L8 Float Representation, Evaluation order for logical expressions

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

C Programming: Tilde Operator (`~`)

What is `~` in C?

- `~` is the **bitwise NOT operator**.
- Flips every bit:
  - `0 → 1`
  - `1 → 0`

Example:

x = 5; // 0000 0101 (in 8 bits)

~x = ? // 1111 1010

Example with Unsigned Integer

```c
#include <stdio.h>
int main() {
    unsigned int x = 5;    // 0000...0101
    printf("x = %u\n", x);
    printf("~x = %u\n", ~x);
}
```

Example with Signed Integer

```c
#include <stdio.h>
int main() {
    int x = 5;   // 0000...0101
    printf("x = %d\n", x);
    printf("~x = %d\n", ~x);
}
```

Floating Point Representation (IEEE 754)

C uses IEEE 754 Standard for float (32-bit) and double (64-bit).

Sign (1 bit)	**Exponent (8 bits)**	**Mantissa (23 bits)**

$$(-1)^{\text{sign}} \times (1.\text{mantissa}) \times 2^{(\text{exponent}-127)}$$

Example

Number: 5.75

- Convert to binary:

  $5.75 = 101.11_2$

  $= 1.0111 \times 2^2$

Sign = 0 (positive)

Exponent = 127 + 2 = 129 = 10000001_2

Mantissa = 0111000...

Final 32-bit pattern:

```
0 10000001 01110000000000000000000
```

Float Representation Example – 0.15625

Convert decimal to binary fraction:

```
0.15625 × 2 = 0.3125 → 0
0.3125 × 2 = 0.625   → 0
0.625 × 2 = 1.25     → 1
0.25 × 2 = 0.5       → 0
0.5 × 2 = 1.0        → 1
```

Binary = 0.00101_2 Normalize: 1.01×2^{-3}. Sign = 0

Exponent = 127 – 3 = 124 = 01111100_2

Mantissa = 0100000...

Final bit pattern: `0 01111100 01000000000000000000000`

Another Float Example – Negative Number

Number: -7.5

Binary: $111.1_2 = 1.111 \times 2^2$

Sign = 1

Exponent = 127 + 2 = 129 = 10000001_2

Mantissa = 1110000...

Final representation:

1 10000001 11100000000000000000000

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

The Problem: Exact Representation

- Not all decimal real numbers can be represented **exactly** in binary.
- Example: `0.1` in base 10 looks simple.
- In **binary (base 2)**, `0.1` = `0.00011001100110011...` (repeating infinitely).
- A `float` has **limited bits**, so it stores only an approximation.

Example in C

```c
#include <stdio.h>
int main() {
    float x = 0.1f;
    if (x == 0.1f)
        printf("Equal\n");
    else
        printf("Not Equal\n");

    printf("x = %.25f\n", x);
    return 0;
}
```

The Problem

- Floating point numbers (`float`, `double`) are stored in **binary (IEEE 754)**.
- Many decimal values **cannot be represented exactly**.
- Equality checks (`==`) often fail due to **rounding errors**.

Example: Equality Failure

```c
#include <stdio.h>

int main() {
    float x = 0.1f;
    float y = 0.2f;
    float z = 0.3f;

    if (x + y == z) {
        printf("Equal\n");
    } else {
        printf("Not Equal\n");
    }
}
```

Why?

Internally (IEEE 754, 32-bit float):

0.1 → 0.10000000149011612

0.2 → 0.20000000298023224

0.3 → 0.30000001192092896

x + y = 0.3000000119…

z = 0.3000000119…

Tiny differences cause == to fail.

The Fix: Use an Epsilon

Instead of ==, check if the difference is within tolerance:

```c
#include <stdio.h>
#include <math.h>

int main() {
    float x = 0.1f, y = 0.2f, z = 0.3f;
    float epsilon = 1e-6;

    if (fabs((x + y) - z) < epsilon) {
        printf("Approximately Equal\n");
    } else {
        printf("Not Equal\n");
    }
}
```

Logical Operators in C

- **AND (&&)**

  True if *both* operands are true.

- **OR (||)**

  True if *at least one* operand is true.

- **NOT (!)**

  Negates a condition.

Evaluation Order – Short-Circuit

In C, evaluation is **left-to-right** with **short-circuiting**:

- `A && B` → If `A` is false, **B is not evaluated**.
- `A || B` → If `A` is true, **B is not evaluated**.

This is called **short-circuit evaluation**.

Example: Short-Circuit AND

```c
#include <stdio.h>
int main() {
    int x = 0;
    if (x != 0 && (10 / x > 1)) {
        printf("Condition true\n");
    } else {
        printf("Condition false\n");
    }
}
```

Example: Short-Circuit OR

```c
#include <stdio.h>
int main() {
    int x = 5;
    if (x == 5 || (10 / x == 2)) {
        printf("True branch\n");
    }
}
```

Example with Side Effects

```c
#include <stdio.h>
int main() {
    int a = 0, b = 1;
    if (a++ > 0 && b++) {
        printf("Inside if\n");
    }
    printf("a = %d, b = %d\n", a, b);
}
```