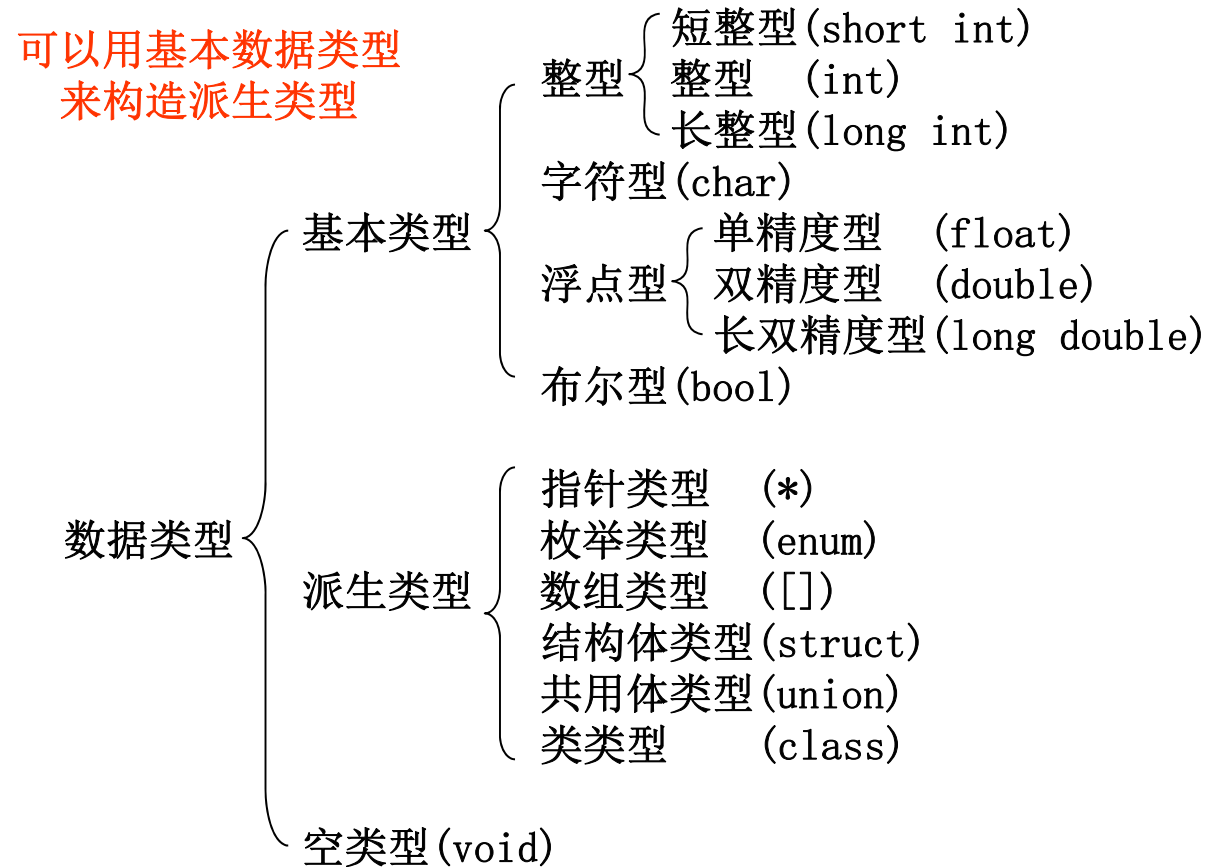


§ 2. 数据类型和表达式

2. 1. C++的数据类型

2. 1. 1. C++的数据类型

P. 18



§ 2. 数据类型和表达式

2. 1. C++的数据类型

2. 1. 2. 各种数据类型所占字节及表示范围

P. 19-20 表2. 1及说明

(VS2017 32bit编译器为基准，浮点数范围有错)

类型	类型标识符	字节	数值范围	数值范围
整型	[signed] int	4	-2147483648 ~ +2147483647	$-2^{31} \sim +2^{31}-1$
无符号整型	unsigned [int]	4	0 ~ 4294967295	$0 \sim +2^{32}-1$
短整型	short [int]	2	-32768 ~ +32767	$-2^{15} \sim +2^{15}-1$
无符号短整型	unsigned short [int]	2	0 ~ 65535	$0 \sim +2^{16}-1$
长整型	long [int]	4	-2147483648 ~ +2147483647	$-2^{31} \sim +2^{31}-1$
无符号长整型	unsigned long [int]	4	0 ~ 4294967295	$0 \sim +2^{32}-1$
字符型	[signed] char	1	-128 ~ +127	$-2^7 \sim +2^7-1$
无符号字符型	unsigned char	1	0 ~ +255	$0 \sim +2^8-1$
单精度型	float	4	$-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$	$-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$
双精度型	double	8	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$
长双精度型	long double	8	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$

§ 2. 数据类型和表达式

2. 1. C++的数据类型

2. 1. 2. 各种数据类型所占字节及表示范围

P. 19-20 表2. 1及说明

(VS2017 32bit编译器为基准，浮点数范围有错)

★ 用sizeof(数据类型)来确定某个具体编译器中每种数据类型所占的字节

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

int main()
{
    cout << sizeof(int) << endl;
    cout << sizeof(unsigned int) << endl;
    cout << sizeof(long) << endl;
    cout << sizeof(unsigned short) << endl;
    cout << sizeof(float) << endl;
    cout << sizeof(double) << endl;

    return 0;
}
```

用VS2017的32位编译器
64位编译器
DevC++的32位编译器
64位编译器
CodeBlocks的32编译器
Linux C++的64位编译器
分别编译运行并观察结果

注:

- 1、给的CodeBlocks软件只包含32位编译器
- 2、给的Linux虚拟机中只包含64位编译器
- 3、VS2017和DevC++如何切换32位和64位编译器?

§ 2. 数据类型和表达式

2.1. C++的数据类型

2.1.2. 各种数据类型所占字节及表示范围

- ★ 对于整型数(3种), 均有signed及unsigned的区别, 缺省为signed
- ★ 对于16/32位编译系统, short型占2字节, long占4字节, int的大小与编译系统有关(2/4字节)
(本课程中若不加以特别说明, 均认为是32位编译系统)
- ★ 对某些32位编译系统, 有8字节的整型数, 表示形式为(__int64, long long int等), 本课程不讨论
- ★ 对某些64位编译系统, int和long所占字节可能存在差异, 了解即可
- ★ 对于整型数, 存储为二进制数形式

P.19 图 2.1 (十进制)85 = 1010101(二进制)

则: int型 : 00000000 00000000 00000000 01010101

long型 : 00000000 00000000 00000000 01010101

short型: 00000000 01010101

char型 : 01010101

- ★ 浮点型数有有效位数的限定, 可能存在一定的误差

§ 2. 数据类型和表达式

2.1. C++的数据类型

2.1.3. 整型数的符号位

含义：表示一个整型数的若干字节的最高bit位 (最左)

0/1表示正负：signed

0/1表示数值：unsigned

P. 20 图2.2:

01111111 11111111 = 32767 (有符号短整型)

32767 (无符号短整型)

11111111 11111111 = -1 (有符号短整型) ?? (用第1章补充知识理解, 应为-32767)

65535 (无符号短整型)

2.1.4. 补码的基本概念

原码：整型数转换为二进制后的形式，最高位可作为符号位

10: 0000000000001010

-10: 1000000000001010

★ 原码的缺陷：+0与-0的二义性问题

1: 1000000000000000

0: 0000000000000000

补码：计算机内整型数值的表示方法

正数与原码相同

负数：绝对值的原码取反+1

问：现在看到一个整数(1**), 到底是unsigned正数还是signed的负数?

答：错误的问题!!!

不是你看到整数后, 再去判断该数是什么类型, 而是要先确定以什么类型去看待这个数, 再去确定该数的值

例：short型整数

数值	二进制表示	原码	补码
100	1100100	0000000001100100	0000000001100100
-10	1010（绝对值）	0000000000001010	111111111110101 +) 1 ----- 111111111110110
0	0(正) 0(负)	0000000000000000 0000000000000000	0000000000000000 111111111111111 +) 1 ----- 1 0000000000000000 (高位溢出，舍去)
-1	1（绝对值）	0000000000000001	111111111111110 +) 1 ----- 111111111111111
-32768	1000000000000000	1000000000000000	011111111111111 +) 1 ----- 1000000000000000
-32767	0111111111111111	0111111111111111	1000000000000000 +) 1 ----- 1000000000000001

§ 2. 数据类型和表达式

2.2. 常量

2.2.1. 基本概念

常量：在程序运行过程中值不能改变的量称为常量

{ 字面常量(直接常量)：直接字面形式表示
符号常量：通过标识符表示

2.2.2. 数值常量

2.2.2.1. 整型常量

三种表示方法：

十进制： 正常方式

二进制： 0b+0~1

八进制： 0+0~7

十六进制： 0x/0X+0~9, A-F, a-f

123 10进制表示

0b1111011 2进制表示

0173 8进制表示

0x7B 16进制表示

四个值相等，都是10进制的123

请把下列三个数从大到小排列：

(1) 123

(2) 0123

(3) 0x123

?

★ 老版本编译器可能无二进制表示方法(但内部存储仍然为二进制补码)

★ 二进制方式表示不方便，今后不再讨论

★ 整型常量缺省是int型，long型通常加l(L)表示，unsigned通常加u(U)，short型无特殊后缀

下列整型常量分别是什么类型？

123 :

123L :

123U :

123UL:

123LU:

?

§ 2. 数据类型和表达式

2.2. 常量

2.2.2. 数值常量

2.2.2.2. 浮点型常量

两种表示方式:

十进制数 (带小数点的数字)

0.123 123.456 123.0

指数形式 (科学记数法)

- e前面为尾数部分, 必须有数, e后为指数部分, 必须为整数

1.23e4	✓
1.23e-4	✓
-1.23e-4	✓
e4	✗
1.23e4.5	✗

- 尾数的整数部分为0, 第1位小数非零的表示形式称为**规范化的指数形式**, 无论源程序中如何表示, 机内存储都是规范化指数形式

123e4	(123 x 10 ⁴)	机内存储形式
12.3e5	(12.3 x 10 ⁵)	
1.23e6	(1.23 x 10 ⁶)	
0.123e7	(0.123 x 10 ⁷)	
0.0123e8	(0.0123 x 10 ⁸)	

§ 2. 数据类型和表达式

2.2. 常量

2.2.2. 数值常量

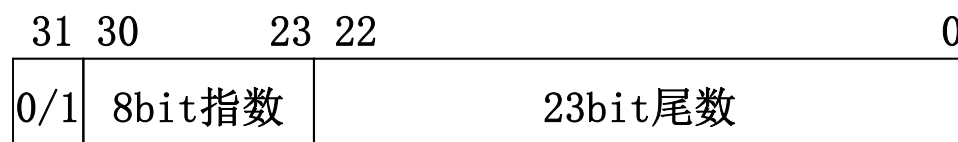
2.2.2.2. 浮点型常量

两种表示方式:

十进制数 (带小数点的数字)

指数形式 (科学记数法)

★ 浮点数在内存中的存储分为三部分, 分别是符号位、指数部分和尾数部分



浮点数的存储遵从 IEEE 754 规范
具体暂时不做要求, 第2学期再说
P. 22 图2.3仅是一个分段示范, 不准确

★ 浮点数有指定有效位数(float:6位/double:15位), 超出有效位数则舍去(四舍五入), 因此会产生误差

例1: 常量1: 123456.7890123456e5

常量2: 123456.7890123457e5

内部存储都是 0.123456789012346e11

例2: 1.0/3*3

不同编译系统, 可能是0.999999或1或1.000001

★ 浮点常量缺省为double型, 如需表示为float型, 可以在后面加f(F)

1.23 : double型, 占8个字节

1.23F : float型, 占4个字节

可自行写测试程序来证明
`cout << sizeof(1.23) << endl;`
`cout << sizeof(1.23F) << endl;`

§ 2. 数据类型和表达式

2.2. 常量

2.2.3. 字符常量与字符串常量

2.2.3.1. ASCII码

★ P. 464 附录A ASCII码表

★ ASCII码占用一个字节，共可表示256个字符

0XXXXXXX : 基本ASCII码 128个(0-127)

1xxxxxxx : 扩展ASCII码 128个(128-255)

★ 基本ASCII码分图形字符和控制字符

0-32, 127: 控制字符, 34个

33-126 : 图形字符, 94个(键盘上都能找到)

★ 几个基本的ASCII码值

0 - 48/0x30 空格 - 32

A - 65/0x41 a - 97/0x61

★ 汉字的表示:

GB2312-80 : 用两个字节表示一个汉字, 共6733个, 两字节的高位为1(与扩展ASCII码冲突)

GBK : 1995年公布, 2字节表示, 收录汉字20000+

GB18030-2005: 2-4个字节表示, 收录汉字70000+

§ 2. 数据类型和表达式

2.2. 常量

2.2.3. 字符常量与字符串常量

2.2.3.2. 普通字符常量与转义字符常量

★ 直接表示 ' 空格或大部分可见的图形字符'

★ 转义符表示 '\字符、八、十六进制数'

- '\字符' : P. 22-23 表2.2 (\v的ASCII码为11)

- '\ddd' : 000-377 (0-255), 超出error

- '\xhh' : 00-ff/FF (0-255)

注意: \x的x必须小写, 否则warning('\X41' 错), 后面字符大小写不限 ('\x1A' / 'x1a' 均可)

相似概念: 整型常量的16进制, 大小写均可0x41 / 0X41

```
#include <iostream>
using namespace std;
int main()
{
    cout << '\377' << endl;
    cout << '\477' << endl;
    cout << '\x41' << endl;
    cout << '\X41' << endl;

    return 0;
}
```

VS2017: error+warning
其余三个编译器: warning

体会编译器的差异

§ 2. 数据类型和表达式

2.2. 常量

2.2.3. 字符常量与字符串常量

2.2.3.2. 普通字符常量与转义字符常量

★ 一个字符常量可有几种表示形式

A (ASCII=65)	'A'	ESC (ASCII=27)	'\33'
	'\101'		'\033'
换行 (ASCII=10)	'\x41' (' \X41' 错!!!)		'\x1b'
			'\x1B'
	'\n'	双引号 (ASCII=34)	'\"'
	'\12'		'\42'
	'\012'		'\042'
	'\xA'		'\x22'
	'\x0A'		
	'\xa'		
	'\x0a'		

★ '0' 与 '\0' 的区别

'0' - ASCII 48 '\60' '\060' '\x30'

'\0' - ASCII 0 '\00' '\000' '\x0' '\x00'

★ 控制字符中，除空格外，都不能直接表示，\ ' " 等特殊图形字符也不能直接表示

2.2.3.3. 字符在内存中的存储

★ 一个字符常量只能表示一个字符，
一个字符在内存中占用一个字节，
字节的值为该字符的ASCII码

§ 2. 数据类型和表达式

2.2. 常量

2.2.3. 字符常量与字符串常量

2.2.3.4. 字符串常量

含义：连续多个字符组成的字符序列

表示：“字符串”

★ 可以是图形字符，也可以转义符

字符串的长度：字符序列中字符的个数

“abc123*#” = ?

“\x61\x62\x63\x061\x62\x063\x2a\x043” = ?

“\r\n\t\\A\\t\x1b\”\1234\xft\x2f\33” = ?

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

```
int main()
{
    cout << strlen("abc123*#") << endl;
    cout << strlen("\x61\x62\x63\x061\x62\x063\x2a\x043") << endl;
    cout << strlen("\r\n\t\\A\\t\x1b\”\1234\xft\x2f\33") << endl;

    return 0;
}
```

strlen是系统函数，
打印字符串的长度

用四个编译器分别编译，体会编译器的差异

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

```
int main()
{
    cout << strlen("abc123*#") << endl;
    cout << strlen("\x61\x62\x63\x061\x62\x063\x2a\x043") << endl;
    cout << strlen("\r\n\t\\A\\t\x1b\”\1234\xft\x2f\33") << endl;

    return 0;
}
```

用四个编译器分别编译，体会编译器的差异

§ 2. 数据类型和表达式

2.2. 常量

2.2.3. 字符常量与字符串常量

2.2.3.4. 字符串常量

含义：连续多个字符组成的字符序列

表示：“字符串”

★ 可以是图形字符，也可以转义符

字符串的长度：字符序列中字符的个数

"abc123*#" = ?

"\x61\x62\x63\061\62\063\x2a\043" = ?

"\r\n\t\\A\\t\x1b\"1234\xft\x2f\33" = ?

"\r\n\t\\A\\t\x1b\"1234\xft\x2f\33" = ?



C/C++在编译8/16进制转义符时有区别：

1、转义符后的合法8/16进制数若多于3/2个

"\1234"：8进制，编译不报错，长度为2

"\x2fa"：16进制，编译报error错

2、转义符后跟非法字符

"\9234"：8进制，编译报warning错，长度为4

"*123"：同上

"\xg123"：16进制，编译报error错

"\x*123"：同上

不再深入讨论，可自行查找资料

在内存中的存放：每个字符的ASCII码+字符串结束标志 '\0' (ASCII 0、尾0)

★ ""与" "的区别

"" - 空字符串，长度为0

" " - 含一个空格的字符串，长度为1

★ 'A'与"A"的区别

'A' - 字符常量，内存中占一个字节

"A" - 字符串常量，内存中占两个字节

★ 暂不讨论字符串中含尾0的情况 (第5、6章再讨论)

"Hello\0ABC"

\0

32 \0

65

65 \0

可写测试程序，观察下面的值

sizeof(""); strlen("");

sizeof(" "); strlen(" ");

sizeof('A'); strlen('A');

sizeof("A"); strlen("A");

问题1：上述8个中哪个错？

2：sizeof和strlen差异？

§ 2. 数据类型和表达式

2.2. 常量

2.2.4. 符号常量

用一个标识符代表的常量称为符号常量

```
#define pi 3.14159
```

★ 优点：含义清晰，修改方便

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

int main()
{
    ...;
    3.14159 * ...;
    ...;
    3.14159 * ...;
    ...;
    3.14159 * ...;
    ...;
    return 0;
}
```

```
#include <iostream>
using namespace std;
#define pi 3.14159
int main()
{
    ...;
    pi * ...;
    ...;
    pi * ...;
    ...;
    pi * ...;
    ...;
    return 0;
}
```

假设共10000处使用 π 值

1、要求降低 π 的精度为3.14

2、要求提高 π 的精度为3.1415926

那种方法方便？易于修改？

§ 2. 数据类型和表达式

2.3. 变量

2.3.1. 基本概念

变量：在程序运行中，值能够改变的量（有名字、值）

2.3.2. 标识符

标识符：用来标识变量名、符号常量名、函数名、数组名、结构体名、类名等的有效字符序列，称为标识符

C++的标识符命名规则：由字母或下划线开头，由字符、数字、下划线组成

合理： A a al _a _al al_b abc

不合理： 1a a-b

★ 标识符区分大小写

★ 长度≤32(或64)

★ 取名时通常按变量的含义

★ 必须先定义、后使用

★ 同级不能同名

2.3.3. 定义变量

数据类型 标识符名，标识符名，…；

int a;

unsigned int b, c;

long e, f;

short i, j;

float f1, f2;

double d1, d2;

char c1;

多个变量间用逗号分隔，
最后有分号

定义一个变量：
变量名为_____
存放一个____类型的数据
在内存中占用_____个字节

§ 2. 数据类型和表达式

2.3. 变量

2.3.3. 定义变量

★ VS2017允许用中文做变量名，通用性差，不建议

```
#include <iostream>
using namespace std;
int main()
{
    int 分数 = 78;
    cout << 分数 << endl;
    return 0;
}
```

★ VS2017支持auto自动定义类型，由初值决定类型，通用性差，不建议



```
#include <iostream>
using namespace std;
int main()
{
    auto x = 12;
    auto y = 1.2;
    cout << sizeof(x) << endl;
    cout << sizeof(y) << endl;
    return 0;
}
```

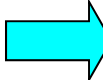
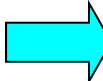
§ 2. 数据类型和表达式

2.3. 变量

2.3.4. 对变量赋初值

变量可以在定义的同时赋初值

<code>int a=10;</code>		<code>int a;</code> <code>a=10;</code>
<code>char b='B';</code>		<code>char b;</code> <code>b='B';</code>

<code>int a=10, b=15*2, c=30;</code>		<code>int a, b, c;</code> <code>a=10;</code> <code>b=15*2;</code> <code>c=30;</code>
<code>int a=10, b=10, c=10;</code>		<code>int a, b, c;</code> <code>a=10;</code> <code>b=10;</code> <code>c=10;</code>

★ 对多个变量赋同一初值，要分开进行

`int a=b=c=10;` (错误)

`int a=10, b=a, c=b+1;` (可用已定义变量赋初值)

★ 若变量定义后未赋值即访问，VS2017直接报error

(其他编译器报warning/不报错，并打印不可预知值)

```
#include <iostream>
using namespace std;
int main()
{
    int k;
    cout << k << endl;
    k=10;
    return 0;
}
```

不可预知值：不同编译系统表现不一样，有些每次都一样，有些每次不同

本例中，k未赋初值即使用
VS2017报error错
其它编译器不报错

§ 2. 数据类型和表达式

2.3. 变量

2.3.4. 对变量赋初值

变量可以在定义的同时赋初值

P.28中间: `float a, b=5.78*3.5, c=2*sin(2.0);`

再次强调:

不同编译器可能在细节处理上有差异, 前几个都是简单的例子, 后面限于时间关系, 均以VS2017为准, 完成多编译器作业时, 要有能力去解决差异

```
#include <iostream>
using namespace std;
int main()
{
    float a, b=5.78*3.5, c=2*sin(2.0);
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    return 0;
}
```

sin是系统的库函数
求sin值, 单位弧度

本例中:

1. a未赋初值即使用
2. 库函数sin未包含对应头文件

VS2017报error
其它编译器不报错
VS2017不报错
其它编译器报error

虽然VS2017可以不包含cmath头文件, 但建议加上, 方便移植

```
#include <iostream>
using namespace std;
int main()
{
    double b=5.78*3.5, c=2*sin(2.0);

    cout << b << endl;
    cout << c << endl;
    return 0;
}
```

VS2017: 0 error 0 warning
其他编译器: 报error

```
#include <iostream>
#include <cmath> //数学函数
using namespace std;
int main()
{
    double a=0, b=5.78*3.5, c=2*sin(2.0);
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    return 0;
}
```

库函数需要加头文件
不需要像例1.3
写出sin的具体实现

四个编译器均正常
0 error 0 warning

§ 2. 数据类型和表达式

2.3. 变量

2.3.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.3.5.1. 整型变量的使用

★ 整型常量缺省是int型，long型后通常加l(L)表示，unsigned通常加u(U)，short型无特殊后缀

123 123L 123U 123UL 123LU

★ 数据的溢出在C++中不认为是错误

★ 同长度的signed与unsigned相互赋值时，可能会有不正确的结果
(机内二进制相同，但十进制表现不同)

★ 不同长度的整型数据相互赋值时，遵循如下规则：

短=>长：低位赋值，高位填充符号位(短为signed)

填充0 (短为unsigned)

长=>短：低位赋值，高位丢弃

(赋值按机内二进制进行，可能导致不正确)

★ 数据的溢出在C++中不认为是错误

```
short a=32767, b=a+1;  
a= 0111111111111111 = 32767  
b= 1000000000000000 = -32768
```

```
short a=-32768, b=a-1;  
a= 1000000000000000 = -32768  
b= 0111111111111111 = 32767
```

```
unsigned short a=65535, b=a+1;  
a= 1111111111111111 = 65535  
b= 1 0000000000000000 = 0  
    (高位溢出, 舍去)
```

```
unsigned short a=0, b=a-1;  
a= 0000000000000000 = 0  
b= 1111111111111111 = 65535  
    (借位不够, 虚借一位)
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    short a1=32767, b1=a1+1;  
    cout << a1 << ' ' << b1 << endl;  
  
    short a2=-32768, b2=a2-1;  
    cout << a2 << ' ' << b2 << endl;  
  
    unsigned short a3=65535, b3=a3+1;  
    cout << a3 << ' ' << b3 << endl;  
  
    unsigned short a4=0, b4=a4-1;  
    cout << a4 << ' ' << b4 << endl;  
  
    return 0;  
}
```

从十进制角度理解,
相当于以几为模?

?

★ 同长度的signed与unsigned相互赋值时，可能会有不正确的结果
(机内二进制相同，但十进制表现不同)

```
short a=100; unsigned short b=a;  
a= 0000000001100100 = 100  
b= 0000000001100100 = 100
```

```
short a=-10; unsigned short b=a;  
a= 111111111110110 = -10  
b= 111111111110110 = 65526
```

```
unsigned short a=100; short b=a;  
a= 0000000001100100 = 100  
b= 0000000001100100 = 100
```

```
unsigned short a=40000; short b=a;  
a= 1001110001000000 = 40000  
b= 1001110001000000 = -25536
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    short a1=100;  
    unsigned short b1=a1;  
    cout << a1 << ' ' << b1 << endl;  
  
    short a2=-10;  
    unsigned short b2=a2;  
    cout << a2 << ' ' << b2 << endl;  
  
    unsigned short a3=100;  
    short b3=a3;  
    cout << a3 << ' ' << b3 << endl;  
  
    unsigned short a4=40000;  
    short b4=a4;  
    cout << a4 << ' ' << b4 << endl;  
  
    return 0;  
}
```

?

★ 短=>长: 低位赋值, 高位填充符号位(短为signed)
填充0 (短为unsigned)

```
short a=100; long b=a;  
a= 填充符号位 0000000001100100 =100  
b= 00000000000000000000000001100100 =100
```

```
unsigned short a=100; long b=a;  
a= 填充0 0000000001100100 =100  
b= 00000000000000000000000001100100 =100
```

```
short a=32767; long b=a;  
a= 填充符号位 0111111111111111 =32767  
b= 00000000000000000111111111111111 =32767
```

```
unsigned short a=32767; long b=a;  
a= 填充0 0111111111111111 =32767  
b= 00000000000000000111111111111111 =32767
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    short a1=100;  
    long b1=a1;  
    cout << a1 << ' ' << b1 << endl;  
  
    unsigned short a2=100;  
    long b2=a2;  
    cout << a2 << ' ' << b2 << endl;  
  
    short a3=32767;  
    long b3=a3;  
    cout << a3 << ' ' << b3 << endl;  
  
    unsigned short a4=32767;  
    long b4=a4;  
    cout << a4 << ' ' << b4 << endl;  
  
    return 0;  
}
```

?

★ 短=>长: 低位赋值, 高位填充符号位(短为signed)
填充0 (短为unsigned)

```
short a=-10; long b=a;  
a= 填充符号位 1111111111110110 =-10  
b= 11111111111111111111111111110110 =-10
```

```
short a=-10; unsigned long b=a;  
a= 填充符号位 1111111111110110 =-10  
b= 11111111111111111111111111110110 =4294967286
```

```
unsigned short a=40000; long b=a;  
a= 填充0 1001110001000000 =40000  
b= 00000000000000001001110001000000 =40000
```

```
unsigned short a=40000; unsigned long b=a;  
a= 填充0 1001110001000000 =40000  
b= 00000000000000001001110001000000 =40000
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    short a1=-10;  
    long b1=a1;  
    cout << a1 << ' ' << b1 << endl;  
  
    short a2=-10;  
    unsigned long b2=a2;  
    cout << a2 << ' ' << b2 << endl;  
  
    unsigned short a3=40000;  
    long b3=a3;  
    cout << a3 << ' ' << b3 << endl;  
  
    unsigned short a4=40000;  
    unsigned long b4=a4;  
    cout << a4 << ' ' << b4 << endl;  
  
    return 0;  
}
```

?

★ 长=>短: 低位赋值, 高位丢弃

```
long a=100; short b=a;  
a= 00000000000000000000000001100100 =100  
b= 0000000001100100 =100
```

```
long a=70000;short b=a;  
a= 000000000000000010001000101110000 =70000  
b= 0001000101110000 =4464
```

```
long a=100000;short b=a;  
a= 000000000000000011000011010100000 =100000  
b= 1000011010100000 =-31072
```

```
long a=100000;unsigned short b=a;  
a= 000000000000000011000011010100000 =100000  
b= 1000011010100000 =34464
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    long a1=100;  
    short b1=a1;  
    cout << a1 << ' ' << b1 << endl;  
  
    long a2=70000;  
    short b2=a2;  
    cout << a2 << ' ' << b2 << endl;  
  
    long a3=100000;  
    short b3=a3;  
    cout << a3 << ' ' << b3 << endl;  
  
    long a4=100000;  
    unsigned short b4=a4;  
    cout << a4 << ' ' << b4 << endl;  
  
    return 0;  
}
```

?


★ 长=>短：低位赋值，高位丢弃

```
long a=-10;   short b=a;  
a= 11111111111111111111111111111111 =-10  
b=                1111111111111111 =-10
```

```
long a=-10;    unsigned short b=a;  
a= 11111111111111111111111111111111 =-10  
b=              11111111111111111111111111111111 =65526
```

```
unsigned long a=4294967286;short b=a;  
a= 111111111111111111111111110110 =4294967286  
b=                1111111111110110 =-10
```

```
unsigned long a=4294967286; unsigned short b=a;  
a= 111111111111111111111111110110 =4294967286  
b=                1111111111110110 =65526
```



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

int main()
{
    long a1=-10;
    short b1=a1;
    cout << a1 << ' ' << b1 << endl;

    long a2=-10;
    unsigned short b2=a2;
    cout << a2 << ' ' << b2 << endl;

    unsigned long a3=4294967286;
    short b3=a3;
    cout << a3 << ' ' << b3 << endl;

    unsigned long a4=4294967286;
    unsigned short b4=a4;
    cout << a4 << ' ' << b4 << endl;

    return 0;
}
```



§ 2. 数据类型和表达式

2.3. 变量

2.3.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.3.5.2. 浮点型变量的使用

- ★ 浮点数在内存中的存储分为三部分，分别是符号位、指数部分和尾数部分
- ★ 浮点数有指定的有效位数，有效位数以外的数字被舍去，会产生误差
- ★ 浮点常量缺省为double型，如需表示为float型，可以在后面加f(F)

1.23 1.23F

- ★ float赋值给double一定正确，double赋值给float不一定正确，且不同于整型的高位丢弃，VS2017会出现 **inf** 的形式，具体原因不作要求

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

int main()
{
    double d=1.23456e38;
    float f1=d;           //warning
    float f2=d*10;        //warning
    cout << d << endl;    1.23456e+38
    cout << f1 << endl;    1.23456e+38
    cout << f2 << endl;    inf
    return 0;
}
```

§ 2. 数据类型和表达式

2.3. 变量

2.3.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.3.5.3. 字符型变量的使用

- ★ 直接表示 ' 空格或大部分可见的图形字符'
 - ★ 转义符表示 '\字符、八、十六进制数'
 - ★ 一个字符常量可有几种表示形式
 - ★ '0' 与 '\0' 的区别
 - ★ 控制字符中，除空格外，都不能直接表示，\ ' " 等特殊图形字符也不能直接表示
 - ★ 存储形式：一个字符常量只能表示一个字符，一个字符在内存中占用一个字节
字节的值为该字符的ASCII码
 - ★ 注意变量与常量的区别
- ```
char a, b;
a='a'; //左边是变量，右边是常量
b='\101';
```

## § 2. 数据类型和表达式

### 2.3. 变量

#### 2.3.5. 各种类型变量的使用 (归纳+补充+重点+难点)

##### 2.3.5.3. 字符型变量的使用

#### ★ 注意变量与常量的区别

```
char a, b;
```

```
a = 'a'; // 左边是变量，右边是常量
```

```
b = '\101';
```

#### ★ 输出形式：数字/字符

P. 23-24 例2.1、2.2

|                                                                                                                                                                                                    |                                                                                  |                                                                                                                                                                                                                                               |                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <pre>//例 2.1 #include &lt;iostream&gt; using namespace std;  int main() {     int i, j;     i = 'A';     j = 'B';     cout &lt;&lt; i &lt;&lt; ' ' &lt;&lt; j &lt;&lt; '\n';     return 0; }</pre> | <pre>'A':      01000001 i : 00..00 01000001       24个  可以理解为1字节整数 赋值给4字节整数</pre> | <pre>//例 2.2 #include &lt;iostream&gt; using namespace std;  int main() {     char c1, c2;     c1 = 'a';     c2 = 'b';     c1 = c1 - 32;     c2 = c2 - 32;     cout &lt;&lt; c1 &lt;&lt; ' ' &lt;&lt; c2 &lt;&lt; endl;     return 0; }</pre> | <pre>c1:      01100001 32: 00..00 00100000       24个  可以理解为1字节整数 减4字节整数，再赋值 给1字节整数 c1:      01000001       (减32后的结果)</pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|

## § 2. 数据类型和表达式

### 2.3. 变量

#### 2.3.5. 各种类型变量的使用 (归纳+补充+重点+难点)

##### 2.3.5.3. 字符型变量的使用

★ 注意变量与常量的区别

★ 输出形式：数字/字符

★ 与整数的互通性：可当作1字节的整数参与运算(signed/unsigned)

```
#include <iostream>
using namespace std;
int main()
{
 char a;
 a=65;
 cout << a << endl;
 return 0;
}
```

A

```
#include <iostream>
using namespace std;
int main()
{
 char a;
 a='A';
 cout << a << endl;
 return 0;
}
```

A

结果虽相同，但过程不同

a=65 : 4字节赋1字节，丢弃24bit 0

a='A' : 1字节赋1字节

a的赋值方法：

1、字符常量形式

a='A' / a='\101' / a='\x41'

2、整数常量形式

a=65 / a=0101 / a=0x41 / a=0X41

## § 2. 数据类型和表达式

### 2.3. 变量

#### 2.3.5. 各种类型变量的使用 (归纳+补充+重点+难点)

##### 2.3.5.4. 字符串型变量的使用

★ C++中无字符串变量，可用一维字符数组来表示字符串变量，第5章讨论

## § 2. 数据类型和表达式

### 2.3. 变量

#### 2.3.6. 常变量

含义：在程序执行过程中值不能改变的变量

形式：const 数据类型 变量名=初值；

```
const int a=10;
```

```
const double pi=3.14159;
```

★ 常变量必须`在定义时赋初值`，且在执行过程中`不能再次赋值`，否则编译错误

```
#include <iostream>
using namespace std;
int main()
{
 const int a;

 return 0;
}
//编译报错
```

```
#include <iostream>
using namespace std;
int main()
{
 const int a=10;
 a=15;

 return 0;
}
//编译报错
```



## § 2. 数据类型和表达式

### 2.3. 变量

#### 2.3.6. 常变量

★ 常变量与符号常量使用方法相似，但本质有区别（**推荐使用常变量**）

|                                                                                                                                                                                                                               |                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include &lt;iostream&gt; using namespace std; #define pi 3.14159 int main() { double r=3, s, v;   s = pi*r*r;   v = pi*r*r*r*4/3;   cout &lt;&lt; s &lt;&lt; endl;   cout &lt;&lt; v &lt;&lt; endl;   return 0; }</pre> | <pre>#include &lt;iostream&gt; using namespace std; int main() { const double pi=3.14159;   double r=3, s, v;   s = pi*r*r;   v = pi*r*r*r*4/3;   cout &lt;&lt; s &lt;&lt; endl;   cout &lt;&lt; v &lt;&lt; endl;   return 0; }</pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

★ 常变量与符号常量的本质有区别（**具体不做要求**），推荐常变量

## § 2. 数据类型和表达式

### 2. 4. C++的运算符

#### 2. 4. 1. C++的运算符的种类

P. 29 15类

#### 2. 4. 2. 运算符的优先级和结合性

优先级：不同运算符进行混合运算时，按优先级的高低依次执行

结合性：**同级**运算符进行混合运算时，按结合性的方向依次进行处理

左结合：**从左至右**进行同级运算符的混合运算

右结合：**从右至左**进行同级运算符的混合运算

P. 465 附录B (**1最高 17最低 熟记!!!**)

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.1. 基本的算术运算符

+   -   \*   /   %

★ 字符型可以参与算术运算，当作1字节的整型数，值为其ASCII码

10 + 'A' => 10 + 65 => 75

★ %的使用(求模，整数相除求余数)，%两侧为整型(int, long, short, char)

|                                                                          |                                                                            |                                                                                |                                                                               |
|--------------------------------------------------------------------------|----------------------------------------------------------------------------|--------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| 10 % 3    1 $\begin{array}{r} 3 \\ 10 \\ \underline{9} \\ 1 \end{array}$ | 10 % -3   1 $\begin{array}{r} -3 \\ 10 \\ \underline{-9} \\ 1 \end{array}$ | -10 % 3    -1 $\begin{array}{r} -3 \\ -10 \\ \underline{-9} \\ -1 \end{array}$ | -10 % -3   -1 $\begin{array}{r} 3 \\ -10 \\ \underline{-9} \\ -1 \end{array}$ |
| 10 % 7    3 $\begin{array}{r} 3 \\ 10 \\ \underline{9} \\ 1 \end{array}$ | 10 % -7   3 $\begin{array}{r} -3 \\ 10 \\ \underline{-9} \\ 1 \end{array}$ | -10 % 7    -3 $\begin{array}{r} -3 \\ -10 \\ \underline{-9} \\ -1 \end{array}$ | -10 % -7   -3 $\begin{array}{r} 3 \\ -10 \\ \underline{-9} \\ -1 \end{array}$ |
| 7 % 3    1 $\begin{array}{r} 3 \\ 7 \\ \underline{6} \\ 1 \end{array}$   | 7 % -3   1 $\begin{array}{r} -3 \\ 7 \\ \underline{-6} \\ 1 \end{array}$   | -7 % 3    -1 $\begin{array}{r} -3 \\ -7 \\ \underline{-6} \\ -1 \end{array}$   | -7 % -3   -1 $\begin{array}{r} 3 \\ -7 \\ \underline{-6} \\ -1 \end{array}$   |
| 3 % 7    3 $\begin{array}{r} 3 \\ 3 \\ \underline{3} \\ 0 \end{array}$   | 3 % -7   3 $\begin{array}{r} -3 \\ 3 \\ \underline{-3} \\ 0 \end{array}$   | -3 % 7    -3 $\begin{array}{r} -3 \\ -3 \\ \underline{-3} \\ 0 \end{array}$    | -3 % -7   -3 $\begin{array}{r} 3 \\ -3 \\ \underline{-3} \\ 0 \end{array}$    |

★ 整数相除 ( / ) 的使用

结果为整数(舍去小数, 非四舍五入)

```
//比较容易犯的错误
#include <iostream>
using namespace std;
int main()
{
 double pi=3.14159, v1, v2;
 int r=3, h=5;
 v1=1/3*pi*r*r*h;
 v2=4/3*pi*r*r*r;
 cout << v1 << endl;
 cout << v2 << endl;
 return 0;
}
```

|                     |         |         |
|---------------------|---------|---------|
|                     | 实际      | 期望      |
| cout << v1 << endl; | 0       | 47.1238 |
| cout << v2 << endl; | 84.8229 | 113.097 |

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.2. 算术运算符的优先级与结合性

优先级：\* / % 4级  
+ - 5级 都是双目运算符

结合性：左结合

#### 2.5.3. 算术表达式

含义：用算术运算符及括号将操作数（运算对象：常量、变量、函数）连接起来，组成符合C++语法规则的算式

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.4. C++的表达式求值 (补充, 重要!!!)

表达式: 由若干操作数和操作符构成的符合C++语法的算式

表达式的求值: 整个表达式从左到右依次分析, 分析原则如下

★ 若只有一个运算符, 则求值

$a+b$

★ 若有两个运算符, 若

① 左边运算符的优先级高于右边运算符 或

② 左边运算符的优先级等于右边运算符, 且该级别运算符是左结合, 则对左边运算符求值, 其值再参与后续的运算

$a*b+c$

$a+b+c$

★ 若有两个运算符, 若

① 左边运算符的优先级低于右边运算符 或

② 左边运算符的优先级等于右边运算符, 且该级别运算符是右结合, 则先忽略左边运算符, 继续向后分析, 直到右边运算符被求值后再次分析左边运算符

$a+=a-=a*a$

$a+b*c$

$a+b*c-d$

$a+b*c*d$

## § 2. 数据类型和表达式

### 2. 5. 算术运算符与算术表达式

#### 2. 5. 4. C++的表达式求值 (补充, 重要!!!)

#### ★ 用栈(LIFO)的形式理解表达式求值过程

LIFO: Last In First Out

运算数栈: 存放运算数

运算符栈: 存放运算符

规则: (1) 运算数/运算符分别进各自的栈

(2) 若欲进栈的运算符级别高于或等于(右结合)栈顶运算符, 则进栈

(3) 若欲进栈的运算符级别低于或等于(左结合)栈顶运算符, 则先将栈顶运算符  
计算完成, 计算数为运算数栈的两个元素, 运算符及运算数出栈, 运算结果进栈

(4) 重复上述步骤至运算符栈为空, 运算数栈只有一个元素, 否则认为语法错

例:  $10 + 'a' + i * f - d / e$  (P. 31中间)

说明: 10 - 整型常量

'a' - 字符型常量

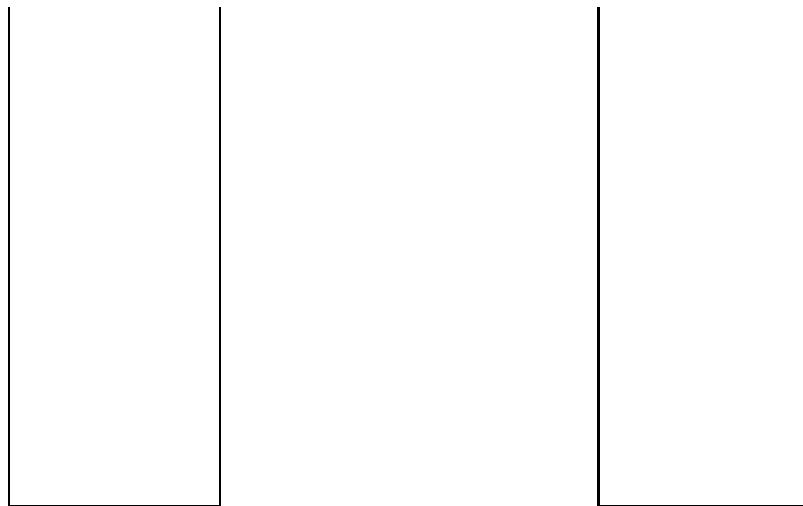
i - 某个int型变量

f - 某个float型变量

d - 某个double型变量

e - 某个long型变量

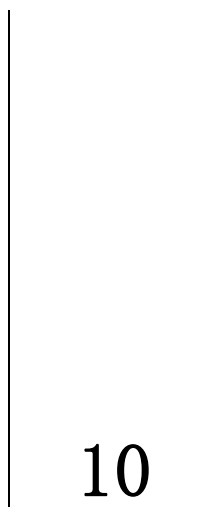
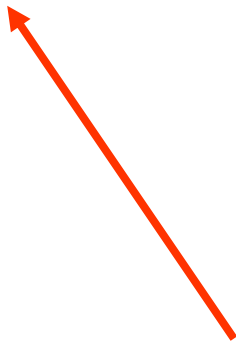
例:  $10 + a' + i * f - d / e$



初始: 两栈均为空



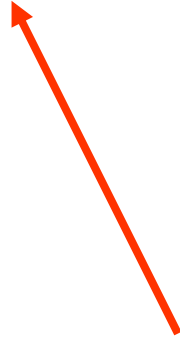
例:  $10 + a' + i * f - d / e$



10进栈



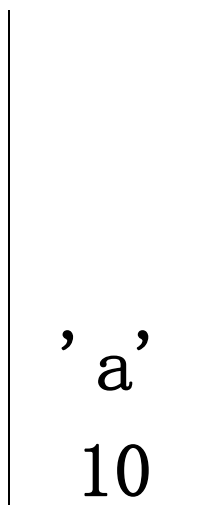
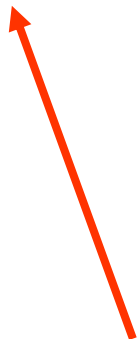
例:  $10 + a' + i * f - d / e$



+进栈



例: 10 + 'a' + i \* f - d / e



'a' 进栈



例:  $10 + 'a' + i * f - d / e$



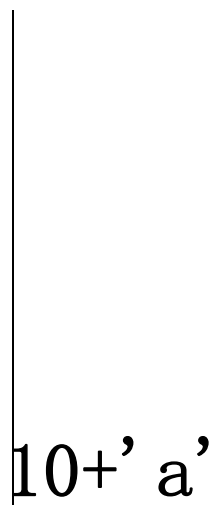
'a'  
10

+

要进栈的 (+) 等于栈顶 (+)，且左结合，求值

例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$



要进栈的 (+) 等于栈顶 (+)，且左结合，求值

例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$



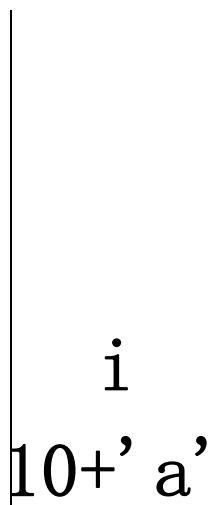
$10 + a'$

+进栈

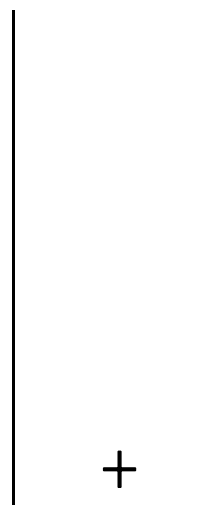
+

例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$



i进栈



例:  $10+'a'+i*f-d/e$

步骤①  $10+'a'$



$i$   
 $10+'a'$

\*进栈

$*$   
 $+$

(要进栈的\*高于栈顶的+)



例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$



f  
i  
 $10 + a'$

f进栈

\*

+

例:  $10+' a' +i*f-d/e$

步骤①  $10+' a'$



f  
i  
 $10+' a'$

\*  
+

要进栈的 (-) 低于栈顶的 (\*), 先计算

例:  $10+' a' +i*f-d/e$

步骤①  $10+' a'$

步骤②  $i*f$



$i*f$   
 $10+' a'$

$+$

要进栈的  $(-)$  低于栈顶的  $(*)$ ，先计算

例:  $10+' a' +i*f-d/e$

步骤①  $10+' a'$

步骤②  $i*f$



$i*f$   
 $10+' a'$

$+$

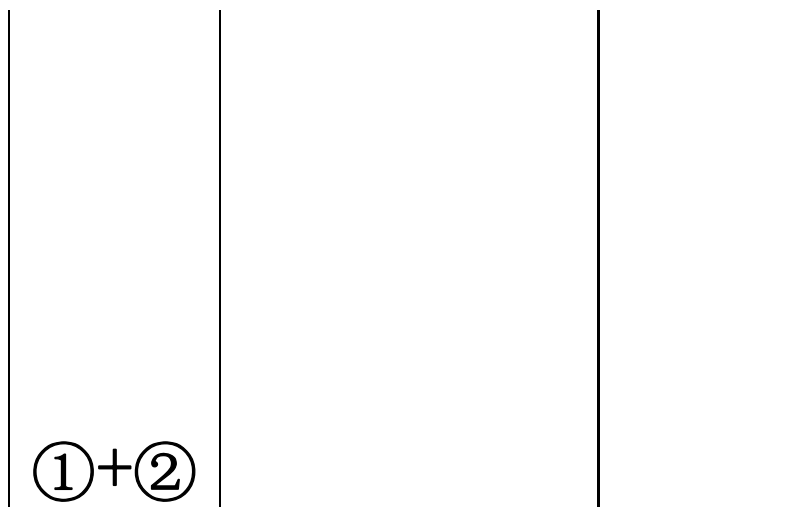
要进栈的  $(-)$  等于栈顶的  $(+)$ ，左结合，先计算

例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$

步骤②  $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)



要进栈的(-)等于栈顶的(+), 左结合, 先计算

例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$

步骤②  $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)



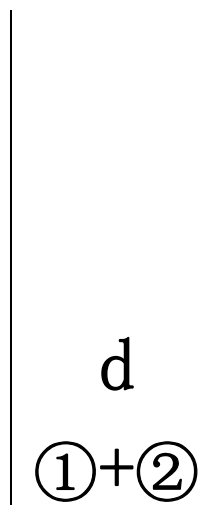
-进栈

例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$

步骤②  $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)



d进栈

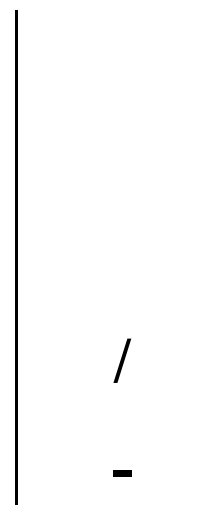
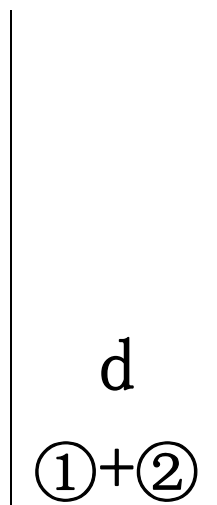


例:  $10+' a' +i*f-d/e$

步骤①  $10+' a'$

步骤②  $i*f$

步骤③ ①+② (步骤①的和 + 步骤②的积)



/进栈 (要进栈的/高于栈顶的-)

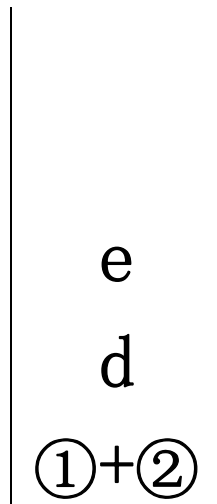


例:  $10+' a' +i*f-d/e$

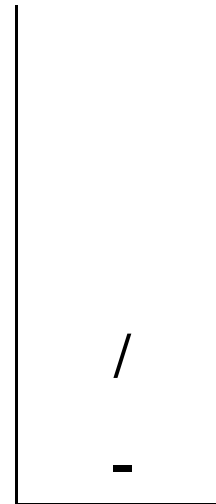
步骤①  $10+' a'$

步骤②  $i*f$

步骤③ ①+② (步骤①的和 + 步骤②的积)



e进栈



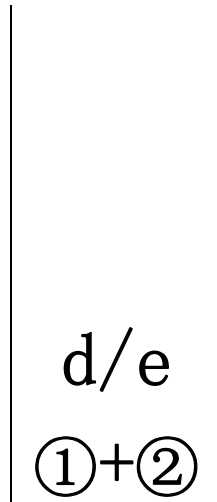
例:  $10 + a' + i * f - d / e$

步骤①  $10 + a'$

步骤②  $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)

步骤④  $d / e$



计算  $d/e$

例:  $10 + a' + i * f - d / e$

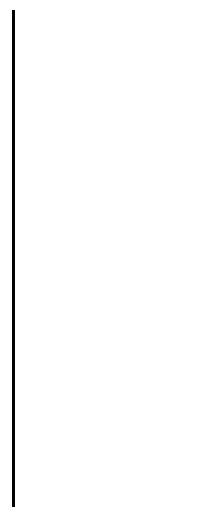
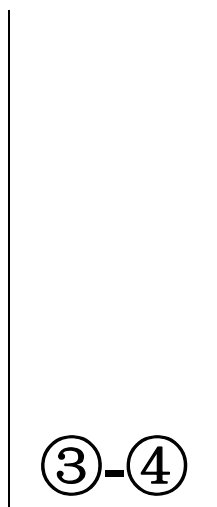
步骤①  $10 + a'$

步骤②  $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)

步骤④  $d / e$

步骤⑤ ③-④ (步骤③的和 - 步骤④的商)



计算 ③-④

例:  $10 + a' + i * f - d / e$

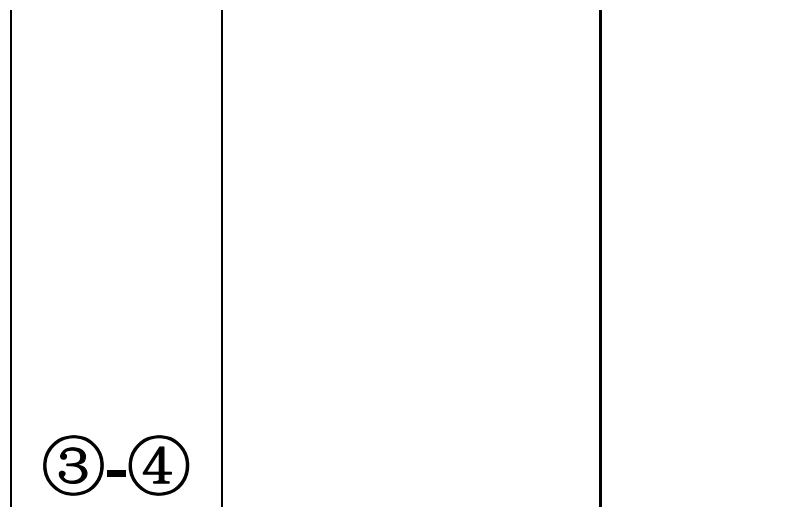
步骤①  $10 + a'$

步骤②  $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)

步骤④  $d / e$

步骤⑤ ③-④ (步骤③的和 - 步骤④的商)



表达式分析并求值完成

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.5. 字符型、整型、实型的混合运算

##### ★ 运算的方法

10+'a'+1.5-8765.1234\*'b' (P. 30最后)

如果某个运算符涉及到两个数据不是同一类型，则需要先转换成同一类型，再进行运算

##### ★ 转换的优先级

P. 31 图2.7 (有错)

高 ↑ double  
float  
unsigned long  
long  
unsigned [int]  
低 int <- char, short

<- 表示必定的转换

```
char a;
short b;

a+b (int + int)
```

思考：怎么证明说法的正确性？（引申：如何解决学习过程中碰到的含义不清的问题？）

```
#include <iostream>
using namespace std;
int main()
{
 short a = 1;
 short b = 32767;

 cout << a+b << endl;

 return 0;
}
```

验证：int <- char / short  
是必定的转换

已知：short a=32767;  
short b=a+1;  
则：b是-32768

预测：结果为-32768 则验证不正确  
结果为 32768 则验证正确

```
#include <iostream>
using namespace std;
int main()
{
 short a = 1;

 cout << sizeof(a+'A') << endl;

 return 0;
}
```

预测：结果为 2 则验证不正确  
结果为 4 则验证正确

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.5. 字符型、整型、实型的混合运算

★ 运算的方法

★ 转换的优先级

★ <-表示必定的转换

10+'a'+i\*f-d/e (P.31 解释中float必转double错)

① 10+'a' int + int

② i\*f float \* float

③ ①+② float + float

④ d/e double / double

⑤ ③-④ double - double

注意：不是将整个算式中优先级最高的最先计算，而是分步进行

★ 同级的signed与unsigned混合时，以unsigned为准

★ 类型转换由系统隐式进行

★ 类型转换时，不是一次全部转换成最高级，而是依次转换

'a'+10+15L+1.2

'a'+10 : char=>int int+int

'a'+10+15L : int=>long long+long

'a'+10+15L+1.2 : long=>double double+double

```
#include <iostream>
using namespace std;
int main()
{ int a=2;
 int b=3;
 cout << a-b << endl;
 return 0;
}
```

-1

```
#include <iostream>
using namespace std;
int main()
{ int a=2;
 unsigned int b=3;
 cout << a-b << endl;
 return 0;
}
```

4294967295

比较容易犯的错误：

float pi=3.14159, v1, v2;

int r=3, h=5;

圆锥体积: v1=1/3\*pi\*r\*r\*h;

球体积: v2=4/3\*pi\*r\*r\*r;

实际

0

84.8229

期望

47.1238

113.097

此例正因为是依次转换，才得到上述结果

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.6. 自增与自减运算符

##### 2.5.6.1. 形式

++ (--) 变量名      先自增/减1, 后使用 (前缀)

变量名 ++ (--)      先使用, 后自增/减1 (后缀)

```
#include <iostream>
using namespace std;
int main()
{ int i=3, j;
 j = ++i; //前缀
 cout << i << endl; 4
 cout << j << endl; 4
 return 0;
}
```

- ① ++优先级高于=, 先计算++
- ② 因为++是前缀, 先++i成为4
- ③ 再将i值4赋值给j

```
#include <iostream>
using namespace std;
int main()
{ int i=3, j;
 j = i++; //后缀
 cout << i << endl; 4
 cout << j << endl; 3
 return 0;
}
```

- ① ++优先级高于=, 先计算++
  - ② 后缀++, 将i的原值3保留到某个中间变量中  
接下来, 因为不同的编译器(例如VS系列和部分gcc), 可能有两种执行顺序
  - ③ 先将原值赋给j      ③ 先进行++, i值为4
  - ④ 再进行++, i值为4      ④ 再将原值赋给j
- 无论哪种顺序, 都是j=3 i=4

```
#include <iostream>
using namespace std;
int main()
{
 int i = 3;
 i = i++;
 cout << i << endl;
 return 0;
}
```

无聊的做法, 仅为了理解  
顺序1      顺序2  
i值为4      i值为3  
VS2017      其余三编译器

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.6. 自增与自减运算符

##### 2.5.6.1. 形式

##### 2.5.6.2. 使用

★ ++/--的**前/后缀**对变量自身无影响，影响的是参与运算的表达式

```
int i=3;
```

```
i++;
```

单独成为语句时，前后缀等价

```
++i;
```

★ 前后缀的优先级和结合性均不相同

后缀：优先级(2)    前缀：优先级(3)

左结合

右结合

```
int i=3, j;
```

```
<1> -++i; i=4
```

```
<2> j = -++i; i=4 j=-4
```

① 前缀++(3)和-(3)优先级相同，右结合，先++

② ++i成为4

③ 4变为-4，存于中间变量中，  
对<1>无意义，所以有编译警告  
对<2>则赋值给j

注意：负号(-)运算符的结果保存在中间变量中，  
不会影响原常量、变量、表达式（i仍然是4）

```
#include <iostream>
using namespace std;
int main()
{ int i=3;
 -++i;
 cout << i << endl;

 return 0;
}
```

4

编译有警告

```
#include <iostream>
using namespace std;
int main()
{ int i=3, j;
 j = -++i;
 cout << i << endl;
 cout << j << endl;
 return 0;
}
```

4

-4



## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.6. 自增与自减运算符

##### 2.5.6.1. 形式

##### 2.5.6.2. 使用

★ ++/--的**前/后缀**对变量自身无影响，影响的是参与运算的表达式

```
int i=3;
```

```
i++;
```

单独成为语句时，前后缀等价

```
++i;
```

★ 前后缀的优先级和结合性均不相同

后缀：优先级(2)    前缀：优先级(3)

左结合

右结合

```
int i=3, j;
```

<1> -++i;    i=4

<2> j = -++i; i=4    j=-4

仿照上例，请自行给出解释

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{ int i=3;
```

```
 -i++;
```

```
 cout << i << endl; 4
```

```
 return 0;
```

```
}
```

编译有警告

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{ int i=3, j;
```

```
 j = -i++;
```

```
 cout << i << endl; 4
```

```
 cout << j << endl; -3
```

```
 return 0;
```

```
}
```

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.6. 自增与自减运算符

##### 2.5.6.1. 形式

##### 2.5.6.2. 使用

★ ++/--的**前/后缀**对变量自身无影响，影响的是参与运算的表达式

★ 前后缀的优先级和结合性均不相同

★ ++/--后，变量的值改变**(不能对常量、表达式)**

```
int i=3, j=4;
```

```
const int k=3;
```

(i+j)++: 无地方存放++后的结果 **(错)**

10++: 常量值不能被改变 **(错)**

k++: 常变量的值不能被改变 **(错)**

★ 不主张对同一个变量的多个++/--出现在同一个表达式中

**(i++)+(i++)+(i++)**

**(不同编译系统处理方式不同，不深入讨论)**

```
//用不同编译器测试本程序
#include <iostream>
using namespace std;
int main()
{
 int i = 3, k;
 k = (i++) + (i++) + (i++);
 cout << i << ' ' << k << endl;
 return 0;
}
```

## § 2. 数据类型和表达式

### 2.5. 算术运算符与算术表达式

#### 2.5.7. 强制类型转换

(类型名) (表达式) / 类型名 (表达式)

(int) (a+b) / int (a+b)

(int) a / int (a)

C形式

C++形式

```
int a;
```

```
float b, c;
```

a+(int)b / a+int(b) int型

a+(int)b+c / a+int(b)+c float型

a+(int)(b+c) / a+int(b+c) int型

C形式

C++形式

★ 必须在程序中显式使用

★ 强制转换后，原变量的值、类型不变(强制转换的结果放在一个中间变量中)

## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.1. 形式

变量 = 数据（常量、变量、表达式）

a=b

a=15

a=b\*c-d

★ 赋值运算符左边必须是变量名

★ 若赋值运算符的左右类型不同，则以左值类型为准进行转换

```
float a;
```

```
a = 1.2; （右值double转换为左值float型，VS2017有warning）
```

## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.2. 字符型、整型、实型间相互赋值的类型转换

★ float/double => char/short/int/long时, 取整

● 保证合理范围, 否则可能溢出 (C++语法不错)

f=123.0 => char=123

f=12345.67 => short 12345

=> char 溢出(值不可信)

f=1234567.89 => long 1234567

=> char 溢出 (值不可信)

=> short 溢出(值不可信)

值不可信: 不同编译器处理不可信值的方法不同, 不需要深究

★ char/short/int/long => float/double, 不变

后面补零, 不会溢出, 但精度受影响

char 123 => float 123

int 12345 => double 12345

long 1234567 => float 1.23457e+06

long 123456700 => float 1.23457e+08

long 123456789 => float 1.23457e+08

相同

● 实型数运算时要四舍五入

★ float/double相互赋值时

float -> double : 不会错

double -> float : 可能溢出 (VS2017中是 inf 形式)

## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.2. 字符型、整型、实型间相互赋值的类型转换

★ char/short/int/long相互赋值时

少字节->多字节：低位赋值，高位补符号位/0

十进制的表示形式可能不同

char 123 => long 123

short -10 => unsigned int 4294967286

多字节->少字节：低位赋值，高位自动截断

可能溢出

long 1234 => short 1234

long 70000 => short 4464

★ signed与unsigned相互赋值时，值的二进制形式不变，但十进制表示形式可能不同

unsigned short a=65535;

short b;

b=a; b=-1

(unsigned)65535 = 1111 1111 1111 1111 = (signed)-1

## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.3. 复合的赋值运算符

变量 复合赋值运算符 表达式 (常量、变量、表达式)

P. 465 附录B 优先级15 共10种 (目前: + - \* / %)

$a+=b \Rightarrow a=a+b$

$a*=b+1 \Rightarrow a=a*(b+1)$

★ 优先级相同 (注意: +、\*优先级不同, 但+=、\*=优先级相同)

## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.4. 赋值表达式

##### 2.6.4.1. 含义

将一个变量和一个表达式用赋值运算符连接起来

##### 2.6.4.2. 形式

变量 = 表达式（常量、变量）

$a=b$

$a=5$

$a=b+c$

$a=(b=5)$     如何理解？



## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.4. 赋值表达式

##### 2.6.4.3. 赋值表达式的值 (含复合赋值表达式)

和变量的值相等

$a=b=c=5 \Rightarrow a=(b=(c=5))$

理解:

- 1、将常量5赋值给c, 赋值表达式  $c=5$  的值也是5
- 2、将赋值表达式  $c=5$  的值5赋值给b, 赋值表达式  $b=(c=5)$  的值也是5
- 3、将赋值表达式  $b=(c=5)$  的值5赋值给a, 赋值表达式  $a=(b=(c=5))$  的值也是5

★ 变量赋初值时, `int a=b=c=5;`不行

|                                 |    |                           |
|---------------------------------|----|---------------------------|
| <code>int a=5, b=5, c=5;</code> | 正确 | <code>int a, b, c;</code> |
| <code>int a=b=c=5;</code>       | 错误 | <code>a=b=c=5;</code> 正确  |

★ 赋值表达式的值可以参与其它表达式的运算

$(a=3*5)=4*3$

$3*5 \quad a=3*5 \quad a=15$

$4*3 \quad a=4*3 \quad a=12$

虽然=的左边是一个表达式, 不是要求的变量  
但先计算该表达式, 使=执行时形式为变量

$a=3*5=4*3$

$3*5$

$4*3$

$4*3 \Rightarrow 3*5$

( $4*3$ 的值赋给表达式 $3*5$ , 错)

## § 2. 数据类型和表达式

### 2.6. 赋值运算符与赋值表达式

#### 2.6.4. 赋值表达式

##### 2.6.4.3. 赋值表达式的值 (含复合赋值表达式)

★ 变量赋初值时, `int a=b=c=5;`不行

★ 赋值表达式的值可以参与其它表达式的运算

```
int a=12;
a+=a-=a*a;
```

| 步骤                           | 原因                                        | 执行的表达式                       | a的值                 |
|------------------------------|-------------------------------------------|------------------------------|---------------------|
| ① <code>a*a</code>           | * 优先级最高                                   | <code>12*12</code>           | <code>a=12</code>   |
| ② <code>a-=a*a</code>        | <code>+=</code> 、 <code>-=</code> 相同, 右结合 | <code>a-=12*12</code>        | <code>a=12</code>   |
| ③ <code>a=a-a*a</code>       | <code>-=</code> 展开                        | <code>a=12-12*12</code>      | <code>a=-132</code> |
| ④ <code>a+=(a=a-a*a)</code>  | <code>+=</code> 最后                        | <code>a+=-132</code>         | <code>a=-132</code> |
| ⑤ <code>a=a+(a=a-a*a)</code> | <code>+=</code> 展开                        | <code>a=(-132)+(-132)</code> | <code>a=-264</code> |

★ 不讨论相同变量的多个赋值表达式同时运算的情况

例: `cout << (a=10)+(a=20) << endl;`

不同编译器可能20、30、40 (VS2017是40, RedFlag5是30)

## § 2. 数据类型和表达式

### 2.7. 逗号运算符和逗号表达式

#### 2.7.1. 形式

表达式1, 表达式2, ..., 表达式n

★ C++中级别最低的运算符(又称为**顺序求值运算符**)

#### 2.7.2. 逗号表达式的值

顺序求表达式1, 2, ..., n的值, 整个逗号表达式的值为第n个表达式的值

|                                                                                                                                                                 |                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <code>a=3*5, a*4</code><br>式1(赋值表达式): <code>a=15</code><br>式2(算术表达式): <code>15*4=60</code><br>整个逗号表达式的值为60                                                      | <code>b=(a=3*5, a*4)</code><br><code>b = 60</code> (赋值表达式, 将逗号表达式的值赋给b)        |
| <code>(a=3*5, a*4), a+5</code><br>式1(逗号表达式)<br>式1-1(赋值表达式)=15   (a=15)<br>式1-2(算术表达式)=60<br>式1 =60<br>式2(算术表达式)=20<br>整个逗号表达式的值为20                              | <code>b = ((a=3*5, a*4), a+5)</code><br><code>b=20</code> (赋值表达式, 将逗号表达式的值赋给b) |
| <code>int a=5, b=4, c=3;</code><br><code>cout &lt;&lt; a &lt;&lt; b &lt;&lt; c;</code><br><code>cout &lt;&lt; (a, b) &lt;&lt; (a, c) &lt;&lt; (a, b, c);</code> |                                                                                |

543433

## § 2. 数据类型和表达式

### 2.8. 关于C++表达式的总结

★ C++中任意类型的表达式均有值

- ┌ 算术表达式
- ├ 赋值表达式、复合赋值表达式
- └ 逗号表达式

★ 表达式按照优先级/结合性逐步求值(借助栈理解)，运算过程中可能还会涉及到类型转换

★ 表达式类型由最后一个运算决定

b = (a=3\*5, a\*4): 赋值表达式

b = a=3\*5, a\*4 : 逗号表达式