```cpp
//SID: 500490778
//Date: 8/22

#include <iostream>     // std::cout
#include <algorithm>    // std::max
#include <cstdlib>

#include "queuesim.h"

//-----------Teller Methods---------------
Teller::Teller(int Num, int AvgCap, bool Print)
    :_AvgCapacity(AvgCap),
    _TellerNumber(Num),
    _QueueLength(0),
    _Print(Print)
{
    if(_Print) {
        std::cout << "CTOR Teller, ID: " << _TellerNumber << std::endl;
    }

}

//Dtor deletes alllll
Teller::~Teller() {
    if(_Print) {
        std::cout << "DTOR Teller, ID: " << _TellerNumber << std::endl;
    }
}


//processes customers one cycle from the queue
void Teller::Process() {

    if(_Print){
        std::cout << "Teller " << _TellerNumber << " processing with queue length " << _QueueLength;
    }

    //uniformly distrubuted number averaging avg_capcity taken away from queue length, equals 0 if queue negative

    _QueueLength = std::max(0, _QueueLength - rand()%(_AvgCapacity*2 + 1));

    if(_Print){
        std::cout << ", reduced after processing to " <<  _QueueLength << std::endl;
    }
};

//returns queue length
int Teller::QueueReport() {
    return _QueueLength;
}

//adds to queue
void Teller::AddQueue() {
    _QueueLength++;

    //Joining message
    if(_Print) {
        std::cout << "Joining a new customer to queue " << _TellerNumber << std::endl;
    }
};



//---------------------Checkout Methods-------------------------

Checkout::Checkout(int Strat, int TellersNum, int NewCust, bool Print, int TellerAvgCap)
    :_NumTellers(TellersNum),
    _AvgNewCustomer(NewCust),
    _ShortestQ(-1),
    _QStrat(Strat),
    _Print(Print)
{
    if(_Print) {
        std::cout << "CTOR Checkout" << std::endl;
    }


    pTellEmployee = new Teller[_NumTellers];

    for(int i=0; i<_NumTellers; i++) {

        pTellEmployee[i] = Teller(i, TellerAvgCap, _Print);

        //std::cout << i << std::endl;
    }

};

Checkout::~Checkout() {
    delete [] pTellEmployee;


    if(_Print) {
        std::cout << "DTOR Checkout" << std::endl;
    }
};

void Checkout::FindShortestQueue() {

    //std::cout << "Checkout::FindShortestQueue" << std::endl;

    _ShortestQ = 0;
    int Length = pTellEmployee[0].QueueReport(); //find length of queue 0


    //loop through all tellers to find shortest queue, if tie lowest number teller is allocated
    for(int i=1; i<_NumTellers; i++) {
        //find if current queue (i) is shorter than shortest queue
        if(Length > pTellEmployee[i].QueueReport()) {
            //assign new shortest queue
            _ShortestQ = i;
            Length = pTellEmployee[i].QueueReport();
        }
    }
};

//adds new customer to existing queues, uses find shortest queue to efficiently disperse new customers
void Checkout::AddNewCustomers() {

    int NewCust = rand() % (_AvgNewCustomer*2 + 1);


    //display allocation customer message
    if(_Print) {
        std::cout << "Allocating " << NewCust << " new customers" << std::endl;
    }


    //find shortest queue and add customer to said queue
    for(int i=0; i<NewCust; i++) {

        //queue strategy
        if(_QStrat == 0) { //allocate customer to the shortest queue
            FindShortestQueue();
            pTellEmployee[_ShortestQ].AddQueue();
        } else if (_QStrat == 1) {
            pTellEmployee[rand() % _NumTellers].AddQueue(); //allocates customer to random queue
        }

    }


};

//tells all tellers to process customers
void Checkout::ProcessTellersOneCycle() {

    //loops through all tellers to process cycles
    for(int i=0; i<_NumTellers; i++) {
        pTellEmployee[i].Process();
    }
};

double Checkout::ReportAvgQLength() {

    double Total = 0;

    //loops through all tellers
    for(int i=0; i<_NumTellers; i++) {
        Total += (double)pTellEmployee[i].QueueReport();
    }

    //std::cout << Total / (double)_NumTellers << std::endl;

    return Total / (double)_NumTellers;
};
```

```cpp

//---------------------Simulation Methdos--------------
Simulator::Simulator(int strat, bool print, int cyc)
    :_TotalCycles(cyc),
    _CurrentCycle(0),
    _QStrat(strat),
    _Print(print),
    _TotalQLength(0)
{
    if(_Print) {
        std::cout << "CTOR Simulator" << std::endl;
    }

};

Simulator::~Simulator() {
    if(_Print) {
        std::cout << "DTOR Simulator" << std::endl;
    }

};

void Simulator::RunSimulation(int TellersNum, int TellerCycleAvg, int AvgCustomerPerCycle) {

    Checkout Coles(_QStrat, TellersNum, AvgCustomerPerCycle, _Print, TellerCycleAvg);

    //loop through each cycle, adding new customers and then processing them
    for(_CurrentCycle=0; _CurrentCycle< _TotalCycles; _CurrentCycle++) {

        if(_Print){
            std::cout << "[RunSimulation] simulation cycle " << _CurrentCycle << std::endl;     //prints simulation cycle number
        }
        //adds 0-MaxCustomers to queue
        Coles.AddNewCustomers();

        //Process Customers from teller queues
        Coles.ProcessTellersOneCycle();

        //Find Q length
        _TotalQLength += Coles.ReportAvgQLength();
    }
};

void Simulator::StratComparison() {

    //Run simulation with shortest Q strat
    _QStrat = 0;
    _TotalQLength = 0;
    _CurrentCycle = 0;
    RunSimulation();

    std::cout << "Shortest Queue Strat Average Queue: " << _TotalQLength / (double)(_CurrentCycle+1) << std::endl;

    _TotalQLength = 0;
    _CurrentCycle = 0;
    _QStrat = 1;

    RunSimulation();
    std::cout << "Random Queue Strat Average Queue: " << _TotalQLength / (double)(_CurrentCycle+1) << std::endl;

};
```

```
// SID: 500490778
// 8/2022
/*
This file stores classes that are used to run a shopping line simulator.
The Simulator class will create an instance of the checkout class, and then
the checkout class will create an multple instances of the teller class. The simulator
class will run the simulation for the inputed number of cycles
Checkout class will is capable of controlling the teller class and will simulate adding new
customers and allocating which line
Teller runs the most basic methods that simulate the its own line
*/

#ifndef _QUEUESIM_H
#define _QUEUESIM_H

//Simulates a queue at a teller
class Teller {

    public:
        //tellers number in checkout, average capacity per cycle,
        Teller(int Num=0, int AvgCap = 3, bool Print = false);
        ~Teller();

        void Process();               // Processes one cycles, will remove _TellerCapacity from queue
        int QueueReport();                  //Returns length of queue
        void AddQueue();                    //adds to teller queue

    private:
        int _AvgCapacity;       // how many customers a teller processes in a cycle
        int _QueueLength;          // length of queue
        int _TellerNumber;         // number teller is at checkout
        bool _Print;               //determines if methods print to terminal
};

//simulates a checkout, runs multiple teller queues
class Checkout {

    public:
        //defualt strat shortest queue, two tellers, default avg 6 new customers, default not print, teller process average 3/cycle
        Checkout(int Strat = 0, int TellersNum = 2, int NewCust = 6, bool Print = false, int TellerAvgCap = 3);
        ~Checkout();

        void FindShortestQueue();            // finds the shortest queue from an array of queue lengths
        void AddNewCustomers();             // adds all new customers to appropriate queues
        void ProcessTellersOneCycle();       // processes all tellers one cycle
        double ReportAvgQLength();          // returns average length of all teller queues

    private:
        int _NumTellers;            // number of tellers in the checkout
        int _ShortestQ;              // current shortest queue
        int _AvgNewCustomer;         //average numbers of new customers added each cycle
        Teller* pTellEmployee;      // pointer to an array of tellers
        int _QStrat;                // 0 for shortest Q strat, 1 for random Q, determines how AddNewCustomers() behaves
        bool _Print;                // determines if class prints to terminal
};

//Runs simulation of checkout multiple cycles
class Simulator {

    public:
        //simulation cycles, number of tellers, average teller process per cycle, average new custmers each cycle
        Simulator(int Strat = 0, bool print = true, int cyc = 1000);
        ~Simulator();

        //controls the running of the simulation and takes all inputs required for simulation
        void RunSimulation(int TellersNum = 3, int TellerCycleAvg = 3, int AvgCustomerPerCycle = 6);

        //Runs RunSimulation() for both strategies, prints average queue length of each
        void StratComparison();

    private:
        int _TotalCycles;           // duration of simulation, in cycles
        int _CurrentCycle;          // current cycle of simulation
        int _QStrat;                // 0 for shortest Q strat, 1 for random Q
        bool _Print;                // if function prints to terminal or not
        double _TotalQLength;       // Total Queue Length in simulation (adds all queues to length every cycles)
};

```

```


#endif
```

```
// SID: 500490778
// 8/2022
/*
This program runs a queueing simulator to compare the average Queue length if new customer allocated
to the shortest Queue vs allocated to a random Q. Read queuesim.h for details of all classes used
*/

#include <iostream>
#include <cstdlib>              // rand
#include <algorithm>            // std::max

#include "queuesim.h"

int main() {

    srand(time(0)); //random number generator same for both strategies, different every run time


    //Runs the simulation
    Simulator* Output = new Simulator(0, true, 100); //shortest Q strat, printing = true, 100 cycles
    Output->RunSimulation();
    delete Output;

    //Compares random Q strat to shortest Q strat
    Simulator Compare(0, false);
    Compare.StratComparison();

}
```