```
 1: // $Id: newtonsqrt.cpp,v 1.21 2016-06-13 13:44:53-07 - - $
 2:
 3: //
 4: // Newton's method to extract square root.
 5: //
 6:
 7: #include <cmath>
 8: #include <cstdlib>
 9: #include <iomanip>
10: #include <iostream>
11: #include <limits>
12: #include <sstream>
13: #include <stdexcept>
14: #include <string>
15:
16: using namespace std;
17:
18: const double EPSILON = numeric_limits<double>::epsilon();
19: const double NOTNUMBER = numeric_limits<double>::quiet_NaN();
20: const int DIGITS = numeric_limits<double>::digits10 + 6;
21:
22: bool are_close (double num1, double num2) {
23:    return fabs (num1 - num2) <= num1 * EPSILON;
24: }
25:
26: double from_string (const string& arg) {
27:    stringstream stream {arg};
28:    double result{};
29:    if (stream >> result and stream.eof()) return result;
30:    return NOTNUMBER;
31: }
32:
```

```
33:
34: double newton_sqrt (double number) {
35:    if (number < 0) throw domain_error ("newton_sqrt");
36:    if (number == 0) return 0;
37:    if (std::isnan (number) or std::isinf (number)) return number;
38:    int exponent;
39:    double fraction = frexp (number, &exponent);
40:    cout << number << " = "
41:         << fraction << " * 2 ** " << exponent << endl;
42:    double guess = ldexp (fraction, exponent / 2);
43:    double result;
44:    for (int count = 0;; ++count) {
45:       cout << "approx(" << count << ") = " << guess << endl;
46:       result = (number / guess + guess) / 2.0;
47:       if (are_close (result, guess)) break;
48:       guess = result;
49:    }
50:    return result;
51: }
52:
53: int main (int argc, char** argv) {
54:    cout << setprecision (DIGITS);
55:    for (int argi = 1; argi < argc; ++argi) {
56:       string arg = argv[argi];
57:       double number = from_string (arg);
58:       cout << endl << "argv[" << argi << "] = \"" << arg << "\" => "
59:            << number << endl;
60:       try {
61:          double value = newton_sqrt (number);
62:          cout << "sqrt (" << number << ") = " << value << endl;
63:       }catch (domain_error& error) {
64:          cout << "domain_error (" << error.what() << ")" << endl;
65:       }
66:    }
67:    return EXIT_SUCCESS;
68: }
69:
70: /*
71: //TEST// valgrind --leak-check=full --show-reachable=yes \
72: //TEST//       --log-file=newtonsqrt.out.grind \
73: //TEST//       newtonsqrt 2 10 100 1000 1e6 1e1000 foo \
74: //TEST//       >newtonsqrt.out 2>&1
75: //TEST// mkpspdf newtonsqrt.ps newtonsqrt.cpp* newtonsqrt.out*
76: */
77:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting newtonsqrt.cpp
    2: checksource newtonsqrt.cpp
    3: ident newtonsqrt.cpp
    4: newtonsqrt.cpp:
    5:      $Id: newtonsqrt.cpp,v 1.21 2016-06-13 13:44:53-07 - - $
    6: cpplint.py.perl newtonsqrt.cpp
    7: Done processing newtonsqrt.cpp
    8: g++ -Wall -Wextra -Wpedantic -Wshadow -fdiagnostics-color=never -std=gnu
++2a -Wold-style-cast -g -O0 newtonsqrt.cpp -o newtonsqrt -lm
    9: rm -f newtonsqrt.o
   10: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished newtonsqrt.cpp
```

```
 1:
 2: argv[1] = "2" => 2
 3: 2 = 0.5 * 2 ** 2
 4: approx(0) = 1
 5: approx(1) = 1.5
 6: approx(2) = 1.4166666666666651864
 7: approx(3) = 1.4142156862745096646
 8: approx(4) = 1.4142135623746986983
 9: approx(5) = 1.4142135623730492343
10: sqrt (2) = 1.4142135623730492343
11:
12: argv[2] = "10" => 10
13: 10 = 0.625 * 2 ** 4
14: approx(0) = 2.5
15: approx(1) = 3.25
16: approx(2) = 3.1634615384615383249
17: approx(3) = 3.1622778816927523878
18: approx(4) = 3.1622776601683870723
19: approx(5) = 3.1622776601683790787
20: sqrt (10) = 3.1622776601683790787
21:
22: argv[3] = "100" => 100
23: 100 = 0.78125 * 2 ** 7
24: approx(0) = 6.25
25: approx(1) = 11.125
26: approx(2) = 10.056882022471910431 7
27: approx(3) = 10.000160863201600136 6
28: approx(4) = 10.000000001293827267 5
29: approx(5) = 10
30: sqrt (100) = 10
31:
32: argv[4] = "1000" => 1000
33: 1000 = 0.9765625 * 2 ** 10
34: approx(0) = 31.25
35: approx(1) = 31.625
36: approx(2) = 31.622776679841898328 5
37: approx(3) = 31.622776601683792563 3
38: sqrt (1000) = 31.622776601683792563 3
39:
40: argv[5] = "1e6" => 1000000
41: 1000000 = 0.95367431640625 * 2 ** 20
42: approx(0) = 976.5625
43: approx(1) = 1000.28125
44: approx(2) = 1000.00003953966074732
45: approx(3) = 1000.00000000000079581
46: approx(4) = 1000
47: sqrt (1000000) = 1000
48:
49: argv[6] = "1e1000" => nan
50: sqrt (nan) = nan
51:
52: argv[7] = "foo" => nan
53: sqrt (nan) = nan
```

```
    1: ==20478== Memcheck, a memory error detector
    2: ==20478== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
    3: ==20478== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
    4: ==20478== Command: newtonsqrt 2 10 100 1000 1e6 1e1000 foo
    5: ==20478== Parent PID: 20477
    6: ==20478==
    7: ==20478==
    8: ==20478== HEAP SUMMARY:
    9: ==20478==     in use at exit: 0 bytes in 0 blocks
   10: ==20478==   total heap usage: 21 allocs, 21 frees, 793 bytes allocated
   11: ==20478==
   12: ==20478== All heap blocks were freed -- no leaks are possible
   13: ==20478==
   14: ==20478== For counts of detected and suppressed errors, rerun with: -v
   15: ==20478== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```