

C/C++ 语言

C/C++ 数据

张晓平

武汉大学数学与统计学院

Table of contents

1. 例子
2. 基本概念
3. 整型数据
4. `char` 类型
5. 浮点型数据

例子

问题

编程序，实现华氏温度到摄氏温度的转换。转化公式为

$$C = \frac{5}{9}(F - 32).$$

其中 F 表示华氏温度， C 表示摄氏温度。

例子

```
1 // fal2cel.c: Convert Fah temperature to Cel
   temperature
2 #include <stdio.h>
3 int main(void)
4 {
5     float fah, cel;
6     printf("Please input the Fah temperature:\n");
7     scanf("%f", &fah);
8     cel = (fah - 32.) * 5./9.;
9     printf("%.2f F = %.2f C\n", fah, cel);
10    return 0;
11 }
```

例子

```
$ gcc fah2cel.c
```

```
$ ./a.out
```

```
Please input the Fahrenheit temperature:
```

```
78
```

```
78.00 F = 25.56 C
```

例子

- 变量声明：

```
float fah, cel;
```

float 类型可以处理小数。

- 格式说明符 %f 用于输出浮点型数据。%.2f 可以精确控制输出格式，使浮点数显示到小数点后两位。
- 使用 scanf() 函数为程序提供键盘输入。
 - %f 指示 scanf() 从键盘读取一个浮点数；
 - &fah 表示变量 fah 的地址，指定将输入值赋给变量 fah 。

- 该程序的最大特点是交互性，交互性使得程序更加灵活。
 - 例如，该程序可以输入任意的华氏温度，而不必每次重写。
- `scanf()` 和 `printf()` 使得这种交互成为可能。
 - `scanf()` 从键盘读取数据并将其传递给程序；
 - `printf()` 则从程序读取数据并将其打印到屏幕。
 - 两者一起使用，就建立起了人机之间的双向通信。

基本概念

定义：常量

在程序执行过程中，值不发生改变的量称为常量。

常量分为两类：

1. 直接常量（或字面常量）
2. 符号常量

直接常量

- 整型常量：12、0、-3；
- 浮点型常量：3.1415、-1.23；
- 字符型常量：'a'、'b'

定义：标识符

用来标识变量名、符号常量名、函数名、数组名、类型名、文件名的有效字符序列。

定义：标识符

用来标识变量名、符号常量名、函数名、数组名、类型名、文件名的有效字符序列。

定义：符号常量

在 C/C++ 中，可以用一个标识符来表示一个常量，称之为符号常量。

符号常量在使用之前必须先定义，其一般形式为：

```
#define 标识符 常量
```

- `#define` 是一条预处理命令，称为**宏定义**。
- 功能是把该标识符定义为其后的常量值。
- 一经定义，在程序的预处理阶段该标识符会被替换成该常量值。

符号常量

```
#include <stdio.h>
#define PRICE 100
int main(void)
{
    int num, total;
    num = 10;
    total = num * PRICE;
    printf("total=%d\n", total);
    return 0;
}
```

符号常量

```
#include <stdio.h>
#define PRICE 100
int main(void)
{
    int num, total;
    num = 10;
    total = num * PRICE;
    printf("total=%d\n", total);
    return 0;
}
```

使用符号常量的好处是：

- 含义清楚；
- 能做到“一改全改”。

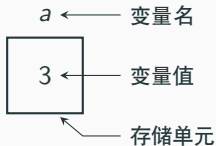
定义：变量

在程序执行过程中，值可以改变的量称为变量。

- 一个变量应该有一个名字，在内存中占据一定的存储单元。
- 变量定义必须放在变量使用之前。

变量

- 在 C 中，变量的定义一般放在函数体的开头部分。
- 在 C++ 中，变量的定义可以放在任何位置。
- 要区分变量名和变量值是两个不同的概念。



- 对于常量，编译器通过书写形式来辨认其类型。

例如，42 是整型，42.0 是浮点型。

- 变量必须在声明语句中指定其类型。

数据类型关键字

int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		

数据类型关键字

- 整型
 - `int`
 - `long`、`short`、`unsigned` 和 `signed`
- 字符型：
 - `char`
- 浮点型
 - `float`、`double` 和 `long double`
- 布尔型：（布尔值为`true` 和 `false`）
 - C: `_Bool`
 - C++: `bool`
- 复数：
 - C: `complex`
 - C++: `complex`

布尔类型

```
#include <stdio.h>
int main(void)
{
    _Bool is_printed = 1;
    if (is_printed)
        printf("Hello world!\n");
    return 0;
}
```

布尔类型

```
#include <stdio.h>
int main(void)
{
    _Bool is_printed = 1;
    if (is_printed)
        printf("Hello world!\n");
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    bool is_printed = true;
    if (is_printed)
        cout << "Hello world!" << endl;
    return 0;
}
```

表 1: C 风格的复数类型及其操作, 需要用到头文件 `complex.h`

函数	功能
<code>creal</code>	获取复数的实部
<code>cimag</code>	获取复数的虚部
<code>conj</code>	获取复数的共轭
<code>cabs</code>	获取复数的模
<code>carg</code>	获取幅角

复数类型

```
#include <stdio.h>
#include <complex.h>
int main(void)
{
    double complex a = 1 + 1I;
    printf("a = (%.2f, %.2f)\n", creal(a), cimag(a));
    double complex b = conj(a);
    printf("b = (%.2f, %.2f)\n", creal(b), cimag(b));
    printf("arg = %.5f\n", carg(a));
    printf("abs = %.5f\n", cabs(a));
    return 0;
}
```

表 2: C++ 风格的复数类型及其操作, 需要用到头文件 `complex`

方法	功能
<code>real</code>	获取复数的实部
<code>imag</code>	获取复数的虚部
<code>conj</code>	获取复数的共轭
<code>abs</code>	获取复数的模
<code>arg</code>	获取复数的幅角

复数类型

```
#include <cstdio>
#include <complex>
using namespace std;
int main(void)
{
    complex<double> a (1.0, 1.0);
    printf("a = (%.2f, %.2f)\n", a.real(), a.imag());
    complex<double> b = conj(a);
    printf("b = (%.2f, %.2f)\n", b.real(), b.imag());
    printf("arg = %.5f\n", arg(a));
    printf("abs = %.5f\n", abs(a));
    return 0;
}
```

整型数据

整数的存储方式

- 数据都是以二进制的形式存储。
- 整数以补码的方式存储。
 1. 正数的补码是其本身
 2. 负数的补码：将其绝对值的二进制形式按位取反再加 1。

整数的存储方式

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

图 1: 正数 10 的存储方式

10 的原码

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

取反

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

加 1, 得-10 的补码

1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

图 2: 负数-10 的存储方式

int 型表示有符号整数，其取值范围依赖于系统。

- C：使用头文件 `limits.h`
 - 它专用于检测整型数据类型（int, short, long）的取值范围。
- C++：使用头文件 `limits`
 - 它提供了 `std::numeric_limits` 模板类，用于替代 C 语言，检测各种基本数据类型的取值范围及其他信息。

int 型

```
// int_info.c
#include<stdio.h>
#include<limits.h>
int main(void)
{
    printf("range of int is %d ~ %d\n", INT_MIN, INT_MAX
);
    printf("sizeof int = %lu bytes\n", sizeof(int));
    return 0;
}
```


int 型

```
// int_info.c
#include<stdio.h>
#include<limits.h>
int main(void)
{
    printf("range of int is %d ~ %d\n", INT_MIN, INT_MAX
);
    printf("sizeof int = %lu bytes\n", sizeof(int));
    return 0;
}
```

```
$ gcc int_info.c
$ ./a.out
range of int is -2147483648 ~ 2147483647
sizeof int = 4 bytes
```

int 型

```
// int_info.cpp
#include <cstdio>
#include <limits>
using namespace std;
int main()
{
    printf( "range of int is %d ~ %d\n",
            numeric_limits<int>::min(),
            numeric_limits<int>::max() );
    printf( "sizeof int = %lu bytes\n", sizeof(int) );
    return 0;
}
```

int 型

```
// int_info.cpp
#include<cstdio>
#include<limits>
using namespace std;
int main()
{
    printf( "range of int is %d ~ %d\n",
            numeric_limits<int>::min(),
            numeric_limits<int>::max() );
    printf( "sizeof int = %lu bytes\n", sizeof(int) );
    return 0;
}
```

```
$ g++ int_info.cpp
$ ./a.out
range of int is -2147483648 ~ 2147483647
sizeof int = 4 bytes
```

int 变量的声明

关键字 `int` 用于声明基本的整型变量，书写格式为

```
int var;  
int var1, var2;
```

要声明多个变量，

- 可以逐个声明每个变量；
- 也可在 `int` 后跟一个变量名列表，各个变量之间用逗号隔开。

int 变量的赋值

int 变量的赋值有如下三种方式：

1. 先声明，后赋值

```
int n;  
n = 1;
```

2. 先声明，后通过 scanf 函数赋值

```
int n;  
scanf("%d", &n);
```

3. 初始化变量

```
int n = 1;
```

int 变量的初始化

初始化变量就是为变量赋一个初始值。

```
int a = 1;  
int b = 2, c = 3;  
int d, e = 4;  // valid, but not good
```

请避免在一个声明语句中同时出现初始化和未初始化的变量。

int 变量的初始化

声明语句为变量创建、标定存储空间并为其指定初始值。

int 变量的初始化

声明语句为变量创建、标定存储空间并为其指定初始值。

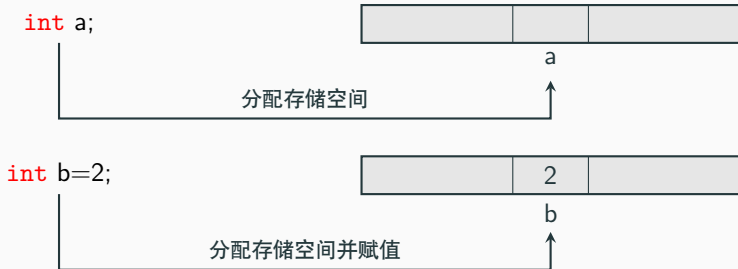


图 3: 定义和初始化变量

int 值的打印

```
// print1.c: display some properties of printf()
#include<stdio.h>
int main(void)
{
    int a = 10;
    int b = 2;
    printf("Doing it right: ");
    printf("%d - %d = %d\n", a, b, a-b);
    printf("Doing it wrong: ");
    printf("%d - %d = %d\n", a);
    return 0;
}
```

int 值的打印

```
// print1.c: display some properties of printf()
#include<stdio.h>
int main(void)
{
    int a = 10;
    int b = 2;
    printf("Doing it right: ");
    printf("%d - %d = %d\n", a, b, a-b);
    printf("Doing it wrong: ");
    printf("%d - %d = %d\n", a);
    return 0;
}
```

```
$ gcc print1.c
```

```
$ ./a.out
```

```
Doing it right: 10 - 2 = 8
```

```
Doing it wrong: 10 - 73832 = 771
```

第二次调用 `print()` 时，程序使用 `a` 为第一个 `%d` 提供打印值，然后用内存中的任意值为其余两个 `%d` 提供打印值。

注意

使用 `printf()` 时，格式说明符的个数与要显示值的数目必须相同。

八进制数和十六进制数的打印

在 C 中，有专门的前缀指明进制。

- 前缀 0x 或 0X 表示十六进制数
 - 16 的十六进制表示为 0x10 或 0X10。
- 前缀 0 表示八进制数
 - 16 的八进制表示为 020。

打印八进制数和十六进制数

```
1 // bases.c: print 100 in decimal, octal and hex
2 #include <stdio.h>
3 int main(void)
4 {
5     int x = 100;
6     printf("dec = %d; oct = %o; hex = %X\n", x, x,
7         x);
8     printf("dec = %d; oct = %#o; hex = %#X\n", x,
9         x, x);
10    return 0;
11 }
```

打印八进制数和十六进制数

```
1 // bases.c: print 100 in decimal, octal and hex
2 #include <stdio.h>
3 int main(void)
4 {
5     int x = 100;
6     printf("dec = %d; oct = %o; hex = %X\n", x, x,
7           x);
8     printf("dec = %d; oct = %#o; hex = %#X\n", x,
9           x, x);
10    return 0;
11 }
```

```
$ gcc bases.c
```

```
$ ./a.out
```

```
dec = 100; oct = 144; hex = 64
```

```
dec = 100; oct = 0144; hex = 0x64
```

八进制数和十六进制数的打印

进制	格式说明符	格式说明符（显示前缀）
十进制	%d	
八进制	%o	%#o
十六进制	%x 或 %X	%#x 或 %#X

其他整数类型

C/C++ 提供 4 个附属关键字修饰 `int` : `short`、`long`、`unsigned` 和 `signed`。

类型	含义	占位符
<code>short (int)</code>	适用于小数值	<code>%hd</code> , <code>%ho</code> , <code>%hx</code>
<code>long (int)</code>	适用于大数值	<code>%ld</code> , <code>%lo</code> , <code>%lx</code>
<code>long long (int)</code>	适用于更大数值	<code>%lld</code> , <code>%llo</code> , <code>%llx</code>
<code>unsigned (int)</code>		<code>%u</code>
<code>unsigned long (int)</code>		<code>%lu</code>
<code>unsigned long long (int)</code>		<code>%llu</code>

其他整数类型

C/C++ 提供 4 个附属关键字修饰 `int` : `short`、`long`、`unsigned` 和 `signed`。

类型	含义	占位符
<code>short (int)</code>	适用于小数值	<code>%hd</code> , <code>%ho</code> , <code>%hx</code>
<code>long (int)</code>	适用于大数值	<code>%ld</code> , <code>%lo</code> , <code>%lx</code>
<code>long long (int)</code>	适用于更大数值	<code>%lld</code> , <code>%llo</code> , <code>%llx</code>
<code>unsigned (int)</code>		<code>%u</code>
<code>unsigned long (int)</code>		<code>%lu</code>
<code>unsigned long long (int)</code>		<code>%llu</code>

关键字 `signed` 可以和任何有符号类型一起使用，使数据类型更加明确。如 `short`、`short int`、`signed short` 和 `signed short int` 表示同一种类型。

问题

为什么会出现多种整数类型？

问题

为什么会出现多种整数类型？

- 有些 CPU 的自然字太小，若认为没有表示更大数的需要，会将 long 类型和 `int` 类型定义相同的长度。
- 很多场合不要用到太大的整数，于是创建了更节省空间的 short 类型。

问题

为什么会出现多种整数类型？

- 有些 CPU 的自然字太小，若认为没有表示更大数的需要，会将 long 类型和 int 类型定义相同的长度。
- 很多场合不要用到太大的整数，于是创建了更节省空间的 short 类型。

注

C/C++ 仅保证 short 类型不会比 int 类型长，long 类型不会比 int 类型短，其目的是为了适应不同的机器。

问题

若整数太大，超出整数类型的范围会发生什么？

整数的上溢

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    int i = INT_MAX;
    unsigned int j = UINT_MAX;
    printf("i = %d, i+1 = %d, i+2 = %d\n", i, i+1, i+2);
    printf("j = %u, j+1 = %u, j+2 = %u\n", j, j+1, j+2);
    return 0;
}
```

```
$ gcc intOverflow.c
$ ./a.out
i = 2147483647, i+1 = -2147483648, i+2 = -2147483647
j = 4294967295, j+1 = 0, j+2 = 1
```

当达到最大值时，将溢出到起始点。

- 对于 `unsigned int` 类型，起始点是 0；
- 对于 `int` 类型，起始点为 -2147483648。

当达到最大值时，将溢出到起始点。

- 对于 `unsigned int` 类型，起始点是 0；
- 对于 `int` 类型，起始点为 -2147483648。

注

当整数溢出时，编译器不会给出任何提示，故编程时必须谨慎对待此类问题。

char 类型

定义：char 型数据

char 型数据是用单引号括起来的一个字符。

例如：

'a', 'b', '=', '+', '?'

都是合法 char 型数据。

char 型数据的特点

- char 型数据只能用单引号括起来。
- char 型数据只能是单个字符。
- 字符可以是字符集（如 ASCII 码）中任意字符，但数字被定义为字符型之后就不能参与数值运算。

如 '5' 和 5 是不同的。'5' 是 char 型数据。

字符变量的声明和初始化

- 字符变量的类型说明符是 `char`，其声明与整型变量相同，如：

```
char a, b;  
char c;
```

- 可以使用以下初始化语句将字符 'A' 赋给 `grade`：

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

字符变量的声明和初始化

- 字符变量的类型说明符是 `char`，其声明与整型变量相同，如：

```
char a, b;  
char c;
```

- 可以使用以下初始化语句将字符 'A' 赋给 `grade`：

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

若不使用单引号，编译器会将 A 视为一个变量名；若使用双引号，编译器将其视为一个字符串。

```
char grade;    //声明一个 char 变量  
grade = 'A';   //可以  
grade = A;     //不可以  
grade = "A";   //不可以  
grade = 65;    //可以
```

字符变量的存储方式

每个字符变量被分配一个字节的内存空间，其中只能存放一个字符。
字符值是以 ASCII 码的形式存放在变量的内存单元之中的。

字符变量的存储方式

每个字符变量被分配一个字节的内存空间，其中只能存放一个字符。字符值是以 ASCII 码的形式存放在变量的内存单元之中的。

```
char a = 'x';  
char b = 'y';
```

因为 'x' 和 'y' 的 ASCII 码为 120 和 121，故字符变量 a 和 b 在内存中的存储方式为：

a:

0	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---

b:

0	1	1	1	1	0	0	1
---	---	---	---	---	---	---	---

字符变量的存储方式

由字符变量的存储方式可看出，可以把字符量看成是整型量。事实上，

- C/C++ 允许对整型变量赋以字符值，也允许对字符型变量赋以整型值。
- 在输出时，允许把字符变量按整型量输出，也允许把整型量按字符量输出。
- 整型量占两个字节，字符量占一个字节。当整型量按字符型量处理时，只有低八位字节参与处理。

打印字符

使用printf() 打印字符,

- 若使用 %c, 将打印字符本身;
- 若使用 %d, 将打印对应的 ASCII 码。

打印字符

使用printf() 打印字符,

- 若使用 %c, 将打印字符本身;
- 若使用 %d, 将打印对应的 ASCII 码。

```
// charcode.c: displays code number for a char
#include <stdio.h>
int main(void)
{
    char c;
    printf("Please input a character:\n");
    scanf("%c", &c);
    printf("the code for %c is %d\n", c, c);
    return 0;
}
```

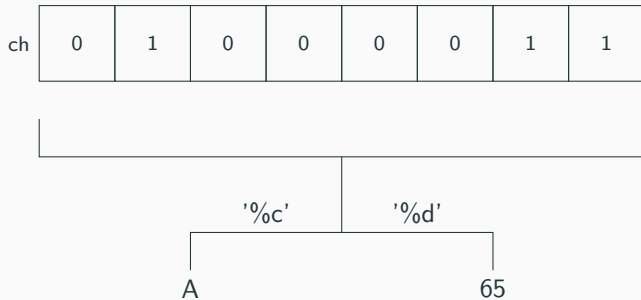
打印字符

使用printf() 打印字符,

- 若使用 %c, 将打印字符本身;
- 若使用 %d, 将打印对应的 ASCII 码。

```
// charcode.c: displays code number for a char
#include <stdio.h>
int main(void)
{
    char c;
    printf("Please input a character:\n");
    scanf("%c", &c);
    printf("the code for %c is %d\n", c, c);
    return 0;
}
```

```
$ gcc charcode.c
$ ./a.out
Please input a character:
A
```



浮点型数据

不同计数法

数字	科学计数法	指数计数法
1 000 000 000	1.0×10^9	1.0e9
123 000	1.23×10^5	1.23e5
322.56	3.2256×10^2	3.2256e2
0.000 056	5.6×10^{-5}	5.6e-5

浮点数的存储方式

C 的浮点数包括

- `float` (单精度)
- `double` (双精度)
- `long double`

浮点数的存储方式

C 标准规定,

- `float` 数据占 32 位, 至少能表示 6 位有效数字, 取值范围至少为 10^{-37} 到 10^{+37} 。
- `double` 数据占 64 位, 至少能表示 10 位有效数字, 最小取值范围和 `float` 相同。
- C 只保证 `long double` 类型至少同 `double` 类型一样精确。

两者在存储方式上都遵从 IEEE 规范, `float` 遵从 IEEE R32.24, 而 `double` 遵从 IEEE R64.53。

浮点型数据

```
// float.c
#include <stdio.h>
#include <float.h>
int main(void)
{
    printf("----float---\n");
    printf(" range: %e ~ %e\n", FLT_MIN, FLT_MAX);
    printf(" epsilon: %e\n", FLT_EPSILON);
    printf(" size:  %lu bytes\n\n", sizeof(float));
    printf("----double---\n");
    printf(" range: %e ~ %e\n", DBL_MIN, DBL_MAX);
    printf(" epsilon: %e\n", DBL_EPSILON);
    printf(" size:  %lu bytes\n\n", sizeof(double));
    printf("----long double---\n");
    printf(" range: %Le ~ %Le\n", LDBL_MIN, LDBL_MAX);
    printf(" epsilon: %Le\n", LDBL_EPSILON);
    printf(" size:  %lu bytes\n", sizeof(long double));
    return 0;
}
```

浮点型数据

----float---

range: $1.175494\text{e-}38 \sim 3.402823\text{e+}38$

epsilon: $1.192093\text{e-}07$

size: 4 bytes

----double---

range: $2.225074\text{e-}308 \sim 1.797693\text{e+}308$

epsilon: $2.220446\text{e-}16$

size: 8 bytes

----long double---

range: $3.362103\text{e-}4932 \sim 1.189731\text{e+}4932$

epsilon: $1.084202\text{e-}19$

size: 16 bytes

浮点型数据 i

```
// float.cpp
#include <iostream>
#include <limits>
using namespace std;
int main(void)
{
    cout << "----float----" << endl
         << "range: " << numeric_limits<float>::min()
         << " ~ "      << numeric_limits<float>::max() << endl
         << "prec: "   << numeric_limits<float>::epsilon() <<
         endl
         << "size: "   << sizeof(float) << endl << endl;

    cout << "----double----" << endl
         << "range: " << numeric_limits<double>::min()
         << " ~ "      << numeric_limits<double>::max() << endl
         << "prec: "   << numeric_limits<double>::epsilon() <<
         endl
```

```
<< "size: " << sizeof(double) << endl;

cout << "----long double----" << endl
    << "range: " << numeric_limits<long double>::min()
    << " ~ "      << numeric_limits<long double>::max() <<
    endl
    << "prec: "   << numeric_limits<long double>::epsilon
    () << endl
    << "size: "   << sizeof(long double) << endl;
return 0;
}
```

浮点型数据

----float----

range: 1.17549e-38 ~ 3.40282e+38

prec: 1.19209e-07

size: 4

----double----

range: 2.22507e-308 ~ 1.79769e+308

prec: 2.22045e-16

size: 8

----long double----

range: 3.3621e-4932 ~ 1.18973e+4932

prec: 1.0842e-19

size: 16

浮点数的存储方式

无论是单精度还是双精度在存储中都分为三个部分：

1. 符号位 (sign): 0 代表正, 1 代表负
2. 指数位 (exponent): 用于存储科学计数法中的指数数据, 并采用移位存储
3. 尾数部分 (mantissa)

浮点数的存储方式

R32.24 和 R64.53 的存储方式都用科学计数法来存储数据。

例

因 $(120.5)_{10} = (1110110.1)_2$ ，故其二进制科学计数法表示为

$$(1.1101101 \times 2^6)_2$$

浮点数的存储方式

R32.24 和 R64.53 的存储方式都用科学计数法来存储数据。

例

因 $(120.5)_{10} = (1110110.1)_2$ ，故其二进制科学计数法表示为

$$(1.1101101 \times 2^6)_2$$

注

任何一个数的二进制科学计数法表示都为

$$1. * * * * \times 2^n.$$

因第一位都是 1，不存储，故 23 位的尾数部分，可表示的精度却是 24 位。

浮点数的存储方式

问题

那么 24 位能精确到小数点后几位呢？

因

$$(9)_{10} = (1001)_2,$$

故四位能精确十进制中的 1 位小数点，24 位就能使 `float` 能精确表示到小数点后 6 位。

浮点数的存储方式

关于指数部分，因指数可正可负，8 位的指数位能表示的指数范围应该是 $-127 \sim 128$ 。所以指数部分的存储采用移位存储，存储的数据为“原数据 +127”。

浮点数的存储方式

以下观察 8.25 的存储方式：

1. 8.25 用二进制的科学计数法表示为 1.0001×2^3

浮点数的存储方式

以下观察 8.25 的存储方式：

1. 8.25 用二进制的科学计数法表示为 1.0001×2^3

2. 符号位为：0，表示为正

指数位为： $3 + 127 = 130 = (1000\ 0010)_2$

尾数部分为： $(0001)_2$

浮点数的存储方式

以下观察 8.25 的存储方式：

1. 8.25 用二进制的科学计数法表示为 1.0001×2^3

2. 符号位为：0，表示为正

指数位为： $3 + 127 = 130 = (1000\ 0010)_2$

尾数部分为： $(0001)_2$

3. 存储方式如下：



$130 = 3 + 127$

000 1

1.0001×2^3

浮点数的存储方式

以下观察 120.5 的存储方式：

1. 120.5 用二进制的科学计数法表示为 1.1101101×2^6

浮点数的存储方式

以下观察 120.5 的存储方式：

1. 120.5 用二进制的科学计数法表示为 1.1101101×2^6

2. 符号位为：0，表示为正

指数位为： $6 + 127 = 133 = (1000\ 0101)_2$

尾数部分为： $(1101101)_2$

浮点数的存储方式

以下观察 120.5 的存储方式：

1. 120.5 用二进制的科学计数法表示为 1.1101101×2^6

2. 符号位为：0，表示为正

指数位为： $6 + 127 = 133 = (1000\ 0101)_2$

尾数部分为： $(1101101)_2$

3. 存储方式如下：

0	1000 0010	110 1101 0000 0000 0000 0000
---	-----------	------------------------------

$$133 = 6 + 127$$

110 1101

$$1.1101101 \times 2^6$$

浮点数的存储方式

问题

给出内存中一段数据

01000010111001011000000000000000,

并告诉你是单精度存储，如何得到该数据的十进制数值？

浮点数的存储方式

1. 将数据分段

0	1000 0101	110 0101 1000 0000 0000 0000
---	-----------	------------------------------

浮点数的存储方式

1. 将数据分段

0	1000 0101	110 0101 1000 0000 0000 0000
---	-----------	------------------------------

2. 符号位为 0，故为正；因 $(1000\ 0101)_2 = 133$ ，故指数为 $133 - 127 = 6$ ；故该数据为

$$(1.1100101 \times 2^6)_2 = (1110010.1)_2 = 114.5.$$

浮点数的存储方式

```
// float2double.c:
#include <stdio.h>
int main(void)
{
    float f1 = 2.2, f2 = 2.25;
    double g1, g2;
    g1 = (double) f1;
    g2 = (double) f2;
    printf("g1 = %.13f, g2 = %.13f\n", g1, g2);
    return 0;
}
```

浮点数的存储方式

```
// float2double.c:
#include <stdio.h>
int main(void)
{
    float f1 = 2.2, f2 = 2.25;
    double g1, g2;
    g1 = (double) f1;
    g2 = (double) f2;
    printf("g1 = %.13f, g2 = %.13f\n", g1, g2);
    return 0;
}
```

```
g1 = 2.2000000476837, g2 = 2.25000000000000
```

问题

为什么在单精度转换为双精度时，2.2 的数值发生了改变而 2.25 却没有改变？

浮点数的存储方式

问题

为什么在单精度转换为双精度时，2.2 的数值发生了改变而 2.25 却没有改变？

- 2.25 的单精度存储方式为

0 1000 0001 001 0000 0000 0000 0000

而双精度存储方式为

0 1000 0001 001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

故在强制转换时，数值没有改变。

浮点数的存储方式

将十进制小数转换为二进制的方法：**将小数乘 2，取整数部分。**

$$\begin{array}{r} 0.2 \\ \times 2 \\ \hline 0.4 \end{array} \quad \text{取整数位0}$$
$$\begin{array}{r} 0.4 \\ \times 2 \\ \hline 0.8 \end{array} \quad \text{取整数位0}$$
$$\begin{array}{r} 0.8 \\ \times 2 \\ \hline 1.6 \end{array} \quad \text{取整数位1}$$
$$\begin{array}{r} 1.6 \\ \times 2 \\ \hline 1.2 \end{array} \quad \text{取整数位1}$$
$$\begin{array}{r} 1.2 \\ \times 2 \\ \hline 0.4 \end{array} \quad \text{取整数位0}$$

2.2 的二进制表示为一个无限循环的排列：

```
10.0011 0011 0011 0011 0011...
```

浮点数的存储方式

2.2 的单精度存储方式为

```
0  1000 0001  000 1100 1100 1100 1100
```

而双精度存储方式为

```
0  1000 0001  000 1100 1100 1100 1100 1100 1100 1100 1100 1100 1100 1100 1100 1100
```

故在强制转换时，数值会发生改变。

浮点型常量

基本形式为：包含小数点的一个带符号的数字序列，接着是字母 e 或 E，然后是代表 10 的指数的一个有符号值。如

`-1.56E+12, 2.87e-3`

浮点型常量

- 可以省略正号

```
+2.87e-3  
2.87e-3
```

- 可以没有小数点或指数部分，但不能同时没有

```
2E5  
19.28  
2          // is an integer
```

- 可以省略小数部分或整数部分，但不能同时省略。

```
3.e12  
.45E-5
```

浮点型常量

- 浮点型常量中不要使用空格

```
1.56 E+12 // wrong
```

- 默认情况下，编译器把浮点型常量当做 `double` 类型。

设 `some` 为一 `float` 变量，

```
some = 4.0 * 2.0;
```

则 `4.0` 和 `2.0` 会被存储为 `double` 类型，用 64 位存储。

注意：乘积运算使用双精度，结果被截取为正常的 `float` 长度，能保证计算精度，但会减慢程序的运行。

浮点型常量

- 可以加后缀 `f` 或 `F` 使编译器把浮点常量当做 `float` 型

```
2.3f
```

```
3.4e9F
```

- 可以加后缀 `l` 或 `L` 使编译器把浮点常量当做 `long double` 型（由于字母 `l` 和数字 `1` 容易混淆，建议使用后缀 `L`）

```
54.3l
```

```
4.3E9L
```

浮点型常量

```
// float1.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.1;
    if (x == 0.1)          printf("IF\n");
    else if (x == 0.1f)    printf("ELSE IF\n");
    else                   printf("ELSE\n");
}
```

浮点型常量

```
// float1.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.1;
    if (x == 0.1)          printf("IF\n");
    else if (x == 0.1f)    printf("ELSE IF\n");
    else                   printf("ELSE\n");
}
```

ELSE IF

浮点型常量

```
// float1.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.1;
    if (x == 0.1)          printf("IF\n");
    else if (x == 0.1f)    printf("ELSE IF\n");
    else                   printf("ELSE\n");
}
```

ELSE IF

解释

x 为单精度浮点数 0.1，常量 0.1f 表示单精度浮点数 0.1，常量 0.1 表示双精度浮点数 0.1，而根据浮点数的存储方式可知 $0.1 \neq 0.1f$ ，故 $x == 0.1f$ 。

浮点型常量

```
// float2.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.1;
    printf("%lu %lu %lu\n",
        sizeof(x), sizeof(0.1), sizeof(0.1f));
    return 0;
}
```

浮点型常量

```
// float2.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.1;
    printf("%lu %lu %lu\n",
           sizeof(x), sizeof(0.1), sizeof(0.1f));
    return 0;
}
```

4 8 4

浮点型常量

```
// float2.c: What is the output?  
#include<stdio.h>  
int main(void)  
{  
    float x = 0.1;  
    printf("%lu %lu %lu\n",  
           sizeof(x), sizeof(0.1), sizeof(0.1f));  
    return 0;  
}
```

4 8 4

解释

原因同上。

浮点型常量

```
// float3.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.5;
    if (x == 0.5)          printf("IF\n");
    else if (x == 0.5f)    printf("ELSE IF\n");
    else                   printf("ELSE\n");
}
```

浮点型常量

```
// float3.c: What is the output?
#include<stdio.h>
int main(void)
{
    float x = 0.5;
    if (x == 0.5)          printf("IF\n");
    else if (x == 0.5f)    printf("ELSE IF\n");
    else                   printf("ELSE\n");
}
```

IF

浮点型常量

```
// float3.c: What is the output?  
#include <stdio.h>  
int main(void)  
{  
    float x = 0.5;  
    if (x == 0.5)          printf("IF\n");  
    else if (x == 0.5f)    printf("ELSE IF\n");  
    else                   printf("ELSE\n");  
}
```

IF

解释

因 x 为单精度浮点数 0.5，常量 0.5f 表示单精度浮点数 0.5，常量 0.5 表示双精度浮点数 0.5，但是根据浮点数的存储方式可知 $0.5 == 0.5f$ ，故条件 $x == 0.5$ 先满足，从而执行第一个分支。

打印浮点型数据

- 使用格式说明符 `%f` 打印 `float`和 `double`型数据，`%e` 打印指数计数的数字。
- 使用格式说明符 `%Lf` 、`%Le` 打印 `long double` 型数据。

打印浮点型数据

```
// showf_pt.c:
#include <stdio.h>
int main(void)
{
    float a = 32000.;
    double b = 2.14e9;
    long double c = 5.32e-5;
    printf("%f can be written as %e\n", a, a);
    printf("%f can be written as %e\n", b, b);
    printf("%Lf can be written as %Le\n", c, c);
    return 0;
}
```

打印浮点型数据

```
// showf_pt.c:
#include <stdio.h>
int main(void)
{
    float a = 32000.;
    double b = 2.14e9;
    long double c = 5.32e-5;
    printf("%f can be written as %e\n", a, a);
    printf("%f can be written as %e\n", b, b);
    printf("%Lf can be written as %Le\n", c, c);
    return 0;
}
```

```
32000.000000 can be written as 3.200000e+04
2140000000.000000 can be written as 2.140000e+09
0.000053 can be written as 5.320000e-05
```

浮点型数据的溢出

```
#include <stdio.h>
int main(void)
{
    float toobig = 1e39;
    printf("toobig = %f\n",toobig);
    return 0;
}
```

浮点型数据的溢出

```
#include <stdio.h>
int main(void)
{
    float toobig = 1e39;
    printf("toobig = %f\n",toobig);
    return 0;
}
```

```
toobig = inf
```

浮点型数据的溢出

```
#include <stdio.h>
int main(void)
{
    float toobig = 1e39;
    printf("toobig = %f\n", toobig);
    return 0;
}
```

```
toobig = inf
```

注

当浮点数超出表示范围时，会发生上溢（overflow），C 会赋予一个代表无穷大的特殊值，即 `inf`。

浮点型数据舍入误差

```
#include <stdio.h>
int main(void)
{
    float a,b;
    a = 2.e20 + 1;
    b = a - 2.e20;
    printf("b = %f\n", b);
    return 0;
}
```

浮点型数据舍入误差

```
#include <stdio.h>
int main(void)
{
    float a,b;
    a = 2.e20 + 1;
    b = a - 2.e20;
    printf("b = %f\n", b);
    return 0;
}
```

```
b = 4008175468544.000000
```

浮点型数据舍入误差

```
#include <stdio.h>
int main(void)
{
    float a,b;
    a = 2.e20 + 1;
    b = a - 2.e20;
    printf("b = %f\n", b);
    return 0;
}
```

```
b = 4008175468544.000000
```

解释

数字 $2.0e20$ 加 1，变化的是第 21 位，要计算正确，至少需要存储 21 位的数字，而 `float` 型数字只有 6、7 位有效数字，故该计算注定不正确。