

We'll have more choices.



C程序设计

T03



数据类型

厦门大学信息学院软件工程系

黄炜 副教授

主要内容

- 常量和变量
- 常数的表示
- 数据类型
 - 整型
 - 浮点型
- 类型的选用
- 常见的编程错误



1. 变量和常量



```
/* platinum.c  -- your weight in platinum */
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

新的数据类型：浮点型

```
float weight; /* user weight */
```

```
float value; /* platinum equivalent */
```

```
printf("Are you worth your weight in platinum?\n");
```

```
printf("Let's check it out.\n");
```

```
printf("Please enter your weight in pounds: ");
```

对普通变量，此处需要&号

```
/* get input from the user */
```

```
scanf("%f", &weight);
```

此处程序暂停，等待用户键盘输入，直到按下回车键 (Enter) 继续

```
/* assume platinum is $1700 per ounce */
```

```
/* 14.5833 converts pounds to ounces */
```

表示常数的新方法：
带小数点表示浮点型

```
value = 1700.0 * weight * 14.5833;
```

一个程序应该有0个、1个或者多个输入



```
printf("Your weight in platinum is worth $%.2f.\n",  
value);  
printf("You are easily worth that! If platinum prices  
drop,\n");  
printf("eat more to maintain  
return 0;  
}
```

打印常数的新方法：printf()中的
%f用于处理浮点数，%.2f只输出2位
有效小数，更加美观。

Are you worth your weight in platinum?
Let's check it out.

Please enter your weight in pounds: 156↓

Your weight in platinum is worth \$3867491.25.

You are easily worth that! If platinum prices drop,
eat more to maintain your value.

蓝色下划线表示需要键盘
输入，↓表示敲击回车键



常量和变量

- 常量变量的划分

- 在程序流程中，一些量的值是不可改变的

- 例如： $\pi=3.14$ 是不能随便改动的

- 一些量的值随着运行需要改变

- 例如：计算1到10的和，指示当前加数的量应不断变化

- 定义

- 常量（ **Constants** ）：运行过程中保持值不变的量

- 变量（ **Variable** ）：运行过程中值可以改变的量



常量和变量

- 常量和变量的名字

- 常量和变量应有唯一的、有意义的名字

- 常量和变量的值

- 用于参与计算，指示其当前值或被改变值

```
value = 1700.0 * weight * 14.5833;
```

1. 取出weight的值，假设为5；
2. 计算 $1700.0 * 5 * 14.5833$ 的值，为123958.05；
3. 改变value的值为赋值号右边的值123958.05。



常量和变量

- 常量 (Constants) : 运行过程中保持值不变的量
 - 有经验的程序员避免直接使用有特殊物理意义的常数
 - 宏和const常量的名字一般不使用小写字母

分类	示例	格式
常数	14.5833	整数、实数、字符、字符串，或其其它进制。
宏定义	#define WEIGHT 14.5833	预编译指令#define，宏名，宏的展开
常量	const float X = 1700.0;	关键字const，数据类型，变量名，赋值符号，值。



常量和变量

- 变量 (Variable) : 运行过程中值可以改变的量

改变方式	示例	格式
通过赋值 改变值	<code>value = 1700.0 * weight * 14.5833;</code>	整数、实数、字符、字符串，或其其它进制。
通过写内存 改变值	<code>scanf("%f", &weight);</code>	需要调用scanf和memset等函数改变内存值



常量和变量的操作

- 声明

- 指明名称所对应变量（或常量）的数据类型

- 赋值

- 改变变量的值（只能在声明时对常量改变值）

- 取值

- 读取变量（或常量）的值



一般变量的声明

• 变量声明的格式

分类	格式	示例
声明一个变量	<类型> <变量名>;	<code>int var;</code>
声明多个变量	<类型> <变量名>, <变量名2>;	<code>int grade, class, x;</code>
声明时赋值	<类型> <变量名>=<表达式>;	<code>int grade, class = 2;</code>

• 注意事项

– 常量和变量必须**先声明后使用**，变量必须**先赋值再使用**

▪ **初学者应养成声明时赋值的好习惯**

– 传统C要求声明应集中在语句块最开头，C99和C11要求声明在使用之前



一般变量的赋值

- 变量赋值的格式

分类	格式	示例
用表达式赋值一个变量	<变量名>=<表达式>;	var = 65;

- 注意事项

- 赋值语句应以分号结尾，赋值表达式不能以分号结尾
- 表达式先求值，再赋值给变量
- 赋值后变量的值为表达式的值



一般变量的使用

• 变量使用的格式

分类	格式	示例
变量是表达式	<变量名>	<code>a = var;</code>
变量可以组成表达式	由<变量名>和<操作符> 组成	<code>a = x + 1;</code>

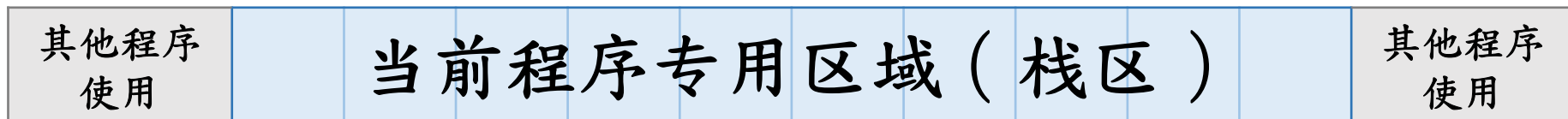
• 注意事项

- 变量是表达式，因而变量有值
- 所有的变量必须**先声明再赋值后使用**，否则结果不可控
- 变量的使用范围应同时满足的条件
 - 声明语句所在语句体中且在声明语句之后的语句



一般变量相关的内存变化

- 原始的内存空间



- 变量声明 (`int a;`) 时，在内存开辟四字节空间
a所在区域



- 变量赋值 (`a=1;`) 时，改变内存内容



- 变量求值 (`a`) 时，其值为该变量内存区域的内容 (1)



修改const常量产生编译错误

- 标记const的常量在内存开辟空间，存储其值
- 尝试修改const常量将产生编译错误

黄色底色的方框中是
错误的示例代码

```
const float mult;  
mult = 23 * weight * 14.5833;
```

— 编译时，将出现错误：赋值左边不可为常量

```
const.c: In function 'main':  
const.c:9:5: error: assignment of read-only variable 'mult'  
    mult = 23;  
    ^
```

绿色底色的方框中是
编译器的输出信息

- 对#define常量或数值常量赋值也将产生编译错误
- 宏是简单替换，不因宏定义而产生额外的空间




```

/* pizza.c -- uses defined constants in a pizza context */
#include <stdio.h>
#define PI 3.14159
int main(void)
{
    float area, circum, radius;

    printf("What is the radius of your pizza?\n");
    scanf("%f", &radius);
    area = PI * radius * radius;
    circum = 2.0 * PI * radius;
    printf("Your basic pizza parameters are as follows:\n");
    printf("circumference = %1.2f, area = %1.2f\n", circum,
area);
    return 0;
}

```

有经验的程序员用宏定义来表示常数，明确常量的物理意义

What is the radius of your pizza?

35

Your basic pizza parameters are as follows:
circumference = 219.91, area = 3848.45



预定义为#define的常量

- 格式 `#define NAME value // comments`
- 行为：预编译器替换所有关键字，再交给编译器编译
 - 文中除注释和字符串外所有完整的NAME都替换为value

使用宏定义的语句	宏展开的语句
<code>circum = 2.0 * PI * radius;</code>	<code>circum = 2.0 * 3.14159 * radius;</code>
<code>"PI is 3.14"</code>	<code>"PI is 3.14"</code>
<code>area = PIs * radius * radius;</code>	<code>area = PIs * radius * radius;</code>
宏定义	<code>s = PI * r * r;</code> 宏展开的语句
<code>#define PI 3.1415</code>	<code>s = 3.1415 * r * r;</code>
<code>#define PI 3 + 0.1415</code>	<code>s = 3 + 0.1415 * r * r;</code>
<code>#define PI = 3.1415</code>	<code>s = = 3.1415 * r * r;</code>



2. 常数的表示



整数常数的表示

- 格式 (大小写无关) [正负号]<十/八/十六进制常数>[整数后缀]

正负号	可省略	示例 (106)	
正号 (默认)	是	+106 或 106	
负号	否	-106	
进制序列	前缀	数字字符集	示例 (106)
十进制	无	0123456789	106
八进制	0	01234567	0152
十六进制	0x	0123456789ABCDEF	0x6a
后缀类型	后缀	示例 (106)	
长型 (默认)	l	106l	
无符号	u	106u	
64位	ll (VS里是i64)	106ll	

常量的数据类型影响着表达式的计算结果



字符常数的表示

- 字符常数（大小写有关） `'<字符序列>'`

- 字符常数的类型是整型，而不是字符型
- 字符序列是指：除单引号（'）、反斜杠（\）或者换行符以外的所有源字符集成员，或者转义序列

转义序列举例	格式	示例
简单转义序列	反斜杠（\），非x小写字母或标点	<code>\a \b \f \n \r \t \v \' \" \\ \?</code>
八进制转义序列	反斜杠（\），1~3位八进制数（01234567）	<code>\101</code>
十六进制转义序列	反斜杠（\），小写字母x，十六进制数（0123456789ABCDEF）	<code>\x41</code>



整数常数的表示形式

• 整数（含字符）常数的主要表示形式

赋值方法	示例	值	特点
十进制写法	var=65;	65	不以0开头
八进制写法	var=0101;	65	以0开头，但不紧接x或X，后续数字是0~7。
十六进制写法	var=0x20;	32	以0x或0X开头，后续数字是0~9，A~F。
可见字符	var='A';	65	以单引号为界，中间只有一个字符
转义序列字符	var='\b';	9	以单引号为界，中间以反斜杠开头，跟着一个字母
八进制字符	var='\101';	65	以单引号为界，中间以反斜杠开头，但不紧接x或X，后续数字是0~7。
十六进制字符	var='\x20';	32	以单引号为界，中间以\x或\x开头，后续数字是0~9，A~F。



实数常数的表示

- 格式（大小写无关） $\langle \text{正负号} \rangle \langle \text{小数常数} \rangle [\text{指数}] [\text{浮点后缀}]$

小数常数序列	示例	值
数字序列+小数点+数字序列	30.106	30.106
小数点+数字序列	.106	0.106
数字序列+小数点	106.	106.0

指数部分序列	示例	值
e（大小写）+正负号（默认正）+数字序列	1.5E-2	0.015

浮点后缀	含义	示例（106）
l（默认）	double类型	-10.6l
f	float类型	10.6f



```
/* escape.c -- uses escape characters */
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    float salary;
```

```
    printf("\aEnter your desired monthly salary:"); /* 1 */
```

```
    printf(" $_____ \b\b\b\b\b\b\b\b"); /* 2 */
```

```
    scanf("%f", &salary);
```

```
    printf("\n\t$%.2f a month is $%.2f a year.\n", salary * 12.0); /* 3 */
```

```
    printf("\rGee!\n"); /* 4
```

利用转义序列
回退光标

光标在此

Enter your desired monthly salary: \$

```
return 0;
```

```
}
```

Enter your desired monthly salary: \$123

打印到此

Gee! \$123.00 a month is \$1476.00 a year.



整数常数的原则

- 书写原则

- 一般不加前后缀，除非不加前后缀会出现错误

- 前缀： $-3 < -2$ （真）； $3 < 2$ （假）

- 后缀： $65536ul * 65536ul > 0$ （真）； $65536 * 65536 > 0$ （假）

- 尊重书写习惯

- 小数点前的0不建议省略

- 小数不建议科学计数法，前导0多的小数应用科学计数法

- 幂数的e/E大小写都可以，但是应全篇统一



3. 整型



整型分类

- 整型类型（按符号划分）

- 有符号（signed，默认）、无符号（unsigned）

- 整型类型（按长度划分）

- 字符型（char）：8位

字符型可以视为取值范围很小的整型

- 短整型（short，16位机器int）：16位

- 长整型（long，32和64位机器int）：32位

- 超长整型（long long）：64位

- 注意：在Linux GCC 64位编译器下，long被视为超长整型



整型分类（32/64位架构）

- 在32或64位架构下整型所有分类
 - signed 和 int 可以省略不写（但不能略至没有类型）

名称	有符号类型	无符号版本
字符型 (<i>n</i> =3)	<code>signed char ; char</code>	<code>unsigned char</code>
短整型 (<i>n</i> =4)	<code>signed short int ; signed short ; short int ; short</code>	<code>unsigned short int ; unsigned short</code>
长整型 (<i>n</i> =5)	<code>signed long int ; signed long ; signed int ; long int ; long ; int ; signed</code>	<code>unsigned long int ; unsigned int ; unsigned ; unsigned long ;</code>
超长整型 (<i>n</i> =6)	<code>signed long long int ; signed long long ; long long int ; long long</code>	<code>unsigned long long int ; unsigned long long</code>



整型的取值范围

- 整型的内存占用长度： $N = 2^n$
- 整型的取值范围（精度为1）
 - 有符号 $[-2^{N-1}, 2^{N-1}-1]$ ，无符号 $[0, 2^N-1]$

名称	长度 ($N=2^n$)	有符号取值范围 ($[-2^{N-1}, 2^{N-1}-1]$ ，精度为1)	无符号取值范围 ($[0, 2^N-1]$ ，精度为1)
字符型 ($n=3$)	8 B	-128 ~ +127	0 ~ 255
短整型 ($n=4$)	16 B	-32,768 ~ +32,767	0 ~ 65,535
长整型 ($n=5$)	32 B	-2,147,483,648 ~ +2,147,483,647	0 ~ +4,294,967,296
超长整型 ($n=6$)	64 B	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807	0 ~ +18,446,744,073,709,551,615



整数的内存格式（字符型为例）

- 字符（ character ）

位置	7	6	5	4	3	2	1	0	值
Bits	1	0	1	1	0	0	1	0	-
Hex	B				2				-
unsigned char	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	178
signed char	—	$256 - 178 = 78$							-78



位、字节和字

- 位 (bit) : 二元不确定性
 - 有2种情况概率相等, 从未知到已知所获得的信息量为1位
 - 有 2^N 种情况概率相等, 从未知到已知所获得的信息量为 N 位
- 字节 (byte) : 内存处理数据的最小单位
 - 1 byte = 8 bits
- 字 (word) : 用于一次性处理事务的固定长度位组
 - 现代计算机的字长通常为32、64位
 - 字长指寄存器的大小 (即指示内存的范围)



进制：看信息的角度

- 根据 x_i 和旧进制 N 算出 $x = \sum_{i=-\infty}^{\infty} x_i \cdot N^i, x_i \in \{0, 1, \dots, N\}$
- 然后再根据新的进制 N' 计算出

$$x'_i = \frac{x}{N'^i} \bmod N'$$

9	0	2	7
9000	0	20	7

2	1	5	0	3
8192	512	320	0	3

例如：将10进制的9027转换为8进制数

$$9 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0 = 9027$$

$$9027/8^4 \bmod 8 \approx 2 \bmod 8 = 2$$

$$9027/8^3 \bmod 8 \approx 17 \bmod 8 = 1$$

$$9027/8^2 \bmod 8 \approx 141 \bmod 8 = 5$$

$$9027/8^1 \bmod 8 \approx 1128 \bmod 8 = 0$$

$$9027/8^0 \bmod 8 \approx 1128 \bmod 8 = 3$$

答案：21503



进制：不同进制的转换

- 不同进制与二进制的转换

数字	BIT 3	BIT 2	BIT 1	BIT 0	数字	BIT 3	BIT 2	BIT 1	BIT 0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10 (A)	1	0	1	0
3	0	0	1	1	11 (B)	1	0	1	1
4	0	1	0	0	12 (C)	1	1	0	0
5	0	1	0	1	13 (D)	1	1	0	1
6	0	1	1	0	14 (E)	1	1	1	0
7	0	1	1	1	15 (F)	1	1	1	1



ASCII码表

- 字符型数据的值是8位（ bit ）整数
 - 其ASCII码所对应的字符是输入输出格式之一
- 表示（ 初学者只需知道它们是有顺序的 ）
 - 0-31：非打印控制字符，用于控制打印机等外围设备。
 - 例如，12代表换页/新页功能，指示打印机跳到下一页的开头。
 - 32-127：可打印字符，能在键盘上找到的字符。
 - 数字127代表 DELETE 命令。
 - 128-255：扩展ASCII打印字符



*ASCII码表

ASCII (1977/1986)

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	<u>NUL</u> 0000 0	<u>SOH</u> 0001 1	<u>STX</u> 0002 2	<u>ETX</u> 0003 3	<u>EOT</u> 0004 4	<u>ENQ</u> 0005 5	<u>ACK</u> 0006 6	<u>BEL</u> 0007 7	<u>BS</u> 0008 8	<u>HT</u> 0009 9	<u>LF</u> 000A 10	<u>VT</u> 000B 11	<u>FF</u> 000C 12	<u>CR</u> 000D 13	<u>SO</u> 000E 14	<u>SI</u> 000F 15
1_	<u>DLE</u> 0010 16	<u>DC1</u> 0011 17	<u>DC2</u> 0012 18	<u>DC3</u> 0013 19	<u>DC4</u> 0014 20	<u>NAK</u> 0015 21	<u>SYN</u> 0016 22	<u>ETB</u> 0017 23	<u>CAN</u> 0018 24	<u>EM</u> 0019 25	<u>SUB</u> 001A 26	<u>ESC</u> 001B 27	<u>FS</u> 001C 28	<u>GS</u> 001D 29	<u>RS</u> 001E 30	<u>US</u> 001F 31
2_	<u>SP</u> 0020 32	<u>!</u> 0021 33	<u>"</u> 0022 34	<u>#</u> 0023 35	<u>\$</u> 0024 36	<u>%</u> 0025 37	<u>&</u> 0026 38	<u>'</u> 0027 39	<u>(</u> 0028 40	<u>)</u> 0029 41	<u>*</u> 002A 42	<u>+</u> 002B 43	<u>,</u> 002C 44	<u>-</u> 002D 45	<u>.</u> 002E 46	<u>/</u> 002F 47
3_	<u>0</u> 0030 48	<u>1</u> 0031 49	<u>2</u> 0032 50	<u>3</u> 0033 51	<u>4</u> 0034 52	<u>5</u> 0035 53	<u>6</u> 0036 54	<u>7</u> 0037 55	<u>8</u> 0038 56	<u>9</u> 0039 57	<u>:</u> 003A 58	<u>;</u> 003B 59	<u><</u> 003C 60	<u>=</u> 003D 61	<u>></u> 003E 62	<u>?</u> 003F 63
4_	<u>@</u> 0040 64	<u>A</u> 0041 65	<u>B</u> 0042 66	<u>C</u> 0043 67	<u>D</u> 0044 68	<u>E</u> 0045 69	<u>F</u> 0046 70	<u>G</u> 0047 71	<u>H</u> 0048 72	<u>I</u> 0049 73	<u>J</u> 004A 74	<u>K</u> 004B 75	<u>L</u> 004C 76	<u>M</u> 004D 77	<u>N</u> 004E 78	<u>O</u> 004F 79
5_	<u>P</u> 0050 80	<u>Q</u> 0051 81	<u>R</u> 0052 82	<u>S</u> 0053 83	<u>T</u> 0054 84	<u>U</u> 0055 85	<u>V</u> 0056 86	<u>W</u> 0057 87	<u>X</u> 0058 88	<u>Y</u> 0059 89	<u>Z</u> 005A 90	<u>[</u> 005B 91	<u>\</u> 005C 92	<u>]</u> 005D 93	<u>^</u> 005E 94	<u>_</u> 005F 95
6_	<u>`</u> 0060 96	<u>a</u> 0061 97	<u>b</u> 0062 98	<u>c</u> 0063 99	<u>d</u> 0064 100	<u>e</u> 0065 101	<u>f</u> 0066 102	<u>g</u> 0067 103	<u>h</u> 0068 104	<u>i</u> 0069 105	<u>j</u> 006A 106	<u>k</u> 006B 107	<u>l</u> 006C 108	<u>m</u> 006D 109	<u>n</u> 006E 110	<u>o</u> 006F 111
7_	<u>p</u> 0070 112	<u>q</u> 0071 113	<u>r</u> 0072 114	<u>s</u> 0073 115	<u>t</u> 0074 116	<u>u</u> 0075 117	<u>v</u> 0076 118	<u>w</u> 0077 119	<u>x</u> 0078 120	<u>y</u> 0079 121	<u>z</u> 007A 122	<u>{</u> 007B 123	<u> </u> 007C 124	<u>}</u> 007D 125	<u>~</u> 007E 126	<u>DEL</u> 007F 127



4. 浮点型



单精度和双精度的精确度和范围

- 浮点数存在**精度问题**，不可以直接比较相等

— 因为：2的任何次方都不以0为结尾

关键字	名称	二进制位数				十进制位数	
		总	符号	指数	小数	小数精度	指数范围
float	单精度	32	1	8	23	6~7	-37~+38
double	双精度	64	1	11	52	15~16	-307~+308
long double	长双精度	80	1	15	64	19~20	-4391~+4392

* long double 为了对齐实际是12B或16B，不足补0。（初学者不需要掌握）

- 浮点型溢出后表示为infs（无限）



浮点型的精度问题

- 设 `float PI=3.14, r=1.5, h=3;`

- 则PI值为3.1400001049041，r和h值为原值

- 不同的乘法顺序得到不同的结果（失之毫厘谬以千里）

表达式	计算次序	打印为%.10lf	打印为%.2lf
<code>r*r*pi*h</code>	2.25 → 7.0650000572	21.1949996948	21.19
<code>r*r*h*pi</code>	2.25 → 6.75	21.1950016022	21.20

- 正确的做法

- 使用精确的数据类型

```
double PI=3.14, r=1.5, h=3;
```

- 先乘大数再乘小数

```
PI * h * r * r
```



浮点数的内存格式

• 单精度浮点数 (float)

– 浮点加法和乘法不符合结合律和分配律。

sign	exponent								fraction																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	1	1	1	1	1	0	0	0	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	2	1	0	1/4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
—	2^{-3}								$1 + 2^{-2}$																							
-1*1.25*0.125=-0.15625																																



双精度浮点数的内存格式

• 双精度浮点数 (double)

— 比浮点数精确

sign	exponent								fraction																							
63	6 2	6 1	6 0	. .	5 5	5 4	5 3	5 2	5 1	5 0	4 9	4 8	4 7	4 6	. .	3 5	3 4	3 3	3 2	. .	2 5	2 4	. .	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 0	
1	0	1	1	.	1	1	0	0	0	1	0	0	0	0	.	0	0	0	0	.	0	0	.	0	0	0	0	0	0	0	0	
—	—	0	0	.	0	0	2	1	0	1 / 4	0	0	0	0	.	0	0	0	0	.	0	0	.	0	0	0	0	0	0	0	0	
—	2^{-3}								$1 + 2^{-2}$																							
-1*1.25*0.125=-0.15625																																



单精度和双精度的精确位数

- 单精度 (float) 的精确度是小数点后6-7位
 - 因为23乘以0.3 ($\log_{10}2$) 等于6~7, $2^{8-1}-1$ 乘以0.3等于38~39
 - 2的-23次方是0.00000011920928955078125
- 双精度 (double) 的精确度是小数点后15-16位
 - 因为52乘以0.3等于15~16, $2^{11-1}-1$ 乘以0.3等于307~308
 - 2的-52次方是2.2204460492503130808472633361816e-16
- 用二进制理解会比十进制直观



```
/* floater.c--demonstrates round-off error */
#include <stdio.h>
int main(void)
{
    float a,b;

    b = 2.0e20 + 1.0;
    a = b - 2.0e20;
    printf("%f \n", a);
    printf("%e \n", a);

    return 0;
}
```

```
4008175468544.000000
4.008175e+012
```



```

/* typesize.c -- print out the sizes of various types
#include <stdio.h>
int main(void)
{
    /* c99 provides
    printf("Type int has a size of %zd bytes.\n", sizeof(int));
    printf("Type char has a size of %zd bytes.\n", sizeof(char));
    printf("Type long has a size of %zd bytes.\n", sizeof(long));
    printf("Type long long has a size of %zd bytes.\n",
    sizeof(long long));
    printf("Type double has a size of %zd bytes.\n",
    sizeof(double));
    printf("Type long double has a size of %zd bytes.\n",
    sizeof(long double));
    return 0;

```

Type int has a size of 4 bytes.
 Type char has a size of 1 bytes.
 Type long has a size of 4 bytes.
 Type long long has a size of 8 bytes.
 Type double has a size of 8 bytes.
 Type long double has a size of 12 bytes.



5. 类型的选用



基本数据类型的使用原则

- 声明少量的变量时，不折腾
 - 优先整数用int，字符用char，实数用double
- 默认情况无法满足要求时，按精度和值域选用
 - 整型int无法表示的整数用long long int
 - 双精度小数double无法表示的实数用long double
- 声明规模较大的数组，尽量选择更小的类型
 - 因为大量的变量使用太大的数据类型，会影响内存占用



```

/* print2.c-more printf() properties */
#include <stdio.h>
int main(void)
{
    unsigned int un = 3000000000; /* system with 32-bit int */
    short end = 200;                /* and 16-bit short */
    long big = 65537;
    long long verybig = 12345678908642;

    printf("un = %u and not %d\n", un, un);
    printf("end = %hd and %d\n", end, end);
    printf("big = %ld and not %hd\n", big, big);
    printf("verybig= %lld and not %ld\n", verybig, verybig);

    return 0;
}

```

```

un = 3000000000 and not -1294967296
end = 200 and 200
big = 65537 and not 1
verybig= 12345678908642 and not 1942899938

```



```
/* charcode.c-displays code number for a character */
#include <stdio.h>
int main(void)
{
    char ch;

    printf("Please enter a character.\n");
    scanf("%c", &ch);    /* user inputs character */
    printf("The code for %c is %d.\n", ch, ch);

    return 0;
}
```

输入以后需要按回车键，才能将缓冲的内容发送给程序

Please enter a character.

C↓

The code for C is 67.



6. 相关编程错误



变量未赋值即使用

- 变量未赋值即使用，结果将不可控
 - 尤其自增减等操作符，需要先读取变量

```
char c = 0;  
c += 3;  
printf("%c", c);
```

```
char c;  
c += 3;  
printf("%c", c);
```

- 使用scanf可能未改变变量值，应判断其返回值

```
char c;  
if (scanf("%c", &c) == 1)  
    printf("%c", c);
```

```
char c;  
scanf("%c", &c);  
printf("%c", c);
```



变量未赋值即使用

- 在代码块（如：循环体）内应先赋值再使用
 - 变量应声明在其使用的最小范围内
 - 除非将该声明语句移至其下代码块会无法得到正确结果
 - 变量应在代码块最前方赋初始值，不应在使用后赋初始值

```
int main() {  
    int n = 0;  
    while (scanf("%d", &n) == 1) {  
        int i, b = 1;  
        for (i = 1; i <= n; i++)  
            b = b * i;  
    }  
}
```

```
int main() {  
    int n = 0, i, b = 1;  
    while (scanf("%d", &n) == 1) {  
        for (i = 1; i <= n; i++)  
            b = b * i;  
        b = 1;  
    }  
}
```



混淆数字字符和数字的概念

- 字符是整型的一种，以%c输出时查询ASCII表

字符型常数	对应的值	字符型常数	对应的值
'\0'	0	'0'	48
...
'\x9'	9	'9'	57

- 键盘输入数字（例如：0）时

- 配合%c，其值为'0'，即48
- 配合%d，其值为'\0'，即0

```
char c;  
scanf("%c", &c);  
if (c == 0)  
    return 0;
```

```
scanf("%c", &c);  
if (c == '0')  
    return 0;
```

```
scanf("%d", &c);  
if (c == 0)  
    return 0;
```



赋值时类型不匹配

- 编译器不报错，取模赋值

warning: overflow in implicit constant conversion [-Woverflow]

错误类型	示例代码片段	运行功能
用过大的数赋值过小的类型	<code>char num = 259;</code>	变量num的值为3。
	<code>char num = 385;</code>	变量num的值为-127。
用实数赋值整型	<code>char num = 259.3;</code>	变量值为127。
	<code>char num = -385.3;</code>	变量值为-128。
用有符号的数给无符号的变量赋值	<code>unsigned char num = -2;</code>	变量值为254。
	<code>unsigned char num = 2.5;</code>	变量值为0。
	<code>unsigned int a = 3;</code> <code>int res = (a - 4) > 0;</code>	变量res的值为0。
有符号写成无符号造成运算结果溢出	<code>int num = 65535;</code> <code>int res = (num*num > 0);</code>	变量res的值为0。



整数类型的溢出

- 整数类型的取值范围（如图）

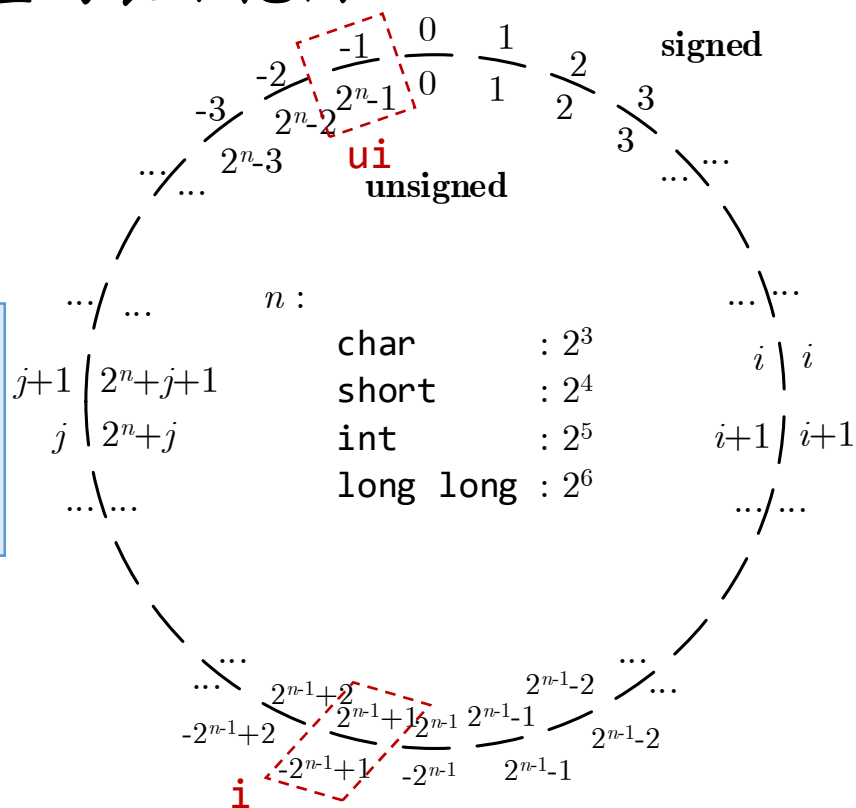
- 溢出：表达式超过了其数据类型的表示范围

- 溢出后的表现

- 根据图示自动转换

```
unsigned int ui = -1; 225-1+1  
int i = 2147483649u; 225-1+1  
printf("%u\n", i); // -2147483647
```

- 程序员应该避免溢出的发生



```

/* toobig.c-exceeds maximum int size on our system */
#include <stdio.h>
int main(void)
{
    int i = 2147483647;
    unsigned int j = 4294967295;

    printf("%d %d %d\n", i, i+1, i+2);
    printf("%u %u %u\n", j, j+1, j+2);

    return 0;
}

```

```

print1.c: In function 'main':
print1.c:6:5: warning: this decimal constant is
unsigned only in ISO C90 [enabled by default]
    unsigned int j = 4294967295;
    ^

```

```

2147483647 -2147483648 -2147483647
4294967295 0 1

```



忽视浮点数精度问题的错误

- 浮点数存在精度问题

- 运算结果应为 x 时，实际结果可能是 $y \in [x - \varepsilon, x + \varepsilon]$

- 原因：十进制小数转换为二进制需要更多的有限数字表示

- 浮点数比较大小关系，会以一定概率出现答案错误

```
( 1.1f - 1.f - .1f ) == 0    // 1.1000000238
```

```
fabs( 1.1f / 9 * 3 * 3 == 1.1f ) < 1e-6
```

$$1.099999904632568359375 = 1 \times 2^0 + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-8} + 1 \times 2^{-9} + 1 \times 2^{-12} \\ + 1 \times 2^{-13} + 1 \times 2^{-16} + 1 \times 2^{-17} + 1 \times 2^{-20} + 1 \times 2^{-21}$$

$$1.10000002384185791015625 = 1 \times 2^0 + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-8} + 1 \times 2^{-9} + 1 \times 2^{-12} \\ + 1 \times 2^{-13} + 1 \times 2^{-16} + 1 \times 2^{-17} + 1 \times 2^{-20} + 1 \times 2^{-21} + 1 \times 2^{-23}$$

$$0.0999999940395355224609375, 0.100000001490116119384765625$$



忽视浮点数精度问题的错误

- 浮点数上下和舍入取整，会以一定概率出现答案错误

– 切勿认为double类型就不会有精度问题 $\sqrt[3]{125} = 5$

```
#include <stdio.h>
#include <math.h>
int main() {
    double x = 125;
    double y = 1. / 3;
    double d = pow(x, y);
    int u = d;
    printf("Pow = %.16lf\n", d);
    printf("Floor Pow = %.16lf\n", floor(d));
    printf("Floor Pow Int = %d\n", u);
    return 0;
}
```

```
Pow = 4.9999999999999991
Floor Pow = 4.0000000000000000
Floor Pow Int = 4
```

数学库函数存在精度问题，很难想象。但解决这一问题的有效方法，还是在于认识全面测试程序的重要性。在浮点数转换为整数时，多留个心眼。



* 布尔类型 (Boolean)

- `_Bool` 类型在C99引入，用于表示布尔值
 - `true` (非0)、`false` (0)
 - 大小：1 bit
 - 其实是用int实现，此时不能用1或0来判别。



* 可移植类型stdint.h和inttypes.h

- 可移植类型应包含 `stdint.h` 和 `inttypes.h`
- `int` 等对不同架构大小不同，需要“确切长度类型”
 - `int8_t` `int16_t` `int32_t` ...
- 最小长度类型：能容纳指定长度的最小类型
 - 如：`int_least8_t`
- 最快最小长度类型：使计算达到最快的最小长度类型
 - 如：`int_fast8_t`



```

/* altnames.c -- portable names for integer types */
#include <stdio.h>
#include <inttypes.h> // supports portable types
int main(void)
{
    int32_t me32;      // me32 a 32-bit signed variable

    me32 = 45933945;
    printf("First, assume int32_t is int: ");
    printf("me32 = %d\n", me32);
    printf("Next, let's not make any assumptions.\n");
    printf("Instead, use a \"macro\" from inttypes.h: ");
    printf("me32 = %" PRIu32 "\n", me32);

    return 0;
}

```

First, assume int32_t is int: me32 = 45933945
 Next, let's not make any assumptions.
 Instead, use a "macro" from inttypes.h: me32 = 45933945



```

/* showf_pt.c -- displays float value in two ways */
#include <stdio.h>
int main(void)
{
    float aboat = 32000.0;
    double abet = 2.14e9;
    long double dip = 5.32e-5;
    printf("%f can be written %e\n", aboat, aboat);
    // next line requires C99 or later compliance
    printf("And it's %a in hexadecimal, powers of 2 notation\n", aboat);
    printf("%f can be written %e\n", abet, abet);
    printf("%Lf can be written %Le\n", dip, dip);
    re 32000.000000 can be written 3.200000e+004
    And it's 0x1.f40000p+14 in hexadecimal, powers of 2 notation
    2140000000.000000 can be written 2.140000e+009
    0.000053 can be written 5.320000e-005
}

```



C程序设计

T03



谢谢

厦门大学信息学院软件工程系

黄炜 副教授