```
  while (m > 0)
  { d = m % 10;    // d will be the right-most digit of m
    m /= 10;       // then remove that digit from m
    n = 10*n + d;  // and append that digit to n
  }
  cout << "The reverse is " << n << endl;
}
```

```
Enter a positive integer: 123456
The reverse is 654321
```

In this run, `m` begins with the value 123,456. In the first iteration of the loop, `d` is assigned the digit 6, `m` is reduced to 12,345, and `n` is increased to 6. On the second iteration, `d` is assigned the digit 5, `m` is reduced to 1,234, and `n` is increased to 65. On the third iteration, `d` is assigned the digit 4, `m` is reduced to 123, and `n` is increased to 654. This continues until, on the sixth iteration, `d` is assigned the digit 1, `m` is reduced to 0, and `n` is increased to 654,321.

4.12 This implements the Babylonian Algorithm:

```
#include <cmath>  // defines the fabs() function
#include <iostream>
using namespace std;
int main()
{ const double TOLERANCE = 5e-8;
  double x = 2.0;
  while (fabs(x*x - 2.0) > TOLERANCE)
  { cout << x << endl;
    x = (x + 2.0/x)/2.0;  // average of x and 2/x
  }
  cout << "x = " << x << ", x*x = " << x*x << endl;
}
```

```
2
1.5
1.41667
1.41422
x = 1.41421, x*x = 2
```

We use a "tolerance" of `5e-8` (= 0.00000005) to ensure accuracy to 7 decimal places. The `fabs()` function (for "floating-point absolute value"), defined in the `<cmath>` header file, returns the absolute value of the expression passed to it. So the loop continues until `x*x` is within the given tolerance of 2.

4.13 This program finds the integer square root of a given number. This method uses an "exhaustive" algorithm to find all the positive integers whose square is less than or equal to the given number:

```
int main()
{ float x;
  cout << "Enter a positive number: ";
  cin >> x;
  int n = 1;
  while (n*n <= x)
    ++n;
  cout << "The integer square root of " << x << " is "
       << n-1 << endl;
}
```

```
Enter a positive number: 1234.56
The integer square root of 1234.56 is 35
```