```
#include <iostream>
#include <vector>
#include <cmath>
#include <sstream>
#include <iomanip>

#include "queue.h"

using namespace single_channel_queue_simulation;


int Customer::customer_id_ = 0;
int Simulator::n_wait_ = 0;
int Simulator::clock_ = 0;

Logger::Logger() {
  log_file_ = std::ofstream("log.txt");
}

Logger::~Logger() {
  log_file_.close();
}

void Logger::Log(std::string log) {
  log_file_ << log;
  log_file_ << std::endl;
}


void EventModel::SetCumProb() {
  float cum;

  for(int i = 0; i < n_options_; i++) {
    cum = 0;
    for(int j = 0; j <= i; j++) {
      cum += probs_[j];
    }
    cum_prob_[i] = cum;
  }
}

void EventModel::SetCumSum() {
  for(int i = 0; i < n_options_; i++) {
    cum_sum_[i] = std::pow(10, n_decimal_) * cum_prob_[i];
  }
}

EventModel::EventModel(int n_decimal, std::vector<int> options,
std::vector<float> probs) {
  options_ = options;
  probs_ = probs;
  n_decimal_ = n_decimal;
  n_options_ = options_.size();
  cum_prob_.resize(n_options_);
  cum_sum_.resize(n_options_);
  SetCumProb();
  SetCumSum();
}

int EventModel::GetEvent() {
```

```
  int r = std::rand();
  int range = std::pow(10, n_decimal_);
  r = r % range;

  if(r == 0) return options_.back();

  for(int i = 0; i < n_options_; i++) {
    if(r <= cum_sum_[i]) return options_[i];
  }

  throw "GET EVENT FAILED";
}

Simulator::Simulator(int n_customer, EventModel& arrival_model,
EventModel& service_model)
  : arrival_model_(arrival_model), service_model_(service_model) {
  n_customer_ = n_customer;
  arrival_model_ = arrival_model;
  service_model_ = service_model;
  total_it_ = total_its_ = total_st_ = total_tcss_ = total_wtq_ = 0;
}

Customer::Customer(int service_time) {
  customer_id_++;
  inter_arrival_time_ = arrival_time_ = time_service_begins_ =
    waiting_time_in_queue_ = idle_time_of_server_ = 0;
  service_time_ = time_service_ends_ = time_customer_spends_in_system_ =
service_time;
}

Customer::Customer(int arrival_interval, int service_time, Customer
prev_customer) {
  customer_id_++;

  inter_arrival_time_ = arrival_interval;
  arrival_time_ = inter_arrival_time_ + prev_customer.arrival_time_;
  service_time_ = service_time;
  time_service_begins_ = std::max(arrival_time_,
prev_customer.time_service_ends_);
  waiting_time_in_queue_ = (arrival_time_ >=
prev_customer.time_service_ends_)
    ? 0
    : prev_customer.time_service_ends_ - arrival_time_;
  time_service_ends_ = time_service_begins_ + service_time_;
  time_customer_spends_in_system_ = service_time_ +
waiting_time_in_queue_;
  idle_time_of_server_ = (arrival_time_ >=
prev_customer.time_service_ends_)
    ? arrival_time_ - prev_customer.time_service_ends_
    : 0;

  Simulator::clock_ = time_service_ends_;

  if(arrival_time_ < prev_customer.time_service_ends_)
Simulator::n_wait_++;
}

void Simulator::LogCustomer(Customer& c) {
  std::stringstream details;
  details << c.customer_id_
```

```
          << std::setw(6) << c.inter_arrival_time_
          << std::setw(6) << c.arrival_time_
          << std::setw(6) << c.service_time_
          << std::setw(6) << c.time_service_begins_
          << std::setw(6) << c.waiting_time_in_queue_
          << std::setw(6) << c.time_service_ends_
          << std::setw(6) << c.time_customer_spends_in_system_
          << std::setw(6) << c.idle_time_of_server_ << std::endl
    ;

  std::string details_string = details.str();
  logger_.Log(details_string);
}

void Simulator::InitializeLogTable() {
  std::stringstream heads;
  heads << "C"
        << std::setw(6) << "IT"
        << std::setw(6) << "AT"
        << std::setw(6) << "ST"
        << std::setw(6) << "TSB"
        << std::setw(6) << "WTQ"
        << std::setw(6) << "TSE"
        << std::setw(6) << "TCSS"
        << std::setw(6) << "ITS" << std::endl
    ;

  std::string heads_string = heads.str();
  logger_.Log(heads_string);
}


void Simulator::UpdateHistory(Customer& c) {
  total_it_ += c.inter_arrival_time_;
  total_its_ += c.idle_time_of_server_;
  total_st_ += c.service_time_;
  total_tcss_ += c.time_customer_spends_in_system_;
  total_wtq_ += c.waiting_time_in_queue_;
}

void Simulator::LogTotals() {
  std::stringstream totals;
  totals << ""
          << std::setw(10) << total_it_
          << std::setw(6) << ""
          << std::setw(6) << total_st_
          << std::setw(6) << ""
          << std::setw(6) << total_wtq_
          << std::setw(6) << ""
          << std::setw(6) << total_tcss_
          << std::setw(6) << total_its_ << std::endl
    ;

  std::string totals_string = totals.str();
  logger_.Log(totals_string);
}

void Simulator::LogMetrics() {
  std::stringstream metrics;
```

```
  metrics << "average waiting time: " << (float)total_wtq_ / n_customer_
<< std::endl
          << "waiting probability: " << (float)n_wait_ / n_customer_ <<
std::endl
          << "server idle probability: " << (float)total_its_ / clock_ <<
std::endl
          <<  "average service time: " << (float)total_st_ / n_customer_
<< std::endl
          << "average inter arrival time: " << (float)total_it_ /
(n_customer_ - 1) << std::endl
          << "average waiting time for queue people: " <<
(float)total_wtq_ / n_wait_ << std::endl
          << "average time in system: " << (float)total_tcss_ /
n_customer_ << std::endl
    ;

  std::string metrics_string = metrics.str();
  logger_.Log(metrics_string);
}

void Simulator::RunSimulation() {
  if(n_customer_ == 0) return;

  InitializeLogTable();

  Customer customer(service_model_.GetEvent());
  LogCustomer(customer);
  UpdateHistory(customer);

  for(int i = 0; i < n_customer_ - 1; i++) {
    int service_time, arrival_interval;
    service_time = service_model_.GetEvent();
    arrival_interval = arrival_model_.GetEvent();
    customer = Customer(arrival_interval, service_time, customer);
    LogCustomer(customer);
    UpdateHistory(customer);
  }

  LogTotals();
  LogMetrics();
}

int main() {
  std::vector<int> arrival_intervals {1, 2, 3, 4, 5, 6, 7, 8};
  std::vector<float> arrival_probs (8, 0.125);
  std::vector<int> service_times {1, 2, 3, 4, 5, 6};
  std::vector<float> service_probs {0.1, 0.2, 0.3, 0.25, 0.1, 0.05};
  int n_customer = 100;

  EventModel arrival_model(3, arrival_intervals, arrival_probs);
  EventModel service_model(2, service_times, service_probs);
  Simulator simulator(n_customer, arrival_model, service_model);

  simulator.RunSimulation();

  return 0;
}
```

C	IT	AT	ST	TSB	WTQ	TSE	TCSS	ITS
1	0	0	4	0	0	4	4	0
2	7	7	5	7	0	12	5	3
3	7	14	2	14	0	16	2	2
4	4	18	3	18	0	21	3	2
5	6	24	5	24	0	29	5	3
6	3	27	2	29	2	31	4	0
7	6	33	2	33	0	35	2	2
8	7	40	3	40	0	43	3	5
9	5	45	2	45	0	47	2	2
10	2	47	2	47	0	49	2	0
11	2	49	3	49	0	52	3	0
12	5	54	4	54	0	58	4	2
13	7	61	2	61	0	63	2	3
14	7	68	2	68	0	70	2	5
15	1	69	2	70	1	72	3	0
16	8	77	3	77	0	80	3	5
17	1	78	1	80	2	81	3	0
18	1	79	3	81	2	84	5	0
19	4	83	4	84	1	88	5	0
20	1	84	3	88	4	91	7	0
21	2	86	3	91	5	94	8	0
22	4	90	4	94	4	98	8	0
23	7	97	2	98	1	100	3	0
24	2	99	3	100	1	103	4	0
25	3	102	2	103	1	105	3	0
26	4	106	4	106	0	110	4	1
27	1	107	2	110	3	112	5	0
28	8	115	4	115	0	119	4	3
29	7	122	4	122	0	126	4	3

30	8	130	4	130	0	134	4	4
31	3	133	4	134	1	138	5	0
32	1	134	2	138	4	140	6	0
33	3	137	2	140	3	142	5	0
34	7	144	1	144	0	145	1	2
35	3	147	2	147	0	149	2	2
36	1	148	3	149	1	152	4	0
37	5	153	5	153	0	158	5	1
38	7	160	3	160	0	163	3	2
39	4	164	4	164	0	168	4	1
40	1	165	4	168	3	172	7	0
41	1	166	3	172	6	175	9	0
42	3	169	1	175	6	176	7	0
43	7	176	4	176	0	180	4	0
44	4	180	4	180	0	184	4	0
45	7	187	3	187	0	190	3	3
46	8	195	6	195	0	201	6	5
47	6	201	3	201	0	204	3	0
48	6	207	4	207	0	211	4	3
49	2	209	2	211	2	213	4	0
50	1	210	5	213	3	218	8	0
51	7	217	3	218	1	221	4	0
52	4	221	4	221	0	225	4	0
53	4	225	4	225	0	229	4	0
54	1	226	1	229	3	230	4	0
55	3	229	1	230	1	231	2	0
56	6	235	5	235	0	240	5	4
57	3	238	3	240	2	243	5	0
58	3	241	4	243	2	247	6	0
59	7	248	3	248	0	251	3	1

60	4	252	5	252	0	257	5	1
61	4	256	2	257	1	259	3	0
62	1	257	2	259	2	261	4	0
63	1	258	2	261	3	263	5	0
64	4	262	4	263	1	267	5	0
65	6	268	4	268	0	272	4	1
66	5	273	2	273	0	275	2	1
67	4	277	3	277	0	280	3	2
68	5	282	3	282	0	285	3	2
69	3	285	4	285	0	289	4	0
70	2	287	4	289	2	293	6	0
71	5	292	2	293	1	295	3	0
72	6	298	4	298	0	302	4	3
73	5	303	2	303	0	305	2	1
74	3	306	2	306	0	308	2	1
75	6	312	4	312	0	316	4	4
76	3	315	2	316	1	318	3	0
77	7	322	3	322	0	325	3	4
78	5	327	2	327	0	329	2	2
79	7	334	3	334	0	337	3	5
80	8	342	3	342	0	345	3	5
81	4	346	3	346	0	349	3	1
82	7	353	5	353	0	358	5	4
83	7	360	2	360	0	362	2	2
84	2	362	3	362	0	365	3	0
85	1	363	6	365	2	371	8	0
86	1	364	4	371	7	375	11	0
87	8	372	2	375	3	377	5	0
88	6	378	3	378	0	381	3	1
89	2	380	5	381	1	386	6	0

90	7	387	4	387	0	391	4	1
91	4	391	3	391	0	394	3	0
92	4	395	3	395	0	398	3	1
93	3	398	2	398	0	400	2	0
94	4	402	1	402	0	403	1	2
95	7	409	2	409	0	411	2	6
96	1	410	5	411	1	416	6	0
97	2	412	3	416	4	419	7	0
98	7	419	5	419	0	424	5	0
99	2	421	2	424	3	426	5	0
100	8	429	2	429	0	431	2	3
		429		309		97	406	122

average waiting time: 0.97
waiting probability: 0.4
server idle probability: 0.283063
average service time: 3.09
average inter arrival time: 4.33333
average waiting time for queue people: 2.425
average time in system: 4.06