地球与空间科学系
DEPARTMENT OF EARTH AND SPACE SCIENCES

C程序设计基础

Introduction to C programming

Lecture 4: Data

2023-3-7

张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

Review on L3

Data types and variables

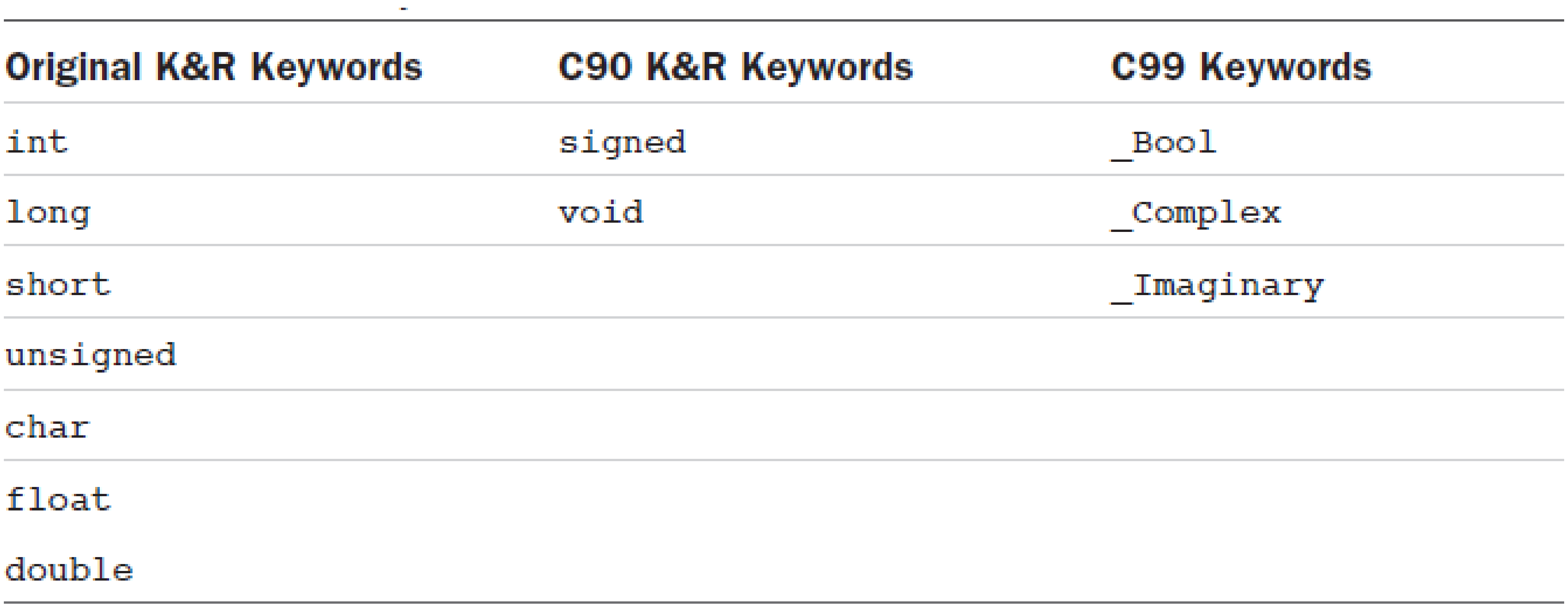
Operations and expressions

Formatted Input/Output

Data types and variables

```
int num = 5;    //整数
float x = 2.14; //浮点数、实数
char  c = ‘T’;  //字符串
char  s[10] = “Hello”;  //字符串
```

Data types



Original K&R Keywords	C90 K&R Keywords	C99 Keywords
int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		

Variables

Variables are placeholders for values, each variable has a **type** defined. The type determines how it is stored and how much space (bit) it needs in machine.

```
type variable;          /*declare*/
type variable = value; /*initialize*/
```

```
int num; //声明
num = 5; //赋值
printf("num = %d", num);
```

```
int num = 5; //声明+赋值
printf("num = %d", num);
```

Variables

A variable name can ONLY be defined once, but its value can be set multiple times!

✓

```
int num = 5; //声明+赋值
printf(“num = %d”, num);

num = 10; //重新赋值
printf(“num = %d”, num);
```

✗

```
int num = 5; //声明+赋值
printf(“num = %d”, num);

int num = 10; //重定义
printf(“num = %d”, num);
```

Rules to name variables?

- Lowercase/uppercase letters, digits and the underscore(_)
- The first character must not be a number.
- Length limit (<=31)
- Case-sensitive

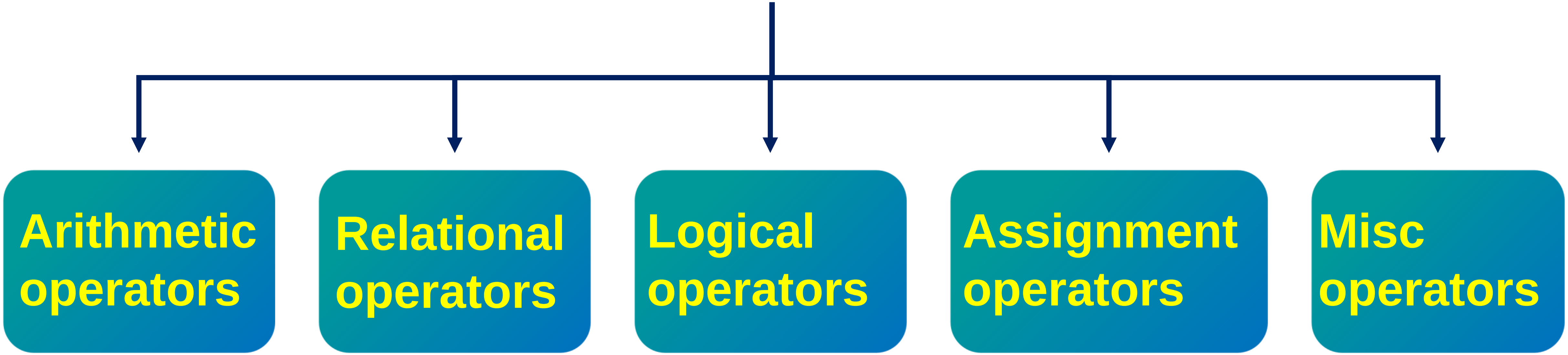
Valid Names	**Invalid Names**
wiggles	$Z]**
cat2	2cat
Hot_Tub	Hot-Tub
TaxRate	tax rate
_kcab	don’t

Rules to name variables?

- Keywords are reserved by C, cannot be used
- Variable names must be **unique**
- Variable names should be **readable**, **meaningful** and **consistent**
    - UpperCamelCase - BodyMassIndex
    - lowerCamelCase - bodyMassIndex
    - snake_case – body_mass_index

Operators

Operator is a symbol that tells compiler to perform specific mathematical or logical operations.



Arithmetic operators

Relational operators

Logical operators

Assignment operators

Misc operators

+, -, *, /, %

>,>=,!=,<

&&,||,!

=,+=, -=, *=

sizeof(),&,?

Arithmetic Operators

More examples on different data types

Operators	int A = 10, B = 20;	float A = 13, B = 6;
+	A + B = 30	A + B = 19
-	A – B = -10	A – B = 7
*	A * B = 200	A * B = 78
/	A / B = 0	A / B = 2.166667
%	A % B = 10	A % B = ? (wrong!)
++	A++ = 11	A++ = 14
--	A-- = 9	A-- = 12

Arithmetic Operators

Post-increment A++
use A; then increment it

```
int A = 20;
int B = A++;

printf(“A=%d\n”,A);
printf(“B=%d\n”,B);
```

```
A=21
B=20
```

Pre-increment ++A
increment it; then use it

```
int A = 20;
int B = ++A;

printf(“A=%d\n”,A);
printf(“B=%d\n”,B);
```

```
A=21
B=21
```

Arithmetic Operators

++

```
#include<stdio.h>
int main(void){
  float shoe;

  shoe = 17.0;
  while (++shoe < 18.5)
  {
    printf("The first Size is: %f\n", shoe);
  }
  shoe = 17.0;
  while (shoe++ < 18.5)
  {
    printf("The second Size is: %f\n", shoe);
  }

  return 0;
}
```

```
The first Size is:18.000000
```

```
The second Size is:18.000000
The second Size is:19.000000
```

Arithmetic Operators

++/ - -

Don’t use increment/decrement on a variable that

- is part of more than one argument of a function;
- appears more than once in an expression.

```
while (num < 21)
   {
      printf("%d %d\n", num, num*num++);
   }
```

```
ans = num/2 + 5*(1 + num++);
```

```
n = 3;
y = n++ + n++;
```

✖

- ➢ 不要写出别人看不懂的也不知道系统会怎样执行程序

Operator precedence(优先级)

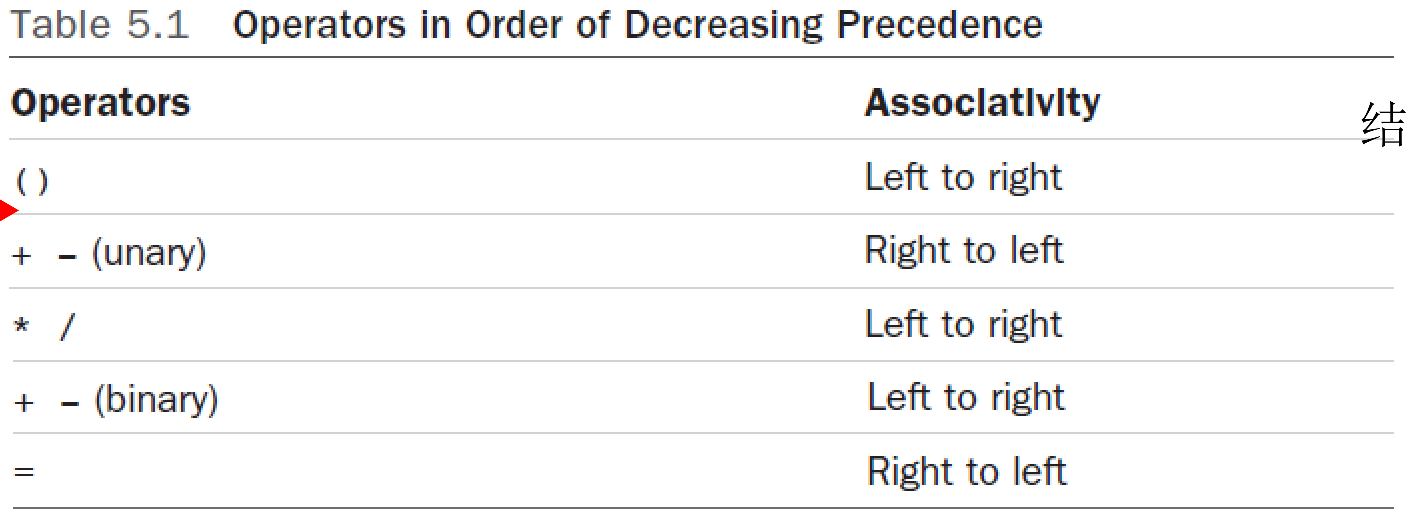


Table 5.1 Operators in Order of Decreasing Precedence

Operators	Associatlvlty
( )	Left to right
+ – (unary)	Right to left
* /	Left to right
+ – (binary)	Left to right
=	Right to left

结合性/结合律

++/ - -

unary(一元)
binary(二元)

```
float a;
a=+4.3/-5.0+-6.6;
```

It’s allowed, but not recommended.

```
a = b += c++ -d + --e / -f;
```

```
(a = (b += (((c++) –d) + ((--e) / (-f)))));
```

Relational Operators

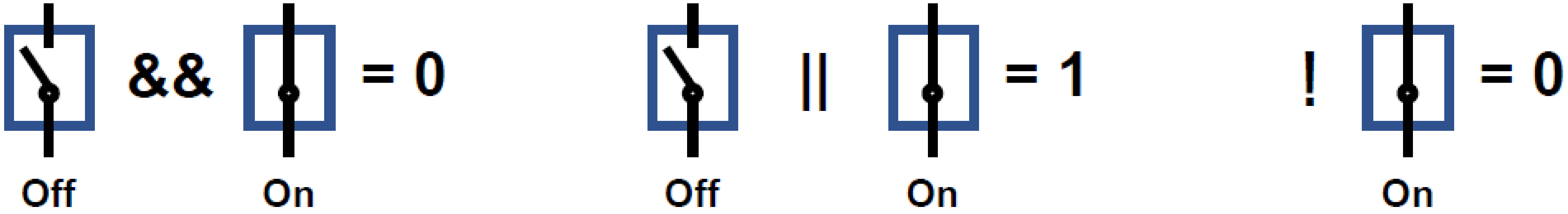
More examples on different data types

Operators	**float A = 3.5, B = 3.5;**	**char A = ‘A’, B = ‘B’;**
==	A==B = 1 (true)	A==B = 0 (false)
!=	A != B = 0 (false)	A != B = 1 (true)
>	A > B = 0 (false)	A > B = 0 (false)
<	A < B = 0 (false)	A < B = 1 (true)
>=	A >= B = 1 (true)	A >= B = 0 (false)
<=	A <= B = 1 (true)	A <= B = 0 (true)

Logical Operators

Define two variables: `int A = 0, B = 1;`

Operators	Description	Example
&&	**AND operator, if both are on, then on**	A&&B = 0 (false)
||	**OR operator, if any is on, then on**	A ||B = 1 (true)
!	**NOT operator, turn opposite**	!A = 1 (true) !B = 0 (false)



Assignment operators

Define two variables: `int A = 5, B = 3;`

Operators	Description	Example
=	Simple assignment	B = B + A = 8
+=	Add and assign	B += A is B = B + A = 8
-=	Subtract and assign	B -= A is B = B – A = -2
***=**	Multiply and assign	B *= A is B = B * A = 15
/=	Divide and assign	B /= A is B = B / A = 0
%=	Modulus and assign	B %= A is B = B % A = 3

Assignment operators

More example:

```
int a=12,b;
b=a+=a-=a*a;
```

- S1: a-=a*a;

```
a = a-a*a;
a = 12-144
a = -132
```

- S2: a+=-132;

```
a =a +(-132)
b = -264
```

Miscellaneous operators

Define a variable: int A = 10; double B = -1.5;

Operator	Description	Example
sizeof()	Return the size of variable (number of bytes)	sizeof(A) = 4 sizeof(B) = 8
&	Return the address of variable	&A = -2072708912 &B = -1602356112
?	Conditional expression	int flag = A>0 ? 1:0;
*****	Pointer points to a variable	*A, *B

Few other important operators supported by C Language.

- **Data types and variables**
- **Operations and expressions**
- **Formatted Input/Output**

getchar() and putchar()

getchar() reads the next available single character and returns an integer representing the character in ASCII table.

putchar() puts the passed character on the screen

```
int c = getchar();
putchar(c);
```

It reads and puts a single character!!!

gets() and puts()

gets() reads a string (a group of characters) from user and puts it into a buffer.
puts() shows the string on the screen

```
gets(char *s);
puts(char *s);
```

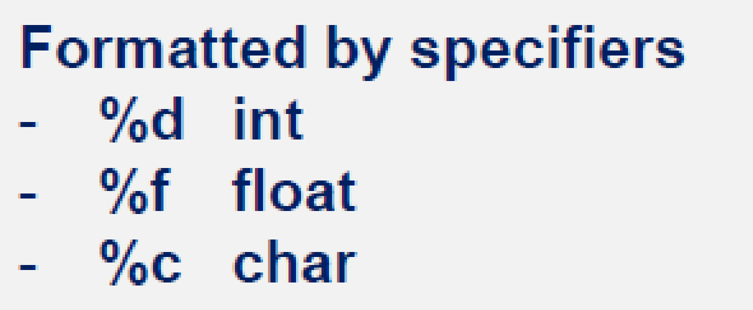
It reads and puts a group of characters!!!

scanf() and printf()

scanf() reads the user input stream and scans it according to the provided format.

printf() writes to the output scream according to the format

```
scanf([formatted text], [arguments]);
printf([formatted text], [arguments]);
```



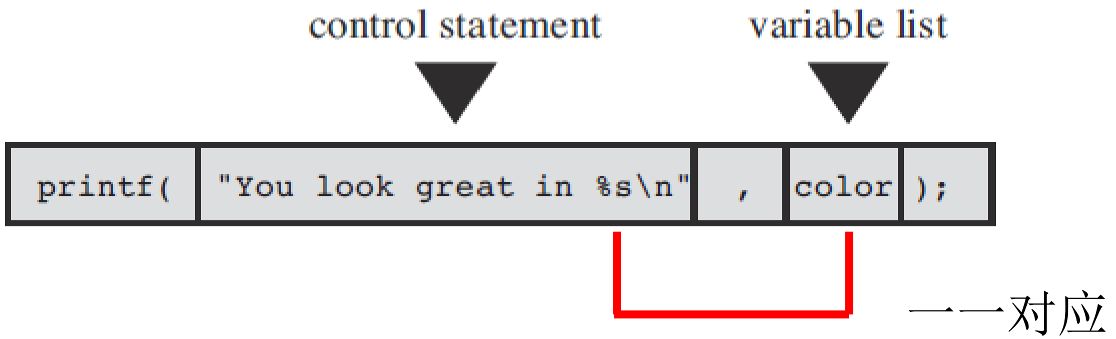
Formatted by specifiers

- %d int
- %f float
- %c char

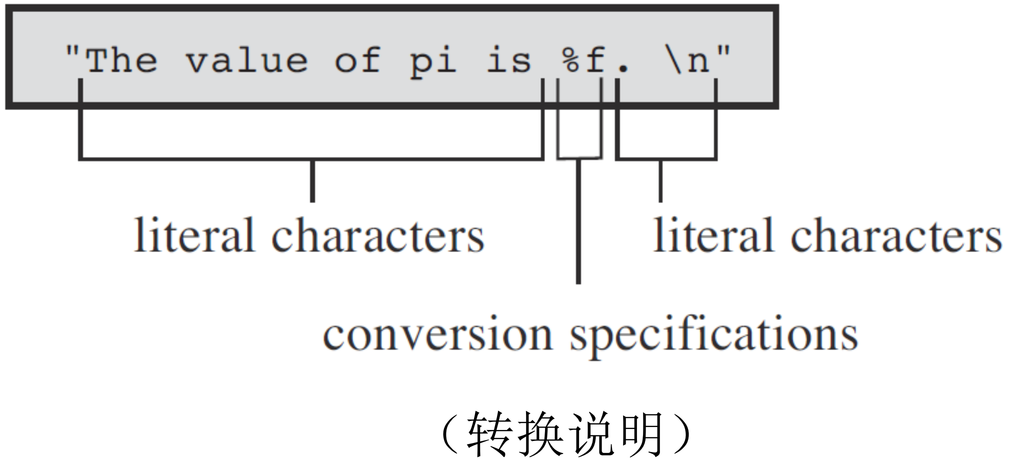
f means formatted!!!

printf()

```
printf([formatted text], [arguments]);
```



```
printf( "You look great in %s\n" , color );
```



```
"The value of pi is %f. \n"
```

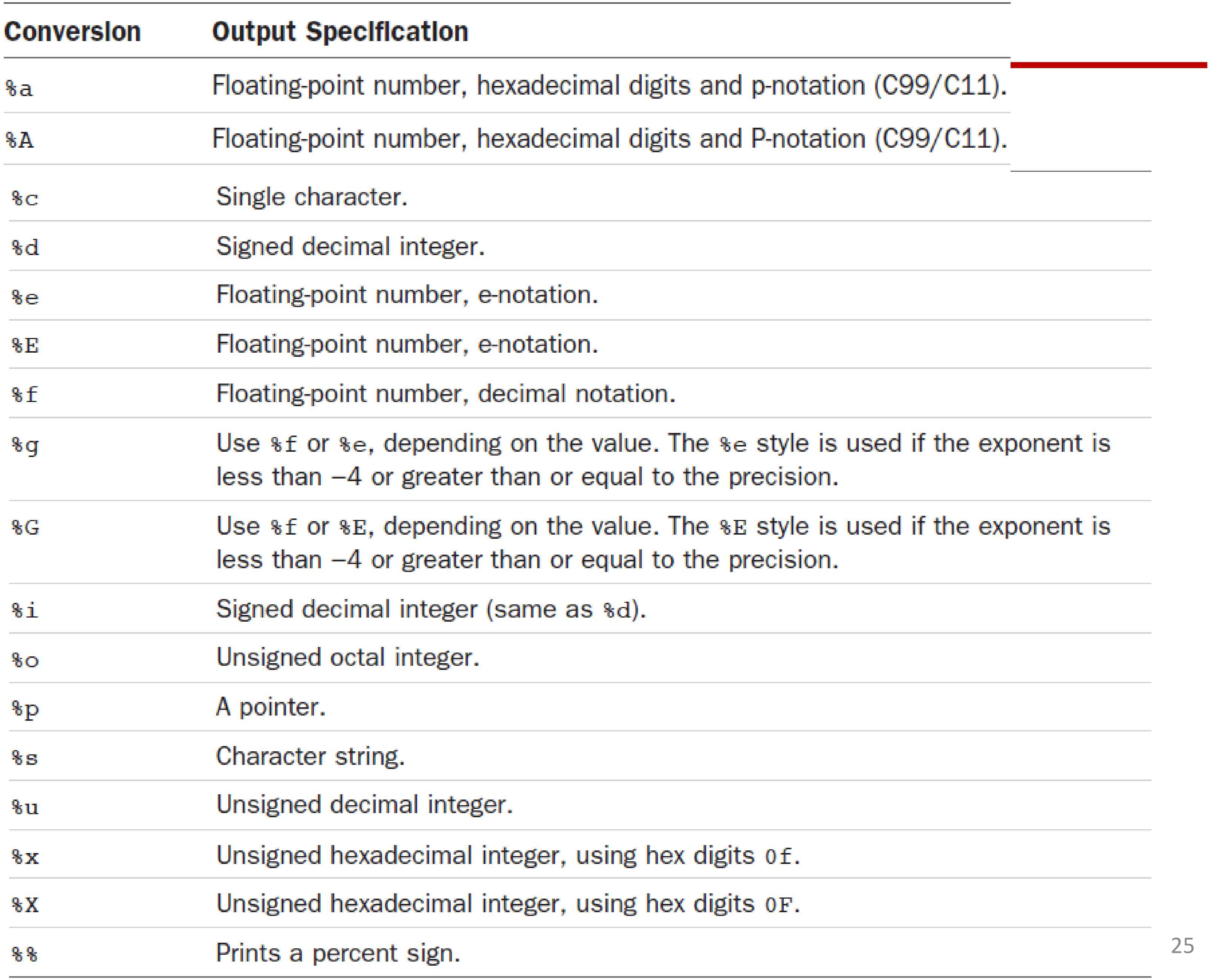
✓

```
printf(“Hello world!\m”);
printf(“%c%d\n”, ‘$’, 2*cost);
```

✗

```
printf(“The score was Squids %d, Slugs %d.\n”,
         score1);
```

printf()

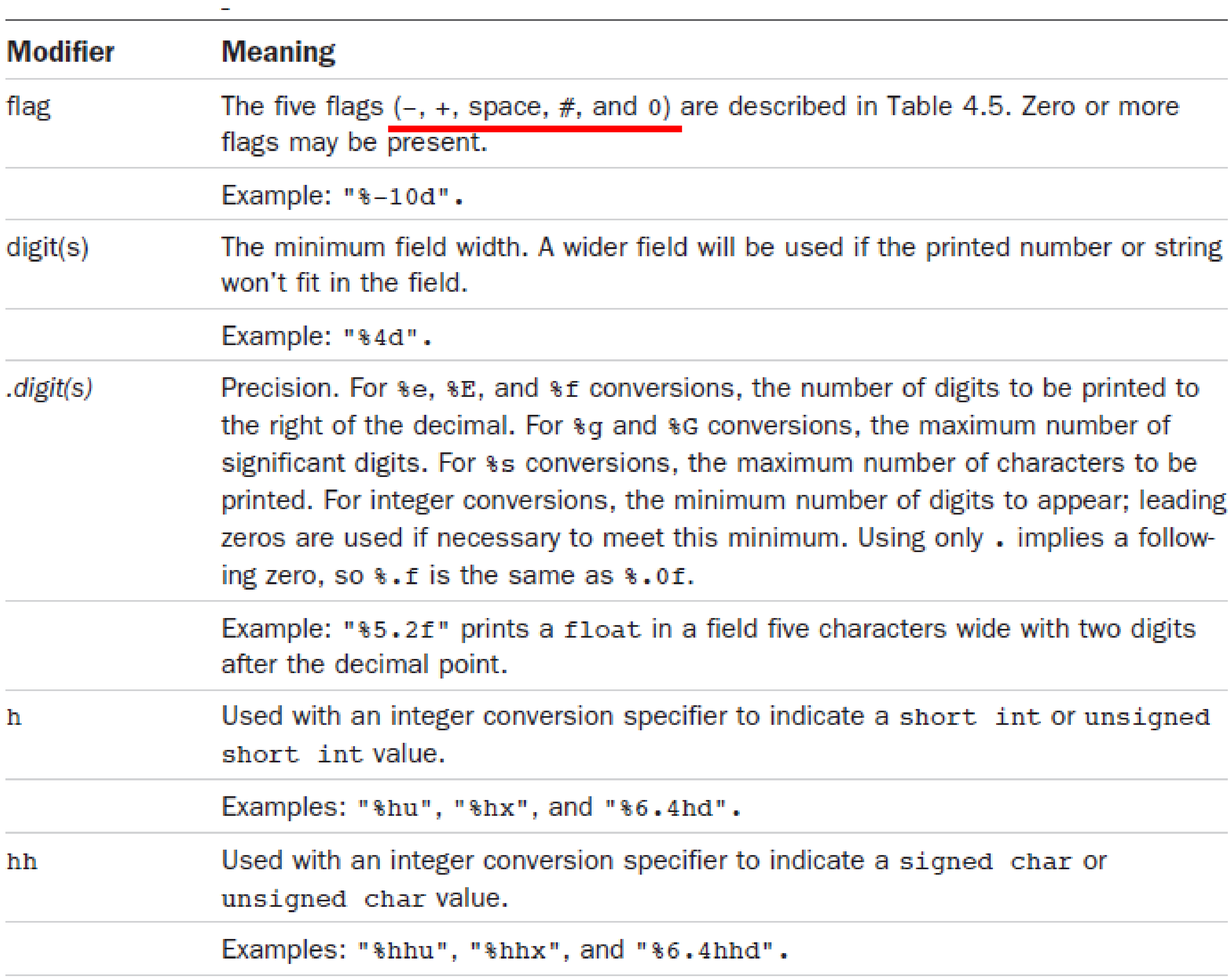


Conversion	Output Specification
`%a`	Floating-point number, hexadecimal digits and p-notation (C99/C11).
`%A`	Floating-point number, hexadecimal digits and P-notation (C99/C11).
`%c`	Single character.
`%d`	Signed decimal integer.
`%e`	Floating-point number, e-notation.
`%E`	Floating-point number, e-notation.
`%f`	Floating-point number, decimal notation.
`%g`	Use `%f` or `%e`, depending on the value. The `%e` style is used if the exponent is less than –4 or greater than or equal to the precision.
`%G`	Use `%f` or `%E`, depending on the value. The `%E` style is used if the exponent is less than –4 or greater than or equal to the precision.
`%i`	Signed decimal integer (same as `%d`).
`%o`	Unsigned octal integer.
`%p`	A pointer.
`%s`	Character string.
`%u`	Unsigned decimal integer.
`%x`	Unsigned hexadecimal integer, using hex digits `0f`.
`%X`	Unsigned hexadecimal integer, using hex digits `0F`.
`%%`	Prints a percent sign.

printf()

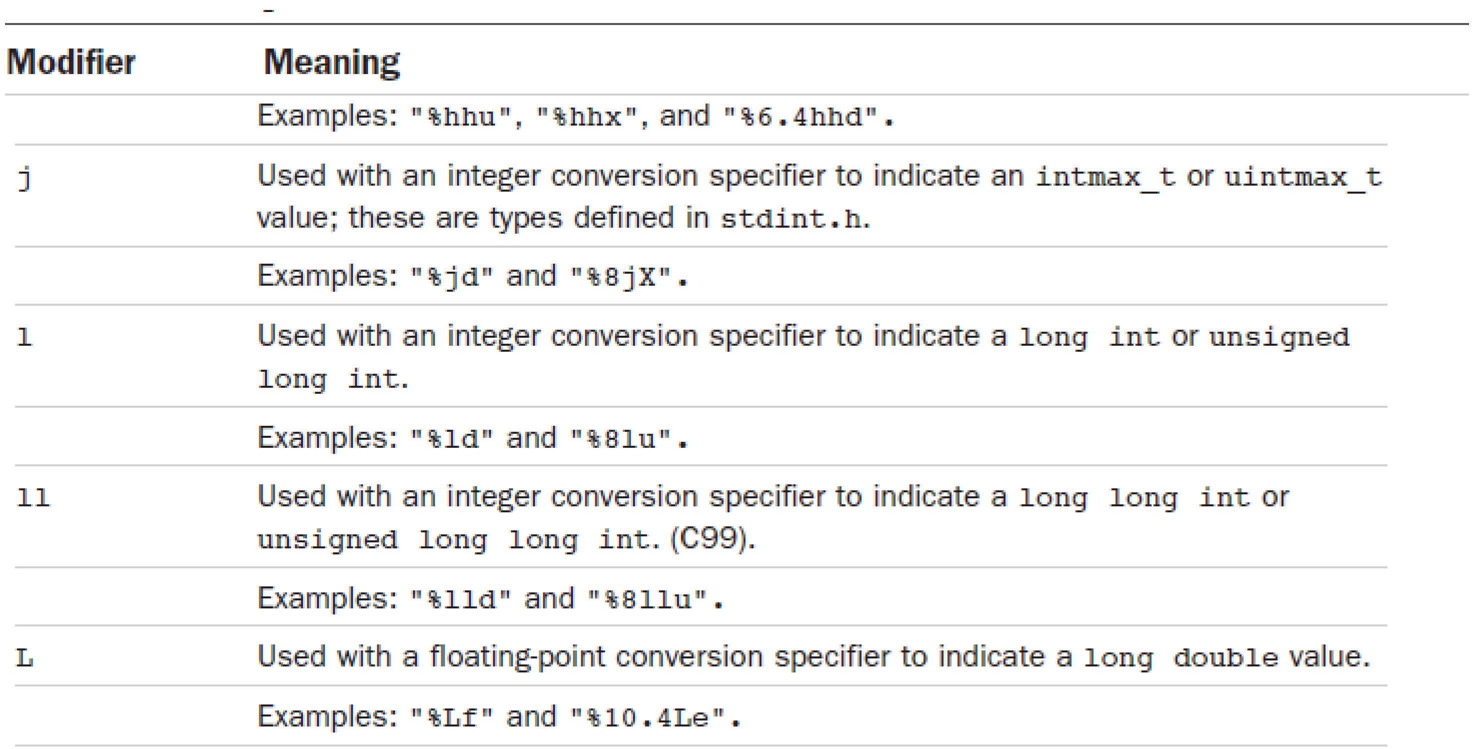
%d:以带符号的十进制形式输出整数
%o:以八进制无符号形式输出整数
%x:以十六进制无符号形式输出整数
%u:以无符号十进制形式输出整数
%c:以字符形式输出，只输出一个字符
%s:输出字符串
%f:以小数形式输出单，双精度数，隐含输出六位小数
%e:以指数形式输出实数
%g:选用%f或%e格式中输出宽度较短的一种格式，不输出无意义的0

Conversion specification modifiers for printf()



Modifier	Meaning
flag	The five flags (`-`, `+`, space, `#`, and `0`) are described in Table 4.5. Zero or more flags may be present.
	Example: `"%-10d"`.
digit(s)	The minimum field width. A wider field will be used if the printed number or string won't fit in the field.
	Example: `"%4d"`.
.digit(s)	Precision. For `%e`, `%E`, and `%f` conversions, the number of digits to be printed to the right of the decimal. For `%g` and `%G` conversions, the maximum number of significant digits. For `%s` conversions, the maximum number of characters to be printed. For integer conversions, the minimum number of digits to appear; leading zeros are used if necessary to meet this minimum. Using only `.` implies a following zero, so `%.f` is the same as `%.0f`.
	Example: `"%5.2f"` prints a `float` in a field five characters wide with two digits after the decimal point.
`h`	Used with an integer conversion specifier to indicate a `short int` or `unsigned short int` value.
	Examples: `"%hu"`, `"%hx"`, and `"%6.4hd"`.
`hh`	Used with an integer conversion specifier to indicate a `signed char` or `unsigned char` value.
	Examples: `"%hhu"`, `"%hhx"`, and `"%6.4hhd"`.

Conversion specification modifiers for printf()



Modifier	Meaning
	Examples: "%hhu", "%hhx", and "%6.4hhd".
j	Used with an integer conversion specifier to indicate an intmax_t or uintmax_t value; these are types defined in stdint.h.
	Examples: "%jd" and "%8jX".
l	Used with an integer conversion specifier to indicate a long int or unsigned long int.
	Examples: "%ld" and "%8lu".
ll	Used with an integer conversion specifier to indicate a long long int or unsigned long long int. (C99).
	Examples: "%lld" and "%8llu".
L	Used with a floating-point conversion specifier to indicate a long double value.
	Examples: "%Lf" and "%10.4Le".

提醒:

新手尽量使用简单格式

scanf()-conversion specifier

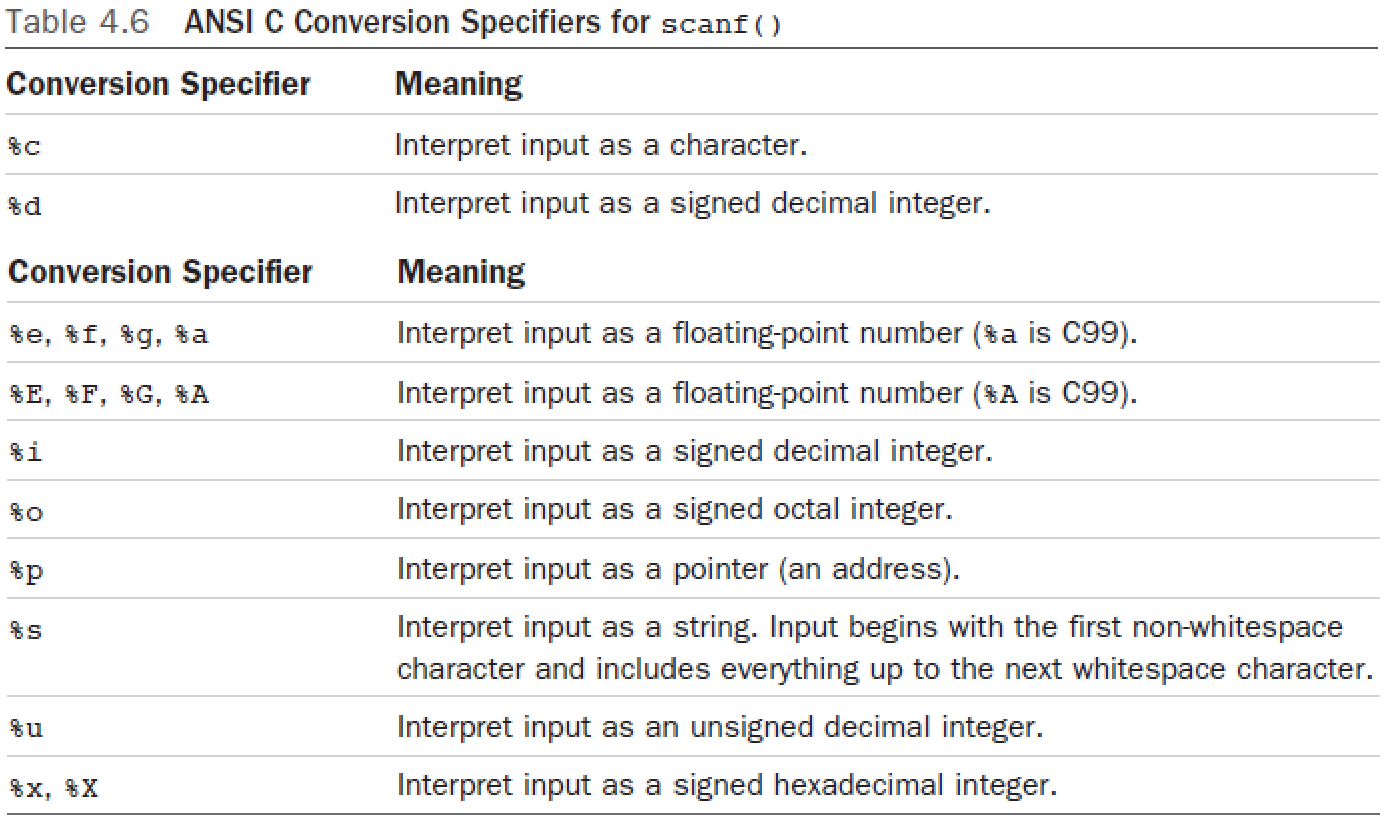


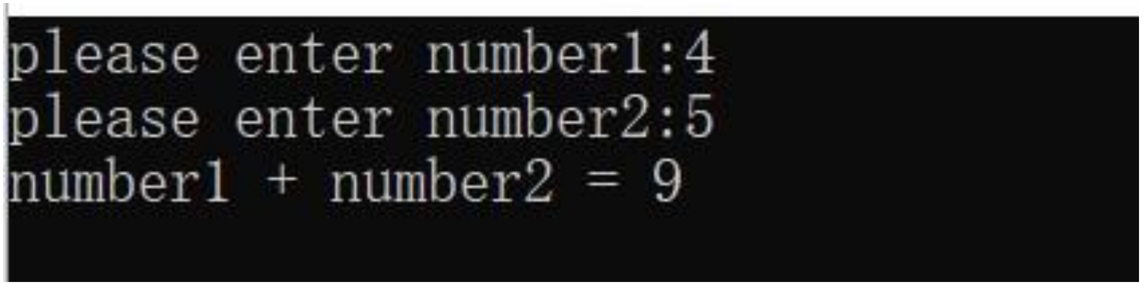
Table 4.6 **ANSI C Conversion Specifiers for `scanf()`**

Conversion Specifier	Meaning
`%c`	Interpret input as a character.
`%d`	Interpret input as a signed decimal integer.
Conversion Specifier	**Meaning**
`%e`, `%f`, `%g`, `%a`	Interpret input as a floating-point number (`%a` is C99).
`%E`, `%F`, `%G`, `%A`	Interpret input as a floating-point number (`%A` is C99).
`%i`	Interpret input as a signed decimal integer.
`%o`	Interpret input as a signed octal integer.
`%p`	Interpret input as a pointer (an address).
`%s`	Interpret input as a string. Input begins with the first non-whitespace character and includes everything up to the next whitespace character.
`%u`	Interpret input as an unsigned decimal integer.
`%x`, `%X`	Interpret input as a signed hexadecimal integer.

scanf() and printf()

Example 1: input 2 integers and make calculation

```c
#include<stdio.h>
int main(void)
{ int num1;
  int num2;
  int num3=0;
  printf("please enter number1:");
  scanf("%d",&num1);
  printf("please enter number2:");
  scanf("%d",&num2);
  num3=num1+num2;
  printf("number1 + number2 = %d\n",num3);
  return 0;
}
```



```
please enter number1:4
please enter number2:5
number1 + number2 = 9
```

int num1;

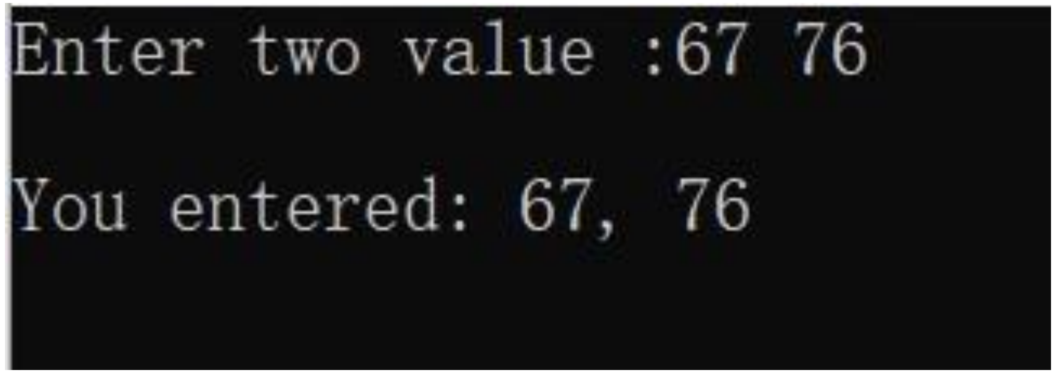
&num1

Get address of num1

scanf() and printf()

Example 2: input 2 integers in char and `int` formats

```
#include <stdio.h>
int main(void)
{
  char str[100];
  int i;

  printf( "Enter two value :");
  scanf("%s %d", str, &i);
  printf( "\nYou entered: %s, %d ", str, i);
  return 0;
}
```



```
Enter two value :67 76

You entered: 67, 76
```

int i; **char str[100];**

&i str

Get address of i

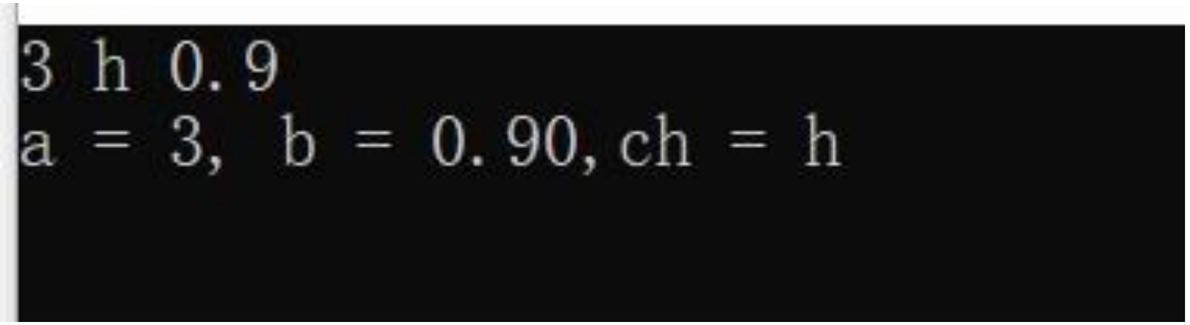
scanf() and printf()

```
scanf("%c %c %c",&a,&b,&c);

 scanf(“ %c %c %c",&a,&b,&c);

 scanf(“ %c %c %c ",&a,&b,&c);

scanf(“ %c, %c %c",&a,&b,&c);

scanf(“ %c , %c %c",&a,&b,&c);
```

???

scanf() and printf()

Example 3: input different types of data



```
3 h 0.9
a = 3,  b = 0.90, ch = h
```

```
#include<stdio.h>
int main(void)
{   int a;
    char ch;
    float b;
    scanf("%d %c %f",&a,&ch,&b);
    printf("a = %d, b = %.2f,ch = %c\n", a, b, ch);
return 0;
}
```

printf() *

```
int main(void)
{
  unsigned width, precision;
  int number = 256;
  double weight = 242.5;
  printf("Enter a field width:\n");
  scanf("%d", &width);
  printf("The number is :%*d:\n", width, number);
  printf("Now enter a width and a precision:\n");
  scanf("%d %d", &width, &precision);
  printf("Weight = %*.*f\n", width, precision, weight);
  printf("Done!\n");
  return 0;
}
```

```
Enter a field width:
6
The number is : 256:
Now enter a width and a precision:
8 3
Weight = 242.500
Done!
```

scanf() *

```
/* skiptwo.c -- skips over first two integers of input
*/
#include <stdio.h>
int main(void)
{
  int n;
  printf("Please enter three integers:\n");
  scanf("%*d %*d %d", &n);
  printf("The last integer was %d\n", n);
  return 0;
}
```

```
Please enter three integers:
2013 2014 2015
The last integer was 2015
```

This skipping facility is useful if, for example, a program needs to read a particular column of a file that has data arranged in uniform columns.

Content

1. Bit and byte
2. Data types
3. Type casting

Content

1. Bit and byte

2. Data types

3. Type casting

Bit and byte

Bit (位)

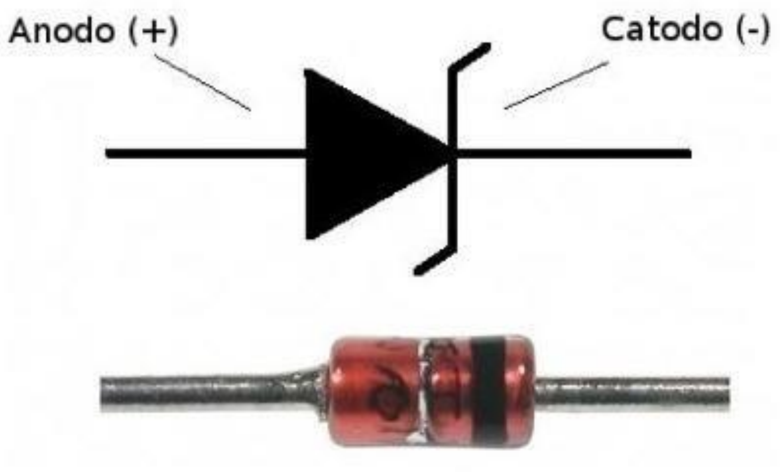
The smallest unit for storage (atomic).It can hold one of two values: **0 or 1**.

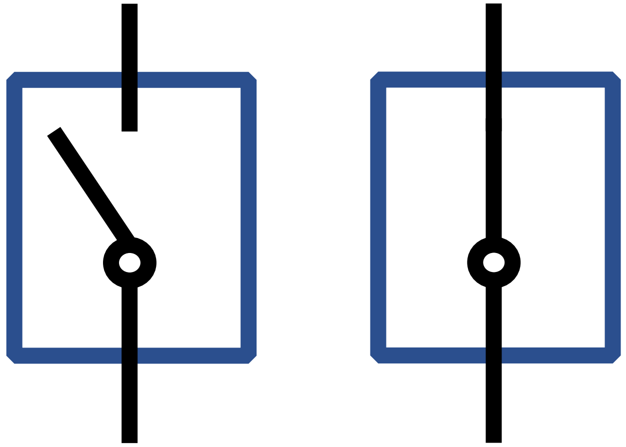
Byte (字节)

The smallest unit for information storage, 1 byte = 8 bits

Bit

Computer is nothing but a vast collection of **diodes (on and off)**, denoting the state of 0 and 1.





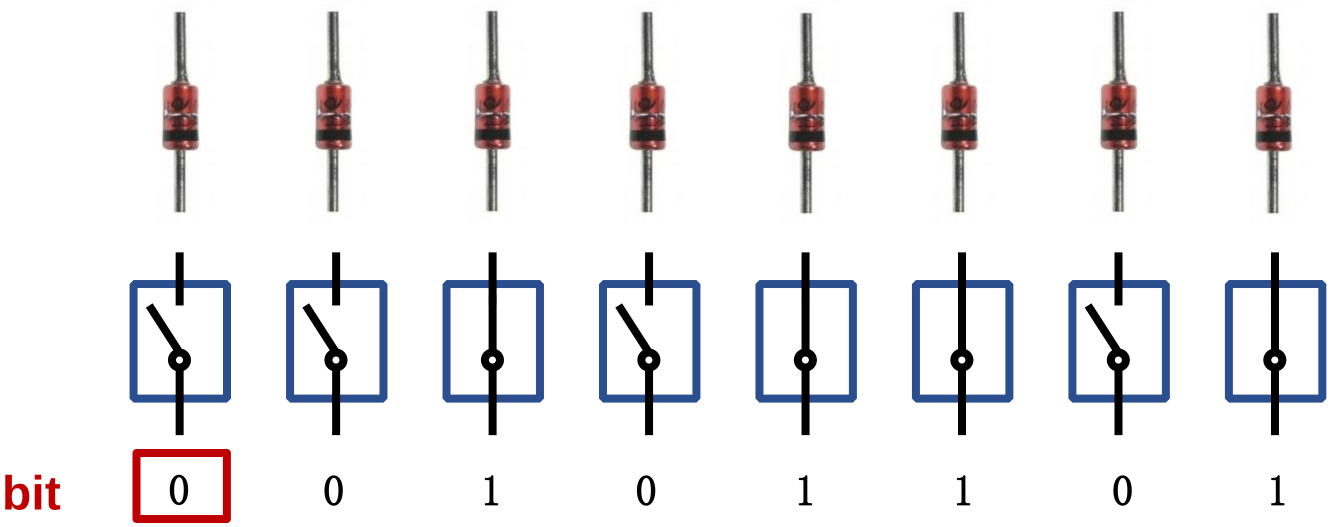
Off **On**

“0” **“1”**

False/No **True/Yes**

Byte

1 byte = 8 bits



More diodes = More bits

↓

More complex information

- 1024 bytes = 1 KB (Kilobyte)
- 1024 KB = 1 MB (Megabyte)
- 1024 MB = 1 GB (Gigabyte)
- 1024 GB = 1 TB (Terabyte)
- 1024 TB = 1 PB (Petabyte)

1 KB = 1024 (2^10) bytes = 8192 bits

Decimal numbering system

2 0 2 2

$$= (2 * 10^3) + (0 * 10^2) + (2 *10^1) + (2 * 10^0)$$

↓

Use 10 as basis

Binary numbering system

1 0 1 0

$$= (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0)$$

$$= 11$$

Use 2 as basis

Binary numbering system

1 0 0 1 0

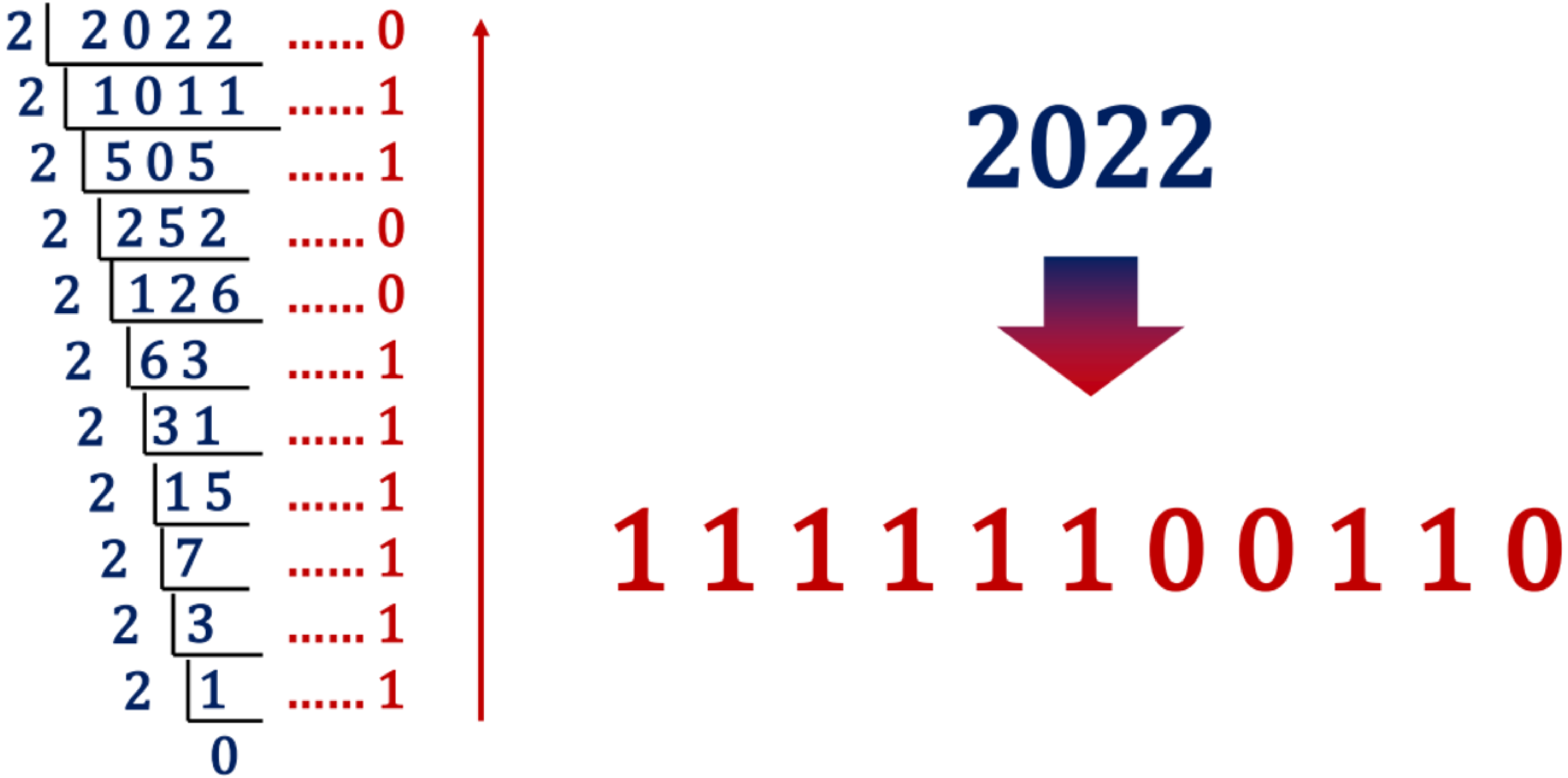
$= (1 * 2^4) + (0 * 2^3) + (0 *2^2) + (1 * 2^1)+(0 * 2^0)$

$=18$

↓

Use 2 as basis

Decimal to binary

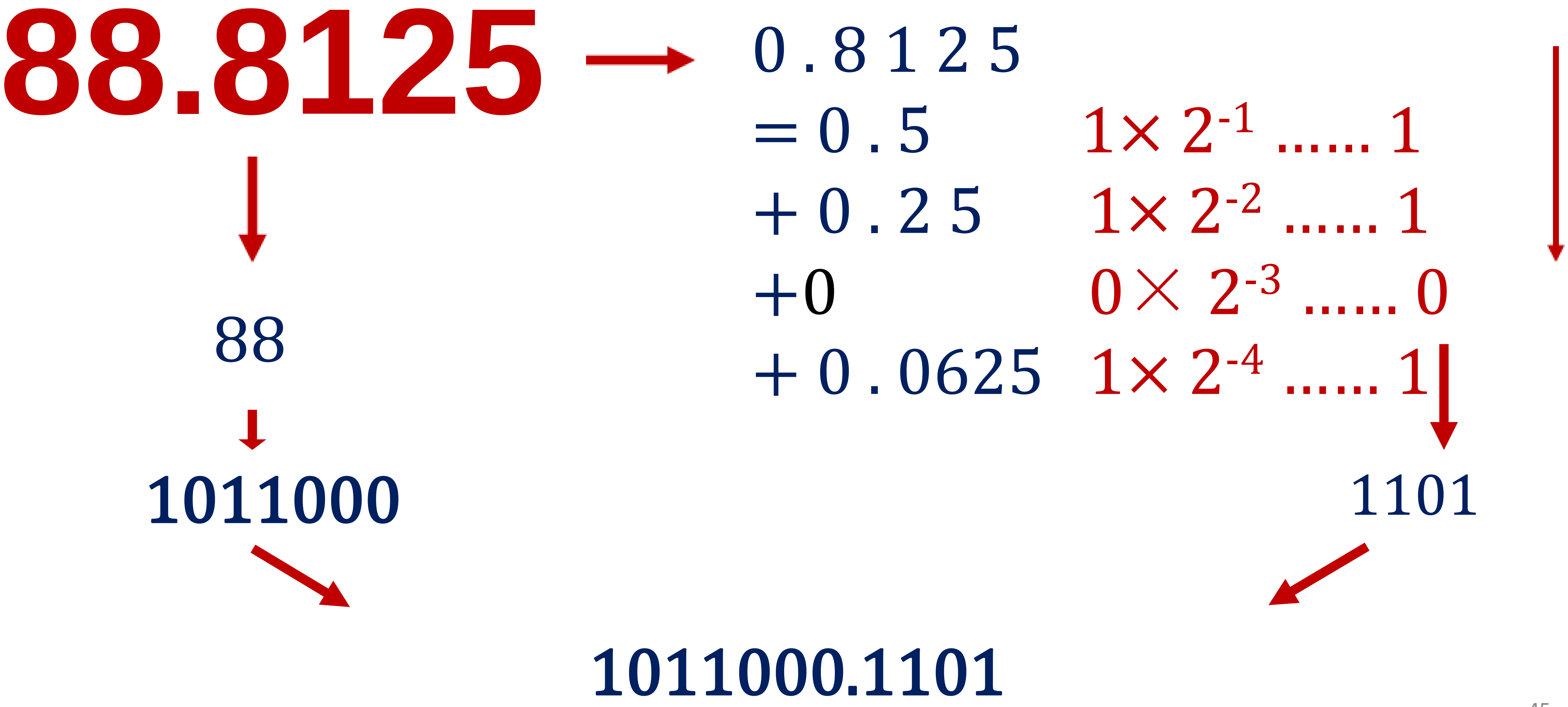


2 | 2022 0
2 | 1011 1
2 | 505 1
2 | 252 0
2 | 126 0
2 | 63 1
2 | 31 1
2 | 15 1
2 | 7 1
2 | 3 1
2 | 1 1
0

2022

1 1 1 1 1 1 0 0 1 1 0

(float)Decimal to binary



88.8125 →

0.8125

$= 0.5$ $\quad 1\times 2^{-1}$ 1

$+ 0.25$ $\quad 1\times 2^{-2}$ 1

$+0$ $\quad 0\times 2^{-3}$ 0

$+ 0.0625$ $\quad 1\times 2^{-4}$ 1

88

1011000

1101

1011000.1101

Binary to (float) decimal

$$1011000.1101$$

$= (1 * 2^{6})+(0 * 2^{5})+(1 * 2^{4})+ (1 * 2^{3}) + (0 *2^{2})$

$+ (0 * 2^{1})+(0 * 2^{0})+(1 * 2^{-1})+(1 * 2^{-2})+(0 * 2^{-3})$

$+(1 * 2^{-4})$

$=88.8125$

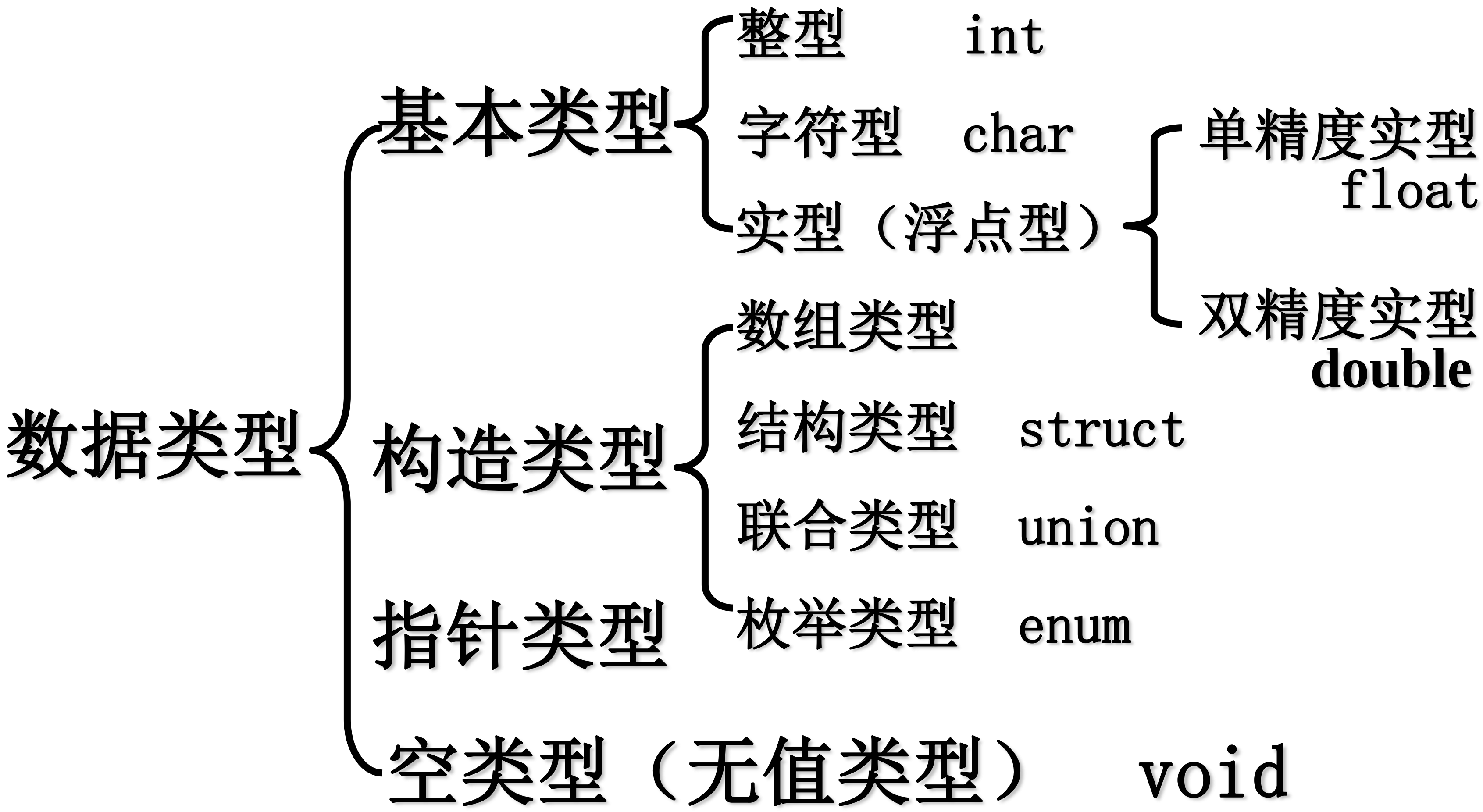
Bit and byte

- Bit is the atomic unit for machine (0 and 1)
- Byte is the smallest unit for information storage, 1 byte = 8 bits
- A collection of bytes can be used to denote integer number, real number, characters
- Machine interprets everything in the binary format

Content

1. Bit and byte
2. **Data types**
3. Type casting

Data type

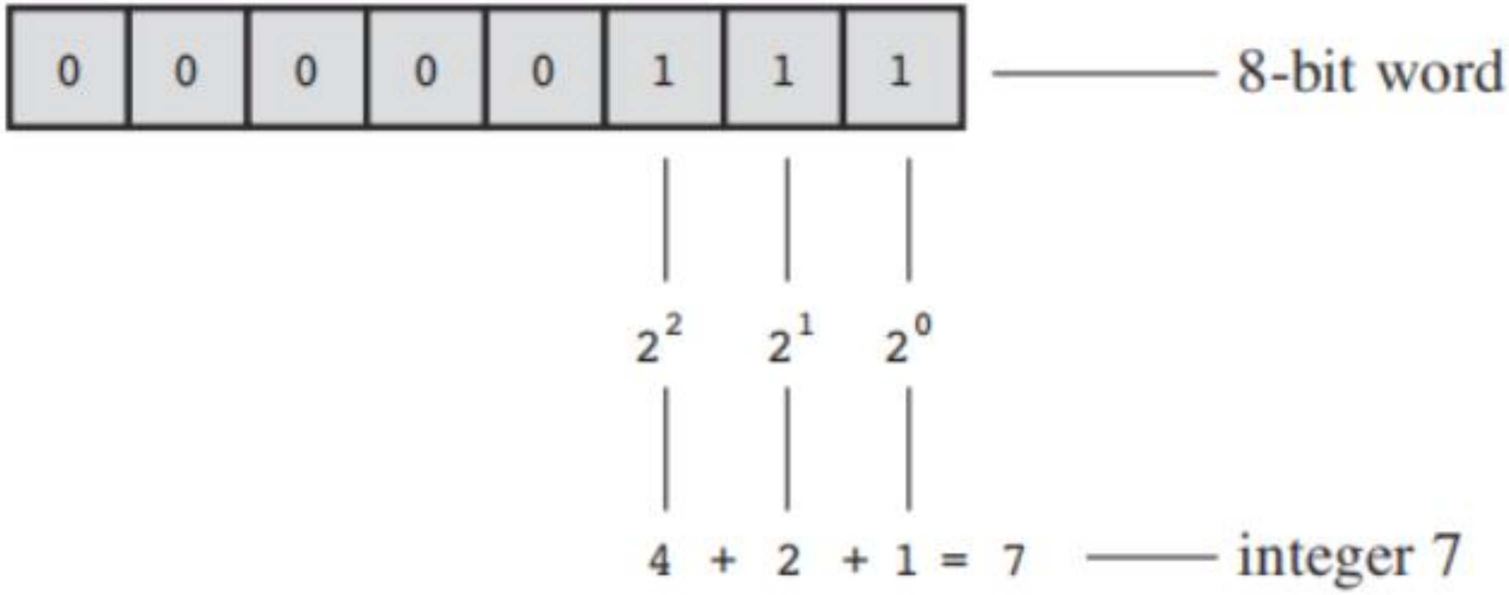


数据类型

- 基本类型
  - 整型 int
  - 字符型 char
  - 实型（浮点型）
    - 单精度实型 float
    - 双精度实型 double
- 构造类型
  - 数组类型
  - 结构类型 struct
  - 联合类型 union
  - 枚举类型 enum
- 指针类型
- 空类型（无值类型） void

Integer

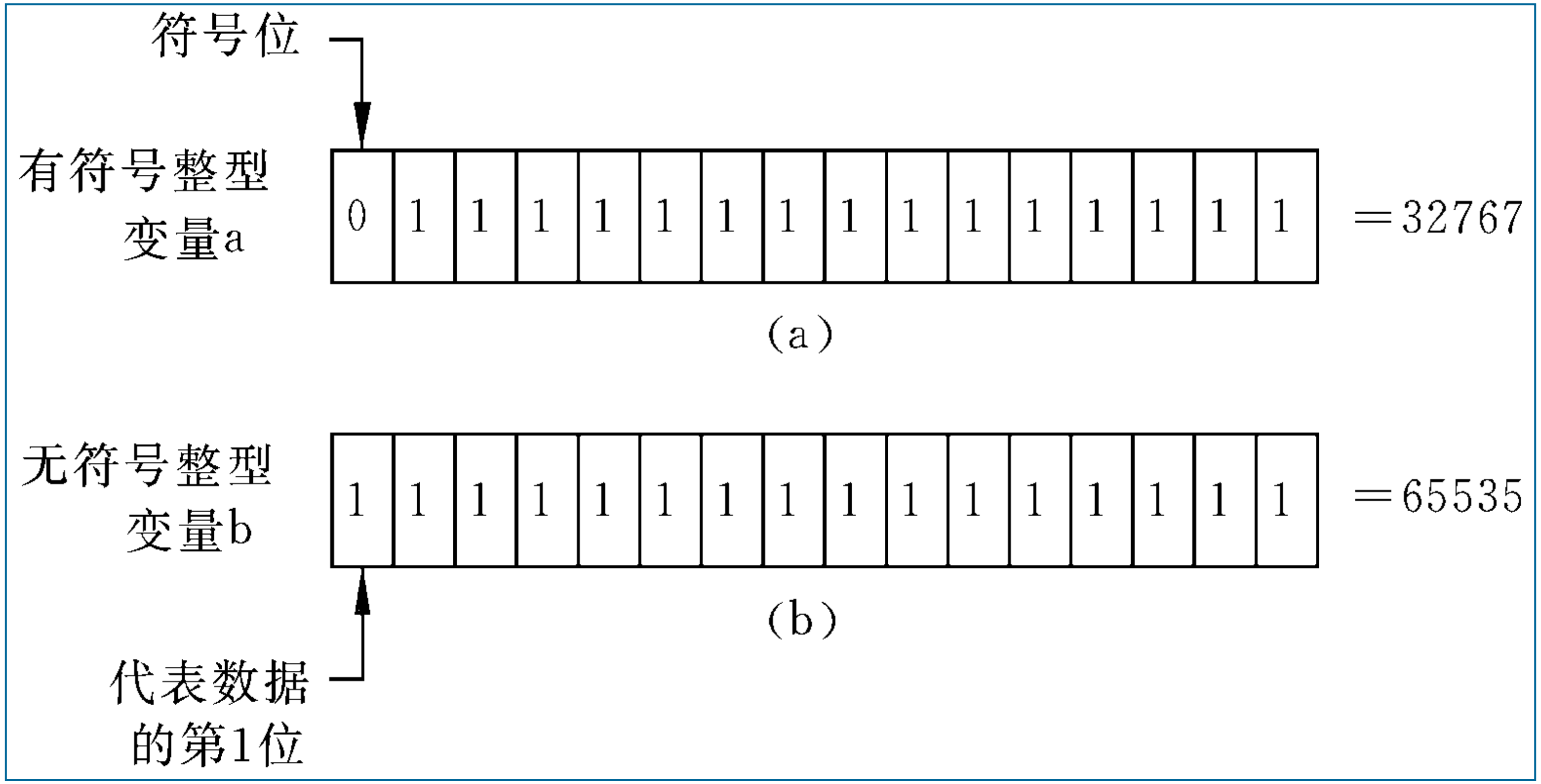
An integer is a number with no fractional part



Storing the integer 7 using a binary code.

Signed and Unsigned

The type **unsigned** int, or unsigned, is used for variables that have only nonnegative values.

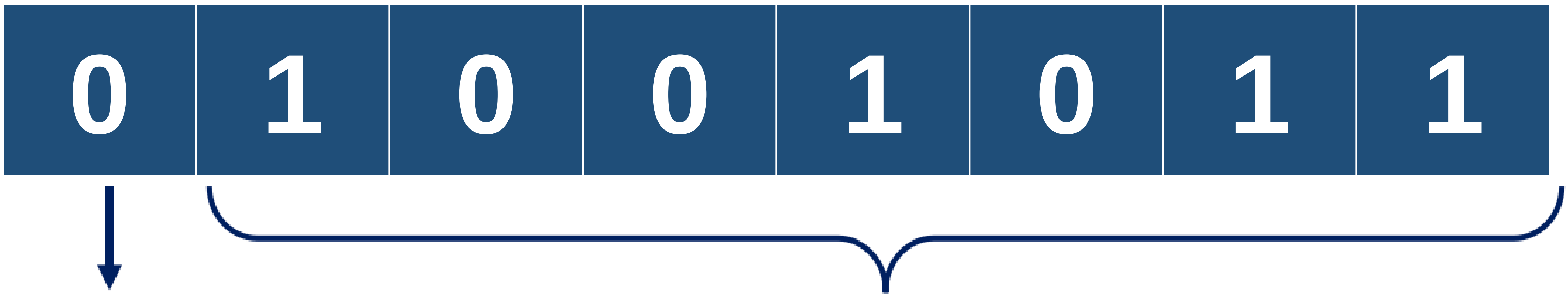


Integer

有符号基本整型	(signed)int
有符号短整型	(signed)short (int)
有符号长整型	(signed) long (int)
无符号基本整型	unsigned int
无符号短整型	unsigned short (int)
无符号长整型	unsigned long (int)

( )表示默认情况

Use byte to store **integer number**



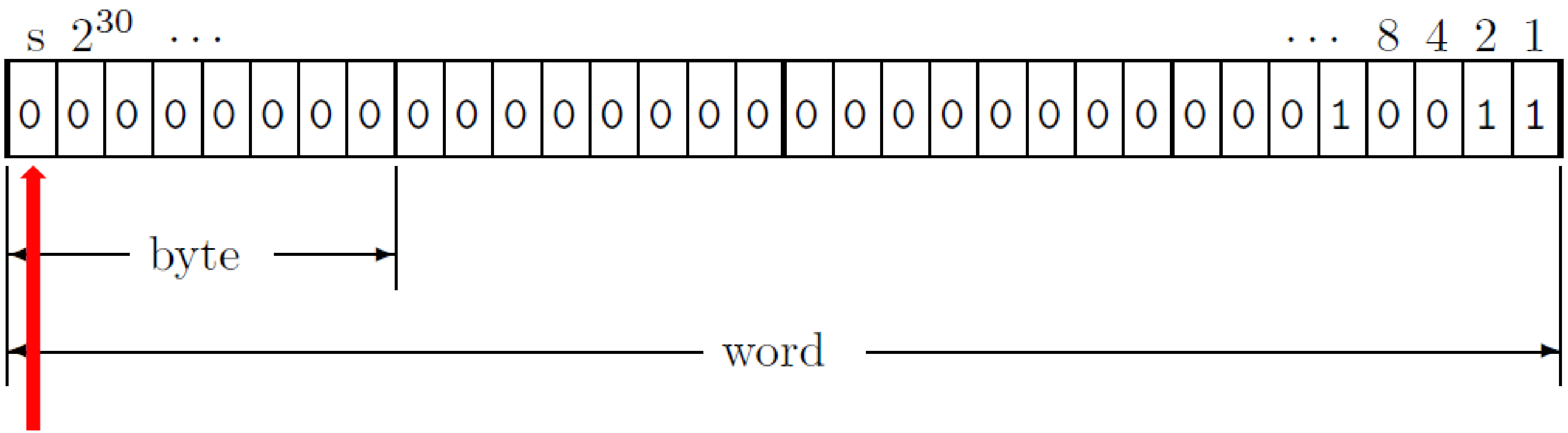
0	1	0	0	1	0	1	1

2^7-1

Sign
0 → -
1 → +

1 byte denotes range [-128, 127]

Use byte to store integer number



for an `int` type

sizeof(int) = 4

Sign bit:

0 for positive

1 for negative

$$\boxed{\sum_{i=1}^{32} 2^{i-1} * r_i} =$$

Positive numbers:	N
Negative numbers:	$2^{32} + N$

Integer

类型	类型说明符	长度	数的范围
基本型	int	4字节（大部分）	-2^{31}～$2^{31}-1$
短整型	short	2字节	-2^{15}～$2^{15}-1$
长整型	long	4字节	-2^{31}～$2^{31}-1$
双长型	long long	8字节	-2^{63}～$2^{63}-1$
无符号整型	unsigned	4字节	0～（$2^{32}-1$）
无符号短整型	unsigned short	2字节	0～65535
无符号长整型	unsigned long	4字节	0～（$2^{32}-1$）

Integer

Check it on your system

```
#include<stdio.h>
int main(void)
{
  printf("Size of unsigned int: %d\n",                 (int) sizeof(unsigned int));
  printf("Size of unsigned short int: %d\n",       (int) sizeof(unsigned short int));
  printf("Size of unsigned long int: %d\n",         (int) sizeof(unsigned long int));
  printf("Size of unsigned long long int: %d\n",  (int) sizeof(unsigned long long int));
  printf("Size of int: %d\n",                   (int) sizeof(int));
  printf("Size of short int: %d\n",        (int) sizeof(short int));
  printf("Size of long int: %d\n",          (int) sizeof(long int));
  printf("Size of long long int: %d\n",  (int) sizeof(long long int));
  return 0;
}
```

```
Size of unsigned int: 4
Size of unsigned short int: 2
Size of unsigned long int: 8
Size of unsigned long long int: 8
Size of int: 4
Size of short int: 2
Size of long int: 8
Size of long long int: 8
```

Integer

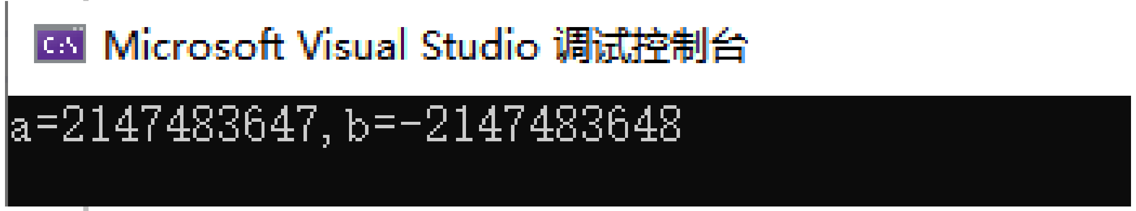
- 一个整数，如果其值超过了-2^{31}～2^{31}-1范围内，则可以将它赋值给一个long long int型变量。

- 一个整常量后面加一个字母u或U，认为是unsigned int型，如12345u，在内存中按unsigned int规定的方式存放(存储单元中最高位不作为符号位，而用来存储数据)。如果写成-12345u，则先将-12345转换成其补码53191，然后按无符号数存储。

- 在一个整常量后面加一个字母l或L，则认为是long int型常量。例如：123l,432L,0L

Integer

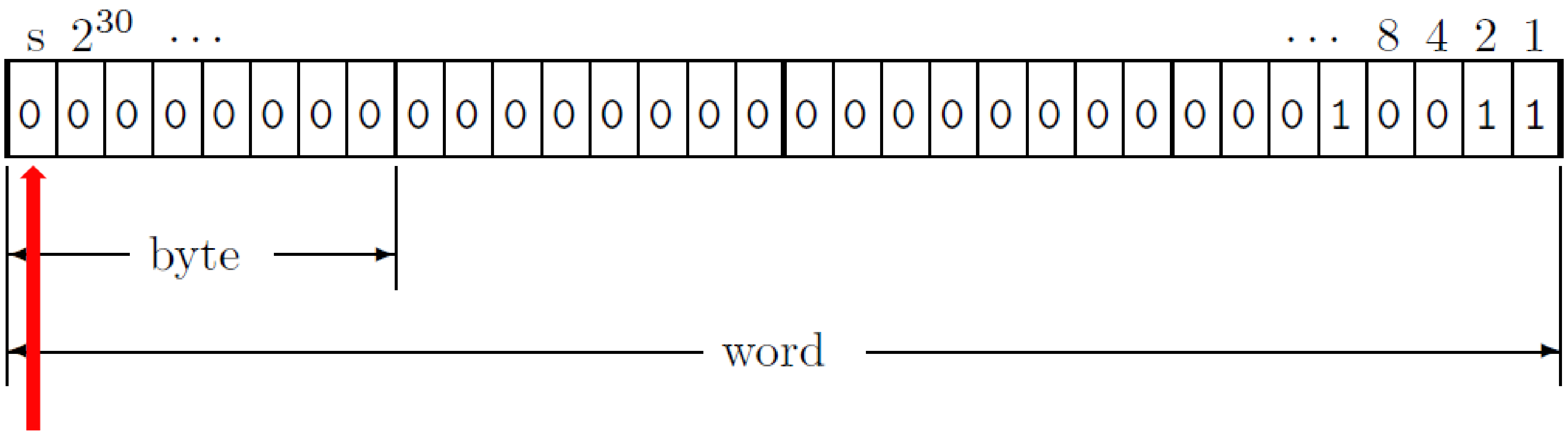
```
#include <stdio.h>
void  main()
{int a,b;
 a=2147483647;
 b=a+1;

printf( “%d,%d\n” ,a,b)
}
```



```
a=2147483647,b=-2147483648
```

说明： 数值是以**补码**表示的。一个整型变量只能容纳-32768～32767范围内的数，无法表示大于32767或小于-32768的数。遇此情况就发生“溢出”。

Integer



Sign bit:

0 for positive

1 for negative

$$\boxed{\sum_{i=1}^{32} 2^{i-1} * r_i} =$$

Positive numbers:	N
Negative numbers:	$2^{32} + N$

Integer

- The largest positive value by an `int`

```
01111111111111111111111111111111
```

$$2^{30} + 2^{29} + 2^{28} + \cdots + 2^{1} + 2^{0} = 2^{31} - 1 = 2147483647 \approx 2 \times 10^{9}$$

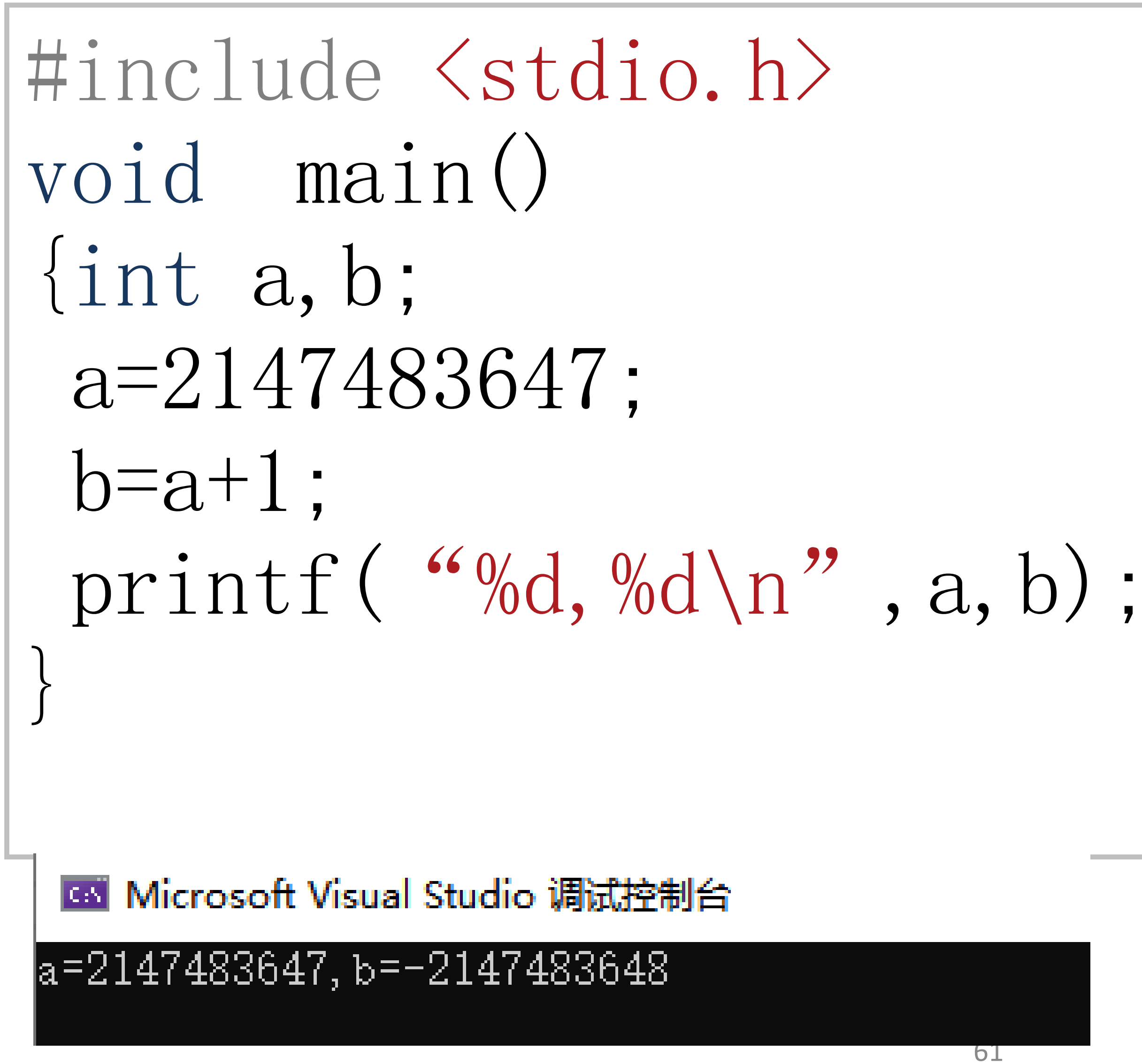
Integer

- Larger than the largest positive value?

$$\begin{array}{r} 01111111111111111111111111111111 \\ +1 \\ \hline 10000000000000000000000000000000 \end{array} = 2^{31}$$

2^{32}+N=2^{31},

➔N=-2^{31}



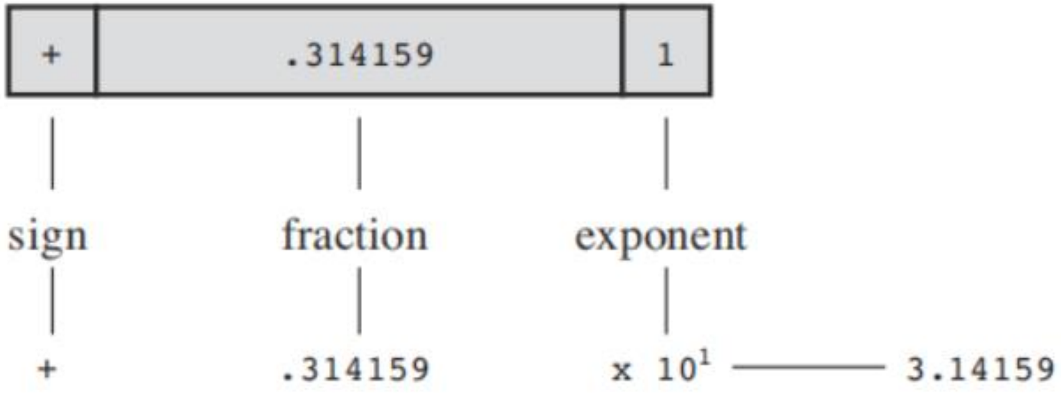
```
#include <stdio.h>
void  main()
{int a,b;
 a=2147483647;
 b=a+1;
 printf( “%d,%d\n” ,a,b);
}
```

```
a=2147483647,b=-2147483648
```

Floating-Point Types

- A floating-point number more or less corresponds to what mathematicians call a real number .
- Real numbers include the numbers between the integers, such as 2.75, 3.16E7, 7.00, and 2e–8.

The scheme used to store a floating-point number is different from the one used to store an integer!



Storing the number pi in floating-point format (decimal version).

Floating-Point Types

浮点型常量的表示方法：

两种表示形式

- 小数**decimal**　0.123
- 指数**exponent**　3e-3

注意:字母e(或E)之前必须有数字，且e后面的指数必须为整数

√　1e3、1.8e-3、-123e-6、-.1e-3

×　e3、2.1e3.5、.e3、e

Binary to decimal

1011000.1101

$= (1 * 2^{6})+(0 * 2^{5})+(1 * 2^{4})+ (1 * 2^{3}) + (0 *2^{2})$

$+ (0 * 2^{1})+(0 * 2^{0})+(1 * 2^{-1})+(1 * 2^{-2})+(0 * 2^{-3})$

$+(1 * 2^{-4})$

=88.8125

not used C for the float data

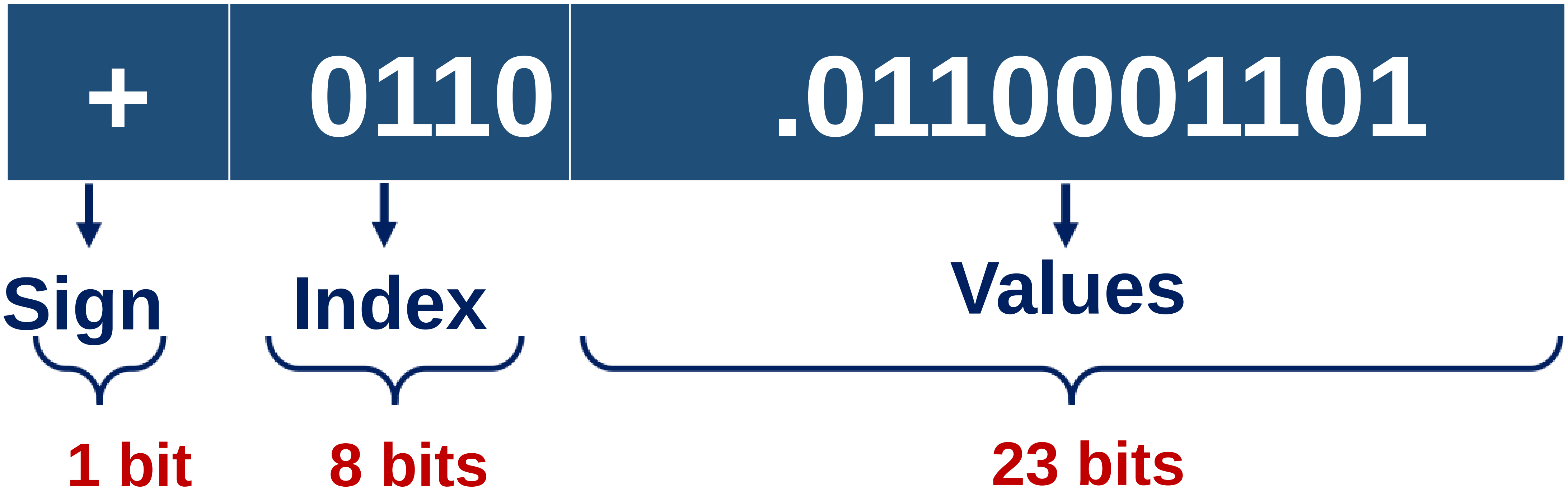
Use byte to store real number

Real number = rational number (10, -0.23) + irrational number (PI, $\sqrt{2}$)

How to use byte to denote 88.8125?!

Use 2 as basis

$$88.8125=1011000.1101=1.0110001101\times 2^6$$



+

0110

.0110001101

Sign

Index

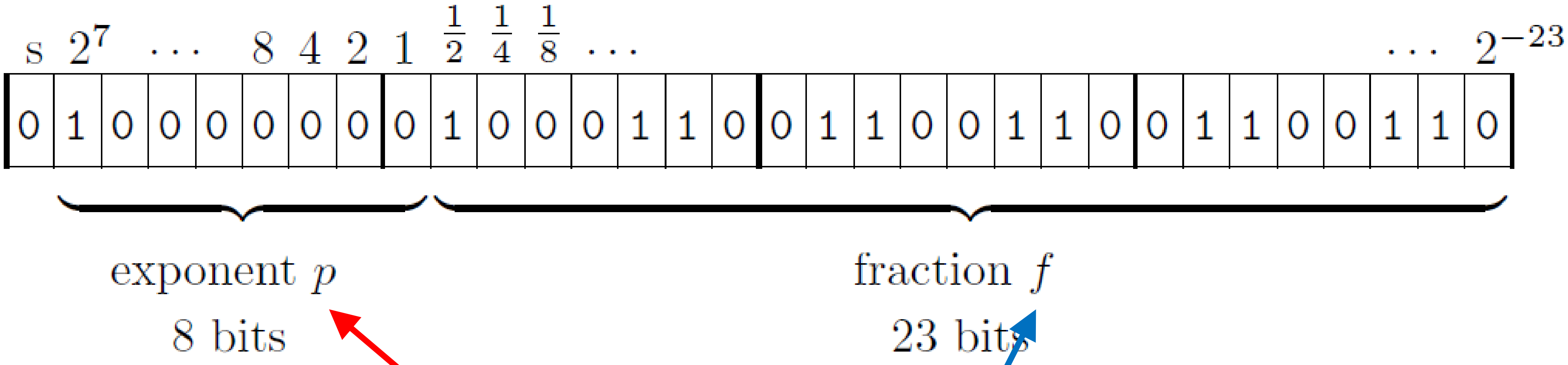
Values

1 bit

8 bits

23 bits

Floating-Point Types

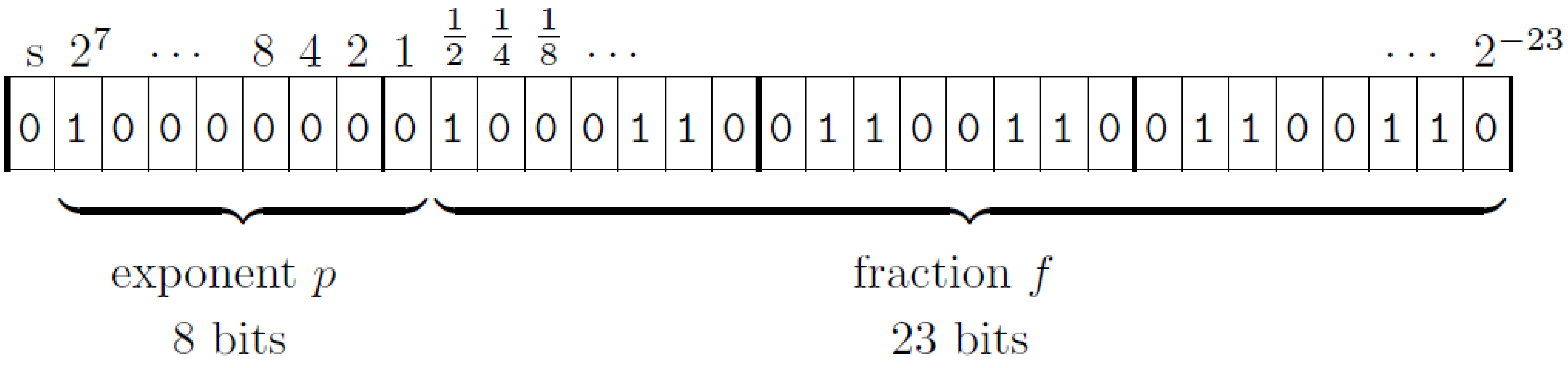


$$r = (-1)^s \times 2^{p-127} \times (1 + f)$$

±mantissa*2$^{\text{exponent}}$

假数(对数之小数部份)

Floating-Point Types



$$r = (-1)^s \times 2^{p-127} \times (1 + f)$$

The p field represents the exponent as a biased number

$2^{-126} - 2^{127}$ for the exponent part

$10^{-38} - 10^{38}$

0000 0000

1111 1111 Reserved

Floating-Point Types

浮点型变量分为单精度（float型）、双精度（double型）和长双精度型（long double）三类形式。

$$r = (-1)^s \times 2^{p-127} \times (1+f)$$

Types	Bits	Exponent bias	Range	Precision digits
float	32(8+23)	127	$10^{-38} \sim 10^{38}$	6-8
double	64(11+52)	1023	$10^{-308} \sim 10^{308}$	15-17
long double	128(15+112)	16383	$10^{-4931} \sim 10^{4932}$	33-34

It depends on the machine

bits = sign bit + (exponent + fraction) bits

Floating-Point Types

浮点型变量分为单精度（float型）、双精度（double型）和长双精度型（long double）三类形式。

```
#include<stdio.h>
int main(void)
{
  printf("Size of float: %d\n",  (int) sizeof(float));
  printf("Size of double: %d\n",  (int) sizeof(double));
  printf("Size of long double: %d\n",  (int) sizeof(long double));
  return 0;
}
```

```
Size of float: 4
Size of double float: 8
Size of long double float: 16
```

```
note:
float f;                sizeof(3.14159) = 8
f = 3.14159f;
```

float: round-off error

```
#include <stdio.h>
void main()
{float a,b;
 a = 1234567890.0f;
 b = a + 20 ;
 printf(“a=%f\n”,a);
 printf(“b=%f\n”,b);
 return 0;
}
```

? ? ?

说明：一个浮点型变量只能保证的有效数字是7位有效数字，后面的数字是无意义的，并不准确地表示该数。应当避免将一个很大的数和一个很小的数直接相加或相减，否则就会“丢失”小的数

float: round-off error

```
#include<stdio.h>
int main(void)
{ float a1, a2, b, c;
  a1 = 123456789.2f;

  b = a1 + 500;
  c = b - 500;
  a2  = c;

  if(a1 == a2)
  {printf("a1 = a2. \n");}
  else
  {printf("a1 /= a2. \n");}
  printf("a1=%f\n", a1);
  printf("a2=%f\n", a2);
  printf("b=%f\n", b);
  printf("c=%f\n", c);
  return 0;
}
```

舍入误差是指运算得到的近似值和精确值之间的差异。比如当用有限位数的浮点数来表示实数的时候(理论上存在无限位数的浮点数)就会产生舍入误差。舍入误差是量化误差的一种形式。 如果在一系列运算中的一步或者几步产生了舍入误差，在某些情况下，误差会随着运算次数增加而积累得很大，最终得出没有意义的运算结果。

```
a1 /= a2.
a1=123456792.000000
a2=123456800.000000
b=123457296.000000
c=123456800.000000
```

```
a1 = 1234.2f;
```

```
a1 = a2.
a1=1234.199951
a2=1234.199951
b=1734.199951
c=1234.199951
```

float: round-off error

Suppose we have a machine that does decimal arithmetic and retains **3 significant digits** in each result. What happen?

$$1.00 + \underbrace{.001 + .001 + \cdots + .001}_{1000 \text{ terms}}$$

1.00

$$\underbrace{.001 + .001 + \cdots + .001}_{1000 \text{ terms}} + 1.00$$

1.00+1.00=2.00

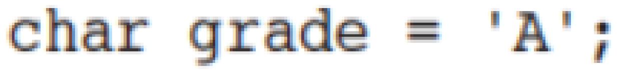
C=A-B

Not always has:

B+C=A

Using Characters: Type char

- The char type is used for storing characters such as letters and punctuation marks, but technically it is an integer type.
- A single character contained between single quotes is a C character constant.



```
char grade = 'A';
```

Use byte to store character(s)

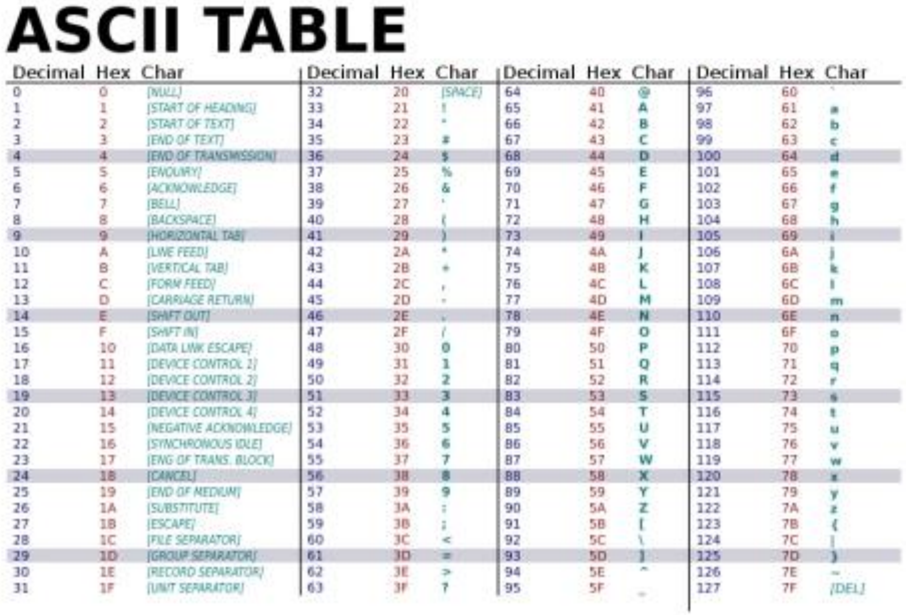
Characters are A, B, C, &, $, %, etc.

0	1	0	0	0	0	0	1

= 65 → ‘A’

find in ASCII table

(American National Standard Code for Information Interchange)

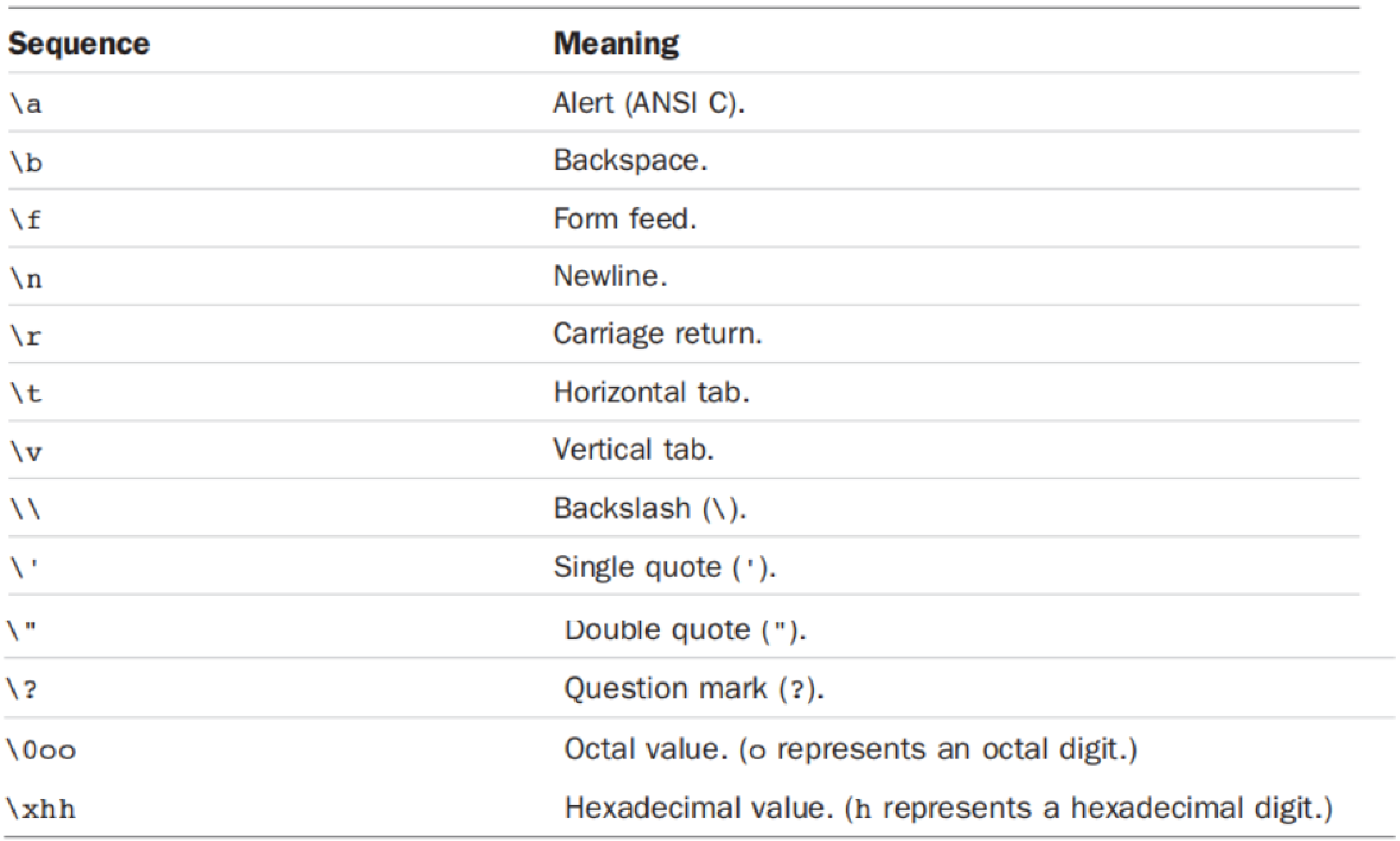


ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Character(s)

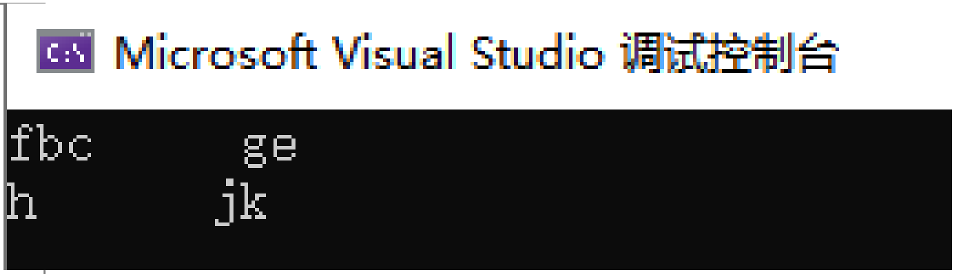
Escape sequences(转义字符)



Sequence	Meaning
\a	Alert (ANSI C).
\b	Backspace.
\f	Form feed.
\n	Newline.
\r	Carriage return.
\t	Horizontal tab.
\v	Vertical tab.
\\	Backslash (\).
\'	Single quote (').
\"	Double quote (").
\?	Question mark (?).
\0oo	Octal value. (o represents an octal digit.)
\xhh	Hexadecimal value. (h represents a hexadecimal digit.)

Character(s)

```
#include<stdio.h>
void main() {
    printf(”abc\tde\rf\tg\n”);
    printf(”h\ti\b\bjk\n”);
}
```



```
fbc     ge
h      jk
```

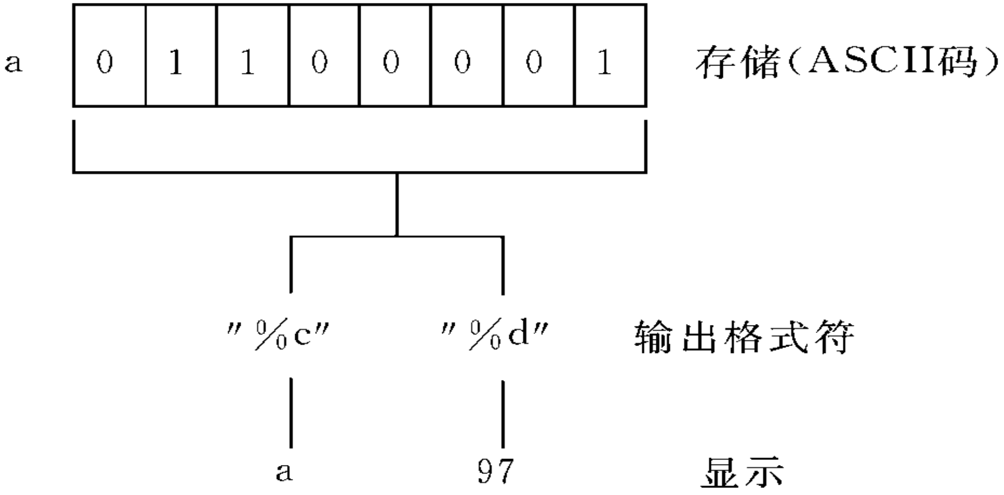
Character(s)

- 字符型变量用来存放字符常量，注意只能放一个字符。
- 一个字符变量在内存中占一个字节。
- 一个字符常量存放到一个字符变量中，实际上并不是把该字符的字型放到内存中去，而是将该字符的相应的ASCII代码放到存储单元中。这样使字符型数据和整型数据之间可以通用。

Character(s)

注意:

一个字符数据既可以以字符形式输出，也可以以整数形式输出。



Character(s)

```
#include<stdio.h>
void main() {
        char c1, c2;
        c1 = 98;
        c2 = 97;
        printf("%c %c\n", c1, c2);
        printf("%d %d\n", c1, c2);
}
```

运行结果：

```
a    b
97   98
```

- **说明：** 在第３和第4行中，将整数97和98分别赋给c1和c2，它的作用相当于以下两个赋值语句：

  c1＝′ a′ ;c2＝′ b′ ;

  因为’a’和’b’的ASCII码为97和98

Character(s)

```
#include <stdio.h>
void main()
   {char c1,c2;
    c1=’ a’ ;
    c2=’ b’ ;
     c1=c1-32;
     c2=c2-32;
    printf( “%c %c″ ,c1,c2) ;
 }
```

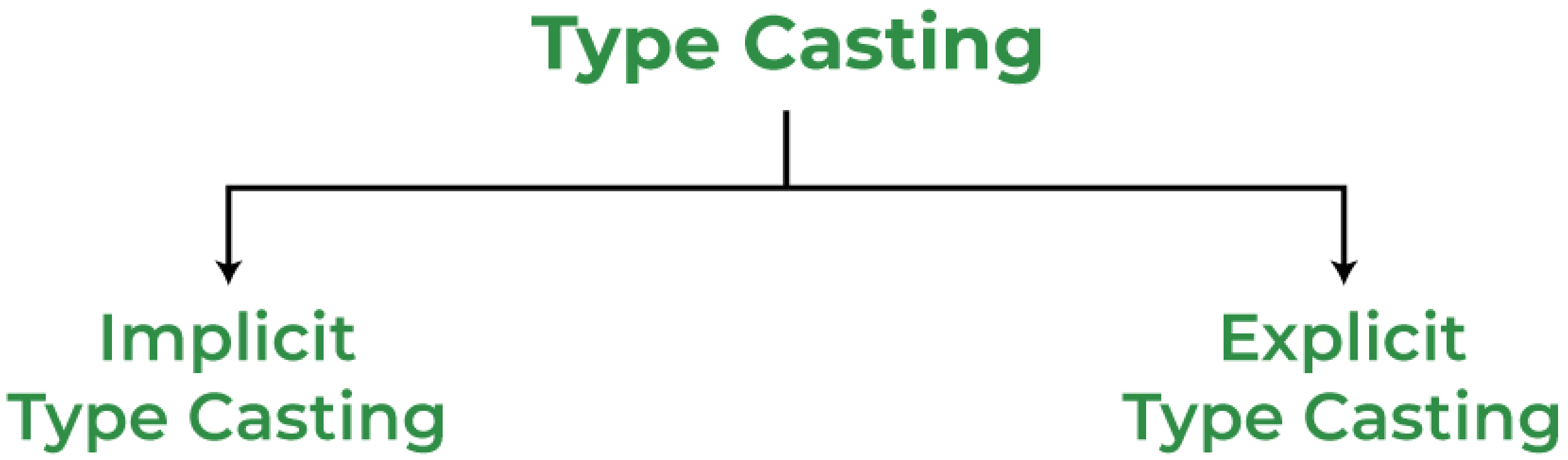
运行结果： A B

说明： 程序的作用是将两个小写字母a和b转换成大写字母A和B。从ＡＳＣＩＩ代码表中可以看到每一个小写字母比它相应的大写字母的ASCII码大32。Ｃ语言允许字符数据与整数直接进行算术运算。

Content

1. Bit and byte
2. Data types
3. **Type casting**

Type casting



Type Casting

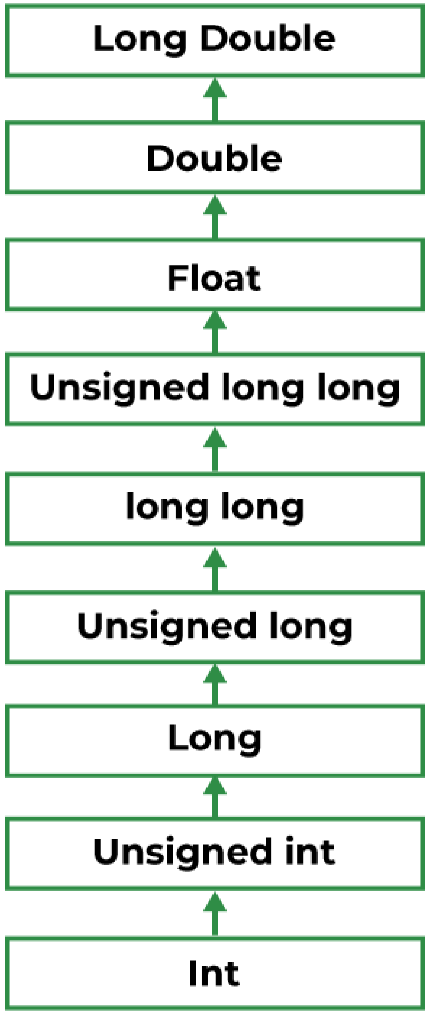
Implicit Type Casting

Explicit Type Casting

Type casting: Implicit Type Casting

- Implicit type casting in C is used to convert the data type of any variable without using the actual value that the variable holds. It performs the conversions without altering any of the values which are stored in the data variable. Conversion of lower data type to higher data type will occur automatically.

- Integer promotion will be performed first by the compiler. After that, it will determine whether two of the operands have different data types. Using the hierarchy below, the conversion would appear as follows if they both have varied data types:



```
1.0f/2;
sqrt(2.0+1);
```

Type casting: Implicit Type Casting

```
// C program to illustrate the use of
//typecasting
#include <stdio.h>
int main()
{       int a = 15, b = 2;
        float div;
        div = a / b;
        printf("The result is %f\n", div);
        return 0;
}
```

Output:

```
The result is 7.000000
```

Type casting: Explicit Type Casting

There are some cases where if the datatype remains unchanged, it can give incorrect output. In such cases, typecasting can help to get the correct output and reduce the time of compilation. In explicit type casting, we have to force the conversion between data types. This type of casting is explicitly defined within the program.

```
#include <stdio.h>
int main()
{     int a = 15, b = 2;
      char x = 'a';
      double div;
      // Explicit Typecasting in double
      div = (double)a / b;

      x = x + 3;
      printf("The result of Implicit typecasting is %c\n", x);
      printf("The result of Explicit typecasting is %f", div);
      return 0;
}
```

Output:

```
The result of Implicit typecasting is d
The result of Explicit typecasting is 7.500000
```

Type casting: Explicit Type Casting

- In C programming, there are 5 built-in type casting functions.
- **atof():** This function is used for converting the string data type into a float data type.
- **atbol():** This function is used for converting the string data type into a long data type.
- **Itoa():** This function is used to convert the long data type into the string data type.
- **itoba()**: This function is used to convert an int data type into a string data type.
- **atoi()**: This data type is used to convert the string data type into an int data type.

Suppl.Round-off errors

- During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and 28 Americans were killed.
- A later study determined that the problem was caused by the inaccuracy of the binary representation of 0.10.

— The Patriot incremented a counter once every 0.10 seconds.

— It multiplied the counter value by 0.10 to compute the actual time.

- However, the (24-bit) binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
- This doesn't seem like much, but after 100 hours the time ends up being off by 0.34 seconds — enough time for a Scud to travel 500 meters!
- Professor Skeel wrote a short article about this.
- Roundoff Error and the Patriot Missile. SIAM News, 25(4):11, July 1992.

Suppl.Floating-point

- With 32 bits, there are 2^{32}, or about 4 billion, different bit patterns.
  - These can represent 4 billion integers or 4 billion reals.
  - But there are an infinite number of reals, and the IEEE format can only represent some of the ones from about -2^{128} to $+2^{128}$.
  - Represent same number of values between 2^n and 2^{n+1} as 2^{n+1} and 2^{n+2}
- Thus, floating-point arithmetic has "issues"
  - Small round-off errors can accumulate with multiplications or exponentiations, resulting in big errors.
  - Rounding errors can invalidate many basic arithmetic principles such as the associative law, $(x + y) + z = x + (y + z)$.
- The IEEE 754 standard guarantees that all machines will produce the same results—but those results may not be mathematically accurate!