ETH zürich

Session 04
Floating Point II, Functions I

Informatik I

Lan Zhang

14.03.2025

Schedule

1. Review of Last Week
2. Main Topics
   - Normalized FP-Systems
   - Functions

Feedback

- Boolean Functions
  - expr == true/1 is equivalent to expr
  - expr == false/0 is equivalent to !expr
  - Bitwise Operators: performs operation on binary representation of numbers
    - `&` (AND)
    - `|` (OR)
    - `^` (XOR) (equivalent to !=)
    - `~` (NOT)
- Fibonacci Primes
  - Update step: `int next = prev + curr; prev = curr; curr = next;`
- Overflow Check
  - `curr < MAX_INT - prev`

Please use correct formatting and indentation (C++ Style Guide - C++ Guides)

Normalized Floating Point Systems

$$F^*(\beta, p, e_{min}, e_{max})$$

$$\pm d_0.d_1 \dots d_{p-1} \times \beta^e$$

- $d_i \in \{0, \dots, \beta - 1\}, e \in \{e_{min}, \dots, e_{max}\}$
- $d_0 \neq 0$
- # Bits for exponent = $log_\beta(e_{max} - e_{min} + 1)$

Calculation:

1. Transform operands to equal exponents
2. Binary addition
3. Renormalization
4. Round to p significant digits

Special values in IEEE Standard 754

	mantisse	exponent
• ±0	all 0	all 0
• ±∞	all 0	all 1
• nan*	all > 0	all 1

$1.001 \cdot 2^{-1} + 1.111 \cdot 2^{-2}$

$1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$

$$\begin{array}{r} 1.001 \cdot 2^{-1} \\ +0.1111 \cdot 2^{-1} \\ \hline 10.0001 \cdot 2^{-1} \end{array}$$

$1.00001 \cdot 2^0$

$1.000 \cdot 2^0$

*not a number

Normalized FP-Systems: Example

State the following numbers in $F^*(2, 3, -1, 2)$

1. the largest number
2. the smallest number
3. the smallest non-negative number

Compute how many numbers are in the set $F^*(2, 3, -1, 2)$

Sol.

1. $1.11 * 2^2 = 7$

2. $-1.11 * 2^2 = 7$

3. $1.00 * 2^{-1} = 0.5$

$$2 * 2^2 * 4 = 32$$

2 * 2 ^ # of variable digits * # of exponents

Binary Representation of Integers & Decimal Numbers

Binary **integers** are composed of powers of 2

e.g. $45 == 0b101101 == 1*2^5 +0*2^4 +1*2^3 +1*2^2 +0*2^1 +1*2^0$

To represent **decimal numbers**, we use negative exponents

e.g. $5.25 == 0b101.01 == 1*2^2 +0*2^1 +1*2^0 +0*2^{-1} +1*2^{-2}$

Conversion

e.g. $11.1 = 0b1011.0\overline{0011}$

x	b_i	$x - b_i$	$2 \cdot (x - b_i)$
0.1	0	0.1	0.2
0.2	0	0.2	0.4
0.4	0	0.4	0.8
0.8	0	0.8	1.6
1.6	1	0.6	1.2
1.2	1	0.2	0.4
0.4	0	0.4	0.8

all arithmetic operations round the exact result to the closest representable number

Floating Point Guidelines

1. Do not test floating point numbers for equality
   - Specify tolerance range `(abs(x-y)<tol);`
   - Use <= / >= instead of == / !=

Example:

```
float a = 1.05f;
if (100*a == 105.0f) <- false since 1.05 not exactly representable
   std::cout << "no output\n"
```

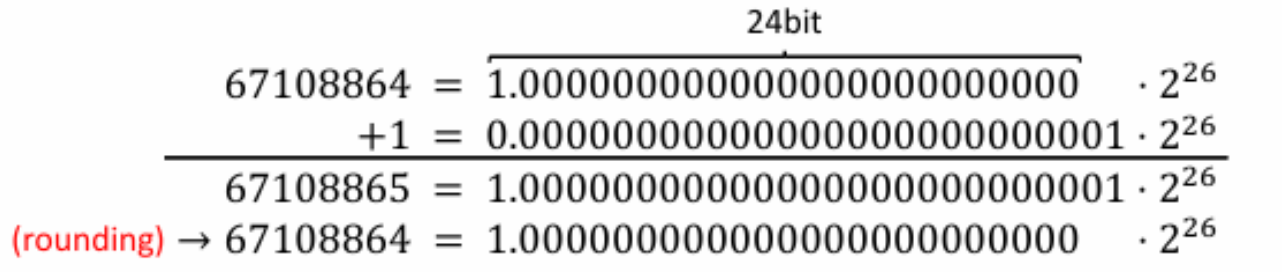
$$1.05 = \overbrace{1.00001100110011001100110}^{\text{24bit}}01\ldots \cdot 2^0$$
$$\rightarrow 1.049999995231\ldots = 1.00001100110011001100110 \cdot 2^0$$

Floating Point Guidelines

2. Do not add floating point numbers of very different sizes
   - Logarithmic number axis

Example:

```
float a = 67108864.0f + 1.0f; -> significand too short
if (a > 67108864.0f)
   std::cout << "This is not output ... \n";
```



Floating Point Guidelines

3. Do not subtract floating point numbers of similar sizes
   - Cancellation: loss of significant digits propagates through subsequent calculations

   Example:

   Consider sequence $x_{n+1} = 6x_n - 1$

   Conputing some sequences for given x_0:

   - e.g. $x_0 = 1 \rightarrow x_1 = 5, x_2 = 29, x_3 = 173, \ldots$
   - e.g. $x_0 = 0.2 \rightarrow x_1 = 0.2, x_2 = 0.2, x_3 = 0.2, \ldots$ -> C++ claims $x_{14} \approx 622.982$

`float` represents 0.2 as 0.20000000298...

Thus: $6 \cdot x_0 - 1 \neq 1.2 - 1$ but rather:

$$x_1 = 0.20000004768 \ldots$$
$$x_2 = 0.20000028610 \ldots$$
$$x_3 = 0.20000171661 \ldots$$
$$\vdots$$

<- accumulation of errors

Functions

- Make frequently used functionalities reusable
- Avoid code duplication
- Structure code

```
Type fname(type1 pname1,type2 pname2,..., typeN pnameN) {
     block;
     return Type;
}
fname(expr1, expr2,...,exprN)// have to be convertible into type1-N
```

- Definition outside main scope
- Passed parameters are stored only temporarily, i.e. do not change value outside the function

Void functions

- only have an effect (no return value)
- do not require a return, unless you want to exit the function

Pre/Post Conditions

```
\\PRE: as general as possible
What must apply to function calls?
What is the definition scope of the function?

\\POST: as detailed as possible
What happens after the function call?
What value does the function return?
```

PRE and POST can be verified with assertion!

```
#include <cassert>
...
assert([condition])
```

- gives information on where and what condition failed
- recognize incorrect (user) entries and avoid subsequent undefined behaviour

Functions: Examples

Does the loop terminate for all values of the variable a, for which the precondition holds?

```
 // PRE: a > 0
void f1(int a) {
     bool b = false;
     for (; a > 0; a -= b) b = !b;
 }

// PRE: a == b * 3n , for some natural number n
void f2(double a, double b) {
     while (true) {
            if (a == b) break;
            a /= 3;
            b *= 3;
     }
}
```