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

#include "news_paper.h"

using namespace news_paper;

const int Simulator::kNDay = 20;

void Logger::SetLogFile(std::string fs) {
  log_file_ = std::ofstream(fs);
}

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

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

template <class T>
void EventModel<T>::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;
  }
}

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

template <class T>
EventModel<T>::EventModel(int n_decimal, std::vector<T> 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();
}

template <class T>
T EventModel<T>::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 (r);
}

Simulator::Simulator(EventModel<DayType>& day_model, EventModel<int>&
good_model,
                     EventModel<int>& fair_model, EventModel<int>&
poor_model)
  : day_model_(day_model), good_model_(good_model),
    fair_model_(fair_model), poor_model_(poor_model) {
  total_revenue_ = total_lost_profit_ = total_salvage_ = total_cost_ =
total_profit_ = n_news_paper_ = 0;
}

void Simulator::ResetTotals() {
  total_revenue_ = total_lost_profit_ = total_salvage_ = total_cost_ =
total_profit_ = 0;
}

bool Logger::HasLogFile() {
  return log_file_.is_open();
}

void Logger::CloseLogFile() {
  log_file_.close();
}

void Simulator::SetNNewsPaper(int n) {
  n_news_paper_ = n;

  if(logger_.HasLogFile()) logger_.CloseLogFile();
  std::stringstream fs;
  fs << "log_" << n << ".txt";
  logger_.SetLogFile(fs.str());
}

Day::Day(int id, int demand, int n_np, DayType dt) {
  id_ = id;
  demand_ = demand;
  n_np_ = n_np;
  day_type_ = dt;

  revenue_ = lost_profit_ = salvage_ = cost_ = profit_ = 0;
}

int Simulator::GetDemand(DayType dt) {
  switch(dt) {
  case DayType::kGood:
    return good_model_.GetEvent();
    break;
  case DayType::kFair:
```

```
    return fair_model_.GetEvent();
    break;
  case DayType::kPoor:
    return poor_model_.GetEvent();
    break;
  default:
    throw (dt);
  }
}

void Day::SetFields() {
  revenue_ = std::min(demand_, n_np_) * 0.5;
  lost_profit_ = (demand_ > n_np_) ? (demand_ - n_np_) * 0.1 : 0;
  salvage_ = (demand_ > n_np_) ? (demand_ - n_np_) * 0.05 : 0;
  cost_ = n_np_ * 0.33;
  profit_ = revenue_ - cost_ - lost_profit_ + salvage_;
}

void Simulator::StepSimulate(int it, Day& day) {
  DayType dt = day_model_.GetEvent();
  int demand = GetDemand(dt);

  day = Day(it, demand, n_news_paper_, dt);
  day.SetFields();
}

void Simulator::InitializeLogTable() {
  std::stringstream heads;
  heads << "D"
        << std::setw(10) << "Type"
        << std::setw(10) << "Demand"
        << std::setw(10) << "Revenue"
        << std::setw(10) << "Lost"
        << std::setw(10) << "Salvage"
        << std::setw(10) << "Cost"
        << std::setw(10) << "Profit" << std::endl
    ;

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

int Day::GetID(){
  return id_;
}

int Day::GetDemand() {
  return demand_;
}

int Day::GetNNP() {
  return n_np_;
}

DayType Day::GetDayType() {
  return day_type_;
}

float Day::GetRevenue() {
  return revenue_;
```

```
}

float Day::GetLostProfit() {
  return lost_profit_;
}

float Day::GetSalvage() {
  return salvage_;
}

float Day::GetCost() {
  return cost_;
}

float Day::GetProfit() {
  return profit_;
}

void Simulator::LogDay(Day& d) {
  std::string dt;

  switch(d.GetDayType()) {
  case kGood:
    dt = "good";
    break;
  case kFair:
    dt = "fair";
    break;
  case kPoor:
    dt = "poor";
    break;
  default:
    throw(d.GetDayType());
  }

  std::stringstream details;
  details << d.GetID() + 1
          << std::setw(10) << dt
          << std::setw(10) << d.GetDemand()
          << std::setw(10) << d.GetRevenue()
          << std::setw(10) << d.GetLostProfit()
          << std::setw(10) << d.GetSalvage()
          << std::setw(10) << d.GetCost()
          << std::setw(10) << d.GetProfit() << std::endl
    ;

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

void Simulator::LogTotals() {
  std::stringstream totals;
  totals << "Totals"
         << std::setw(6) << ""
         << std::setw(10) << ""
         << std::setw(10) << total_revenue_
         << std::setw(10) << total_lost_profit_
         << std::setw(10) << total_salvage_
         << std::setw(10) << total_cost_
         << std::setw(10) << total_profit_ << std::endl
```

```
    ;

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

void Simulator::UpdateTotals(Day& day) {
  total_revenue_ += day.GetRevenue();
  total_lost_profit_ += day.GetLostProfit();
  total_salvage_ += day.GetSalvage();
  total_cost_ += day.GetCost();
  total_profit_ += day.GetProfit();
}

void Simulator::RunSimulation() {
  if(kNDay == 0) return;
  Day day(0, 0, 0, DayType::kGood);

  for(int i = 0; i < 1000; i++) {
    StepSimulate(i, day);
  }

  InitializeLogTable();

  for(int i = 0; i < kNDay; i++) {
    StepSimulate(i, day);
    UpdateTotals(day);
    LogDay(day);
  }

  LogTotals();
}

float Simulator::GetTotalProfit() {
  return total_profit_;
}

int main() {
  int n_runs = 3;
  std::vector<int> n_np = {60, 70, 80};
  std::vector<int> demands = {40, 50, 60, 70, 80, 90, 100};
  std::vector<float> good_demands_prob = {0.03, 0.05, 0.15, 0.2, 0.35,
0.15, 0.07};
  std::vector<float> fair_demands_prob = {0.1, 0.18, 0.4, 0.2, 0.08,
0.04, 0};
  std::vector<float> poor_demands_prob = {0.44, 0.22, 0.16, 0.12, 0.06,
0, 0};
  std::vector<DayType> day_types = {DayType::kGood, DayType::kFair,
DayType::kPoor};
  std::vector<float> day_types_prob = {0.35, 0.45, 0.2};

  Logger logger();

  EventModel<DayType> day_model(2, day_types, day_types_prob);
  EventModel<int> good_model(2, demands, good_demands_prob),
    fair_model(2, demands, fair_demands_prob),
    poor_model(2, demands, poor_demands_prob);
  Simulator simulator(day_model, good_model, fair_model, poor_model);

  std::vector<float> profits;
```

```
  for(int i = 0; i < n_runs; i++) {
    simulator.ResetTotals();
    simulator.SetNNewsPaper(n_np[i]);
    simulator.RunSimulation();
    profits.push_back(simulator.GetTotalProfit());
  }

  int max_ind = 0;

  for(int i = 1; i < n_runs; i++) {
    if(profits[i] > profits[max_ind]) max_ind = i;
  }

  std::cout << "Best performance was for: " << n_np[max_ind] <<
std::endl;


  return 0;
}
```

D	Type	Demand	Revenue	Lost	Salvage	Cost	Profit
1	fair	50	25	0	0	19.8	5.2
2	good	100	30	4	2	19.8	8.2
3	poor	40	20	0	0	19.8	0.200001
4	fair	60	30	0	0	19.8	10.2
5	fair	60	30	0	0	19.8	10.2
6	poor	70	30	1	0.5	19.8	9.7
7	good	80	30	2	1	19.8	9.2
8	fair	60	30	0	0	19.8	10.2
9	fair	60	30	0	0	19.8	10.2
10	poor	40	20	0	0	19.8	0.200001
11	fair	50	25	0	0	19.8	5.2
12	fair	50	25	0	0	19.8	5.2
13	poor	40	20	0	0	19.8	0.200001
14	fair	50	25	0	0	19.8	5.2
15	poor	60	30	0	0	19.8	10.2
16	fair	60	30	0	0	19.8	10.2
17	good	60	30	0	0	19.8	10.2
18	good	80	30	2	1	19.8	9.2
19	good	70	30	1	0.5	19.8	9.7
20	poor	50	25	0	0	19.8	5.2
Totals			545	10	5	396	144

D	Type	Demand	Revenue	Lost	Salvage	Cost	Profit
1	fair	70	35	0	0	23.1	11.9
2	fair	50	25	0	0	23.1	1.9
3	fair	50	25	0	0	23.1	1.9
4	poor	50	25	0	0	23.1	1.9
5	good	80	35	1	0.5	23.1	11.4
6	fair	70	35	0	0	23.1	11.9
7	good	60	30	0	0	23.1	6.9
8	fair	80	35	1	0.5	23.1	11.4
9	fair	40	20	0	0	23.1	-3.1
10	poor	50	25	0	0	23.1	1.9
11	good	80	35	1	0.5	23.1	11.4
12	good	90	35	2	1	23.1	10.9
13	fair	60	30	0	0	23.1	6.9
14	fair	40	20	0	0	23.1	-3.1
15	fair	60	30	0	0	23.1	6.9
16	poor	50	25	0	0	23.1	1.9
17	poor	40	20	0	0	23.1	-3.1
18	good	100	35	3	1.5	23.1	10.4
19	fair	50	25	0	0	23.1	1.9
20	good	60	30	0	0	23.1	6.9
Totals			575	8	4	462	109

D	Type	Demand	Revenue	Lost	Salvage	Cost	Profit
1	fair	60	30	0	0	26.4	3.6
2	good	100	40	2	1	26.4	12.6
3	fair	60	30	0	0	26.4	3.6
4	fair	60	30	0	0	26.4	3.6
5	poor	60	30	0	0	26.4	3.6
6	fair	50	25	0	0	26.4	-1.4
7	poor	40	20	0	0	26.4	-6.4
8	fair	80	40	0	0	26.4	13.6
9	fair	60	30	0	0	26.4	3.6
10	fair	90	40	1	0.5	26.4	13.1
11	fair	90	40	1	0.5	26.4	13.1
12	good	100	40	2	1	26.4	12.6
13	fair	60	30	0	0	26.4	3.6
14	fair	60	30	0	0	26.4	3.6
15	poor	40	20	0	0	26.4	-6.4
16	poor	70	35	0	0	26.4	8.6
17	fair	50	25	0	0	26.4	-1.4
18	poor	60	30	0	0	26.4	3.6
19	fair	80	40	0	0	26.4	13.6
20	good	100	40	2	1	26.4	12.6
Totals			645	8	4	528	113