

□□□□□□

- □□__
- □□□□□_____
- □□__
- □□□□___

开始

本书用的是 ***visualstdio***软件进行编写代码与演示的

1.开头语句与固定格式

```
#include <iostream>
using namespace std;
int main()
{
  return 0;
}
```

如你所见，上面的这一段代码就是每一个 c++程序必须写的代码，下面我们将会一一解释其中的含义。

#include 这一行代码是用来引用头文件的，就像想要打扫房间就必须有工具一样，后面的: iostream就是c++所需的工具。

using namespace std;这一段先不用理解，后面的章节会讲到，先照抄就行。

int main () 这一段是最重要的，这个 main 函数里包括了程序开始时最先运行的代码，如果没有这一行，编译器会报错。代码在大括号中写。

return 0; 这一段是返回值，也是记住就行，后面会讲。

2.关于程序运行的顺序

几乎所有编译器的运行顺序是从上到下，从左到右。

在 c++中，每一个在函数内的语句结束后都要加分号。

输入输出流

在我们之前的章节里讲过，#include是用来引用头文件 iostream的，而我们今天讲的输入输出流就包括在了iostream里。

最基本的语法如下

```
//输出流
cout<<"Hello World"<<endl;
//输入流
int a;
cin>>a;
```

这就是最简单的一个输入输出流。

cout用来把后面的内容输出到控制台， cin用来把控制台用户输入的信息给到程序里。

1.cout

这里的双引号用来告诉编译器这是文字，不是函数名或变量名。

<<这个符号就像压缩机，把这一行的代码从右向左压成一个包，发给控制台。

endl的作用是换行。

2.cin

>>这个符号就跟 cout里的 <<作用相反。把控制台里用户输入的东西打包发给程序。

后面接的只能是变量名。

意思是把控制台内的用户输入的数字或字符传给程序内的变量，实现人机接口。

3.应用

现在让我们结合上面写一个简单的代码吧

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

```
int main()
  {
  cout<<"Hello World"<<endl;
  int a;
  cin>>a;
  cout<<a<<endl;
  return 0;
  }
```

变量

变量，顾名思义，就是可以改变的量，下面我们将会学习如何创建变量与改变变量。

1.变量

1.变量的定义

```
int a=0;
```

这里就是定义变量的方法。前面的 int为变量的类型，后面的为名称。这里将一个 int类型的变量初始化为了 0。

变量的类型有很多种。

有:

- 布尔： `bool`
- 字符： `char`
- 整数：
  - `short`
  - `int`
  - `long`
  - `long long`
- 浮点：
  - `float`
  - `double`
  - `long double`

bool类型的变量只能存储 true与 false

char类型的变量只能存储单个字符，例如 ‘a’或 ‘n’。但是单个中文不能存储，因为中文在程序里算两个字符。

整数类型分为很多种，但是都大差不差，存储都只能存整数，不能存小数与字符。这些类型的差别为存储容量的不同，每一个不同类型的整数变量存储数字的大小上线都不同。

浮点型是存储小数的，注意，在使用 float时做相等时无论什么数都要后面加个 .0例如: ***float a=0.0;***
即使这个数为整数。在存储 double时不仅要加 .0也要加 f例如 ***double a=0.0f;***

2.变量的赋值与运算

```
int a = 10;  // 初始化
a = 20;      // 赋值：把 20 存进 a
a = a + 5;   // 先算 a+5 得 25，再存回 a
```

局部变量如果只写 `int a;` 而没有初始化，它的值是不确定的，不要直接读取。建议总是初始化：

```
int a = 0;
double price = 0.0;
```

3 初始化 vs 赋值

```
int a = 1;  // 初始化：变量创建时给值
a = 2;      // 赋值：变量已经存在，改它的值
```

`const` 变量只能初始化，不能赋值：

加了 const的变量只能在初始化时进行赋值，这个代码便于后面防止误修改变量而设置很好用。

```
const int MAX = 100;
// MAX = 200; // 错误
```

---

2. 赋值号左边能放什么？

赋值号左边必须是“可以被修改的变量类东西”，通常叫左值。常见可以放：

```
int a;
a = 10;          // 变量
```

```
int arr[3];
arr[0] = 5;       // 数组元素

int* p = &a;
*p = 20;          // 解引用指针

int& r = a;
r = 30;           // 引用，改的是 a
```

不能放：

```
// 5 = a;         // 错误，5 不是变量
// a + 1 = 3;     // 错误，表达式不能赋值
// const int c=1; c=2; // 错误，const 不能改
```

注意：引用赋值不是“改绑”

```
int x = 1, y = 2;
int& r = x;  // r 是 x 的别名
r = y;       // 等价于 x = y，x 变成 2
             // r 仍然引用 x，不会改成引用 y
```

---

3. 赋值表达式本身也有值

 不只是一个动作，它本身也是一个表达式，值是赋值后的。所以可以链式赋值：

```
int a, b, c;
a = b = c = 0;  // 从右往左：c=0，然后 b=0，然后 a=0
```

也可以：

```
int a;
int b = (a = 3); // a 变成 3，b 也得到 3
```

但初学阶段建议少写复杂链式赋值，分步写更清楚。

---

4. 算术运算

C++ 常见算术运算符：

运算符	含义	例子
+	?	a + b
-	?	a - b
*	?	a * b
/	?	a / b
%	?????????	a % b
-a	??	-a
+a	??	+a

例子：

```
int a = 10, b = 3;
int sum = a + b;   // 13
int diff = a - b;  // 7
int prod = a * b;  // 30
int quo = a / b;   // 3
int rem = a % b;   // 1
```

4.1 整数除法会“截断”

如果 / 两边都是整数，结果还是整数，小数部分直接丢掉：

```
int a = 5, b = 2;
int c = a / b;       // 2，不是 2.5
double d = a / b;    // 2.0，因为先算 5/2 得 2，再转 double
```

想要小数除法，至少让一边是浮点数：

```
double d1 = 5.0 / 2;  // 2.5
double d2 = 5 / 2.0;  // 2.5
double d3 = static_cast<double>(5) / 2; // 2.5, 推荐写法
```

负数除法是 向零截断 :

```
-5 / 2   // -2
-5 % 2   // -1
```

C++11 起，取余结果的符号和被除数相同。

4.2 除零和取余零

```
int a = 10;
// int b = a / 0; // 错误/未定义，程序可能崩溃
// int c = a % 0; // 错误
```

取余 % 只能用于整数， 5.0 % 2 是错的。

4.3 溢出

```
int max = 2147483647;
// max + 1 的有符号溢出行为不可依赖
```

无符号整数溢出会按 2 的位数回绕:

```
unsigned char c = 255;
c++;  // c 变成 0
```

初学阶段记住: 不要让整数运算超出范围 。

4.4 浮点数有精度问题

```
double x = 0.1 + 0.2;
// x 不一定精确等于 0.3
```

不要直接用 == 比较两个计算出来的浮点数:

```cpp
#include <cmath>

double a = 0.1 + 0.2;
if (std::fabs(a - 0.3) < 1e-9) {
    // 认为它们足够接近
}
```

---

5. 复合赋值

复合赋值是“先运算，再存回左边”：

```cpp
a += b;   // 等价于 a = a + b
a -= b;   // a = a - b
a *= b;   // a = a * b
a /= b;   // a = a / b
a %= b;   // a = a % b
```

还有位运算版本：

```cpp
a &= b;
a |= b;
a ^= b;
a <<= 1;
a >>= 1;
```

例子：

```cpp
int a = 10;
a += 5;  // a 变成 15
a *= 2;  // a 变成 30
a /= 4;  // a 变成 7，整数除法
```

注意类型转换：

```cpp
int a = 10;
a += 2.5;  // 等价于 a = a + 2.5，先得 12.5，再转成 int，a 变成 12
```

---

6. 自增和自减

```
int i = 5;
i++;  // i 变成 6
i--;  // i 变成 5
```

前置和后置在单独使用时效果一样：

```
i++;
++i;
i--;
--i;
```

但在表达式里，它们有区别：

```
int i = 5;
int a = i++;  // 后置：先把旧值 5 给 a，然后 i 变成 6
// 此时 a == 5，i == 6

int b = ++i;  // 前置：i 先变成 7，再把 7 给 b
// 此时 b == 7，i == 7
```

记忆：

- ⬛：先用旧值，再加一
- ⬛：先加一，再用新值

初学阶段重要建议

不要在一个表达式里对同一个变量多次自增，例如：

```
// int a = i++ + ++i; // 不要这样写
```

这种写法结果不可靠，应拆成多步：

```
int i = 1;
int a = i;
i++;
int b = ++i;
```

7. 关系运算和逻辑运算

7.1 关系运算

运算符	含义
==	??
!=	???
<	??
>	??
>=	????
<=	????

关系运算的结果是 bool，即 true 或 false：

```
int a = 5, b = 3;
bool ok = a > b;  // true
```

千万不要把 == 写成 =：

```
int a = 5;
if (a == 0) { /* 比较 */ }

if (a = 0) {
    // 这是把 0 赋给 a，然后条件为 false
}
```

7.2 不要连写数学不等式

```
int x = 5;

// if (0 < x < 10) // 错误理解!
// 实际是 (0 < x) < 10, 即 true/false 再和 10 比较
```

应该写:

```
if (x > 0 && x < 10) {
    // ...
}
```

7.3 逻辑运算

运算符	含义	例子
&&	??????????	a > 0 && b > 0
||	?????????	a == 0 || b == 0
!	?????????	!ok

```
int a = 5, b = 3;
bool ok = (a > 0) && (b > 0);  // true
bool bad = (a == 0) || (b == 0); // false
```

7.4 短路求值

&& 左边为假时，右边不再计算; || 左边为真时，右边不再计算:

```
int a = 10, b = 0;

if (b != 0 && a / b > 2) {
    // b 为 0 时，左边 false，右边不会算，避免除零
}
```

但位运算 & 和 | 不短路:

```
// if (b != 0 & a / b > 2) // 不推荐这样写，右边仍可能计算
```

条件判断中，非零为真，零为假：

```
if (a)      // 等价于 if (a != 0)
if (!a)     // 等价于 if (a == 0)
```

---

8. 类型转换

8.1 隐式转换

混合类型运算时，通常会把范围小的转成范围大的：

```
int a = 5;
double b = 2.0;
double c = a + b;  // a 转成 double，结果是 7.0
```

赋值时，右边会转成左边的类型：

```
int a = 3.99;   // a 变成 3，小数被截掉
double d = 5;   // d 变成 5.0
bool b = 5;     // b 变成 true
int n = true;   // n 变成 1
```

`char` 参与算术通常提升为整数：

```
char c = 'A';
int x = c + 1;   // 'A' 的编码通常是 65，x 是 66
char d = c + 1;  // d 是 'B'
```

8.2 显式转换

推荐用 `static_cast`：

```
int a = 7, b = 2;
double r = static_cast<double>(a) / b; // 3.5
```

也可以：

```
double d = 3.9;
int n = static_cast<int>(d); // 3, 向零截断
```

不要依赖 C 风格转换的复杂规则，初学阶段用 [redacted] 更清楚。

8.3 有符号和无符号混用的坑

```
unsigned u = 1;
int n = -1;

if (n < u) {
    // 可能不是你想象的结果
}
```

因为 [redacted] 可能被转成很大的无符号数。初学阶段尽量少混用 [redacted] 和 [redacted]。

---

9. 位运算简介

位运算直接操作整数的二进制位：

运算符	含义
[redacted]	???
[redacted]	???
[redacted]	????
[redacted]	????
[redacted]	??
[redacted]	??

```
int x = 5;      // 二进制 0101
int y = x << 1; // 1010, 即 10
int z = x & 1;  // 判断奇偶: 1 表示奇数
```

初学阶段知道它们存在即可，注意：

- [redacted] 和 [redacted] 不同
- [redacted] 和 [redacted] 不同
- 移位位数不要超过类型宽度，不要移负数
- [redacted] 容易出错，应该写 [redacted]

---

10. 条件运算符和 [redacted]

条件运算符：

```
int a = 10, b = 20;
int max = (a > b) ? a : b;  // 如果 a>b 取 a，否则取 b
```

[redacted] 求类型或变量占多少字节：

```
int a = 0;
std::cout << sizeof(int) << '\n';
std::cout << sizeof a << '\n';
```

[redacted] 的结果类型是 [redacted]，通常是无符号整数。

---

11. 运算符优先级：不确定就加括号

常见优先级从高到低简化如下：

优先级	运算符	说明
?	[redacted]	???????????
	[redacted]	??
	[redacted]	???
	[redacted]	??

	[illegible]	??
	[illegible]	??
	[illegible]	??
	[illegible]	??
	[illegible]	???
	[illegible]	??
	[illegible]	???
	[illegible]	???
	[illegible]	??
	[illegible] ?	??
?	[illegible]	??

几个容易错的例子：

```
int a = 1, b = 2, c = 3;

int x = a + b * c;   // 先乘后加，x = 7
int y = (a + b) * c; // 先加后乘，y = 9

// if (a & b == c) 实际可能被理解成 a & (b == c)
// 应该写 if ((a & b) == c)

int m = a > b ? a : b; // 条件运算符
```

赋值运算符是 右结合 ：

```
a = b = c = 0;  // 等价于 a = (b = (c = 0))
```

优先级只决定 “怎么分组”，不决定 “先算左边还是右边”。例如 [illegible] 中，[illegible] 和 [illegible] 谁先执行并不保证。不要依赖复杂表达式的求值顺序。

12. 常见错误清单

1. 变量没初始化就使用

```
int a;
// std::cout << a; // 值不确定
```

2. 整数除法丢小数

```
double d = 5 / 2; // 2.0, 不是 2.5
```

3. 把 [redacted] 写成 [redacted]

```
if (a = 0) { } // 赋值，不是比较
```

4. 连写不等式

```
// if (0 < x < 10) // 错
if (x > 0 && x < 10)
```

5. 浮点数直接用 [redacted] 比较
6. 在复杂表达式里多次自增同一变量
7. 整数溢出、除零、取余零
8. 有符号和无符号混用
9. 优先级记错，不加括号
10. [redacted] 变量赋值、数组整体赋值

```
const int c = 1;
// c = 2; // 错

int x[3], y[3];
// x = y; // 错，数组不能整体赋值
```

---

13. 一个完整示例

```
#include <iostream>
#include <cmath>

int main() {
    int a = 10, b = 3;
```

```
    std::cout << "a + b = " << a + b << '\n';   // 13
    std::cout << "a - b = " << a - b << '\n';   // 7
    std::cout << "a * b = " << a * b << '\n';   // 30
    std::cout << "a / b = " << a / b << '\n';   // 3
    std::cout << "a % b = " << a % b << '\n';   // 1
    std::cout << "a / 2.0 = " << a / 2.0 << '\n'; // 5

    double r = static_cast<double>(a) / b;
    std::cout << "r = " << r << '\n';  // 3.33333

    int x = 5;
    int old = x++;  // old = 5□x = 6
    int now = ++x;  // x = 7□now = 7
    std::cout << old << ' ' << now << ' ' << x << '\n'; // 5 7 7

    a += 5;  // a = 15
    a *= 2;  // a = 30
    std::cout << "a = " << a << '\n'; // 30

    bool ok = (a > b) && (b != 0);
    int max = (a > b) ? a : b;
    std::cout << ok << ' ' << max << '\n'; // 1 30

    double p = 0.1 + 0.2;
    std::cout << std::boolalpha
              << (std::fabs(p - 0.3) < 1e-9) << '\n'; // true
}
```

14. □□□

1. □□□□□□□

```
int a = 7, b = 2;
std::cout << a / b << ' ' << a % b;
```

2. □□ □□□□

```
double d = 7 / 2;
```

答案: [redacted]。因为先做整数除法。改成 [redacted] 才是 [redacted]。

3. 下面 [redacted] 是多少?

```
int a = 3;
a += 2.6;
```

答案: [redacted]。先得 [redacted], 再转成 [redacted] 截断为 [redacted]。

4. 下面 [redacted]、[redacted]、[redacted] 分别是多少?

```
int i = 1;
int a = i++;
int b = ++i;
```

答案: [redacted], [redacted] 先变 [redacted]; 然后 [redacted] 让 [redacted] 变 [redacted], [redacted]。所以 [redacted]。

5. 下面写法为什么错?

```
if (0 < x < 10) { }
```

应改为:

```
if (x > 0 && x < 10) { }
```

□□□

1.□□□

1.1if□□

if□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

if□□□□□□□□□□□□□□□□□□ ……