POLLING SIMULATION PROGRAM

MAINTENANCE GUIDE

Table of Contents:

Introduction:

This program is designed to perform calculations involved with queue theory. Currently, the program is configured for simulating polling times at designated voting precincts. It requires 3 different input files which are included in the package as well as instructions for altercation. This document is designed to serve as an introduction to the inner workings of the program and contains in depth breakdowns within the code itself. The code in this program is formatted to match the requirements of the Google C++ Style Guide.

1. Main .cc .h:

Main serves as the front end for the program. It handles setting up all the variables involved with input and output files, as well as all major function. Main uses not only the following local variables, but member functions and variables from the included Utilities file. For further information, see the Utilities section.

1.1 Local Variables:

string config_filename - configuration file.txt, defined by argument 1.

string pct_filename - precinct file.txt, defined by argument 2.

string out_filename - output file.txt, defined by argument 3.

string log_filename - log file.txt, defined by argument 4.

string service_time_filename - service-times file.txt, defined by argument 5.

string outstring - used to store output, output to log and output files.

ofstream out_stream - output stream used to write to out_filename

Scanner config_stream - scanner used for reading the configuration file.

Scanner pct_stream - scanner used for reading the precinct file.

Scanner service_stream - scanner used for reading the service-times file.

Configuration config - Object of Configuration class, used to configure the computation with a configure file.

Simulation simulation - Object of Simulation class, used to do the actual queuing process and statistics.

MyRandom random - Object of MyRandom class, used to generate random numbers.

2. Config .cc .h:

The Config class is responsible for assigning independent variables necessary to run the simulation. It reads in values given by a config file and passes them on to other classes in the program. It also reads and handles the service time file.

2.1 Public Variables:

int seed_ - seed used by MyRandom, if unspecified, 1 is the default.

int election_day_length_hours_ - hours in an election day.

int election_day_length_seconds_ - seconds in an election day, calculated, not input.

int time_to_vote_mean_seconds_ - average time to vote in seconds, calculated.

int max_expected_to_simulate_ - max voters expected to simulate.

int min_expected_to_simulate_ - minimum voters expected to simulate.

int wait_time_minutes_that_is_too_long_ - wait time which is considered too long.

int number_of_iterations_ - number of iterations to perform.

double arrival_zero_ - percentage of voters who arrive at time 0.

vector<int> actual_service_times_ - stores the values from the service-times file.

vector<double> arrival_fractions_ - stores the doubles from the second line of the configuration file.

2.2 General Functions:

int GetMaxServiceSubscript() const;

void ReadConfiguration(Scanner& instream);

void ReadServiceTimes(Scanner& instream);

string ToString();

GetMaxServiceSubscript() - returns the longest service times from the service times vector.

ReadConfiguration() - reads the Configuration file.

ReadServiceTimes() - reads the Service-Time file.

ToString() - outputs a formatted form of the variables including GetMaxServiceSubscript().

2.3 Creating Configuration Files: (file_name.txt)

The Configuration class is responsible for configuring the statistical computation needed by this election simulator. Thus, this class is the cast that the 'rest of the program' is poured into. This class fulfills some of the much needed independent variables needed to simulate a queue, in this case an election, as specified by traditional Queueing theory. This class requires two input files that are given by the user in the command line arguments and passed into this module via input streams; hence, those two files are the Configuration file and Service-Time file. The Configuration file has the input delimited by spaces; thus, the input should be formatted like this, let ';' be the space delimiter:

2.3.1 Configuration File:

FirstLine:

RN_SEED;number of hours in election day; mean time to vote; max number of voters per pct;

waiting time(minutes);"too long" number of iterations to perform.

Example: 35 13 105 50 5000 30 3

SecondLine:

percent at time zero; pct arrival percentages per hour (13 numbers).

Example: 0.0 10.0 10.0 10.0 5.0 5.0 5.0 10.0 10.0 5.0 5.0 5.0 10.0 10.0

2.3.2 Service-Time File:

The Service-Time file should be formatted like a column, and all the values should be given as integers in sorted ascending order.

3. Simulation .cc .h:

The Simulation class handles the front-end of precinct simulation. To begin, the Simulation class reads in individual precinct data with the ReadPrecincts() member function. After the precinct data is read in, RunSimulation() iterates through pcts_ and runs simulations via OnePct::RunSimulationPct().

3.1 Private Variables:

map<int, OnePct> pcts_; - a map of the precincts.

3.2 General Functions:

void ReadPrecincts(Scanner& infile);

void RunSimulation(const Configuration& config, MyRandom& random, ofstream& out_stream);

string ToString();

ReadPrecincts() - reads from the Precinct file, creates a precinct, then stores it in the pcts_ map.

RunSimulation() - iterates through pcts_, checks to make sure the expected number of voters is with the minimum and maximum expected voters, then proceeds to run simulations.

ToString() - outputs precinct data to a string via OnePct()::ToString().

4. OnePct .cc .h:

The OnePct class is where the bulk of the work is done. It uses data input from the Configuration file to perform computations involving queue theory. It also creates and uses voters from the OneVoter class.

4.1 Private Variables:

int pct_expected_voters_ - the number of voters expected by a precinct.

int pct_expected_per_hour_ - voters expected per hour.

double pct_minority_ - percent of precinct that is a minority voter.

string pct_name_ - precinct name.

int pct_number_ - precinct number (ID).

double pct_turnout_ - the percentage turnout of voters.

int pct_stations_ - number of voting stations.

int pct_num_voters_ - total number of voters.

double wait_dev_seconds_ - deviation of the mean wait time, calculated.

double wait_mean_seconds_ - mean wait time, calculated.

set<int> stations_to_histo_;

vector<int> free_stations_ ;

multimap<int, OneVoter> voters_backup_ - a backup of all voters.

multimap<int, OneVoter> voters_done_voting_ - a map of voters who have finished voting.

multimap<int, OneVoter> voters_pending_ - a map of voters waiting to vote.

multimap<int, OneVoter> voters_voting_ - a map of voters currently voting.

4.2 Accessors:

int GetExpectedVoters() const - returns the expected voter turnout for a precinct.

int GetPctNumber() const - returns the precinct ID.

4.3 Public Functions:

void ReadData(Scanner& infile);

void RunSimulationPct(const Configuration& config, MyRandom& random, ofstream& out_stream);

string ToString();

string ToStringVoterMap(string label, multimap<int, OneVoter> themap);

ReadData() - called by the Simulation class and creates a precinct with data from an input file.

RunSimulationPct() - sets a min and max station count, runs a simulation, then calls the private member function DoStatistics().

ToString() - outputs a string of data involving a precinct.

ToStringVoterMap() - outputs the map of voters with information for each voter

4.4 Private Functions:

void CreateVoters(const Configuration& config, MyRandom& random, ofstream& out_stream);

int DoStatistics(int iteration, const Configuration& config, int station_count,
map<int, int>& map_for_histo, ofstream& out_stream);

void ComputeMeanAndDev();

void RunSimulationPct2(int stations);

CreateVoters() - creates voters based on data from a precinct.

DoStatistics() - calculates how many voters had to wait too long as well as the mean and deviation of the wait time via ComputeMeanAndDev().

ComputeMeanAndDev() - calculates the mean wait time and deviation of the mean.

RunSimulationPct2() - assigns voters to voting booths and records which booth they used. Repeats if voters are still pending.

4.5 Creating Precint File: (filename.txt)

The OnePct class is used to hold the data for any given precinct that the simulation runs. As such, the file needs to have the information that was recorded at each precinct so that it can do calculations and compare them to the values from the configuration class. Like the Configration file, the OnePct file is delimited by spaces; thus, the format for the file is as shown, using ';' to represent the whitespace.

One Line: pct_number_;pct_name_;pct_turnout_;pct_num_voters;pct_expected_voters_;

pct_expected_per_hour;pct_stations_;pct_minority_;stat1;stat2;stat3

(The pct_turnout and pct_minority should be doubles, pct_name is a string; all other values are integers)

Example: 000 XXX00000 19.2 10101 0 235 8 10.1 0 0 0

This is repeated for each precinct that the simulation needs to run through.

5. OneVoter .cc .h:

The OneVoter class is where all the information that pertains to a voter is created and saved. It uses values specified in the configuration and OnePct classes to generate random voters for use in calculations.

5.1 Parameterized Constructor:

OneVoter(int sequence, int arrival_seconds, int duration_seconds);

5.2 Private Variables:

int sequence_;

int time_arrival_seconds_;

int time_done_voting_seconds_;

int time_start_voting_seconds_;

int time_vote_duration_seconds_;

int time_waiting_seconds_;

int which_station_;

5.3 Accessors:

int GetStationNumber() const - returns the station in use by a voter.

int GetTimeArrival() const - returns the arrival time of a voter in seconds.

int GetTimeDoneVoting() const - returns the sum of the time started and the duration spent by a voter.

int GetTimeWaiting() const - returns the amount of time a voter has been waiting in seconds.

int GetTimeInQ() const - returns the start time minus the arrival times.

5.4 Public Functions:

void AssignStation(int station_number, int start_time_seconds);

void DoneVoting();

string ToString();

static string ToStringHeader();

AssignStation() - called by OneVoter::RunSimulationPct2(), a voter is assigned to a free station.

DoneVoting() - find done voting time, never used (?)

ToString() - outputs data for a single voter.

ToStringHeader() - creates a header for the ToString() member function.

5.5 Private Functions:

string ConvertTime(int time_in_seconds) const;

string GetTOD(int time) const;

ConvertTime() - returns time from seconds to hour:minute:second format.

GetTOD() - returns the formatted time of day.

6. MyRandom .cc .h:

This class contains 4 member functions used for generating random numbers based on given parameters. If a seed is not passed as a parameter, the default constructor assigns the seed value to 1.

6.1 Private Variables:

int seed_;

6.2 Default Constructor:

MyRandom();

6.3 Parameterized Constructor:

MyRandom(unsigned seed);

6.4 Public Functions:

int RandomExponentialInt(double mean);

double RandomNormal(double mean, double dev);

double RandomUniformDouble(double lower, double upper);

int RandomUniformInt(int lower, int upper);

int RandomExponentialInt() - Pick a Random Integer from a Exponential Distribution.

double RandomNormal() - Pick a Random Double from a Normal Distribution.

double RandomUniformDouble() - Pick a Random Double from a Uniform Distribution.

int RandomUniformInt() - Pick a Random Integer from a Uniform Distribution.

Utilities:

7. Scanline .cc .h:

Scanline uses takes lines supplied by the scanner and has the ability to tokenize them. This is useful when reading in the precinct file. While this program does not utilize all the available member functions, they may prove handy in later program builds.

7.1 Public Functions:

bool HasMoreData();

bool HasNext();

void OpenString();

string Next();

double NextDouble();

int NextInt();

string NextLine();

LONG NextLONG();

HasMoreData() - returns true if there are any characters left in input string.

HasNext() - returns true if there are any characters left in input string.

OpenString() - allows string to be read through, spliting by deliminator.

Next() - returns anything until it hits white space, which can be carriage, new line, return, or tab.

NextDouble() - returns next double until it hits whitespace, crashes if a double is not the next token.

NextInt() - returns next int until it hits whitespace, crashes if an integer is not the next token.

NextLine() - returns the rest of the line.

NextLONG() - returns the next LONG until it hits whitespace, crashes if a LONG is not the next token.

8. Scanner .cc .h:

The scanner class provides tools for reading in .txt files line by line. These lines are then ‘opened’ by Scanline in order to be tokenized. As with Scanline, not all of these member functions are used but may prove handy in later program builds.

8.1 Public Functions:

void Close();

bool HasNext();

string Next();

double NextDouble();

int NextInt();

string NextLine();

LONG NextLONG();

void OpenFile();

Close() - closes scanner stream.

HasNext() - returns true if there is anything left in the file.

Next() - returns next token that is not whitespace. Whitespace can be carriage, new line, return, or tab.

NextDouble() - returns next double until it hits whitespace, crashes if a double is not the next token.

NextInt() - returns next int until it hits whitespace, crashes if an integer is not the next token.

NextLine() - returns the rest of the line.

NextLONG() - returns the next LONG until it hits whitespace, crashes if a LONG is not the next token.

OpenFile() - allows a file to be opened as a "Scanner."

9. Utils .cc .h:

The Utils class houses a multitude of member functions all surrounding handling of input and output. This program utilizes a variety of the output formatting functions as well as the stream functions listed below. Also included is an argument checking function.

9.1 Public Functions:

void Utils::FileOpen(std::ifstream& in_stream, std::string filename);

void Utils::FileOpen(std::ofstream& out_stream, std::string filename);

Utils::FileOpen() - an overloaded function that opens an input or output file based on the parameters passed.

Maintenance Guide Authors:

C.J. Waldron Abraham Khan

Stephen Page Thomas O'Hara

Program Author:

Duncan Buell