Playing Computer to Improve Our Programs

What will output as a result of executing the following piece of code?

```
…
sum = 0;
for (int i=1; i <=3; i++) {
        cout << i;
        sum += i;
        }
cout << sum;
…
```

Let's "play computer" – also known as tracing your program – to produce the output. In this type of exercise, we forget that we are human, and try not to do anything too smart. We pretend to be a computer, and simply follow the program code, one instruction at a time. To do this, we need (1) the program code (above) (2) a representation of main memory (3) the input stream (if any) (4) a representation of the output window.

In this case we have no input so, in addition to the program code, we will need to show what happens in memory and what goes to the output stream (e.g., the console window).

First, sum is initialized to zero:

MAIN MEMORY

sum
0

(No output so far.) Now we enter the for loop. The index variable i is created and initialized to 1.

MAIN MEMORY

sum	i
0	1

Is i<=3? Well, right now i is equal to 1, so this condition evaluates to TRUE.

MAIN MEMORY

sum	i	i ≤ 3?
0	1	T

PROGRAM OUTPUT

Now execution can enter the body of the loop.

BODY OF LOOP PROCESSING:
cout << i; //Send value of i to output.
sum += i; //add i to sum.

Then, don't forget to increment i (the i++ in the for loop header). Here's what we have now:

MAIN MEMORY

sum	i	i ≤ 3?
~~0~~ 1	~~1~~ 2	T

PROGRAM OUTPUT

1

We come around to loop again and the first thing to do is test. Is i still <= 3? Since i is now 2, and 2 IS less than 3, the answer is TRUE, and we loop again.

By the time we finish the program (at least the piece of the program that we can see), here's what we will have in "main memory" and in our "program output":

MAIN MEMORY

sum	i	i ≤ 3?
~~0~~	~~1~~	T
~~1~~	~~2~~	T
~~3~~	~~3~~	T
6	4	F

PROGRAM OUTPUT

1
2
3
6

For all the exercises in this set of lecture notes, play computer on your own, even if the representation of memory and of the program output is not here. Create your own for every problem.

Why do we play computer with our program? It is an essential tool to help us indentify run-time, or logic, errors. The compiler helps us to identify syntax errors, but we often don't know there is anything wrong with our program until we try to run it – it may produce the wrong output; it may hang; etc.

Exercise:
What, if anything, is wrong with the following piece of code?

```
…
for (i = 0; i < 3; i++)
        {
        cout << i;
        i++;
        }
…
```

Playing computer:

MAIN MEMORY		*PROGRAM OUTPUT*
i	i < 3?	
~~0~~	T	0
~~1~~		
~~2~~	T	2
~~3~~		
4	F	

What happened? It looks like the programmer intended to output the sequence 0, 1, 2 and then stop (i<3). However, only 0 and 2 were output. The problem here is that i is incremented TWICE – once in the BODY of the loop (the statement i++;) and once because the third piece of the for header tells the computer to increment every time around the loop.

Problem:
What's wrong with the following piece of code:

```
…
while (x != 0)                          // version 1
        sum = sum + x;
…
```

At first glance, this looks like it is probably a loop that adds up a bunch of data values all called x. So what's wrong with it?

Perhaps you say that we must make sure that one of the unseen statements before this piece of code initializes sum to zero. That's very good and let's make that explicit:

```
…
sum=0;                                  // version 2
while (x != 0)
        sum = sum + x;
…
```

What else? Hmmm… (Pause for thinking … ☺)

If you start playing computer with this code, the first thing you see is that x does not have an initial value before getting into the loop. Perhaps a value is assigned earlier, in the *unseen* portion of the code.

If that is the case, there are two possibilities when the condition is evaluated on the while:

Either x is equal to zero or it is something other than zero.

If x = 0, then the while loop is skipped completely and program continues at the very next statement after the loop.

If x is NOT = 0, in other words it is any other number, just NOT zero, then the loop is entered. Each time around the loop, the value of x (whatever it is) is added to sum. So the first time around the loop, sum is 0 and x is whatever x is, they are added together and 0+x is stored in sum. The second time around, x is added to the sum, so sum is now equal to 2x. The third time around, sum becomes 3x. And so on.

What is the value of sum when the loop finally ends?

Oops! Wait a minute! How does the loop end?

It turns out that if execution goes into the loop once – because x is some number other than zero – then it can never stop looping because x never gets another value. The value of x never changes inside the loop. This is an infinite loop.

So, how do we fix it? I'm guessing that probably the author of the program wanted to read (input) data values into x and add up all those values. So the body of the loop needs another statement, to input data from the input data stream:

```
...
sum = 0;                                    // version 3
while (x != 0)
{
        cin >> x;
        sum = sum + x;
}
...
```

This is better, but we still have the problem that we can't clearly see what the value of x is when the condition on the while is evaluated.

One possible solution is to just use brute force to assign a value to x that will get execution into the loop so the data can be input and summed. How about this?

```
…
sum = 0;                                        // version 4
// x is assigned a "dummy" value, any non-zero number
x = 1;
while (x != 0)
{
        cin >> x;
        sum = sum + x;
}

…
```

Now this looks right! (Wishful thinking!) Let's add some output so we can test it:

```
…
sum = 0;                                        // version 5
x = 1;
while (x != 0)
{
        cin >> x;
        sum = sum + x;
}
cout << "The sum is " << sum;
…
```

Play computer with this piece of code and use the following input stream:
10
20
30
0

What's the output?

The sum is 60

This looks right. Cool!

However, there is actually a problem with this piece of program. The problem becomes clearer if we use a different "sentinel value" to end out data stream, say, -5.

```
…
sum = 0;                                    // version 5
x = 1;
while (x != -5)
{
        cin >> x;
        sum = sum + x;
}

…
```

The input stream would be
10
20
30
-5

And the output would be:

```
The sum is 55
```

Oops!

The problem is that our loop is supposed to end as soon as the sentinel data value is input, but the processing (adding x to sum) is always done right after data input, no matter whether the data is 'good' data or 'bad' data (the sentinel value). The sentinel data value should never be processed. It's not real data.

One solution is to reverse the statements in the body of the loop so that the input is done last, at the very end of the loop. This has the advantage that execution will follow immediately with testing the condition on the while to see whether to loop again or get out of the loop. That's what we want – to test for the sentinel value immediately after input of the data value.

```
…
sum = 0;                                    // version 6
x = 1;                          //this is no longer a good idea
while (x != -5)
{
        sum = sum + x;
        cin >> x;
}
…
```

Now we have a new problem. The initial value for x (x=1) was supposed to just get us into the loop artificially. We don't want to process THAT. So instead of giving x an artificial initial value, why not just read in the first data value before getting into the loop? Like this:

```
…
sum = 0;                                        // version 7
cin >> x;          // This only reads the FIRST data value
while (x != -5)
{
        sum = sum + x;
        cin >> x;     // This reads all the REST of the data
}
…
```

Exercise: Play computer with the above code using the input stream: 10 20 30 0. What is output
Ans.: 60 (Try it.)

Exercise: What's wrong with the following piece of program code?

```
…
sum = 0;
cin >> x;
while (x != 0)
{
        cin >> x;
        sum = sum + x;
}
…
```

Ans. There are two things wrong here:

1- the first data value is read but never processed. As soon as the loop is entered the next value of x is input.
2- the last data value (the sentinel value) is processed although it is not supposed to be added to sum. It is not a real data value – it should not be processed like real data.

Exercise:
What, if anything, is wrong with the following piece of program code? How can you improve it?

```
…
while (A != -999)
{
        cout << "enter A:";
        cin >> A;
        cout << "enter B:";
        cin >> B;
        cout << "You have entered" << A << " and " << B;
}
…
```

Ans: answers in class

More on methods for ending the input data stream.

Another method to input data until we run out of data. Just ask the user if there is any data to enter before asking for more data. For example:

```
…
while (more = 'y')
{
cout << "?: ";
cin >> x;
sum += x;
count ++;
cout << "Is there any more data (y or n)?";
cin >> more;
}
…
```