6. Data Types

Hengfeng Wei (魏恒峰)
hfwei@nju.edu.cn

Nov. 01, 2024

Review

Functions

Function Definition

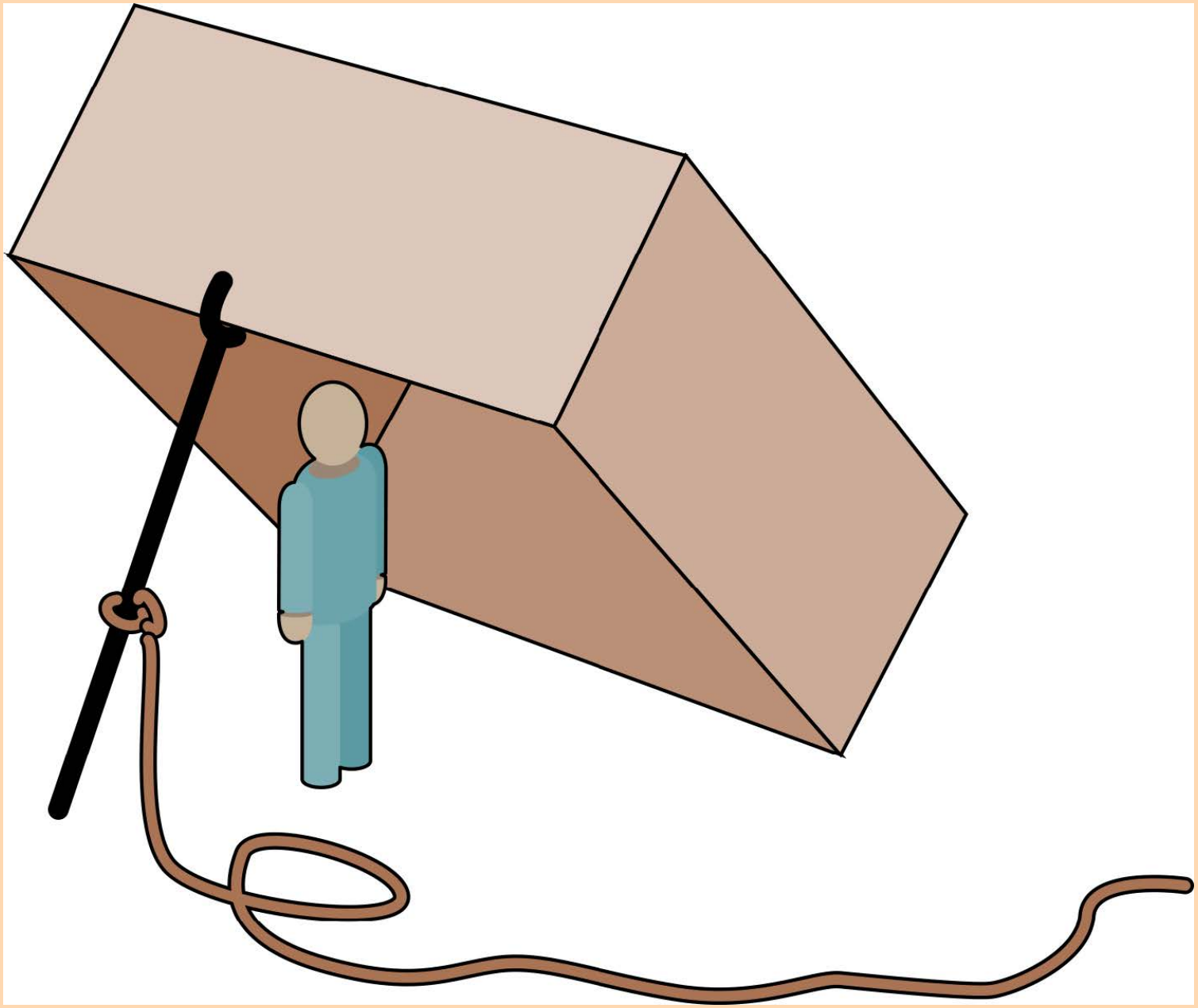
Function Declaration

Arrays as Parameters

Pass by Value

Overview

(Basic) Data Types

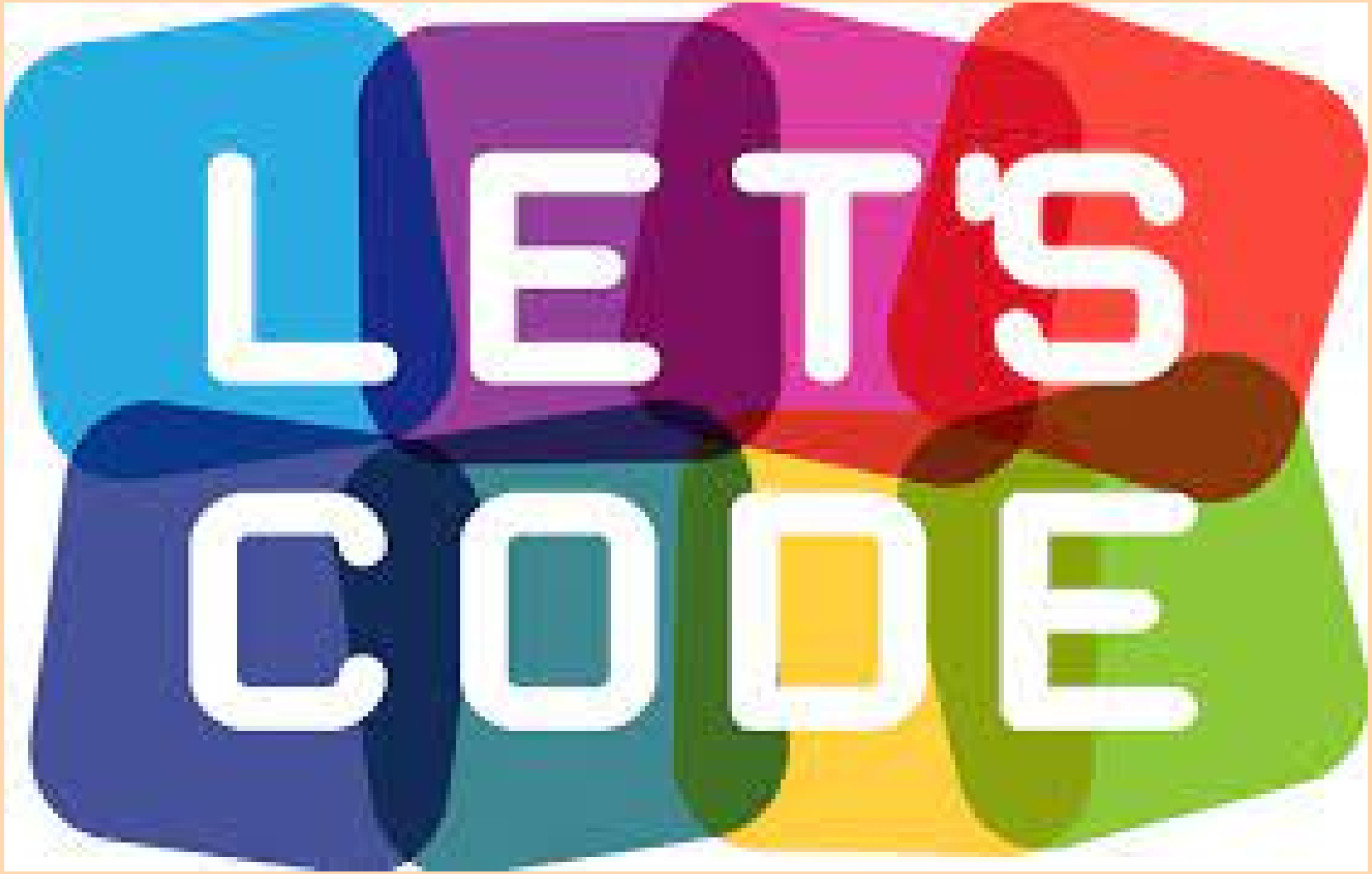


Two Major Reasons

Architectures May Vary

Finite vs. Infinite


THEORY
PRACTICE
THEORY
INTO
PRACTICE


LETS
CODE

Object

3.15

1 **object**

region of data storage in the execution environment, the contents of which can represent valu

2 **Note 1 to entry:** When referenced, an object may be interpreted as having a particular type; see 6.3.2.1.

Object Types **Function Types**

Data Types

The **type** of a variable determines

- the set of **values** it may take on and
- what **operations** can be performed on them.

int **char** **bool** **(_Bool)** **double**

[ ]

Integral Types (size.c)

- (unsigned) short (int)
- unsigned int
- unsigned long (int)
- unsigned long long (int)

When **sizeof** is applied to an operand that has type **char**, **unsigned char**, or **signed char**, (or a qualified version thereof) the result is 1. When applied to an operand that has array type, the result is the total number of bytes in the array.[105] When applied to an operand that has structure or union type, the result is the total number of bytes in such an object, including internal and trailing padding.

The value of the result of both operators is implementation-defined, and its type (an unsigned integer type) is **size_t**, defined in <stddef.h> (and other headers).

Integral Types

- (unsigned) short (int)
- (unsigned) int
- (unsigned) long (int)
- (unsigned) long long (int)

The *precision* of an integer type is the number of bits it uses to represent values, excluding any sign and padding bits. The *width* of an integer type is the same but including any sign bit; thus for unsigned integer types the two values are the same, while for signed integer types the width is one greater than the precision.

Integral Types (int-limits.c)

- (unsigned) short (int)
- (unsigned) int
- (unsigned) long (int)
- (unsigned) long long (int)

Integral Types (exact-width.c)

int8_t int16_t int32_t int64_t

7.20.1.1 Exact-width integer types

The typedef name **int***N***_t** designates a signed integer type with width *N*, no padding bits, and a two's complement representation. Thus, **int8_t** denotes such a signed integer type with a width of exactly 8 bits.

The typedef name **uint***N***_t** designates an unsigned integer type with width *N* and no padding bits. Thus, `uint24_t` denotes such an unsigned integer type with a width of exactly 24 bits.

These types are optional. However, if an implementation provides integer types with widths of 8, 16, 32, or 64 bits, no padding bits, and (for the signed types) that have a two's complement representation, it shall define the corresponding typedef names.

stdint.h

Signed and Unsigned (unsigned.c)

Be careful when MIXING signed and unsigned types.

typedef

```
typedef unsigned __int64 size_t

#define _ _int64 long long

typedef long long time_t
```

char (char.c)

Use `char` only for representing characters.

Do **NOT** assume `signed char` or `unsigned char`.

Overflow

(unsigned-wrap.c　for-unsigned.c　unsigned-wrap-fix.c)

无符号整数运算中没有溢出, 取而代之的是**回绕 (wrap)**现象

Overflow

(signed-overflow-fix.c)

有符号整数运算中发生溢出, 程序的行为是**未定义的**

Implicit Conversion

(implicit-conversion.c)

- 算术表达式、逻辑表达式 (先做整值提升)
- 定义初始化、赋值 (类型转换)
- 函数调用时 (类型转换)
- 函数返回时 (类型转换)

Be careful about narrowing conversions!!!

Implicit Conversion

Integer promotions (integer-promotion.c)

Integer conversion rank (Section 7.4.3)

Usual arithmetic conversions (Section 7.4.1)

Explicit Conversion

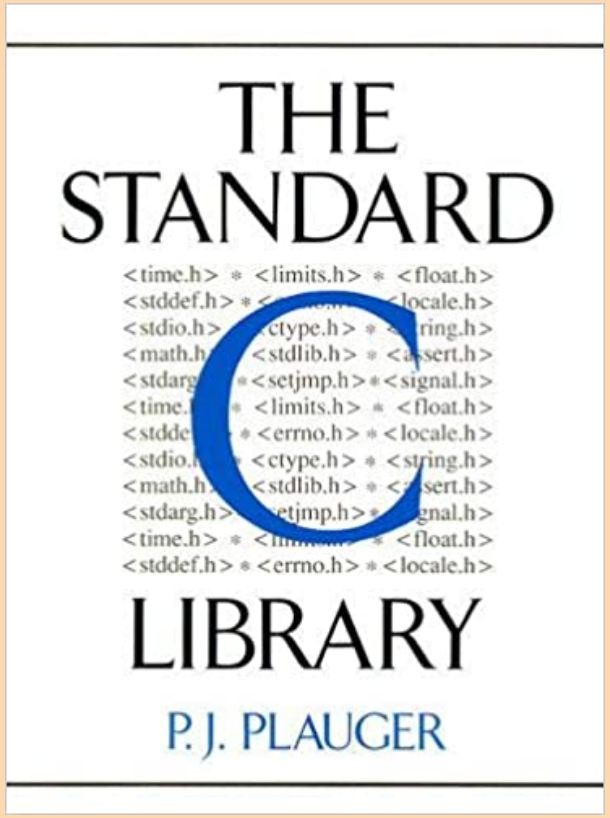
(explicit-conversion.c)

(type) expression

Floating-point Numbers

(float-limits.c)

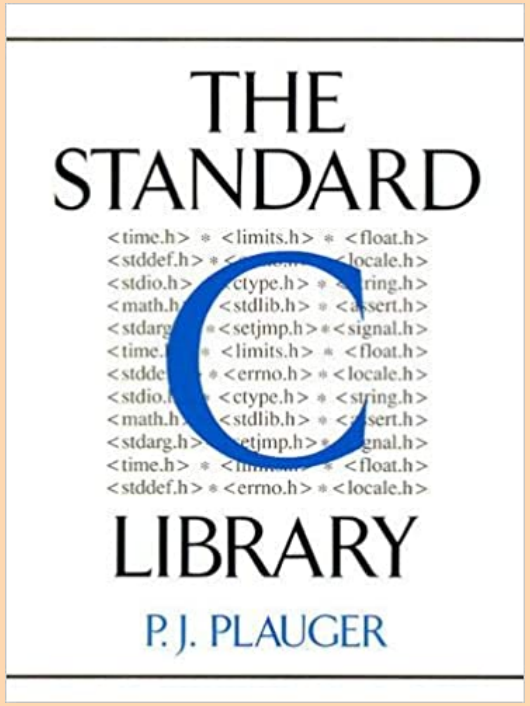
- float (F)
- double
- long double (L)



"Floating-point Arithmetic is Hard."

(Section 23.1 `float.h`)

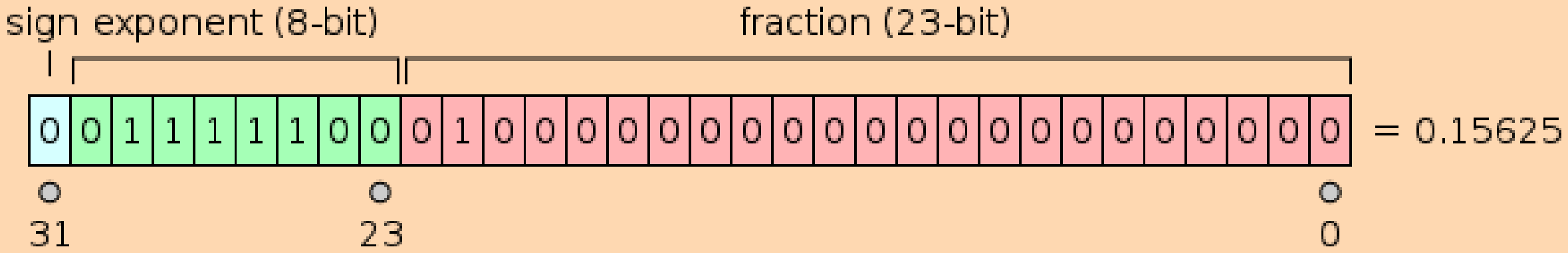
"Many applications don't need floating-point arithmetic at all."



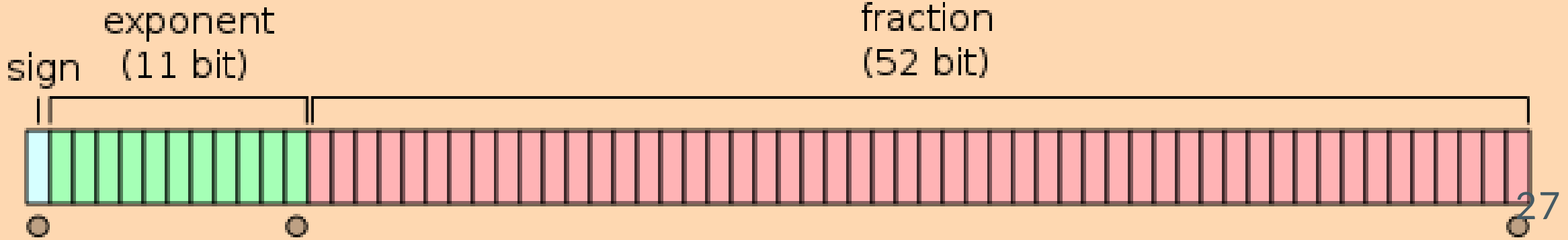
Use `math.h` (Section 23.3) whenever possible.

- 7.12 Mathematics <math.h>
  - 7.12.1 Treatment of error conditions
  - 7.12.2 The FP_CONTRACT pragma
  - 7.12.3 Classification macros
  - 7.12.4 Trigonometric functions
  - 7.12.5 Hyperbolic functions
  - 7.12.6 Exponential and logarithmic functions
  - 7.12.7 Power and absolute-value functions
  - 7.12.8 Error and gamma functions
  - 7.12.9 Nearest integer functions
  - 7.12.10 Remainder functions
  - 7.12.11 Manipulation functions
  - 7.12.12 Maximum, minimum, and positive difference functions
  - 7.12.13 Floating multiply-add
  - 7.12.14 Comparison macros

IEEE 754



$24\ (\approx 6)$ $53\ (\approx 16)$



5.2.4.2.2 Characteristics of floating types <float.h>

The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.[21] The following parameters are used to define the model for each floating-point type:

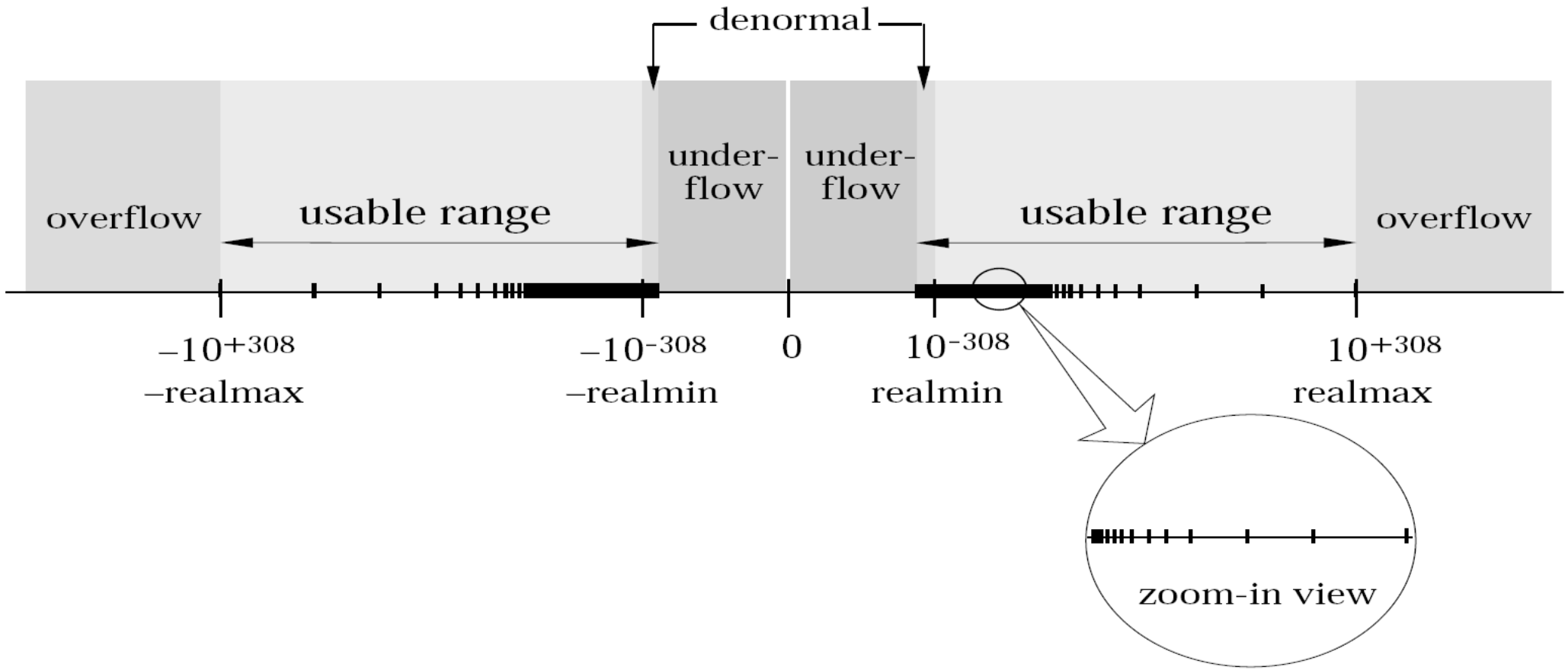
s	sign (± 1)
b	base or radix of exponent representation (an integer > 1)
e	exponent (an integer between a minimum $e_{\min}$ and a maximum $e_{\max}$)
p	precision (the number of base-b digits in the significand)
f_k	nonnegative integers less than b (the significand digits)

A *floating-point number* (x) is defined by the following model:

$$x = sb^e \sum_{k=1}^{p} f_k b^{-k}, \quad e_{\min} \le e \le e_{\max}$$

In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to contain other kinds of floating-point numbers, such as *subnormal floating-point numbers* ($x \neq 0$, $e = e_{\min}$, $f_1 = 0$) and *unnormalized floating-point numbers* ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a floating-point exception; a *signaling NaN* generally raises a floating-point exception when occurring as an arithmetic operand.[22]

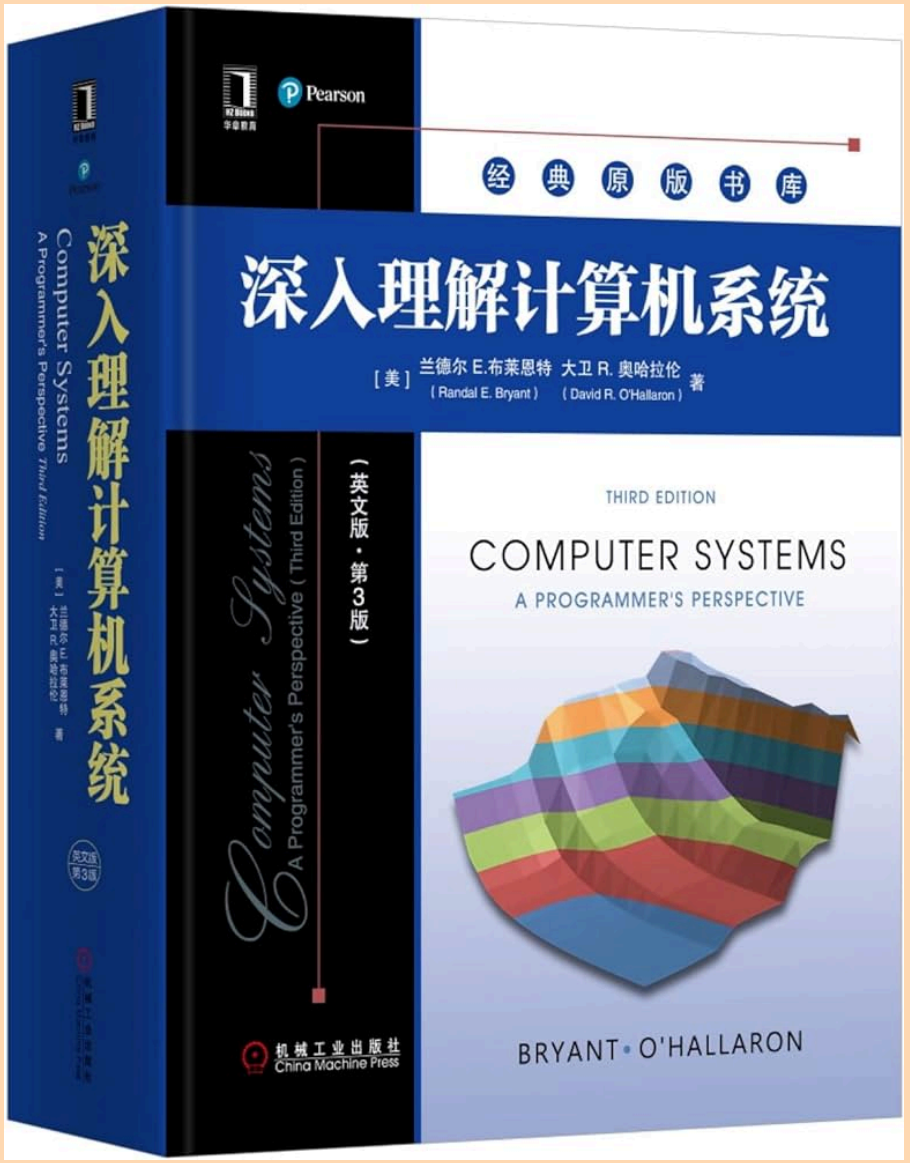
Floating Point Number Line

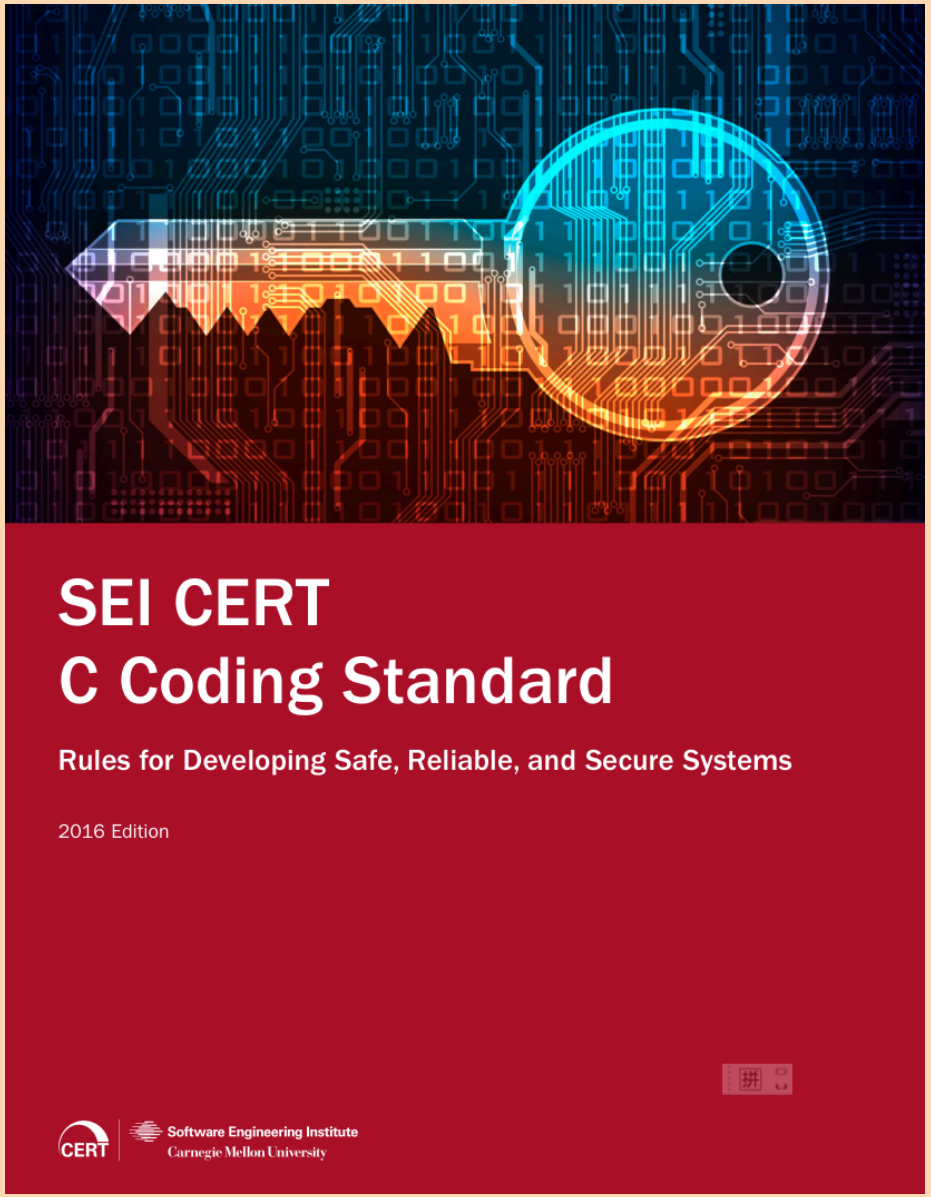


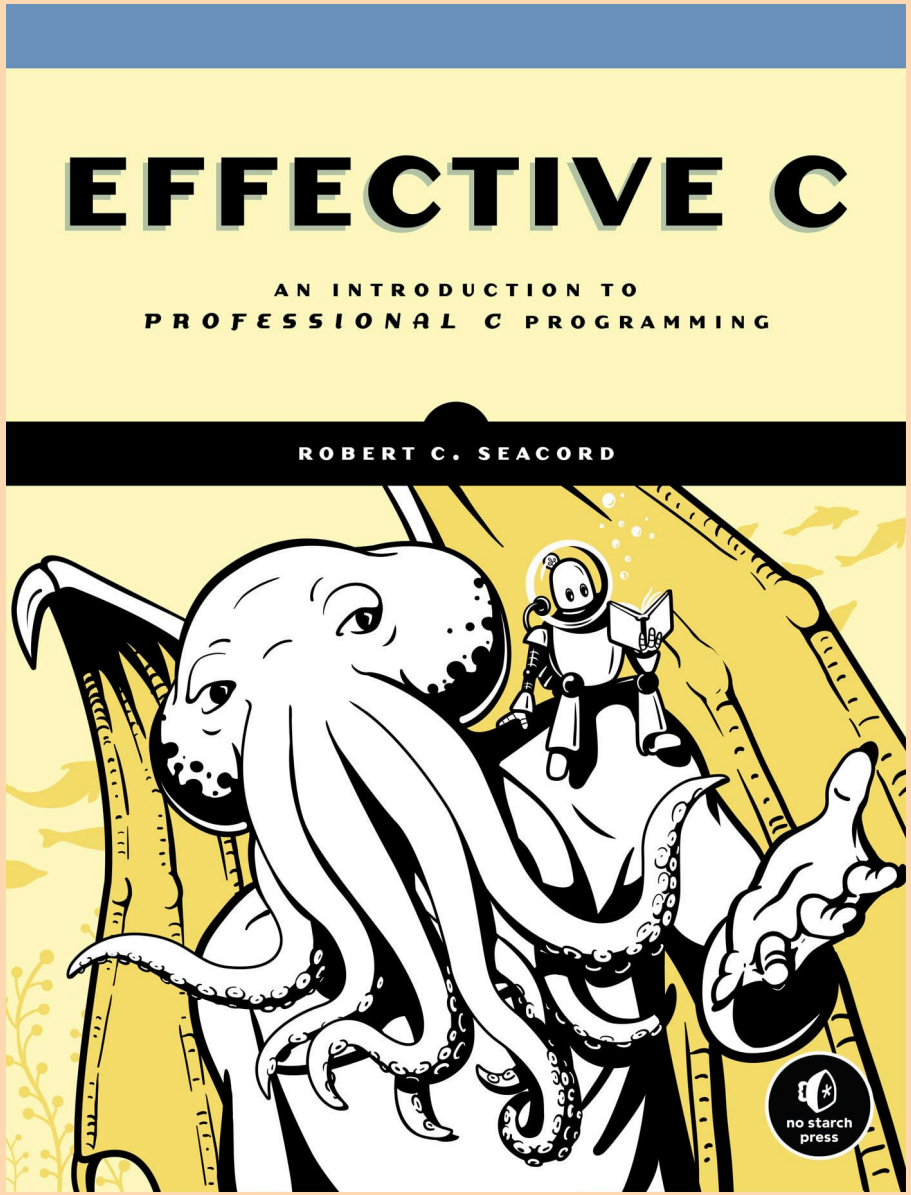
What is a subnormal floating point number? @ stackoverflow

implicit-conversion.c

sum-product.c **loop.c** **compare.c**


Pearson
经典原版书库
深入理解计算机系统
[美] 兰德尔 E.布莱恩特 大卫 R. 奥哈拉伦 著
(Randal E. Bryant) (David R. O'Hallaron)
Computer Systems
A Programmer's Perspective (Third Edition)
(英文版·第3版)
机械工业出版社
China Machine Press
THIRD EDITION
COMPUTER SYSTEMS
A PROGRAMMER'S PERSPECTIVE
BRYANT • O'HALLARON
深入理解计算机系统
Computer Systems
A Programmer's Perspective Third Edition


SEI CERT
C Coding Standard
Rules for Developing Safe, Reliable, and Secure Systems
2016 Edition
CERT
Software Engineering Institute
Carnegie Mellon University


EFFECTIVE C
AN INTRODUCTION TO
PROFESSIONAL C PROGRAMMING
ROBERT C. SEACORD
no starch
press


SEE
YOU
NEXT
WEEK