Integer Overflow: Introduction

- Any variable in C takes up a certain amount of memory (bits).
- This limits the range of values that can be represented.
- Any time you try to go past this limit it is called an “overflow”

Integer Overflow

- A variable of type int allocates 32 bits of memory.
- We want to be able to represent negative and positive numbers, 
  so roughly half of this range is negative and roughly half is 
  positive.
- Using this logic, Integers range from $-2^{31}$ to $2^{31} - 1$, or 
  $-2147483648$ to $2147483647$
Overflow

As an **INT** it is impossible to represent outside of the range of:

| INT_MIN   | $-2^{31}$ | $-2147482648$ |
| INT_MAX   | $2^{31} - 1$ | $2147482647$ |

which is why we have **other** data-types

---

**Integer Overflow Example**

The following function can overflow for large values of a and b.

```c
// find_mid(low, high): returns the middle integer between
//   two boundaries, low and high, inclusively
//   [round down to the whole integer]
// Requires: 0 <= low <= high
int find_mid(int low, int high) {
    return (low + high) / 2;
}
```

Even though it can never return a number larger than **INT_MAX**, the result of computing \((a + b)\) is undetermined.

**Practice:** On seashell, implement the `find\_mid` function that would fix the issue above.

---

**Practice Problem: Overflow**

The function `not\_overflow\_add(a, b)` returns true if adding non-negative integers a and b will not cause overflow, otherwise, returns false.

For example,

```c
not\_overflow\_add(1, 0); // => true
not\_overflow\_add(INT\_MAX, 1); // => false
```
### Data Types

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Description</th>
<th>Printf</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer (numbers)</td>
<td>%d</td>
</tr>
<tr>
<td>char</td>
<td>Characters</td>
<td>%c</td>
</tr>
<tr>
<td>float</td>
<td>Floating Point decimal numbers</td>
<td>%f</td>
</tr>
<tr>
<td>double</td>
<td>Double precision floating value</td>
<td>%f</td>
</tr>
</tbody>
</table>

### Characters

Characters are **integers** that are typically used to hold single pieces of text data. They are 8-bit (max value of 127).

```c
char nine = '9';
char not_nine = 9; // Be careful! This is a tab!

char a = 'a';
char also_a = 97; // Equivalent, but bad practice.

char space = ' ';
char newline = '\n'; // Some characters use escape codes.
```

Later on, we will learn how to use a series of characters in a row to represent more complicated text like words and sentences.

### Debugging Tips

- Use trace statements:
  - Print out the values of variables.
  - Print out statements to show control flow.
- Automate:
  - Write your own tests!
- Simplify:
  - Comment out parts that aren’t a likely cause.
  - Remove components until you isolate the problem.
  - Writing modular code helps immensely.