DR B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR

LAB FILE
OF
System Simulation and Modeling
CSX-424

Session: Jan - May 2021

SUBMITTED TO-
Dr. Kuldeep Kumar
Assistant Professor
CSE Department

SUBMITTED BY-
Ankit Goyal
Roll No.- 17103011
Group - G-1
Branch - CSE

INDEX

Experiment -1

Aim: Implement Chi Square Test.

Theory:

One of the primary tasks involved in any supervised Machine Learning venture is to select the best features from the given dataset to obtain the best results. One way to select these features is the Chi-Square Test.

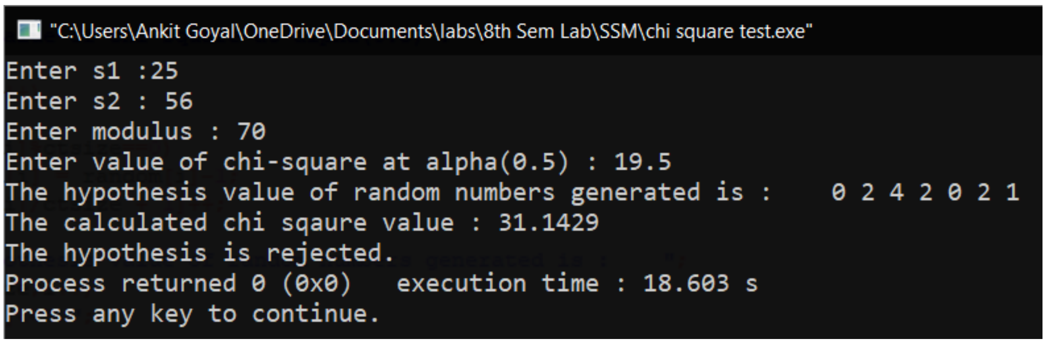
Mathematically, a Chi-Square test is done on two distributions two determine the level of similarity of their respective variances. In its **null hypothesis**, it assumes that the given distributions are independent. This test thus can be used to determine the best features for a given dataset by determining the features on which the output class label is most dependent on. For each feature in the dataset, the value is calculated and then ordered in descending order according to the value. The higher the value, the more dependent the output label is on the feature and higher the importance the feature has on determining the output.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   float chi;
   int random[20];
   int s1,s2,mod,i=2;
   cout<<"Enter s1 :";
   cin>>s1;
   cout<<"Enter s2 : ";
   cin>>s2;
   cout<<"Enter modulus : ";
   cin>>mod;

   /// Generating Random numbers
   random[0] = (s1+s2)%mod;
   random[1] = (s2+random[0])%mod;
   for(i=2;i<20;i++)
      random[i] = (random[i-1] + random[i-2])%mod;

   int ctsize = ceil(mod/10);
   int ct[20];
   memset(ct,0,sizeof(ct));

   cout<<"Enter value of chi-square at alpha(0.5) : ";
   cin>>chi;
   for(i=0;i<20;i++)
   {
      if(random[i]%ctsize==0)
```

```
            random[i] = random[i]-1;
        ct[random[i]/ctsize + 1]++;
    }
    cout<<"The hypothesis value of random numbers generated is :    ";
    for(i=0;i<ctsize;i++)
        cout<<ct[i]<<" ";
    cout<<endl;

    float chisq  = 0;
    for(i=0;i<ctsize;i++)
    {
        chisq += ((ctsize - (float)ct[i])*(ctsize - (float)ct[i]))/ctsize;
    }
    cout<<"The calculated chi sqaure value : "<<chisq<<"\n";
    if(chisq<chi)
        cout<<"The hypothesis is true: The numbers are uniformly distributed";
    else
        cout<<"The hypothesis is rejected.";

}
```

Output:



"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\chi square test.exe"

```
Enter s1 :25
Enter s2 : 56
Enter modulus : 70
Enter value of chi-square at alpha(0.5) : 19.5
The hypothesis value of random numbers generated is :    0 2 4 2 0 2 1
The calculated chi sqaure value : 31.1429
The hypothesis is rejected.
Process returned 0 (0x0)   execution time : 18.603 s
Press any key to continue.
```

Experiment -2

Aim: To generate random numbers using:

1. Mid square method
2. Residue method
3. Arithmetic congruence method

Theory:

Mid Square Method was proposed by Van Neumann. In this method, we have a seed and then the seed is squared and its midterm is fetched as the random number. Consider if we have a seed having N digits we square that number to get a 2N digits number if it doesn't become 2N digits we add zeros before the number to make it 2N digits.

The most common method of generating the random number sequence is known as **the residue method**. Multiply the previous random number by the constant a, add on another constant c, take the modulus by M, and then keep just the fractional part (remainder) as the next random number.

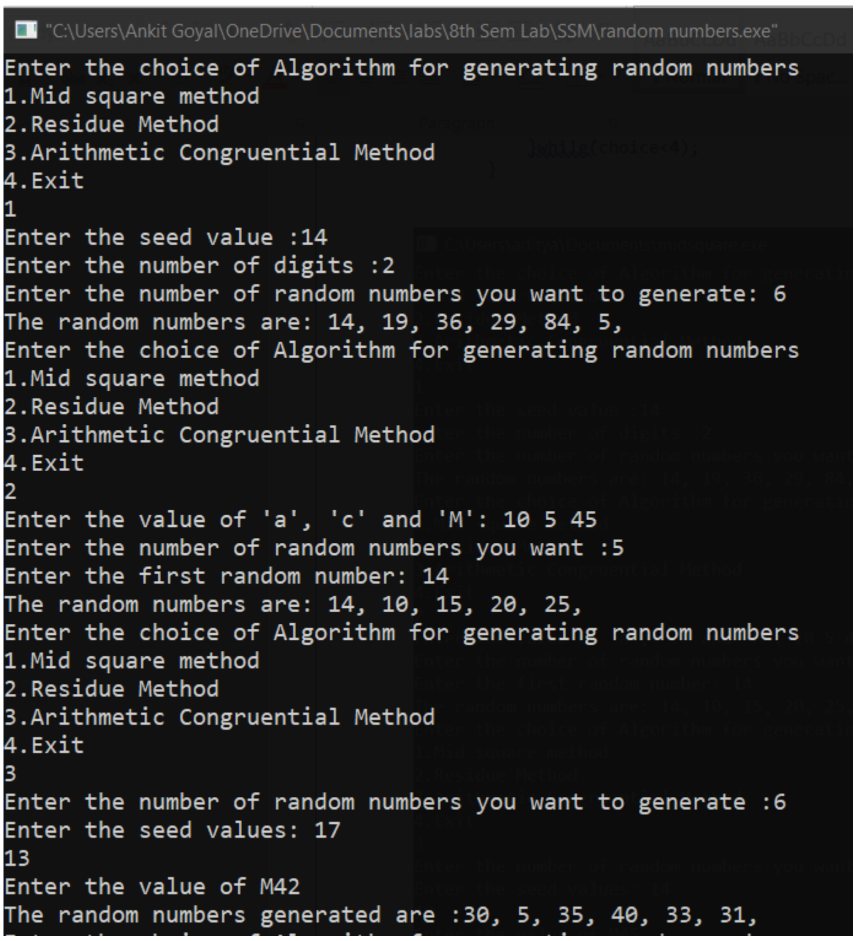
Arithmetic Congruential Method uses 2 seed values to generate random numbers. It is described by formula :
$R_{n+1} = (R_{n-1} + R_n) \bmod M$

Code:

```
#include<bits/stdc++.h>
using namespace std;

long long  a[] = { 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000,
100000000 };
int middleSquareNumber(int numb, int dig)
{
   long long int sqn = numb * numb, next_num = 0;
   int trim = (dig / 2);
   sqn = sqn / a[trim];
   for (int i = 0; i < dig; i++)
   {
      next_num += (sqn % (a[trim])) * (a[i]);
      sqn = sqn / 10;
   }
   return next_num;
}
void midsq()
{
  cout<<"Enter the seed value :";
  int seed;
  cin>>seed;
  int dig;
  cout<<"Enter the number of digits :";
  cin>>dig;
  int n;
  cout<<"Enter the number of random numbers you want to generate: ";
  cin>>n;
  cout<<"The random numbers are: ";
  cout<<seed<<", ";
  int ni=seed;
  for(int i=1; i<n; ++i)
  {
      ni = middleSquareNumber(ni,dig);
      cout<<ni<<", ";
  }
  cout<<"\n";


}
void residue()
{
  int a,c,M,r;
  cout<<"Enter the value of 'a', 'c' and 'M': ";
  cin>>a>>c>>M;
  cout<<"Enter the number of random numbers you want :";
  int n;
  cin>>n;
  cout<<"Enter the first random number: ";
  cin>>r;
  cout<<"The random numbers are: ";
  cout<<r<<", ";
  int rd=r;
  for(int i=1; i<n; ++i)
  {
     r= (a*rd+ c)%M;
     rd=r;
     cout<<r<<", ";
  }
  cout<<"\n";

}
void arithmeticCong()
{

int n,a,b,m;
cout<<"Enter the number of random numbers you want to generate :";
cin>>n;
cout<<"Enter the seed values: ";
cin>>a>>b;
cout<<"Enter the value of M";
cin>>m;
int r;
cout<<"The random numbers generated are :";
```

```
for(int i=0; i<n; ++i)
{
   r=(a+b)%m;
   b=a;
   a=r;
   cout<<r<<", ";
}
cout<<"\n";
}
int main()
{

   int choice;
   do{
   cout<<"Enter the choice of Algorithm for generating random numbers\n1.Mid
square method\n2.Residue Method\n3.Arithmetic Congruential Method\n4.Exit\n";
   cin>>choice;

   switch(choice)
   {
   case 1: midsq();
      break;
   case 2: residue();
      break;
   case 3: arithmeticCong();
      break;
   case 4:
      break;
   default:
      cout<<"Wrong choice\n";
      break;
   }
   }while(choice<4);
}
```

Output:



```
"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\random numbers.exe"
Enter the choice of Algorithm for generating random numbers
1.Mid square method
2.Residue Method
3.Arithmetic Congruential Method
4.Exit
1
Enter the seed value :14
Enter the number of digits :2
Enter the number of random numbers you want to generate: 6
The random numbers are: 14, 19, 36, 29, 84, 5,
Enter the choice of Algorithm for generating random numbers
1.Mid square method
2.Residue Method
3.Arithmetic Congruential Method
4.Exit
2
Enter the value of 'a', 'c' and 'M': 10 5 45
Enter the number of random numbers you want :5
Enter the first random number: 14
The random numbers are: 14, 10, 15, 20, 25,
Enter the choice of Algorithm for generating random numbers
1.Mid square method
2.Residue Method
3.Arithmetic Congruential Method
4.Exit
3
Enter the number of random numbers you want to generate :6
Enter the seed values: 17
13
Enter the value of M42
The random numbers generated are :30, 5, 35, 40, 33, 31,
```

Experiment -3

Aim: Implement a program to perform Kolmogorov Smirnov (KS) test.

THEORY:

Kolmogorov–Smirnov test a very efficient way to determine if two samples are significantly different from each other. It is usually used to check the uniformity of random numbers. Uniformity is one of the most important properties of any random number generator and Kolmogorov–Smirnov test can be used to test it.

The Kolmogorov–Smirnov test may also be used to test whether two underlying one-dimensional probability distributions differ. It is a very efficient way to determine if two samples are significantly different from each other.

The Kolmogorov–Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples.

To use the test for checking the uniformity of random numbers, we use the CDF (cumulative distribution function) of U[0, 1].

$F(x)= x$ for $0<=x<=1$

Empirical CDF, $S_n(x)=$ (number of R1, R2…Rn < x)/N array of random numbers, the random numbers must be in the range of [0, 1].

PROGRAM:

```
#include<bits/stdc++.h>
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?:"<<endl;
                        cin>>n;
                        cout<<"Enter "<<n<<" numbers"<<endl;
                        for(i=0;i<n;i++)
```

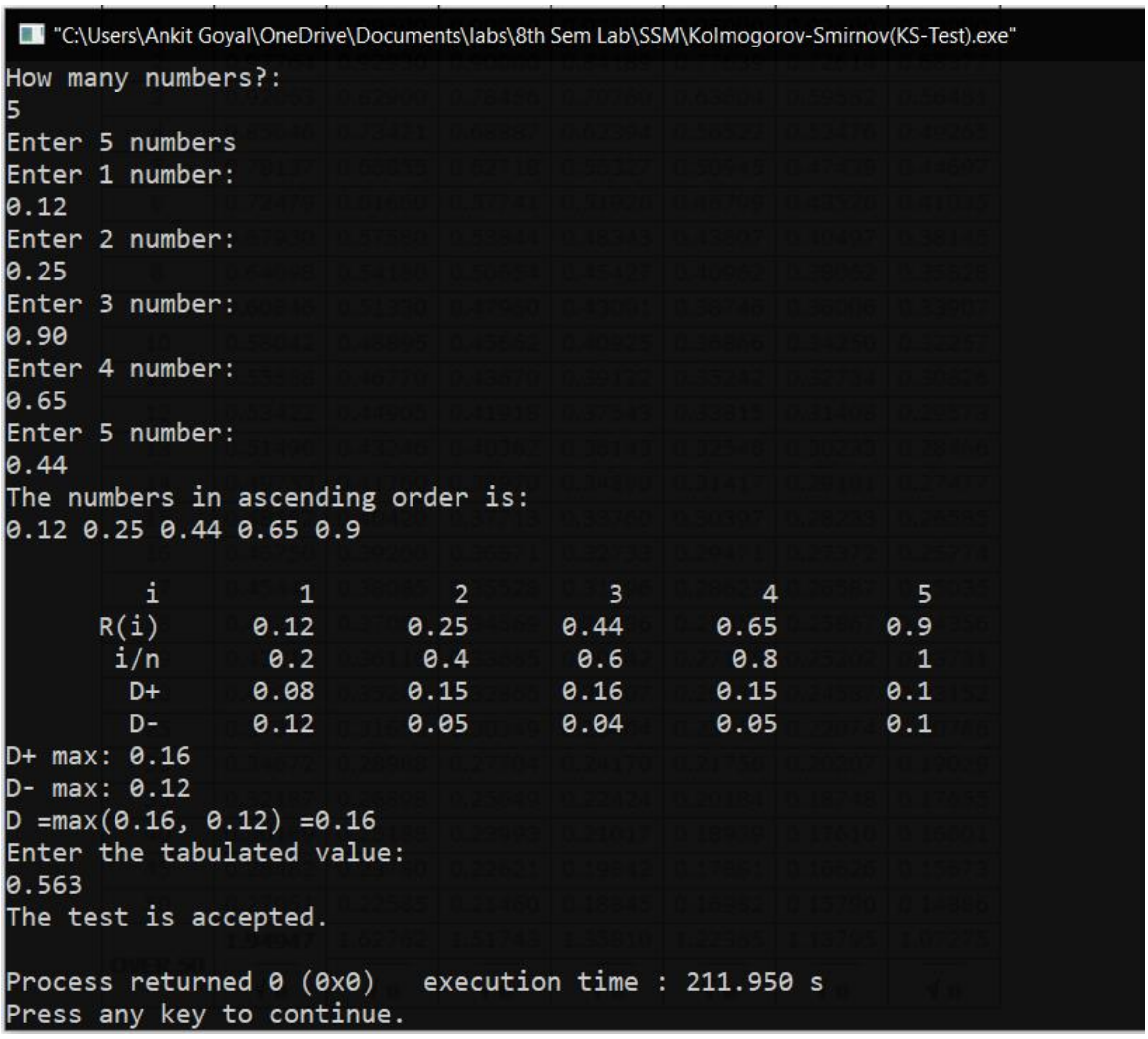
```
		{
			cout<<"Enter "<<i+1<<" number:"<<endl;
			cin>>numbers[i];
		}
	}
	void BubbleSort() // arrange the number in increasing order
	{
		int i,j;
		float temp;
		for(i=0;i<n-1;i++)
		{
			for(j=0;j<n-i-1;j++)
			{
				if(numbers[j]>numbers[j+1])
				{
					temp=numbers[j];
					numbers[j]=numbers[j+1];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

```
cout<<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;
            cout<<endl;

        }
    void conclusion() // asking tabulated D and comparing it with D(calculated)
    {
        cout<<"Enter the tabulated value:"<<endl;
        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(); //function calls
    ks1.BubbleSort();
    ks1.calculate();
    ks1.display();
    ks1.conclusion();
    return(0);
```

}

OUTPUT:



```
"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\Kolmogorov-Smirnov(KS-Test).exe"
How many numbers?:
5
Enter 5 numbers
Enter 1 number:
0.12
Enter 2 number:
0.25
Enter 3 number:
0.90
Enter 4 number:
0.65
Enter 5 number:
0.44
The numbers in ascending order is:
0.12 0.25 0.44 0.65 0.9

         i         1         2         3         4         5
      R(i)      0.12      0.25      0.44      0.65       0.9
       i/n       0.2       0.4       0.6       0.8         1
        D+      0.08      0.15      0.16      0.15       0.1
        D-      0.12      0.05      0.04      0.05       0.1
D+ max: 0.16
D- max: 0.12
D =max(0.16, 0.12) =0.16
Enter the tabulated value:
0.563
The test is accepted.

Process returned 0 (0x0)   execution time : 211.950 s
Press any key to continue.
```

Experiment -4

Aim: To implement a Monte Carlo Simulation.

Theory:

Monte Carlo simulation is a computerized mathematical technique that allows people to account for risk in quantitative analysis and decision making. The technique is used by professionals in such widely disparate fields as finance, project management, energy, manufacturing, engineering, research and development, insurance, oil & gas, transportation, and the environment. Monte Carlo simulation furnishes the decision-maker with a range of possible outcomes and the probabilities they will occur for any choice of action. It shows the extreme possibilities the outcomes of going for broke and for the most conservative decision along with all possible consequences for middle-of-the-road decisions.

Example:

Dr. Ravi, a dentist schedules all his patients for 30 minute appointments. Some of the patients take more or less than 30 minutes depending on the type of dental work to be done. The following table shows the summary of the various categories of work, their probabilities and the time actually needed to complete the work.

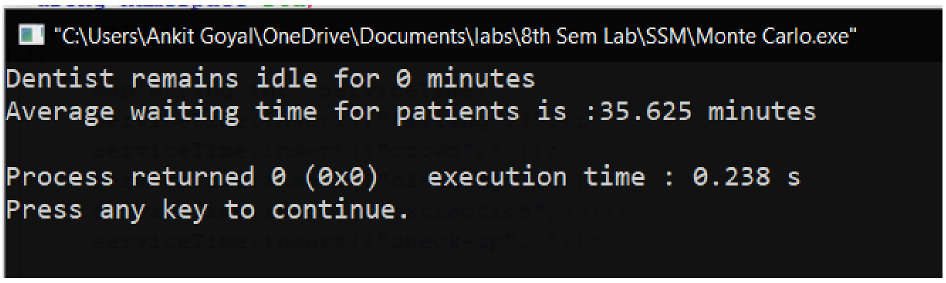
Category	Time Required(minutes)	No. of Patients
Filling	45	40
Crown	60	15
Cleaning	15	15
Extraction	45	10
Check-up	15	20

Simulate the dentist's clinic for four hours and determine the average waiting time for the patients as well as the idleness of the doctor. Assume that all the patients show up at the clinic exactly at their scheduled arrival time, starting at 8.00 am. Use the following random numbers for handling the above problem: 40, 82, 11, 34, 25, 66, 17, 79

Code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   unordered_map<string,int>
serviceTime={{"filling",45},{"crown",60},{"cleaning",15},{"extraction",45},{"check-
up",15}};
   vector<int> numbers{40,82,11,34,25,66,17,79};
   int curTime=0,maxTime=0;
   int idle=0;
   int wait=0;
   for(int i=0;i<8;i++)
   {
      if(curTime>maxTime)
      {
         idle+=(curTime-maxTime);
         maxTime=curTime;
      }
         wait+=(maxTime-curTime);
         curTime+=30;
         string type="";
         if(numbers[i]<40)
            type="filling";
         else if(numbers[i]<55)
            type="crown";
         else if(numbers[i]<70)
            type="cleaning";
         else if(numbers[i]<80)
            type="extraction";
         else
            type="check-up";
         maxTime+=serviceTime[type];
   }
   cout<<"Dentist remains idle for "<<idle<<" minutes\n";
   cout<<"Average waiting time for patients is :"<<((float)wait/8.0)<<" minutes\n";
   return 0;
}
```

Output:



```
"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\Monte Carlo.exe"
Dentist remains idle for 0 minutes
Average waiting time for patients is :35.625 minutes

Process returned 0 (0x0)   execution time : 0.238 s
Press any key to continue.
```

Experiment -5

AIM: Implement a program to perform Geometric and Poisson Distribution.

THEORY:

1. GEOMETRIC DISTRIBUTION:

The geometric distribution represents the number of failures before you get a success in a series of Bernoulli trials. This discrete probability distribution is represented by the probability density function:

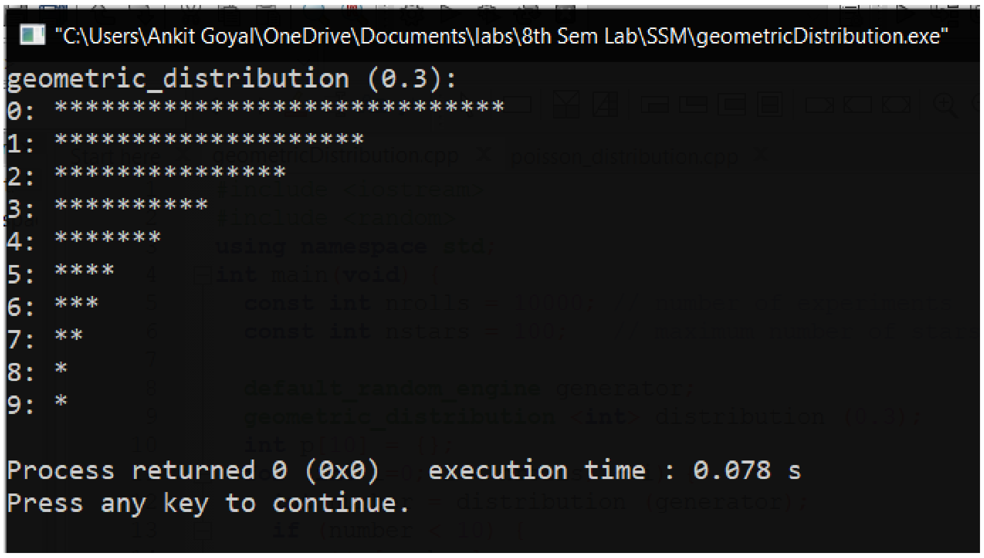
f(x) = (1 - p)x - 1p

For example, you ask people outside a polling station who they voted for until you find someone that voted for the independent candidate in a local election. The geometric distribution would represent the number of people who you had to poll before you found someone who voted independent. You would need to get a certain number of failures before you got your first success.

Program:

```
#include <iostream>
#include <random>
using namespace std;
int main(void) {
  const int nrolls = 10000; // number of experiments
  const int nstars = 100;   // maximum number of stars to distribute

  default_random_engine generator;
  geometric_distribution <int> distribution (0.3);
  int p[10] = {};
  for (int i=0; i < nrolls; ++i) {
    int number = distribution (generator);
    if (number < 10) {
                    ++p[number];
            }
  }
  cout << "geometric_distribution (0.3):" << endl;
  for (int i = 0; i < 10; ++i)
    cout << i << ": " << string(p[i] * nstars / nrolls, '*') << endl;
  return 0;
}
```

Output:



```
"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\geometricDistribution.exe"
geometric_distribution (0.3):
0: *****************************
1: ********************
2: ***************
3: **********
4: *******
5: ****
6: ***
7: **
8: *
9: *

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

2. POISSON DISTRIBUTION:

A Poisson distribution is a tool that helps to predict the probability of certain events from happening when you know how often the event has occurred. It gives us the probability of a given number of events happening in a fixed interval of time.
The Poisson Distribution pmf is:

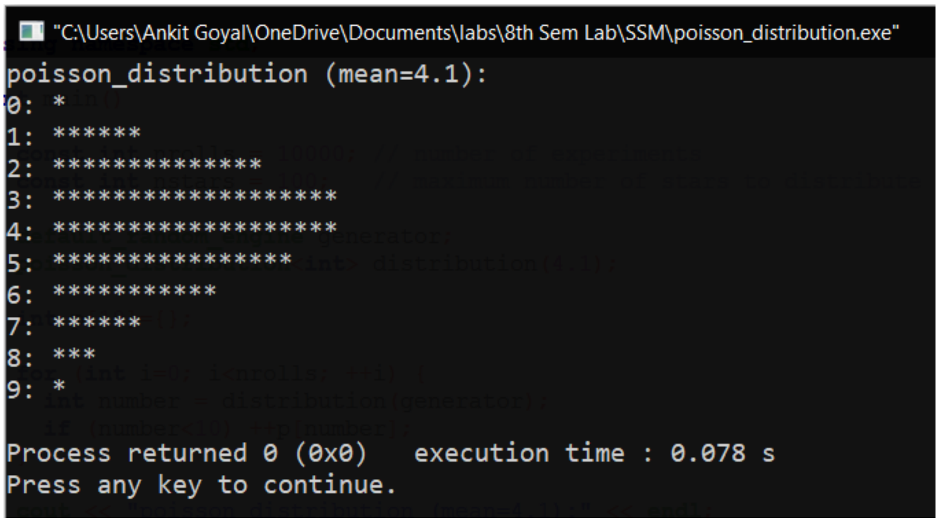
$$P(x; \mu) = (e-\mu * \mu x) / x!$$

Where:
- The symbol "!" is a factorial.
- μ (the expected number of occurrences) is sometimes written as λ. Sometimes called the **event rate** or rate parameter.

Program:

```
#include <iostream>
#include <random>

using namespace std;

int main()
{
  const int nrolls = 10000; // number of experiments
  const int nstars = 100;   // maximum number of stars to distribute

  default_random_engine generator;
  poisson_distribution<int> distribution(4.1);

  int p[10]={};

  for (int i=0; i<nrolls; ++i) {
    int number = distribution(generator);
    if (number<10) ++p[number];
  }

  cout << "poisson_distribution (mean=4.1):" << endl;
  for (int i=0; i<10; ++i)
    cout << i << ": " << string(p[i]*nstars/nrolls,'*') << endl;

  return 0;
}
```

Output:



"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\poisson_distribution.exe"

```
poisson_distribution (mean=4.1):
0: *
1: ******
2: **************
3: *******************
4: *******************
5: ****************
6: ***********
7: ******
8: ***
9: *

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

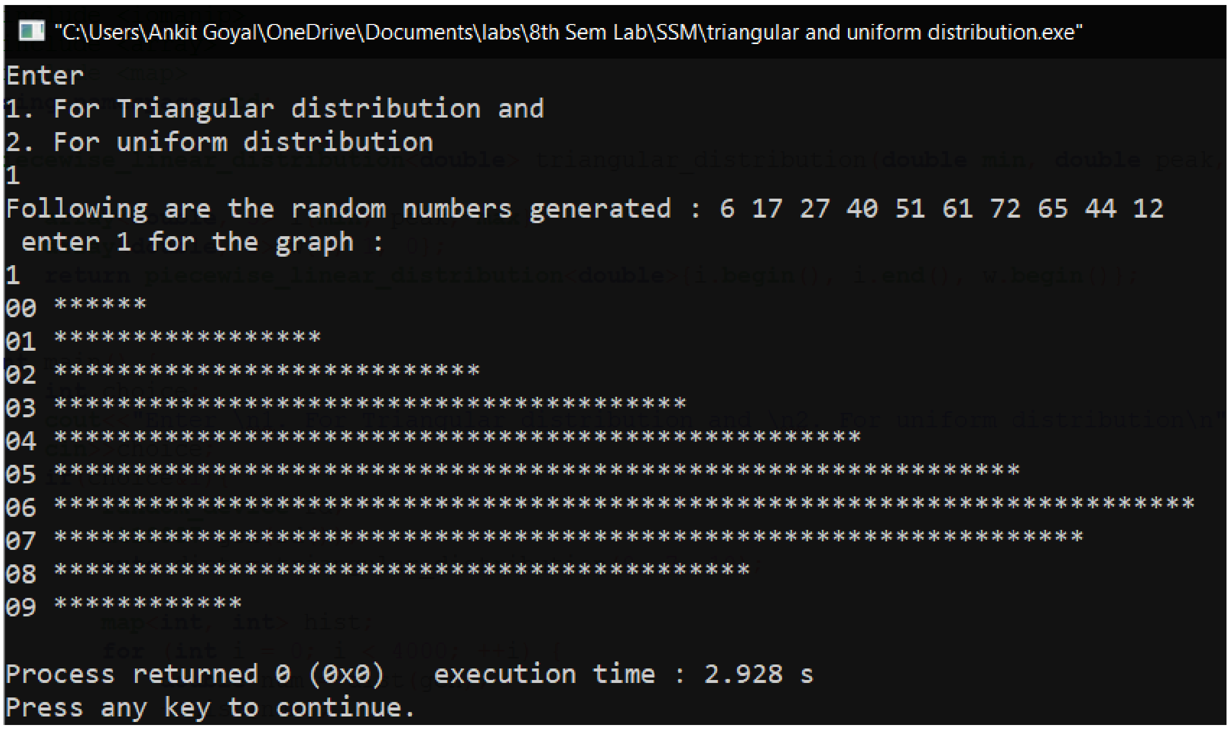
Experiment -6

Aim: Implement a menu driven program to generate random numbers using: a) Triangular distribution b) Uniform distribution.

Program:

```
#include <random>
#include <iostream>
#include <iomanip>
#include <array>
#include <map>
using namespace std;

piecewise_linear_distribution<double> triangular_distribution(double min, double peak,
double max)
{
   array<double, 3> i{min, peak, max};
   array<double, 3> w{0, 1, 0};
   return piecewise_linear_distribution<double>{i.begin(), i.end(), w.begin()};
}

int main() {
   int choice;
   cout<<"Enter \n1. For Triangular distribution and \n2. For uniform distribution\n";
   cin>>choice;
   if(choice&1){
      random_device rd;
      mt19937 gen(rd());
      auto dist = triangular_distribution(0, 7, 10);

      map<int, int> hist;
      for (int i = 0; i < 4000; ++i) {
         double num = dist(gen);
         ++hist[num];
      }
      cout<<"Following are the random numbers generated : ";
      for(auto p : hist) {
         cout << p.second/10<< " ";
      }
      cout<<"\n enter 1 for the graph : \n";
      int x;
      cin>>x;
      if(x&1){
         for(auto p : hist) {
            cout << setw(2) << setfill('0') << p.first << ' '
               << string(p.second/10,'*') << '\n';
         }
      }
   }
   else{
        const int nrolls=500;
        const int nstars=95;
        const int nintervals=10;

        default_random_engine generator;
        uniform_real_distribution<double> distribution(0.0,1.0);

        int p[nintervals]={};
        cout<<"Random number generated : ";
        for (int i=0; i<nrolls; ++i) {
          double number = distribution(generator);
          ++p[int(nintervals*number)];
          cout<<number<<" ";
        }
        cout<<endl;
        cout << "uniform_real_distribution (0.0,1.0):" << endl;
        cout << fixed; cout.precision(1);

        for (int i=0; i<nintervals; ++i) {
          cout << float(i)/nintervals << "-" << float(i+1)/nintervals << ": ";
          cout << string(p[i]*nstars/nrolls,'*') << endl;
        }

    }
}
```

Output:



```
"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\triangular and uniform distribution.exe"
Enter
1. For Triangular distribution and
2. For uniform distribution
1
Following are the random numbers generated : 6 17 27 40 51 61 72 65 44 12
 enter 1 for the graph :
1
00 ******
01 *****************
02 ***************************
03 ****************************************
04 ***************************************************
05 *************************************************************
06 ************************************************************************
07 *****************************************************************
08 ********************************************
09 ************

Process returned 0 (0x0)   execution time : 2.928 s
Press any key to continue.
```

Experiment -7

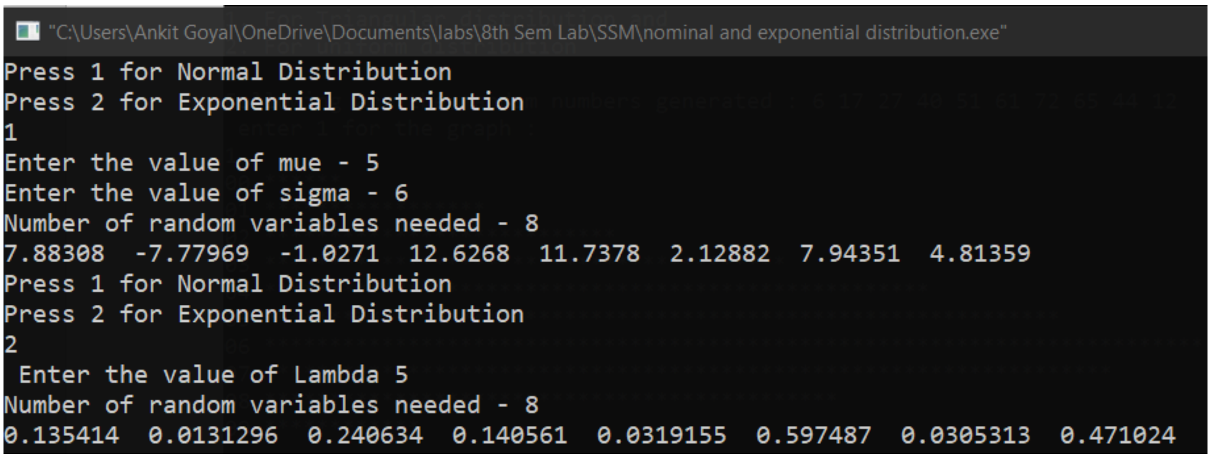
Aim: Implement a menu driven program to generate random numbers using: a) Nominal distribution b) Exponential distribution.

Program:

```
#include <bits/stdc++.h>
using namespace std;
void normal()
{
   int i, j, m, nn;
   float t, sum, x, mue, sigma;

   cout << "Enter the value of mue - ";
   cin >> mue;
   cout << "Enter the value of sigma - ";
   cin >> sigma;
   cout << "Number of random variables needed - ";
   cin >> nn;

   for (m = 1; m <= nn; m++)
   {
      sum = 0;
      for (i = 1; i <= 12; i++)
      {
         x = float(rand()) / float(RAND_MAX);
         sum = sum + x;
      }
      t = mue + sigma * (sum - 6.);
      cout << t << "  ";
   }
}
void expo()
{
   int i, j, k, m, nn;
   double lambda;
   cout << " Enter the value of Lambda ";
   cin >> lambda;
   cout << "Number of random variables needed - ";
   cin >> nn;
   for (m = 1; m <= nn; m++)
   {
      double u = float(rand()) / float(RAND_MAX);
      double x = log(1 - u) / (-lambda);
      cout << x << "  ";
   }
}
int main()
```

```
{
    int c;
    while (true)
    {
        cout << "Press 1 for Normal Distribution " << endl;
        cout << "Press 2 for Exponential Distribution " << endl;
        cin >> c;

        if (c == 1)
            normal();
        else
            expo();
        cout << endl;
    }
    return 0;
}
```

Output:



```
"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\nominal and exponential distribution.exe"
Press 1 for Normal Distribution
Press 2 for Exponential Distribution
1
Enter the value of mue - 5
Enter the value of sigma - 6
Number of random variables needed - 8
7.88308  -7.77969  -1.0271  12.6268  11.7378  2.12882  7.94351  4.81359
Press 1 for Normal Distribution
Press 2 for Exponential Distribution
2
 Enter the value of Lambda 5
Number of random variables needed - 8
0.135414  0.0131296  0.240634  0.140561  0.0319155  0.597487  0.0305313  0.471024
```

Experiment -8

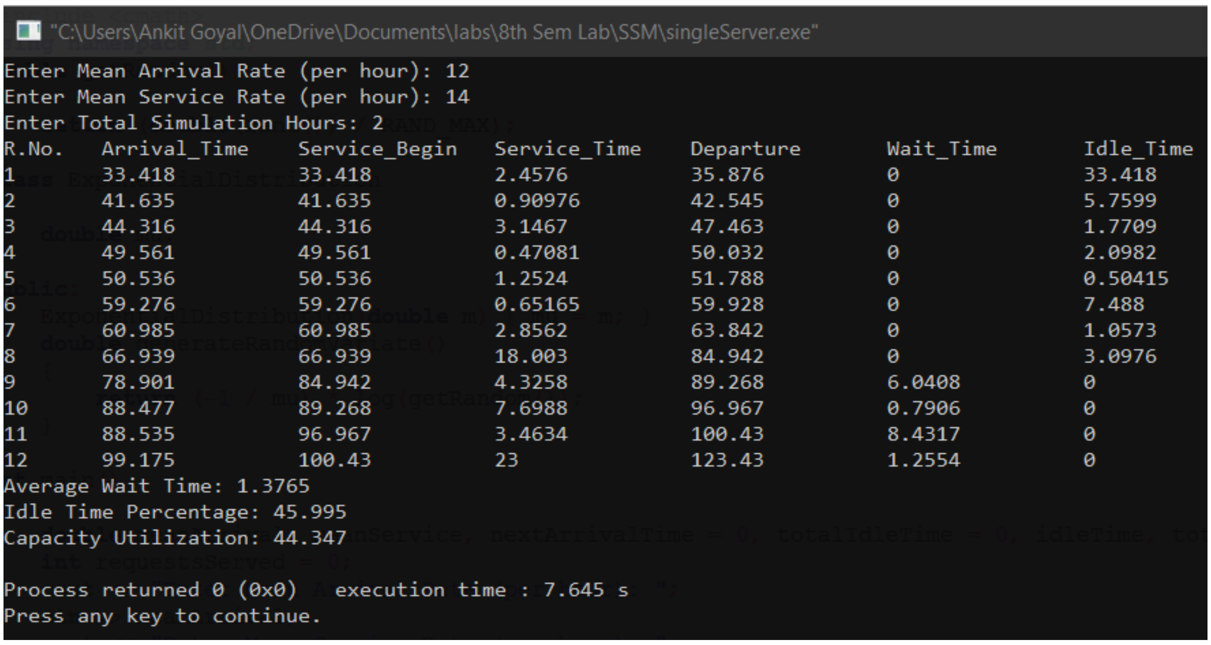
Aim: Implement a Single Server Queuing System.

CODE:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
double getRandom() {
   return (double(rand())/RAND_MAX);
}
class ExponentialDistribution {
   double mu;
public:
   ExponentialDistribution(double m) {mu=m;}
   double generateRandomVariate() {
      return (-1/mu)*log(getRandom());
   }
};
int main() {
   double meanArrival, meanService, nextArrivalTime=0, totalIdleTime=0, idleTime,
totalWaitTime=0, waitTime, nextDepartureTime=0, nextServiceBeginTime, service,
totalMinutes;
   int requestsServed=0;
   cout<<"Enter Mean Arrival Rate (per hour): "; cin>>meanArrival;
   cout<<"Enter Mean Service Rate (per hour): "; cin>>meanService;
   cout<<"Enter Total Simulation Hours: "; cin>>totalMinutes;
   totalMinutes=totalMinutes*60;
   ExponentialDistribution interArrivalTime(meanArrival/60), serviceTime(meanService/60);

cout<<"R.No.\tArrival_Time\tService_Begin\tService_Time\tDeparture\tWait_Time\tIdle_Ti
me"<<endl;
   while(nextDepartureTime<=totalMinutes)
   {
      nextArrivalTime+=interArrivalTime.generateRandomVariate();
      if(nextArrivalTime<=nextDepartureTime)
      {
         nextServiceBeginTime=nextDepartureTime;
         waitTime=nextDepartureTime-nextArrivalTime;
         totalWaitTime+=waitTime;
         idleTime=0;
      }
      else
      {
         nextServiceBeginTime=nextArrivalTime;
         idleTime=nextArrivalTime-nextDepartureTime;
         totalIdleTime+=idleTime;
         waitTime=0;
```

```
        }
        service=serviceTime.generateRandomVariate();
        nextDepartureTime=nextServiceBeginTime+service;
        ++requestsServed;

cout<<setprecision(5)<<requestsServed<<"\t"<<nextArrivalTime<<"\t\t"<<nextServiceBegi
nTime<<"\t\t"<<service<<"\t\t"<<nextDepartureTime<<"\t\t"<<waitTime<<"\t\t"<<idleTime
<<endl;
    }
    cout<<"Average Wait Time: "<<totalWaitTime/requestsServed<<endl;
    cout<<"Idle Time Percentage: "<<totalIdleTime/totalMinutes*100<<endl;
    cout<<"Capacity Utilization: "<<(nextArrivalTime-
totalIdleTime)/nextArrivalTime*100<<endl;
    return 0;
}
```

OUTPUT:


"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\singleServer.exe"

```
Enter Mean Arrival Rate (per hour): 12
Enter Mean Service Rate (per hour): 14
Enter Total Simulation Hours: 2
R.No.   Arrival_Time    Service_Begin   Service_Time    Departure       Wait_Time       Idle_Time
1       33.418          33.418          2.4576          35.876          0               33.418
2       41.635          41.635          0.90976         42.545          0               5.7599
3       44.316          44.316          3.1467          47.463          0               1.7709
4       49.561          49.561          0.47081         50.032          0               2.0982
5       50.536          50.536          1.2524          51.788          0               0.50415
6       59.276          59.276          0.65165         59.928          0               7.488
7       60.985          60.985          2.8562          63.842          0               1.0573
8       66.939          66.939          18.003          84.942          0               3.0976
9       78.901          84.942          4.3258          89.268          6.0408          0
10      88.477          89.268          7.6988          96.967          0.7906          0
11      88.535          96.967          3.4634          100.43          8.4317          0
12      99.175          100.43          23              123.43          1.2554          0
Average Wait Time: 1.3765
Idle Time Percentage: 45.995
Capacity Utilization: 44.347

Process returned 0 (0x0)   execution time : 7.645 s
Press any key to continue.
```

Experiment -9

Aim: Implement Two server(multi) queuing system.

Theory:

The system consists of multiple servers and a common queue for all items. When any item requests for the server, it is allocated if at-least one server is available. Else the queue begins to start until the server is free. In this system, we assume that all servers are identical, i.e. there is no difference which server is chosen for which item. There is an exception of utilization.

Multi server queue has two or more service facilities in parallel providing identical service. All the. customers in the waiting line can be served by more than one station. The arrival time and the service time. follow poisson and exponential distribution.

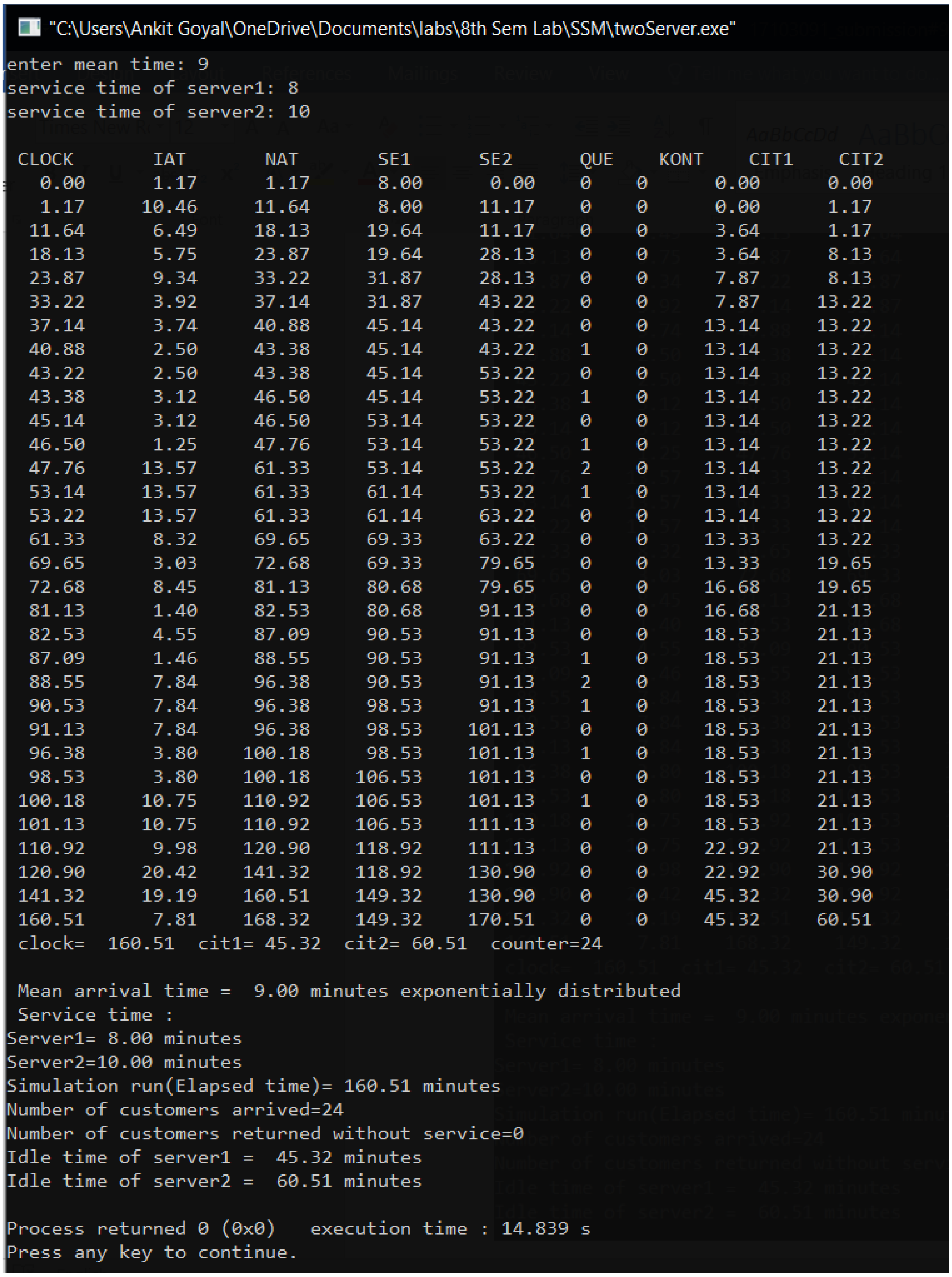
Program:

```
#include<bits/stdc++.h>
using namespace std;
constexpr int FLOAT_MIN = 0;
constexpr int FLOAT_MAX = 1;
int main()
{

  std::random_device rd;
   std::default_random_engine eng(rd());
   std::uniform_real_distribution<float> distr(FLOAT_MIN, FLOAT_MAX);
  float r,iat,clock=0,nat,it1,it2,run=150,cit1=0,cit2=0;
  float mean, lemda1, lemda2;
   cout<<"enter mean time: ";
   cin>>mean;
   cout<<"service time of server1: ";
   cin>>lemda1;
   cout<<"service time of server2: ";
   cin>>lemda2;
   float se1=0,se2=0;
  int k,q=0,qmax=3,kont=0,counter;
  printf("\n CLOCK      IAT      NAT      SE1      SE2      QUE    KONT    CIT1    CIT2");
  r=distr(eng);
  iat=(-mean)*log(1-r);
  nat=nat+iat;
  se1=lemda1;
  counter=1;

  printf("\n %6.2f    %6.2f    %6.2f    %6.2f    %6.2f    %d    %d    %6.2f    %6.2f
",clock,iat,nat,se1,se2,q,kont,cit1,cit2);
  while(clock<=run)
  {
     if(nat<=se1 && nat<=se2)
```

```
{
   clock=nat;
   q=q+1;
   r=distr(eng);
   iat = (-mean)*log(1-r);
   nat=nat+iat;
   counter=counter+1;
}
else if (se1<=nat && se1<=se2)
clock=se1;
else
   clock=se2;

   if(q>qmax)
   {
      kont=kont+1;
      q=q-1;
   }
   else if(q>=1 && se1<=clock)
   {
      it1=clock-se1;
      cit1=cit1+it1;
      se1=clock+lemda1;
      q=q-1;
   }

   else if(q>=1 && se2<=clock)
   {
      it2=clock-se2;
      cit2=cit2+it2;
      se2=clock+lemda2;
      q=q-1;
   }
   else if(q==0&& se1<=clock)
   {
      clock=nat;
      it1=clock-se1;
      cit1=cit1+it1;
      se1=nat+lemda1;
      r=distr(eng);
      iat=(-mean)*log(1-r);
      nat=nat+iat;
      counter=counter+1;
   }
   else if(q==0 && se2<=clock)
   {
      clock=nat;
      it2=clock-se2;
      cit2=cit2+it2;
      se2=nat+lemda2;
```

```
                r=distr(eng);
                iat=(-mean)*log(1-r);
                nat=nat+iat;
                counter=counter+1;
            }
              printf("\n %6.2f    %6.2f    %6.2f    %6.2f    %6.2f    %d    %d    %6.2f    %6.2f
",clock,iat,nat,se1,se2,q,kont,cit1,cit2);
    }

        printf("\n clock=%8.2f  cit1=%6.2f  cit2=%6.2f  counter=%d",clock,cit1,cit2,counter);
        printf("\n\n Mean arrival time = %5.2f minutes exponentially distributed",mean);
        printf("\n Service time : \nServer1=%5.2f minutes\nServer2=%5.2f
minutes",lemda1,lemda2);
        printf("\nSimulation run(Elapsed time)=%7.2f minutes",clock);
        printf("\nNumber of customers arrived=%d",counter);
        printf("\nNumber of customers returned without service=%d",kont);
        printf("\nIdle time of server1 = %6.2f minutes",cit1);
        printf("\nIdle time of server2 = %6.2f minutes\n",cit2);
}
```

OUTPUT:



"C:\Users\Ankit Goyal\OneDrive\Documents\labs\8th Sem Lab\SSM\twoServer.exe"

```
enter mean time: 9
service time of server1: 8
service time of server2: 10

 CLOCK       IAT       NAT       SE1       SE2      QUE   KONT     CIT1     CIT2
   0.00      1.17      1.17      8.00      0.00     0    0       0.00     0.00
   1.17     10.46     11.64      8.00     11.17     0    0       0.00     1.17
  11.64      6.49     18.13     19.64     11.17     0    0       3.64     1.17
  18.13      5.75     23.87     19.64     28.13     0    0       3.64     8.13
  23.87      9.34     33.22     31.87     28.13     0    0       7.87     8.13
  33.22      3.92     37.14     31.87     43.22     0    0       7.87    13.22
  37.14      3.74     40.88     45.14     43.22     0    0      13.14    13.22
  40.88      2.50     43.38     45.14     43.22     1    0      13.14    13.22
  43.22      2.50     43.38     45.14     53.22     0    0      13.14    13.22
  43.38      3.12     46.50     45.14     53.22     1    0      13.14    13.22
  45.14      3.12     46.50     53.14     53.22     0    0      13.14    13.22
  46.50      1.25     47.76     53.14     53.22     1    0      13.14    13.22
  47.76     13.57     61.33     53.14     53.22     2    0      13.14    13.22
  53.14     13.57     61.33     61.14     53.22     1    0      13.14    13.22
  53.22     13.57     61.33     61.14     63.22     0    0      13.14    13.22
  61.33      8.32     69.65     69.33     63.22     0    0      13.33    13.22
  69.65      3.03     72.68     69.33     79.65     0    0      13.33    19.65
  72.68      8.45     81.13     80.68     79.65     0    0      16.68    19.65
  81.13      1.40     82.53     80.68     91.13     0    0      16.68    21.13
  82.53      4.55     87.09     90.53     91.13     0    0      18.53    21.13
  87.09      1.46     88.55     90.53     91.13     1    0      18.53    21.13
  88.55      7.84     96.38     90.53     91.13     2    0      18.53    21.13
  90.53      7.84     96.38     98.53     91.13     1    0      18.53    21.13
  91.13      7.84     96.38     98.53    101.13     0    0      18.53    21.13
  96.38      3.80    100.18     98.53    101.13     1    0      18.53    21.13
  98.53      3.80    100.18    106.53    101.13     0    0      18.53    21.13
 100.18     10.75    110.92    106.53    101.13     1    0      18.53    21.13
 101.13     10.75    110.92    106.53    111.13     0    0      18.53    21.13
 110.92      9.98    120.90    118.92    111.13     0    0      22.92    21.13
 120.90     20.42    141.32    118.92    130.90     0    0      22.92    30.90
 141.32     19.19    160.51    149.32    130.90     0    0      45.32    30.90
 160.51      7.81    168.32    149.32    170.51     0    0      45.32    60.51
 clock=  160.51  cit1= 45.32  cit2= 60.51  counter=24

 Mean arrival time =  9.00 minutes exponentially distributed
 Service time :
Server1= 8.00 minutes
Server2=10.00 minutes
Simulation run(Elapsed time)= 160.51 minutes
Number of customers arrived=24
Number of customers returned without service=0
Idle time of server1 =  45.32 minutes
Idle time of server2 =  60.51 minutes

Process returned 0 (0x0)   execution time : 14.839 s
Press any key to continue.
```