```
 1: // $Id: powerint.cpp,v 1.14 2016-06-13 13:44:54-07 - - $
 2:
 3: //
 4: // NAME
 5: //    powerint - power function raising a double to an int power
 6: //
 7: // SYNOPSIS
 8: //    double powi (double base, int exponent);
 9: //
10: // DESCRIPTION
11: //    The powi() function computes the value of base raised to the
12: //    power exponent.
13: //
14: // RETURN VALUES
15: //    Upon successful completion, powi() returns the value of base
16: //    raised to the power of exponent.  If exponent is 0, 1.0 is
17: //    returned regardless of the value of base (a number, 0, NaN,
18: //    or +-Infinity).  Otherwise multiplication and division are
19: //    subject to the rules of IEEE-754 floating point arithmetic,
20: //    for CPUs which support that.
21: //
22: // MATHEMATICS USED
23: //
24: //    powi (x, - n)   => powi (1 / x, n)
25: //    powi (x, 2 n)   => powi (x ** 2, n / 2)
26: //    powi (x, n + 1) => x * powi (x, n)
27: //
28: //    .EQ
29: //    x sup { - n }   = { ( 1 / x ) ) } sup n
30: //    x sup { 2 n }   = ( x sup 2 ) sup { n / 2 }
31: //    x sup { n + 1 } = x ( x sup n )
32: //    .EN
33: //
34: //    $$
35: //    x ^ { - n}    = { ( 1 / x ) } ^ n
36: //    x ^ { 2 n }   = ( x ^ 2 ) ^ ( n / 2 )
37: //    x ^ { n + 1 } = x ( x ^ n )
38: //    $$
39: //
40: //
41:
42: #include <cstdlib>
43: #include <iomanip>
44: #include <iostream>
45: #include <libgen.h>
46: #include <limits>
47: #include <sstream>
48: #include <string>
49: #include <typeinfo>
50: #include <vector>
51:
52: using namespace std;
53:
54: const int DIGITS = numeric_limits<double>::digits10 + 6;
```

```
 55:
 56: template <typename item_t>
 57: item_t from_string (const string& arg) {
 58:    stringstream stream {arg};
 59:    item_t result;
 60:    stream >> result;
 61:    return result;
 62: }
 63:
 64: void print_powi (double base, int exponent, double result) {
 65:    cout << "powi: " << base << " ** " << exponent << " * "
 66:         << result << endl;
 67: }
 68:
 69: double powi (double base, int exponent) {
 70:    double result = 1.0;
 71:    print_powi (base, exponent, result);
 72:    if (exponent < 0) {
 73:       base = 1.0 / base;
 74:       exponent = - exponent;
 75:       print_powi (base, exponent, result);
 76:    }
 77:    while (exponent > 0) {
 78:       if (exponent % 2 == 0) { /* is even */
 79:          base *= base;
 80:          exponent /= 2;
 81:       }else {
 82:          result *= base;
 83:          --exponent;
 84:       }
 85:       print_powi (base, exponent, result);
 86:    }
 87:    return result;
 88: }
 89:
 90: int main (int argc, char** argv) {
 91:    vector<string> args (&argv[1], &argv[argc]);
 92:    for (auto arg = args.cbegin(); arg != args.cend(); ++arg) {
 93:       double base = from_string<double> (*arg);
 94:       if (++arg == args.cend()) break;
 95:       int exponent = from_string<int> (*arg);
 96:       cout << endl << setprecision (DIGITS);
 97:       double result = powi (base, exponent);
 98:       cout << "ANSWER: " << base << " ** " << exponent
 99:            << " == " << result << endl;
100:    }
101:    return 0;
102: }
103:
104: /*
105: //TEST// valgrind --leak-check=full --show-reachable=yes \
106: //TEST//       --log-file=powerint.out.grind \
107: //TEST//       powerint 2 2 2 9 2 15 2 16 2 24 \
108: //TEST//       >powerint.out 2>&1
109: //TEST// mkpspdf powerint.ps powerint.cpp* powerint.out*
110: */
111:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting powerint.cpp
    2: checksource powerint.cpp
    3: ident powerint.cpp
    4: powerint.cpp:
    5:       $Id: powerint.cpp,v 1.14 2016-06-13 13:44:54-07 - - $
    6: cpplint.py.perl powerint.cpp
    7: Done processing powerint.cpp
    8: g++ -Wall -Wextra -Wpedantic -Wshadow -fdiagnostics-color=never -std=gnu
++2a -Wold-style-cast -g -O0 powerint.cpp -o powerint -lm
    9: rm -f powerint.o
   10: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished powerint.cpp
```

```
 1:
 2: powi: 2 ** 2 * 1
 3: powi: 4 ** 1 * 1
 4: powi: 4 ** 0 * 4
 5: ANSWER: 2 ** 2 == 4
 6:
 7: powi: 2 ** 9 * 1
 8: powi: 2 ** 8 * 2
 9: powi: 4 ** 4 * 2
10: powi: 16 ** 2 * 2
11: powi: 256 ** 1 * 2
12: powi: 256 ** 0 * 512
13: ANSWER: 2 ** 9 == 512
14:
15: powi: 2 ** 15 * 1
16: powi: 2 ** 14 * 2
17: powi: 4 ** 7 * 2
18: powi: 4 ** 6 * 8
19: powi: 16 ** 3 * 8
20: powi: 16 ** 2 * 128
21: powi: 256 ** 1 * 128
22: powi: 256 ** 0 * 32768
23: ANSWER: 2 ** 15 == 32768
24:
25: powi: 2 ** 16 * 1
26: powi: 4 ** 8 * 1
27: powi: 16 ** 4 * 1
28: powi: 256 ** 2 * 1
29: powi: 65536 ** 1 * 1
30: powi: 65536 ** 0 * 65536
31: ANSWER: 2 ** 16 == 65536
32:
33: powi: 2 ** 24 * 1
34: powi: 4 ** 12 * 1
35: powi: 16 ** 6 * 1
36: powi: 256 ** 3 * 1
37: powi: 256 ** 2 * 256
38: powi: 65536 ** 1 * 256
39: powi: 65536 ** 0 * 16777216
40: ANSWER: 2 ** 24 == 16777216
```

```
    1: ==20532== Memcheck, a memory error detector
    2: ==20532== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
    3: ==20532== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
    4: ==20532== Command: powerint 2 2 2 9 2 15 2 16 2 24
    5: ==20532== Parent PID: 20530
    6: ==20532==
    7: ==20532==
    8: ==20532== HEAP SUMMARY:
    9: ==20532==     in use at exit: 0 bytes in 0 blocks
   10: ==20532==   total heap usage: 26 allocs, 26 frees, 891 bytes allocated
   11: ==20532==
   12: ==20532== All heap blocks were freed -- no leaks are possible
   13: ==20532==
   14: ==20532== For counts of detected and suppressed errors, rerun with: -v
   15: ==20532== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```