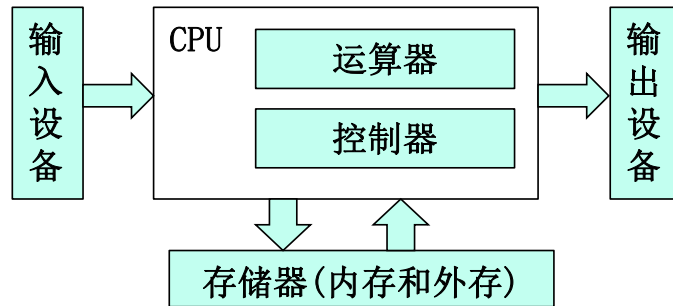




§ 2. 基础知识

2.1 计算机的组成

CPU = Central Processing Unit



- ★ 程序：让CPU(运算器和控制器)工作的一串指令
 - 高级语言：人能理解的程序
 - 机器语言：CPU能理解并执行的程序
- ★ 高级语言程序设计：用高级语言编写程序的一套语法规则



§ 2. 基础知识

2. 2. 二进制

2. 2. 1. 进制的基本概念

★ 进制：按进位原则进行记数的方法叫做**进位记数制**，简称为**进制**

十进制 : 0-9, 逢十进一

二十四进制: 0-23, 逢二十四进一

六十进制 : 0-59, 逢六十进一

=> N进制, 0 ~ N-1, 逢N进一

★ 基数：指各种位制中允许选用基本数码的个数

十进制 : 0-9 => 基数为10

二十四进制: 0-23 => 基数为24

六十进制 : 0-59 => 基数为60

★ 位权：在N进制数中，每个数码所表示的数值等于该数码乘以一个与数码所在位置相关的常数，这个常数叫做位权。

其大小是以基数为底、数码所在位置的序号为指数的整数次幂

$$215 = 2 \times 10^2 + 1 \times 10^1 + 5 \times 10^0$$

2的位权是 10^2 (百位, 基数10为底, 序号为2)

1的位权是 10^1 (十位, 基数10为底, 序号为1)

5的位权是 10^0 (个位, 基数10为底, 序号为0)



§ 2. 基础知识

2. 2. 二进制

2. 2. 2. 二进制的基本概念

★ 二进制：基数为2，只有0、1两个数码，逢二进一

★ 二进制是计算机内部表示数据的方法，因为计算机就其本身来说是一个电器设备，为了能够快速存储/处理/传递信息，其内部采用了大量电子元件；在这些电子元件中，电路的**通和断**、电压的**高与低**，这两种状态最容易实现，也最稳定、也最容易实现对电路本身的控制。我们将计算机所能表示这样的状态，用0、1来表示、即用二进制数表示计算机内部的所有运算和操作

★ 位与字节：每个二进制位只能表示0/1两种状态，当表示较大数据时，所用位数就比较长，为便于管理，每8位称为一个字节
(位：bit 字节：byte)

8 bit = 1 byte

bit ：计算机内表示数据的最小单位

byte ：计算机表示数据的基本单位（数据表示为1-n个byte）



§ 2. 基础知识

2.2.2. 二进制的概念

★ 位与字节：每个二进制位只能表示0/1两种状态，当表示较大数据时，所用位数就比较长，为便于管理，每8位称为一个字节
(位: bit 字节: byte)

8 bit = 1 byte

bit : 计算机内表示数据的最小单位

byte : 计算机表示数据的基本单位 (数据表示为1-n个byte)

● 二进制的大数表示及不同单位的换算如表所示

● 简写的时候, b=bit/B=byte

例：某智能手机，存储空间 128GB
某智能电视，存储空间 128Gb

二进制大数的表示单位：

2^{10}	= 1024	=1 KB	(KiloByte)
2^{20}	= 1024 KB	=1 MB	(MegaByte)
2^{30}	= 1024 MB	=1 GB	(GigaByte)
2^{40}	= 1024 GB	=1 TB	(TeraByte)
2^{50}	= 1024 TB	=1 PB	(PeraByte)
2^{60}	= 1024 PB	=1 EB	(ExaByte)
2^{70}	= 1024 EB	=1 ZB	(ZettaByte)
2^{80}	= 1024 ZB	=1 YB	(YottaByte)
2^{90}	= 1024 YB	=1 BB	(BrontoByte)
2^{100}	= 1024 BB	=1 NB	(NonaByte)
2^{110}	= 1024 NB	=1 DB	(DoggaByte)
2^{120}	= 1024 DB	=1 CB	(CorydonByte)

● 实际表述中，K/M/G既可以表示 $2^{10}/2^{20}/2^{30}$ ，也可以是 $10^3/10^6/10^9$ ，因此部分表述有二义性，折算时有误差，要根据语境理解 (计算机内一般按二进制理解，其余十进制)

例：某程序猿工资：18K = 18000

宽带速率：20Mbps = 20×10^6 (bit per second)

笔记本内存：8GB = 8×2^{30}

硬盘：1TB = 1×10^{12} (厂商标注)

= 1×2^{40} (计算机理解) → 约9%误差



§ 2. 基础知识

2. 2. 二进制

2. 2. 2. 二进制的基本概念

★ 计算机内数据的表示：因为采用二进制，只有两个数码（0/1），任何复杂的数据都是由0/1的基本表示组成的

{ 数字(有数值含义的数字序列)
文本(包括数码，即无数值含义的数字序列)
静态图像
动态视频
声音

课上会有简单的介绍，了解即可；
听不懂也不影响后续学习



§ 2. 基础知识

2.2.3. 二进制与十进制的互相转换

★ 数制转换：计算机内部采用二进制，但用计算机解决实际问题时，数据的输入输出通常使用十进制(人易于理解)。

因此，使用计算机进行数据处理时必须先将十进制转换为二进制，处理完成后再将二进制转换为十进制，这种将数字由一种数制转换成另一种数制的方法称为**数制转换**

★ 十进制转二进制(整数)：除2取余法

基本方法：

- (1) 要转换的整数除以2，得商和余数(整除)
- (2) 继续用商除以2，得新的商和余数(整除)
- (3) 重复(2)直到商为零时为止
- (4) 把所有余数逆序排列，即为转换的二进制数

★ 二进制转十进制(整数)：按权相加法

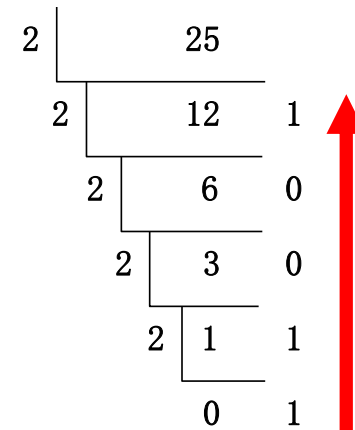
基本方法：

- (1) 把二进制数写成加权系数展开式
- (2) 按十进制加法规则求和

$$\begin{aligned} 11001 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 16 + 8 + 0 + 0 + 1 \\ &= 25 \end{aligned}$$

★ 暂不考虑负数(后续讲)

★ 二进制/十进制小数的相互转换(需自学，作业要求)



$$(25)_{10} = (11001)_2$$



§ 2. 基础知识

2.2.4. 八进制、十六进制

★ 八进制：基数为8，数码为0-7，逢八进一

★ 十六进制：基数为16，数码为0-15，逢十六进一，为避免歧义，用A-F(a-f)替代10-15表示

★ 十进制转八、十六进制：除8/16取余法

8	11418	
8	1427	2
8	178	3
8	22	2
8	2	6
	0	2

$$(11418)_{10} = (26232)_8$$

16	11418	
16	713	10
16	44	9
16	2	12
	0	2

$$(11418)_{10} = (2C9A)_{16}$$

★ 八、十六进制转十进制：按权相加法

$$\begin{aligned}(26232)_8 &= 2 \times 8^4 + 6 \times 8^3 + 2 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 \\ &= 2 \times 4096 + 6 \times 512 + 2 \times 64 + 3 \times 8 + 2 \times 1 \\ &= 11418\end{aligned}$$

$$\begin{aligned}(2C9A)_{16} &= 2 \times 16^3 + 12 \times 16^2 + 9 \times 16^1 + 10 \times 16^0 \\ &= 2 \times 4096 + 12 \times 256 + 9 \times 16 + 10 \times 1 \\ &= 11418\end{aligned}$$



§ 2. 基础知识

2.2.4. 八进制、十六进制

★ 二进制转八、十六进制：低位开始，每3/4位转1位

$$(1011100101011)_2 = 1\ 011\ 100\ 101\ 011 = (13453)_8$$

$$(1011100101011)_2 = 1\ 0111\ 0010\ 1011 = (172B)_{16}$$

★ 八、十六进制转二进制：每1位转3/4位，不足补0

$$(13453)_8 = 001\ 011\ 100\ 101\ 011 = (1011100101011)_2$$

$$(172B)_{16} = 0001\ 0111\ 0010\ 1011 = (1011100101011)_2$$

★ 暂不考虑负数(后续讲)

★ 八进制和十六进制的相互转换（需自学，作业要求）

阅读：下发的附录（A-计数系统）



§ 2. 基础知识

2.3. C++程序简介

2.3.1. 最简单的C++程序

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, World." << endl;
    return 0;
}
```

例1

功能：在屏幕上输出“Hello, World.”

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, sum;
    cin >> a >> b;
    sum=a+b;
    cout << "a+b=" << sum << endl;
    return 0;
}
```

例2

功能：从键盘上输入两个整数(第1行，空格分开)
在屏幕上输出和(第2行)

```
#include <iostream>
using namespace std;
int my_max(int x, int y)
{
    int z;
    if (x>y)
        z=x;
    else
        z=y;
    return z;
}
```

例3

功能：从键盘上输入两个整数(第1行)
在屏幕上输出大的那个(第2行)

```
int main()
{
    int a, b, m;
    cin >> a >> b;
    m = my_max(a, b);
    cout << "max=" << m << '\n';
    return 0;
}
```



§ 2. 基础知识

2. 3. C++程序简介

2. 3. 2. 程序结构的基本形式

包含的头文件

命名空间

暂 常量定义

函数的定义

无 全局变量的定义

函数1

...

...

函数n

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

int main()
{
    int a,b,sum;
    cin >> a >> b;
    sum=a+b;
    cout<<"a+b="<<sum<< endl;
    return 0;
}
```

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

例 3

```
int my_max(int x, int y)
{
    int z;
    if (x>y)
        z=x;
    else
        z=y;
    return (z);
}
```

函数my_max

```
int main()
{
    int a,b,m;
    cin >> a >> b;
    m = my_max(a, b);
    cout<<"max="<<m<<' \n' ;
    return 0;
}
```

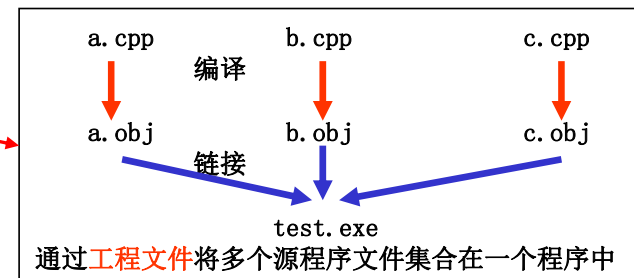
函数main

★ C++程序由函数组成，函数是C++程序的基本单位

★ 一个C++程序可由若干源程序文件 (*.cpp) 组成，每个源程序文件可以包含若干函数

★ 有且仅有一个名为main()的函数，称为主函数，程序的执行从它开始

★ C++提供许多库函数 (已做好的)





§ 2. 基础知识

2.3. C++程序简介

2.3.3. 函数的组成

函数返回类型 函数名（形式参数表）

```
{  
}
```

函数体（局部变量的定义、函数体的可执行部分）

```
int my_max(int x, int y)  
{  
    int z;  
    if (x>y) z=x;  
    else z=y;  
    return (z);  
}
```

```
#include <iostream>  
using namespace std;  
int my_max(int x, int y)  
{ ...  
}  
int main()  
{ ...  
}
```

```
#include <iostream>  
using namespace std;  
  
//此处需补函数my_max的声明(暂略)  
int main()  
{ ...  
}  
int my_max(int x, int y)  
{ ...  
}
```

★ 从main()开始执行，函数相互间的位置不影响程序的正确性



§ 2. 基础知识

2.3. C++程序简介

2.3.3. 函数的组成

★ 从main()开始执行，函数相互间的位置不影响程序的正确性

★ 函数平行定义，嵌套调用

★ 函数由语句组成，一个语句以;结尾（必须有），语句分为定义语句和执行语句，定义语句用于声明某些信息，执行语句用于完成特定的操作

★ 书写格式自由，可以一行多个语句，也可以一个语句多行（以\表示分行）

例3

```
#include <iostream>
using namespace std;
int my_max(int x, int y)
{
    int z;
    if (x>y) z=x;
    else z=y;
    return (z);
}

int main()
{
    int a,b,m;
    cin >> a >> b;
    m = my_max(a,b);
    cout<<"max="<<m<<' \n';
    return 0;
}
```

<pre>#include <iostream> using namespace std; int main() { cout << "Hello, World." << endl; return 0; }</pre>	<pre>#include <iostream> using namespace std; int main() { cout << "Hello, World." << endl; return 0;}</pre>
<pre>#include <iostream> using namespace std; int main() { cout \ << \ "Hello, World." \ << endl; return 0; }</pre>	<pre>#include <iostream> using namespace std; int main() { cout << "Hello, World." << endl; return 0; }</pre>

同一个程序，这些格式
编译器都认为是正确的



§ 2. 基础知识

2.3. C++程序简介

2.3.3. 函数的组成

★ 书写格式自由，可以一行多个语句，也可以一个语句多行（以\表示分行）

- 不同书写格式，编译器都正确，但对人的阅读而言，友好程度不同

- 建议：一句一行（本课程强制要求：一句一行）

=> 违反格式规定的，本题分数扣20%

<pre>#include <iostream> using namespace std; int main() { cout << "Hello, World." << endl; return 0; }</pre> ✓	<pre>#include <iostream> using namespace std; int main() { cout << "Hello, World." << endl;return 0;}</pre> ✗	<pre>int max(int x, int y) { int z; if (x>y) z=x; else z=y; return z; }</pre> ✓	<pre>int max(int x, int y) { int z; if (x>y) z=x; else z=y; return z; }</pre> ✗
<pre>#include <iostream> using namespace std; int main() { cout \ << \ "Hello, World." \ << endl; return 0; }</pre> ✗	<pre>#include <iostream> using namespace std; int main() { cout << "Hello, World." << endl; return 0; }</pre> ✗	<pre>int max(int x, int y) { return 0; }</pre> ✓	<pre>int max(int x, int y) { return 0; }</pre> ✗



§ 2. 基础知识

2.3. C++程序简介

2.3.3. 函数的组成

★ 可以用/* ... */(**多行**)或 //...(**单行**)两种方式加入注释, 注释中的内容是为了增加程序的可读性, 因此不需要符合C++的语法及规定

```
#include <iostream>
using namespace std;    //命名空间

int main()              /* function main */
{
    cout << "Hello, World." << endl; //Ausgabe(德文-输出)
    return 0;            /* fanhui :-) ^-^ */
}
```

```
#include <iostream>
using namespace std;    //命名空间
/* 下面是主函数, 仅用于输出一个字符串,
   输出后程序即运行结束 */

int main()              /* function main */
{
    cout << "Hello, World." << endl;
    return 0;           /* fanhui :-) ^-^ */
}
```

★ 系统提供的库函数和自己编写的函数调用方法相同



§ 2. 基础知识

2. 4. 编译器的安装与使用、编程的基本步骤

2. 4. 1. VS2022的安装与基本使用

2. 4. 2. DevC++的安装与基本使用

2. 4. 3. 编程的基本过程

2. 4. 4. 本课程作业的格式要求

文档已下发
认真阅读



§ 2. 基础知识

2.5. C++的数据类型

2.5.1. 整型数的符号位

问题：如果用2字节（16bit）表示一个整数，则数据范围为多少？

$$00000000\ 00000000 = 0$$

$$11111111\ 11111111 = 2^{16}-1 = 65535$$

答：数据范围为0-65535

=> 这种表示方法不考虑负数，称为**无符号数**

续问：如何能同时表示正负数（称为**有符号数**），数据范围是多少？

前提：计算机内任何数据都表示为0/1，包括正负号，假设0/1表示+/-

方案1：单独用额外的一个bit表示+/-，再加16bit数据

例：0 00000000 00000001 = +1 (max = +65535)

1 00000000 00000001 = -1 (min = -65535)

=> 无法用两字节表示正负数（**放弃**）

方案2：最高(最左)bit位表示+/-，再加15bit数据

例：0 00000000 00000001 = +1 (max = +32767)

1 00000000 00000001 = -1 (min = -32767)

=> 可用两字节表示正负数，但数据范围减半（**选**）



§ 2. 基础知识

2.5. C++的数据类型

2.5.1. 整型数的符号位

★ 数据的二进制原码

将表示一个整型数的若干字节的最高bit位(最左)的0/1理解为正负号, (该bit位称为**符号位**), 后续的所有bit理解为数值的方式称为数据的二进制原码表示

00000000 00000001 = +1

01111111 11111111 = +32767

10000000 00000001 = -1

11111111 11111111 = -32767

结论: 原码不适合在计算机内表示整数

● 原码的缺陷1: +0与-0的二义性问题

00000000 00000000 = + 0

10000000 00000000 = - 0

计算机思维方式的引申:

计算机内部如何判断两个数据相等?

方法1: 对应字节完全一致(选)

=> 缺陷1需要解决

方法2: 某些数据做特判(放弃)

● 原码的缺陷2: 含负数的情况下无法正常运算

仅含正数运算: 1 + 1

二进制	十进制
00000000 00000001	1
+) 1	+) 1

00000000 00000010 (+2)	2

含负数运算: -1 + 1

二进制	十进制
10000000 00000001	-1
+) 1	+) 1

10000000 00000010 (-2)	0



§ 2. 基础知识

2.5. C++的数据类型

2.5.2. 补码的基本概念

★ 补码：解决 ± 0 的二义性及含负数运算的问题，是计算机内整型数值的真正表示方法

- 正数与原码相同
- 负数：绝对值的原码取反+1
 - Step1: 求绝对值的原码
 - Step2: 绝对值的原码取反
 - Step3: 取反后再+1



例：以2字节（16bit）整数为例

数值	绝对值的二进制表示	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	1111111111111110 +) 1 ----- 111111111111111
-32768	1000000000000000 （绝对值当无符号数，1表示 2^{15} ）	1000000000000000	0111111111111111 +) 1 ----- 1000000000000000
-32767	0111111111111111	0111111111111111	1000000000000000 +) 1 ----- 1000000000000001

原码方式的 -32767

原码方式的 -0（补码当有符号数）

原码方式的 -1

形式一样，理解不同

问题1：计算机内的整数，正数用原码表示，负数用补码表示，该说法是否正确？

问题2：现在看到一个以1开始的二进制整数(1***)，应该是无符号的正数，还是有符号的负数？

例：以2字节（16bit）整数为例

	原码理解	补码理解
0 0000000 00000000	+0	+0
0 1111111 11111111	+32767	+32767
=====		
	原码理解	补码理解
1 0000000 00000000	-0	-32768
1 0000000 00000001	-1	-32767
1 0000000 00000010	-2	-32766
...		
1 1111111 11111101	-32765	-3
1 1111111 11111110	-32766	-2
1 1111111 11111111	-32767	-1

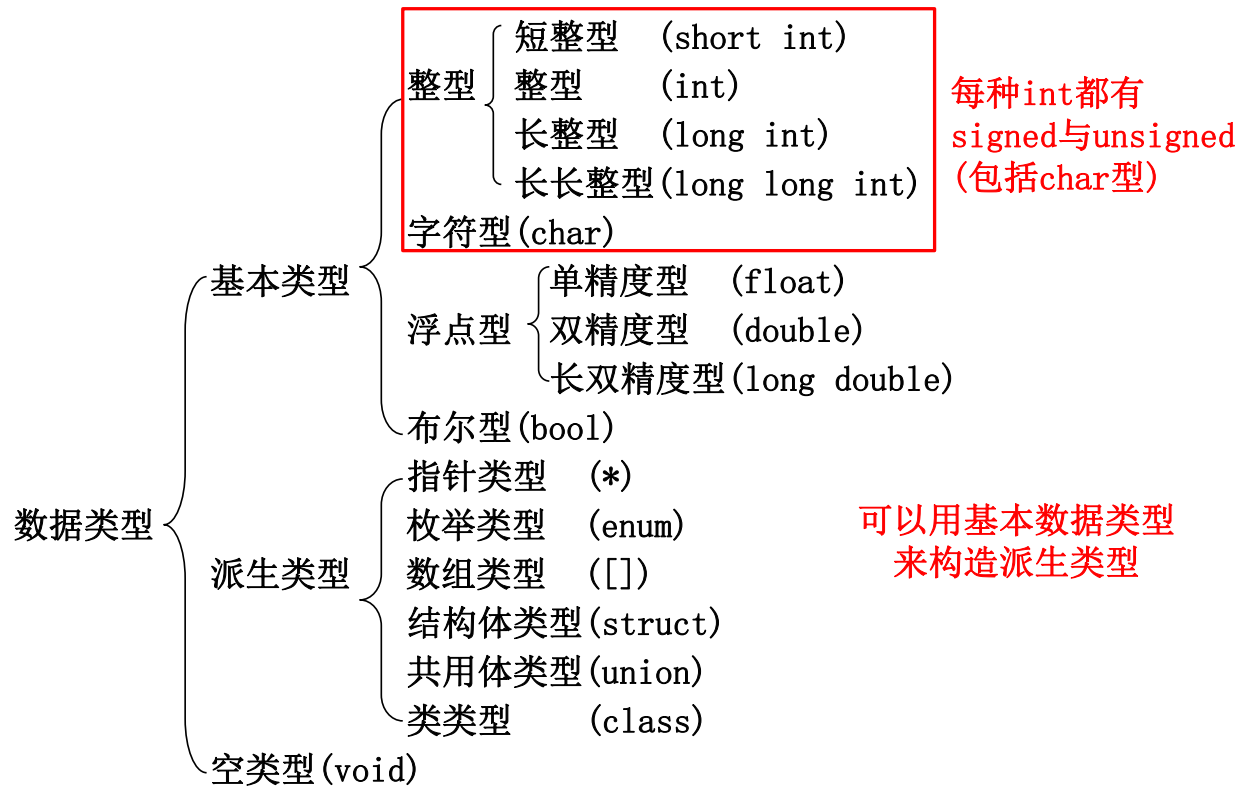
=> 2字节（16bit）的整数范围为 $-32768 \sim +32767$
 $-2^{15} \sim +2^{15}-1$



§ 2. 基础知识

2. 5. C++的数据类型

2. 5. 3. 数据类型分类





§ 2. 基础知识

2. 5. C++的数据类型

2. 5. 4. 各种数据类型所占字节及表示范围(以VS2022 x86/32bit为基准)

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	$0 \sim +2^{32}-1$
长长整型	long long [int]	8	-9223372036854775808 ~ 9223372036854775807	$-2^{63} \sim +2^{63}-1$
无符号长长整型	unsigned long long [int]	8	0 ~ 18446744073709551615	$0 \sim +2^{64}-1$
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	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$

[]表示可省略
例: signed int } 等价
int

long } 等价
long int }



§ 2. 基础知识

2.5.C++的数据类型

2.5.4. 各种数据类型所占字节及表示范围(以VS2022 x86/32bit为基准)

★ 如何确定数据类型所占的字节: `sizeof(类型)`

```
#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(unsigned long long) << endl;
    cout << sizeof(float) << endl;
    cout << sizeof(long double) << endl;

    return 0;
}
```

用VS2022的32位编译器

64位编译器

DevC++的32位编译器

64位编译器

分别编译运行并观察结果

注意: VS2022和DevC++如何
切换32位和64位编译器?



§ 2. 基础知识

2.5. C++的数据类型

2.5.4. 各种数据类型所占字节及表示范围 (以VS2022 x86/32bit为基准)

★ 如何确定数据类型的上下限:

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

int main()
{
    cout << INT_MIN    << endl;
    cout << UINT_MAX   << endl;
    cout << LLONG_MAX  << endl;
    cout << SHRT_MAX   << endl;

    return 0;
}
```

方法1: 根据sizeof的结果自行推算

方法2: 打印系统预定义的值

- 可能需要包含climits头文件
(目前所用双编译器均不需要)
- 预置定义见P.36 表3.1

趣味思考题(学到P.26后再想):

前提条件:

- 1、不准用INT_MIN等预置值(或不知道)
- 2、知道sizeof()
- 3、不准用其它工具(书/网络/计算器)

问题: 想知道long long型(或其它)
数据的最大值, 怎么办?



§ 2. 基础知识

2.5. C++的数据类型

2.5.4. 各种数据类型所占字节及表示范围(以VS2022 x86/32bit为基准)

★ 对于整型数(含char), 均有signed及unsigned的区别, 缺省为signed

★ 不同的编译系统中, 不同数据类型的所占字节/表示范围可能不同

● 本课程中若不加以特别说明, 均认为:

short : 2字节

int/long : 4字节

long long : 8字节

long double : 8字节

● 所有C/C++编译器中, 不同整型数据的相互关系遵循的基本准则

以int型数据为基准大小, short不大于int, long不小于int, longlong不小于long

★ 因为数据必须占用一定字节, 因此计算机不可能表示数学中的无穷概念

(可以有特殊定义表示无穷大, 但无法体现增量)

★ 对于整型数, 存储为二进制数形式, 占满对应字节长度

例: 85(十进制) = 1010101(二进制)

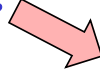
则: int型 : 00000000 00000000 00000000 01010101

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



例:	short	int	long	long long
Turbo C (1987年)	2	2	4	无
VS2022	2	4	4	8
RockyLinux	2	4	8	8



§ 2. 基础知识

2.6. 常量

2.6.1. 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

2.6.2. 数值常量

2.6.2.1. 整型常量

整型常量在C/C++源程序中的四种表示方法：

- 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	请把下列三个数从大到小排列： (1) 123 (2) 0123 (3) 0x123 ?
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		



§ 2. 基础知识

2.6. 常量

2.6.2. 数值常量

2.6.2.1. 整型常量

整型常量在C/C++源程序中的四种表示方法：

- 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

内部表示
输出显示
额外修饰 } 注意区分

```
#include <iostream>
using namespace std;
int main()
{
    cout << 119 << endl;
    cout << 0b1110111 << endl;
    cout << 0167 << endl;
    cout << 0x77 << endl;

    return 0;
}
```

输出？

```
#include <iostream>
using namespace std;
int main()
{
    cout << hex << 119 << endl;
    cout << hex << 0b1110111 << endl;
    cout << hex << 0167 << endl;
    cout << hex << 0x77 << endl;

    return 0;
}
```

输出？

```
#include <iostream>
using namespace std;
int main()
{
    cout << dec << 119 << endl;
    cout << "0x" << hex << 0b1110111 << endl;
    cout << "(" << oct << 0167 << ")8" << endl;

    return 0;
}
```

输出？

三个例题的结论：

- ★ 无论在源程序表示为何种进制，在**内存中**只有二进制补码一种
- ★ 无论在源程序表示为何种进制，和输出无关，输出缺省为十进制，可加**前导进制转换**(hex/oct/dec)改为其它进制
- ★ 输出只负责最基本的内容(**可能引发二义性**)，其余需要自行按需添加



§ 2. 基础知识

2.6. 常量

2.6.2. 数值常量

2.6.2.1. 整型常量

整型常量在C/C++源程序中的四种表示方法：

- 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

★ 无直接的二进制输出方法

★ 老版本编译器无源程序中的二进制表示方法

★ 二进制方式表示不方便，今后不再讨论 (以后也可能会说三种表示方法)

★ 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:

?

另：L不建议小写，为什么？

问题：这几种在vs下都是4字节，如何证明？

答：C++提供typeid运算符，基本用法为typeid(数据).name

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

```
int main()
{
    cout << typeid(123).name() << endl;
    cout << typeid(123L).name() << endl;
    cout << typeid(123U).name() << endl;
    cout << typeid(123UL).name() << endl;
    cout << typeid(123LU).name() << endl;

    return 0;
}
```



§ 2. 基础知识

2.6. 常量

2.6.2. 数值常量

2.6.2.1. 整型常量

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

int main()
{
    cout << typeid(bool).name() << endl;
    cout << typeid(char).name() << endl;
    cout << typeid(unsigned char).name() << endl;
    cout << typeid(signed short).name() << endl;
    cout << typeid(unsigned short).name() << endl;
    cout << typeid(int).name() << endl;
    cout << typeid(unsigned).name() << endl;
    cout << typeid(unsigned long).name() << endl;
    cout << typeid(long long).name() << endl;
    cout << typeid(unsigned long long).name() << endl;
    cout << typeid(float).name() << endl;
    cout << typeid(double).name() << endl;
    cout << typeid(long double).name() << endl;
    return 0;
}
```

VS打印完整信息，Dev打印
单字母，后续不再解释Dev，
有兴趣自行研究即可

VS

```
Microsoft Visual Studio 调试控制台
bool
char
unsigned char
short
unsigned short
int
unsigned int
unsigned long
__int64
unsigned __int64
float
double
long double
```

Dev

```
D:\Wo
b
c
h
s
t
i
j
m
x
y
f
d
e
```



§ 2. 基础知识

2.6. 常量

2.6.2. 数值常量

2.6.2.2. 浮点型常量

★ 只有十进制这一种进制，但可以表示为两种形式：

{ 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	✗



§ 2. 基础知识

2.6. 常量

2.6.2. 数值常量

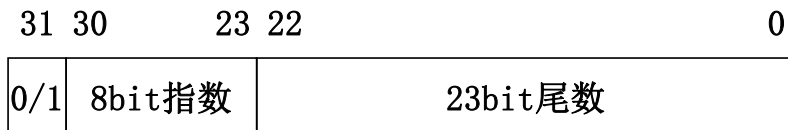
2.6.2.2. 浮点型常量

★ 只有十进制这一种进制，但可以表示为两种形式：

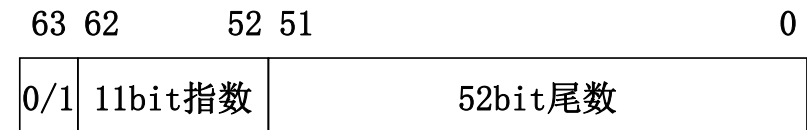
- 十进制数（带小数点的数字）
- 指数形式（科学记数法）

★ 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

float单精度：



double双精度：



- long double 的具体位数分配暂忽略，原理相同
- 浮点数的存储遵从 IEEE 754 规范
(具体如何存储通过作业完成并理解[专业文档的阅读能力培养])



§ 2. 基础知识

2.6. 常量

2.6.2. 数值常量

2.6.2.2. 浮点型常量

★ 只有十进制这一种进制，但可以表示为两种形式：

- 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 0.1234567890123456e11

例2：1.0/3*3

不同编译系统，可能是0.999999或1（本课程的双系统都是1）

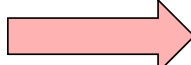
● 后续通过作业理解

★ 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 << typeid(1.23).name() << endl;
cout << sizeof(1.23F) << endl;
cout << typeid(1.23F).name() << endl;
```



§ 2. 基础知识

2.6. 常量

2.6.3. 字符常量与字符串常量

2.6.3.1. ASCII码

★ P.845 附录C ASCII码表

★ ASCII码占用一个字节，共可表示256个字符

0XXXXXX : 基本ASCII码 128个(0-127)

1XXXXXX : 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 - 32/0x20

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. 基础知识

2.6. 常量

2.6.3. 字符常量与字符串常量

2.6.3.2. 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. 43 表3.2

在表格后加: '\ddd' : 000-377 (0-255) : 8进制值对应的ASCII码

'\xhh' : 00-ff/FF (0-255) : 16进制值对应的ASCII码

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;
}
```

VS2022/DevC++: 分别提示什么信息?

体会编译器的差异



§ 2. 基础知识

2.6. 常量

2.6.3. 字符常量与字符串常量

2.6.3.2. 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' '\101' '\x41' ('\X41' 错!!!)	ESC (ASCII=27)	'\33' '\033' '\x1b' '\x1B'
换行 (ASCII=10)	'\n' '\12' '\012' '\xA' '\x0A' '\xa' '\x0a'	双引号 (ASCII=34)	'\"' '\42' '\042' '\x22'

★ '0' 与 '\0' 的区别 (非常重要!!!)

'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



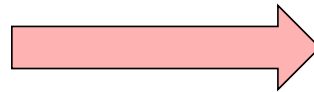
§ 2. 基础知识

2.6. 常量

2.6.3. 字符常量与字符串常量

2.6.3.3. 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



```
#include <iostream>
using namespace std;
int main()
{
    cout << 'A' << endl;
    cout << sizeof('A') << endl;
    cout << typeid('A').name() << endl;
    //打印A的ASCII码值的方法(强制类型转换, 后面会专题讲到)
    cout << int('A') << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
A
1
char
65
```

=> 推论: 汉字不能表示为字符常量形式

- 为什么?
- 样例的结论不再讨论, 初学者不建议深究

```
#include <iostream>
using namespace std;
int main()
{
    cout << '汉' << endl;
    cout << sizeof('汉') << endl;
    cout << typeid('汉').name() << endl;
    return 0;
}
```

输出

显示输出来源(S): 生成

已启动生成...

1> 已启动生成: 项目: demo-cpp, 配置: Debug Win32

1>demo.cpp

1>demo-cpp.vcxproj -> D:\Workspace\VS2022-demo\Debug\demo-cpp.exe

生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个

Microsoft Visual Studio 调试控制台

```
47802
4
int
```

VS : 0 error, 0 warning
Dev: 0 error, 3 warning

行	列	单元	信息
5	13	D:\Workspace\VS2022-demo\demo-cpp\demo.cpp	[Warning] multi-character character constant [-Wmultichar]
6	20	D:\Workspace\VS2022-demo\demo-cpp\demo.cpp	[Warning] multi-character character constant [-Wmultichar]
7	20	D:\Workspace\VS2022-demo\demo-cpp\demo.cpp	[Warning] multi-character character constant [-Wmultichar]

D:\Workspace\

```
47802
4
i
```



§ 2. 基础知识

2.6. 常量

2.6.3. 字符常量与字符串常量

2.6.3.4. 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" = ?

"\r\n\t\\A\\t\x1b"\9234\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

思考：（课上未讲，也不再深入讨论，有兴趣可自行查找资料）

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



§ 2. 基础知识

2.6. 常量

2.6.3. 字符常量与字符串常量

2.6.3.4. 字符串常量

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

\0	
32	\0

★ '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

65	
65	\0

★ 暂不讨论字符串中含尾0的情况 (后续模块再讨论)

"Hello\0ABC"

★ 字符串常量方式可表示汉字

```
#include <iostream>
using namespace std;
int main()
{
    cout << "中" << endl;
    cout << sizeof("中") << endl;

    cout << "同济" << endl;
    cout << sizeof("同济") << endl;

    return 0;
}
```

Microsoft Visual
中
3
同济
5

```
#include <iostream>
using namespace std;
int main()
{
    cout << sizeof("") << endl;
    cout << strlen("") << endl;
    cout << sizeof(" ") << endl;
    cout << strlen(" ") << endl;

    cout << sizeof('A') << endl;
    cout << strlen('A') << endl;
    cout << sizeof("A") << endl;
    cout << strlen("A") << endl;

    return 0;
}
```

问题

- 1、上述8个中哪句编译错？
- 2、其余正确的语句，输出是什么？
- 3、sizeof和strlen差异？



§ 2. 基础知识

2.6. 常量

2.6.4. 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

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

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？

在VS2022下编译运行下面的代码，
观察结果

```
#define 喵喵喵 main
#define 喵喵 int
#define 喵 (
#define 喵喵呜 )
#define 喵喵喵喵 {
#define 喵呜喵 }
#define 呜喵喵 <<
#define 呜呜 cout
#define 呜呜 endl
#define 呜呜呜 "喵喵喵!"
#define 呜呜呜呜 return
#define 呜喵 0
#define 喵呜 ;
#include <iostream>
using namespace std;
喵喵 喵喵喵 喵 喵喵呜
喵喵喵喵
呜 呜喵喵 呜呜呜 呜喵喵 呜呜
喵喵
呜呜呜呜 呜喵 喵呜
喵喵喵
```



§ 2. 基础知识

2.7. 变量

2.7.1. 基本概念

变量：在程序运行中，值能够改变的量（有名字、值）

问：为什么要用变量？

=> 最开始的例2，从键盘输入两个整数到a、b中（`cin >> a >> b;`），因为每次输入不同，a、b必须是变量

2.7.2. 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，由字母、数字、下划线组成

合理： A a al _a _al al_b abc （前例：`#define pi 3.14159` 中，pi 也是标识符）

不合理： la a-b

★ 标识符区分大小写（也称为大小写敏感）

★ 长度<=32（或其它值，每种编译器还有参数可设，不再讨论）

★ 取名时通常按变量的含义

★ 必须先定义、后使用（某些语言不定义可直接使用，即如果第一次使用不存在，会自动创建）

★ 同级不能同名（不同级的同名问题后续讨论）

★ 标识符不能与关键字（int/float等）同名

例：int float; ✗

各有利弊，遵循各语言的规则即可：

例：定义时 cno1(字母1)
使用时 cno1(数字1)



§ 2. 基础知识

2.7. 变量

2.7.3. 定义变量

数据类型 标识符名, 标识符名, ...;

多个变量间用逗号分隔, 最后有分号

```
int a;  
unsigned int b, c;  
long e, f;  
short i, j;  
float f1, f2;  
double d1, d2;  
char c1;
```

定义一个变量后的多重含义:

- 1、变量名为_____
- 2、存放一个_____类型的数据
(隐含含义: 数值范围从____到____之间)
- 3、在内存中占用从_____开始的_____个字节

Microsoft Visual Studio 调试控制台
4
0077FCC0

定义一个变量后的多重含义:

- 1、变量名为 a
- 2、存放一个 int (有符号4字节整型) 类型的数据
(隐含含义: 数值范围从 -2147483648 到 +2147483647 之间)
- 3、在内存中占用从 0x0077FCC0 开始的 4 个字节 (0x0077FCC0~FCC3)

```
demo.cpp  demo-cpp  (全局范围)  
1  #include <iostream>  
2  using namespace std;  
3  int main()  
4  {  
5      int a;  
6      cout << sizeof(a) << endl;  
7      cout << &a << endl;  
8      return 0;  
9  }
```

&a = a在内存中的起始地址
(默认为16进制表示)



§ 2. 基础知识

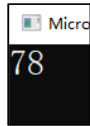
2.7. 变量

2.7.3. 定义变量

★ VS2022允许用中文做变量名，通用性差，不建议(Dev C++编译报错)

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

int main()
{
    int 分数 = 78;
    cout << 分数 << endl;
    return 0;
}
```



★ C++11 标准支持auto自动定义类型，由初值决定类型，易错，
对使用者的基本概念清晰程度要求很高，初学者不建议(本课程禁用)

```
#include <iostream>
using namespace std;
int main()
{
    auto x1 = 12;
    auto x2 = 12U;
    cout << x1 - 13 << endl;
    cout << x2 - 13 << endl;

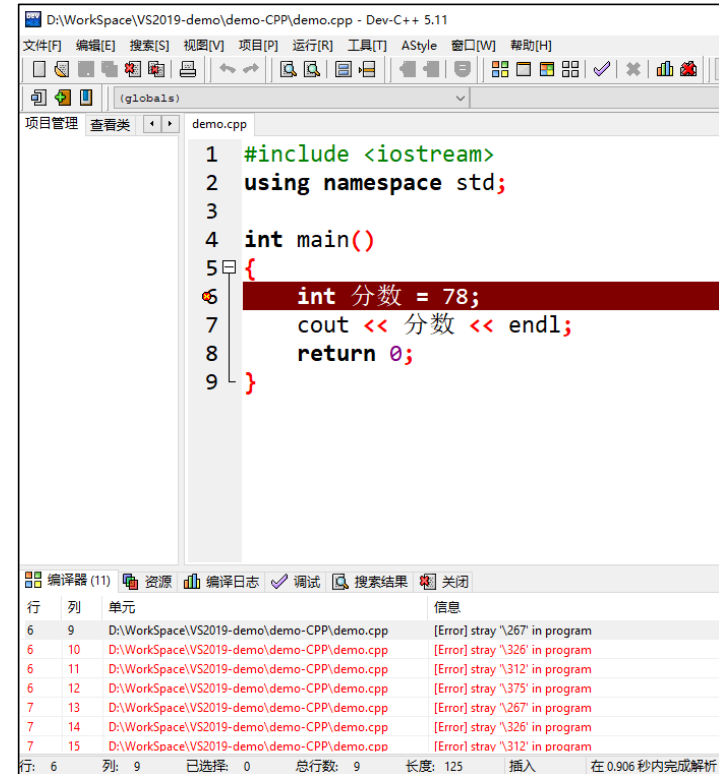
    auto y = 1.2F;
    auto z = 1.2;
    cout << sizeof(y) << endl;
    cout << sizeof(z) << endl;
    return 0;
}
```

本课程双编译器：
直接支持

某些编译器：

缺省设置编译出错，加-std=c++11后正确

● 具体方法不做要求，了解即可





§ 2. 基础知识

2.7. 变量

2.7.4. 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)

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

int main()
{
    int a = b = c = 10;
    return 0;
}
```

```
error C2065: "b": 未声明的标识符
error C2065: "c": 未声明的标识符
```

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

int main()
{
    int c, b, a = b = c = 10;
    return 0;
}
```

编译器思维培养:

- 1、左侧报什么编译错误?
- 2、右侧为什么是正确的?
- 3、编译器的检查流程是怎样的?



§ 2. 基础知识

2.7. 变量

2.7.4. 对变量赋初值

★ 若变量定义后未赋值即访问，不同编译器表现不同

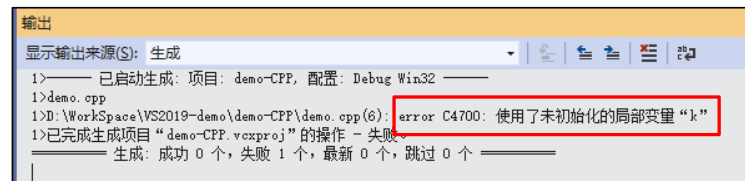
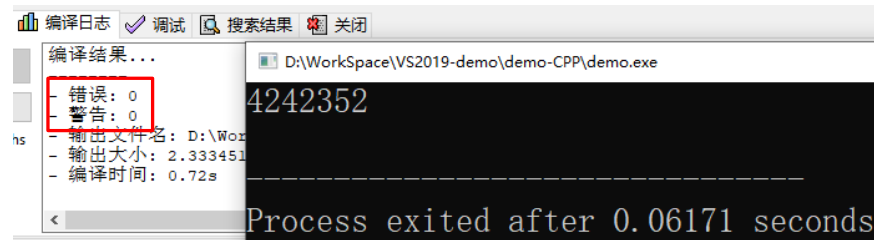
VS : error

Dev C++: 不报错，输出不可预知值

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

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

int main()
{
    int k;
    cout << k << endl;
    k=10;
    return 0;
}
```





§ 2. 基础知识

2.7. 变量

2.7.4. 对变量赋初值

例：在VS和DevC++中分别运行这几个程序，观察error及warning信息

```
#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;
}
```

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

```
#include <iostream>
#include <cmath> //数学函数对应头文件，VS可省略
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;
}
```

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

```
#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;
}
```

```
#include <iostream>
#include <cmath> //数学函数对应头文件，VS可省略
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;
}
```

只有提交这个版本，
得分才能超过50% !!!

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VS为准，完成多编译器作业时，
要有能力去解决差异



§ 2. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.1. 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

例：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
+) 0000000000000001
-----
b= 1000000000000000 = -32768
```

含计算过程

注：学过2.9.5 转换
优先级后，再来
思考本例忽略了
什么步骤!!!

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

计算过程略

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

计算过程略

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

计算过程略

```
#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= 1111111111110110 = -10  
b= 1111111111110110 = 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;  
}
```



```
//P.23 趣味思考题的解法  
#include <iostream>  
#include <climits>  
using namespace std;  
int main()  
{  
    //学完P.26的解法  
    cout << 0xFFFFFFFFFFFFFFFF << endl;  
  
    //学完本页知识点后的解法  
    unsigned long long x = -1LL;  
    cout << x/2 << endl;  
    return 0;  
}
```

P.23 趣味思考题的两种解法
要想明白为什么!!!



★ 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 0 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 0 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 1 1111111111110110 =-10
```

```
b= 11111111111111111111111111110110 =-10
```

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

```
a= 填充符号位 1 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;  
}
```

?

绝对值之和=?
为什么?



★ 长=>短：低位赋值，高位丢弃

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

[illegible]

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

```
unsigned long a=4294967286; unsigned short b=a;
a= 111111111111111111111111111111110110 =4294967286
b=                111111111111111111110110 =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. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.2. 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

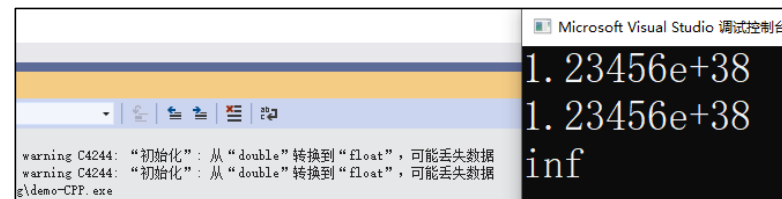
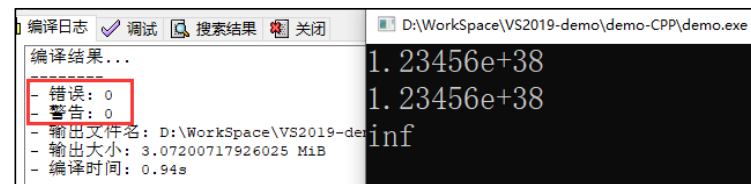
★ 浮点常量缺省为double型，如需表示为float型，可以在后面加f(F)

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，VS2022会出现 **inf** (infinity) 的形式(有warning)，具体原因不作要求

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

int main()
{
    double d=1.23456e38;
    float f1=d;
    float f2=d*10;
    cout << d << endl;
    cout << f1 << endl;
    cout << f2 << endl;
    return 0;
}
```





§ 2. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.3. 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. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.3. 字符型变量的使用

★ 注意变量与常量的区别

★ 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, b, c, d, e;
```

```
    a=65;
    b=0b1000001;    右侧为各种数制
    c=0101;          的int型常量
    d=0x41;
    e=0X41;
```

```
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    cout << d << endl;
    cout << e << endl;
    return 0;
```

```
}
```

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

```
    char a, b, c;
```

```
    a='A';           右侧为各种形式
    b='\101';         的char型常量
    c='\x41'; //不可以 '\X41'
```

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

```
}
```

结果虽相同，但过程不同

a=65 : 4字节赋值给1字节，丢弃24bit 0

a='A' : 1字节赋值给1字节

输出形式：字符(cout根据输出变量类型决定)



§ 2. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.3. 字符型变量的使用

★ 注意变量与常量的区别

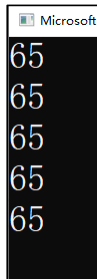
★ 与整数的互通性：可当作1字节的整数参与运算(signed/unsigned)

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

```
    int a, b, c, d, e;
    a=65;
    b=0b1000001;
    c=0101;
    d=0x41;
    e=0X41;
```

```
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    cout << d << endl;
    cout << e << endl;
    return 0;
```

```
}
```



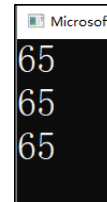
```
Microsoft
65
65
65
65
65
```

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

```
    int a, b, c;
    a='A';
    b='\101';
    c='\x41'; //不可以 '\X41'
```

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

```
}
```



```
Microsoft
65
65
65
```

结果虽相同，但过程不同

a=65 : 4字节赋值给4字节

a='A' : 1字节赋值给4字节，补高24bit

输出形式：数字(cout根据输出变量类型决定)



§ 2. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.3. 字符型变量的使用

★ 注意变量与常量的区别

★ 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 c1, c2;
```

```
    c1 = 'a';
```

```
    c2 = 'b';
```

```
    c1 = c1-32;
```

```
    c2 = c2-32;
```

```
    cout << c1 << ' ' << c2 << endl;
```

```
    return 0;
```

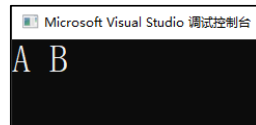
```
}
```

换为int，输出什么？

1、__字节常量赋值给__变量

2、__字节变量 - __常量 (不同字节相减规则待定)，再赋值给__字节变量

3、输出为__形式(字符/数字)





§ 2. 基础知识

2.7. 变量

2.7.5. 各种类型变量的使用 (归纳+补充+重点+难点)

2.7.5.4. 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（后续再讨论）



§ 2. 基础知识

2.7. 变量

2.7.6. 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；

数据类型 const 变量名=初值；

例：const int a=10;
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;
}
```

error C2734: “a”：如果不是外部的，则必须初始化常量对象

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

```
int main()
{
    const int a=10;
    a=15;

    return 0;
}
```

error C3892: “a”：不能给常量赋值



§ 2. 基础知识

2.7. 变量

2.7.6. 常变量

★ 常变量与符号常量使用方法相似，但本质有区别（具体不做要求，推荐使用常变量）

┌ 常变量：有类型、有空间、有初始值（除了值不可变，其余同变量）
└ 符号常量：一个标识符替代一串字符

```
#include <iostream>
using namespace std;
#define pi 3.14159 //①无分号
int main()
{
    double r=3, s, v;
    s = pi*r*r; //②
    v = pi*r*r*r*4/3; //③
    cout << s << endl;
    cout << v << endl;
    return 0;
}
```

- 1、将①改为#define pi 3.14159;（有分号）报什么错？
- 2、在①有分号的基础上，②改为s=r*r*pi;
③改为v=r*r*r*4/3*pi;（均有分号），
能否正确？如何解释？
- 3、在①有分号的基础上，②改为s=r*r*pi
③改为v=r*r*r*4/3*pi（均无分号），
能否正确？如何解释？

```
#include <iostream>
using namespace std;
int main()
{
    const double pi=3.14159; //①有分号
    double r=3, s, v;
    s = pi*r*r; //②
    v = pi*r*r*r*4/3; //③
    cout << s << endl;
    cout << v << endl;
    return 0;
}
```

- 1、将①改为const double pi=3.14159（无分号），报什么错？
- 2、在①无分号的基础上，②改为s=r*r*pi
③改为v=r*r*r*4/3*pi（均无分号），
报什么错？

回忆课上所讲，自行测试



§ 2. 基础知识

2.8. C++的运算符

2.8.1. C++的运算符的种类

总：下发的附录（D-运算符优先级）

★ 运算符根据参与运算的操作数的个数，分为一元/二元/三元运算符，也称为单目/双目/三目运算符

★ 唯一的三目运算符是第15组的“:?”

★ 勘误 => 附录D P. 850 优先级第5组

*

/

% （书上印为^）

2.8.2. 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(L-R)：从左至右进行同级运算符的混合运算

右结合(R_L)：从右至左进行同级运算符的混合运算

附录D（1最高 18最低 熟记!!!）

=> 随着学习过程慢慢加入新的运算符



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.1. 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

'A' * 10 => 65 * 10 => 650

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

Microsoft Visual

650

★ %的使用(求模，整数相除求余数)，%两侧为整型

(char, short, int, long, long long及对应的unsigned)

10 % 3 = 1 10 % 7 = 3 7 % 3 = 1 3 % 7 = 3	10 % -3 = 1 10 % -7 = 3 7 % -3 = 1 3 % -7 = 3	10 % 3 = -1 10 % 7 = -3 7 % 3 = -1 3 % 7 = -3	10 % -3 = -1 10 % -7 = -3 7 % -3 = -1 3 % -7 = -3
--	--	--	--

★ 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;
}
```

Microsoft Visual Studio 调试控制台

0
84.8229

实际 期望
0 47.1238
84.8229 113.097
4/3更有欺骗性!!!



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.2. 算术运算符的优先级与结合性

优先级：* / % 5级
+ - 6级 都是双目运算符

结合性：左结合 (L-R)

2.9.3. 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. 基础知识

2.9. 算术运算符与算术表达式

2.9.4. 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. 基础知识

2.9. 算术运算符与算术表达式

2.9.4. 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

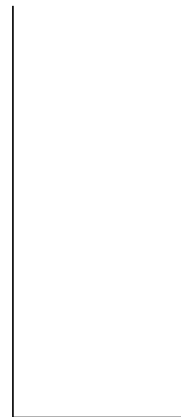
(2) 若欲进栈的运算符级别高于或等于(右结合)栈顶运算符, 则进栈

(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

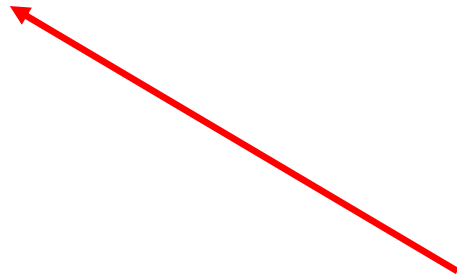
例: $10 + 'a' + i * f - d / e$

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



初始: 两栈均为空

例: $10 + 'a' + i * f - d / e$

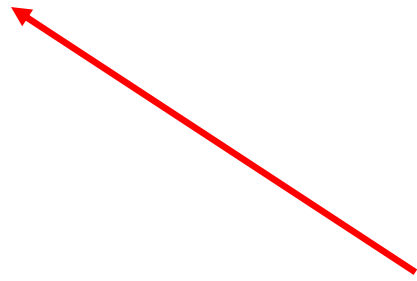


10进栈

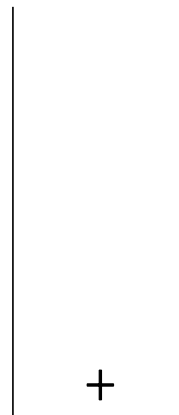
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量



例: 10+'a'+i*f-d/e



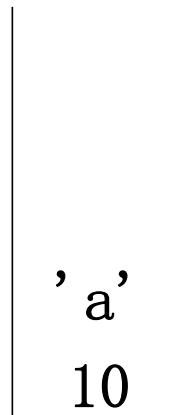
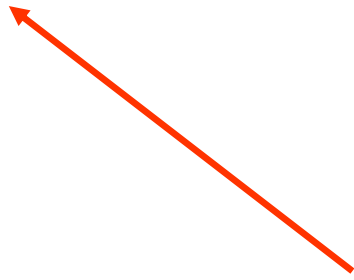
+进栈



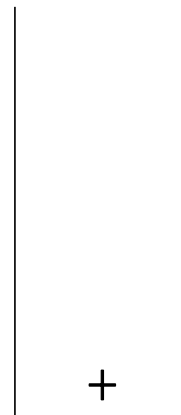
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量



例: $10 + 'a' + i * f - d / e$



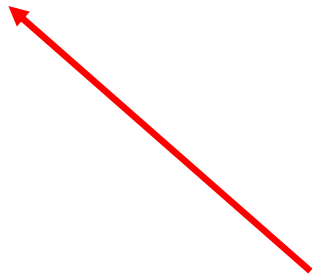
'a' 进栈



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



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



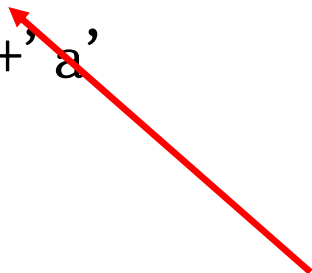
'a'
10

+

要进栈的(+)等于栈顶(+), 且左结合, 求值

例: $10 + 'a' + i * f - d / e$

步骤① $10 + 'a'$



$10 + 'a'$

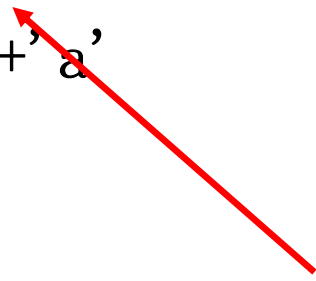
要进栈的 (+) 等于栈顶 (+), 且左结合, 求值

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: $10 + 'a' + i * f - d / e$

步骤① $10 + 'a'$



$10 + 'a'$

+进栈

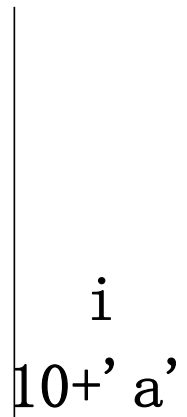
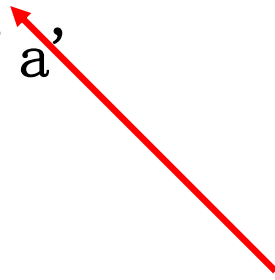
+

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

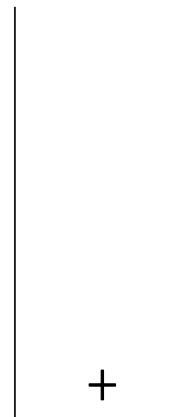


例: $10 + 'a' + i * f - d / e$

步骤① $10 + 'a'$



i进栈



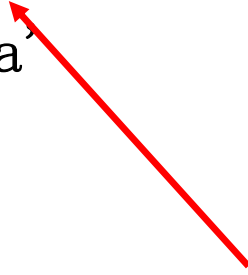
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量





例: $10 + 'a' + i * f - d / e$

步骤① $10 + 'a'$



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

i
 $10 + 'a'$

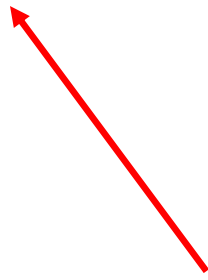
*

+

*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

*

+

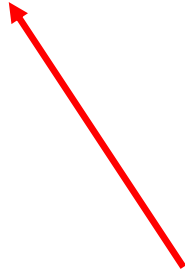
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量





例: $10 + 'a' + i * f - d / e$

步骤① $10 + 'a'$



f
i
 $10 + 'a'$

*

+

要进栈的(-) 低于栈顶的(*), 先计算

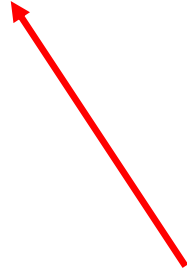
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量



例: $10 + 'a' + i * f - d / e$

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

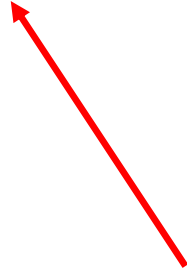
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量



例: $10 + 'a' + i * f - d / e$

步骤① $10 + 'a'$

步骤② $i * f$



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

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

+

要进栈的(-)等于栈顶的(+), 左结合, 先计算



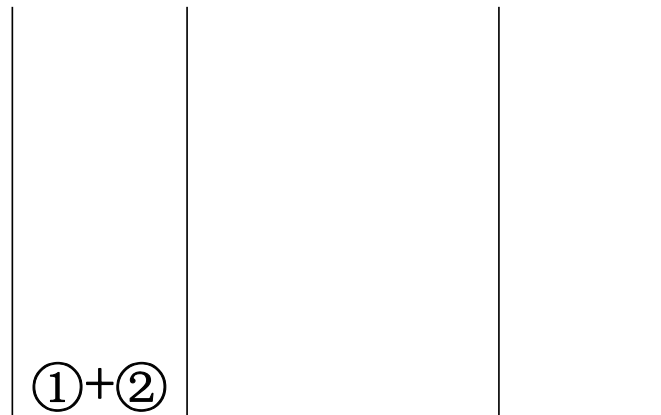
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量

例: $10 + 'a' + i * f - d / e$

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



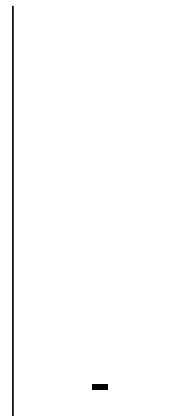
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量

例: $10 + 'a' + i * f - d / e$

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



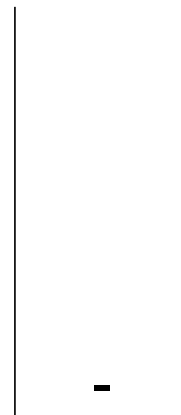
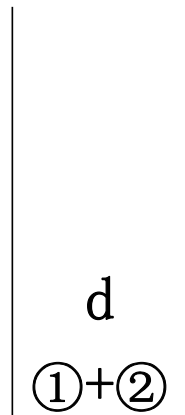
例: $10 + 'a' + i * f - d / e$

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)

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



d进栈



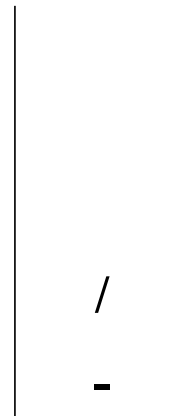
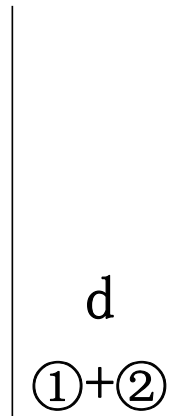
例: $10 + 'a' + i * f - d / e$

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)

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



/进栈 (要进栈的/高于栈顶的-)



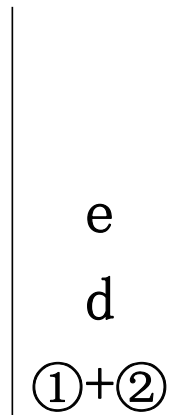
说明: 10 - 整型常量
'a' - 字符型常量
i - 某个int型变量
f - 某个float型变量
d - 某个double型变量
e - 某个long型变量

例: $10 + 'a' + i * f - d / e$

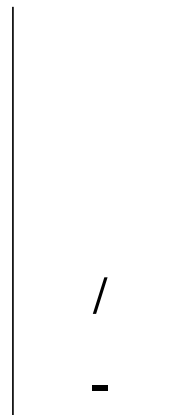
步骤① $10 + 'a'$

步骤② $i * f$

步骤③ ①+② (步骤①的和 + 步骤②的积)



e进栈





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: $10 + 'a' + i * f - d / e$

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

d / e
①+②

计算 d / e

-



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: $10 + 'a' + i * f - d / e$

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)

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

③-④

说明:

本例只是一个简单的说明, 表述也不够准确, 且未考虑到括号、单目、三目运算符等, 真实的表达式求值比这个复杂得多, 待后续课程再进行深入学习

● 做为初学者理解, 够用了

表达式分析并求值完成 (运算符栈为空, 运算数栈只有一个数)



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.5. 字符型、整型、实型的混合运算

★ 运算的方法

`10+'a'+1.5-8765.1234*'b'`

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（阅读：P. 53~57 3.4.4 类型转换）

高 ↑
long double
double
float
unsigned long long
long long
unsigned long
long
unsigned [int]
低
int ← char, u_char, short, u_short

```
char a;  
short b;  
  
a+b    (int + int)
```

←整型提升(表示必定的转换)

整型提升的等价说法：
参与算术运算的最小数据类型是int



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.5. 字符型、整型、实型的混合运算

★ 转换的优先级（阅读：P. 53~57 3.4.4 类型转换）

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;
}
```

思考：b=a+1中
问1：a+1=?
什么类型？
问2：b为什么是-32768？



已知：short a=32767, b=a+1;
则：b是-32768
验证：int <- char / short是必定的转换
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 = 32767;
    short b = 1;
    cout << sizeof(a + 'A') << endl;
    cout << typeid(a + b).name() << endl;

    return 0;
}
```

其它两种可用的方法



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.5. 字符型、整型、实型的混合运算

例：10+'a'+i*f-d/e (刚才用栈解释表达式求值的例子)

- 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. 基础知识

2.9. 算术运算符与算术表达式

2.9.5. 字符型、整型、实型的混合运算

★ 运算的方法

`10+'a'+1.5-8765.1234*'b'`

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（阅读：P. 53~57 3.4.4 类型转换）

特别说明：转换优先级的规则很复杂，这里只是一个适合初学者的简易规则，不完全正确，如果与实际编译器的表现有不一致的地方，以编译器为准

<https://zh.cppreference.com/w/c/language/conversion>

https://zh.cppreference.com/w/cpp/language/operator_arithmetic

高 ↑ long double
double
float
unsigned long long
long long
unsigned long
long
unsigned [int]
低 ↓ int <- char, u_char, short, u_short
<- 整型提升(表示必定的转换)

4) 否则两个操作数均为整数。此情况下，

首先，两个操作数都会经历整型提升（见后述）。然后

- 若两类型在提升后相同，则该类型即为共用类型
- 否则，若两操作数在提升后有相同的符号性（均为有符号或均为无符号），则拥有较低转换等级（见后述）者会隐式转换成拥有较高转换等级的操作数的类型
- 否则，两者符号性不同：若无符号类型操作数拥有大于或等于有符号类型操作数的转换等级，则有符号类型操作数会隐式转换成无符号类型
- 否则，两者符号性不同且有符号操作数的等级大于无符号操作数的等级。此情况中，若有符号类型可以表达无符号类型的所有值，则有无符号类型的操作数被隐式转换成有符号操作数的类型。
- 否则，两个操作数都会经历隐式转换，到有符号类型的无符号类型对应者。

```
1.f + 20000001; // int 被转换成 float，给出 20000000.00
                // 相加后舍入到 float，给出 20000000.00
(char)'a' + 1L; // 首先，char 被提升回 int。
                // 这是有符号+有符号的情形，等级不同
                // int 被转换成 long，结果是 signed long 的 98
2u - 10; // 无符号/有符号，等级相同
         // 10 被转换成无符号，无符号数学运算为模 UINT_MAX+1
         // 对于 32 位 int，结果是 unsigned int 类型的 4294967288（即 UINT_MAX-7）
0UL - 1LL; // 无符号/有符号，相异等级，有符号的等级较大。
           // 若 sizeof(long) == sizeof(long long)，则有符号数不能表示所有无符号数
           // 这是最后一种情况：两个操作数都被转换成 unsigned long long
           // 结果是 unsigned long long 类型的 18446744073709551615（ULLONG_MAX）
```



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.6. 自增与自减运算符

2.9.6.1. 形式

++ (--) 变量名 先自增/减1, 后使用 (前缀, 优先级第3组)

变量名 ++ (--) 先使用, 后自增/减1 (后缀, 优先级第2组)

```
#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gcc系列), 可能两种执行顺序
 - 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
VS DevC++



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.6. 自增与自减运算符

2.9.6.1. 形式

2.9.6.2. 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++;
```

```
++i;
```

单独成为语句时，前后缀等价

思考：在不考虑CPU硬件优化，编译器软件优化的情况下，
纯语句角度考虑，哪种效率更高？为什么？

★ 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



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.6. 自增与自减运算符

2.9.6.1. 形式

2.9.6.2. 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)++: 编译器认为和是只读的 **(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

例: (i++)+(i++)+(i++)

(不同编译系统处理方式不同，不深入讨论)

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int i = 3, j = 4;
6      const int k = 3;
7      (i + j)++;
8      10++;
9      k++;
10     return 0;
11 }
```

error C2105: “++” 需要左值
error C2105: “++” 需要左值
error C2105: “++” 需要左值

//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. 基础知识

2.9. 算术运算符与算术表达式

2.9.7. 强制类型转换

(类型名)(表达式) / 类型名(表达式) / static_cast<类型名>(表达式)

(int)(a+b) / int(a+b) / static_cast<int>(a+b)

(int)a / int(a) / static_cast<int>(a)

C形式

C++形式

C++形式

<pre>#include <iostream> using namespace std; int main() { int a=10; double b=3.5, c=3.6; cout << sizeof(a + (int)b) << endl; cout << sizeof(a + (int)b + c) << endl; cout << sizeof(a + (int)(b+c)) << endl; cout << endl; cout << a + (int)b << endl; cout << a + (int)b + c << endl; cout << a + (int)(b + c) << endl; cout << endl; cout << typeid(a + (int)b).name() << endl; cout << typeid(a + (int)b + c).name() << endl; cout << typeid(a + (int)(b + c)).name() << endl; return 0; }</pre>	C方式	<pre>#include <iostream> using namespace std; int main() { int a=10; double b=3.5, c=3.6; cout << sizeof(a + int(b)) << endl; cout << sizeof(a + int(b) + c) << endl; cout << sizeof(a + int(b+c)) << endl; cout << endl; cout << a + int(b) << endl; cout << a + int(b) + c << endl; cout << a + int(b + c) << endl; cout << endl; cout << typeid(a + int(b)).name() << endl; cout << typeid(a + int(b) + c).name() << endl; cout << typeid(a + int(b + c)).name() << endl; return 0; }</pre>	C++方式	<pre>#include <iostream> using namespace std; int main() { int a=10; double b=3.5, c=3.6; cout << sizeof(a + static_cast<int>(b)) << endl; cout << sizeof(a + static_cast<int>(b) + c) << endl; cout << sizeof(a + static_cast<int>(b+c)) << endl; cout << endl; cout << a + static_cast<int>(b) << endl; cout << a + static_cast<int>(b) + c << endl; cout << a + static_cast<int>(b + c) << endl; cout << endl; cout << typeid(a + static_cast<int>(b)).name() << endl; cout << typeid(a + static_cast<int>(b) + c).name() << endl; cout << typeid(a + static_cast<int>(b + c)).name() << endl; return 0; }</pre>	C++方式
--	-----	--	-------	---	-------

Microsoft Visual
4
8
4
13
16.6
17
int
double
int
三个程序输出一致



§ 2. 基础知识

2.9. 算术运算符与算术表达式

2.9.7. 强制类型转换

(类型名)(表达式) / 类型名(表达式) / static_cast<类型名>(表达式)

(int)(a+b) / int(a+b) / static_cast<int>(a+b)

(int)a / int(a) / static_cast<int>(a)

C形式

C++形式

C++形式

```
int a;
```

```
double b, c;
```

```
a+(int)b / a+int(b) / a+static_cast<int>(b) int型
```

```
a+(int)b+c / a+int(b)+c / a+static_cast<int>(b)+c double型
```

```
a+(int)(b+c) / a+int(b+c) / a+static_cast<int>(b+c) int型
```

C形式

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)

```
int a = -3;
```

```
unsigned int b = 2;
```

```
cout << a+b << endl;
```

?

```
int a = -3;
```

```
unsigned int b = 2;
```

```
cout << int(a+b) << endl;
```

?



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 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，VS有warning）
```

warning C4305: “=” : 从“double”到“float”截断



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 2. 字符型、整型、实型间相互赋值的类型转换

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

f=1234567.89 => long 1234567

=> char 溢出

=> short 溢出

```
#include <iostream>
using namespace std;
int main()
{
    float f1 = 123.0F;
    char c1 = f1;
    cout << int(c1) << endl;

    double f2 = 12345.67;
    char c2 = f2;
    short s2 = f2;
    cout << int(c2) << ' ' << s2 << endl;

    double f3 = 1234567.89;
    char c3 = f3;
    short s3 = f3;
    long l3 = f3;
    cout << int(c3) << ' ' << s3 << ' ' << l3 << endl;

    return 0;
}
```

- 1、观察运行结果
- 2、int(c1)的目的是什么？
- 3、溢出的结果与长字节整数赋短字节的规律是否相同？

Microsoft Visual Studio 调试控制台

```
123
57 12345
-121 -10617 1234567
```




§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 2. 字符型、整型、实型间相互赋值的类型转换

★ char/short/int/long/long 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

自行构造测试
程序并理解

● 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 (VS中是 inf 形式)



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 2. 字符型、整型、实型间相互赋值的类型转换

★ char/short/int/long/long 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

★ 同类型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. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 3. 复合的赋值运算符

含义：将算术运算符/**位运算符**和赋值组合在一起，同时完成计算并赋值

形式：变量 复合赋值运算符 表达式(常量、变量、表达式)

附录D 优先级第16组 共10种(**目前要求：+ - * / %，位运算符暂略**)

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

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

★ 优先级相同 (**注意：+、*优先级不同，但+=、*=优先级相同**)



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 4. 赋值表达式

2. 10. 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. 10. 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. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 4. 赋值表达式

2. 10. 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

`int a=5, b=5, c=5;` 正确

`int a=b=c=5;` 错误

`int a, b, c;`

`a=b=c=5;` 正确

```
error C2065: "b": 未声明的标识符
error C2065: "c": 未声明的标识符
```



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 4. 赋值表达式

2. 10. 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> | <code>*</code> 优先级最高 |
| ② <code>a-=a*a</code> | <code>+=</code> 、 <code>-=</code> 相同, 右结合 |
| ③ <code>a=a-a*a</code> | <code>-=</code> 展开 |
| ④ <code>a+=(a=a-a*a)</code> | <code>+=</code> 最后 |
| ⑤ <code>a=a+(a=a-a*a)</code> | <code>+=</code> 展开 |

<code>12*12</code>	<code>a=12</code>
<code>a-=12*12</code>	<code>a=12</code>
<code>a=12-12*12</code>	<code>a=-132</code>
<code>a+=-132</code>	<code>a=-132</code>
<code>a=(-132)+(-132)</code>	<code>a=-264</code>

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

int main()
{
    int a = 12;
    a += a -= a * a;
    cout << a << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

-264



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 4. 赋值表达式

2. 10. 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

思考: 观察下面的程序, 想明白为什么左侧是正确而右侧是错误的, 错在哪个=`=`上?

<pre>(a=3*5)=4*3 3*5 a=3*5 a=15 4*3 a=4*3 a=12</pre> 虽然<code>=</code>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<code>=</code>执行时形式为变量	<pre>a=3*5=4*3 3*5 4*3 4*3 => 3*5 (4*3的值赋给表达式3*5, 错)</pre>
<pre>#include <iostream> using namespace std; int main() { int a; (a = 3 * 5) = 4 * 3; cout << a << endl; return 0; }</pre> Microsoft Visual Studio 调试控制台 12	<pre>#include <iostream> using namespace std; int main() { int a; a = 3 * 5 = 4 * 3; cout << a << endl; return 0; }</pre> error C2106: “=”: 左操作数必须为左值



§ 2. 基础知识

2. 10. 赋值运算符与赋值表达式

2. 10. 4. 赋值表达式

2. 10. 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

★ 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 (VS是40, RedFlag5是30)

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

```
int main()
{
    int a;
    cout << (a = 10) + (a = 20) << endl;
    return 0;
}
```

不同编译器取值不同:

VS/DevC++ : 40

RedFlag5 gcc : 30

```
[root@RF5-X64 ~]# cat test.cpp
#include <iostream>
using namespace std;
int main()
{
    int a;
    cout << (a = 10) + (a = 20) << endl;
    return 0;
}
```

```
[root@RF5-X64 ~]# g++ -o test test.cpp
```

```
[root@RF5-X64 ~]# ./test
```

```
30
```

```
[root@RF5-X64 ~]#
```




§ 2. 基础知识

2. 11. 逗号运算符和逗号表达式

2. 11. 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. 11. 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> 式1(赋值表达式): <code>a=15</code> 式2(算术表达式): <code>15*4=60</code> 整个逗号表达式的值为60	<code>b=(a=3*5, a*4)</code> <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> 式1(逗号表达式) 式1-1(赋值表达式)=15 (a=15) 式1-2(算术表达式)=60 式1 =60 式2(算术表达式)=20 整个逗号表达式的值为20	<code>b = ((a=3*5, a*4), a+5)</code> <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> <code>cout << a << b << c;</code> <code>cout << (a, b) << (a, c) << (a, b, c);</code>	543433 为什么? 再次强调: 只输出最基本的信息



§ 2. 基础知识

2. 12. 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