```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include <cmath>
#include <utility>
#include <limits>
#include <list>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include "drw.h"

// Type Definitions
typedef float real; //float can be changed to double or long double to increase the precision, float is used to make the program faster
class Cmns; //Class of Meniscus
typedef std::vector<std::vector<real>> Treal; //Table of real numbers
typedef std::vector<std::vector<Cmns>> Tmns;

//GENERAL CONSTANTS
const real PI = std::acos(-1);
const real HUGE = std::numeric_limits<real>::max();

//Physical Characterstics of the experiment
const real PRESSURE_BOTTOM = 100;
const real PRESSURE_TOP = 2;
const real SIGMA = 0; //7.56e-2; // FOR water at 20C in SI units, produces 75Pa pressure difference for tube of radius 1mm
const real TUBE_LENGTH = 1;//0.1;
const real MU1 = 1;//1e-3; // viscosity of the invading liquid: water
const real MU2 = 1;//1e-5; // viscosity of defending liquid: air

//Parameters of simulation
const real MAX_WETTING_PROPORTION = 0.98;
const real THRESHOLD_FILL = 0.001; //if any meniscus is smaller than this proportion, then it is destroyed
const real TIME_DIV = 4; // if the nearest meniscus by time is further, then L / TIME_DIV is prefered
```

```
37  const int IMAGE_SIZE = 1000;
38  const real FINE_RADIUS_RANDOMNESS = 1e4;
39
40  //Input Output of File names
41  const std::string FILE_NAME_RADIUS = "radius.txt";
42  const std::string FILE_NAME_MNS = "fill.txt";
43  const std::string FOLDER_SAVE_PIC = "pic/";
44
45
46  class Cmns
47  {
48      /* FUNCTION DESCRIPTION - scontb
49       *
50       * when we write the equation, about the volume
51       * q = dV/dt = PI / 8 / MU * r ^ 4 / L * [(P[i] - P[j]) + 2 * SIGMA
         / r)
52       * If there is high pressure and mescible fluid in the [i],
53       * a positive sign before 2 * SIGMA / r means
54       * the pressure difference is made higher by the interface meniscus
55       * 1) blue: 0
56       * 2) grey: 0
57       * 3) blue|grey: +1
58       * 4) grey|blue: -1
59       * 5) blue|grey|blue: 0
60       * 6) grey|blue|grey: 0
61       *
62       * sign going down means direction is 2 or 3,
63       * for which type 0 means the meniscus is oriented away from [i],
64       * giving a negative contribution of cappilary pressure
65       */
66
67      static real _scontb_sig(bool condition) //scontb's sign function
68      {
69          return condition ? -1 : 1;
70      }
71
72      /* generate positions long version
73       *
74       * the generated std::vector<real> can be of 3 types:
75       * 1) [0, 1]
```

```
76          * 2) [0, pos1, 1]
77          * 3) [0, pos1, pos2, 1]
78          *
79          * it is distinguished from the short version which is of the form
80          * 1) [], n = 0
81          * 2) [a], n = 1
82          * 3) [a, b], n = 2
83          */
84
85         std::vector<real> gen_pos_long_after_dspl(real vel, real l) const
86         {
87             auto pos_long_after_dslp = gen_pos_long();
88             if(vel < 0)
89             {
90                 pos_long_after_dslp.front() = l;
91             }
92             else
93             {
94                 pos_long_after_dslp.back() -= l;
95             }
96
97             return pos_long_after_dslp;
98         }
99
100        //generate compartments of the configuration which exists
101        typedef std::list<std::pair<bool, real>> Ccmprt;
102        Ccmprt gen_cmprt_existing(real vel, real l) const
103        {
104            const auto pos_long_after_dspl = gen_pos_long_after_dspl(vel, l);
105            Ccmprt cmprt_existing;
106            for(int i = 1; i < pos_long_after_dspl.size(); ++ i)
107            {
108                cmprt_existing.push_back({(i + type - 1) % 2,
                    pos_long_after_dspl[i] - pos_long_after_dspl[i - 1]});
109            }
110
111            return cmprt_existing;
112        }
113
114        static Ccmprt merge_existing_and_new_cmprts(Ccmprt& cmprt_existing,
```

```
    const Ccmprt& cmprt_new, real vel)
115 {
116     if(vel > 0)
117     {
118         cmprt_existing.insert(cmprt_existing.begin(),
            cmprt_new.crbegin(), cmprt_new.crend());
119     }
120     else
121     {
122         cmprt_existing.insert(cmprt_existing.end(),
            cmprt_new.begin(), cmprt_new.end());
123     }
124     return cmprt_existing;
125 }
126
127 struct CmnsAfterDspl
128 {
129     bool type;
130     std::vector<real> v;
131 };
132
133 static std::vector<real> gen_pos_from_segmented(std::vector<real>
    pos_segmented)
134 {
135     pos_segmented.pop_back();
136     for(int i = 1; i < pos_segmented.size(); ++ i)
137     {
138         pos_segmented[i] += pos_segmented[i - 1];
139     }
140
141     return pos_segmented;
142 }
143
144 CmnsAfterDspl gen_pos_new_and_type(const Ccmprt& cmprt_new, const
    real threshold_fill) const
145 {
146     std::vector<std::pair<int, real>> cmprt_new_temp_vector;
147     for(const auto& x: cmprt_new) // Step-1 Filter out anything
        smaller than threshold_fill
148     {
```

```
149             if(x.second >= threshold_fill)
150             {
151                 cmprt_new_temp_vector.push_back({x.first, x.second});
152             }
153         }
154 
155         for(int i = 1; i < cmprt_new_temp_vector.size(); ++ i) // Step-2 Merge any two compartments of the smae fluid type
156         {
157             if(cmprt_new_temp_vector[i - 1].first == cmprt_new_temp_vector[i].first)
158             {
159                 cmprt_new_temp_vector[i - 1].first = -1;
160                 cmprt_new_temp_vector[i].second += cmprt_new_temp_vector[i - 1].second;
161             }
162         }
163 
164         int type_begin_temp = -1;
165         std::vector<real> pos_segmented;
166         for(const auto& x: cmprt_new_temp_vector)
167         {
168             if(x.first != -1)
169             {
170                 if(type_begin_temp == -1)
171                 {
172                     type_begin_temp = x.first;
173                 }
174                 pos_segmented.push_back(x.second);
175             }
176         }
177 
178         const bool type_begin = type_begin_temp;
179         const auto pos_new = gen_pos_from_segmented(pos_segmented);
180 
181         if(pos_new.size() < 3) // Depending on the number of meniscus, recombinate or preenet as it is
182         {
183             return {type_begin, pos_new};
184         }
```

```
185         if(pos_new.size() == 3)
186         {
187             const real l1 = 0;
188             const real l2 = pos_new[0];
189             const real l3 = pos_new[1];
190             const real l4 = pos_new[2];
191 
192             const real d1 = l2 - l1;
193             const real d2 = l4 - l3;
194             const real d = d1 + d2;
195             const real c1 = (l1 + l2) / 2;
196             const real c2 = (l3 + l4) / 2;
197 
198             const real L1 = (c1 * d1 + c2 * d2) / d - d / 2;
199             const real L2 = L1 + d;
200 
201             return {!type_begin, {L1, L2}};
202         }
203         if(pos_new.size() == 4)
204         {
205             const real l1 = pos_new[0];
206             const real l2 = pos_new[1];
207             const real l3 = pos_new[2];
208             const real l4 = pos_new[3];
209 
210             const real d1 = l2 - l1;
211             const real d2 = l4 - l3;
212             const real d = d1 + d2;
213             const real c1 = (l1 + l2) / 2;
214             const real c2 = (l3 + l4) / 2;
215 
216             const real L1 = (c1 * d1 + c2 * d2) / d - d / 2;
217             const real L2 = L1 + d;
218 
219             return {type_begin, {L1, L2}};
220         }
221 
222         std::cout << "ER3-oversized decompartalization" << std::endl;
223         return {type_begin, {-1, -1}};
224     }
```

```
225
226 public:
227     int n; //number of meniscus present
228     bool type; // 0 - corresponds to blue fluid - which is invading
229     std::vector<real> pos; // positions of meniscus
230
231     Cmns(): n(0), type(1), pos(2) {} //by default everything is the
        defending fluid
232     Cmns(int n, bool type, real p1, real p2): n(n), type(type), pos{p1,
        p2} {}
233
234     real mu(const real mu1, const real mu2) const
235     {
236         std::vector<real> muv{mu1, mu2};
237         const auto pos_long = gen_pos_long();
238
239         real sum = 0;
240         for(int i = 1; i < pos_long.size(); ++ i)
241         {
242             sum += muv[(i - 1 + type) % muv.size()] * (pos_long[i] -
                pos_long[i - 1]);
243         }
244
245         return sum;
246     }
247
248     real time(const real velocity, const real length, const real
        time_div) const
249     {
250         if(n == 0)
251         {
252             return length / velocity / time_div;
253         }
254
255         real dspl = (velocity >= 0 ? (1 - pos[n - 1]): pos.front());
256         dspl = std::min(1.0f / time_div, dspl);
257         return length * dspl / velocity;
258     }
259
260     void update(const real vel, const real r, const std::vector<real>&
```

```
    add, const real threshold_fill)
261 {
262     const real area = PI * std::pow(r, 2);
263     const real l1 = add.front() / area;
264     const real l2 = add.back() / area;
265     const real l = l1 + l2;
266 
267     auto cmprt_existing = gen_cmprt_existing(vel, l);
268     Ccmprt cmprt_new{{0, l1}, {1, l2}};
269     auto cmprt = merge_existing_and_new_cmprts(cmprt_existing,
        cmprt_new, vel);
270     auto pos_new_and_type = gen_pos_new_and_type(cmprt,
        threshold_fill);
271     n = pos_new_and_type.v.size();
272     type = pos_new_and_type.type;
273     pos = pos_new_and_type.v;
274     pos.resize(2);
275 }
276 
277 real scontb(int direction) const
278 {
279     return _scontb_sig(direction > 1) * _scontb_sig(type) * (n % 2);
280 }
281 
282 /*      vel | [true]    | [false]   |
283  * drec     | above(<2) | below(>=2)|
284  * [true]+1 | out-0     | in-1      |
285  * [flase]-1| in-1      | out-0     |
286  */
287 
288 bool is_flow_into_node(const int direction, const real velocity) const
289 {
290     return (direction < 2) ^ (velocity >= 0);
291 }
292 
293 bool type_fluid_into_node(int direction) const
294 {
295     if(direction < 2) // if fluid is coming from the above
296     {
297         return type; // whatever is at the lowest part is what gets
```

```
            into the node
        }
        /*
         * What is on the top part?
         *
         * n    | type=0 | type=1 |
         * 0    | 0      | 1      |
         * 1    | 1      | 0      |
         * 2    | 0      | 1      |
         */

        return type ^ (n % 2);
    }

    real sum_type_first() const
    {
        const auto pos_long = gen_pos_long();
        real sum = 0;
        for(int i = 1 + type; i < pos_long.size(); i += 2)
        {
            sum += pos_long[i] - pos_long[i - 1];
        }

        return sum;
    }

    std::vector<real> gen_pos_long() const
    {
        std::vector<real> v(n + 2);
        for(int i = 0; i < n; ++ i)
        {
            v[i + 1] = pos[i];
        }
        v.back() = 1;

        return v;
    }
};

std::ifstream& operator>> (std::ifstream& fin, Cmns& val)
```

```
337 {
338     fin >> val.n >> val.type >> val.pos.front() >> val.pos.back();
339     return fin;
340 }
341
342 std::ofstream& operator<< (std::ofstream& fout, const Cmns& val)
343 {
344     fout << '\n' << val.n << ' ' << val.type << ' ' << val.pos.front()
        << ' ' << val.pos.back();
345     return fout;
346 }
347
348 struct Coordinate
349 {
350     real x;
351     real y;
352 };
353
354 template <class T>
355 class FTable
356 {
357 public:
358     int nrows;
359     int ncols;
360     std::vector<std::vector<T>> v;
361
362     FTable() = default;
363     FTable(int nrows, int ncols, const T& val = T()): nrows(nrows),
        ncols(ncols), v(nrows, std::vector<T>(ncols, val)) {}
364
365     bool read(const std::string& file_name)
366     {
367         std::ifstream fin(file_name);
368         if(!(fin >> nrows >> ncols))
369         {
370             std::cout << "-ER2-" << file_name << " is corrupted!" << '\n';
371             return false;
372         }
373
374         std::vector<T> w;
```

```
375             T val;
376 
377             while(fin >> val)
378             {
379                 w.push_back(val);
380             }
381 
382             if(nrows * ncols != w.size())
383             {
384                 std::cout << "-ER2-" << file_name << " has incorrect
                    diamensions." << '\n';
385                 return false;
386             }
387 
388             v.resize(nrows, std::vector<T>(ncols));
389             for(int i = 0; i < w.size(); ++ i)
390             {
391                 v[i / ncols][i % ncols] = w[i];
392             }
393 
394             return true;
395         }
396 
397         Coordinate _coordinate (int row, int col) const
398         {
399             return {0.5f + col, -0.5f + nrows - row};
400         }
401 
402         void write(const std::string& file_name) const
403         {
404             std::ofstream fout(file_name);
405             fout << nrows << ' ' << ncols << "\n\n";
406             for(const auto& row: v)
407             {
408                 for(const auto& val: row)
409                 {
410                     fout << val << ' ';
411                 }
412 
413                 fout << '\n';
```

```
414             }
415         }
416
417         bool between(real x, real a, real b) const
418         {
419             return x >= a && x <= b;
420         }
421
422         bool inside(real x1, real x2, real y1, real y2, const Coordinate&
    coordinate) const
423         {
424             return between(coordinate.x, x1, x2) && between(coordinate.y,
    y1, y2);
425         }
426
427         void update(real x1, real x2, real y1, real y2, const T& val)
428         {
429             real xmin = std::min(x1, x2);
430             real xmax = std::max(x1, x2);
431
432             real ymin = std::min(y1, y2);
433             real ymax = std::max(y1, y2);
434             for(int i = 0; i < nrows; ++ i)
435             {
436                 for(int j = 0; j < ncols; ++ j)
437                 {
438                     if(inside(xmin, xmax, ymin, ymax, _coordinate(i, j)))
439                     {
440                         v[i][j] = val;
441                     }
442                 }
443             }
444         }
445
446         void print() const
447         {
448             for(const auto& row: v)
449             {
450                 for(const auto& val: row)
451                 {
```

```
452                 std::cout << val << ' ';
453             }
454             std::cout << '\n';
455         }
456     }
457 };
458 
459 typedef FTable<real> FTradius;
460 typedef FTable<Cmns> FTmns;
461 
462 bool FCheckValidity()
463 {
464     FTradius radius;
465     FTmns mns;
466 
467     bool validity = true;
468     if(radius.read(FILE_NAME_RADIUS))
469     {
470         std::cout << "-FDK-" << FILE_NAME_RADIUS << " is valid" << '\n';
471     }
472     else
473     {
474         validity = false;
475     }
476 
477     if(mns.read(FILE_NAME_MNS))
478     {
479         std::cout << "-FDK-" << FILE_NAME_MNS << " is valid, " << '\n';
480     }
481     else
482     {
483         validity = false;
484     }
485 
486     if(validity)
487     {
488         if((radius.nrows == mns.nrows) && (radius.ncols == mns.ncols))
489         {
490             std::cout << "-FDK-" << "diamensions of " <<                  ↵
                FILE_NAME_RADIUS << " and " << FILE_NAME_MNS << " match" <<  ↵
```

```
                '\n';
491         }
492         else
493         {
494             std::cout << "-ERR-" << "diamensions of " <<
                FILE_NAME_RADIUS << " and " << FILE_NAME_MNS << " do not
                match!" << '\n';
495             validity = false;
496         }
497     }
498
499     return validity;
500 }
501
502 void FPrintValidityStatus()
503 {
504     if(FCheckValidity())
505     {
506         std::cout << "-FDK-" << FILE_NAME_RADIUS << ", " <<
            FILE_NAME_MNS << " is okay" << '\n';
507     }
508     else
509     {
510         std::cout << "-ERR-" << std::string(30, '!') << '\n';
511     }
512 }
513
514 Treal FReadFileRadius()
515 {
516     FTradius radius;
517     radius.read(FILE_NAME_RADIUS);
518
519     for(auto& row: radius.v)
520     {
521         for(auto& cell: row)
522         {
523             cell += (rand() % 100) / FINE_RADIUS_RANDOMNESS;
524         }
525     }
526     return radius.v;
```

```
527 }
528
529 Tmns FReadFileFill()
530 {
531     FTmns mns;
532     mns.read(FILE_NAME_MNS);
533     return mns.v;
534 }
535
536 int FLinearLocNode(int i, int j, int m)
537 {
538     return (i * (m + 1) + (i % 2)) / 2 + j;
539 }
540
541 std::pair<int, int> FConnectionEnds(int r, int c, int m)
542 {
543     return {FLinearLocNode(r, c / 2 + (c % 2) * ((r + 1) % 2), m),
        FLinearLocNode(r + 1, c / 2 + (c % 2) * (r % 2), m)};
544 }
545
546 int FTotalNodes(int n, int m)
547 {
548     return ((n + 1) * (m + 1) + 1) / 2;
549 }
550
551 struct Connections
552 {
553     bool a = true;
554     int r;
555     int c;
556     int p;
557 };
558
559 std::vector<Connections> FGenConnectionsEqu(int r, int c, int n, int m)
560 {
561     const auto p = FLinearLocNode(r, c, m);
562     std::vector<Connections> v
563     {
564         {true, r - 1, 2 * c - 1 + r % 2, p - m / 2 - 1},
565         {true, r - 1, 2 * c - 0 + r % 2, p - m / 2 - 0},
```

```
566             {true, r - 0, 2 * c - 0 + r % 2, p + m / 2 + 1},
567             {true, r - 0, 2 * c - 1 + r % 2, p + m / 2 + 0}
568         };
569 
570         if(r % 2)
571         {
572             return v;
573         }
574 
575         if(r == 0)
576         {
577             v[0].a = false;
578             v[1].a = false;
579         }
580         if(c == 0)
581         {
582             v[0].a = false;
583             v[3].a = false;
584         }
585         if(2 * c == m)
586         {
587             v[1].a = false;
588             v[2].a = false;
589         }
590         if(r == n)
591         {
592             v[2].a = false;
593             v[3].a = false;
594         }
595 
596         return v;
597     }
598 
599     std::vector<real> FGaussElimination(Treal M)
600     {
601         //std::cout << "okay-gauss gaussian eleimination" << std::endl;
602         const int n = M.front().size() - 1;
603         for(int i = 0; i < n; ++ i)
604         {
605             real divider = M[i][i];
```

```
606             for(int j = 0; j <= n; ++ j)
607             {
608                 M[i][j] /= divider;
609             }
610 
611             for(int j = 0; j < n; ++ j)
612             {
613                 if(i == j)
614                 {
615                     continue;
616                 }
617 
618                 real coeff = M[j][i];
619 
620                 for(int k = 0; k <= n; ++ k)
621                 {
622                     M[j][k] -= M[i][k] * coeff;
623                 }
624             }
625         }
626         std::vector<real> v;
627 
628         for(auto& row: M)
629         {
630             v.push_back(row.back());
631         }
632 
633         return v;
634     }
635 
636     void TFPrintMatrix(const std::string& s, const Treal& matrix)
637     {
638         std::cout << "\n\n-----------------------------------\n";
639         std::cout << s << '\n';
640 
641         std::cout << std::setw(7) << -1 << " | ";
642         for(int j = 0; j < matrix.front().size(); ++ j)
643         {
644             std::cout << std::setw(7) << (float)j << ' ';
645         }
```

```
646         std::cout << '\n';
647         for(int i = 0; i < matrix.size(); ++ i)
648         {
649             std::cout << std::setw(7) << (float)i << " | ";
650             for(int j = 0; j < matrix.front().size(); ++ j)
651             {
652                 std::cout << std::setw(7) << matrix[i][j] << ' ';
653             }
654             std::cout << '\n';
655         }
656     }
657
658     void TFPrintMatrix(const std::string& s, const std::vector<float>& v,
    const int n, const int m)
659     {
660         std::cout << "\n\n-----------------------------------\n";
661         std::cout << s << '\n';
662
663         int count = 0;
664         for(int i = 0; i <= n; ++ i)
665         {
666             int mt = m / 2 - i % 2;
667             for(int j = 0; j <= mt; ++ j)
668             {
669                 std::cout << std::setw(7) << v[count++] << ' ';
670             }
671             std::cout << '\n';
672         }
673     }
674
675     Treal FGenEquForGauss(const Treal& radius, const Tmns& mns)
676     {
677         //std::cout << "okay-FGenEquForGauss" << std::endl;
678         const int n = radius.size();
679         const int m = radius.front().size();
680         const int total_nodes = FTotalNodes(n, m);
681         Treal equation(total_nodes, std::vector<real>(total_nodes + 1));
682
683         //std::cout << "okay-FGenEquForGauss" << std::endl;
684         //std::cout << "total_nodes=" << total_nodes << std::endl;
```

```
685
686         for(int i = 0; i <= n; ++ i)
687         {
688             int mt = m / 2 - (i % 2);
689             //std::cout << "n= " << n << ", m=" << m << ", mt=" << mt <<
                std::endl;
690             for(int j = 0; j <= mt; ++ j)
691             {
692
693                 //std::cout << "i=" << i << ", j=" << j << std::endl;
694
695                 const int l = FLinearLocNode(i, j, m);
696                 auto& e = equation[l];
697                 if(i == 0)
698                 {
699                     e[l] = 1;
700                     e.back() = PRESSURE_TOP;
701                     continue;
702                 }
703                 if(i == n)
704                 {
705                     e[l] = 1;
706                     e.back() = PRESSURE_BOTTOM;
707                     continue;
708                 }
709
710                 //derection: 0-topleft, 1-topright, 2-bottomright,
                    3-bottomleft
711                 const auto connections = FGenConnectionsEqu(i, j, n, m);
712
713                 for(int i = 0; i < connections.size(); ++ i)
714                 {
715                     const auto& c = connections[i];
716                     //std::cout << "connection, a=" << c.a << " c=" << c.c
                        << ", r=" << c.r << ", p=" << c.p << std::endl;
717                     if(c.a)
718                     {
719                         const auto& r = radius[c.r][c.c];
720                         const auto& f = mns[c.r][c.c];
721                         const auto& s = f.scontb(i);
```

```
722
723                     const real K = std::pow(r, 3) / f.mu(MU1, MU2);
724                     e[l] += r * K;
725                     e[c.p] -= r * K;
726                     e.back() -= SIGMA * 2 * s * K;
727                 }
728             }
729         }
730     }
731
732     //std::cout << "okay-FGenEquForGauss" << std::endl;
733     //TFPrintMatrix("Gauss", equation);
734     return equation;
735 }
736
737 std::vector<real> FCalcPressure(const Treal& radius, const Tmns& mns)
738 {
739     //std::cout << "okay-gauss Fclac pres" << std::endl;
740     return FGaussElimination(FGenEquForGauss(radius, mns));
741 }
742
743
744
745 Treal FCalcVelocity(const std::vector<real>& pressure, const Treal&
    radius, const Tmns& mns)
746 {
747     const int n = radius.size();
748     const int m = radius.front().size();
749     auto velocity = radius;
750     for(int i = 0; i < n; ++ i)
751     {
752         for(int j = 0; j < m; ++ j)
753         {
754             const auto locs = FConnectionEnds(i, j, m);
755             const auto delp = pressure[locs.second] -
                pressure[locs.first];
756             const auto& r = radius[i][j];
757             const auto& mu = mns[i][j].mu(MU1, MU2);
758             const auto& s = mns[i][j].scontb(0);
759             velocity[i][j] = r / 8 / mu / TUBE_LENGTH * (delp * r + s *
```

```
                2 * SIGMA);
            }
        }

        return velocity;
    }

    Treal FCalcVolume(Treal velocity, const Treal& radius, const real time_step)
    {
        for(int i = 0; i < velocity.size(); ++ i)
        {
            auto& v = velocity[i];
            for(int j = 0; j < v.size(); ++ j)
            {
                v[j] = std::abs(v[j] * std::pow(radius[i][j], 2) * PI * time_step);
            }
        }

        return velocity;
    }

    real FDetermineTimeStep(const Tmns& mns, const Treal& velocity)
    {
        real min_time = HUGE;
        for(int i = 0; i < mns.size(); ++ i)
        {
            for(int j = 0; j < mns[i].size(); ++ j)
            {

                const real temp_time = mns[i][j].time(velocity[i][j], TUBE_LENGTH, TIME_DIV);
                min_time = std::min(temp_time, min_time);
            }
        }

        return min_time;
    }

```

```cpp
struct IntegrationResult
{
    Tmns mns;
    real fluid1in;
    real fluid1out;
    real fluid2in;
    real fluid2out;
};

int FCountConnections(const std::vector<Connections>& connections)
{
    int count = 0;

    for(const auto& connection: connections)
    {
        count += connection.a;
    }

    return count;
}

struct TubeWhereFlowOut
{
    real radius;
    int r;
    int c;
};

bool Fcomparison_outflow(const TubeWhereFlowOut& first, const TubeWhereFlowOut& second)
{
    return first.radius < second.radius;
}

std::vector<real> FAmountVolumeToBePushedIn(real volume, std::vector<real>& tank)
{
    auto v = tank;
    v.front() = std::min(tank.front(), volume);
    v.back() = volume - v.front();
```

```
834
835     for(int i = 0; i < tank.size(); ++ i)
836     {
837         tank[i] -= v[i];
838     }
839
840     return v;
841 }
842
843
844 Tmns FCombineFillAndAdditions(Tmns mns, const Treal& velocity, const
    Treal& radius, const std::vector<std::vector<std::vector<real>>>&
    additions)
845 {
846     for(int i = 0; i < mns.size(); ++ i)
847     {
848         auto& f = mns[i];
849         for(int j = 0; j < f.size(); ++ j)
850         {
851             f[j].update(velocity[i][j], radius[i][j], additions[i][j],
                THRESHOLD_FILL);
852         }
853     }
854
855     return mns;
856 }
857
858 Tmns FPerformIntegration(const Tmns& mns, const Treal& volume, const
    Treal& velocity, const Treal& radius)
859 {
860     const int n = volume.size();
861     const int m = volume.front().size();
862
863     real fluid1in = 0;
864     real fluid1out = 0;
865     real fluid2in = 0;
866     real fluid2out = 0;
867
868     std::vector<std::vector<std::vector<real>>> additions(n,
    std::vector<std::vector<real>>(m));
```

```
869
870         for(int i = 0; i <= n; ++ i)
871         {
872             int mt = m / 2 - i % 2;
873             for(int j = 0; j <= mt; ++ j)
874             {
875                 //std::cout << "Performing integration i=" << i << ", j="
                    << j << std::endl;
876                 const auto connections = FGenConnectionsEqu(i, j, n, m);
877
878                 /*
879                 for(const auto& connection: connections)
880                 {
881                     std::cout << "connection, a=" << connection.a << " c="
                        << connection.c << ", r=" << connection.r << ", p=" <<
                        connection.p << std::endl;
882                 }
883                 */
884                 std::vector<real> vol_in(2);
885                 std::vector<TubeWhereFlowOut> tubes_flow_out;
886                 for(int direction = 0; direction < connections.size(); ++
                    direction)
887                 {
888                     const auto& c = connections[direction];
889                     if(c.a)
890                     {
891                         const auto& f = mns[c.r][c.c];
892                         const auto& vel = velocity[c.r][c.c];
893                         const auto& vol = volume[c.r][c.c];
894                         const auto& r = radius[c.r][c.c];
895                         if(f.is_flow_into_node(direction, vel))
896                         {
897                             vol_in[f.type_fluid_into_node(direction)] += vol;
898                         }
899                         else
900                         {
901                             tubes_flow_out.push_back({r, c.r, c.c});
902                         }
903                     }
904                 }
```

```
905
906             //for(const auto& tpshin: tubes_flow_out)   std::cout <<
                "tube_push_out before short: radius=" << tpshin.radius <<
                ", r=" << tpshin.r << ", c=" << tpshin.c << std::endl;
907
908             //std::cout << "second stage reached!" << std::endl;
909             if(i == 0)
910             {
911                 fluid1out += vol_in.front();
912                 fluid2out += vol_in.back();
913                 continue;
914             }
915             if(i == n) // NOTE might remove else
916             {
917                 for(const auto& tpshin: tubes_flow_out)
918                 {
919                     additions[tpshin.r][tpshin.c] =
                        {volume[tpshin.r][tpshin.c], 0};
920                 }
921                 continue;
922
923             }
924
925             std::sort(tubes_flow_out.begin(), tubes_flow_out.end(),
                *Fcomparison_outflow);
926             for(const auto& tpshin: tubes_flow_out)
927             {
928                 //std::cout << "tube_push_out after sort: radius=" <<
                    tpshin.radius << ", r=" << tpshin.r << ", c=" <<
                    tpshin.c << std::endl;
929
930                 additions[tpshin.r][tpshin.c] =
                    FAmountVolumeToBePushedIn(volume[tpshin.r][tpshin.c],
                    vol_in);
931             }
932         }
933     }
934     //std::cout << "-------FCombineFillAndAdditions" << std::endl;
935     return FCombineFillAndAdditions(mns, velocity, radius, additions);
936 }
```

```
937
938 //Tested works Correctly
939 void FPlot(Tmns mns, Treal radius, real clock, int count)
940 {
941     std::reverse(mns.begin(), mns.end());
942     std::reverse(radius.begin(), radius.end());
943
944     real max_radius = -1;
945     real min_radius = 1e12;
946
947     for(const auto& x: radius)
948     {
949         for(auto y: x)
950         {
951             max_radius = std::max(max_radius, y);
952             min_radius = std::min(min_radius, y);
953         }
954     }
955
956     const int image_size = IMAGE_SIZE;
957     const int length = mns.front().size();
958     const int height = mns.size();
959
960     const int effective_length = image_size / (std::max(length, height)
    + 2);
961
962     drw::bmp a(image_size, image_size);
963
964     const int start_y = effective_length;
965     const int start_x = effective_length;
966     const real max_thick = effective_length;
967     const real min_thick = effective_length / 6.0;
968
969
970     int y = start_y;
971     for(int row = 0; row < mns.size(); ++ row)
972     {
973         const auto& w = mns[row];
974         int x = start_x + effective_length * (row % 2);
975         for(int col = 0; col < w.size(); ++ col)
```

```
976             {
977                 int sign = (1 - 2 * (row % 2)) * (1 - 2 * (col % 2));
978                 const real r = radius[row][col];
979                 real thick = min_thick;
980                 if(max_radius != min_radius)
981                 {
982                     thick  += (r - min_radius) * (max_thick - min_thick) /
                        (max_radius - min_radius);
983                 }
984                 a.drawVector(x, y, effective_length, thick, sign, w[col].n,
                    w[col].pos, w[col].type);
985                 x += 2 * effective_length * (sign > 0);
986             }
987 
988             y += effective_length;
989         }
990 
991         //a.save(FOLDER_SAVE_PIC + "pic-" + std::to_string(count) + "_t-" +
            std::to_string(clock) + ".bmp");
992         a.save(FOLDER_SAVE_PIC + "pic-" + std::to_string(count) + ".bmp");
993     }
994 
995     void FPlotWithoutRadius(Tmns mns, int count)
996     {
997         std::reverse(mns.begin(), mns.end());
998 
999         const int image_size = IMAGE_SIZE;
1000        const int n_cols = mns.front().size();
1001        const int n_rows = mns.size();
1002 
1003        const int length = image_size / (std::max(n_rows, n_cols) + 2);
1004 
1005        drw::bmp a(image_size, image_size);
1006 
1007        const int start_y = length;
1008        const int start_x = length;
1009        const int thick = length / 10;
1010 
1011 
1012        int y = start_y;
```

```
1013        for(int row = 0; row < mns.size(); ++ row)
1014        {
1015            const auto& w = mns[row];
1016            int x = start_x + length * (row % 2);
1017            for(int col = 0; col < w.size(); ++ col)
1018            {
1019                int sign = ((row % 2) ^ (col % 2) ? -1 : 1);
1020                a.drawStrip(x, y, length, thick, sign,
                    w[col].gen_pos_long(), w[col].type);
1021                x += 2 * length * (sign > 0);
1022            }
1023            y += length;
1024        }
1025
1026        //a.save(FOLDER_SAVE_PIC + "pic-" + std::to_string(count) + "_t-" +
        std::to_string(clock) + ".bmp");
1027        a.save(FOLDER_SAVE_PIC + "stp-" + std::to_string(count) + ".bmp");
1028    }
1029
1030    /*
1031    class Diamension
1032    {
1033    public:
1034        int m;
1035        int n;
1036
1037        Diamension(int number_cols, int number_rows): m(number_cols),
        n(number_rows) {}
1038        Diamension(const Treal& table): m(table.front().size()),
        n(table.size()) {}
1039
1040        std::pair<int, int> FConnectionEnds(int r, int c, int m)
1041        {
1042            return {FLinearLocNode(r, c/2 + 1 - (r % 2), m),
            FLinearLocNode(r + 1, c/2 + (r % 2), m)};
1043        }
1044
1045        int FTotalNodes(int n, int m)
1046        {
1047            return ((n + 1) * (m + 1) + 1) / 2;
```

```
1048        }
1049
1050        int FLinearLocNode(int i, int j, int m)
1051        {
1052            return (i * (m + 1) + (i % 2)) / 2 + j;
1053        }
1054    };
1055    */
1056
1057
1058
1059    real FMeasureWettingFluidProportion(const Tmns& mns, const Treal& radius)
1060    {
1061        real total = 0;
1062        real type_first = 0;
1063        for(int i = 0; i < radius.size(); ++ i)
1064        {
1065            for(int j = 0; j < radius[i].size(); ++ j)
1066            {
1067                const real rsq = std::pow(radius[i][j], 2);
1068                type_first += mns[i][j].sum_type_first() * rsq;
1069                total += rsq;
1070            }
1071        }
1072
1073        return type_first / total;
1074    }
1075
1076
1077
1078    void FSimulate(const Treal& radius, Tmns& mns)
1079    {
1080        TFPrintMatrix("radius", radius);
1081
1082        real clock = 0;
1083        int count = 10000;
1084
1085        FPlot(mns, radius, clock, count);
1086        FPlotWithoutRadius(mns, count);
1087
```

```
1088        real wetting_fluid_proportion;
1089        while((wetting_fluid_proportion =
            FMeasureWettingFluidProportion(mns, radius)) <=
            MAX_WETTING_PROPORTION)
1090        {
1091            std::cout << "PRS-" << count << ", clock=" << clock << ",
            proportion=" << wetting_fluid_proportion << std::endl;
1092            const auto pressure = FCalcPressure(radius, mns);
1093            //TFPrintMatrix("pressure", pressure, radius.size(),
            radius.front().size());
1094
1095            const auto velocity = FCalcVelocity(pressure, radius, mns);
1096            TFPrintMatrix("velocity", velocity);
1097
1098            const auto time_step = FDetermineTimeStep(mns, velocity);
1099            const auto volume = FCalcVolume(velocity, radius, time_step);
1100
1101            TFPrintMatrix("volume", volume);
1102
1103            mns = FPerformIntegration(mns, volume, velocity, radius);
1104            clock += time_step;
1105            ++ count;
1106            FPlot(mns, radius, clock, count);
1107            FPlotWithoutRadius(mns, count);
1108        }
1109    }
1110    //ffmpeg -framerate 10 -i filename-%03d.jpg output.mp4
1111    int main()
1112    {
1113        std::srand((unsigned)std::time(nullptr));
1114        FPrintValidityStatus();
1115        const auto radius = FReadFileRadius();
1116        auto mns = FReadFileFill();
1117
1118        std::cout << std::fixed << std::setprecision(4);
1119
1120        FSimulate(radius, mns);
1121        return 0;
1122    }
1123
```