Babylonian method for square root

Algorithm:

This method can be derived from (but predates) Newton–Raphson method.

```
1 Start with an arbitrary positive start value x (the closer to the
   root, the better).
2 Initialize y = 1.
3. Do following until desired approximation is achieved.
  a) Get the next approximation for root using average of x and y
  b) Set y = n/x
```

Implementation:

```
/*Returns the square root of n. Note that the function */
float squareRoot(float n)
{
  /*We are using n itself as initial approximation
   This can definitely be improved */
  float x = n;
  float y = 1;
  float e = 0.000001; /* e decides the accuracy level*/
  while(x - y > e)
  {
    x = (x + y)/2;
    y = n/x;
  }
  return x;
}

/* Driver program to test above function*/
int main()
{
  int n = 50;
  printf ("Square root of %d is %f", n, squareRoot(n));
  getchar();
}
```

Example:

```
n = 4 /*n itself is used for initial approximation*/
```

```
Initialize x = 4, y = 1
Next Approximation x = (x + y)/2 (= 2.500000),
y = n/x  (=1.600000)
Next Approximation x = 2.050000,
y = 1.951220
Next Approximation x = 2.000610,
y = 1.999390
Next Approximation x = 2.000000,
y = 2.000000
Terminate as (x - y) > e now.
```

If we are sure that n is a perfect square, then we can use following method. The method can go in infinite loop for non-perfect-square numbers. For example, for 3 the below while loop will never terminate.

```
/*Returns the square root of n. Note that the function
  will not work for numbers which are not perfect square
unsigned int squareRoot(int n)
{
  int x = n;
  int y = 1;
  while(x > y)
  {
    x = (x + y)/2;
    y = n/x;
  }
  return x;
}

/* Driver program to test above function*/
int main()
{
  int n = 49;
  printf (" root of %d is %d", n, squareRoot(n));
  getchar();
}
```