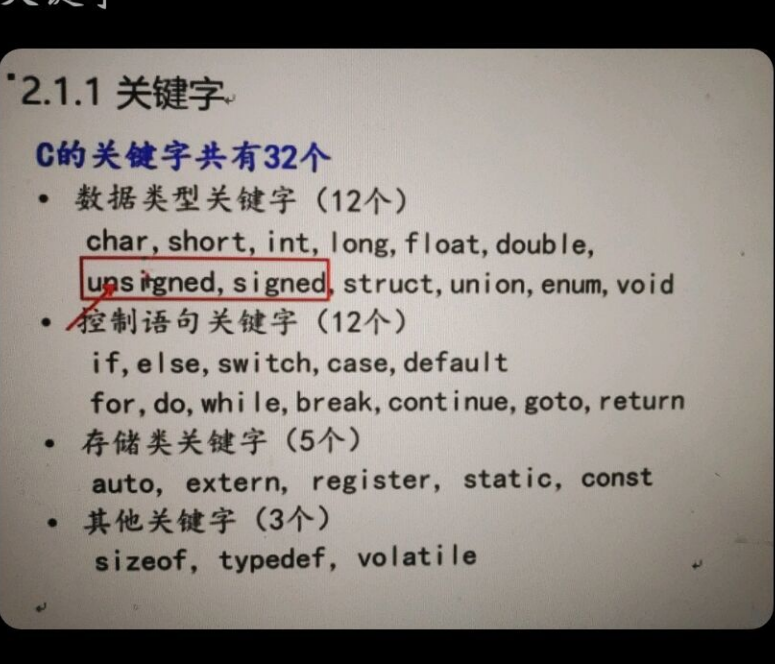
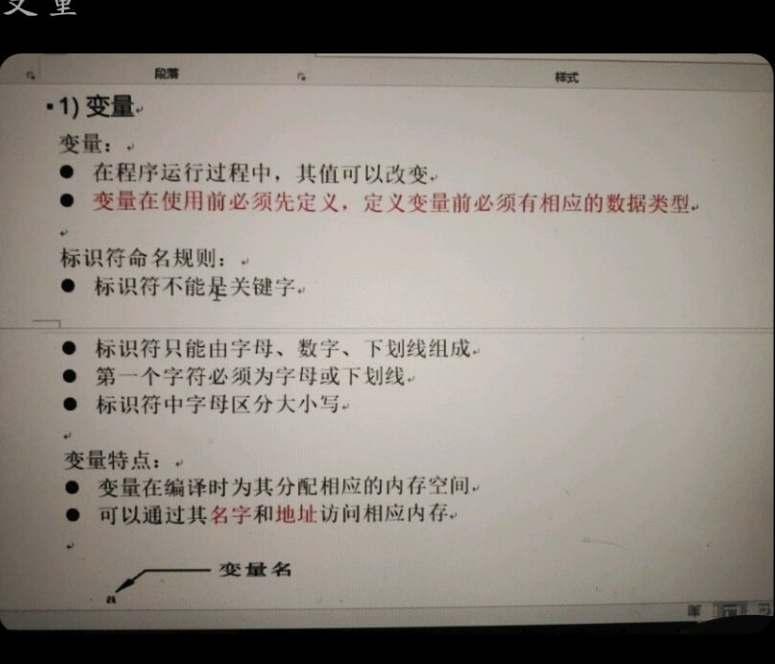
数据类型

关键字


2.1.1 关键字

C的关键字共有32个
- 数据类型关键字（12个）
  char, short, int, long, float, double,
  unsigned, signed, struct, union, enum, void
- 控制语句关键字（12个）
  if, else, switch, case, default
  for, do, while, break, continue, goto, return
- 存储类关键字（5个）
  auto, extern, register, static, const
- 其他关键字（3个）
  sizeof, typedef, volatile

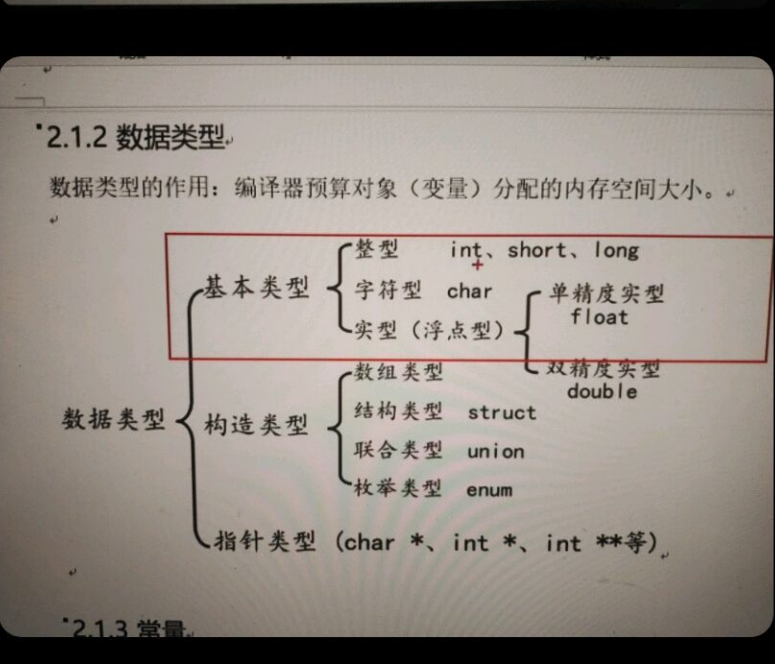
变量


1) 变量

变量：
- 在程序运行过程中，其值可以改变。
- 变量在使用前必须先定义，定义变量前必须有相应的数据类型。

标识符命名规则：
- 标识符不能是关键字。
- 标识符只能由字母、数字、下划线组成。
- 第一个字符必须为字母或下划线。
- 标识符中字母区分大小写。

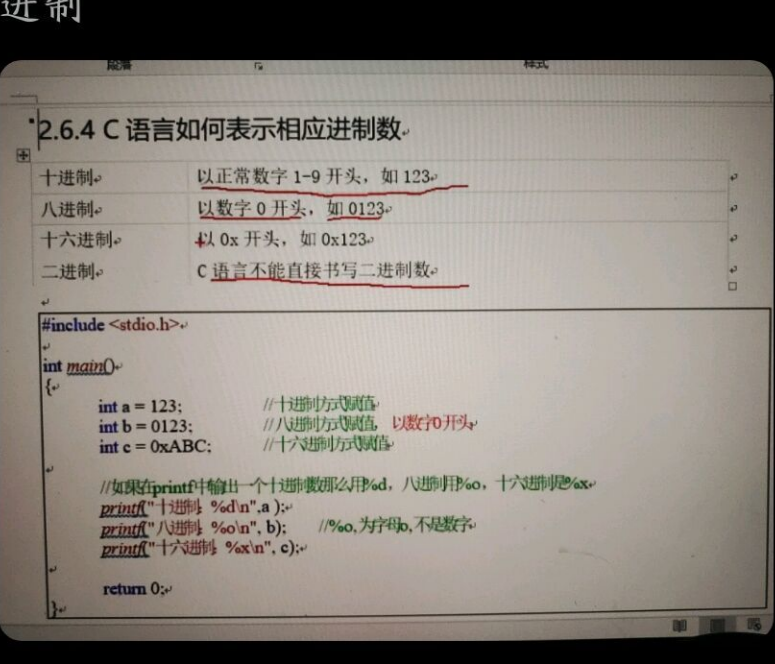
变量特点：
- 变量在编译时为其分配相应的内存空间。
- 可以通过其名字和地址访问相应内存。


2.1.2 数据类型

数据类型的作用：编译器预算对象（变量）分配的内存空间大小。

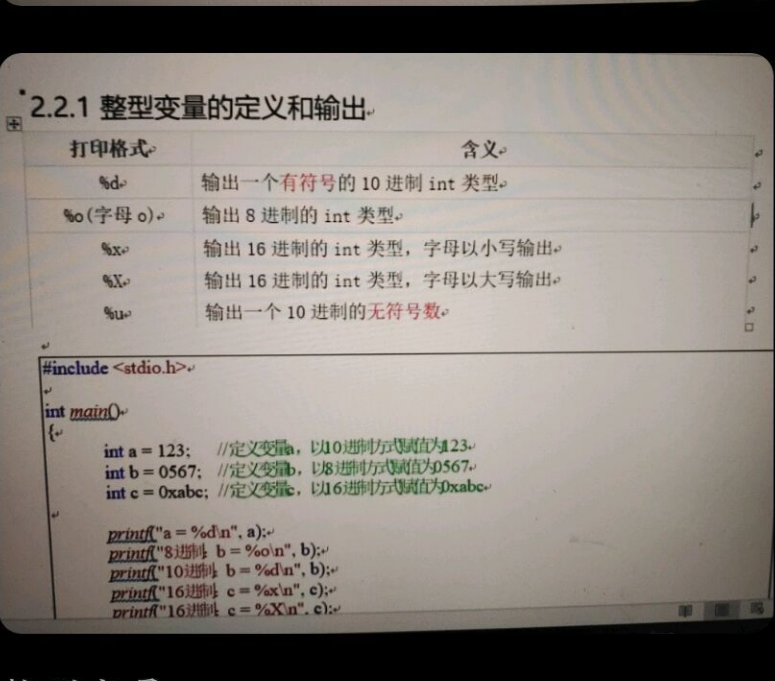
float 默认保留六位小数

进制


2.6.4 C 语言如何表示相应进制数

十进制	以正常数字 1-9 开头，如 123
八进制	以数字 0 开头，如 0123
十六进制	以 0x 开头，如 0x123
二进制	C 语言不能直接书写二进制数

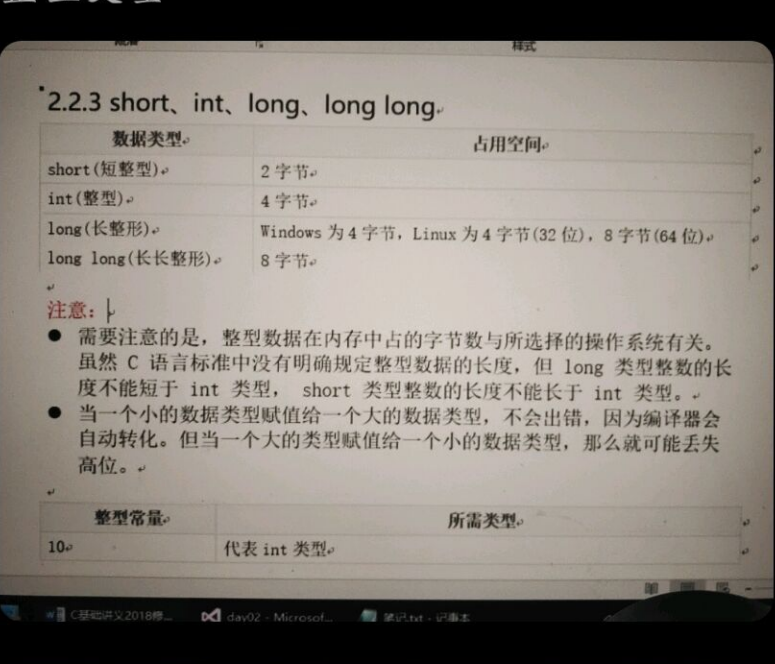
```
#include <stdio.h>

int main()
{
	int a = 123;		//十进制方式赋值
	int b = 0123;		//八进制方式赋值，以数字0开头
	int c = 0xABC;		//十六进制方式赋值

	//如果在printf中输出一个十进制数那么用%d，八进制用%o，十六进制是%x
	printf("十进制：%d\n",a );
	printf("八进制：%o\n", b);	//%o,为字母o,不是数字
	printf("十六进制：%x\n", c);

	return 0;
}
```


2.2.1 整型变量的定义和输出

打印格式	含义
%d	输出一个有符号的 10 进制 int 类型
%o(字母 o)	输出 8 进制的 int 类型
%x	输出 16 进制的 int 类型，字母以小写输出
%X	输出 16 进制的 int 类型，字母以大写输出
%u	输出一个 10 进制的无符号数

```
#include <stdio.h>

int main()
{
	int a = 123;	//定义变量a，以10进制方式赋值为123
	int b = 0567;	//定义变量b，以8进制方式赋值为0567
	int c = 0xabc;	//定义变量c，以16进制方式赋值为0xabc

	printf("a = %d\n", a);
	printf("8进制：b = %o\n", b);
	printf("10进制：b = %d\n", b);
	printf("16进制：c = %x\n", c);
	printf("16进制：c = %X\n", c);
```

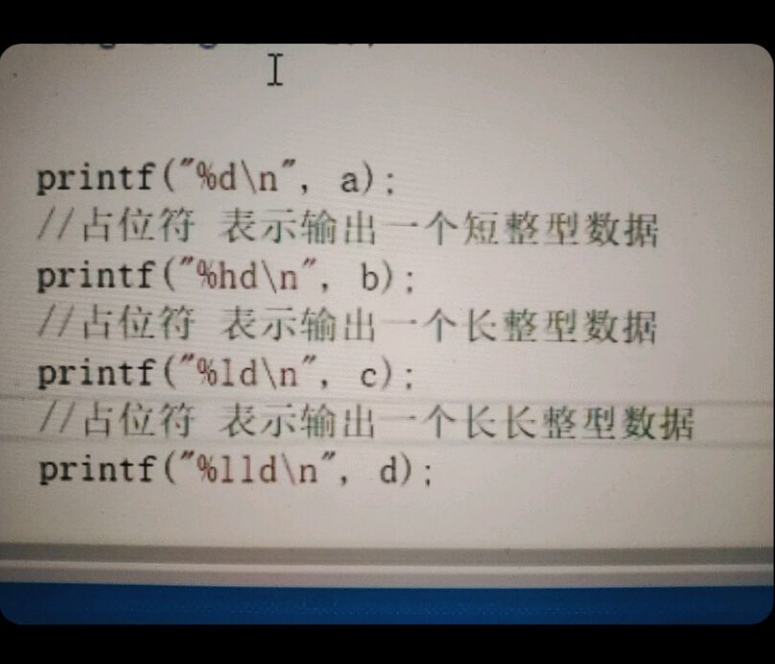
整型变量


2.2.3 short、int、long、long long

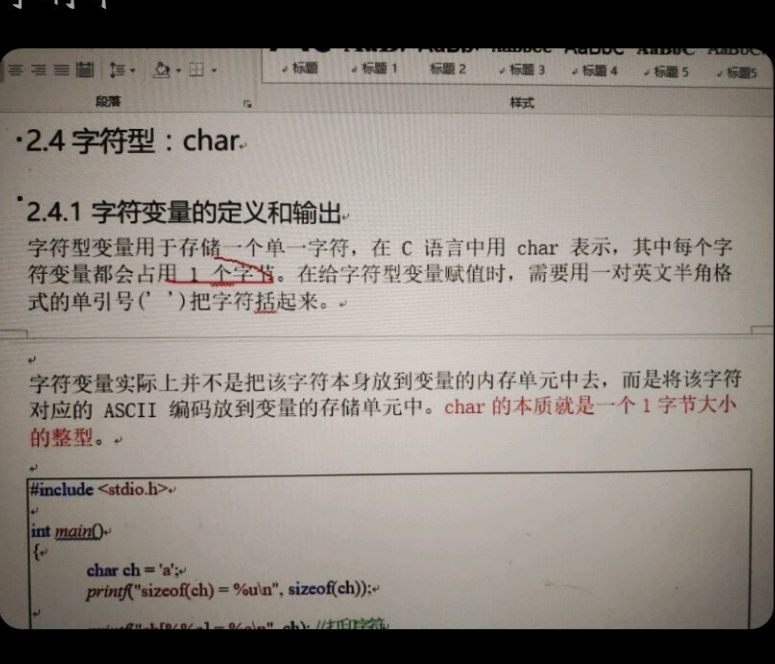
数据类型	占用空间
short(短整型)	2 字节
int(整型)	4 字节
long(长整形)	Windows 为 4 字节，Linux 为 4 字节(32 位)，8 字节(64 位)
long long(长长整形)	8 字节

注意：
- 需要注意的是，整型数据在内存中占的字节数与所选择的操作系统有关。虽然 C 语言标准中没有明确规定整型数据的长度，但 long 类型整数的长度不能短于 int 类型，short 类型整数的长度不能长于 int 类型。
- 当一个小的数据类型赋值给一个大的数据类型，不会出错，因为编译器会自动转化。但当一个大的类型赋值给一个小的数据类型，那么就可能丢失高位。

整型常量	所需类型
10	代表 int 类型


```
printf("%d\n", a);
//占位符 表示输出一个短整型数据
printf("%hd\n", b);
//占位符 表示输出一个长整型数据
printf("%ld\n", c);
//占位符 表示输出一个长长整型数据
printf("%lld\n", d);
```

字符串


2.4 字符型：char

2.4.1 字符变量的定义和输出

字符型变量用于存储一个单一字符，在 C 语言中用 char 表示，其中每个字符变量都会占用 1 个字节。在给字符型变量赋值时，需要用一对英文半角格式的单引号(' ')把字符括起来。

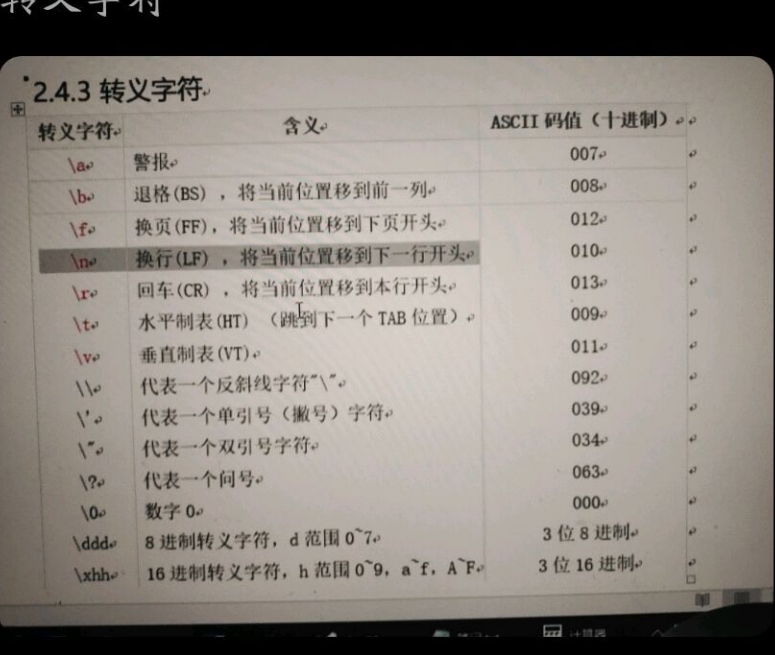
字符变量实际上并不是把该字符本身放到变量的内存单元中去，而是将该字符对应的 ASCII 编码放到变量的存储单元中。char 的本质就是一个 1 字节大小的整型。

```
#include <stdio.h>

int main()
{
	char ch = 'a';
	printf("sizeof(ch) = %u\n", sizeof(ch));
```

字符和字符串的区别

字符串有结束符，会多占一个字节，字符没有

转义字符


2.4.3 转义字符

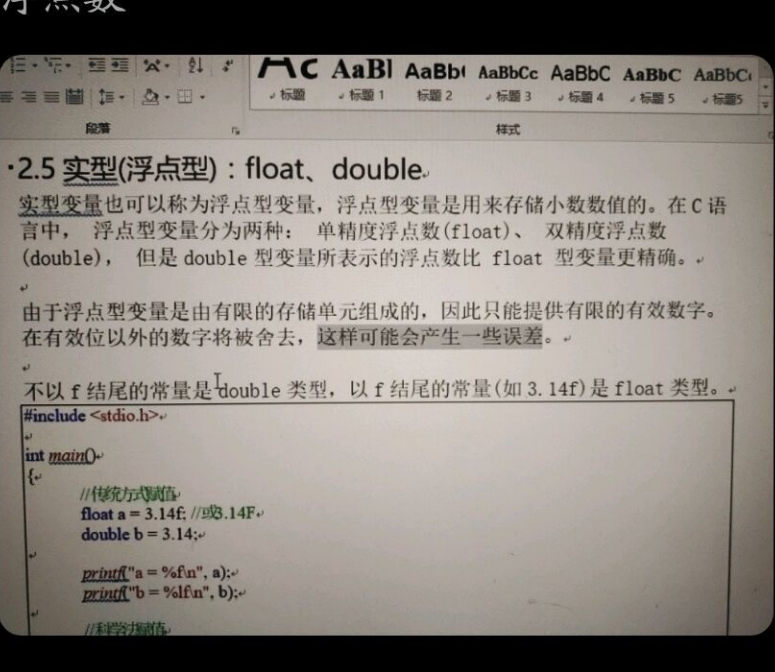
转义字符	含义	ASCII 码值（十进制）
\a	警报	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	换行(LF)，将当前位置移到下一行开头	010
\r	回车(CR)，将当前位置移到本行开头	013
\t	水平制表(HT)（跳到下一个 TAB 位置）	009
\v	垂直制表(VT)	011
\\	代表一个反斜线字符"\"	092
\'	代表一个单引号（撇号）字符	039
\"	代表一个双引号字符	034
\?	代表一个问号	063
\0	数字 0	000
\ddd	8 进制转义字符，d 范围 0~7	3 位 8 进制
\xhh	16 进制转义字符，h 范围 0~9，a~f，A~F	3 位 16 进制

两个反斜杠 等于 一个正斜杠

打印百分号 % 需要 % %

注意小数打印的时候默认都是六位，但是实际上不是六位

浮点数


2.5 实型(浮点型)：float、double

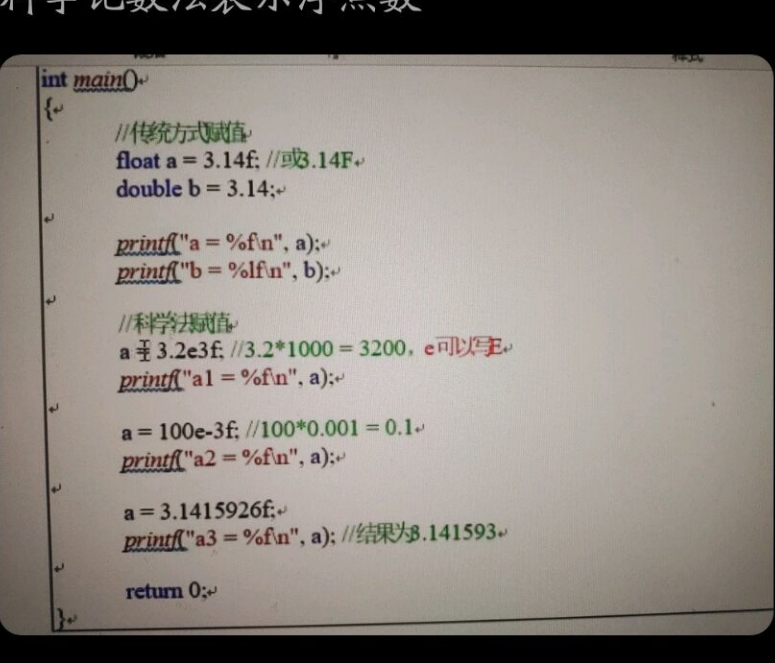
实型变量也可以称为浮点型变量，浮点型变量是用来存储小数数值的。在 C 语言中，浮点型变量分为两种：单精度浮点数(float)、双精度浮点数(double)，但是 double 型变量所表示的浮点数比 float 型变量更精确。

由于浮点型变量是由有限的存储单元组成的，因此只能提供有限的有效数字。在有效位以外的数字将被舍去，这样可能会产生一些误差。

不以 f 结尾的常量是 double 类型，以 f 结尾的常量(如 3.14f)是 float 类型。

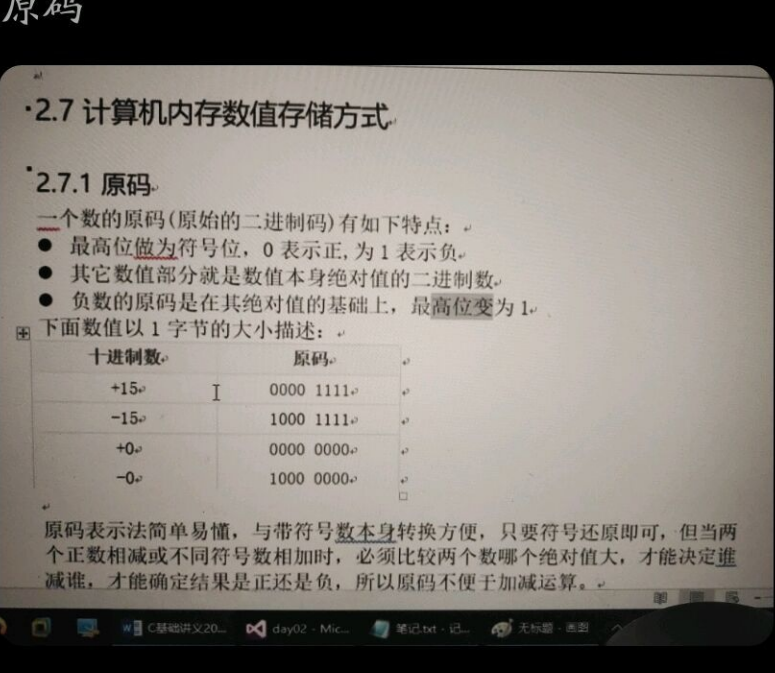
```
#include <stdio.h>

int main()
{
	//传统方式赋值
	float a = 3.14f; //或3.14F
	double b = 3.14;

	printf("a = %f\n", a);
	printf("b = %lf\n", b);
```

科学记数法表示浮点数


```
int main()
{
	//传统方式赋值
	float a = 3.14f; //或3.14F
	double b = 3.14;

	printf("a = %f\n", a);
	printf("b = %lf\n", b);

	//科学法赋值
	a = 3.2e3f; //3.2*1000 = 3200，e可以E
	printf("a1 = %f\n", a);

	a = 100e-3f; //100*0.001 = 0.1
	printf("a2 = %f\n", a);

	a = 3.1415926f;
	printf("a3 = %f\n", a); //结果为3.141593

	return 0;
}
```

计算机内存储数据的方式

原码


2.7 计算机内存数值存储方式

2.7.1 原码

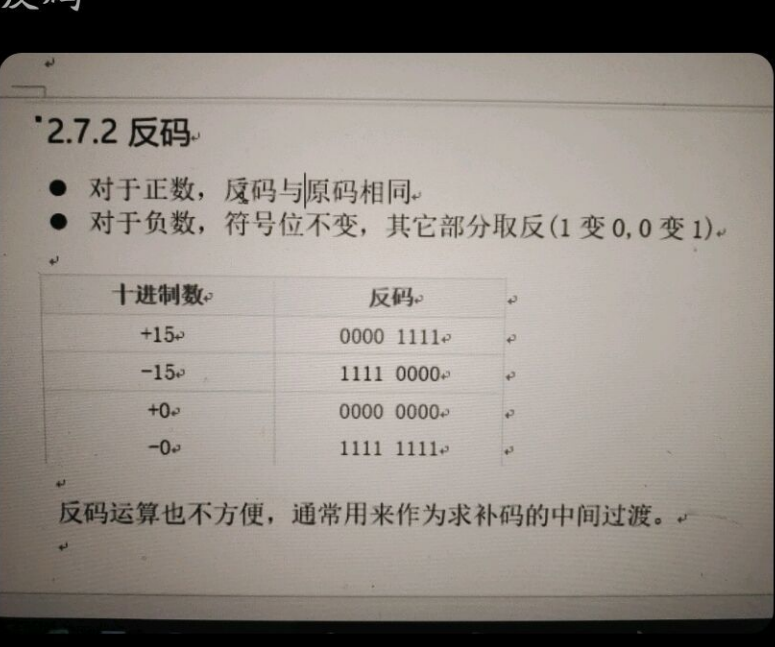
一个数的原码(原始的二进制码)有如下特点：
- 最高位做为符号位，0 表示正，为 1 表示负
- 其它数值部分就是数值本身绝对值的二进制数
- 负数的原码是在其绝对值的基础上，最高位变为 1

下面数值以 1 字节的大小描述：

十进制数	原码
+15	0000 1111
-15	1000 1111
+0	0000 0000
-0	1000 0000

原码表示法简单易懂，与带符号数本身转换方便，只要符号还原即可，但当两个正数相减或不同符号数相加时，必须比较两个数哪个绝对值大，才能决定谁减谁，才能确定结果是正还是负，所以原码不便于加减运算。

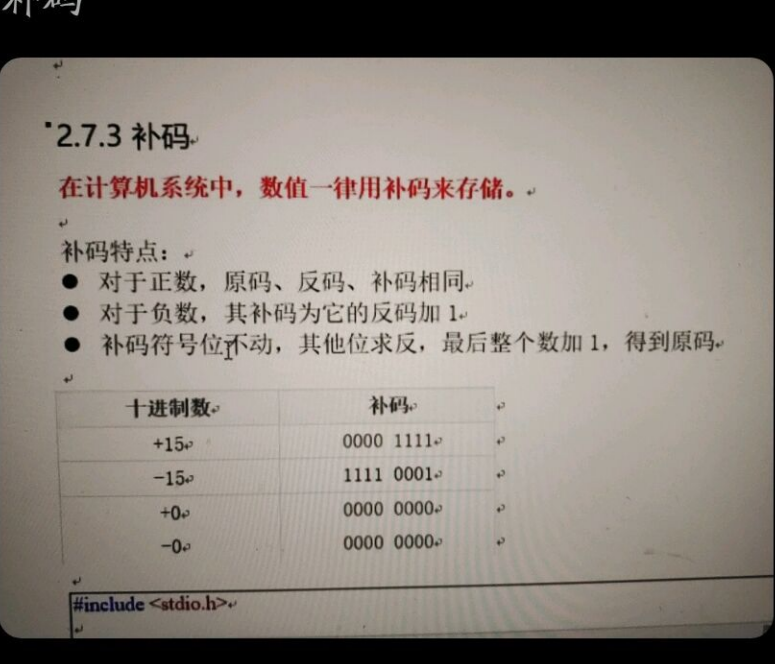
反码


2.7.2 反码
- 对于正数，反码与原码相同。
- 对于负数，符号位不变，其它部分取反(1 变 0，0 变 1)。

十进制数	反码
+15	0000 1111
-15	1111 0000
+0	0000 0000
-0	1111 1111

反码运算也不方便，通常用来作为求补码的中间过渡。

补码


2.7.3 补码

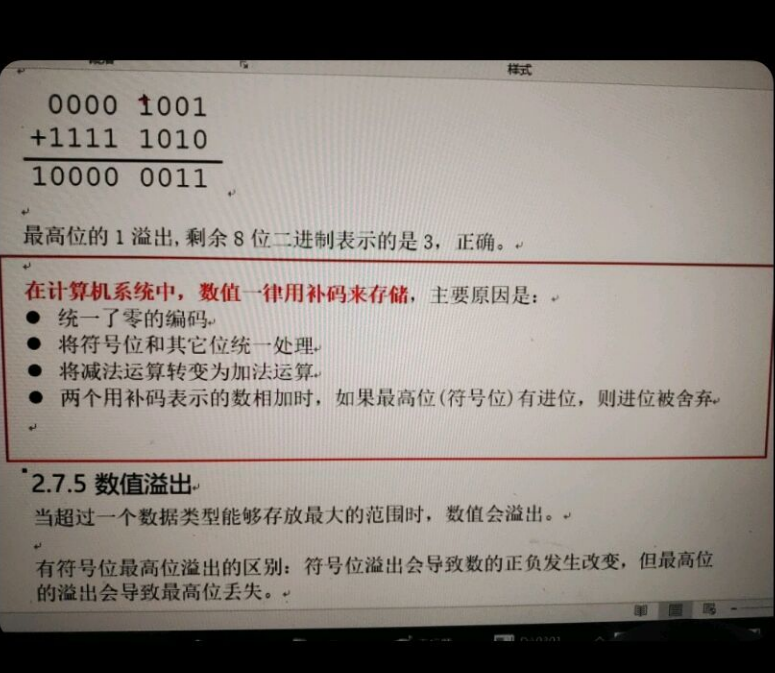
在计算机系统中，数值一律用补码来存储。

补码特点：
- 对于正数，原码、反码、补码相同。
- 对于负数，其补码为它的反码加 1。
- 补码符号位不动，其他位求反，最后整个数加 1，得到原码。

十进制数	补码
+15	0000 1111
-15	1111 0001
+0	0000 0000
-0	0000 0000

#include <stdio.h>

补码求原码就是再补一次


```
  0000 1001
+1111 1010
 10000 0011
```

最高位的 1 溢出，剩余 8 位二进制表示的是 3，正确。

在计算机系统中，数值一律用补码来存储，主要原因是：
- 统一了零的编码
- 将符号位和其它位统一处理
- 将减法运算转变为加法运算
- 两个用补码表示的数相加时，如果最高位(符号位)有进位，则进位被舍弃。

2.7.5 数值溢出

当超过一个数据类型能够存放最大的范围时，数值会溢出。

有符号位最高位溢出的区别：符号位溢出会导致数的正负发生改变，但最高位的溢出会导致最高位丢失。

正负零 在补码中都是0000 0000

八位补码

注意都是补码补码和正常会有点怪，不要奇怪

正数最大0111 1111　127

126　　0111 1110

0　　0000 0000

-127　1000　0001

负数最大1000 0000　-128

零0000　0000

有符号位的存储范围

同理的存储范围

8位　二的七次方/127　二的七次方减一

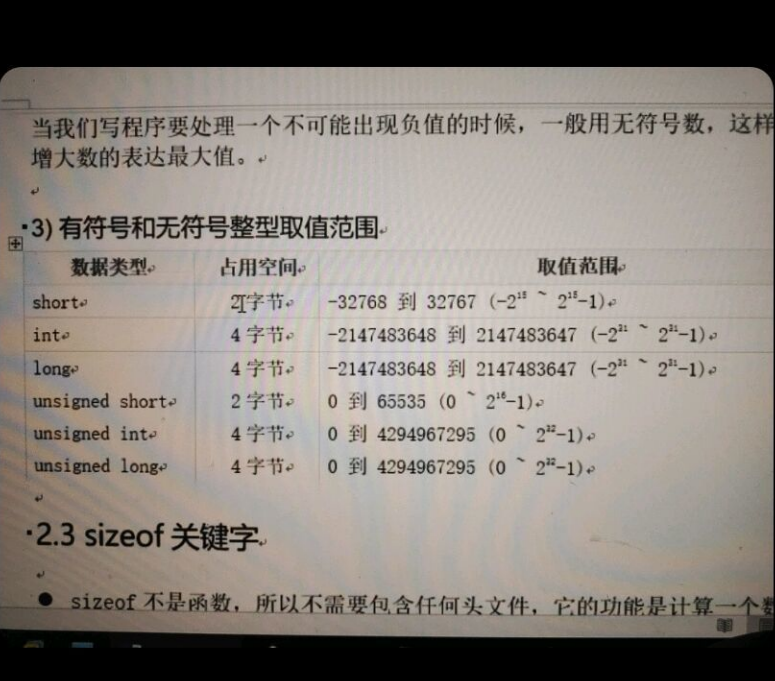
16位　二的十六次方/二十六次方减一

64位同理

无符号数的存储范围

8位　0到255（二的八次方减一）

其他同理


当我们写程序要处理一个不可能出现负值的时候，一般用无符号数，这样增大数的表达最大值。

3) 有符号和无符号整型取值范围

数据类型	占用空间	取值范围
short	2字节	-32768 到 32767 (-2^{15} ~ $2^{15}-1$)
int	4字节	-2147483648 到 2147483647 (-2^{31} ~ $2^{31}-1$)
long	4字节	-2147483648 到 2147483647 (-2^{31} ~ $2^{31}-1$)
unsigned short	2字节	0 到 65535 (0 ~ $2^{16}-1$)
unsigned int	4字节	0 到 4294967295 (0 ~ $2^{32}-1$)
unsigned long	4字节	0 到 4294967295 (0 ~ $2^{32}-1$)

2.3 sizeof 关键字
- sizeof 不是函数，所以不需要包含任何头文件，它的功能是计算一个

没写无符号的关键字，都是有符号

数据溢出

进位和溢出的区别如下：

发生条件不同

进位通常发生在无符号数的运算中，当无符号操作中产生了超过表示范围的结果时，会发生进位；

溢出发生在有符号数的运算中，当有符号操作结果超出了有符号数的表示范围时，即结果无法用有限位数表示时，会发生溢出。a 有符号数也进位，所以需要一个方法来判断目前的进位是溢出还是进位

结果不同 。进位不会导致程序执行错误的结果；溢出可以导致程序执行错误的结果。

有符号数判断是进位还是溢出的方法是根据运算的最高位和次高位是否有进位或借位。

有符号数怎么判断目前是进位还是溢出

在加法运算中，如果次高位相加形成进位，而最高位相加没有进位，则结果溢出；如果次高位无进位，而最高位有进位，则结果溢出。在减法运算中，如果次高位不需要借位，而最高位需要借位，则结果溢出；如果次高位需要借位，而最高位不需借位，则结果溢出。

溢出举例

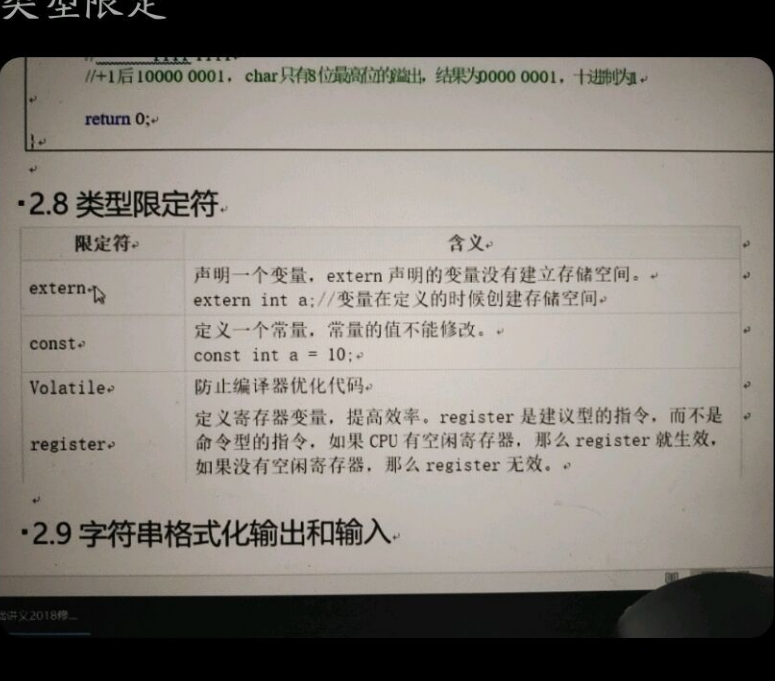
（溢出发生在补码中）

127：　0111 1111　再加1

　　1111 1111　-1

两个正数相加，变成负数

类型限定

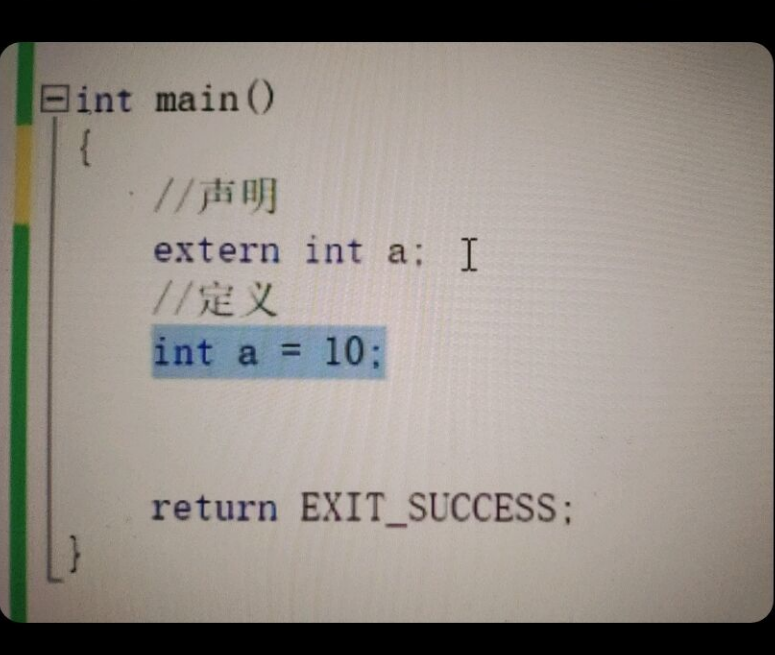

//+1后10000 0001，char只有8位最高位的溢出，结果为0000 0001，十进制为1

return 0;

2.8 类型限定符

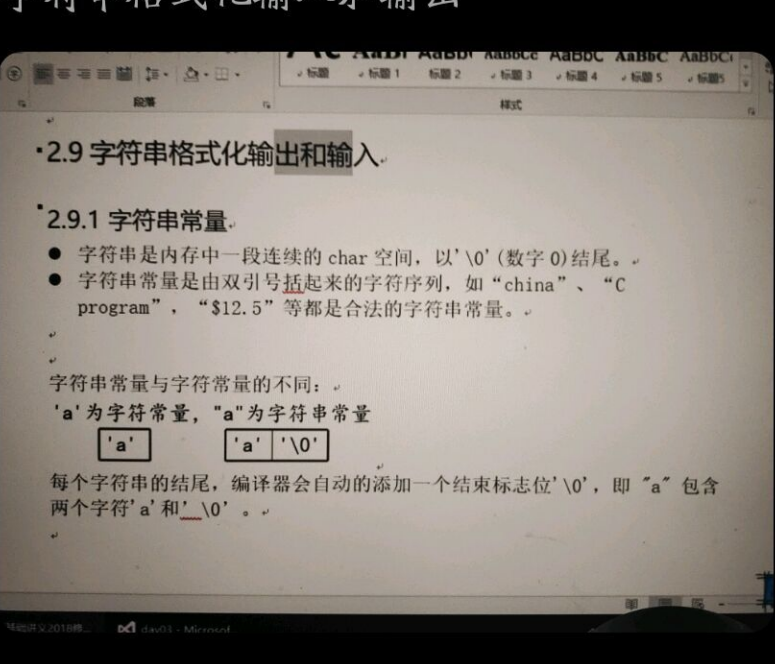
限定符	含义
extern	声明一个变量，extern 声明的变量没有建立存储空间。extern int a;//变量在定义的时候创建存储空间
const	定义一个常量，常量的值不能修改。const int a = 10;
Volatile	防止编译器优化代码
register	定义寄存器变量，提高效率。register 是建议型的指令，而不是命令型的指令，如果 CPU 有空闲寄存器，那么 register 就生效，如果没有空闲寄存器，那么 register 无效。

2.9 字符串格式化输出和输入


```
int main()
{
	//声明
	extern int a;
	//定义
	int a = 10;


	return EXIT_SUCCESS;
}
```

声明不占用内存，定义是一种特殊的声明，

字符串格式化输入和输出


2.9 字符串格式化输出和输入

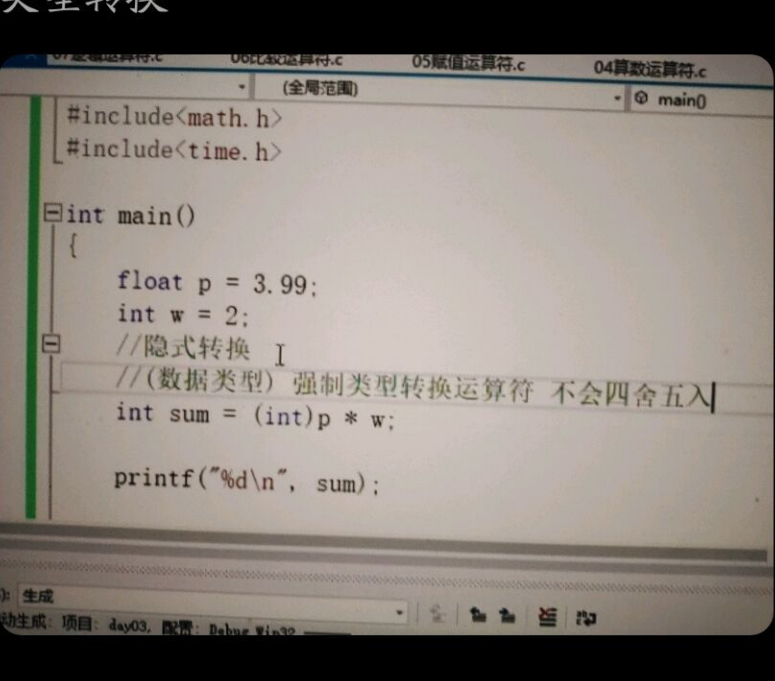
2.9.1 字符串常量
- 字符串是内存中一段连续的 char 空间，以'\0'(数字 0)结尾。
- 字符串常量是由双引号括起来的字符序列，如“china”、“C program”，“$12.5”等都是合法的字符串常量。

字符串常量与字符常量的不同：

'a'为字符常量，"a"为字符串常量

每个字符串的结尾，编译器会自动的添加一个结束标志位'\0'，即 "a" 包含两个字符'a'和'\0'。

类型转换


```
#include<math.h>
#include<time.h>

int main()
{
	float p = 3.99;
	int w = 2;
	//隐式转换
	//(数据类型) 强制类型转换运算符 不会四舍五入
	int sum = (int)p * w;

	printf("%d\n", sum);
```