

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | Fu Ruoxuan |
| **Student ID:** | 222019321062060 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Queue and Simulation | | | |
| Date | | Dec，2020 | Type | | □Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Understand the concept of container adapter    2. Know the implementation the queue container adapter in the STL    3. Grasp the use of queue container in a real application    4. Know the concept of simulation and can use simulation to solve a real problem    5. Know about the queueing theory and the exponential distribution theory | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)   Task 1  Improve the car wash simulation problem in the following ways based on the codes and slides sent to you:   * 1. Do not restrict the capacity of the car wash station.   2. The inter-arrival time should be generated from an exponential distribution randomly. An exponential distribution has a distribution function   The parameter is based on the mean inter-arrrival time from the user’s input, that is,   * 1. The service time for each car should be generated from an exponential distribution   ,  where  with mean service time provided by the user. Note that mean service time should be less than mean arrival time which means should be larger than   * 1. To generate a sequence satisfying exponential distributions, you could adopt the formula:   or  with *p* a random value in (0, 1) by uniform distribution.   * 1. Output the average waiting time and maximal queue length, using a large amount of simulation data. See if your calculated average waiting time equals:   Task1 Code  car.h   1. #ifndef CAR\_H 2. #define CAR\_H 4. **class** Car 5. { 6. **private**: 7. **double** ArrivalTime; 8. **double** DepartureTime; 9. **double** WaitingTime; 10. **double** ServiceTime; 12. **public**: 13. Car(); 14. Car(**double** ArrivalT,**double** ServiceT); 16. **double** getArrivalTime(); 17. **double** getDepartureTime(); 18. **double** getWaitingTime(); 20. **void** setDepartAndWaitTime(**double** CurrTime); 21. **void** printCarDeparture(); 22. **void** printCarArrival(); 23. };  26. #endif   washcmp.h   1. #ifndef WASH\_H 2. #define WASH\_H 4. #include <queue> 5. #include "car.h" 7. **class** WashCmp 8. { 9. **private**: 10. std::queue<Car> carQueue; 11. **int** MaxQueueLength; 12. **int** CarNumber; 13. **double** totalWaitingTime;  16. **double** getMeanArrivalRate(); //accept user input to get the mean arrival Rate 17. **double** getMeanServiceRate();//accept user input to get the mean service Rate 18. **int** getCarNumber();//accept user input to get the car number 19. **double** getArrivalTime(**double** MeanArrivalRate); 20. **double** getServiceTime(**double** MeanServiceRate); 21. **void** processArrivalNonEmptyQ(**double** currTime,**double** ServiceTime); //a car arrives and the waiting queue is not empty 22. **void** processArrivalEmptyQ(**double** currTime,**double** ServiceTime); //a car arrives and the waiting queue is empty 23. **void** processDeparture(); 24. **void** processRemain(); //no more arriving cars, process the remaining cars in the waiting queue 26. **public**: 27. **double** time;//Use this variable to record the time that has passed 28. WashCmp(); 29. **void** simulation(); 30. **void** printCmpStatistic(); 32. };  35. #endif   car.cpp   1. #include "car.h" 2. #include <iostream> 4. Car::Car() 5. { 6. ArrivalTime = 0; 7. DepartureTime = 0; 8. WaitingTime = 0; 9. } 11. Car::Car(**double** ArrivalT,**double** ServiceT) 12. { 13. ArrivalTime = ArrivalT; 14. ServiceTime = ServiceT; 15. } 17. **double** Car::getArrivalTime() 18. { 19. **return** ArrivalTime; 20. } 22. **double** Car::getDepartureTime() 23. { 24. **return** DepartureTime; 25. } 26. **double** Car::getWaitingTime() 27. { 28. **return** WaitingTime; 29. } 31. **void** Car::setDepartAndWaitTime(**double** startServiceTime) 32. { 33. DepartureTime = startServiceTime + **this**->ServiceTime; 34. WaitingTime = startServiceTime - ArrivalTime; 35. } 37. **void** Car::printCarDeparture() 38. { 39. std::cout << "A car is washed and departs!\n"; 40. std::cout << "\tArrival time: " << ArrivalTime << std::endl; 41. std::cout << "\tDeparture time: " << DepartureTime << std::endl; 42. std::cout << "\tWaiting time: " << WaitingTime << std::endl; 43. } 45. **void** Car::printCarArrival() 46. { 47. std::cout << "A car arrives!\n"; 48. std::cout << "\tArrival time: " << ArrivalTime << std::endl; 49. }   washcmp.cpp   1. #include "washCmp.h" 2. #include <iostream> 3. **using** **namespace** std; 5. WashCmp::WashCmp() 6. { 7. totalWaitingTime = 0; 8. MaxQueueLength = 0; 9. CarNumber = 0; 10. time = 0; 11. } 13. **void** WashCmp::simulation() 14. { 15. **double** MeanArrivalRate = getMeanArrivalRate(); //get the next arrival time from keyboard input 16. **double** MeanServiceRate = getMeanServiceRate(); 17. CarNumber = getCarNumber(); 18. **double** ArrivalTime = getArrivalTime(MeanArrivalRate); 19. **double** ServiceTime = getServiceTime(MeanServiceRate); 20. **int** count = 0;//Record the cars that have been serviced 21. **while**(count!=CarNumber) 22. { 23. **if** (carQueue.empty()) //queue empty, process arrival 24. { 25. cout << "current time is " << time << endl; 26. processArrivalEmptyQ(ArrivalTime,ServiceTime); 27. ArrivalTime = getArrivalTime(MeanArrivalRate); 28. ServiceTime = getServiceTime(MeanServiceRate); 29. } 30. **else** **if** (ArrivalTime < carQueue.front().getDepartureTime()) //arrival first, process arrival 31. { 32. processArrivalNonEmptyQ(ArrivalTime,ServiceTime); 33. ArrivalTime = getArrivalTime(MeanArrivalRate); 34. ServiceTime = getServiceTime(MeanServiceRate); 35. } 36. **else** //departure first or of the same time, process departure 37. { 38. processDeparture(); //no need to get next arrival 39. count++;//Counting vehicles that have been serviced when the car leaves 40. } 41. **if** (carQueue.size() > MaxQueueLength) 42. { 43. MaxQueueLength = carQueue.size(); 44. cout << "now the MaxQueueLength is " << MaxQueueLength << endl; 45. } 46. } 48. //no more arrival, process the remaining cars in the queue 49. processRemain(); 50. } 52. **double** WashCmp::getMeanArrivalRate() 53. { 54. **double** MeanArrivalRate; 55. cout << "Please input the mean arrival rate" << endl; 56. cin >> MeanArrivalRate; 57. **return** MeanArrivalRate; 58. } 60. **double** WashCmp::getMeanServiceRate() 61. { 62. **double** MeanServiceRate; 63. cout << "Please input the mean service rate" << endl; 64. cin >> MeanServiceRate; 65. **return** MeanServiceRate; 66. } 68. **int** WashCmp::getCarNumber() 69. { 70. **int** CarNumber; 71. cout << "Please input the number of cars to be simulated" << endl; 72. cin >> CarNumber; 73. **return** CarNumber; 74. } 76. **double** WashCmp::getArrivalTime(**double** MeanArrivalRate) 77. { 79. **double** p1 = rand() / **double**(RAND\_MAX + 1); 80. **double** ArrivalTime = -(1/MeanArrivalRate) \* log(1 - p1); 81. time += ArrivalTime; 82. **return** time; 83. } 85. **double** WashCmp::getServiceTime(**double** MeanServiceRate) 86. { 87. **double** p1 = rand() / **double**(RAND\_MAX + 1); 88. **double** ServiceTime = -(1/ MeanServiceRate) \* log(1 - p1); 89. **return**  ServiceTime; 90. } 92. **void** WashCmp::processArrivalNonEmptyQ(**double** ArrivalTime,**double** SeriveTime) 93. { 95. Car ArrivedCar = Car(ArrivalTime,SeriveTime); 96. ArrivedCar.printCarArrival(); //print the arrival information of the arrived car 97. carQueue.push(ArrivedCar); //set the arrival time of the arrived car 98. } 100. **void** WashCmp::processArrivalEmptyQ(**double** ArrivalTime,**double** ServiceTime) 101. { 102. Car ArrivedCar = Car(ArrivalTime,ServiceTime); 103. ArrivedCar.setDepartAndWaitTime(ArrivalTime); //set the departure and waiting time of the arrived car 104. ArrivedCar.printCarArrival(); //print the arddzrival information of the arrived car 105. carQueue.push(ArrivedCar); //set the arrival time of the arrived car 106. }   110. **void** WashCmp::processDeparture() 111. { 112. **double** currTime = carQueue.front().getDepartureTime(); //get the current time 114. totalWaitingTime += carQueue.front().getWaitingTime(); //update statistics 116. carQueue.front().printCarDeparture(); //print departure information 117. carQueue.pop(); //departs 119. **if** (!carQueue.empty())                                 //set the departure and waiting time of 120. carQueue.front().setDepartAndWaitTime(currTime);   //the current front car in the queue 121. } 123. **void** WashCmp::processRemain() //wash the remaining cars in the queue 124. { 125. **while** (!carQueue.empty()) 126. processDeparture(); 127. } 129. **void** WashCmp::printCmpStatistic() 130. { 132. std::cout << "Total waiting time: " << totalWaitingTime << std::endl; 133. std::cout << "Max queue length: " << MaxQueueLength << endl; 134. std::cout << "The average waiting time is: " << (**double**)totalWaitingTime / CarNumber << std::endl; 135. }   main.cpp   1. #include "car.h" 2. #include "washCmp.h" 3. #include <iostream> 4. #include <time.h> 5. #include <math.h> 6. **using** **namespace** std; 8. **int** main() 9. { 10. WashCmp cmp; 11. cmp.simulation(); 12. cmp.printCmpStatistic(); 13. **return** 0; 14. }   Result  CarNumber=1000  CarNumber=5000    The average waiting time is 2.083333333333333 calculated with the formula. | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Analysis of experimental results  In this simulation program, a timer is redefined, and the arrival time and service time of each car is a time interval generated by random numbers obeying an exponential distribution, and the arrival time point and service time point are obtained by summing with the values of the timer. The average waiting time obtained by varying the number of vehicles shows that as the number of simulated vehicles increases, the average waiting time obtained is closer to the value obtained by formula  Harvest  A queue is an adapter for a type of container, FIFO (First In First Out), where elements are inserted into one end of said container and fetch operations are performed from its other end.  A queue is implemented as a container adapter, which is wrapped inside an object using a specific container class as its underlying container class, providing a specific set of member functions to access its elements.  The elements are pressed into the "back" of the specified container and popped out of the "front" of it.  The underlying container can be one of the standard container class templates or some other specially designed container class.  This base container should include support for at least the following operations.  empty  size  front  back  push\_back  pop\_front  The deque and list in the standard library container satisfy the above requirements. For queue instances without a specified container, the standard library container deque is used by default.  Existing Problems  No polynomial fit was used to determine the correctness of the generated results. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |