HOW NUMBERS ARE STORED

Run your arithmetic series sum program for $N = 1000$. Does everything look fine? Now run it for $N = 65550$. What do you notice happening!?

The program is pooping out at 65536. Does anyone recognize this number? Since we are dealing with computers, it's good to think about powers of 2. Let's start with $2^{10} = 1024$. Continuing with powers of 2, we get 1024, 2048, 4096, 8192, 16384, 32768, 65536. This program is going haywire precisely at 2^{16}. To understand why, we need to figure out how computers store numbers.

Let's start by reviewing base 10. It's like going back to elementary school. What do we mean by the number 4567? Well, 4567 is shorthand for four thousands, five hundreds, six tens and 7 ones. In other words,

$$4567 = 4 * 10^3 + 5 * 10^2 + 6 * 10^1 + 7 * 10^0$$

Notice here I used $10^0 = 1$, which makes the pattern clear.

Base 10 uses ten different symbols: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. The basic idea is that when we run out of symbols we need to start using two digit numbers: $10, 11, 12, 13, ...$

Base 2 is the same idea. We are only going to use two symbols: $0, 1$ Why? (Answer: computers store information with little magnets. The north pole can be up or the down pole can be up. There are only two choices, so only two "symbols".) So in base two, the only one digit numbers are $0, 1$. To count higher, we need to start using more digits. We count in base 2 thusly: $0, 1, 10, 11, 100, 101, 110, 111, 1000, \cdots$.

In analogy with base 10

$$\begin{aligned}
10 &= 1 * 2^1 + 0 * 2^0 = 2 \quad \text{(in base 10)} \\
11 &= 1 * 2^1 + 1 * 2^0 = 3 \quad \text{(in base 10)} \\
100 &= 1 * 2^2 + 0 * 2^1 + 0 * 2^0 = 4 \quad \text{(in base 10)} \\
101 &= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5 \quad \text{(in base 10)} \\
110 &= 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 6 \quad \text{(in base 10)}
\end{aligned}$$

So counting in base 2: $0, 1, 10, 11, 100, 101, 110, 111, 1000, \cdots$. is the same as base 10: $0, 1, 2, 3, 4, 5, 6, 7, 8, \cdots$.

We are now almost in a position to see why we get in trouble at 65536. Think about the biggest numbers you can store with certain numbers of digits in base 2. With 1, 2, 3, 4, 5 digits, the largest numbers are $1, 11, 111, 1111, 11111$. (These are the analogs of $9, 99, 999, 9999, 99999$, the largest numbers you can store with 1, 2, 3, 4, 5 digits in base 10.) Convert the largest base 2 numbers to base 10. You get $1, 3, 7, 15, 31$. Do you recognize the pattern? These are $2^n - 1$, that is $2-1, 4-1, 8-1, 16-1, 32-1$ where $2, 4, 8, 16, 32$ are powers of 2.

When you declare a variable to be an integer, the compute allocates 16 bits to store. The largest integer is then $2^{16}-1$, and your computer starts getting into trouble at $2^{16} = 65536$.

So now we understand whats going wrong with your arithmetic series program.

[1] Are there ways to fix this? Are there data types that can count higher? (Answer: use

```
long long int sum;
```

When you do this, you need to read things in and write tthem out with the appropriate format, eg:

```
scanf("%Ld",&N);
printf("%20Ld",sum);
```