# C++程序设计基础 第二章

# *前言*

*本文档由 @ItsJiale 创作，作者博客：https://jiale.domcer.com/，作者依据数学与大数据学院 2024 级大数据教学班的授课重点倾向编写而成。所有内容均为手动逐字录入，其中加上了不少自己的理解与思考，耗费近一周时间精心完成。*

*此文档旨在助力复习 C++ 程序设计基础，为后续学习数据结构筑牢根基。信计专业的同学，也可参考本文档规划复习内容。需注意，若个人学习过程中存在不同侧重点或对重难点的理解有差异，应以教材内容为准。倘若文档内容存在任何不妥之处，恳请各位读者批评指正。*

*By：ItsJiale*

*2025.4.8*

# 第2章 基本数据类型和表达式

## 2.1 C++语句基本元素

它们包括标识符、关键字、字面值常量、运算符以及标点符号

### 2.1.1 标识符

C++标识符由字母、数字和下画线组成，而且必须要以字母或下画线开头。

***注意：C++严格区分大小写***

### 2.1.2 关键字

关键字不能作为用户自定义的标识符名

## 2.2 基本数据类型

### 2.2.1 内置类型

常用的数据类型

| 类型 | 含义 | 字节（Visual C++） |
| --- | --- | --- |
| bool | 布尔型 | 1 |
| char | 字符型 | 1 |
| short | 短整型 | 2 |
| int | 整型 | 4 |
| long | 长整型 | 4 |
| long long | 双长整型 | 8 |
| float | 单精度浮点型 | 4 |
| double | 双精度浮点型 | 8 |
| long double | 扩展的精度浮点型 | 8 |

相关概念：

位（bit）是计算机存储数据的最小单位，指二进制中的一个位数，其值为0或1

字节（Byte）是计算机存储容量的基本单位，一个字节由8位二进制数组成，即8个比特位

如果想要表达无符号类型，需要在这些类型名前面添加unsigned修饰符，例如unsigned int

#### 1. 原码

原码是一种最简单的机器数表示法，最高位为符号位，0 表示正数，1 表示负数，其余位表示数值的绝对值。

以 8 位二进制数为例，十进制数 +5 的原码是 0000 0101， −5 的原码是 1000 0101。

#### 2. 反码

反码的表示规则是：正数的反码与原码相同；负数的反码是在原码的基础上，符号位不变，其余位按位取反。
以 8 位二进制数为例，十进制数 +5 的反码是 0000 0101， −5 的原码是 1000 0101，反码是 1111 1010 。

#### 3. 补码

补码的表示规则是：***正数的补码与原码相同***；***负数的补码是在反码的基础上加 1***。补码解决了原码在进行减法运算时的问题，在计算机中，有符号数通常用补码表示。
以 8 位二进制数为例，十进制数 +5 的补码是 0000 0101， −5 的原码是 1000 0101，反码是 1111 1010，补码是 1111 1011。

***反码***可以看作是从原码到补码转换过程中的***中间码***。

### 2.2.2 常量

#### 1. 整型常量

以0开头的数代表八进制数，八进制数由数字0~7组成。以0x或0X开头的数代表十六进制数，十六进制数由数字0~9和字母A~F（大小写均可以）组成。

#### 2. 实型常量

实型常量以小数或指数表示。指数形式由尾数、阶数和E或e组成

#### 3. 字符和字符串常量

用单引号引起来单个字符称为字符常量，由双引号引起来的零个或多个字符称为字符串常量

## 2.3 对象

#### 2.3.1 对象的定义和初始化

#### 1. 对象的定义

以类型说明符开头，紧跟一个或多个对象名，其中对象名以逗号分隔，并以分号结束

```
int age;
```

```
int age, score ;
```

#### 2. 对象的初始化

```
int age =18;
```

### 2.3.2 对象的声明

声明对象，需要利用关键字extern，而且不能提供初始值

```
extern int i ;
```

## 2.4 常量修饰符

### 2.4.1 const修饰符

不能对const修饰的对象进行改写

```
const int c =100;
```

## 2.5 表达式

### 2.5.1 基本知识

#### 1. 左值和右值

任何一个表达式，要么是左值，要么是右值。对于程序员来说，左值所在的内存空间的地址是可以获取（用取地址符&获取）的，但右值的地址是无法得到的。因此，左值对象既可以读又可以写，而对右值对象只能进行读操作，不能对它进行写操作。显然，常量都是右值，而由程序员定义的用来存放并能够改变值的对象是左值。

**2. 优先级和结合性**

通常一般都是先乘除后加减

## 2.5.2 算术运算符

| 运算符 | 功能 | 用法 |
| --- | --- | --- |
| + | 加法 | 3+2 |
| – | 加法 | 3–2 |
| * | 乘法 | 3*2 |
| / | 除法 | 3/2 |
| % | 取模 | 3%2 |

## 2.5.3 赋值运算符

赋值运算符“=”的功能是，把赋值符号右侧表达式的值写入赋值符号左侧的操作对象里，所以赋值运算符的左侧操作对象必须是一个支持写操作的左值。运算的结果是左侧操作对象本身，且是左值，表达式的值是左侧对象的值。

```
int i = 0 ,  j = i ;      //初始化而非赋值
```

```
i = 0 ;                     //赋值而非初始化
```

```
i+j =10;                  //错误：算术表达式为右值
```

## 2.5.4 自增和自减运算符

*后置 j = i ++ 先调用后自增*

*前置 j =++i  先自增后调用*

```
int a =10,b= 20 ;
```

```
b=a++;      //先把a的值赋给b  然后  a再自增
```

结果是b=10，a=11

## 2.5.5. 逻辑和关系运算符

在这些运算符中，①逻辑非的优先级最高–>②算术运算符–>③关系运算符–>④逻辑与–>⑤逻辑或–>⑥赋值运算

| 运算符 | 功能 | 结合性 | 用法 | 语义 |
| --- | --- | --- | --- | --- |
| ! | 逻辑非 | 右 | ！5 | 操作数的值为真，结果为假；反之结果为真 |
| < | 小于 | 左 | 5<4 | 左小于右，结果为真；反之为假 |
| <= | 小于等于 | 左 | 5<=4 | 左小于或等于右，结果为真，反之为假 |
| > | 大于 | 左 | 5>4 | 左大于右，结果为真；反之为假 |
| >= | 大于等于 | 左 | 5>=4 | 左大于或等于右，结果为真；反之为假 |
| == | 等于 | 左 | 5==4 | 左等于右，结果为真；反之为假 |
| != | 不等于 | 左 | 5!=4 | 左不等于右，结果为真；反之为假 |
| && | 逻辑与 | 左 | 5&&4 | 两个数均为真，结果为真；反之为假 |
| \|\| | 逻辑或 | 左 | 5\|\|4 | 两个数的值有一个或全为真，结果为真；反之为假 |

**关于逻辑非：**

*在大多数编程语言中，数值 0 被视为逻辑假，非 0 数值被视为逻辑真。所以这里的操作数 5，它的值在逻辑上被认为是“真”。**对于“!5”，因为操作数 5（逻辑上为真），经过逻辑非运算后，结果就变为逻辑假。***

***短路最值：***

对于&&来说，仅仅当左侧运算对象的值为真时，才计算右侧运算对象的值

对于||来说，仅仅当左侧运算对象的值为假时，才计算右侧运算对象的值

## 2.5.6 逗号运算符

逗号表达式求值的方法为依次***从左往右计算每个运算对象，表达式的结果为最右边的运算对象***

***即：整体取最右边的值赋给i***

```
int i , j;
```

```
i = (j=3 , j += 6 ,5+6);          // i 的值为11，j的值为9
```

## 2.5.7 条件运算符

条件运算符（？：）是唯一的一个三目运算符

*条件表达式 ？表达式1 : 表达式2*

- *若 `条件表达式` 的结果为 `true`，则整个三目运算符表达式的值是 `表达式1` 的值。*
- *若 `条件表达式` 的结果为 `false`，则整个三目运算符表达式的值是 `表达式2` 的值。*

## 2.5.8 sizeof 运算符

sizeof 运算符返回一个表达式或一个类型所占内存的***字节数***。一般格式为

`sizeof(type)`或

`siezeof(expr)`

### 2.5.9 位运算符

位运算符包括 ~（按位取反 0换1 、1换0）、<<（左移  左移比特位，用0来补空缺）、>>（右移）、&（位与）、|（位或）和^（位异或）

C++

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

int main() {
    int a = 5; // 二进制: 0101
    int b = 3; // 二进制: 0011

    // 按位与 (&)
    int andResult = a & b; // 二进制: 0001, 十进制: 1
    cout << "按位与结果: " << andResult << endl;

    // 按位或 (|)
    int orResult = a | b; // 二进制: 0111, 十进制: 7
    cout << "按位或结果: " << orResult << endl;

    // 按位异或 (^)
    int xorResult = a ^ b; // 二进制: 0110, 十进制: 6
    cout << "按位异或结果: " << xorResult << endl;

    // 按位取反 (~)
    int notResult = ~a; // 二进制补码运算结果, 通常为负数
    cout << "按位取反结果: " << notResult << endl;

    // 左移 (<<)
    int leftShiftResult = a << 2; // 二进制: 010100, 十进制: 20
    cout << "左移结果: " << leftShiftResult << endl;

    // 右移 (>>)
    int rightShiftResult = a >> 1; // 二进制: 0010, 十进制: 2
    cout << "右移结果: " << rightShiftResult << endl;

    return 0;
}
```

# 2.6 类型转换

## 2.6.1 隐式类型转换

一般情况下，比int类型小的整型类型提升为较大的整型类型 ***（小转大）***

## 2.6.2 显式类型转换

概念不好说明，以例子展示

```
float x =3.14;
```

```
int a =  (int) x;
```

结果是3