Homework 4

1. Please see '*pi.cpp*' for the answer.

2.

```
The pi estimate is: 3.1321568489074707031
The last term included is: 0.0053280056454241275787
The total number of terms included is: 7
```

3.

```
The pi estimate is: 3.1414794921875
The last term included is: 6.0588237829506397247e-05
The total number of terms included is: 13
```

4. Yes, the previous two results support the assertion that the last term included indicates the error in the estimate. Let's take a look at the results from Question #2. The 'pi' estimate, rounded to 4 significant figures, is 3.1321. The last term included, rounded to 4 significant figures is 0.0053. In Question #3, the 'pi' estimate, rounded to 4 significant figures, is 3.1415. The last term included is 6.0588237829506397247e-05. We can see that the 'pi' estimate for Question 3 is more accurate than Question 2. Now, let's compare the last terms side-by-side:

Last Term For Q2: 0.0053280056454241275787
Last Term For Q3: 0.000060588237829506397247

Comparing the results from these two questions, clearly shows that the last term included gets significantly smaller and supports the assertion that the last term indicates the error in the estimate. The last term for Q2 has two zeros after the decimal point, and the last term for Q3 has four zeros after the decimal point - double the amount of zeros! Furthermore, the first 7 digits after the decimal for both terms is are not completely zero. There are non-zero terms, which indicates that precision has not been lost.

5.

```
The pi estimate is: 3.1415925025939941406
The last term included is: 5.9704066961648404166e-21
The total number of terms included is: 65
```

6. No, the last result does not support the assertion that the last term included indicates the error in the estimate. This is because the last term included is so small that there is no perceptible change to the 'pi' estimate. The last term included is '5.9704066961648404166e-21'. A more apt representation of this is:

0.0000000000000000000059704066961648404166

As you can see, the first 20 digits after the decimal are zeros. Which means that any change to the 'pi' estimate will be incredibly tiny and unnoticeable. Furthermore, computers cannot properly represent floating point numbers due to limitations in the binary system. Only the first 7 digits after the decimal are accurate, the rest are not. The term above has 20 zeros after the decimal. Thus, this term is extremely inaccurate and does not support the assertion that the last term included indicates the error in the estimate.

Now, let's take a look at the 'pi' estimate from Question 5: 3.1415925025939941406
Only the first 7 digits are accurate. The rest are not the true values of 'pi'. In fact, the estimate for 'pi' stops changing after the 22nd term (roughly), and no longer becomes accurate after the 6th decimal. See below.

```
$ ./pi
Welcome to the pi estimator program
Please enter how precise you want the estimate to be
>>> 0.0000001
Our pi estimate is 3.1415925025939941406
This differs from true pi by -1.5099579897537296347e-07
The smallest term included was 9.0608978098316583782e-08
The number of terms summed was 22
```

The output above demonstrates that the estimate for 'pi' stops changing around the 22nd term, and the estimate no longer becomes accurate and the error in the estimate becomes meaningless.

7. When 0 is entered for 'small', the 'pi.cpp' program produces the following:

```
Our pi estimate is 3.1415925025939941406
This differs from true pi by -1.5099579897537296347e-07
The smallest term included was 0
The number of terms summed was 147
```

At the 147th iteration, the value becomes so small, that the computer cannot keep track of it, and rounds the term to 0. The computer rounds the term to 0 because its binary number system cannot properly represent floating point numbers. Once the value of 'term' becomes 0, the 'if-statement' at the end of the 'for-loop' is satisfied and the program breaks out of the loop. The algorithm does not get stuck in an infinite loop because the statement (0 <= 0) is true. Thus, the ‘break’ is reachable and the program exits the loop.

8. When -1 is entered for 'small', the program gets stuck in an infinite loop. This is because the 'if-statement' at the end of the 'for-loop' is never satisfied. After 147 iterations, the value of 'term' becomes 0. However, there is no way for 'term' to decrease anymore; it will always remain zero. Thus, the statement: (0 <= -1) is never satisfied and the program gets stuck in an infinite loop. On every iteration, 'term' is multiplied by 0, and the result, zero, is added to 'sum'.

To fix this, simply add an 'if-statement' to check user input in the beginning. If user input is less than 0, stop the program by returning a non-zero int.

9. Please see '*pi.cpp*' for the answer.