RunningTotal Program Notes

1) The human algorithm is something like:

```
// initialize total to 0
// loop until the user is done entering values
// (i.e., continue looping as long as user hasn’t indicated “quit”) 
// { read a value from the user
//  add that value to total
// }
// OK, you’ve now to the final total so display it for the user
```

However, a direct translation of this algorithm into code will generally result in problems (“run-time, logic error”), as discussed below.

2) A WHILE loop is used (vs. a FOR loop) because the program doesn’t know how many times to go around the loop until the program is actually run. So the loop condition has to watch for the indicator of ”THE END”. Hence, an EVENT-controlled loop structure is needed.

3) Since the user is entering values (repeatedly), he/she has to be told how to indicate ”DONE”. This is the event that the while loop's CONDITION has to watch for. The value is referred to as the SENTINEL value.

4) The SENTINEL value has to be:
   a. specified in the prompt (so the user knows what to do)
   b. some legal value in terms of the correct data type
   c. some value that could not possibly be NORMAL data

Programmers often use a -1 for integer input since it’s usually not normal data, but it’s still a legal int. A 0 is sometimes appropriate, but programmers don’t usually get into a habit of using it since too often it’s an unusual value, but yet a valid value (e.g., exam score, number of sales for today).

5) WHILE loops are a PRE-test loops (as are FOR loops) (unlike DO/WHILE loops which are POST-test loops) - i.e., the while’s CONDITION is checked BEFORE the loop body code is executed. This allows for the possibility that the loop body may NEVER be executed (e.g., NumCustomers on Christmas Day at McDonalds). This means that in normal processing, when you expect the loop body code to execute, the condition must evaluate to TRUE. This leads us to consider the appropriate choice of loop structure (see below).

6) WHILE loops (as well as FOR loops and DO/WHILE loops) use a CONTINUE CONDITION (vs. an EXIT CONDITION). So:
   a. before the loop starts, the input/sentinel variable in the condition must contain a value that makes the condition TRUE. This could be done by INITIALIZING the variable to a valid value – however, this approach creates other problems (e.g., the value actually gets “processed”, even though it was not part of the user’s input). A better approach is to use the “process-read” loop structure (see below).
   b. the operation which eventually WILL make while’s condition FALSE (meaning “exit the loop”) must be done:
      i. INSIDE the loop body
      ii. at the BOTTOM of the loop body so that it’s executed IMMEDIATELY BEFORE the condition is checked

This situation is best handled with a “process-read” loop structure (see below).
7) This example DOUBLE-USES the main variable for both:
   a. storing the input number from the user each time
   b. storing the SENTINEL value indicating the user wants to quit.
   This is further reason to use the “process-read” loop structure (see below).

8) This example uses a PROCESS-READ” loop structure with PRIMING READ - that is, in pseudocode
   // priming read (i.e., get an input value)
   // while (you didn't just read in the sentinel "quit" indicator)
   // {
   //     process the data value that you just read in
   //     do another read here at the bottom
   // }
   So altogether, there will be N+1 reads (i.e., if there are 5 good scores, there will be 6 reads altogether).
   RULE: With this "sentinel" (double-use of the input variable) approach, always
         a. do a priming “read” just BEFORE the while loop starts
         b. put another identical “read” at the BOTTOM of the loop body

RULE: Always test that programs containing while loops will work correctly if the user indicates QUIT right away,
      without ever entering a valid value.