```
    1: /*
    2:  * Time slice based calculation of a combustion engine;
    3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
    4:  *
    5:  * This program is free software; you can redistribute it and/or modify
    6:  * it under the terms of the GNU General Public License as published by
    7:  * the Free Software Foundation; either version 2 of the License, or
    8:  * (at your option) any later version.
    9:  *
   10:  * This program is distributed in the hope that it will be useful,
   11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
   12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   13:  * GNU General Public License for more details.
   14:  *
   15:  * You should have received a copy of the GNU General Public License along
   16:  * with this program; if not, write to the Free Software Foundation, Inc.,
   17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
   18:  *
   19:  */
   20:
   21: #include "combustionengine.h"
   22:
   23: CombustionEngine* CombustionEngine::_pInst = NULL;
   24:
   25: CombustionEngine* CombustionEngine::getInst(){
   26:         if (_pInst == NULL){
   27:                 _pInst = new CombustionEngine();
   28:         }
   29:         return _pInst;
   30: }
   31:
   32: CombustionEngine::CombustionEngine() {
   33:         int i = 0;
   34:         _oil = Oil();
   35:         _ecu = Ecu();
   36:         _w = 0.0;
   37:         _cnt = 0;
   38:         _intake = GasComponent(V_intake, Environment::getInst()->getAmbientAir()->getT(),
   39:                         Environment::getInst()->getAmbientAir()->getP(), Environment::getInst()->getAmbient
Air()->getNu());
   40:         _exhaust = GasComponent(V_exhaust, Environment::getInst()->getAmbientAir()->getT(),
   41:                         Environment::getInst()->getAmbientAir()->getP(), Environment::getInst()->getAmbient
Air()->getNu());
   42: //      _thrPos = 0.0;
   43:         _T_CW = 340.0;
   44:         _M_Shaft = 0.0;
   45:         for (i = 0; i < Ncyl; i++) {
   46:                 _phiCyl[i] = -4*M_PI/Ncyl*(double)i;
   47:                 _cyl[i] = Cylinder(_phiCyl[i], &_intake, &_exhaust, &_ecu, &_oil);
   48:         }
   49:
   50: }
   51: /**
   52:  * Parameters:
   53:  * w ... speed of rotation in [rad/s];
   54:  * thrPos ... throttle position, absolute value ==> [0..1]
   55:  */
   56: void CombustionEngine::run(double w, double thrPos) {
   57:         int i = 0;
   58:         _ecu.setThrottlePosition(thrPos, w);
   59:         _M_Shaft = 0.0;
   60:         double dphi = (_w + w)/2.0 * Ts;
   61:         _w = w;
   62:         _cnt++;
   63:         for (i = 0; i < Ncyl; i++) {
   64:                 _cyl[i].calcCylinder(dphi);
   65:                 _M_Shaft += _cyl[i].getM_G() + _cyl[i].getM_P();
   66:         }
   67:         Environment::getInst()->getAmbientAir()->calcGasExchange(A_intake, &_intake);
   68:         //_exhaust.calcGasExchange(A_exhaust, Environment::getInst()->getExhaustGas());
   69:         Environment::getInst()->getExhaustGas()->calcGasExchange(A_exhaust, &_exhaust);
   70: }
   71:
   72: void CombustionEngine::setPhiSpark(double sparkAngle) {
   73:         _ecu.setPhiSpark(sparkAngle);
   74: }
   75:
   76: void CombustionEngine::setPhiInjection(double injectionAngle){
   77:         _ecu.setPhiInjection(injectionAngle);
   78: }
   79:
   80:
   81: const GasComponent& CombustionEngine::getExhaust() const {
```

```
   82:         return _exhaust;
   83: }
   84:
   85: const GasComponent& CombustionEngine::getIntake() const {
   86:         return _intake;
   87: }
   88:
   89: double CombustionEngine::getThrPos() const {
   90:         return _ecu.getThrottlePosition();
   91: }
   92:
   93: double CombustionEngine::getW() const {
   94:         return _w;
   95: }
   96:
   97: double CombustionEngine::getMShaft() const {
   98:         return _M_Shaft;
   99: }
  100:
  101: double CombustionEngine::getPhi() const {
  102:         return _cyl[0].getPhi();
  103: }
  104:
  105: double CombustionEngine::getp_Cyl(int cylNum) const {
  106:         return _cyl[cylNum].getPressure();
  107: }
  108:
  109: double CombustionEngine::getT_Cyl(int cylNum) const {
  110:         return _cyl[cylNum].getTemperature();
  111: }
  112:
  113: double CombustionEngine::getCyl_T(int cylNum) const {
  114:         return _cyl[cylNum].getT_Cyl();
  115: }
  116:
  117: double CombustionEngine::getFuelConsumption() const {
  118:         return _ecu.getFuelConsumed();
  119: }
  120:
  121: double CombustionEngine::getH_Cooling() const {
  122:         double result = 0.0;
  123:         for (int i = 0; i < Ncyl; i++) {
  124:                 result += _cyl[i].getH_Cool();
  125:         }
  126:         return result;
  127: }
  128:
  129: unsigned long CombustionEngine::getCnt() const {
  130:                 return _cnt;
  131:         }
  132:
  133: void CombustionEngine::setTempCoolingWater(double T_CW){
  134:         _T_CW = T_CW;
  135:         for (int i = 0; i < Ncyl; i++) {
  136:                 _cyl[i].setT_CW(_T_CW);
  137:         }
  138: }
  139:
  140:
```

```
 1: /*
 2:  * Time slice based calculation of a combustion engine;
 3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
 4:  *
 5:  * This program is free software; you can redistribute it and/or modify
 6:  * it under the terms of the GNU General Public License as published by
 7:  * the Free Software Foundation; either version 2 of the License, or
 8:  * (at your option) any later version.
 9:  *
10:  * This program is distributed in the hope that it will be useful,
11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13:  * GNU General Public License for more details.
14:  *
15:  * You should have received a copy of the GNU General Public License along
16:  * with this program; if not, write to the Free Software Foundation, Inc.,
17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
18:  *
19:  */
20:
21: #ifndef ENGINE_H
22: #define ENGINE_H
23:
24: #include "definitions.h"
25: #include "cylinder.h"
26: #include "ecu.h"
27: #include "oil.h"
28: #include "environment.h"
29:
30: class CombustionEngine
31: {
32: public:
33:         static CombustionEngine *getInst();
34:         /** Method: run(w, thrPos)
35:          * Parameters:
36:          * w ... speed of rotation in [rad/s];
37:          * thrPos ... throttle position, absolute value ==> [0..1]
38:          */
39:         void run(double w, double thrPos);
40:         void setPhiSpark(double sparkAngle);
41:         void setPhiInjection(double injectionAngle);
42:         void setTempCoolingWater(double T_CW);
43:         const GasComponent& getExhaust() const;
44:         const GasComponent& getIntake() const;
45:         double getThrPos() const;
46:         double getW() const;
47:         double getMShaft() const;
48:         double getPhi() const;
49:         double getp_Cyl(int cylNum) const;
50:         double getT_Cyl(int cylNum) const;
51:         double getCyl_T(int cylNum) const;
52:         double getFuelConsumption() const;
53:         double getH_Cooling() const;
54:         unsigned long getCnt() const;
55:
56: protected:
57: private:
58:         CombustionEngine();
59:         static CombustionEngine* _pInst;
60:
61:         Ecu _ecu;
62:         Oil _oil;
63:         double _w;
64:         Cylinder _cyl[Ncyl];
65:         double _phiCyl[Ncyl]; // angle offset off cylinders
66:         GasComponent _intake;
67:         GasComponent _exhaust;
68:         double _T_CW;
69:         double _M_Shaft;
70:         unsigned long _cnt;
71:
72: };
73:
74: #endif // ENGINE_H
```

```
 1: /*
 2:  * <one line to give the program's name and a brief idea of what it does.>
 3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
 4:  *
 5:  * This program is free software; you can redistribute it and/or modify
 6:  * it under the terms of the GNU General Public License as published by
 7:  * the Free Software Foundation; either version 2 of the License, or
 8:  * (at your option) any later version.
 9:  *
10:  * This program is distributed in the hope that it will be useful,
11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13:  * GNU General Public License for more details.
14:  *
15:  * You should have received a copy of the GNU General Public License along
16:  * with this program; if not, write to the Free Software Foundation, Inc.,
17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
18:  *
19:  */
20:
21: #include "cylinder.h"
22:
23: Cylinder::Cylinder(){
24:         _phi = 0.0;
25:         _x_p = 0.0;
26:         _dx_p = 0.0;
27:         _v_p = 0.0;
28:         _M_p = 0.0;
29:         _M_g = 0.0;
30:         _F_fr = 0.0;
31:         _n_Fuel = 0.0;
32:         _H_cooling = 0.0;
33:         _H_hx_gas = 0.0;
34:         _T_cyl = T_ref;
35:         _T_CW = T_ref;
36:         _gc = GasComponent(Vcyl*(1+1/chi), T_ref, p_ref);
37:         _pinlet = NULL;
38:         _pexhaust = NULL;
39:         _pecu = NULL;
40: }
41:
42: Cylinder::Cylinder(double phi0, GasComponent *pinlet, GasComponent *pexhaust, Ecu *pecu, Oil *poil){
43:         _phi = phi0;
44:         _x_p = 0.0;
45:         _dx_p = 0.0;
46:         _v_p = 0.0;
47:         _M_p = 0.0;
48:         _M_g = 0.0;
49:         _F_fr = 0.0;
50:         _n_Fuel = 0.0;
51:         _H_cooling = 0.0;
52:         _H_hx_gas = 0.0;
53:         _T_cyl = T_ref;
54:         _T_CW = T_ref;
55:         _gc = GasComponent(Vcyl*(1+1/chi), T_ref, p_ref);
56:         _pinlet = pinlet;
57:         _pexhaust = pexhaust;
58:         _pecu = pecu;
59:         _poil = poil;
60: }
61:
62: void Cylinder::setT_CW(double T_CW) {
63:         _T_CW = T_CW;
64: }
65:
66: double Cylinder::getH_Cool() const {
67:         return _H_cooling;
68: }
69:
70: double Cylinder::getX_p() const {
71:         return _x_p;
72: }
73:
74: double Cylinder::getdX_p() const {
75:         return _dx_p;
76: }
77:
78: double Cylinder::getV_p() const {
79:         return _v_p;
80: }
81:
82: double Cylinder::getdV_p() const {
83:         return _dv_p;
```

```
 84: }
 85:
 86: double Cylinder::getM_G() const {
 87:         return _M_g;
 88: }
 89:
 90: double Cylinder::getM_P() const {
 91:         return _M_p;
 92: }
 93:
 94: double Cylinder::getT_Cyl() const {
 95:         return _T_cyl;
 96: }
 97:
 98: double Cylinder::getPhi() const {
 99:         return _phi;
100: }
101:
102: double Cylinder::getPressure() const {
103:         return _gc.getP();
104: }
105:
106: double Cylinder::getTemperature() const {
107:         return _gc.getT();
108: }
109:
110: void Cylinder::calcCylinder(double dPhi){
111:         _phi += dPhi;
112:         if(_phi >= 4.0*M_PI){
113:                 _phi -= 4.0*M_PI;
114:         }
115:         if(_phi < 0.0){
116:                 _phi += 4.0*M_PI;
117:         }
118:         _dx_p = r_cs*( sqrt(pow(l2r,2.0) - pow(sin(_phi),2.0)) - cos(_phi) -(l2r-1.0)) -_x_p; // change in
x pos (from 0)
119:         _x_p += _dx_p;
120:         _dv_p = _v_p - _dx_p/Ts;
121:         _v_p = _dx_p/Ts;
122:         //force of friction: F = eta(T) * A * v/d; M = F*r(phi)
123:         _F_fr = _poil->getEta(_T_cyl)* 2*r_cs*M_PI*h_Piston * _v_p / d_Piston;
124:         _M_p = m_Piston*_dv_p / Ts * r_cs * sin(_phi)- fabs(_F_fr * r_cs*sin(_phi)); // speed dep. && frict
ion
125:         if(passedAngle(_pecu->getPhiInjection(), dPhi)){
126:                 _n_Fuel = _pecu->fillInjector(_pinlet->getP(), _pinlet->getT());
127:         }
128:         if(passedAngle(_pecu->getPhiSpark(), dPhi)){
129:                 _gc.setCombustionStarted(true);
130:         }
131:         _gc.calcGasExchange(_pecu->getValveOut_A(_phi), _pexhaust);
132:         _pinlet->calcGasExchange(_pecu->getValveIn_A(_phi), &_gc);
133:         _gc.calcStateChange(getCmpFactor() , getHeatExchangeEnthalpy(), calcFuelInj());//, *_pinlet, *_pexh
aust);
134:         _M_g = -ACyl*(_gc.getP() - Environment::getInst()->getAmbientAir()->getP())*r_cs*sin(_phi);
135: }
136:
137: double Cylinder::getCylArea(double x_pos) const {
138:         return 2*ACyl*(1.0 + (hCyl-_x_p));
139: }
140:
141: /*
142:  * Heat exchange
143:  *  gas <-> cyl. wall <-> cooling water
144:  */
145: double Cylinder::getHeatExchangeEnthalpy(){
146:         double hx_factor = _gc.getspecV()/(R*T_ref/p_ref) * getCylArea(hCyl - _x_p); // v_ref/v(p,T) * A [m
3/mol / m3/mol * m2]
147:         _H_cooling = hx_a_CW * (_T_cyl - _T_CW)*Ts;
148:         _H_hx_gas = hx_a_CG* hx_factor * (_T_cyl - _gc.getT())*Ts;
149:         _T_cyl += (-_H_hx_gas - _H_cooling + fabs(_F_fr*_dx_p))/hx_C_W; // gas hx && friction
150:         return _H_hx_gas;
151: }
152:
153: double Cylinder::getCmpFactor() const {
154:         return 1.0/ (1.0 + _dx_p/(hCyl-_x_p));
155: }
156:
157: bool Cylinder::passedAngle(double alpha, double dphi) const {
158:         return (alpha < _phi && _phi <= alpha + dphi);
159: }
160:
161: double Cylinder::getHxGas() const {
162:         return _H_hx_gas;
```

```
163: }
164: /**
165:  * calc the mols of fuel to be injected
166:  */
167: double Cylinder::calcFuelInj() {
168:         double result = 0.0;
169:         if(_n_Fuel > EPSILON && _phi >= _pecu->getPhiInjection()){
170:                 if(_n_Fuel > Fuel_n_Inject){
171:                         result = Fuel_n_Inject;
172:                         _n_Fuel -= Fuel_n_Inject;
173:                 }else{
174:                         result = _n_Fuel;
175:                         _n_Fuel = 0.0;
176:                 }
177:         }
178:         return result;
179: }
180:
```

```
    1: /*
    2:  * Time slice based calculation of a combustion engine;
    3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
    4:  *
    5:  * This program is free software; you can redistribute it and/or modify
    6:  * it under the terms of the GNU General Public License as published by
    7:  * the Free Software Foundation; either version 2 of the License, or
    8:  * (at your option) any later version.
    9:  *
   10:  * This program is distributed in the hope that it will be useful,
   11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
   12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   13:  * GNU General Public License for more details.
   14:  *
   15:  * You should have received a copy of the GNU General Public License along
   16:  * with this program; if not, write to the Free Software Foundation, Inc.,
   17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
   18:  *
   19:  */
   20:
   21: #ifndef CYLINDER_H
   22: #define CYLINDER_H
   23:
   24: #include "definitions.h"
   25: #include "gascomponent.h"
   26: #include "ecu.h"
   27: #include "environment.h"
   28: #include "oil.h"
   29:
   30: class Cylinder
   31: {
   32: public:
   33:         Cylinder();
   34:     Cylinder(double phi0, GasComponent *pinlet, GasComponent *pexhaust, Ecu *pecu, Oil *poil);
   35:
   36:
   37:     void setT_CW(double T_CW);
   38:
   39:     void calcCylinder(double dPhi);
   40:
   41:         double getH_Cool() const;
   42:         double getM_G() const;
   43:         double getM_P() const;
   44:
   45:         double getX_p() const;
   46:         double getdX_p() const;
   47:         double getV_p() const;
   48:         double getdV_p() const;
   49:
   50:         double getT_Cyl() const;
   51:         double getPhi() const;
   52:         double getHxGas() const;
   53:         double getPressure() const;
   54:         double getTemperature() const;
   55:
   56: private:
   57:     double _phi, _x_p, _dx_p, _v_p, _dv_p; //cs angle, abs. pos of piston, delta pos, abs. velocity, delta
in velocity[m/s];
   58:     double _M_p, _M_g, _F_fr;
   59:     double _H_cooling, _T_CW, _T_cyl, _H_hx_gas; // to water [J], [K], [K]
   60:     double _n_Fuel;
   61:     GasComponent _gc;
   62:     GasComponent *_pinlet, *_pexhaust;
   63:     Ecu *_pecu;
   64:     Oil *_poil;
   65:
   66:     double getHeatExchangeEnthalpy();
   67:     double getCylArea(double x_pos) const;
   68:     double getCmpFactor() const;
   69:     bool passedAngle(double alpha, double dphi) const;
   70:     double calcFuelInj();
   71: };
   72:
   73: #endif // CYLINDER_H
```

```
 1: /*
 2:  * <one line to give the program's name and a brief idea of what it does.>
 3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
 4:  *
 5:  * This program is free software; you can redistribute it and/or modify
 6:  * it under the terms of the GNU General Public License as published by
 7:  * the Free Software Foundation; either version 2 of the License, or
 8:  * (at your option) any later version.
 9:  *
10:  * This program is distributed in the hope that it will be useful,
11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13:  * GNU General Public License for more details.
14:  *
15:  * You should have received a copy of the GNU General Public License along
16:  * with this program; if not, write to the Free Software Foundation, Inc.,
17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
18:  *
19:  */
20:
21: #include "definitions.h"
22:
```

```
    1: /*
    2:  * <one line to give the program's name and a brief idea of what it does.>
    3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
    4:  *
    5:  * This program is free software; you can redistribute it and/or modify
    6:  * it under the terms of the GNU General Public License as published by
    7:  * the Free Software Foundation; either version 2 of the License, or
    8:  * (at your option) any later version.
    9:  *
   10:  * This program is distributed in the hope that it will be useful,
   11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
   12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   13:  * GNU General Public License for more details.
   14:  *
   15:  * You should have received a copy of the GNU General Public License along
   16:  * with this program; if not, write to the Free Software Foundation, Inc.,
   17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
   18:  *
   19:  */
   20:
   21: #ifndef DEFINITIONS_H
   22: #define DEFINITIONS_H
   23:
   24: //#include <stdio.h>
   25: #include <math.h>
   26: #include "shomate.h"
   27:
   28: // calc consts
   29: const double Ts = 10.0*pow(10.0,-6.0); //[s] cycle time
   30: const double EPSILON = pow(10.0,-6.0); // max deviation of double numbers
   31:
   32: // engine consts
   33: // mechanical
   34: const int Ncyl = 1; // number of cylinders
   35: const double Vcyl = 0.5*pow(10.0,-3.0); //[m3] cylinder volume
   36: const double chi = 18.0; // compression factor
   37: const double w_engine_min = 150.0*M_PI/30.0; // minimum engine speed -- injection starting point
   38: const double w_engine_max = 12000.0*M_PI/30.0; // maximum engine speed -- injection end point
   39: const double l2r = 2.5; //[m/m] pleuel to crank shaft radius
   40: const double r_cs = pow(Vcyl/(2.0*M_PI),1.0/3.0);// crank shaft radius =(Vcyl/2pi)^1/3: V_h(r^2 pi 2r);
   41: const double m_Piston = 0.35; // [kg] piston mass
   42: const double h_Piston = 2.0*r_cs; // height of piston (https://www.thm.de/me/images/user/herzog-91/Kolbenma
schinen/Kolbenmaschinen_6_Konstruktionselemente.pdf)
   43: const double d_Piston = pow(10.0,-4.0); // distance between piston and cylinder [m];
   44: const double ACyl = pow(r_cs,2.0) * M_PI;
   45: const double hCyl = 2.0*r_cs*(1.0+1.0/chi); //[m] height from lower dead center to cylinder head
   46: // fluid dynamics
   47: const double k_Aval = 0.2; //ratio A_valves/A_cyl
   48: const double A_Valve_out = k_Aval * ACyl; // cross section of outlet valves
   49: const double A_Valve_in = k_Aval * ACyl; // cross section of inlet valves
   50: const int num_Valve = 2; // number of inlet and outlet valve per cylinder;
   51:
   52: const double A_intake = A_Valve_in * (1.0+Ncyl/2.0); // cross section of intake
   53: const double A_exhaust = A_intake; // (min.) cross section of the exhaust gas pipe
   54: const double V_intake = Vcyl*(10.0+Ncyl*2.0); //[m3] volume of intake manifold
   55: const double V_exhaust = V_intake*2.0; //[m3] volume of exhaust pipe
   56: // heat transfer
   57: const double hx_a_CW = 400.0; // [J/(K s)] heat transfer coefficient cooling water <-> wall
   58: const double hx_a_CG = 2000.0; // [J/(K m2 s)] heat transfer coefficient wall <-> gas (delta T ~300K; 10kW
thermal flow rate)
   59: const double hx_C_W = 15000.0; // [J/K] heat capacity of wall
   60: //const double hx_gamma = hx_C_W / hx_a_CW; // ratio hx_C_W / hx_a_CW
   61: //const double hx_alpha = hx_a_CG / hx_a_CW; // ratio hx_a_WG / hx_a_CW
   62:
   63: // standard definitions
   64: const double R = 8.314462175; // [J/mol K]
   65: const double p_ref = pow(10.0,5.0); // [Pa]
   66: const double T_ref = 295.15;// [K]
   67: const double nu_Air[defs::Fuel+1]={0.7825, 0.2099, 0.0076, 0.0, 0.0, 0.0, 0.0, 0.0}; // [mol/mol] N2, O2, H
2O, CO2, CO
   68: const double eta_is = 0.95; // isentropic efficiency
   69: const double MolWeights[defs::Fuel+1] = {0.028, 0.032, 0.018, 0.044, 0.028, 0.002, 0.012, 0.114};//molare w
eight [kg/mol]
   70:
   71: // oil definitions
   72: // https://de.wikipedia.org/wiki/Viskosit%C3%A4t#Typische_Werte; last 2 values are "burnt oil" -- blocking
eng?!
   73: const double Oil_p[6] = {273.15 + 25.0, 0.1, 273.15 + 150.0, 0.003, 500.0, 10}; // T1, eta1, T2, eta2, end
of life...
   74:
   75: // fuel definitions
   76: //static const double Hf_ref = 50.0*1000.0*1000.0; // [J/kg]
   77:   //defined in ShData_Fuel: static const double H_fuel_form = -208700; // [J/mol] enthalpy of formation (ht
```

```
tp://webbook.nist.gov/cgi/cbook.cgi?ID=C111659&Units=SI&Mask=1)
   78: const double Fuel_n_C = 10.0;
   79: const double Fuel_O2_req = Fuel_n_C * 1.5 + 0.5; //[mol_O2/mol_Fuel] mols of O2 for stoichiometric reaction
 of an alkane
   80: // injection
   81: const double Fuel_n_Inject = nu_Air[defs::O2]*(p_ref*Vcyl/(R*T_ref))/Fuel_O2_req*(Ts/(2*pow(10.0,-3.0))); /
/[mol/s] molare amount injected per sample (def.: duration = 2 ms for "std filled" cyl)
   82: const double Fuel_T_Autoignition = 273.15 + 255.0; //https://de.wikipedia.org/wiki/Z%C3%BCndtemperatur
   83:
   84: // phys/chem/math consts [Fuel, H2O, CO, CO2]
   85: //const double ChemRectionRate[4] = {12.0, 40.0, 10.0, 5.0}; // was too slow at ~5000rpm
   86: const double ChemRectionRate[4] = {50.0, 40.0, 10.0, 3.0}; // reaction rate at T&p_ref
   87:
   88: //const double ChemRectionRate[4] = {0.40, 0.026, 0.026, 0.026}; // reaction rate at T&p_ref
   89: //const double ChemRectionRate[4] = {30.0, 30.0, 30.00, 15.0}; // reaction rate at T&p_ref
   90:
   91: const double AntoinePars[3] = {5.40221, 1838.675, -31.737}; // Antoine pars for 273 <= T <= 303 K
   92:
   93: const ShDataEntry ShData_N2[1] = {
   94:                 ShDataEntry(298.0, 6000.0,26.092, 8.218801, -1.976141, 0.159274, 0.044434, -7.98923, 221.02
, 0.0)};
   95: const ShDataEntry ShData_O2[1] = {
   96:                 ShDataEntry(298.0, 6000.0, 29.659, 6.137261, -1.186521, 0.09578, -0.219663, -9.861391, 237.
948, 0.0)};
   97: const ShDataEntry ShData_H2O[2] = {
   98:                 ShDataEntry(500.0, 1700.0, 30.092, 6.832514, 6.793435, -2.53448, 0.082139, -250.881, 223.39
67, -241826.4),
   99:                 ShDataEntry(1700.0, 6000.0, 41.96426, 8.622053, -1.49978, 0.098119, -11.15764, -272.1797, 2
19.7809, -241826.4)};
  100: const ShDataEntry ShData_CO2[2] = {
  101:                 ShDataEntry(298.0, 1200.0, 24.99735, 55.18696, -33.69137, 7.948387, -0.136638, -403.6075, 2
28.2431, -393522.4),
  102:                 ShDataEntry(1200.0, 6000.0, 58.16639, 2.720074, -0.492289, 0.038844, -6.447293, -425.9186,
263.6125, -393522.4)};
  103: const ShDataEntry ShData_CO[2] = {
  104:                 ShDataEntry(298.0, 1300.0, 25.56759, 6.09613, 4.054656, -2.671301, 0.131021, -118.0089, 227
.3665, -110527.1),
  105:                 ShDataEntry(1300.0, 6000.0, 35.1507, 1.300095, -0.205921, 0.01355, -3.28278, -127.8375, 231
.712, -110527.1)};
  106: const ShDataEntry ShData_Fuel[1] = {
  107:                 ShDataEntry(298.0, 6000.0, 351.455, 279.288, 0.0, 0.0, 0.0, 0.0, 0.0, -249700.0)}; // Decan
e; only Hf for C10H22, others for Octane
  108:                 //ShDataEntry(298.0, 6000.0, 351.455, 279.288, 0.0, 0.0, 0.0, 0.0, 0.0, -208700.0)}; //Octa
ne
  109:
  110:
  111: const ShData ShDataDB[defs::Fuel+1] = {ShData(1, ShData_N2), ShData(1, ShData_O2),
  112:                 ShData(2, ShData_H2O), ShData(2, ShData_CO2), ShData(2, ShData_CO),
  113:                 ShData(), ShData(), ShData(1, ShData_Fuel)
  114: };
  115:
  116: #endif // DEFINITIONS_H
```

```
 1: /*
 2:  * ecu.cpp
 3:  *
 4:  *  Created on: 23.11.2014
 5:  *      Author: alex
 6:  */
 7:
 8: #include "ecu.h"
 9:
10: Ecu::Ecu() {
11:         _phiSpark = (1.0 - 10.0/180.0)*M_PI;
12:         _phiInjection = (1.0 - 10.0/180.0)*M_PI;
13:         _phiValveOutOpen = (2.0 -       10.0/180.0)*M_PI;
14:         _phiValveOutClose = (3.0+       10.0/180.0)*M_PI;
15:         _phiValveInOpen = (3.0  -       10.0/180.0)*M_PI;
16:         _phiValveInClose = (4.0 +       10.0/180.0)*M_PI;
17:         _valve_exh = Valve(A_Valve_out, num_Valve,      _phiValveOutOpen, _phiValveOutClose);
18:         _valve_inl = Valve(A_Valve_in, num_Valve,       _phiValveInOpen, _phiValveInClose);
19:         _throttlePosition = 0.0;
20:         _w = 0.0;
21:         _n_fuel_consumed = 0.0;
22: }
23:
24: double Ecu::getValveOut_A(double phi){
25:         return _valve_exh.getCrosssection(phi);
26: }
27:
28: double Ecu::getValveIn_A(double phi){
29:         return _valve_inl.getCrosssection(phi);
30: }
31:
32: void Ecu::setPhiValveInClose(double phiValveInClose) {
33:         _phiValveInClose = phiValveInClose;
34: }
35:
36: void Ecu::setPhiValveInOpen(double phiValveInOpen) {
37:         _phiValveInOpen = phiValveInOpen;
38: }
39:
40: void Ecu::setPhiValveOutClose(double phiValveOutClose) {
41:         _phiValveOutClose = phiValveOutClose;
42: }
43:
44: void Ecu::setPhiValveOutOpen(double phiValveOutOpen) {
45:         _phiValveOutOpen = phiValveOutOpen;
46: }
47:
48:
49: /**
50:  * valve class should do this!!!
51:
52: double Ecu::getPhiValveInClose() const {
53:         return _phiValveInClose;
54: }
55:
56: double Ecu::getPhiValveInOpen() const {
57:         return _phiValveInOpen;
58: }
59:
60: double Ecu::getPhiValveOutClose() const {
61:         return _phiValveOutClose;
62: }
63:
64: double Ecu::getPhiValveOutOpen() const {
65:         return _phiValveOutOpen;
66: }*/
67:
68:
69: double Ecu::getPhiSpark() const {
70:         return _phiSpark;
71: }
72:
73: void Ecu::setPhiSpark(double sparkAngle) {
74:         if(sparkAngle >= -40.0 && sparkAngle <= 20.0){
75:                 _phiSpark = (1.0 + sparkAngle / 180.0)*M_PI;
76:         }
77: }
78:
79: double Ecu::getPhiInjection() const {
80:         return _phiInjection;
81: }
82:
83: void Ecu::setPhiInjection(double phiInjection) {
```

```
   84:         if(phiInjection >= -80.0 && phiInjection <= 20.0){
   85:                 _phiInjection = (1.0 + phiInjection / 180.0)*M_PI;
   86:         }
   87: }
   88:
   89: double Ecu::getThrottlePosition() const {
   90:         return _throttlePosition;
   91: }
   92:
   93: void Ecu::setThrottlePosition(double throttlePosition, double w) {
   94:         _w = w;
   95:         if(throttlePosition >= 0.0 && throttlePosition <= 1.0) {
   96:                 _throttlePosition = throttlePosition;
   97:         }
   98:         if( _w < w_engine_min || _w > w_engine_max){
   99:                 _throttlePosition = 0.0;
  100:         }
  101: }
  102:
  103: double Ecu::getFuelConsumed() const{
  104:         return _n_fuel_consumed;
  105: }
  106:
  107: /**
  108:  * mols of O2 in the "default filled cylinder"...
  109:  * _n_Fuel = n_O2 / (Fuel_O2_req * 1.05) * _pecu->getThrottlePosition();
  110:  */
  111: double Ecu::fillInjector(double p, double T){
  112:         double result = p*Vcyl/(R*T)*nu_Air[defs::O2]/ (Fuel_O2_req * 1.05) * _throttlePosition;
  113:         _n_fuel_consumed += result;
  114:         return result;
  115: }
```

```
 1: /*
 2:  * ecu.h
 3:  *
 4:  *  Created on: 23.11.2014
 5:  *      Author: alex
 6:  */
 7:
 8: #ifndef ECU_H_
 9: #define ECU_H_
10:
11: #include "definitions.h"
12: #include "valve.h"
13:
14: class Ecu{
15:
16: public:
17:         Ecu();
18:         double getValveOut_A(double phi);
19:         double getValveIn_A(double phi);
20: //      double getPhiValveInClose() const;
21:         void setPhiValveInClose(double phiValveInClose);
22: //      double getPhiValveInOpen() const;
23:         void setPhiValveInOpen(double phiValveInOpen);
24: //      double getPhiValveOutClose() const;
25:         void setPhiValveOutClose(double phiValveOutClose);
26: //      double getPhiValveOutOpen() const;
27:         void setPhiValveOutOpen(double phiValveOutOpen);
28:
29:         double getPhiSpark() const;
30:         void setPhiSpark(double sparkAngle);
31:         double getPhiInjection() const;
32:         void setPhiInjection(double phiInjection);
33:         double getThrottlePosition() const;
34:         void setThrottlePosition(double throttlePosition, double w);
35:         double getFuelConsumed() const; // get mols of fuel that were consumed
36:         double fillInjector(double p, double T); // consumes the fuel
37:
38: private:
39:         double _phiSpark;
40:     double _phiValveInOpen, _phiValveInClose, _phiValveOutOpen, _phiValveOutClose;
41:     double _phiInjection;
42:     double _throttlePosition;
43:         double _w;
44:     Valve _valve_exh, _valve_inl;
45:     double _n_fuel_consumed;
46:
47: };
48:
49:
50:
51: #endif /* ECU_H_ */
```

```
 1: /*
 2:  * Enviroment.cpp
 3:  *
 4:  *  Created on: 09.11.2014
 5:  *      Author: alex
 6:  */
 7:
 8: #include "environment.h"
 9:
10: Environment* Environment::_pinst = NULL;
11:
12: Environment::Environment() {
13:         _pAmbientAir = new GasComponent(pow(10.0,9.0), T_ref, p_ref, nu_Air, true);
14:         _pExhaustGas = new GasComponent(EPSILON, T_ref, p_ref, nu_Air, true);
15: }
16:
17: GasComponent* Environment::getAmbientAir(){
18:         return _pAmbientAir;
19: }
20: GasComponent* Environment::getExhaustGas(){
21:         return _pExhaustGas;
22: }
23:
24:
25: Environment* Environment::getInst() {
26:         if(!_pinst){
27:                 _pinst = new Environment;
28:         }
29:         return _pinst;
30: }
31:
```

```
 1: /*
 2:  * Enviroment.h
 3:  *
 4:  *  Created on: 09.11.2014
 5:  *      Author: alex
 6:  */
 7:
 8: #ifndef ENVIRONMENT_H_
 9: #define ENVIRONMENT_H_
10:
11: #include "definitions.h"
12: #include "gascomponent.h"
13:
14: class Environment {
15: public:
16:         static Environment* getInst();
17:         GasComponent* getAmbientAir();
18:         GasComponent* getExhaustGas();
19: /*      const GasComponent*& getAmbientAir() const;
20:         const GasComponent*& getExhaustGas() const;
21: */
22: private:
23:         Environment();
24:         static Environment* _pinst;
25:
26:         GasComponent* _pAmbientAir;
27:         GasComponent* _pExhaustGas;
28: };
29:
30:
31:
32: #endif /* ENVIRONMENT_H_ */
```

```
    1: /*
    2:  * <one line to give the program's name and a brief idea of what it does.>
    3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
    4:  *
    5:  * This program is free software; you can redistribute it and/or modify
    6:  * it under the terms of the GNU General Public License as published by
    7:  * the Free Software Foundation; either version 2 of the License, or
    8:  * (at your option) any later version.
    9:  *
   10:  * This program is distributed in the hope that it will be useful,
   11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
   12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   13:  * GNU General Public License for more details.
   14:  *
   15:  * You should have received a copy of the GNU General Public License along
   16:  * with this program; if not, write to the Free Software Foundation, Inc.,
   17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
   18:  *
   19:  */
   20:
   21: #include "gascomponent.h"
   22:
   23: /*
   24:  * returns: p [Pa];
   25:  * params: T [K], relHum [%];
   26:  */
   27: double getPressureFromRelHum(double T, double relHum){
   28:         return pow(10, 3 + AntoinePars[0] - AntoinePars[1]/(AntoinePars[2]+T)) * relHum; // [Pa] = 10^5 [ba
r] * [%] / 100
   29: }
   30:
   31: GasComponent::GasComponent(){
   32:         _T=T_ref;
   33:         _p=p_ref;
   34:         _V=1.0;
   35:         _n_g = calcMols();
   36:         _v=_V/_n_g;
   37:         int i = 0;
   38:         for (i = 0; i < defs::Fuel+1; i++) {
   39:                 _nu[i]=nu_Air[i];
   40:         }
   41:         _MW = calcMolareWeight();
   42:         _cp = Shomate::getInst()->getHeatCapacity(_T, _nu);
   43:         _H = _n_g * _cp * _T;
   44:         _combustionStarted = false;
   45:         _isContainer = false;
   46: }
   47:
   48: GasComponent::GasComponent(double V, double T, double p){
   49:         _T=T;
   50:         _p=p;
   51:         _V=V;
   52:         _n_g = calcMols();
   53:         _v=_V/_n_g;
   54:         int i = 0;
   55:         for (i = 0; i < defs::Fuel+1; i++) {
   56:                 _nu[i]=nu_Air[i];
   57:         }
   58:         _MW = calcMolareWeight();
   59:         _cp = Shomate::getInst()->getHeatCapacity(_T, _nu);
   60:         _H = _n_g * _cp * T;
   61:         _combustionStarted = false;
   62:         _isContainer = false;
   63: }
   64:
   65: GasComponent::GasComponent(double V, double T, double p, const double nu[defs::Fuel+1]){
   66:         _T=T;
   67:         _p=p;
   68:         _V=V;
   69:         _n_g = calcMols();
   70:         _v=_V/_n_g;
   71:         int i = 0;
   72:         for (i = 0; i < defs::Fuel+1; i++) {
   73:                 _nu[i]=nu[i];
   74:         }
   75:         normalizeMols();
   76:         _MW = calcMolareWeight();
   77:         _cp = Shomate::getInst()->getHeatCapacity(_T, _nu);
   78:         _H = _n_g * _cp * T;
   79:         _combustionStarted = false;
   80:         _isContainer = false;
   81: }
   82:
```

```
   83: GasComponent::GasComponent(double V, double T, double p, const double nu[defs::Fuel+1], bool isContainer){
   84:         _T=T;
   85:         _p=p;
   86:         _V=V;
   87:         _n_g = calcMols();
   88:         _v=_V/_n_g;
   89:         int i = 0;
   90:         for (i = 0; i < defs::Fuel+1; i++) {
   91:                 _nu[i]=nu[i];
   92:         }
   93:         normalizeMols();
   94:         _MW = calcMolareWeight();
   95:         _cp = Shomate::getInst()->getHeatCapacity(_T, _nu);
   96:         _H = _n_g * _cp * T;
   97:         _combustionStarted = false;
   98:         _isContainer = isContainer;
   99: }
  100:
  101:
  102: void GasComponent::calcGasExchange(double A_crosssection, GasComponent *pgc){
  103:         if(A_crosssection > 0 && fabs(_p - pgc->_p)>EPSILON){
  104:                 double deltaN = 0.0;
  105:                 if(_p > pgc->_p){
  106:                         deltaN = A_crosssection * pow( (_p*(_p - pgc->_p))/(2.0*_MW*R*_T) , 0.5)*Ts;
  107:                         pgc->transferFrom(deltaN, *this);
  108:                 }else{
  109:                         deltaN = A_crosssection * pow( (pgc->_p*(pgc->_p - _p))/(2.0*pgc->_MW*R*pgc->_T) ,
0.5)*Ts;
  110:                         transferFrom(deltaN, *pgc);
  111:                 }
  112:         }
  113: }
  114:
  115: void GasComponent::calcStateChange(double cmpFactor, double H_cooling, double n_Fuel){
  116:                 //,const GasComponent &inlet, GasComponent &exhaust){
  117:         double deltaH = H_cooling;
  118:         deltaH += isentropicStateChange(cmpFactor);
  119:         deltaH += injection(n_Fuel);
  120:         deltaH += chemReaction();
  121:         double dT_est = _T*deltaH/_H; // == dH/(n*cp)
  122:         _cp = Shomate::getInst()->getHeatCapacity(_T + dT_est, _nu);
  123:         _MW = getMolareWeight();
  124:         _H += deltaH;
  125:         _T = _H/(_n_g * _cp);
  126:         if(_T > Fuel_T_Autoignition) {
  127:                 _combustionStarted = true;
  128:         }
  129:         _p = R*_T/_v;
  130:
  131: }
  132:
  133: void GasComponent::setCombustionStarted(bool combustionStarted) {
  134:         _combustionStarted = combustionStarted;
  135: }
  136:
  137: bool GasComponent::isCombustionStarted() const {
  138:         return _combustionStarted;
  139: }
  140:
  141: double GasComponent::getSpecHeatCapacity() const {
  142:         return _cp;
  143: }
  144:
  145: double GasComponent::getEnthalpy() const {
  146:         return _H;
  147: }
  148:
  149: double GasComponent::getMolareWeight() const {
  150:         return _MW;
  151: }
  152:
  153: double GasComponent::getMols() const {
  154:         return _n_g;
  155: }
  156:
  157: const double* GasComponent::getNu() const {
  158:         return _nu;
  159: }
  160:
  161: double GasComponent::getP() const {
  162:         return _p;
  163: }
  164:
```

```
165: double GasComponent::getT() const {
166:         return _T;
167: }
168:
169: double GasComponent::getspecV() const {
170:         return _v;
171: }
172:
173: double GasComponent::getV() const {
174:         return _V;
175: }
176:
177: // --- private methods
178:
179: /*
180:  * cmpFactor: V_i/V_(i-1)
181:  */
182: double GasComponent::isentropicStateChange(double cmpFactor) {
183:         double deltaH = 0.0;
184:         if(fabs(cmpFactor-1.0)>EPSILON){
185:                 _V*=cmpFactor;
186:                 _v*=cmpFactor;
187:                 deltaH = _H*(pow(cmpFactor, 1.0/(1.0 - _cp/R)) - 1.0);
188:                 if(cmpFactor > 1.0){// expansion
189:                         deltaH *= eta_is;
190:                 }else{ // compression
191:                         deltaH /= eta_is;
192:                 }
193:         }
194:         return deltaH;
195: }
196:
197: /*double GasComponent::isochoricStateChange(double deltaH) {
198:         return deltaH;
199: }*/
200:
201: void GasComponent::transferFrom(double dn, GasComponent &gc) {
202:         if(dn > EPSILON && gc._n_g > dn){ //do not take it from an "near empty" component
203:                 // remove gas from gc
204:                 double dH = dn* gc._cp * gc._T;
205:                 double cmpFactor = 1 - dn/gc._n_g;
206:                 gc._H -= dH;
207:                 gc._n_g -= dn;
208:
209:                 if(!gc._isContainer){ //isentropic expansion (V const)
210:                         gc._H = gc._H * pow(cmpFactor, 1.0/(gc._cp/R - 1.0));
211:                         gc._T = gc._H/(gc._n_g * gc._cp);
212:                         gc._v = gc._V/gc._n_g;
213:                         gc._p = R*gc._T / gc._v;
214:                 }else{ //isobaric expansion (p,T,v const)
215:                         gc._V *= cmpFactor;
216:                 }
217:
218:                 // add to 'this'
219:                 _H += dH;
220:                 cmpFactor = dn/_n_g;
221:                 int i = 0;
222:                 for (i = 0; i < defs::Fuel+1; i++) {
223:                         _nu[i] = (_nu[i]+cmpFactor*gc._nu[i])/(1+cmpFactor);
224:                 }
225:                 _n_g += dn;
226:                 _MW = calcMolareWeight();
227:
228:                 if(!_isContainer){
229:                         _H = _H * pow( (1+cmpFactor) , 1.0/(_cp/R - 1.0));
230:                         _T = _H/(_n_g * _cp);
231:                         _v = _V/_n_g;
232:                         _p = R*_T / _v;
233:                 }else{
234:                         _V *= (1+cmpFactor);
235:                 }
236:                 _combustionStarted &= gc._combustionStarted;
237:
238:         }
239: }
240:
241: double GasComponent::injection(double n_Fuel){
242:         double deltaH = 0.0;
243:         double k = 1.0;
244:         if(n_Fuel > EPSILON){
245:                 deltaH = n_Fuel * (Shomate::getInst()->getFuelHeatCapacity(T_ref)*T_ref - Shomate::getInst(
)->getFuelHeatCapacity(_T)*_T);
246:                 k = n_Fuel/_n_g;
```

```
247:                 _nu[defs::Fuel] += k;
248:                 _p *= (1+k);
249:                 normalizeMols();
250:                 _v = _V / _n_g;
251:         }
252:         return deltaH;
253: }
254:
255: double GasComponent::chemReaction(){
256:         double deltaH = 0.0;
257:         if(_combustionStarted){
258:                 double k_Tpt =  _p/p_ref * exp( _T/T_ref - 1.0)*Ts;
259:                 if(k_Tpt < 0) k_Tpt = 0.0;
260:                 double k_O2 = sqrt(_nu[defs::O2]);
261:                 // Fuel ==> 8*C + 9*H2
262:                 _n_chemR[0] = ChemRectionRate[0]*k_Tpt*_nu[defs::Fuel];
263:                 // H2 + 1/2*O2 ==> H20
264:                 _n_chemR[1] = ChemRectionRate[1]*k_Tpt*_nu[defs::H2]*k_O2;
265:                 // C + 1/2*O2 ==> C0
266:                 _n_chemR[2] = ChemRectionRate[2]*k_Tpt*_nu[defs::C]*k_O2;
267:                 // CO + 1/2*O2 ==> C02
268:                 _n_chemR[3] = ChemRectionRate[3]*k_Tpt*_nu[defs::CO]*k_O2;
269:                 deltaH =  _n_chemR[0] * Shomate::getInst()->getEnthalpyOfFormation(defs::Fuel);
270:                 deltaH -= _n_chemR[1] * Shomate::getInst()->getEnthalpyOfFormation(defs::H2O);
271:                 deltaH -= _n_chemR[2] * Shomate::getInst()->getEnthalpyOfFormation(defs::CO);
272:                 deltaH -= _n_chemR[3] * (Shomate::getInst()->getEnthalpyOfFormation(defs::CO2) - Shomate::g
etInst()->getEnthalpyOfFormation(defs::CO));
273:                 deltaH *= _n_g;
274:                 _nu[defs::Fuel] -= _n_chemR[0];
275:                 _nu[defs::H2]   += _n_chemR[0]*(Fuel_n_C+1.0) - _n_chemR[1];
276:                 _nu[defs::H2O]  += _n_chemR[1];
277:                 _nu[defs::C]    += _n_chemR[0]*Fuel_n_C - _n_chemR[2];
278:                 _nu[defs::CO]   += _n_chemR[1] - _n_chemR[3];
279:                 _nu[defs::CO2]  += _n_chemR[3];
280:                 _nu[defs::O2]   -= 0.5*(_n_chemR[1] + _n_chemR[2] + _n_chemR[3]);
281:                 normalizeMols();
282:                 _v = _V / _n_g;
283:                 if(_nu[defs::O2] < EPSILON ||
284:                                 ( _nu[defs::H2] < EPSILON && _nu[defs::C] < EPSILON && _nu[defs::CO] < EPSI
LON && _nu[defs::Fuel] < EPSILON)){
285:                         _combustionStarted = false;
286:                 }
287:         }
288:         return deltaH;
289: }
290:
291: // helper methods
292:
293: double GasComponent::calcMolareWeight(){
294:         double MW=0.0;
295:         int i=0;
296:         for (i = 0; i < defs::Fuel+1; i++) {
297:                 MW += _nu[i] * MolWeights[i];
298:         }
299:         return MW;
300: }
301:
302: double GasComponent::calcMols(){
303:         return _p*_V/(_T*R);
304: }
305:
306: /**
307:  * recalcs the num of mols
308:  * sum(nu_i) == 1.0 && nu_i >= 0.0;
309:  */
310: void GasComponent::normalizeMols(){
311:         double k = 0.0;
312:         int i = 0;
313:         for (i = 0; i < defs::Fuel+1; i++) {
314:                 if(_nu[i] > EPSILON){
315:                         k += _nu[i];
316:                 }else{
317:                         _nu[i] = 0.0;
318:                 }
319:         }
320:         if( fabs(k-1.0) > EPSILON ){
321:                 _n_g *= k;
322:                 for (i = 0; i < defs::Fuel+1; i++) {
323:                         _nu[i] /= k;
324:                 }
325:         }
326:         _MW = calcMolareWeight();
327: }
```

328:
329:

```
    1: /*
    2:  * <one line to give the program's name and a brief idea of what it does.>
    3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
    4:  *
    5:  * This program is free software; you can redistribute it and/or modify
    6:  * it under the terms of the GNU General Public License as published by
    7:  * the Free Software Foundation; either version 2 of the License, or
    8:  * (at your option) any later version.
    9:  *
   10:  * This program is distributed in the hope that it will be useful,
   11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
   12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   13:  * GNU General Public License for more details.
   14:  *
   15:  * You should have received a copy of the GNU General Public License along
   16:  * with this program; if not, write to the Free Software Foundation, Inc.,
   17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
   18:  *
   19:  */
   20:
   21: #ifndef GASCOMPONENT_H
   22: #define GASCOMPONENT_H
   23:
   24: #include <stdio.h>
   25: #include "definitions.h"
   26:
   27: double getPressureFromRelHum(double T, double relHum);
   28:
   29: class GasComponent
   30: {
   31: public:
   32:         GasComponent();
   33:         GasComponent(double V, double T, double p);
   34:         GasComponent(double V, double T, double p, const double nu[defs::Fuel+1]);
   35:         GasComponent(double V, double T, double p, const double nu[defs::Fuel+1], bool isContainer);
   36:         /*
   37:          * remove mols from higher pressured component and mix it to the other
   38:          */
   39:         void calcGasExchange(double A_crosssection, GasComponent *pgc);
   40:         void calcStateChange(double cmpFactor, double H_cooling, double n_Fuel);//, const GasComponent &inl
et, GasComponent &exhaust);
   41: //setter methods
   42:         void setCombustionStarted(bool combustionStarted);
   43:         //void setMols(double n);
   44: //getter methods
   45:         double getSpecHeatCapacity() const;
   46:         double getEnthalpy() const;
   47:         double getMolareWeight() const;
   48:         double getMols() const;
   49:         const double* getNu() const;
   50:         double getP() const;
   51:         double getT() const;
   52:         double getspecV() const;
   53:         double getV() const;
   54:         bool isCombustionStarted() const;
   55:
   56: protected:
   57:         double _n_g, _nu[defs::Fuel+1];
   58:         double _T,_p,_v;
   59:         double _V,_H;
   60:         double _MW,_cp;
   61:         bool _combustionStarted;
   62:         bool _isContainer;
   63:
   64:         void transferFrom(double n, GasComponent &gc);
   65:         void normalizeMols();
   66:
   67:
   68: private:
   69:
   70:         double _n_chemR[4];
   71:
   72:         double isentropicStateChange(double cmpFactor);
   73:         //double isochoricStateChange(double deltaH);
   74:         double injection(double n_Fuel);
   75:         double chemReaction();
   76:
   77:         double calcMolareWeight();
   78:         double calcMols();
   79: };
   80:
   81:
   82: #endif // GASCOMPONENT_H
```

```
/* Copyright 2003-2004 The MathWorks, Inc. */

// ***************************************************************
// **** To build this mex function use: mex mexengine.cpp ****
// ***************************************************************

#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME mexengine

// Need to include simstruc.h for the definition of the SimStruct and
// its associated macro definitions.
#include "simstruc.h"
#include "combustionengine.h"


#define IS_PARAM_DOUBLE(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal) &&\
!mxIsEmpty(pVal) && !mxIsSparse(pVal) && !mxIsComplex(pVal) && mxIsDouble(pVal))


// Function: mdlInitializeSizes ===============================================
// Abstract:
//    The sizes information is used by Simulink to determine the S-function
//    block's characteristics (number of inputs, outputs, states, etc.).
static void mdlInitializeSizes(SimStruct *S){
    // No expected parameters
    ssSetNumSFcnParams(S, 0);

    // Parameter mismatch will be reported by Simulink
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return;
    }

    // Specify I/O
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 5);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 32);

    ssSetNumSampleTimes(S, 1);

    // Reserve place for C++ object
    ssSetNumPWork(S, 1);

    ssSetSimStateCompliance(S, USE_CUSTOM_SIM_STATE);

    //ssSetOptions(S, SS_OPTION_WORKS_WITH_CODE_REUSE | SS_OPTION_EXCEPTION_FREE_CODE);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);

}


// Function: mdlInitializeSampleTimes =========================================
// Abstract:
//   This function is used to specify the sample time(s) for your
//   S-function. You must register the same number of sample times as
//   specified in ssSetNumSampleTimes.
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}

// Function: mdlStart =======================================================
// Abstract:
//   This function is called once at start of model execution. If you
//   have states that should be initialized once, this is the place
//   to do it.
#define MDL_START
static void mdlStart(SimStruct *S)
{
    // Store new C++ object in the pointers vector
    CombustionEngine *engine  = CombustionEngine::getInst();
    ssGetPWork(S)[0] = engine;
}

// Function: mdlOutputs =======================================================
// Abstract:
//   In this function, you compute the outputs of your S-function
//   block.
static void mdlOutputs(SimStruct *S, int_T tid){
    // Retrieve C++ object from the pointers vector
```

```
    CombustionEngine *engine = static_cast<CombustionEngine *>(ssGetPWork(S)[0]);
    // Get data addresses of I/O
    InputRealPtrsType  u = ssGetInputPortRealSignalPtrs(S,0);
              real_T *y = ssGetOutputPortRealSignal(S, 0);

    // Call AddTo method and return peak value
        engine->setPhiSpark(*u[2]); // [degree] if(sparkAngle >= -40.0 && sparkAngle <= 20.0)
        engine->setPhiInjection(*u[3]);
        engine->setTempCoolingWater(*u[4]);
        y[0] = 0.0;
        y[2] = engine->getPhi(); // get phi from prev. time step
    for (int i = 0; i < 10; i++) {
                engine->run(*u[0], *u[1]); // [rad/s] , [-] 0..1
                y[0] += engine->getMShaft();
        }
    y[0]/=10.0;
        y[1] = engine->getPhi();
        if((y[2] > y[1])&&(y[2]-y[1] > 11)){ // phi reset!
                y[2] -= 4.0*M_PI;
        }
        y[2]=(y[1]-y[2])/(2*M_PI)*10000; //frequency of rotation
        /*
         * [0::2]  M, phi, freq
         */

        y[3] = 0.0; engine->getCyl_T(0);
        for (int i = 0; i < 6; i++) {
                if (i < Ncyl){
                        y[3] += engine->getCyl_T(i);
                        y[16+i] = engine->getp_Cyl(i);
                        y[24+i] = engine->getT_Cyl(i);
                }else{
                        y[16+i] = 0.0;
                }
        }
        y[3] /= Ncyl;
        y[4] = engine->getFuelConsumption();
        y[5] = engine->getIntake().getP();
        y[6] = engine->getIntake().getT();
        y[7] = engine->getExhaust().getP();
        y[8] = engine->getExhaust().getT();
        y[9] = engine->getExhaust().getNu()[defs::O2];
        y[10] = engine->getExhaust().getNu()[defs::H2O];
        y[11] = engine->getExhaust().getNu()[defs::CO2];
        y[12] = engine->getExhaust().getNu()[defs::CO];
        y[13] = engine->getExhaust().getNu()[defs::H2];
        y[14] = engine->getExhaust().getNu()[defs::C];
        y[15] = engine->getExhaust().getNu()[defs::Fuel];
        // p1..6
        y[22] = Environment::getInst()->getAmbientAir()->getMols();
        y[23] = Environment::getInst()->getExhaustGas()->getMols();
        // T1..6
        y[30] = engine->getH_Cooling()/Ts;
        y[31] = Environment::getInst()->getExhaustGas()->getP();
}

/* Define to indicate that this S-Function has the mdlG[S]etSimState methods */
#define MDL_SIM_STATE

/* Function: mdlGetSimState =================================================
 * Abstract:
 */
#ifdef ARTELAB
        //RT - version
        static double mdlGetSimState(SimStruct* S) {
           return 0.0;
        }
#else
        //nonRT - version
        static mxArray* mdlGetSimState(SimStruct* S) {
           // Retrieve C++ object from the pointers vector
           return mxCreateDoubleScalar(0.0);
        }
#endif


/* Function: mdlGetSimState =================================================
 * Abstract:
 *
 */
static void mdlSetSimState(SimStruct* S, const mxArray* ma)
{
    // Retrieve C++ object from the pointers vector
```

```
167:     CombustionEngine *engine = static_cast<CombustionEngine*>(ssGetPWork(S)[0]);
168:         engine->run(mxGetPr(ma)[0], mxGetPr(ma)[0]);
169: }
170:
171: // Function: mdlTerminate =====================================================
172: // Abstract:
173: //   In this function, you should perform any actions that are necessary
174: //   at the termination of a simulation.  For example, if memory was
175: //   allocated in mdlStart, this is the place to free it.
176: static void mdlTerminate(SimStruct *S)
177: {
178:     // Retrieve and destroy C++ object
179:     CombustionEngine *engine = static_cast<CombustionEngine*>(ssGetPWork(S)[0]);
180:     delete engine;
181: }
182:
183:
184: // Required S-function trailer
185: #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
186: #include "simulink.c"      /* MEX-file interface mechanism */
187: #else
188: #include "cg_sfun.h"       /* Code generation registration function */
189: #endif
```

```
 1: /*
 2:  * Oil.cpp
 3:  *
 4:  *  Created on: Apr 15, 2018
 5:  *      Author: alex
 6:  */
 7:
 8: #include "oil.h"
 9:
10: Oil::Oil() {
11:         _eta_base = Oil_p[3]/Oil_p[1];
12:         _eta_exp = Oil_p[1];
13:         _k = Oil_p[2]/(Oil_p[0]-Oil_p[2]);
14: }
15:
16: Oil::~Oil() {
17: }
18:
19: /**
20:  * implementation of the Arrhenius eq.
21:  * eta = eta_0 exp(k/T);
22:  * with the 2-point definition the bases changes to eta1/eta2...
23:  *
24:  * eta = eta1 * (eta2/eta1) ^ ( (1/T - 1/T1) / (1/T2 - 1/T1) )
25:  *     = eta1 * _eta_base^(-_k) * _eta_base^(_k*T1/T)
26:  */
27: double Oil::getEta(double T) const {
28:         double eta = Oil_p[5];
29:         if (T < Oil_p[4]){
30:                 eta = Oil_p[1]*pow(_eta_base, -_k)*pow(_eta_base, _k*Oil_p[0]/T);
31:         }else{
32:                 if (T < 250){
33:                         eta = 1;
34:                 }
35:         }
36:         return eta;
37: }
```

```
 1: /*
 2:  * Oil.h
 3:  *
 4:  *  Created on: Apr 15, 2018
 5:  *      Author: alex
 6:  */
 7:
 8: #ifndef OIL_H_
 9: #define OIL_H_
10:
11: #include "definitions.h"
12:
13: class Oil {
14: public:
15:         Oil();
16:         ~Oil();
17:         double getEta(double T) const;
18:
19: private:
20:         double _eta_base, _eta_exp, _k;
21: };
22:
23: #endif /* OIL_H_ */
```

```
 1: /*
 2:  * <one line to give the program's name and a brief idea of what it does.>
 3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
 4:  *
 5:  * This program is free software; you can redistribute it and/or modify
 6:  * it under the terms of the GNU General Public License as published by
 7:  * the Free Software Foundation; either version 2 of the License, or
 8:  * (at your option) any later version.
 9:  *
10:  * This program is distributed in the hope that it will be useful,
11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13:  * GNU General Public License for more details.
14:  *
15:  * You should have received a copy of the GNU General Public License along
16:  * with this program; if not, write to the Free Software Foundation, Inc.,
17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
18:  *
19:  */
20: #include <stdio.h>
21:
22: #include "shomate.h"
23: #include "definitions.h"
24:
25: TRange::TRange(){
26:   T_min_ = 0.0;
27:   T_max_ = 0.0;
28: }
29:
30: TRange::TRange(double Tmin, double Tmax){
31:   TRange();
32:   set(Tmin, Tmax);
33: }
34:
35: void TRange::set(double Tmin, double Tmax){
36:   T_min_ = Tmin;
37:   T_max_ = Tmax;
38: }
39:
40:
41:
42: ShParDef::ShParDef(){
43: }
44:
45: ShParDef::ShParDef(double A, double B, double C, double D, double E, double F, double G, double Hf){
46:   set(A, B, C, D, E, F, G, Hf);
47: }
48:
49: void ShParDef::set(double A, double B, double C, double D, double E, double F, double G, double Hf){
50:   data_[shomate::A] = A;
51:   data_[shomate::B] = B;
52:   data_[shomate::C] = C;
53:   data_[shomate::D] = D;
54:   data_[shomate::E] = E;
55:   data_[shomate::F] = F;
56:   data_[shomate::G] = G;
57:   data_[shomate::Hf]= Hf;
58: }
59:
60: void ShParDef::add(const double factor, const ShParDef * pars) {
61:         for (int i = 0; i <= shomate::Hf; i++) {
62:                 if(pars != 0){ // && (pars->data_[i]) > 0.0){
63:                         data_[i] += (pars->data_[i])* factor;
64:                 }
65:         }
66: }
67:
68: void ShParDef::reset(){
69:   set(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
70: }
71:
72: ShDataEntry::ShDataEntry(double Tmin, double Tmax, double A, double B, double C, double D, double E, double
F, double G, double Hf){
73:           T_ = TRange(Tmin, Tmax);
74:           pars_= ShParDef(A, B, C, D, E, F, G, Hf);
75: }
76:
77: ShData::ShData(){
78:         defs_ = 0;
79: }
80:
81: ShData::ShData(int dataSets){
82:   if(dataSets > 0 && dataSets <= 5){
```

```
   83:           defs_ = dataSets;
   84:   }
   85: }
   86:
   87: ShData::ShData(ShDataEntry shEntry){
   88:         defs_ = 1;
   89:         T_[0] = shEntry.T_;
   90:         pars_[0] = shEntry.pars_;
   91: }
   92:
   93: ShData::ShData(int dataSets, const ShDataEntry shEntry[]){
   94:         defs_ = 0;
   95:         if(dataSets > 0 && dataSets <= 5){
   96:                   defs_ = dataSets;
   97:                   int i = 0;
   98:                   for (i = 0; i < defs_; i++) {
   99:                           T_[i] = shEntry[i].T_;
  100:                           pars_[i] = shEntry[i].pars_;
  101:                   }
  102:           }
  103: }
  104:
  105: void ShData::set(int dataSet, double Tmin, double Tmax, double A, double B, double C, double D, double E, d
ouble F, double G, double Hf){
  106:   if(dataSet >=0 && dataSet < defs_){
  107:           T_[dataSet].set(Tmin, Tmax);
  108:           pars_[dataSet].set(A, B, C, D, E, F, G, Hf);
  109:   }
  110: }
  111:
  112: const ShParDef * ShData::getParams(double T) const  {
  113:   const ShParDef *result = NULL;
  114:   if(defs_ > 0 ){
  115:           if(defs_ == 1 || T <= T_[0].T_max_){
  116:                 result = &pars_[0];
  117:           } else if(defs_ == 2 ){
  118:                 result = &pars_[1];
  119:           } else{
  120:                 for(int i = 1; i<defs_;i++){
  121:                   if(T <= T_[i].T_max_){
  122:                         result = &pars_[i];
  123:                         break;
  124:                   }
  125:                 }
  126:           }
  127:   }
  128:   return result;
  129: }
  130:
  131: ShData::~ShData(){
  132: }
  133:
  134:
  135: Shomate* Shomate::_pInst = NULL;
  136:
  137: Shomate::Shomate(){
  138:         _pActShParams = new ShParDef();
  139:         _pActShParams->reset();
  140: }
  141:
  142: double Shomate::getHeatCapacity(double T, const double *pn_def){
  143:   double cp = 0.0;
  144:   double t = T/1000.0;
  145:   int i = 0;
  146:   _pActShParams->reset();
  147:   for(i=0;i<=defs::Fuel;i++){
  148:           _pActShParams->add( *(pn_def+i), ShDataDB[i].getParams(T));
  149:   }
  150:   cp= _pActShParams->data_[shomate::A] + _pActShParams->data_[shomate::B]*t + _pActShParams->data_[shomate:
:C]*pow(t,2.0)
  151:                   + _pActShParams->data_[shomate::D]*pow(t,3.0) + _pActShParams->data_[shomate::E]/pow(t,2.
0);
  152:   return cp;
  153: }
  154:
  155: double Shomate::getFuelHeatCapacity(double T){
  156:         return ShDataDB[defs::Fuel].pars_[0].data_[shomate::A] + ShDataDB[defs::Fuel].pars_[0].data_[shomat
e::B] * T/1000.0;
  157: }
  158:
  159: double Shomate::getEnthalpyOfFormation(enum defs::eChemList substance){
  160:         return ShDataDB[substance].pars_[0].data_[shomate::Hf];
  161: }
```

```
162:
163: Shomate* Shomate::getInst() {
164:         if (!_pInst){
165:                         _pInst = new Shomate();
166:                 }
167:                 return _pInst;
168: }
169:
170: Shomate::~Shomate() {
171: }
172:
173:
174:
```

```
    1: /*
    2:  * <one line to give the program's name and a brief idea of what it does.>
    3:  * Copyright (C) 2014  Alex Luschan <alexander.luschan@gmail.com>
    4:  *
    5:  * This program is free software; you can redistribute it and/or modify
    6:  * it under the terms of the GNU General Public License as published by
    7:  * the Free Software Foundation; either version 2 of the License, or
    8:  * (at your option) any later version.
    9:  *
   10:  * This program is distributed in the hope that it will be useful,
   11:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
   12:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   13:  * GNU General Public License for more details.
   14:  *
   15:  * You should have received a copy of the GNU General Public License along
   16:  * with this program; if not, write to the Free Software Foundation, Inc.,
   17:  * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
   18:  *
   19:  */
   20:
   21: #ifndef SHOMATE_H
   22: #define SHOMATE_H
   23: #include <iostream>
   24: #include <stdio.h>
   25: #include <stdlib.h>
   26:
   27:
   28: // chemical substances
   29: namespace defs{
   30:   enum eChemList {N2=0, O2=1, H2O=2, CO2=3, CO=4, H2=5, C=6, Fuel=7};
   31: }
   32:
   33: namespace shomate{
   34:         enum shName{A ,B ,C ,D ,E ,F ,G ,Hf};
   35: }
   36:
   37: class TRange
   38: {
   39: public:
   40:   double T_min_, T_max_;
   41:   TRange();
   42:   TRange(double Tmin, double Tmax);
   43:   void set(double Tmin, double Tmax);
   44: };
   45:
   46: /* unit of Hf: [J/mol]*/
   47: class ShParDef{
   48: public:
   49:   ShParDef();
   50:   ShParDef(double A,double B,double C,double D,double E,double F,double G,double Hf);
   51:   void set(double A,double B,double C,double D,double E,double F,double G,double Hf);
   52:   //void add(double *factor, ShParDef * pars);
   53:   void add(const double factor, const ShParDef *pars);
   54:   void reset();
   55:   double data_[shomate::Hf+1];
   56: };
   57:
   58: class ShDataEntry{
   59: public:
   60:         TRange T_;
   61:         ShParDef pars_;
   62:         ShDataEntry(double Tmin, double Tmax, double A,double B,double C,double D,double E,double F,double
G,double Hf);
   63: };
   64:
   65: class ShData
   66: {
   67: public:
   68:         ShData();
   69:         ShData(int dataSets);
   70:         ShData(ShDataEntry shEntry);
   71:         ShData(int dataSets, const ShDataEntry shEntry[]);
   72:         ~ShData();
   73:         void set(int dataSet, double Tmin, double Tmax, double A,double B,double C,double D,double E,double
 F,double G,double Hf);
   74:         TRange T_[5];
   75:         ShParDef pars_[5];
   76:         const ShParDef* getParams(double T)  const ;
   77: private:
   78:         int defs_;
   79: };
   80:
   81: class Shomate{
```

```
82: public:
83:         static Shomate* getInst();
84:
85:         double getHeatCapacity(double T, const double *pn_def);
86:         double getFuelHeatCapacity(double T);
87:         double getEnthalpyOfFormation(enum defs::eChemList substance);
88:
89: private:
90:         Shomate();
91:         virtual ~Shomate();
92:         static Shomate* _pInst;
93:
94:         ShParDef* _pActShParams;
95: };
96:
97: #endif // SHOMATE_H
```

```
 1: /*
 2:  * valve.cpp
 3:  *
 4:  *  Created on: 31.01.2016
 5:  *      Author: alex
 6:  */
 7:
 8: #include "valve.h"
 9:
10: /**
11:  *
12:  */
13: Valve::Valve(double A_Valves, int num_Valves, double phi_open, double phi_close){
14:         _phi_m = (phi_open+phi_close)/2;
15:         _dphi = (phi_close-phi_open)/2;
16:         _num = num_Valves;
17:         _r = sqrt(A_Valves/((double)num_Valves * M_PI));
18:         _stroke = _r*2;
19: }
20:
21: double Valve::getPhiM() {
22:         return _phi_m;
23: }
24:
25: double Valve::getDPhi() {
26:         return _dphi;
27: }
28:
29: void Valve::setPhiM(double phi_M) {
30:         _phi_m = phi_M;
31: }
32:
33: void Valve::setDPhi(double d_Phi) {
34:         _dphi = d_Phi;
35: }
36:
37: double Valve::getCrosssection(double phi) {
38:         double result = getActStroke(phi);
39:         if(result > 0.0){
40:                 if(result > _r/2.0){            // fully opened
41:                         result = _r*_r*M_PI*_num;
42:                 }else{                                          // partially opened
43:                         result *= 2.0*_r*M_PI*_num;
44:                 }
45:         }
46:         return result;
47: }
48:
49: double Valve::getActStroke(double phi){
50:         double result = 0.0;
51:         if (!( _phi_m + _dphi > 4*M_PI && phi < M_PI)) { // closing might happen in between 0.. (<M_PI)
52:                 result = (1 - fabs(_phi_m - phi)/_dphi)*_stroke;
53:         }else{
54:                 result = (1 -fabs(_phi_m - phi - 4*M_PI)/_dphi) * _stroke;
55:         }
56:         return result;
57: }
58:
59: Valve::Valve(){
60:         _phi_m = 0.0;
61:         _dphi = 0.0;
62:         _num = 1;
63:         _r = 0.0;
64:         _stroke = 0.0;
65: }
66:
67: Valve::~Valve(){
68:
69: }
```

```
    1: /*
    2:  * valve.h
    3:  *
    4:  *  Created on: 31.01.2016
    5:  *      Author: alex
    6:  */
    7:
    8: #ifndef VALVE_H_
    9: #define VALVE_H_
   10:
   11: #include "definitions.h"
   12:
   13: class Valve{
   14:
   15: public:
   16:         Valve();
   17:         Valve(double A_Valves, int num_Valves, double dphi_open, double dphi_close);
   18:         ~Valve();
   19:         double getPhiM();
   20:         double getDPhi();
   21:         void setPhiM(double phi_M);
   22:         void setDPhi(double d_Phi);
   23:         double getCrosssection(double phi);
   24:
   25: private:
   26:         double _phi_m;          // angle of middle position of camshaft (in means of crankshaft working cyc
le)
   27:         double _dphi;           // angle open<->middle and middle<->close position
   28:         double _r;                      // radius of a single valve
   29:         double _stroke;         // stroke of valve(s)
   30:         int _num;                       // number of valves
   31:
   32:         double getActStroke(double phi);
   33: };
   34:
   35:
   36:
   37: #endif /* VALVE_H_ */
```