EECE 5643: Simulation and Performance Evaluation
Professor Ningfang Mi

Homework 3

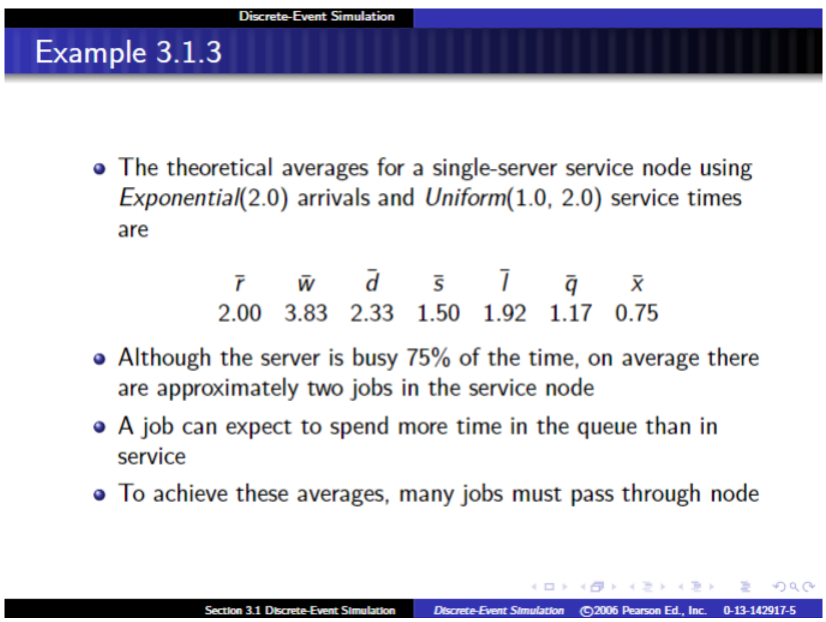
- Assignment Due: 02/16/2023 -

Harrison Sun
Monday, Thursday 11:45 am - 1:25 pm
Completed: February 16, 2023

1 Ex. 3.1.1

(a) Modify program ssq2 to use $Exponential(1.5)$ **service times.**
(b) Process a relatively large number of jobs, say 100000, and report what changes this produces relative to the statistics in Example 3.1.3.
(c) Explain (or conjecture) why some statistics change and others do not.

	123456	123456789	975312468	97531	246810
interarrival time	2.00	2.00	2.00	2.00	2.00
wait time	5.99	6.04	6.10	5.95	6.01
delay time	4.49	4.53	4.60	4.45	4.51
service time	1.50	1.50	1.50	1.50	1.51
number in node	3.00	3.02	3.05	2.97	3.00
number in queue	2.25	2.27	2.30	2.22	2.25
utilization	0.75	0.75	0.75	0.75	0.75



The average interarrival time ($\bar{r}$), average service time ($\bar{s}$), and utilization ($\bar{x}$) remain the same. The average wait time ($\bar{w}$), average delay time ($\bar{d}$), average number in the node ($\bar{l}$), and average number in the queue ($\bar{q}$) increase. This is because the service time distribution changes from a $Uniform(1.0, 2.0)$ to an $Exponential(1.5)$. While the mean stays the same, the average service time increases due to the significantly higher possible service times caused by the exponential distribution. Whenever this happens, the number in the queue (and thus, the number in the node) increases and propagates to higher wait times as the subsequent jobs arrive.

```
hlsun: Simulation-and-Perforr

(base) hlsun:Simulation-and-Performance-Evaluation$ conda deactivate
hlsun:Simulation-and-Performance-Evaluation$ make clean
rm Homework3.3 Homework3.4 Homework3.1 Homework3.2
hlsun:Simulation-and-Performance-Evaluation$ make
g++ Homework3.3.cpp c_lib/rng.c -o Homework3.3
g++ Homework3.4.cpp c_lib/rvgs.c c_lib/rngs.c -o Homework3.4
g++ Homework3.1.cpp c_lib/rng.c -o Homework3.1
g++ Homework3.2.cpp c_lib/rvgs.c c_lib/rngs.c -o Homework3.2
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.1 r 100000 s 123456

for 100000 jobs
   average interarrival time =   2.00
   average wait ........... =   5.99
   average delay .......... =   4.49
   average service time .... =   1.50
   average # in the node ... =   3.00
   average # in the queue .. =   2.25
   utilization ............ =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.1 r 100000 s 123456789

for 100000 jobs
   average interarrival time =   2.00
   average wait ........... =   6.04
   average delay .......... =   4.53
   average service time .... =   1.50
   average # in the node ... =   3.02
   average # in the queue .. =   2.27
   utilization ............ =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.1 r 100000 s 975312468

for 100000 jobs
   average interarrival time =   2.00
   average wait ........... =   6.10
   average delay .......... =   4.60
   average service time .... =   1.50
   average # in the node ... =   3.05
   average # in the queue .. =   2.30
   utilization ............ =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.1 r 100000 s 97531

for 100000 jobs
   average interarrival time =   2.00
   average wait ........... =   5.95
   average delay .......... =   4.45
   average service time .... =   1.50
   average # in the node ... =   2.97
   average # in the queue .. =   2.22
   utilization ............ =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.1 r 100000 s 246810

for 100000 jobs
   average interarrival time =   2.00
   average wait ........... =   6.01
   average delay .......... =   4.51
   average service time .... =   1.51
   average # in the node ... =   3.00
   average # in the queue .. =   2.25
   utilization ............ =   0.75
hlsun:Simulation-and-Performance-Evaluation$
```

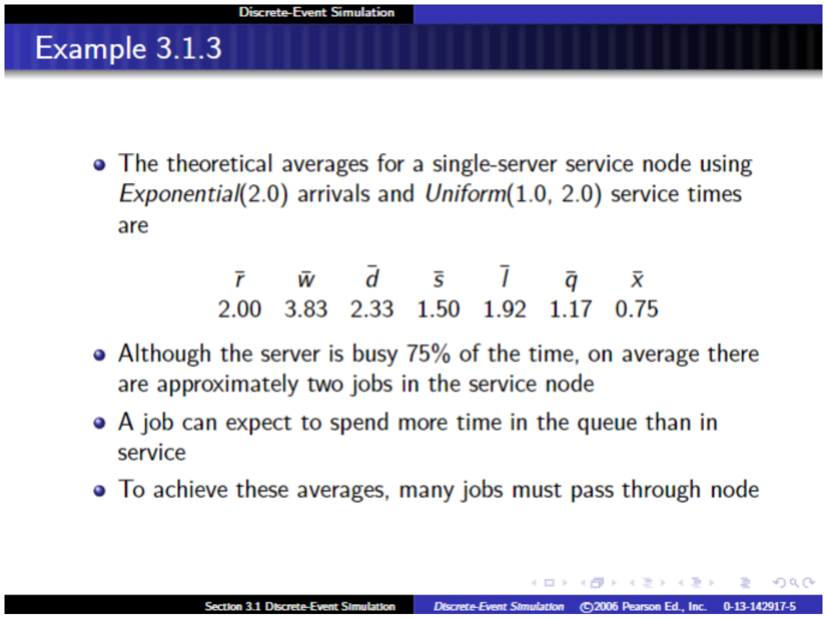
```c
/**
 * Modified by Harrison Sun
 * sun.har@northeastern.edu
 * February 11, 2023
 */

/* -------------------------------------------------------------------------
 * This program - an extension of program ssq1.c - simulates a single-server
 * FIFO service node using Exponentially distributed interarrival times and
 * Uniformly distributed service times (i.e. a M/U/1 queue).
 *
 * Name              : ssq2.c  (Single Server Queue, version 2)
 * Author            : Steve Park & Dave Geyer
 * Language          : ANSI C
 * Latest Revision   : 9-11-98
 * -------------------------------------------------------------------------
 */

#include <exception>
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <stdio.h>
#include <string>
#include <math.h>
#include "c_lib/rng.h"

#define LAST         10000L                   /* number of jobs processed */
#define START        0.0                      /* initial time             */


double Exponential(double m)
/* ---------------------------------------------------
 * generate an Exponential random variate, use m > 0.0
 * ---------------------------------------------------
 */
{
    return (-m * log(1.0 - Random()));
}


double Uniform(double a, double b)
/* --------------------------------------------
 * generate a Uniform random variate, use a < b
 * --------------------------------------------
 */
{
    return (a + (b - a) * Random());
}


double GetArrival(void)
/* ---------------------------
 * generate the next arrival time
 * ---------------------------
 */
{
    static double arrival = START;

    arrival += Exponential(2.0);
    return (arrival);
}


//double GetService(void)
```

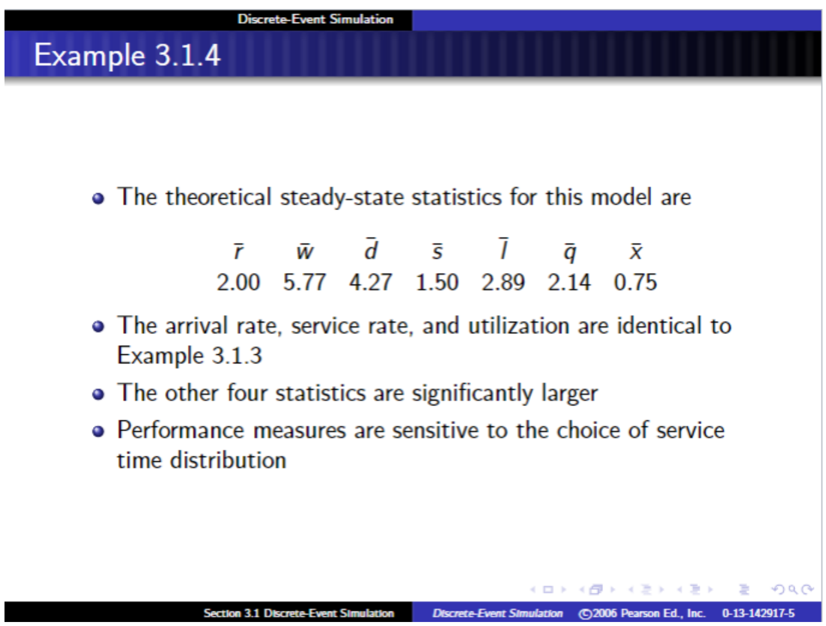
```
///* ------------------------------------
// * generate the next service time
// * ------------------------------------
// */
//{
//    return (Uniform(1.0, 2.0));
//}

/* Changing the GetService(void) function to return Exponential(1.5) service times */
double GetService(void)
{
    /* Generate the next service time */

    return (Exponential(1.5));
}

/* function to check if the input is a number */
bool checkArg(char* input)
{
    try
    {
        if (strlen(input) > 9)
        {
            throw std::logic_error("Number is too large.");
        }

        for (int i = 0; i < strlen(input); ++i)
        {

            if (std::isdigit(input[i])) continue;
            else
            {
                std::string errorMessage;
                errorMessage.append((std::string)input);
                errorMessage.append(" is not a digit.");
                throw std::logic_error(errorMessage);
            }
        }
        return 1;
    }

    catch (const std::logic_error& error)
    {
        std::cerr << error.what() << std::endl;
        return 0;
    }
}

int main(int argc, char* argv[])
{
    long   index = 0;                          /* job index            */
    double arrival = START;                     /* time of arrival      */
    double delay;                                /* delay in queue       */
    double service;                              /* service time         */
    double wait;                                 /* delay + service      */
    double departure = START;                    /* time of departure    */
    struct {                                     /* sum of ...           */
        double delay;                              /*   delay times        */
        double wait;                               /*   wait times         */
        double service;                            /*   service times      */
        double interarrival;                       /*   interarrival times */
    } sum = { 0.0, 0.0, 0.0 };

    long numRuns{};                                 /* number of runs */

```

```
    // Set the seed
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 's' && checkArg(argv[i + 1]))
        {
            PutSeed(std::stol(argv[i + 1]));
            break;
        }
        else
        {
            PutSeed(123456789);
        }
    }

    // Set the number of runs
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'r' && checkArg(argv[i + 1]))
        {
            numRuns = std::stol(argv[i + 1]);
            break;
        }
        else
        {
            numRuns = 10000;
        }
    }

    while (index < numRuns) {
        index++;
        arrival = GetArrival();
        if (arrival < departure)
            delay = departure - arrival;         /* delay in queue    */
        else
            delay = 0.0;                         /* no delay          */
        service = GetService();
        wait = delay + service;
        departure = arrival + wait;              /* time of departure */
        sum.delay += delay;
        sum.wait += wait;
        sum.service += service;
    }
    sum.interarrival = arrival - START;

    printf("\nfor %ld jobs\n", index);
    printf("   average interarrival time = %6.2f\n", sum.interarrival / index);
    printf("   average wait ............ = %6.2f\n", sum.wait / index);
    printf("   average delay ........... = %6.2f\n", sum.delay / index);
    printf("   average service time .... = %6.2f\n", sum.service / index);
    printf("   average # in the node ... = %6.2f\n", sum.wait / departure);
    printf("   average # in the queue .. = %6.2f\n", sum.delay / departure);
    printf("   utilization ............. = %6.2f\n", sum.service / departure);

    return (0);
}
```

2 Ex. 3.1.5

(a) Verify that the mean service time in Example 3.1.4 is 1.5.
(b) Verify that the steady-state statistics in Example 3.1.4 seem to be correct.
(c) Note that the arrival rate, service rate, and utilization are the same as those in Example 3.1.3. Explain (or conjecture) why this is so. Be Specific.





```
hlsun:Simulation-and-Performance-Evaluation$ make clean
rm Homework3.3 Homework3.4 Homework3.1 Homework3.2
hlsun:Simulation-and-Performance-Evaluation$ make
g++ Homework3.3.cpp c_lib/rng.c -o Homework3.3
g++ Homework3.4.cpp c_lib/rvgs.c c_lib/rngs.c -o Homework3.4
g++ Homework3.1.cpp c_lib/rng.c -o Homework3.1
g++ Homework3.2.cpp c_lib/rvgs.c c_lib/rngs.c -o Homework3.2
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.2 r 100000 s 123456

for 100000 jobs
   average interarrival time =   2.00
   average wait ............ =   5.61
   average delay ........... =   4.10
   average service time .... =   1.50
   average # in the node ... =   2.80
   average # in the queue .. =   2.05
   utilization ............. =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.2 r 100000 s 246810

for 100000 jobs
   average interarrival time =   2.00
   average wait ............ =   5.82
   average delay ........... =   4.31
   average service time .... =   1.50
   average # in the node ... =   2.90
   average # in the queue .. =   2.15
   utilization ............. =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.2 r 100000 s 1357911

for 100000 jobs
   average interarrival time =   2.00
   average wait ............ =   5.58
   average delay ........... =   4.09
   average service time .... =   1.50
   average # in the node ... =   2.79
   average # in the queue .. =   2.04
   utilization ............. =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.2 r 100000 s 987654321

for 100000 jobs
   average interarrival time =   2.00
   average wait ............ =   5.73
   average delay ........... =   4.23
   average service time .... =   1.50
   average # in the node ... =   2.86
   average # in the queue .. =   2.11
   utilization ............. =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.2 r 100000 s 12357

for 100000 jobs
   average interarrival time =   2.00
   average wait ............ =   5.66
   average delay ........... =   4.16
   average service time .... =   1.49
   average # in the node ... =   2.82
   average # in the queue .. =   2.08
   utilization ............. =   0.75
hlsun:Simulation-and-Performance-Evaluation$
```

	123456	246810	1357911	987654321	12357
interarrival time	2.00	2.00	2.00	2.00	2.00
wait time	5.61	5.82	5.58	5.73	5.66
delay time	4.10	4.31	4.09	4.23	4.16
service time	1.50	1.50	1.50	1.50	1.49
number in node	2.80	2.90	2.79	2.86	2.82
number in queue	2.05	2.15	2.04	2.11	2.08
utilization	0.75	0.75	0.75	0.75	0.75

The arrival rate is the same because the interarrival time is defined by the same distribution as in Example 3.1.3: $r \sim Exponential(2.0)$.

The average service rate is the same as in Example 3.1.3 due to the distribution of the service times. The average number of tasks $\bar{t}$ of $t \sim 1 + Geometric(0.9)$ is 1 plus the inverse of $p = 0.9$. Therefore, the average number of tasks is 10. The average of $Uniform(0.1, 0.2)$ is 0.15. Multiplying this and the number of tasks together results in an average service rate $\bar{s} = 1.5$, which is the same as the service rate in Example 3.1.3.

The server utilization is the same as in Example 3.1.3 because the average interarrival times and average service rates are the same in both examples. The server utilization is a ratio of the interarrival and service time averages.

```
/**
 * Homework 3.2
 * EECE 5643 - Simulation and Performance Evaluation
 * Author: Harrison Sun
 * Email: sun.har@northeastern.edu
 */

#include <cstdlib>
#include <cstring>
#include <stdio.h>
#include <exception>
#include <iostream>
#include <math.h>
#include <string>
#include "c_lib/rvgs.h"
#include "c_lib/rngs.h"

#define LAST         10000L                   /* number of jobs processed */
#define START        0.0                      /* initial time             */

 /**
  * double GetArrival()
  *
  * @param void
  * @return arrival - the next arrival time
  *
  * This function calculates the arrival times for each process.
  */

double GetArrival()
{
    static double arrival = START;

    arrival += Exponential(2.0);
    return (arrival);
}


/**
 * double GetService()
 *
 * @param void
 * @return sum - the total service time for the process
 *
 * This function calculates the service times for each process.
 */

double GetService()
{
    long k{};
    double sum{ 0.0 };
    long tasks = 1 + Geometric(0.9);
    for (k = 0; k < tasks; ++k)
    {
        sum += Uniform(0.1, 0.2);
    }
    return sum;
}

/**
 * bool checkArg()
 *
 * @param char* input - the input string literal from the console
 * @return bool - true if the input is a number, false otherwise
 *
```

```
 * This function determines whether the argument is a number.
 */

bool checkArg(char* input)
{
    try
    {
        if (strlen(input) > 9)
        {
            throw std::logic_error("Number is too large.");
        }

        for (int i = 0; i < strlen(input); ++i)
        {

            if (std::isdigit(input[i])) continue;
            else
            {
                std::string errorMessage;
                errorMessage.append((std::string)input);
                errorMessage.append(" is not a digit.");
                throw std::logic_error(errorMessage);
            }
        }
        return 1;
    }

    catch (const std::logic_error& error)
    {
        std::cerr << error.what() << std::endl;
        return 0;
    }
}

/**
 * int main()
 *
 * @param int argc - the number of arguments
 * @param char* argv[] - the arguments
 *
 * @return int - 0 if the program runs successfully
 */

int main(int argc, char* argv[])
{
    long   index = 0;                        /* job index            */
    double arrival = START;                  /* time of arrival      */
    double delay;                            /* delay in queue       */
    double service;                          /* service time         */
    double wait;                             /* delay + service      */
    double departure = START;                /* time of departure    */
    struct {                                 /* sum of ...           */
        double delay;                        /*   delay times        */
        double wait;                         /*   wait times         */
        double service;                      /*   service times      */
        double interarrival;                 /*   interarrival times */
    } sum = { 0.0, 0.0, 0.0 };

    long numRuns{};                          /* number of runs */

    // Set the seed
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 's' && checkArg(argv[i + 1]))
        {
```

```
            PutSeed(std::stol(argv[i + 1]));
            break;
        }
        else
        {
            PutSeed(123456789);
        }
    }

    // Set the number of runs
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'r' && checkArg(argv[i + 1]))
        {
            numRuns = std::stol(argv[i + 1]);
            break;
        }
        else
        {
            numRuns = 10000;
        }
    }

    while (index < numRuns) {
        index++;
        arrival = GetArrival();
        if (arrival < departure)
            delay = departure - arrival;        /* delay in queue    */
        else
            delay = 0.0;                        /* no delay          */
        service = GetService();
        wait = delay + service;
        departure = arrival + wait;             /* time of departure */
        sum.delay += delay;
        sum.wait += wait;
        sum.service += service;
    }
    sum.interarrival = arrival - START;

    printf("\nfor %ld jobs\n", index);
    printf("   average interarrival time = %6.2f\n", sum.interarrival / index);
    printf("   average wait ............ = %6.2f\n", sum.wait / index);
    printf("   average delay ........... = %6.2f\n", sum.delay / index);
    printf("   average service time .... = %6.2f\n", sum.service / index);
    printf("   average # in the node ... = %6.2f\n", sum.wait / departure);
    printf("   average # in the queue .. = %6.2f\n", sum.delay / departure);
    printf("   utilization ............. = %6.2f\n", sum.service / departure);
    return 0;
}
```

3 Ex. 3.3.1

Let β be the probability of feedback and let the integer-valued random variable $\mathcal{X}$ be the number of times a job feeds back.

(a) For x = 0,1,2,... what is $Pr(\mathcal{X} = x)$?

The probability of feedback $Pr(\mathcal{X} = x)$ is equal to $\beta^x \times (1 - \beta)$. That is, the probability of each feedback is defined by β and the single successful job completion is defined by $1 - \beta$.

(b) How does this relate to the discussion of acceptance/rejection in Section 2.3 (i.e., Example 2.3.8)?

The probability of feedback is analogous to rejection of the job. Therefore, it can be modeled similarly to the acceptance/rejection model with the acceptance criteria encompassing $1 - \beta$ of the feature space and β encompassing the remainder (outside of the acceptance area) of the feature space. In this case, a two dimensional feature space is not required and can be flattened to a single line.

```
/**
 * Homework 3.3
 * EECE 5643 - Simulation and Performance Evaluation
 * Author: Harrison Sun
 * Email: sun.har@northeastern.edu
 */

#define DEFAULT_BETA 0.9        // default value for beta
#define DEFAULT_RUNS 100000L    // default value for number of runs

#include <cstdlib>
#include <cstring>
#include <stdio.h>
#include <exception>
#include <iostream>
#include <math.h>
#include <string>
#include <vector>
#include "c_lib/rng.h"

/**
 * bool checkArg()
 *
 * @param char* input - the input string literal from the console
 * @return bool - true if the input is a number, false otherwise
 *
 * This function determines whether the argument is a number.
 */

bool checkArg(char* input)
{
    try
    {
        if (strlen(input) > 9)
        {
            throw std::logic_error("Number is too large.");
        }

        for (int i = 0; i < strlen(input); ++i)
        {

            if (std::isdigit(input[i]) || (input[i] == '.')) continue;
            else
            {
                std::string errorMessage;
                errorMessage.append((std::string)input);
                errorMessage.append(" is not a number.");
                throw std::logic_error(errorMessage);
            }
        }
        return 1;
    }

    catch (const std::logic_error& error)
    {
        std::cerr << error.what() << std::endl;
        return 0;
    }
}

/**
 * int main()
 *
 * @param int argc - the number of arguments
 * @param char* argv[] - the arguments
```

```
 *
 * @return int - returns 0 if the program runs successfully
 */

int main(int argc, char* argv[])
{
    long numRuns{};
    long Beta{};
  std::vector<int> feedback;

    // Set the seed
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 's' && checkArg(argv[i + 1]))
        {
            PutSeed(std::stol(argv[i + 1]));
            break;
        }
        else
        {
            PutSeed(123456789);
        }
    }

    // Set the number of runs
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'r' && checkArg(argv[i + 1]))
        {
            numRuns = std::stol(argv[i + 1]);
            break;
        }
        else
        {
            numRuns = DEFAULT_RUNS;
        }
    }

    // Set the number of runs
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'B' && checkArg(argv[i + 1]))
        {
            Beta = std::stol(argv[i + 1]);
            break;
        }
        else
        {
            Beta = DEFAULT_BETA;
        }
    }

    for (int i = 0; i < numRuns; ++i)
    {
        int numFeedback{ 0 };
        double random = Random();
        /* Test if feedback */
        while (random > Beta)
        {
            numFeedback++;
            random = Random();
        }
    if (numFeedback < feedback.size())
            feedback[numFeedback]++;
        else
```

```
          {
        feedback.resize(numFeedback + 1);
        feedback[numFeedback]++;
          }
    }
    std::cout << feedback.size() << std::endl;
    for (int i = 0; i < feedback.size(); ++i)
    {
    feedback[i] /= numRuns;
    std::cout << "Probability of " << i << " feedbacks: " << feedback[i] << std::endl;
    }

  return 0;
}
```

4 Ex. 3.3.4

Modify program ssq2 to account for a finite queue capacity.

(a) For the queue capacities 1,2,3,4,5, and 6, construct a table of the estimated steady-state probability of rejection.

Uniform(1.0, 2.0)	123456	246810	97531
Rejection Q = 1	18.53	18.45	18.19
Rejection Q = 2	8.98	9.05	8.78
Rejection Q = 3	4.77	4.76	4.55
Rejection Q = 4	2.65	2.61	2.51
Rejection Q = 5	1.45	1.52	1.35
Rejection Q = 6	0.83	0.85	0.74

(b) Also, construct a similar table if the service-time distribution is changed to be $Uniform(1.0, 3.0)$.

Uniform(1.0, 3.0)	123456	246810	97531
Rejection Q = 1	27.66	27.58	27.24
Rejection Q = 2	18.45	18.30	18.11
Rejection Q = 3	13.82	13.69	13.41
Rejection Q = 4	10.93	10.77	10.55
Rejection Q = 5	9.05	8.86	8.60
Rejection Q = 6	7.74	7.55	7.43

(c) Comment on how the probability of rejection depends on the service process.

The probability of rejection increases when the service time increases. This makes sense, as a higher service time allows the queue to build up, particularly when the interarrival rate is faster than the service time.

(d) How did you convince yourself these tables are correct?

This makes sense, as the increased service time increases the rejection rate. Additionally, a longer queue makes the rejection rate decrease, as the service node can handle multiple jobs arriving in a short period of time given that the queue has space.

```
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 1 u 1.0 2.0
   percent of rejections.... =  18.53
   utilization ............ =   0.61
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 2 u 1.0 2.0
   percent of rejections.... =   8.98
   utilization ............ =   0.68
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 3 u 1.0 2.0
   percent of rejections.... =   4.77
   utilization ............ =   0.72
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 4 u 1.0 2.0
   percent of rejections.... =   2.65
   utilization ............ =   0.73
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 5 u 1.0 2.0
   percent of rejections.... =   1.45
   utilization ............ =   0.74
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 6 u 1.0 2.0
   percent of rejections.... =   0.83
   utilization ............ =   0.75
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 1 u 1.0 2.0
   percent of rejections.... =  18.45
   utilization ............ =   0.61
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 2 u 1.0 2.0
   percent of rejections.... =   9.05
   utilization ............ =   0.68
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 3 u 1.0 2.0
   percent of rejections.... =   4.76
   utilization ............ =   0.71
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 4 u 1.0 2.0
   percent of rejections.... =   2.61
   utilization ............ =   0.73
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 5 u 1.0 2.0
   percent of rejections.... =   1.52
   utilization ............ =   0.74
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 6 u 1.0 2.0
   percent of rejections.... =   0.85
   utilization ............ =   0.74
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 1 u 1.0 2.0
   percent of rejections.... =  18.19
   utilization ............ =   0.61
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 2 u 1.0 2.0
   percent of rejections.... =   8.78
   utilization ............ =   0.68
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 3 u 1.0 2.0
   percent of rejections.... =   4.55
   utilization ............ =   0.71
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 4 u 1.0 2.0
   percent of rejections.... =   2.51
   utilization ............ =   0.73
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 5 u 1.0 2.0
   percent of rejections.... =   1.35
   utilization ............ =   0.74
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 6 u 1.0 2.0
   percent of rejections.... =   0.74
   utilization ............ =   0.74
hlsun:Simulation-and-Performance-Evaluation$
```

```
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 1 u 1.0 3.0
   percent of rejections.... =  27.66
   utilization ............ =   0.73
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 2 u 1.0 3.0
   percent of rejections.... =  18.45
   utilization ............ =   0.82
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 3 u 1.0 3.0
   percent of rejections.... =  13.82
   utilization ............ =   0.86
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 4 u 1.0 3.0
   percent of rejections.... =  10.93
   utilization ............ =   0.89
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 5 u 1.0 3.0
   percent of rejections.... =   9.05
   utilization ............ =   0.91
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 123456 q 6 u 1.0 3.0
   percent of rejections.... =   7.74
   utilization ............ =   0.92
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 1 u 1.0 3.0
   percent of rejections.... =  27.58
   utilization ............ =   0.72
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 2 u 1.0 3.0
   percent of rejections.... =  18.30
   utilization ............ =   0.81
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 3 u 1.0 3.0
   percent of rejections.... =  13.69
   utilization ............ =   0.86
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 4 u 1.0 3.0
   percent of rejections.... =  10.77
   utilization ............ =   0.89
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 5 u 1.0 3.0
   percent of rejections.... =   8.86
   utilization ............ =   0.91
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 246810 q 6 u 1.0 3.0
   percent of rejections.... =   7.55
   utilization ............ =   0.92
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 1 u 1.0 3.0
   percent of rejections.... =  27.24
   utilization ............ =   0.72
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 2 u 1.0 3.0
   percent of rejections.... =  18.11
   utilization ............ =   0.81
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 3 u 1.0 3.0
   percent of rejections.... =  13.41
   utilization ............ =   0.86
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 4 u 1.0 3.0
   percent of rejections.... =  10.55
   utilization ............ =   0.89
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 5 u 1.0 3.0
   percent of rejections.... =   8.60
   utilization ............ =   0.91
hlsun:Simulation-and-Performance-Evaluation$ ./Homework3.4 r 100000 s 97531 q 6 u 1.0 3.0
   percent of rejections.... =   7.43
   utilization ............ =   0.92
hlsun:Simulation-and-Performance-Evaluation$
```

```
/**
 * Modified by Harrison Sun
 * sun.har@northeastern.edu
 * February 11, 2023
 */

 /* -------------------------------------------------------------------------
  * This program is a next-event simulation of a single-server FIFO service
  * node using Exponentially distributed interarrival times and Uniformly
  * distributed service times (i.e., a M/U/1 queue).  The service node is
  * assumed to be initially idle, no arrivals are permitted after the
  * terminal time STOP, and the service node is then purged by processing any
  * remaining jobs in the service node.
  *
  * Name            : ssq3.c  (Single Server Queue, version 3)
  * Author          : Steve Park & Dave Geyer
  * Language        : ANSI C
  * Latest Revision : 10-19-98
  * -------------------------------------------------------------------------
  */

#include <cstdlib>
#include <cstring>
#include <stdio.h>
#include <exception>
#include <iostream>
#include <math.h>
#include <string>
#include "c_lib/rvgs.h"
#include "c_lib/rngs.h"

#define START         0.0              /* initial time                   */
#define STOP      20000.0              /* terminal (close the door) time */
//#define INFINITY   (10000.0 * STOP)  /* must be much larger than STOP  */
#define MAXQUEUE     6                 /* max. # of jobs in queue        */


double Min(double a, double c)
/* ------------------------------
 * return the smaller of a, b
 * ------------------------------
 */
{
    if (a < c)
        return (a);
    else
        return (c);
}

double GetArrival()
/* ---------------------------------------------
 * generate the next arrival time
 * ---------------------------------------------
 */
{
    static double arrival = START;

    SelectStream(0);
    arrival += Exponential(2.0);
    return (arrival);
}


double GetService(double lb, double ub)
/* ---------------------------------------------
```

```
 * generate the next service time with rate 2/3
 * ---------------------------------------------
 */
{
    SelectStream(1);
    return (Uniform(lb, ub));
}

/**
* bool checkArg()
*
* @param char* input - the input string literal from the console
* @return bool - true if the input is a number, false otherwise
*
* This function determines whether the argument is a number.
*/

bool checkArg(char* input)
{
    try
    {
        if (strlen(input) > 9)
        {
            throw std::logic_error("Number is too large.");
        }

        for (int i = 0; i < strlen(input); ++i)
        {

            if (std::isdigit(input[i]) || (input[i] == '.')) continue;
            else
            {
                std::string errorMessage;
                errorMessage.append((std::string)input);
                errorMessage.append(" is not a number.");
                throw std::logic_error(errorMessage);
            }
        }
        return 1;
    }

    catch (const std::logic_error& error)
    {
        std::cerr << error.what() << std::endl;
        return 0;
    }
}

int main(int argc, char* argv[])
{
    struct {
        double arrival;                 /* next arrival time */
        double completion;              /* next completion time */
        double current;                 /* current time */
        double next;                    /* next (most imminent) event time */
        double last;                    /* last arrival time */
    } t;
    struct {
        double node;                    /* time integrated number in the node */
```

```
        double queue;                   /* time integrated number in the queue */
        double service;                 /* time integrated number in service */
    } area = { 0.0, 0.0, 0.0 };
    long index{};                       /* used to count departed jobs */
  long reject{};                      /* number of jobs rejected because of full queue    */
    long number{};                      /* number in the node */
    double endtime{};
    int queuesize{};
    double lowerBound{};
    double upperBound{};
    // Set the seed
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 's' && checkArg(argv[i + 1]))
        {
            PlantSeeds(std::stol(argv[i + 1]));
            break;
        }
        else
        {
            PlantSeeds(123456789);
        }
    }

    // Set the number of runs
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'r' && checkArg(argv[i + 1]))
        {
            endtime = std::stol(argv[i + 1]);
            break;
        }
        else
        {
            endtime = STOP;
        }
    }

    // Set the queue size
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'q' && checkArg(argv[i + 1]))
        {
            queuesize = std::stol(argv[i + 1]);
            break;
        }
        else
        {
            queuesize = MAXQUEUE;
        }
    }

    // Set the bounds for uniform distribution (service time)
    for (int i = 0; i < argc; ++i)
    {
        if (*argv[i] == 'u' && checkArg(argv[i + 1]) && checkArg(argv[i + 1]))
        {
      lowerBound = std::stol(argv[i + 1]);
      upperBound = std::stol(argv[i + 2]);
            break;
        }
```

```c
        else
        {
      lowerBound = 1.0;
      upperBound = 2.0;
        }
    }
    t.current = START;            /* set the clock                         */
    t.arrival = GetArrival();     /* schedule the first arrival            */
    t.completion = INFINITY;        /* the first event can't be a completion */

    while ((t.arrival < endtime) || (number > 0)) {
        t.next = Min(t.arrival, t.completion);          /* next event time   */
        if (number > 0) {                               /* update integrals  */
            area.node += (t.next - t.current) * number;
            area.queue += (t.next - t.current) * (number - 1);
            area.service += (t.next - t.current);
        }
        t.current = t.next;                          /* advance the clock */

        if (t.current == t.arrival) {                /* process an arrival */
            if (number < queuesize)
            {
                number++;
                t.arrival = GetArrival();
                if (t.arrival > endtime) {
                    t.last = t.current;
                    t.arrival = INFINITY;
                }
                if (number == 1)
                    t.completion = t.current + GetService(lowerBound, upperBound);
            }
            else
            {
        reject++;
                t.arrival = GetArrival();
                if (t.arrival > endtime) {
                    t.last = t.current;
                    t.arrival = INFINITY;
                }
                if (number == 1)
                    t.completion = t.current + GetService(lowerBound, upperBound);
            }
        }

        else {                                         /* process a completion */
            index++;
            number--;
            if (number > 0)
                t.completion = t.current + GetService(lowerBound, upperBound);
            else
                t.completion = INFINITY;
        }
    }

    printf("\nfor %ld jobs\n", index);
    printf("   average interarrival time = %6.2f\n", t.last / index);
    printf("   average wait ........... = %6.2f\n", area.node / index);
    printf("   average delay .......... = %6.2f\n", area.queue / index);
    printf("   average service time .... = %6.2f\n", area.service / index);
    printf("   average # in the node ... = %6.2f\n", area.node / t.current);
    printf("   average # in the queue .. = %6.2f\n", area.queue / t.current);
  printf("   number of rejections..... = %6.2ld\n", reject);
  printf("   percent of rejections.... = %6.2f\n", ((double)reject / (index+reject)) * 100);
    printf("   utilization ............ = %6.2f\n", area.service / t.current);

```

```
    return (0);
}
```