ...  Simulation   main  ?5 ~8 -5  11:09  0.616s  cd "c:\Users\firoj\OneDrive\
?) { g++ lab_work6_b.cpp -o lab_work6_b } ; if ($?) { .\lab_work6_b }
==================================================
| Multiplicative Congruential Random Numbers     |
| Parameters: X0=13, m=1000, a=15, c=7           |
==================================================
| Generated Numbers (100):                       |
==================================================
   13   202    37   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
  437   562   437   562   437   562   437   562   437   562
==================================================
```

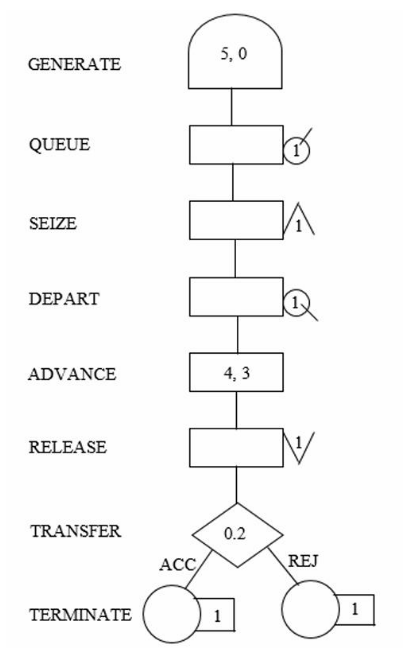
Qn: Why is GPSS called transaction flow-oriented language? A machine tool in a manufacturing shop is turning out parts at the rate of every 5minutes. When they are finished, the parts are sent to an inspect or, who takes 4 ± 3 minutes to examine each one and rejects 20% of the parts.

Draw and explain a block diagram for it and write a GPSS program to simulate using the concept of FA CILITY.

Solution:

GPSS (General Purpose Simulation System) is a highly structured and special purpose simulation language based on process interaction approach and oriented toward queuing systems. The system being simulated is described by the block diagram using various GPSS blocks. Each block represents events, delays or other actions that affect the transaction flow. Therefore, GPSS is called transaction flow-oriented language.

The block diagram for the given problem using GPSS is given below:



A GENERATE block is used to represent the output of the machine by creating one transaction every fiv e units of time. A QUEUE block places the transaction in the queue and SEIZE block allows a transactio n to engage a facility if it is available. If more than one inspector is available, the transaction leaves the q ueue which is denoted by DEPART block and enters ADVANCE block. An ADVANCE block with a me ans of 4 and modifier of 3 is used to represent inspection. The time spent on inspection will therefore be any one of the values 1,2, 3, 4, 5, 6 or 7, with equal probability given to each value. Upon completion of the inspection, RELEASE block allows a transaction to disengage the facility and transaction go to a TR

ANSFER block with a selection factor of 0.2, so that 80 % of the parts go to the next location called AC C, to represent accepted parts and 20 % go to another location called REJ to represent rejects. Both locat ions reached from the TRANSFER block are TERMINATE blocks.

Code:

```
GENERATE 5.0
QUEUE 1
SEIZE 1
DEPART 1
ADVANCE 4.0 3.0
RELEASE 1
TRANSFER 0.2 ACC REJ
ACC TERMINATE 1
REJ TERMINATE 1
```

Report:

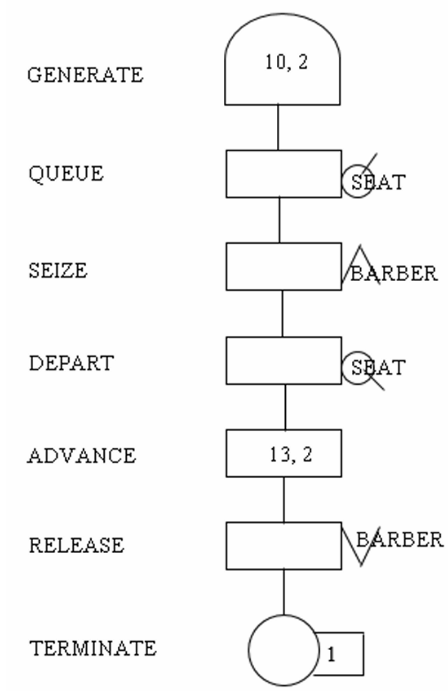
```
              GPSS World Simulation Report - lab7.5.1


                  Wednesday, June 11, 2025 15:18:43

           START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
                0.000            505.946     9         1          0


              NAME                       VALUE
          ACC                             8.000
          REJ                             9.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE           101             0       0
                    2    QUEUE              101             0       0
                    3    SEIZE              101             1       0
                    4    DEPART             100             0       0
                    5    ADVANCE            100             0       0
                    6    RELEASE            100             0       0
                    7    TRANSFER           100             0       0
ACC                 8    TERMINATE           74             0       0
REJ                 9    TERMINATE           26             0       0


FACILITY         ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 1                  101    0.755      3.784  1       101    0    0     0      0


QUEUE              MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 1                   1    1    101     61     0.106      0.532      1.342   0


CEC XN   PRI          M1      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
   101    0        505.000    101      3       4


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
   102    0        510.000    102      0       1
```

Lab Report 8 **Date:** June 11, 2025

Qn: Create a GPSS model and program to simulate a barber shop for a day (9am to 4pm), where a customer enters the Shop every 10±2 minutes and a barber takes 13±2 for a haircut.

Solution:

GPSS Model to simulate barber shop:



Code:

```
GENERATE 10,2
QUEUE SEAT
SEIZE BARBER
DEPART SEAT
ADVANCE 15,3
RELEASE BARBER
TERMINATE

TIMER GENERATE 420
TERMINATE 1
```

Generated Simulation Report:

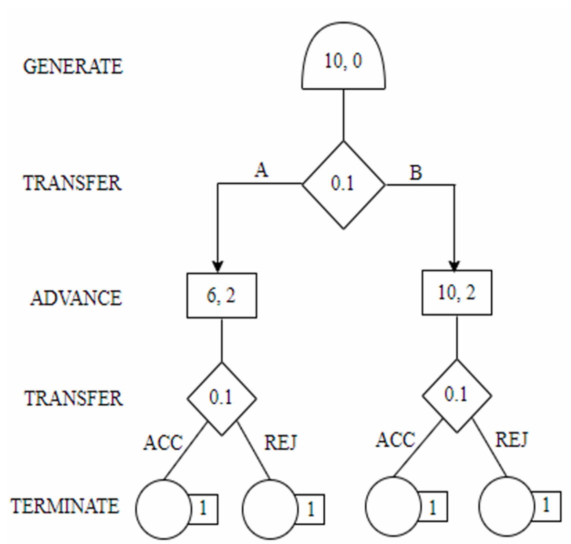
```
                GPSS World Simulation Report - lab8.1.1


                     Wednesday, June 11, 2025 15:35:18

             START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
                  0.000         420000.000     9        1          0


                NAME                       VALUE
            BARBER                      10001.000
            SEAT                        10000.000
            TIMER                           8.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE         42043             0       0
                    2    QUEUE            42043         14026       0
                    3    SEIZE            28017             0       0
                    4    DEPART           28017             0       0
                    5    ADVANCE          28017             1       0
                    6    RELEASE          28016             0       0
                    7    TERMINATE        28016             0       0
TIMER               8    GENERATE          1000             0       0
                    9    TERMINATE         1000             0       0


FACILITY         ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 BARBER           28017    1.000      14.990  1    28684    0    0     0   14026


QUEUE              MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 SEAT             14026 14026  42043      1  7019.115  70119.366  70121.034   0


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
 28684    0      420000.571   28684      5      6
 43044    0      420001.108   43044      0      1
 43045    0      420420.000   43045      0      8
```

Lab Report 9 **Date:** June 11, 2025

Qn: Parts are being made at the rate of one every 10 minutes. They are of two types, A and B. And are mixed randomly with about 10% being type B. A separate inspector is assigned to examine each part. Inspection of part A takes 6+2 minutes while B takes 10+2 minutes. Both inspectors reject 10% of the parts they inspect. Draw GPSS block diagram to simulate the above problem for 100 parts.

Solution:

The block diagram for given problem using GPSS:



Code:

```
GENERATE 10.0 0.0
TRANSFER 0.1 A B

A ADVANCE 12.0 6.0
TRANSFER 0.1 ACC REJ

B ADVANCE 10.0 2.0
TRANSFER 0.1 ACC REJ

ACC TERMINATE 1
REJ TERMINATE 1
```

Simulation Report:

```
              GPSS World Simulation Report - lab9.7.1


                   Wednesday, June 11, 2025 15:57:58

           START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
                0.000          10011.608     8         0          0


              NAME                       VALUE
          A                               3.000
          ACC                             7.000
          B                               5.000
          REJ                             8.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE          1001             0       0
                    2    TRANSFER          1001             0       0
A                   3    ADVANCE            866             1       0
                    4    TRANSFER           865             0       0
B                   5    ADVANCE            135             0       0
                    6    TRANSFER           135             0       0
ACC                 7    TERMINATE          895             0       0
REJ                 8    TERMINATE          105             0       0


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
  1002    0       10020.000   1002      0      1
  1001    0       10022.568   1001      3      4
```