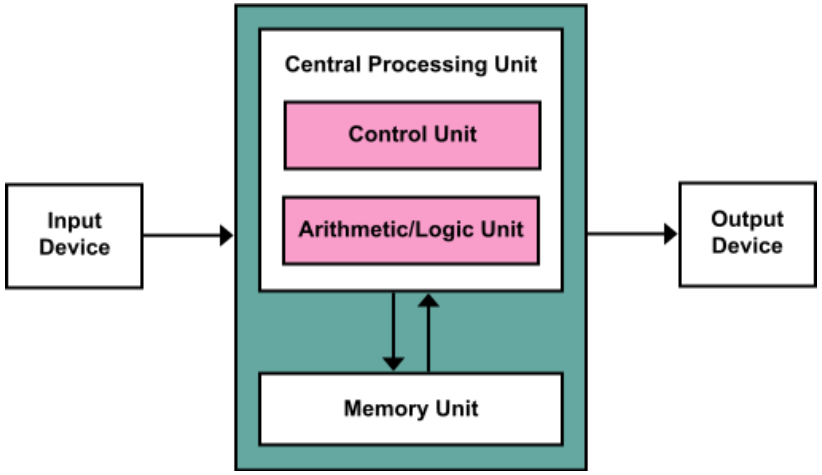
CS 1502H 程序设计思想与方法 (C++)
Principles and Methodology in Programming
Chapter 2. 顺序（1）

陈东尧
上海交通大学
2025年秋季

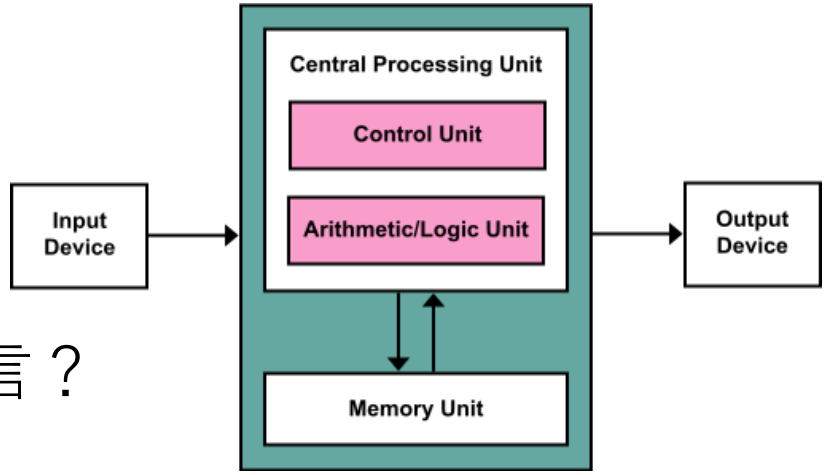
上一讲回顾

- 什么是冯诺依曼架构？



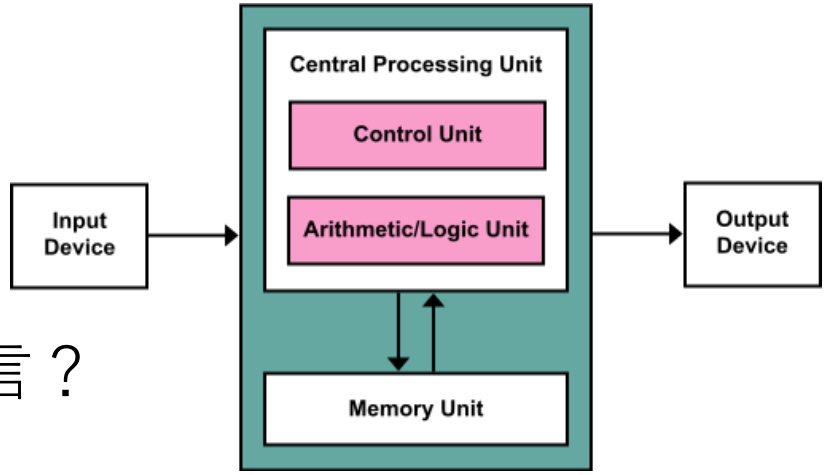
上一讲回顾

- 什么是冯诺依曼架构？五大组成部分
  - 输入
  - 输出
  - 存储器
  - 计算逻辑单元
  - 控制单元
- 按照不同的层次划分，共有哪三种编程语言？



上一讲回顾

- 什么是冯诺依曼架构？五大组成部分
  - 输入
  - 输出
  - 存储器
  - 计算逻辑单元
  - 控制单元
- 按照不同的层次划分，共有哪三种编程语言？
  - 机器语言（machine code）
  - 汇编语言（Assembly language）
  - 高级语言（High-level language）



本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型


> HELLO WORLD!
> _

第一个例子：Hello World

- C++源程序组成：
  - 注释部分
  - 编译预处理
  - 主程序部分

```
// File: hello.cpp
// this program prints the message
// “Hello World!” on the screen

#include <iostream>
using namespace std;

int main()
{
  cout << “Hello World!” << endl;
  return 0;
}
```

程序注释

注释(Comment)

- 注释是写给我们看的（For better readability）
- 解释程序中一些比较难理解的部分。
- C++的注释有两种：
  - 单行注释：从//开始到本行结束，C++风格。
  - 界定符注释：C风格的注释，即从/*与*/之间所有的文字都是注释，可以是连续的几行。

给程序添加注释是良好的程序设计风格

本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型

C++程序的基本组成

- 基本的C++程序结构

```
// File: hello.cpp
// this program prints the message
// “Hello World!” on the screen
#include <iostream>
using namespace std;

int main()
{
  cout << “Hello World!” << endl;
  return 0;
}
```

编译预处理命令

编译预处理

- C++的编译分成两个阶段：预编译和编译。
- 编译预处理命令（预编译命令，precompiled header），即那些以#开头的指令，不需要以分号（；）结尾
- 编译预处理主要有：
  - 库(library)包含：用#include实现，表示程序使用了某个库
  - 宏(macro)定义：用#define实现。用定义的名字来代替所赋予的含义

库包含格式

- 库（Library）是预先做好的一些工具程序。
- 每个库要提供一个接口（头文件）
  - 告诉库的用户如何使用库提供的功能
  - 以便编译器检查程序中对库的调用是否正确。
- 库包含就是把库的头文件插入源文件
- 库包含格式：
  - #include <filename>：包含了一个系统库
  - #include “filename.h”：包含了一个用户自定义的库

宏定义格式

- **宏(macro)**定义：用#define实现。宏包括：
  - 不带参数的宏，通常用来定义符号常量。例如

    #define YEAR 2024

    cout<<“This year is:”<<YEAR; //输出This year is 2024
  - 带参数的宏，用来定义一些较为复杂的操作。例如

    #define SUM(a,b,c) a + b + c

    int result = SUM(1, 2, 3) // 结果为6

本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型

C++程序的基本组成：名字空间

```
// File: hello.cpp
// this program prints the message
// “Hello World!” on the screen

#include <iostream>
using namespace std;
int main()
{
  cout << “Hello World!” << endl;
  return 0;
}
```

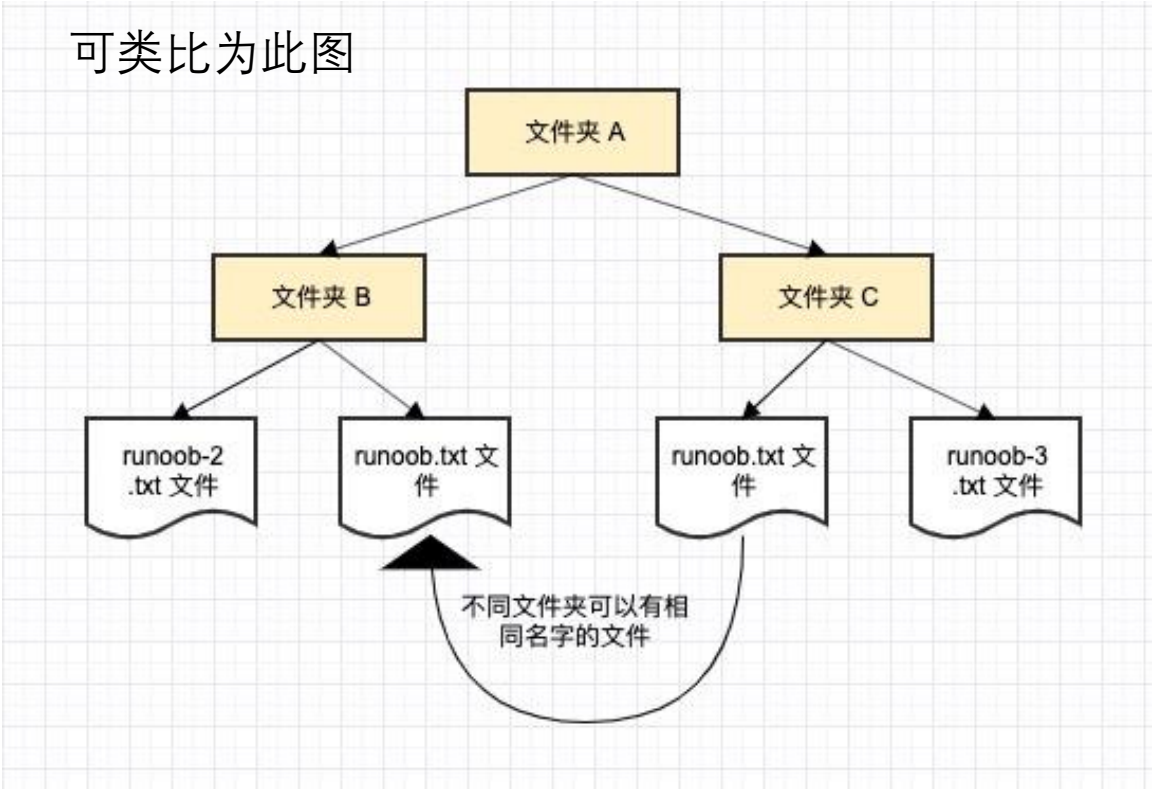
名字空间

名字空间(namespace)

- 名字空间：C++中引入的概念，把一组程序实体组合在一起，构成一个作用域。
- 用于解决重名问题，同一个名字空间中不能有重名。
- std （standard的缩写）命名空间是C++中标准库类型对象的命名空间。
- 不同的使用方式
  - using namespace std;
  - using std::cout;

名字空间(namespace)

- 名字空间：C++中引入的概念，把一组程序实体组合在一起，构成一个作用域。
- 用于解决重名问题，同一个名字空间中不能有重名。
- std （standard的缩写）命名空间是C++中标准库类型对象的命名空间。
- 不同的使用方式
  - using namespace std;
  - using std::cout;



本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型

C++程序的基本组成：主程序（main）

- 主程序由一个或多个函数组成。
- 逻辑上，函数是一组语句，共同执行一个任务。
- 每个程序都必须有一个名为main的函数，它是程序的入口。
- main函数的构成：

```
int main()
{
  cout << “Hello World!” << endl;
  return 0;
}
```

函数头

函数体

本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型

输出流对象（cout）

- “流(stream)”指的是设备之间传递的数据流
- 输出流是传给输出设备的数据流， cout代表标准输出。
- 流插入运算符：<<
- 格式
  - 将hello显示在屏幕上：cout << “hello”；
  - cout << “Hello World!” << endl；
  - endl：换行符, C风格的换行符是’¥n’

回头看第一个例子

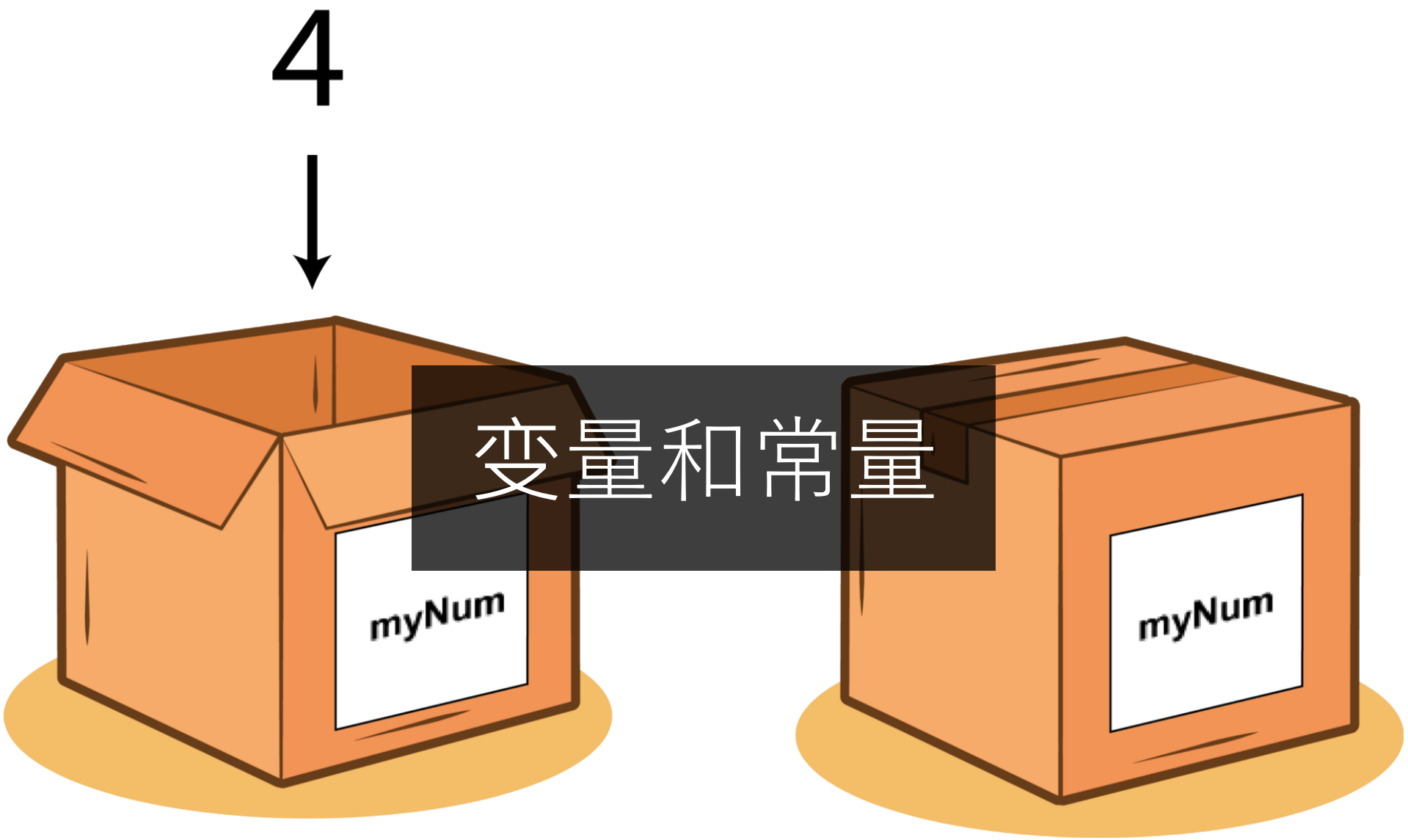
```
/* file: hello.cpp
*  This program prints the message “Hello World!”
on the screen */

#include <iostream>
using namespace std;
int main()
{
  cout << “Hello World!.” << endl;
  return  0;
}
```

本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型

变量和常量



变量（variable）的概念

- 为什么需要变量？
  - 临时存储输入数据、中间结果和最终结果。
- 什么是变量？
  - 提供一个具名的、可供程序操作的存储空间（内存中）：即，Variables are containers for storing data values
- C++中的每个变量都有其数据类型，数据类型决定着:
  - 变量占内存空间的大小和布局方式
  - 存储的值的范围
  - 变量能参与的运算

变量定义

- 变量有**三个重要属性**：数据类型（type）、名称（name）、值（value）
- 变量定义就是告诉编译器变量的名字及该变量中可以存放哪一类数据类型的值。
- C++中变量定义的格式:

  类型名 变量名1，变量名2，…，变量名n；
- *type name = value;*
  - *int num = 15;*
- 在C++中，每个变量在使用前必须被定义，以便编译器检查变量使用的合法性。

变量初始化（initialization）

- 术语
  - 初始化：int num = 3;
  - 定义：int num;
  - 赋值：num = 3;　　//把num的当前值擦除，而以3来替代。

  注意“=”在程序设计中的作用为**赋值**

- 变量初始化方式
  - int num1 = 1;
  - int num2(2);

变量命名

- 开头：名字必须以**字母（大小写不限）或下划线**开头。
- 名字中的其它字符必须是字母、数字或下划线，**不得使用空格或其它特殊符号**
- 名字不可以是系统的关键字，如：int, double, for, return等保留字，它们在C++语言中有特殊用途
- 名字长度：通常在255个字符（character）以下。
- C++语言中，名字中出现的大写和小写字母**被看作是不同的字符**，因此ABC, Abc，abc是三个独立的变量名。
- 名字应有较好的**易读性**：使读者易于明白其存储的值是什么

变量命名

- 驼峰命名法（camel case）：通过大小写来界定

```
variableOne
variableTwo
```

- 蛇形命名法（snake case）：通过下划线来界定

```
variable_one
variable_two
```

- 帕斯卡命名法（Pascal case）：通过大小写来界定（首字母大写）

```
VariableOne
VariableTwo
```

- 匈牙利命名法（Hungarian Notation）：变量的类型或用途来开头，后面接变量的具体含义（通过驼峰命名法来分开）

```
arrDistrubuteGroup  //Array called “Distribute Group”
sUserName           //String called “User Name”
iRandomSeed         //Integer called “Random Seed”
```

扩展阅读：https://curc.readthedocs.io/en/latest/programming/coding-best-practices.html

本讲框架：程序的基本构成

1. 注释
2. 预编译处理命令（库、宏）
3. 名字空间
4. 主程序
5. 输出对象
6. 变量和常量
7. 数据类型：
    1. 计算机的码制
    2. 整形、浮点型、字符型、布尔型、枚举型

数据类型

- 整型（integer）
  - 长整型：无符号unsigned long、有符号long int / long
  - 标准整型：无符号unsigned int、有符号int
  - 短整型：无符号unsigned short int、有符号short int / short
- 实型(real-valued)
  - 单精度：float
  - 双精度：double
  - 长双精度：long double
- 字符型（character）：char
- 布尔型（boolean）：bool

整型(integer)

- 根据占用内存空间的长度，整型数分为：基本整型、长整型和短整型
- C++规定各类型整型数所占的字节数：
  - 短整型：2 Bytes
  - 基本型和长整型：4 B
- 具体占用内存由各个编译器指定，常见的如下所示
  - 基本型 int： 4B -2^{31}～$(2^{31}-1)$
  - 长整型 long int：4B -2^{31}～$(2^{31}-1)$
  - 短整型 short int：2B -2^{15}～$(2^{15}-1)$

无符号（unsigned）整数

- 在某些应用中，不可能出现负数，则整型数中有一半的数值范围是被浪费的。因此在C/C++中可以将所有的数都看成正整数，称为无符号数。
- 无符号数的定义：在各种整数类型前加上关键词unsigned，变成unsigned int, unsigned short, unsigned long。

unsigned int	$0 \sim 2^{32}-1$
unsigned short	0～65535
unsigned long	$0 \sim 2^{32}-1$

整型数的表示 --- 码制

- 如何将符号位数字化:
  - 0表示正数
  - 1表示负数

- 数字的三种编码方式为：
  - 原码(true form)
  - 反码(one’s compliment)
  - 补码(two’s compliment)

原码

- 用符号位和数值表示带符号数。
  - 正数的符号位为0，负数的符号位为1。
  - 数值部分用二进制表示。
  - 如用一个字节表示数值：

$[62]_{原}$=0 0111110

$[-62]_{原}$=1 0111110

- 练习

  $[90]_{原}$ =

  $[-78]_{原}$ =

【扩展阅读】十进制-二进制转换：https://www.electronics-tutorials.ws/binary/bin_2.html#:~:text=An%20easy%20method%20of%20converting,into%20its%20binary%20number%20equivalent.

原码（Sign-magnitude）

- 用符号位和数值表示带符号数。
  - 正数的符号位为0，负数的符号位为1。
  - 数值部分用二进制表示。
  - 如用一个字节表示数值：

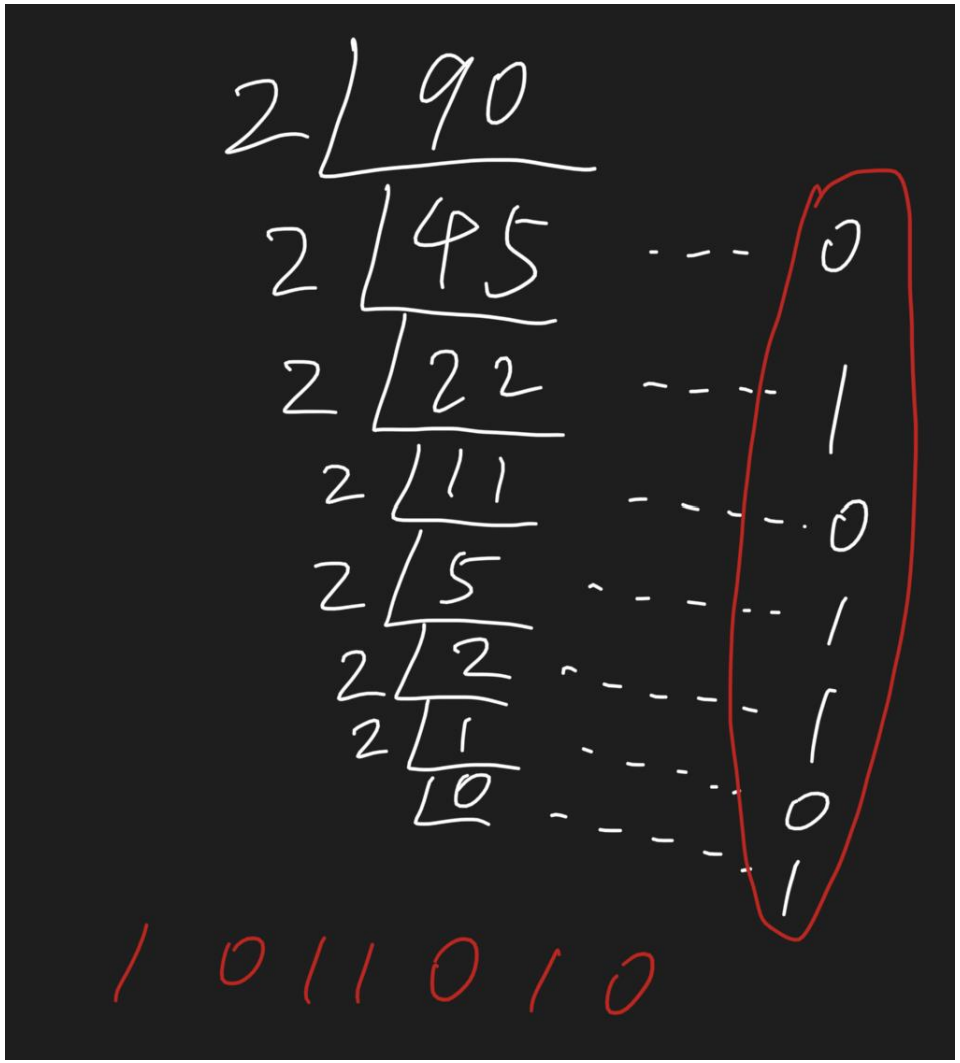
$[62]_{原}$=0 0111110

$[-62]_{原}$=1 0111110

- 练习

  $[90]_{原}$ =

  $[-78]_{原}$ =



【扩展阅读】十进制-二进制转换：https://www.electronics-tutorials.ws/binary/bin_2.html#:~:text=An%20easy%20method%20of%20converting,into%20its%20binary%20number%20equivalent.

原码（Sign-magnitude）

- 用符号位和数值表示带符号数。
  - 正数的符号位为0，负数的符号位为1。
  - 数值部分用二进制表示。
  - 如用一个字节表示数值：

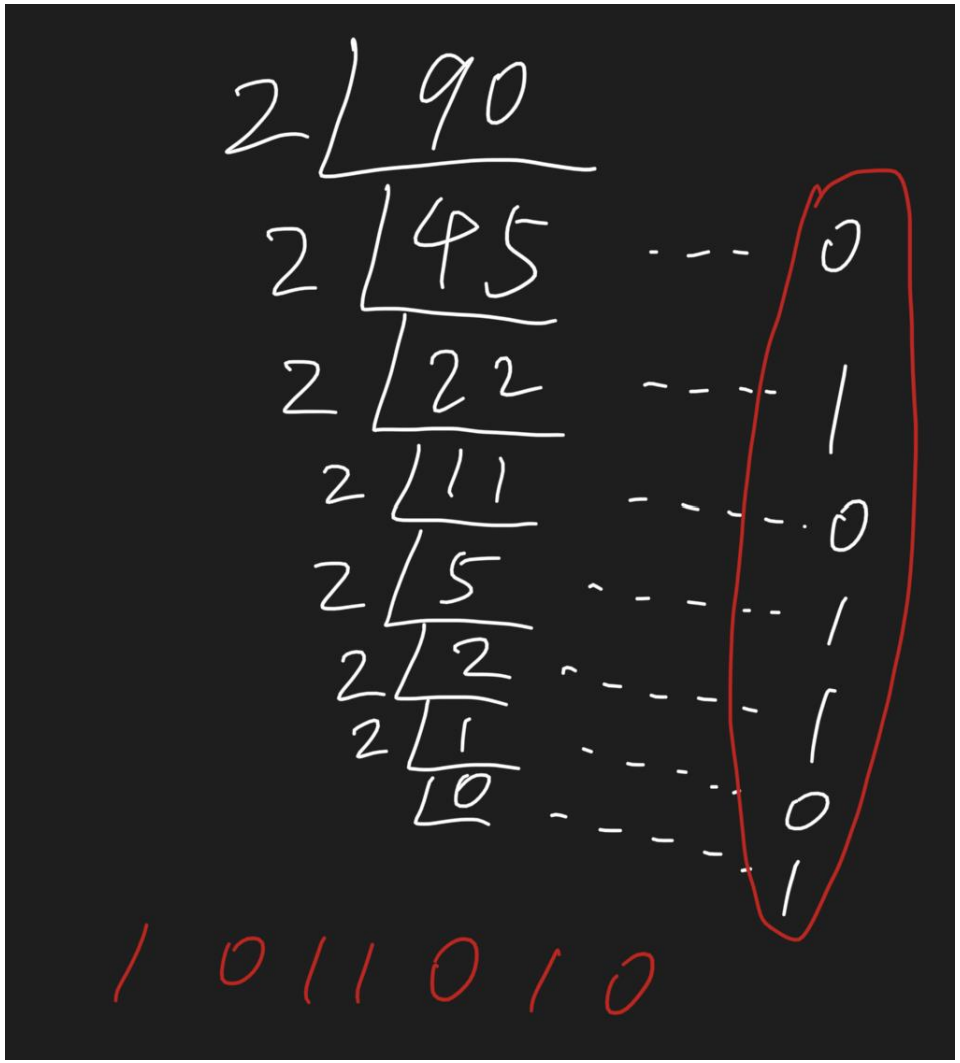
$[62]_{原}$=0 0111110

$[-62]_{原}$=1 0111110

- 练习

  $[90]_{原}$ = 01011010

  $[-78]_{原}$ =11001110



【扩展阅读】十进制-二进制转换：https://www.electronics-tutorials.ws/binary/bin_2.html#:~:text=An%20easy%20method%20of%20converting,into%20its%20binary%20number%20equivalent.

原码的缺点

- 原码不能直接参加运算，可能会出错。例如数学上，1+(-1)=0，而在二进制中00000001+10000001=10000010，换算成十进制为-2。显然出错了。
- 所以原码的符号位不能直接参与运算，必须和其他位分开，这就增加了硬件的开销和复杂性。

反码(Ones' Complement)

- 正数的反码与原码相同，负数的反码为该数的对应正数按位取反。如：

  $[62]_{反}$=0 0111110

  $[-62]_{反}$=1 1000001　　//解释：$[-62]_{原}$=1 0111110；之后对“0111110”按位取反

- 练习

  $[90]_{反}$=

  $[-78]_{反}$=

反码(Ones' Complement)

- 正数的反码与原码相同，负数的反码为该数的对应正数按位取反。如：

  $[62]_{反}$=0 0111110

  $[-62]_{反}$=1 1000001 // $[-62]_{原}$=1 0111110

- 练习

  $[90]_{反}$= 01011010

  $[-78]_{反}$=10110001

补码(Two's complement)

- 正数的补码与原码相同，负数的补码为该数的对应正数按位取反再加1。如：

  $[62]_{补}$=0 0111110

  $[-62]_{补}$=1 1000010　　// 解释：$[-62]_{反}$=1 1000001；之后对“1000001”加一

- 练习

  $[90]_{补}$=

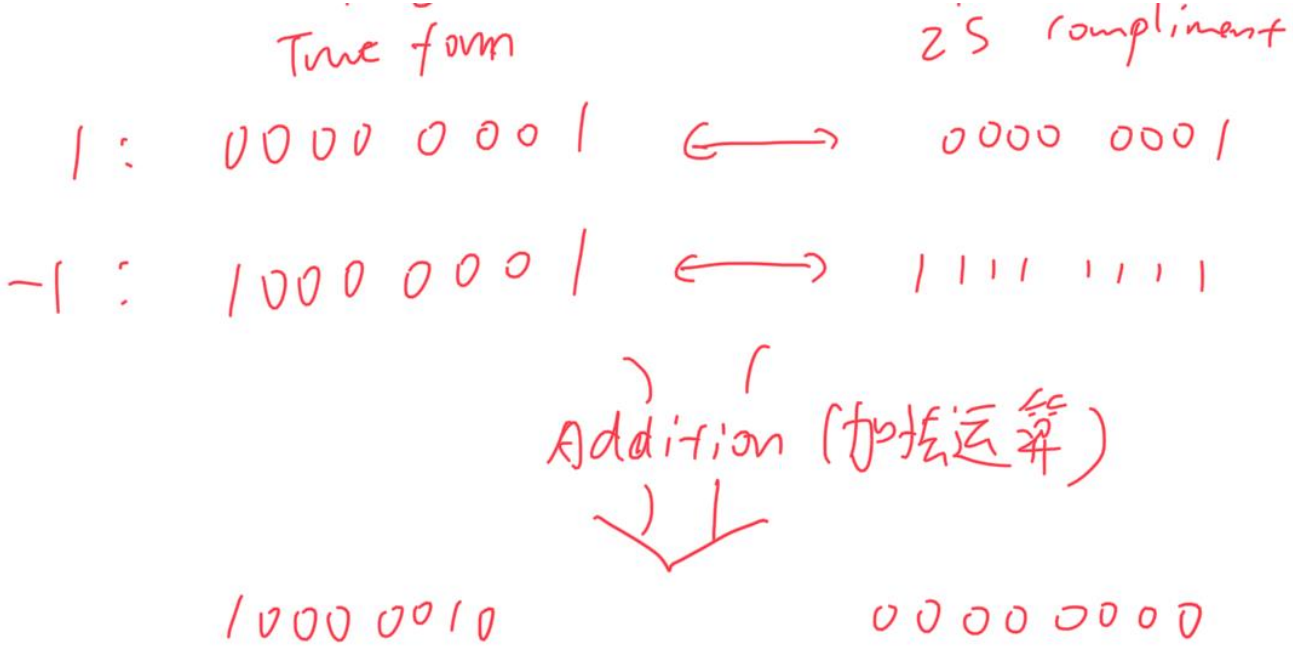
  $[-78]_{补}$=

- 计算机中为何要使用反码和补码

https://www.cnblogs.com/zhangziqiu/archive/2011/03/30/computercode.html

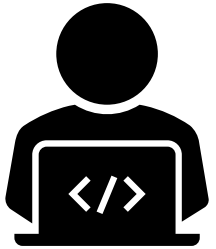
补码的优点

- 补码系统的最大优点是可以在加法或减法处理中，不需因为数字的正负而使用不同的计算方式。
  - 加法：只要一种加法电路就可以处理各种有号数加法
  - 减法：可以用一个数加上另一个数的补码来表示。即： 不管是加法还是减法都是加法运算。这在电路设计上相当方便。



整数的内部表示

- 整数在计算机内部通常用补码表示。
- 整数运算时要注意数据的表示范围。
  - 如果整数用两个字节表示，正整数 32767 加 1 的结果为 -32768。这称为整数运算的溢出（overflow）。
  - $[32767]_{补}$=0111 1111 1111 1111
- 系统不检查这样的错误，程序员必须自己保证程序中不出现这样的错误。



代码演示

chapter2-add-overflow.cpp

具体分析

1. $[32767]_{补}$=0111 1111 1111 1111
2. 上述加1，你会得到？
   - 1000 0000 0000 0000
3. 在有符号数的情况下，最左边的一位代表符号，所以电脑会认为这是“-0”
4. 定义其为当前16位系统中的最小负数-32768

1000 0000 0000 0000 ↔ -32768
1000 0000 0000 0001 ↔ -32767
… …
1111 1111 1111 1111 ↔ -1
0000 0000 0000 0000 ↔ 0
… …
0111 1111 1111 1111 ↔ 32767

可实现：
数据范围首尾相连

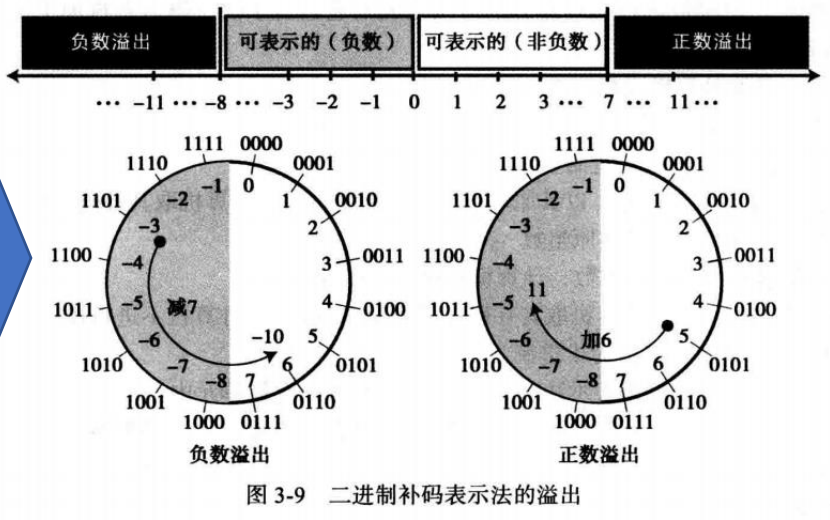


图 3-9　二进制补码表示法的溢出

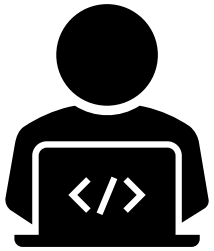
了解溢出后，结合int的定义，我们来分析如下一个例子

```
#include <iostream>
using namespace std;

int main()
{
        // Value of p 10^5
        int p = 100000;

        // Value of q 10^5
        int q = 100000;

        int result = p * q;
        cout << result << endl;
        return 0;
}
```



代码演示

chapter2-int_overflow.cpp

1410065408

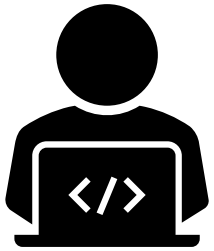
在32位机上，最大的signed integer是2,147,483,647 (2^31 - 1)
在64位机上，是不会出错的

```
#include <iostream>
using namespace std;

int main()
{
        // Value of p 10^5
        int p = 100000;

        // Value of q 10^5
        int q = 100000;

        long  long  int  result = p * q;
        cout << result << endl;
        return 0;
}
```



代码演示

chapter2-int_overflow.cpp

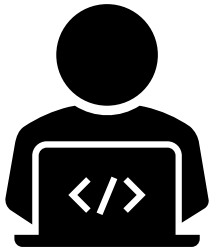
1410065408

```
#include <iostream>
using namespace std;

int main()
{
        // Value of p 10^5
        int p = 100000;

        // Value of q 10^5
        int q = 100000;

        long long int result = (long long int)p * q;
        cout << result << endl;
        return 0;
}
```



代码演示

chapter2-int_overflow.cpp

10000000000

先显式类型转换p为long long int

```
#include <iostream>
using namespace std;

int main()
{
        // Value of p 10^5
        int p = 100000;

        // Value of q 10^5
        int q = 100000;

        long long int result = (long long int)p * q;
        cout << result << endl;
        return 0;
}
```

C++的设计包括了隐式类型转换（implicit type conversion），其中的原则之一就是会自动向更大的数据类型转换。
这里正确的写法是：

```
long long int result = (long long int)p * q;
```

那么两个乘数类型不同，就会自动转换q的类型。
注意如果是这样：

```
long long int result = p * q;
```

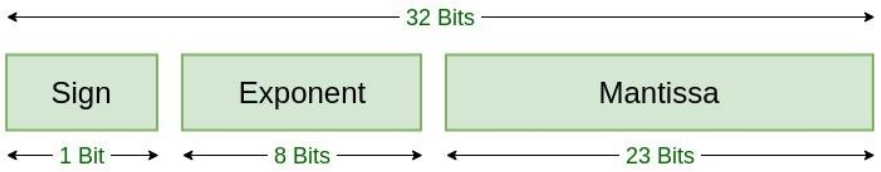
赋值符号同侧的p和q，符号是一样的，那就不会触发自动类型转换，两个都是int，那么乘法的结果就会溢出，这个溢出后的值赋给result，就会得出错误结果。

10000000000

先显式类型转换p为long long int

数据类型---实型（浮点型）

- 实型数表示成$a*2^b$的形式， a为尾数 (mantissa)， b为指数 (exponent)
- 这种形式又称为浮点表示法， 浮点数无法精确表示。
- 一个浮点数分成尾数和指数两部分。
  - 尾数表示数的有效数值
  - 指数表示小数点在该数中的位数
- 实型分为：
  - 单精度(single-precision) float、双精度double、长双精度：long double



Single Precision
IEEE 754 Floating-Point Standard

数据类型-字符型

- 字符类型（char)：存放一个字母或符号， 占一个字节， 存放的是字符的机内码。
- 字符的机内表示：用字符编码表示。常用的有ASCII, BCD, EBCDIC等。PC机中都用ASCII.
- ASCII码的重要特性
  - 数字‘0’到‘9’是顺序存放的
  - 字母被分成二段：大写的和小写的。大写字母是连续的， 小写字母也是连续的
- 字符型变量可执行的运算
  - 算术、比较

ASCII 码表

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

字符运算

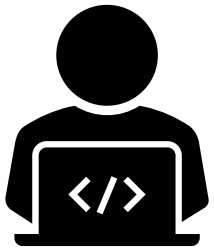
- 赋值

  char c1, c2;

  c1=’a’; c2=’b’;

  c1=97; c2= 98;

  //上述两种方法等效



代码演示

chapter2-char-operation.cpp

```
c
2
43
+
+
```

```
char c1, c2;
c1='a'; c2='b'; // c1=97; c2= 98;

/* c1 = ’a’; c1 = c1 + 2; c1 的值应为？*/
c1 = c1 + 2;
cout <<c1<<endl;

/* c中存放的是小写字母，则c - ‘a’ + 1表示什么？ */
cout<<'b' - 'a' + 1<<endl;

/* 如c中存放的是数字（‘0’ ~ ‘9’），则c - ‘0’表示什么？*/
cout<<'b' - '7'<<endl;
cout<<char('b' - '7')<<endl;
char c3 = 'b'-'7';
cout<<c3<<endl;
```

数据类型-布尔（boolean）型

- 布尔型：标准C中没有布尔型数据，这是C++中新增的数据类型。占一个字节。它的值为：true，false
- 布尔型数据的内部表示：
  - true为1
  - false为0
- 布尔类型不能直接输入输出
- 布尔型数据可以进行算术、比较、逻辑运算

数据类型-枚举类型

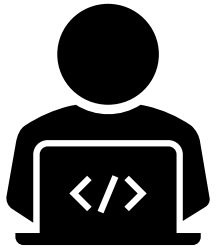
- 有时在设计程序时会用到一些特殊的对象，这些对象的取值范围是有限可数的。
  - 星期中的每一天
- 解决方法
  - 采用编码：假设0表示星期日，1表示星期一，…，6表示星期六。然后用一个整型变量如weekday表示这个对象。缺点是可读性差、健壮性弱
  - 符号常量：用#define功能将这些数字定义为符号常量，但可扩展性差
  - 定义一个新类型：枚举类型

定义枚举类型

- 格式：enum 枚举类型名 {元素表}；
- 定义一个表示一周中每天的名字的枚举类型：

  enum weekdayT { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
- 定义一个枚举类型的变量：

  weekdayT weekday；
- 枚举类型变量的操作：
  - 赋值： weekday = Friday；
  - 比较：Monday < Friday 比较这两个值的内部表示
- 枚举类型不能直接输入输出



代码演示

chapter2-enum-operation.cpp

枚举类型的内部表示

- 在内部，枚举类型采用编码表示。当定义

  enum weekdayT { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}; 时，默认用0代表Sunday，1代表Monday，…，6表示Saturday

- C++语言的编译器也允许明确指出枚举类型的元素的内部表示。例如，希望从1而不是0开始编号，可以这样定义

  enum weekdayT { Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

- 也可以从中间某一个开始重新指定，如

  enum weekdayT { Sunday, Monday, Tuesday=5, Wednesday, Thursday, Friday, Saturday};

枚举类型

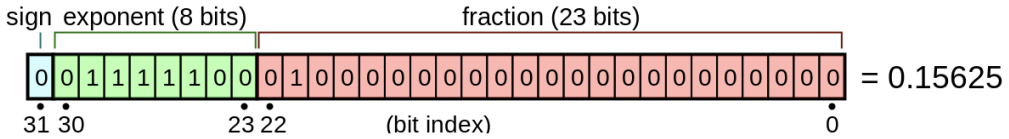
- 自定义枚举类型的好处：
  - 编译器自动编码，解放程序员工作量
  - 变量可以用有意义的类型名，提高程序的可读性
  - 为编译器提供信息（例如取值范围或允许的操作），程序的调试更容易
- 注意：虽然枚举型变量中存储的是一个整数，但不能将一个整数赋给一个自定义的枚举型变量，否则会出现编译错误。
- 布尔型就是C++事先定义好的一个枚举类型。

数据类型小结

- 理解数据类型的作用，主要用于规定：
  - 占用空间
  - 取值范围
  - 支持运算
- 可以直接用cin和cout对象输入输出的类型
  - 整型
  - 实型
  - 字符型
- 不能直接用cin和cout对象输入输出的类型
  - 布尔型：true显示1，false显示0
  - 枚举类型：显示编码值

扩展阅读

- Float类型的数据范围：
  - https://zhuanlan.zhihu.com/p/84453627
  - https://en.wikipedia.org/wiki/Single-precision_floating-point_format



The real value assumed by a given 32-bit *binary32* data with a given *sign*, biased exponent *e* (the 8-bit unsigned integer), and a *23-bit fraction* is

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2,$$

which yields

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i}2^{-i}\right).$$

In this example:

- $\text{sign} = b_{31} = 0,$
- $(-1)^{\text{sign}} = (-1)^0 = +1 \in \{-1, +1\},$
- $E = b_{30}b_{29}\ldots b_{23} = \sum_{i=0}^{7} b_{23+i}2^{+i} = 124 \in \{1, \ldots, (2^8 - 1) - 1\} = \{1, \ldots, 254\},$
- $2^{(E-127)} = 2^{124-127} = 2^{-3} \in \{2^{-126}, \ldots, 2^{127}\},$
- $1.b_{22}b_{21}\ldots b_0 = 1 + \sum_{i=1}^{23} b_{23-i}2^{-i} = 1 + 1 \cdot 2^{-2} = 1.25 \in \{1, 1 + 2^{-23}, \ldots, 2 - 2^{-23}\} \subset [1; 2 - 2^{-23}] \subset [1; 2).$

thus:

- $\text{value} = (+1) \times 2^{-3} \times 1.25 = +0.15625.$