**Lab 6**

**1.WAP to generate 50 random numbers using Mixed Congruential Method where X0=11, m=100, a = 5 and c = 13.**

**Code:**

#include <stdio.h>

int main() {

// Given parameters

int X = 11; // Seed value

int a = 5; // Multiplier

int c = 13; // Increment

int m = 100; // Modulus

int i;

// Generate 50 random numbers

printf("Generated random numbers:\n");

for(i = 0; i < 50; i++) {

X = (a \* X + c) % m; // LCG formula

printf("%d ", X);

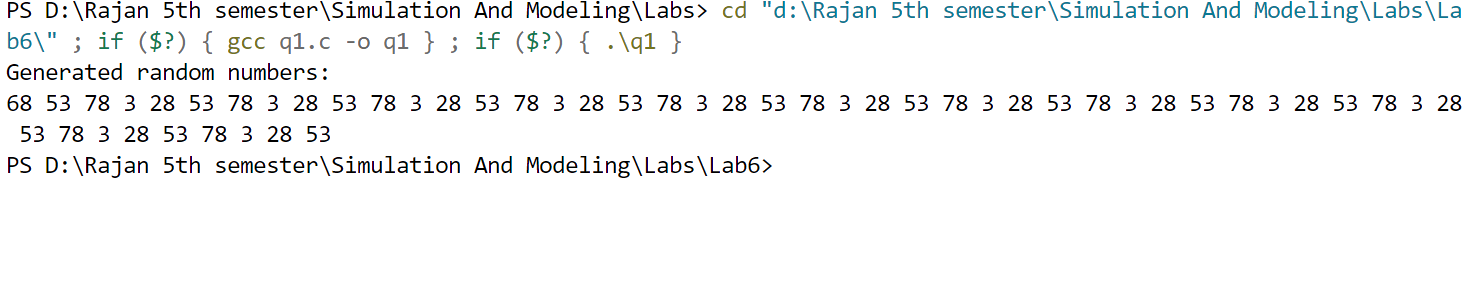
}

printf("\n");

return 0;

}

**Output:**

****

**2.WAP to generate 50 random numbers using Multiplicative Congruential Method where X0=13, m =1000, a = 15 and c = 7.**

**Code:**

#include <stdio.h>

int main() {

// Given parameters

int X = 13; // Seed value

int a = 15; // Multiplier

int m = 1000; // Modulus

int i;

// Generate 50 random numbers

printf("Generated random numbers:\n");

for(i = 0; i < 50; i++) {

X = (a \* X) % m; // Multiplicative Congruential Formula

printf("%d ", X);

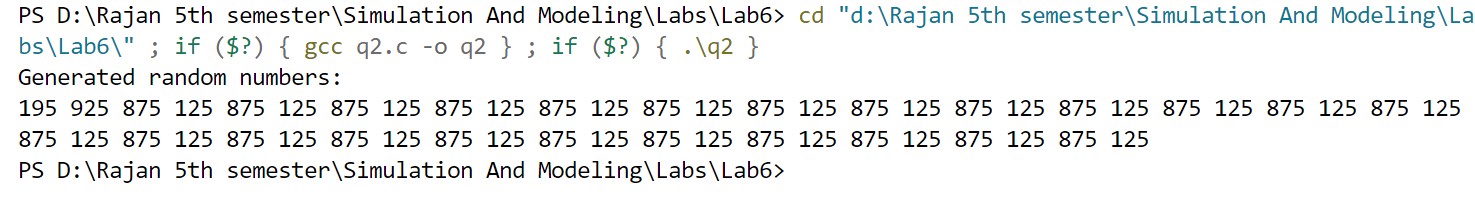
}

printf("\n");

return 0;

}

**Output:**

****

**3.WAP to implement Kolmogorov – Smirnov test.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 10 // Number of sample points

int main() {

double sample[N] = {0.12, 0.34, 0.56, 0.78, 0.90, 0.23, 0.45, 0.67, 0.89, 0.99};

double D = 0.0, D\_plus, D\_minus;

int i, j;

// Sorting the sample using Bubble Sort

for (i = 0; i < N - 1; i++) {

for (j = 0; j < N - i - 1; j++) {

if (sample[j] > sample[j + 1]) {

double temp = sample[j];

sample[j] = sample[j + 1];

sample[j + 1] = temp;

}

}

}

// Compute the maximum difference (D-statistic)

for (i = 0; i < N; i++) {

double Fn = (i + 1) / (double)N; // Empirical CDF

double F\_theoretical = sample[i]; // Expected CDF (Uniform[0,1])

D\_plus = fabs(Fn - F\_theoretical);

D\_minus = fabs(F\_theoretical - i / (double)N);

if (D\_plus > D) D = D\_plus;

if (D\_minus > D) D = D\_minus;

}

// Print the results

printf("Kolmogorov-Smirnov D-statistic: %lf\n", D);

// Compute the critical value (for alpha = 0.05, N = 10)

double D\_critical = 1.36 / sqrt(N);

// Decision Rule

if (D > D\_critical) {

printf("Reject the null hypothesis (Sample does not follow uniform distribution)\n");

} else {

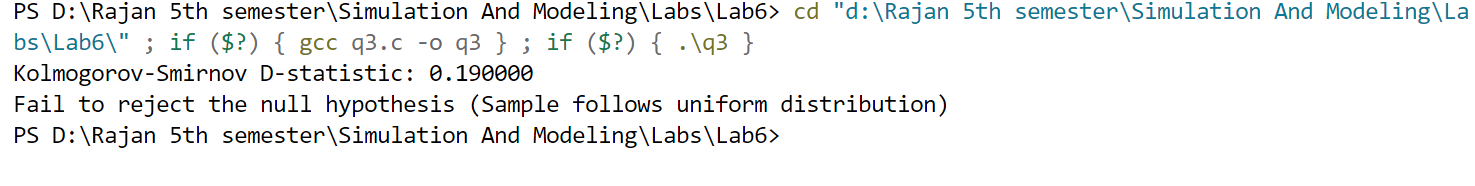
printf("Fail to reject the null hypothesis (Sample follows uniform distribution)\n");

}

return 0;

}

**Output:**

****

**Lab 7**

**1.WAP to implement auto correlation test.**

**code:**

#include <stdio.h>

#define N 10 // Number of random numbers

int main() {

double random\_numbers[N] = {0.23, 0.11, 0.87, 0.54, 0.33, 0.66, 0.77, 0.12, 0.98, 0.45};

int d = 2; // Lag distance

int i, count = 0;

double sum = 0.0, mean, variance, R;

// Compute mean

for (i = 0; i < N; i++) {

sum += random\_numbers[i];

}

mean = sum / N;

// Compute Auto-correlation coefficient

sum = 0.0;

for (i = 0; i < N - d; i++) {

sum += (random\_numbers[i] - mean) \* (random\_numbers[i + d] - mean);

}

// Compute Variance

double var\_sum = 0.0;

for (i = 0; i < N; i++) {

var\_sum += (random\_numbers[i] - mean) \* (random\_numbers[i] - mean);

}

variance = var\_sum / N;

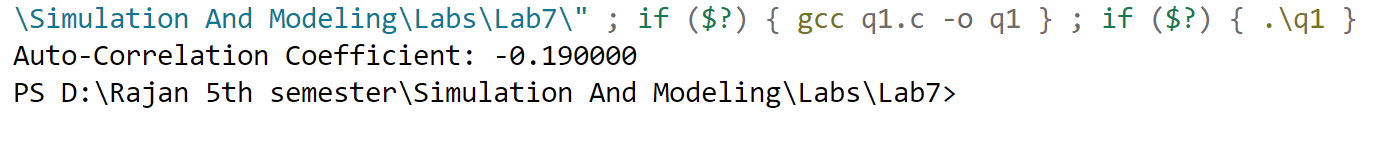
R = sum / ((N - d) \* variance);

printf("Auto-Correlation Coefficient: %lf\n", R);

return 0;

}

**Output:**

****

**2.WAP to implement Poker test.**

**Code:**

#include <stdio.h>

#define N 10 // Number of random numbers

int main() {

int random\_numbers[N] = {123, 456, 789, 222, 333, 777, 888, 999, 111, 555};

int i, j, k, count[10] = {0};

int frequency[4] = {0}; // 0: All different, 1: One pair, 2: Three of a kind

// Count occurrences of digits

for (i = 0; i < N; i++) {

int digits[10] = {0};

int num = random\_numbers[i];

// Count digit occurrences

while (num > 0) {

digits[num % 10]++;

num /= 10;

}

// Classify numbers based on repetition

int pairs = 0, triples = 0;

for (j = 0; j < 10; j++) {

if (digits[j] == 2) pairs++;

if (digits[j] == 3) triples++;

}

if (triples == 1) frequency[2]++; // Three of a kind

else if (pairs == 1) frequency[1]++; // One pair

else frequency[0]++; // All different

}

// Print Results

printf("Poker Test Results:\n");

printf("All different: %d times\n", frequency[0]);

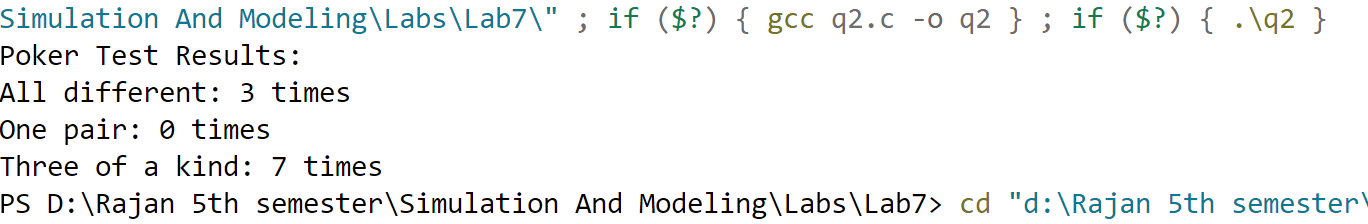
printf("One pair: %d times\n", frequency[1]);

printf("Three of a kind: %d times\n", frequency[2]);

return 0;

}

**Output:**

****

**3.WAP to implement chi-square test.**

**Code:**

#include <stdio.h>

#include <math.h>

#define N 10 // Number of random numbers

#define K 5 // Number of bins

int main() {

double random\_numbers[N] = {0.12, 0.34, 0.56, 0.78, 0.90, 0.23, 0.45, 0.67, 0.89, 0.99};

int observed[K] = {0};

int i, bin;

double expected = N / (double)K; // Expected frequency per bin

double chi\_square = 0.0;

// Count observed frequencies

for (i = 0; i < N; i++) {

bin = (int)(random\_numbers[i] \* K); // Assign bin (0 to K-1)

observed[bin]++;

}

// Compute Chi-Square Statistic

for (i = 0; i < K; i++) {

chi\_square += ((observed[i] - expected) \* (observed[i] - expected)) / expected;

}

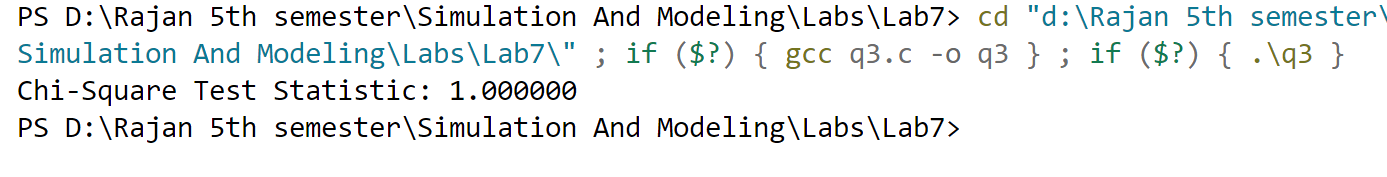
// Print Results

printf("Chi-Square Test Statistic: %lf\n", chi\_square);

return 0;

}

**Output:**

****

**Lab 8**

**1.Write a program to determine point estimation and its bias for a sample of data with a given population mean.**

**Code:**

#include <stdio.h>

int main() {

// Sample data

float sample[] = {52, 55, 60, 49, 58};

int n = sizeof(sample) / sizeof(sample[0]);

// Given population mean

float populationMean = 55.0;

// Calculate sample mean (point estimate)

float sum = 0.0;

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

sum += sample[i];

}

float sampleMean = sum / n;

// Calculate bias

float bias = sampleMean - populationMean;

// Output results

printf("Sample Data: ");

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

printf("%.1f ", sample[i]);

}

printf("\nSample Mean (Point Estimate): %.2f\n", sampleMean);

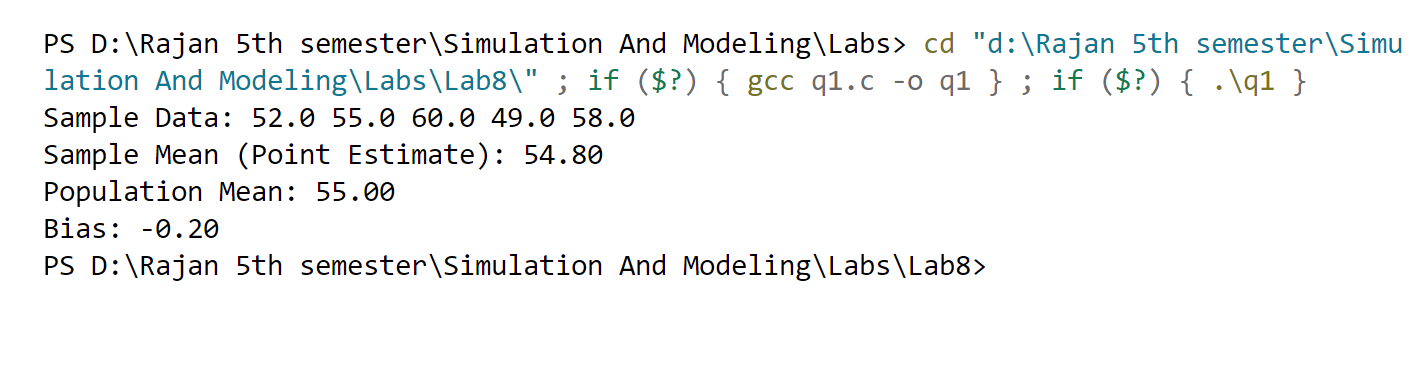
printf("Population Mean: %.2f\n", populationMean);

printf("Bias: %.2f\n", bias);

return 0;

}

**Output:**

****

**2.Write a program to determine interval/ confidence interval estimation for a sample of data with a given population mean.**

**Code:**

#include <stdio.h>

#include <math.h>

int main() {

// Sample data

float sample[] = {52, 55, 60, 49, 58};

int n = sizeof(sample) / sizeof(sample[0]);

// Confidence level: 95% => Z-score = 1.96

float z = 1.96;

// Calculate mean

float sum = 0.0;

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

sum += sample[i];

}

float mean = sum / n;

// Calculate standard deviation

float variance\_sum = 0.0;

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

variance\_sum += pow(sample[i] - mean, 2);

}

float stddev = sqrt(variance\_sum / (n - 1)); // Sample standard deviation

// Calculate margin of error

float margin = z \* (stddev / sqrt(n));

// Calculate confidence interval

float lower = mean - margin;

float upper = mean + margin;

// Output results

printf("Sample Mean: %.2f\n", mean);

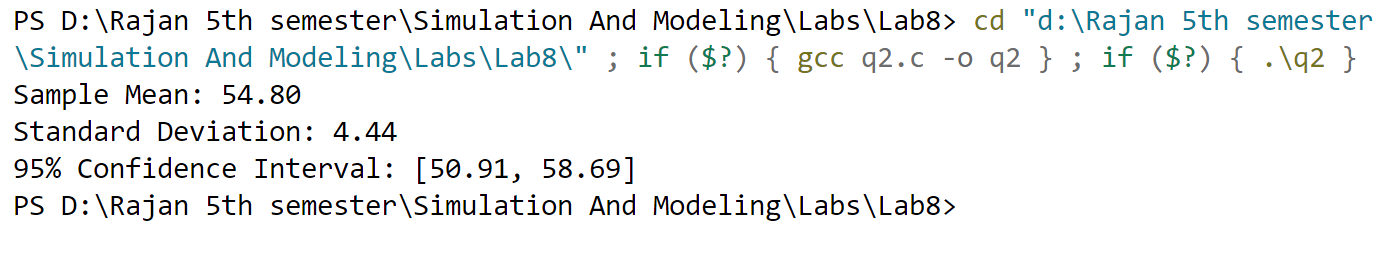
printf("Standard Deviation: %.2f\n", stddev);

printf("95%% Confidence Interval: [%.2f, %.2f]\n", lower, upper);

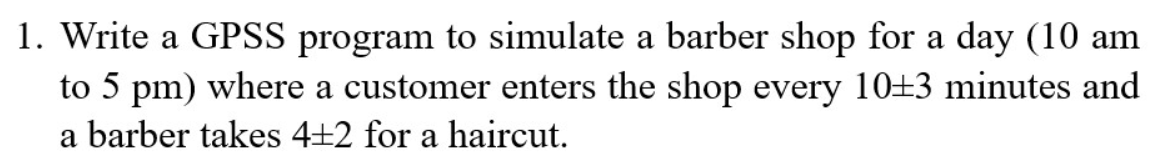
return 0;

}

**Output:**

****

**Lab 9**

****

**Code:**

GENERATE 600,180

SEIZE BARBER

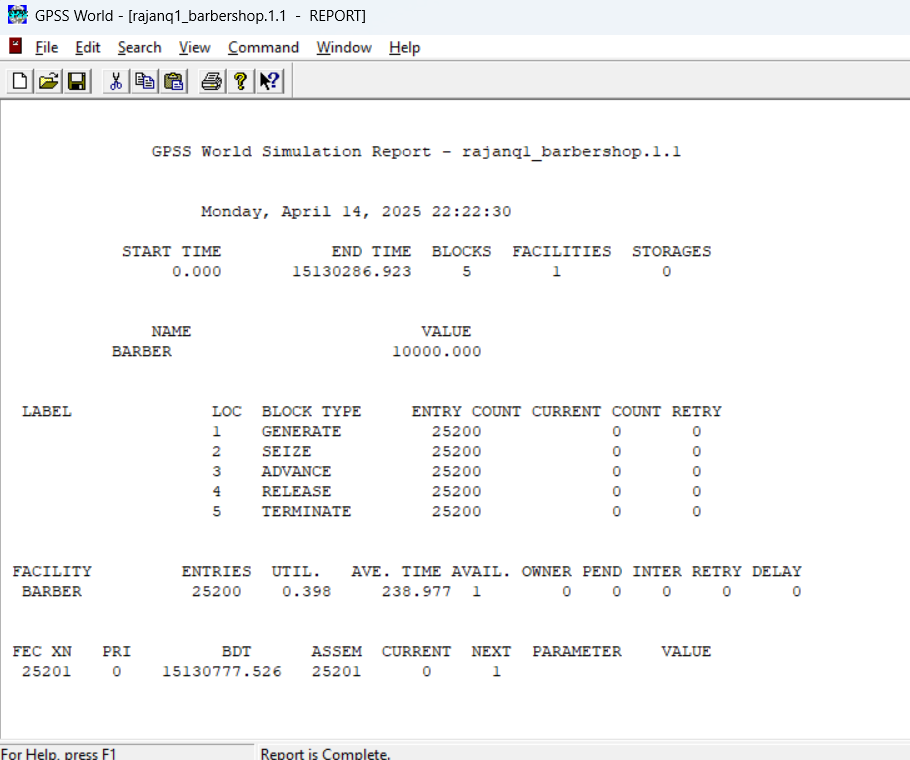
ADVANCE 240,120

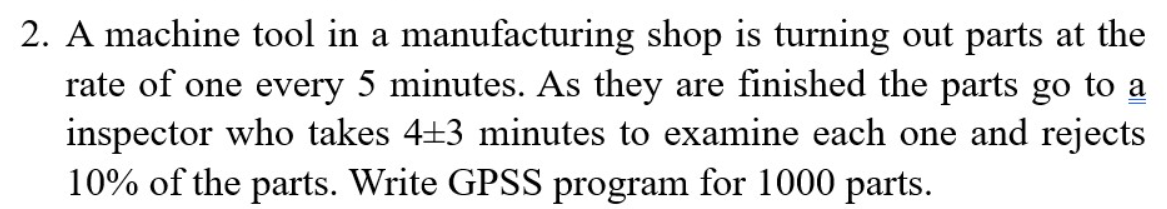
RELEASE BARBER

TERMINATE 1

START 25200

Output:



****

**Code:**

SIMULATE

GENERATE 300

ADVANCE 240,180

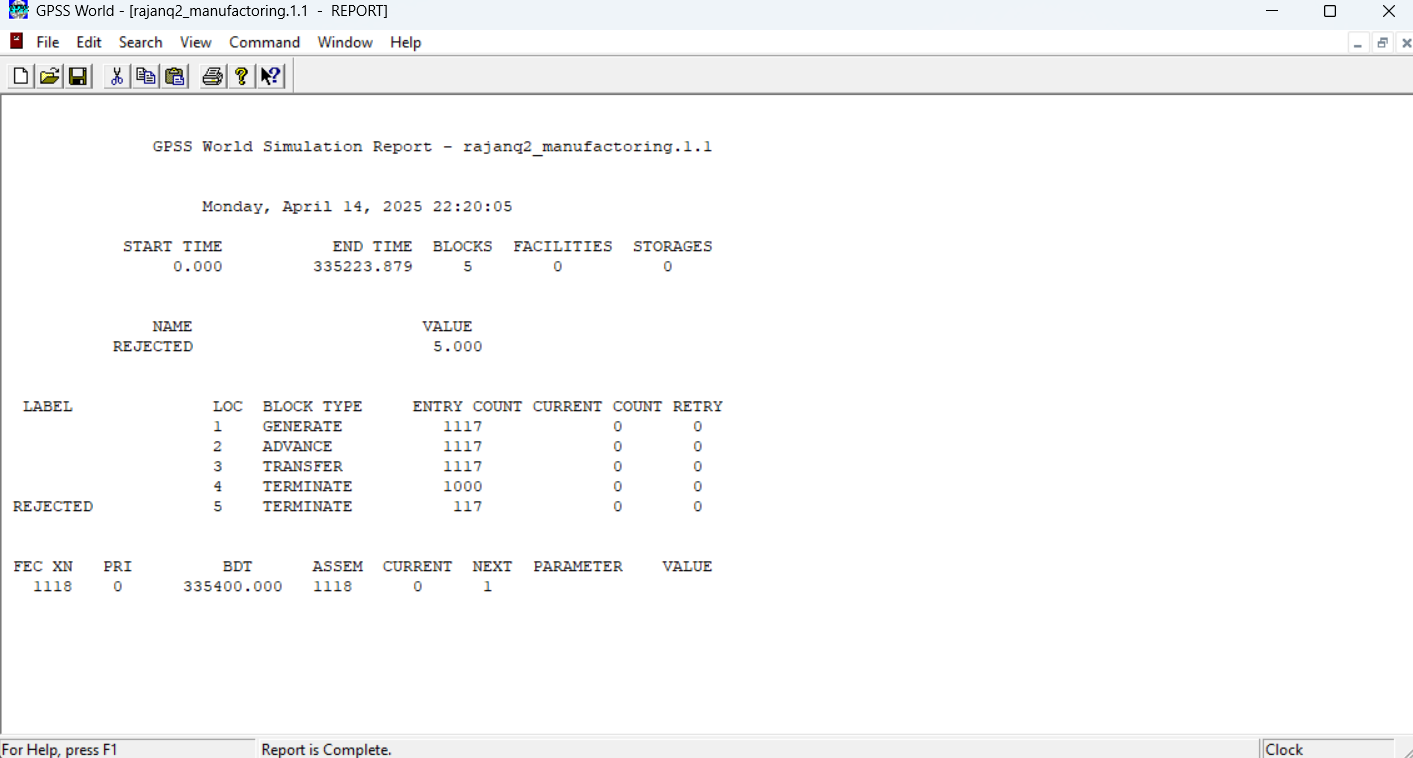
TRANSFER .1, REJECTED

TERMINATE 1

REJECTED TERMINATE 0

START 1000

**Output:**

****

**3.Implement q2 using facility.**

**Code:**

GENERATE 5,,,1000

SEIZE INSPECTOR

ADVANCE 4,3

RELEASE INSPECTOR

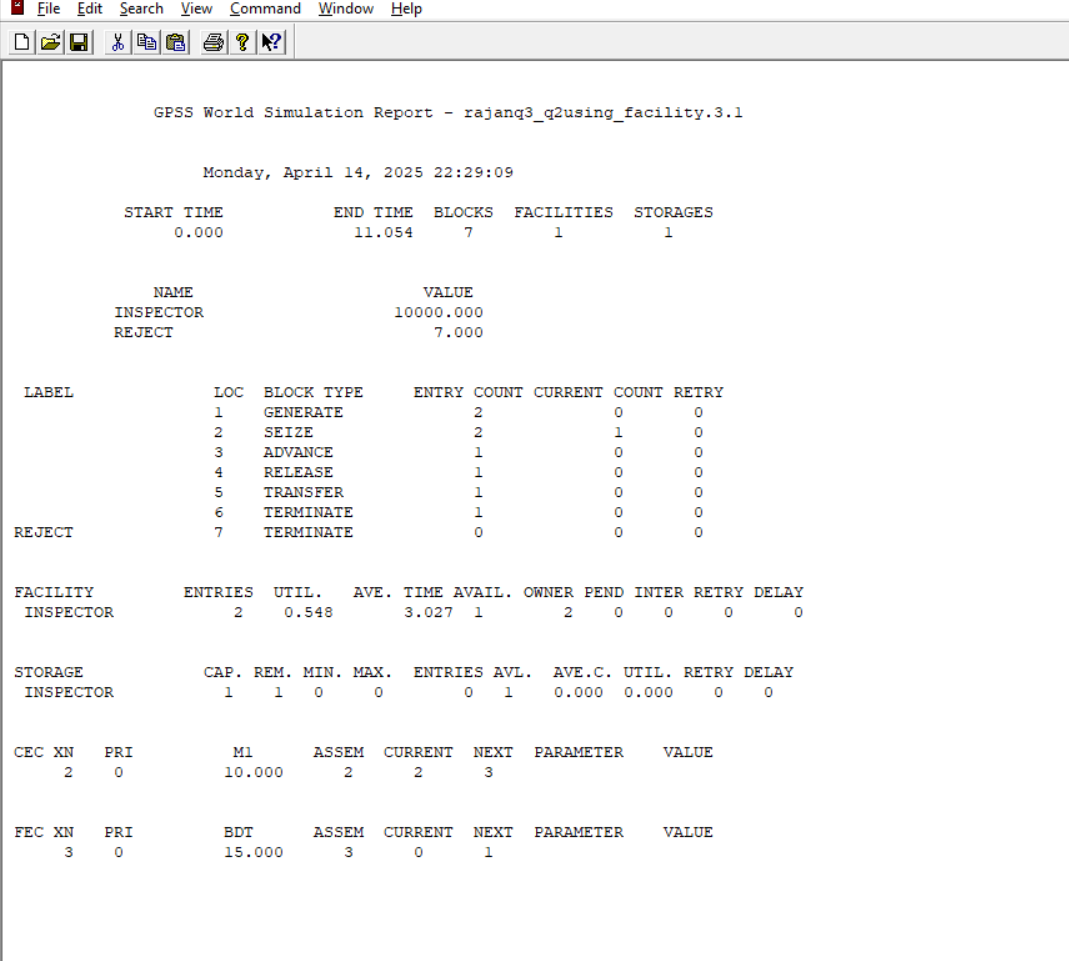
TRANSFER .1, REJECT

TERMINATE 1

REJECT TERMINATE 1

INSPECTOR STORAGE 1

**Output:**

****

**4.A machine tool in a manufacturing shop is turning out parts at the rate of every 5 minutes. When they are finished, the parts are sent to an inspector, who takes 4±3 minutes to examine each one and rejects 15% of the parts. Write a GPSS program to simulate using the concept of facility.**

**Code:**

GENERATE 5,,,1000

SEIZE INSPECTOR

ADVANCE 4,3

RELEASE INSPECTOR

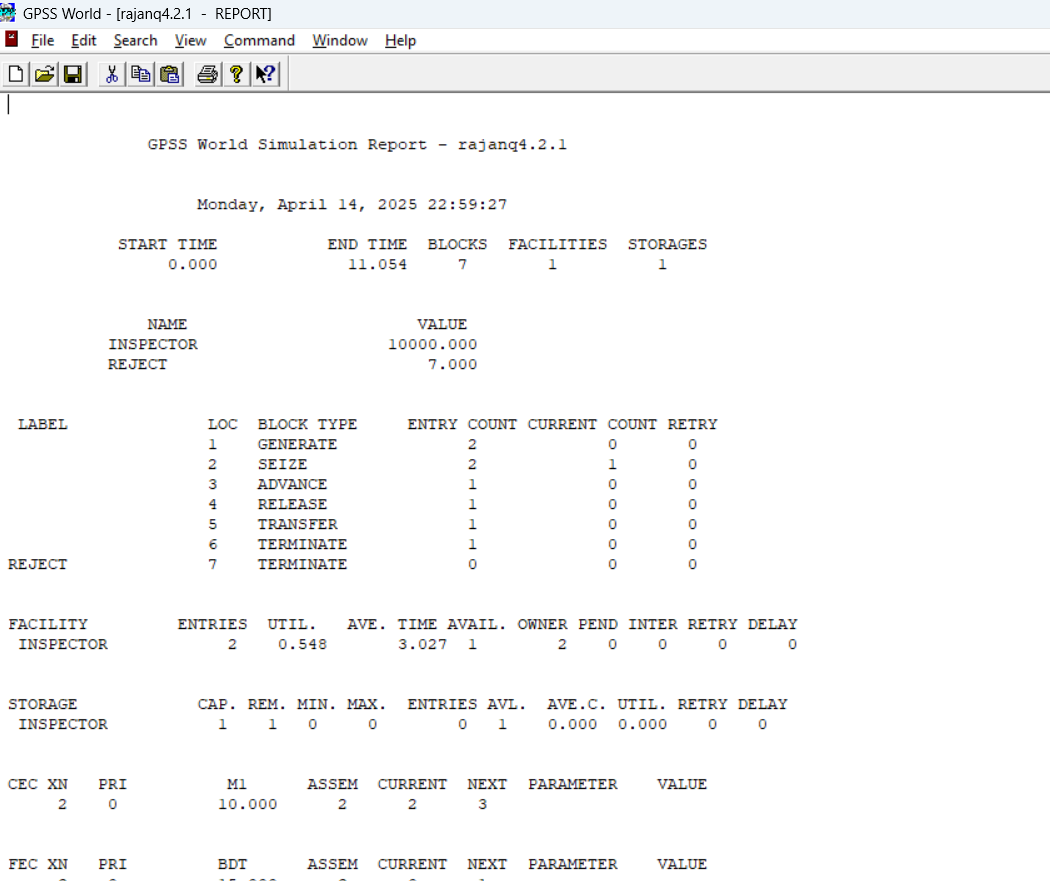
TRANSFER .15, REJECT

TERMINATE 1

REJECT TERMINATE 1

INSPECTOR STORAGE 1

**Output:**

****

**Lab 10**

**1.A machine tool in a manufacturing shop is turning out parts at the rate of every 5 minutes. When they are finished, the parts are sent to an inspector, who takes 4±3 minutes to examine each one and rejects 15% of the parts. Write a GPSS program to simulate using the concept of storage. (Here storage has 3 servers and timing parameters are changed.)**

**Code:**

GENERATE 5,,,1000

ENTER INSPECTOR,1

ADVANCE 4,3

LEAVE INSPECTOR,1

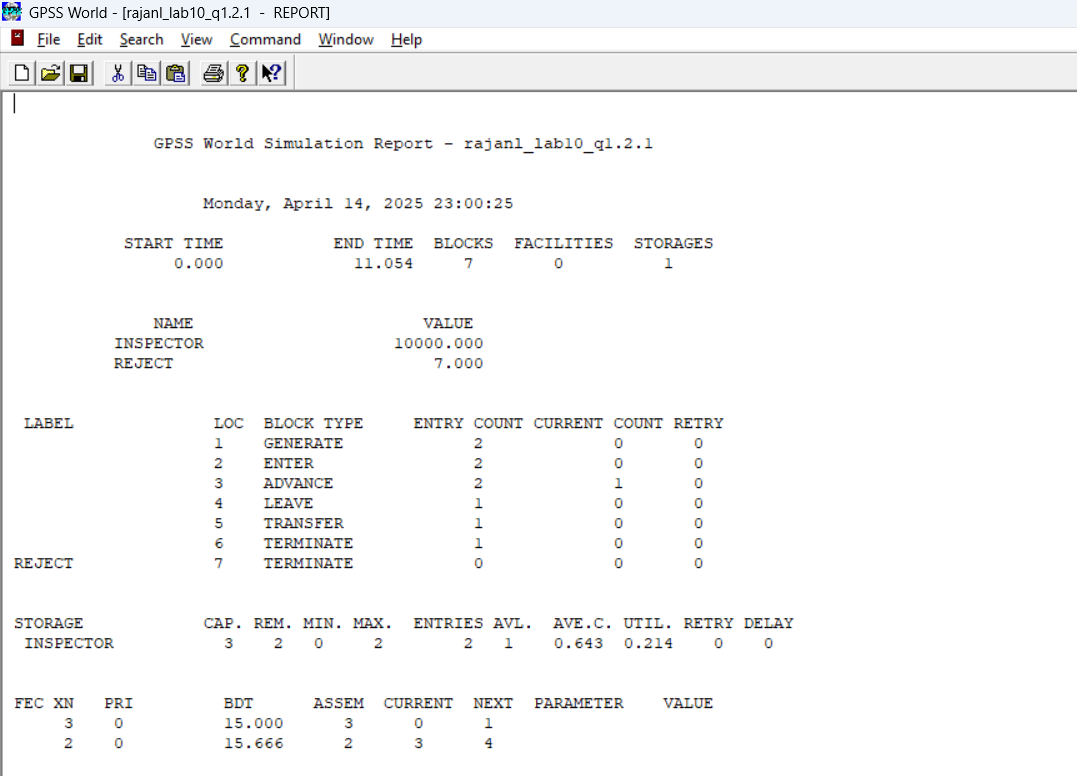
TRANSFER .15, REJECT

TERMINATE 1

REJECT TERMINATE 1

INSPECTOR STORAGE 3

**Output:**

****

**2.Fuel Station Simulation: One vehicle arrives every 2±2 minutes. It takes 5±2 minutes to fuel one vehicle. Number of nozzles = 2. Fuel station operates 10 hours a day. Write a GPSS program to simulate operation of fuel station for entire day**

**Code:**

GENERATE 2,2

QUEUE FUELQ

ENTER NOZZLE,1

DEPART FUELQ

ADVANCE 5,2

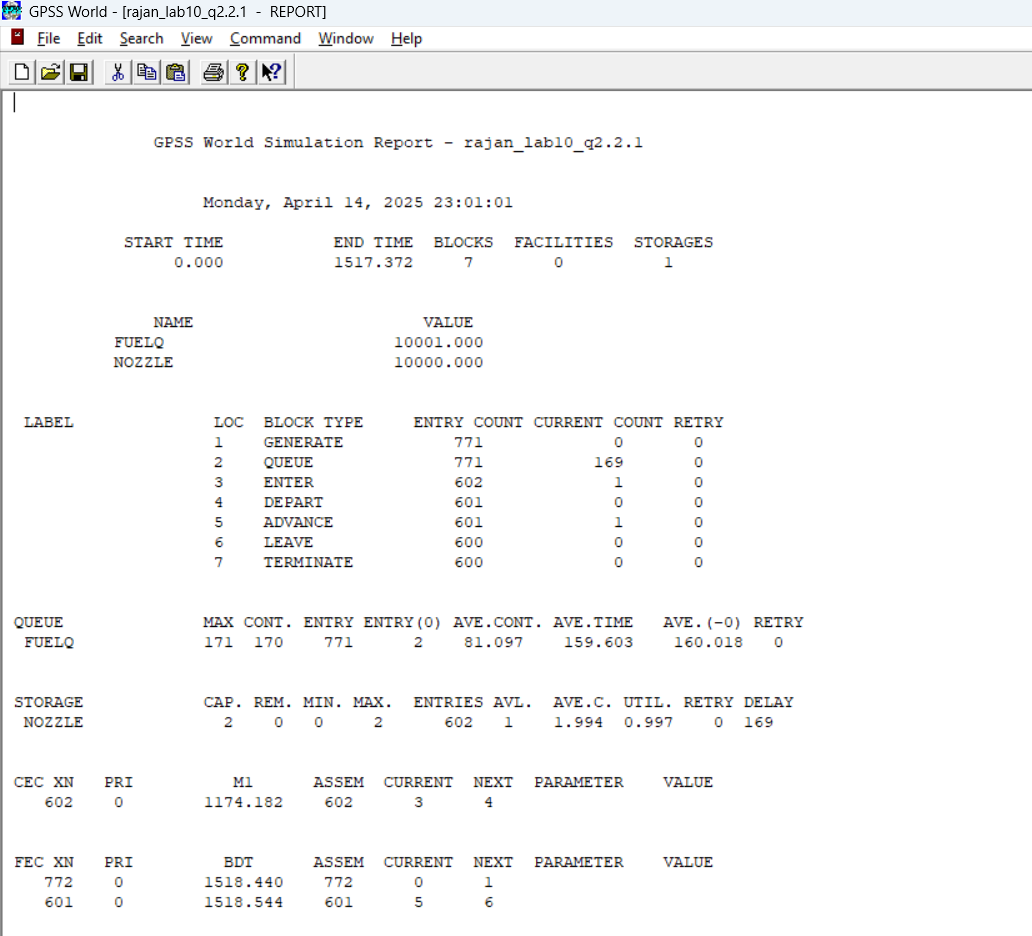
LEAVE NOZZLE,1

TERMINATE 1

NOZZLE STORAGE 2

START 600

**Output:**

****

**Lab 11**

**1.Customers arrive at Joey’s Barbershop one every 15±3 minutes and it takes Joey 18±2 minutes to cut hair of a customer. Write a GPSS program to simulate a Barbershop using the concept of facility and run the simulation for 9 hours.**

**Code:**

GENERATE 15,3

QUEUE BARBERQ

SEIZE JOEY

DEPART BARBERQ

ADVANCE 18,2

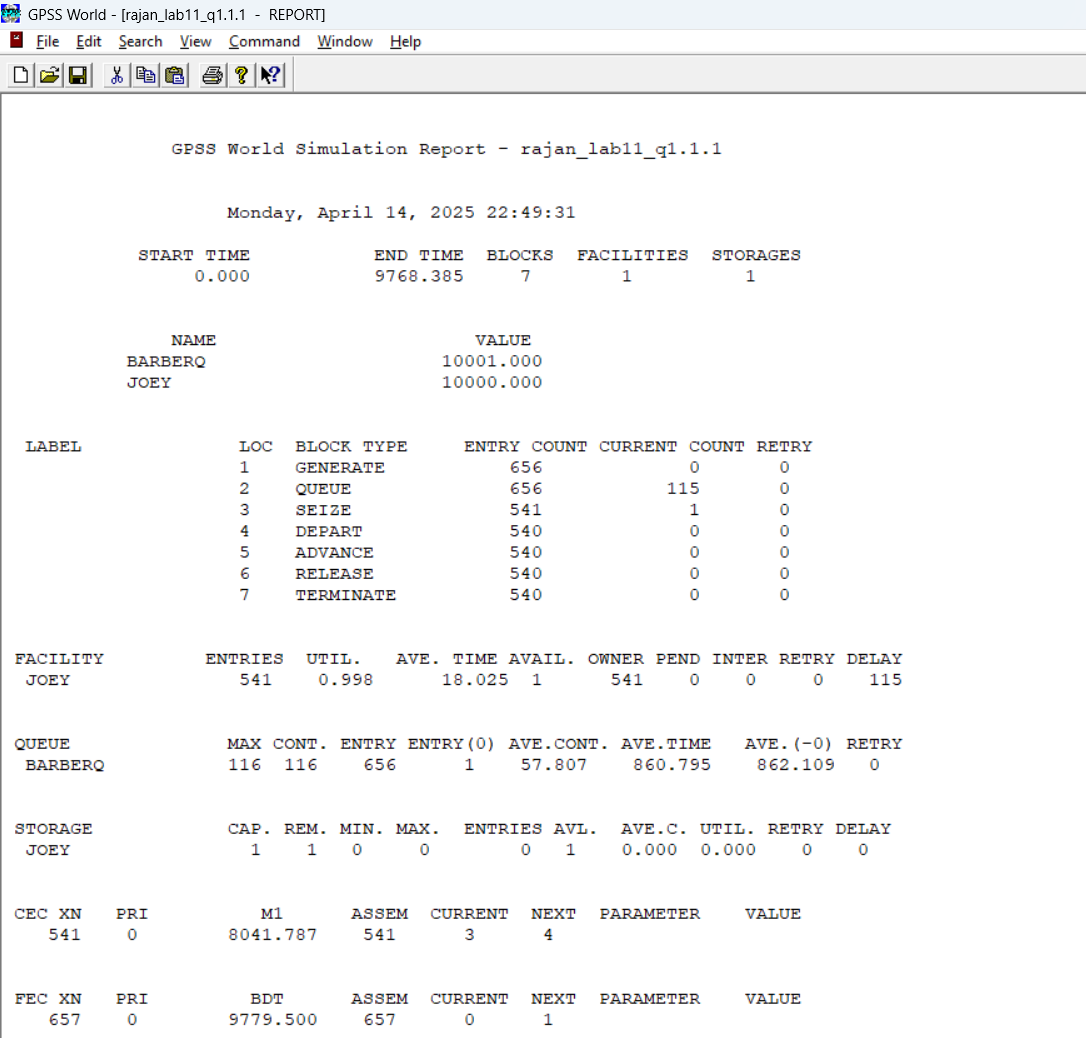
RELEASE JOEY

TERMINATE 1

JOEY STORAGE 1

START 540

**Output:**

****

**2.Simulate random walk problem or a drunkard problem in any programming language.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

int position = 0;

int steps, i;

printf("Enter number of steps: ");

scanf("%d", &steps);

srand(time(0));

for (i = 0; i < steps; i++) {

if (rand() % 2 == 0)

position--;

else

position++;

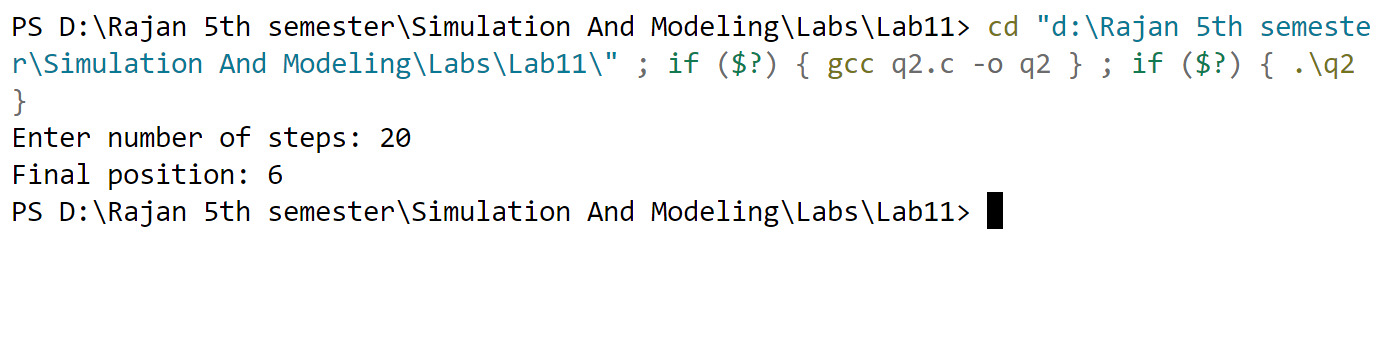
}

printf("Final position: %d\n", position);

return 0;

}

**Output:**

****