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

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

int main(){
	int x, a, c, m, n;
	cout<<"Random numbers generation using Mixed Congurential
Method\n";
	cout<<"Enter Seed: ";
	cin>>x;
	cout<<"Enter constant multiplier: ";
	cin>>a;
	cout<<"Enter incrementer(c != 0): ";
	cin>>c;
	if (c == 0){
		cout<<"df";
	}
	cout<<"Enter modulus(+ve): ";
	cin>>m;
	cout<<"Enter sequence length: ";
	cin>>n;
	for(int i=0; i<n; i++){
		x= (a * x + c)  % m;
		cout<<x<<" ";
	}
	return 0;
}
```

```
Random numbers generation using Mixed Congurential Method
Enter Seed: 23
Enter constant multiplier: 2
Enter incrementer(c != 0): 5
Enter modulus(+ve): 57
Enter sequence length: 11
51 50 48 44 36 20 45 38 24 53 54
--------------------------------
```

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

```
#include <iostream>
using namespace std;
int main(){
        int M = 1000, a = 15, X = 13, i;

        cout<<"Multiplicative Congurential Method \n";
        cout<<"First 50 random numbers: \n";

        for(i=0; i<50; i++) {
                X=(a * X ) % M;
                cout << X << " ";
        }
        return 0;
}
```

```
Multiplicative Congurential Method
First 50 random numbers:
195 925 875 125 875 125 875 125 875 125 875 125 875 125
 875 125 875 125 875 125 875 125 875 125 875 125 875 12
5 875 125 875 125 875 125 875 125 875 125 875 125 875 1
25 875 125 875 125 875 125 875 125
--------------------------------
```

3.WAP to implement (i) Kolmogorov – Smirnov test and (ii) Chi-Square Test

```
// Kolmogorov – Smirnov test for uniformity of random nunbers

#include<iostream>
#include<conio.h>
#include<iomanip>
using namespace std;

class KS{
	private:
		float numbers[20];
		float D,tabulatedD;
		float Dplusmax,Dminusmax;
		float Dplus[20],Dminus[20];
		float ratio[20],ratiominus[20];
		int i,j,n;
	public:
		void getdata(){ //to get the random numbers
			cout<<"How many numbers?: ";
			cin>>n;
			cout<<"Enter "<<n<<" numbers: ";
			for(i=0;i<n;i++){
				cin>>numbers[i];
			}
		}

		void BubbleSort(){ // arrange the number in increasing order
			int i,j; float temp;
			for(i=0;i<n-1;i++){
				for(j=0;j<n-i-1;j++){
					if(numbers[j]>numbers[j+1]){
						temp=numbers[j];
						numbers[j]=numbers[j+1];
						numbers[j+1]=temp;
					}
				}
			}
			cout<<"The numbers in ascending order is:"<<endl;
			for(i=0;i<n;i++)
				cout<<setprecision(2)<<numbers[i]<<" ";
		}

		void calculate(){ // find D+, D-
```

```
        for(i=0;i<n;i++){
                int j;
                j=i+1;
                ratio[i]=(float)j/n;
                ratiominus[i]=(float)i/n;
                Dplus[i]=ratio[i]-numbers[i];
                Dminus[i]=numbers[i]-ratiominus[i];
        }
}

void display(){ // display the tabulated format and find D
        cout<<endl<<endl;
        cout<<setw(10)<<"i";
        for(i=1;i<=n;i++)
                cout<<setw(10)<<i;
        cout<<endl;
        cout<<setw(10)<<"R(i)";
        for(i=0;i<n;i++)
                cout<<setw(10)<<numbers[i];
        cout<<endl;
        cout<<setw(10)<<"i/n";
        for(i=0;i<n;i++)
                cout<<setw(10)<<setprecision(2)<<ratio[i];
        cout<<endl;
        cout<<setw(10)<<"D+";
        for(i=0;i<n;i++)
                cout<<setw(10)<<setprecision(2)<<Dplus[i];
        cout<<endl;
        cout<<setw(10)<<"D-";
        for(i=0;i<n;i++)
                cout<<setw(10)<<setprecision(2)<<Dminus[i];
        cout<<endl;

        Dplusmax=Dplus[0];
        Dminusmax=Dminus[0];
        for(i=1;i<n;i++){
                if(Dplus[i]>Dplusmax)
                        Dplusmax=Dplus[i];
                if(Dminus[i]>Dminusmax)
                        Dminusmax=Dminus[i];
        }
        cout<<"D+ max: "<<Dplusmax<<endl;
        cout<<"D- max: "<<Dminusmax<<endl;
```

```
                cout<<"D =max("<<Dplusmax<<", "<<Dminusmax<<") =";
                if(Dplusmax>Dminusmax)
                        D=Dplusmax;
                else
                        D=Dminusmax;
                cout<<D<<endl;
        }

        void conclusion(){ // asking tabulated D and comparing it with
D(calculated)
                cout<<"Enter the tabulated value: ";
                cin>>tabulatedD;
                if(D<tabulatedD)
                        cout<<"The test is accepted."<<endl;
                else
                        cout<<"The test is rejected."<<endl;
        }
};

int main(){ //main function
        KS ks1; //object of KS class ks1.getdata();

        cout<<"Kolmogorov - Smirnov test\n";
        //function calls
        ks1.getdata();
        ks1.BubbleSort();
        ks1.calculate();
        ks1.display();
        ks1.conclusion();
        getch();
        return(0);
}
```

```
Kolmogorov - Smirnov test
How many numbers?: 10
Enter 10 numbers: 0.44 .19 .88 .27 .55 .13 .63 .74 .24 .33
The numbers in ascending order is:
0.13 0.19 0.24 0.27 0.33 0.44 0.55 0.63 0.74 0.88

        i        1        2        3        4        5        6        7        8        9       10
     R(i)     0.13     0.19     0.24     0.27     0.33     0.44     0.55     0.63     0.74     0.88
      i/n      0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8      0.9        1
       D+    -0.03     0.01     0.06     0.13     0.17     0.16     0.15     0.17     0.16     0.12
       D-     0.13     0.09     0.04    -0.03    -0.07    -0.06    -0.05    -0.07    -0.06    -0.02
D+ max: 0.17
D- max: 0.13
D =max(0.17, 0.13) =0.17
Enter the tabulated value: .410
The test is accepted.
```

```
// chi sq test for uniformity of random nunbers

#include<iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

class kstest{
int O[10],E[10],N;
float diff[10];
float chisquare,chitab;
public:
        void getdata(int n){
                int temp,i;
                cout<<"Enter frequencies of the interval: ";
                for(i=0;i<n;i++)
                        cin>>O[i];
                N=0;
                for(i=0;i<n;i++)
                        N+=O[i];
                temp=N/n;
                for(i=0;i<n;i++)
                        E[i]=temp;
        }

        void calculatechi(int n){
                int i;
                cout<<"\nCalculated differences: ";
                for(i=0;i<n;i++){
                        diff[i]=(pow((O[i]- E[i]),2))/E[i];
                        cout<<"\t"<<diff[i];
                }
                chisquare=0;
                for(i=0;i<n;i++)
                        chisquare+=diff[i];
        }
        void decide(float chi){
                cout<<"\nObtained chi square value:"<<chisquare;
                if(chitab>chisquare)
                        cout<<"\nAccepted :The given distributions are
uniform.\n";
                else
```

```
                    cout<<"\nRejected:The given distributions are not
uniform.\n";
        }
};

int main(){
        kstest calc;
        float n,chitab;

        cout<<"------- Chi Square test-------\n";
        cout<<"Enter the number of classes or values:";
        cin>>n;
        cout<<"Enter the Tabulated value of chi:";
        cin>>chitab;
        calc.getdata(n);
        calc.calculatechi(n);
        calc.decide(chitab);
        return 0;
}
```

```
------- Chi Square test-------
Enter the number of classes or values:10
Enter the Tabulated value of chi:16.9
Enter frequencies of the interval: 8 8 10 9 12 8 10 14 10 11

Calculated differences:         0.4     0.4     0       0.1     0.4     0.4
    0       1.6     0       0.1
Obtained chi square value:3.4
Rejected:The given distributions are not uniform.

--------------------------------
```

4.WAP to implement (i) Autocorrelation Test, (ii) Poker Test

```
// // auto-correlation test for independence of random nunbers
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <cstdlib>
#include <ctime>

using namespace std;

// Function to compute mean of a sequence
double computeMean(const vector<double>& data) {
    double sum = 0.0;
    for (double val : data) sum += val;
    return sum / data.size();
}

// Function to perform autocorrelation test
double autocorrelationTest(const vector<double>& data, int lag) {
    int n = data.size();
    double mean = computeMean(data);

    double numerator = 0.0;
    double denominator = 0.0;

    for (int i = 0; i < n - lag; ++i) {
        numerator += (data[i] - mean) * (data[i + lag] - mean);
    }

    for (int i = 0; i < n; ++i) {
        denominator += (data[i] - mean) * (data[i] - mean);
    }

    return numerator / denominator;
}

int main() {
    int n, lag;
    cout<<"----- Auto-correlation test -----\n";
    cout << "Enter number of random numbers to generate: ";
    cin >> n;
```

```
    cout << "Enter lag value: ";
    cin >> lag;

    if (lag >= n || lag <= 0) {
        cout << "Invalid lag. It must be > 0 and < n.\n";
        return 1;
    }

    vector<double> randomNumbers(n);
    srand(time(0));

    cout << "Generated Random Numbers:\n";
    for (int i = 0; i < n; ++i) {
        randomNumbers[i] = (double)rand() / RAND_MAX;
        cout << fixed << setprecision(4) << randomNumbers[i] << " ";
    }
    cout << endl;

    double r_d = autocorrelationTest(randomNumbers, lag);

    cout << "\nAutocorrelation Coefficient (lag " << lag << "): " << fixed <<
setprecision(5) << r_d << endl;

    if (abs(r_d) < 0.1)
        cout << "=> Likely independent (good randomness)\n";
    else
        cout << "=> Possible correlation (poor randomness)\n";

    return 0;
}
```

```
----- Auto-correlation test -----
Enter number of random numbers to generate: 23
Enter lag value: 3
Generated Random Numbers:
0.8367 0.2608 0.8668 0.8672 0.5531 0.5076 0.8858 0.6897
 0.9718 0.0157 0.7969 0.4645 0.6798 0.1248 0.7808 0.550
1 0.3989 0.7768 0.1878 0.8669 0.9537 0.2389 0.4942

Autocorrelation Coefficient (lag 3): 0.00078
=> Likely independent (good randomness)
```

Poker test

```
#include <iostream>
#include <vector>
#include <map>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <iomanip>
#include <algorithm>

using namespace std;

// Classify digit group
string classifyGroup(const string& group) {
    map<char, int> freq;
    for (char c : group) freq[c]++;

    vector<int> counts;
    for (auto& p : freq) counts.push_back(p.second);

    sort(counts.rbegin(), counts.rend());

    if (counts[0] == 4) return "Four of a kind";
    if (counts[0] == 3 && counts[1] == 1) return "Three of a kind";
    if (counts[0] == 2 && counts[1] == 2) return "Two pairs";
    if (counts[0] == 2) return "One pair";
    return "All different";
}

int main() {
    int totalDigits, groupSize;
    cout<<"----------- Poker Test (Independence Test) -----------\n";
         cout << "Enter total number of random digits to generate (e.g., 20000): ";
    cin >> totalDigits;

    cout << "Enter size of each group (e.g., 4): ";
    cin >> groupSize;

    if (totalDigits % groupSize != 0) {
        cout << "Error: Total digits must be divisible by group size.\n";
        return 1;
    }
```

```
int numGroups = totalDigits / groupSize;
map<string, int> patternFreq;

srand(time(0));

// Generate and classify groups
for (int i = 0; i < numGroups; ++i) {
    string group = "";
    for (int j = 0; j < groupSize; ++j) {
        group += '0' + rand() % 10;
    }
    string pattern = classifyGroup(group);
    patternFreq[pattern]++;
}

// Show observed frequencies
cout << "\nObserved Frequencies:\n";
for (auto& p : patternFreq) {
    cout << setw(20) << left << p.first << ": " << p.second << "\n";
}

// Theoretical probabilities for 4-digit Poker test (base-10):
// Based on known probabilities from statistical tables
map<string, double> expectedProbs = {
    {"All different", 0.504},
    {"One pair",      0.432},
    {"Two pairs",     0.027},
    {"Three of a kind", 0.036},
    {"Four of a kind",  0.001}
};

// Chi-square test
double chi_square = 0.0;

cout << "\nChi-Square Components:\n";
for (auto& p : expectedProbs) {
    int observed = patternFreq[p.first];
    double expected = numGroups * p.second;

    double diff = observed - expected;
    chi_square += (diff * diff) / expected;
```

```
        cout << setw(20) << left << p.first
             << "O=" << observed << ", E=" << fixed << setprecision(2) << expected
             << ", contrib=" << (diff * diff) / expected << endl;
    }

    cout << "\nChi-Square Statistic: " << fixed << setprecision(4) << chi_square <<
endl;

    int degrees_of_freedom = expectedProbs.size() - 1;  // k - 1
    cout << "Degrees of freedom: " << degrees_of_freedom << endl;
    cout << "Compare with chi-square critical value for df = " <<
degrees_of_freedom << " at 0.05 significance level: ~9.488\n";

    if (chi_square < 9.488)
        cout << "Sequence passes the Poker test (likely independent/random).\n";
    else
        cout << "Sequence fails the Poker test (may not be truly random).\n";

    return 0;
}
```

```
---------- Poker Test (Independence Test) ----------
Enter total number of random digits to generate (e.g., 20000): 4500
Enter size of each group (e.g., 4): 4

Observed Frequencies:
All different       : 556
Four of a kind      : 1
One pair            : 509
Three of a kind     : 36
Two pairs           : 23

Chi-Square Components:
All different       O=556, E=567.00, contrib=0.21
Four of a kind      O=1, E=1.12, contrib=0.01
One pair            O=509, E=486.00, contrib=1.09
Three of a kind     O=36, E=40.50, contrib=0.50
Two pairs           O=23, E=30.38, contrib=1.79

Chi-Square Statistic: 3.6064
Degrees of freedom: 4
Compare with chi-square critical value for df = 4 at 0.05 significance level: ~9.488
Sequence passes the Poker test (likely independent/random).
```

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

```
// point estimation

#include <iostream>
#include <vector>
#include <numeric>  // for accumulate
#include <iomanip>  // for setprecision

using namespace std;

int main() {
    int n;
    double population_mean;

    // Input population mean
    cout << "Enter the population mean: ";
    cin >> population_mean;

    // Input sample size
    cout << "Enter the number of samples: ";
    cin >> n;

    vector<double> sample(n);

    // Input sample data
    cout << "Enter the sample data:\n";
    for (int i = 0; i < n; ++i) {
        cin >> sample[i];
    }

    // Calculate sample mean
    double sum = accumulate(sample.begin(), sample.end(), 0.0);
    double sample_mean = sum / n;

    // Calculate bias
    double bias = sample_mean - population_mean;
    // Output results
    cout << fixed << setprecision(4);
    cout << "\nSample Mean (Point Estimate): " << sample_mean << endl;
    cout << "Bias = Sample Mean - Population Mean: " << bias << endl;
    return 0;
}
```

```
Enter the population mean: 29
Enter the number of samples: 10
Enter the sample data:
3 43 20 14 11 35 29 23 19 20

Sample Mean (Point Estimate): 21.7000
Bias = Sample Mean - Population Mean: -7.3000

--------------------------------
```

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

```
//interval/ confidence interval estimation for a sample of data with given
population mean.

#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <numeric> // for accumulate

using namespace std;

// Function to calculate standard deviation
double calculate_std_dev(const vector<double>& data, double mean) {
    double sum = 0.0;
    for (double x : data) {
        sum += (x - mean) * (x - mean);
    }
    return sqrt(sum / (data.size() - 1));  // sample standard deviation
}

int main() {
    int n;
    double population_mean;
    double confidence_level;

    cout << "Enter the number of samples: ";
    cin >> n;

    vector<double> sample(n);

    cout << "Enter the sample data:\n";
    for (int i = 0; i < n; ++i) {
        cin >> sample[i];
    }

    cout << "Enter population mean (if known, else enter 0): ";
    cin >> population_mean;

    cout << "Enter confidence level (e.g. 0.95 for 95%): ";
    cin >> confidence_level;
```

```
    // Z-scores for common confidence levels (you can extend this)
    double z = 0;
    if (confidence_level == 0.90) z = 1.645;
    else if (confidence_level == 0.95) z = 1.960;
    else if (confidence_level == 0.99) z = 2.576;
    else {
        cout << "Unsupported confidence level. Use 0.90, 0.95, or 0.99.\n";
        return 1;
    }
    // Calculate sample mean
    double sum = accumulate(sample.begin(), sample.end(), 0.0);
    double sample_mean = sum / n;

    // Calculate sample standard deviation
    double std_dev = calculate_std_dev(sample, sample_mean);

    // Calculate margin of error
    double margin = z * (std_dev / sqrt(n));

    // Confidence interval
    double lower = sample_mean - margin;
    double upper = sample_mean + margin;

    // Output
    cout << fixed << setprecision(4);
    cout << "\nSample Mean: " << sample_mean << endl;
    cout << "Sample Standard Deviation: " << std_dev << endl;
    cout << confidence_level * 100 << "% Confidence Interval: ["
         << lower << ", " << upper << "]\n";

    return 0;
}
```

```
Enter the number of samples: 10
Enter the sample data:
43 23 44 20 34 34 55 32 29 30
Enter population mean (if known, else enter 0): 0
Enter confidence level (e.g. 0.95 for 95%): .95

Sample Mean: 34.4000
Sample Standard Deviation: 10.4478
95.0000% Confidence Interval: [27.9244, 40.8756]

--------------------------------
```

7. Write a program to implement markov chain to predict weather condition.

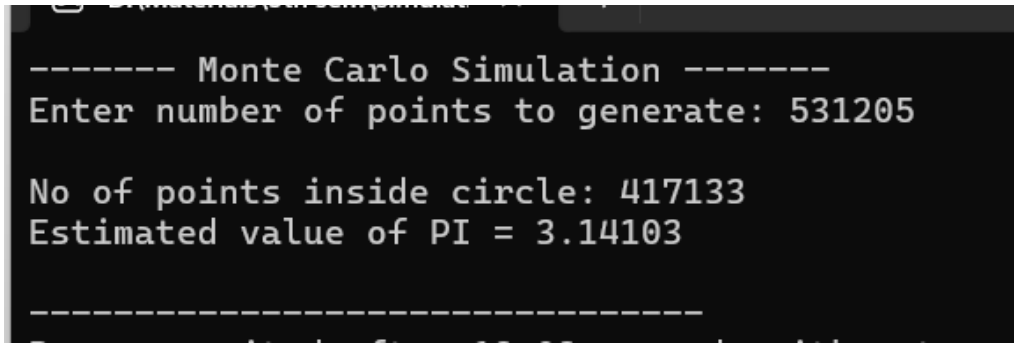
```
// markov chain to predict weather condition.

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>

using namespace std;

// Weather states
enum Weather { Sunny, Cloudy, Rainy };

// Transition matrix
const double transition[3][3] = {
   {0.6, 0.3, 0.1},  // From Sunny
   {0.2, 0.5, 0.3},  // From Cloudy
   {0.3, 0.4, 0.3}   // From Rainy
};

// Convert state to string
string getWeatherName(Weather state) {
   switch (state) {
      case Sunny: return "Sunny";
      case Cloudy: return "Cloudy";
      case Rainy: return "Rainy";
      default: return "Unknown";
   }
}

// Get next state based on current state and probabilities
Weather getNextWeather(Weather current) {
   double randVal = (double)rand() / RAND_MAX;
   double cumulative = 0.0;

   for (int i = 0; i < 3; ++i) {
      cumulative += transition[current][i];
      if (randVal < cumulative) {
         return static_cast<Weather>(i);
      }
   }
```

```
    // fallback
    return Sunny;
}

int main() {
    srand(time(0));  // Seed RNG

    int days;
    cout << "----- Markov Chain(Weather Prediction) ----\n";
         cout << "Enter number of days to simulate: ";
    cin >> days;

    Weather current = Sunny;  // Initial state
    cout << "Day 0: " << getWeatherName(current) << endl;

    for (int i = 1; i <= days; ++i) {
        current = getNextWeather(current);
        cout << "Day " << i << ": " << getWeatherName(current) << endl;
    }

    return 0;
}
```

```
----- Markov Chain(Weather Prediction) ----
Enter number of days to simulate: 6
Day 0: Sunny
Day 1: Cloudy
Day 2: Cloudy
Day 3: Sunny
Day 4: Sunny
Day 5: Cloudy
Day 6: Rainy

--------------------------------
```

8. Write a program to estimate the value of PI using Monte Carlo Simulation.

```
#include <iostream>
#include <cstdlib>  // for rand() and RAND_MAX
#include <ctime>    // for time()
using namespace std;

int main() {
   int num_points;
   int inside_circle = 0;

   cout<<"------- Monte Carlo Simulation -------\n";
        cout << "Enter number of points to generate: ";
   cin >> num_points;

   // Seed the random number generator
   srand(time(0));

   for (int i = 0; i < num_points; ++i) {
      // Generate random (x, y) in [0, 1]
      double x = (double)rand() / RAND_MAX;
      double y = (double)rand() / RAND_MAX;

      // Check if the point is inside the quarter circle
      if (x * x + y * y <= 1.0) {
         inside_circle++;
      }
   }
   // Estimate PI
   double pi_estimate = 4.0 * inside_circle / num_points;

   cout<<"\nNo of points inside circle: "<<inside_circle<<endl;
cout << "Estimated value of PI = " << pi_estimate << endl;

   return 0;
}
```



9. Write a program to estimate the area under the curve using the Monte Carlo Simulation.

```
// area under curve using monte carlo simulation

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using namespace std;

// Function to define the curve
double f(double x) {
   return x * x;  // You can change this function
}

int main() {
   int num_points;
   double a = 0.0, b = 1.0;       // Interval [a, b]
   double f_max = 1.0;            // Max value of f(x) on [0,1] for x^2 is 1.0

        cout<<"--- Area Under Curve (Monte Carlo) ----\n";
   cout << "Enter number of random points to generate: ";
   cin >> num_points;

   srand(time(0));  // Seed RNG

   int points_below_curve = 0;

   for (int i = 0; i < num_points; ++i) {
      double x = a + (double)rand() / RAND_MAX * (b - a);
      double y = (double)rand() / RAND_MAX * f_max;

      if (y <= f(x)) {
         points_below_curve++;
      }
   }

   double area_estimate = ((double)points_below_curve / num_points) * (b - a) *
f_max;
        cout<<"\nPoints below curve: "<<points_below_curve;
```

```
    cout << "\nEstimated area under the curve y = x^2 from " << a << " to " << b << "
is: " << area_estimate << endl;

    return 0;
}
```

```
--- Area Under Curve (Monte Carlo) ----
Enter number of random points to generate: 4308745

Points below curve: 1436531
Estimated area under the curve y = x^2 from 0 to 1 is: 0.333399

--------------------------------
```

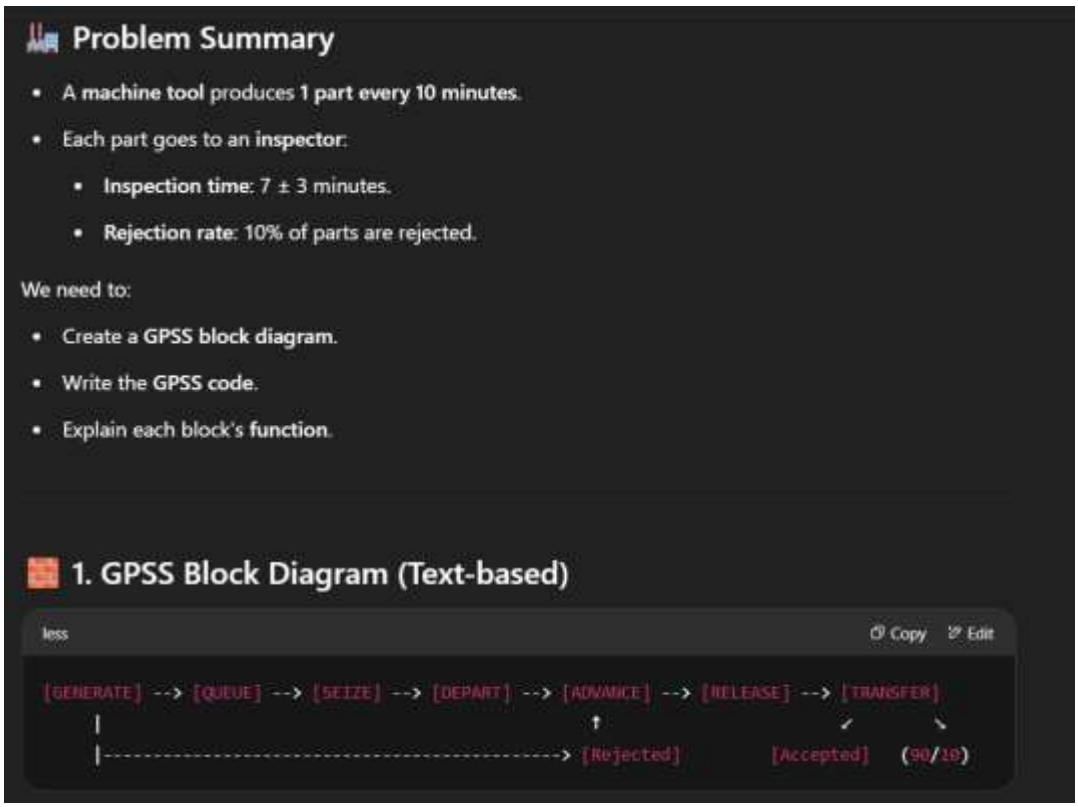
10. Write a program to calculate measures of a M/M/1 Queue for a given value of Arrival Rate and Service Rate.

```
// M M 1 queuing system

#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    double lambda, mu;

    // Input values
    cout << "Enter Arrival Rate : ";
    cin >> lambda;

    cout << "Enter Service Rate : ";
    cin >> mu;

    // Check stability condition
    if (lambda >= mu) {
        cout << "System is unstable. Arrival rate must be less than service rate (λ <
μ)." << endl;
        return 1;
    }
```

```
    // Calculating traffic intensity
    double rho = lambda / mu;

    // Calculating M/M/1 queue measures
    double L = lambda / (mu - lambda); // Average number in system
    double Lq = (lambda * lambda) / (mu * (mu - lambda)); // Average number in
queue
    double W = 1 / (mu - lambda); // Average time in system
    double Wq = lambda / (mu * (mu - lambda)); // Average time in queue

    // Displaying results
    cout << fixed << setprecision(3);
    cout << "\nM/M/1 Queue Performance Measures:\n";
    cout << "Traffic Intensity (ρ): " << rho << endl;
    cout << "Average number in system (L): " << L << endl;
    cout << "Average number in queue (Lq): " << Lq << endl;
    cout << "Average time in system (W): " << W << endl;
    cout << "Average time in queue (Wq): " << Wq << endl;

    return 0;
}
```

```
Enter Arrival Rate : 5
Enter Service Rate : 7.5

M/M/1 Queue Performance Measures:
Traffic Intensity (╧ü): 0.667
Average number in system (L): 2.000
Average number in queue (Lq): 1.333
Average time in system (W): 0.400
Average time in queue (Wq): 0.267

--------------------------------
```

Consider a machine tool in a manufacturing shop is turning out parts at the rate of one every 10 minutes. As they are finished, the parts go to an inspector, who takes 7 ± 3 minutes to examine each one and rejects about 10% of the parts. Develop a block diagram and write the code for simulating the above problem using GPSS, and also explain the function of each block used in the block diagram in detail. (Model 1 to 5 from Gordon Book)



```
* Machine Tool and Inspection Simulation

* Part generation every 10 minutes
GENERATE 10

* Inspector queue and processing
QUEUE    INSPECTOR
SEIZE    INSPECTOR
DEPART   INSPECTOR

* Inspection time: 7 ± 3 minutes
ADVANCE  7,3

* Inspector is free
RELEASE  INSPECTOR

* Decide rejection (10% chance)
TRANSFER .1, REJECTED

* Accepted part - end transaction
TERMINATE 1

* Rejected part processing
REJECTED  TERMINATE 1

* Define the inspector resource (no explicit declaration required in GPSS)
* Simulation control: run for 1000 parts total
START 1000
```

```
          START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
               0.000          10008.107     9        1          0


            NAME                        VALUE
          INSPECTOR                   10000.000
          REJECTED                        9.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE          1000             0       0
                    2    QUEUE             1000             0       0
                    3    SEIZE             1000             0       0
                    4    DEPART            1000             0       0
                    5    ADVANCE           1000             0       0
                    6    RELEASE           1000             0       0
                    7    TRANSFER          1000             0       0
                    8    TERMINATE          906             0       0
REJECTED            9    TERMINATE           94             0       0


FACILITY         ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 INSPECTOR         1000    0.691       6.915  1        0    0    0     0       0


QUEUE              MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 INSPECTOR           1    0   1000   1000     0.000      0.000      0.000   0


FEC XN    PRI         BDT       ASSEM  CURRENT  NEXT  PARAMETER    VALUE
  1001     0       10010.000    1001      0      1
```

🧩 3. Block-by-Block Explanation

Block	Function
`GENERATE 10`	Creates 1 part every 10 minutes.
`QUEUE INSPECTOR`	Puts the part in line if inspector is busy.
`SEIZE INSPECTOR`	Tries to get control of the inspector (a facility).
`DEPART INSPECTOR`	Removes the part from the inspector queue.
`ADVANCE 7,3`	Simulates inspection time (7 ± 3 mins → Uniform between 4 and 10).
`RELEASE INSPECTOR`	Frees the inspector for the next part.
`TRANSFER .1, REJECTED`	Sends 10% of parts to `REJECTED` label (i.e., 10% are rejected).
`TERMINATE 1` (Accepted)	Ends the transaction for accepted parts.
`REJECTED TERMINATE 1`	Ends the transaction for rejected parts.
`START 1000`	Starts simulation for 1000 parts (i.e., 1000 terminations).

📊 Possible Extensions

- To count rejected vs accepted parts, you can add counters via `TABULATE`, `SAVEVALUE`, or `COUNT` blocks.
- You can simulate multiple inspectors by handling the facility as multi-capacity (`STORAGE` instead of `FACILITY`).

🧪 Sample GPSS Behavior

Time	Event
0	Part 1 generated
10	Part 2 generated
12	Part 1 done with inspection
20	Part 3 generated
...	Rejected parts routed out

Write a GPSS model to simulate a barber shop where each costumer enters the Shop every 10 ± 2 minutes and a barber takes 13 ± 2 for a haircut. Run the simulation for 1 hour and prepare the report.

```
* Barber Shop Simulation

* Generate customers every 10 ± 2 minutes
GENERATE   10,2

* Wait in queue if barber is busy
QUEUE      BARBER
SEIZE      BARBER
DEPART     BARBER

* Haircut takes 13 ± 2 minutes
ADVANCE    13,2

* Release barber after haircut
RELEASE    BARBER

* Customer leaves system
TERMINATE  1

* Timer to stop simulation after 60 minutes
GENERATE   60
TERMINATE  1

* Start simulation with termination counter = 1
START      1
```

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.000	25.893	9	1	0

NAME	VALUE
BARBER	10000.000

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

FACILITY	ENTRIES	UTIL.	AVE. TIME	AVAIL.	OWNER	PEND	INTER	RETRY	DELAY
BARBER	2	0.561	7.262	1	3	0	0	0	0

QUEUE	MAX	CONT.	ENTRY	ENTRY(0)	AVE.CONT.	AVE.TIME	AVE.(-0)	RETRY
BARBER	1	1	2	1	0.132	1.707	3.413	0

CEC XN	PRI	M1	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
3	0	22.480	3	3	4		

FEC XN	PRI	BDT	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
4	0	31.522	4	0	1		
2	0	60.000	2	0	8		