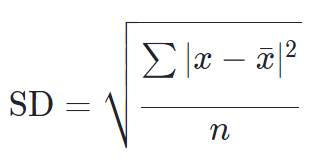
For this assignment you will get use to writing C++ code and the variety of syntax it uses. You will need to read from a file and perform some basic statistics on the numbers. The file given to you, “Homework1\_Infile.txt”, contains a series of numbers. The very first number in the file is an integer. This number tells how many other numbers there are in the file. You must read in this number and use it to create an array of doubles. Once the array is created, you need to read all the other numbers into the array. (The array is already sorted in ascending order, which will be relevant later)

You will need to take this array of numbers and compute some basic statistics: mean, median, and standard deviation. You can create the array and populate it in the main function, but you must create a function called Mean, Median, and StandardDev used for the previously mentioned stats. Below is the description of how the functions should compute.

**Mean**: To compute the mean you compute the standard average. Sum up the total value of the entire array and divide this total by the number of elements.

**Median**: The median of a set is the middle value. As mentioned earlier, the array is already sorted so it is easier to find the middle value. If there are an odd number of values, it is the exact middle value. If there are an even number, you find the median by taking the two middle values, and finding the mean between them.

**Standard Deviation**: To compute the standard deviation we follow this formula.



To translate this to code, follow these steps:

1. Find the mean (x with the bar).
2. For each element in the array (regular x) subtract it from the mean.
3. Square the computation for each element in step 2.
4. Sum up each computation from step 3.
5. Divide the sum by the number of elements.
6. Take the square root.

**Square Root:** For computing standard deviation, you should implement your own square root. DO NOT use any math libraries for this. Below is a description of the Newton-Raphson (<https://en.wikipedia.org/wiki/Newton%27s_method>) method for approximating a square root. I will also be providing the algorithm for translation into code.



The above function looks at successive solutions to a given function. The function for a square root is defined as *f(x) = x2 – C = 0*, where *C* is the number we are finding the square root. Now we have f(x), we find the derivative *f ’(x) = 2x.* We start with an intial value called x0 which well will set to be 1: x0 = 1. We use this to solve the equation the first time. Each solution gets stored into the new x, xn+1 and gets used in the next calculation. Each time we are approximating an answer and getting closer. This will not yield the actual result (unless the value is a perfect square) but an approximation. We decided when we are close enough and quit iteration. Example, find the square root of 10:

1. X0 = 1
2. f(1) = 1 – 10, f ‘(1) = 2
3. x1 =1 - -9/2 = 5.5
4. x1 = 5.5
5. Take x1 and go back to step 2 and find x2
   1. f(5.5) = 30.25 – 10 = 20.25
   2. f ‘(5.5) = 11
   3. x2 = 5.5 – 20.25/11 = 3.659
   4. Use x2 to compute x3

Eventually you will get close to 3.16228. When you get here you can stop (will explain in the algorithm). So how does this all translate to code? We start by breaking the entire process up into 3 parts. We have f(x), we have f ‘(x), and we have the main function that loops through iterations. Let us start by defining a function to compute our f(x) function and its derivative f ‘(x):

def f(x, value):

return x \* x - value

def fprime(x):

return 2 \* x

From this we can now write our main function to iterate through calculation.

def squareRoot(value):

stop <= false

x0 <= 1

x1 <= 0

while stop variable is not false

do

x1 <= x0 - f(x0, value) / fprime(x)

if x1 is equivalent to x0 then

stop <= true

else

x0 <= x1

end while

return x1

We check if the values are the same because in C++ we will lose floating point precision at some point. This loss of precision will eventually make the two floating point numbers the same making our estimation close enough.