CSE 351 Section 3 – Numerical Representation Limits

IEEE 754 Floating Point Standard

Goals

- ★ Represent a large range of values (*i.e.*, both very small and very large numbers)
- ★ Include a high amount of precision
- ★ Allow for real arithmetic results (*e.g.*, ∞ and NaN)

Encoding

The value of a real number can be represented in normalized scientific binary notation as:

$$(-1)^{sign} \times \text{Mantissa}_2 \times 2^{\text{Exponent}} = (-1)^S \times 1.M_2 \times 2^{E\text{-bias}}$$

The binary representation for floating point encodes these three components into separate fields:

	S	E	M
`float` bits:	1	8	23
`double` bits:	1	11	52

- `S`: The sign of the number (0 for positive, 1 for negative)
- `E`: The exponent in biased notation (unsigned with a bias of 2^{w-1}-1)
- `M`: The mantissa (also called the significand or fraction) *without* implicit leading 1

Special Cases

E	M	Interpretation
`0b0…0`	`0b0…0`	± 0
`0b0…0`	non-zero	± denormalized num
everything else	anything	± normalized num
`0b1…1`	`0b0…0`	± ∞
`0b1…1`	non-zero	NaN

Floating Point Limitations

- Overflow: Result has larger magnitude/exponent than largest normalized number (→ ∞)
- Underflow: Result has smaller magnitude/exponent than smallest denormalized number (→ 0)
- Rounding: Result falls between two representable numbers (→ one of the representable numbers)

Mathematical Properties

- Not associative: $(2 + 2^{50}) - 2^{50} \neq 2 + (2^{50} - 2^{50})$
- Not distributive: $100 \times (0.1 + 0.2) \neq 100 \times 0.1 + 100 \times 0.2$
- Not cumulative: $2^{25} + 1 + 1 + 1 + 1 \neq 2^{25} + 4$

Exercise 1: Averaging

```
/* Returns the average of the numbers in arr */

float average(int* arr, int n) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += arr[i];
  }
  return sum / n;
}
```

Issue?

Fix?

Exercise 2: Bank Account

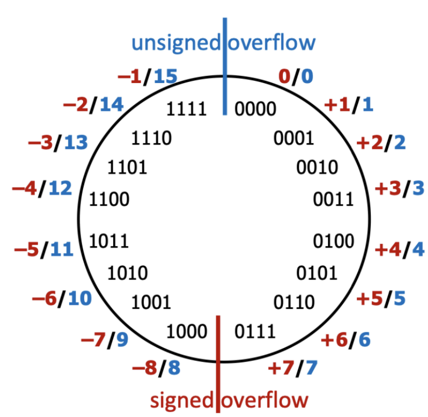
```
/* Subtracts the given amount from the bank account balance, then returns
the amount withdrawn */

float withdraw(float* balance, float amt) {
  if (amt > *balance) { return 0; }
  *balance -= amt;
  return amt;
}
```

Issue?

Fix?

Integers and Arithmetic Overflow



Arithmetic overflow occurs when the result of a calculation can't be represented in the current encoding scheme (*i.e.*, it lies outside of the representable range of values), resulting in an incorrect value.

- Unsigned overflow: The result lies outside of [UMin, UMax]; an indicator of this is when you add two numbers and the result is smaller than either number.
- Signed overflow: The result lies outside of [TMin, TMax]; an indicator of this is when you add two numbers with the same sign and the result has the opposite sign.

Lab 1b Prep: Tennis! (Past Midterm Question)

Doubles tennis is played with 2 players to a side. Your tennis club is tracking the performances of player pairings using a 16-bit encoding scheme:

```
          +-----------------------------------------+
          | perf score |    ID #1    |    ID #2    |
          +-----------------------------------------+
   Bits:       15-12         11-6           5-0
```

- Each player at the club has a unique ID.
- **Player IDs** in a pairing encoding must be distinct unsigned integers (*e.g.*, ID 0b111000 = ID 56).
- The **performance score** is encoded using Two's Complement, with more positive scores indicating better performance.

Exercise 3: Encodings

How many unique player IDs can be represented in this scheme?

Exercise 4: Limitations

What are the constraints/limitations on the range of values for the performance score?

Exercise 5: Bit Masking

Write a bit mask that will result in the binary of the performance score of a given pairing `x`. Your answer should have the 4 most significant bits set to the score and the 12 least significant bits set to `0`.

Exercise 6: Bit Manipulation

Write a series of bit manipulations (*i.e.*, shifting, using bitwise operators) to extract the ID of player #1 from a given pairing `x`. Your answer should have the 10 most significant bits set to 0 and the 6 least significant bits set to the ID of player #1.