Mathematical Software Programming (02635)

Module 2 — Fall 2016

Instructor: Martin S. Andersen

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

```c
/* 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

```c
/* intnum.c */
#include <stdio.h>

int main(void) {

    int a = 1 << 30; /* a = 2^30 = 1073741824 */

    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

$$x = s \cdot (d_0.d_1 d_2 \ldots d_n)_b \cdot b^e$$

- b is the base (e.g., 2 or 10)
- s represents the *sign*
- $d_0.d_1 d_2 \ldots d_n$ is the so-called *mantissa* or *significant*
- d_i is the ith digit of the mantissa
- e is the *exponent*
- x is *normal* is $d_0 \neq 0$; otherwise x is *subnormal*
- Invalid operations yield `NaN` (not a number)
  - $\sqrt{-1}$, $0 \cdot \infty$, $0/0$, ∞/∞, $\infty - \infty$
  - condition `x != x` is true only if `x` is `NaN`

IEEE Standard for FP Arithmetic (IEEE 754)

- Technical standard for floating-point computation established in 1985
- Several binary and decimal formats

Single precision (binary32)

- base 2
- 32 bits: 1 sign, 8 exponent, 23 mantissa
- In C: `float`

Double precision (binary64)

- base 2
- 64 bits: 1 sign, 11 exponent, 52 mantissa
- In C: `double`

Classifying floating-point numbers (C99)

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

- `isfinite(x)`, `isnormal(x)`, `isnan(x)`, `isinf(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");
if (x != x)
    printf("x is not a number\n");
```

Exercises

Exercise 5-2 in "WSS"

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:

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