```
// Compile with "make" / "make run" or
// g++ *.cpp -O3 -ffast-math -lopenblas -llapacke -o main


#include "task_3.h"
#include "task_4+6.h"
#include "task_5.h"
#include "task_7.h"
#include "timing.h"
#include <cblas.h>                // cBLAS Library
#include <lapacke.h>

#include <iomanip>
#include <iostream>

void task_1() {
    printf("\n\n-------------- Task 1 --------------\n\n");
    printf("See comment in main.cpp");

    // -------------------------------------------------------------
    // STREAM version $Revision: 5.10 $
    // -------------------------------------------------------------
    // This system uses 8 bytes per array element.
    // -------------------------------------------------------------
    // Array size = 80000000 (elements), Offset = 0 (elements)
    // Memory per array = 610.4 MiB (= 0.6 GiB).
    // Total memory required = 1831.1 MiB (= 1.8 GiB).
    // Each kernel will be executed 20 times.
    //     The *best* time for each kernel (excluding the first iteration)
    //     will be used to compute the reported bandwidth.
    // -------------------------------------------------------------
    // Your clock granularity/precision appears to be 1 microseconds.
    // Each test below will take on the order of 116886 microseconds.
    //     (= 116886 clock ticks)
    // Increase the size of the arrays if this shows that
    // you are not getting at least 20 clock ticks per test.
    // -------------------------------------------------------------
    // WARNING -- The above is only a rough guideline.
    // For best results, please be sure you know the
    // precision of your system timer.
    // -------------------------------------------------------------
    // Function    Best Rate MB/s  Avg time     Min time     Max time
    // Copy:           29569.4     0.048585     0.043288     0.059164
    // Scale:          17644.0     0.082248     0.072546     0.102548
    // Add:            21030.1     0.100620     0.091298     0.124700
    // Triad:          21230.7     0.100758     0.090435     0.120631
    // -------------------------------------------------------------
    // Solution Validates: avg error less than 1.000000e-13 on all three arrays
    // -------------------------------------------------------------
    // ./flops.exe

    //     FLOPS C Program (Double Precision), V2.0 18 Dec 1992

    //     Module     Error        RunTime      MFLOPS
    //                             (usec)
    //         1      4.0146e-13      0.0024   5827.9076
    //         2     -1.4166e-13      0.0007  10037.8942
    //         3      4.7184e-14      0.0039   4371.9185
    //         4     -1.2557e-13      0.0034   4355.5711
    //         5     -1.3800e-13      0.0066   4415.6439
    //         6      3.2380e-13      0.0065   4441.6299
    //         7     -8.4583e-11      0.0053   2277.1707
    //         8      3.4867e-13      0.0069   4367.6094

    //     Iterations      =  512000000
    //     NullTime (usec) =     0.0000
    //     MFLOPS(1)       =  7050.6178
    //     MFLOPS(2)       =  3461.6233
```

```
69:     //      MFLOPS(3)       =  4175.0442
70:     //      MFLOPS(4)       =  4389.7311
71: }
72:
73: void task_2() {
74:     printf("\n\n-------------- Task 2 --------------\n\n");
75:     printf("See comment in main.cpp");
76:
77:     // Memory needed (double 64-bit, 8 bytes):
78:     // (A) (2N + 1) * 8 bytes
79:     // (B) (M*N + M + N) * 8 bytes
80:     // (C) (M*L + L*N + M*N) * 8 bytes
81:     // (D) (N + N + p) * 8 bytes
82:
83:     // Floating point operations:
84:     // (A) 2N
85:     // (B) M * 2N
86:     // (C) M * 2L * N
87:     // (D) 2 * N * p (Horner Schema)
88:
89:     // Read/Write operations:
90:     // (A) Read: 2N         Write: 1
91:     // (B) Read: M*2N       Write: M*N
92:     // (C) Read: M*2L*N     Write: M*L*N
93:     // (D) Read: 2*N*p      Write: N*P
94: }
95:
96: void task_3() {
97:     printf("\n\n-------------- Task 3 --------------\n\n");
98:     printf("Functions implemented in task_3.cpp");
99: }
100:
101: void task_4(bool cblas = false) {
102:     if (cblas == false) {printf("\n\n-------------- Task 4 --------------\n\n");}
103:     size_t M, N, L, p, NLOOPS;
104:
105:     { //          Scalar product
106:     printf("----- Benchmark (A) -----\n");
107:     // Initialization
108:         N = 50'000'000;
109:         NLOOPS = 50;
110:         auto [x,y] = init_A(N);
111:     // Benchmark
112:         tic();
113:         benchmark_A(x, y, NLOOPS, cblas);
114:         double sec = toc() / NLOOPS;
115:     // Timings and Performance
116:         size_t memory = 2 * N;
117:         size_t flops  = 2 * N;
118:         print_performance(sec, memory, flops, sizeof(x[0]));
119:     printf("------------------------\n");
120:     }
121:
122:     { //      Matrix-Vector product
123:     printf("----- Benchmark (B) -----\n");
124:     // Initialization
125:         M = 8'000;
126:         N = 12'000;
127:         NLOOPS = 30;
128:         auto [A,x] = init_B(M,N);
129:     // Benchmark
130:         tic();
131:         benchmark_B(A, x, NLOOPS, cblas);
132:         double sec = toc() / NLOOPS;
133:     // Timings and Performance
134:         size_t memory = M*N + M + N;
135:         size_t flops  = 2 * M * N;
136:         print_performance(sec, memory, flops, sizeof(A[0]));
```

```
137:     printf("------------------------\n");
138:     }
139:
140:     { //     Matrix-Matrix product
141:     printf("----- Benchmark (C) -----\n");
142:     // Initialization
143:         M = 1'000;
144:         N = 2'000;
145:         L = 500;
146:         NLOOPS = 20;
147:         auto [A,B] = init_C(M,N,L);
148:     // Benchmark
149:         tic();
150:         benchmark_C(A, B, L, NLOOPS, cblas);
151:         double sec = toc() / NLOOPS;
152:     // Timings and Performance
153:         size_t memory = M*L + L*N + M*N;
154:         size_t flops  = M * 2*L * N;
155:         print_performance(sec, memory, flops, sizeof(A[0]));
156:     printf("------------------------\n");
157:     }
158:
159:     if (cblas == false)
160:     { //      Polynomial evaluation
161:     printf("----- Benchmark (D) -----\n");
162:     // Initialization
163:         N = 1'000'000;
164:         p = 200;
165:         NLOOPS = 20;
166:         auto [x,a] = init_D(N,p);
167:     // Benchmark
168:         tic();
169:         benchmark_D(x, a, NLOOPS);
170:         double sec = toc() / NLOOPS;
171:     // Timings  and Performance
172:         size_t memory = 2.0 * N;
173:         size_t flops  = 2.0 * N * p;
174:         print_performance(sec, memory, flops, sizeof(x[0]));
175:     printf("------------------------\n");
176:     }
177: }
178:
179: void task_5() {
180:     printf("\n\n------------- Task 5 -------------\n\n");
181:
182:     printf("----- Benchmark norm -----\n");
183:     // Initialization
184:         size_t N =50'000'000;
185:         size_t NLOOPS = 50;
186:         vector<double> x = init_norm(N);
187:     // Benchmark
188:         tic();
189:         benchmark_norm(x, NLOOPS);
190:         double sec = toc() / NLOOPS;
191:     // Timings and Performance
192:         size_t memory = N;
193:         size_t flops  = 2 * N;
194:         print_performance(sec, memory, flops, sizeof(x[0]));
195:     printf("------------------------\n");
196:     printf("What do you observe? Why?\n");
197:     printf("-> Faster per loop than scalar product, only loads elements of 1 vector,
instead of 2.");
198: }
199:
200: void task_6() {
201:     printf("\n\n------------- Task 6 -------------\n\n");
202:     printf("Benchmarks using cBLAS\n");
203:     task_4(true);
```

```
  204: }
  205:
  206: void task_7() {
  207:     printf("\n\n-------------- Task 7 --------------\n\n");
  208:     { // Check Ax=b
  209:     size_t N=5, Nrhs=2;
  210:     auto [A,b] = init_M(N,Nrhs);
  211:     vector<double> A_og = A;
  212:
  213:     printf("A =");
  214:     print_matrix(A,N,N);
  215:     printf("b =");
  216:     print_matrix(b,N,Nrhs);
  217:
  218:     int lda=N, ldb=Nrhs;
  219:     vector<int> ipiv(N);
  220:     LAPACKE_dgetrf(LAPACK_ROW_MAJOR, N, N, A.data(), lda, ipiv.data());
  221:     LAPACKE_dgetrs(LAPACK_ROW_MAJOR, 'N', N, Nrhs, A.data(), lda, ipiv.data(), b.dat
a(), ldb);
  222:
  223:     printf("L + U =");
  224:     print_matrix(A,N,N);
  225:     printf("x =");
  226:     print_matrix(b,N,Nrhs);
  227:
  228:     int ldc=Nrhs;
  229:     vector<double> C(N*Nrhs);
  230:     cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, Nrhs, N, 1.0, A_og.dat
a(), lda, b.data(), ldb, 0.0, C.data(), ldc);
  231:     printf("Check solution:\nA * x = ");
  232:     print_matrix(C,N,Nrhs);
  233:     }
  234:     // ################################
  235:     // Benchmark
  236:
  237:     cout << fixed << setprecision(4);
  238:     size_t NLOOPS = 1000;
  239:     cout << "N    = " << "  | 1       | 2       | 4       | 8       | 16      | 32 " << e
ndl;
  240:     cout <<       "---------|--------|--------|--------|--------|--------|-------" <
< endl;
  241:     for (int exp = 1; exp < 10; ++exp) {
  242:         cout << "Nrhs = " << static_cast<size_t>(pow(2,exp));
  243:         for (size_t N : {1, 2, 4, 8, 16, 32}) {
  244:             tic();
  245:             for (size_t i = 0; i < NLOOPS; ++i) {
  246:                 benchmark_lapacke(N, static_cast<size_t>(pow(2,exp)));
  247:             }
  248:             double sec = toc();
  249:             cout << " | " << sec;
  250:         }
  251:         cout << endl;
  252:     }
  253:     printf("\nFor fixed n, the solution time per rhs does not slow down consistently
 and scales very well.\nIts faster than expected.");
  254: }
  255:
  256: int main() {
  257:     task_1();
  258:     task_2();
  259:     task_3();
  260:     task_4();
  261:     task_5();
  262:     task_6();
  263:     task_7();
  264:     printf("\n\n");
  265:
  266:     return 0;
```

```
  267: }
```

```cpp
#include "task_3.h"
#include <vector>
#include <cassert>
#include <cmath>
using namespace std;


double scalar(vector<double> const &x, vector<double> const &y) {
    assert(x.size() == y.size());
    size_t const N = x.size();
    double sum = 0.0;
    for (size_t i = 0; i < N; ++i) {
        sum += x[i] * y[i];
    }
    return sum;
}


vector<double> matrix_vec(vector<double> const &A, vector<double> const &x) {
    size_t const N = x.size();
    size_t const M = A.size() / N;
    vector<double> b(M);

    for (size_t i = 0; i < M; ++i) {
        for (size_t j = 0; j < N; ++j) {
            b[i] += A[i*N + j] * x[j];
        }
    }
    return b;
}


vector<double> matrix_matrix(vector<double> const &A, vector<double> const &B, size_t const &M) {
    size_t const L = A.size() / M;
    size_t const N = B.size() / L;
    vector<double> C(M*N,0);

    for (size_t i = 0; i < M; ++i) {
        for (size_t k = 0; k < L; ++k) {
            for (size_t j = 0; j < N; ++j) {
                C[i*N + j] += A[i*L + k] * B[k*N + j];
            }
        }
    }
    return C;
}


vector<double> poly(vector<double> const &x, vector<double> const &a) {
    size_t N = x.size();
    size_t p = a.size();
    vector<double> y(N);
    for (size_t i = 0; i < N; ++i) {
        y[i] = a[p];
        for (size_t k = 1; k < p; ++k) {
            y[i] = y[i]*x[i] + a[p-k];
        }
    }
    return y;
}
```

```
    1: #pragma once
    2: #include <vector>
    3: using namespace std;
    4:
    5: double scalar(vector<double> const &x, vector<double> const &y);
    6: vector<double> matrix_vec(vector<double> const &A, vector<double> const &x);
    7: vector<double> matrix_matrix(vector<double> const &A, vector<double> const &B, size_
t const &M);
    8: vector<double> poly(vector<double> const &x, vector<double> const &a);
```

```
    1: #include "task_3.h"
    2: #include "task_4+6.h"
    3: #include "timing.h"
    4: #include <cblas.h>               // cBLAS Library
    5: #include <vector>
    6: #include <iostream>
    7: using namespace std;
    8:
    9: void print_performance(double sec, size_t memory, size_t flops, unsigned int size) {
   10:     printf("Memory allocated  : %.3f GByte\n", 1.0 * memory / 1024 / 1024 / 1024 * s
ize);
   11:     printf("Duration per loop : %.3f sec\n", sec);
   12:     printf("GFLOPS            : %.3f\n", 1.0 * flops / sec / 1024 / 1024 / 1024);
   13:     printf("GiByte/s          : %.3f\n", 1.0 * memory / sec / 1024 / 1024 / 1024 * s
ize);
   14: }
   15:
   16: tuple<vector<double>, vector<double>> init_A(size_t N) {
   17:     vector<double> x(N), y(N);
   18:     for (size_t i = 0; i < N; ++i) {
   19:         x[i] = i%219 + 1.0;
   20:         y[i] = 1.0 / x[i];
   21:     }
   22:     return make_tuple(x, y);
   23: }
   24:
   25: void benchmark_A(vector<double> const &x, vector<double> const &y, size_t NLOOPS, bo
ol cblas) {
   26:     size_t N = x.size();
   27:
   28:     double s(0.0), sum(0.0);
   29:     if (cblas == false) {
   30:         for (size_t i = 0; i < NLOOPS; ++i) {
   31:             s = scalar(x, y);
   32:             sum += s;
   33:         }
   34:     } else if (cblas == true) {
   35:         for (size_t i = 0; i < NLOOPS; ++i) {
   36:             s = cblas_ddot(N, x.data(), 1, y.data(), 1);
   37:             sum += s;
   38:         }
   39:     }
   40:
   41:     // Check correctness
   42:     if (static_cast<size_t>(sum) != N*NLOOPS) {printf("  !!   W R O N G  result   !!
\n");}
   43: }
   44:
   45: tuple<vector<double>, vector<double>> init_B(size_t M, size_t N) {
   46:     vector<double> A(M*N), x(N);
   47:     for (size_t i = 0; i < M; ++i) {
   48:         for (size_t j = 0; j < N; ++j) {
   49:             A[i*N + j] = (i+j)%219 + 1.0;
   50:         }
   51:     }
   52:     for (size_t j = 0; j < N; ++j) {
   53:         x[j] = 1.0/A[17*N + j];
   54:     }
   55:     return make_tuple(A, x);
   56: }
   57:
   58: void benchmark_B(vector<double> const &A, vector<double> const &x, size_t NLOOPS, bo
ol cblas) {
   59:     size_t N = x.size();
   60:     size_t M = A.size() / N;
   61:     vector<double> b(M);
   62:     double sum(0.0);
   63:
```

```
    64:     if (cblas == false) {
    65:         for (size_t i = 0; i < NLOOPS; ++i) {
    66:             b = matrix_vec(A,x);
    67:             sum += b[17];
    68:         }
    69:     } else if (cblas == true) {
    70:         for (size_t i = 0; i < NLOOPS; ++i) {
    71:             cblas_dgemv(CblasRowMajor, CblasNoTrans, M, N, 1.0, A.data(), N, x.data(
), 1, 0, b.data(), 1);
    72:             sum += b[17];
    73:         }
    74:     }
    75:
    76:     // Check correctness
    77:     if (static_cast<size_t>(sum) != N*NLOOPS) {printf("  !!   W R O N G  result   !!
\n");}
    78: }
    79:
    80: tuple<vector<double>, vector<double>> init_C(size_t M, size_t N, size_t L) {
    81:     vector<double> A(M*L), B(L*N);
    82:     for (size_t i = 0; i < M; ++i) {
    83:         for (size_t j = 0; j < L; ++j) {
    84:             A[i*L + j] = (i+j)%219 + 1.0;
    85:         }
    86:     }
    87:     // B chosen such that C[0,17]=L
    88:     // so B[i,17] = 1/A[0,i]
    89:     for (size_t i = 0; i < L; ++i) {
    90:         for (size_t j = 0; j < N; ++j) {
    91:             if (j==17) {
    92:                 B[i*N + 17] = 1.0/A[i];
    93:             } else {
    94:                 B[i*N + j] = (i+j)%219 + 1.0;
    95:             }
    96:         }
    97:     }
    98:     return make_tuple(A, B);
    99: }
   100:
   101: void benchmark_C(vector<double> const &A, vector<double> const &B, size_t L, size_t
NLOOPS, bool cblas) {
   102:     size_t M = A.size() / L;
   103:     size_t N = B.size() / L;
   104:     vector<double> C(M*N);
   105:     double sum(0.0);
   106:
   107:     if (cblas == false) {
   108:         for (size_t i = 0; i < NLOOPS; ++i) {
   109:             C = matrix_matrix(A,B,M);
   110:             sum += C[17];
   111:         }
   112:     } else if (cblas == true) {
   113:         for (size_t i = 0; i < NLOOPS; ++i) {
   114:             cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, M, N, L, 1.0, A.d
ata(), L, B.data(), N, 0.0, C.data(), N);
   115:             sum += C[17];
   116:         }
   117:     }
   118:
   119:     // Check correctness
   120:     if (static_cast<size_t>(sum) != L*NLOOPS) {printf("  !!   W R O N G  result   !!
\n");}
   121: }
   122:
   123: tuple<vector<double>, vector<double>> init_D(size_t N, size_t p) {
   124:     // x_i = i/N for i=0,...,N-1
   125:     // a_j = 1   for j=0,...,p-1
   126:     vector<double> x(N), a(p);
```

```
  127:     for (size_t i = 0; i < N; ++i) {
  128:         x[i] = static_cast<double>(i) / N;
  129:     }
  130:     for (size_t j = 0; j < p; ++j) {
  131:         a[j] = 1.0;
  132:     }
  133:     return make_tuple(x, a);
  134: }
  135:
  136: void benchmark_D(vector<double> const &x, vector<double> const &a, size_t NLOOPS) {
  137:     size_t N = x.size();
  138:     vector<double> y(N);
  139:     double sum(0.0);
  140:
  141:     for (size_t i = 0; i < NLOOPS; ++i) {
  142:         y = poly(x,a);
  143:         sum += y[0];
  144:     }
  145:
  146:     // Check correctness
  147:     if (static_cast<size_t>(sum) != NLOOPS) {printf("  !!   W R O N G  result sum =
%f  !!\n", sum);}
  148: }
```

```
    1: #pragma once
    2: #include <vector>
    3: using namespace std;
    4:
    5: void print_performance(double sec, size_t memory, size_t flops, unsigned int size);
    6: tuple<vector<double>, vector<double>> init_A(size_t N);
    7: tuple<vector<double>, vector<double>> init_B(size_t M, size_t N);
    8: tuple<vector<double>, vector<double>> init_C(size_t M, size_t N, size_t L);
    9: tuple<vector<double>, vector<double>> init_D(size_t N, size_t p);
   10:
   11: void benchmark_A(vector<double> const &x, vector<double> const &y, size_t NLOOPS, bo
ol cblas);
   12: void benchmark_B(vector<double> const &A, vector<double> const &x, size_t NLOOPS, bo
ol cblas);
   13: void benchmark_C(vector<double> const &A, vector<double> const &B, size_t L, size_t
NLOOPS, bool cblas);
   14: void benchmark_D(vector<double> const &x, vector<double> const &a, size_t NLOOPS);
```

```cpp
#include "task_4+6.h"
#include "task_5.h"
#include "timing.h"
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;

double norm(vector<double> const &x) {
    size_t N = x.size();
    double sum = 0.0;
    for (size_t i = 0; i < N; ++i) {
        sum += x[i] * x[i];
    }
    return sqrt(sum);
}

vector<double> init_norm(size_t N) {
    vector<double> x(N);
    for (size_t i = 0; i < N; ++i) {
        x[i] = i%219 + 1.0;
    }
    return x;
}

void benchmark_norm(vector<double> const &x, size_t NLOOPS) {
    double s(0.0), sum(0.0);
    for (size_t i = 0; i < NLOOPS; ++i) {
        s = norm(x);
        sum += s;
    }
    printf("||x|| = %f\n", sum/NLOOPS);
}
```

```
1: #pragma once
2: #include <vector>
3: using namespace std;
4:
5: double norm(vector<double> const &x);
6: vector<double> init_norm(size_t N);
7: void benchmark_norm(vector<double> const &x, size_t NLOOPS);
```

```cpp
#include "task_7.h"
#include <lapacke.h>
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;

tuple<vector<double>, vector<double>> init_M(size_t N, size_t Nrhs) {
    vector<double> A(N*N), b(N*Nrhs);
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            if (i != j) {
                A[i*N + j] = 1.0 / pow(abs(1.0*i-1.0*j),2);
            } else {
                A[i*N + j] = 4;
            }
        }
        for (size_t j=0; j < Nrhs; ++j) {
            b[i*Nrhs + j] = 1.0*j;
        }
    }
    return make_tuple(A, b);
}

void print_matrix(vector<double> &A, size_t M, size_t N) {
    printf("\n");
    for (size_t i = 0; i < M; ++i){
        for (size_t j = 0; j < N; ++j) {
            printf("%f ", A[i*N + j]);
        }
        printf("\n");
    }
    printf("\n\n");
}

void benchmark_lapacke(int N, int Nrhs) {
    auto [A,b] = init_M(N,Nrhs);
    int lda=N, ldb=Nrhs;
    vector<int> ipiv(N);
    LAPACKE_dgetrf(LAPACK_ROW_MAJOR, N, N, A.data(), lda, ipiv.data());
    LAPACKE_dgetrs(LAPACK_ROW_MAJOR, 'N', N, Nrhs, A.data(), lda, ipiv.data(), b.data(), ldb);
}
```

```
1: #pragma once
2: #include <vector>
3: using namespace std;
4:
5: tuple<vector<double>, vector<double>> init_M(size_t N, size_t Nrhs);
6: void print_matrix(vector<double> &A, size_t M, size_t N);
7: void benchmark_lapacke(int N, int Nrhs);
```

```
    1: //
    2: //    Gundolf Haase, Oct 18 2024
    3: //
    4: #pragma once
    5: #include <chrono>                   // timing
    6: #include <stack>
    7:
    8: //using Clock = std::chrono::system_clock;   //!< The wall clock timer chosen
    9: using Clock = std::chrono::high_resolution_clock;
   10: using TPoint= std::chrono::time_point<Clock>;
   11:
   12: // [Galowicz, C++17 STL Cookbook, p. 29]
   13: inline
   14: std::stack<TPoint> MyStopWatch; //!< starting time of stopwatch
   15:
   16: /** Starts stopwatch timer.
   17:  *  Use as @code tic(); myfunction(...) ; double tsec = toc();  @endcode
   18:  *
   19:  *  The timining can be nested and the recent time point is stored on top of the sta
ck.
   20:  *
   21:  *  @return recent time point
   22:  *  @see toc
   23:  */
   24: inline auto tic()
   25: {
   26:     MyStopWatch.push(Clock::now());
   27:     return MyStopWatch.top();
   28: }
   29:
   30: /** Returns the elapsed time from stopwatch.
   31:  *
   32:  * The time point from top of the stack is used
   33:  * if time point @p t_b is not passed as input parameter.
   34:  * Use as @code tic(); myfunction(...) ; double tsec = toc();  @endcode
   35:  * or as @code auto t_b = tic(); myfunction(...) ; double tsec = toc(t_b);  @endcode
   36:  * The last option is to be used in the case of
   37:  * non-nested but overlapping time measurements.
   38:  *
   39:  * @param[in]  t_b start time of some stop watch
   40:  * @return elapsed time in seconds.
   41:  *
   42: */
   43: inline double toc(TPoint const &t_b = MyStopWatch.top())
   44: {
   45:     // https://en.cppreference.com/w/cpp/chrono/treat_as_floating_point
   46:     using Unit       = std::chrono::seconds;
   47:     using FpSeconds = std::chrono::duration<double, Unit::period>;
   48:     auto t_e = Clock::now();
   49:     MyStopWatch.pop();
   50:     return FpSeconds(t_e-t_b).count();
   51: }
```