Numeros decimales y operadores

Por Ariel Parra

[Γα=Ω5]

Números Decimales

Los números decimales en C se representan mediante tipos de datos de punto flotante. Los tipos más comunes son:

- **float**: Precisión simple (4 bytes).
- **double**: Precisión doble (8 bytes).
- **long double**: Precisión extendida (10 bytes (x86) o 16 bytes (x64)).

Representación Binaria

Los números enteros se representan directamente en binario, usando potencias de 2. Por ejemplo, el número decimal 13 se representa en binario como `1101`, que es igual a

$$2^3 + 2^2 + 2^0$$

Los números decimales se representan en binario, usando potencias de 2 para los enteros y potencias negativas de 2 para la parte decimal. Por ejemplo, el número 13.25 se representa en binario como `1101.01`, que es igual a

$$2^3 + 2^2 + 2^0 + 2^{-2}$$

Conversión de Números Decimales a Binario

Cuando se convierte un número decimal fraccionario a binario, se descompone en una suma de potencias negativas de 2:

- **Parte Entera**: Se convierte como en la representación binaria normal.
- **Parte Fraccionaria**: Se convierte multiplicando por 2 y extrayendo la parte entera en cada paso.

Ejemplo: Convertir 0.625 a Binario

1. Multiplica 0.625 por 2:

$$0.625 \times 2 = 1.25$$

2. Toma la parte fraccionaria restante (0.25) y multiplícala por 2:

$$0.25 \times 2 = 0.5$$

3. Toma la parte fraccionaria restante (0.5) y multiplícala por 2:

$$0.5 \times 2 = 1.0$$

Entonces, 0.625 en decimal se representa en binario como:

$$0.101_2$$

Representación en IEEE 754

Supongamos que queremos representar el número decimal 5.75 en precisión simple (32 bits (tipo `float`)):

1. **Convertir a Binario**: 5.75 en binario es 101.11
2. **Normalizar**: La representación normalizada es 1.0111 × 2^2.
3. **Codificar en IEEE 754**:
   - **Signo**: 0 (positivo).
   - **Exponente**: 2 + 127 (sesgo) = 129 (en binario: 10000001).
   - **Mantisa**: 0111 (23 bits).

Entonces, el número 5.75 se representa en IEEE 754 como:

Signo	Exponente	Mantisa
0	10000001	01110000000000000000000

Operadores aritméticos

```
cout << "Suma: " << (a + b) << endl; // Suma
cout << "Resta: " << (a - b) << endl; // Resta
cout << "Multiplicación: " << (a * b) << endl; // Multiplicación
cout << "División: " << (a / b) << endl; // División (entera: redondea hacia abajo)
cout << "Módulo: " << (a % b) << endl; // Módulo
```

Operaciones simples con **módulo** `%`:

```
const int MOD = 1e9 + 7; //limite de salida
cout << (a + b ) % MOD;
cout << (a - b + MOD) % MOD; // resta
cout << (a * b ) % MOD;
cout << (a / b ) % MOD; // INCORRECTO uso del modular inverso, ocupa el teorema de Fermat
cout << ( min + rand() % (max - min + 1) ); // número aleatorio limitado en mínimo y máximo
```

Funciones en <cmath>

```
min(a, b); min({a, b, c, d});
max(a, b); max({a, b, c, d});
pow(base, exp); powl(base, exp);
pow(p, 1.0 / n); // raíz n-ésima de `p`
fmin(a.b,c.d);
fmax(a.b,c.d);
log(num); //logaritmo natural ln
log10(num);
cos(num);
sin(num);
tan(num);
sqrt(num);
sqrtl(num);
inverseSqrt(n);// raíz inversa de `n`
```

constantes de `<cmath>`: `M_PI`, `M_E`,`M_SQRT2`, etc.

Funciones de manipulación de decimales

imprimir `n` cantidad de decimales:

```
#include <iomanip> // Para usar fixed y setprecision
cout << fixed << setprecision(digits) << var; // Si digits == 0 -> redondea.
```

manipulación de decimales en `<cmath>`:

```
round(num); // 1.45 -> 1 , 1.5 -> 2
trunc(num); // 1.5 -> 1
ceil(num);  // 1.5 -> 2, con int ceil de `a/b` es: `(a + b - 1) / a`
floor(num); // 1.5 -> 1, con int `a/b` siempre sera floor
abs(num);   // -1.5 -> 1.5, 1.5 -> 1.5
```

Not A Number

Cuando intentamos calcular la raíz cuadrada de un número negativo en C++, el resultado no es un número real, por lo que la función `sqrt(-1)` devuelve un valor especial llamado `NAN` (el cual es una cosntante de C).

¿Qué tipo de `NAN`?

Existen dos tipos de `NAN` en C++:

1. **Quiet NaN (`quiet_NaN`)**:
    - Se utiliza para representar cálculos que no tienen un resultado válido, pero que no generan una excepción.
    - Se puede obtener utilizando `std::numeric_limits<double>::quiet_NaN();`.
2. **Signaling NaN (`signaling_NaN`)**:
    - Representa un valor que, cuando se utiliza en una operación, debería generar una excepción o señal.
    - Se puede obtener con `std::numeric_limits<double>::signaling_NaN();`.

Puedes verificar si un valor es `NAN` utilizando la función `isnan()`.

Infinito

El infinito en C se maneja de manera similar con la constante `INFINITY`. Este valor es devuelto por operaciones matemáticas que resultan en un valor infinito, como una división entre cero. `1.0 / 0.0 = INFINITY`

Verificación y obtención de `INFINITY`

- Para verificar si un valor es infinito, se utiliza la función `isinf()`.
- El valor de infinito se puede obtener con los limtes de double en c++ `std::numeric_limits<double>::infinity();`.

Problemas

- **1A** Theatre Square ⮥
- **219158U** Float or int ⮥

Referencias

- Kaze Emanuar. (2023). *The Truth about the Fast Inverse Square Root on the N64* [video]. Recuperado de https://youtu.be/tmb6bLbxd08?si=MmDNXxpHCrbcor92
- Bobater. (2023). *How can Computers Calculate Sine, Cosine, and More? | Introduction to the CORDIC Algorithm #SoME3*. Recuperado de https://www.youtube.com/watch?v=bre7MVlxq7o
- Nemean. (2020). *Fast Inverse Square Root — A Quake III Algorithm* [video]. Recuperado de https://youtu.be/p8u_k2LIZyo?si=4mprBbVnW_ANZqNF
- Creel. (2012) *Floating Point Bit Hacks Every Programmer Should Know (Including Fast Inverse Square Root - Quake)* [video]. Recuperado de https://youtu.be/ReTetN51r7A?si=lB7LSkmvFdlfvq16
- Wiffin, E. (2023). *Floating Point Math*. Recuperado de https://0.30000000000000004.com/
- Wikipedia. (2024). *IEEE 754*. Recuperado de https://es.wikipedia.org/wiki/IEEE_754/