A word about numbers...

```
int // Integer 32 bits
long long // Long integer 64 bits
float // Floating point number (uses decimal) 32 bits
double // Double precision floating point number 64 bits
char // Character 1 byte
```

for an int

low value: -2,147,483,648 -2^31

high value: 2,147,483,647 +2^31-1

Why the -1? Simple explanation is that in order to represent negative numbers - to the computer, taking the negative of a number, that is, subtracting a number from 0, is the same as inverting the bits and adding one

Enter the following code and then test it out to see what happens to your input numbers.

```
#include <stdio.h>
#include <cs50.h>

int main(void)
{
    // plain integer with no decimals - less than 10 digits
    // this works fine for a number that only needs 32 bits
    printf("number with no decimals but less than 10 digits total: \n");
    printf("defined as an integer -  32 bits - no problem with holding this number:
\n");
    int little_num = GetInt();
    printf("little number is: %i\n\n", little_num);

    // plain integer - more than 10 digits
    printf("number with no decimals but more than 10 digits and less than 20 total:
\n");
    printf("defined as an integer - 32 bits - so it can't hold the number correctly: \n
");
    int fail_num = GetInt();
    printf("fail number is: %i\n\n", fail_num);

    // a long long will hold up to 64 bits - this is what you need to use
    printf("number with no decimals but more than 10 digits but less than 20 total:\n
");
```

```
    printf("defined as a long long - 64 bits - no problem here  \n");
    long long big_num = GetLongLong();
    printf("big number number is: %lli\n", big_num);
}
```