```cpp
#include <cmath>
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <string>

using namespace std;
class ElectricWaterHeater {
  /*!
    class described in detail in documentation
    */

private:
  int TIMESTEP = 15; // TMESTEP OF THE SIMULATION (MINUTES)

  const float rho = 997.; // water density (kg / m3)
  const float Cp = 4186.; // water specific heat (J/ kg)

  bool is_heating = false; // initntial state: not heating
  bool is_weekend; // = true for water consumption profile during the
weekends
  int number_of_users_;       // number of people using the heater
  int number_of_showers_ = 1; // number of showers available for use
  float Q_ = 0.;              // Q used to heat the water (W)
  float max_Q_; // max Q used to het the water during current Xmin
timestep of
                // the simulation
  float time_of_max_Q_;             // minute at which max_Q is first used
  float T_;                         // current temperture of the water
(◦C)
  float water_consumption[24 * 60]; // array desrbining water consumption
for
                                    // 24hours for each minute (l/min)
  float V_out_;       // flow of water because of water consumption (l/
min)
  float total_V_out_; // total water consumption during Xmin timestep (l)
  float power_;       // electrical power of the heater (W)
  float Tset_;        // set temperature of water in the tank (◦C)
  float S_;           // surface of the tank (m2)
  float V_;           // volume of the tank(liters)
  float Text_;        // external temperature outside the tank (◦C)
  float R_;           // tank thermal resistance (m2 * (◦C) / W);
  float Tin_;         // temperature of feed water(◦C)
  float efficiency_;  // efficiency of electrical heating (from 0 to 1)
  float deadband_;    // temperature band for controling the heater(◦C)
  float energy_;   // energy consumed by the heater during Xmin timestep
(kWh)
  int start_time_; // from 0 to 24*4;
  int time_;       // time of the simulation (minute from o to 24*4)
  bool water_usage_from_file_ =
      false;         // true if how water usage profile is read from a
file
  std::string path_; // path to HW usage profile
```

```
  float calculate_temperature(float timestep, float m_out, float Q) {
    float G = S_ / R_;
    float B = rho * m_out * Cp;
    // B: (density of water)*(water usage)*(specific heat of water);
    float C = V_ * rho * Cp;
    // C: (volume of tank)*(density of water)*(specific heat of water);
    float D = 1 / (B + G);

    // calculate Temperature
    T_ = T_ * exp(-(1 / (D * C)) * (timestep)) +
         (G * D * Text_ + B * D * Tin_ + Q * efficiency_ * D) *
             (1 - exp(-(1 / (D * C)) * (timestep)));

    return T_;
  }

  float calculate_power(float t) {

    // get water consumption for current minute of the day and change l/
min to
    // m3/second
    V_out_ = water_consumption[int(t)] / 60. / 1000.;

    // If the heating is OFF and the water temperature is LOWER than the
    // deadband than turn the heating ON. If the heating is ON and the
water
    // temperature is HIGHER than the deadband than turn the heating OFF
    if (is_heating) {
      if (T_ - Tset_ > deadband_) {
        is_heating = false;
      }

    } else {
      if (Tset_ - T_ > deadband_) {
        is_heating = true;
      }
    }

    if (is_heating) {
      Q_ = power_;
    } else {
      Q_ = 0.;
    }

    // calucalte temperature of water after 60 seconds
    T_ = calculate_temperature(60, V_out_, Q_);

    return Q_;
  }

  void build_water_consumption_profile(int number_of_occupants,
                                       bool is_weekend = false,
```

```
                                        int number_of_showers = 1) {

    for (int minute = 0; minute < 24 * 60; minute += 1) {
      water_consumption[minute] = 0;
    }
    if (water_usage_from_file_ == true) {

      std::ifstream file(path_);

      if (!file.is_open())
        std::cout << "Error while opening the file";
      int number_of_draws;
      int index;
      std::string number_of_draws_string;

      getline(file, number_of_draws_string, ';');
      number_of_draws = atof(number_of_draws_string.c_str());
      std::string auxiliary_array[number_of_draws * 2];

      int i = 0;
      while (file.good()) {
        for (i = 0; i < number_of_draws * 2; i++) {
          getline(file, auxiliary_array[i], ';');
        }
      }

      for (i = 0; i < number_of_draws; i++) {
        index = atof(auxiliary_array[i].c_str());
        i = i + 1;
        water_consumption[index] = atof(auxiliary_array[i].c_str());
      }

    } else {

      if (is_weekend == true) {

        int shower_start_time = 7 * 60 + (rand() % 90);
        for (int occupant = 1; occupant <= number_of_occupants; occupant
+= 1) {

          // shower
          int delay = 7 * int(occupant / number_of_showers);
          shower_start_time = shower_start_time + delay;
          for (int minute = shower_start_time; minute <= shower_start_time
+ 5;
               minute += 1) {
            water_consumption[minute] = water_consumption[minute] + 4;
          }

          // tap / kitchen sink
          for (int small_water_draw = 0; small_water_draw < 5;
               small_water_draw += 1) {
            int morning_use_time = shower_start_time + (rand() % 60);
```

```
            water_consumption[morning_use_time] =
                water_consumption[morning_use_time] +
                (0.5 + 0.5 * (rand() % 3));
          }

          // tap / kitchen sink
          for (int small_water_draw = 0; small_water_draw < 5;
               small_water_draw += 1) {
            int evening_use_time = 14 * 60 + (rand() % 360);
            water_consumption[evening_use_time] =
                water_consumption[evening_use_time] +
                (0.5 + 0.5 * (rand() % 3));
          }
        }

      } else {
        int shower_start_time = 7 * 60 + (rand() % 30);
        for (int occupant = 1; occupant <= number_of_occupants; occupant
+= 1) {
          // shower
          int delay = 7 * int(occupant / number_of_showers);
          shower_start_time = shower_start_time + delay;
          for (int minute = shower_start_time; minute <= shower_start_time
+ 5;
               minute += 1) {
            water_consumption[minute] = water_consumption[minute] + 4;
          }

          // tap / kitchen sink
          for (int small_water_draw = 0; small_water_draw < 5;
               small_water_draw += 1) {
            int morning_use_time = shower_start_time + (rand() % 60);
            water_consumption[morning_use_time] =
                water_consumption[morning_use_time] +
                (0.5 + 0.5 * (rand() % 3));
          }

          // tap / kitchen sink
          for (int small_water_draw = 0; small_water_draw < 5;
               small_water_draw += 1) {
            int evening_use_time = 18 * 60 + (rand() % 240);
            water_consumption[evening_use_time] =
                water_consumption[evening_use_time] +
                (0.5 + 0.5 * (rand() % 3));
          }
        }
      }
    }
  }

public:
  // Class default constructor
  ElectricWaterHeater() {
```

```
    power_ = 1500;
    Tset_ = 60;
    V_ = 60. / 1000.; // divison by 1000 converts liters to m3
    S_ = 0.9;
    Text_ = 22;
    Tin_ = 15;
    R_ = 0.9;
    efficiency_ = 0.98;
    deadband_ = 2.25;

    number_of_users_ = int(V_ / 25.);
    build_water_consumption_profile(number_of_users_);

    T_ = Tset_;
    start_time_ = 0;
  }

  // Class constructor
  ElectricWaterHeater(float power, float Tset = 60, float V = 60,
                      int number_of_users = 2, float S = 0.9, float Tin =
15,
                      float Text = 22, float R = 0.9, float efficiency =
0.98,
                      float deadband = 2.25) {
    power_ = power;
    Tset_ = Tset;
    S_ = S;
    V_ = V / 1000.; // divison by 1000 converts liters to m3
    number_of_users_ = number_of_users;
    Tin_ = Tin;
    Text_ = Text;
    R_ = R;
    efficiency_ = efficiency;
    deadband_ = deadband;

    build_water_consumption_profile(number_of_users);

    T_ = Tset_;
    start_time_ = 0;
  }

  void simulate_timestep(int timestep_number) {

    // reseting energy, max power and total water flow vairables
    energy_ = 0.;
    max_Q_ = 0.;
    total_V_out_ = 0.;
    time_of_max_Q_ = 0.;

    // starting minute (between 0-24*4)
    int t_start = (timestep_number - 1) * TIMESTEP + start_time_;
    // ending minute (between 0-24*4)
    int t_end = timestep_number * TIMESTEP + start_time_;
```

```
    // for each minute in TIMESTEP-minutes timestep
    for (int t = t_start; t < t_end; t += 1) {

      time_ = t;
      // calcualtes Q for given minute.
      Q_ = calculate_power(t);

      if (Q_ > max_Q_) {
        max_Q_ = Q_;
        time_of_max_Q_ = t + 1;
      }
      energy_ = energy_ + Q_ * 1. / 60. / 1000.;
      total_V_out_ = total_V_out_ + V_out_ * 60 * 1000;

      // decomment this line to print the behaviour of the heater minute
by
      // minute
      // cout << Q_ << ";" << T_ << ";" << V_out_ * 1000 * 60 << endl;
    }
  }

  // fucntions to get calcualted values
  float get_max_power() { return max_Q_; }
  float get_temperature() { return T_; }
  float get_water_flow() { return total_V_out_; }
  float get_energy_kWh() { return energy_; }
  float get_energy_joule() { return energy_ * 1000. * 3600.; }
  float get_current_minute_of_the_day() { return time_ + 1; }
  float get_minute_with_max_power() { return time_of_max_Q_; }

  // functions to set starting conditions
  void set_start_time(int hour, int quoter) {
    start_time_ = 60 * hour + 15 * (quoter - 1);
  }

  void set_water_usage_from_file(std::string path) {
    path_ = path;
    water_usage_from_file_ = true;
    build_water_consumption_profile(number_of_users_, is_weekend,
                                    number_of_showers_);
  }

  void water_usage_default() {
    water_usage_from_file_ = false;
    build_water_consumption_profile(number_of_users_, is_weekend,
                                    number_of_showers_);
  }

  void set_temperature(float T_input) { T_ = T_input; }

  void set_number_of_users(int number_of_users) {
    number_of_users_ = number_of_users;
```

```
    build_water_consumption_profile(number_of_users, is_weekend,
                                    number_of_showers_);
  }

  void set_is_it_weekend(bool is_it_weekend) {
    is_it_weekend = is_weekend;
    build_water_consumption_profile(number_of_users_, is_weekend,
                                    number_of_showers_);
  }

  void set_number_of_showers(bool number_of_showers) {
    number_of_showers_ = number_of_showers;
    build_water_consumption_profile(number_of_users_, is_weekend,
                                    number_of_showers_);
  }

  void set_set_temperature(float Tset) { Tset_ = Tset; }
  void set_power(float power) { power_ = power; }
  void set_deadband(float deadband) { deadband_ = deadband; }
  void set_volume(float volume) { V_ = volume / 1000; }
  void set_surface(float surface) { S_ = surface; }
  void set_resistance(float resistance) { R_ = resistance; }
  void set_efficiency(float efficiency) { efficiency_ = efficiency; }
  void set_external_temperature(float Text) { Text_ = Text; }
  void set_inlet_temperature(float Tin) { Tin_ = Tin; }
  void set_timestep(int timestep) { TIMESTEP = timestep; }
};

int main() {
  ElectricWaterHeater One(2000);
  One.set_timestep(15);
  One.set_set_temperature(60);
  One.set_volume(60);
  One.set_number_of_users(2);
  std::ofstream myfile;
  myfile.open("Water15min.csv");
  for (int iter = 1; iter <= 60 * 24; iter += 1) {
    One.simulate_timestep(iter);
    myfile << One.get_energy_joule() << " ; " << One.get_max_power() << "
; "
           << One.get_temperature() << " ; " << One.get_water_flow() <<
endl;
  }
  return 0;
}
```