**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №4

з дисципліни «Моделювання та Аналіз Програмного Забезпечення» на тему:

«Використання розподілу ймовірності в системі імітаційного моделювання SIMC; Генератор випадкових чисел»

**Виконав:**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли:**

Викладач: Ж. К. Камінська

2024

Використання розподілу ймовірності в системі імітаційного моделювання SIMC; Генератор випадкових чисел

Мета роботи

Метою роботи є вивчення методів описання рівномірного та нерівномірного розподілу безперервних та дискретних випадкових величин в SIMC та способів їх практичного використання при моделюванні систем масового обслуговування

Результати виконання

Код програми

#include "../simc/simc.h"

#include <iostream>

using namespace std;

void one()

{

  auto Modeling\_Hours=40;

  auto Total\_Modeling\_Time=Modeling\_Hours\*60;

  auto Assembly\_Delay=30;

  auto Firing\_Delay=8;

  auto Worker\_Hourly\_Salary=50;

  auto Firing\_Daily\_Price=200;

  auto Material\_Price=2;

  auto Product\_Price=7;

  auto Workers\_Min=4;

  auto Workers\_Max=6;

  auto Best\_Result=0;

  auto Best\_Count=0;

  pqueue Assembly\_Queue;

  pstorage Assembly\_Facility;

  pqueue Firing\_Queue;

  pfacility Firing\_Facility;

  table Assembly\_Table;

  table Firing\_Table;

  Assembly\_Table.x[1] = 25;

  Assembly\_Table.p[1] = 0.01;

  Assembly\_Table.x[2] = 26;

  Assembly\_Table.p[2] = 0.04;

  Assembly\_Table.x[3] = 27;

  Assembly\_Table.p[3] = 0.09;

  Assembly\_Table.x[4] = 28;

  Assembly\_Table.p[4] = 0.19;

  Assembly\_Table.x[5] = 29;

  Assembly\_Table.p[5] = 0.37;

  Assembly\_Table.x[6] = 30;

  Assembly\_Table.p[6] = 0.63;

  Assembly\_Table.x[7] = 31;

  Assembly\_Table.p[7] = 0.81;

  Assembly\_Table.x[8] = 32;

  Assembly\_Table.p[8] = 0.91;

  Assembly\_Table.x[9] = 33;

  Assembly\_Table.p[9] = 0.96;

  Assembly\_Table.x[10] = 34;

  Assembly\_Table.p[10] = 0.99;

  Assembly\_Table.x[11] = 35;

  Assembly\_Table.p[11] = 1.0;

  Firing\_Table.x[1] = 6;

  Firing\_Table.p[1] = 0.05;

  Firing\_Table.x[2] = 7;

  Firing\_Table.p[2] = 0.3;

  Firing\_Table.x[3] = 8;

  Firing\_Table.p[3] = 0.7;

  Firing\_Table.x[4] = 9;

  Firing\_Table.p[4] = 0.95;

  Firing\_Table.x[5] = 10;

  Firing\_Table.p[5] = 1.0;

  initlist(Total\_Modeling\_Time);

  initcreate(1, 0);

  newqueue(Assembly\_Queue, "\"Assembly Queue\"");

  newqueue(Firing\_Queue, "\"Firing Queue\"");

  newfac(Firing\_Facility, "\"Firing Facility\"");

  for (auto Workers\_Count=Workers\_Min;Workers\_Count<=Workers\_Max;Workers\_Count++) {

    for (auto j=Workers\_Min;j<=Workers\_Count;j++) initcreate(1,0);

    newstorage(Assembly\_Facility, "\"Assembly Facility\"", 3);

    auto Parts\_Assembled=0;

    while (systime<Total\_Modeling\_Time) {

      plan();

      switch (sysevent) {

        case 1: inqueue(Assembly\_Queue); break;

        case 2: enter(Assembly\_Facility, 1); break;

        case 3: outqueue(Assembly\_Queue); break;

        case 4: delayt(randdtable(Assembly\_Table, v1)); break;

        case 5: leave(Assembly\_Facility, 1); break;

        case 6: inqueue(Firing\_Queue); break;

        case 7: seize(Firing\_Facility); break;

        case 8: outqueue(Firing\_Queue); break;

        case 9: delayt(randdtable(Firing\_Table, v1)); break;

        case 10: outfac(Firing\_Facility); Parts\_Assembled+=1; break;

        case 11: next(1); break;

      }

    }

    auto Workers\_Salary=Worker\_Hourly\_Salary\*Modeling\_Hours\*Workers\_Count;

    auto Firing\_Facility\_Cost=Firing\_Daily\_Price/8\*Modeling\_Hours;

    auto Materials\_Cost=Parts\_Assembled\*Material\_Price;

    auto Parts\_Cost=Parts\_Assembled\*Product\_Price;

    auto Total\_Expenses=Workers\_Salary+Firing\_Facility\_Cost+Materials\_Cost;

    auto Profit=Parts\_Cost-Total\_Expenses;

    if (abs(Profit)>abs(Best\_Result)) { Best\_Result=Profit; Best\_Count=Workers\_Count; }

    cout << "Workers: " << Workers\_Count << " Profit: " << Profit << endl;

  }

  cout << "\nBest Count: " << Best\_Count << " Best Result: " << Best\_Result << endl << endl;

  printall();

}

void two() {

  pfacility Packing\_Facility;

  // pack = 12x

  auto Packs\_Count=0;

  auto Modeling\_Time=1\*60\*60;

  initlist(Modeling\_Time);

  initcreate(1, 0);

  newfac(Packing\_Facility, "\"Packing Facility\"");

  while (systime < Modeling\_Time) {

    plan();

    switch (sysevent) {

      case 1: split(12, 2); break;

      case 2: seize(Packing\_Facility); break;

      case 3: delayt(randab(13,19,v1)); break;

      case 4: outfac(Packing\_Facility); break;

      case 5: assemble(12); break;

      case 6: Packs\_Count+=1; next(1); break;

    }

  }

  printall();

  clear();

  cout << "Total Packs: " << Packs\_Count << endl << endl;

}

void three()

{

  auto Modeling\_Time = 8 \* 60 \* 60;

  auto Maximum\_Queue = 0;

  auto Maximum\_Bags = 0;

  pfacility Cash\_Register;

  pstorage Store\_Baskets;

  pqueue Store\_Queue;

  initlist(Modeling\_Time);

  initcreate(1, 0);

  newfac(Cash\_Register, "\"Cash Register\"");

  newstorage(Store\_Baskets, "\"Store Rows\"", 100);

  newqueue(Store\_Queue, "\"Store Queue\"");

  while (systime < Modeling\_Time) {

    plan();

    switch (sysevent) {

      case 1: create(randexp(75, v1)); break;

      case 2: enter(Store\_Baskets, 1); break;

      case 3: if (rand01(v1) > 0.75) { trans->pi[1] += randab(2, 4, v1); delayt(randab(60, 180, v1)); } else next(4); break;

      case 4: if (rand01(v1) > 0.55) { trans->pi[1] += randab(3, 5, v1); delayt(randab(120, 180, v1)); } else next(5); break;

      case 5: if (rand01(v1) > 0.82) { trans->pi[1] += randab(4, 6, v1); delayt(randab(75, 165, v1)); } else next(6); break;

      case 6: inqueue(Store\_Queue); trans->pi[1] += randab(1,3,v1); break;

      case 7: seize(Cash\_Register); break;

      case 8: outqueue(Store\_Queue); break;

      case 9: delayt(3\*trans->pi[1]); break;

      case 10: outfac(Cash\_Register); break;

      case 11: leave(Store\_Baskets, 1); break;

      case 12: destroy(); break;

    }

    if (Store\_Baskets->sm > Maximum\_Bags) Maximum\_Bags = Store\_Baskets->sm;

    if (Store\_Queue->mq > Maximum\_Queue) Maximum\_Queue = Store\_Queue->mq;

  }

  cout << "Maximum Queue: " << Maximum\_Queue << endl

       << "Maximum Bags: " << Maximum\_Bags << endl << endl;

  printall();

  clear();

}

int main()

{

  one();

  two();

  three();

  return 0;

}

Виконання програми

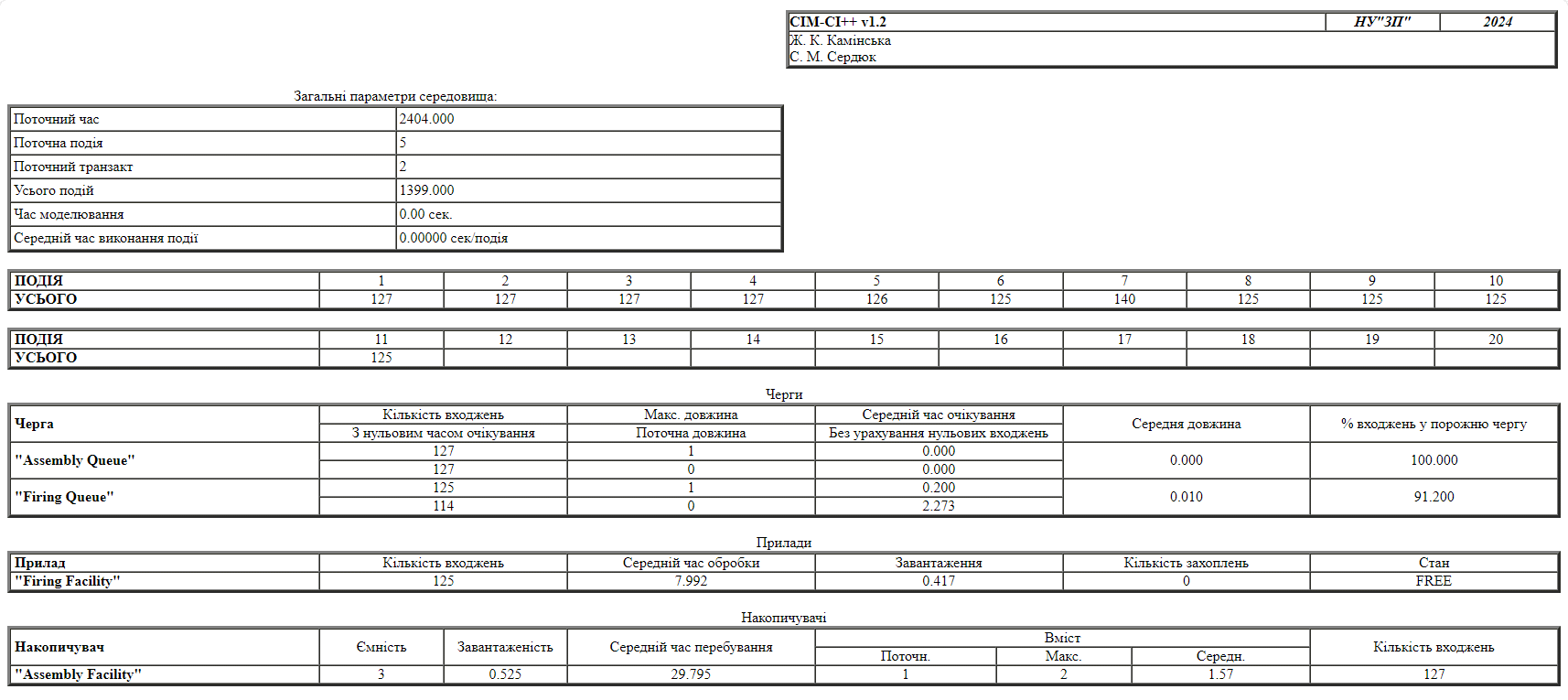


Рисунок 1.1 – Загальне завдання – браузер

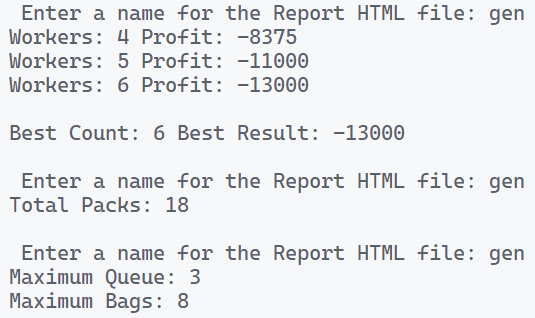


Рисунок 1.2 – Загальне завдання – консолька ♰

Висновки

Таким чином ми вивчили методи описання рівномірного та нерівномірного розподілу безперервних та дискретних випадкових величин в SIMC та способи їх практичного використання при моделюванні систем масового обслуговування

Контрольні питання

Генерація випадкових величин

В процесі моделювання коли потрібен генератор чисел то використовуємо такі функції:

rand01(v) - генерує числа рівномірно в інтервалі [0,1)

randab(a,b,v) - генерує числа рівномірно в інтервалі [a,b)

randexp(lambda,v) - генерує числа експоненційно з інтенсивністю lambda

randnorm(xmean,disp,v) - генерує числа за нормальним законом з середнім xmean і дисперсії disp

Процедури роботи з ансамблями в СІМ SIMC

Для роботи з ансамблями в SIMC існують такі процедури:

split(n,e) - де n то є число транзактів на створення, а e то є номер події де транзакти направляються

assemble(n) - де n то є число транзактів на збір

priority(p) - де p то є нове значення пріоритету

params(p) - параметр p застосовується до всіх транзактів

Реалізація безперервних та дискретних випадкових величин заданих в SIMC вигляді таблиці

Для безперервного розподілу є функція double randtable(table t, long v) де t то є таблиця а v то є число джерело

Для дискретного розподілу є функція double randdtable(table t, long v) де t то є таблиця а v то є число джерело