4.2 Problem Solving: Hand-Tracing

Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.

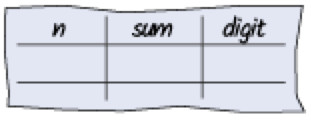
In Programming Tip 3.6, you learned about the method of hand-tracing. When you hand-trace code or pseudocode, you write the names of the variables on a sheet of paper, mentally execute each step of the code and update the variables.

It is best to have the code written or printed on a sheet of paper. Use a marker, such as a paper clip, to mark the current line. Whenever a variable changes, cross out the old value and write the new value below. When a program produces output, also write down the output in another column.

Consider this example. What value is displayed?

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```

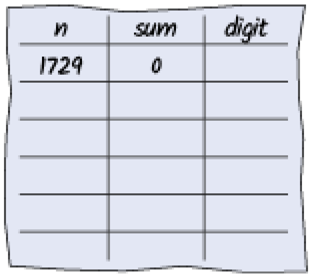
There are three variables: `n`, `sum`, and `digit`.



n	sum	digit

The first two variables are initialized with 1729 and 0 before the loop is entered.

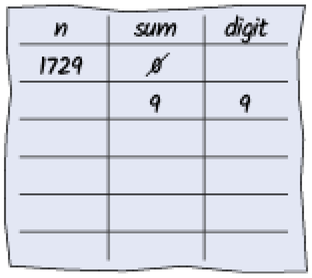
```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```



n	sum	digit
1729	0	

Because `n` is greater than zero, enter the loop. The variable `digit` is set to 9 (the remainder of dividing 1729 by 10). The variable `sum` is set to 0 + 9 = 9.

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```

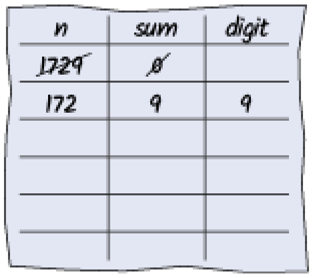


n	sum	digit
1729	~~0~~	
	9	9

Finally, n becomes 172. (Recall that the remainder in the division 1729 / 10 is discarded because both arguments are integers.)

Cross out the old values and write the new ones under the old ones.

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```

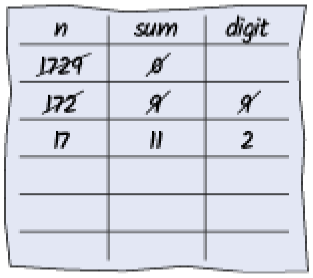


n	sum	digit
~~1729~~	~~0~~	
172	9	9

Now check the loop condition again.

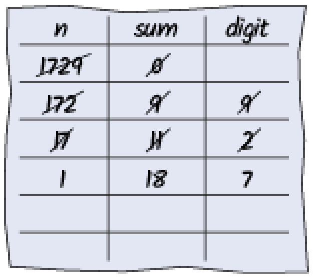
```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```

Because n is still greater than zero, repeat the loop. Now digit becomes 2, sum is set to $9 + 2 = 11$, and n is set to 17.



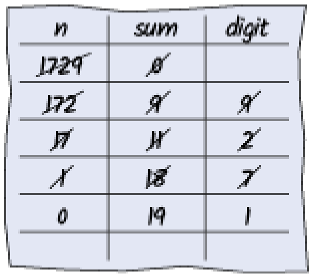
n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
17	11	2

Repeat the loop once again, setting digit to 7, sum to $11 + 7 = 18$, and n to 1.



n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
1	18	7

Enter the loop for one last time. Now digit is set to 1, sum to 19, and n becomes zero.



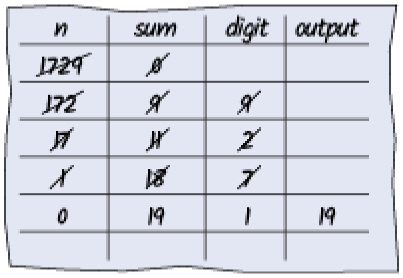
n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
~~1~~	~~18~~	~~7~~
0	19	1

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```

Because n equals zero, this condition is not true.

The condition `n > 0` is now false. Continue with the statement after the loop.

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
cout << sum << endl;
```



n	sum	digit	output
~~1729~~	~~0~~		
~~172~~	~~9~~	~~9~~	
~~17~~	~~11~~	~~2~~	
~~1~~	~~18~~	~~7~~	
0	19	1	19

This statement is an output statement. The value that is output is the value of `sum`, which is 19.

Of course, you can get the same answer by just running the code. However, hand-tracing can give you an *insight* that you would not get if you simply ran the code. Consider again what happens in each iteration:

- We extract the last digit of `n`.
- We add that digit to `sum`.
- We strip the digit off `n`.

Hand-tracing can help you understand how an unfamiliar algorithm works.

In other words, the loop forms the sum of the digits in `n`. You now know what the loop does for any value of `n`, not just the one in the example. (Why would anyone want to form the sum of the digits? Operations of this kind are useful for checking the validity of credit card numbers and other forms of ID numbers—see Exercise ••• Business P4.21.)

Hand-tracing does not just help you understand code that works correctly. It is a powerful technique for finding errors in your code. When a program behaves in a way that you don't expect, get out a sheet of paper and track the values of the variables as you mentally step through the code.

Hand-tracing can show errors in code or pseudocode.

You don't need a working program to do hand-tracing. You can hand-trace pseudocode. In fact, it is an excellent idea to hand-trace your pseudocode before you go to the trouble of translating it into actual code, to confirm that it works correctly.

SELF CHECK

•• 1.

Trace through the following loop.

```
int n = 1796;
int count = 0;
```

```
while (n > 0)
{
   int digit = n % 10;
   if (digit == 6 || digit == 7)
   {
      count++;
   }
   n = n / 10;
}
cout << count << endl;
```

n count digit Output

• **2.** What did the loop of the preceding problem do?

Compute the sum of all digits in `n`

Compute the sum of all digits in `n` that are 6 or 7

Count all digits in `n`

Count all digits in `n` that are 6 or 7

• **3.** Trace the following code, assuming that both `first` and `second` begin with a value of 1. How many values are printed?

```
while (second <= 10)
{
   cout << second << endl;
   int temp = first + second;
   first = second;
   second = temp;
}
```

1

4

5

6

•• **4.**

Trace through the following loop.

```
s = "Fred"
r = ""
i = 0
while i < length of s
  c = ith character of s
  r = c + r
  i++
Print r
```

r i c Output

• **5.** What did the loop of the preceding problem do?

Print all characters in the string s

Print the string s in reverse order

Print every other character in the string s

Count the number of characters in s

••• **6.** Implement the pseudocode of the preceding problem, prompting the user for the value of the string s.

▶ Show Code to be Completed

Complete the code in your IDE or go to CODE CHECK to complete the code and evaluate your solution.

Practice It

• **1.** Trace the following code, assuming that both `first` and `second` begin with a value of 1.   What is the last value printed?

```
while (second <= 10)
{
   cout << second << endl;
   int temp = first + second;
   first = second;
   second = temp;
}
```

5

8

10

13

•• **2.**

Rearrange the following lines of code to produce a program that prints all digits of a positive integer `n` in reverse order. Not all lines are useful.

```
-int result = 1;
.while (n > 0)
.{
. int digit = n % 10;
- if (digit != 0)
- {
-  result = result * digit;
- }
. cout << digit;
, n = n / 10;
.}
.cout << endl;
-cout << result << endl;
```

•• **3.** Write a program that prints all digits of a positive integer in reverse order. *Hint:* Start with the “sum of digits” program in this section.

▶ Show Code to be Completed

Complete the code in your IDE or go to CODE CHECK to complete the code and evaluate your solution.

••• **4.** Write a program that prints all digits of *any* integer in reverse order.

▶ Show Code to be Completed

Complete the code in your IDE or go to CODE CHECK to complete the code and evaluate your solution.

•• **5.** Trace through the following statements:

```
int n = 1;
while (n <= 3)
{
   int r = n * n;
   cout << r + ",";
   n++;
}
cout << endl;
```

n r Output

•• **6.** The preceding walkthrough showed a potential error. There is a comma after the last value. Usually, commas are between values only. Rearrange the following lines of code to produce a loop that does not have this problem. Use all lines.

```
.int n = 1;
.while (n <= 3)
.{
. if (n > 1) { cout << ","; }
, int r = n * n;
. cout << r;
. n++;
.}
.cout << endl;
```

• **7.**

The following pseudocode is intended to count the number of digits in the positive integer *n*:

```
count = 1
temp = n
```

```
While temp > 10
   Increment count.
   Divide temp by 10.0.
```

Trace the pseudocode for $n = 123$, $n = 100$, and $n = 3$. What errors do you find, and how do you fix the code?

The code is wrong for all inputs. The loop condition should be *temp != 10*

The code is wrong for inputs that are divisible by ten. The loop condition should be *temp >= 10*

The code is wrong for inputs that are less than ten. The loop condition should be *temp > 0*.

There are no errors.