遗留问题

```
_Bool flag=1;
for (d = 3; d < =sqrt(n); d++)
  if (0 == n % d) {
     flag = 0;
     break;} //合数
if(flag) printf("%d is prime\n", n);
else
  printf("%d is divisible by %d\n", n, d);
```

第7章

基本类型

基本类型

- C语言的基本（内置的）类型：
  - 整型（int）：long、short和unsigned
  - 浮点型：float，double和long double
  - 字符型（char）
  - 布尔型（_Bool (C99)）

整数：有符号、无符号

- 有符号数：二进制最高位（最左边）为符号位，0：正数，1：负数
  - 最大16位整数：32,767（2^{15} - 1），
  - 二进制表示形式0111111111111111。
  - 最大32位整数是01111111111111111111111111111111，数值为2,147,483,647（2^{31} - 1）。
- 无符号整数：最左边位是数值的一部分
  - 最大的16位无符号整数65,535（2^{16} - 1）
  - 最大的32位无符号整数是4,294,967,295（2^{32} - 1）。

有符号和无符号整数

- 整型变量默认有符号
- 无符号数：
  - 强制声明unsigned类型。
  - 主要用于系统编程和低级的、与机器相关的应用。

整型

- 有/无符号+三种长度(关键字)产生6种类型组合：

short int	unsigned short int
int	unsigned int
long int	unsigned long int

- 关键字的顺序没有要求，“xxx +int”“int”可省略
  - long int可以缩写为long。

整型

- 6种整型取值范围随机器不同而不同
- 同种类型取值范围不一定统一
  - int，有些2byte，有些4byte
  - 类似公路，高速有些100，有些120
- C标准要求：
  - short、int和long中的每一种类型都要覆盖一个确定的最小取值范围
  - 其次，长度：long int>=int>=short int

整型

- 16位机器上整型通常的取值范围：

类型	*最小值*	*最大值*
`short int(2Byte)`	-32,768	32,767
`unsigned short int`	0	65,535
`int`	-32,768	32,767
`unsigned int`	0	65,535
`long int(4byte)`	-2,147,483,648	2,147,483,647
`unsigned long int`	0	4,294,967,295

整型

- 32位机器上整型通常的取值范围：

类型	*最小值*	*最大值*
`short int(2byte)`	-32,768	32,767
`unsigned short int`	0	65,535
`int(4byte)`	-2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int(4byte)`	-2,147,483,648	2,147,483,647
`unsigned long int`	0	4,294,967,295

整型

- 64位机器上整型通常的取值范围：

类型	*最小值*	*最大值*
`short int`	-32,768	32,767
`unsigned short int`	0	65,535
`int(4byte)`	-2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int(8byte)`	-2^{63}	$2^{63}-1$
`unsigned long int`	0	$2^{64}-1$

- 上述取值范围并非C标准强制
- 标准库 头文件 `<limits.h>` 可以找到定义了每种整数最大值和最小值的宏。

整型常量

- 在程序中以文本形式出现的数
  - 变量：温度
  - 常量：密度，圆周率，汇率
- 不一定宏定义
  - 宏定义是给常量命名
- C语言允许用十进制、八进制和十六进形式书写整型常量。
- 本质上？
  - 二进制

整型常量

- 十进制常量包含数字0~9，以非零数字开头：
  - 15　255　32767
- 八进制常量包含数字0~7，以零开头
  - 017　0377　077777
- 十六进制常量包含数字0~9和字母a~f
  -

不管什么进制，对数本身没有影响，计算机只识二进制。

八进制和十六进制数

- 八进制数：0~7编写。每一位表示一个8次幂。
  - 237[八进制]→十进制数：

    $2 \times 8^2 + 3 \times 8^1 + 7 \times 8^0$

    $= 128 + 24 + 7 = 159$。
- 十六进制数：0~9加上A~F，A~F分别10~15。
  - 1AF [十六进制]→十进制：

    $1 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$

    $= 256 + 160 + 15 = 431$。

整型常量

- 常量没有声明（大小）
- 十进制常量默认：int类型
  - 超过int范围，用long int，还大，编译器会将其作为unsigned long int 来处理。
- 八或十六进制：
  - 编译器将遍历int, unsigned int, long int和 unsigned long int类型，从小到大，直到找到适合的表示常量的类型。

整型常量

- 常量后加上字母L（或l）：强制编译器把常量作为长整型数来处理：
  - 15L　0377L　0x7fffL
- 常量后边加上字母U（或u），指明无符号常量：
  - 15U　0377U　0x7fffU
- 常量长且无符号：组合字母L和U：
  - 0xffffffffUL
- 字母L和U的顺序和大小写都没有关系

整型溢出

- 算术运算结果超出表示范围。
- 溢出表现跟操作数有无符号有关
- 有符号溢出
  - 程序行为没有定义（不确定的）
- 无符号溢出时，掐头取尾
  - 结果为正确结果模（余）除2^n的结果，丢掉进位，保留余数

eg.两位十进制相加，结果取两位：
56+78=134，取值34

读/写整数

- 无符号整数、短和长整型整数需要一些新的转换说明符。
- 无符号整数，使用字母u、o或x。

```
unsigned int u;
scanf( “%u”, &u); //十
printf(″%u″, u);
scanf(″%o″, &u); //八
printf(″%o″, u);
scanf(″%x″, &u); //十六
printf(″%x″, u);
```

读/写整数

- 短整型：在d、o、u或x前面加上字母h：

```
short s;

scanf("%hd", &s);
printf("%hd", s);
```

- 长整型：在d、o、u或x前面加上l。
- 长长整型：在d、o、u或x前面加上ll。

程序：数列求和（改进版）

- 数列项或和可能超出int表示范围。
- 在16位int机器上，可能发生：

```
Enter integers (0 to terminate):
10000 20000 30000 0
The sum is: -5536
```

- 将变量改换成long int型进行改进。

浮点型

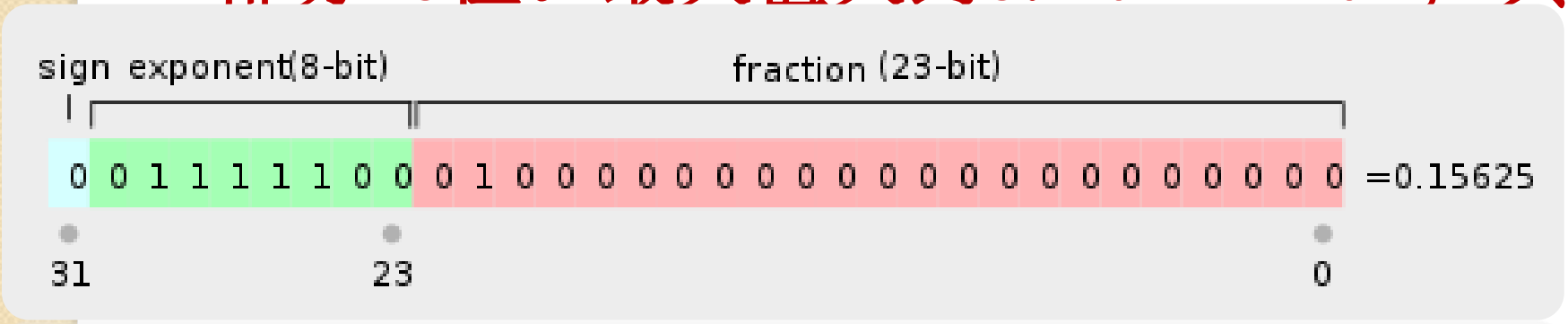
- float：单精度浮点数
- double：双精度浮点数
- long double：扩展双精度浮点数

浮点型

- float适合精度要求不严格的浮点数。
- double为大部分的程序提供了足够的精度。
- long double 很少用到。
- C标准没有说明三种float具体精度
  - 不同计算机存储浮点数的方法可以不同。
  - 大多数现代计算机和工作站都遵循IEEE 754标准。

IEEE浮点标准（IEEE754-1985）

- 提供了两种主要的浮点数格式：
  - float(32位)和double(64位)。
- 数值以科学计数法的形势存储：
  - 符号、指数和小数。
- float：
  - 符号1位、指数8位（含1位符号），小数部分23位。最大值大约3.40×10^{38}，其



浮点类型

- IEEE标准：

类型	*最小正值*	*最大值*	*精度*
`float`	1.17549×10^{-38}	3.40282×10^{38}	6 digits
`double`	2.22507×10^{-308}	1.79769×10^{308}	15 digits

- 不遵循IEEE标准的计算机上，该表无效。
- 在一些机器上，float可以有和double相同的数值集合，或**double**可以有和**long double**相同的数值。

浮点常量——多种书写方式

- 如果有指数（10的幂），必须在指数前放置字母E（或e）。
- 符号（+或-）可以出现在字母E（或e）的后边。
- 57.0的有效写法如下：
  - 57.0　57.　57.0e0　57E0　5.7e1　5.7e+1　.57e2　570.e-1

浮点常量

- 默认情况下，以双精度数的形式存储
- 只需单精度，常量末尾加F（或f）（如57.0F)
- 以long double格式存储，常量末尾加L（或l）(如57.0L)

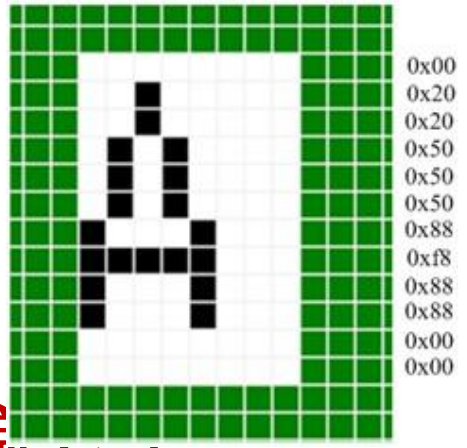
读/写浮点数

- 单精度浮点数：用转换说明符%e、%f和%g。
- double float，在e、f或g前放置字母l：

```
double d;
scanf("%lf", &d);
```

- 注意：
  - 只能在scanf函数格式串中使用l
  - 不能在printf函数格式串中使用，e、f和g可用来格式输出float或double型值。C99可加l，但不起作用
    - l控制scanf读缓存
- long double，在e、f或g前放置字母L。
  - scanf("%Lf", &d);
  - printf("%Lf", d);

字符型char



- 字符如何表示？A, a?
  - 计算机用数字对字符编码（编号）
  - 类似学号
- 字符如何显示？
  - 模子，printf(’A’);
  - 计算机中，12*8点阵

- 编码值可根据计算机的不同而不同，因不同的机器可能会有不同的字符集
- 例如，学号、身份证号

字符集

- 最常用的字符集是*ASCII*（美国信息交换标准码）
  - 7位代码（1byte）表示128个字符。
  - A-Z:1000001-1011010(65-90)(97-122)
  - 0-9:0110000-0111001(48-57)
    - 数字，数vs字，eg，9和’9’
- ASCII常被扩展为8位代码用于表示256个字符代码，被称为*Latin-1*,提供一些西欧和许多非洲语言所需的字符。

字符ASCII表

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

字符集

- char型变量可赋值为任何单个字符：

```
char ch;

ch = 'a';    /* lower-case a */
ch = 'A';    /* upper-case A */
ch = '0';    /* zero         */
ch = ' ';    /* space        */
```

- 注意，字符常量需要用单引号括起来，而不是双引号。

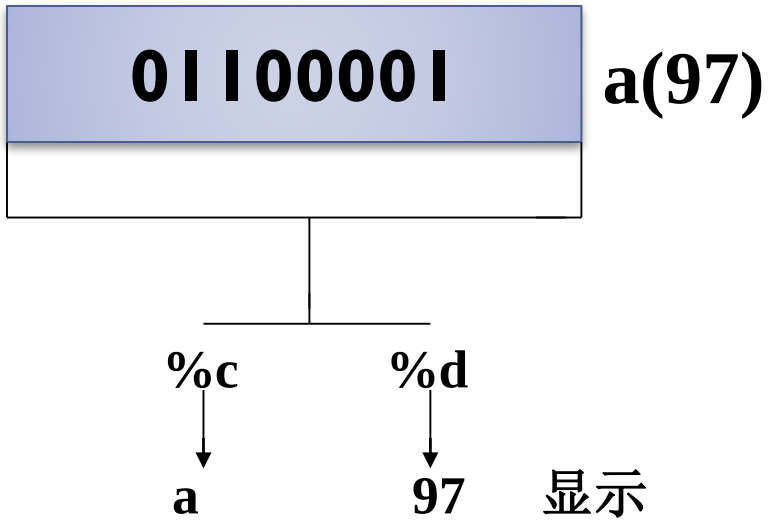
字符操作

- C语言以ASCII码存贮字符
  - 1Byte(2进制)：
    00000000~01111111(0~127)
- 按小整数的方式处理字符。

```
char c1,c2;
c1= ‘a’ ,c2=98;
printf("%c,%c\n",c1,c2);
printf("%d,%d\n",c1,c2);}
```

运行结果1：a,b

运行结果2：97,98



字符常量

- 不是char型,而是int型。
  - char ch = ‘a’ ;——ch =97;
  - #define CH ‘A’ ——define CH 65

97

00 00 00 97

不用管，实际值相同

字符操作

- 字符本质：int数值（小整数）
- 可以使用它对应的整数值赋值、比较、算术运算

字符操作——赋值运算

- 采用ASCII码字符集为前提:

```
char ch;     int i;
i = 'a';          /* i is now 97    */
ch = 65;          /* ch is now 'A' */
ch = ch + 1;      /* ch is now 'B' */
ch++;             /* ch is now 'C' */
```

字符操作——大小写转换

- 大小写转换：
  - eg，‘A’ -》 ‘a’
- A(65)，a(97),间隔32
  - ‘A’ + 32 ——》 ‘a’
  - ‘B’ + 32 ——》 ‘b’
- ASCII码中每个字母大小写都间隔32

32：转换因子
‘a’-’A’：转换因子

字符操作——比较

- 测试ch是否是小写字母；是则转化对应大写。

```
if( ‘a’ <=ch && ch <= ‘z’ )//测试
    ch = ch - ( ‘a’ - ‘A ‘);//转换
```

- 具体数值随字符集而有所不同
  - 程序使用<, <=, >, 和 >= 来进行字符比较可能不易移植。

字符操作

- 字符数的属性之好处：
  - ch++：实现在字符间游走（遍历）
  - eg. for循环控制：

```
for (ch = 'A'; ch <= 'Z'; ch++) …
```

- 字符数的属性之缺点：
  - 可能会导致编译器无法检查出来的错误。
  - 导致编写出诸如‘a’ * ‘b’ / ‘c’的无意义的表达式。
  - 可能会妨碍程序的可移植性
    - Eg,上述for语句利用A到Z连续编码

有符号和无符号字符

- 存在有符号和无符号两种char类型。
  - signed char sch;
  - unsigned char uch;
- 有符号字符：－128 ~ 127，无符号型字符：0~255。
- 一些编译器按有符号处理，一些编译器按无符号。
- 大多数时候，没有太大关系，不用关心。

算术类型

- 整数类型
  - char
  - int
  - 枚举（有序集合）：学校专业
- 浮点类型
  - 实数
  - 复数

转义序列

- 字符：
  - 普通字符：0-9，a-z，A-Z等，常量书写：单引号括起来
  - 特殊字符：不可见、（无法打印的或键入），如换行、制表符，需特殊表示法
    - \n, \t
- 转义序列呈现这类特殊符号
  - 字符转义序列
  - 数字转义序列

字符转义序列

名称	转义序列
Alert (bell)	\a
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t
Vertical tab	\v
Backslash	\\
Question mark	\?
Single quote	\’
Double quote	\″

- 方便但表示范围有限（仅常用特殊字符）
  - 不包含所有无法打印的ASCII字符，如ESC
  - 无法表示ASCII字符以外的字符，西欧、非洲等8位（256）Latin-1
- 用编号表示：数字转义序列

数字转义序列

- 可以表示任何字符。
  - 格式：\???
- ???：字符的八进制或十六进制编码（非十进制）
  - 八进制：如 \33 或 \033。最多三位八进制数
  - 十六进制：如\x1b 或\x1B。最多两位十六进制数，x小写，十六进制数字不限大小写。
- 作为字符常量：单引号括起来，’\141’
  - 0141=97
- 数字转义序列隐晦，#define命名意义明确：
  - #define ESC ’\33’
- 转义序列可嵌入字符串中使用。
  - “hello\n”

字符处理函数

- toupper库函数：
  - 小写字母转换成大写字母，tolower反之
  - ch = toupper(ch);//ch-’a’+’A’
- toupper在ctype.h中声明
  - #include <ctype.h>
- C函数库提供了其他有用的字符处理函数。

读/写字符

- scanf和printf用转换说明符%c读/写单个字符
- %c读字符不会跳过空白、回车等字符
- 需求：扫描一行的末尾（结束符）
  - 从行首开始，逐个读入并检查字符是否为换行符

```
do{//至少执行一次读取和检查，do循环
        scanf(“%c”, &ch);
}while(ch != ‘\n’);
```

- 循环结束后
  - ch=？？行末的“\n”
  - 循环读走当前行所有字符，下次scanf从新行开始
- 如需跳过空白等字符，%c前面加空格：
  - scanf(" %c", &ch);

温故而知新

- 基本类型：整型、浮点型、字符、布尔
  - 整型
    - 三种长度：short、int、long、long long[C99]
    - 符号：有/无
    - 读写：%[h/l]d/u/o/x
  - 浮点
    - float、double、long double
    - 读写：%[l/L]e/f/g

温故而知新——字符

- char：二进制编码来表示（编号）
- 最常用字符集：ASCII码，1Byte（8bits）二进制
- 本质：小整数 00000000~01111111(0~127)
  - printf(),%c, %d

温故而知新——转义序列

- 字符转义序列
  - 仅常用特殊字符，\n,\t方便但表示范围有限
- 数字转义序列
  - \ASCII码（8/16进制）

温故而知新——字符操作

- 字符数之本质
- 赋值：char i= ‘a’
  - 字符-》整型：int i = ‘a’;
  - 整型-》字符：char ch = 97;
- 比较
  - 遍历：for(ch = ’A’; ch <= ’Z’; ch++)
    …
- 算术运算
  - 大小写转换：ch = ch +/- 转换因子
    (‘a’ - ‘A’);
  - ch++; ch+=5;
  - ‘a’ * ‘b’ / ‘c’

读/写字符

- getchar()和putchar()代替scanf函数和printf函数读取字符。
- 格式：

```
putchar(ch);//printf(“%c”,ch)
ch = getchar();//scanf(“%c”,&ch)
```

- getchar()和putchar()节约时间
  - 实现简单
  - 作为宏实现
    - #define getchar() getc(stdin)

读/写字符

- gechar还优于scanf()函数——返回字符

  ch = getchar();直接取回

  scanf(“%c”,&ch);取回放某处

- 用getchar()扫描行结束(末尾’\n’)：

```
do{
    ch = getchar();//scanf(“%c”,ch);
}while (ch != '\n');
```

- 循环可转换为

```
do {;}
while ((ch = getchar()) != '\n');
```

读/写字符

- 进一步转换（删除do{;}）

  ```
  while ((ch = getchar()) != '\n') ;
  ```

  - 与do版本等价：getchar()至少执行一次
- 甚至变量ch都可以不需要：

  ```
  while (getchar() != '\n') ;
  ```

  - 类似等式代入

读/写字符

- 行首或存在空白，找非空行首
- 用getchar跳过若干空格，遇到第一个非空字符循环终止。

```
while((ch = getchar()) == ' ')  ;
```

精简：

```
while(getchar() == ' ')  ;
```

统计除行首空字符外的字符数

```
int len=0;
while(getchar() == ' ') ;
while (getchar() != '\n') len++;
```

程序：确定用户输入消息的长度

Enter a message: Brevity is the soul of wit.

Your message was 27 character(s) long.

- 长度包括空格和标点符号，不含结尾换行符。
- 可采用scanf或getchar函数读取字符，大多数C程序员愿意采用getchar函数。

```
#include <stdio.h>

int main(void)
{
   char ch;
   int len = 0;

   printf("Enter a message: ");

   while (getchar() != '\n')
       len++;

   printf("Your message was %d character(s)
   long.\n", len);

   return 0;
}
```

类型转换

- （第四章）不同类型操作数运算时，精度自适应攀高——类型转换
- 计算机执行算术运算，通常要求操作数具有相同的位数和存储方式，对齐
  - eg. 竖式运算
- 操作数类型不同时，C编译器先类型转换，以方便硬件实现。
  - a（16位int）+b（32位long int），a先转化成32位long int
  - int与float运算，将int转换为float。

类型转换类型

- 隐式转换：编译器自动处理的转换，无需程序员介入。
- 显示转换：程序员使用强制运算符——强制类型转换。

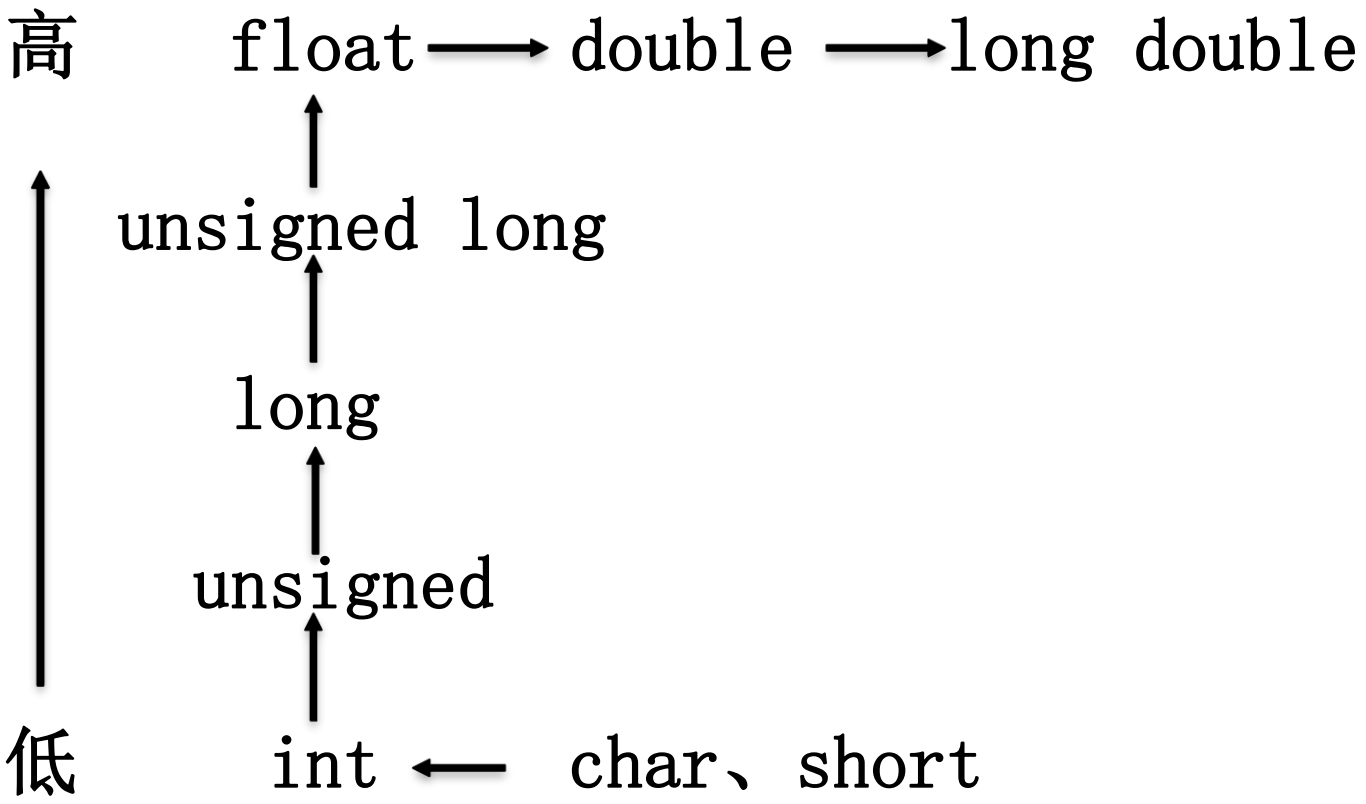
隐式类型转换

- 基本数据类型（算术类型）种类多，规则略微复杂
  - 算术转换：操作数的类型不同
  - 赋值转换：=左右类型不匹配
  - 函数调用转换：实参与形参类型不匹配
  - 函数返回转换：return 表达式与函数返回类型不匹配

常用算术转换策略

- 统一操作数类型：类型提升——将相对较狭小类型的操作数转换成另一个操作数的类型。
- 整型提升：short int——》int。

类型提升规则（跟大碗）



最好尽量避免使用无符号整数，
特别是不要把它和有符号整数混合使用。

赋值类型转换（削足适履）

- lv=expr：expr脚——》lv鞋的类型
- 小脚穿大鞋

```
char c;int i;
float f;    double d;
i = c;    /* c——》int    */
f = i;    /* i——》float */
d = f;    /* f——》double */
```

大脚小鞋，削足适履

- 浮点数赋值给整型变量，削小数部分。

```
int i;

i = 842.97;  /* i is now 842 */
i = -842.97; /* i is now -842 */
```

- expr结果超出左值取值范围（溢出），掐头取尾：

```
char c;

c = 256+78;（0000 0001 0100 1110）

c = 10000;     /*** WRONG ***/
i = 1.0e20;    /*** WRONG ***/
f = 1.0e100;   /*** WRONG ***/
```

函数调用与返回类型转换

- 本质上也是赋值
- 函数调用转换：实参赋值给形参
  - func(int a)→func(b)
- 函数返回转换：return 表达式返回给外部调用者

```
int func()
{
    …
    return 4.34;
}
```

强制类型转换

- 隐式转换自动完成，更大程度控制类型转换需主动进行转换。
  - average=sum/10;
- 强制类型转换表达式格式如下：

  （ 类型名 ）表达式

强制类型转换

- 求float型值小数部分：

```
float f, frac_part;
frac_part = f - (int) f;
```

- 求均值

```
float average;
int sum, n;
average = (float)sum/n;
```

强制类型转换优先级

- 一元运算符优先级高于二元运算符：
  - `(float) sum / n`
- 解释为
  - `((float) sum) / n`
- 其他方法可以实现同样的效果：
  - `average = sum / (float) n;`
  - `average = (float) sum / (float) n;`
- 强制转任一操作数，其余隐式转换

强制类型转换

- 使用强制类型转换来避免溢出：
  - long i;
  - int j = 1000;
  - i = j * j; /* overflow may occur */
- 用强制类型转换来避免这个问题：
  - i = (long) j * j;
  - i = (long) (j * j);/*** WRONG ***/
  - 溢出在强制类型转换之前就已经发生了

类型定义

- #define创建一个布尔类型的宏：

```
#define BOOL int
```

- 更好方法——类型定义给类型别名：

```
typedef int Bool;//(int别名Bool)
Bool flag;    /* same as int flag; */
```

- 同：都是别名
- 异：
  - 宏定义，指令，只能在程序头部，只在预编译时处理（替换）——静态别名
  - 类型定义，语句，程序运行时处理——动态别名

类型定义的优点

- 使程序更易于理解：
- 假设变量cash_in、cash_out存储收支的美元数量：
  - typedef float Dollars;
  - Dollars cash_in, cash_out;
- 比原始用法更有实际意义：
  - float cash_in, cash_out;

类型定义的优点

- 使程序更容易修改：
- 重定义Dollars为double类型，只需改变类型定义就可以了：
  - typedef double Dollars;
- 如果不使用类型定义，则需要找到所有用于存储美元数量的float型变量并改变它们的声明。

类型定义和移植性

- 类型定义是编写可移植程序的一种重要工具。
- 程序移植常出现的问题之一：不同计算机上的类型取值范围可能不同。

```
int i;
i = 100000;
```

  - 在32位整数的机器上没有问题，在16位整数的机器上就会出错。

类型定义和移植性

- Eg. 假设程序需要变量——产品数量（num），取值范围在0－50,000。
- 声明变量时，考虑int 还是long int。
  - 取值范围：long int，保证不溢出；
  - 用户方便：用户更愿意使用int
  - 算术运算速度：int型快；
  - 占用内存空间：int型变量少。
- 针对不同计算机（int，long int表示范围不同），用typedef调整程序

类型定义和移植性

- 定义自己的“数量”类型，而避免使用int类型声明数量变量：

```
typedef int Quantity;
Quantity q;
```

- 程序移植到小值整数(int 2 byte)机器，改变Quantity定义：

```
typedef long Quantity;
```

- 注意
  - 改变Quantity定义可能会影响Quantity类型变量的使用方式
  - 如printf（），scanf（），%d,or %ld

sizeof运算符

- 确定指定类型或变量所需存储空间的大小——字节数。

  sizeof (type-name)

- sizeof(char) 始终为1，其他类型计算出的值可能会有所不同。
- 在大多数32位机器上，sizeof(int)的值为4。

sizeof运算符

- 也可应用于常量、变量和表达式。
  - type i, j;
  - sizeof(i)——》sizeof(type)
  - sizeof(i + j)——》sizeof(type)
- 此时，圆括号不是必须的。
  - sizeof i等价于sizeof(i)。
- 优先级：圆括号可能还是会需要的。
  - sizeof i + j——》(sizeof i) + j,
  - sizeof（一元运算符）优先级高于二元运算符+。