



程序设计与算法(一)

C语言程序设计

郭 炜

微信公众号



微博: <http://weibo.com/guoweiofpku>

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄



北京大学
PEKING UNIVERSITY

信息科学技术学院

指定教材：

《新标准C++程序设计教程》

郭炜 编著

清华大学出版社

重点大学计算机专业系列教材

新标准C++程序设计教程

郭炜 编著



清华大学出版社



北京大学
PEKING UNIVERSITY

信息科学技术学院

输入输出控制符



罗马大斗兽场遗址

输入输出控制符

在printf和scanf中可以使用以"%"开头的控制符，指明要输入或输出的数据的类型以及格式。

```
int n = 3;  
printf("I want to buy %d books for %f dollars",3,4.5);  
double f;  
scanf("%d%f",&n,&f);
```

输入输出控制符

在printf和scanf中可以使用以"%"开头的控制符，指明要输入或输出的数据的类型以及格式。

常用格式控制符	作 用
%d	读入或输出int变量
%c	读入或输出char变量
%f	读入或输出float变量，输出时保留小数点后面6位
%lf	读入或输出double变量，输出时保留小数点后面6位
%x	以十六进制读入或输出整型变量
%lld	读入或输出long long 变量(64位整数)
%nd (如%4d,%12d)	以n字符宽度输出整数，宽度不足时用空格填充
%0nd (如 %04d,%012d)	以n字符宽度输出整数，宽度不足时用0填充
%.nf (如%.4f,%.3f)	输出double或float值，精确到小数点后n位

用scanf读入不同类型的变量

用scanf可以一次读入多个类型不同的变量，只要输入的各项之间用空格分隔即可。

```
#include <iostream>
#include <cstdio> //使用printf,scanf要有这一行
using namespace std;
int main()
{
    int n; char c; float m;
    scanf("%d%c%f", &n, &c, &m);
    printf("%d %c %f", n, c, m);
    return 0;
}
```

● 依次输入一个整数，一个字符，再一个小数，则它们会被分别放入n,c,m

34k 234.45 ✓

34 k 234.449997

用scanf读入不同类型的变量

用scanf可以一次读入多个类型不同的变量，只要输入的各项之间用空格分隔即可。

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int n; char c; float m;
    scanf("%d%c%f", &n, &c, &m);
    printf("%d %c %f", n, c, m);
    return 0;
}
```

- 依次输入一个整数，一个字符，再一个小数，则它们会被分别放入n,c,m
- **&n**代表“取n的地址”

34k 234.45 ✓

34 k 234.449997

用scanf读入不同类型的变量

用scanf可以一次读入多个类型不同的变量，只要输入的各项之间用空格分隔即可。

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int n; char c; float m;
    scanf("%d%c%f", &n, &c, &m);
    printf("%d %c %f", n, c, m);
    return 0;
}
```

- 依次输入一个整数，一个字符，再一个小数，则它们会被分别放入n,c,m
- **&n**代表“取n的地址”
- **%c**代表等待输入一个字符

34k 234.45 ✓

34 k 234.449997

用scanf读入不同类型的变量

用scanf可以一次读入多个类型不同的变量，只要输入的各项之间用空格分隔即可。

```
#include <cstdio>
#include <iostream>
using namespace std;
int main()
{
    int n; char c; float f;
    scanf("%d%c%f", &n, &c, &f);
    printf("%d %c %f", n, c, f);
    return 0;
}
```

- 依次输入一个整数，一个字符，再一个小数，则它们会被分别放入n,c,m
- **&n**代表“取n的地址”
- **%c**代表等待输入一个字符
- **%f**代表等待输入一个float小数

<u>34k 234.45 ✓</u> 34 k 234.449997
--

%f用于输出double或float类型的值，保留小数点后面6位

用scanf读入不同类型的变量

输入字符时，不会跳过空格(空格也会被当作字符读入)，
输入其他类型的数据时，会跳过空格

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int n; char c; float f;
    scanf("%d%c%f", &n, &c, &f);
    printf("%d %c %f", n, c, f);
    return 0;
}
```

34 k 456 ✓
34 0.000000

c = ' ', 读入f时，对应输入是'k'，导致出错

用scanf跳过输入中的字符

如果在输入中有scanf中出现的非控制字符,
则这些字符会被跳过

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int n,m; char c; float f;
    scanf("%d %c,%f:%d",&n,&c,&f,&m);
    printf("%d,%c,%f,%d",n,c,f,m);
    return 0;
}
```

12 k,3.75:290 ✓
12,k,3.750000,290

控制printf 输出整数的宽度

<code>%nd</code> (如 <code>%4d,%12d</code>)	以n字符宽度输出整数, 宽度不足时用空格填充
<code>%0nd</code> (如 <code>%04d,%012d</code>)	以n字符宽度输出整数, 宽度不足时用0填充

```
int n = 123;  
printf("%05d,%5d,%5d,%05d",n,n,123456,123456);
```

00123, 123,123456,123456

控制printf 输出浮点数的精度

<code>%.nf</code> (如 <code>%.4f</code> , <code>%.3f</code>)	输出浮点数, 精确到小数点后n位
--	------------------

```
float a = 123.45;  
double b = 22.37362723;  
printf("%.5f %.2f %.12f", 12.3, a, b);
```

12.30000 123.45 22.373627230000

尽量使用double

- double 精度高于float, 所以一般尽量使用double!
- double类型的变量用 `%lf` 输入!

```
int main()
{
    float f,c;
    scanf("%f",&f);
    c = 5 * (f-32)/9;
    printf("%.5f",c,f);
    return 0;
}
```

输入: 1501

输出: 816.11108

```
int main()
{
    double f,c;
    scanf("%lf",&f);
    c = 5 * (f-32)/9;
    printf("%.5f",c,f);
    return 0;
}
```

输入: 1501

输出: 816.11111

使用 double 的结果更精确!!!!

格式控制符%x和%u

- **%x**: 以十六进制形式读入或输出整数
- **%u**: 以无符号整数形式输出整数

```
printf ("%x,%d,%u", 0xffffffff, 0xffffffff, 0xffffffff) ;  
=> ffffffff,-1,4294967295
```

格式控制符%x和%u

- **%x**: 以十六进制形式读入或输出整数
- **%u**: 以无符号整数形式输出整数

```
printf ("%x,%d,%u", 0xffffffff, 0xffffffff, 0xffffffff);  
=> ffffffff,-1,4294967295
```

```
int n;  
scanf ("%x", &n);  
printf ("%d", n);
```

12 ✓

18

a0 ✓

160

用C++的cout进行输出

```
#include <iostream>
using namespace std;
int main()    {
    int n=5;
    double f = 3.9;
    char c = 'a';
    cout << "n=" << n << ",f=" << f << endl; //endl换行
    cout << 123 << ", c=" << c;
    return 0;
}
```

```
n=5,f=3.9
123, c=a
```

用C++的cin进行输入

```
#include <iostream>
using namespace std;
int main()    {
    int n1,n2;
    double f;
    char c;
    cin >> n1 >> n2 >> c >> f;
    cout << n1 << "," << n2 << "," << c << "," << f;
    return 0;
}
```

5 10k 1.23 ✓
5,10,k,1.23

用cin读入所有输入的字符，包括空格，回车

```
#include <iostream>
using namespace std;
int main()
{
    int c;
    while((c = cin.get()) != EOF) {
        cout << (char)c ;
    }
    return 0;
}
```

用scanf读入所有输入的字符，包括空格，回车

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    char c;
    while (scanf("%c",&c) != EOF) {
        printf("%c",c);
    }
    return 0;
}
```

用cin,cout和scanf,printf

- **cin,cout 速度比scanf,printf慢,输入输出数据量大时用后者**
- **一个程序里面不要同时用cin和scanf,不要同时用cout和printf**



北京大学
PEKING UNIVERSITY

信息科学技术学院

赋值运算符
算术运算符
算术表达式



威尼斯

赋值运算符

- 赋值运算符用于给变量赋值，常用有以下六种

=

+=

-=

*=

/=

%=

赋值运算符

```
int a;  
a = 1;           // a的值变为1  
a = a + 1;       // a的值变为2  
a = 4 + a;       // a的值变为6  
a += b;          // 等效于 a = a + b, 但是执行速度更快
```

`-=`, `*=`, `/=`, `%=` 用法与 `+=` 类似

表达式 `x = y` 的值, 就是 `y` 的值

算术运算符

七种算术运算符用于数值运算
运算符+操作数构成表达式

加 +

减 -

乘 *

除 /

求余数 %

自增 ++

自减 --

加、减、乘运算符

$a+b$ 、 $a-b$ 、 $a*b$ 这三个表达式的值，就是 a 和 b 做算术运算的结果。表达式的值的类型，以操作数中精度高的类型为准。

加、减、乘运算符

$a+b$ 、 $a-b$ 、 $a*b$ 这三个表达式的值，就是 a 和 b 做算术运算的结果。表达式的值的类型，以操作数中精度高的类型为准。

精度：

`double > long long > int > short > char`

加、减、乘运算符

$a+b$ 、 $a-b$ 、 $a*b$ 这三个表达式的值，就是 a 和 b 做算术运算的结果。表达式的值的类型，以操作数中精度高的类型为准。

精度：

`double > long long > int > short > char`

`2 * 0.5 => 1.0`

加、减、乘运算的溢出

两个整数类型进行加、减、乘都可能导致计算结果超出了结果类型所能表示的范围，这种情况就叫做**溢出**。

加、减、乘运算的溢出

两个整数类型进行加、减、乘都可能导致计算结果超出了结果类型所能表示的范围，这种情况就叫做**溢出**。

计算结果的溢出部分直接被丢弃。

加、减、乘运算的溢出

两个整数类型进行加、减、乘都可能导致计算结果超出了结果类型所能表示的范围，这种情况就叫做**溢出**。

计算结果的溢出部分直接被丢弃。

实数（浮点数）运算也可能溢出，结果不易预测。

加、减、乘运算的溢出

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int n1 = 0xffffffff;
    cout << n1 << endl;    //输出4294967295
    unsigned int n2 = n1 + 3; //导致溢出
    cout << n2 << endl;    //输出2
    return 0;
} // 0xffffffff + 3 的结果, 应该是 0x1000000002,
```


加、减、乘运算的溢出

- 有时计算的最终结果似乎不会溢出，但中间结果可能溢出，这也会导致程序出错

加、减、乘运算的溢出

- 有时计算的最终结果似乎不会溢出，但中间结果可能溢出，这也会导致程序出错

- 例：(a+b)/2 未必等于 a/2+b/2

```
printf("%d", (2147483646 + 6) / 2);    => -1073741822  
printf("%d", 2147483646 / 2 + 6 / 2);  => 1073741826
```

加、减、乘运算的溢出

- 有时计算的最终结果似乎不会溢出，但中间结果可能溢出，这也会导致程序出错

- 例：(a+b)/2 未必等于 a/2+b/2

```
printf("%d", (2147483646 + 6) / 2);    => -1073741822  
printf("%d", 2147483646 / 2 + 6 / 2);  => 1073741826
```

- 解决溢出的办法是尽量使用更高精度的数据类型（两个int进行运算会溢出，用两个 long long 进行运算可能就不会溢出）

除法运算

- 除法的计算结果，类型和操作数中精度高的类型相同

除法运算

- 除法的计算结果，类型和操作数中精度高的类型相同
- 两个整数做除法，结果是商。余数忽略

$$22 / 5 = 4$$

$$-22 / 5 = -4$$

除法运算

```
int main()    {  
    int a = 10;  
    int b = 3;  
    double d = a/b; // a/b 的值也是整型, 其值是3  
    cout << d << endl; //输出 3  
    d = 5/2;           //d的值变为2.0  
    cout << d << endl; //输出 2  
    d = 5/2.0;  
    cout << d << endl; //输出 2.5  
    d = (double)a/b;  
    cout << d << endl; //输出 3.33333  
    return 0;  
}
```

模运算

求余数的运算符 “%” 也称为模运算符。它是双目运算符，两个操作数都是整数类型的。a % b 的值就是a除以b的余数。

$$22 \% 5 = 2$$

$$-22 \% 5 = -2$$

模运算

求余数的运算符 “%” 也称为模运算符。它是双目运算符，两个操作数都是整数类型的。a % b 的值就是a除以b的余数。

$$22 \% 5 = 2$$

$$-22 \% 5 = -2$$

除法运算和模运算的除数都不能为0，否则程序会崩溃！！！！

自增运算符 “++”

- 单目运算符，操作数为整数类型变量或实数型变量

自增运算符 “++”

- 单目运算符，操作数为整数类型变量或实数型变量
- 有前置和后置两种用法

自增运算符 “++”

- 单目运算符，操作数为整数类型变量或实数型变量

- 有前置和后置两种用法

- 前置用法：

++a; 将a的值加1，表达式返回值为a加1后的值

自增运算符 “++”

- 单目运算符，操作数为整数类型变量或实数型变量

- 有前置和后置两种用法

- 前置用法：

`++a;` 将a的值加1，表达式返回值为a加1^后的值

- 后置用法：

`a++;` 将a的值加1，表达式返回值为a加1^前的值

自增运算符 “++”

```
#include <iostream>
using namespace std;
int main()
{
    int n1 ,   n2 = 5;
    n2 ++;    // n2变成6
    ++ n2;    // n2变成 7
    n1 = n2 ++;    // n2变成8,n1变成7
    cout << n1 << "," << n2 << endl;    //输出 7,8
    n1 = ++ n2;    //n1和n2都变成9
    cout << n1 << "," << n1 << endl;    //输出 9,9
    return 0;
}
```

自减运算符 “--”

自减运算符 “--”

用于将整型或实数型变量的值减1。它的用法和 “++” 相同

取相反数运算符 -

单目运算符 "-"

用于取整型或实数型变量的值的相反数

```
int a = 4;  
int b = -a;  
int c = -5 * 3;
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

关系运算符
逻辑运算符
逻辑表达式



承德小布达拉宫

关系运算符

- 六种关系运算符用于数值的比较

相等 ==

不等 !=

大于 >

小于 <

大于等于 >=

小于等于 <=

- 比较的结果是bool类型，成立则为true,反之为false

bool类型

- bool类型变量只有两种取值，true或false
- false等价于0, true等价于非0整型值

```
int n = true, m = false;  
printf("%d,%d", n, m);    => 1,0
```

关系运算符

```
int main()
{
    int n1 = 4,  n2 = 5,  n3;
    n3 = ( n1 > n2 );    // n3 的值变为 0
    cout << n3 << ", ";  // 输出 0,
    n3 = ( n1 < n2 );    // n3 的值变为非0值
    cout << n3 << ", ";  // 输出 1,
    n3 = (n1 == 4);      // n3 的值变为非 0 值
    cout << n3 << ", ";  // 输出 1,
    n3 = (n1 != 4);      // n3 的值变为0
    cout << n3 << ", ";  // 输出 0,
    n3 = (n1 == 5);      // n3 的值变为0
    cout << n3 ;         // 输出 0,
    return 0;
}
```

逻辑运算符和逻辑表达式

逻辑运算符用于表达式的逻辑操作，有 `&&`, `||`, `!` 三种，操作的结果是true或false

●与 `&&`

`exp1 && exp2` 当且仅当exp1和exp2的值都为真（或非0）时，结果为true

```
int n = 4;  
n > 4 && n < 5      => false  
n >= 2 && n < 5     => true  
5 && 0              => false  
4 && 1              => true
```

逻辑运算符和逻辑表达式

- 或 **||**
exp1 || exp2 当且仅当exp1和exp2的值都为假（或0）时，结果为false

```
int n = 4;  
n > 4 || n < 5    => true  
n <= 2 || n > 5   => false
```

逻辑运算符和逻辑表达式

●非 !

! exp exp值为真(或非0),结果为false,exp值为false(0),结果为true

! (4 < 5) => false

! 5 => false

! 0 => true

逻辑运算符和逻辑表达式

逻辑表达式是**短路计算**的，即对逻辑表达式的计算，在整个表达式的值已经能够断定的时候即会停止

- `exp1 && exp2` : 如果已经算出表达式`exp1`为假，那么整个表达式的值肯定为假，于是表达式`exp2`就不需要再计算
- `exp1 || exp2` : 如果已经算出`exp1`为真，那么整个表达式必定为真，于是`exp2`也不必计算

逻辑运算符和逻辑表达式

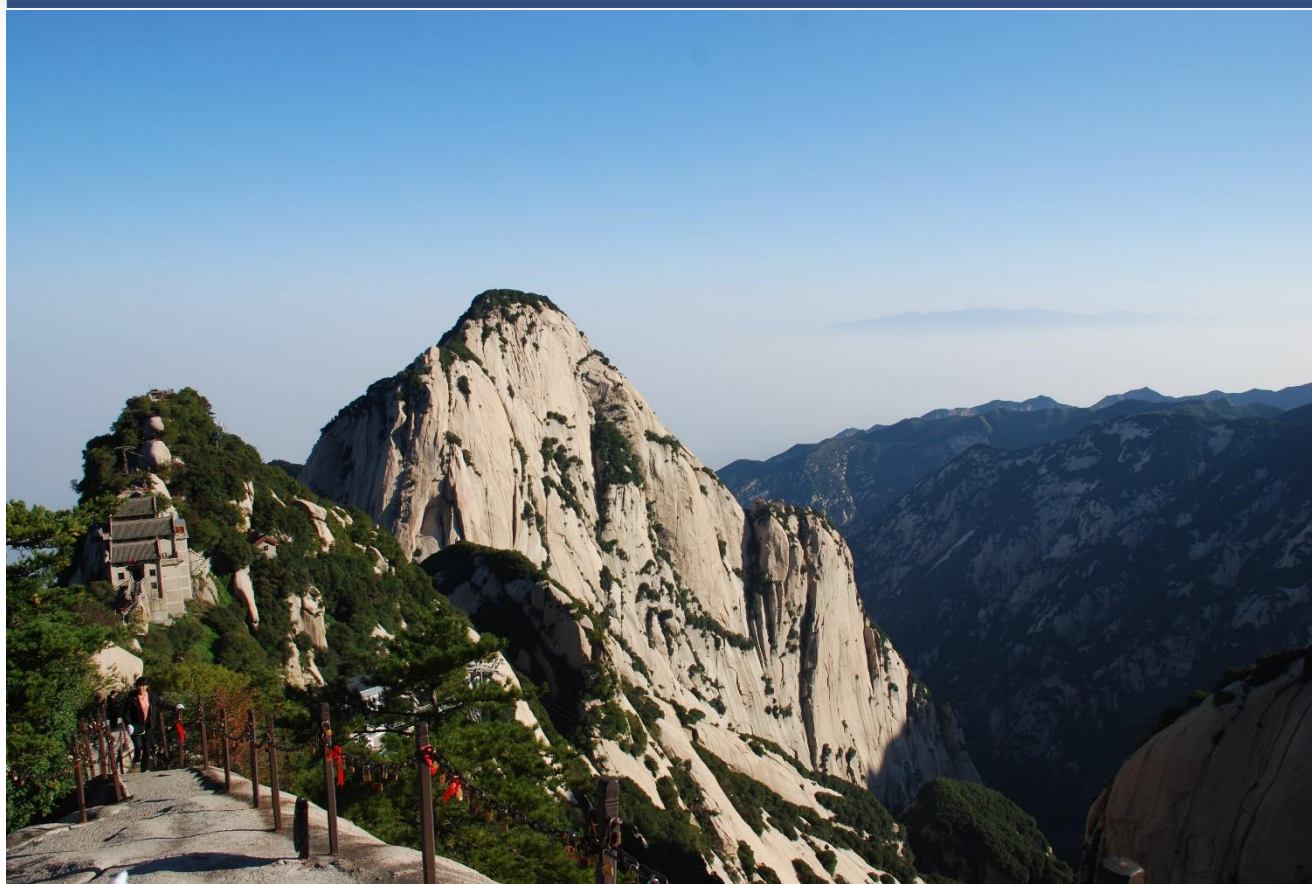
```
#include <iostream>
using namespace std;
int main()
{
    int a = 0, b = 1;
    bool n = (a++) && (b++) ;    // b++不被计算
    cout << a << ", " << b << endl; //输出 1,1
    n = a++ && b++ ;    // a++和b++都要计算
    cout << a << ", " << b << endl; //输出 2,2
    n = a++ || b++ ;    // b++不被计算
    cout << a << ", " << b << endl; //输出 3,2
    return 0;
}
```




北京大学
PEKING UNIVERSITY

信息科学技术学院

强制类型转换运算符 及 运算符优先级



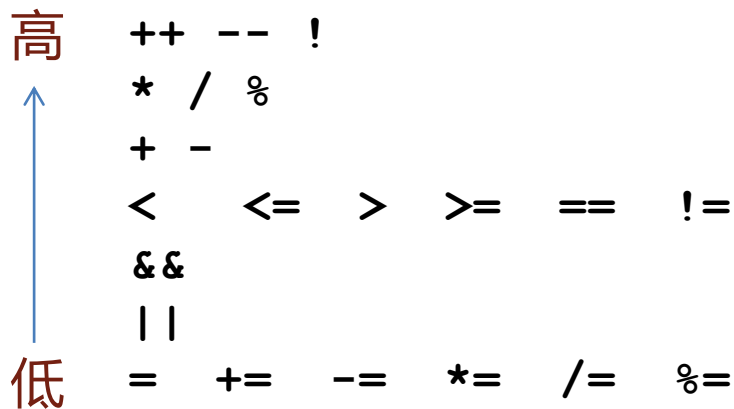
华山北峰

强制类型转换运算符

- 类型名本身就是一个运算符，叫“强制类型转换运算符”
用于将操作数转换为指定类型

```
double f = 9.14;  
int n = (int) f;    //n=9  
f = n / 2;          //f=4.0  
f = double(n) / 2;  //f=4.5
```

部分运算符的优先级



A diagram illustrating the relative precedence of various C++ operators. On the left, a blue arrow points upwards, indicating increasing priority. The word '高' (High) is at the top and '低' (Low) is at the bottom. Operators are listed in rows from highest to lowest priority: ++, --, !; *, /, %; +, -; <, <=, >, >=, ==, !=; &&; ||; and finally =, +=, -=, *=, /=, %=.

高	++	--	!
	*	/	%
	+	-	
	<	<=	> >= == !=
	&&		
低	=	+=	-= *= /= %=

- 可以用()改变运算顺序, 如 `a*(b+c)`。勤用括号以避免优先级错误
- `printf("%d,%d",a+++b,a);` 输出什么?

部分运算符的优先级

高	++	--	!	- (取相反数)					
	*	/	%						
	+	-							
	<	<=	>	>=	==	!=			
	&&								
低	=	+=	--	*=	/=	%=			

- 可以用()改变运算顺序, 如 `a*(b+c)`。勤用括号以避免优先级错误
- `printf("%d,%d",a+++b,a);`
`a+++b` 等价于 `(a++)+b`