

第二章 程序设计基础

C++简明双链教程（李昕著，清华大学出版社）

作者：李昕

PPT制作者：孙百乐

qingline.net/cppbook

目录

01 数据类型

02 整型

03 浮点类型

04 其他数据类型

05 数据类型转换

06 操作符

07 获取用户输入

08 时间处理

09 常用数学函数

10 底层效率分析*

01

数据类型

中国石油大学(华东)

qingline.net/cppbook

1.1 常见数据类型

C++中的常见数据类型包括:

用单引号表示, 一对单引号中**有且仅有一个字符**, 空格也是一个字符, 单引号中不能为空;

true或false
(1或0)



整型			浮点型			字符	字符串	布尔类型
int	short	long long	float	double	long double	char	string	bool
32位	16位	64位	32位	64位	128位			



用双引号表示, 一对双引号中可以为空, 也可以是多个字符;

数据类型和基本输入输出:

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a,d;
6      float b=2.578;
7      char c='a';
8      string s;
9      cin>>a>>d;
10     cin>>s;
11     cout<<"a="<<a<<"<<d="<<d<<"<<a+b="<<a+d<<endl;
12     cout<<"b="<<b<<endl;
13     cout<<"c="<<c<<endl;
14     cout<<s<<endl;;
15     return 0;
16 }
```

变量

常量

样例输入

3 5
我喜欢中国石油大学 (华东)

样例输出

a=3; d=5; a+b=8
b=2.578
c=a
我喜欢中国石油大学 (华东)

注意区分第5行的a和第7行的'a'。其中a是一个变量，被定义为int类型，可以被赋值；'a'是一个常量，代表小写字母a，不能被修改。



变量a



变量b



变量c



变量d

数据类型和基本输入输出:

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a,d;
6      float b=2.578;
7      char c='a';
8      string s;
9      cin>>a>>d;
10     cin>>s;
11     cout<<"a="<<a<<"<<d="<<d<<"<<a+b="<<a+d<<endl;
12     cout<<"b="<<b<<endl;
13     cout<<"c="<<c<<endl;
14     cout<<s<<endl;;
15     return 0;
16 }
```

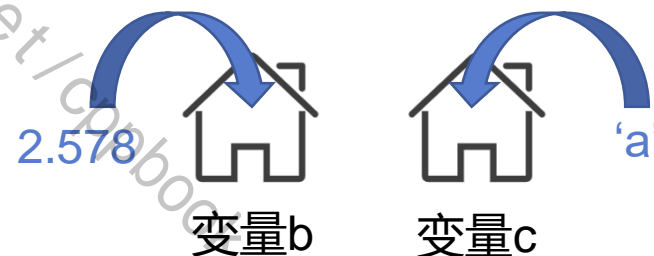
样例输入

3 5
我喜欢中国石油大学 (华东)

样例输出

a=3; d=5; a+b=8
b=2.578
c=a
我喜欢中国石油大学 (华东)

第6-7行中的=表示赋值，将右侧计算的结果赋值给左侧的变量。注意赋值号左边只能写一个变量，也称为左值。不能对多个变量同时赋值，也就是说，左值必须是唯一的。



数据类型和基本输入输出:

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a,d;
6      float b=2.578;
7      char c='a';
8      string s;
9      cin>>a>>d;
10     cin>>s;
11     cout<<"a="<<a<<"<<d="<<d<<"<<a+b="<<a+d<<endl;
12     cout<<"b="<<b<<endl;
13     cout<<"c="<<c<<endl;
14     cout<<s<<endl;
15     return 0;
16 }
```

cin是C++的常用输入方式

cout是C++的常用输出方式

endl是C++中的一个关键字，
表示end of line，即回车换行符。

样例输入

3 5
我喜欢中国石油大学（华东）

样例输出

a=3; d=5; a+b=8
b=2.578
c=a
我喜欢中国石油大学（华东）

cin是C++的标准输入方式，将输入内容赋值给变量；cout是C++的标准输出方式；采用这样的方式，可以让用户交互式的为变量赋值，并进行打印输出。

数据类型和基本输入输出:

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a,d;
6      float b=2.578;
7      char c='a';
8      string s;
9      cin>>a>>d;
10     cin>>s;
11     cout<<"a="<<a<<"<<d<<"<<a+b<<"<<a+d<<endl;
12     cout<<"b="<<b<<endl;
13     cout<<"c="<<c<<endl;
14     cout<<s<<endl;;
15     return 0;
16 }
```

样例输入

3 5

我喜欢中国石油大学 (华东)

样例输出

a=3; d=5; a+b=8

b=2.578

c=a

我喜欢中国石油大学 (华东)

在OJ系统中，如果给定的样例输出比较长，请复制粘贴到自己的程序中。这样可以防止潜在的书写错误，或中英文符号的混淆。

本节要点

索引	要点	正链	反链
T211	区分字符和字符串，单引号中有且仅有一个字符，字符串用双引号		T241 , T243
	洛谷： U269720 (LX201)		
T212	区分变量和常量，掌握标准输入和标准输出。如果要求的输出比较长，或含有特殊字符，请用复制粘贴的方式放到程序输出中，尽量不要手工输入，防止潜在错误的发生。		T271
	洛谷： U269763 (LX202)		
T213	赋值号为一个=，其左侧只能有一个变量，称为左值		T261

1.2 转义字符

转义字符:

C/C++的字符串中以\开头的字符称为转义字符，表示特殊含义的字符。

转义字符在书写上呈现两个字符甚至多个字符，但是实际上代表一个特殊字符。下表列举了一些常见的转义字符。

转义字符	意义	转义字符	意义
\n	换行(LF)，将当前位置移到下一行开头	\'	代表一个单引号（撇号）字符
\r	回车(CR)，将当前位置移到本行开头	\"	代表一个双引号字符
\t	水平制表(HT)（跳到下一个TAB位置）	\?	代表一个问号
\\	代表一个反斜线字符\"'	\0	空字符(NULL)

本节要点

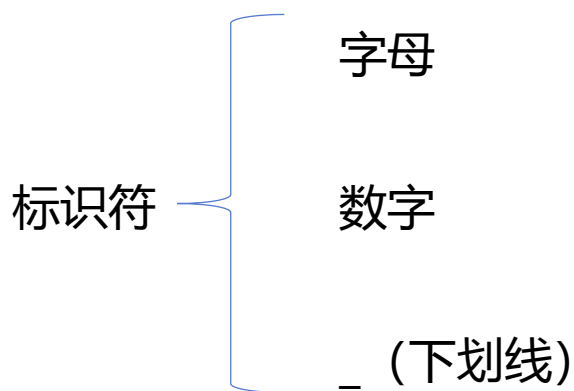
索引	要点	正链	反链
T214	理解字符串的转移字符, \在字符串中的特殊含义, 打印一个\要写\\		T216
	洛谷: U269723 (LX203)		

1.3 标识符

标识符:

标识符是用于表示变量名、常量名或函数名等的字符序列。标识符可以由字母、数字和 _ (下划线) 组成, 变量名必须以字母或 _ (下划线) 开头。标识符是大小写敏感的, 即区分大小写。

以下都是合法的数据类型和标识符, 建议使用有意义的名字作为变量名, 在程序阅读时, 通过变量名就可以理解程序的含义。



```
1  int myNum = 5;           // Integer (whole number)
2  float myFloatNum = 5.99; // Floating point number
3  double myDoubleNum = 9.987868; // Floating point number
4  char myLetter = 'D';     // Character
5  bool myBoolean = true;   // Boolean
6  string myText = "Hello"; // String
```

1.4 C语言的输出方式

1. printf

样例输出

```
a=3 |b= 4|c=0
d=3.14|e=3.14 |f=0.000000
character A
```

```
1  #include <stdio.h> //C语言标准输入scanf和标准输出printf的头文件
2  using namespace std;
3  int main()
4  {
5      int a=3,b=4,c;
6      double d=3.1415926,e=3.1415927,f;
7      printf("a=%-4d|b=%4d|c=%d\n",a,b,c); //注意宽度控制,-表示左对齐
8      printf("d=%.2f|e=%-8.2f|f=%8f\n",d,e,f); //注意精度控制,-表示左对齐
9      printf("character %c\n",'A'); //常量也可以采用占位符形式输出
10 }
```

printf是C语言的屏幕输出方式，其中的%4d，%.2f和%c称为占位符，分别表示整型(int)、单精度浮点型(float)和字符类型(char)，对应变量（a, b, c）的值将在占位符位置进行输出。

1.4 C语言的输出方式

1. printf

```
1  #include <stdio.h> //C语言标准输入scanf和标准输出printf的头文件
2  using namespace std;
3  int main()
4  {
5      int a=3,b=4,c;
6      double d=3.1415926,e=3.1415927,f;
7      printf("a=%-4d|b=%4d|c=%d\n",a,b,c); //注意宽度控制,-表示左对齐
8      printf("d=%.2f|e=%-8.2f|f=%8f\n",d,e,f); //注意精度控制,-表示左对齐
9      printf("character %c\n",'A'); //常量也可以采用占位符形式输出
10 }
```

%.2f表示输出的浮点数精度保留两位小数，如果是双精度浮点型double类型，采用%.2lf的形式。

占位符还可以进行宽度控制，例如%4d表示打印内容占4个字符宽度的整型，且右对齐。%-8.2f表示占10个字符宽度，精度为2的浮点数，-表示左对齐。

样例输出

```
a=3 |b= 4|c=0
d=3.14|e=3.14 |f=0.000000
character A
```

1.4 C语言的输出方式

在以上代码中：

- 注意printf中的精度控制、宽度控制和左对齐；
- 因为%作为占位符的标记，因此如果想输出一个一个%，需要使用两个%%，与\\同理；
- 常量也可以采用占位符的形式进行输出，但是意义不大，因为常量可以直接写进字符串；
- 在变量定义的时候直接进行赋值，叫做变量初始化，例如a, b, c, d；
- c 和 f 在使用前并未进行赋值，因此其结果是任意的。

1.4 C语言的输出方式

变量在使用前必须初始化：

以上程序在运行时会出现警告：

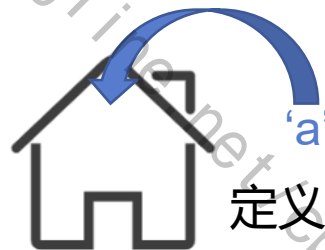


'f' is used uninitialized in this function

当一个变量定义后，会根据指定的数据类型分配内存空间。但是由于未初始化，对应内存空间内的数值不会发生任何改变，把该内存空间中的二进制数值按照变量的数据类型进行解析，因此对应变量的值可以认为是任意值。



定义但未初始化



定义且初始化

PS:这是非常危险的操作，它可能导致代码在本地运行结果正确，但是提交到服务器上在线评测的时候就发生了错误。变量未初始化给出警告，但不是错误，因此程序可以正常执行。

1.4 C语言的输出方式

2. cout

printf 书写比较复杂，还要进行类型控制，在C++中不推荐使用，C++的cout形式输出更加常用。cin 和 cout 能够自动进行类型检测，不需要书写占位符，使用更加简单方便。

cout << “我更简单”;

PS:绝大部分情况下可以使用 cout 进行屏幕打印，但是如果涉及精度和宽度控制时，cout 书写比较复杂，建议采用printf 形式。

本节要点

索引	要点	正链	反链
T215	正确使用占位符，控制宽度、精度等		T231
	洛谷: U269724 (LX204)		
T216	因为%表示占位符，所以输出时用%%表示一个%	T214	
	洛谷: U269725 (LX205)		
T217	变量在使用前必须先赋值		T443

02

整型

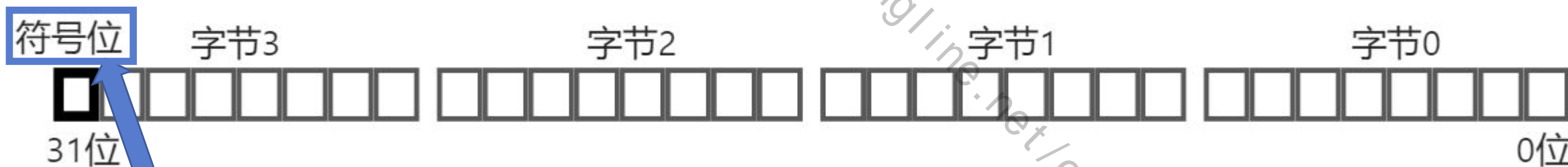
中国石油大学(华东)

qingline.net/cppbook

2.1 整型的数值范围

以 int 类型为例：

最高位作为符号位，剩余31位，因此int类型的数值范围为 $-2^{31} \sim 2^{31}-1$ ，最大值减一是因为0要占一种表示方式。如果最高位不作为符号位，即unsigned int类型，数值范围转换为 $0 \sim 2^{32}-1$ 。



int由连续的4字节构成，共32比特，它在内存中的分配如上图所示

对于有符号数据类型，符号位用0表示“正数”，用1表示“负数”

2.1 整型的数值范围

最大值都有一个减1，这是为0保留一个空间。这些在头文件<climits>定义的常量都可以直接使用。

其它类型：

与之类似，long long为64位的整型，char是8位的整型。下表列举了<climits>头文件中定义的部分常量，从这些常量中可以看到char，int，long long数据类型的表示范围。

类型	字节	最小值 常量名称	最小值	最大值 常量名称	最大值	unsigned最大值 常量名称	unsigned 最大值
char	1	CHAR_MIN	-2^7	CHAR_MAX	$+2^7-1$	UCHAR_MAX	$+2^8-1$
short	2	SHRT_MIN	-2^{15}	SHRT_MAX	$+2^{15}-1$	USHRT_MAX	$+2^{16}-1$
int	4	INT_MIN	-2^{31}	INT_MAX	$+2^{31}-1$	UINT_MAX	$+2^{32}-1$
long long	8	LLONG_MIN	-2^{63}	LLONG_MAX	$+2^{63}-1$	ULLONG_MAX	$+2^{64}-1$

2.1 整型的数值范围

其它类型:

与之类似, long long为64位的整型, char是8位的整型。下表列举了<climits>头文件中定义的部分常量, 从这些常量中可以看到char, int, long long数据类型的表示范围。

类型	字节	最小值 常量名称	最小值	最大值 常量名称	最大值	unsigned最大值 常量名称	unsigned 最大值
char	1	CHAR_MIN	-2^7	CHAR_MAX	$+2^7-1$	UCHAR_MAX	$+2^8-1$
short	2	SHRT_MIN	-2^{15}	SHRT_MAX	$+2^{15}-1$	USHRT_MAX	$+2^{16}-1$
int	4	INT_MIN	-2^{31}	INT_MAX	$+2^{31}-1$	UINT_MAX	$+2^{32}-1$
long long	8	LLONG_MIN	-2^{63}	LLONG_MAX	$+2^{63}-1$	ULLONG_MAX	$+2^{64}-1$

C++对C语言兼容, C语言的函数库在C++中都可以使用。使用时可以采用C语言方式#include<limits.h>; 也可以采用C++方式, 去掉文件后缀.h, 在文件名前加字母c, 例如#include<climits>。

2.1 整型的数值范围

溢出:

因为每种类型都有其存储范围，如果数值过大，无法在限定的空间表示，高位多出的部分就会被自动忽略，结果就可能会发生错误。这种现象被称为**溢出**或**截断操作**。因此在程序设计时，一定要确保对应的数据类型能够保证所有的可能值能被正确的存储。



数据类型的截断：

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      char a = 97;
6      cout<<bitset<32>(97)<<endl;
7      cout<<bitset<8>(a)<<endl;
8      char c = 0x1061;
9      cout<<bitset<32>(0x1061)<<endl;
10     cout<<bitset<8>(c)<<endl;
11     cout<<c<<endl;
12     return 0;
13 }
```

bitset是C++中的一个类，用于将值进行二进制表示，bitset<N>(V)中N表示二进制的位数，V表示需要处理的值。以上代码中将相同的值用不同的二进制位进行表示，帮助理解溢出和截断；

样例输出

```
000000000000000000000000000000001100001
01100001
00000000000000000000000001000001100001
01100001
a
```



```
m>  
td;  
  
32>(97)<<endl;  
8>(a)<<endl;  
61;  
32>(0x1061)<<endl;  
8>(c)<<endl;  
;
```

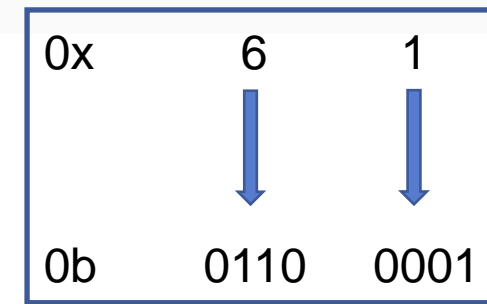
进制。因为16是2的以表示为4位二进制 $0x6=0b0110$ 。当存将每一位十六进制转到一起即可，例如 $0b0001$ 的拼接，即

样例输出

```
000000000000000000000000000000001100001  
01100001  
0000000000000000000000001000001100001  
01100001
```

a

在C++中，0b开头表示二进制，0x开头表示十六进制。因为16是2的4次幂，因此1位十六进制可以表示为4位二进制，例如0x1=0b0001，0x6=0b0110。当存在多位十六进制时，简单地将每一位十六进制转换为二进制，然后顺序拼接到一起即可，例如0x61可以表达为0b0110和0b0001的拼接，即0b01100001；



数据类型的截断:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      char a = 97;
6      cout<<bitset<32>(97)<<endl;
7      cout<<bitset<8>(a)<<endl;
8      char c = 0x1061;
9      cout<<bitset<32>(0x1061)<<endl;
10     cout<<bitset<8>(c)<<endl;
11     cout<<c<<endl;
12     return 0;
13 }
```

同理, 将变量c赋值为十六进制0x1061时, 高位部分0x1000超出1字节的表示范围, 被自动截断。剩余部分0x61等于十进制的97, 即小写字母a的ASCII码, 因此变量c会输出字符a。

注意这里的字面常量97本质上是一个int整数(32位), 对应的二进制为0b1100001。字符变量a只能存放1个字节, 即8个二进制位, 存储空间超限。所以会发生截断操作。

截断的规则: 低位保留, 高位舍弃。97<256(1字节)存储大小, 因此第9-31位的高位部分全部为0, 截断后a的值依旧是97。

样例输出

```
000000000000000000000000000000001100001
01100001
```

```
0000000000000000000000001000001100001
01100001
a
```

数据类型的截断：

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      char a = 97;
6      cout<<bitset<32>(97)<<endl;
7      cout<<bitset<8>(a)<<endl;
8      char c = 0x1061;
9      cout<<bitset<32>(0x1061)<<endl;
10     cout<<bitset<8>(c)<<endl;
11     cout<<c<<endl;
12     return 0;
13 }
```

0001000001100001

截断

01100001

00000000000000000000000000000000000001100001

截断

01100001

样例输出

```
000000000000000000000000000000001100001
01100001
```

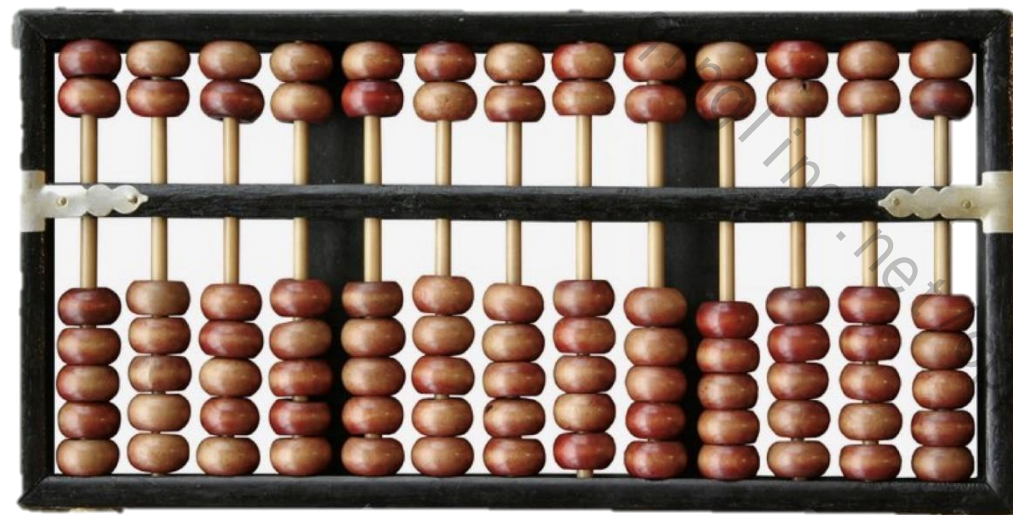
a

本节要点

索引	要点	正链	反链
T221	理解整型的二进制存储形式；数值范围与存储空间的关系；溢出和截断产生的原因；掌握有符号整型和无符号整型（unsigned）的区别。		T253 , T2A2
T222	理解字面量0b和0x的表示方法，掌握二进制与十六进制的关系		T271

2.2 整数N进制转十进制

所谓十进制就是逢十进一，二进制就是逢二进一，进一步泛化，N进制就是逢N进一。可以用指数求和的形式表示一个整数，例如 $1978 = 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$ 。对于一个二进制可以用同样的办法转换为十进制，例二进制0b10110可以表示为 $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22$ 。N进制数都可以采用这种方法转换为十进制。



二进制转十进制:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      unsigned int c = 0b10110; //0b开始表示一个二进制数
6      cout<<c<<'\\t'<<bitset<8>(c)<<endl;
7      return 0;
8  }
```

样例输出

22 00010110

第6行:

第一个输出, 因为没有指定进制, 按照默认的十进制进行输出, 得到结果22。

第二个输出`bitset<8>(c)`表示把变量`c`按照8位二进制进行表示, 因此输出结果为 00010110。

二进制转十进制:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      unsigned int c = 0b10110; //0b开始表示一个二进制数
6      cout<<c<<'\\t'<<bitset<8>(c)<<endl;
7      return 0;
8  }
```

样例输出

22 00010110

在线评测网站上的题目经常出现数值范围为 $-10^9 \sim 10^9$ 或 $-10^{18} \sim 10^{18}$ 等类似的数值范围说明。但是数据类型根据内存占用位数只能比较容易获取二进制表示的范围，不好和这些数值范围做比较。可以通过估算法确定采用何种数据类型。这种方法在线评测的时候经常会用得到。

估算法：因为 $2^{10} = 1024$ 约等于 10^3 ，也就是说每10位二进制就可以表示3位十进制。因此int（31位）可以保留 10^9 以下的整数，long long（63位）可以保留 10^{18} 以下的整数。

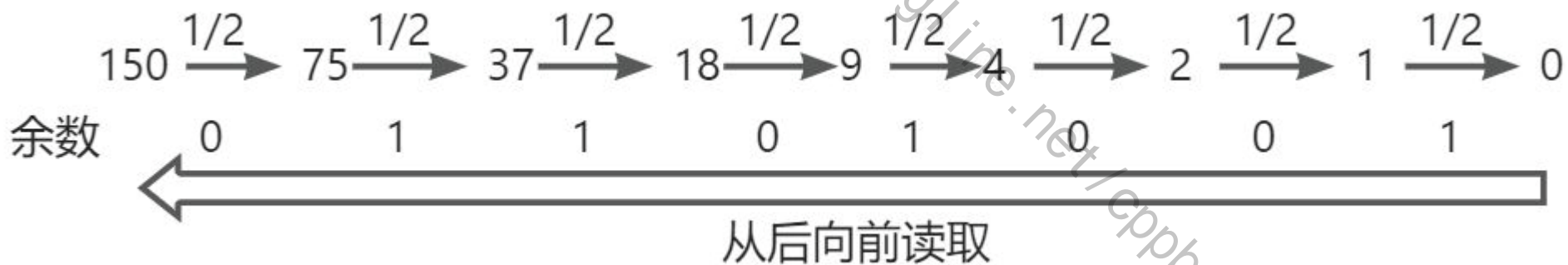
本节要点

索引	要点	正链	反链
T223	掌握N进制转换为十进制的算法		
	洛谷: U269727 (LX206)		
T224	能够根据题目给定的数值范围, 通过估算法确定整型的数据类型		T216
	洛谷: U269729 (LX207), U269742 (LX210), U269748 (LX213)		

2.3 整数十进制转换为N进制

除2取余法:

整数十进制转换为二进制采用的经典方法称为**除2取余法**。具体做法是：用2整除十进制整数，可以得到一个商和余数；再用2去除商，又会得到一个商和余数，如此进行，直到商为0停止。最后把先得到的余数作为二进制数的低位有效位，后得到的余数作为二进制数的高位有效位，依次排列起来，就得到了对应的二进制。除N取余法可以将十进制转换为N进制。



本节要点

索引	要点	正链	反链
T225	掌握十进制转换为N进制的算法		T471
	洛谷: U269732 (LX208)		

03

浮点类型

中国石油大学(华东)

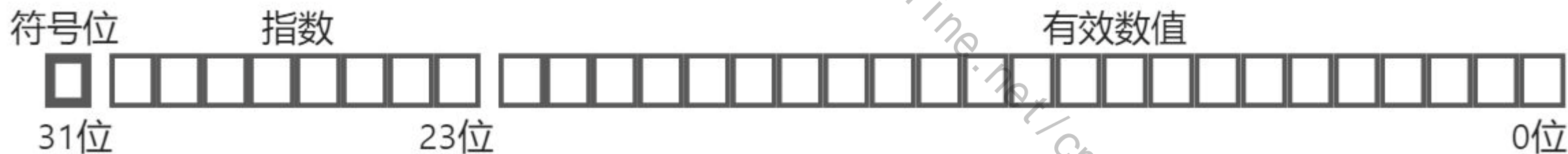
qingline.net/cppbook

3.1 浮点数的内存表示

以 float 类型为例：

浮点数就是所谓的小数，一个float类型的对象占据4个字节（32比特）。这32个比特以类似于科学计数法的形式来表达一个浮点数，即浮点数 $= (1 + \text{有效数值}) \times 2^{\text{指数}}$ 。

按照IEEE 754标准，如图所示，最高的1位(第31位)用做符号位，接着的8位(第23-30位)是指数E，剩下的23位(第0-22位)为有效数字M。例如 $3.14159 = 1.570795 \times 2^1$ ，其中0.570795表示有效数值，记录了精度信息， 2^1 中的1表示指数。



按照这样的表示方法，浮点数的表示精度是有限的，float类型大约能够存储小数点后6位精度的十进制数。double类型为64位，它的表示精度更大一些，大约为12-15位精度。

代码分析:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      float a=1;
6      int b = 1;
7      printf("%d\n",a);
8      printf("%f\n",b);
9  }
```

样例输出

0
0.000000

int类型分配4字节空间，然后将变量b和这个空间绑定。按照整型的存储标准，将1转换为二进制，存储到对应的空间中。当后继代码使用b时，将读取这块空间，并按照整型的存储格式进行解读。

以上代码通过占位符将整型变量按照浮点数方式解读，将浮点数按照整型方式进行解读，虽然初始值都为1，但是得到的结果都是0，这就是存储内容和解读方式不对应的原因。

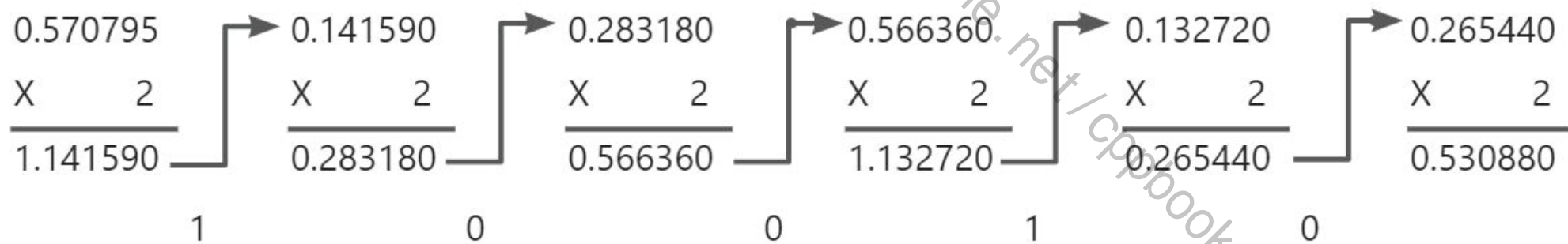
本节要点

索引	要点	正链	反链
T231	整型与浮点型的存储格式不同，因此解析方式也不同	T215	
T232	浮点数存储的精度有限		T234

3.2 纯小数十进制转换为二进制

乘2取整法：

纯小数十进制转换为二进制采用的经典方法称为**乘2取整法**。具体做法是：用2乘十进制小数，将积的整数部分取出，再用2乘余下的小数部分，又得到一个积，再将积的整数部分取出，如此进行，直到积中的小数部分为零，或者达到所要求的精度为止。最后把先得到的余数作为二进制数的高位有效位，后得到的余数作为二进制数的低位有效位，依次排列起来，就得到了对应的二进制。连接为二进制的时候从左向右，**这与除2取整法是相反的**。



3.2 纯小数十进制转换为二进制

浮点数无法精确比较：

可以尝试用该方法转换其他纯小数，就会发现除了 2^{-n} 或多个 2^{-n} 之和外，其他数值转换为二进制小数时都是无限循环小数，而每个浮点数在计算机中的表示一定是有**精度限制**。由此可以得到结论，**绝大部分十进制小数在计算机中都无法精确存储**。

例如：

0.570795

$\times 2$
1.141590

1

0.141590

$\times 2$
0.283180

0

0.283180

$\times 2$
0.566360

0

0.566360

$\times 2$
1.132720

1

0.132720

$\times 2$
0.265440

0

0.265440

$\times 2$
0.530880

0.10010的十进制为 $1 \times 2^{-1} + 1 \times 2^{-4} = 0.5625$

对于指定的浮点数据类型，该数据类型的有效数值部分的二进制位越长，这个精度差就会越小。

不相等，存在精度差

浮点数的精确比较:

```
1  #include <iostream>
2  #include <cmath> //函数fabs的头文件
3  using namespace std;
4  int main()
5  {
6      float a=0.9876;
7      int b = 9876;
8      cout<<(a==0.9876)<<endl; //错误浮点数比较方式
9      cout<<(fabs(a-0.9876)<0.0001)<<endl; //正确浮点数比较方式
10     cout<<(b==9876)<<endl; //整型因为能精确存储,可以精确比较
11 }
```

样例输出

0
1
1

相同的浮点数, 因为精度问题, 在内存中存储的内容可能会有细微差别, 而等于运算是一种精确的比较, 导致第8行输出false。按照科学计算的规定, 第9行才是浮点数的正确比较方式, 其中的0.0001是用户自定义的误差阈值, 根据问题需要的精度确定。第10行中是两个整数的比较, 整型能精确存储, 因此可以精确比较。

C++中0表示false, 1表示true

本节要点

索引	要点	正链	反链
T233	掌握十进制纯小数转换为二进制的算法，即乘2取整法		
	洛谷： U269740 (LX209)		
T234	绝大部分十进制浮点数无法精确存储，因此无法精确比较。必须掌握浮点数的误差比较法	T232	T291
	洛谷： U269744 (LX211)		

04

其他数据类型

中国石油大学(华东)

qingline.net/cppbook

4.1 字符

ASCII表:

计算机中的所有内容都是以二进制形式进行存储的，数值类型可以转换为二进制存储到计算机中，但是字符是不可以的。为了存储字符，只能先把字符映射成数值，把读到的数值按照字符进行理解。国际统一的字符映射表称为ASCII表。

ASCII 值	控制字 符	ASCII 值	控制字 符	ASCII 值	控制字 符	ASCII 值	控制字 符
0	NUT	32	(space)	64	@	96	,
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p

ASCII 值	控制字 符	ASCII 值	控制字 符	ASCII 值	控制字 符	ASCII 值	控制字 符
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	`
31	US	63	?	95	_	127	DEL

4.1 字符

只需要知道这个表的存在，了解每个字符在计算机中实际上是一个整数。并不需要记住每个字符对应的数值。但以下规律是需要掌握：

- 数字字符是连续的，小写字符是连续的，大写字符是连续的，但是大写和小写字符是不连续的。
- 一对匹配的括号，(),[],{>,ASCII码值相差不超过2。
- 对于一个字符，可以通过加减运算转换为另外一个字符，两个字符相减可以得到二者之间在ASCII码表上的差距，但是两个字符相加是没有物理含义的。

小写转大写:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char b = 'b';
6      cout<<b<<" , "<<(int)b<<endl;
7      b = b - 'a' + 'A';
8      cout<<b<<" , "<<(int)b<<endl;
9      return 0;
10 }
```

样例输出

b, 98
B, 66

第5行中的b是一个变量, 'b'是一个常量字符, 必须能将二者进行清晰区分。

第6行将变量b分别以char类型和int类型进行输出, int类型的输出实际上就是该字符的ASCII码值。其中(int)b表示将b强制转换为int类型进行输出, 转换的结果存储在一个临时空间中, b本身没有发生任何改变。由此可见, char就是1字节长度的整型, 默认时按字符解读, 本质上就是一个整数。

将一个浮点型强制转换为整型时, 小数部分会被舍掉, 并不会发生四舍五入。例如int a = (int)2.7;。则a的值为2。

小写转大写:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char b = 'b';
6      cout<<b<<"", "<<(int)b<<endl;
7      b = b - 'a' + 'A';
8      cout<<b<<"", "<<(int)b<<endl;
9      return 0;
10 }
```

样例输出

b, 98
B, 66

第7行将一个小写字母转换为对应的大写字母。因为字符本身的连续性，因此对应大小写字母之间的差值都是相等的，即'a'-'A'。很多初学者将其写为b=b-32;。虽然结果是正确的，但是32不具有实际含义，代码阅读性差，强烈建议采用第7行的方式进行大小写字母转换。

4.1 字符

字符判断和转换函数：

在C语言标准头文件<cctype>中包含了一系列字符判断和转换函数，其作用与函数名称相同，建议使用。

判断函数都是以is开头，返回类型为int类型，非0表示true，0表示false；

函数名	作用	函数名	作用	函数名	作用
isalpha	是否为字母	isdigit	是否为数字字符	isalnum	是否为字母或数字字符
islower	是否为小写	isupper	是否为大写	isspace	是否为空白符
tolower	转换为小写字符	toupper	转换为大写字符		

空白符不是指空格，而是包括空格、制表符和回车符等

转换函数都是以to开头，在进行转换前，判断了输入是否符合要求，因此使用更加安全；

代码分析:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      char a = 97,b=97;           //ex开始表示一个十六进制数
6      cout<<bitset<8>(a)<<endl;
7      cout<<bitset<8>(b)<<endl;
8      char c = a+b;
9      cout<<bitset<8>(c)<<endl;
10     printf("%d\n",c);           //与cout<<(int)c<<endl;等价
11     return 0;
12 }
```

样例输出

```
01100001
01100001
11000010
-62
```

char在本质上就是一个无符号的8位整型（1字节），因此按照整型执行加分运算。结果c的首位1被作为符号位，表示负数，其余部分1000010是62的补码，补码的详细定义参见10.1节。因此c按照整型输出结果为-62。

随堂练习

将一个数字字符转换为对应的数字，例如将'5'转换为5。

本节要点

索引	要点	正链	反链
T241	掌握字符类型就是1字节的整型，可以直接作为整型进行数值运算，一个字符通过偏移（加减一个整数）可以得到另外一个字符。	T211	T851
	洛谷： U269746 (LX212)		
T242	掌握大小写转换，数字和数字字符直接转换的正确方法 建议使用<cctype>库中的字符判断和转换函数		T549
	洛谷： U269748 (LX213)， U270340 (LX308)		

4.2 字符串

字符串类型:

C语言中用字符数组表示字符串，对于初学者比较复杂。但C++中提出了字符串类型，使字符串的操作变得容易。string 是 C++ 中常用的一个类，使用 string 类需要包含头文件<string>。

三种对字符串进行初始化的方式

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      string s1;
6      string s2 = "c plus plus";
7      string s3 = s2;
8      string s4 (5, 's');
9      cout<<s2.length()<<'\\t'<<s4.size()<<endl; //获取字符串的长度，length和size等价
10     cout<<"第二个字符是："<<s3[2]<<endl; //获取第2个字符，注意语法形式
11     return 0;
12 }
```

定义字符串

//用C常量字符串进行初始化
//用s2对s3进行初始化
//用5个s构成字符串进行初始化，即"sssss"

样例输出

11 55
p

4.2 字符串

字符串类型:

C语言中用字符数组表示字符串，对于初学者比较复杂。但C++中提出了字符串类型，使字符串的操作变得容易。string 是 C++ 中常用的一个类，使用 string 类需要包含头文件<string>。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      string s1;           //空字符串
6      string s2 = "c plus plus"; //用C常量字符串进行初始化
7      string s3 = s2;      //用s2对s3进行初始化
8      string s4 (5, 's');  //用5个s构成字符串进行初始化，即"sssss"
9      cout<<s2.length()<<'\t'<<s4.size()<<endl; //获取字符串的长度，length和size等价
10     cout<<"第二个字符是："<<s3[2]<<endl; //获取第2个字符，注意语法形式
11     return 0;
12 }
```

第9行length和size函数都可以获取字符串中字符的数量，二者完全等价，只是因为历史原因才出现了两个名字。

字符串是由字符构成，可以采用字符串名称[i]的方式获取字符串的第i个字符，注意下标i从0开始。

样例输出

11 55
p

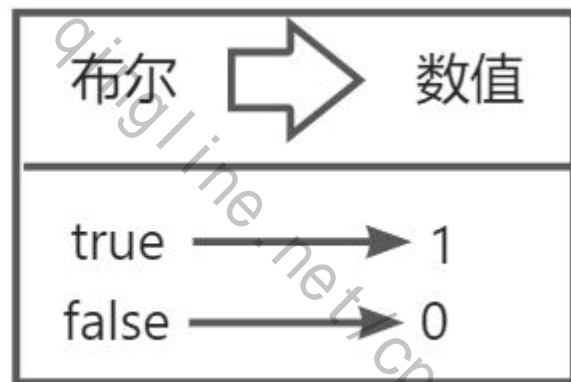
本节要点

索引	要点	正链	反链
T243	掌握字符串的初始化、赋值、获取长度和获取第i个字符的语法形式。	T211	
	洛谷: U269748 (LX213)		

4.3 布尔类型

布尔类型:

C/C++中的布尔值为true/false，其实这是两个宏，本质上就是1/0。这是布尔值和整型值的对应关系。但是整型值映射到布尔值有一个特殊的规定：非0值映射为true，0映射为false，特别强调这里的非0值是包括负数的。



代码分析：

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

上面代码中求解非0值的数量，就是充分利用了整型和布尔型之间的转换规则。

样例输入

样例输出

3 0 5

2

a的输入值为整数3，被强制转换为布尔类型后变成了true，随后因为加法运算需要整型值，又被作为1处理。

同样道理，b最终为0，c最终为1，因此输出结果为2。

代码分析:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      bool a,b;
7      cin>>boolalpha>>a>>b;
8      cout<<a<<' '<<b<<endl;
9      cout<<boolalpha<<a<<' '<<b<<endl;
10     return 0;
11 }
```

布尔类型在屏幕打印时默认输出为1/0，如果想输入或输出true/false，需要在输出前增加boolalpha。

样例输入	样例输出
true false	1 0 true false

本节要点

索引	要点	正链	反链
T244	数值->布尔: 非0值映射为true, 0映射为false; 布尔->数值: true和1等价, false和0等价		T267
	洛谷: U269749 (LX214), U270323 (LX304)		
T245	输入输出true/false要用boolapha		
	洛谷: U269744 (LX211), U269749 (LX214)		

05

数据类型转换

中国石油大学(华东)

qingline.net/cppbook

5.1 隐式类型转换

隐式转换：

为了让计算机执行算术运算，通常要求操作数有**相同的大小**（即位的数量相同），并且要求**存储的方式也相同**。计算机可能可以直接将两个32位整数相加，但是不能直接将32位整数和64位整数相加，也不能直接将32位整数和32位浮点数相加。另一方面，C/C++语言允许在表达式中混合使用基本数据类型。在单独一个表达式中可以组合整数、浮点数、甚至是字符。

例如：

int(32位) + long long(64位)

int + float



C/C++允许，而计算机不允许

5.1 隐式类型转换

在这种情况下C/C++语言编译器可能需要生成一些指令将某些操作数转换成不同类型，使得硬件可以对表达式进行计算。

需要转换：

int(32位) + long long(64位)

转换

int(64位) + long long(64位)

int + float

转换

float + float

因为编译器可以自动处理这些转换而无需程序员介入，所以这类转换称为隐式转换 (implicit conversion) 。

5.1 隐式类型转换

当发生下列情况时会进行隐式转换：

1. 当算术表达式或逻辑表达式中操作数的类型**不相同**时。
2. 当赋值运算符右侧表达式的类型和左侧变量的类型**不匹配**时。

隐式类型转换规则：

C/C++ 自动转换的基本原则是：**低精度类型向高精度类型转换，有符号类型向无符号类型转换。**

具体如下：



由上图可以看到，当数据类型的空间**小于int**时，**都会转换为int**，这是“**整型提升**”的原因，详见本章5.3节。

另一个需要注意的是虽然long long有64位，float只有32位，但是二者进行运算的时候，会转换为float类型。可以认为**整型与浮点型进行运算时，会转换为浮点型。**

5.1 隐式类型转换

C++可以使用typeid(变量).name()的方式获取变量的数据类型。数据类型和返回名称的对应关系如下表：

类型	名称	类型	名称	类型	名称	类型	名称
char	c	unsigned char	h	short	s	unsigned short	t
int	i	unsigned int	j	long long	x	unsigned long long	y
float	f	double	d	bool	b	long double	e

以下代码以 + 运算为例，展现了不同数据类型之间进行运算时的隐式类型转换：

```
1  #include <iostream>
2  using namespace std;
3  #define type(obj) typeid(obj).name()
4
5  int main()
6  {
7      char a;
8      unsigned char b;
9      bool ba;
10     cout<<type(a)<<' '<<type(b)<<' '<<type(ba)<<' '<<type(a+b)<<' '<<type(a+ba)<<endl;
11     short sa;
12     unsigned short sb;
13     cout<<type(sa)<<' '<<type(sb)<<' '<<type(sa+sb)<<' '<<type(sa+a)<<endl;
14     int ia;
15     unsigned int ib;
16     cout<<type(ia)<<' '<<type(ib)<<' '<<type(ia+ib)<<endl;
17     long long lla;
18     unsigned long long llb;
19     cout<<type(lla)<<' '<<type(llb)<<' '<<type(lla+llb)<<' '<<type(lla+ia)<<endl;
20     float fa;
21     double lfa;
22     cout<<type(fa)<<' '<<type(lfa)<<' '<<type(fa+ia)
23         <<' '<<type(fa+lla)<<' '<<type(fa+lfa)<<endl;
24     return 0;
25 }
```

样例输出

```
chbii
stii
ijj
xyyx
fdffd
```

5.1 隐式类型转换

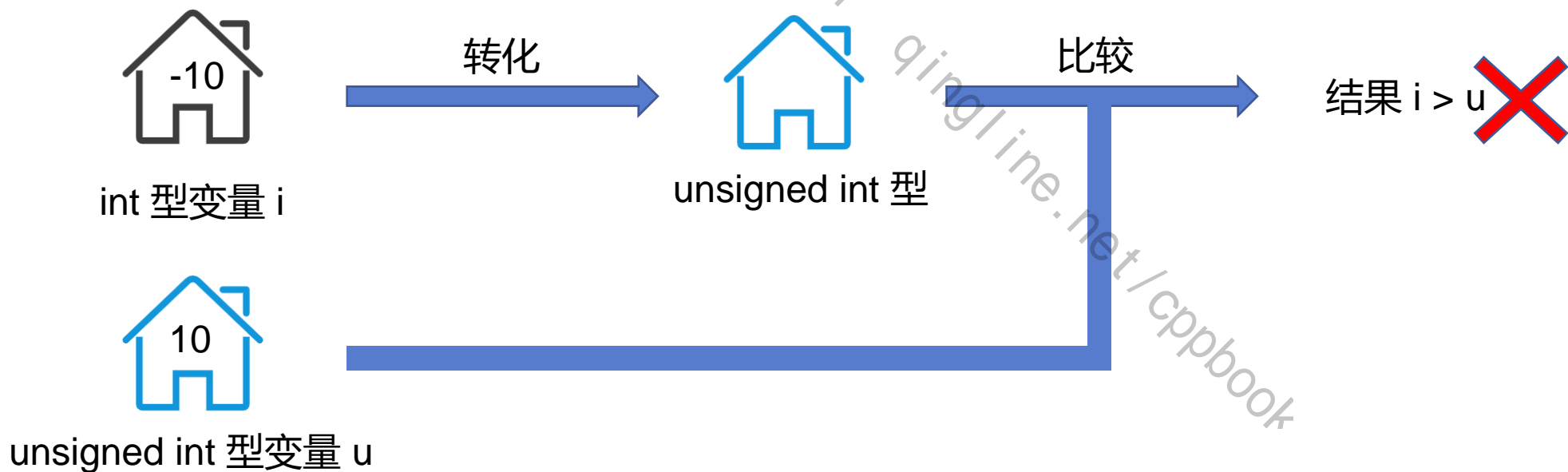
注意某些隐蔽的编程错误：

当把有符号操作数和无符号操作数整合时，会通过把符号位看成数的位的方法把有符号操作数“转换”成无符号的值。这条规则可能会导致某些隐蔽的编程错误。



5.1 隐式类型转换

假设int型的变量i的值为-10，而且unsigned int型的变量u的值为10。如果用 < 运算符比较变量i和变u，那么期望的结果应该是1（真）。但是，在比较前，变量i转换成为unsigned int类型。因为负数不能被表示成无符号整数，所以转换后的数值将不再为-10，而是一个大的正数（将变量i中的位看作是**无符号数**）。因此i < u比较的结果将为0。



5.1 隐式类型转换

由于此类陷阱的存在，所以最好尽最避免使用无符号整数，特别是不要把它和有符号整数混合使用。



代码分析:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      int a = -1;
6      cout<<(a<sizeof(a))<<endl;
7      cout<<bitset<32>((unsigned int)a)<<'\\t'<<((unsigned int)a)<<endl;
8      return 0;
9  }
```

样例输出

```
0
11111111111111111111111111111111 4294967295
```

-1显然小于4，但是为什么输出结果为false(0)呢？

因为sizeof返回的数据类型为unsigned int，在进行比较操作时，做了隐式类型转换，将a转换为unsigned int类型，因此将符号位的1作为数值看待，-1就变成了一个非常大的数。这里-1的二进制是补码形式，详见本章10.1节。

5.1 隐式类型转换

混合运算的计算顺序：

除了类型转换，混合运算时还要注意运算的顺序，否则可能产生错误。

数据溢出错误：例如三个整数 a , b , c ，数学上保证 $a*b$ 一定能被 c 整除，在书写时，一定要写成 $a*b/c$ ，而不能写成 $a/c*b$ 或 $b/c*a$ ，因为除法可能产生小数。



本节要点

索引	要点	正链	反链
T251	当进行混合运算时，低精度类型会自动转换为高精度类型		T291
	洛谷： U269751 (LX215)， U269761 (LX223)		

5.2 显式类型转换

显示转换:

C/C++语言还允许程序员通过使用强制运算符执行显式转换 (explicit conversion) 。

```
1  #include <iostream>
2  #include<cmath>
3  using namespace std;
4  int main()
5  {
6      double a=0.3,b=0.6;
7      int ai = (int)a;    //C语言格式
8      int bi = int(b);    //C++推荐格式
9      cout<<ai<<'\\t'<<bi<<endl;
10     return 0;
11 }
```

样例输出

0 0

C语言的类型转换格式为: (目标类型) 表达式;
C++推荐的格式为: 目标类型 (表达式)

浮点型转换为整型, 直接舍掉小数部分取整, 不存在四舍五入。

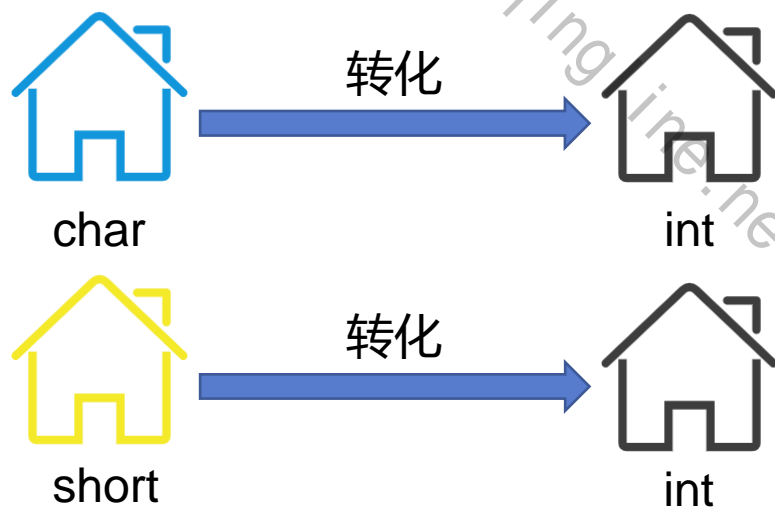
本节要点

索引	要点	正链	反链
T252	显示类型转换的方法，特别注意浮点数转换整数时会舍弃小数取整		T256 , T622
	洛谷: U269740 (LX209)		

5.3 整型提升*

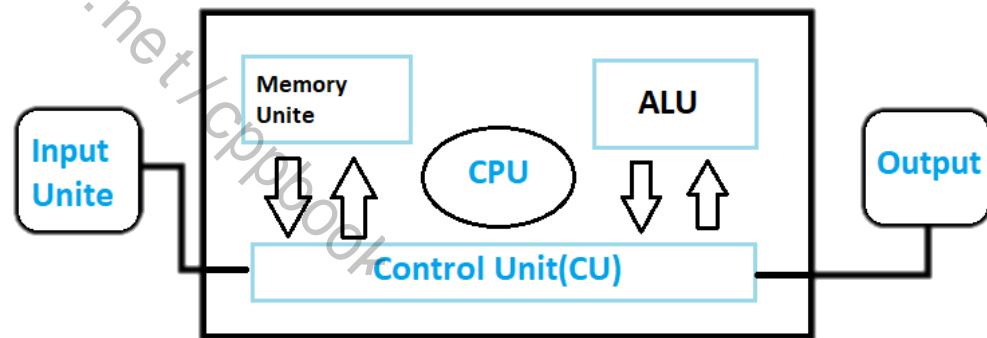
整型提升:

C/C++的整型算数运算总是至少以默认的整型（int型）的精度来进行，也就是说参与运算的操作数最小也不能小于4个字节的精度，如若精度小于4个字节该操作数就必须提升成整型的精度。为了获得这个精度，表达式中字符型（char，1字节）和短整型（short，2字节）操作数在使用之前会被转换为普通整型，这种转换被称为：整型提升。



5.3 整型提升*

这主要是因为表达式的整型运算是由CPU的**整型运算器 (ALU)** 执行，而该运算器操作对象的字节长度一般就是int型的字节长度。因此CPU是无法实现直接对2个char类型的操作数的运算，而是通过先转换为CPU内整型操作数的标准长度然后再进行加法运算。



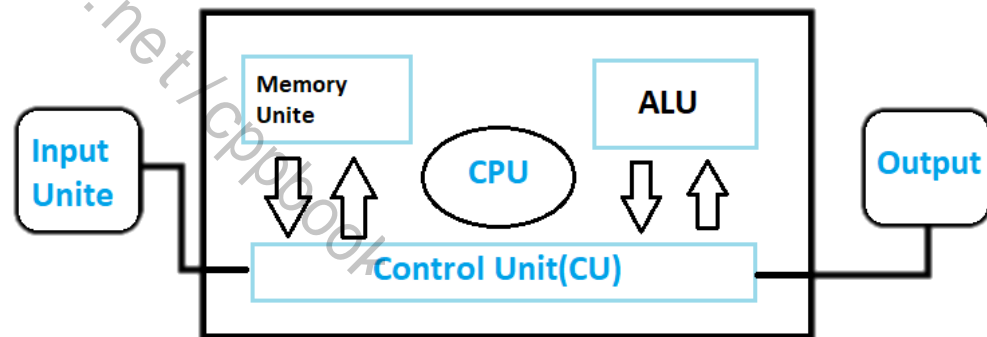
5.3 整型提升*

整型提升前提:

当表达式中出现长度可能小于int型的整型值时，才须要对该值进行整型提升转换为int或unsigned int型，然后再送入CPU去执行运算。

整形提升规则:

对于有符号的整形变量，整型提升是在高位补变量的符号位；
而对于无符号的整型变量，整型提升是直接高位补0。



代码分析:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      char a = 97;
5      cout<<sizeof(a)<<endl;
6      cout<<sizeof(a+a)<<endl;
7      return 0;
8  }
```

样例输出

1
4

char是一个字节，当让char参与加法运算时，被自动提升为int，因此结果为4个字节。

代码分析:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      char a = 0xb6;
6      int b = 0xb6000000;
7      cout<<(a==0xb6)<<endl;
8      cout<<(b==0xb6000000)<<endl;
9      cout<<bitset<32>(0xb6)<<'\\t'<<sizeof(0xb6)<<endl;
10     cout<<bitset<32>(a)<<'\\t'<<sizeof(a)<<endl;
11     cout<<bitset<32>(b)<<'\\t'<<sizeof(b)<<endl;
12     return 0;
13 }
```

仔细观察第9-10行的输出结果，如果按照1字节比较
(需修改)

样例输出

```
0
1
0000000000000000000000000000000010110110 4
1111111111111111111111111111111110110110 1
1011011000000000000000000000000000000000 4
```

第7行结果为false (0)，主要因为在进行关系运算时，char类型变量进行了整型提升，而且提升时因为符号位为1，前面补1（见第10行输出），0xb6默认为int类型，其结果当然在32位下前面全部为0，因此二者不相等。而int类型不需要进行整型提升，因此第8行结果为true (1)。

5.4 类型转换的精度损失

类型转换的精度损失:

提升数据的精度通常是一个平滑无损害的过程。但是降低数据的精度可能导致问题。原因很简单：一个较低精度的类型存储空间不够大，不能存放一个具有更高精度的完整的数据。



代码分析:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      long long b = 654321 * 654321;
5      cout << b << endl;
6      long long c = 654321LL * 654321;
7      cout << c << endl;
8      long long d = (long long)654321 * 654321;
9      cout << d << endl;
10     return 0;
11 }
```

样例输出

```
-1360758559
428135971041
428135971041
```

//LL表示long long类型

654321 * 654321的结果虽然能在long long的表示范围内，但是654321默认数据类型为int，654321 * 654321的计算结果也为int，保存在一个临时的int类型的空间中。第4行将int类型的中间结果转换为long long，转换之前结果已经溢出。因此第5行输出结果错误。

代码分析:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      long long b = 654321 * 654321;
5      cout << b << endl;
6      long long c = 654321LL * 654321;
7      cout << c << endl;
8      long long d = (long long)654321 * 654321;
9      cout << d << endl;
10     return 0;
11 }
```

样例输出

```
-1360758559
428135971041
428135971041
```

//LL表示long long类型

第6行将一个操作数转换为long long，计算结果向高精度的long long进行类型转换，存储临时结果的变量也是long long类型的，因此c能显示正确的结果。

代码分析:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      long long b = 654321 * 654321;
5      cout << b << endl;
6      long long c = 654321LL * 654321;
7      cout << c << endl;
8      long long d = (long long)654321 * 654321;
9      cout << d << endl;
10     return 0;
11 }
```

样例输出

```
-1360758559
428135971041
428135971041
```

//LL表示long long类型

第8行做了强制类型转换，将第一个数转换为long long类型，转换之后的结果也是放在一个临时存储空间中，用这个中间结果与第2个操作数654321进行运行，结果也是long long类型的。保证了计算结果的正确性。

本节要点

索引	要点	正链	反链
T253	溢出和截断会保存在中间临时变量里，溢出发生后的类型转换不起作用	T221	

5.5 四舍五入和趋零舍入

C/C++中提供了round函数，对浮点数进行四舍五入取整，该函数在<cmath>头文件中。

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main(){
5      printf("round(50.2) = %f \n", round(50.2));
6      printf("round(50.8) = %f \n", round(50.8));
7      printf("round(0.2) = %f \n", round(0.2));
8      printf("round(-50.2) = %f \n", round(-50.2));
9      printf("round(-50.8) = %f \n", round(-50.8));
10     cout<<round(M_PI*100)/100<<endl;    //保留两位数字精度
11     return
12 }
```

样例输出

```
round(50.2) = 50.000000
round(50.8) = 51.000000
round(0.2) = 0.000000
round(-50.2) = -50.000000
round(-50.8) = -51.000000
3.14
```

round函数可以对整数进行四舍五入，第10行是它的一种特殊用法，可以保留指定精度进行四舍五入。首先乘100进行整数的四舍五入，然后再除100，回退到原有的值域范围。从而实现了指定精度的四舍五入。

随堂练习

两个double类型变量a和b，a为已知，且有5位精度。采用round函数用a对b进行赋值，且b四舍五入地保留a的两位精度。

例如a=3.14159，则b应该为3.14。如果a=3.14591，则b应该为3.15。

在计算中可能会大概率遇到的问题：

```
1  #include <iostream>
2  #include<cmath> //round函数的头文件
3  using namespace std;
4  int main(){
5      double x=99;
6      x=x/100;
7      int a=2/(1-x);
8      cout<<a<<endl;
9      cout<<2/(1-x)<<endl;
10     printf("%lf\n",2/(1-x));
11     printf("%.14lf\n",2/(1-x));
12     a = round(2/(1-x));
13     cout<<a<<endl;
14     printf("%.14lf\n",round(2/(1-x)));
15     return 0;
16 }
```

数学推导悖论：经过简单的数学运算，可知第7行的运算结果应该为200，但第8行的输出结果却是199

//输出为199，是一个错误的结果
//200，结果正确，趋零舍入的原因
//6位精度时结果正确，第7位四舍五入的原因
//14位精度时，整数部分为199
//结果为200
//小数部分四舍五入为整数

样例输出

```
199
200
200.000000
199.999999999999983
200
200.0000000000000000
```

结果初步验证：第9-10行的结果为200，证明预期结果应该没问题。第9行保留默认6位，会在第7位上进行四舍五入。第10行因为采用cout输出，会进行趋零舍入。

在计算中可能会大概率遇到的问题：

```
1  #include <iostream>
2  #include<cmath> //round函数的头文件
3  using namespace std;
4  int main(){
5      double x=99;
6      x=x/100;
7      int a=2/(1-x);
8      cout<<a<<endl;
9      cout<<2/(1-x)<<endl;
10     printf("%lf\n",2/(1-x));
11     printf("%.14lf\n",2/(1-x));
12     a = round(2/(1-x));
13     cout<<a<<endl;
14     printf("%
15     return 0;
16 }
```

样例输出

```
199
200
200.000000
199.99999999999983
200
200.0000000000000000
```

//输出为199，是一个错误的结果

//200，结果正确，趋零舍入的原因

//6位精度时结果正确，第7位四舍五入的原因

//14位精度时，整数部分为199

//结果为200

错误原因分析：第11行保留14位精度进行输出，找到了错误的原因。 $2/(1-x)$ 的运算结果是一个非常接近200的数，但是因为浮点数无法精确表示，在计算机中保留为一个非常接近200，但是却小于200的数。在第8行转换为整数时，按照计算机中的取整规则，将小数点后的数据自动截断，就得到了一个错误的结果。

在计算中可能会大概率遇到的问题:

```
1  #include <iostream>
2  #include<cmath> //round函数的头文件
3  using namespace std;
4  int main(){
5      double x=99;
6      x=x/100;
7      int a=2/(1-x);
8      cout<<a<<endl;
9      cout<<2/(1-x)<<endl;
10     printf("%lf\n",2/(1-x));
11     printf("%.14lf\n",2/(1-x));
12     a = round(2/(1-x));
13     cout<<a<<endl;
14     printf("%.14lf\n",round(2/(1-x)));
15     return 0;
16 }
```

样例输出

```
199
200
200.000000
199.999999999999983
200
200.0000000000000000
```

//输出为199, 是一个错误的结果

//200, 结果正确, 趋零舍入的原因

//6位精度时结果正确, 第7位四舍五入的原因

//14位精度时, 整数部分为199

//结果为200

//小数部分四舍五入为整数

错误修正方式: 在这样情况下, 应该采用第12行的形式, 使用round函数, 保证结果的正确性。

本节要点

索引	要点	正链	反链
T254	printf的精度控制会进行四舍五入		
	洛谷: U269724 (LX204)		
T255	cout对浮点数的输出会进行趋零舍入		
T256	round函数的正确使用, 尤其对于浮点运算结果	T252	T321
	洛谷: U269753 (LX216)		

06

操作符

中国石油大学(华东)

qingline.net/cppbook

6.1 运算符

常用运算符：

编程常用 +、-、*、/ 来分别执行加、减、乘、除数学运算，用 = 来执行赋值操作，关系运算符 >、>=、<、<=、==、!= 与小学数学中的用法相同。

特别强调，等于运算符是两个等号 ==，初学者常用赋值号代替等于判断，从而产生了无法预知的结果。初学者在进行等于判断的时候要特别注意 = 的数量。

有时为了书写方便，还会采用复合运算符 +=、-=、*=、/= 表示。例如 $a+=2$ 表示 $a=a+2$ 。注意 $a*=m+1$ 表示 $a=a*(m+1)$ 。

C/C++ 中没有幂次运算符，初学者常把 ^ 当做幂运算符，实际上它是“位或”操作符。可以包含头文件 <cmath>，使用 pow 函数进行代替。但是要特别注意，pow 的返回值为浮点数类型。

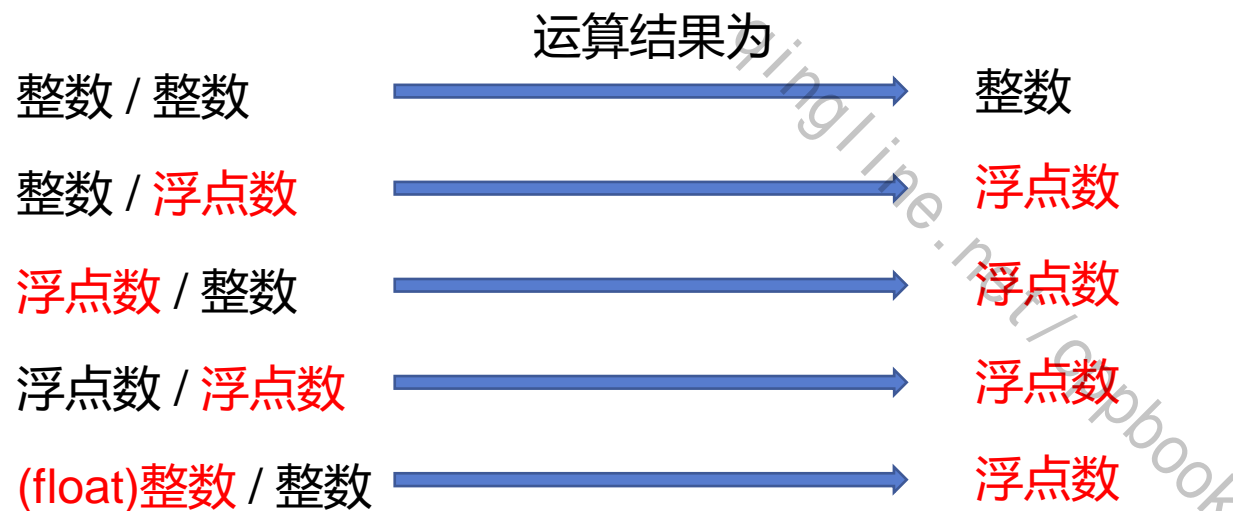
本节要点

索引	要点	正链	反链
T261	等于判断是==，初学者经常容易写成=，这是初学者的经典错误！	T213	T317
T262	掌握复合运算符的语法书写方式		
T263	C/C++中没有幂次运算符；幂运算函数为pow，但是其返回值是浮点型，不建议使用，一定要使用时要注意浮点舍入的潜在问题		T291

6.2 除法和整除

C/C++中除法和整除都用相同的操作符 `/`。

二者的区别是：如果分子和分母都为整数，则商自动取整，即整除运算。如果分子和分母中有一个为浮点数，则商为浮点数。当分子和分母都为整数时，为了取得浮点数的结果，需要将其中之一强制转换为浮点数。



6.2 除法和整除

从理论上讲，整数除整数等于整数，浮点数除浮点数等于浮点数，因为隐式类型转换的存在，混合运算时整型会转换为浮点型，因此只要分子和分母其中之一为浮点数即可。

除法和整除的区分是初学者的常犯错误之一，常常忽略了整除而导致结果不正确

除法和整除运算：

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main()
5  {
6      cout<<(1/3)<<endl;
7      cout<<(1.0/3)<<endl;
8      cout<<(1./3)<<endl;
9      int a=1,b=3;
10     cout<<(a/b)<<endl;
11     cout<<((float)a/b)<<endl;
12     cout<<(float(a)/b)<<endl;
13     return 0;
14 }
```

样例输出

```
0
0.333333
0.333333
0
0.333333
0.333333
```

第8行将1写为1.后，也将1作为浮点数。

第11行是类型强制转换，形成了一个新的浮点型临时变量，第12行是用a构造一个float型对象，二者的含义不同，但达到的效果相同。

除法和整除运算：

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main()
5  {
6      cout<<(1/3)<<endl;
7      cout<<(1.0/3)<<endl;
8      cout<<(1./3)<<endl;
9      int a=1,b=3;
10     cout<<(a/b)<<endl;
11     cout<<((float)a/b)<<endl;
12     cout<<(float(a)/b)<<endl;
13     return 0;
14 }
```

样例输出

```
0
0.333333
0.333333
0
0.333333
0.333333
```

计算顺序：因为整除的原因，一定要特别注意计算顺序。例如三个整数 a , b , c ，如果数学上保证 $a*b$ 一定能被 c 整除，在书写时，一定要写成 $a*b/c$ ，而不能写成 $a/c*b$ 或 $b/c*a$ ，因为整除会导致小数精度的丢失，从而导致结果错误。如果能数学上能保证 a 或 b 都能被 c 整除，那么建议使用 $a/c*b$ 或 $b/c*a$ ，这样会避免因为 $a*b$ 数值过大而导致的内存溢出。

本节要点

索引	要点	正链	反链
T264	/ 既可以进行浮点除法（被除数和除数其中之一为浮点数），也可以进行整除（被除数和除数都为整数），一定要能区分二者的不同，混合运算时要特别注意计算顺序	T251	
	洛谷： U269751 (LX215)， U269701 (LX224)		

6.3 求模运算

$a\%b$ 称为a对b求模(modulus)，简言之就是求a除以b的余数，要求a和b必须都为整数。求模运算在程序设计中比较常见，主要有以下功能：

- 倍数判断。如果 $a\%b==0$ ，则a是b的倍数。
- 奇偶判断。如果 $a\%2==0$ ，则a是偶数，否则a为奇数。
- 取n进制的个位数。如果 $a\%n$ 的结果为m，则m是a作为n进制的各位数。例如 $153\%10=3$ 表示153在十进制下的个位数为3， $153\%2=1$ 表示153在二进制下的个位数为1。

例题

输入一个两位数，输出每个数位上的数值，以空格分隔。

样例输入	样例输出
78	7 8

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int v;
7      cin>>v;
8      cout<<v/10<<' '<<v%10<<endl;
9      return 0;
10 }
```

取余可以得到个位数值，整除操作可以让其他数位移动到个位上。

例题

Tom 和 Mary 玩取石子的游戏： n 颗石子码成一堆，从 Tom 开始，两人轮流取石子，最少取 1 颗、最多取 2 颗，谁取到最后一颗石子，谁就失败。两个人都极聪明，不会放过任何取胜的机会。请同样聪明的你编写程序，输入石子的数量，输出 Mary 是否获胜。

题目解析：当只有1颗时，只能Tom取走，因此Mary获胜。当有2-3颗时，Tom可以取走1-2颗，把最后一颗留个Mary，因此Tom获胜。而当有4颗时，无论Tom怎么取，都会把主动权交给Mary，因此Mary获胜。当有5-6颗时，Tom重新掌握了主动权。由此可知，当两个人都非常聪明时，结果出现了循环，石子数为3的倍数加1时，Mary获胜，石子数为3的倍数加2或3的倍数时，Tom获胜。

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n;
7      cin>>n;
8      cout<<boolalpha<<(n%3==1)<<endl;
9      return 0;
10 }
```

样例输入1	样例输出1
1	true
样例输入2	样例输出2
2	false

本节要点

索引	要点	正链	反链
T265	对于周期性运算，首先要考虑取余运算%，经典使用场景为奇偶判断、倍数判断和整数的数位分解		T26A
	洛谷： U269727 (LX206)， U269732 (LX208)， U269746 (LX212)		

6.4 逻辑运算符

运算符	说明	范例	
&&	逻辑与	A && B	如果 A 和 B 的值都为真，那么结果为真，否则结果为假。 如果 A 的值为假，那么不会计算 B 的值，这叫做短路。
	逻辑或	A B	只要 A 和 B 的值一个为真，那么结果为真，否则结果为假。 如果 A 的值为真，那么不会计算 B 的值，这叫做短路。
!	逻辑非	!A	如果原来 A 为真，那么结果为假。如果原来 A 为假，那么结果为真。

因为逻辑与和逻辑或存在短路运算，所以要将重要的条件放在前面，如果前面的条件不成立，将不会考虑后面的条件。短路运算在很多场合都可以大幅提升计算的效率。

特别注意，C/C++中不支持连续比较：

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int m=4;
6      cout<<(3<m<5)<<endl;
7      m=2;
8      cout<<(3<m<5)<<endl;
9      m=6;
10     cout<<(3<m<5)<<endl;
11 }
```

样例输出

1
1
1

从输出结果可以看到，无论m为2或4或6，结果都为1，即判断结果都为true。这是因为首先判断3<m，无论结果是0或1，都是小于5。因此最终结果为1。如果需要连续关系判断，必须使用逻辑运算，即改为3<m && m<5，才能得到预期结果。这也是初学者的常犯错误之一。

可以将4.3 布尔类型中的代码用!改写:

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

样例输入

样例输出

3 0 5

2

第7行可以用逻辑非改写为
`cout<<(!a+!b+!c);`

以a=3为例, 3为非0值, 表示true, 则!3表示false, 则!!3表示true, 参与算术运算时被计算为1。同理!!0表示0, !!5表示1, 因此计算结果和代码2.11相同。

随堂练习

能被400整除的为闰年，或能被4整除并且不能被100整除的为闰年。完成以下程序的标记为TODO的缺失部分，闰年输出为1，否则输出为0。

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int year;
6      cin>>year;
7      cout<<( /*TODO*/ )<<endl;
8      return 0;
9  }
```

本节要点

索引	要点	正链	反链
T266	C/C++中没有连续关系判断，多条件时必须采用 && 和 运算符		
	洛谷： U270346 (LX311)		
T267	注意取反运算！在数值类型和逻辑类型中的映射方式变化	T244	

6.5 自增和自减运算

前自增（自减）、后自增（自减）：

在 C++ 中， $i++$ （后自增）或 $++i$ （前自增）都表示 $i=i+1$ 。 $i++$ 具有赋值运算，改变 i 的值，而 $i+1$ 没有赋值运算，不改变 i 的值，因此 $i++$ 和 $i+1$ 是不同的。

前自增与后自增的区别是前自增是先自增后赋值，后自增是先赋值后自增。与之类似，前自减与后自减的区别是前自减是先自减后赋值，后自减是先赋值后自减。

```
1  #include <iostream>
2  using namespace std;
3  int main(int argc, char **argv)
4  {
5      int a = 100;
6      int b = a++;
7      int c = 100;
8      int d = ++c;
9      cout << "a = " << a << " b = " << b << endl;
10     cout << "c = " << c << " d = " << d << endl;
11 }
```

样例输出

```
a = 101 b = 100
c = 101 d = 101
```

本节要点

索引	要点	正链	反链
T268	在复合语句中，要注意自增和自减运算的发生时刻。建议初学者把自增和自减运算形成独立语句，仅利用其书写上的便捷性，不要因为运算顺序的错误而产生潜在问题		

6.6 sizeof运算符

在 C/C++ 中，sizeof 运算符用于获取一个变量或者数据类型所占的内存的字节大小。注意sizeof是一个运算符，而不是一个函数。函数求值是在运行的时候，而关键字 sizeof 求值是在编译的时候。

```
1  #include <iostream>
2  using namespace std;
3  int main(int argc, char **argv)
4  {
5      int sizeofInt = sizeof(int);
6      int sizeofLL = sizeof(long long);
7      int sizeofChar = sizeof(char);
8      int sizeofFloat = sizeof(float);
9      int sizeofDouble = sizeof(double);
10     cout <<"sizeofInt = "<<sizeofInt<<" sizeofLL = "<<sizeofLL<<endl;
11     cout <<"sizeofChar = "<<sizeofChar<<endl;
12     cout <<"sizeofFloat = "<<sizeofFloat<<" sizeofDouble = "<<sizeofDouble<<endl;
13 }
```

样例输出

```
sizeofInt = 4
sizeofLL = 8
sizeofChar = 1
sizeofFloat = 4
sizeofDouble = 8
```

本节要点

索引	要点	正链	反链
T269	通过sizeof运算符，了解变量和常量的空间占用状况		

qingline.net/cppbook

6.7 位运算*

位运算的示例：

计算机中的一切最终都是以二进制形式表示的，一个1/0称为一位（bit）。C++中提供了针对二进制的运算，直接操作一个对象的某个比特位。以 $a=22=0b10110$, $b=28=0b11100$ 为例，下表（见下页）展示了位运算的示例。请注意，运算时不需要特意将数值转换为二进制。

运算	符号	说明	样例	结果	二进制
按位取反	~	将操作数按位取反，即对于每个二进制位，1变0，0变1。	~a	4294967273	11101001
按位与	&	将a与b对应二进制位逐一进行按位与运算。当且仅当a与b中对应二进制位均为1时，结果位为1，否则为0。	a&b	20	10100
按位或		将a与b对应二进制位逐一进行按位或运算。当且仅当a与b中对应二进制位中至少有一个1时，结果位为1，否则为0。	a b	30	11110
按位异或	^	将a与b对应二进制位逐一进行按位异或运算。当且仅当a与b中对应二进制位不同时，结果位为1，否则为0。	a^b	10	1010
左移位	<<	a<<n表示将对象a的二进制位逐次左移n位，超出左端的二进制位丢弃，并用0填充右端空出的位置，相当于乘 2^n 。	a<<2	88	1011000
右移位	>>	a>>n表示将对象a的二进制位逐次右移n位，超出右端的二进制位丢弃。如果a是无符号整数，用0填充左端空位，如果a为有符号整数，填充值取决于具体的机器，可以是0，也可以是符号位，相当于整除 2^n 。	a>>3	2	10

位运算基本操作:

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4  int main(){
5      unsigned int a = 0b10110, b=0b11100; //0b开始表示一个二进制数
6      cout<<a<<'\t'<<bitset<8>(a)<<endl;
7      cout<<b<<'\t'<<bitset<8>(b)<<endl;
8      cout<<~a<<'\t'<<bitset<8>(~a)<<endl;
9      cout<<(a&b)<<'\t'<<bitset<8>(a&b)<<endl;
10     cout<<(a|b)<<'\t'<<bitset<8>(a|b)<<endl;
11     cout<<(a^b)<<'\t'<<bitset<8>(a^b)<<endl;
12     cout<<(a<<2)<<'\t'<<bitset<8>(a<<2)<<endl;
13     cout<<(a>>3)<<'\t'<<bitset<8>(a>>3)<<endl;
14     return 0;
15 }
```

样例输出

```
22 00010110
28 00011100
233 11101001
20 00010100
30 00011110
10 00001010
88 01011000
2 00000010
```

当需要进行二进制相关的运算时，位运算能发挥极好的作用。例如，计算一个数的二进制中有几个1时，就可以通过位与操作进行逐位判断，详见第四章9.4节。

6.7 位运算*

用位运算代替正常的数学运算：

因为二进制是数据在计算机中的终极表达形式，所以位运算具有较高的性能。在一些特殊需求下，可以用位运算代替正常的数学运算。

- 二进制每右移一位，相当于整除2；每左移一位，相当于放大2倍；当需要进行 2^n 倍放大或缩小时，可以通过移位操作代替数学的乘除法。
- 二进制中，奇数个位为1，偶数个位为0，因此奇偶判断除了前文提到的 $n\%2$ 外，还可以用 $n\&1$ 进行判断，而且显然后者的效率要高很多。

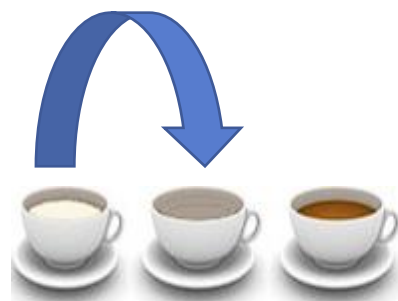
本节要点

索引	要点	正链	反链
T26A	理解位运算的基本含义，增强对计算机底层结构的理解。位运算的效率 high，移位运算可以进行 2^n 的乘法或除法， $n \& 1$ 可以代替奇偶判断，初学者可以忽略位运算	T265	T26C , T2A2

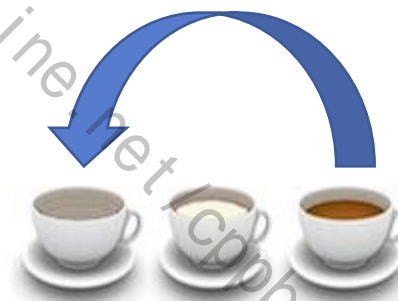
6.8 三个层次的变量交换

两个变量的值交换：

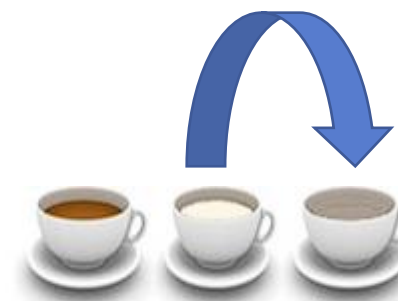
- 基本形式
- 数学方法
- 位运算形式



1) 牛奶倒入空杯



2) 咖啡倒入牛奶杯



3) 牛奶倒入咖啡杯

基本形式

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

数学方法

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

位运算形式

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

为了防止值被覆盖，需要引入一个临时变量，保留被覆盖的值。

基本形式

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

数学方法

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

位运算形式

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

如果要求不使用第三个变量，可以通过数学方法解决。○ 第5行保留了两个变量的和；第6行用和减去原来的b，就得到了原来的a，赋值给b；第7行用和减去原来的a，就得到了原来的b，实现了两个变量的交换。

位运算形式：数学方法还有一点点小的弊端，如果两个变量都非常大，相加后可能超出了存储范围。可以采用异或运算进行代替。异或有个特点：任何数和自身异或都等于0，而任何数和0异或都等于自身。由此推导出一个新的结论，一个值a另一个值b异或2次，又变成了值a。第5行用a保留了 a^b 的结果，因此第6行相当于 $a^b^b = a^{(b^b)} = a^0 = a$ 并赋值给b；第7行相当于 $a^b^a = (a^a)^b = 0^b = b$ ，从而实现了两个变量的交换。位运算不会存在数据溢出的风险。

位运算形式

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

C++中提供了模板函数swap，用来交换两个相同类型变量的值。例如：swap(a,b)。

本节要点

索引	要点	正链	反链
T26B	变量交换的基本形式是必须掌握的内容，在C++中可以简易的使用swap。		T312 , T811
	洛谷： U269757 (LX217)		
T26C	异或运算是计算机中的最基本操作之一，要掌握使用方法。特别注意位运算是在二进制级别进行对位操作，不会产生溢出。	T26A	

07

获取用户输入

中国石油大学(华东)

qingline.net/cppbook

7.1 整型和浮点型的cin输入

cin:

cin 可以理解为 c 和 in，表示C++中的标准输入。cin 根据变量的数据类型，将输入信息进行转换，赋值给相应的变量。当遇到与当前变量类型不匹配的字符时，将会自动停止。

代码分析：

```
1  #include <iostream>
2  using namespace std;
3  int main ()
4  {
5      float a,b;
6      cin>>a>>b;
7      cout<<a+b<<endl;
8      int c,d;
9      cin>>c;
10     cout<<c<<endl;
11     cin>>d;
12     cout<<d<<endl;
13 }
```

样例输入	样例输出
1.2 3.4	4.6
12.34	12
	0

“空格”，“制表符”和“回车”被统称为空白符。

当输入数据并回车后，输入的内容被添加到一个输入缓冲区里，程序从缓冲区内读取数据。当执行到cin时，cin读取缓冲区，如果缓冲区内有内容，直接从缓冲区读取。如果缓冲区为空，光标闪烁，等待用户输入。

cin在对整型和浮点型进行输入的时候，首先忽略空白符，然后读取有效的字符进行解析，到第一个与当前变量类型不匹配的字符时停止。被读取的有效字符从缓冲区内被清除，而从第一个无效字符开始的内容依旧保留在缓冲区里。

代码分析：

```
1  #include <iostream>
2  using namespace std;
3  int main ()
4  {
5      float a,b;
6      cin>>a>>b;
7      cout<<a+b<<endl;
8      int c,d;
9      cin>>c;
10     cout<<c<<endl;
11     cin>>d;
12     cout<<d<<endl;
13 }
```

样例输入	样例输出
1.2 3.4	4.6
12.34	12
	0

第6行对a赋值时，1.2前面的所有空白符被忽略，1.2被输入，遇到1.2后的空格时，输入自动停止。1.2被解析为浮点数并赋值给a。被读取的内容从缓冲区内被清除，而 3.4被继续保留在缓冲区里。

对b进行赋值时，因为缓冲区非空，直接从缓冲区读取，3.4被赋值给b。特别注意，回车符作为第一个无效字符，被保留在缓冲区中。

代码分析：

```
1  #include <iostream>
2  using namespace std;
3  int main ()
4  {
5      float a,b;
6      cin>>a>>b;
7      cout<<a+b<<endl;
8      int c,d;
9      cin>>c;
10     cout<<c<<endl;
11     cin>>d;
12     cout<<d<<endl;
13 }
```

样例输入	样例输出
1.2 3.4	4.6
12.34	12
	0

第9行对c进行输入时，忽略缓冲区中残留的回车，未得到有效信息，因此光标闪烁，等待用户输入。用户输入12.34后，因为c是整型，.被认为是无效字符，输入停止。因此c被赋值为12。.34被保留在缓冲区中。

第9行对d进行输入时，遇到.34，这是一个非法输入，标志位被设置为异常，cin不再接受输入，直接跳过。d被赋值为0。

遇到非法输入时，可以使用cin.clear();重置标志位，然后使用cin.sync();或cin.ignore();清除缓冲区。这种操作比较复杂，初学者可以先行忽略缓冲区非法输入的问题，保证输入的合法性。

在一些特殊的应用场景下，需要输入或输出8进制或16进制数据。在c++中主要体现在oct（八进制）、dec（十进制）和hex（十六进制）三个关键字。但是在cin和cout要分别设置。：

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int v;
7      cin>>oct>>v;
8      cout<<"十进制:"<<v<<"八进制:"<<oct<<v<<endl;
9      cin>>hex>>v;
10     cout<<"八进制:"<<v<<"十进制:"<<dec<<v<<"十六进制:"<<hex<<v<<endl;
11     return 0;
12 }
```

样例输入

11
aa

样例输出

十进制:9;八进制:11
八进制:252;十进制:170;十六进制:aa

默认的通道是十进制，因此第8行直接输出v时，是按照十进制进行输出的。

因为第8行中已经将输出通道改为八进制了，因此在第10行直接进行输出时，按照八进制进行输出，直到改为dec后才改为十进制。

本节要点

索引	要点	正链	反链
T271	必须掌握输入的基本原理，输入和缓冲区的关系	T212	T273 , T275 , T277
T272	掌握8进制和16进制的输入和输出方法	T222	
	洛谷: U269748 (LX213)		

7.2 字符串的输入

cin:

与整型和浮点型类似，字符串也可以采用cin的方式进行读取。但是因为字符串包含的字符范围比较多，只有遇到空白符时输入才结束。空白符之后的内容不会被读取。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main ()
5  {
6      string s;
7      cin>>s;
8      cout<<s<<endl;
9  }
```

样例输入	样例输出
first second	first

7.2 字符串的输入

getline:

可以推断, cin 方式读取的字符串中不包括空白符。如果字符串中需要包含空格, 或者读入空字符串, 需要采用 getline 方式, 该方式需要包含 `<string>` 头文件。该方式读取一行字符串, 并且把末尾的回车符从缓冲区中清除。

代码分析:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main ()
5  {
6      string s1,s2,s3;
7      getline(cin,s1);
8      getline(cin,s2);
9      getline(cin,s3);
10     cout<<"s1:"<<s1<<endl;
11     cout<<"s2:"<<s2<<endl;
12     cout<<"s3:"<<s3<<"\ts3 size: "<<s3.size()<<endl;
13     cout << s3[5] << ' ' << s3[6] << endl;
14 }
```

第一行输入一个空行，第二行两个单词中间有一个空格，但是整行字符串被统一输入给s2。

样例输入

(空行)

first second

first\tsecond

样例输出

s1:

s2:first second

s3:first\tsecond s3

size: 13

\t

当输入中有专业字符时，如s3的输入，用cin读入时会将转义字符识别成两个字符'\'和't'，而不会自己作为转义字符。

本节要点

索引	要点	正链	反链
T273	区分字符串输入时cin（空白符分隔）和getline（回车符分隔）的区别	T271	T276 , T277
	洛谷： U269755 (LX218)		
T274	特别注意输入时的转义字符会被逐个字符读取，不会作为转义字符	T214	

7.3 字符的输入

cin.get():

C++中采用 cin.get() 读取一个任意字符，包括空白符。可以用cout<<或cout.put()输出字符。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main ()
5  {
6      char a;
7      a = cin.get();cout.put(a);
8      a = cin.get();cout<<a;
9      a = cin.get();cout.put(a);
10     a = cin.get();cout<<a;
11 }
```

样例输入

ab
c

样例输出

ab
c

//这里读取的是回车符

注意第9行读取的字符是回车符，第10行读取的字符是字符'c'，因此字符'c'的输入才会在新行上出现。

本节要点

索引	要点	正链	反链
T275	特别注意在输入时空白符也是一个字符	T271	T276

7.4 数字和字符的混合输入

cin.ignore():

如7.1所示，无论是整型或浮点型，在读取正确的输入后，如果后继是一个空白符，空白符会被残留在缓冲区中。如果接下来读取一个字符，或者用getline读取一个字符串，这个残留的空白符也会被作为有效字符进行输入，这与程序的目的可能存在违背。这时需要调用cin.ignore()函数，去除缓冲区中残留的空白符。

代码分析:

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  int main()
6  {
7      int num;
8      string str;
9      cin >> num;
10     getline(cin,str);
11     cout << "Number :" << num << ", String:" << str << "#" << endl;
12 }
```

样例输入

34

样例输出

Number :34, String:#

执行以上程序时会发现，只输入34并回车后，程序就会输出并结束。这是因为34后面的回车残留在缓冲区中，执行getline时，会读取一个空字符串。

代码分析:

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  int main()
6  {
7      int num;
8      string str;
9      cin >> num;
10     cin.ignore();
11     getline(cin,str);
12     cout << "Number :" << num << ", String:" << str << "#" << endl;
13 }
```

样例输入

34
apple

样例输出

Number :34, String:apple#

cin.ignore() 的作用就是从缓冲区中读取一个字符并抛弃，这样残留的回车符就被清除，执行到第11行的时候，因为缓冲区为空，光标闪烁等待用户输入。

7.4 数字和字符的混合输入

cin.ignore():

当需要把无效字符到当前输入行的所有内容都进行清空时，可以调用

cin.ignore(numeric_limits<std::streamsize>::max(), '\n'); 该语句表示一直忽略到回车符。

通常输入的字符串都不会那么长，比如最多只有256个字符，也可以简写为 *cin.ignore(256, '\n');*。

这条语句需要头文件<limits>

ignore函数的第一个参数表示最大抽取的字符数，第二个参数表示结束的字符。无参时表示只抽取并抛弃一个字符。

本节要点

索引	要点	正链	反链
T276	在输入完数值再使用getline时，要特别注意残留回车的影响，一定要使用cin.ignore去除残留回车的影响，否则不能得到正确的输入	T273 , T275	
	洛谷: U269756 (LX219)		

7.5 逗号分隔的数值

注意分隔符：

当输入多个数据时，绝大多数情况都是用空白符分隔的，但是有时也会用其他字符进行分隔，例如逗号分隔。在处理题目时一定要特别注意输入数据的分隔符。



例题

输入三个逗号分隔的整数，依次对其输出，输出时以制表符分隔。

样例输入	样例输出
2, 3, 5	2 3 5

这时一定要对分隔符进行特殊处理。

方法一：用ignore的方式忽略中间的逗号

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a,b,c;
7      cin >> a;
8      cin.ignore();
9      cin >> b;
10     cin.ignore();
11     cin >> c;
12     cin.ignore();
13     cout << a << '\t' << b << '\t' << c << endl;
14 }
```

例题

输入三个逗号分隔的整数，依次对其输出，输出时以制表符分隔。

样例输入	样例输出
2, 3, 5	2 3 5

方法二：用一个字符变量读取中间的逗号

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a,b,c;
7      char ch;
8      cin >> a>>ch>>b>>ch>>c;
9      cout << a << '\t' << b << '\t' << c << endl;
10 }
```

例题

输入三个逗号分隔的整数，依次对其输出，输出时以制表符分隔。

样例输入	样例输出
2, 3, 5	2 3 5

如果输入中存在数值外的常量字符或字符串，scanf是一个比较方便的函数，它与printf相对应，第一个参数也是一个控制字符串。如果输入中有需要忽略的常量字符或字符串，可以直接写到控制字符串中，例如例题2.3中的逗号。

方法三：C语言的scanf方式（推荐方式）

```
1  #include<cstdio>
2  using namespace std;
3
4  int main()
5  {
6      int a,b,c;
7      scanf("%d,%d,%d",&a,&b,&c);
8      printf("%d\t%d\t%d",a,b,c);
9  }
```

特别注意在使用scanf的时候，每个变量前要加&，表示取变量的地址，这是语法要求。

本节要点

索引	要点	正链	反链
T277	当输入数值用非空白符分隔，或有额外字符时，可以采用ignore或字符填充法输入，scanf此时更具有优势，但要特别注意scanf的语法	T271 , T273	T612
	洛谷： U269744 (LX211)		

08

时间处理

中国石油大学(华东)

qingline.net/cppbook

例题

国家安全局获得一份珍贵的材料，上面记载了一个即将进行的恐怖活动的信息。不过，国家安全局没法获知具体的时间，因为材料上的时间使用的是Linux的时间戳，即是从2011年1月1日0时0分0秒开始到该时刻总共过了多少秒。此等重大的责任现在落到了你的肩上，给你该时间戳，请你计算出恐怖活动在某一天实施？（为了简单起见，规定一年12个月，每个月固定都是30天）

【输入】

一个整数n，表示从2011年1月1日0时0分0秒开始到该时刻过了n秒。

【输出】

输出一行，分别是三个整数y，m，d，表示恐怖活动在y年m月d日实施。

起始是一个日期，差值n是一个整数，为了计算最终结果，可以有两种方式：1) 将n转换为日期，与起始日期相加求和；2) 将起始日期转换为整数，求和后转换为日期类型。第一种方式在逻辑上较为顺畅，但是实际运算难度较大，因为日和月的进位都不规整。第二种方式在计算时更为顺畅。下面代码为第二种方式。

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int seconds;
6      cin>>seconds;
7      int days = 60*60*24;
8      int months = 30;
9      int years = months*12;
10     int day = seconds/days;
11     int year = 2011+day/years;
12     day = day-day/years*years;
13     int month = day/ months;
14     day = day-month* months;
15     cout<<year<< ' ' <<month+1<< ' ' <<day+1<<endl;
16     return 0;
17 }
```

样例输入

130432457

样例输出

2015 3 10

//一天包含的秒数

//总共过去了多少天

//整除后再乘，去掉余数部分

//月和日都是从1开始计数

第12行去掉余数的方法在计算中经常被使用。这与数学的概念是不一致的，这里的除法表示的是整除。

7.5 逗号分隔的数值

时间戳:

Unix时间戳(Unix timestamp), 或称Unix时间(Unix time)、POSIX时间(POSIX time), 是一种时间表示方式, 定义为从格林威治时间1970年01月01日00时00分00秒起至现在的总秒数。Unix时间戳不仅被使用在Unix 系统、类Unix系统中, 也在许多其他操作系统中被广泛采用。因为以秒为单位, 数值比较大, 更建议在计算时采用long long类型, 防止数值溢出。

1970年01月01日
00时00分00秒

到当前时间的
总秒数

本节要点

索引	要点	正链	反链
T281	当涉及时间相关运算时，推荐先将时间转换为整数，处理后再转换回来。可以极大程度避免时间单位进位不统一而产生的问题。		
	洛谷： U269758 (LX220)， U270342 (LX309)		

09

常用数学函数

中国石油大学(华东)

qingline.net/cppbook

9 常用数学函数

数学函数在计算中非常常用，C/C++在<cmath>头文件中包括以下常用数学函数：

函数名	功能描述	样例	样例结果
sqrt	平方根函数	sqrt(9)	3
fabs	浮点数绝对值	fabs(-3.5)	3.5
log10, log	以10为底和以e为底的自然对数	log10(1000)	3
pow	幂运算	pow(10,3)	1000
sin, cos	正弦和余弦	sin(M_PI/4)	0.707107
round	四舍五入取整	round(3.7)	4
ceil	向上取整	ceil(5.2)	6
min, max	最小值，最大值	min(5.2, 3.7)	3.7

M_PI在<cmath>头文件中定义，表示 π 。

9 常用数学函数

数学函数在计算中非常常用，C/C++在<cmath>头文件中包括以下常用数学函数：

函数名	功能描述	样例	样例结果
sqrt	平方根函数	sqrt(9)	3
fabs	浮点数绝对值	fabs(-3.5)	3.5
log10, log	以10为底和以e为底的自然对数	log10(1000)	3
pow	幂运算	pow(10,3)	1000
sin, cos	正弦和余弦	sin(M_PI/4)	0.707107
round	四舍五入取整	round(3.7)	4
ceil	向上取整	ceil(5.2)	6
min, max	最小值，最大值	min(5.2, 3.7)	3.7

round和ceil函数的返回值都是double类型，如果需要整型返回值，需要做显示类型转换。

9 常用数学函数

数学函数在计算中非常常用，C/C++在<cmath>头文件中包括以下常用数学函数：

如果对整型调用pow函数，因为返回值为浮点数，因此其结果可能因为浮点数不能精确表示而产生错误。此时建议使用int(round(pow(x,n))), 确保得到正确的整型值。

函数名	功能描述	样例	样例结果
sqrt	平方根函数	sqrt(9)	3
fabs	浮点数绝对值	fabs(-3.5)	3.5
log10, log	以10为底和以e为底的自然对数	log10(1000)	3
pow	幂运算	pow(10,3)	1000
sin, cos	正弦和余弦	sin(M_PI/4)	0.707107
round	四舍五入取整	round(3.7)	4
ceil	向上取整	ceil(5.2)	6
min, max	最小值，最大值	min(5.2, 3.7)	3.7

9 常用数学函数

数学函数在计算中非常常用，C/C++在<cmath>头文件中包括以下常用数学函数：

对于整数a的平方和立方，建议直接写为a*a或a*a*a，对于高次方，建议使用循环。不建议使用函数pow，这样不会转换为浮点数，不会产生精度问题。

函数名	功能描述	样例	样例结果
sqrt	平方根函数	sqrt(9)	3
fabs	浮点数绝对值	fabs(-3.5)	3.5
log10, log	以10为底和以e为底的自然对数	log10(1000)	3
pow	幂运算	pow(10,3)	1000
sin, cos	正弦和余弦	sin(M_PI/4)	0.707107
round	四舍五入取整	round(3.7)	4
ceil	向上取整	ceil(5.2)	6
min, max	最小值，最大值	min(5.2, 3.7)	3.7

例题

求一个整数的位数。

样例输入	样例输出
123	3

答案: $(\text{int})\log_{10}(x)+1$

解析: 任何一个整数都可以表达为 10^y 形式, 其中 y 是一个浮点数, 而且 y 的整数部分为 x 的位数减1。因此答案中先调用函数 \log_{10} , 强制转换为整型相当于舍弃小数部分, 最后加1得到 x 的位数。

qingline.net/cppbook

随堂练习

已知三角形的三个边长 a, b, c , 根据海伦公式编程计算三角形的面积, 保留四位小数精度。海伦公式为 $S = \sqrt{p(p-a)(p-b)(p-c)}$, 其中 $p = (a+b+c)/2$ 。

样例输入	样例输出
1 2 2	0.9682

随堂练习

鸡兔同笼是中国古代的数学名题之一。大约在1500年前，《孙子算经》中就记载了这个有趣的问题。书中叙述为：今有雉兔同笼，上有三十五头，下有九十四足，问雉兔各几何？这四句话的意思是：有若干只鸡兔同在一个笼子里，从上面数，有35个头，从下面数，有94只脚。问笼中各有多少只鸡和兔？编程实现鸡兔同笼问题，输入头和脚的数量，输出鸡和兔的数量。

C/C++不具备解方程的能力，当需要处理方程式，需要人工推导出计算公式，然后交给计算机进行计算。

样例输入	样例输出
35 94	23 12

本节要点

索引	要点	正链	反链
T291	将数学表达式改写成程序代码，是程序设计的基本要求，注意其中发生的数据类型隐式转换	T234 , T251 , T263	T321
	洛谷： U269742 (LX210), U269759 (LX221), U269760 (LX222)		

10 运算效率的底层分析*

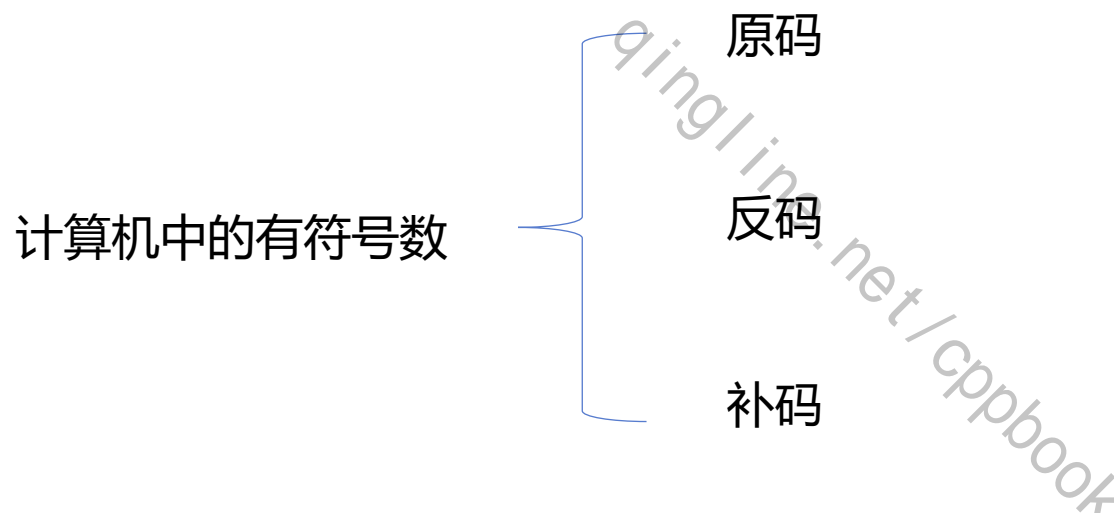
中国石油大学(华东)

qingline.net/cppbook

10.1 负数和补码

计算机中的有符号数：

计算机中的有符号数有三种表示方法，即原码、反码和补码。



10.1 负数和补码

原码:

除了最高位作为符号位之外（0代表正，1代表负），原码可以理解为将数值直接转换为二进制。

例如，用8位二进制表示一个数，
+11的原码为00001011，
-11的原码就是10001011

原码不能直接参加运算，可能会出错。

例如，数学上， $1 + (-1) = 0$ ，而在二进制中
 $00000001 + 10000001 = 10000010$ ，换算
成十进制为-2，显然错误。

10.1 负数和补码

反码:

反码可以理解为正数的反码与原码相同，除符号位之外，各位取反。

例如，+11的反码为00001011，
-11的反码就是11110100。

补码:

补码可以理解为正数的补码与原码相同，负数的补码等于反码+1，

例如，+11的补码为00001011，
-11的补码就是11110101。

10.1 负数和补码

原码和反码的问题：

1. 不能直接参与运算外
2. 0被表示为+0和-0两种表示方法。-0在实际运算中是没有任何意义的，因此原码和反码在表示时都存在问题。

例如：

(+0)原码=00000000,

(-0)原码=10000000,

(+0)反码=00000000,

(-0)反码=11111111。

10.1 负数和补码

补码:

为了解决以上两个问题，提出了补码的概念。在进行实际分析之前，先看一个日常生活中的例子，当前时间为3点，想将其调整为8点，可以按顺时针调整5个小时，也可以逆时针回拨7个小时，即 $3-7=3+12-7=3+5$ ，对于模12而言，5和7互为补数。当需要减数a时，其效果与加补数是相同的。



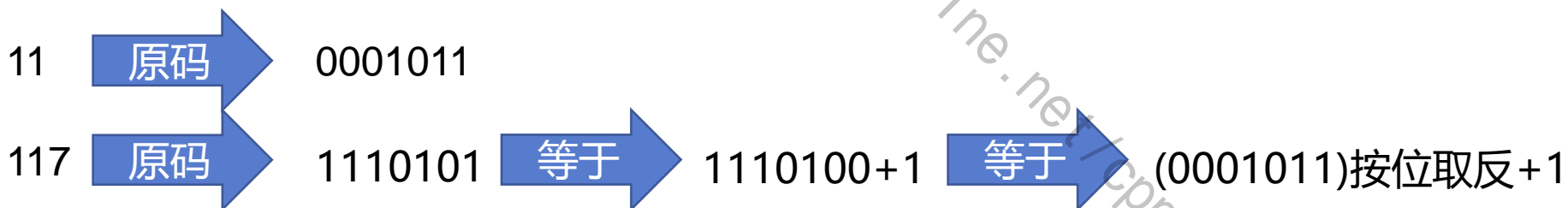
10.1 负数和补码

补码:

给定位数的存储空间，例如7位。当结果超过位数限制时，高位自动被舍弃。例如

$$11 + 117 = 128 = 27 = (10000000)_2。$$

因为存储空间位7位的限定，最高位的1被舍弃，所以 $11 + 117 = 0$ 。也就是说11和117在模27下互为补数。



10.1 负数和补码

补码：

由此得到一个结论，对于 n 位存储空间的限定，一个数的补码等于其相对与模 2^n 的补数。如前所述，当需要减一个数时，等价于加这个数对模 2^n 的补数。因此需要减去一个数时，可以将这个数用补码表示，然后加上这个补码，就完成了减法操作。

实际上，计算机中只有加法器，而没有减法器，所有的减法都是由加补码完成的。

10.1 负数和补码

补码解决了原码和反码出现的两个问题：

1) 补码可以直接参与运算。 $25 + (-11) = (00011001)\text{补码} + (11110101)\text{补码} = (00001110)\text{补码} = 14$ ，结果正确。

2) $+0 = (00000000)\text{原码} = (00000000)\text{补码}$ ， $-0 = (10000000)\text{原码} = (11111111)\text{反码} = (00000000)\text{补码}$ ，在补码下， $+0$ 和 -0 的表示方式完全相同。

0的表示唯一了，那么原来 -0 的原码 10000000 表示什么？

$(00000001)\text{补码} + (10000000)\text{补码} = (10000001)\text{补码} = -127$ ，也就是说 $1 + (10000000)\text{补码} = -127$ ，因此 $(10000000)\text{补码} = -128 = 27$ 。这就是在2.1节中，`<climits>`头文件里有符号类型最小值的绝对值比最大值的绝对值大1的原因。

类型溢出:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      cout << (1<<7) << endl;
5      cout << (1<<31) << endl;
6      cout << unsigned(1<<31) << endl;
7      cout << (1U<<31) << endl;
8      cout << (1LL<<31) << endl;
9      return 0;
10 }
```

//U表示unsigned int类型
//LL表示long long类型

样例输出

```
128
-2147483648
2147483648
2147483648
2147483648
-1360758559
428135971041
```

这段代码中使用了移位运算符<<, 详见本章6.7节。它表示对一个数值的二进制表达形式下向左平移, 每平移一位相当于对原有数值扩大两倍, 因此 $1<<n$ 等价于 2^n 。因此第4行输出结果为 $128=2^7$ 。

类型溢出:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      cout << (1<<7) << endl;
5      cout << (1<<31) << endl;
6      cout << unsigned(1<<31) << endl;
7      cout << (1U<<31) << endl;
8      cout << (1LL<<31) << endl;
9      return 0;
10 }
```

样例输出

```
128
-2147483648
2147483648
2147483648
2147483648
-1360758559
428135971041
```

对于初学者最疑惑的部分在于，通过左移数值不断扩大，可能出现输出结果为负数。这是因为在有符号数据类型中，最高位的1是符号位，表示负数。默认情况下，字面量1被按照int类型处理，231的二进制为1后面跟随31个0，表示负数，而负数的二进制是用补码表示，因此第5行输出结果为-231=-2147483648。同样道理，对于一个任意的正数，如果某位上的1被左移到最高位，整个数值就会被解读为负数。

类型溢出:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      cout << (1<<7) << endl;
5      cout << (1<<31) << endl;
6      cout << unsigned(1<<31) << endl;
7      cout << (1U<<31) << endl;
8      cout << (1LL<<31) << endl;
9      return 0;
10 }
```

//U表示unsigned int类型
//LL表示long long类型

样例输出

```
128
-2147483648
21474836488
2147483648
2147483648
-1360758559
428135971041
```

对于无符号类型unsigned, 最高位不作为符号位, 因此不会出现左移后变为负数的情况。第6行显示正确的结果。

类型溢出:

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      cout << (1<<7) << endl;
5      cout << (1<<31) << endl;
6      cout << unsigned(1<<31) << endl;
7      cout << (1U<<31) << endl; //U表示unsigned int类型
8      cout << (1LL<<31) << endl; //LL表示long long类型
9      return 0;
10 }
```

样例输出

```
128
-2147483648
2147483648
2147483648
2147483648
-1360758559
428135971041
```

在数值后面添加字面量，可以修改数值的默认类型，其中1U表示unsigned int类型的1，1LL表示long long类型的1。这两种类型的数值有效范围都大于31位，因此都能显示正确结果。

本节要点

索引	要点	正链	反链
T2A1	了解负整数在计算机中的表示方式与正整数不同即可，这种方式叫补码		

qingline.net/cppbook

10.2 整型的极限值


数据类型的极值：

在进行数值运算时，会用到一个数据类型的极值。

有时为了防止数据溢出，也会对极限值进行判断。

如果记不住这些关键字，也可以从二进制的角度进行记忆。有符号整型的最小值可以理解为负0，即符号位为1，其余位为0，最大值为符号位为0，其余位为1。

符号整型的最小值  10...0

符号整型的最大值  01...1

例如用一个变量m记录多个数的最大值时，需要先将m的默认值设置为对应数据类型中的最小值；同样在求最小值时，需要先将m的默认值设置为它能表示的最大值。

如第2.1小节所示，其实这些极限值都在头文件<climits>中有定义，例如int类型的最大值为INT_MAX，最小值为INT_MIN，unsigned int的最大值为UINT_MAX等。

10.2 整型的极限值

以整型为例：

以int为例，最小值为 $1 \ll 31$ ，最大值为 $\sim(1 \ll 31)$ 。这是因为字面量1的默认数据类型为int。

对于long long类型，对应的最小最大值分别为 $1LL \ll 63$ 和 $\sim(1LL \ll 63)$ 。因为溢出位被自动截断的原因，数值表示构成循环，最小值减1对应的也是最大值，即int和long long的最大值也可以表示为 $(1 \ll 31) - 1$ 和 $(1LL \ll 63) - 1$ 。

而无符号整型的最大值为全1，unsigned int和unsigned long long的最大值分别为 $\sim 0U$ 和 $\sim 0LLU$ 。因为溢出位被自动截断的原因，数值表示构成循环，-1的内存表示与对应最大值的内存表示相同，所以也可以表示为 $-1U$ 和 $-1LLU$ 。

10.2 整型的极限值

int和long long极值的位运算表示法:

类型	字节	最小值	值	最大值	值	unsigned最大值	值
int	4	$1 \ll 31$	-2^{31}	$\sim(1 \ll 31)$	$+2^{31} - 1$	-1U或~0U	$+2^{32} - 1$
long long	8	$1LL \ll 63$	-2^{63}	$\sim(1LL \ll 63)$	$+2^{63} - 1$	-1LLU或~0LLU	$+2^{64} - 1$

本节要点

索引	要点	正链	反链
T2A2	int和long long以及对应的无符号整数的极值表示	T221 , T26A	

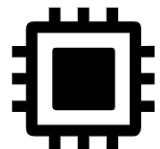
10.3 常见运算的效率分析

基础运算之间有效率差距，当数据量比较大时，考虑用效率高的运算代替效率低的运算。以下是几种基础运算效率高低的比较：

移位 > 赋值 > 大小比较 > 加法 > 减法 > 乘法 > 取模 > 除法



10.3 常见运算的效率分析



CPU中的算术逻辑单元ALU在算术上只干了两件事，**加法**，**移位**，顶多加上**取反**，在逻辑上，只有**与**、**或**、**非**、**异或**4种操作。**移位**功能部件也是最基础的部件。



减法是取补码相加（补码即对一个数的二进制表示进行取反加1）实现的。



乘法和除法都是靠移位实现的。左移 n 位执行乘 2^n ，右移 n 位执行除 2^n 。以此为基础，乘法器是一步一步乘(移位)出来的，每次取乘数的一位与被乘数操作，1则把被乘数照写，0则为0，然后乘数右移。这样循环，最后把每一步结果加起来。最后将上面每一步的结果，然后直接相加。



而除法是每步的结果作加法或减法(加减交替法)，有的算法还需要恢复上一次的结果(余数恢复法)，而且每一步加减后还要进行移位，此外，每次移位后比乘法还多出一试错操作，所以除法效率最低。

综上所述，计算机用加法器、移位器和基本逻辑门电路就构成一个简单的算术逻辑单元ALU。

THANKS

中国石油大学(华东)

李昕

qingline.net/cppbook