



第三章 程序设计基础

模块3.1：数据基础

主讲教师：同济大学计算机科学与技术学院 陈宇飞
同济大学计算机科学与技术学院 龚晓亮



目录

- C++发展背景
- 二进制
- 整数的补码
- C++的数据类型



目录

- C++发展背景

- 计算机内数据的表示
- 计算机程序设计语言的发展
- C++的发展历史
- C语言的特点
- C++语言的特点



3.1.1 C++发展背景

1. 计算机内数据的表示

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

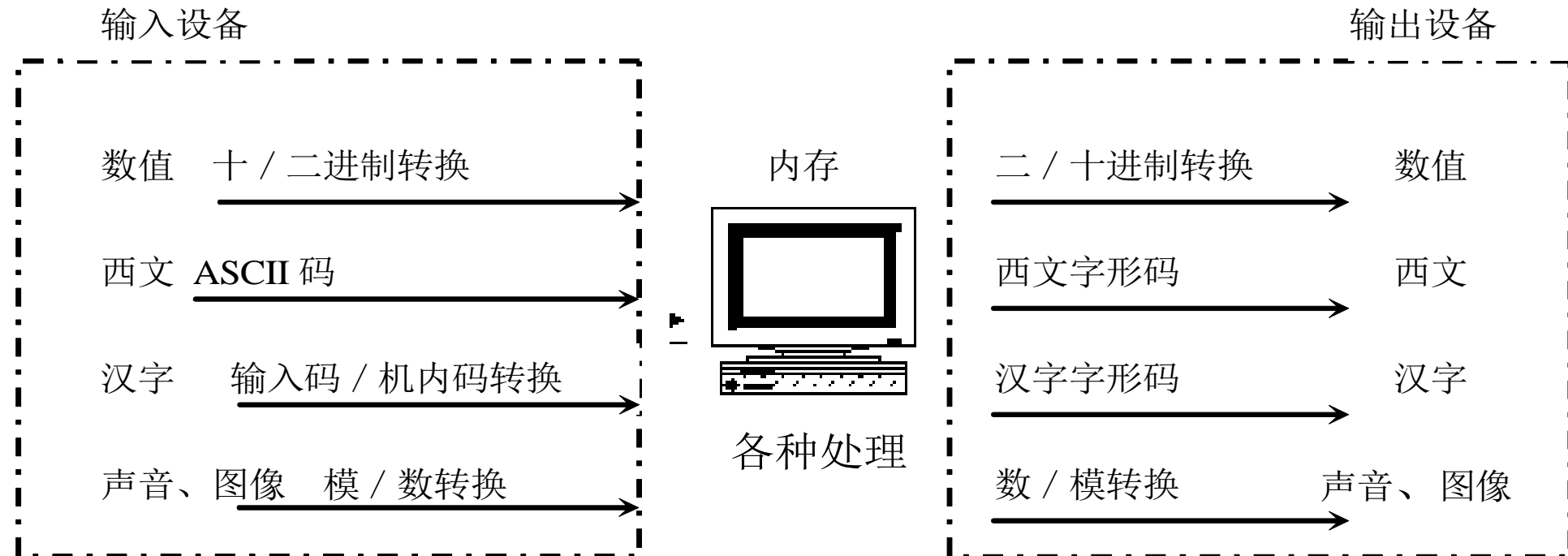
- 数字(有数值含义的数字序列)
- 文本(包括数码，即无数值含义的数字序列)
- 静态图像
- 动态视频
- 声音



3.1.1 C++发展背景

1. 计算机内数据的表示

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





3.1.1 C++发展背景

2. 计算机程序设计语言的发展

- ✓ 机器语言：只有0/1，面向具体机器，可读性差，无移植性
- ✓ 汇编语言：ADD、MOV等助记符，面向具体机器，可读性差，可移植性差
- ✓ 高级语言：自然语言表达，与具体机器无关，可读性好，可移植性好；但与硬件差距较大，对系统控制不易
- ✓ 自然语言：大模型智能编程



3.1.1 C++发展背景

2. 计算机程序设计语言的发展

- ✓计算机能直接识别的只有机器语言
 - ✓将汇编、高级语言转换为机器语言的工具就是语言处理程序
- 转换的过程称为**编译和链接**



3. 1. 1 C++发展背景

3. C++的发展历史

1960	-	1963	-	1967	-	1970	-	1972	-	1983	-	1989	-	1990	-	1999	-	2011
ALGOL60		CPL		BCPL		B		C		开始ANSI		ANSI-C89		ISO-C90		C99		C11

1982	-	1997	-	1998	-	2003	-	2011	-	2014	-	2017
C++		ANSI		ISO		C++03		C++11		C++14		C++17



3.1.1 C++发展背景

4. C语言的特点

- ✓语言简洁，使用方便，书写自由
- ✓运算符丰富
- ✓数据结构种类多
- ✓面向过程，具备结构化的控制语句，易于实现结构化的程序设计



3.1.1 C++发展背景

4. C语言的特点

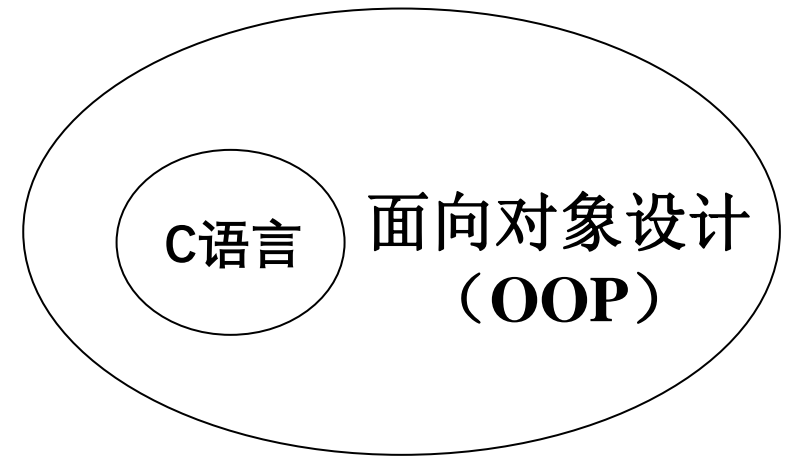
- ✓语法限制不严，灵活程度大（也易带来错误）
- ✓能直接访问物理地址，实现位操作，既具有高级，又具有低级语言的特性，适用于编写系统软件（UNIX、LINUX）
- ✓目标代码质量高，执行速度快
- ✓可移植性好



3.1.1 C++发展背景

5. C++语言的特点

- ✓与C完全兼容，原来的C语言代码可直接在C++下编译
- ✓对C语言的功能进行了扩充
- ✓支持面向对象机制，易于开发大型的软件



C++的组成



目录

- C++发展背景
- 二进制
- 整数的补码
- C++的数据类型



目录

- 二进制

- 进制的基本概念
- 二进制的基本概念
- 十进制与二进制的转换
- 八进制、十六进制



3.1.2 二进制

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, $0 \sim N-1$, 逢N进一



3.1.2 二进制

1. 进制的基本概念

四个概念

- ✓ 数码：数制中表示基本数值大小的不同数字符号
- ✓ 基数：数制中使用数码的个数
- ✓ 位权：数制中每个位置的价值
- ✓ 标识（后缀）：为了区分不同的进制，在数字后面加上相应的字母或者括号外加上数字下标。比如：
 $(66)_{10}$ 和66D都表示十进制数：66



3.1.2 二进制

1. 进制的基本概念

✓ 基数：指各种位制中允许选用基本数码的个数

十进制 : 0-9 \Rightarrow 基数为10

二十四进制 : 0-23 \Rightarrow 基数为24

六十进制 : 0-59 \Rightarrow 基数为60



3.1.2 二进制

1. 进制的基本概念

✓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

$$215 = 2*10^2 + 1*10^1 + 5*10^0$$

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）



3.1.2 二进制

1. 进制的基本概念

✓位权展开式

十进制数的表示，如678.34的位权展开式

$$678.34 = 6 \times 10^2 + 7 \times 10^1 + 8 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2}$$

数码

基数

位权

七进制数4532.1的位权展开式

$$4532.1 = 4 \times 7^3 + 5 \times 7^2 + 3 \times 7^1 + 2 \times 7^0 + 1 \times 7^{-1}$$



3. 1. 2 二进制

1. 进制的基本概念

进位制	数码	基数	位权	标识
二进制 Binary	0, 1	2	2^n	B
八进制 Octal	0, 1, 2, 3, 4, 5, 6, 7	8	8^n	O
十进制 Decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10	10^n	D或省略
十六进制 Hexadecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	16	16^n	H
R进制	$0 \sim R-1$	R	R^n	



3.1.2 二进制

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



3.1.2 二进制

2. 二进制的基本概念

- ✓ 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)

$$8 \text{ bit} = 1 \text{ byte}$$

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)



3.1.2 二进制

2. 二进制的基本概念

- ✓ 二进制的大数表示及不同单位的换算如表所示

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)



3.1.2 二进制

2. 二进制的基本概念

- ✓ 简写的时候, $b=\text{bit}/B=\text{byte}$
- ✓ 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: 12K

宽带速率: $20\text{Mbps} = 20 \times 10^6 \text{ (bit per second)}$

笔记本内存: $8\text{GB} = 8 \times 2^{40}$

硬盘: $1\text{TB} = 1 \times 10^{12}$ (厂商标注)
 $= 1 \times 2^{40}$ (计算机理解)

标称=>机内显示
 $10^{12}/2^{40}=0.9095$
 $1 \text{ TB} \Rightarrow 931 \text{ GB}$



3.1.2 二进制

2. 二进制的基本概念

- ✓ 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




3.1.2 二进制

3. 十进制与二进制的转换

✓ 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

2		25	
2	12	1	
2	6	0	
2	3	0	
2	1	1	
	0	1	

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



3.1.2 二进制

3. 十进制与二进制的转换

✓ 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

$$\begin{aligned}(11001)_2 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 16 + 8 + 0 + 0 + 1 \\ &= (25)_{10}\end{aligned}$$



3.1.2 二进制

3. 十进制与二进制的转换

✓ 十进制转二进制（小数）：乘2取整法

基本方法:

- (1) 要转换的小数乘以2，取整
- (2) 继续用小数部分乘以2，取整
- (3) 重复(2)直到满足精度或等于0为止
- (4) 把所有整数**顺序排列**，即为转换的二进制数

$$\begin{array}{r}
 2 \overline{) 100} \\
 2 \overline{) 50} \quad 0 \\
 2 \overline{) 25} \quad 0 \\
 2 \overline{) 12} \quad 1 \\
 2 \overline{) 6} \quad 0 \\
 2 \overline{) 3} \quad 0 \\
 2 \overline{) 1} \quad 1 \\
 \quad \quad 1
 \end{array}$$

$$\begin{array}{r}
 0.345 \\
 \times 2 \\
 \hline
 0.690 \\
 \times 2 \\
 \hline
 1.380 \\
 \times 2 \\
 \hline
 0.760 \\
 \times 2 \\
 \hline
 1.520 \\
 \times 2 \\
 \hline
 1.04
 \end{array}$$

The final result is $\frac{1}{16}$.

例 $(100.345)_{10} \approx (1100100.01011)_2$



3.1.2 二进制

3. 十进制与二进制的转换

✓ 二进制转十进制（**小数**）：**按权相加法**

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}
1	1	1	1	1	1	1	1	.	1	1
128	64	32	16	8	4	2	1		0.5	0.25

$$(11111111.11)_2 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 + 0.5 + 0.25 = (255.75)_{10}$$

$$\text{再例如: } (110111.01)_2 = 32 + 16 + 4 + 2 + 1 + 0.25 = (55.25)_{10}$$



3.1.2 二进制

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**



3.1.2 二进制

4. 八进制、十六进制

✓ 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

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



3. 1. 2 二进制

4. 八进制、十六进制

✓ 八、十六进制转十进制：按权相加法

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

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



3. 1. 2 二进制

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

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

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



3. 1. 2 二进制

4. 八进制、十六进制

✓ 八、十六进制转二进制：每1位转3/4位，不足补0

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

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

八进制	二进制	十六进制	二进制	十六进制	二进制
0	000	0	0000	8	1000
1	001	1	0001	9	1001
2	010	2	0010	A	1010
3	011	3	0011	B	1011
4	100	4	0100	C	1100
5	101	5	0101	D	1101
6	110	6	0110	E	1110
7	111	7	0111	F	1111



3. 1. 2 二进制

4. 八进制、十六进制

✓ 八进制和十六进制的相互转换： **二进制做桥梁**

$$\begin{aligned}(710.62)_8 &= (111\ 001\ 000.110\ 010)_2 \\ &= 1, 11\ 00, 1\ 000.110\ 0, 1000 \\ &= (1\ C\ 8\ .\ C\ 8)_{16}\end{aligned}$$

八进制	二进制	十六进制	二进制	十六进制	二进制
0	000	0	0000	8	1000
1	001	1	0001	9	1001
2	010	2	0010	A	1010
3	011	3	0011	B	1011
4	100	4	0100	C	1100
5	101	5	0101	D	1101
6	110	6	0110	E	1110
7	111	7	0111	F	1111

思考：

如何快速地将十进制数如456.78
分别转换成二、八、十六进制？



3.1.2 二进制

4. 八进制、十六进制

✓ 八进制和十六进制的相互转换：二进制做桥梁

思考：
如何快速地将十进制数如456.78
分别转换成二、八、十六进制？



$$\begin{array}{r|l} 8 & 456 \\ \hline 8 & 57 \\ \hline 8 & 7 \\ \hline & 0 \end{array}$$

$$\begin{array}{r} 0 \\ 1 \\ 7 \end{array}$$

$$\begin{array}{r} 0.78 \\ \times 8 \\ \hline 6.24 \\ \times 8 \\ \hline 1.92 \end{array}$$

八进制	二进制	十六进制	二进制	十六进制	二进制
0	000	0	0000	8	1000
1	001	1	0001	9	1001
2	010	2	0010	A	1010
3	011	3	0011	B	1011
4	100	4	0100	C	1100
5	101	5	0101	D	1101
6	110	6	0110	E	1110
7	111	7	0111	F	1111

$$\begin{aligned} (456.78)_{10} &\approx (7\ 1\ 0\ .\ 6\ 2)_8 \\ &= (111\ 001\ 000.\ 110\ 010)_2 \\ &= 1,\ 11\ 00,\ 1\ 000.\ 110\ 0,\ 1000 \\ &= (1\ C\ \ \ \ 8\ .\ \ C\ \ \ 8)_{16} \end{aligned}$$



目录

- C++发展背景
- 二进制
- 整数的补码
- C++的数据类型



目录

- 整数的补码

- 整型数的符号位

- 补码的基本概念



3.1.3 整数的补码

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**



3.1.3 整数的补码

1. 整型数的符号位

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**)



3.1.3 整数的补码

1. 整型数的符号位

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2：最高(最左)bit位表示+/-，再加15bit数据

例： **0**0000000 00000001 = +1 (max = +32767)

10000000 00000001 = -1 (min = -32767)

=> 可用两字节表示正负数，但数据范围减半 **(选)**



3.1.3 整数的补码

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

二进制原码表示

$$00000000 \ 00000001 = +1$$

$$01111111 \ 11111111 = +32767$$

$$10000000 \ 00000001 = -1$$

$$11111111 \ 11111111 = -32767$$



3.1.3 整数的补码

1. 整型数的符号位

✓ 数据的二进制原码

$$00000000 \ 00000001 = +1$$

$$01111111 \ 11111111 = +32767$$

$$10000000 \ 00000001 = -1$$

$$11111111 \ 11111111 = -32767$$

$$10000000 \ 00000000 = -0$$

$$00000000 \ 00000000 = +0$$

原码的缺陷1: +0与-0的二义性问题
=> 需要解决该缺陷



3.1.3 整数的补码

1. 整型数的符号位

✓ 数据的二进制原码

正数加法 $1 + 1$:

二进制		十进制
00000000	00000001	1
+) 1	1	1
-----		-----
00000000	00000010	(+2) 2

负数加法 $-1 + 1$:

二进制		十进制
10000000	00000001	-1
+) 1	1	1
-----		-----
10000000	00000010	(-2) 0

✓ 原码的缺陷2: 含负数的情况下无法正常运算



3.1.3 整数的补码

1. 整型数的符号位

✓ 数据的二进制原码

□ 原码的缺陷1: $+0$ 与 -0 的二义性问题

□ 原码的缺陷2: 含负数的情况下无法正常运算

结论: 原码不适合在计算机内表示整数



3.1.3 整数的补码

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	0000000000001010	1111111111110101
-----	------	------------------	------------------

+) 1

1111111111110110

0	0(正)	0000000000000000	0000000000000000
---	------	------------------	------------------

	0(负)	0000000000000000	1111111111111111
--	------	------------------	------------------

+) 1

1 0000000000000000

(高位溢出，舍去)



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
-1	1	0000000000000001	1111111111111110 +) 1 ----- 1111111111111111
-32768	1000000000000000 (1表示 2^{15})	1000000000000000	0111111111111111 +) 1 ----- 1000000000000000
-32767	0111111111111111	0111111111111111	1000000000000000 +) 1 ----- 1000000000000001



3.1.3 整数的补码

2. 补码的基本概念

0 0000000 00000000

0 1111111 11111111 原码理解: $+0 \sim +32767$

补码理解: $+0 \sim +32767$

=====

1 0000000 00000000

1 1111111 11111111 原码理解: $-0 \sim -32767$

补码理解: $-1 \sim -32768$

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



3.1.3 整数的补码

2. 补码的基本概念

✓ 补码：解决 ± 0 的二义性问题，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

问题1：计算机的正数用原码表示，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？



3.1.3 整数的补码

2. 补码的基本概念

✓ 补码：解决 ± 0 的二义性问题，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

问题1：计算机的正数用原码表示，负数用补码表示，该说法是否正确？

答：错误!!!

计算机内整数只有补码表示一种，只是正数的补码与原码一致而已



3.1.3 整数的补码

2. 补码的基本概念

✓ 补码：解决 ± 0 的二义性问题，是计算机内整型数值的真正表示方法

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？

答：错误的问题!!!

不是看到这个二进制整数后，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，再去确定该数的值即：必须通过类型确定值，而不是通过值确定类型!!!



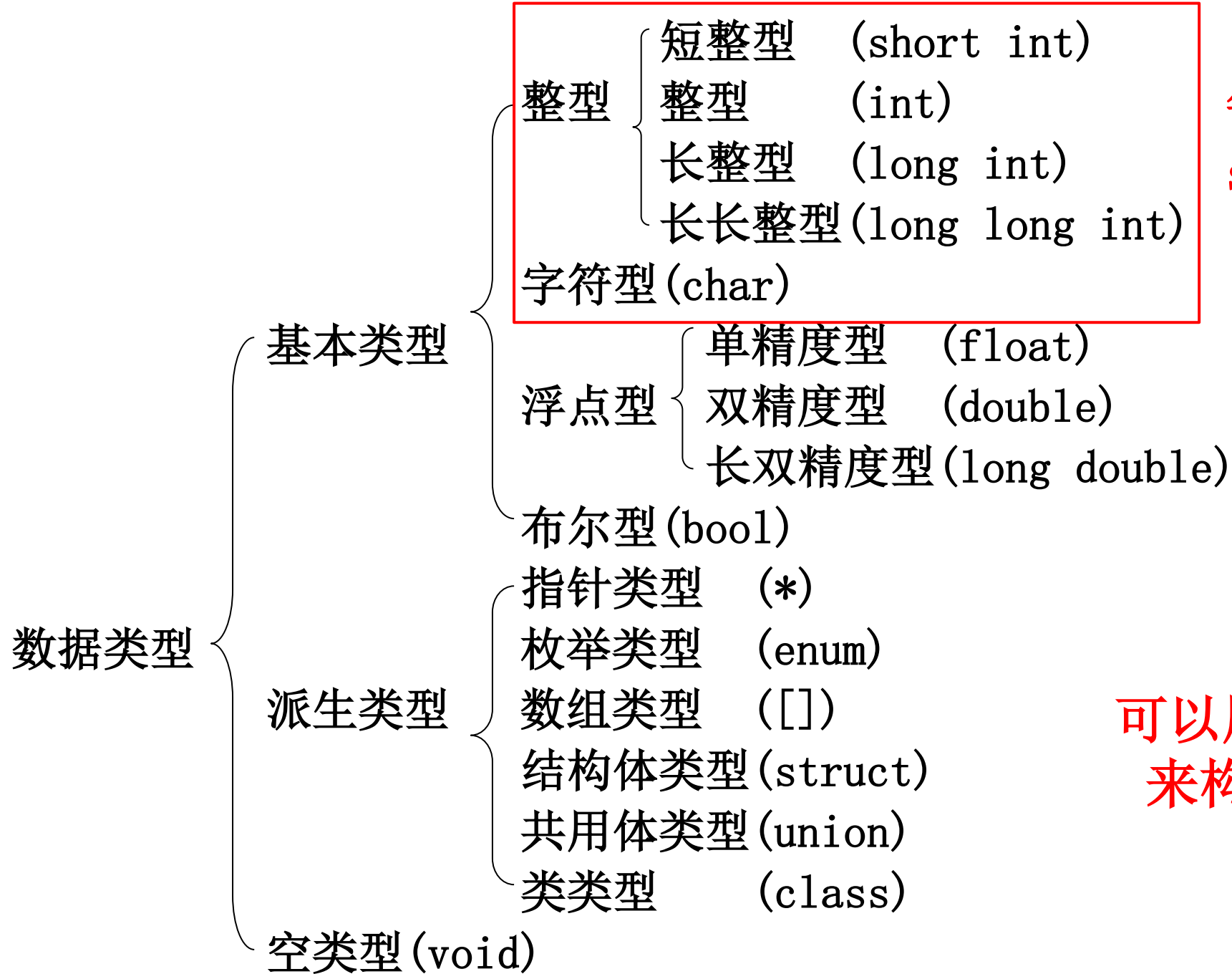
目录

- C++发展背景
- 二进制
- 整数的补码
- C++的数据类型



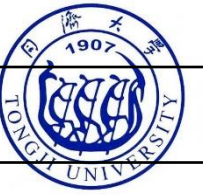
目录

- C++的数据类型
 - C++的数据类型
 - 数据类型所占字节及表示范围



每种int都有
signed与unsigned
(包括char型)

可以用基本数据类型
来构造派生类型



类型	类型标识符	字节	数值范围	
整型	[signed] int	4	$-2^{31} \sim +2^{31}-1$	
无符号整型	unsigned [int]	4	注：[]表示可省略	$0 \sim +2^{32}-1$
短整型	short [int]	2		$-2^{15} \sim +2^{15}-1$
无符号短整型	unsigned short [int]	2		$0 \sim +2^{16}-1$
长整型	long [int]	4		$-2^{31} \sim +2^{31}-1$
无符号长整型	unsigned long [int]	4		$0 \sim +2^{32}-1$
长长整型	long long [int]	8		$-2^{63} \sim +2^{63}-1$
无符号长长整型	unsigned long long [int]	8		$0 \sim +2^{64}-1$
字符型	[signed] char	1		$-2^7 \sim +2^7-1$
无符号字符型	unsigned char	1		$0 \sim +2^8-1$
单精度型	float	4		$-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$
双精度型	double	8		$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$
长双精度型	long double	8		$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$



3.1.4 C++的数据类型

```
#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(float) << endl;
    cout << sizeof(double) << endl;
    return 0;
}
```



注：不同编译器结果可能不同



3.1.4 C++的数据类型

- ✓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, 默认是32位编译系统



3.1.4 C++的数据类型

✓对于整型数，存储为二进制数形式

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



3.1.4 C++的数据类型

- ✓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)，本课程暂不讨论
- ✓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
- ✓浮点型数有有效位数的限定，可能存在一定的误差!!!



总结

• C++发展背景

- 计算机内数据的表示
- 计算机程序设计语言的发展
- C++的发展历史
- C语言的特点
- C++语言的特点

• 二进制

- 进制的基本概念
- 二进制的基本概念
- 八进制、十六进制
- 进制转换

• 整数的补码

- 整型数的符号位
- 补码的基本概念

• C++的数据类型

- C++的数据类型
- 数据类型所占字节及表示范围