



第二讲 C++ 编程基础

潘建瑜@MATH.ECNU



1

C++ 语言概述

2

C++ 编程基础

3

简单输入输出

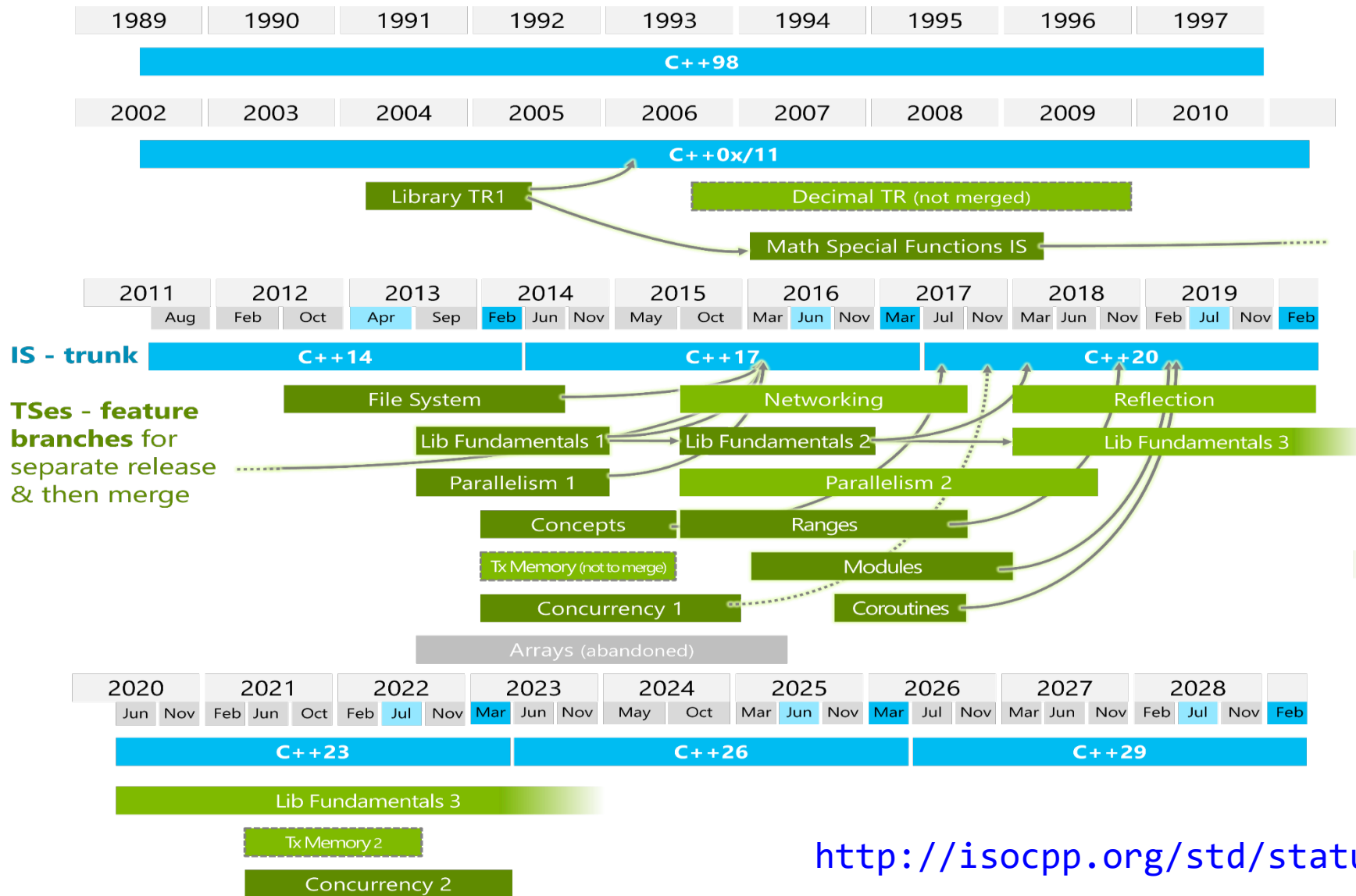
1

C++ 语言概述

- C++ 的发展
- C++ 源程序结构与书写规范
- C++ 编译器和集成开发环境

- C++ 是从 C 语言发展演变而来，可以看成是 C 的超集
- 1980 年由 Bjarne Stroustrup 开发创建
- 1983 年正式取名 C++，1989 年开始 C++ 的标准化工作
- 1994 年制定了 ANSI C++ 标准草案
- **1998 年由 ISO 批准为国际标准，通称 C++98**
- **2011 年发布 C++11**，被称为是**里程碑式更新**，注入了现代编程语言的活力，标志着进入**现代 C++** 时代
- Recent milestones: C++23 done; C++26 work has begun
- C++ 可用于软件/系统/游戏开发、高性能计算、单片机和嵌入式系统等





<http://isocpp.org/std/status>

程序示例分析

```
#include <iostream> // 预处理指令，载入头文件
```

```
using namespace std; // 使用标准的命名空间
```

```
int main()
```

```
{
```

```
    cout << "Hello!" << endl;
```

```
    cout << "Welcome to C++! " << endl;
```

```
    return 0;
```

```
}
```

主函数

cout : 标准输出，通常指屏幕

<< : 插入

endl : 换行并刷新流

C++源程序结构

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello!" << endl;
    cout << "Welcome to C++! " << endl;

    return 0;
}
```

C++ 源程序结构:

- ▶ 一个 C++ 源程序由一个或多个源文件组成
- ▶ 每个源文件可由一个或多个函数组成
- ▶ 一个源程序有且只能有一个 **main 函数**，即主函数
- ▶ 程序执行从 main 开始，在 main 中结束
- ▶ 源程序中可以有预处理命令，通常应放在源文件或源程序的最前面
- ▶ 一行可以写多个语句，一个语句可以分几行书写

C++ 程序书写规范

- ❑ 每条语句以分号 “;” 结尾，（预处理命令，函数头和花括号 “}” 之后除外）
- ❑ 标识符、关键字之间用空格隔开（若已有明显的间隔符，可不加）
- ❑ 区分大小写
- ❑ 注释：// 和 /* */
- ❑ 适当的缩进：锯齿形书写格式
- ❑ 所有标点符号必须在英文状态下输入

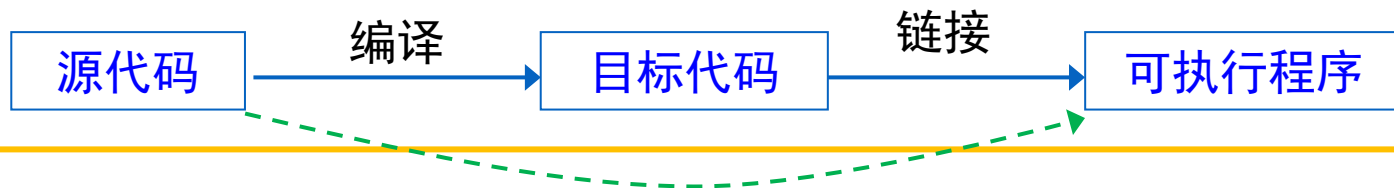
代码书写建议：

- ▶ { } 要对齐
- ▶ 一行写一个语句，一个语句写一行
- ▶ 缩进一致，可使用 TAB
- ▶ 有合适的空行，有足够的注释

C++ 编译器

什么是编译器

- ❑ 编译器就是将“高级语言”翻译为“机器语言”的程序
- ❑ 一个现代编译器的主要工作流程：



常见的 C++ 编译器

- ❑ **GNU C++** 开源免费，Linux/Unix 平台，非常优秀
- ❑ **Intel DPC++** Intel 编译器，基于 LLVM 框架，Win/Linux 都适用
- ❑ **Nvidia C++** NVIDIA提供的 C++ 编译器，基于 LLVM 框架，支持 CPU 和 GPU 并行
- ❑ **Visual C++** 微软，Windows平台，集成在 Visual Studio 中
- ❑ **Clang** LLVM 的编译器前端，an LLVM native C/C++/Objective-C compiler

Integrated Development Environment

IDE (集成开发环境)

- ❑ 用于程序开发的应用软件，一般包括编辑器、编译器、调试器和图形界面等
- ❑ 常见的 **C++** 集成开发环境
 - **Dev C++** : 小巧免费，功能简单，适合初学者和教学
 - **VS Code + MinGW / Remote SSH**: 微软免费IDE + GCC (微软有配置方法指导)
 - **Visual Studio** : 微软出品，大而全，有社区版

Dev C++

- ❑ **Dev-Cpp 5.11** (gcc 4.9.2, 支持 C11/C++11, 缺省为 C90/C++98, 机房电脑)
- ❑ **Embarcadero Dev-Cpp 6.3** (gcc 9.2, 支持 C17/C++17, 缺省为 C11/C++14)

2

C++ 编程基础 一些基本概念

- C++ 字符集：标识符，关键字
- C++ 数据类型与类型转换
- 变量、常量、符号常量
- 运算符、运算优先级
- 语句与表达式

字符集

C++ 字符集

- 字母 (大写和小写, 共 **52** 个)
- 数字 (**0** 到 **9** 共 **10** 个)
- 空白符 (空格符、制表符、换行符)
- 标点 和 特殊字符

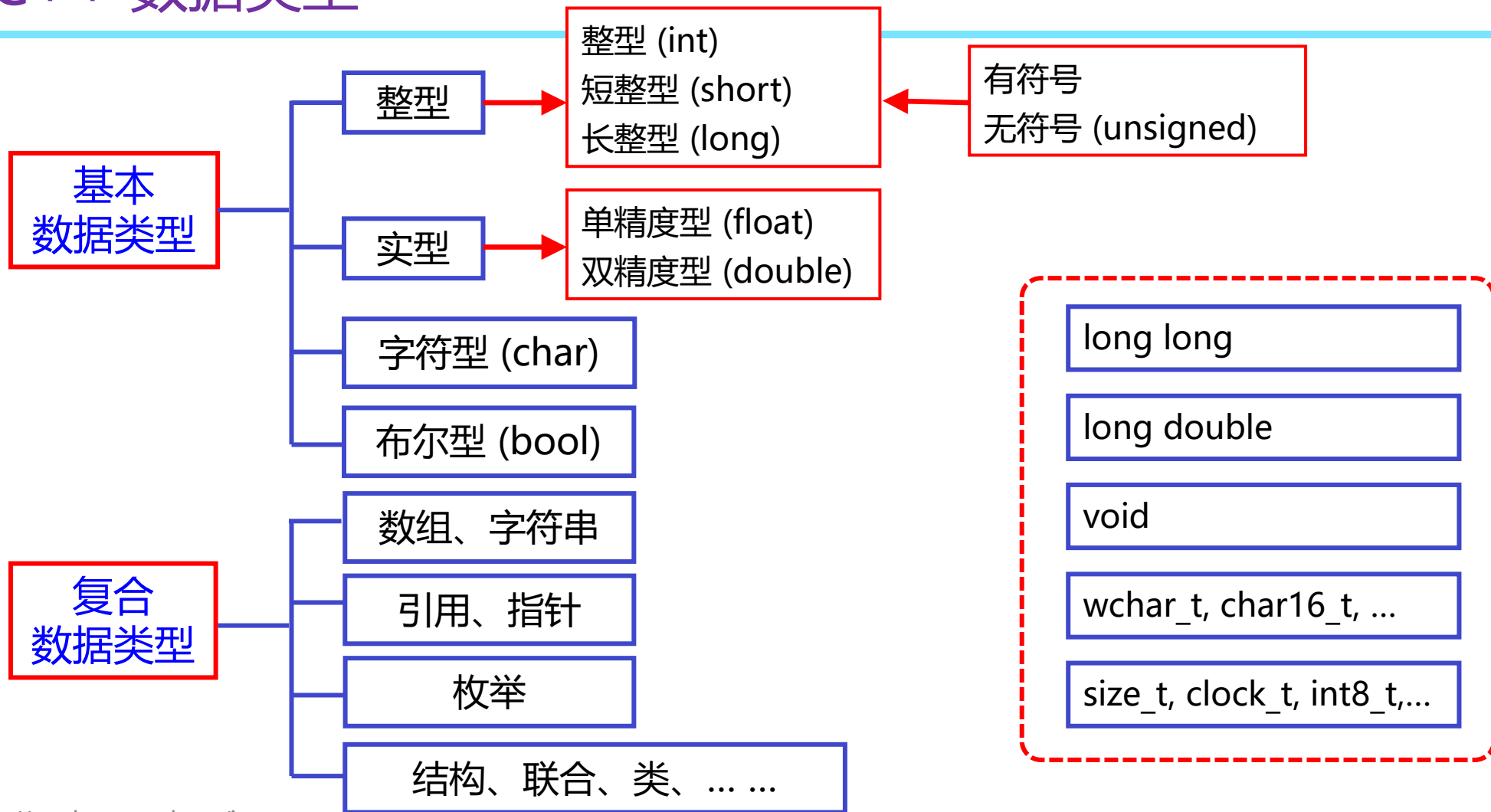
!	#	%	^	&	*	()	[]	{ }	_	+	
=	-	~	<	>	/	\	'	"	;	.	,

† 注: 这里的标点符号都是指在英文状态下的标点

词法记号/词汇

- ❑ **标识符**：用来标识变量名、函数名、对象名等的字符序列
 - 由字母、数字、下划线组成，第一个字符必须是字母或下划线
 - 区分大小写，不能用关键字
 - C++ 不限制标识符长度，实际长度与编译器有关
 - 命名原则：见名知意、不宜混淆
- ❑ **关键字**：具有特定意义的字符串，通常也称为保留字
 - 包括：类型说明符（类型标识符）、语句定义符（控制命令）、编译预处理命令等
- ❑ **运算符**（详见后面介绍）
- ❑ **分隔符**：空格、逗号、冒号、分号、()、{ }
- ❑ **注释符**：以 “ /* ” 开头并以 “ */ ” 结尾（大段注释）；或 “ // ” 开头（行注释）
- ❑ **文字**：直接用字符表示的数据，即常量，如数字、字符串等

C++ 数据类型



基本数据类型 / 原生数据类型

ex02_sizeof.cpp

类型	类型说明符/类型标识符	所占字节数	表示范围
整型	short	2	$-2^{15} \sim 2^{15} - 1$
	int	2 / 4	$-2^{15} \sim 2^{15} - 1 / -2^{31} \sim 2^{31} - 1$
	long	4 / 8	$-2^{31} \sim 2^{31} - 1 / -2^{63} \sim 2^{63} - 1$
	unsigned short	2	$0 \sim 2^{16} - 1$
	unsigned int	2/4	$0 \sim 2^{16} - 1 / 0 \sim 2^{32} - 1$
	unsigned long	4/8	$0 \sim 2^{32} - 1 / 0 \sim 2^{64} - 1$
实型	float	4 (6-7)	$10^{-38} \sim 10^{38}$
	double	8 (15-16)	$10^{-308} \sim 10^{308}$
	long double	16 (18-19)	$10^{-4932} \sim 10^{4932}$
布尔型	bool	1	true, false
字符型	char	1	

► C++ 标准没有规定每种数据类型的字节数，只规定大小顺序，具体长度由处理器和编译器确定

数据运算基本准则和数据类型转换

相同类型的数据才能直接运算，运算结果为同类型数据

数据类型转换：自动转换/隐式转换

- ❑ 不同类型的数据进行运算，需先转换成同一类型
- ❑ 转换按数据长度增加的方向进行，以保证精度不降低
- ❑ 所有的浮点运算都是以双精度进行的
- ❑ 赋值号两边的数据类型不同时，将右边表达式的值转换为左边变量的类型

char → short → int → long
→ unsigned long → double ← float

```
int    i = 2;  
double x = 3.2, y;  
y = i + x;
```

数据类型转换：强制转换

数据类型转换：强制转换/显式转换

类型标识符(表达式) // C++ 风格

(类型标识符)表达式 // C 风格

`static_cast`<类型标识符>(表达式) // C++, 静态转换

❑ 将表达式的 **值** 转换成指定的数据类型，作用是临时性的

```
int a = 2, b = 5;
double x, y, z;
x = b/a;           // x = 2.0
y = double(b)/a;   // y = 2.5
z = double(b/a);   // z = 2.0
```

ex02_datatype_conversion.cpp

```
int    a = 2, b = 5;
double c = static_cast<double>(b)/a;
```



注意：类型转换不会改变变量本身的数据类型！

数据类型转换规则

- ❑ 浮点型转整型：直接丢掉小数部分
- ❑ 字符型转整型：取字符的 **ASCII** 码
- ❑ 整型转字符型：取 **ASCII** 码对应的字符

ex02_datatype_conversion.cpp

```
int i;  
char c;  
i = 3.6; cout << "i = " << i << endl; // i = 3  
i = -3.6; cout << "i = " << i << endl; // i = -3  
i = 'm'; cout << "i = " << i << endl; // i = 109  
c = 90; cout << "a = " << a << endl; // c = Z
```

变量

变量：用于存储数据，值可以改变

- ❑ **变量名：**要求与标识符相同
- ❑ **变量类型：**整型、实型、字符型、布尔型
- ❑ 变量必须 **先声明，后使用**

变量的声明

类型说明符 变量名列表;

- ❑ **变量的初始化：**三种方式（赋值号，小括号，大括号）

```
int i = 20;  
int j(2);  
int k{4}; // C++11
```

常量： 常数常量

常量： 在程序运行中值不能改变的量

常数常量

- ❑ 整型常量： 整数，后面加 l 或 L 表示长整型，加 u 或 U 表示无符号整型
- ❑ 实型常量： 缺省为双精度，
后面加 f 或 F 表示单精度，加 l 或 L 表示 long double
- ❑ 字符型常量： 用单引号括起来的单个字符或转义字符
- ❑ 字符串常量： 用双引号括起来的字符序列
- ❑ 布尔常量： true 和 false

```
123, -456, 123L, 456U;  
1.2, 1.2F, 1.2L, 1.2e8, 1.2e8F  
'M', 'A', 'T', 'H', '?', '$'  
"MATH@ECNU"
```

常量：符号常量

符号常量

`const` 类型说明符 变量名 = 表达式;

- 符号常量在声明时必须初始化
- 符号常量的值在程序中不能被修改（不能重新赋值）

```
const float PI = 3.1415926; // 可以用常数初始化  
const float x = PI/2 + 2.6; // 也可以用表达式初始化
```

常量表达式 constexpr (C++11)

constexpr 类型说明符 变量名 = 表达式;

- ❑ 值不会变 (与 const 类似)
- ❑ 在编译时就能得到计算结果 (const 可能会在运行时才得到结果)
- ❑ 初始化时, 右边的表达式必须是常量表达式

```
int x = 11; const int y = 22;  
constexpr int z1 = 11 + y; // OK  
constexpr int z2 = x + 22; // ERROR
```

ex02_constexpr.cpp

- ❑ 编译器保证 constexpr 对象是常量表达式, 即在编译时就能得到结果
- ❑ constexpr 函数 (能用于常量表达式的函数) 可以把运行期的计算迁移至编译期, 从而使得程序运行更快 (但会增加编译时间)

AUTO 类型推导 (C++11)

auto 类型推导

- 根据 初始化表达式 对推导出变量的数据类型

```
auto a = 2; // OK, 推导出 a 的类型为 int  
auto b = a + 3.0; // OK, 推导出 b 的类型为 double  
auto c ; // ERROR, 没有初始化, 无法推导 c 的类型
```

ex02_auto.cpp

decltype 类型推导 (C++11)

decltype 类型推导

- ❑ 在编译时进行类型推导，分析表达式获取其类型

```
int a = 2;  
auto b = 0.0;  
decltype(a+b) c; // double d;
```

ex02_decltype.cpp

- ❑ 与 auto 类似，但用于不同的场景：decltype 的主要用途是推导表达式的类型，而不是像 auto 那样从变量的初始化表达式中推导类型。

为已有的数据类型说明符取别名

给类型说明符取别名

typedef 已有数据类型说明符 类型别名;

using 类型别名 = 已有数据类型说明符; // C++11

- ❑ “原数据类型说明符” 必须是已经存在或定义过的
- ❑ 可以同时取多个别名

```
typedef double area, volume; // 可以同时定义多个别名，用逗号隔开
using real = float;
area x;
volume y;
real z;
```

ex02_typedef_using.cpp

运算符

- ❑ 算术运算符：+、-、*、/、%、++ (自增)、-- (自减)
- ❑ 赋值运算符：=、+=、-=、*=、/=、%=、&=、|=、^=、>>=、<<=
- ❑ 逗号运算符：, (把若干表达式组合成一个表达式)
- ❑ 求字节数运算符：sizeof (计算数据类型所占的字节数，运算符，不是函数)

- ❑ 关系运算符：用于比较运算，>、<、==、>=、<=、!=
- ❑ 逻辑运算符：用于逻辑运算，&&、||、!
- ❑ 条件运算符：是一个三目运算符，用于条件求值 (?:)
- ❑ 指针运算符：* (取内容)、& (取地址)
- ❑ 位运算符：按二进制位进行运算，&、|、^ (异或)、~ (取反)、<< (左移)、>> (右移)

赋值运算

□ 标准赋值语句： **变量 = 表达式**

```
x=3;  
y=z=4; // 这种方式不能用于初始化
```

□ 自增自减（前置和后置）： **++、--**

<code>x++;</code>	↔	<code>x = x+1;</code>
<code>++x;</code>	↔	<code>x = x+1;</code>
<code>y = x++*3;</code>	↔	<code>y = x*3; x = x+1;</code>
<code>y = ++x*3;</code>	↔	<code>x = x+1; y = x*3;</code>

```
int i = 10, j, k;  
j = i++; // j = ?  
k = ++i; // k = ?
```

ex02_increment.cpp

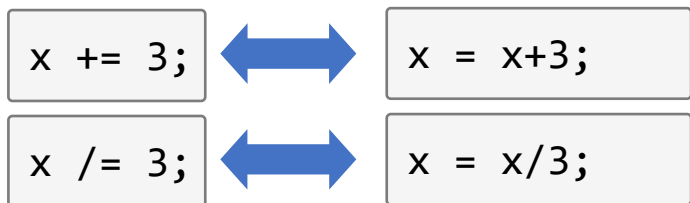
- ▶ **前置**：先自增自减，然后使用
- ▶ **后置**：先使用，然后自增自减

不要在同一语句中包含一个变量的多个++或--，因为它们的解释在C/C++标准中没有规定，完全取决于编译器的个人行为，同时也应避免 `y=x++*x` 及类似语句

复合赋值与逗号运算

□ 复合赋值语句:

`+=、-=、*=、/=、%=`



```
int a, b, c, d, e;
a = 5;
b = a + 3;
a = a + (c = 6);    // a=?, c=?
d = e = f = a;
e *= d;             // e=?
f /= c - 2;         // f=?
```

`ex02_shortcut_assignment.cpp`

□ 逗号运算符:

`表达式 1, 表达式 2`

► 先计算表达式 1 的值, 再计算表达式 2 的值, 并将表达式 2 的值作为整个表达式的结果

```
int a=2, b1, b2;
b1 = a++, a+10;    // b1=?
b2 = (a++, a+10);  // b2=?
```

`ex02_comma.cpp`

注意运算的优先级!
(逗号运算最低)

位运算

□ 位运算符：按二进制位进行运算

&、|、^ (异或)、~ (取反)、<< (左移)、>> (右移)

```
short a = 5, b = 14;  
short c1, c2, c3, c4, c5, c6;  
c1 = a & b;  
c2 = a | b;  
c3 = ~ a;  
c4 = a ^ b;  
c5 = a << 3;  
c6 = a >> 2;
```

ex02_bitwise.cpp

a : 00000000 00000101
b : 00000000 00001110

a&b	:	00000000	00000100
a b	:	00000000	00001111
~ a	:	11111111	11111010
a ^ b	:	00000000	00001011
a<<3	:	00000000	00101000
a>>2	:	00000000	00000001

其他运算符将会再后面的讲义中陆续介绍

运算优先级

高



低

() ++(后置) --(后置) 强制类型转换
! ++(前置) --(前置) +(正号) -(负号)
* / %
+ -
< <= > >=
== !=
&&
||
赋值(= += -= *= /= 等)
逗号运算(,)

更多详见讲义或课程主页

sizeof

- 返回对象或数据类型所占内存字节数 (注意: sizeof 是操作符, 不是函数)

sizeof(数据类型)

返回指定数据类型所占的字节数

sizeof(变量名)

返回存储指定变量所需的字节数

sizeof(表达式)

返回存储表达式结果所需的字节数

ex02_sizeof.cpp

```
int a, b, c, d;  
a = sizeof(int);  
b = sizeof(a);  
c = sizeof(3 + 5);  
d = sizeof(3.0L + 5);
```

常用数学函数

需加头文件 `#include <cmath>`

绝对值	<code>abs(x)</code>
平方根	<code>sqrt(x)</code>
指数函数	<code>exp(x)</code>
x^y	<code>pow(x,y)</code>
对数函数	<code>log(x)</code> , <code>log10(x)</code>
取整函数	<code>ceil(x)</code> , <code>floor(x)</code> , <code>round(x)</code> , <code>trunc(x)</code>
三角函数	<code>sin</code> , <code>cos</code> , <code>tan</code> , <code>asin</code> , <code>acos</code> , <code>atan</code>
双曲三角函数	<code>sinh</code> , <code>cosh</code> , <code>tanh</code> , <code>asinh</code> , <code>acosh</code> , <code>atanh</code>

`ex02_math.cpp`

表达式与语句

- ❑ 表达式：由运算符连接常量、变量、函数所组成的式子，通常返回一个具体的值
- ❑ 程序由语句构成，C++的语句包括：
 - ▶ 空语句（只有分号）
 - ▶ 声明语句
 - ▶ 表达式语句（赋值表达式）
 - ▶ 复合语句（将多个语句用 { } 括起来组成的一个语句）
 - ▶ 选择语句
 - ▶ 循环语句
 - ▶ 跳转语句
 - ▶

3

简单输入输出

- 标准输出: `cout`
- 标准输入: `cin`
- 操纵符: 控制输出的格式

标准输出

□ 输出到标准输出设备（显示器）：cout

```
x = 3.14159;  
cout << "x = " << x << endl;
```

► 转义字符：在输出语句中有特殊含义的字符

<code>\a</code>	响铃	<code>\r</code>	回车	<code>\\</code>	反斜杠
<code>\b</code>	退后一格	<code>\t</code>	水平制表符	<code>\'</code>	单引号
<code>\n</code>	换行	<code>\v</code>	垂直制表符	<code>\"</code>	双引号

```
x = 3.14159; y = 2.71828;  
cout << "x = " << x << "\t y = " << y << "\n";
```

ex02_IO_01.cpp

标准输入

- ❑ 从标准输入设备（键盘）中输入数据：cin

ex02_IO_02.cpp

```
cout << "Please input x: ";  
cin >> x ;
```

- ▶ 在输入语句前通常需要输出一些 提示信息

操纵符

❑ 操纵符：控制输入输出格式

需加头文件 `#include <iomanip>`

操纵符	含义
<code>endl</code>	插入换行符，并刷新流
<code>setw(n)</code>	设置域宽
<code>right</code>	右对齐（缺省方式）
<code>left</code>	左对齐
<code>setfill(字符)</code>	设置填充
<code>fixed</code>	使用定点形式
<code>scientific</code>	使用指数形式（科学计数法）
<code>setprecision(n)</code>	设置输出的有效数字个数； 若在 <code>scientific</code> 后使用，则设置小数位数

▶ 更多用法见第七讲：输入输出与文件操作

操纵符

❑ 操纵符的作用范围

- ▶ `setw`: 只对紧随其后的输出起作用
- ▶ 其它操纵符: 直至下一个同类型命令为止

ex02_IO_03.cpp

```
double x = 3.14159, y = 12.3456789;
cout << setprecision(5);
cout << "x = " << x << endl;
cout << "y = " << y << endl;

cout << fixed; cout << setprecision(5);
cout << "x = " << x << endl;
cout << "y = " << y << endl;

cout << left;
cout << "x = " << setw(20) << x << "MATH" << endl;
cout << setw(20) << "x = " << x << "MATH" << endl;
```

举例：贷款问题

例：银行贷款问题

已知贷款总额、贷款月利率和贷款年限，计算每月需偿还的金额和偿还总额。

□ 等额本息方式：

L ：贷款总额

r_m ：月利率

y ：还贷年限

$$\text{每月需偿还的金额: } L \times \frac{r_m (1 + r_m)^{12y}}{(1 + r_m)^{12y} - 1}$$

□ 等额本金方式：

R ：已偿还的本金总额

$$\text{每月需偿还的金额: } \frac{L}{12y} + (L - R) \times r_m$$

等额本息

ex02_loan.cpp

```
#include <iostream>
#include <cmath> // 数学函数
using namespace std;

int main()
{
    double Loan, rate_year, rate_month, year;
    double payment_month, payment_total;

    cout << "input loan amount: " ;           cin >> Loan;
    cout << "input yearly interest rate: ";   cin >> rate_year;
    rate_month = rate_year/1200;
    cout << "input number of years: ";        cin >> year;
    double vtmp = pow(1+rate_month,12*year); // 计算每月还款金额
    payment_month = Load*rate_month*vtmp/(vtmp-1);
    payment_total = payment_month*year*12;

    cout << "Monthly payment: " << payment_month << ", ";
    cout << "Total payment: " << payment_total;

    return 0;
}
```

举例：显示系统时间

例：显示系统当前时间

头文件 `ctime` 中函数 `time(0)` 或 `time(NULL)` 返回当前时间与1970年1月1日零时的时间差（格林威治时间，以秒为单位）

ex02_showtime.cpp

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
    long second, minute, hour;
    second = time(NULL);
    minute = second/60;
    hour = minute/60;

    cout << "当前北京时间是 ";
    cout << (hour+8) % 24 << ":" << minute % 60 << ":" << second % 60 << endl;
    return 0;
}
```

// 北京时间：格林威治时间 + 8小时

第二讲上机作业

- 1、编写程序，从键盘读入圆柱体的半径和高度，计算其表面积和体积，并将结果在屏幕上输出。
(π 取值 3.14159265，输入要有提示，程序取名 hw02_01.cpp)
- 2、银行提供两种 5 年定期存款方式：
一年期方式：年利率 10%，每年到期后，自动将本年度的利息加入本金中；
五年期方式：年利率 11%，五年后本金和利息一起归还储户。
编写程序，分别以两种方式存入 100 万，输出五年后各得多少？（程序取名 hw02_02.cpp）
- 3、修改程序 ex02_showtime.cpp，使得输出的时、分、秒都占两个位置，如：
14 点 25 分 10 秒显示为 14:25:10，9 点 8 分 5 秒显示为 09:08:05
(提示：setw 和 setfill，程序取名 hw02_03.cpp)