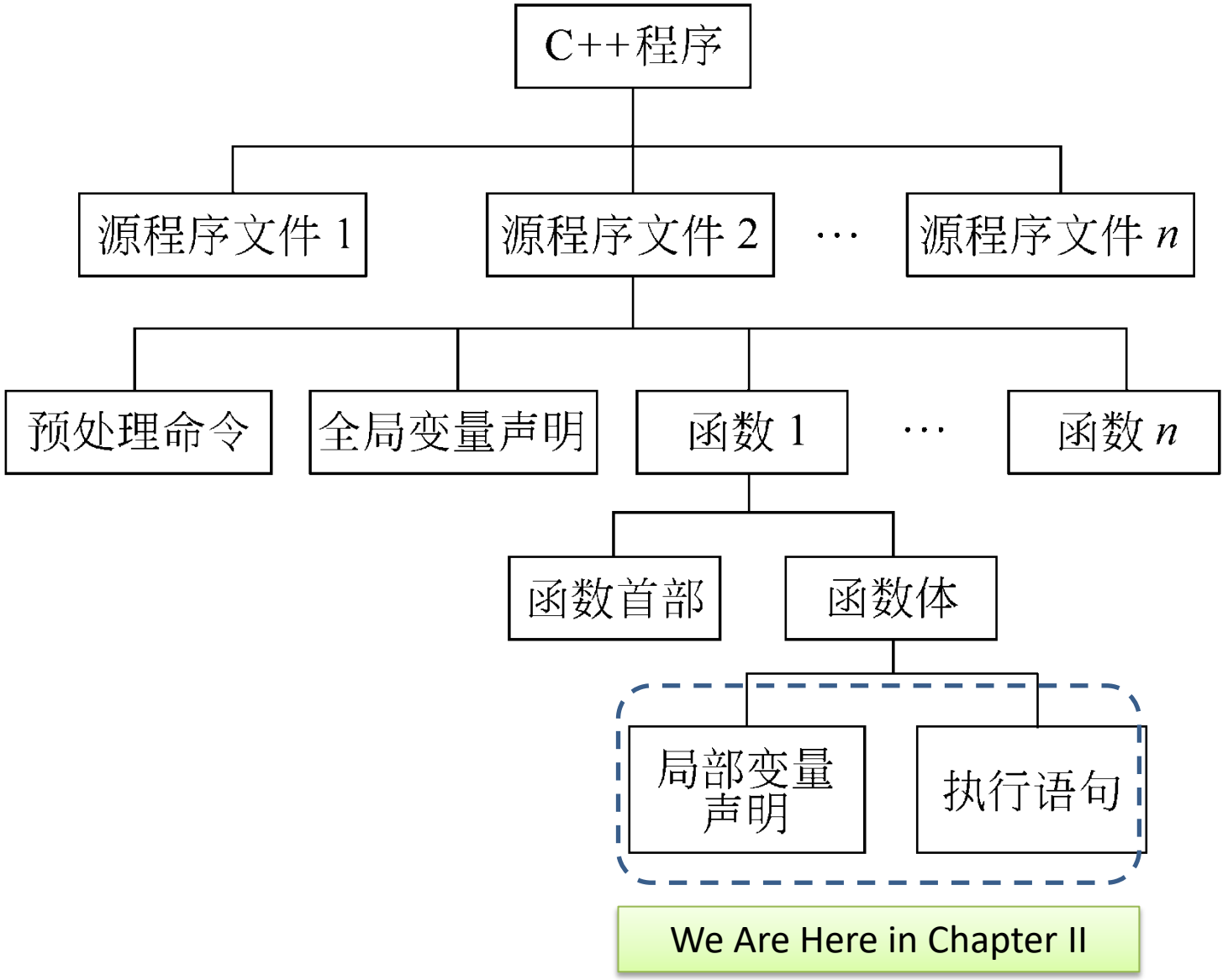
数据类型与语句

C++程序设计

徐延宁 xyn@sdu.edu.cn
数字媒体技术教育部工程研究中心
山东大学软件学院

数据类型与语句



输入十个整数，输出其中最大数值。

```
#include <iostream>
using namespace std;
int main(){
    int a, m;
    cin >> a;
    m =a ;
    for (int j = 1; j < 10; j++){
       cin >> a;
       if (a > max) max = a;
    }
    cout << "max=" << m << endl;
    return 0;
}
```

本章内容达标

数据类型与语句

- 主要内容：
  - **基本数据类型**
  - 运算表达式（类型转换，优先级）
  - 输入输出语句，
  - 分支循环控制流程
- 典型应用：
  - 输入十个数，求和，均值，最大，最小

数据类型

基本数据类型解读

表 2.1　数值型和字符型数据的字节数和数值范围

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

关注1. unsigned 无符号数，非负数

关注2. char是一个字节，不支持汉字

- 关注3. C/C++**规定**，long不短于int, int不短于short，**实际长度由编译器决定**
  - 问题1：long有多长，4？8？ long long有多长，long double有多长，没有准确定义
  - 问题2：先搞清楚你所使用的编译器的基本数据类型的长度

基本数据类型-sizeof（）运算符

- 对于数据类型或者变量，可以使用**sizeof**运算其字节长度。格式为：
  - sizeof（<类型>）或者 sizeof(<变量>) int a; sizeof(a)

sizeof（int） //值为4　　sizeof（float） //值为4

sizeof（double) //值为8　　sizeof（char) //值为1

sizeof(long double) vc 8, mingw 16

- 写一个程序，输出每种你所知道的基本数据类型数据所占空间，顺便输出string

```
cout << "long double\t" << sizeof(long double) << endl;
```

基本数据类型-sizeof（）运算符

```
#include <iostream>
using namespace std;
int main(){
    cout << "char 长度： " << sizeof(char) << endl;
    cout << "bool 长度： " << sizeof(bool) << endl;
    cout << "int 长度： "<< sizeof(int) << endl;
    cout << "unsigned int 长度： "<< sizeof(unsigned int) << endl;
    cout<< “long 长度 ” << sizeof(long) << endl;
    cout<< “long long长度 ” << sizeof(long long) << endl;
    cout << "float 长度： "<< sizeof(float) << endl;
    cout << "double 长度： " << sizeof(double) << endl;
    cout << "long double 长度： " << sizeof(long double) << endl;
    return 0;
}
```

```
char 长度：1
bool 长度：1
int 长度：4
unsigned int 长度：4
long长度 4
long long长度 8
float 长度：4
double 长度：8
long double 长度：16
```

基本数据类型的数值-常量，变量

- 常量：在程序运行过程中，其值一直保持不变的量。
- 整数常量：
  - 十进制：30
  - 八进制、十六进制：前缀：030（24的八进制表示），0x30(48的十六进制表示)
  - 后缀：30l, 30L, 30ul 等价于(unsigned long)30
- 实数常量：
  - 十进制形式： 23.0 24.5 3.56789l （long double）
  - 指数形式： 23E2 145e-2 356789e2
- 编译器根据表面形式（字面值，literal value）来判断常量类型。
  - 30，40 为整型(默认int)，30.0，40.0为浮点型（默认double），

基本数据类型解读-常量

- 字符常量：用一个字节（8位）存储一个字符，表示为 ‘a’ ， ‘D’
- 每个字符对应一个ASCII码，也可以通过单引号括起来的转义符表示该字符；请一定记住下面字符的ASCII码

‘0’	‘\x30’	‘\060’	ASCII码48
‘ ’	‘\x20’	‘\040’	空格， ASCII码32

‘A’	‘\x41’	‘\101’	ASCII码65
‘a’	‘\x61’	‘\141’	ASCII码97

续表

字符形式	含　义	ASCII 代码
\"	双引号字符	34
\0	空字符	0
\ddd	1～3 位八进制数所代表的字符	
\xhh	1～2 位十六进制数所代表的字符	

- 字符常量怎么表示汉字，不支持， ‘我’ 是一个错误，
  - **怎么表示汉字，使用wchar_t，宽字符，常量加L前缀，** L ‘我’ 可以，具体百度，细节有毒

ASCII控制字符

二进制	十进制	十六进制	缩写	可以显示的表示法	名称/意义
0000 0000	0	00	NUL	NUL	空字符（Null）
0000 0001	1	01	SOH	SOH	标题开始
0000 0010	2	02	STX	STX	本文开始
0000 0011	3	03	ETX	ETX	本文结束
0000 0100	4	04	EOT	EOT	传输结束
0000 0101	5	05	ENQ	ENQ	请求
0000 0110	6	06	ACK	ACK	确认回应
0000 0111	7	07	BEL	BEL	响铃
0000 1000	8	08	BS	BS	退格
0000 1001	9	09	HT	HT	水平定位符号
0000 1010	10	0A	LF	LF	换行键
0000 1011	11	0B	VT	VT	垂直定位符号
0000 1100	12	0C	FF	FF	换页键
0000 1101	13	0D	CR	CR	归位键
0000 1110	14	0E	SO	SO	取消变换（Shift out）
0000 1111	15	0F	SI	SI	启用变换（Shift in）
0001 0000	16	10	DLE	DLE	跳出数据通讯
0001 0001	17	11	DC1	DC1	设备控制一（XON 启用软件速度控制）
0001 0010	18	12	DC2	DC2	设备控制二
0001 0011	19	13	DC3	DC3	设备控制三（XOFF 停用软件速度控制）
0001 0100	20	14	DC4	DC4	设备控制四
0001 0101	21	15	NAK	NAK	确认失败回应
0001 0110	22	16	SYN	SYN	同步用暂停

ASCII可显示字符

二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
00	32	20	（空格）(SP)	0100 0000	64	40	@	0110 0000	96	60	`
01	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
10	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
11	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
00	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
01	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
10	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
11	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
00	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
01	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
10	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
11	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
00	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
01	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
10	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
11	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
00	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
01	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
10	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
11	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
00	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
01	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
10	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
11	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
00	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
01	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
10	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
11	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
00	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

基本数据类型解读-常量

- 所有字符都可以用单引号括起来的转义符表示，有些控制字符（回车、换行、分页、缩进）必须要通过转义符表示（表2.2）。很多控制字符用不到了，不需要深究是啥了。牢记**\n \t \\ \' \"**

表 2.2 转义字符及其含义

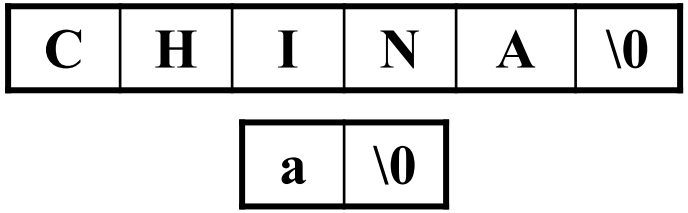
字符形式	含　义	ASCII 代码
\a	响铃	7
\n	换行,将当前位置移到下一行开头	10
\t	水平制表(跳到下一个 tab 位置)	9
\b	退格,将当前位置移到前一列	8
\r	回车,将当前位置移到本行开头	13
\f	换页,将当前位置移到下页开头	12
\ v	竖向跳格	8
\\	反斜杠字符“\”	92
\'	单引号(撇号)字符	39

Alarm
newline
Tab
Backspace
Return

\f在打印机上换页。在屏幕上没有页。

基本数据类型解读-字符与整数

- 字符串常量：用“”界定。
- 对于**标准C**，标准类库没有string，字符串在内存中以字符数组存放，并且以‘\0’结束。字符串的组织方式，开发人员必须要知道。
  - 如：“China”
  - ‘a’在内存中占一个字节， "a"占两个字节



- 而C++ 或者Java中的 String对象，String数据如何组织开发人员不知道。

单选题 10分

设置

下面关于C++源代码中，浮点数1.0的写法，哪一种错误

A 1.0f

B 1.0l

C 1.0

D 1.0ul

提交

填空题 13分

设置

```
cout << sizeof(1) <<sizeof(1l) << sizeof(1ll)<< sizeof(1.0f)  <<
sizeof(1.0)  << sizeof('1') <<sizeof("1") << endl;
```

上面代码的运行结果是什么： [填空1] [填空2] [填空3] [填空4] [填空5] [填空6] [填空7]

作答

变量命名-标识符

- 标识符：作用起名字（变量名、数组名、函数名），通过名字而非地址访问
  1. 以大写字母、小写字母或下划线(_) (以及难忘的$)**开始**。
  2. **后续**可以由大写字母、小写字母、下划线(_)、$或数字0～9组成。
  3. 大写字母和小写字母代表不同的标识符。 （case sensitive）
  4. 不能是C++关键字或操作符。如 int if while + =等。
- 标识符命名应遵循一定规则：
  - 全局变量见字知义，名字不要过短；局部变量，循环控制i，j，k
  - 命名风格统一：匈牙利命名法:iCount，dSum, fScore；Camel命名法（驼峰式）:studentName, numberOfStudent
  - **x_1, x_2,…,x_10,…, x_99, 最差的可读性最，好的加密手段**

基本数据类型解读-变量

- 变量：一个数值内存空间，可以通过名字读写其中的数值。
- C/C++中，变量需要严格遵循定义(分配)、赋值、使用、回收的次序。
- 理解编译器的动作：
  - 定义：类型 名字　int i;
    - 根据类型，分配内存空间，将名字关联内存空间，将来可以通过名字访问该空间
  - 赋值：对内存空间初始化。
  - 使用：获取，或者改变内存空间的数值。
  - 回收：需要理解变量的回收机制
- 变量有一定的**生存周期**与**作用范围**

变量定义的特例-auto（不考）

- 每个变量都要有一个类型，但编译器其实可以根据数值推断类型，
- auto 编译器自动推断变量类型。
  - **auto** ch = ‘x’; // 相当于 **char** ch = ‘x’;
  - **auto ch; ch = ‘x’; //WRONG** ch必须在定义时赋值，
  - 直接 char ch多好，**auto意义何在?**
- 引入auto的原因
  - 配合**泛型**。写代码的时候类型未知，运行代码的时候，根据第一次赋值类型，动态确定变量的类型。
  - **for (auto a: myArray){ cout << a; }**
- ~~C++ 11 之前，auto是c的关键字，有其他含义~~

单选题 10分

设置

下面四段代码输出65的包括：

A

```
auto x;
x = 65;cout << x;
```

B

```
auto x = 'A';
x = 65.9;cout << x;
```

C

```
int v[] = { 6,5 };
for (auto x : v)
    cout << x;
```

D

```
auto x = 65, y = 0.9;
cout << x+y;
```

提交

基本数据类型解读-常变量

- C++使用const定义**常变量(constant variable)**
- 常变量：本质是变量，有名字的内存空间，但约束数值在程序运行生命周期不能改变
  - const float pi=3.1415926; //C++风格
  - Java的做法： final float pi=3.1415926;
  - C语言风格：预编译指令 #define pi 3.1415926，严格意义上不是常变量
  - constexpr float pi=3.1415926; //C++ 11特性，**编译时常量，类似#define**（不考）

单选题 10分

设置

constexpr *vs* const： 编译时赋值（命中注定） vs 运行时赋值

- 关于赋值语句A，B，正确的说法是

A 语法都正确

B 语法都不正确

C A对，B错

D A错，B对

```
int i;
cin >> i;

const int k = i;   //A
constexpr int j = i;   //B
```

提交

单选题 10分

设置

关于赋值语句A，B，正确的说法是

```
int i = 10;
const int k = i;   //A
constexpr int j = 10;   //B
```

A AB都正确

B AB都不正确

C A对，B错

D A错，B对

提交

单选题 10分

设置

左边一段代码输出是什么?

A 6597

B 65a

C Aa

D A97

```
#include <iostream>
using namespace std;
int main() {
    int i = 'A';
    char ch = 'A' + 32;
    cout << i << ch << endl;
    return 0;
}
```

提交

运算符

- 算术运算符　　+　-　*　/　%(求余)
- 赋值运算符　　=　　+=　　-=
- 关系运算符　　<　>　<=　>=　　==　　!=
- 逻辑运算符　　与&&　　或||　　非!
- 逗号运算符
- 其他运算符　　例如位运算符

运算符-自增、自减运算符

- 自增、自减运算和Java相同 i++, ++i, i--, --i
  - i的类型应当是整数，但浮点数也不算错
  - `float fa= 1.5; cout << --fa << endl;    // 0.5`

- 自增自减运算符作用于变量

3++　　(x+y)++　　(−i)++

- 注意 ++在前在后的差别，++在前, 先++,后参与运算；++在后，先参与运算，后++

```
int i = 3, j = ++i * 5;
```
j 20

```
int i = 3, j = i++ * 5;
```
j 15

```
int i = 3, j = i++ + i;    //  3 + ; i++ ; 3 + 4 = 7
```

关系表达式与真假值

- 在C++语言中，字符、bool 转换为整数参与算数运算
- 用非0代表真，用0表示假，例如 if (3+2) 成立，
  - if (x = 4-2*2) <=> x= 4-2*2; if(0); 不成立； Java语言中，语法错误
- 关系表达式的结果以及布尔变量的值只有两个，真为1，假为0，例如
  - bool b = 3 + 4 < 5 < 1; \\ 7 < 5 < 1 0 < 1 最后 b = 1
- 布尔变量与布尔常量(true/false)占1个字节，true为1，false为0，

关系表达式与真假值

- 课堂难度（求表达式的值）

已知

int a=2　b=3　c=4

b=='a'+1	0
c-a==a	1
(a==2) + 1	2

- 考试难度（求表达式的值）

```
bool flag = 2025; // flag 1
cout <<(flag > 2020) + flag * 2 + bool(4) << endl;
```

0	2	1

要点

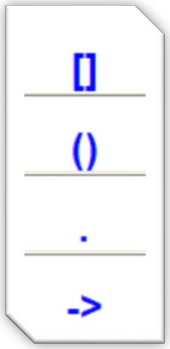
- 混合表达式（考试实际难度低）
  - 优先级
  - 类型转换与溢出
- 位运算（不考）
- 输入输出（考试实际难度低）

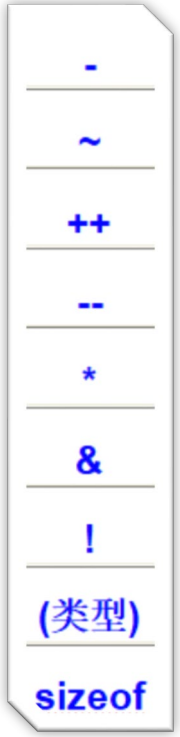
优先级

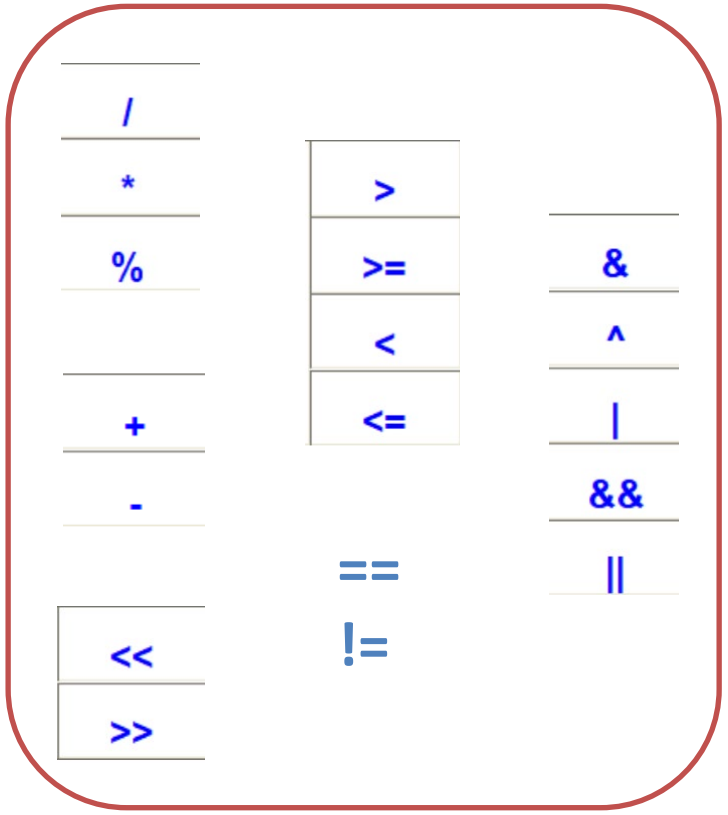
- **分析**表达式时，从左往右看；
- **计算**表达式时，先算**优先级**高的；
- 若是同级，看**结合**方向，
  - 括号、双目、逗号从左到右；单目，三目，赋值从右到左；
- 由于小括号优先级最高，可以用小括号改变（明确）优先级；

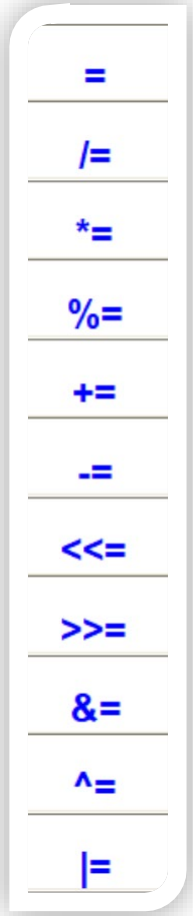
表达式

- 优先级：括号>单目>**双目**>三目>赋值>逗号









同级从左到右
a = 3, a = 3+4

同级从左到右
My.birthday.month =3;
i = 3;
a[i++][i++]=10;
a[3][4]= 10;

同级从右到左运算
+-+-a;
!!-a;
p++;等价(p++)
(int)a[6]

- 同级从左到右 a+b-c
- 四则运算高于关系运算，高于逻辑运算
- << >> & ^ |是位运算

同级从右到左
int a = b= c = 3+1;

逗号运算符和逗号表达式

- 表达式1，表达式2，表达式3，…，表达式n
- 整个表达式从左到右求解，结果为最后一个表达式的值，
- 逗号表达式 优先级最低（比赋值还要低）。

a=(3+4, 5*6, 2+1);　　a=3

a=3*3, a+6, a+7;　　a=9　　表达式结果 16

(a=3*5, a*4), a+5　　a=15　　20

- 逗号表达式的作用：
  - 除了考试，平时用到的较少：a=3, b = 4;　　v.s.　　a = 3; b= 4;
  - 变量定义赋值　　**int a = 3, b = 4;**
  - 某些语法限定　**for(i=0,j=0; i<100; i++,j++)**

优先级

•编译优化1：局部规约，非全局规约 a + b + c * d; 先算 a+b 再算 c*d

•编译优化2：使用逻辑与&&和逻辑或||，短路。是否执行编译优化，可能影响运行结果

已知： x=4 y=5

i= ++x= =5 || ++y= =6

x=5 y=5 i=1 （短路）

i= x++= =5&& y++= =6

x=5 y=5 i=0 （短路）

++的进一步说明，假设 int a = 3;

- 规则1：++a 可以作为左值，a++ 不能作为左值。
  - ++a = 5；　// ++a出现在=左边，正确　a = 5;
  - a++ = 5;　//a++作为左值，语法错误
- 规则2：从左到右解析，同级从右到左计算。例如:
  - **+a, ++a, ~~+++a,~~ ++++a, ~~+++++a~~**
  - +++a, 编译器视角，++(+a)=》++(3)
    - 注意 a 与 +a的区别。a是变量，可以作为左值，a = 3;
    - +a是计算结果，不能作为左值，即不能有 +a = 3;
    - 可以有 ++a，不能有 ++(+a)
  - ++++a，理解为++(++a)　正确；　+++++a，理解为 ++(++(+a))错误

++的进一步说明，假设 int a = 3;

- 规则3：a++（后加）优先级高，++a（先加）与+a（单目+）同级。例如:
  - –++a++ 后加高于先加，所以理解为 ++（a++），又因为a++不能作为左值，所以语法错误
  - –(++a)++ （）优先级最高，先有++a，又因为++a可以作为左值，因此可以再++;
  - –b=-++a，-与前加优先级相同，自右向左结合，等价于-（++a），结果a 4, b -4;
  - –b=-a++，-低于后加，等价于 b=-(a++)，a++表达式结果是3，所以 b = -3，而a = 4;
  - –b=a+++a 后加高于+，所以理解为 （a++）+a ； a++表达式结果是3，所以理解为3+a;
    a++高于3+a，所以先有 a = 4，后有b=7
  - –b = -a + a++；先a++, 结果为3，a为4；b = -a + 3 = -4 + 3 = -1;
  - –a++++为什么错？？

填空题 2分

设置

```
int a = 3, b = 2;
b*= ++a+=a++*a << 1 + 1;
```

执行完上述代码后，a,b的数值各是多少

[填空1] [填空2]

正常使用填空题需3.0以上版本雨课堂

作答

多选题 10分

设置

已知 c=4，分别执行表达式1,2,3，正确的结果描述包括

1. (c=1)&&(c=3)&&(c=5)
2. (c==1)||(c=2) || (c=5)
3. (c!=2) && (c!=4) &&(c>=1)&&(c<=5)

A c的数值分别是5，2，4

B 表达式的结果分别是1，1，0

提交

填空题 8分

设置

已知， int i = 2 ,j = 2; 下列语句中表达式中i, j的值各为多少

i=3, (j++)+i ; [填空1] [填空2]

i*=j+=3; [填空3] [填空4]

j=i=((i=3)*2); [填空5] [填空6]

已知， int i; double d; 赋值后 i, d 的数值各为多少

d = i = 1.5; [填空7] [填空8]

正常使用填空题需3.0以上版本雨课堂

作答

混合运算

- 运算时，根据当前处理的运算符，决定运算数转换的类型。转换会根据原有数据生成临时数据参与运算，原有数据不变。
  - 10+'a'+d-87.65*'b'
- 运算符两边都是整数的运算为整数运算，结果为整数　1 / 3
- 数据类型转换（以及运算）可能会导致两类问题：溢出，精度损失
  - char ch = 3.1415926 * 100;
- C/C++如何避免溢出？-**开发者自己保证转换安全**

混合运算-类型转换

- C++的转换语法不像Java那么严格，且整数还包括unsigned，**每一次数据转换都是有风险的**-开发者自己保证转换安全
- 有符号转无符号，需要开发者保证是有符号数是正数。
  - unsigned int b= -1 ; cout << b<< hex << b ; //
  - 4294967295ffffffff //4294967295是最大的无符号整数，其十六进制(hex)形式为FF FF FF FF
- 无符号转有符号，需要开发者保证不越界
  - unsigned int ch1 = 0x8fffffff;
  - int x = ch1; // 越界，但依然有结果，x为负数
- 多字节数据类型向少字节数据类型转换，需要开发者保证不越界
  - `int x = 1234567;`
  - `short y = x;`
- double可以自动转换出float，浮点数自动/隐式转换得到整数，但自由（不强制使用类型转换）的代价是风险增大　　int x = 1.7; // **居然可以**

混合运算-整数类型转换

- 两个兼容的不同类型的值相加，结果自动转为更大的数（正数）的类型。
  - short+long， 结果要转为long；
  - unsigned+signed，结果要转为unsigned

```
unsigned int a = 6;
int b = -7, c = -3;
cout<< a+b<< endl;
cout << a+c << endl;
cout << -7 + a << endl;
cout << -6 + a << endl;
cout << -7 + 6 << endl;
```

```
4294967295
3
4294967295
0
-1
```

单选题 1分

设置

```
unsigned int a = 6;
int b = -20, c = -3;
cout<< ((a+b)>6)<< ((a+c)==3);
```

其输出结果为

A 11

B 01

C 10

D 00

// 2023年C++程序设计第一题

提交

混合运算-强制类型转换

- C++支持的强制类型转化语法　（类型名）（表达式）

1. Java风格，C风格:　　**(double) a**　　**(int) (x+y)**　　**(int) 6.2%4**
2. **C++风格，类型作为函数名，()作为参数：double(a)， int(x+y) int(6) %4 ，**
3. 复杂风格，引入注目，突出这儿有类型转换：

– int ii = static_cast<int>(5.5); //等效于　int ii = 5.5;　等效于 int ii = int(5.5);

static_cast <type-id> (expression)

- 在强制类型运算后原变量不变，但得到一个所需类型的中间变量。

位运算(不要求)

- 用一个字节来存储多个（最多8个）信息，表示一个事物的多种状态。
- 用途（位运算不是存储匮乏年代的老古董）：
  - 文件的读权限，写权限，执行权限，分别用0001，0010，0100表示，则0101表示可读可执行，不能写（修改）。
  - 编码系统，按位编码：身份证、学号、车牌号(鲁A NB001)，不同位代表不同含义
  - 图像、视频编码的基础
  - 计算机组成原理，数字逻辑、汇编等课程的基础知识
- 实际应用中，考虑效率，通常用unsigned int定义单个编码，将其看做32个0/1位（bit）进行的运算；或者 unsigned char[]定义大量编码。例如
  - unsigned char img[1920][1080][4]，定义一个1920*1080像素的图像，每个像素点由RGBA信息构成

位运算(不要求)

- 典型的位运算包括：
  - 运算符 ~ 取反， ^位异或， & 位乘， |位加， <<左移，>>右移

表 4.3：位运算符（左结合律）

运算符	功能	用法
~	位求反	~ expr
<<	左移	expr1 << expr2
>>	右移	expr1 >> expr2
&	位与	expr & expr
^	位异或	expr ^ expr
|	位或	expr | expr

- 移位运算

```
unsigned char bits = 0233;
```

1	0	0	1	1	0	1	1

```
bits << 8 // bits 提升成 int 类型，然后向左移动 8 位
```

0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 1 1 0 1 1	0 0 0 0 0 0 0 0

```
bits << 31 // 向左移动 31 位，左边超出边界的位丢弃掉了
```

1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

```
bits >> 3 // 向右移动 3 位，最右边的 3 位丢弃掉了
```

0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 0 1 1

位运算

- 位取反

```
unsigned char bits = 0227;
```

1 0 0 1 0 1 1 1

~bits

1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 1 1 0 1 0 0 0

- 位运算(双目运算):
  - 与&
  - 或|
  - 异或^
  - cout << (2 | 3) << (2 || 3) << endl; // 31
  - 异或: 半加（无符号加），0 ^ 0 = 0, 1 ^ 1 = 0, 0 ^ 1 = 1, 1 ^ 0 = 1
  - 性质: X ^ 0 = X X ^ X = 0 X ^ Y = Y ^ X X ^ Y ^ Z = X ^ (Y ^ Z)

```
unsigned char b1 = 0145;
unsigned char b2 = 0257;
```

		0 1 1 0 0 1 0 1
		1 0 1 0 1 1 1 1
b1 & b2	*24个高阶位都是0*	0 0 1 0 0 1 0 1
b1 | b2	*24个高阶位都是0*	1 1 1 0 1 1 1 1
b1 ^ b2	*24个高阶位都是0*	1 1 0 0 1 0 1 0

推论: x ^ y ^ y -> x ^ (y ^ y) -> x ^ 0 -> x

- 典型计算：
  - 使用异或擦写

    x ^ y ^ y ->x 用y涂改x两次，第二次涂改相当于还原

  - Swap(x,y)而不用tmp(中间变量) tmp = x; x = y; y = tmp;

    x ^= y;
    y ^= x;　　Y = Y ^ X = Y ^ (X ^ Y) = X ^ Y ^ Y = X
    x ^= y;　　X = X ^ Y = (X ^ Y) ^ X = Y ^ X ^ X = Y

  - 取x的第k位（从0开始）：if (x >> k &1) 或者 if (x & 1<<k)
  - 快速乘法，位移快于乘法。
  - y = x * 10;　　？？　　y = (x << 3) + (x << 1);

- 位运算诸多优点：节省内存，运算速度快
- ICS(Introduction to Computer Sciences，计算机科学导论)课程，其Datalab实验（可以百度）的一个主题：**只能利用位运算与加法运算（廉价的运算单元），**实现若干复杂运算或者函数：
  - **实现减法，不必设计一套减法运算单元，x-y => x+(-y) 如何由y得到-y，补码：各位取反，末位加1**
  - **int isGreater(int x, int y)** 功能：当 x > y 时，返回1，否则返回0
  - **int isPower2(int x)** 功能：判断x是否恰好等于 2^n，如果等于则返回1，否则返回0

- **位运算**缺点是太专业，过于底层，可读性不强，不够直观。

- 数据类型

- 语句

- 空语句 –只包含一个语句结束符 “;”
- 声明语句
  - 例如：变量声明 int i;
- 表达式语句
  - 在表达式末尾添加语句结束符。例如： a=3; z = f(x)+g(x,y); f(x);
  - 3; 3+5;没有意义但语法正确
- 流程控制语句(与Java相同)
  - 选择语句、循环语句、跳转语句 goto, break, continue, return
- 标号语句:配合 goto, break, continue使用
  - 在语句前附加标号，通常用来与跳转语句配合 watchHere: x = x + 1;
- 复合语句（与Java相同）：用“{}”括起来的多条语句

输入输出

- 在C++中，输入输出通过输入输出流来实现。
- cin和cout是预定义的流类对象，其定义位于头文件iostream中。
  - cin用来处理标准输入，即键盘输入。**对应c语言scanf getxxx**
  - cout用来处理标准输出，即屏幕输出。**对应c语言printf putxxx**
  - #include <iostream>
- 键盘输入：cin >> （提取运算符）。
  - cin >> 变量1 >>变量2…;
  - 提取符可以连续写多个，每个后面跟变量，输入时用空白符间隔。
- 键盘输出：cout << <表达式> << <表达式>

```
int a, b;
cin >> a >> b;
cout << “a +b =”<< a + b << endl
```

输入

- 在缺省的情况下，cin自动跳过输入的空白符（空格，TAB，Return），
- 使用函数cin.get() 读空格，使用cin.getline()读取一行
- 如果希望按照八进制、十六进制解释读到的数据。。。 cin>>hex>>i;
- 输入数据的格式、个数和类型与cin中所列举的变量类型一一对应
- 如果不对应，后果未定义，不知道错哪儿比异常更可怕

```
float  f;
int  i1,i2;
char  ch1,ch2;
cin>>i1>>f>>i2>>ch1>>ch2;
```

输入：34 5.678 1a c<CR>

i1:34 f:5.678 i2:1 ch1:a ch2:c

格式化输出

- cout可以搭配控制符(manipulators)，实现**格式化**输出，对应Java ***Format
  - 数据的进制，宽度，对齐方式，小数点位数；
  - cout<<setfill('*');
  - cout <<setw(3)<< 4<<setiosflags(ios::left) <<12<<setw(4)<< 4 <<endl;
  - 用*填充，字符位置宽度（每次都要设置），对齐方式（更改时设置）

4124*

- 头文件<iomanip>定义了manipulators，通常需要#include到你的源代码中
- 卷面考试不要求（大量记忆），但PTA作业要用到，实践中要用到，格式问题是脸面问题，很重要

常用输出操纵符

- 输出十进制、Hex 十六进制、oct 八进制整数；
  - cout<<12<<hex<<12<<oct<<12<<12;　　12c1414
- 对于浮点数（float、double和long double），格式包括科学计数法输出，有效数字位数等
  - cout << setprecision (5) << 12.34567 << endl; // 12.236
- 除了通过插入操作符进行输出外，也可以用ostream类提供函数输出，例如：

```
//输出一个字节。
cout.put('A');
```

```
//输出info数组的n个字节。
char info[n]; ……
cout.write(info,n);
```

格式化输出

表 3.1　标准输入输出流的控制符

控　制　符	作　　用
dec	设置数值的基数为 10
hex	设置数值的基数为 16
oct	设置数值的基数为 8
setfill(c)	设置填充字符 c,c 可以是字符常量或字符变量
setprecision(n)	设置浮点数的精度为 n 位。在以一般十进制小数形式输出时,n 代表有效数字。在以 fixed(固定小数位数)形式和 scientific(指数)形式输出时,n 为小数位数
setw(n)	设置字段宽度为 n 位
setiosflags(ios::fixed)	设置浮点数以固定的小数位数显示
setiosflags(ios::scientific)	设置浮点数以科学记数法(即指数形式)显示
setiosflags(ios::left)	输出数据左对齐
setiosflags(ios::right)	输出数据右对齐
setiosflags(ios::skipws)	忽略前导的空格
setiosflags(ios::uppercase)	数据以十六进制形式输出时字母以大写表示
setiosflags(ios::lowercase)	数据以十六进制形式输出时字母以小写表示
setiosflags(ios::showpos)	输出正数时给出“+”号


金宝汇商业管理公司(华研超市店)
票号0002326 机号2031 工号1005
交易时间020-11-25 16:11:21
2135812028060 1.76 2.806 4.94
红薯
2151143035065 0.96 3.506 3.37
土豆
2151143036925 0.96 3.692 3.54
土豆
2151143031241 0.96 3.124 3.00
土豆
金额: 14.85件数: 4
折扣: 0.00找零: 0.15
人民币 15.00
商品质量问题七日内退换
生鲜食品八小时内退换
客服电话:8118259
购物卡、团购电话:8118259
此单为唯一购物凭证发票当日索取
关注消防 生命至上
东胜区消防提醒您:冬季火灾易发

控制流程语句

- 与Java完全相同

- 分支：
  - if else
  - ？ ： （三目运算）
  - switch case break default

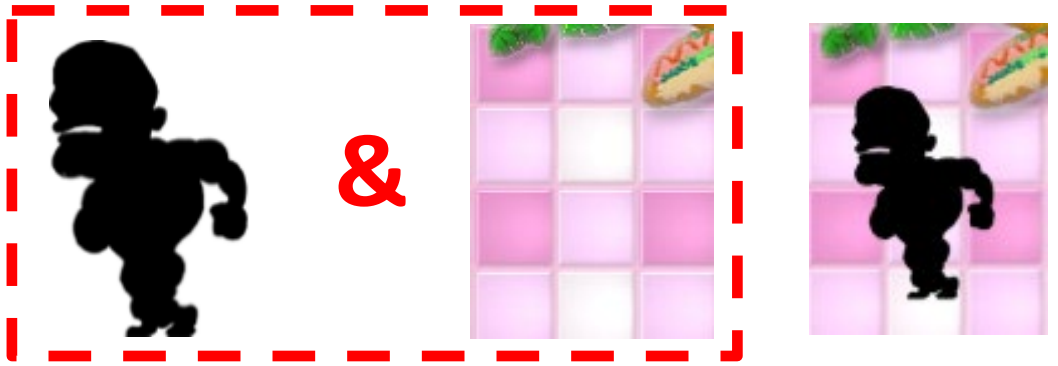
- 循环：
  - while
  - do{} while
  - for
  - continue
  - break

课程安排

- 简单C++程序：数据类型与语句；
- Next：**函数**
- 数组与指针
- 自定义数据类型
- 类和对象使用
- 运算符重载
- 继承和派生
- 模板与容器

位运算

- 位运算做PS中的图层叠加，用0，1组成的蒙版图像，区分全景背景



0 & X = 0

1 & X = X

0 | X = X

1 | X = 1

数据类型

- 关于unsigned int，位运算，16进制那些事
- unsigned一个用途是编码（Unicode），习惯上从0开始编码，编码不用负数
- unsigned另一个用途是位运算，例如
  - 1110 0010
  - & 1101 0011
  - 11000010
- 所有int(float)都是二进制编码的，普通int，最高位表示正负符合，并且负数采用补码表示，所以不适合位运算
- Cpp规范不支持二进制输入，太长了，10进制不够直观，所以用16进制或者8进制常量
  - 0xe2 & 0xd3 =

投票 最多可选1项

设置

C++开发环境，我选择了

A Visual Studio C++

B VSCode + w64devkit

C Code::Blocks 或者DevC++

D JetBrains的CLion

E Xcode或者其他开发环境

F 我还没有开始或者没有成功编写HelloWorld

提交

混合运算-类型转换

- 计算机中表示有符号的整数，通常用原码表示正整数，用补码表示负整数，用最高位的0表示整数，最高位的1表示负数，例如：8为二进制表示
  - 0 001 1111 正数，1*16 + 15 = 31
  - 1 001 1111 负数，是不是 -31呢，**不是**
- 为了用一套加法电路可以实现统一实现加、减运算（体系结构（组成原理）知识），对负数引入了补码表示
- 负数的补码由源码变换得到：
  - 对应原码的最高位（符号）=1，（原本最高位=1的正数溢出）
  - 原码其余各位取反，末位+1，
- 例如求-1的补码表示：
  - 1 的原码 0000 0000 0000 0001
  - 原码最高位置1 **1** 000 0000 0000 0001
  - 各位取反 **1** 111 1111 1111 1110
  - 末位+1 1 111 1111 1111 1111 对应十六进制 FFFF
- -10的补码
  - 0000 0000 0000 0110 =》 1000 0000 0000 0110 =》1111 1111 1111 1001 =》 1111 1111 1111 1010

混合运算-类型转换

- 已知
  - unsigned int b= -1 ; cout <<\n<< hex << b ; //
  - 输出为
  - 4294967295
  - ffffffff
- 下面代码
  - unsigned int b= -3 ; cout << b<<\n<< hex << b ; //
- 输出为?
- 3 对应 0 000 0000 0000 0011
- 其补码为 1 111 1111 1111 1101

为啥要补码表示？转补码后，用一套加法电路可以实现加、减运算。体系结构（组成原理）课程知识