We'll have more choices.

C 程 序 设 计
C Programming

3

数据类型

理论课程

厦門大學
XIAMEN UNIVERSITY

信息学院 黄 煒
（国家示范性软件学院）博士，副教授
School of Informatics Dr. Wei Huang

知识框架

- 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

内容纲要

```c
/* platinum.c  -- your weight in platinum */
#include <stdio.h>
int main(void)
{
    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: ");
    /* get input from the user                    */
    scanf("%f", &weight);
    /* assume platinum is $1700 per ounce         */
    /* 14.5833 converts pounds avd to ounces troy */
    value = 1700.0 * weight * 14.5833;
```

新的数据类型：浮点型

对普通变量，此处需要&号

此处程序暂停，等待用户键盘输入，直到按下回车键（Enter）继续

表示常数的新方法：带小数点表示浮点型

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 your value.\n");
    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

- 定义
  - 常量（Constants）：运行过程中保持值不变的量
    - 例如：π=3.14 是不能随便改动的
  - 变量（Variable）：运行过程中值可以改变的量
    - 例如：计算1到10的和s，指示当前加数的量s不断变化
- 属性
  - 名字：在作用域内应有唯一的、有意义的名字
    - 作用域：可以用该名字指代该量的代码行范围
  - 值：存于内存，用于参与计算，指示其当前值或被改变值

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

改变其值

指示当前值

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
    - 单词之间用下划线区隔

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 PPO 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
  - 有经验的程序员避免将常量声明为变量
  - 变量的名字一般较少甚至不使用大写字母
    - 单词之间用下划线区隔

改变方式	示例	格式
通过赋值改变值	`value = 1700.0 * weight * 14.5833;`	变量名，赋值符号，表达式；分号。
通过写内存改变值	`scanf("%f", &weight);`	需要调用scanf和memset等函数改变内存值

取地址操作符&

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
- 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

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>;	`int var;`
声明多个变量	<类型> <变量名>, <变量名2>;	`int grade, class, x;`
声明时赋值	<类型> <变量名>=<表达式>;	`int grade, class = 2;`

- 变量声明语句的位置
  - 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
  - 赋值后变量的值为赋值符号右侧表达式的值

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>	a = var;
变量可以组成表达式	由<变量名>和<操作符> 组成	a = x + 1;

- 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4字节空间

- 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

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

黄色底色的方框中是错误的示例代码

- 尝试修改常量
  - 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

```c
/* 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
`circum = 2.0 * PI *radius;`	`circum = 2.0 * 3.14159 *radius;`
`"PI is 3.14"`	`"PI is 3.14"`
`area = PIs * radius * radius;`	`area = PIs * radius * radius;`

宏定义	s = PI * r * r; 宏展开的语句
`#define PI 3.1415`	`s = 3.1415 * r * r;`
`#define PI 3 + 0.1415`	`s = 3 + 0.1415 * r * r;`
`#define PI = 3.1415`	`s = = 3.1415 * r * r;`

内容纲要

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	\a \b \f \n \r \t \v \' \" \\ \?
八进制转义序列	反斜杠（\），1~3位八进制数（01234567）	\101
十六进制转义序列	反斜杠（\），小写字母x，十六进制数（0123456789ABCDEF）	\x41

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）

<正负号><小数常数>[指数][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

```c
/* escape.c -- uses escape characters */
#include <stdio.h>
int main(void)
{
    float salary;

    printf("\aEnter your desired monthly salary:");/* 1 */
    printf(" $_______\b\b\b\b\b\b\b");              /* 2 */
    scanf("%f", &salary);
    printf("\n\t$%.2f a month is $%.2f a year.", salary,
           salary * 12.0);                          /* 3 */
    printf("\rGee!\n");                             /* 4 */
    return 0;
}
```

利用转义序列回退光标

光标在此

```
Enter your desired monthly salary: $_______
```

打印到此

```
Enter your desired monthly salary: $123↲___

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

内容纲要

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
  - 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

字符型可以视为取值范围很小的整型

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（*n*=3）	signed char ; char	unsigned char
短整型（*n*=4）	signed short int ; signed short ; short int ; short	unsigned short int ; unsigned short
长整型（*n*=5）	signed long int ; signed long ; signed int ; long int ; long ; int ; signed	unsigned long int ; unsigned int ; unsigned ; unsigned long ;
超长整型（*n*=6）	signed long long int ; signed long long ; long long int ; long long	unsigned long long int ; unsigned long long

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 ~ +4,294,967,295
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 $x = \sum_{i=-\infty}^{\infty} x_i \cdot N^i, x_i \in 0,1,\ldots,N$
- 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

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

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

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

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

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

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

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

内容纲要

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
r*r*pi*h	2.25 → 7.0650000572	21.1949996948	21.19
r*r*h*pi	2.25 → 6.75	21.1950016022	21.20

- 正确的做法
  - 使用精确的数据类型
  - 先乘大数再乘小数

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

```
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	09	08	07	06	05	04	03	02	01	00
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	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	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	62	61	60	. .	55	54	53	52	51	50	49	48	47	46	. .	35	34	33	32	. .	25	24	. .	07	06	05	04	03	02	01	00
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2.220446049250313080847263336181 6e-16
- 用二进制理解会比十进制直观

```c
/* floaterr.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;
}
```

```
400817546854 4.000000
4.008175e+012
```

```c
/* typesize.c -- pr
#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.
```

内容纲要

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

```c
/* 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
```

```c
/* 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.
```

内容纲要

1	变量和常量
2	常数的表示
3	整型和浮点型
4	相关的编程错误
5	小结

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

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

- 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

```
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]
```

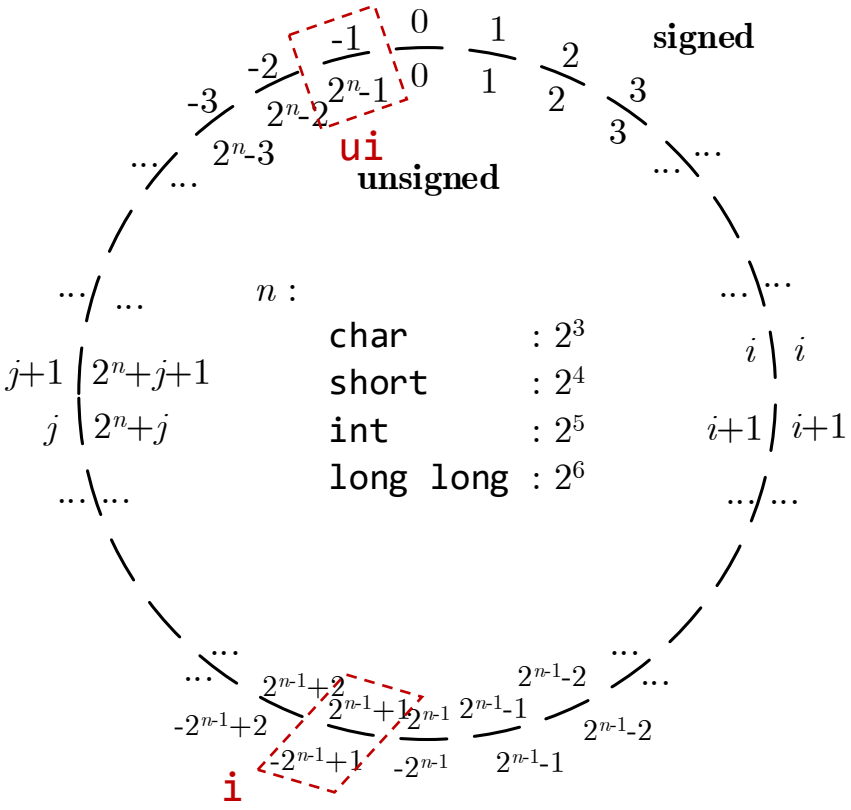
错误类型	示例代码片段	运行功能
用过大的数赋值过小的类型	`char num = 259;`	变量num的值为3。
	`char num = 385;`	变量num的值为-127。
用实数赋值整型	`char num = 259.3;`	变量值为127。
	`char num = -385.3;`	变量值为-128。
用有符号的数给无符号的变量赋值	`unsigned char num = -2;`	变量值为254。
	`unsigned char num = 2.5;`	变量值为0。
	`unsigned int a = 3;` `int res = (a - 4) > 0;`	变量res的值为1。
有符号写成无符号造成运算结果溢出	`int num = 65535;` `int res = (num*num > 0);`	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;  2^(2^5) − 1
int i = 2147483649u;   2^(2^5−1) + 1
printf("%u\n", i); // -2147483647
```

- 程序员应该避免溢出的发生



```c
/* 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
```

$$1.09999990463256835937 5 = 1\times2^{0}+1\times2^{-4}+1\times2^{-5}+1\times2^{-8}+1\times2^{-9}+1\times2^{-12}+1\times2^{-13}+1\times2^{-16}+1\times2^{-17}+1\times2^{-20}+1\times2^{-21}$$

$$1.10000002384185791015625 = 1\times2^{0}+1\times2^{-4}+1\times2^{-5}+1\times2^{-8}+1\times2^{-9}+1\times2^{-12}+1\times2^{-13}+1\times2^{-16}+1\times2^{-17}+1\times2^{-20}+1\times2^{-21}+1\times2^{-23}$$

$$0.09999999403953552246093 75, 0.10000000149011611938476 5625$$

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 $\sqrt[3]{125} = 5$

```c
#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 = 4.999999999999991
Floor Pow = 4.000000000000000
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）

扩展内容

- _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

扩展内容

- 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

```c
/* 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 = %" PRId32 "\n", me32);

    return
}
```

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

```c
/* 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
```

内容纲要

1	变量和常量
2	常数的表示
3	整型和浮点型
4	相关的编程错误
5	小结

3

谢谢

理论课程

厦門大學
XIAMEN UNIVERSITY

信息学院 黄 煒
（国家示范性软件学院） 博士，副教授
School of Informatics Dr. Wei Huang