1. Design Solution and Implementation for Each Requirement:
2. Implement the queue using “../lib/list.h”, the basic operations of queue can be emulated by list, as the following( list’s function -> queue’s function):
   1. Void Append (T item) -> void push (T item);
   2. T RemoveFront () -> T pop();
   3. Bool IsEmpty () -> bool empty();
   4. T Front () -> T front ();
   5. Int NumInList () -> int size ()
3. Design the Customer Class
   * Firstly, we need to analyze what property should a customer have, it would be:

*int UID; // users' ID*

*int itemNum; // value was set from 5-40*

*int waitTimeBegin; // the time that shopper begin to wait*

*int beginCheckoutTime; // the time that shopper begin checkout*

*int endCheckoutTime; // the time that shopper end checkout*

*int serviceTime; // the service time left to be checkout.*

* + Secondly, as those property are private, it means we need to write getter/setter function to get the property or set the property. Notice that we need to override the operator==, and operator!=, for the reason that in list.h, there is “assert” to check if the customer exists.

1. Design how customers’ pattern
   1. In this part, the first comes to mind is that how many customers are there every minute. According to the problems, it depends on the time period (peak time or regular time). Thus, we use for-loop to iterate 300 times, which mean 300mins (2pm-7pm,5 hours). Then in every minute, we use rand() to create a random number of customer number. At here, I wrote a help function to produce random number from target range, named *randomNum(int min, int max).* Also, the number of customers’ items were initialed by this function.

*// return a random number range from min to max*

*int randomNum(int min, int max) {*

*return (rand() % (max - min + 1)) + min;*

*}*

* 1. Secondly, how customers move. It would that, as a customer initialed, then enqueue to the waiting line. After all the customers produced in this minute was enqueued, check whether exists cashier available to enqueue (using *bool cashierAvaliable (List<Customer> \*cashierLine,int cashierNum)*), if so, then pop a customer from waiting line to cashier line until all the cashier are unavailable.

*// check whether exists cashier is not full*

*bool cashierAvaliable(List<Customer> \*cashierLine,int cashierNum) {*

*int flag = false;*

*for (int i = 0; i < cashierNum; i++) {*

*flag = flag || (cashierLine[i].NumInList() != maxCashierLine);*

*}*

*return flag;*

*}*

1. Desgin Cashier’s Pattern
   1. *As cashier has only two mode. One is open a new cashier line, when the number of waiting line is greater than 10 AND the current cashiers are all full AND the number of cashier line is 10 at most.*

*while (waitingLine.NumInList() > 10 && !cashierAvaliable(cashierLine, cashierNum) && cashierNum < 10 ) {*

*cashierNum++;*

*cout << "Open a new cashier line " << cashierLine << endl;*

*for (int i = 0; i < 5; i++) {*

*Customer shopper = waitingLine.Front();*

*waitLineToCashierLine(cashierLine, shopper, timeRecord, cashierNum);*

*waitingLine.RemoveFront();*

*}*

*}*

* 1. Another mode is to close a cashier line, when waiting line is empty AND this cashier is empty.

*// Check if there exists cashier need to be close.*

*while (waitingLine.IsEmpty() && cashierHasEmpty(cashierLine, cashierNum) && (cashierNum > 2)) {*

*cashierNum = closeACashier(cashierLine, cashierNum);*

*}*

1. Collect the data for summary
   1. To store data, I wrote a data structure named summary to store the information needed;

*// data structure to store the summary inforamtion*

*struct summary {*

*int customerNum = 0;*

*int minWaitTime = -1, maxWaitTime = -1, totalWaitTime = 0, minSerTime = -1, maxSerTime = -1, totalSerTime = 0;*

*double avgWaitTime, avgSerTime;*

*int maxOpenCashier = -1, totalOpenCashier = 0, cashierHalfFull = 0;*

*int minCusNumInWaitLine = -1, maxCusNumInWaitLine = -1, totalCusNumInWaitLine = 0;*

*double avgOpenCashier, avgCusNumInLine;*

*} hourSummary[5], totalSummary, testSummary;*

* 1. As for the customers’ information, every time a customer dequeue from cashier line, it means this customer has completed the order, then record this customer, by using “summary updateCusInfo(summary sum, Customer shopper)”;

*summary updateCusInfo(summary sum, Customer shopper) {*

*sum.customerNum++;*

*int waitTime = shopper.getbeginCheckoutTime() - shopper.getwaitTimeBegin();*

*sum.totalWaitTime += waitTime;*

*if (waitTime > sum.maxWaitTime)*

*sum.maxWaitTime = waitTime;*

*if (waitTime < sum.minWaitTime || sum.minWaitTime == -1)*

*sum.minWaitTime = waitTime;*

*int seriveTime = shopper.getendCheckoutTime() - shopper.getbeginCheckoutTime();*

*sum.totalSerTime += seriveTime;*

*if (seriveTime > sum.maxSerTime)*

*sum.maxSerTime = seriveTime;*

*if (seriveTime < sum.minSerTime || sum.minSerTime == -1 )*

*sum.minSerTime = seriveTime;*

*return sum;*

*}*

* 1. As for the information of cashier line and waiting line, every minute will execute the function to record this minute’s information, by using “summary updateCusInfo(summary sum, Customer shopper) ”.

*// every minute, record the cashier and waiting line's information*

*summary updateCashierInfo(summary sum, int cashierNum,*

*int waitLineSize, List<Customer> \*cashierLine) {*

*if (sum.maxOpenCashier < cashierNum || sum.maxOpenCashier == -1)*

*sum.maxOpenCashier = cashierNum;*

*sum.totalOpenCashier += cashierNum;*

*if (isCashierHalfFull(cashierLine, cashierNum))*

*sum.cashierHalfFull++;*

*if (waitLineSize < sum.minCusNumInWaitLine*

*|| sum.minCusNumInWaitLine == -1)*

*sum.minCusNumInWaitLine = waitLineSize;*

*if (waitLineSize > sum.maxCusNumInWaitLine*

*|| sum.maxCusNumInWaitLine == -1)*

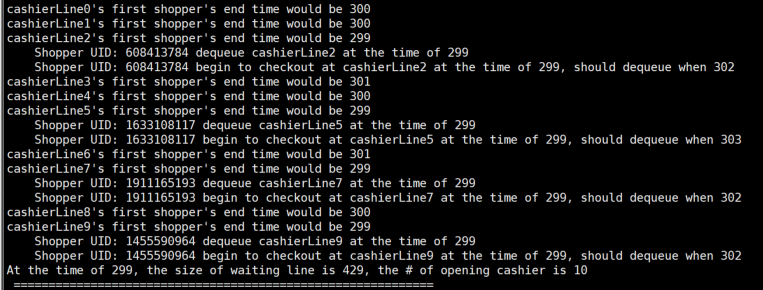
*sum.maxCusNumInWaitLine = waitLineSize;*

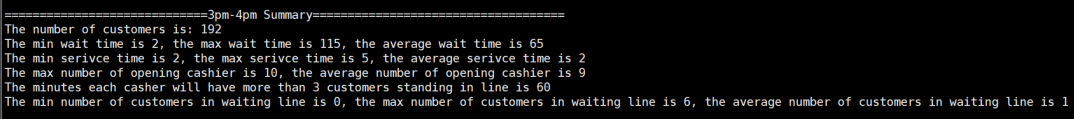
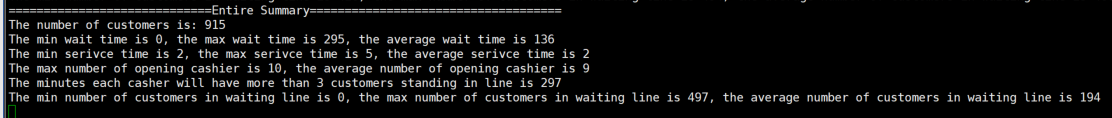
*sum.totalCusNumInWaitLine += waitLineSize;*

*return sum;*

*}*

1. Output Snapshots:
2. Every minute, print the cashier line’s enqueue and dequeue situation, shoppers’ move pattern, and the overall information (current time, the size of waiting line, the number of opening cashier)



1. Every hour, print hourly summary according to supermarket’s need
2. Finally, print entire summary according to supermarket’s needs.