Mathematical Software Programming (02635)

Lecture 2 — September 13, 2018

Instructor: Martin S. Andersen

Fall 2018

Checklist — what you should know by now

- How to write a simple program in C (`int main(void) {}`)
- Basic data types (`int`, `long`, `float`, `double`, ...)
- Basic input/output (`printf`, `scanf`)
- Implicit/explicit typecasting
- How to compile and run a program from the terminal / command prompt

This week

Topics

- Control statements and loops
- Finite precision arithmetic
- Application: numerical integration

Learning objectives

- Evaluate discrete and continuous mathematical expressions
- Choose appropriate data types and data structures for a given problem

Example 1: fpnum.c

```
#include <stdio.h>

int main(void) {

    double a = 1.0, b = 1e-16, c = -1.0;
    printf("(a + b) + c = %.4e\n",(a+b)+c);
    printf("a + (b + c) = %.4e\n",a+(b+c));

    return 0;
}
```

Output

```
$ ./fpnum
(a + b) + c = 0.0000e+00
a + (b + c) = 1.1102e-16
```

Example 2: intnum.c

```c
#include <stdio.h>

int main(void) {

    /* a = 2^30 = 1073741824 */
    int a = 1 << 30;
    printf("  a = %d\n",a);
    printf("2*a = %d\n",2*a);

    return 0;
}
```

Output

```
$ ./intnum
  a = 1073741824
2*a = -2147483648
```

What went wrong?

- Associative property of addition

$$(a+b)+c = a+(b+c)$$

  does not hold for finite-precision floating point arithmetic
- We need to learn about floating point numbers!
- Integer operations may overflow!
- Without overflow, integer arithmetic satisfies commutative, associative, and distributive properties
  - commutative: $x+y=y+x$ and $xy=yx$
  - associative: $(x+y)+z=x+(y+z)$ and $(xy)z=x(yz)$
  - left distributive: $x(y+z)=(xy)+(xz)$
  - right distributive: $(y+z)x=(yx)+(zx)$

Floating point numbers

$$x = s \cdot (d_0.d_1 d_2 \ldots d_{p-1})_b \cdot b^E$$

- b is the base (e.g., 2 or 10)
- s represents the sign
- $d_0.d_1 d_2 \ldots d_{p-1}$ is the so-called *mantissa* or *significant*
- d_i is the ith digit of the mantissa
- E is the *exponent*
- p is the *precision*
- x is *normal* is $d_0 \neq 0$; otherwise x is *subnormal*

Floating point numbers (continued)

Machine epsilon

$$\epsilon = (0.00\ldots01)_b = b^{-(p-1)}$$

Warning: some books/authors use a different definition!

Unit round-off

$$u = \frac{\epsilon}{2}$$

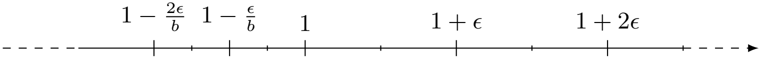
Unit in the last place (ulp)

$$\mathrm{ulp}(x) = (0.00\ldots01)_b \cdot b^E = \epsilon \cdot b^E$$

$\mathrm{ulp}(x)$ is the gap between $|x|$ and the next larger FP number

Representable positive numbers

Floating-point number system with precision p, base b, and exponent $E \in \{E_{\min}, \ldots, E_{\max}\}$



- largest number (let $E = E_{\max}$ and $d_i = b - 1$)

$$N_{\max} = (b-1)\sum_{i=0}^{p-1} b^{E_{\max}-i} = b^{E_{\max}}(b - b^{-(p-1)})$$

- smallest *normal* number (let $E = E_{\min}$, $d_0 = 1$, and $d_i = 0$ for $i > 0$)

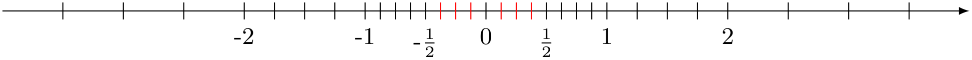
$$N_{\min} = b^{E_{\min}}$$

- smallest *subnormal* number (let $E = E_{\min}$, $d_{p-1} = 1$, and $d_i = 0$ for $i < p - 1$)

$$b^{E_{\min}-(p-1)}$$

Example

Suppose $b = 2$, $p = 3$, and $E \in \{-1, 0, 1\}$



- machine epsilon: $\epsilon = 2^{-2} = 0.25$
- largest number: $N_{\max} = 2^2(1 - 2^{-3}) = 3.5$
- smallest *normal* number: $N_{\min} = 2^{-1} = 0.5$
- smallest *subnormal* number: $2^{-3} = 0.125$

Rounding

- round to nearest
  - ties to even (aka *round to even*, *banker's rounding*, and *scientific rounding*)
  - ties away from zero
- directed rounding
  - round toward zero
  - round toward $+\infty$ (or $-\infty$)

Example

Suppose $b = 10$ and $p = 2$, hence $\epsilon = 0.1$

x	nearest (even)	nearest (away)	zero	$+\infty$	$-\infty$
1.05	1.0	1.1	1.0	1.1	1.0
1.15	1.2	1.2	1.1	1.2	1.1
-1.05	-1.0	-1.1	-1.0	-1.0	-1.1
-1.15	-1.2	-1.2	-1.1	-1.1	-1.2

Computational model (no underflow/overflow)

Operation "**op**" (addition/subtraction/multiplication/division with *round to nearest*)

$$\mathrm{fl}(x \ \mathbf{op}\ y) = (x \ \mathbf{op}\ y)(1+\delta) \qquad |\delta| \le u$$

Function evaluation (for example, exp, log, sin, or cos)

$$\mathrm{fl}(f(x)) = (1+\delta_1) f((1+\delta_2)x)$$

Absolute and relative error

$$e_{\mathrm{abs}} = |\mathrm{fl}(f(x)) - f(x)|$$

$$e_{\mathrm{rel}} = \frac{|\mathrm{fl}(f(x)) - f(x)|}{|f(x)|}$$

Cancellation

Cancellation is **loss of significance** in calculations with finite-precision arithmetic

Suppose $b = 10$ and $p = 3$ ($\epsilon = 10^{-2}$) and let $x = 1.02$, $y = 1.01$, and $z = 1.23 \cdot 10^{-2}$

$$\begin{aligned} \mathrm{fl}(x - z) &= 1.01 \cdot 10^{0} & e_{\mathrm{abs}} &= 2.3 \cdot 10^{-3} \quad e_{\mathrm{rel}} \approx e_{\mathrm{abs}} \\ \mathrm{fl}(x - y) &= 1.00 \cdot 10^{-2} & e_{\mathrm{abs}} &= e_{\mathrm{rel}} = 0 \\ \mathrm{fl}(\mathrm{fl}(x + z) - y) &= 2.00 \cdot 10^{-2} & e_{\mathrm{abs}} &= 2.3 \cdot 10^{-3} \quad e_{\mathrm{rel}} \approx 10^{-1} \end{aligned}$$

Subtraction may cause many significant digits to disappear

Catastrophic cancellation

Relative error increases substantially more than absolute error

IEEE 754: Standard for FP Arithmetic

- Technical standard for floating-point computation established in 1985, updated in 2008
- Several binary and decimal formats (i.e., $b = 2$ or $b = 10$)
- Defines rounding modes and required operations (arith., conversions, total ordering, ...)

binary32 (single precision)

- base 2
- 32 bits: 1 sign, 8 exponent, 23 mantissa ($p = 24$, d_0 is implicit)
- C data type: `float`

binary64 (double precision)

- base 2
- 64 bits: 1 sign, 11 exponent, 52 mantissa ($p = 53$, d_0 is implicit)
- C data type: `double`

Example: binary64 (“double precision”) floating-point numbers

Floating-point number system with $b = 2$, $p = 53$, and $E \in \{-1022, \ldots, 1023\}$

- $\epsilon = 2^{-52} \approx 2.22 \cdot 10^{-16}$
- $N_{\max} = 2^{1023}(2 - 2^{-52}) \approx 1.8 \cdot 10^{308}$
- $N_{\min} = 2^{-1022} \approx 2.2 \cdot 10^{-308}$

Representing special values

11 bits for exponent: 2048 possible values (normal numbers use only 2046 of these)

- Signed zero and subnormal numbers
- Signed `INFINITY` (divide-by-zero, overflow, $\log(0)$, ...) and `NAN` (not a number)

Invalid operations yield `NAN`, e.g.,

$$\sqrt{-1}, \quad 0 \cdot \infty, \quad 0/0, \quad \infty/\infty, \quad \infty - \infty$$

Decimal floating point numbers

- standardized in 2008
- limited hardware support (software implementation: `libdfp`)
- emulate exact decimal rounding (e.g., in finance)
- different binary representations allowed (binary coded / decimal coded)

decimal32 (32 bits)

Floating-point number system with $b = 10$, $p = 7$, and $E \in \{-95, \ldots, 96\}$

decimal64 (64 bits)

Floating-point number system with $b = 10$, $p = 16$, and $E \in \{-383, \ldots, 384\}$

Floating-point unit (FPU)

Floating-point operations (`+`, `-`, `*`, `/`, square root, bit shifting) may be carried out by

- Integrated FPU
- Add-on FPU
- FPU emulator (floating-point library)

Some FPUs also support transcendental functions (e.g., log, exp, trig.)

Intel x87 and extended precision

- Initially add-on FPU (Intel 8087)
- Floating-point related subset of x86 instruction set
- 80-bit double-extended precision used internally ($p = 64$, $\epsilon \approx 10^{-19}$)
- Many systems implement `long double` as double-extended on the x86 architecture

Classifying floating-point numbers (C99)

Header file `math.h` includes macros and functions:

- `isfinite(x)`, `isnormal(x)`, `isnan(x)`, `isinf(x)`, `fpclassify(x)`

`fpclassify(x)` returns one of the following values:

- `FP_NAN`, `FP_INFINITE`, `FP_ZERO`, `FP_SUBNORMAL`, `FP_NORMAL`

Example

Check if x is `NAN`

```
double x = 0.0/0.0;
if (fpclassify(x) == FP_NAN)
    printf("x is not a number\n");
if (isnan(x))
    printf("x is not a number\n");
```

Floating-point environment (C99)

Header file `fenv.h` includes macros and functions for controlling FP environment

- `fesetround(int mode)`, `fegetround(void)`, ...
- rounding modes: `FE_DOWNWARD`, `FE_TONEAREST`, `FE_TOWARDZERO`, `FE_UPWARD`

Inform compiler that application might access the floating-point environment

```
#pragma STDC FENV_ACCESS ON  //C99 (currently not supported by GCC/Clang)
```

Example

Set rounding mode to *round toward zero*

```
if (fesetround(FE_TOWARDZERO)) {
    fprintf(stderr, "Failed to set rounding mode\n");
    exit(EXIT_FAILURE);
}
```

GCC compiler option: `-frounding-math`

Exercises

Exercise 5-2 in "Writing Scientific Software"

Evaluate

$$\frac{1-\cos(x)}{x^2}$$

for $x = 10^{-k}$, $k = 0, 1, 2, \ldots, 16$.
Taylor expansion of $\cos(x)$ around 0:

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

Double angle identity:

$$\cos(2x) = 1 - 2\sin^2(x)$$

Things to avoid with floating-point numbers

- Do **not** test for equality between floating-point numbers
- Do **not** use floating-point numbers as loop counters
- Avoid subtracting nearly equal quantities and then divide by something small