EGR115
Running Totals

Running Totals
Since first grade, students learn that addition requires two terms:

$$T = a + b$$

In fact, most learn that when there are more terms they can write it as a long sequence:

$$T = a + b + c + d + e$$

Of course, we cannot perform all the additions at once. We can only add two items – anybody who has ever added multiple items has performed it in this manner:

$$a + b \rightarrow T_1$$
$$T_1 + c \rightarrow T_2$$
$$T_2 + d \rightarrow T_3$$
$$T_3 + e \rightarrow T$$

This is beginning to indicate a repetitive process, but it requires that we know the first two values before we can start adding. Even this is problematic as we frequently do not have even two pieces of data at the same time. This is easily overcome by realizing that we can make the first addition just like the successive ones: we define T initially as 0, then change it with each additive term:

$T = 0$	T is now 0
$T + a \rightarrow T$	T is now a
$T + b \rightarrow T$	T is now a + b
$T + c \rightarrow T$	T is now a + b + c
$T + d \rightarrow T$	T is now a + b + c + d
$T + e \rightarrow T$	T is now a + b + c + d + e

When complete, the result is the same as the first-grade method.

This repetitive process is called a "**running total**". This new method is equivalent to the old method and can be applied for *any* series of successive additions:

$$T = 3 + 4 + \cdots + 9 + 10 \equiv 0 + 3 + 4 + \cdots + 9 + 10$$

Note that we started our addition with 0 indicating the initial definition of T as 0.

You've been using this since you played your first game that had a score: the score starts at 0, you earn some points, and your score goes up. The score is T and the points are a, b, c, etc.

By using running totals we can summarize any repetitive addition as:

Define the result to be 0 initially
Repeat the next step until we are out of terms:
Define the result to be the sum of the next term and the result so far

The process of running totals is fundamental to all of computer programming. It is so fundamental and so common that most programmers don't even realize at what point they learned it!

1. Use the highlighted description above to show every step of the running total process (each on its own line) to compute the sum of these values: 3, 15, -2, 12.2, -34.5

0 → T
Are there terms remaining? Yes
T + 3 → T
Are there terms remaining? Yes
T + 15 → T
Are there terms remaining? Yes
T + (-2) → T
Are there terms remaining? Yes
T + 12.2 → T
Are there terms remaining? Yes
T + (-34.5) → T
Are there terms remaining? No

Running Products

We can use a similar approach for multiplication: $P = a \cdot b \cdot c \cdot d \cdot e$

$P = 1$
$P \cdot a \rightarrow P$
$P \cdot b \rightarrow P$
$P \cdot c \rightarrow P$
$P \cdot d \rightarrow P$
$P \cdot e \rightarrow P$

Note that we start the product with 1. (Why?) This approach is called the "**running product**".

Using 1 as a multiplier is equivalent math:

$$P = 1 \cdot 7 \cdot 8 \cdot 9 \cdots 11 \cdot 12 \equiv 7 \cdot 8 \cdot 9 \cdots 11 \cdot 12$$

Note that we started our multiplication with 1 indicating the initial definition of P as 1.

By using running products we can summarize any repetitive multiplication as:

Define the result to be 1 initially
Repeat the next step until we are out of terms:
Define the result to be the product of the next term and the result so far

2. Recall that factorial of some integer, *N*, is the multiplication of all integers between 1 and *N*, inclusive. It is denoted as *N!*. Factorial is a running product. Use the highlighted description above to show every step of the running product (each step on its own line) that computes 6!

$1 \rightarrow P$
Are there terms remaining? Yes
$P \cdot 1 \rightarrow P$
Are there terms remaining? Yes
$P \cdot 2 \rightarrow P$
Are there terms remaining? Yes
$P \cdot 3 \rightarrow P$
Are there terms remaining? Yes
$P \cdot 4 \rightarrow P$
Are there terms remaining? Yes
$P \cdot 5 \rightarrow P$
Are there terms remaining? Yes
$P \cdot 6 \rightarrow P$
Are there terms remaining? No