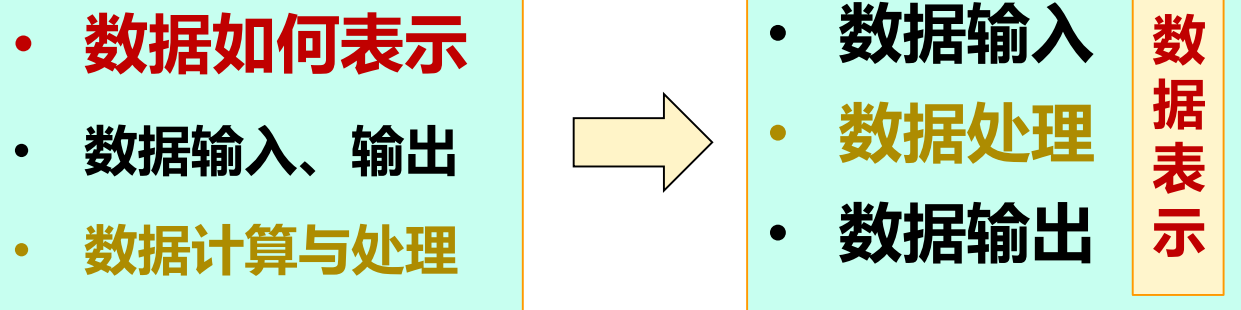


## 第二章(Chapter 2)

# C语言编程概览 overview

### 学习要点

1. 基本的数据类型  
(常量、变量、整型、浮点型、字符型、字符串型)
2. 特别的常量：字符的ASCII编码、转义字符
3. 变量命名规则
4. 变量的赋值、类型转换、访问限定
5. 格式化输入输出 (scanf , printf)
6. 算术运算符、算术表达式
7. 赋值运算、复合赋值运算
8. 关系运算 (符)、逻辑运算 (符)
9. 一维数组简介
10. 常量的符号表示 (define)



- 数据如何表示
- 数据输入、输出
- 数据计算与处理

- 数据输入
- 数据处理
- 数据输出

数据表示

计算机命令我们的动作，我们必须严格遵守，计算机才会为我们工作。

输入(input)

变量的赋值

常量/变量

格式化输入(scanf等)

常量的符号表示(define)

变量命名规则

整型

浮点型

字符型

字符串型

转义符

数组

按程序（规定的动作），  
计算机为我们工作

处理(processing)

复合赋值运算

算术运算（符）

赋值运算

关系运算（符）

位运算

逻辑运算（符）

core

我们命令计算机为我们工作，  
得到我们想要的结果

输出(output)

格式化输出（printf等）



I



P



O

不同类型的数据，输入、输出、处理，都不相同。如何表示数据？如何输入、输出、处理？

# 本讲提纲

2.1 常量

2.2 变量

2.3 输入输出简介

2.4 算术运算

2.5 类型转换

2.6 关系运算与逻辑运算

2.7 运算符的优先级

2.8 一维数组简介

2.9 常量的符号表示方法简介

## 2.1 常量

常量：就是在程序中固定（不可以被程序改变）的数值

	数字	“非” 数字常量
整数	整数 (integer)	字符 (character) 字符串 (string)
非整数	实数 (double, float)	

常数：pi, e(自然常数, 2.71828..),  
phi(黄金分割数, 或黄金比率) , ...

```
#include <stdio.h>
int main()
{
    int a = 1;
    double pi = 3.14;
    char ch = 'A';

    printf("%d\n", a);
    printf("%f\n", pi);
    printf("%c\n", ch);

    return 0;
}
```

# (1) 整数 (integer)

整数的各种进制表示：十进制与二进制、八进制、十六进制

- C语言中默认的表示方式是十进制：0, 123, 45678, -234, -987, -0
- 也可以指定进制（略）
  - ◆ 十六进制：0xA2, 0xFF11（以0x开头）
  - ◆ 八进制：012, 034（以数字零开头）

```
int main()
{
    int a = 19;
    int b = 0x13;
    int c = 023;
    .....
    return 0;
}
```

进制	十进制Dec	八进制Oct	十六进制Hex
基本数字	0 ~ 9	0 ~ 7	0 ~9, A~F (or a~f)
基数	10	8	16
规则	逢10进1	逢8进1	逢16进1
实例	19	023	0x13

Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Oct	00	01	02	03	04	05	06	07	010	011	012	013	014	015	016	017	020	021	022
Hex	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF	0x10	0x11	0x12

## (2) 实数 (浮点数)

实数 (浮点数) (双精度 double, 单精度 float)

- 一般的表示方法

-0.016, 1233.6

0.28 // 表示小数0.28

**.28** // 表示小数0.28

**8000.** // 表示小数8000.0

- 科学计数法

-1.6E-2 // 表示 $-1.6 \times 10^{-2}$

2.8E-1 // 表示0.28

1.2345e3 // 表示1234.5

**8e3** // 表示小数8000.0

- 三个不同: 实数与整数在计算机中的**编码**方式不同; 数值表示的**范围**和**精度**不同; 计算的一些**方法**和**结果**不同。
- 没有小数部分的实数看上去跟整数相同 (数值相同, 但实际上不同, 如8e3为8000.0而不是8000)。
- 默认 (缺省) 的实数类型是double, 要表示float类型 (比double表示的范围和精度小), 在数据后面加后缀f或F, 如:  
-1.6E-2f, .28F, 1233.6f

```
int main()
{
    int a = 1;
    double PI = 3.14;
    double eps = 1e-8;
    float phi = 0.6184f;

    char ch = 'A';
    .....
```

### (3) 字符常量

**字符常量：**用一对单引号括起来字符(character)称为字符常量，如：'A', 'b', '?', ...

**ASCII码：**一个字符常量的值在计算机中通常对应一个唯一的小整数，常用**ASCII码** (American Standard Code for Information Interchange)来表示字符的编码，是一个整数值，如：'A' 的值为65，'b' 的值为98，'?' 的值为63



```
.....  
int main()  
{  
    int a = 1;  
    char ch = 'A';  
    .....  
    printf("%d\n", a);  
    printf("%c\n", ch);  
    printf("%d\n", ch);  
    .....  
}
```

输出

注意：输出中字符常量不带单引号。但是在程序中的字符常量，必须带单引号。

1  
A  
65

### (3) 字符常量

- 用数字表示的字符与数字值是不一样的，如**字符** '0' 的ASCII编码是十进制的 48，而不是**数字**0（两者差值为48）
- 字符实际上是一个小整数（字符的ASCII编码，0~127）
- 数字字符 0~9（即'0' ~ '9'），字母'a' ~ 'z'，'A' ~ 'Z'之间的所有字符都是连续编码的，如：  
'0' ↔ 48, '1' ↔ 49, ... ; 'a' ↔ 97, 'b' ↔ 98, ...
- 字符常量可像其它整数一样参与数值运算，如：  
'5' 的编码等于 '0'+5，即 53； 'a'+2 等于 'c' ；  
'8' - '0' 等于 8

```
char x = '0'; // int x = '0';  
printf("%d\n", x); // 输出整数 48  
printf("%c\n", x); // 输出字符 0
```

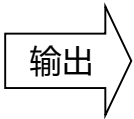
```
int ch = 'a';  
printf("%c\n", ch); // 输出字符 a  
  
ch = 97; // 97是'a'的ASCII码值  
printf("%c\n", ch);  
  
printf("%d\n", ch); // 输出什么
```

两个  
输出  
结果  
相同



# 转义字符

- 在C语言中，还有一类特殊的字符常量，称为转义字符常量，如： '\n', '\0', '\t'
- 输出特殊符号%: `printf("%%");`
- 转义字符输出示例: `printf("\\\\%%\\");`



输出



转义字符及其含义

转义字符	含义	转义字符	含义
\n	换行	\t	水平制表Tab(Table)
\v	垂直制表	\b	退格
\r	回车	\f	换页
\a	响铃	\\	反斜线
\'	单引号	\"	双引号

## 例2-0：转义符的应用

```
#include <stdio.h>

int main()
{
    printf("\\\\%%\\n");
    printf("test\n");
    printf("\"test\"n");
    printf("\\test\\n");

    return 0;
}
```

请读者自行运行程序，观察输出结果，理解转义符的含义。

读者可以写测试代码printf("x\n")来理解各个转义字符的含义（把\n替换为其他转义字符）

# ASCII编码表 (PPT看不清楚? 没关系, 查阅资料! )

ASCII值	字符 (含义)	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUL, 空字符	32	(space)	64	@	96	`
1	SOH, 标题开始	33	!	65	A	97	a
2	STX, 正文开始	34	”	66	B	98	b
3	ETX, 正文结束	35	#	67	C	99	c
4	EOT, 传输结束	36	\$	68	D	100	d
5	ENQ, 请求	37	%	69	E	101	e
6	ACK, 收到通知	38	&	70	F	102	f
7	BEL, 响铃	39	,	71	G	103	g
8	BS, 退格	40	(	72	H	104	h
9	HT, 水平制表符(\t)	41	)	73	I	105	i
10	LF, 换行(\n)	42	*	74	J	106	j
11	VT, 垂直制表符	43	+	75	K	107	k
12	FF, 换页键	44	,	76	L	108	l
13	CR, 回车键	45	-	77	M	109	m
14	SO, 不启用切换	46	.	78	N	110	n
15	SI, 启用切换	47	/	79	O	111	o
16	DLE, 数据链路转义	48	0	80	P	112	p
17	...	49	1	81	Q	113	q
18	...	50	2	82	R	114	r
19	...	51	3	83	X	115	s
20	...	52	4	84	T	116	t
21	...	53	5	85	U	117	u
22	...	54	6	86	V	118	v
23	...	55	7	87	W	119	w
24	...	56	8	88	X	120	x
25	...	57	9	89	Y	121	y
26	...	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	...	60	<	92	/	124	
29	...	61	=	93	]	125	}
30	...	62	>	94	^	126	~
31	...	63	?	95	_	127	DEL

# ASCII码的更多介绍（自学）

ASCII 控制字符 (字符编码: 0-31)						
在ASCII码表中，前32个字符是不能用于打印控制的编码，而是用于控制像打印机一样的外围设备。						
十进制 DEC	八进制 OCT	十六进制 HEX	二进制 BIN	符号 Symbol	HTML 实体编码	中文解释 Description
0	000	00	00000000	NUL	&#000;	空字符
1	001	01	00000001	SOH	&#001;	标题开始
2	002	02	00000010	STX	&#002;	正文开始
3	003	03	00000011	ETX	&#003;	正文结束
4	004	04	00000100	EOT	&#004;	传输结束
5	005	05	00000101	ENQ	&#005;	询问
6	006	06	00000110	ACK	&#006;	收到通知
7	007	07	00000111	BEL	&#007;	铃
8	010	08	00001000	BS	&#008;	退格
9	011	09	00001001	HT	&#009;	水平制表符
10	012	0A	00001010	LF	&#010;	换行键
11	013	0B	00001011	VT	&#011;	垂直制表符
12	014	0C	00001100	FF	&#012;	换页键
13	015	0D	00001101	CR	&#013;	回车键
14	016	0E	00001110	SO	&#014;	移出
15	017	0F	00001111	SI	&#015;	移入
16	020	10	00010000	DLE	&#016;	数据链路转义
17	021	11	00010001	DC1	&#017;	设备控制 1
18	022	12	00010010	DC2	&#018;	设备控制 2
19	023	13	00010011	DC3	&#019;	设备控制 3
20	024	14	00010100	DC4	&#020;	设备控制 4
21	025	15	00010101	NAK	&#021;	拒绝接收
22	026	16	00010110	SYN	&#022;	同步空闲
23	027	17	00010111	ETB	&#023;	传输块结束
24	030	18	00011000	CAN	&#024;	取消
25	031	19	00011001	EM	&#025;	介质中断
26	032	1A	00011010	SUB	&#026;	替换
27	033	1B	00011011	ESC	&#027;	换码符
28	034	1C	00011100	FS	&#028;	文件分隔符
29	035	1D	00011101	GS	&#029;	组分隔符
30	036	1E	00011110	RS	&#030;	记录分离符
31	037	1F	00011111	US	&#031;	单元分隔符

ASCII 打印字符 (字符编码: 32-127)						
32 ~ 126(共95个)是字符：32是空格，其中48 ~ 57为0到9十个阿拉伯数字，65 ~ 90为26个大写英文字母，97 ~ 122号为26个小写英文字母，其余为一些标点符号、运算符号等。第127个字符表示的是键盘上的删除命令。						
十进制 DEC	八进制 OCT	十六进制 HEX	二进制 BIN	符号 Symbol	HTML 实体编码	中文解释 Description
32	040	20	00100000		&#032;	空格
33	041	21	00100001	!	&#033;	感叹号
34	042	22	00100010	"	&#034;	双引号
35	043	23	00100011	#	&#035;	井号
36	044	24	00100100	\$	&#036;	美元符
37	045	25	00100101	%	&#037;	百分号
38	046	26	00100110	&	&#038;	与
39	047	27	00100111	'	&#039;	单引号
40	050	28	00101000	(	&#040;	左括号
41	051	29	00101001	)	&#041;	右括号
42	052	2A	00101010	*	&#042;	星号
43	053	2B	00101011	+	&#043;	加号
44	054	2C	00101100	,	&#044;	逗号
45	055	2D	00101101	-	&#045;	连字号或减号
46	056	2E	00101110	.	&#046;	句点或小数点
47	057	2F	00101111	/	&#047;	斜杠
48	060	30	00110000	0	&#048;	0
49	061	31	00110001	1	&#049;	1
50	062	32	00110010	2	&#050;	2
51	063	33	00110011	3	&#051;	3
52	064	34	00110100	4	&#052;	4
53	065	35	00110101	5	&#053;	5
54	066	36	00110110	6	&#054;	6
55	067	37	00110111	7	&#055;	7
56	070	38	00111000	8	&#056;	8
57	071	39	00111001	9	&#057;	9
58	072	3A	00111010	:	&#058;	冒号
59	073	3B	00111011	;	&#059;	分号
60	074	3C	00111100	<	&#060;	小于
61	075	3D	00111101	=	&#061;	等号
62	076	3E	00111110	>	&#062;	大于
63	077	3F	00111111	?	&#063;	问号

# 需要但又忘记ASCII码表时？

## 例2-1: ASCII的字符形式(ascii2char)

```
int i, ch;
printf("      ");
for(i=0; i<=9; i++)
    printf("%2c ", '0'+i); // 输出表头（行）
printf("\n");

printf("%2c: ", '0') ;
for(i=0; i<=127; i++)
{
    if( ((i+1)%10) != 0 )
        printf("%2c ", i); // 输出某行前9个
    else
    {
        printf("%2c\n", i); // 输出某行最后一个，然后换行
        printf("%2d: ", i/10+1); // 输出表头（列）
    }
}
```

输出

D:\a1ac\example\chap2\c2-1.exe

	0	1	2	3	4	5	6	7	8	9
0:										
1:										
2:										
3:				!	"	#	\$	%	&	'
4:	(	)	*	+	,	-	.	/	0	1
5:	2	3	4	5	6	7	8	9	:	;
6:	<	=	>	?	@	A	B	C	D	E
7:	F	G	H	I	J	K	L	M	N	O
8:	P	Q	R	S	T	U	V	W	X	Y
9:	Z	[	\	]	^	_	`	a	b	c
10:	d	e	f	g	h	i	j	k	l	m
11:	n	o	p	q	r	s	t	u	v	w
12:	x	y	z	{		}	~			

# 需要但又忘记ASCII码表时？

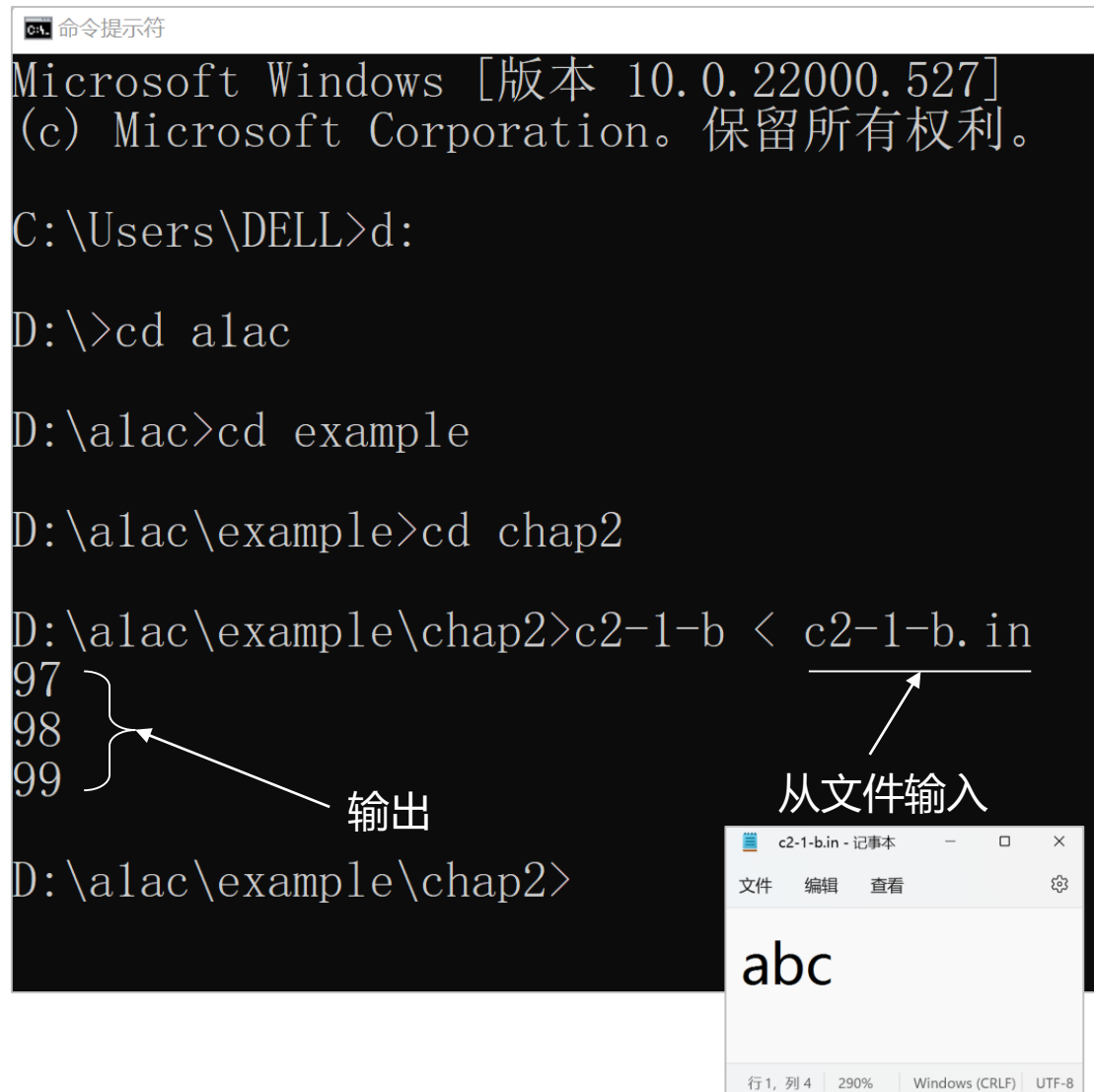
## 例2-1-b: 字符的ASCII形式(char2ascii)

```
// c2-1-b.c
#include <stdio.h>

int main( )
{
    char x;
    while(scanf("%c", &x) != EOF)
    {
        printf("%d\n", x);
    }

    return 0;
}
```

注: EOF (End of File, 是一个常量 -1 )表示输入时读到文件末尾, 2.3节详述。



The screenshot shows a Windows command prompt window titled "命令提示符" (Command Prompt) with the following text:

```
Microsoft Windows [版本 10.0.22000.527]
(c) Microsoft Corporation。保留所有权利。

C:\Users\DELL>d:
D:\>cd alac
D:\alac>cd example
D:\alac\example>cd chap2
D:\alac\example\chap2>c2-1-b < c2-1-b.in
97
98
99
```

Arrows point from the text "输出" (Output) to the numbers 97, 98, and 99, and from "从文件输入" (Input from file) to the command `c2-1-b < c2-1-b.in`.

In the bottom right corner, a Notepad window titled "c2-1-b.in - 记事本" (c2-1-b.in - Notepad) is open, showing the text "abc". The status bar at the bottom of the Notepad window indicates "行 1, 列 4" (Line 1, Column 4), "290%", "Windows (CRLF)", and "UTF-8".

## (4) 字符串常量

字符串常量：用一对双引号括起来的字符串，如 "hello"

- 注意：所有字符串均以 '\0' 结束（编码值为 0 的字符，是不可见的字符，即，“存在”但“看不见”），因此，"x" 和 'x' 不同，"x" 其实包括两个字符 'x' 和 '\0'。字符串末尾的 '\0' 由编译程序自动添加。
- 字符串中可以有转义字符，如：

```
printf("Welcome \n to \nBeijing!\n");
```

输出

Welcome  
to  
Beijing!

```
printf("\"%%This is a string\"\n");
```

输出

"%This is a string"

💡 编程再提示：特别的转义符  
输出%的用法： `printf("%%");`

## 2.2 变量

常量是固定的值，存储常量的，则就称为变量。

常用的变量基本类型有：

数据类型	关键字
整型 (integer)	<b>int</b> , short, <b>long long</b> , unsigned...
实型 (float, double )	float, <b>double</b>
字符 (character)	<b>char</b>
字符串 (string)	char型的数组 <b>char []</b> 或指针 <b>char *</b>

## 2.2 变量

### 例2-2: a+b回顾

```
int a, b, sum;  
  
scanf("%d %d", &a, &b);  
sum = a + b;  
printf("%d + %d = %d\n", a, b, sum);
```

#### B 星幽增强

时间限制: 1000ms 内存限制: 65536kb

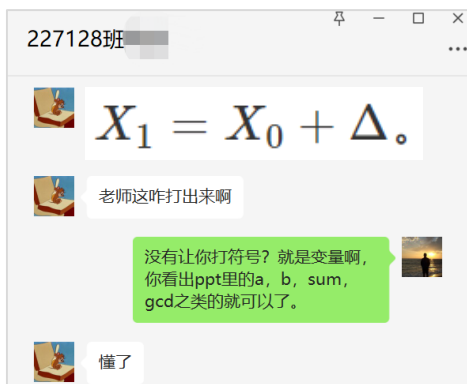
通过率: 1222/1288 (94.88%) 正确率: 1222/1980 (61.72%)

#### 题目描述

式部茉优需要计算学生们的星幽能力值经过增强化后的数值。现有原能力值  $X_0$  和加值  $\Delta$ ，需要你计算出增强后的数值  $X_1$ ，满足  $X_1 = X_0 + \Delta$ 。

#### 变量定义

- ◆ int: 变量类型（整型）（还可以定义其他类型，如 float, double, char...）
- ◆ 定义三个整型变量，分别为 a, b, sum，称为变量名。
- ◆ 定义多个变量时，用逗号分开。也可以分开定义，如  
int a;  
int b;  
int sum;
- ◆ 变量名命名规则（后文介绍）
- ◆ 变量是给人看的，在计算机中，变量会对应一个存储数据的位置（内存地址），因此，只要符合命名规则，任意定义变量，都能实现相应的功能。





# 变量命名规则

## 例2-2: a+b回顾

```
int a, b, sum;

scanf("%d %d", &a, &b);
sum = a + b;
printf("%d + %d = %d\n", a, b, sum);
```

### B 星幽增强

时间限制: 1000ms 内存限制: 65536kb

通过率: 1222/1288 (94.88%) 正确率: 1222/1980 (61.72%)

#### 题目描述

式部荣优需要计算学生们的星幽能力值经过增强后的数值。现有原能力值  $X_0$  和加值  $\Delta$ ，需要你计算出增强后的数值  $X_1$ ，满足  $X_1 = X_0 + \Delta$ 。

- ◆ **变量命名规则**：变量的名字必须是合法的标识符（identifier）。一个标识符是由字母（或\_）开头，由字母、数字和下划线组成的字符串。
- ◆ **系统保留的关键字不能作为标识符**，如 main, int, return 等。
- ◆ 标识符在C语言中可作为变量名、函数名、参数(parameter)名、自定义类型名、枚举(enumeration)名和标号(label)等。

## 下面哪些是非法的标识符？

Beijing	beijing	C_Programming
a8673	8a673	_1024
sin(25)	c[2]	x6    b.3    sum*co
#stu	a2	
xMin	topLeft	numOfStdudent
x_min	top*left	num_of_stdudent

- ◆ C语言区分大小写，因此变量名 Beijing 和 beijing 不一样。
- ◆ 为了增加程序的可读性，变量名通常由表示特定含义的英语单词组成，几个单词（或其缩写）由下划线或单词首字母大写连接在一起。

## (变量) 命名法

---

- 遵循一致的命名约定会对提高代码的可用性起到重大作用
- 使得许多开发人员使用统一框架成为可能
- 遵守命名规范，帮助你在阅读代码的时候快速跟踪代码逻辑

# (变量) 命名法

常用的变量命名法：匈牙利法、骆驼法、下划线法、帕斯卡 (pascal) 法

## (1) 匈牙利命名法

- ◆ 为了纪念匈牙利籍的Microsoft程序员 Charles Simonyi (查尔斯·西蒙尼) (Excel 早期版本的主要成员 )
- ◆ 在变量名前面加上相应的小写字母的符号标识作为前缀，标识出变量的作用域、类型等。

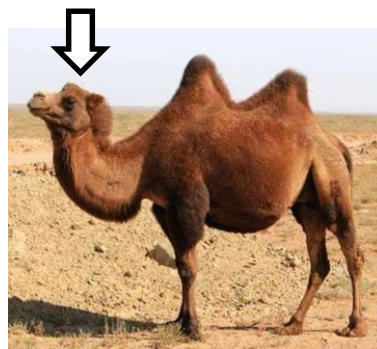
```
int i_num;  
char c_name;  
double d_averScore;
```

# (变量) 命名法

## (2) 骆驼命名法

- ◆ 混合使用大小写字母来构成变量和函数的名字。
- ◆ 近年来越来越流行了，在许多新的函数库和Microsoft Windows这样的环境中，它使用得相当多。

头下去！



```
double stuScore;  
printEmployeePaychecks();
```

名字中的每一个逻辑断点都由一个大写字母来标记

流量明星，通常用于函数名

## (变量) 命名法

### (3) 下划线法

下划线法是C出现后开始流行起来的，在许多经典的程序和UNIX这样的环境中，它的使用非常普遍。

```
double stu_score;  
print_employee_paychecks();
```

名字中的每一个逻辑断点都由一个下划线来标记

老而弥坚，通常用于变量名

## (变量) 命名法

### (4) 帕斯卡 (pascal) 命名法

- ◆ 帕斯卡命名法与骆驼命名法类似。只不过骆驼命名法是首字母小写，而帕斯卡命名法是首字母大写。
- ◆ 在C#中，以帕斯卡命名法和骆驼命名法居多。

```
double StuScore;  
public void DisplayInfo();  
string UserName;
```

重点突出，建议用于文件名

# (变量) 命名法风格小结

如下是什么命名法?

```
int i_num;
```

```
double stuScore;
```

```
double stu_score;
```

```
double StuScore;
```

## 💡 编程提示

变量命名时:

- 风格同一性
- 最小化长度&&最大化信息量原则
- 避免过于相似, 如 stu, Stu
- 避免易混淆的字符, 如 score1, scorel, scoreI
- 避免在不同级别的作用域中重名

采用哪种风格, 跟个人的喜好、团队的规则等有关, 没有绝对标准, 但**不建议**在一个开发任务中“无序”使用, 造成“混乱”!

```
double stuScore;
```

```
int stu_num;
```





# C语言保留的关键字

关键字(*keyword*): 已经被系统使用的标识符, 具有预先定义好的特殊意义, 不能作为变量名、函数名及标号等使用。

int	struct	auto	goto	if
float	union	static	return	else
char	void	extern	break	while
short	enum	register	continue	for
long	typedef			do
unsigned	sizeof			switch
double	const			case
signed	volatile			default

define, undef, include, ifdef, ifndef, endif, 及line, 虽不是关键字, 但是最好把它们看作关键字, 它们主要用于C预处理程序中。

## 【\*】基本的数据类型及其能表示的范围

数据类型	类型含义	数据长度（二进制位数）	位数说明
int	整型	32	数据长度是目前常用编译器的长度，C语言并没有严格规定，不同平台、不同编译器可能有所不同。 int前可以加若干长度修饰符号。char前面也可以加unsigned修饰符号，即只表示不小于0的部分。
short (int)	短整型	16	
long long (int)	长整型	64	
unsigned (int)	无符号整型	32	
float	单精度浮点型	32	
double	双精度浮点型	64	
char	字符型	8	

1. 在C语言中，数据是有范围的；如对于32位机的int，其数值范围为  
 $-2,147,483,648 \sim 2,147,483,647$  ( $-2^{32-1} \sim 2^{32-1}-1$ )

0	0	0
1	0	1
2	1	0
3	1	1

2. 在不同的系统平台，同一数据类型（如int）范围可能不同。  
一般来说，不同的系统（如Windows, Linux），相同数据类型长度可能有差异。但有个原则是：短整型(short)不能长于普通整型(int)；长整型(long)不能短于普通整型(int)。
3. 浮点数使用标准数据格式（IEEE-754），float的有效数字大约相当于十进制的7位，表示范围大约为  $-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$ ，能表示的绝对值最小数约为  $10^{-44.85}$ ；double能表示的范围和精度更大。浮点数的表示是近似值，如显示的是1.0，计算机中实际可能是0.99999999...，也可能是1.00000001...。

# a+b回顾

## 例2-2: a+b回顾

```
int a, b, sum;
```

```
scanf("%d %d", &a, &b);
```

```
sum = a + b;
```

```
printf("%d + %d = %d\n", a, b, sum);
```

运行程序

100

200

100 + 200 = 300

再运行，分别输入十亿和二十亿

1000000000

2000000000

结果不正确。为什么？

# 变量的赋值和类型转换

```
#include <stdio.h>
int main( )
{
    int a, b; // 定义变量
    double d;

    a = 55;
    b = a; } // b = a = 55;
    d = 123.456;

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

    return 0;
}
```

- ◆ **变量定义后才能使用。**
- ◆ **赋值运算符 ‘=’**：表示把运算符右边的值（常量、变量、或表达式，如 `b=a+5`）赋给左边。
- ◆ **赋值运算可以多个连写，此时从右往左结合。**如  
`b = a = 55;`

输出?

`a = 55, b = 55`  
`d = 123.456000`

# 变量的赋值和类型转换

```
#include <stdio.h>
int main( )
{
    int a, b; // 定义变量
    double d;

    a = 55;
    b = a; // b = a = 55;
    d = 123.456;

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

    return 0;
}
```

增加两条语句?

输出

a = 55, b = 55  
d = 123.456000

```
#include <stdio.h>
int main( )
{
    int a, b;
    double d;

    a = 55;
    b = a; // b = a = 55;
    d = 123.456;

    a = d; // 两边类型不一致!
    d = a; // 两边类型不一致!

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

    return 0;
}
```

输出?

💡 编程提示

当赋值运算左右两边**类型不一致**时，可能会出现数据损失。

## 变量的初始化

```
#include <stdio.h>
int main( )
{
    int a, b;

    b = 60;

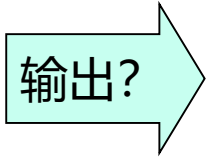
    printf("a = %d\nb = %d", a, b);

    return 0;
}
```

变量赋值后才能访问：

当局部变量定义时没有赋值，  
变量的值是不确定的；

强行访问，得到的值不可信。



输出?

## 2.3 数值输入输出简介

- 标准库函数 `stdio.h` 提供相应的 I/O 操作，常用的是两个函数：`scanf()`, `printf()`  
[ `stdio.h` : standard input output . header ]
- 标准输入函数：`scanf`，其函数原型（使用格式）为  
`int scanf(char *format, ...);` 【例 `scanf("%d%d", &a, &b);`】

```
#include <stdio.h>
int main()
{
    int a;
    double x;
    char ch;
    scanf("%d", &a);
    scanf("%lf", &x);
    scanf("%c", &ch);
    ...
}
```

格式控制字符 (输入占位符)	输入形式	对应参数类型	输入实例	备注
%d	把输入解析为有符号的十进制整数	int	+123	
%x	十六进制整数形式	int	8C	
%c	一个字符	char	x	
%s	一个字符串（不含空白符）	字符数组或字符指针	hello123 yes	yes不会被输入
%f	小数形式（单精度浮点数）	float	-1.23	
%lf	小数形式（双精度浮点数）	double	-1.23	

# 数值输入输出：标准输入函数 scanf

scanf, 其函数原型（使用格式）为：int scanf(char \*format, ...); 【例 scanf("%d%d", &a, &b); 】

- **format**格式控制串是占位符字符串，其中的转换控制字符（以%开始）（占位符）决定了其他参数（输入数据）的个数和输入格式。
- 输入数值类型，如 %d, %f, %lf 时，scanf 会跳过输入数据前的空格符、制表符(Tab)、换行符等空白符（也把空白符看成多个数值输入的分隔符），从第一个有效**数值型**字符开始读入数据并尝试按规定格式转换（转换成功，数据读入内存（变量）；转换失败，读入停止）。
- **format**格式串中其它字符必须原样输入。
- scanf( )函数返回**成功赋值的数据项数**，读到文件末尾而没有数据时返回 EOF ( -1 )。

```
int x, y, z, i;  
i = scanf("%d%d,%d", &x, &y, &z);  
printf("%d\n", i);  
printf("%d\n", x);  
printf("%d\n", y);  
printf("%d", z);
```

运行时分别给出如下**输入**，输出是什么？

5 6,7

5 6 7

5  
6,7

5 6,7



# 数值输入输出： 1) 常见的编程平台实例，单组数据输入

## 例2-2：a+b回顾

输入两个数a和b，求a+b的值（输入输出都在int范围）。

【输入两个数 $X_0$ 和 $\Delta$ ，求 $X_0+\Delta$ 的值】

输入：一行，分别为整数 a 和 b（空格分开）。

输出：一个数，表示a+b的值。

输入样例：

1 2

输出样例：

3

此问题的程序可以这样写：

```
#include <stdio.h>

int main()
{
    int a, b;

    scanf("%d%d", &a, &b);
    printf("%d", a+b);

    return 0;
}
```

### B 星幽增强

时间限制：1000ms 内存限制：65536kb

通过率：1222/1288 (94.88%) 正确率：1222/1980 (61.72%)

#### 题目描述

式部茉优需要计算学生们的星幽能力值经过增强化后的数值。现有原能力值  $X_0$  和加值  $\Delta$ ，需要你计算出增强后的数值  $X_1$ ，满足  $X_1 = X_0 + \Delta$ 。

## 数值输入输出： 2) 常见的编程平台实例，多组数据输入

### 例2-3: a+b进阶

输入两个数a和b，求a+b的值  
(输入输出都在int范围)。

输入：多组数据，每组一行，分别为整数 a 和 b (空格分开)。

输出：对每行输入，输出为一个数，表示该输入行的和。

输入样例：

1 2

5 6

输出样例：

3

11

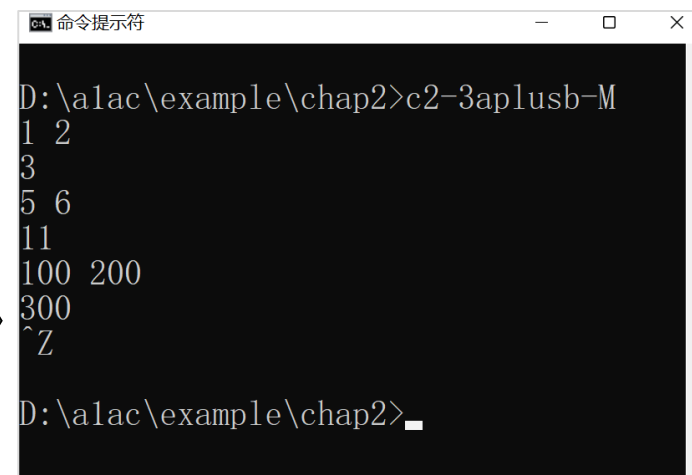
程序可以这样写：

```
#include <stdio.h>

int main()
{
    int a, b;

    while (scanf("%d%d", &a, &b) != EOF)
    {
        printf("%d\n", a + b);
    }
    return 0;
}
```

运行示例



在IDE或命令行下运行测试时，每输入两个数，立刻就会输出一个数（两数之和），跟题目中描述的输入（输出）样例都排列在一起显示的效果不太一样。在计算机中，输入与输出在物理上通常是两个不同的文件，但在这里，输入和输出都是同一个控制台程序，交替使用，因此结果依序显示到屏幕上。在windows系统，按键 ctrl+Z 可结束输入。

## 数值输入输出：2) 常见的编程平台实例，多组数据输入

“输入、输出样例”与“运行测试”呈现方式不一样？

输入样例：

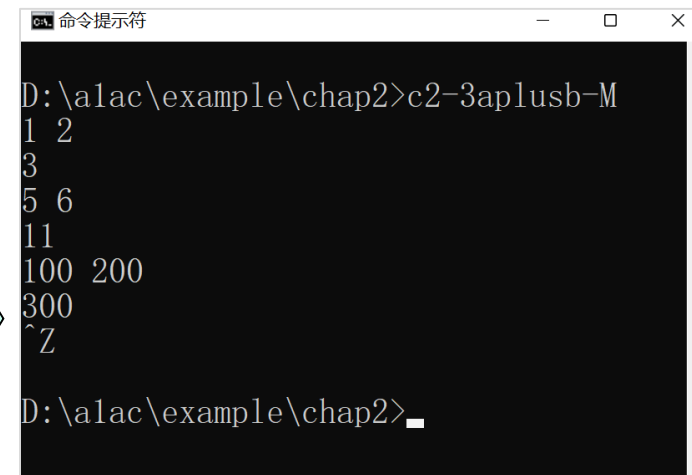
1 2  
5 6

输出样例：

3  
11

```
#include <stdio.h>
int main()
{
    int a, b;
    while (scanf("%d%d", &a, &b) != EOF)
    {
        printf("%d\n", a + b);
    }
    return 0;
}
```

运行示例



```
D:\a1ac\example\chap2>c2-3aplusb-M
1 2
3
5 6
11
100 200
300
^Z
3
11
303
^Z
D:\a1ac\example\chap2>_
```

输入和输出都在同一个控制台程序界面上，于是，得到第1组输入，计算后显示对应的输出，接着进行第2组.....

知识点：scanf() 函数有返回值。若输入成功，返回输入的数据项数；输入错误返回已成功读入的数据个数；读到文件末尾时返回 EOF ( -1 )。

## 数值输入输出： 2) 常见的编程平台实例， 多组数据输入

这个程序有  
“bug” ？

怎么办？

```
#include <stdio.h>
int main()
{
    int a, b;

    while (scanf("%d%d", &a, &b) != EOF)
    {
        printf("%d\n", a + b);
    }

    return 0;
}
```

运行测试

2 3

5

8 d

11

11

11

11

.

.

.

第1组输入，格式正确

第1组输出，结果正确

第2组输入，敲错了！

输出根本就停不下来！  
程序进入死循环（程序不是真的死了，而是永生，程序永远不停地在运行）。

知识点：scanf( )函数有返回值。若输入成功，返回输入的数据项数；  
输入错误返回已成功读入的数据个数；读到文件末尾时返回 EOF ( -1 )。

## 数值输入输出：2) 常见的编程平台实例，多组数据输入

这个程序有 “bug” ？ 怎么办？

```
#include <stdio.h>
int main()
{
    int a, b;

    while (scanf("%d%d", &a, &b) != EOF)
    {
        printf("%d\n", a + b);
    }

    return 0;
}
```

2 3  
5  
8 d  
11  
11  
11  
11  
.  
.  
.

死  
循  
环  
了

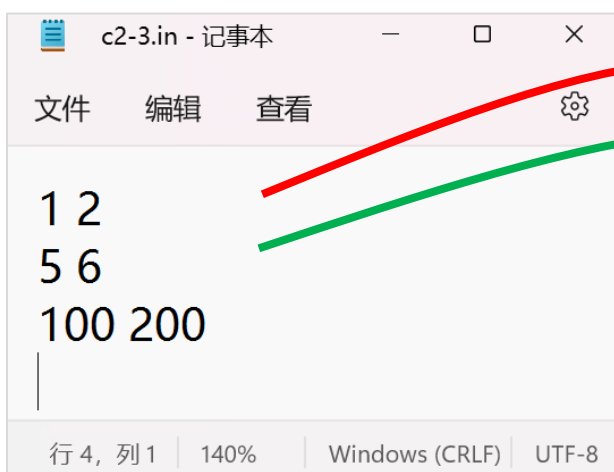
有很多其他实现方法，可以进行输入的容错处理。

但本代码满足了题意要求，输入出错导致的“闪屏”是程序员犯的错误，不能完全怪到计算机头上（因为计算机完全按人的要求在工作）！

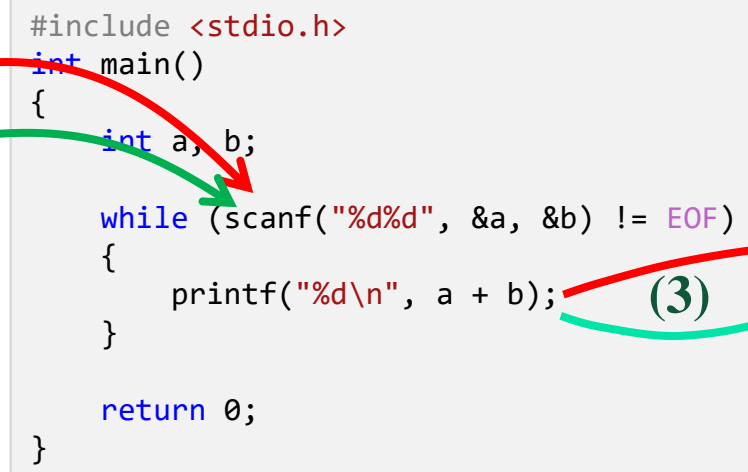
## 数值输入输出： 3) 从文件进行输入重定向

### 例2-3: a+b进阶

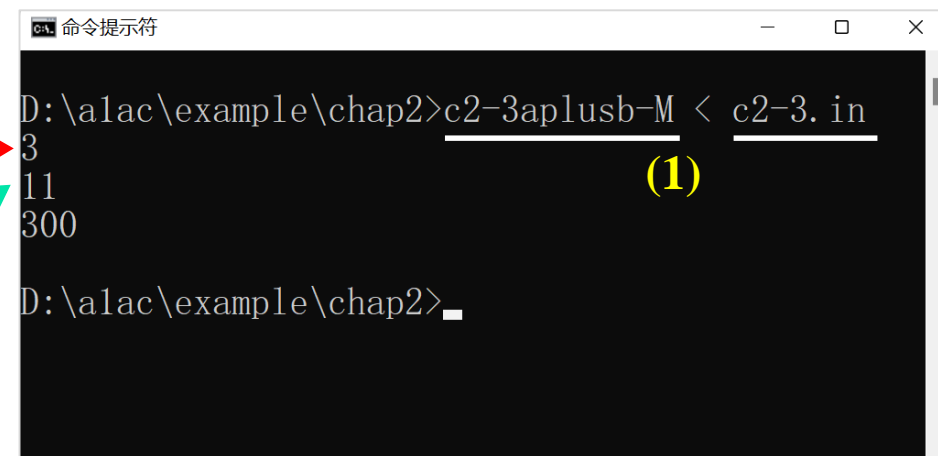
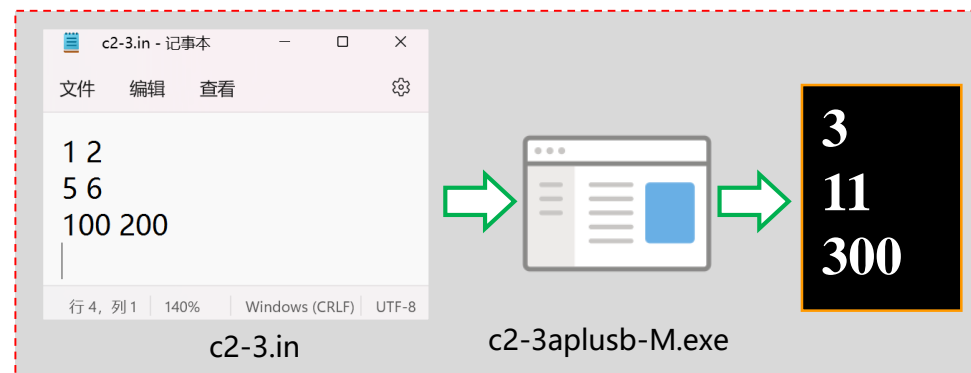
模拟 OJ 评测机的输入原理（从文件中输入）。输出与评测的过程（即，输出也放到文件中），后文再介绍。



```
c2-3.in - 记事本
文件 编辑 查看
1 2
5 6
100 200
行 4, 列 1 140% Windows (CRLF) UTF-8
```



```
#include <stdio.h>
int main()
{
    int a, b;
    while (scanf("%d%d", &a, &b) != EOF)
    {
        printf("%d\n", a + b);
    }
    return 0;
}
```



```
命令提示符
D:\a1ac\example\chap2>c2-3apusb-M < c2-3.in
3
11
300
D:\a1ac\example\chap2>
```

- (1) 运行程序 `c2-3apusb-M.exe`，用流输入运算符 `<`，从文件 `c2-3.in` 进行输入（输入数据提前在文件中准备好）
- (2) 循环读入一组数（两个数），若读入成功，则到第(3)步；若读入不成功（读到文件结束时），结束循环
- (3) 对每组输入数进行计算，输出，然后回到第(2)步

# 数值输入输出：标准输出函数 printf

## printf 其函数原型为

`int printf(char *format, ...);` 【例 `printf("%d\n%d\n", a, b);`】

格式控制字符 (输出占位符)	输出形式	对应参数类型	输出实例	备注
%d	解析为有符号的十进制整数形式	int	123	
%x	十六进制整数形式	int	8C	
%c	一个字符	char, int	x	
%s	一个字符串	字符数组或字符指针	hello123 yes	"..."中的...被输出，如 <code>printf("%s", "hello123 yes");</code>
%f	小数形式	double	-1.23	没有特定的float输出
%e	小数形式（科学计数表示）	double	-1.23e-3	0.00123

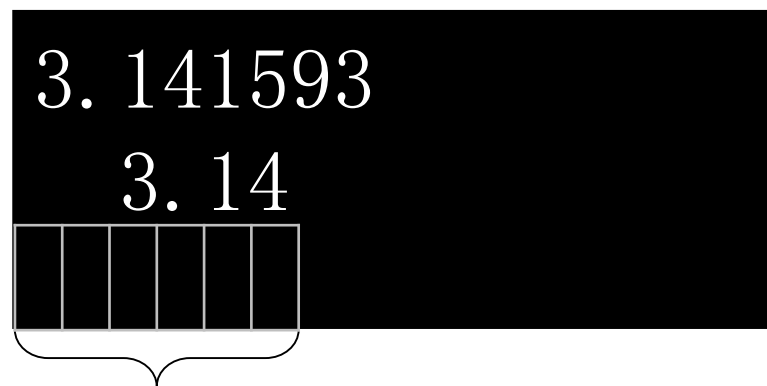
在控制字符前还可以加数字，如：

%-4d：输出最小域宽为4个字符的整数，左对齐（右边如有富裕补空格）。

%6.2f：输出最小域宽为6个字符的浮点数，并且小数点后占两位。

## 格式化输出实例

```
#include <stdio.h>
int main()
{
    double y = 3.14159265;
    printf("%f\n", y);
    printf("%6.2f\n", y);
    return 0;
}
```



3.141593  
3.14

占位符含义：

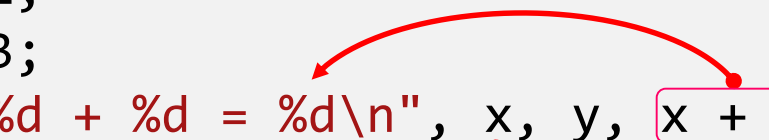
- **%f**，输出double类型，默认保留到小数点后6位，小数点后不足6位时右补零。
- **%6.2f**，输出double类型，保留到小数点后2位，输出的数最少占6个字符（若输出的数据少于6个字符宽度，则右对齐输出，左边空余的部分填空格；若输出的数据多于6个字符宽度，则按实际数据输出。）
- 更多的输出格式控制，可以查阅网络资料。



# 数值输入输出常见问题1: 多个数据

- 同时输出多个数据?

```
int x = 2;  
int y = 3;  
printf("%d + %d = %d\n", x, y, x + y);
```



输出

输出结果

**2 + 3 = 5**

- 同时输入多个数据?

```
int n;  
float r;  
scanf("%d%f", &n, &r);
```

输入

输入格式示例

**5 6.6**

**5  
6.6**

💡 编程提示: 输入多个数值型数据时, 用空格 (可以多个) 或回车换行 (或多个) 分隔, 但在占位符字符串中不用给出空格或回车换行符。

## 数值输入输出常见问题2：正确匹配

```
// c2-test.c  
float r;  
scanf("r=%f", &r );
```

```
double r;  
scanf("%f", &r);
```

输入格式示例

**r=3.14**

必须原样输入占位符字符串中的其他字符，即，必须原样输入 **r=** 进行占位匹配，然后输入实数，比如，输入 3.14 匹配实数占位符 %f，实数将被读入到变量 **r** 中（由占位符 %f 决定）。

💡 编程提示：熟记输入与输出常用数据类型的格式控制字符。

## 数值输入输出常见问题3：特殊匹配

### 例2-2-p: a+b特例

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

```
int main()  
{
```

```
    int a, b;  
    int sum;
```

```
    scanf("%d%d\n", &a, &b);  
    //scanf("%d%d", &a, &b); //常规写法  
    sum = a + b;  
    printf("%d\n", sum);
```

```
    return 0;
```

```
}
```

输入两个整数，  
按回车，无法  
读入数据！

scanf中的\n与多个  
空白符匹配，直到有  
新的非空白符输入出  
现，\n的匹配结束。

命令提示符 - c2-2p

D:\a1ac\example\chap2>c2-2p

1 2

💡 编程提示：一般情况下，scanf 中强烈不建议使用\n

在scanf格式字符串中\n和空格的作用是一样的，比如，  
scanf("%d %d\n%d", &a, &b, &c)

读入三个空格分开的三个整数是可以的。

scanf格式控制字符串中的占位符\n用来匹配（跳过）1到多个连续空白字符（回车，空格，tab等非可见字符），一旦发生至少1个空白字符匹配，且此时输入缓冲区中没有待输入的字符时，scanf会一直等待直到有新的字符输入且不是空白字符时结束\n的匹配，然后才继续往后解析。

用scanf输入整数时，格式控制字符串中的%d会跳过输入该数前面的若干个空白符，因此，输入多个数时，多个%d通常连着写，不加空白符，如：  
scanf("%d%d%d", &a, &b, &c); 这么写就对了！（实数原理相同）

## 数值输入输出常见问题3：特殊匹配

### 例2-2-r: scanf 中有 \n 时也有特别的用处！

输入：整数 $n$  ( $1 \leq n \leq 100$ )，换行；然后输入 $n$ 行数据，每行1个四则运算符 $op$ ，2个整数 $a$ 和 $b$ ，用空格分开。

输出：对每行数据，输出四则运算  
 $a \ op \ b$  的结果。

数据约定：输入、输出都在`int`范围内。

输入样例：

```
2
* 2 3
+ 3 5
```

输出样例：

```
6
8
```

# 数值输入输出常见问题3：特殊匹配

## 例2-2-r: scanf 中有 \n 时也有特别的用处！

输入：整数 $n$  ( $1 \leq n \leq 100$ )，换行；然后输入 $n$ 行数据，每行1个四则运算符 $op$ ，2个整数 $a$ 和 $b$ ，用空格分开。

输出：对每行数据，输出四则运算 $a \text{ op } b$ 的结果。

数据约定：输入、输出都在`int`范围内。

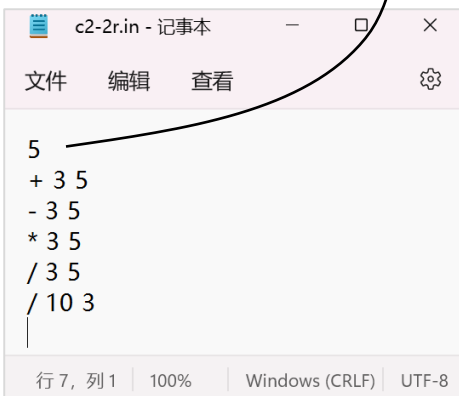
输入样例：

```
2
* 2 3
+ 3 5
```

输出样例：

```
6
8
```

某个实际输入

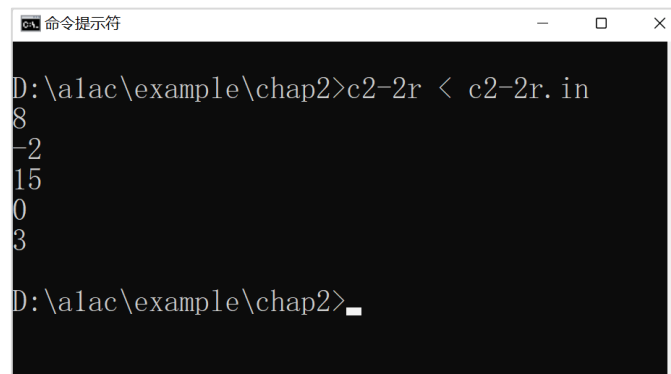


```
int n, i, a, b, c;
char op;

scanf("%d\n", &n);
for(i=1; i<=n; i++)
{
    if(i < n)
        scanf("%c%d%d\n", &op,&a,&b);
    else
        scanf("%c%d%d", &op,&a,&b);
    if(op == '+')
        c = a+b;
    if(op == '-')
        c = a-b;
    if(op == '*')
        c = a*b;
    if(op == '/')
        c = a/b;
    printf("%d\n", c);
}
```

%d解析输入中的整数5，将5读入变量`n`。  
\n匹配5之后的空白符（包括换行、空格等），直到遇到非空白符（比如，输入第2行的+，本例中+被下一个输入语句中的%c解析，将输入字符读入变量`op`）。  
如果没有\n，5之后的换行会被程序中下一个scanf中的%c解析，读入字符变量`op`中，导致正确的运算符无法被读入字符变量。

运行与输出



一般不建议在scanf中使用\n，使用的时候要特别分析具体应用场景。

## 数值输入输出小结

- 标准输入函数: scanf, 其函数原型为 (格式控制字符串中有非%引导的字符, 需原样输入)  
`int scanf(char *format, ...);` 【例 `scanf("%d %d", &a, &b);`】
- 标准输出函数: printf, 其函数原型为  
`int printf(char *format, ...);` 【例 `printf("%d\n%d\n", a, b);`】
- `scanf()`和`printf()` 都有返回值, 但`printf`返回值通常舍弃不用。
- 一个有用的输入返回值利用情况: `while ( scanf(...) == ?) { ... }`

```
int x, y, z, i;  
i = scanf("%d %d,%d", &x, &y, &z);  
  
printf("%d\n", i);  
printf("%d\n", x);  
printf("%d\n", y);  
printf("%d\n", z);
```

输入输出示例

输入

5 6,7 ✓

输出

3  
5  
6  
7

5 6 7 ✗

2  
5  
6  
4200704

## 2.4 算术运算：算术表达式与算术运算符

- 算术表达式(arithmetic expression)：由操作符和操作数构成的具有计算结果的表达式

- ◆ 操作数 (operand): 常量 或 变量
- ◆ 操作符 (operator): + - \* / %
- ◆ 例如:  $1 + 2$ ,  $s + 3$ ,  $a * b$

```
int a, b;  
float div;  
scanf("%d%d", &a, &b);  
div = a / b;
```

- 算术表达式的结果有数据类型，由其操作数的类型决定。如果操作数类型不同，则需要类型转换。例如：
  - ◆  $1.0 + 1$  (1自动转变为1.0, 结果为2.0, double型)
  - ◆  $2 / 3$  (2和3均为int型, 结果也为int型, 值为0)
  - ◆  $2.0f / 3$  (3自动转变为3.0f, 结果为0.666667f, float型)
  - ◆  $2 / 4 * 1.0 = ?$

## 2.4 算术运算

### 例2-4：求圆的面积和周长

```
// to calculate the circle's area and perimeter
#include <stdio.h>
#define PI 3.141592653589
int main()
{
    double radius, area, perimeter;

    scanf("%lf", &radius);
    ▲

    area = PI * radius * radius;
    perimeter = 2 * radius * PI;
    printf("Radius = %6.2f\n", radius);
    printf("Area = %6.2f\n", area);
    printf("Perimeter = %6.2f\n", perimeter);
    return 0;
}
```

输入

1

输出

Radius = 1.00

Area = 3.14

Perimeter = 6.28



## 2.4 算术运算

常见算术运算符

C操作	算术运算符	代数表达式	C表达式
加	+	$b + 8$	<code>b + 8</code>
减	-	$f - h$	<code>f - h</code>
乘	*	$ab$	<code>a * b</code>
除	/	$a/b$ 或 $a \div b$	<code>a / b</code>
求模	%	$r \bmod s$	<code>r % s</code>

### 例2-4：求圆的面积和周长

...

```
area = PI * radius * radius;
```

...

- 二元运算符的概念：运算符取两个操作数（常量、变量；表达式、函数；括号等），如： $a * (b+c)$ ;
- 直线形式的输入：C/C++编译器中不接受代数符号，如  $\frac{a+b+c}{3}$ ，应该表达为  $(a+b+c)/3$ ;
- 运算符优先级： $z = p * r \% q + w / x - y$ ;
- 括号的使用（“多余”的括号有时并不多余！），如： $d = (a * (b + c)) + (c * (d + e));$

# 算术运算实例：求圆的面积和周长（漂亮的输出格式）

```
#include <stdio.h>
#define PI 3.141592653589
int main()
{
    double radius, area, perimeter;
    scanf("%lf", &radius);
    area = PI * radius * radius;
    perimeter = 2 * radius * PI;

    printf("Radius = %6.2f\n", radius);
    printf("Area = %6.2f\n", area);
    printf("Perimeter = %6.2f\n\n", perimeter);


    printf("Radius      = %6.2f\n", radius);
    printf("Area          = %6.2f\n", area);
    printf("Perimeter    = %6.2f\n\n", perimeter);

    printf("      Radius = %6.2f\n", radius);
    printf("      Area   = %6.2f\n", area);
    printf("Perimeter = %6.2f\n\n", perimeter);

    printf("%12s = %6.2f\n", "Radius", radius);
    printf("%12s = %6.2f\n", "area", area);
    printf("%12s = %6.2f\n\n", "perimeter", perimeter);

    printf("%-12s = %6.2f\n", "Radius", radius);
    printf("%-12s = %6.2f\n", "area", area);
    printf("%-12s = %6.2f\n", "perimeter", perimeter);
    return 0;
}
```

输入 ▶



```
1
Radius = 1.00
Area = 3.14
Perimeter = 6.28

Radius = 1.00
Area = 3.14
Perimeter = 6.28

      Radius = 1.00
      area = 3.14
perimeter = 6.28

Radius      = 1.00
area        = 3.14
perimeter   = 6.28
```

常用的输出格式：

- 左对齐,
- 右对齐,
- 占指定宽度,
- 等等

输出

## 算术运算实例：一元二次方程求根

例2-5：一元二次方程  $ax^2 + bx + c = 0$  求根，输入实数系数a, b, c，求方程的根？

题目分析：根据求根公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

第2行：包括标准库头文件math.h，其中定义了若干（数学）标准库函数。

double sqrt(double)标准库函数，是在math.h中定义的求开方函数。

```
#include <stdio.h>
#include <math.h>
int main()
{
    double a, b, c, r1, r2;
    scanf("%lf%lf%lf", &a, &b, &c); // input test 1 -5 6

    r1 = (-b + sqrt(b * b - 4.0 * a * c)) / (2.0 * a);
    r2 = (-b - sqrt(b * b - 4.0 * a * c)) / (2.0 * a);

    printf("r1 = %.5f, r2 = %.5f\n", r1, r2);

    return 0;
}
```

**思考1：**  $b^2 - 4ac < 0$  时，该程序无法求解。试一试这样的输入，程序会出现怎样的情况。

**思考2：**  $a = 0$  时，求根公式求解也行不通。试一试这样的输入，程序会出现怎样的情况。

# 算术运算 -- 自增和自减运算符

- C语言提供自增 ++ (变量递增1) 运算, 用于取代赋值运算, 如  $a = a + 1$
- 同样也提供自减 -- (变量递减1) 运算
- 自增 ++ 和自减 -- 运算符可放在变量的前面也可放在变量后面

运算符	名称	示例表达式	说明
++	前置自增	++a	将a加1, 然后在a出现的表达式中使用新值
++	后置自增	a++	在a出现的表达式中使用当前值, 然后将a加1
--	前置自减	--a	将a减1, 然后在a出现的表达式中使用新值
--	后置自减	a--	在a出现的表达式中使用当前值, 然后将a减1

样例分析:

```
int i = 1, j = 2;
int m = 5, n = 6;
int u, v, x, y;
u = i++; // u is ?, i is ?
v = ++j; // v is ?, j is ?
x = m--; // x is ?, m is ?
y = --n; // y is ?, n is ?
```

注意: 在单独一条语句中, 前置自增 (减) 与后置自增 (减) 相同, 但在表达式中两者有差别。如

$a++;$   $\Leftrightarrow$   $++a;$   $\Leftrightarrow$   $a = a + 1;$

$c = a++;$        $c = ++a;$

$\Leftrightarrow$

$\Leftrightarrow$

$c = a;$   
 $a = a + 1;$

$a = a + 1;$   
 $c = a;$

自增和自减运算  
通常在循环语句  
中控制条件变量  
的改变。

```
while (i <= n)
{
    .....
    i++;
}
```

# 算术运算 -- 复合赋值运算符

- 对一个变量的值在其原有值基础上修改时，赋值运算符可以缩写成赋值表达式，如

$a += 5$  等价于  $a = a + 5$  ,      $x *= y + 5$  等价于  $x = x * (y + 5)$

- 其它二元运算符均可写成类似的形式，即

$(v) = (v) \text{ op } (\text{expression})$     可写为     $(v) \text{ op} = (\text{expression})$

赋值运算符	示例表达式	等价的含义
+=	$c += 7$	$c = c + 7$
-=	$d -= 4$	$d = d - 4$
*=	$e *= 5$	$e = e * 5$
/=	$f /= 3$	$f = f / 3$
%=	$g \% = 9$	$g = g \% 9$

- 使用复合符合赋值运算符可以使程序员更快地编写程序（精炼程序）；
- 可以使编译器更快地编译程序（提高编译效率）。

- 赋值运算符：  $+=$  ,  $-=$  ,  $*=$  ,  $/=$  ,  $\% =$  ,  $>> =$  ,  $<< =$  ,  $\& =$  ,  $\wedge =$  ,  $| =$
- 注意：  $y *= n + 1$ ; 等价于  $y = y * (n + 1)$ ; 而不是  $y = y * n + 1$ ;

# 算术运算小结

## 常见算术运算符

C操作	算术运算符	代数式	C表达式
加	+	$b + 8$	$b + 8$
减	-	$f - h$	$f - h$
乘	*	$ab$	$a * b$
除	/	$a/b$ 或 $a \div b$	$a / b$
求模	%	$r \bmod s$	$r \% s$

## 复合赋值运算符

赋值运算符	示例表达式	说明
+=	$c += 7$	$c = c + 7$
-=	$d -= 4$	$d = d - 4$
*=	$e *= 5$	$e = e * 5$
/=	$f /= 3$	$f = f / 3$
%=	$g \% = 9$	$g = g \% 9$

## 自增和自减运算符

运算符	名称	示例表达式	说明
++	前置自增	$++a$	将a加1，然后在a出现的表达式中使用新值
++	后置自增	$a++$	在a出现的表达式中使用当前值，然后将a加1
--	前置自减	$--a$	将a减1，然后在a出现的表达式中使用新值
--	后置自减	$a--$	在a出现的表达式中使用当前值，然后将a减1

## 2.5 类型转换

```
...  
char c = '1'; // '1' is 49, or 0x31  
int a = 1;  
int b = 8;  
double d = 1;
```

```
a = c;  
printf("%d\n", a);
```

表达式的数学  
值: 258.5

```
a = 60 + 4 * 50 - 3 / 2.0;  
printf("%d\n", a);
```

实际输出

```
d = (3 + 2) * (7 - 2) / 4;  
printf("%f\n", d);
```

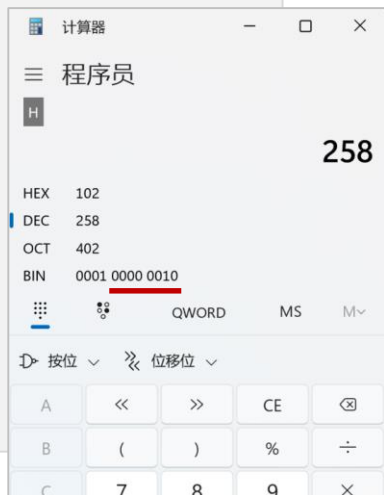
数学值:  
6.25

```
c = a;  
printf("%d\n", c);
```

c ← a ← 258

```
d = (double)(b / 5);  
printf("%f\n", d);
```

```
d = (double)(b) / 5;  
printf("%f\n", d);  
...
```



实际输出跟数学值  
不一样, 为什么?

49

258

6.000000

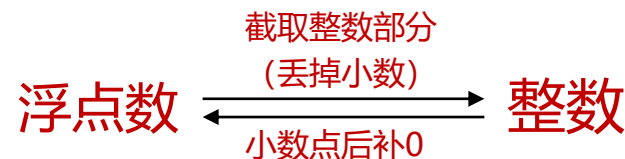
2

分析各个输出产生的原因。

运算中进行类型自动转换, 也称为隐式类型转换, 原则如下:

1. 字符与整数: 可以用整数的地方就可以用字符
2. 混合类型算术表达式中的每个值的类型会升级为此表达式中的“最高”类型 (第2个表达式右边部分)
3. 赋值号右边表达式的类型会自动转换为左边变量类型

(1) 浮点数 to (from) 整数 (表达式2和3)



(2) 整数 to 字符, 超出8位就将高位丢掉 (表达式4)

# 类型转换：强制类型转换

```
...  
char c = '1'; // '1' is 49, or 0x31  
int a = 1;  
int b = 8;  
double d = 1;
```

```
a = c;  
printf("%d\n", a);
```

```
a = 60 + 4 * 50 - 3 / 2.0;  
printf("%d\n", a);
```

```
d = (3 + 2) * (7 - 2) / 4;  
printf("%f\n", d);
```

```
c = a;  
printf("%d\n", c);
```

```
d = (double)(b / 5);  
printf("%f\n", d);
```

数学值:  $8/5=1.6$

```
d = (double)(b) / 5;  
printf("%f\n", d);  
...
```

强制类型转换，也称为显式类型转换。

实际输出

1.000000

1.600000

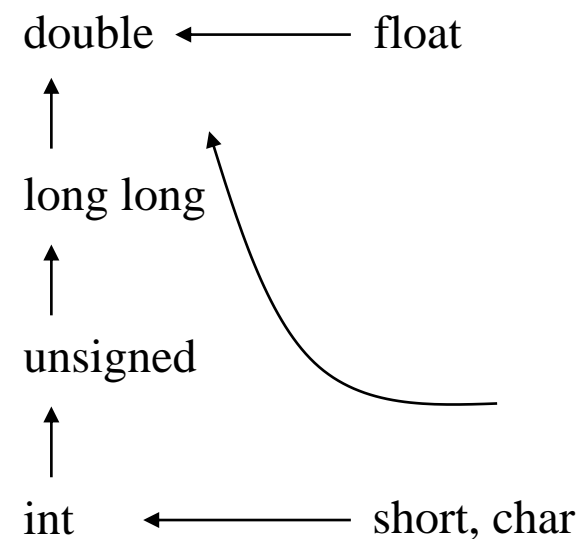
强制类型转换的一个典型应用实例：

```
// 进行浮点数 x 的整数和小数分离  
int i_part;  
double x = 2022.0913, d_part;  
i_part = x; // 获取x的整数部分  
d_part = x - (int)x; // 获取x的小数部分
```



# 基本类型数据的“高低”层次关系

	↑
Data types	
long double	
double	
float	
unsigned long long int	(synonymous with unsigned long long)
long long int	(synonymous with long long)
unsigned int	(synonymous with unsigned)
int	
unsigned short int	(synonymous with unsigned short)
short int	(synonymous with short)
unsigned char	
char	



类型自动转换的顺序示意（部分）

当发生由“较高精度类型数据”转换到“较低精度类型数据”时，大多数编译器会产生一个警告，因为发生了可能的精度损失。

# 类型转换应用：求成绩平均分

例2-6：输入多个成绩（有效成绩0~100，输入-1结束），求平均分（保留小数点后两位）。  
(例c1-8.c的改进版本)

隐式(自动)  
类型转换

```
#include <stdio.h>
int main()
{
    int score = 0, n = 0;
    double sum = 0;
    scanf("%d", &score);

    while (score != -1)
    {
        sum += score; // sum = sum + score
        n++;
        scanf("%d", &score);
    }
    if (n > 0)
    {
        printf("%.2f/%d = %.2f", sum, n, sum / n);
    }
    return 0;
}
```

输入：

31

42

51

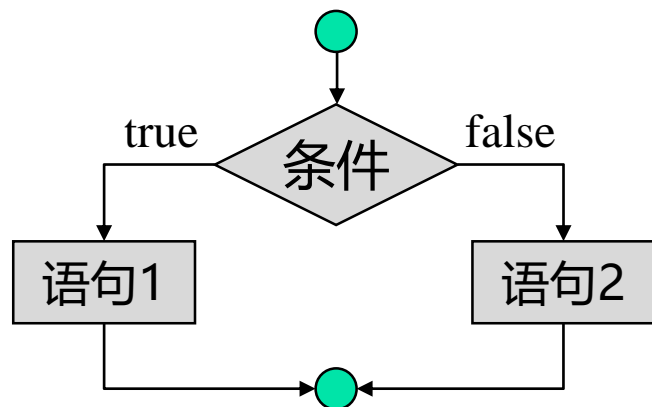
62

-1

输出：

186.00/4 = 46.50

## 2.6 关系运算符与逻辑运算符



if(expression)是一个选择语句

```
if(score >= 80 && score < 90)
    printf("成绩：良");
```

例：设 `int x = 1;`

`(5 >= 3) || (x == 5)`

值为?

`(2 == 3) && (x >= 7)`

值为?

- 程序中对不同分支的执行取决于给定的**条件**
- **逻辑运算**：C语言中的条件由**逻辑表达式**来描述。逻辑表达式通常是一个**关系表达式**（由**关系运算符**描述），也可以是由**逻辑运算符**连接起来的多个关系表达式，或者就是一个变量或常量【如 `if(x)`, `if(6)` 等】
- 逻辑表达式产生确定的逻辑值，结果为0或1【假或真，False或True，F或T】（C语言中用数“非0”表示逻辑值1）
- 关系运算符：> , < , >= , <= , == , !=
- 逻辑运算符：与 && , 或 || , 非 !

&&		A	
		T	F
B	T	T	F
	F	F	F

电路的串并联?

		A	
		T	F
B	T	T	T
	F	T	F

A	T	F
!A	F	T

开关切换?

# 关系运算符与逻辑运算符

- 关系运算符, 6个:  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $==$ ,  $!=$
- 逻辑运算符, 3个: 与  $\&\&$ , 或  $\|\|$ , 非  $!$

例: 设 `int x = 1;`

`(x == 5) || (5 >= 3)`      值为 1 (这个" 1" 占4字节)

`(2 == 3) &\& (x >= 7)`      值为 0 (这个" 0" 占4字节)

`X &\& Y`: 若X为0 (假), 则Y不必求值, 结果一定为0 (假)

`X || Y` : 若X为非0 (真), 则Y不必求值, 结果一定为非0 (真)

如上逻辑表达式的这一重要性质也称为

**"短路求值"**, 在程序设计中经常会用到, 如

**\*\*注意:**

1. 逻辑值是一个Bool (布尔) 类型, 但在C89标准中, 没有定义Bool类型, 通常用整数模拟Bool类型, 用非0值表示真 (True), 用0表示假 (False), 通常占4字节。在C中, 任何一个表达式都可作为条件, 如 `if(6+3)`, `while(x-5)`, 等。
2. C编译器在给出逻辑运算结果时, 以数值非0代表真, 以数值0代表假。

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

```
if ( (c >= 'a') &\& (c <= 'z') )  
{  
    // 若输入大写字母 A, 则第一个关系运算后, 逻辑运算就结束  
    .....  
}
```

# 逻辑运算实例1：大小写转换

例2-7：输入若干字符（可以多行），把输入中的小写字母转换为大写字母，其他字符保持不变 (s2S.c)。

输入文件

```
s2S.in - 记事本
文件 编辑 查看
2022, i came to beijing.
i love beijing!
i love buaa!
行 4, 列 1 | 140% | Windows (CRLF) | UTF-8
```

输出

```
命令提示符
C:\Users\DELL>d:
D:\>cd D:\alac\example\chap2
D:\alac\example\chap2>s2S < s2S.in
2022, I CAME TO BEIJING.
I LOVE BEIJING!
I LOVE BUAA!
D:\alac\example\chap2>
```

逻辑表达式的“短路求值”实例：

```
#include <stdio.h>
int main()
{
    int c; // not char
    while ((c = getchar()) != EOF)
    {
        if (((c >= 'a') && (c <= 'z')))
        {
            printf("%c", c - 32);
        }
        else
        {
            printf("%c", c);
        }
    }
    return 0;
}
```

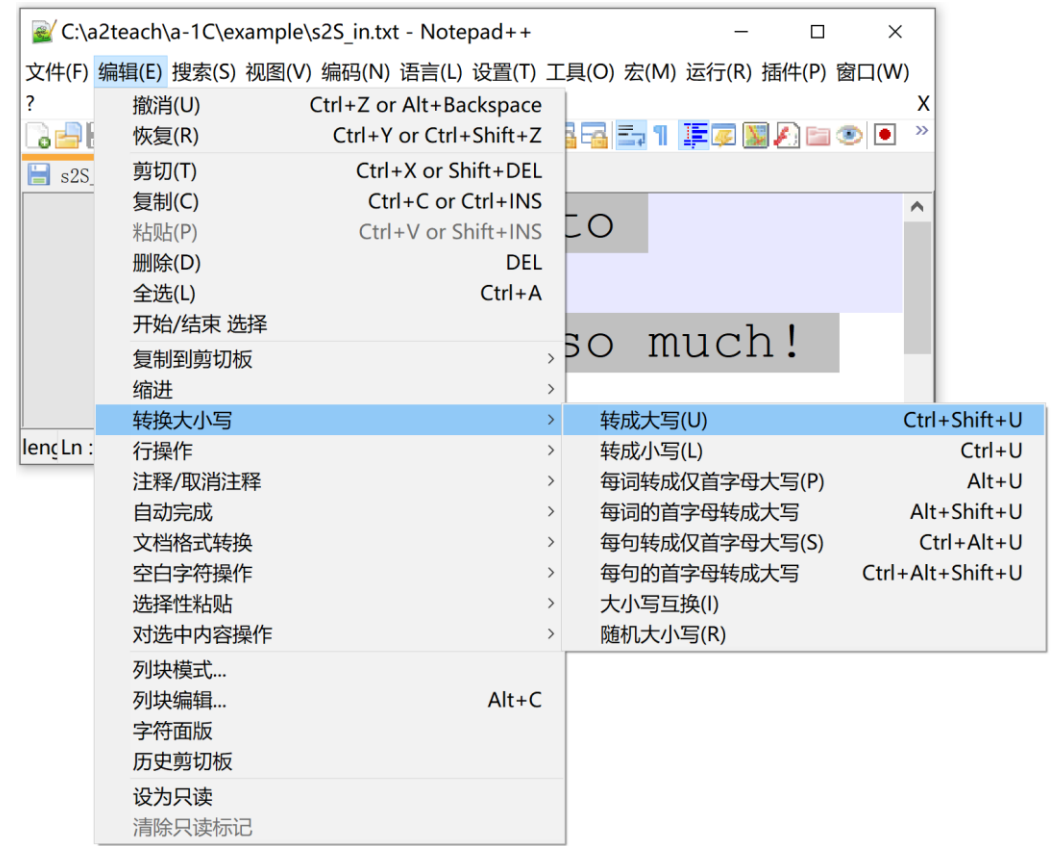
# 逻辑运算实例1：大小写转换

例：小写转大写

```
int c; // not char
while ((c = getchar()) != EOF)
{
    if ((c >= 'a') && (c <= 'z'))
    {
        printf("%c", c - 32);
    }
    else
    {
        printf("%c", c);
    }
}
```

实现此功能

**int getchar(void)**：输入字符，返回字符的ascii码。用户输入多个字符后输入回车，输入进入缓冲区，getchar从缓冲区中依序读入字符。读到文件末尾返回EOF。



## 逻辑运算实例2：猜名次

例2-8：名次预测。有人在赛前预测A~F六名选手在比赛中会按顺序分获第1名到第6名，如果他在这一预测中刚好猜对了三人的名次，则输出他是一个正常的人（不笨，也不是天才）。

### 题目分析

选手	预测名次	实际名次示例（某个输入）	猜对情况
A	1	1	1
B	2	2	1
C	3	5	
D	4	4	1
E	5	6	
F	6	3	

穷举（枚举），逐一枚举上述预测中只有3项正确的所有可能组合，则逻辑表达式应为：

```
if( ( (A == 1) && (B == 2) && (C == 3) && (D != 4) && (E != 5) && (F != 6) ) ||  
    ( (A == 1) && (B == 2) && (C != 3) && (D == 4) && (E != 5) && (F != 6) ) ||  
    (...) || (...) || ... )
```

这里由或连接起来“多个逻辑与表达式”有  $C_6^3 = 20$  项。每个逻辑与表达式中有6个关系表达式，有  $20 \times 6 = 120$  个关系表达式！多长的表达式啊！太复杂！

## 逻辑运算实例2：猜名次

例2-8：名次预测。有人在赛前预测A~F六名选手在比赛中会按顺序分获第1名到第6名，如果他在这一预测中刚好猜对了三人的名次，则输出他是一个正常的人（不笨，也不是天才）。

```
int a, b, c, d, e, f;

// input the actual ranking of sporters
scanf("%d%d%d", &a, &b, &c);
scanf("%d%d%d", &d, &e, &f);

if( (a == 1) + (b == 2) + (c == 3) +
    (d == 4) + (e == 5) + (f == 6) == 3 )
    printf("Got it! You are good.\n");
```

输入输出示例

```
1
2
5
4
6
3
Got it! You are good.
```

这里充分利用了逻辑表达式的真值等于 1 这个特点。

如果用穷举（枚举）方法，逐一枚举上述预测中只有 3 项正确的所有可能组合，则要写一个非常冗长的逻辑表达式：

```
( (a == 1) && (b == 2) && (c == 3) &&
  (d != 4) && (e != 5) && (f != 6) ) ||
( (a == 1) && (b == 2) && (c != 3) &&
  (d == 4) && (e != 5) && (f != 6) ) || (...) || (...) || ...
```



## 逻辑运算更多实例

- 判断整型变量n的值为一个0到10之间的值：

**$(0 \leq n \ \&\& \ n \leq 10)$**

注意：不能写为

$(0 \leq n \leq 10)$

该表达式是一个合法的逻辑表达式，测试一下其值为多少？为什么？

```
int n = 11;
```

```
printf("%d", 0 <= n <= 10);
```

- 判断字符变量c是字母：

$((c \geq 'a') \ \&\& \ (c \leq 'z')) \ || \ ((c \geq 'A') \ \&\& \ (c \leq 'Z'))$

- 判断某年y是平年还是闰年（闰年为能被4整除但不能被100整除，或能被400整除）：

$((y \% 4 == 0) \ \&\& \ (y \% 100 != 0)) \ || \ (y \% 400 == 0)$

## 逻辑运算更多实例

自然语言描述的条件转换为逻辑表达式通常不唯一，如：

- a不大于b:  $(a \leq b)$ ，也可以为  $!(a > b)$
- a不等于b:  $(a \neq b)$ ，也可以为  $!(a == b)$
- x不大于a且不大于b:

$x \leq a \ \&\& \ x \leq b$

也可以为  
还可以为

$!(x > a) \ \&\& \ !(x > b)$

$!((x > a) \ || \ (x > b))$

德.摩根定理  
(反演律)

可读性  
更强

例：判断某年y是平年还是闰年

常规写法:  $((y \% 4 == 0) \ \&\& \ (y \% 100 \neq 0)) \ || \ (y \% 400 == 0)$

更简写法:  $(!(y \% 4) \ \&\& \ (y \% 100)) \ || \ !(y \% 400)$

## 2.7 运算符的优先级

一些基本的优先级规则：

- 先乘除（模），后加减
- 关系运算优于逻辑运算
- 括号优先
- .....
- 提示：  
遇到不清楚的地方  
要善于加括号

```
x = a - b * c;
```

```
if( score >= 80 && score < 90 )  
{ ..... }
```

```
char c;  
if( ( ( c = getchar( ) ) != EOF) && (c >= 'a') && (c <= 'z') )  
    .....  
{...}
```

## \*2.7 运算符的优先级（有些还没有学过，不用理会）

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	后置++	后置自增运算符	变量名++	左到右	
	后置--	后置自减运算符	变量名--		
	[]	数组下标	数组名[整型表达式]		
	()	圆括号	(表达式) / 函数名(形参表)		
	.	成员选择（对象）	对象.成员名		
	->	成员选择（指针）	对象指针->成员名		
2	-	负号运算符	-表达式	右到左	单目运算符
	(类型)	强制类型转换	(数据类型)表达式		
	前置++	前置自增运算符	++变量名		单目运算符
	前置--	前置自减运算符	--变量名		单目运算符
	*	取值运算符	*指针表达式		单目运算符
	&	取地址运算符	&左值表达式		单目运算符
	!	逻辑非运算符	!表达式		单目运算符
	~	按位取反运算符	~表达式		单目运算符
	sizeof	长度运算符	sizeof 表达式/sizeof(类型)		
3	/	除	表达式/表达式	左到右	双目运算符
	*	乘	表达式*表达式		双目运算符
	%	余数（取模）	整型表达式%整型表达式		双目运算符

## \* 运算符的优先级（有些还没有学过，不用理会）

优先级	运算符	名称或含义	使用形式	结合方向	说明
4	+	加	表达式+表达式	左到右	双目运算符
	-	减	表达式-表达式		双目运算符
5	<<	左移	表达式<<表达式	左到右	双目运算符
	>>	右移	表达式>>表达式		双目运算符
6	>	大于	表达式>表达式	左到右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符
	<=	小于等于	表达式<=表达式		双目运算符
7	==	等于	表达式==表达式	左到右	双目运算符
	!=	不等于	表达式!= 表达式		双目运算符
8	&	按位与	整型表达式&整型表达式	左到右	双目运算符
9	^	按位异或	整型表达式^整型表达式	左到右	双目运算符
10		按位或	整型表达式 整型表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12		逻辑或	表达式  表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符

## \* 运算符的优先级 (有些还没有学过, 不用理会)

优先级	运算符	名称或含义	使用形式	结合方向	说明
14	=	赋值运算符	变量=表达式	右到左	
	/=	除后赋值	变量/=表达式		
	*=	乘后赋值	变量*=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
	=	按位或后赋值	变量 =表达式		
15	,	逗号运算符	表达式,表达式,...	左到右	从左向右顺序运算

# 运算符的优先级原则

- 基本的优先级原则：

- ◆ 指针（以后讲）最优。
- ◆ 单目运算优于双目运算，如正负号，如  $-5+6$ 。
- ◆ 先乘除（模），后加减。
- ◆ 逻辑运算最后计算。
- ◆ 不清楚的地方，善用小括号！

- 很多优先级是比较显然的。记住一些常用运算的优先级。记不住怎么办？加括号！

```
char c;  
if ( ( ( c = getchar() ) != EOF) && (c >= 'a') && (c <= 'z') )  
{  
    ...  
}
```

## 2.8 一维数组简介

类似排行榜这样的表格，涉及很多数据，数据如何存储与处理？

AC 编程 主页 小组 题目 赛事 宋友 登出									
E1-2022级程序设计基础第一次练习 比赛排名 更新中，上次更新于 2022-09-10 23:26:14									
简介 题目 排名 我的提交 提问&&公告									
服务器当前时间 2022-09-10 23:26:34 比赛结束时间 2022-09-14 23:30:00 比赛剩余时间 96:03:25									
« < 1 2 3 4 5 > »									
排名	用户	得分	罚时	A 450/479	B 438/444	C 249/278	D 282/353	E 221/254	F 123/148
1	用户姓名不显示	110	12:58:58	0:03:31(+1)	0:05:18	0:07:53	0:09:54	0:12:45	0:28:09(+1)
2		110	20:56:53	0:03:13	0:04:49	0:12:20	0:16:51(+1)	0:28:00(+1)	1:11:43
3		110	21:42:12	1:00:43	1:04:14	1:13:41	1:18:28(+1)	1:27:25	1:44:09
4		110	22:38:50	1:03:21	1:10:29	1:25:23	1:30:37(+1)	1:36:10	1:58:18(+5)
5		110	33:17:43	1:36:23	1:38:37	1:50:36(+1)	1:57:55(+1)	2:11:28(+1)	2:24:23(+1)
6		110	34:04:10	2:32:42(+1)	2:34:56	2:40:51	2:45:55	2:49:51	3:03:43(+4)
7		110	52:01:35	3:44:32	3:47:44(+1)	3:52:59	3:57:04	4:03:31	4:12:55
8		110	53:34:01	3:44:29	3:47:05	3:52:33	3:57:17(+2)	4:01:29	4:12:11
9		110	70:13:52	4:38:27	4:40:36	4:46:14	4:51:49(+1)	4:57:18	11:36:58(+5)
10		110	83:27:40	7:02:30	7:06:31	7:14:44	7:18:34	7:26:12	7:39:04
11		110	83:40:23	7:11:11(+1)	7:15:46	7:22:32	7:27:10(+1)	7:31:24	7:55:17(+4)
12		110	100:43:44	8:58:50	9:01:03(+1)	9:05:24	9:08:48(+1)	9:12:51	10:33:53(+3)
13		109.8	123:03:37	10:37:24(+1)	10:39:54	11:04:11	11:16:57(+2)	11:26:41(+2)	11:34:31
14		109.2	19:10:19	0:03:52	0:06:18	0:16:25(+1)	0:23:27(+2)	0:30:21	0:46:41(+1)
15		109.2	57:03:57	4:06:46	4:09:20	4:13:48	4:17:43(+1)	4:22:18	4:35:48(+1)
16		109.2	57:23:25	3:37:52(+1)	3:40:16(+1)	3:50:09	3:50:31	3:55:07	4:23:00(+1)
17		109.2	62:58:25	4:00:05	4:02:38	4:18:29(+3)	4:28:02	4:34:29	7:34:13
18		109.2	90:24:06	7:43:18	7:47:01	7:52:38	7:57:57(+1)	8:08:05	8:39:46(+1)
19		109.2	93:03:18	7:59:13(+1)	8:02:27	8:11:51	8:15:44	8:21:31	9:12:12(+6)
20		109.2	109:49:11	9:06:10(+1)	9:08:58	9:18:27	9:25:46(+1)	9:33:08	10:33:06(+2)



# 一维数组简介

示例：定义一个包含12个int整数类型的数组，数组命名为 a

- 数组名的命名规则与其他变量名相同
- 数组元素的标号从 0 开始（注意：不是 1）
- 第一个元素称为数组元素0 (zeroth element)，这样，数组a中的元素：第1个元素为a[0]；第2个元素为a[1]； ..... ；第 i 个元素为a[i-1]
- 要引用数组中的特定元素，就要指定数组中的特定位置(position)
- 方括号中位置整数称为下标(subscript)，下标应为整数或返回整数的表达式

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]

```
int i = 0;  
int a[12];
```

先定义。  
定义时方括号中的数表示数组大小，用常量。大小要合适。

```
while(i < 12)  
{  
    a[i] = (i+1)*(i+1);  
    i++;  
}
```

后使用。  
使用时方括号中的数表示访问某个数组元素，变量常量均可。访问时不要越界。

# 一维数组初始化与数组访问

- 数组定义后，每一个数组元素的赋值和访问与单个变量一样
- 数组赋值的一些方法

```
int a[6] = {1, 3, 5, -2, -4, 6}; // 定义数组a并初始化
```

```
int b[6] = {1, 3}; // 定义数组b，并部分初始化
```

```
int e[6]; // 定义数组e，未初始化
```

- 程序执行中进行赋值和访问
- 每一个数组元素的访问与单个变量一样
- 可以跟变量一样，定义不同类型的数组

```
int x[10];
```

```
double y[12];
```

```
char z[128];
```

.....

**注意：**  
不要将循环条件写为  $i \leq 6$ 。  
数组越界访问可能导致严重问题。初学者须谨记。

```
int i=0, a[6];  
  
while ( i < 6 )  
{  
    a[i] = (i+1)*(i+1);  
    i++;  
}  
  
for(i = 0; i < 6; i++ )  
{  
    printf("%d\n", a[i]);  
}
```

## 数组应用实例：成绩平均分

例2-8：求平均分。

输入：多人成绩。人数少于200人，成绩为0~100的整数，输入-1结束。

输出：(1) 平均分（保留小数点后两位）；(2) 成绩大于平均分的输入序号和成绩（序号从1开始编号），用空格、冒号、空格分开。

思考：将该程序与c1-8.c（未使用数组）进行对比分析，体会使用数组的意义。

输入样例：

76  
82  
61  
92  
50  
-1

输入输出  
样例

输出样例：

72.20  
1 : 76  
2 : 82  
4 : 92

```
int sum = 0, n = 0, i = 0, score[200] = {0};  
double ave;
```

定义数组，  
200足够。

```
scanf("%d", &score[0]);  
while (score[n] != -1)  
{
```

输入整数，  
存入数组

```
    sum += score[n];  
    n++;  
    scanf("%d", &score[n]);  
}
```

```
if(n > 0)  
{
```

为什么要用强制  
类型转换？

```
    ave = (double)sum / n;  
    printf("%.2f\n", ave);
```

```
    while (i < n)  
    {
```

```
        if((double)score[i] > ave)
```

```
        {  
            printf("%d : %d\n", i+1, score[i]);  
        }  
        i++;  
    }
```

```
}
```

## 数组应用实例：成绩平均分（更通用的版本）

例2-8-a：求平均分。

输入：多人成绩。人数少于200人，成绩为0~100的整数。  
(注意：没有输入-1作为输入结束的标记。)

输出：(1) 若有成绩输入，输出与例2-8的输出要求一样；  
(2) 若没有成绩输入，输出no input。

输入样例：

76  
82  
61  
92  
50

输入输出  
样例

输出样例：

72.20  
1 : 76  
2 : 82  
4 : 92

```
int sum = 0, n = 0, i, score[200] = {0};
double ave;
while (scanf("%d", &score[n]) != EOF) // EOF: End of File
{
    n++;
}
for (i = 0; i < n; i++)
{
    sum += score[i];
}
if (n > 0)
{
    ave = (double)sum / n;
    printf("%d/%d = %.2f\n", sum, n, ave);
    for (i = 0; i < n; i++)
    {
        if ((double)score[i] > ave)
        {
            printf("%d : %d\n", i + 1, score[i]);
        }
    }
}
else
{
    printf("no input");
}
```

scanf输入正确，返回1（输入元素个数）；输入到“文件”末尾（键盘可以理解为文件），返回EOF（windows，命令行模式下用Ctrl+Z表示输入结束）。

思考题：对比该例与“例2-8”的区别和实现方式的不同。

---

# 本讲提纲

2.8 一维数组简介

2.9 常量的符号表示方法简介

## 2.9 常量的符号表示方法简介

- **#define** 编译预处理中宏定义
- 在C语言源程序中允许用一个标识符来表示（定义）一个字符串，称为“宏”。在编译预处理时，对程序中所有出现的“宏”，都用宏定义中的字符串去替换，这称为“宏替换”或“宏展开”。宏替换是由预处理程序在编译前完成的。(EOF 就是 stdio.h 中定义的常量-1)
- 宏替换是纯粹的文字替换
- 使用宏定义的好处：(1) 可提高程序的可读性；(2) 程序的可移植性更好；(3) 可维护性更好

```
#include <stdio.h>
#define PI 3.141592653589
int main()
{
    double radius, area, perimeter;

    scanf("%lf", &radius);

    area = PI * radius * radius;
    perimeter = 2 * radius * PI;
    .....
```

宏替换

```
#include <stdio.h>
#define PI 3.141592653589
int main()
{
    double radius, area, perimeter;

    scanf("%lf", &radius);

    area = 3.141592653589 * radius * radius;
    perimeter = 2 * radius * 3.141592653589;
    .....
```

# #define 的更多知识

- 宏定义常量通常用于数组大小的定义，如 `#define N 10005`

- 宏也可以带参数，替换时，相应的参数也进行替换，如

```
#define _DIV(a,b) a/b
```

(1) 程序中出现 `_DIV(1,2)` 时就替换为 `1/2`

(2) 程序中出现 `_DIV(3,4)` 时就替换为 `3/4`

(3) 程序中出现 `_DIV(1+3,3+5)` 时就替换为 `1+3/3+5`

【此处，题意是希望计算 $(1+3)/(3+5)$ ，但替换后

没有实现此功能，如何修改？】

- 宏定义中的常见错误

- ◆ 宏定义的最后加上多余的分号，如 `#define N 10;`

- ◆ 带参数的宏中漏写了必要的括号，如

```
#define _DIV(a,b) a/b 应写成
```

```
#define _DIV(a,b) (a)/(b)
```

则(3)中的错误就可以避免。请读者自行体会。

## 几个典型的宏定义示例

```
#include <stdio.h>
#define N 200
#define _F(i, Lf, Rt) for (i = Lf; i < Rt; i++)
#define _CUBE(a) (a) * (a) * (a)
#define LL long long

int main()
{
    LL sum = 0, score[N] = {0}, cubesum = 0;
    int n = 0, i;

    while (scanf("%d", &score[n]) != EOF)
    {
        n++;
    }
    _F(i, 0, n) // for (i = 0; i < n; i++)
    {
        sum += score[i];
        cubesum += _CUBE(score[i]);
    }
    printf("%lld\n", sum);
    printf("%lld\n", cubesum);
    return 0;
}
```

# #define 的更多知识

```
#include <stdio.h>
#define N 200
#define _F(i, Lf, Rt) for (i = Lf; i < Rt; i++)
#define _CUBE(a) (a) * (a) * (a)
#define LL long long

int main()
{
    LL sum = 0, score[N] = {0}, cubesum = 0;
    int n = 0, i;

    while (scanf("%d", &score[n]) != EOF)
    {
        n++;
    }
    _F(i, 0, n) // for (i = 0; i < n; i++)
    {
        sum += score[i];
        cubesum += _CUBE(score[i]);
    }
    printf("%lld\n", sum);
    printf("%lld\n", cubesum);
    return 0;
}
```

宏替换?

```
#include <stdio.h>
#define N 200
#define _F(i, Lf, Rt) for (i = Lf; i < Rt; i++)
#define _CUBE(a) (a) * (a) * (a)
#define LL long long

int main()
{
    long long sum = 0, score[200] = {0}, cubesum = 0;
    int n = 0, i;

    while (scanf("%d", &score[n]) != -1)
    {
        n++;
    }
    {
        sum += score[i];
        cubesum += 
    }
    printf("%lld\n", sum);
    printf("%lld\n", cubesum);
    return 0;
}
```

课后练习



# 综合练习1：大小写转换

例2-9：输入若干字符（可以多行），把输入中的小写字母转换为大写字母，其他字符保持不变 (s2S.c)。

输入文件

```
s2S.in - 记事本
文件 编辑 查看
2022, i came to beijing.
i love beijing!
i love buaa!
行 4, 列 1 140% Windows (CRLF) UTF-8
```

输出

```
命令提示符
C:\Users\DELL>d:
D:\>cd D:\alac\example\chap2
D:\alac\example\chap2>s2S < s2S.in
2022, I CAME TO BEIJING.
I LOVE BEIJING!
I LOVE BUAA!
D:\alac\example\chap2>
```

逻辑表达式的“**短路求值**”实例：

```
#include <stdio.h>
int main()
{
    int c; // not char
    while ((c = getchar()) != EOF)
    {
        if (((c >= 'a') && (c <= 'z')))
        {
            printf("%c", c - 32);
        }
        else
        {
            printf("%c", c);
        }
    }
    return 0;
}
```

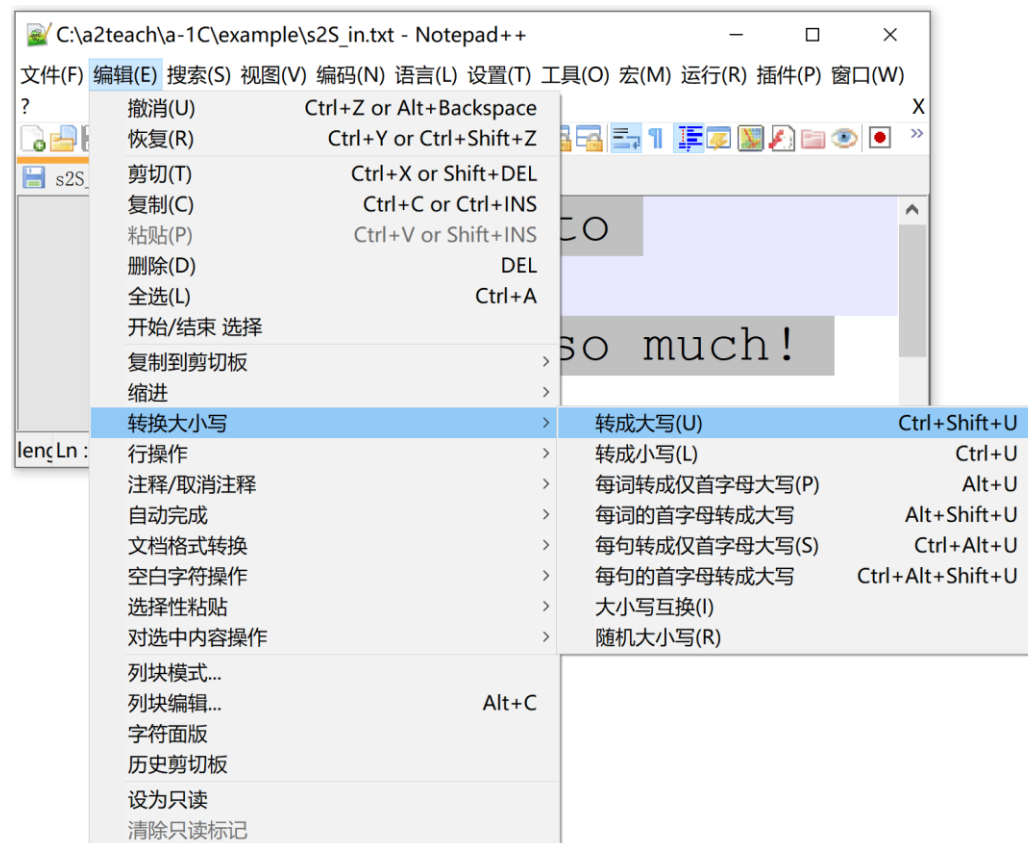
# 综合练习1：大小写转换

例：小写转大写

```
int c; // not char
while ((c = getchar()) != EOF)
{
    if ((c >= 'a') && (c <= 'z'))
    {
        printf("%c", c - 32);
    }
    else
    {
        printf("%c", c);
    }
}
```

实现此功能

**int getchar(void)**：输入字符，返回字符的ascii码。用户输入多个字符后输入回车，输入进入缓冲区，getchar从缓冲区中依序读入字符。读到文件末尾返回EOF。





## 综合练习2：大小写转换

str[0]	str[1]	str[2]	str[3]	str[4]	...	str[i-1]	str[i]		
i	n		s	e	...	*	*		
						↑	↑		

例2-10：输入一段文字（不超过1000个字符），将每个单词首字母转换为大写字母。

用数组实现比较易懂，问题求解思路（伪代码）：

1. 把输入的字符依序存入字符数组str；
2. 如果str的首字母str[0]为小写字母，str[0]转换；
3. 从str的首字符之后的字符开始顺序遍历，如果str[i]为小写字母，且str[i-1]不为字母，则str[i]转换；
4. 如果文字没有遍历完成，到第3步；
5. 输出str。

```
while ((c = getchar()) != EOF)
    str[n++] = c;
if ((str[0] >= 'a') && (str[0] <= 'z')) // 处理首字符
    str[0] = str[0] - 32;

for (i = 1; i < n; i++)                // 处理后续字符
{
    if ( ( (str[i] >= 'a') && (str[i] <= 'z') ) &&
          !( (str[i - 1] >= 'a') && (str[i - 1] <= 'z')) ||
            ((str[i - 1] >= 'A') && (str[i - 1] <= 'Z')) ) )
        str[i] = str[i] - 32;
}
for (i = 0; i < n; i++)
    putchar(str[i]); // printf("%c", str[i]);
```

## \*综合练习2: 大小写转换

str[0]	str[1]	str[2]	str[3]	str[4]	...	str[i-1]	str[i]		
i	n		s	e	...	*	*		
						↑	↑		

例2-10: 输入一段文字 (不超过1000个字符), 将每个单词首字母转换为大写字母。

用数组实现比较易懂, 问题求解思路 (伪代码):

1. 把输入的字符依序存入字符数组str;
2. 如果str的首字母str[0]为小写字母, str[0]转换;
3. 从str的首字符之后的字符开始顺序遍历, 如果str[i]为小写字母, 且str[i-1]不为字母, 则str[i]转换;
4. 如果文字没有遍历完成, 到第3步;
5. 输出str。

```
while ((c = getchar()) != EOF)
    str[n++] = c;
if ((str[0] >= 'a') && (str[0] <= 'z')) // 处理首字符
    str[0] = str[0] - 32;

for (i = 1; i < n; i++)                // 处理后续字符
{
    if ( islower(str[i]) && !(isalpha(str[i-1])) )
        str[i] = str[i] - 32;
}
for (i = 0; i < n; i++)
    putchar(str[i]); // printf("%c", str[i]);
```

## 综合练习2：大小写转换

str[0]	str[1]	str[2]	str[3]	str[4]	...	str[i-1]	str[i]		
i	n		s	e	...	*	*		
						↑	↑		

例2-10：输入一段文字（不超过1000个字符），将每个单词首字母转换为大写字母。

用数组实现比较易懂，问题求解思路（伪代码）：

1. 把输入的字符依序存入字符数组str；
2. 如果str的首字母str[0]为小写字母，str[0]转换；
3. 从str的首字符之后的字符开始顺序遍历，如果str[i]为小写字母，且str[i-1]不为字母，则str[i]转换；
4. 如果文字没有遍历完成，到第3步；
5. 输出str。

```
#include <stdio.h>
#define N 1005
char str[N]; // 前几讲，学习字符串之前，字符数组先放在main之外

int main()
{
    int c, i = 0, n = 0;
    int flag_str_i = 0; // str中第i个字符是否为小写字母的标记
    int flag_str_iLast = 0; // 第i-1个字符是否为字母的标记
    while ((c = getchar()) != EOF)
        str[n++] = c;

    if ((str[0] >= 'a') && (str[0] <= 'z')) // 处理首字符
        str[0] = str[0] - 32;

    for (i = 1; i < n; i++) // 处理后续字符
    {
        flag_str_i = _____; // 第i个字符是小写字母
        flag_str_iLast = _____; // 第i-1个字符是字母

        if ( _____ )
            _____;
        flag_str_i = 0; // 标记重置
        flag_str_iLast = 0; // 标记重置
    }
    for (i = 0; i < n; i++)
        putchar(str[i]); // printf("%c", str[i]);

    return 0;
}
```

## 综合练习2：大小写转换（非数组版）

例2-10-b：输入一段文字（不超过1000个字符），将每个单词首字母转换为大写字母。

参考“例2-9”和“例2-10”，问题求解思路（伪代码）：

1. x1存放第一个输入，x2存放第二个输入；
2. 首字母为小写字母，则转换后输出，否则直接输出；
3. x1存放第 i-1 次输入，x2存放第 i 次输入，处理逻辑同“例2-10”第3步，处理完输出；
4. 保存当前输入状态，处理下一个输入，如果文字没有遍历完成，到第3步；
5. 结束。

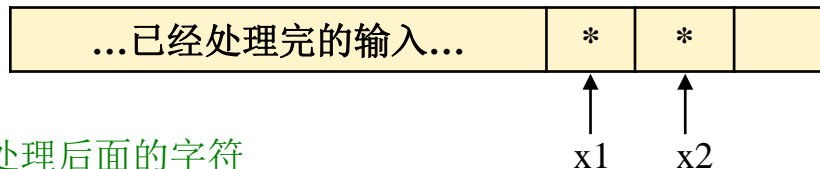
```
int c;  
int flag_str_i = 0; // str中第i个字符是否为小写字母的标记  
int flag_str_iLast = 0; // 第i-1个字符是否为字母的标记  
char x1, x2;
```

```
c = getchar();  
x1 = c;  
if((x1 >= 'a') && (x1 <= 'z')) // 处理首字符  
    x1 = x1 - 32;  
putchar(x1);
```

```
while((c = getchar()) != EOF) // 处理后面的字符  
{
```

```
    x2 = c;  
    flag_str_i = _____; // 第i个字符是小写字母  
    flag_str_iLast = _____; // 第i-1个字符是字母  
    _____; // 记录当前输入，作为下一次循环时的输入的前一个字符  
    if(_____)  
        _____;  
    putchar(x2);  
    flag_str_i = 0; // 标记重置  
    flag_str_iLast = 0; // 标记重置
```

```
}
```



## 第二章小结

- 了解C语言的基本数据类型，取值范围
- 理解ASCII编码规则和使用方法
- 熟练掌握变量的应用：变量的定义（命名规则并区分关键字）、赋值、使用
- 熟练掌握scanf、printf函数的功能与使用方法
- 熟练掌握 +、-、\*、/、%、++、-- 等的使用方法和计算表达式的值
- 熟练掌握赋值运算、复合赋值运算
- 熟练掌握关系运算、逻辑运算，以及两者的关系
- 熟练掌握类型转换的规则及使用方法
- 简单掌握一维数组的使用方法
- 了解常量的符号表示形式（#define的用法与含义）
- 简单掌握库函数的使用方法，如<math.h>中的 sqrt

- **数据输入**
- **数据处理**
- **数据输出**

**数据表示**