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

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

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

  const float rho = 1.2; // air density (kg / m3)
  const float Cp = 1005; // air specific heat (J/ kg)

  bool is_heating = false; // initntial state: not heating
  bool has_thermostate_control_ =
      true;                     // true = there is a regualtor
                                // which turns the heater on only when it is
needed
  bool has_kitchen_ = false; // true if there is a kitchen (cooking) in
the
                             // heated space (affects ventilation)
  bool has_bathroom_ = false; // true if there is a bathroom (laundry) in
the
                              // heated space (affects ventilation)
  int number_of_inhabitants_; // number of people occupying the heated
space
  float Q_ = 0.; // Q used to heat the air (including Q_reduction) (W)
  float Inertia; // This paremter represents the inertia of the building
that
                 // counters heating and cooling of air inside the room
  float max_Q_;  // Maximal power used to heat the air during current Xmin
                 // timestep of the simualtion (W)
  float time_of_max_Q_; // time (minute) of the timestep when Qmax is
first used
  float T_;              // temperature of air inside the room (∘C)
  float V_out_; // volume of air that is exiting the room due to
ventilatiob
                // (m3/s)
  float total_V_out_; // total valume of air that is exciting the room in
a
                      // Xmin timestep of the simualtion (m3)
  float power_;       // power of the electrical space heater (W)
  float Tset_;        // set temperature of the air in the room (∘C)
  float Swall_;  // total surface of the walls (minus windows) THAT ARE
ADJECTED
                 // TO EXTERNAL TEMPERATURE! (m2)
  float Sfloor_; // total floor surface (m2)
  float Swindow_; // total surface of windows THAT IS ADJECTED TO EXTERNAL
```

```
                    // TEMPERATURE (m2)
  float Sroof_;     // total surface of roof THAT IS ADJECTED TO EXTERNAL
                    // TEMPERATURE (m2)
  float V_;         // total volume of the heated space(m3)
  float Text_;      // external temperature outside of the heated space(∘C)
  float ext_temp_profile[24]; // array desribing external temperature
hoour by
                              // hour for 24 hours(∘C)
  float Rwall_;               // walls thermal resistance (m2 ∗ (∘C) / W);
  float Rfloor_;              // floor thermal resistance (m2 ∗ (∘C) / W);
  float Rwindow_;             // window thermal resistance (m2 ∗ (∘C) /
W);
  float Rroof_;               // roof thermal resistance (m2 ∗ (∘C) / W);
  float efficiency_;          // efficienct of electric heating (from 0 to
1)
  float deadband_;            // temperature band to control the heating
(∘C)
  float energy_;              // energy consumed by the heater (kWh)
  int start_time_;            // from 0 to 24*4 (minute of the day)
  int time_; // current time of the simulation from 0 to 24*4 (minute of
the
             // day)
  bool ext_temp_from_file_ =
      false;          // true if external temperature profile is read from
a file
  std::string path_; // path to Text profile

  float calculate_temperature(float timestep, float m_out, float) {

    // calculate coefficients for Temperature Equation
    float G = Swall_ / Rwall_ + Swindow_ / Rwindow_ + Sfloor_ / Rfloor_ +
              Sroof_ / Rroof_;
    float B = rho * m_out * Cp;
    float C = V_ * rho * (Cp + Inertia);
    float D = 1 / (B + G);

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

    return T_;
  }

  float calculate_power(int t) {

    // If external temperature profile is loaded Text is taken from this
profile
    if (ext_temp_from_file_ == true) {
      int hour = int(t / 60);
      Text_ = ext_temp_profile[hour];
    }
```

```
    // If the heating is OFF and the air temperature is LOWER than the
    // deadband than turn the heating ON. If the heating is ON and the air
    // temperature is HIGHER than the deadband than turn the heating OFF
    if (is_heating) {
      if (T_ - Tset_ > deadband_) {
        is_heating = false;
        // heating reduction (heat inetria of the building) counters the
heat
        // losses
      }
    } else {
      if (Tset_ - T_ > deadband_) {
        is_heating = true;
        // heating reduction (heat inetria of the building) counters the
heating
      }
    }

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

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

    return Q_;
  }

  void calculate_air_circulation(float volume, int number_of_inhabitants,
                                 bool has_kitchen, bool has_bathroom) {
    // please note that all values are in m3/h
    V_out_ = 0;

    // ventialtion need for air change (simplified approach - 1 change per
hour)
    V_out_ = V_out_ + volume;

    // ventilation needed to provide confort for inhabitants
    V_out_ = V_out_ + Sfloor_ * 0.4 + 4 * number_of_inhabitants;

    // ventilation needed to remove CO2 produced by inhabitants
    V_out_ = V_out_ + 2.8 * number_of_inhabitants;

    // extra ventilation needed to remove humidity caused
    // by cooking in the kitchen
    if (has_kitchen == true) {
      V_out_ = V_out_ + 47;
```

```
    }

    // exra ventilation needed to remove remove humidty caused by laundry
    if (has_bathroom == true) {
      V_out_ = V_out_ + 78.6;
    }
    // conversion from m3/h to m3/s
    V_out_ = V_out_ / 3600;

    V_out_ = V_out_ * 0.35;
    // 0.35 is temperature difference reduction coefficient
    // It represents the assumption that air that enters through
ventilation is
    // not that cold and the air outside. (It is assumed that a heat
    // recuperation is used)
  }

  void calculate_inertia(float Sfloor) { Inertia = (22 * Sfloor * 60); }

public:
  // Class constructor
  ElectricSpaceHeater(float power, float Tset = 21, float V = 75,
                      int number_of_inhabitants = 2, float Text = 10,
                      float efficiency = 0.98, float deadband = 2.25,
                      bool has_thermostate_control = true) {
    power_ = power;
    Tset_ = Tset;
    V_ = V;
    number_of_inhabitants_ = number_of_inhabitants;
    Text_ = Text;
    efficiency_ = efficiency;
    deadband_ = deadband;
    has_thermostate_control_ = has_thermostate_control;

    calculate_air_circulation(V_, number_of_inhabitants, has_kitchen_,
                              has_bathroom_);

    T_ = Tset_;
    start_time_ = 0;
  }

  // Class default constructor
  ElectricSpaceHeater() {
    power_ = 1500;
    Tset_ = 21;
    V_ = 5 * 5 * 3;
    Text_ = 10;
    efficiency_ = 0.98;
    deadband_ = 2.25;
    number_of_inhabitants_ = 2;
    has_thermostate_control_ = true;

    calculate_air_circulation(V_, number_of_inhabitants_, has_kitchen_,
```

```
                                            has_bathroom_);

    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;

      // decomment this line to print heater behaviour minute by minute
during a
      // timestep
      cout << Q_ << " ; " << Inertia << ";" << T_ << ";" << V_out_ <<
endl;
    }
  }

  // fucntions to get calcualted values
  float get_max_power() { return max_Q_; }
  float get_temperature() { return T_; }
  float get_air_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_; }
  float get_power() { return power_; }

  // fucntion to calcualte actual resistance
  void create_wall_surface(float surface, float resistance = 1. / 0.3) {
```

```cpp
    Rwall_ = resistance; //   m2K/W / m2 = K/W
    Swall_ = surface;
  }

  void create_floor_surface(float surface, float resistance = 1. / 0.5) {
    Rfloor_ = resistance * 2;
    // 2 is temperature difference reduction coefficient
    // It represents the assumption that the temperature difference room -
    // ground is only  50% of the temperature difference room- outside
    Sfloor_ = surface;
    calculate_air_circulation(V_, number_of_inhabitants_, has_kitchen_,
                              has_bathroom_);
    calculate_inertia(Sfloor_);
  }

  void create_roof_surface(float surface, float resistance = 1. / 0.5) {
    Rroof_ = resistance; //   m2K/W
    Sroof_ = surface;
  }

  void create_window_surface(float surface, float resistance = 1. / 1.3) {
    Rwindow_ = resistance; //   m2K/W
    Swindow_ = surface;
  }

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

  void ext_temp_from_file(std::string path) {
    path_ = path;
    ext_temp_from_file_ = true;

    std::ifstream file(path_);

    if (!file.is_open())
      std::cout << "Error while reading file";
    int N = 24;
    std::string auxiliaryarray[N];

    int i = 0;
    while (file.good()) {
      for (i = 0; i < N; i++) {
        getline(file, auxiliaryarray[i], ';');
      }
    }

    for (i = 0; i < N; i++) {
      ext_temp_profile[i] = atof(auxiliaryarray[i].c_str());
    }
  }
```

```
  void ext_temp_default(float constant_T_ext) {
    constant_T_ext = Text_;
    ext_temp_from_file_ = false;
  }

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

  void set_number_of_inhabitants(int number_of_inhabitants) {
    number_of_inhabitants_ = number_of_inhabitants;
    calculate_air_circulation(V_, number_of_inhabitants, has_kitchen_,
                              has_bathroom_);
  }

  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; }
  void set_efficiency(float efficiency) { efficiency_ = efficiency; }
  void set_external_temperature(float Text) { Text_ = Text; }
  void set_has_kitchen(bool has_kitchen) { has_kitchen_ = has_kitchen; }
  void set_has_bathroom(bool has_bathroom) { has_bathroom_ = has_bathroom;
}
  void set_thermostate_control(bool has_thermostate_control) {
    has_thermostate_control_ = has_thermostate_control;
  }

  void set_timestep(int timestep) { TIMESTEP = timestep; }

  void find_correct_power(int power_search_step = 50) {
    bool memory = has_thermostate_control_;
    has_thermostate_control_ = false;
    int proposed_power = 0; // kWh
    float difference = 10000;
    float old_difference = 1000000;
    float start_temperature = get_temperature();
    while (difference < old_difference) {
      proposed_power = proposed_power + 50;
      power_ = proposed_power;
      old_difference = difference;
      T_ = Tset_;
      for (int iter = 1; iter <= 30; iter += 1) {
        simulate_timestep(iter);
      }
      float end_temperature = get_temperature();
      difference = abs(end_temperature - start_temperature);
    }
    power_ = power_ - power_search_step;
    T_ = Tset_;
    has_thermostate_control_ = memory;
  }
};

int main() {
```

```
  ElectricSpaceHeater Two(500, 21, 30);
  Two.set_timestep(1);
  Two.set_number_of_inhabitants(2);
  Two.set_deadband(2);
  Two.create_floor_surface(10);
  Two.create_wall_surface(20);
  Two.create_window_surface(2);
  Two.create_roof_surface(0);
  Two.set_external_temperature(0);
  std::ofstream myfile;
  myfile.open("ESH1final.csv");
  for (int iter = 1; iter <= 60 * 24; iter += 1) {
    Two.simulate_timestep(iter);
    myfile << Two.get_energy_joule() << " ; " << Two.get_max_power() << "
; "
           << Two.get_temperature() << " ; " << Two.get_air_flow() << " ;
"
           << endl;
  }
  return 0;
}
```