# **INDEX**

|  |  |
| --- | --- |
| **Problems** | **Remarks** |
| 1. Simulate Linear Congruential Method for Random Number Generation |  |
| 1. Simulate KS Test |  |
| 1. Calculate the value of PI using Monte Carlo Method |  |
| 1. Simulate Barber Shop in GPSS |  |
| 1. Simulate M-M-1 Queuing System |  |

# **C Program of Linear Congruential Method**

#include<stdio.h>

#include<conio.h>

int main()

{

int xo,x1; /\*xo=seed, x1=next random number that we will generate \*/

int a,c,m; /\*a=constant multiplier, c=increment, m=modulus \*/

int i,n; /\*i for loopcontrol, n for how many random numbers \*/

int array[20]; /\*to store the random numbers generated \*/

printf("Enter the seed value xo: ");

scanf("%d",&xo);

printf("\n");

printf("Enter the constant multiplier a: ");

scanf("%d",&a);

printf("\n");

printf("Enter the increment c: ");

scanf("%d",&c);

printf("\n");

printf("Enter the modulus m: ");

scanf("%d",&m);

printf("\n");

printf("How many random numbers you want to generate: ");

scanf("%d",&n);

printf("\n");

for(i=0;i<n;i++) /\* loop to generate random numbers \*/

{

x1=(a\*xo+c) %m;

array[i]=x1;

xo=x1;

}

printf("The generated random numbers are: ");

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

{

printf("%d",array[i]);

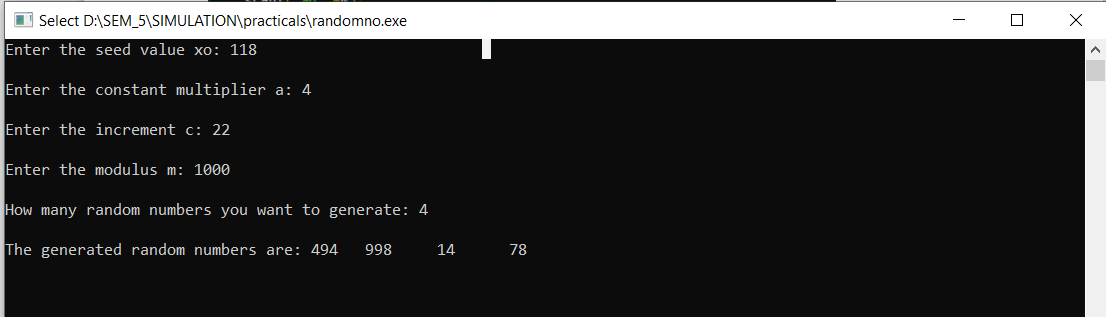
printf("\t");

}

getch();

return(0);

}



# **C++ Program of Kolmogorov Smirnovks (KS) Test**

#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?:"<<endl;

cin>>n;

cout<<"Enter "<<n<<" numbers"<<endl;

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

{

cout<<"Enter "<<i+1<<" number:"<<endl;

cin>>numbers[i];

}

}

float 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();

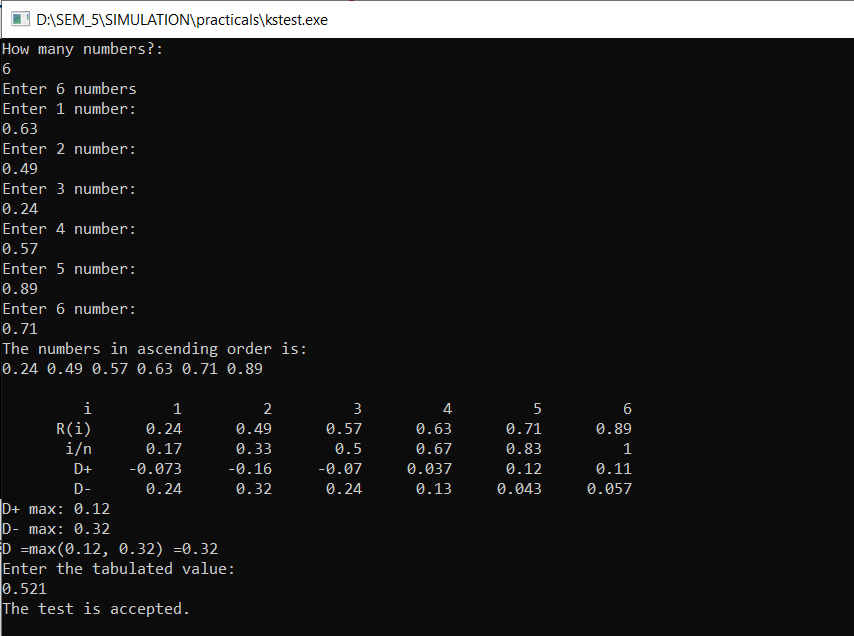
ks1.display();

ks1.conclusion();

getch();

return(0);

}



# **Estimate the value of PI using Monte Carlo Method**

import random

INTERVAL= 1000

circle\_points= 0

square\_points= 0

# Total Random numbers generated= possible x

# values\* possible y values

for i in range(INTERVAL\*\*2):

    # Randomly generated x and y values from a

    # uniform distribution

    # Range of x and y values is -1 to 1

    rand\_x= random.uniform(-1, 1)

    rand\_y= random.uniform(-1, 1)

    # Distance between (x, y) from the origin

    origin\_dist= rand\_x\*\*2 + rand\_y\*\*2

    # Checking if (x, y) lies inside the circle

    if origin\_dist<= 1:

        circle\_points+= 1

    square\_points+= 1

    # Estimating value of pi,

    # pi= 4\*(no. of points generated inside the

    # circle)/ (no. of points generated inside the square)

    pi = 4\* circle\_points/ square\_points

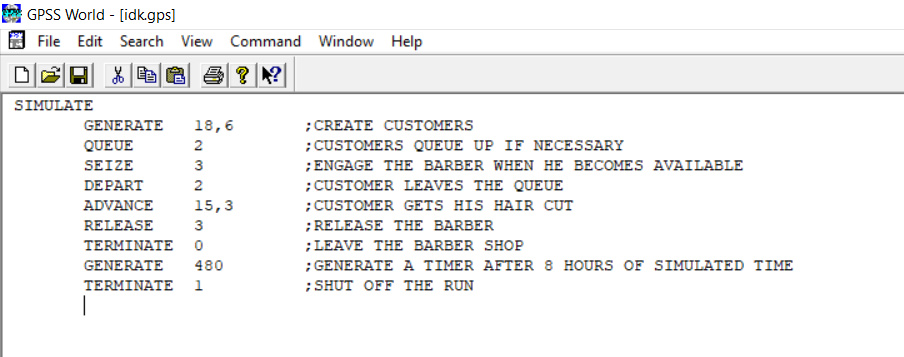
## print(rand\_x, rand\_y, circle\_points, square\_points, "-", pi)

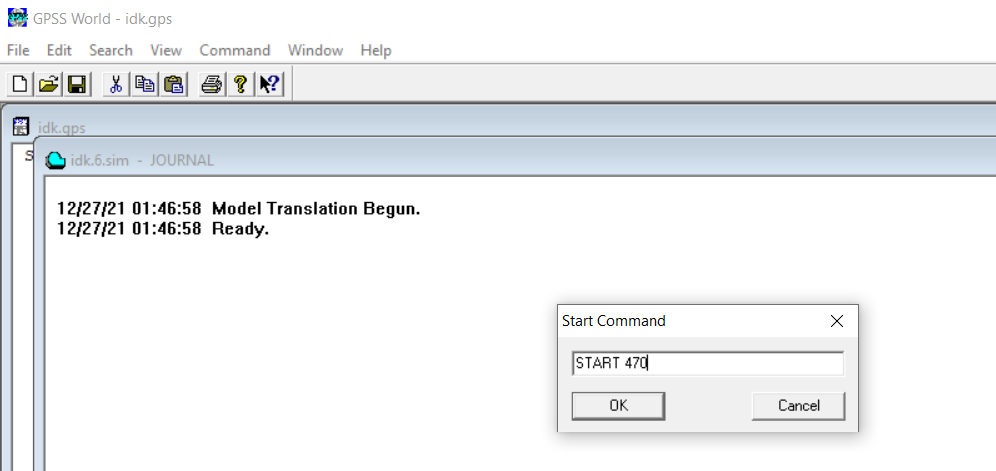
## print("\n")

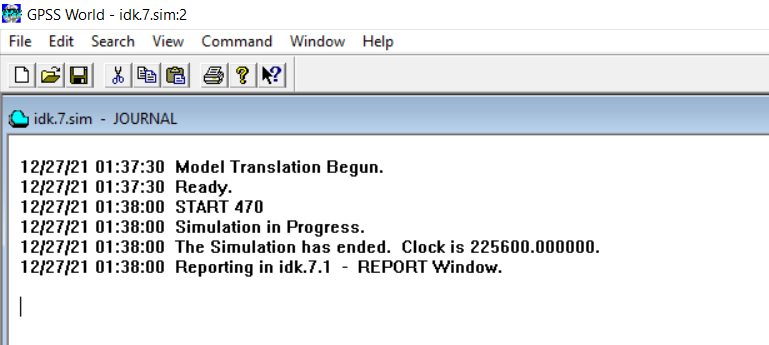
print("Final Estimation of Pi=", pi)

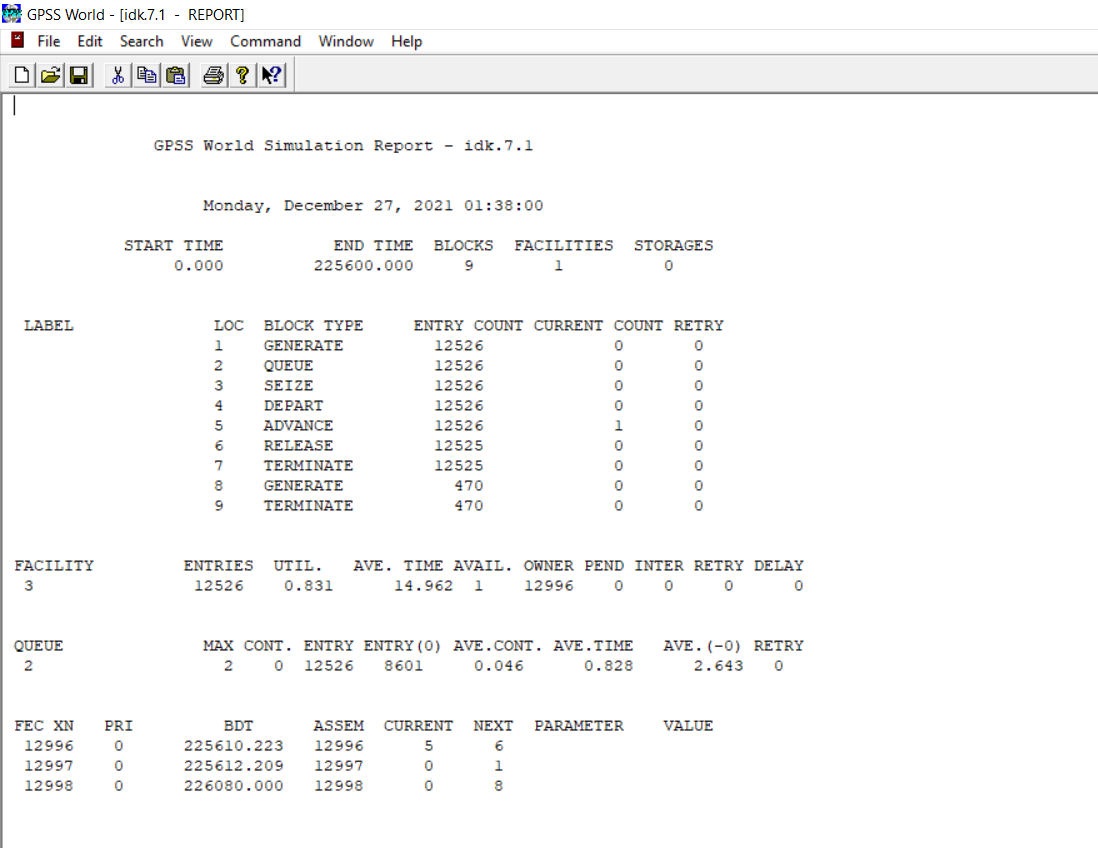


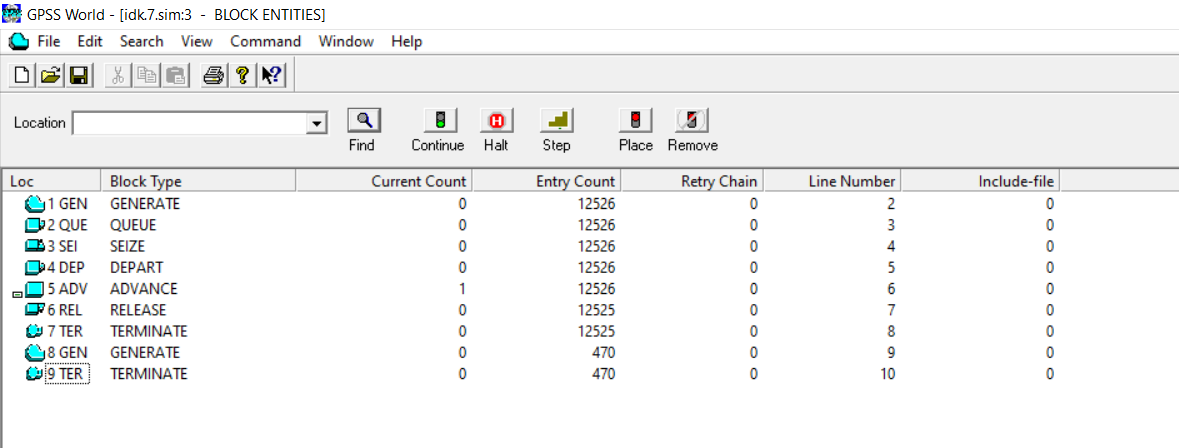
# **GPSS Barber Shop Simulation**











# **M-M-1 Queue Simulation**

import numpy as np

import queue

import copy

import matplotlib.pyplot as plt

# Input Parameters

total\_time = int(input("Enter time for simulation (Hours): "))

IAT\_rate = int(input("Enter Job Arrival Rate (/Hour): "))

ST\_rate = int(input("Enter Job Service Rate (/Hour): "))

rho = IAT\_rate/ST\_rate

# Initialize Parameters

qu = queue.Queue()

curr\_process = None

IAT = []

ST = []

AT = []

wait\_time = []

server\_busy = False

list\_wait = []

list\_delay = []

num\_processes = int(np.random.poisson(IAT\_rate)\*

total\_time)

num\_processes\_served = 0

# Populate Inter-Arrival-Times (IAT)

for i in range(num\_processes):

    temp = np.random.exponential(1/IAT\_rate)\*60\*60

    if i==0:

        IAT.append(0)

    else:

        IAT.append(int(temp - temp%1))

# Populate Service-Times (ST) (where ST[i]!=0)

while not len(ST) == num\_processes:

    temp = np.random.exponential(1/ST\_rate)\*60\*60

    if not int(temp- temp%1)<1:

        ST.append(int(temp - temp%1))

# Save a copy of ST

ST\_copy = copy.deepcopy(ST)

# Get Arrival-Times (AT) from IAT starting at t=0

# and initialize Waiting-Times to 0

for i in range(num\_processes):

    if i == 0:

        AT.append(0)

    else:

        AT.append(AT[i-1] + IAT[i])

    wait\_time.append(0)

# Simulation of M/M/1 Queue (i represents current time)

for i in range(total\_time\*60\*60):

    if server\_busy:

        for item in list(qu.queue):

            wait\_time[item] = wait\_time[item] + 1

        ST[curr\_process] = ST[curr\_process] - 1

        if ST[curr\_process] == 0:

            server\_busy = False

            num\_processes\_served = num\_processes\_served + 1

    for j in range(num\_processes):

        if i== AT[j]:

            qu.put(j)

    if not server\_busy and not qu.empty():

        curr\_process = qu.get()

        server\_busy = True

    sum\_wait = 0

    sum\_delay = 0

    for i in range(num\_processes\_served):

        sum\_wait = sum\_wait + wait\_time[i]

        sum\_delay = sum\_delay + wait\_time[i] + ST\_copy[i]

    if num\_processes\_served == 0:

        list\_wait.append(0)

        list\_delay.append(0)

    else:

        list\_wait.append(sum\_wait/(num\_processes\_served\*60\*60))

        list\_delay.append(sum\_delay/(num\_processes\_served\*60\*60))

plt.plot([i+1 for i in range(total\_time\*60\*60)], list\_wait)

plt.ylabel("Avg Wait Times")

plt.show()

plt.plot([i+1 for i in range(total\_time\*60\*60)], list\_delay)

plt.ylabel("Avg Delay Times")

plt.show()



