C 数据

张晓平

武汉大学数学与统计学院

homepage: xpzhang.me

2018 年 3 月 6 日

目录

例子

编制程序，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
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

- 新的变量声明：

```
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的位置，指定将输入值赋给变量fah。

例子

- 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。

常量

定义

常量
在程序执行过程中，其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’

符号常量

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。

符号常量

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 语言中，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

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，称为宏定义命令。
- 功能是把该标识符定义为其后的常量值。
- 一经定义，以后在程序中所有出现该标识符的地方均代之以该常量值。

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

使用符号常量的好处是：

- 含义清楚；
- 能做到“一改全改”。

变量

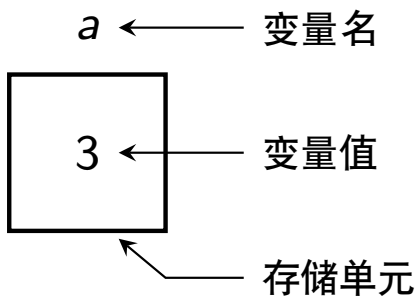
定义

变量
在程序执行过程中，其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。

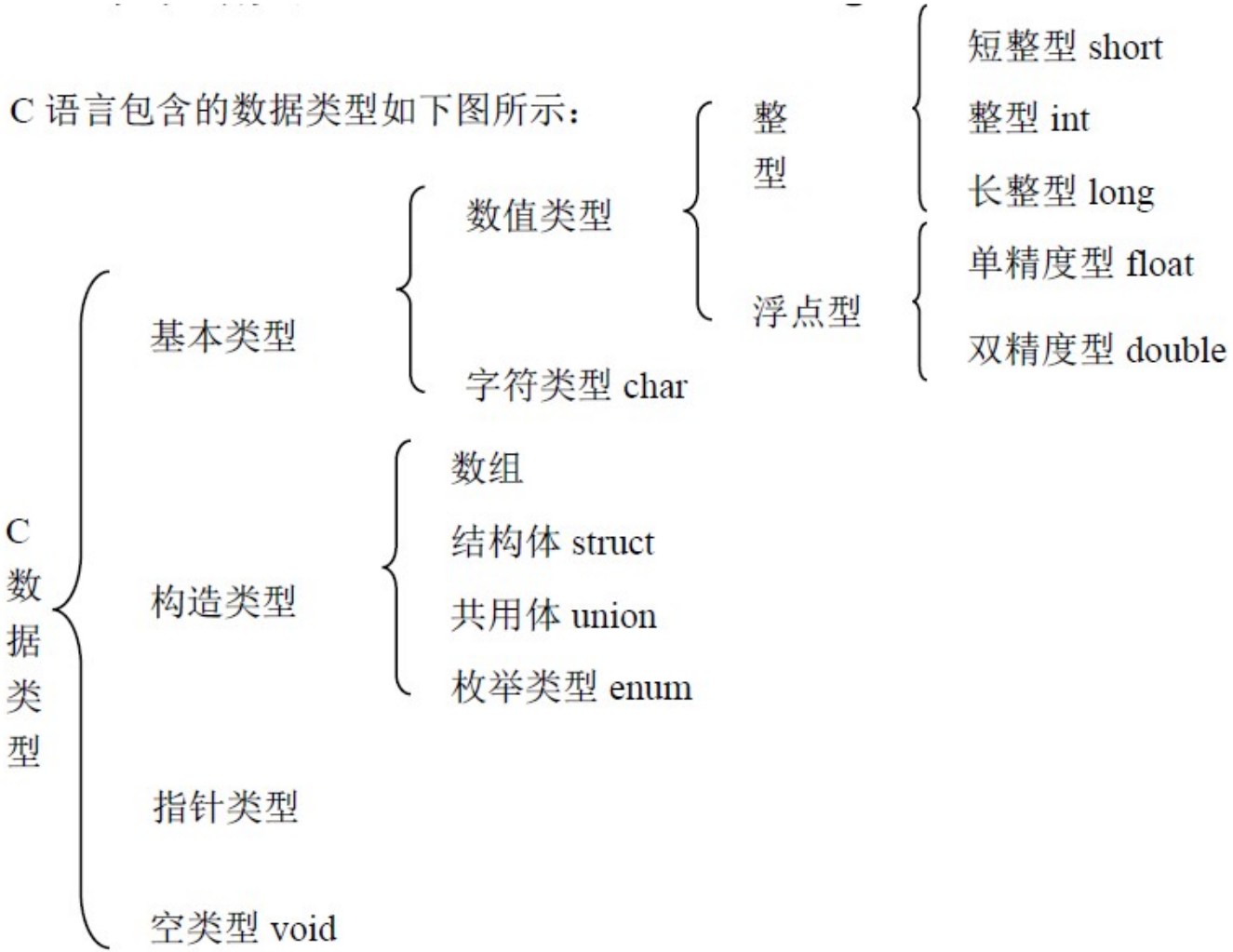
变量

- 一般放在函数体的开头部分。
- 要区分变量名和变量值是两个不同的概念。



数据类型

C 语言包含的数据类型如下图所示：



数据类型

- 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

- int提供基本整型，long、short、unsigned和signed为其变种。
- char用于表示字母及其他字符（如#，$，%,*等），也可表示小的整数。
- float、double和long double表示浮点型数。
- _Bool表示布尔值（true 和 false）。
- _Complex和_Imaginary分别表示复数和虚数。

这些类型按其存储方式被分为两类：整型和浮点型。

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: 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: 负数-10 的存储方式

int型

int型表示有符号整数，其取值范围依赖于系统，可通过以下代码查看。

```
1
2 #include<stdio.h>
3 #include<limits.h>
4 int main(void)
5 {
6   printf("range of int is %d ~ %d\n", INT_MIN,
    INT_MAX);
7   printf("sizeof int = %lu bytes\n", sizeof(int));
8   return 0;
9 }
```

int型

```
$ gcc int_max_min.c
$ ./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;
```

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。

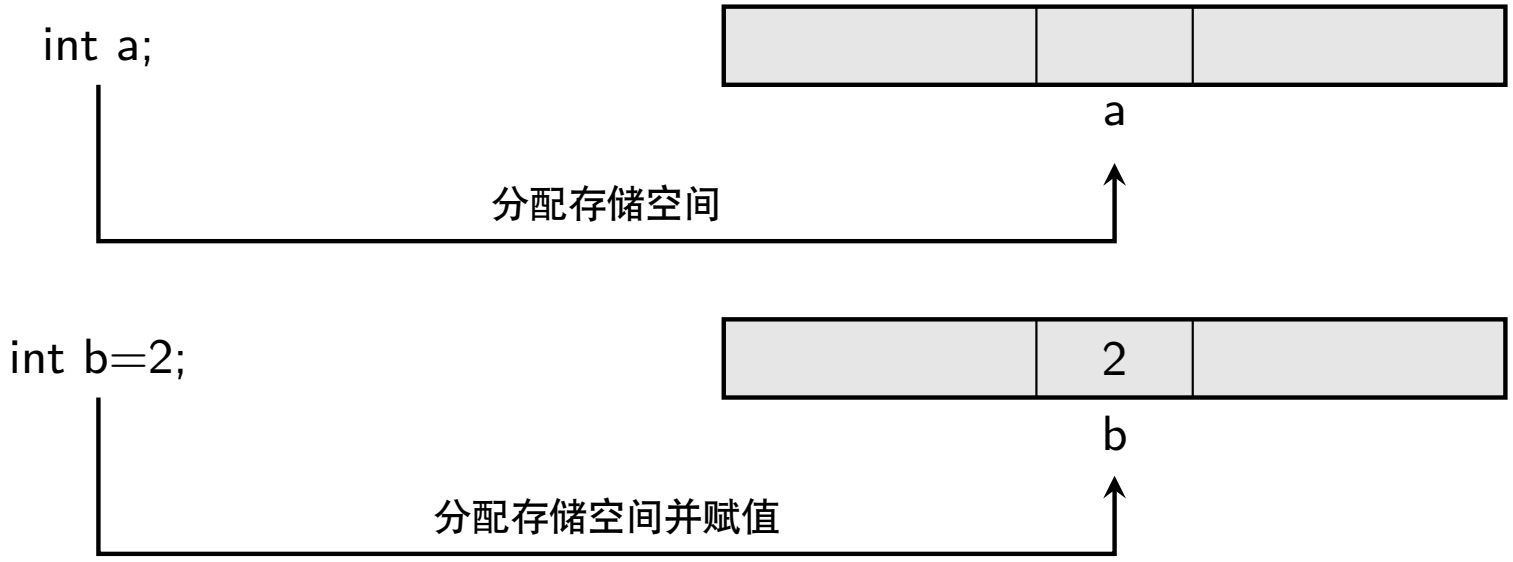


图: 定义和初始化变量

int值的打印

```
1
2  #include<stdio.h>
3  int main(void)
4  {
5     int a = 10;
6     int b = 2;
7     printf("Doing it right: ");
8     printf("%d - %d = %d\n", a, b, a-b);
9     printf("Doing it wrong: ");
10    printf("%d - %d = %d\n", a);
11    return 0;
12 }
```

int值的打印

```
1
2  #include<stdio.h>
3  int main(void)
4  {
5     int a = 10;
6     int b = 2;
7     printf("Doing it right: ");
8     printf("%d - %d = %d\n", a, b, a-b);
9     printf("Doing it wrong: ");
10    printf("%d - %d = %d\n", a);
11    return 0;
12 }
```

```
$ gcc print1.c
$ ./a.out
Doing it right: 10 - 2 = 8
Doing it wrong: 10 - 73832 = 771
```

int值的打印

在第二次调用 print 函数时，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 函数时，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
2 #include <stdio.h>
3 int main(void)
4 {
5   int x = 100;
6   printf("dec = %d; octal = %o; hex = %X\n", x, x, x)
    ;
7   printf("dec = %d; octal = %#o; hex = %#X\n", x, x,
    x);
8   return 0;
9 }
```

打印八进制数和十六进制数

```
1
2 #include <stdio.h>
3 int main(void)
4 {
5   int x = 100;
6   printf("dec = %d; octal = %o; hex = %X\n", x, x, x)
    ;
7   printf("dec = %d; octal = %#o; hex = %#X\n", x, x,
    x);
8   return 0;
9 }
```

```
$ gcc bases.c
$ ./a.out
dec = 100; octal = 144; hex = 64
dec = 100; octal = 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 提供 3 个附属关键字修饰`int`: `short`、`long`和`unsigned`。

其他整数类型

类型	含义	占位符
`short` (`int`)	用于仅需小数值的场合	`%hd`, `%ho`, `%hx`
`long` (`int`)	用于使用大数值的场合	`%ld`, `%lo`, `%lx`
`long long` (`int`)	用于使用更大数值的场合 (C99 标准)	`%lld`, `%llo`, `%llx`

其他整数类型

类型	含义	占位符
`unsigned (int)`	用于只使用非负值的场合。16 位的取值范围是 0-65535。	%u
`unsigned long (int)`	(C90 标准)	%lu
`unsigned long long (int)`	(C99 标准)	%llu

其他整数类型

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。

为什么会出现多种整数类型？

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。

- 有些 CPU 的自然字大小，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。

整数的上溢

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

```
1  #include <stdio.h>
2  #include <limits.h>
3  int main(void)
4  {
5     int i = INT_MAX;
6     unsigned int j = UINT_MAX;
7     printf("i = %d, i+1 = %d, i+2 = %d\n",
8        i, i+1, i+2);
9     printf("j = %u, j+1 = %u, j+2 = %u\n",
10       j, j+1, j+2);
11    return 0;
12 }
```

整数的上溢

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

整数的上溢

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。

注意：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，不能参与运算。

声明字符变量

字符变量的类型说明符是`char`。字符变量的声明与整型变量相同，如：

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

字符常量及其初始化

可以使用以下初始化语句将字符 A 赋给 grade：

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

字符常量及其初始化

```
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
```

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。

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，因此只能存放一个字符。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

```
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 语言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

printf() 使用格式说明符%c打印一个字符。若用格式说明符%d打印字符型变量，将输出一个整数。

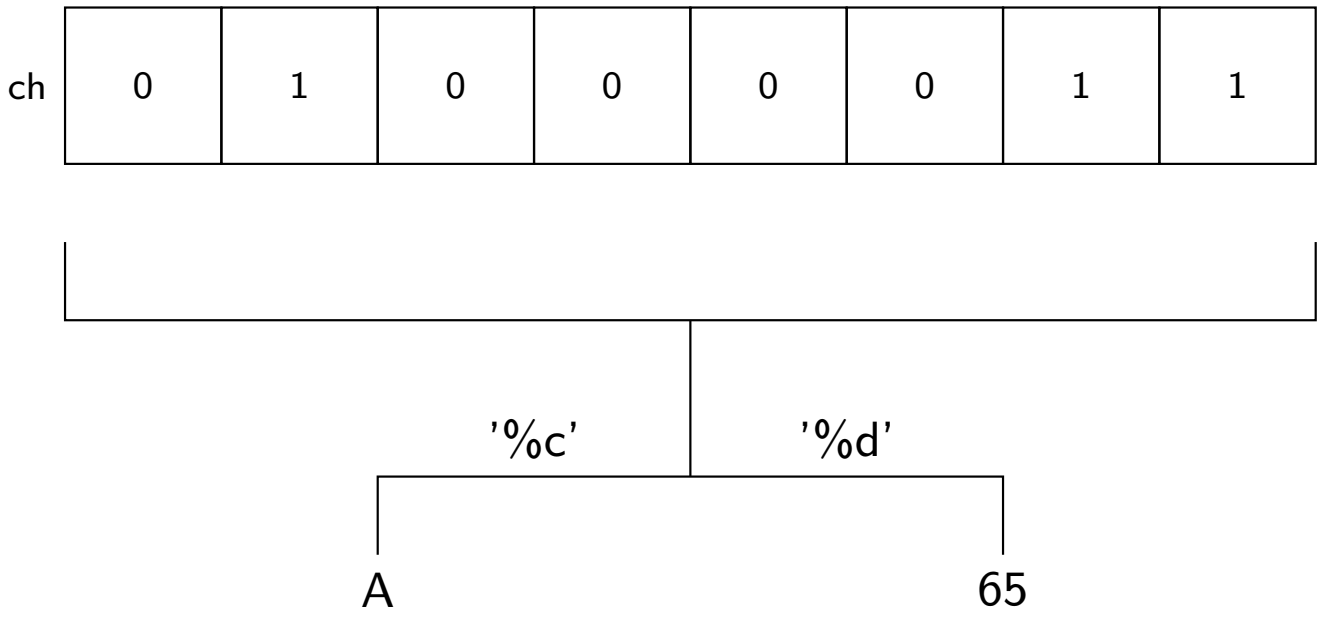
打印字符

```
1
2  #include <stdio.h>
3  int main(void)
4  {
5     char c;
6     printf("Please input a character:\n");
7     scanf("%c", &c);
8     printf("the code for %c is %d\n", c, c);
9     return 0;
10 }
```

打印字符

```
$ gcc charcode.c
$ ./a.out
Please input a character:
A
the code for A is 65
```

打印字符



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）和`long double`类型。

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

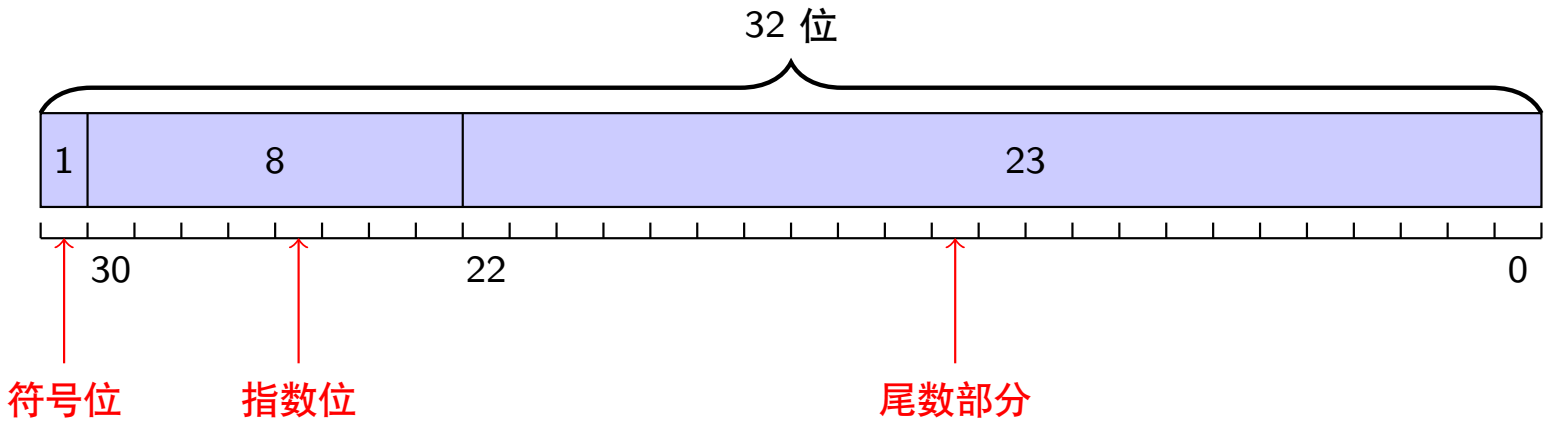


图: `float`数据的存储方式

浮点数的存储方式

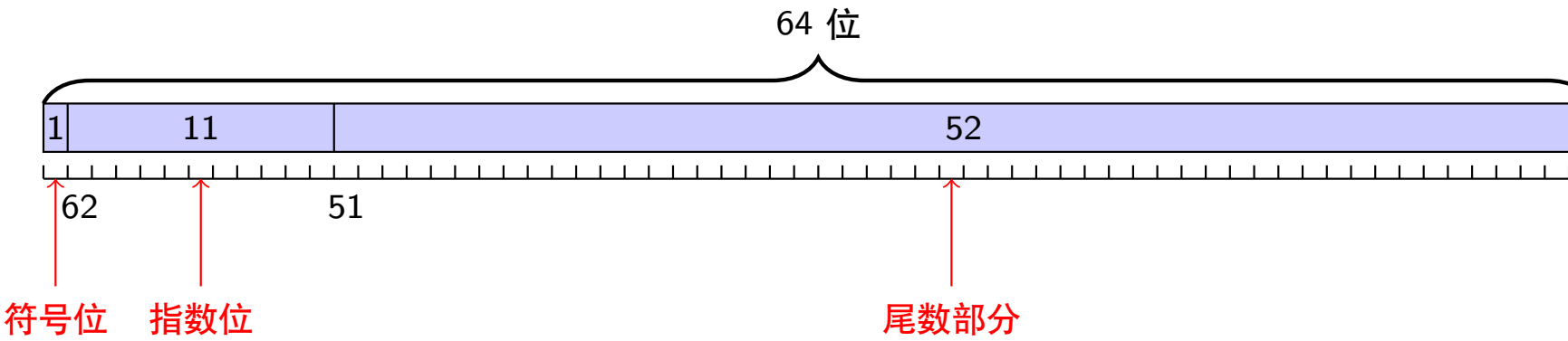


图: double数据的存储方式

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

120.5 的二进制表示为 *1110110.1*，二进制科学计数法表示为 1.1101101×2^6。

浮点数的存储方式

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-127 128。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：

0	1000 0010	000 1000 0000 0000 0000 0000

$130 = 3 + 127$ 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

1.1101101×2^6

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

阅读如下代码，观察运行结果：

```
1
2  #include <stdio.h>
3  int main(void)
4  {
5     float f1 = 2.2, f2 = 2.25;
6     double g1, g2;
7     g1 = (double) f1;
8     g2 = (double) f2;
9     printf("g1 = %.13f, g2 = %.13f\n", g1, g2);
10    return 0;
11 }
```

浮点数的存储方式

```
$ gcc float_double.c
$ ./a.out
g1 = 2.2000000476837, g2 = 2.2500000000000
```

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？

浮点数的存储方式

- 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。**

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

浮点数的存储方式

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
```

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
```

- 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
```

- 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
```

练习

```
1
2  #include<stdio.h>
3  int main(void)
4  {
5    float x = 0.1;
6    if (x == 0.1)
7      printf("IF\n");
8    else if (x == 0.1f)
9      printf("ELSE IF\n");
10   else
11     printf("ELSE\n");
12 }
```

练习

```
$ gcc float1.c
$ ./a.out
ELSE IF
```

练习

```
$ gcc float1.c
$ ./a.out
ELSE IF
```

因 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 0.1f 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 != 0.1f，故 x == 0.1f。

练习

```
1
2 #include<stdio.h>
3 int main(void)
4 {
5   float x = 0.1;
6   printf("%lu %lu %lu\n", sizeof(x), sizeof(0.1),
    sizeof(0.1f));
7   return 0;
8 }
```

练习

```
$ gcc float2.c
$ ./a.out
4 8 4
```

练习

```
$ gcc float2.c
$ ./a.out
4 8 4
```

原因同上。

练习

```
1
2  #include<stdio.h>
3  int main(void)
4  {
5    float x = 0.5;
6    if (x == 0.5)
7      printf("IF\n");
8    else if (x == 0.5f)
9      printf("ELSE IF\n");
10   else
11     printf("ELSE\n");
12 }
```

练习

```
$ gcc float2.c
$ ./a.out
IF
```

练习

```
$ gcc float2.c
$ ./a.out
IF
```

因 x 为单精度浮点数 0.5，常量 0.1f 表示单精度浮点数 0.5，常量 0.5f 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 打印指数计数法的数字。
- 使用格式说明符%Lf、%Le 打印long double型数据。

打印浮点型数据

```
1
2  #include <stdio.h>
3  int main(void)
4  {
5     float a = 32000.;
6     double b = 2.14e9;
7     long double c = 5.32e-5;
8     printf("%f can be written as %e\n", a, a);
9     printf("%f can be written as %e\n", b, b);
10    printf("%Lf can be written as %Le\n", c, c);
11    return 0;
12 }
```

打印浮点型数据

```
$ gcc showf_pt.c
$ ./a.out
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
```

浮点型数据的上溢和下溢


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

浮点型数据的上溢和下溢


```
$ gcc float_overflow.c
$ ./a.out
toobig = inf
```

浮点型数据的上溢和下溢

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

```
1 #include <stdio.h>
2 int main(void)
3 {
4   float a,b;
5   a = 2.e20 + 1;
6   b = a - 2.e20;
7   printf("b = %f\n", b);
8   return 0;
9 }
```

浮点型数据舍入误差

```
$ gcc float_err.c
$ ./a.out
b = 4008175468544.000000
```

浮点型数据舍入误差

为什么会出现如此奇怪的结果？原因是计算机缺乏足够的进行正确运算所需的十进制位数。

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。