


# 【C++语法】Lesson 1 讲义

---

## 【C++语法】Lesson 1 讲义

- 0. C++与算法竞赛介绍
- 1. 环境搭建
- 2. 第一个 C++ 程序——Hello World!
  - ※ 注释
- 3. 变量与数据类型
  - 3.1. 变量
    - 3.1.1. 变量命名规则
  - 3.2. 数据类型
    - 3.2.1. 整型
    - 3.2.2. 浮点型
      - ※ 输出固定位数的浮点数
    - 3.3.3. 字符型
      - ※ ASCII 码
      - ※ 转义字符
    - 3.2.4. 布尔型
    - 3.2.5. 空类型
      - ※ 非基本数据类型
      - ※ 进制数表示
      - ※ 强制类型转换
    - ※ 科学计数法
- 4. 运算符
  - 4.1. 算术运算符
  - 4.2. 赋值运算符
  - 4.3. 比较运算符
  - 4.4. 逻辑运算符
    - ※ 运算符优先级问题
- 5. 输入输出
  - 5.1. cin 和流提取运算符 (>>)
  - 5.2. cout 和流插入运算符 (<<)
  - 5.3. 格式化输出
- ※ 如何基于各大 OJ 有效刷题?
  - 读懂输入输出
  - 锁定数据类型
  - 选择合适算法
  - 技巧：基于数据范围反推时间复杂度
  - 技巧：基于提交反馈进行 Debug
  - 【例题】A+B Problem

本文档为程序设计算法协会C++语法课培训第一节课的讲义，制  
作人：漂至

## 0. C++与算法竞赛介绍

---

- C语言是咱们学院包括很多学院接触到的第一门编程语言，它是众多现代编程语言如C++、JAVA、Python的基础
- C++是C语言的扩展，支持面向对象编程，具有跨平台性，同时性能高效，广泛应用于游戏开发、嵌入式系统等领域
- 同时，在算法竞赛中C++也广受欢迎，主要原因是：
  1. C++性能高效，编译代码执行速度非常快，这也是为什么在很多编程竞赛中，题目对C/C++语言的时间限制会更短，通常为1s，而Java为2s

### 运行限制

语言	最大运行时间	最大运行内存
C++	1s	256M
C	1s	256M
Java	2s	256M
Python3	3s	256M
PyPy3	3s	256M
Go	3s	256M
JavaScript	3s	256M

2. C++中自带的标准模板库STL大大简化了代码编写，能够更快地实现复杂的数据结构，如：动态数组/向量vector、栈stack、队列queue、哈希表map、集合set等等
- Q1：没有C语言基础或编程语言基础是否能学习C++？
 

A1：完全可以的，我甚至推荐大家直接学习C++，据我了解，大二的同学下学期才会学习C语言，如果能通过培训对C++有所掌握，那么学习C语言会非常轻松
  - Q2：常见的算法比赛有哪些？
 

A2：对计算机专业的学生，可以参加的竞赛主要可分为：信奥算法类、数学建模类、开发类、人工智能类、创新创业类、其他类，学习算法可以一条龙参加很多比赛，大多都在大学生竞赛榜单上，对自己的升学、求职都有帮助，以下整理了一些我较为熟

悉的比赛，如有错误请联系我

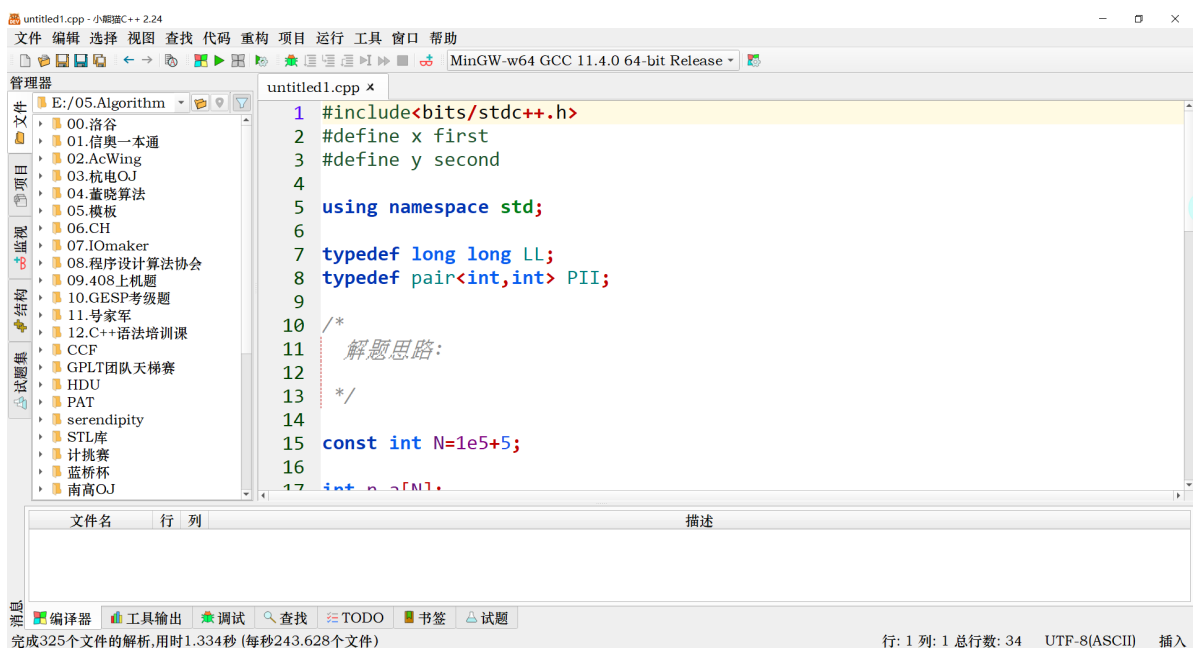
名称	时间	介绍
ICPC国际大学生程序设计竞赛	网络赛9月→区域赛→EC final亚洲总决赛→world final世界总决赛	难度大，含金量高，可加分，3人一队
CCPC中国大学生程序设计竞赛	（XCPC）省赛/邀请赛(上下半年都有)→区域赛→全国总决赛	难度大，含金量高，可加分，3人一队
CCF CAF 全国算法精英大赛（CCF）	第一场2月，第二场预估11月	难度大，含金量较高，可加分，3人一队（ACM赛制）
GPLT团体程序设计天梯赛	每年3月	难度偏上，含金量相对较高，可加分，10人一队
蓝桥杯	省赛4月，国赛5月	难度适中，含金量相对较高，可加分，个人赛
百度之星	初赛6月共三场，决赛	难度偏上，含金量相对较高，可加分（2023），个人赛（全国社会人士都可以参加）
传智杯	每年11月或12月	难度偏上，含金量相对较高，不可加分，个人赛
全国高校计算机能力挑战赛	每年12月	难度一般，含金量一般，不可加分，个人赛

参加算法竞赛对升学/求职有什么帮助? [参加算法竞赛「信奥」有什么用? - 知乎 \(zhihu.com\)](#)

- 介绍了这么多, 接下来正式进入C++的学习吧~

## 1. 环境搭建

- 对于C/C++, 常用的IDE (集成开发环境) 有:
  - 对于初学者:
    1. Dev C++
    2. 小熊猫C++ (国产, 个人较为推荐)
  - 对于进阶者:
    1. CP Editor
    2. Sublime Text (需自己配置)
    3. Vscode
    4. Visual Studio (适用于大型项目)
- 在本次课程中, 我使用的IDE是小熊猫C++, 下载地址为: [下载 \(roygh.net\)](#), 根据电脑位数安装相应的版本即可, 或者使用我在群里发布的安装包进行安装也可



- 小熊猫C++中有非常多方便的快捷键，可参考官方文档：[基本功能与快捷键 | 小熊猫C++ \(royqh.net\)](https://royqh.net/)
- 目前我们需要掌握的快捷键：
  - Ctrl+N：新建C++源代码文件，其后缀名为 .cpp（其实cpp就是c plus plus，即C++，C语言源文件代码后缀名为 .c）
  - Ctrl+S：保存
  - F11：编译并运行

## 2. 第一个 C++ 程序——Hello World!

```
1 #include<iostream> // include命令引入iostream头文件,i:input(输入),o:output(输出),stream(流)
2 using namespace std; // 直译:使用std命名空间
3 // main函数是C++程序的入口,C++程序将从main函数开始执行
4 int main() {
5     cout<<"Hello world"<<endl; // cout和endl都是std命名空间中的,cout是标准输出流对象,endl操作符:end of line,换行,<<流插入运算符
6     return 0; // 函数都要有返回值,main函数同理,return 0表示返回值是0
7 }
```

- `#include<iostream>` 用于告诉编译器在编译程序之前包含一个名为 `iostream` 的头文件，它定义了输入输出流的类和对象，例如 `cin`（用于输入）、`cout`（用于输出）等
- `using namespace std;` 是 C++ 中的一条语句，它的作用是引入 `std` 命名空间中的所有名称到当前作用域中，即可以直接使用这些名称，而不需要 `std::` 前缀
- 在C++中，`<<`是流插入运算符，指向输出对象，最终把`<<`中的内容输出到控制台中
- 每句代码后都需要加上引号；这是C++语言的规则，否则编译会报错

- 上述模板大家需要熟练掌握

## ※ 注释

```
1 // 单行注释
2
3 /*
4     多行注释
5     八百标兵奔北坡，
6     炮兵并排北边跑，
7     炮兵怕把标兵碰，
8     标兵怕碰炮兵炮。
9 */
```

## 3. 变量与数据类型

### 3.1. 变量

- 观察一下下面两段代码

```
1 cout<<111<<endl; // ①
2
3 int a=111; // ②
4 cout<<a<<endl;
```

- 第一段，对于在程序中直接给出的值，我们称作字面量
- 第二段，我们将值存储到一个名为a的变量中，随后输出这个变量，两段代码得到的结果是一样的
- 变量：一个具名的、可供程序操作的存储空间，那么a之前的int是什么意思呢？

### 3.1.1. 变量命名规则

- 变量命名规则是为了增强代码的可读性和容易维护性。以下为C++必须遵守的变量命名规则：
  1. 变量名只能是字母（A-Z，a-z）和数字（0-9）或者下划线（\_）组成
  2. 第一个字母必须是字母或者下划线开头
  3. 不能使用C++关键字来命名变量，以免冲突
  4. 变量名区分大小写

## 3.2. 数据类型

```
1 声明一个变量： 数据类型 变量名； // int a;  
2 声明一个变量同时赋值： 数据类型 变量名 = 具体值； // int  
   a=3;  
3 声明多个变量： 数据类型 变量名1,变量名2;  
4 声明多个变量同时赋值： 数据类型 变量名1 = 具体值1, 变量名  
5 2=具体值2;
```

- 在C++中，常用数据类型有：整型、浮点型、字符型、布尔型、空类型，我们一一介绍

### 3.2.1. 整型

数据类型	占用空间	取值范围
short(短整型)	2字节	$(-2^{15} - 2^{15} - 1)$
int(整型)	4字节	$(-2^{31} - 2^{31} - 1)$
long(长整型)	Windows为4字节，Linux为4字节(32位)、8字节(64位)	$(-2^{31} - 2^{31} - 1)$
long long(长长整型)	8字节	$(-2^{63} - 2^{63} - 1)$

- 以最常用的int型和long long为例

```

1 int a= 4 ;
2 long long b= 16 ;
3 cout<<a<<endl;
4 cout<<b<<endl;

```

- 为什么int型的取值范围是： $(-2^{31} - 2^{31} - 1)$ ？
- 这跟数据在计算机中的存储形式有关，我们将变量中的最高位作为符号位，符号位为0表示正数，为1表示负数，4字节共有32位，拿1位做符号位，则表示数值大小的仅有31位，见推导



在计算机中，数据以二进制形式存储，那么全为1的31位二进制数的十进制表示数应为

$$2^0 + 2^1 + 2^2 + \dots + 2^{30} = 2^{31} - 1 \quad (\text{简单的等比数列求和})$$

- 所以int型装不下小于 $-2^{31}$ 的负整数，也装不下大于 $2^{31} - 1$ 的正整数
- 熟记： $2^{31} \approx 2 \times 10^{10}$  (int型最大值)， $2^{63} \approx 9 \times 10^{19}$  (long long型最大值)



```
1 | int a=99999999999; // 显然不合法
```

- 同时，可以使用sizeof函数可以验证每种变量的大小是否真的和我讲的一样

```
test.cpp x
1 | #include<iostream>
2 | using namespace std;
3 |
4 | int main() {
5 |     short a=1;
6 |     int b=2;
7 |     long c=3;
8 |     long long d=4;
9 |     cout<<sizeof(a)<<endl;
10 |    cout<<sizeof(b)<<endl;
11 |    cout<<sizeof(c)<<endl;
12 |    cout<<sizeof(d)<<endl;
13 |    return 0;
14 | }
```

E:\05.Algorithm\test.exe

```
2
4
4
8
-----
Process exited after 0.08589
Press ANY key to exit...
```

### 3.2.2. 浮点型

数据类型	占用空间	取值范围
float(单精度浮点型)	4字节	7位有效数字
double(双精度浮点型)	8字节	15~16位有效数字

- 同理，double能表示的有效数字范围更多，我们更经常使用double

```
1 | float a=3.14159;
2 | float b=3.1415926535; // 超过 7 位有效数字了,不合法
3 | double c=3.1415926535;
4 |
5 | cout<<sizeof(a)<<endl;
6 | cout<<sizeof(c)<<endl;
```

- 千万要注意这里是有效数字而不是小数点后6~7位！

```
1 float a=5.99999;  
2 cout<<a<<endl; // 输出5.99999  
3 float b=5.999999;  
4 cout<<b<<endl; // 会输出6而不是5.999999  
5 float c=54.99999;  
6 cout<<c<<endl; // 会输出55而不是54.99999
```

### ※ 输出固定位数的浮点数

- 使用 `fixed<<setprecision(i)` 来输出固定位数的小数

```
1 // 比如我们定义了一个10位小数  
2 double x=3.1415926535;  
3 // 但是只想在控制台输出前3位, 自带四舍五入  
4 cout<<fixed<<setprecision(3)<<x<<'\n';
```

### 3.3.3. 字符型

数据类型	占用空间	取值范围
char(字符型)	1字节	只能存储一个字符

- 运行下面一段代码，我们发现将字符型变量强转成int型时可以输出一个数字，这是为什么呢？

```

1 #include<iostream>
2 using namespace std;
3 int main() {
4     char a='A'; // 变量a存储字符A
5     cout<<a<<endl;
6     cout<<sizeof(a)<<endl;
7     cout<<(int)a<<endl; // 表示强制将a转换成int型,此时输出一个数字,为什么?
8     return 0;
9 }

```

## ※ ASCII 码

- ASCII码是一种字符编码标准，用于将特定的数字值与字符进行对应
- ASCII码最初是为了使计算机能够读取和交换文本数据而设计的，它使用 7 位二进制数（0 到 127）来表示 128 个不同的字符
- 可以在 c++API 参考文档中找到，一般来说对于初学者只需要记住以下三个字符的ASCII码即可：'a'=97，'A'=65，'0'=48，' '=0，因为最常用，记住了a就等于记住了a—z，记住了A就等于记住了A—Z，记住了0就等于记住了0—9，其中'a'='A'+32

C/C++ 语言参考

cppreference.com -> ASCII 码表				
ASCII 码表				
下面的 ASCII 码表包含数值在0-127之间的字符的十进制、八进制以及十六进制表示。				
十进制	八进制	十六进制	字符	描述
0	0	00	NUL	
1	1	01	SOH	start of header
2	2	02	STX	start of text
3	3	03	ETX	end of text
4	4	04	EOT	end of transmission
5	5	05	ENQ	enquiry
6	6	06	ACK	acknowledge
7	7	07	BEL	bell
8	10	08	BS	backspace
9	11	09	HT	horizontal tab
10	12	0A	LF	line feed
11	13	0B	VT	vertical tab
12	14	0C	FF	form feed
13	15	0D	CR	carriage return
14	16	0E	SO	shift out
15	17	0F	SI	shift in
16	20	10	DLE	data link escape
17	21	11	DC1	no assignment, but usually XON
18	22	12	DC2	
19	23	13	DC3	no assignment, but usually XOFF

## ※ 转义字符

- 转义字符可使普通字符表示不同的意义，同可见c++API.chm
- 比较常用的有：`\t` 水平制表符，`\n` 换行符
- 这里讲解一下容易被大部分人忽略的关于 `\n` 和 `endl` 的区别：
  1. `\n` 仅仅表示换行符
  2. `endl` 表示插入换行符 `\n` 并刷新输出缓冲区，多了一个操作，性能上比 `\n` 更慢，害怕 TLE 的同学可以留意一下
- 想详细了解一下可参考：[endl 和 '\n' 的区别\\_换行符后面加endl-CSDN博客](#)

### 3.2.4. 布尔型

数据类型	占用空间	取值范围
bool(布尔型)	1字节	true or false

```
1 bool a=true; // 真
2 bool b=false; // 假
3 cout<<a<<' '<<b<<'\n'; // 打印出来为1和0
```

- 在计算机中表示true为 1，表示false为 0

```
1 int c=2;
2 int d=999;
3 int e=0;
4 int f=-2;
5 cout<<bool(c)<<' '<<bool(d)<<' '<<bool(e)<<'
  '<<bool(f)<<'\n'; // 真真假真
```

- 得以验证，0 值的布尔值为false，非 0 值的布尔值都为true，只是true表示的数值是 1

### 3.2.5. 空类型

数据类型	占用空间	取值范围
void(空类型)	1字节(可用sizeof验证)	无

- void 常用于对函数的参数类型、返回值进行声明，或表示一种未知类型，不能代表一个真实的变量

```
20 int main() {  
21     void a;  
22     return 0;  
23 }
```

### ※ 非基本数据类型

- 除了以上几种基本数据类型外，C++中还有其他非基本数据类型，例如string（字符串）

```
1 #include<iostream>  
2 #include<cstring> // 使用string数据类型需引入头文件  
3 using namespace std;  
4 int main() {  
5     string str="Hello world";  
6     cout<<str<<'\n';  
7     return 0;  
8 }
```

### ※ 进制数表示

- 在讲解ASCII码的时候体现了十进制数、八进制数、十六进制数，所以这里我们特地讲一下二、八、十、十六进制数在计算机中是如何表示的

- 若一个数的进制数为X，那么这个数每一位的取值只能是  $[0, X-1]$ ，从低到高第k位的位权为  $X^{k-1}$
- 举个例子，对于一个2进制数，每一位的取值只能是 $[0, 1]$ ，第1位的位权是 $2^0$ ，第2位是 $2^1$  ...
- 值得一提的是16进制数，每一位的取值只能是  $[0, 15]$ ，但是一位怎么表示两位数呢？因此在16进制表示法中，A代表10，B代表11，...，F代表15

```

1  #include<iostream>
2  using namespace std;
3  int main() {
4      int a=0b1101; // 2进制数前缀为0b,表示1101是二进制数,转换为十进制输出为13
5      int b=013; // 8八进制数前缀为0,表示17是个八进制数,转换成十进制数输出为11
6      int c=25; // 10进制数无前缀
7      int d=0x1A; // 16进制数前缀为0x,表示1A是个十六进制数,转换为十六进制输出为26
8      cout<<a<<' '<<b<<' '<<c<<' '<<d<<' '<<endl;
9      // ' '是空字符字面量
10     return 0;
11 }
```

## ※ 强制类型转换

- 在讲ASCII码的时候我们强制将char类型的'A'转换为了数字65并输出，因此在此处简单谈一下强制类型转换问题
- 小范围或低精度向大范围或高精度转换→通常不会丢失精度
  1. int→long long
  2. float→double
  3. 整数→浮点数，浮点数类型通常可以精确表示整数

int 型正整数最大值,  $2^{31}-1$

0 1 1 1 1 1 1 1 1 1 1 ... 1 1 1 1 (符号位为1, 因为是正数, 剩下每一位都是1, 共31个1)

若转为64位的long long, 最大值是 $2^{63}-1$ , 高位补32个0 (多出4个字节)

0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 1 1 1 1 1 1 1 1 1 ... 1 1 1 1 (并不影响值的大小)

- 大范围或高精度向小范围或低精度转换→精度丢失!

1. long long→int
2. double→float
3. 浮点数→整数, 丢失小数部分

```
1 // 高向低
2 long long x=LONG_LONG_MAX;
3 cout<<x<<endl;
4 cout<<(int)x<<endl; // C语言写法, 溢出变负数
5 cout<<int(x)<<endl; // C++写法, 溢出变负数
6 double y=3.14159265358979; // 有效数字15位, 超过
  float有效位
7 cout<<fixed<<setprecision(14)<<y<<endl;
8 cout<<setprecision(14)<<float(y)<<endl; // 不足14
  位, 后面的数字随机
9 float z=3.14;
10 cout<<setprecision(2)<<z<<endl;
11 cout<<int(3.14)<<endl; // 丢失小数部分
```

## ※ 科学计数法

- $aeb$ : 表示a的b次方

```
1 int a=1e3; // 1000
2 int b=1e+3; // 等价于1e3
3 int c=-1.3e2; // -130
4 float d=1.3e-2; // 0.013
5 float e=-1.3e-2; // -0.013
```

## 4. 运算符

---

### 4.1. 算术运算符

- 用于处理四则运算，本小节介绍前七个，自增自减和赋值运算符一起讲

运算符	术语	示例	结果
+	正号	+3	3
-	负号	-3	-3
+	加	10+5	15
-	减	10-5	5
*	乘	10*5	50



运算符	术语	示例	结果
/	除(除数不能为0)	10/5	2
%	取模(取余, 只能用于整数之间)	10%3	1
++a	前置自增(先增后用)	a=2; b=++a;	a=3; b=3;
a++	后置自增(先用后增)	a=2; b=a++;	a=3; b=2;
--a	前置自减(先减后用)	a=2; b=--a;	a=1; b=1;
a--	后置自减(先用后减)	a=2; b=a--;	a=1; b=2;

```

1  #include<iostream>
2  #include<iomanip>
3  using namespace std;
4  int main() {
5      int a=+10;
6      int b=3;
7      int c=-3;
8
9      cout<<a<<' '<<b<<' '<<endl;
10
11     cout<<a+b<<endl;
12     cout<<a-b<<endl;
13     cout<<a*b<<endl;
14     cout<<a/b<<endl; // 整数之间做除法只能整除,余数需
                        要通过模运算拿到
15     cout<<a%b<<endl; // 取模运算只能用于整数之间
16

```

```

17     int d=0;
18     // cout<<a/d<<endl; // 除数不能为0
19
20     double e=0.67;
21     double f=0.31;
22     cout<<e/f<<endl; // 浮点数之间做除法得到浮点数,默认情况下对于无穷位数的小数输出的位数共7位
23     cout<<fixed<<setprecision(3)
24     <<0.67/0.31<<endl; // 需引入 iomanip 头文件,表示保留小数点后3位
25
26     double x=3.14;
27     double y=1.1;
28     // cout<<x%y<<endl; // 小数之间不能取模,取模的被除数也不能为0
29
30     return 0;
31 }

```

## 4.2. 赋值运算符

- 用于将表达式的值赋值给变量

运算符	术语	示例	结果
=	赋值	a=2; b=3;	a=2; b=3;
+=	加等于	a=0; a+=2;	a=2;
-=	减等于	a=5; a-=3;	a=2;
*=	乘等于	a=2; a*=2;	a=4;
/=	除等于	a=4; a/=2;	a=2;

运算符	术语	示例	结果
%=	模等于	a=3; a%=2;	a=1;

```

1 #include<iostream>
2 using namespace std;
3 int main() {
4     // 赋值运算符
5     int a=10;
6     a+=2;
7     cout<<a<<endl;
8     a-=2;
9     cout<<a<<endl;
10    a*=2;
11    cout<<a<<endl;
12    a/=2;
13    cout<<a<<endl;
14
15    // 自增/自减运算
16    a++; // 相当于a=a+1
17    cout<<a<<endl;
18    a--; // 相当于a=a-1;
19    cout<<a<<endl;
20
21    return 0;
22 }

```

- 算术运算符中的自增运算与赋值运算符相结合，是初学者非常容易混淆的问题
- 请读者回答下列几段代码的输出值

```

1 int a=2; // a=2
2 int b=a++; // int b=a++, int b=a; a++, b=a
3 cout<<a<<' '<<b<<endl;

```

```

1 int a=2;
2 int b=++a;
3 cout<<a<<' '<<b<<endl;

```

```

1 int a=2;
2 int b=a--;
3 cout<<a<<' '<<b<<endl;

```

```

1 int a=2;
2 int b=--a;
3 cout<<a<<' '<<b<<endl;

```

```

1 int a=2;
2 int b=a++*3;
3 cout<<a<<' '<<b<<endl;

```

```

1 int a=2;
2 int b=++a/3;
3 cout<<a<<' '<<b<<endl;

```

## 4.3. 比较运算符

- 用于表达式的比较，返回真或假

运算符	术语	示例	结果
==	相等	4==3	0
!=	不等	4!=3	1
<	小于	4<3	0
>	大于	4>3	1
<=	小于等于	4<=3	0

运算符	术语	示例	结果
>=	大于等于	4>=1	1

```

1 int a=1,b=2,c=3;
2 cout<< (a==b) <<endl;
3 cout<< (a<=b) <<endl;
4 cout<< (b>=c) <<endl;
5 cout<< (a!=b) <<endl;

```

## 4.4. 逻辑运算符

- 用于根据表达式的值返回真或假
- 逻辑与 和 逻辑或两种运算，建议记前半部分，逻辑与：全真则真，逻辑或：有真则真

运算符	术语	示例	结果
&&	与	a&&b	全真为真，有假则假
	或	a  b	有真则真，全假为假
!	非	!a	如果a为假，则!a为真；如果a为真，则!a为假

- 请读者回答下面一段代码的输出

```

1 #include<iostream>
2 using namespace std;
3 int main() {
4     int a=10;
5     int b=-999;
6     int c=0;
7     cout<< (a&&b) <<endl; // 注意这里需要加括号
8     cout<< (a&&c) <<endl;
9     cout<< (b||c) <<endl;
10    cout<< !c <<endl;
11

```

```

12 // 为什么不输出true或false呢? → 逻辑运算符的结果被
    用于cout输出时,会隐式地转为整数类型
13     return 0;
14 }

```

## ※ 运算符优先级问题

类别	运算符	结合性
后缀	() [] -> . ++ --	从左到右
一元	+ - ! ~ ++ -- (type)* & sizeof	从右到左
乘除	* / %	从左到右
加减	+ -	从左到右
移位	<< >>	从左到右
关系	< <= > >=	从左到右
相等	== !=	从左到右
位与 AND	&	从左到右
位异或 XOR	^	从左到右
位或 OR		从左到右
逻辑与 AND	&&	从左到右
逻辑或 OR		从左到右
条件	?:	从右到左
赋值	= += -= *= /= %= >>= <<= &= ^=  =	从右到左
逗号	,	从左到右

- 举个例子, 对于 `b=a+=3*5` 这个表达式, 究竟是先计算 `b=a` 还是 `a+=3`, `*5` 又是计算在哪部分身上的呢?
- 经过测试, 我们可以看到结果是 `17 17`, 所以原式的计算顺序应该是 `3*5=15`, `a+=15`, `b=a`, 原式中的运算符一共有 3 个, `*`, `+=`, `=`, 查表可知乘的优先级更高, 因此先计算 `3*5`, `+=`, `=` 的优先级相同, 但结合性是从右到左计算, 因此先计算 `a+=15`, 再计算 `b=a`

```
1 int a=2;
2 int b=a+=3*5;
3 cout<<a<<' '<<b<<endl; // 17 17
```

- 关于运算符优先级问题，只要大家平时在写代码的时候满足代码规范，可读性高，一般就不会出现这种问题，只是这种问题容易出现在C语言期末考试里考考你

## 5. 输入输出

### 5.1. cin 和流提取运算符 (>>)

- 在C++中，cin 和 cout 是用于输入和输出的标准流对象。cin用于从标准输入（通常是键盘）读取数据，而 cout 用于向标准输出（通常是屏幕）输出数据
- 输入：cin >> 变量，使用流提取运算符从标准输入读取数据并存储到指定变量中

```
1 int a;
2 cin>>a;
3 b=a+3;
4 cout<<a<<'\n'<<b;
```

### 5.2. cout 和流插入运算符 (<<)

- 输出：cout << 数据，使用流插入运算符将数据输出到标准输出

## 5.3. 格式化输出

```
1 #include <iostream>
2 #include <cstdint> // 包含用于标准整数类型的头文件
3 #include <cmath>
4 #include <iomanip> // 包含头文件以使用格式化输出相关功能
5
6 using namespace std;
7 int main() {
8     // 1.指定输出宽度
9     cout << setw(10) << "Hello" << endl; // 输出
10    宽度为 10
11
12    // 2.指定精度
13    cout << setprecision(3) << 3.14159 << endl;
14    // 输出精度为 3
15
16    // 3.控制对齐方式:
17    cout << left << setw(10) << "Hello" <<
18    endl; // 左对齐
19    cout << right << setw(10) << "Hello" <<
20    endl; // 右对齐
21
22    // 4.设置填充字符:
23    cout << setfill('-') << setw(10) << "Hello"
24    << endl; // 使用 '-' 填充
25
26    // 5.输出格式设置
27    cout << fixed << 3.14159 << endl; // 固定小
28    数点输出
29    cout << scientific << 3.14159 << endl; //
30    科学计数法输出
31    cout << hex << 255 << endl; // 十六进制输出
32    cout << oct << 255 << endl; // 八进制输出
33}
```



```
27     return 0;
28 }
```

## ※ 如何基于各大 OJ 有效刷题？

所谓OJ就是 Online Judge，是用于自动评判编程题解的在线平台，通常用于编程竞赛和练习，帮助程序员测试和验证他们的代码是否正确并且有效

- 我们以洛谷平台为例，以最经典的 `A+B Problem` 为例，附链接：[P1001 A+B Problem - 洛谷](#)

## 读懂输入输出

### 输入格式

两个以空格分开的整数。

### 输出格式

一个整数。

请严格遵守输入输出格式

如本题中，输出只有一个整数，不得包含多余的空格或空行！！

### 输入输出样例

输入 #1

复制

输出 #1

复制

20 30

50

## 锁定数据类型

- 基于题目描述中的数据范围通常可以锁定出我们需要使用的数据类型( $2.1 \times 10^9$ )

**题目描述** 题目说了a和b的值都不超过 $10^9$ ，则结果不超过 $2 \times 10^9$

输入两个整数  $a, b$ ，输出它们的和 ( $|a|, |b| \leq 10^9$ )。而int型的最大值是 $2^{31}-1$ ，能够容纳

# 选择合适算法

- 通常来说，对于一道算法题，我们有多种解决方法，每种解题方法解决一道题目所花的时间、所用的内存可能有所不同
- 同样的，每道题目有对应的时间限制和内存限制
- C++ 1s内的操作次数大致在  $[10^7, 10^8]$ ，若时间限制为1s，则需要把计算量控制在  $[10^7, 10^8]$  以内

洛谷 / 题目列表 / 题目详情

**P1001 A+B Problem**

这道题共被洛谷用户提交过1.54million次，有855.21k次是通过了的，我们常说的AC

时间限制为1s：每道题目通常有多个测试点，每个测试点需要在1s内得到结果，某个测试点超过1s不通过

内存限制为512M：你所提交的代码所使用内存空间不得超过512MB

提交答案 加入题单 复制题目

提交	通过	时间限制	内存限制
1.54M	855.21k	1.00s	512.00MB

## 技巧：基于数据范围反推时间复杂度

### [由数据范围反推算法复杂度以及算法内容 - AcWing](#)

- 一道算法题中需要多次使用、计算的变量叫做**问题规模**
- 通过问题规模的数据范围和C++在1s内的操作次数，可以大致反推出通过本题所需要的时间复杂度及算法内容

## 由数据范围反推算法复杂度以及算法内容

作者: yxc, 2018-09-10 22:27:18, 所有人可见, 阅读 79582

▲ 一般ACM或者笔试题的时间限制是1秒或2秒。  
3723 在这种情况下，C++代码中的操作次数控制在  $10^7 \sim 10^8$  为最佳。

▼ 下面给出在不同数据范围下，代码的时间复杂度和算法该如何选择：

- ★
- 6232
1.  $n \leq 30$ , 指数级别, dfs+剪枝, 状态压缩dp
  2.  $n \leq 100 \Rightarrow O(n^3)$ , floyd, dp, 高斯消元
  3.  $n \leq 1000 \Rightarrow O(n^2)$ ,  $O(n^2 \log n)$ , dp, 二分, 朴素版Dijkstra, 朴素版Prim, Bellman-Ford
  4.  $n \leq 10000 \Rightarrow O(n * \sqrt{n})$ , 块状链表、分块、莫队
  5.  $n \leq 100000 \Rightarrow O(n \log n) \Rightarrow$  各种sort, 线段树、树状数组、set/map、heap、拓扑排序、dijkstra+heap、prim+heap、Kruskal、spfa、求凸包、求半平面交、二分、CDQ分治、整体二分、后缀数组、树链剖分、动态树
  6.  $n \leq 1000000 \Rightarrow O(n)$ , 以及常数较小的  $O(n \log n)$  算法  $\Rightarrow$  单调队列、hash、双指针扫描、BFS、并查集、kmp、AC自动机, 常数比较小的  $O(n \log n)$  的做法: sort、树状数组、heap、dijkstra、spfa
  7.  $n \leq 10000000 \Rightarrow O(n)$ , 双指针扫描、kmp、AC自动机、线性筛素数
  8.  $n \leq 10^9 \Rightarrow O(\sqrt{n})$ , 判断质数
  9.  $n \leq 10^{18} \Rightarrow O(\log n)$ , 最大公约数, 快速幂, 数位DP
  10.  $n \leq 10^{1000} \Rightarrow O((\log n)^2)$ , 高精度加减乘除
  11.  $n \leq 10^{100000} \Rightarrow O(\log k \times \log \log k)$ ,  $k$ 表示位数, 高精度加减、FFT/NTT

# 技巧：基于提交反馈进行 Debug

- 一道编程题提交到OJ时会经过以下几个步骤，编译→运行(跑测试)→输出到控制台
- 我们的代码正确与否、是否能解决问题都是通过提交后的反馈来决定的，通常来说反馈有如下几种（自总结）：

专业名	全称	含义	可能导致原因
AC	Accepted	代码通过了所有测试用例，表示解答正确	本题通过，皆大欢喜
WA	Wrong Answer	代码输出与期望结果不匹配	代码逻辑错误导致输出了错误的结果、未考虑到所有情况（边界情况处理不足，代码健壮性不强）、输出了不必要的空格或空行
TLE	Time Limit Exceeded	代码执行时间超过限制（通常1~2s）	算法复杂度过高、循环不终止、递归深度太大
MLE	Memory Limit Exceeded	代码使用的内存超过限制（通常是256MB或512MB）	使用了过大的数据结构

专业名	全称	含义	可能导致原因
RE	Runtime Error	代码在运行时发生错误	除零错误、数组越界、指针错误、非法内存访问
CE	Compliation Error	代码编译错误	语法错误、缺少头文件
Segfault	Segmentation Fault	段错误，属于RE的一种，单独拿出来说一下，段是操作系统中的概念，和内存有关	指针错误、数组越界、递归过深

## 【例题】 A+B Problem

- 灰常简单啦

```

1  #include<iostream>
2  using namesapce std;
3  int main() {
4      int a,b;
5      cin>>a>>b;
6      int sum=a+b;
7      cout<<sum;
8      return 0;
9  }
```