

```

1  #include <stdio>
2  #include <stdlib>
3  #include <string>
4  #include <float>
5  #include <csignal>
6  #include <fenv.h>
7
8  /* Manually alter mask value on stack
9   * This will work only on 32-bit system */
10
11 void handler(int signum, siginfo_t *siginfo, void *context) {
12     unsigned int base;
13     unsigned short *control;
14
15     base = (unsigned int)&base; // get data structure on stack
16     control = (unsigned short*)(unsigned int*)(base + 260); // get value ptr on stack
17
18     if(siginfo->si_code == 6) { // turn on mask for UNDERFLOW and INEXACT
19         *control |= (FE_UNDERFLOW | FE_INEXACT);
20     }
21     else if(siginfo->si_code == 4) { // turn on mask for OVERFLOW
22         *control |= FE_OVERFLOW;
23     }
24
25     fprintf(stderr, "fp exception %x at address %x\n", siginfo->si_code, siginfo->si_addr);
26     return;
27 }
28
29 int main() {
30     // turn off mask for all float exception
31     #pragma STDC FENV_ACCESS ON
32     feenableexcept(FE_DIVBYZERO | FE_INEXACT | FE_INVALID | FE_OVERFLOW | FE_UNDERFLOW);
33
34     // register sigaction
35     struct sigaction act;
36     memset(&act, 0, sizeof(act));
37
38     act.sa_sigaction = &handler;
39     act.sa_flags = SA_SIGINFO;
40
41     if (sigaction(SIGFPE, &act, NULL) < 0) {
42         fprintf(stderr, "Set sigaction failed.\n");
43         exit(EXIT_FAILURE);
44     }
45
46     // start calculation
47     double x;
48     x = DBL_MIN;
49     printf("min_normal = %g\n", x);
50
51     __asm("mov $0x12345678,%esi");
52     x = x / 13.0;
53     printf("min_normal / 13.0 = %g\n", x);
54
55     x = DBL_MAX;
56     printf("max_normal = %g\n", x);
57
58     x = x * x;
59     printf("max_normal * max_normal = %g\n", x);
60
61     return 0;
62 }
63

```

```

min_normal = 2.22507e-308
fp exception 6 at address 8048853
min_normal / 13.0 = 1.7116e-309
max_normal = 1.79769e+308
fp exception 4 at address 8048895
max_normal * max_normal = 1.79769e+308

```

```

1  function main()
2  n = 500;
3  M = rand(n, n);
4  M = M * M' + eye;
5
6  % One-for version
7  fprintf('One-for version\n');
8  t = cputime; tic;
9  A = M;
10 for k = 1:n
11     A(k:n, k) = A(k:n, k) / sqrt(A(k, k));
12     A(k, k + 1:n) = 0;
13     A(k + 1:n, k + 1:n) = A(k + 1:n, k + 1:n) - A(k + 1:n, k) * A(k + 1:n, k)';
14 end
15 fprintf('CPU time used: %f\n', cputime - t);
16 fprintf('Elapsed time used: %f\n\n', toc);
17
18 % Two-for version
19 fprintf('Two-for version\n');
20 t = cputime; tic;
21 A = M;
22 for k = 1:n
23     A(k:n, k) = A(k:n, k) / sqrt(A(k, k));
24     A(k, k + 1:n) = 0;
25     for j = k + 1:n
26         A(k + 1:n, j) = A(k + 1:n, j) - A(k + 1:n, k) * A(j, k);
27     end
28 end
29 fprintf('CPU time used: %f\n', cputime - t);
30 fprintf('Elapsed time used: %f\n\n', toc);
31
32 % Three-for version
33 fprintf('Three-for version\n');
34 t = cputime; tic;
35 A = M;
36 for k = 1:n
37     A(k:n, k) = A(k:n, k) / sqrt(A(k, k));
38     A(k, k + 1:n) = 0;
39     for j = k + 1:n
40         for i = k + 1:n
41             A(i, j) = A(i, j) - A(i, k) * A(j, k);
42         end
43     end
44 end
45 fprintf('CPU time used: %f\n', cputime - t);
46 fprintf('Elapsed time used: %f\n\n', toc);
47
48 % Matlab version
49 fprintf('Matlab version\n');
50 t = cputime; tic;
51 A = chol(M, 'lower');
52 fprintf('CPU time used: %f\n', cputime - t);
53 fprintf('Elapsed time used: %f\n\n', toc);

```

```

>> main()
One-for version
CPU time used: 1.154407
Elapsed time used: 0.564912

Two-for version
CPU time used: 1.170007
Elapsed time used: 1.044152

Three-for version
CPU time used: 1.294808
Elapsed time used: 1.291551

Matlab version
CPU time used: 0.000000
Elapsed time used: 0.005667

```

fx >> |