

## 第6章 回顾 数据类型和表达式

### 6.1 教学要点

本章作为前五章的回顾，详细总结了 C 语言中可以使用的数据类型，要求学生准确掌握各种数据类型的定义并能做到熟练使用。在此基础上，C 语言提供了多种运算符，和数据组合后形成表达式，要求学生掌握各种表达式的语法和求解规则，丰富的运算符和表达式使得 C 语言功能十分完善，鼓励学生灵活应用增加编程技巧。

6.1 节通过从数据在计算机内的存储格式入手，介绍整型、字符型、单精度和双精度实型四种基本数据类型的存储方式。教师在讲解过程中，可以适当补充计算机概论的知识，帮助学生加深理解。接下来，通过实例介绍了四种基本数据类型的定义、取值范围及其常量形式，要求学生重点掌握 ASCII 字符集和转义字符的应用。

6.2 节通过实例介绍了在 C 语言中如何通过函数调用完成数据的输入和输出，要求学生能熟练使用 `scanf()`、`printf()`、`getchar()` 和 `putchar()` 这四个基本输入输出函数，重点为格式控制说明符。对于字符类型的数据，还要求学生能灵活运用这两组输入输出函数。

6.3 节通过实例介绍了 C 语言中不同类型数据混合运算的预处理—自动类型转换和强制类型转换，使学生了解类型转换的原则和意义，避免在以后的实际开发中出现类似的错误。

6.4 节通过大量实例介绍了 C 语言的多种表达式和运算符，重点掌握各种运算符的优先级和结合性。运算符丰富、表达式灵活是 C 语言不同于其他高级编程语言的特点，希望学生能充分运用，熟能生巧，使编程能力得到进一步的提高。

本教材是以程序设计为主线，以编程应用为驱动，通过案例和问题引入内容，重点讲解程序设计的思想和方法，穿插介绍相关的语言知识，因此本章将散布在前五章中的数据类型和表达式等内容做了归纳性的汇总。由于知识点较多，需要学生在理解的基础上加强记忆，最好能结合实践加深印象，提高学生的综合编程能力，培养良好的编程风格。

讲授学时：3 学时，实验学时：2 学时。

本章的知识能力结构图见图 6.1。

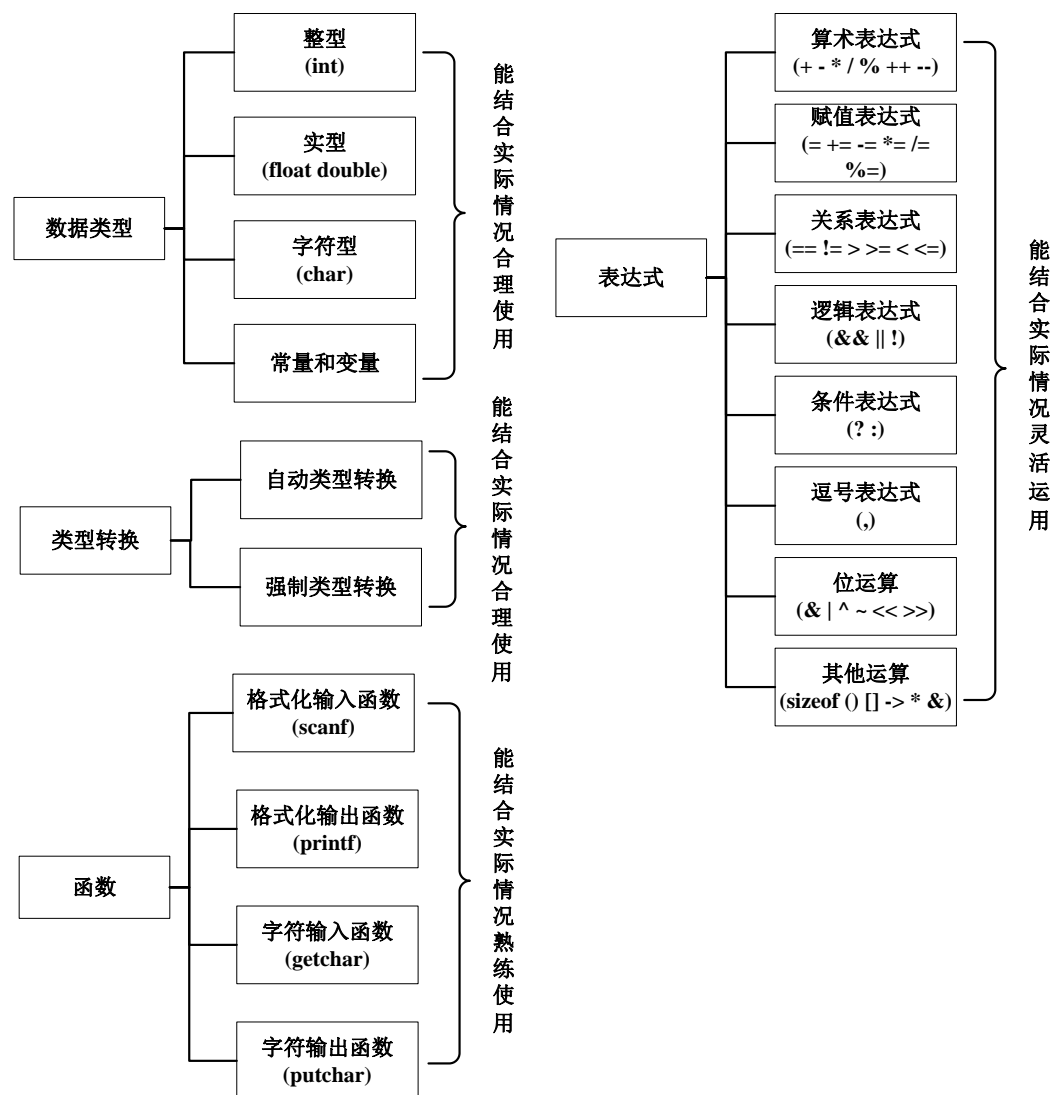






图 6.1 知识能力结构图

## 6.2 讲稿

1	<div> </div> <h3>Chap 6 回顾 数据类型和表达式</h3> <p>6.1 数据的存储和基本数据类型</p> <p>6.2 数据的输入和输出</p> <p>6.3 类型转换</p> <p>6.4 表达式</p>	本章分 4 节。
---	---	----------

2	 <h2>本章要点</h2> <ul style="list-style-type: none"> <li>■ C语言的基本数据类型有哪些？</li> <li>■ 各种基本数据类型的常量有哪些表现形式？</li> <li>■ C语言有哪些表达式？各种表达式的求解规则是什么？</li> </ul>	提出本章的学习要点。
3	 <h2>数据类型和表达式</h2> <ul style="list-style-type: none"> <li>■ 数据类型 <ul style="list-style-type: none"> <li>□ 基本数据类型 <ul style="list-style-type: none"> <li>■ 整型int</li> <li>■ 实型（浮点型）float double</li> <li>■ 字符型char</li> </ul> </li> <li>□ 构造数据类型 <ul style="list-style-type: none"> <li>数组、结构、联合、枚举</li> </ul> </li> <li>□ 指针类型</li> <li>□ 空类型</li> </ul> </li> <li>■ 运算：对数据的操作 <ul style="list-style-type: none"> <li>运算符+数据 → 表达式</li> </ul> </li> </ul>	介绍 C 语言提供了四种基本的数据类型，其中用这些基本类型还可以构造出许多导出类型。C 语言还提供了许多运算符，可以对不同类型的数据进行处理。这些运算符与数据组合后便形成了表达式。
4	 <h2>6.1 数据的存储和基本数据类型</h2> <h3>6.1.1 数据的存储</h3> <p>整型、实型、字符型数据的存储</p> <h3>6.1.2 基本数据类型</h3> <p>整型与整型常量（整数） 字符型与字符型常量 实型与实型常量（实数）</p>	不同类型的数据在计算机内存中的存放方式也是各异的，了解数据的存储方式可以帮助学生理解数据类型的概念和使用方法。
5	 <h3>6.1.1 数据的存储—整型数据</h3> <p>设整数在内存中用2个字节存储</p> <pre> 1 000 0001 1000 0001 0 000 0001 1000 0001 </pre> <p>↑ 符号位 1：负数 0：正数</p>	<p>整型数据的存储要点：最高位为符号位，其余部分为该整型数据的绝对值。</p> <p>提问：ppt 上所示的两个整数的值分别是什么？</p> <p>解答：-385 和+385。</p>

6	<div> <div></div> <div>数值的表示方法—原码 反码 补码</div> <ul style="list-style-type: none"> <li>正数的原码、反码和补码相同           <ul style="list-style-type: none"> <li>1 的补码 0 000 0000 0000 0001</li> <li>.....</li> <li>32767 的补码 0 111 1111 1111 1111</li> <li>(<math>2^{15}-1</math>, 2个字节的存储单元能表示的最大正数)</li> </ul> </li> <li>负数的原码、反码和补码不同           <ul style="list-style-type: none"> <li>-1               <ul style="list-style-type: none"> <li>原码 1 000 0000 0000 0001</li> <li>反码 1 111 1111 1111 1110 原码取反 (符号位保持不变)</li> <li>补码 1 111 1111 1111 1111 反码+1</li> </ul> </li> </ul> </li> </ul> </div>	<p>介绍数值的三种码制表示方法以及正数和负数的表示方法要点。即：正数的原码、反码和补码都相同；负数的反码即对原码取反，补码即反码再加 1，要点是符号位保持不变。</p>
7	<div> <div></div> <div>原码 反码 补码</div> <div>32767</div> <ul style="list-style-type: none"> <li>补码 0 111 1111 1111 1111</li> </ul> <div>-32767</div> <ul style="list-style-type: none"> <li>原码 1 111 1111 1111 1111</li> <li>反码 1 000 0000 0000 0000 原码取反 (符号位保持不变)</li> <li>补码 1 000 0000 0000 0001 反码+1</li> </ul> <div>-32768 = -32767-1</div> <ul style="list-style-type: none"> <li>补码 1 000 0000 0000 0000</li> <li>(2个字节的存储单元能表示的最小负数)</li> </ul> </div>	<p>举例介绍正数和负数的原码、反码以及补码的表示方法。由此可以看出，2 个字节的存储单元所能表示的最小负数为 -32768，那么提出问题：2 个字节的存储单元所能表示的最大正数为多少？答案即为 32767。</p>
8	<div> <div> <div>-32768</div> <div>-1 0 1</div> <div>32767</div> </div> <div> <div>32767 0111 1111 1111 1111</div> <div>.....</div> <div>1 0000 0000 0000 0001</div> <div>0 0000 0000 0000 0000</div> <div>-1 1111 1111 1111 1111</div> <div>-2 1111 1111 1111 1110</div> <div>.....</div> <div>-32767 1000 0000 0000 0001</div> <div>-32768 1000 0000 0000 0000</div> </div> <div> <div>32767 + 1 = 32768 ?</div> <div>1000 0000 0000 000 = -32768</div> <div>-32768 - 1 = -32769 ?</div> <div>0111 1111 1111 1111 = 32767</div> </div> </div>	<p>以 2 个字节为例给出整型数据的表示范围：-32768~+32767，以及二进制码的变化规律，那么下面 2 个问题的答案即可得出：(可提出溢出概念)</p> <p>(1) 32767+1=32768?</p> <p>(2) -32768-1=-32769?</p>
9	<div> <div></div> <div>实型和字符型数据的存储</div> <ul style="list-style-type: none"> <li>实型数据的存储           <ul style="list-style-type: none"> <li>-1.2345e+02</li> <li>符号位 阶码 尾数</li> </ul> </li> <li>字符型数据的存储           <ul style="list-style-type: none"> <li>一个字节存储ASCII码。</li> </ul> </li> </ul> <p>例如字符型常量 'A' 的 ASCII 码为 65，它在内存中以下列形式存放：</p> <p>0 1 0 0 0 0 1</p> </div>	<p>简单介绍实型数据和字符型数据的存储方式。</p> <p>实型数据存储示例：实数-1.2345e+02 是负数，阶码是 2，尾数是 1.2345。</p> <p>字符型数据存储示例：字符 'A' 的 ASCII 码为 65，它在内存中以下列形式存放：</p> <div>0 1 0 0 0 0 1</div>

10	<h3>6.1.2 基本数据类型</h3> <ul style="list-style-type: none"> <li>■ 整型 <ul style="list-style-type: none"> <li>有符号整型    无符号整型    数据长度</li> <li>int    unsigned [int]    16或32位</li> <li>short [int]    unsigned short [int]    16位</li> <li>long [int]    unsigned long [int]    32位</li> </ul> </li> <li>■ 字符型 <ul style="list-style-type: none"> <li>char    8位</li> </ul> </li> <li>■ 实型（浮点型） <ul style="list-style-type: none"> <li>单精度浮点型    float    32位</li> <li>双精度浮点型    double    64位</li> </ul> </li> </ul>	<p>本小节介绍三种基本数据类型的数据长度，尤其要注意整型数据的长度取决于机器的字长。</p>																		
11	<h3>基本数据类型—整型</h3> <p>扩展的整数类型: short long unsigned [int]</p> <table> <tr> <td>有符号整型</td> <td>无符号整型</td> <td>数据长度</td> </tr> <tr> <td>int</td> <td>unsigned [int]</td> <td>16或32位</td> </tr> <tr> <td>short [int]</td> <td>unsigned short [int]</td> <td>16位</td> </tr> <tr> <td>long [int]</td> <td>unsigned long [int]</td> <td>32位</td> </tr> </table> <p>有符号 short</p> <p>1 000 0000 0000 0000 -32768 -2<sup>15</sup></p> <p>0 111 1111 1111 1111 32767 2<sup>15-1</sup></p> <p>无符号 unsigned short</p> <p>0000 0000 0000 0000 0</p> <p>1111 1111 1111 1111 65535 2<sup>16-1</sup></p>	有符号整型	无符号整型	数据长度	int	unsigned [int]	16或32位	short [int]	unsigned short [int]	16位	long [int]	unsigned long [int]	32位	<p>提出无符号整型数据的概念，顾名思义，不带符号的整数，即零或正整数，不包括负数。存储有符号的整型数据时，存储单元的最高位是符号位，其余各位表示数值；存储无符号（指定 unsigned）的整型数据时，存储单元全部用于表示数值。</p> <p>此处可引出问题：</p> <p>(1) 哪些数据需要用有符号整型数据表示？试举例说明。</p> <p>(2) 哪些数据又可以用无符号整型数据表示即可？试举例说明。</p>						
有符号整型	无符号整型	数据长度																		
int	unsigned [int]	16或32位																		
short [int]	unsigned short [int]	16位																		
long [int]	unsigned long [int]	32位																		
12	<h3>整数类型的取值范围</h3> <table> <tr> <td>int    32位</td> <td>-2147483648 ~ 2147483647</td> <td>-2<sup>31</sup> ~ 2<sup>31-1</sup></td> </tr> <tr> <td>short [int]    16位</td> <td>-32768 ~ 32767</td> <td>-2<sup>15</sup> ~ 2<sup>15-1</sup></td> </tr> <tr> <td>long [int]    32位</td> <td>-2147483648 ~ 2147483647</td> <td>-2<sup>31</sup> ~ 2<sup>31-1</sup></td> </tr> </table> <table> <tr> <td>unsigned [int]    32位</td> <td>0 ~ 4294967295</td> <td>0 ~ 2<sup>32-1</sup></td> </tr> <tr> <td>unsigned short [int]    16位</td> <td>0 ~ 65535</td> <td>0 ~ 2<sup>16-1</sup></td> </tr> <tr> <td>unsigned long [int]    32位</td> <td>0 ~ 4294967295</td> <td>0 ~ 2<sup>32-1</sup></td> </tr> </table>	int    32位	-2147483648 ~ 2147483647	-2 <sup>31</sup> ~ 2 <sup>31-1</sup>	short [int]    16位	-32768 ~ 32767	-2 <sup>15</sup> ~ 2 <sup>15-1</sup>	long [int]    32位	-2147483648 ~ 2147483647	-2 <sup>31</sup> ~ 2 <sup>31-1</sup>	unsigned [int]    32位	0 ~ 4294967295	0 ~ 2 <sup>32-1</sup>	unsigned short [int]    16位	0 ~ 65535	0 ~ 2 <sup>16-1</sup>	unsigned long [int]    32位	0 ~ 4294967295	0 ~ 2 <sup>32-1</sup>	<p>一个无符号整型变量中可以存放的数的范围比一般整型变量中数的范围扩大一倍。</p>
int    32位	-2147483648 ~ 2147483647	-2 <sup>31</sup> ~ 2 <sup>31-1</sup>																		
short [int]    16位	-32768 ~ 32767	-2 <sup>15</sup> ~ 2 <sup>15-1</sup>																		
long [int]    32位	-2147483648 ~ 2147483647	-2 <sup>31</sup> ~ 2 <sup>31-1</sup>																		
unsigned [int]    32位	0 ~ 4294967295	0 ~ 2 <sup>32-1</sup>																		
unsigned short [int]    16位	0 ~ 65535	0 ~ 2 <sup>16-1</sup>																		
unsigned long [int]    32位	0 ~ 4294967295	0 ~ 2 <sup>32-1</sup>																		
13	<h3>整型常量（整数）</h3> <ul style="list-style-type: none"> <li>■ 整数的表示</li> </ul> <p>三种表现形式：</p> <ul style="list-style-type: none"> <li>□ 十进制整数：正、负号，0~9，首位不是0 例：10, 123</li> <li>□ 八进制整数：正、负号，0~7，首位是0 例：010, 0123</li> <li>□ 十六进制整数：正、负号，0~9, a~f, A~F，前缀是0x, 0X 例：0x10, 0X123</li> </ul>	<p>介绍C语言中整数的三种表现形式及其特点。</p>																		

14	<div><div></div><div><h3>整数的表示</h3><p>123 = 01111011 (B) 二进制 =173 (O) 八进制 =7B (X) 十六进制</p><p>123 0173 0x7b 16 020 0x10 10 012 0xA 10 010 0x10</p><ul style="list-style-type: none"><li>不能超出整型数据的取值范围</li><li>比长整型数还要大的数只能用实数来表示</li></ul></div></div>	举例说明一个整数可以用不同的进制形式来表示，注意括号内的进制标识代码。								
15	<div><div></div><div><h3>整数的类型</h3><p>判断整数的类型</p><ul style="list-style-type: none"><li>整数后的字母后缀<ul style="list-style-type: none"><li>123L long</li><li>123U unsigned</li><li>123LU unsigned long</li></ul></li><li>整数的值</li></ul></div></div>	整数的类型取决于字母后缀。								
16	<div><div></div><div><h3>基本数据类型—字符型</h3><ul style="list-style-type: none"><li>字符具有数值特征 'A' 65 0100 0001</li><li>整型变量和字符变量的定义和赋值可以互换【ASCII码范围】<table><tr><td>char c;</td><td>int i;</td></tr><tr><td>c = 'A';</td><td>i = 65;</td></tr><tr><td>或</td><td>或</td></tr><tr><td>c = 65;</td><td>i = 'A';</td></tr></table></li></ul></div></div>	char c;	int i;	c = 'A';	i = 65;	或	或	c = 65;	i = 'A';	字符型数据存储时是将该字符相应的ASCII 码放到存储单元中，实际上是以二进制形式存放的，因此字符具有数值特征。  比较左右两个例子中 c 和 i 变量的定义类型以及它们的赋值方式。进一步考虑对 c 和 i 两个变量分别可以进行什么运算？输出又有哪些方式可以选择？后面有详细介绍。
char c;	int i;									
c = 'A';	i = 65;									
或	或									
c = 65;	i = 'A';									
17	<div><div></div><div><h3>字符型常量</h3><ul style="list-style-type: none"><li>字符常量 'a'、'A'、'9'、'+'、'\$'（注意：单引号）</li><li>ASCII字符集 列出所有可用的字符（256个）<ul style="list-style-type: none"><li>每个字符：惟一的次序值（ASCII 码）</li><li>'0' ~'9' 升序排列</li><li>'A' ~'Z'</li><li>'a' ~'z'</li></ul></li></ul></div></div>	重点介绍 ASCII 字符集的排列规律，以及一些特殊字符的 ASCII 值。此处可以结合教材附录 2 ASCII 码集进行讲解，如大小写英文字母、数字 0~9 的 ASCII 码范围、常用符号和特殊控制符的 ASCII 码值等。								

18

## 字符的数值特征

### 字符—ASCII 码

对字符进行运算 — 对字符的ASCII 码进行运算

区分数字字符和数字

例如：  
'A' 的 ASCII 码 65  
则：'A'+1=66，对应字符 'B'

19

转义字符

字符形式	所表示字符
\n	换行
\t	横向跳格（即输出若干个空格）
\b	退格（显示输出时，刷新左边一个字符）
\r	回车（输出位置重新移到行首）
\\	反斜杠字符“\”
\'	单引号（撇号）
\ddd	八进制数 ddd 所代表字符，如 \007 为“响”声，\40 即空格。
\xhh	十六进制数 hh 所代表的字符，如 \x41 即'A'，\x20 即空格。

- 反斜杠后跟一个字符或数字
- 字符常量，代表一个字符  
'\n' '\101' '\x41' 'A'
- 所有字符都可以用转义字符表示

20

## 基本数据类型—实型

### 实型（浮点型）数据

- 单精度浮点型 float
- 双精度浮点型 double

	存储	数据精度 (有效数字)	取值范围
float	4字节	七/八位	$\pm(10^{-38} \sim 10^{38})$
double	8字节	十六位	$\pm(10^{308} \sim 10^{308})$

21

## 数据精度和取值范围

- 数据精度与取值范围是两个不同的概念：  
float x = 1234567.89; 1234567.80  
    虽在取值范围内，但无法精确表达。  
float y = 1.2e55;  
    y 的精度要求不高，但超出取值范围。
- 并不是所有的实数都能在计算机中精确表示
- 实型常量的类型都是double

由于 C 语言中的字符具有数值特征，可以像整数一样参加运算，此时相当于对字符的 ASCII 码进行运算。

强调区分数字和数字字符是两个不同的概念。

转义字符使用场合：有一些字符，如回车、退格等控制码，它们不能在屏幕上显示，也无法从键盘输入，只能用转义字符来表示。

转义字符表示方法：由反斜杠跟上一个字符或数字组成，它把反斜杠后面的字符或数字转换成别的意思。

需要强调：虽然转义字符形式上由多个字符组成，但它是字符常量，只代表一个字符，它的使用方法与其他字符常量相同。

实数在 C 语言中又称浮点数，有两种表示形式：十进制形式和指数形式。实型又分为单精度和双精度两类，它们的精度和取值范围不同，以实型为例介绍精度和取值范围是两个不同的概念。

强调实型常量的类型是 double，但一个实型常量也可以赋给一个 float 型变量。如：  

```
float pi;  
pi=3.14159;
```

22	<div><div></div><h3>实型常量（实数、浮点数）</h3><ul style="list-style-type: none"><li>■ 实数的表示<ul style="list-style-type: none"><li>□ 浮点表示法 0.123 123.4 12. .12</li><li>□ 科学计数法 6.026E-27 1.2e+30 1E-5</li></ul></li><li>■ 实数的类型 double</li></ul></div>	<p>浮点表示法：实数由正号、负号、阿拉伯数字 0~9 和小数点组成，必须有小数点，并且小数点的前、后至少一边要有数字。实数的浮点表示法又称实数的小数形式。</p> <p>科学计数法：实数由正号、负号、数字和字母 e（或 E）组成，e 是指数的标志，在 e 之前要有数据，e 之后的指数只能是整数。实数的科学计数法又称实数的指数形式。例如：</p> <p><b>3.14 和 6.026E-27—&gt;合法</b> <b>0.2E2.3 和 E-5—&gt;非法</b></p>																				
23	<div><div></div><h2>6.2 数据的输入和输出</h2><ul style="list-style-type: none"><li>6.2.1 整型数据的输入和输出</li><li>6.2.2 实型数据的输入和输出</li><li>6.2.3 字符型数据的输入和输出</li></ul></div>	<p>本节介绍不同类型数据的输入和输出方法。主要围绕下列四个基本输入输出函数来介绍：</p> <p><b>scanf()</b> <b>printf()</b> <b>getchar()</b> <b>putchar()</b></p>																				
24	<div><div></div><h3>6.2.1 整型数据的输入输出</h3><p><b>printf</b> (格式控制, 输出参数1, ..., 输出参数n); <b>scanf</b> (格式控制, 输入参数1, ..., 输入参数n); 格式控制说明 %...</p><table><tr><th></th><th>十进制</th><th>八进制</th><th>十六进制</th></tr><tr><td>int</td><td>%d</td><td>%o</td><td>%x</td></tr><tr><td>long</td><td>%ld</td><td>%lo</td><td>%lx</td></tr><tr><td>unsigned</td><td>%u</td><td>%o</td><td>%x</td></tr><tr><td>unsigned long</td><td>%lu</td><td>%lo</td><td>%lx</td></tr></table></div>		十进制	八进制	十六进制	int	%d	%o	%x	long	%ld	%lo	%lx	unsigned	%u	%o	%x	unsigned long	%lu	%lo	%lx	<p>本小节介绍整型数据所使用的输入输出函数 scanf()和 printf()的格式。重点需要掌握格式控制符与数据类型的对应关系。</p>
	十进制	八进制	十六进制																			
int	%d	%o	%x																			
long	%ld	%lo	%lx																			
unsigned	%u	%o	%x																			
unsigned long	%lu	%lo	%lx																			
25	<div><div></div><h3>输出整型数据示例（1）</h3><pre>#include &lt;stdio.h&gt; int main(void) {     printf("%d, %o, %x\n", 10, 10, 10);     printf("%d, %d, %d\n", 10, 010, 0x10);     printf("%d, %x\n", 012, 012);     return 0; }</pre><div>10, 12, a 10, 8, 16 10, a</div></div>	<p>根据格式控制说明，可以选用十进制、八进制和十六进制 3 种形式来输出一个整数，同时，该整数也可以有十进制、八进制和十六进制 3 种表现形式，二者可以不一致，输出结果以格式控制说明为准，但该整数的数值是确定的。</p>																				




26	<div> <div> <div></div> <div>输入整型数据示例 (2)</div> </div> <div> <pre>#include &lt;stdio.h&gt; int main(void) {     int a, b;      printf("input a, b:");     scanf("%x%d", &amp;a, &amp;b);     printf("%d%5d\n", a, b); /*%5d指定变量b的输出宽度为5*/     printf("%x, %d\n", a, b);     return 0; }</pre> <div>input a, b: 17 17 15 17 f, 17</div> </div> </div>	<p>该例说明，输入时，用格式控制说明指定的形式来读入数据。以八进制形式读入 17，相当于将 017（即 15）赋值给变量 a。思考：如果将 <code>scanf("%o%d", &amp;a, &amp;b);</code> 改为 <code>scanf("%x%d", &amp;a, &amp;b);</code> 输入数据不变，输出结果会有什么不同？</p> <p>答案： <b>23    17(中间有三个空格)</b> <b>17, 17</b></p>
27	<div> <div> <div></div> <div>6.2.2 实型数据的输入和输出</div> </div> <div> <ul style="list-style-type: none"> <li>■ 输入 <code>scanf()</code> <ul style="list-style-type: none"> <li>□ float: %f 或 %e 以小数或指数形式输入一个单精度浮点数</li> <li>□ double: %lf 或 %le 以小数或指数形式输入一个双精度浮点数</li> </ul> </li> <li>■ 输出 <code>printf()</code> <ul style="list-style-type: none"> <li>float 和 double 使用相同的格式控制说明</li> <li>□ %f 以小数形式输出浮点数，保留6位小数</li> <li>□ %e 以指数形式输出</li> </ul> </li> </ul> </div> </div>	<p>本小节介绍实型数据所使用的输入输出函数 <code>scanf()</code> 和 <code>printf()</code> 的格式。重点需要掌握格式控制符与数据类型的对应关系。尤其要强调 float 型和 double 型数据在输入时格式控制说明符的区别。</p> <p>强调：当程序运行时输入实型数据有误，建议首先检查是否混用了 %f 与 %lf。初学者经常会犯这个错误，务必需要牢记。</p>
28	<div> <div> <div></div> <div>实型数据输出格式示例</div> </div> <div> <pre>#include &lt;stdio.h&gt; int main(void) {     double d = 3.1415926;      printf("%f, %e\n", d, d);     printf("%5.3f, %5.2f, %.2f\n", d, d, d);     return 0; }</pre> <div>3.141593, 3.14159e+00 3.142, 3.14, 3.14</div> <p>一共5位，小数部分3位，小数点占1位</p> </div> </div>	<p>在输出格式控制说明中，可以加宽度限定词，指定实型数据的输出宽度。例如，输出格式控制说明 <code>%m.nf</code>，指定输出浮点型数据时，保留 n 位小数，且输出宽度是 m（包括符号位和小数点）。若数据的实际位数小于 m，左端补空格，若大于 m，按实际位数输出。示例结果讲解如下：</p> <p>输出 d 的值 3.1415926 时，<code>%5.3f</code> 输出 3.142（保留 3 位小数），<code>%5.2f</code> 输出 3.14（保留 2 位小数，左端补 1 个空格），<code>%.2f</code> 输出 3.14（保留 2 位小数，按实际位数）。</p>
29	<div> <div> <div></div> <div>实型数据输入输出示例</div> </div> <div> <p>假定 float 的精度为 7 位，double 的精度为 16 位</p> <pre>#include &lt;stdio.h&gt; int main(void) {     float f;     double d;      printf("input f, d:");     scanf("%f%lf", &amp;f, &amp;d);     printf("f = %f\n d = %f\n", f, d);     d = 1234567890123.12;     printf("d = %f\n", d);     return 0; }</pre> <div>input f, d: 1234567890123.123456 1234567890123.123456 f = 1234567954432.000000 d = 1234567890123.123540 d = 1234567890123.120120</div> </div> </div>	<p>举例说明实型数据的输入和输出应用。示例结果讲解如下：</p> <p>由于 float 型只能保存 7 位有效数字，double 型可以保存 16 位有效数字，输出数据的下划线部分即为有效数字，由此可知，由于精度的关系，并不是所有实数都可以在计算机中精确表示。</p>

30	<div data-bbox="300 203 347 241" data-label="Image"></div> <h3 data-bbox="331 241 695 277">6.2.3 字符型数据输入输出</h3> <ul style="list-style-type: none"> <li data-bbox="344 293 549 416">■ <b>scanf() 和 printf()</b>  <pre data-bbox="373 322 549 416">%c char ch; scanf("%c", &amp;ch); printf("%c", ch);</pre> </li> <li data-bbox="344 445 596 568">■ <b>getchar() 和 putchar()</b>  <pre data-bbox="373 474 517 568">char ch; ch = getchar(); putchar(ch);</pre>           输入输出一个字符         </li> </ul>	<p>本小节介绍字符型数据所使用的输入输出函数的格式。重点需要掌握格式控制符与数据类型的对应关系。</p> <p><b>scanf("%c", &amp;ch) / ch=getchar()</b>  <b>printf("%c", ch) / putchar(ch)</b></p> <p>注意：getchar()和 putchar()都是针对单个字符的输入和输出。通常情况下成对使用。</p>
31	<div data-bbox="300 620 347 658" data-label="Image"></div> <h3 data-bbox="344 658 576 685">输入输出字符示例</h3> <pre data-bbox="344 692 533 999"># include &lt;stdio.h&gt; int main(void) {     char ch1, ch2;      ch1=getchar();     ch2=getchar();     putchar(ch1);     putchar('#');     putchar(ch2);     return 0; }</pre> <div data-bbox="628 763 687 831" data-label="Text"> <p>Ab A#b</p> </div>	<p>举例说明 getchar()和 putchar()的用法。</p> <p>注意：</p> <ol style="list-style-type: none"> <li>(1) getchar()函数中参数可以省略，括号不能省略；</li> <li>(2) putchar()函数也可以输出单个字符常量，参数用单引号括起来。</li> </ol>
32	<div data-bbox="300 1037 347 1075" data-label="Image"></div> <h3 data-bbox="344 1075 576 1102">输入输出字符示例</h3> <pre data-bbox="320 1108 783 1361"># include &lt;stdio.h&gt; int main(void) {     char ch1, ch2, ch3;      scanf("%c%c%c", &amp;ch1, &amp;ch2, &amp;ch3);     printf("%c%c%c%c", ch1, '#', ch2, '#', ch3);     return 0; }</pre> <div data-bbox="544 1122 608 1178" data-label="Text"> <p>AbC A#b#C</p> </div> <div data-bbox="667 1122 730 1178" data-label="Text"> <p>A bC A# #b</p> </div>	<p>输入多个字符时，这些字符之间不能有间隔。如果使用了间隔符（如空格 ' '），由于它本身也是字符，该间隔符就被作为输入字符。在第二组输入数据中，输入字符 A 后，输入了一个空格，所以 ch2 的值是 ' '，ch3 的值是 'b'。</p>
33	<div data-bbox="300 1453 347 1491" data-label="Image"></div> <h3 data-bbox="339 1491 655 1518">例6-1 大小写字母转换</h3> <p data-bbox="320 1525 427 1545">输入一行字符</p> <ul style="list-style-type: none"> <li>将大写字母转换为相应的小写字母后输出</li> <li>小写字母转换为相应的大写字母后输出</li> <li>其他字符原样输出</li> </ul> <div data-bbox="555 1592 788 1637" data-label="Text"> <p>input characters: Reold 123? rEOLD 123?</p> </div> <pre data-bbox="331 1648 743 1794">ch = getchar(); while(ch != '\n'){     while((ch = getchar()) != '\n')         if(ch &gt;= 'A' &amp;&amp; ch &lt;= 'Z') ch = ch - 'A' + 'a';         else if((ch &gt;= 'a' &amp;&amp; ch &lt;= 'z') ch = ch - 'a' + 'A';     putchar(ch);     ch = getchar(); }</pre> <div data-bbox="592 1749 775 1816" data-label="Text"> <p>(ch = getchar()) != '\n' ch = getchar() != '\n' 等价吗？</p> </div>	<p>该例题是一个包含数据表达、运算和流程控制的典型示例，程序中涉及到字符型数据的表示，算术、赋值和关系运算，在实现过程中使用了循环和分支结构，编程时综合运用了前 5 章介绍的编程思想和语言知识。</p>

34	<div><div><div>大小写英文字母转换</div><div>'b' - 'a' = 'B' - 'A'</div><div>.....</div><div>'z' - 'a' = 'Z' - 'A'</div><div></div><div>'m' &lt;=&gt; 'M'</div></div><div><div>'a' -&gt; 'A'</div><div>'A' -&gt; 'a'</div><div>'m' - 'a' + 'A' = 'M'</div><div>'M' - 'A' + 'a' = 'm'</div></div><div><div>数字字符和数字转换</div><div>9 - 0 = '9' - '0'</div><div>'9' = 9 + '0'</div><div></div><div>'8' &lt;=&gt; 8</div><div>'8' -&gt; 8</div><div>8 -&gt; '8'</div><div>'8' - '0' = 8</div><div>8 + '0' = '8'</div></div></div> <div>字符运算</div>	<p>介绍一些在编程中经常用到的字符运算，要求寻找并总结规律。假设 ch 为 char 类型，则：</p> <ol style="list-style-type: none"><li>(1) 大写字母—&gt;小写字母: ch=ch-'A'+‘a’;</li><li>(2) 小写字母—&gt;大写字母: ch=ch-'a'+‘A’;</li><li>(3) 数字字符—&gt;数值: ch=ch-'0’;</li><li>(4) 数值—&gt;数字字符: ch=ch+'0’;</li><li>(5) 特殊转换，如 ‘A’—&gt;‘Z’， ‘B’—&gt;‘Y’，..... ‘Z’—&gt; ‘A’: ch=‘A’+‘Z’-ch;</li></ol>
35	<div><div>6.3 类型转换</div><div>不同类型数据的混合运算，先转换为同一类型，再运算。</div><div>6.3.1 自动类型转换</div><div><div>非赋值运算的类型转换</div><div>赋值运算的类型转换</div></div><div>6.3.2 强制类型转换</div></div>	<p>本节主要介绍不同类型数据的混合运算，运算的前提是这些数据首先要转换成同一类型。数据类型的转换包括：</p> <ol style="list-style-type: none"><li>(1) 自动转换：由 C 语言编译系统自动完成；</li><li>(2) 强制转换：通过特定的运算完成。</li></ol>
36	<div><div>6.3.1 自动类型转换（非赋值运算）</div><div><div>水平方向：自动</div><div>垂直方向：低 → 高</div></div><div><div>高</div><div>↑</div><div>double</div><div>↑</div><div>unsigned long</div><div>↑</div><div>unsigned</div><div>↑</div><div>int</div><div>低</div></div><div><div>← float</div><div>← long</div><div>← unsigned short</div><div>← char, short</div></div></div>	<p>自动类型转换之非赋值运算方式：转换规则所遵循的原则为保证运算的精度不降低。</p>
37	<div><div>自动类型转换（非赋值运算）</div><div>'A' + 12 - 10.05</div><div>65</div><div>77</div><div>66.95</div><div><div>高</div><div>↑</div><div>double</div><div>↑</div><div>unsigned long</div><div>↑</div><div>unsigned</div><div>↑</div><div>int</div><div>低</div></div><div><div>← float</div><div>← long</div><div>← unsigned short</div><div>← char, short</div></div></div>	<p>举例说明自动转换的详细步骤。</p>

38	<div><div><div>自动类型转换（赋值运算）</div><div>变量 = 表达式</div><div><div><div>■ 计算赋值运算符右侧表达式的值</div><div>■ 将赋值运算符右侧表达式的值赋给左侧的变量</div></div><div><div>将赋值运算符右侧表达式的类型 自动转换成 赋值号左侧变量的类型</div></div></div></div></div>	自动类型转换之赋值运算方式：赋值运算时，将赋值号右侧表达式的类型自动转换成赋值号左侧变量的类型。
39	<div><div><div>自动类型转换（赋值运算）</div><div><div><div>double x; x = 1;      x = ?</div><div>short a = 1000; char b = 'A'; long c; c = a + b;      c = ?</div><div>int ai; ai = 2.56;      ai = ?</div><div>short bi; bi = 0x12345678L bi = ?</div></div></div></div></div>	利用这条规则时，如果赋值号右侧表达式的类型比赋值号左侧变量的类型级别高，运算精度会降低。因此，在赋值运算时，赋值号两侧数据的类型最好相同，至少右侧数据的类型比左侧数据的类型级别低，或者右侧数据的值在左侧变量的取值范围内，否则，会导致运算精度降低，甚至出现意想不到的结果。
40	<div><div><div>6.3.2 强制类型转换</div><div>强制类型转换运算符 (类型名) 表达式</div><div><div><div>(double)3 (int)3.8 (double)(5/2) (double)5/2</div><div><div>3.0 3 2.0 2.5</div></div></div></div></div></div>	使用强制类型转换运算符，可以将一个表达式转换成给定的类型。注意：强制类型转换是运算符而不是函数，故 (类型名) 表达式      √ 类型名 (表达式)      × 强制类型转换运算符的优先级较高，与自增运算符 ++ 相同，它的结合性是从右到左。例如： (int) 3.8 + 1.3    ->    4.3 (int) (3.8 + 1.3) ->    5
41	<div><div><div>强制类型转换示例</div><div><div>#include &lt;stdio.h&gt; int main(void) {     int i;     double x;      x = 3.8;     i = (int) x;     printf("x = %f, i = %d \n", x, i);     printf("(double)(int)x = %f\n", (double)(int)x);     printf(" x mod 3 = %d\n", (int)x % 3);     return 0; }</div><div><div>x = 3.800000, i = 3 (double)(int)x = 3.000000 x mod 3 = 0</div></div></div></div></div>	注意：无论是自动类型转换，还是强制类型转换，都是为了本次运算的需要，对数据的类型进行临时转换，并没有改变数据的定义。

42	<div><div></div><h2>6.4 表达式</h2><p>表达式：由运算符和运算对象（操作数）组成的有意义的运算式子，它的值和类型由参加运算的运算符和运算对象决定。</p><ul style="list-style-type: none"><li>运算符：具有运算功能的符号</li><li>运算对象：常量、变量和函数等表达式</li></ul><p>算术表达式、赋值表达式、关系表达式、逻辑表达式、条件表达式和逗号表达式等</p></div>	本节主要介绍了C语言的多种表达式和相应的运算符。要求准确掌握表达式的语法规则以及运算符的运算规则，并能做到灵活使用。																				
43	<div><div></div><h3>6.4.1 算术表达式—算术运算符</h3><ul style="list-style-type: none"><li>单目 <code>+</code> <code>-</code> <code>++</code> <code>--</code></li><li>双目 <code>+</code> <code>-</code> <code>*</code> <code>/</code> <code>%</code></li></ul><p>注意</p><ul style="list-style-type: none"><li><code>/</code> 整数除整数，得整数 <code>1/4 = 0</code>, <code>10/3 = 3</code></li><li><code>%</code> 模(求余)：针对整型数据 <code>5%6 = 5</code>, <code>9%4 = 1</code>, <code>100%4 = 0</code></li><li><code>+</code> 和 <code>-</code><ul style="list-style-type: none"><li>单目运算符，<code>+10</code> 和 <code>-10</code></li><li>双目运算符，<code>x+10</code> 和 <code>y-10</code></li></ul></li><li>双目运算符两侧操作数的类型要相同，否则，自动类型转换后，再运算。</li></ul></div>	本小节主要介绍算术表达式的语法规则和运算规律。算术运算符分为单目运算符和双目运算符两类，单目运算符只需要一个操作数，而双目运算符需要两个操作数。 注意： (1) 整除和取余运算都针对整型数据； (2) <code>+</code> 和 <code>-</code> 既可以当作单目运算又可以看作双目运算。																				
44	<div><div></div><h3>自增运算符++和自减运算符--</h3><pre>int n;</pre><p><code>n++</code> <code>++n</code> <code>n--</code> <code>--n</code> (只适合变量运算)</p><ul style="list-style-type: none"><li>使变量的值增1或减1<ul style="list-style-type: none"><li><code>++n</code> <code>n++</code>     <code>n = n + 1</code></li><li><code>--n</code> <code>n--</code>     <code>n = n - 1</code></li></ul></li><li>取变量的值作为表达式的值<ul style="list-style-type: none"><li><code>++n</code>: <code>n = n + 1</code>; 取<code>n</code>值作为表达式<code>++n</code>的值</li><li><code>n++</code>: 取<code>n</code>值作为表达式<code>n++</code>的值; <code>n = n + 1</code></li></ul></li></ul></div>	自增运算符和自减运算符的运算对象只能是变量，不能是常量或表达式。如： <b>3++ 或 ++(i+j) 一&gt;非法的表达式</b>																				
45	<div><div></div><h3>自增运算和自减运算</h3><pre>int n, m;</pre><table><tr><td><code>n=n+1</code></td><td><code>n=2;</code></td><td><code>n=3</code></td><td><code>m=3</code></td></tr><tr><td><code>m=n</code></td><td><code>m=++n;</code></td><td></td><td></td></tr><tr><td colspan="4"></td></tr><tr><td><code>m=n</code></td><td><code>n=2;</code></td><td><code>n=3</code></td><td><code>m=2</code></td></tr><tr><td><code>m=n+1</code></td><td><code>m=n++;</code></td><td></td><td></td></tr></table></div>	<code>n=n+1</code>	<code>n=2;</code>	<code>n=3</code>	<code>m=3</code>	<code>m=n</code>	<code>m=++n;</code>							<code>m=n</code>	<code>n=2;</code>	<code>n=3</code>	<code>m=2</code>	<code>m=n+1</code>	<code>m=n++;</code>			举例说明自增和自减运算的用法，注意 <code>n++</code> 和 <code>++n</code> 的区别。 (1) <code>++n</code> 的运算顺序是：先执行 <code>n = n + 1</code> ，再将 <code>n</code> 的值作为表达式 <code>++n</code> 的值； (2) <code>n++</code> 的运算顺序是：先将 <code>n</code> 的值作为表达式 <code>n++</code> 的值，再执行 <code>n = n + 1</code> 。
<code>n=n+1</code>	<code>n=2;</code>	<code>n=3</code>	<code>m=3</code>																			
<code>m=n</code>	<code>m=++n;</code>																					
<code>m=n</code>	<code>n=2;</code>	<code>n=3</code>	<code>m=2</code>																			
<code>m=n+1</code>	<code>m=n++;</code>																					

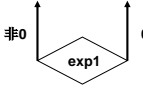
46	<p><b>算术运算符的优先级和结合性</b></p> <p>单目 + - ++ --      高      从右向左</p> <p>双目 * / %      ↑</p> <p>双目 + -      低</p> <p><math>-5 + 3\%2 = (-5) + (3\%2) = -4</math>  <math>3 * 5 \% 3 = (3*5) \% 3 = 0</math>  <math>-i++</math>      <math>-(i++)</math></p>	<p>在 C 语言中，计算表达式的值也需要按运算符的优先级从高到低顺序计算。</p> <p>如果操作数两侧运算符的优先级相同，则按结合性（结合方向）决定计算顺序，若结合方向为“从左到右”，则操作数先与左面的运算符结合；若结合方向为“从右到左”，则操作数先与右面的运算符结合。具体可参阅教材附表 2。</p>
47	<p><b>写出C表达式</b></p> <p>数学式 → C算术表达式</p> <p><math>s(s-a)(s-b)(s-c)</math></p> <p><math>(x+2)e^{2x}</math></p> <p><math>\frac{-b + \sqrt{b^2 - 4ac}}{2a}</math></p>	<p>C 算术表达式分别为：</p> <p><math>s*(s-a)*(s-b)*(s-c)</math>  <math>(x+2)*\exp(2*x)</math>  <math>(-b+\text{sqrt}(b*b-4*a*c))/(2*a)</math></p>
48	<p><b>6.4.2 赋值表达式</b></p> <p>■ 赋值运算符 =</p> <p><math>x = 3*4</math>          优先级较低，结合性从右向左  <math>x = y = 3</math>    <math>x = (y = 3)</math></p>	<p>本小节主要介绍赋值表达式的语法规则和运算规律。</p> <p>赋值运算符的特点：优先级别较低，结合性是从右向左。</p>
49	<p><b>赋值表达式</b></p> <p>右侧表达式的类型自动转换成左侧变量的类型</p> <p>变量 = 表达式</p> <ul style="list-style-type: none"> <li>□ 计算赋值运算符右侧表达式的值</li> <li>□ 将赋值运算符右侧表达式的值赋给左侧的变量</li> <li>□ 将赋值运算符左侧的变量的值作为表达式的值</li> </ul> <pre>int n; double x, y; n = 3.14 * 2; x = 10 / 4; x = (y = 3);</pre>	<p>解答如下：</p> <p><b>n=6</b>          解析：先计算 <math>3.14*2</math> 得 6.28，再通过赋值运算符将该结果赋给整型变量 n，赋值过程中自动转换为整型，因此 n 的值为 6；</p> <p><b>x=2.0</b>          解析：先计算 10 整除 4 得 2，再通过赋值运算符将该结果赋给实型变量 x，赋值过程中自动转换为实型，因此 x 的值为 2.0；</p> <p><b>x=3.0</b>          解析：先计算赋值表达式 <math>y=3</math> 的值得 3.0，再通过赋值运算符将该结果赋给实型变量 x，因此 x 的值为 3.0。</p>





50	<div> <div></div> <div>复合赋值运算符</div> <ul style="list-style-type: none"> <li>■ 赋值运算符 <ul style="list-style-type: none"> <li>□ 简单赋值运算符 =</li> <li>□ 复合赋值运算符 <ul style="list-style-type: none"> <li>■ 复合算术赋值运算符 += -= *= /= %=</li> <li>■ 复合位赋值运算符</li> </ul> </li> </ul> </li> <li>■ 赋值表达式 <div>变量 赋值运算符 表达式</div> <div><math>x += \text{exp}</math> 等价于 <math>x = x + \text{exp}</math></div> <div><math>x *= y - 3</math>      <math>x = x * (y - 3)</math></div> </li> </ul> </div>	<p>注意: <math>x *= y - 3</math> 等价于 <math>x = x * (y - 3)</math>, 而不是 <math>x = x * y - 3</math>。</p>
51	<div> <div></div> <div>6.4.3 关系表达式—关系运算符</div> <ul style="list-style-type: none"> <li>■ 比较两个操作数, 比较的结果: 真 假 <div> <math>x &lt; y</math>   <math>x &lt;= y</math>   <math>x == y</math>  <math>x &gt; y</math>   <math>x &gt;= y</math>   <math>x != y</math> </div> </li> <li>■ 优先级 <div> <div>□ 算术运算符</div> <div>□ &lt; &lt;= &gt; &gt;=</div> <div>□ == !=</div> <div>□ 赋值运算符</div> </div> <div> <math>a &gt; b == c</math>      <math>(a &gt; b) == c</math>  <math>d = a &gt; b</math>      <math>d = (a &gt; b)</math>  <math>ch &gt; 'a' + 1</math>    <math>ch &gt; ('a' + 1)</math>  <math>d = a + b &gt; c</math>   <math>d = ((a + b) &gt; c)</math>  <math>3 &lt;= x &lt;= 5</math>    <math>(3 &lt;= x) &lt;= 5</math>  <math>b - 1 == a != c</math>   <math>((b - 1) == a) != c</math> </div> </li> <li>■ 左结合</li> </ul> </div>	<p>本小节主要介绍关系表达式的语法规则和运算规律。</p> <p>关系运算符的特点: 优先级低于算术运算符, 高于赋值运算符和逗号运算符, 结合性是从左向右。</p> <p>逗号表达式常用于 for 循环语句中对循环变量进行初始化。</p>
52	<div> <div></div> <div>关系表达式</div> <ul style="list-style-type: none"> <li>■ 用关系运算符将2个表达式连接起来的式子哪些是关系表达式? <div> <math>a &gt; b == c</math>      0    <math>\text{char ch} = 'w';</math>  <math>d = a &gt; b</math>        0    <math>\text{int a} = 2, b = 3, c = 1, d, x = 10;</math>  <math>ch &gt; 'a' + 1</math>     1  <math>d = a + b &gt; c</math>     1  <math>b - 1 == a != c</math>   0  <math>3 &lt;= x &lt;= 5</math>      1 </div> </li> <li>■ 关系运算的结果 <div> <div>□ 真    1</div> <div>□ 假    0</div> </div> </li> </ul> </div>	<p>关系表达式的运算结果是逻辑量。由于 C 语言没有逻辑型数据, 就用整数 1 代表“真”, 0 代表“假”。</p> <p>解析: 判断一个复杂表达式是否为关系表达式, 只要看其最后一步运算是否为关系运算即可。</p> <p>注意: 表达式 <math>3 &lt;= x &lt;= 5</math> 等价于 <math>(3 &lt;= x) &lt;= 5</math>, 无论 x 取何值, 关系表达式 <math>3 &lt;= x</math> 的值不是 1 就是 0, 都小于 5, 即 <math>3 &lt;= x &lt;= 5</math> 的值恒为 1。由此看出, 表达式 <math>3 &lt;= x &lt;= 5</math> 无法正确表示代数式 “<math>3 &lt;= x &lt;= 5</math>”。</p>
53	<div> <div></div> <div>6.4.4 逻辑表达式—逻辑运算符</div> <div> <math>(ch &gt;= 'a' \ \&amp;\&amp; \ ch &lt;= 'z') \    \ (ch &gt;= 'A' \ \&amp;\&amp; \ ch &lt;= 'Z')</math>  <math>ch == ' ' \    \ ch == '\n'</math>  <math>x &gt;= 3 \ \&amp;\&amp; \ x &lt;= 5</math> </div> <ul style="list-style-type: none"> <li>■ &amp;&amp;    !</li> <li>■ 逻辑运算结果: 1(真) 0(假)</li> <li>■ 逻辑运算对象: 关系表达式或逻辑量 <div> <math>x &gt;= 3 \ \&amp;\&amp; \ x &lt;= 5</math>      !x </div> </li> <li>■ 判断逻辑量的真假: 非0(真) 0(假)</li> </ul> </div>	<p>本小节主要介绍逻辑表达式的语法规则和运算规律。</p> <p>强调: 逻辑运算对象是值为“真”或“假”的逻辑量, 它可以是任何类型的数据, 如整型、浮点型、字符型等, C 编译系统以非 0 和 0 判定“真”和“假”。</p>



54	<div><div></div><div>逻辑运算的规则—真值表</div><table><tr><th>a</th><th>b</th><th>a&amp;&amp;b</th><th>a  b</th><th>!a</th></tr><tr><td>1</td><td>1</td><td>真</td><td>真</td><td>假</td></tr><tr><td>1</td><td>0</td><td>假</td><td>真</td><td>假</td></tr><tr><td>0</td><td>1</td><td>假</td><td>真</td><td>真</td></tr><tr><td>0</td><td>0</td><td>假</td><td>假</td><td>真</td></tr></table></div>	a	b	a&&b	a  b	!a	1	1	真	真	假	1	0	假	真	假	0	1	假	真	真	0	0	假	假	真	要求掌握如何填写真值表。
a	b	a&&b	a  b	!a																							
1	1	真	真	假																							
1	0	假	真	假																							
0	1	假	真	真																							
0	0	假	假	真																							
55	<div><div></div><div>逻辑运算符的优先级和结合性</div><div><div><div>■ 优先级</div><div><div><div><div>□ !</div><div>算术运算符</div><div>关系运算符</div><div>&amp;&amp;</div><div>  </div><div>赋值运算符</div></div><div>左结合</div></div><div><div>a    b &amp;&amp; c</div><div>!a &amp;&amp; b</div><div>x &gt;= 3 &amp;&amp; x &lt;= 5</div><div>!x == 2</div><div>a    3 + 10 &amp;&amp; 2</div></div><div><div>a    (b &amp;&amp; c)</div><div>(!a) &amp;&amp; b</div><div>(x &gt;= 3) &amp;&amp; (x &lt;= 5)</div><div>(!x) == 2</div><div>a    ((3 + 10) &amp;&amp; 2)</div></div></div></div></div></div>	逻辑运算符的特点：逻辑非的优先级高于算术运算符和关系运算符，其次按顺序为逻辑与、逻辑或，均高于赋值运算符和逗号运算符，结合性是从左向右。																									
56	<div><div></div><div>逻辑表达式</div><div>用逻辑运算符将关系表达式或逻辑量连接起来的式子</div><div><div>哪些是逻辑表达式？</div><table><tr><td>a &amp;&amp; b</td><td>0</td></tr><tr><td>a    b &amp;&amp; c</td><td>1</td></tr><tr><td>!a &amp;&amp; b</td><td>0</td></tr><tr><td>a    3+10 &amp;&amp; 2</td><td>1</td></tr><tr><td>!(x == 2)</td><td>1</td></tr><tr><td>!x == 2</td><td>0</td></tr><tr><td>ch    b</td><td>1</td></tr></table></div><div><div>char ch = 'w'; int a = 2, b = 0, c = 0; float x = 3.0;</div><div><div>exp1 &amp;&amp; exp2</div><div>先算exp1，若其值为0， STOP</div><div>exp1    exp2</div><div>先算exp1，若其值为1， STOP</div></div></div></div>	a && b	0	a    b && c	1	!a && b	0	a    3+10 && 2	1	!(x == 2)	1	!x == 2	0	ch    b	1	<p>解析：判断一个复杂表达式是否为逻辑表达式，只要看其最后一步运算是否为逻辑运算即可。</p> <p>强调：求解用逻辑运算符 &amp;&amp; 或者    连接的逻辑表达式时，按从左到右的顺序计算该运算符两侧的操作数，一旦能得到表达式的结果，就停止计算。例如：</p> <p>（1）解逻辑表达式 exp1 &amp;&amp; exp2 时，先算 exp1，若其值为 0，则 exp1 &amp;&amp; exp2 的值一定是 0。此时，已经没有必要计算 exp2 的值；</p> <p>（2）求解逻辑表达式 exp1    exp2 时，先算 exp1，若其值为非 0，则 exp1    exp2 的值一定是 1。此时，也不必计算 exp2 的值。</p>											
a && b	0																										
a    b && c	1																										
!a && b	0																										
a    3+10 && 2	1																										
!(x == 2)	1																										
!x == 2	0																										
ch    b	1																										
57	<div><div></div><div>例[6-4]写出满足要求的逻辑表达式</div><div><div><div>■ x 为零</div><div><div><div>□ 关系表达式 x == 0</div><div>等价</div><div>□ 逻辑表达式 !x</div></div><div>■ x 不为零</div><div><div><div>□ x != 0</div><div>□ x</div></div><div>■ x 和 y 不同时为零</div><div><div><div>□ !(x == 0 &amp;&amp; y == 0)</div><div>□ x != 0    y != 0</div><div>□ x    y</div></div></div></div><div><div><div><div>x取0</div><div>x取非0</div></div><div><div>x==0 真</div><div>x==0 假</div></div></div><div><div><div><div>x取0</div><div>x取非0</div></div><div><div>!x 真</div><div>!x 假</div></div></div></div></div></div></div></div></div>	要求练习用真值表证明。																									



58	<div data-bbox="300 203 794 551"> <h3>6.4.5 条件表达式</h3> <p><b>exp1 ? exp2 : exp3</b></p>  <math display="block">y = \begin{cases} x+2 &amp; x&gt;0 \\ x^2 &amp; x \leq 0 \end{cases}</math> <p><math>y = (x&gt;0) ? x+2 : x*x;</math></p> <pre> int n; (n&gt;0) ? 2.9 : 1 n = 10      2.9 n = -10     1.0  if ( a&gt;b )     z = a; else     z = b;  z = (a&gt;b) ? a : b;  if ( x&gt;0 )     y=x+2; else     y=x*x; </pre> </div>	<p>本小节主要介绍条件表达式的语法规则和运算规律。条件运算符是 C 语言中唯一的一个三目运算符，它将 3 个表达式连在一起，组成条件表达式。一般形式是：<b>表达式 1 ? 表达式 2 : 表达式 3</b></p> <p>运算过程是：先计算表达式 1 的值，如果它的值为非 0（真），将表达式 2 的值作为条件表达式的值，否则，将表达式 3 的值作为条件表达式的值；</p> <p>条件运算符的优先级较低，只比赋值运算符高。它的结合方向是自右向左。</p>
59	<div data-bbox="300 660 794 987"> <h3>6.4.6 逗号表达式</h3> <p><b>表达式1, 表达式2, ....., 表达式n</b></p> <p>先计算表达式 1，然后计算表达式 2, ....., 最后计算表达式 n 的值，并将表达式 n 的值作为逗号表达式的值。</p> <pre> int a, b, c; (a=2), (b=3), (c=a+b);    a=2, b=3, c=a+b  逗号运算符的优先级最低，左结合 </pre> </div>	<p>本小节主要介绍逗号表达式的语法规则和运算规律。</p> <p>逗号运算符的优先级是所有运算符中最低的，它的结合性是从左到右。</p>
60	<div data-bbox="300 1077 794 1458"> <h3>逗号表达式的用途</h3> <pre> sum = 0; for (i = 0; i &lt;= 100; i++){     sum = sum + i; }  for (i = 0, sum = 0; i &lt;= 100; i++){     sum = sum + i; }  for (i = 0, sum = 0; i &lt;= 100; i++){     sum += i; } </pre> </div>	<p>逗号表达式一个常用的场合就是在 for 循环中为多个循环控制变量做初始化。</p>
61	<div data-bbox="300 1494 794 1843"> <h3>6.4.7 位运算</h3> <ul style="list-style-type: none"> <li>■ 位逻辑运算 <ul style="list-style-type: none"> <li>~ 按位取反                      单目 右结合</li> <li>&amp; 按位与</li> <li>^ 按位异或：相同取0，不同取1</li> <li>  按位或</li> </ul> </li> <li>■ 移位运算 <ul style="list-style-type: none"> <li>&lt;&lt; 对操作数左移给出的位数</li> <li>&gt;&gt; 对操作数右移给出的位数</li> </ul> </li> <li>■ 复合位赋值运算</li> </ul> </div>	<p>本小节主要介绍位运算符的运算规则。</p> <p>在使用位运算符时，注意以下几点：</p> <ol style="list-style-type: none"> <li>① 位运算符中除~是单目运算，其余均为二目运算；</li> <li>② 位运算符所操作的操作数只能是整型或字符型的数据以及它们的变体；</li> <li>③ 操作数的移位运算不改变原操作数的值。</li> </ol>

62	<div>  <h3>位逻辑运算</h3> <p> ~ 按位取反  &amp; 按位与  ^ 按位异或：相同取0，不同取1    按位或 </p> <p> x=0    00000000 00000000  y=3    00000000 00000011  x &amp; y   00000000 00000000  x   y   00000000 00000011  x ^ y   00000000 00000011 </p> <p>1010 ^ 0101 =1111</p> </div> <div> <p>注意区分：</p> <p>&amp; 和  </p> <p>&amp;&amp; 和   </p> <p>x &amp;&amp; y得 0</p> <p>x    y得 1</p> </div>	<p>位逻辑运算符的运算规则：</p> <p>先将两个操作数（int 或 char 类型）化为二进制数，然后按位运算。</p> <p>对于位异或运算 ^ 有几个特殊的操作：</p> <p>(1) a ^ a 的值为 0；</p> <p>(2) a ^ ~a = 二进制全 1 （如果 a 以 16 位二进制表示，则为 65535）；</p> <p>(3) ~(a ^ ~a) = 0；</p> <p>(4) a 和 b 的值互换，可执行语句：</p> <p>a ^= b ^= a ^= b；</p>
63	<div>  <h3>位移位运算</h3> <p> &lt;&lt; 对操作数左移给出的位数  &gt;&gt; 对操作数右移给出的位数 </p> <p> x&lt;&lt;3 将x向左移3位，空出的位用零填补  00111010 &lt;&lt; 3  11010000 </p> <p> x&gt;&gt;3 将x向右移3位  00111010 &gt;&gt; 3  00000111 </p> </div>	<p>移位运算是指对操作数以二进制位为单位进行左移或右移的操作。移位运算具体实现有 3 种方式：循环移位、逻辑移位和算术移位（带符号）。</p> <p>①循环移位：在循环移位中，移入的位等于移出的位；</p> <p>②逻辑移位：在逻辑移位中，移出的位丢失，移入的位取 0；</p> <p>③算术移位：在算术移位中（带符号），移出的位丢失，左移入的位取 0，右移入的位取符号位，即最高位代表数据符号，保持不变。</p>
64	<div>  <h3>复合位赋值运算符</h3> <p> &amp;=   =  ^=  &gt;&gt;=  &lt;&lt;= </p> <p> a &amp;= b 相当于 a = a &amp; b  a &lt;&lt;= 2 相当于 a = a &lt;&lt; 2 </p> </div>	<p>复合位赋值运算符就是在 = 前加上位运算符。</p>
65	<div>  <h3>6.4.8 其他运算</h3> <p> ■ 长度运算符 <b>sizeof</b>  单目运算符，计算变量或数据类型的字节长度  int a;  sizeof(a)  求整型变量 a 的长度，值为4(bytes)  sizeof(int)  求整型的长度，值为4 (bytes)  sizeof(double)  求双精度浮点型的长度，值为8 (bytes) </p> </div>	<p>本小节主要介绍其他一些比较特殊的，具有专门用途运算符。</p> <p>长度运算符 sizeof 是一个单目运算符，用来返回变量或数据类型的字节长度。优点在于使用长度运算符可以增强程序的可移植性，使之不受具体计算机数据类型长度的限制。</p>

66	<p><b>运算符的优先级和结合性</b></p> <ul style="list-style-type: none"> <li>■ ( )</li> <li>■ ! - + ++ -- (类型名) sizeof</li> <li>■ * / %</li> <li>■ + -</li> <li>■ &lt; &lt;= &gt; &gt;=</li> <li>■ == !=</li> <li>■ &amp;&amp;</li> <li>■   </li> <li>■ ? :</li> <li>■ = += -= *= /= %=</li> <li>■ ,</li> </ul>	<p>总结本章所介绍的运算符的优先级和结合性，特点如下：</p> <p>(1) C 语言中，运算符共分 15 个优先级，分别用 1~15 来表示，1 表示优先级最高，15 表示优先级最低。各个运算符的优先级可参见附录 1；</p> <p>C 语言中运算符的结合性分两类，左结合（从左到右）和右结合（从右到左）。单目运算符、三目运算符和赋值运算符的结合性是从右到左，其他运算符的结合性是从左到右。</p>
67	<p><b>例6-5 统计单词个数</b></p> <p>统计一行字符串中单词的个数 单词：单词之间用空格分隔，空格数可以是多个</p> <pre> word = 0; ch = getchar(); while (ch != '\n'){     if (ch == ' '){         word = 0;     }else if (word == 0){ /* 单词的开始 */         count++;         word = 1;     }     ch = getchar(); } </pre> <p>Let's go to room 209. 5</p> <p>67</p>	<p>该例题中，变量 word 的值为 0 表示前一个字符是空格，为 1 表示前一个字符不是空格。</p> <p>当前一个字符是空格且当前字符是非空格时是新单词的开始，count++</p>

## 6.3 练习与习题参考答案

### 6.3.1 练习参考答案

6-1 输入一个十进制数，输出相应的八进制数和十六进制数。例如：输入 31，输出 37 和 1F。

解答：

```

#include <stdio.h>
int main(void)
{
    int x;

    printf("Enter x:");
    scanf("%d", &x);
    printf("%o %x\n", x, x);
    return 0;
}

```

6-2 在程序段：

```
printf("input a, b:");
```

```
scanf("%o%d", &a, &b);
printf("%d%5d\n", a, b);
```

中，如果将 `scanf("%o%d", &a, &b)` 改为 `scanf("%x%d", &a, &b)`，仍输入 17 17，输出是什么？

解答： 23 17(中间有三个空格)

6-3 英文字母转换。输入一行字符，将其中的英文字母转换后输出，其他字符按原样输出。其中英文字母（a~z 或 A~Z）的转换规则是：将当前字母替换为字母表中的后一个字母，同时将小写字母转换为大写，大写字母转换为小写字母，如 'a' -> 'B'、'C' -> 'd'，但是 'Z' -> 'a'、'z' -> 'A'。试编写相应程序。

解答：

```
#include <stdio.h>
int main(void)
{
    char ch;

    while((ch=getchar())!='\n'){
        if(ch>='a' && ch<='z'){
            ch=ch-('a'-'A')+1;
            if(ch>'Z') ch='A';
        }
        else if(ch>='A' && ch<='Z'){
            ch=ch+('a'-'A')+1;
            if(ch>'z') ch='a';
        }
        putchar(ch);
    }
    return 0;
}
```

6-4 证明下列等价关系。

(1)  $a \&\&(b|c)$  等价于  $a\&\&b||a\&\&c$

解答：

a	b	c	$a\&\&(b c)$	$a\&\&b$	$a\&\&c$	$a\&\&b  a\&\&c$
0(假)	0(假)	0(假)	0	0	0	0
0(假)	0(假)	非 0(真)	0	0	0	0
0(假)	非 0(真)	0	0	0	0	0
0(假)	非 0(真)	非 0(真)	0	0	0	0
非 0(真)	0(假)	0(假)	0	0	0	0

非 0(真)	0(假)	非 0(真)	1	0	1	1
非 0(真)	非 0(真)	0(假)	1	1	0	1
非 0(真)	非 0(真)	非 0(真)	1	1	1	1

(2)  $a \parallel (b \& \& c)$  等价于  $(a \parallel b) \& \& (a \parallel c)$

解答:

a	b	c	$b \& \& c$	$a \parallel (b \& \& c)$	$a \parallel b$	$a \parallel c$	$(a \parallel b) \& \& (a \parallel c)$
0(假)	0(假)	0(假)	0	0	0	0	0
0(假)	0(假)	非 0(真)	0	0	0	1	0
0(假)	非 0(真)	0	0	0	1	0	0
0(假)	非 0(真)	非 0(真)	1	1	1	1	1
非 0(真)	0(假)	0(假)	0	1	1	1	1
非 0(真)	0(假)	非 0(真)	0	1	1	1	1
非 0(真)	非 0(真)	0(假)	0	1	1	1	1
非 0(真)	非 0(真)	非 0(真)	1	1	1	1	1

(3)  $!(a \& \& b)$  等价于  $!a \parallel !b$

解答:

a	b	$a \& \& b$	$!(a \& \& b)$	$!a$	$!b$	$!a \parallel !b$
0(假)	0(假)	0	1	1	1	1
0(假)	非 0(真)	0	1	1	0	1
非 0(真)	0(假)	0	1	0	1	1
非 0(真)	非 0(真)	1	0	0	0	0

(4)  $!(a \parallel b)$  等价于  $!a \& \& !b$

解答:

a	b	$a \parallel b$	$!(a \parallel b)$	$!a$	$!b$	$!a \& \& !b$
0(假)	0(假)	0	1	1	1	1
0(假)	非 0(真)	1	0	1	0	0
非 0(真)	0(假)	1	0	0	1	0
非 0(真)	非 0(真)	1	0	0	0	0

### 6.3.2 习题参考答案

#### 一. 选择题

1	2	3	4	5	6
D	A	C	D	C	B

#### 二. 填空题

1	原码: 1000000001111111	2	7
	反码: 1111111110000000		
	补码: 1111111110000001		
3	1	4	x != 0
	0		
	1		
5	1		
	135		

#### 三. 程序设计题

1. 分类统计字符个数: 输入一行字符, 统计出其中的英文字母、空格、数字和其他字符的个数。试编写相应程序。

解答:

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

```
int main(void)
```

```
{
```

```
    char c;
```

```
    int blank, digit, letter, other;
```

```
    printf("请输入一行字符, 以回车符结束: ");
```

```
    c = getchar();
```

```
    blank = digit = letter = other = 0;
```

```
    while(c != '\n'){
```

```
        if(c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
```

```
            letter++;
```

```
        else if(c >= '0' && c <= '9')
```

```
            digit++;
```

```
        else if(c == ' ')
```

```
            blank++;
```

```
        else
```

```
            other++;
```

```
        c = getchar();
```

```
    }
```

```
    printf("letter = %d, blank = %d, digit = %d, other = %d\n", letter, blank, digit, other);
```

```
    return 0;
```

```
}
```

2. 使用函数累加由 n 个 a 构成的整数之和: 输入两个正整数 a 和 n, 求 a+aa+aaa+aa...a(n 个

a)之和。要求定义并调用函数 `fn(a,n)`，它的功能是返回 `aa...a`(`n` 个 `a`)。例如，`fn(3,2)`的返回值是 33。试编写相应程序。

解答：

```
#include <stdio.h>
long fn(long a, int n);
int main(void)
{
    int i, n;
    long a, sn;
    scanf("%ld%d", &a, &n);
    sn=0;
    for(i=1; i<=n; i++)
        sn=sn+fn(a, i);
    printf("sum=%ld\n", sn);
    return 0;
}
long fn(long a, int n)
{
    int i;
    long tn=0;
    for(i=1; i<=n; i++){
        tn=tn+a;
        a=a*10;
    }
    return tn;
}
```

3. 使用函数输出指定范围内的完数：输入两个正整数 `m` 和 `n`( $1 \leq m, n \leq 1000$ )，输出 `m ~ n` 之间的所有完数，完数就是因子和与它本身相等的数。要求定义并调用函数 `factorsum(number)`，它的功能是返回 `number` 的因子和。例如，`factorsum(12)`的返回值是 16 ( $1+2+3+4+6$ )。试编写相应程序。

解答：

```
#include <stdio.h>
int factorsum(int number);
int main(void)
{
    int i, m, n;
    scanf("%d%d", &m, &n);
    for (i=m; i<=n; i++)
        if(factorsum(i)==i) printf("%d\n", i);
    return 0;
}
int factorsum(int number)
{
    int i, sum;
    sum = 1;
    for(i=2; i<=number/2; i++)
```

```

        if(number%i==0) sum = sum + i;
    return sum;
}

```

4. 使用函数输出指定范围内的 Fibonacci 数：输入两个正整数  $m$  和  $n$  ( $1 \leq m, n \leq 10000$ )，输出  $m \sim n$  之间所有的 Fibonacci 数。Fibonacci 序列（第 1 项起）：1 1 2 3 5 8 13 21……。要求定义并调用函数 `fib(n)`，它的功能是返回第  $n$  项 Fibonacci 数。例如，`fib(7)` 的返回值是 13。试编写相应程序。

解答：

```

#include <stdio.h>
long fib(int n);
int main(void)
{
    int i, m, n;

    scanf("%d%d", &m, &n);
    i=1;
    while(fib(i)<=n){
        if(fib(i)>=m)
            printf("%ld ", fib(i));
        i++;
    }
    return 0;
}
long fib(int n)
{
    int i;
    long x1,x2,x;
    if(n==1 || n==2)
        return 1;
    else{
        x1=1;x2=1;
        for(i=3;i<=n;i++){
            x=x1+x2;
            x1=x2;
            x2=x;
        }
        return x;
    }
}

```

5. 使用函数验证哥德巴赫猜想：任何一个不小于 6 的偶数均可表示为两个奇素数之和。例如  $6=3+3$ ， $8=3+5$ ， $\dots$ ， $18=5+13$ 。将 6~100 之间的偶数都表示成两个素数之和，打印时一行打印 5 组。试编写相应程序。

解答：

```

#include "stdio.h"

```



```

#include "math.h"
int prime(int m);
int main(void)
{
    int count, i, number;

    count = 0;
    for(number = 6; number <= 100; number = number + 2){
        for(i = 3; i <= number/2; i = i + 2)
            if(prime(i) && prime(number - i)){
                printf("%d=%d+%d ", number, i, number - i);
                count++;
                if(count % 5 == 0) printf("\n");
                break;
            }
    }
    return 0;
}

int prime(int m)
{
    int k, i;
    if(m == 1) return 0;
    k = sqrt(m);
    for(i = 2; i <= k; i++)
        if(m % i == 0) return 0;
    return 1;
}

```

6. 使用函数输出一个整数的逆序数：输入一个整数，将它逆序输出。要求定义并调用函数 `reverse(number)`，它的功能是返回 `number` 的逆序数。例如，`reverse(12345)` 的返回值是 54321。试编写相应程序。

解答：

```

#include <stdio.h>
int reverse(int n);
int main(void)
{
    int x;
    scanf("%d", &x);
    printf("%d 的逆向是%d\n", x, reverse(x));
    return 0;
}

int reverse (int n)
{
    int m, res;

```

```

    if(n >= 0)
        m = n;
    else
        m = -n;
    res = 0;
    while(m != 0){
        res= res* 10 + m % 10;
        m = m / 10;
    }
    if(n >= 0)
        return res;
    else
        return -res;
}

```

7. 简单计算器：模拟简单运算器的工作，输入一个算式（没有空格），遇等号"="说明输入结束，输出结果。假设计算器只能进行加、减、乘、除运算，运算数和结果都是整数，4种运算符的优先级相同，按从左到右的顺序计算。例如，输入 1+2\*10-10/2=后，输出 10。试编写相应程序。

解答：

```

#include <stdio.h>
int main(void)
{
    char op;
    int operand1, operand2, res;

    printf("请输入一个算式，以=结束： ");
    scanf("%d", &operand1);
    op = getchar();
    while(op != '='){
        scanf("%d", &operand2);
        switch(op){
            case '+': res = operand1 + operand2; break;
            case '-': res = operand1 - operand2; break;
            case '*': res = operand1 * operand2; break;
            case '/': res = operand1 / operand2; break;
            default: res = 0;
        }
        operand1 = res;
        op = getchar();
    }
    printf("%d\n", res);
    return 0;
}

```

8. 单词首字母大写：输入一行字符，将每个单词的首字母改为大写后输出。所谓“单词”是指连续不含空格的字符串，各单词之间用空格分隔，空格数可以是多个。试编写相应程序。

解答：

```
#include <stdio.h>
int main(void)
{
    char c;
    int count, word;
    printf("请输入一行字符，以回车符结束：");
    c = getchar();
    word = 0;
    while(c != '\n'){
        if(c == ' ')
            word=0;
        else if(word == 0){
            word = 1;
            if(c>='a' && c<='z')    c=c-32;
        }
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

## 6.4 实验指导教材参考答案

### 一、调试示例

近似求 PI：根据下式求  $\pi$  的近似值，直到最后一项小于给定精度  $\epsilon$ 。

$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{2!}{3 \times 5} + \frac{3!}{3 \times 5 \times 7} + \frac{4!}{3 \times 5 \times 7 \times 9} + \frac{i!}{3 \times 5 \times \dots (2 \times i + 1)} + \dots$$

解答：略

### 二、基础编程题

(1) 英文字母替换加密（大小写转换+后移 1 位）。为了防止信息被别人轻易窃取，需要把电码明文通过加密方式变换成为密文。变换规则是：将明文中的所有英文字母替换为字母表中的后一个字母，同时将小写字母转换为大写字母，大写字母转换为小写字母。例如，字母 a->B、b->C、…、z->A、A->b、B->c、…、Z->a。输入一行字符，将其中的英文字母按照以上规则转换后输出，其他字符按原样输出。试编写相应程序。

解答：参见练习 6-3。

(2) 使用函数求特殊 a 串数列和：输入两个正整数 a 和 n，求  $a+aa+aaa+aa\cdots a$  (n 个 a) 之和。要求定义并调用函数  $fn(a, n)$ ，它的功能是返回  $aa\cdots a$  (n 个 a)。试编写相应程序

解答：参见习题程序设计第 2 题。

(3) 单词首字母大写：输入一行字符，将每个单词的首字母改为大写后输出。所谓“单词”是指连续不含空格的字符串，各单词之间用空格分隔，空格数可以是多个。试编写相应程序。  
解答：参见习题程序设计第 8 题。

(4) 简单计算器：编写程序，模拟简单运算器的工作。输入一个算式（没有空格且至少有一个操作数），遇等号“=”说明输入结束，输出结果；如果除法分母为 0 或有非法运算符，则输出错误信息“ERROR”。假设计算器只能进行加、减、乘、除运算，运算数和结果都是整数，四种运算符的优先级相同，按从左到右的顺序计算。  
解答：参见习题程序设计第 7 题。

(5) 使用函数验证哥德巴赫猜想：任何一个不小于 6 的偶数均可表示为两个奇素数之和。输入两个正整数  $m$  和  $n$  ( $0 < m \leq n \leq 100$ )，将  $m$  和  $n$  之间的偶数表示成两个素数之和，打印时一行打印 5 组。要求定义并调用函数 `prime(m)` 判断  $m$  是否为素数，当  $m$  为素数时返回 1，否则返回 0。素数就是只能被 1 和自身整除的正整数，1 不是素数，2 是素数。  
解答：参见习题程序设计第 5 题。

### 三、改错题

使用函数输出一个整数的逆序数：输入一个整数  $n$ ，输出其逆序数。要求定义并调用函数 `reverse(n)`，它的功能是返回  $n$  的逆序数。例如 `reverse(123)` 的返回值是 321。

错误行号：	<u>2</u>	正确语句：	<u><code>int reverse( int n );</code></u>
错误行号：	<u>20</u>	正确语句：	<u><code>digit = n % 10;</code></u>
错误行号：	<u>21</u>	正确语句：	<u><code>n = n / 10;</code></u>
错误行号：	<u>23</u>	正确语句：	<u><code>}while( n != 0 );</code></u>

### 四、拓展编程题

(1) 使用函数输出指定范围内的完数：输入两个正整数  $m$  和  $n$  ( $0 < m \leq n \leq 10000$ )，找出  $m$  和  $n$  之间的所有完数，并输出每个完数的因子累加形式的分解式，其中完数和因子均按递增顺序给出；若  $m$  和  $n$  之间没有完数，则输出“No perfect number”。所谓完数就是该数恰好等于除自身外的因子之和。例如， $6 = 1 + 2 + 3$ ，其中 1、2、3 为 6 的因子。要求定义并调用函数 `factorsum(number)`，它的功能是返回 `number` 的因子和；定义和调用函数 `print_pn(m, n)` 输出给定范围  $[m, n]$  内每个完数的因子累加形式的分解式，其中完数和因子均按递增顺序给出。

解答：参见习题程序设计第 3 题。

(2) 使用函数输出指定范围内的 Fibonacci 数：输入两个正整数  $m$  和  $n$  ( $0 < m \leq n \leq 10000$ )，输出  $m$  和  $n$  之间所有的 Fibonacci 数；如果给定区间内没有 Fibonacci 数，则输出“No Fibonacci number”。所谓 Fibonacci 数列就是满足任一项数字是前两项的和（最开始两项均定义为 1）的数列，从第 1 项起为 1、1、2、3、5、8、13、21……。要求定义并调用函数 `fib(n)`，它的功能是返回第  $n$  项 Fibonacci 数。例如，`fib(7)` 的返回值是 13；定义和调用函数 `print_fn(m, n)` 输出给定范围  $[m, n]$  内的所有 Fibonacci 数，相邻数字间有一个空格，行末不得有多余空格。

解答：参见习题程序设计第 4 题。

(3) 统计单词的长度：输入一行字符，统计每个单词的长度。所谓“单词”是指连续不含空格的字符串，各单词之间用空格分隔，空格数可以是多个。试编写相应程序。

解答：

```
#include <stdio.h>
int main(void)
{
    char c;
    int sum, word;
    printf("Enter characters: ");
    c = getchar();
    word = 0;
    sum=0;
    while(c != '\n'){
        if(c == ' '){
            word=0;
            if(sum>0) { printf("%d ", sum); sum=0;}
        }
        else if(word == 0){
            word = 1;
            sum++;
        }
        else
            sum++;
        c = getchar();
    }
    if(word==1) printf("%d", sum);
    return 0;
}
```

(4) 使用函数输出水仙花数：输入两个正整数  $m$  和  $n$  ( $100 \leq m \leq n \leq 10000$ )，输出区间  $[m, n]$  内所有的水仙花数。所谓水仙花数是指一个  $n$  位正整数 ( $n \geq 3$ )，它的各位数字的  $n$  次幂之和等于它本身。例如 153 的各位数字的立方和是  $1^3+5^3+3^3=153$ 。要求定义并调用函数 `narcissistic (number)` 判断 `number` 是否为水仙花数，是则返回 1，否则返回 0；定义和调用函数 `print_n(m, n)` 按从小到大的顺序打印区间  $[m, n]$  内所有的水仙花数。

解答：

```
#include <stdio.h>
int narcissistic (int number);
void print_n(int m, int n);
int main(void)
{
    int i, m, n;
    printf("Enter m, n: ");
    scanf("%d%d", &m, &n);
```

```

        print_n(m, n);
        return 0;
    }
int narcissistic (int number)
{
    int digit, sum, temp;
    temp = number;
    sum = 0;
    while(temp != 0){
        digit = temp % 10;
        temp = temp / 10;
        sum = sum + digit * digit * digit;
    }
    return sum == number;
}
void print_n(int m, int n)
{
    int i;
    for(i = m; i <= n; i++)
        if(narcissistic (i) != 0)
            printf("%d\n", i);
}

```