C 语言入门教程

Grant Lee

2025 年 11 月 14 日

目录

4 从字面量（Literal）到数据类型

4.1 何为魔法数字？

首先来解释一下什么是字面量。在 C 语言中，字面量指的是直接出现在源代码中的常量值，无需标识符即可使用。说得大白话一点，就是代码中一个又一个数据。如代码 10 所示，代码中出现的一个又一个常量数字如 `42`、`'A'`、`3.1416`、`Hi` 就是字面量。

代码 10: C 语言中字面量的示例

```c
#include <stdio.h>

int main(void) {
    int x = 42;           // 42 是整型字面量
    char c = 'A';         // 'A' 是字符字面量
    double pi = 3.1416;   // 3.1416 是浮点字面量
    printf("%s\n", "Hi"); // "Hi" 是字符串字面量
    return 0;
}
```

在我们日常生活中，我们每天都会接触大量的数据，比如购物、计数。但我们从来不会纠结于数据类型或是字面量这些乱七八糟的东西。这是因为在日常生活中，我们普遍使用**十进制**，所以对于我们彼此交流的时候，从来都不会纠结于进制、类型问题。

但在计算机中可不是这样，首先在计算机底部存储数字使用二进制，而其他八进制、十六进制等进制也是常见的表示方式，再加上由于计算机存储数据的方式不同又诞生了浮点数、整型数、字符。所以当你冷不丁地给计算机抛出一个数字 42 时，计算机并不能理解这个数字到底是什么进制、类型，如代码 11 所示，计算机并不能理解这是十进制的 42，还是八进制的 `34`，还是十六进制的 `68`。它更无法理解这到底是 `int`、`double` 还是 `float`。

计算机此时只能按照默认处理规则进行处理，默认情况下这个 `42` 将被理解为十进制、`int` 类型的数据。但如果你的本意并不是这样，而是想表达其他进制或类型的数据，那么计算机就会误解你的意思。那么在工程场景下，往往就会引发严重的错误。

代码 11: 魔法数字实例

```
42
```

计算机遇到这种情况时，它是无法理解的，因为它不知道这个数字到底是什么进制、类型。这就好像一个麻瓜突然看到一个魔法师念咒语一样（来自哈利波特），它根本无法理解魔法师到底在说什么。对于这种情况，我们形象地称为魔法数字（Magic Number）。

4.2 定义与概念

在 **C 语言标准**（如 ISO/IEC 9899:2018, C17）中，“字面量”（*literal* 或 *literal constant*）是一个正式存在的术语。它指的是 **直接出现在源代码中的常量值**，无需标识符即可使用。

字面量类型	示例	说明
整型字面量 (integer literal)	`123`, `0x7B`, `010`	十进制、十六进制、八进制整数
浮点型字面量 (floating-point literal)	`3.14`, `1e-3`	双精度浮点数
字符字面量 (character constant)	`'A'`, `'\n'`	单个字符，本质为 `int` 类型
字符串字面量 (string literal)	`"Hello"`	以 `\0` 结尾的 `char` 数组

4.3 字面量的具体区分方式

C 语言里面通过“前缀（`prefix`）”和“后缀（`suffix`）”来对字面量（`literal`）进行类型和进制的区分。本节总结了 C 语言中通过前缀 （`Prefix`）与后缀 （`Suffix`）来区分字面量的进制与类型的方式。内容适用于 C89/C99/C11/C23 标准。

4.3.1 整数：通过前缀区分进制

整数字面量可使用前缀（`prefix`）确定进制，如 表 1 所示。

表 1: 整数前缀及其含义

前缀	进制	示例
(无前缀)	十进制	`123`
`0`	八进制	`0123` （十进制 83）
`0x` / `0X`	十六进制	`0x123` （十进制 291）
`0b` / `0B`（C23）	二进制	`0b1010` （十进制 10）

注意：

- 以 `0` 开头的整数默认按八进制解析（如 `0123` $\neq$ `123`）。所以在 C 语言中避免使用前导零，以免引起歧义。这算是一个非常经典的错误原因呢。
- C89/C99/C11 没有二进制前缀，直到 C23 才正式加入 `0b` 作为二进制前缀，有些编译器（如 GCC）较早支持。
- C 语言中只有十六进制浮点数可以带前缀，其他进制的浮点数一律不允许带前缀。（具体见后文）。

4.3.2 整数：通过后缀区分类型

整数字面量的类型可由后缀（Suffix）确定。表 2 汇总了常见后缀。

表 2: 整数类型后缀

后缀	类型	示例
u / U	unsigned int	123U
l / L	long int	123L
ll / LL	long long int	123LL
ul / UL	unsigned long etc	123UL, 123LU
组合使用	unsigned long long 等	123ULL, 123LLU

如代码 12所示，不同的类型其实意味着存储空间和取值范围的不同。所以当你需要表示更大范围的整数时，可以使用相应的后缀来指定类型。如果这个范围不匹配，编译器会报错或发出警告。

代码 12: 后缀示例

```
123U      // unsigned int              0 ~ 4294967295
123L      // long int                -2147483648 ~ 2147483647 (32位系统)
123UL     // unsigned long int         0 ~ 4294967295 (32位系统)
123LL     // long long int           -9223372036854775808 ~ 9223372036854775807
123LLU    // unsigned long long int    0 ~ 18446744073709551615

//以上给出的数据范围基于常见的32位和64位系统，
//实际范围基于编译器、CPU 架构和操作系统选择的数据模型而异。
```

注意

作为整数字面量后缀：llu 和 ull 完全等价，可以随便使用，没有任何问题。但在 printf 格式字符串中：必须用%llu，不能用%ull。所以我推荐大家统一记忆 llu。

4.4 浮点数：无前缀（Prefix 不可用）

浮点数 **不能使用前缀**来表示进制，不能像整数那样写成八进制、二进制或普通十六进制形式：

- `012.3` （非法，浮点不能为八进制）
- `0b1.01` （非法，C 不支持二进制小数）
- `0x3.14` （非法的普通十六进制形式）

因此：

浮点字面量不允许使用任何前缀。

4.5 浮点数的后缀（常用）

浮点字面量只能用后缀确定类型，如表 3 所示。

表 3: 浮点字面量后缀

后缀	类型	示例
(无后缀)	double	`3.14`
`f` / `F`	float	`3.14f`
`l` / `L`	long double	`3.14L`

示例：

```
3.14f        // float
2.718L       // long double
1e-3         // double
```

4.6 特殊情况：十六进制浮点（C99 起）

C99 引入了特殊语法的十六进制浮点常量，格式为：

`0x<hex>.<hex>p<指数>`

其中 `p` 的指数表示 2 。示例：

```
double x = 0x1.8p2;   // = (1 + 8/16) * 2^2 = 6.0
```

注意：

- 这是一种“特殊语法”，不是普通意义的“前缀决定进制”。
- 仅 C99 及之后标准支持。

4.7 总结

- 整数字面量：**前缀区分进制，后缀区分类型**。
- 浮点字面量：**无前缀，只能用后缀区分类型**。
- C99 起提供十六进制浮点字面量（如 `0x1.2p3`）。

名称	是否存在于语法中	是否有标识符	示例	说明
字面量 (*literal*)	是	否	`3.14`, `'A'`, `"abc"`	直接出现在代码中
常量 (*constant*)	是	是	`const int a = 10;`	存储在命名空间中

因此，`10` 是一个字面量，而在 `const int a = 10;` 中的 `a` 是一个常量。

4.8 延伸说明

- C 语言仅定义了四类基本字面量：整型、浮点型、字符和字符串。
- C++ 对字面量进行了扩展，例如用户自定义字面量 `42_km`。
- 在 C17 标准第 §6.4 节中，“literal” 一词被正式用于描述这些常量。

4.9 总结

在 C 语言标准中，“字面量”（literal）是正式术语，表示直接出现在源代码中的常量值，如 `10`、`'A'`、`"abc"` 等。它们是语法结构的一部分，而非常量变量。