

## 第5章 函数

### 5.1 教学要点

在学生具备初步使用函数的基础上,本章将系统介绍函数的定义和函数调用。学习如何针对具体问题,确定需要使用函数的功能要求,再将功能用函数程序实现,并考虑如何调用定义好的函数,实现主调函数与被调函数的连接。本章后半部分将讨论函数与变量间的关系,局部变量、全局变量和静态变量,不同形式的变量在函数中起的作用将不同。本章难点是:函数功能模块的确定;函数参数确定,函数参数与函数内局部变量的角色分配;函数参数的传递是值传递,形参的改变不会影响实参的变化。

5.1 节以计算圆柱体体积为例,通过定义体积计算功能的函数和主函数调用的例子,引出函数定义的一般形式:函数首部加函数体,且在函数结尾处通过 `return` 语句返回结果。本节要重点掌握两点:

(1) 形参、实参及参数传递。形参只能用变量表示,在函数中起作用;实参可以是变量、常量或表达式,在主调函数中。实参与形参要个数一致,类型一致。参数传递时,实参把变量、常量的值或者表达式结果值传递给形参。

(2) 函数定义首部与声明。二者形式相似,函数声明的目的主要是说明函数的类型和参数的情况,以保证程序编译时对函数调用是否正确做出判断,声明位置通常在主函数前面。而函数定义首部位于函数定义的第一行,与函数体连续书写组成函数定义。

接下来,通过 3 个例子学习使用函数编写程序的基本方法。第 1 个例子定义的是判断函数,用整数 1 代表结果为真,0 代表假。第 2 个例子更像计算数学意义上的函数,有  $\pi$  的计算结果。第 3 个例子,介绍了判断素数的函数,该函数的写法较为经典,要求学生充分理解算法,并能灵活掌握该函数的其他实现形式。

5.2 节以数字金字塔为例,介绍函数的另一种形式,即不返回结果的函数。不返回结果的函数在定义、调用、参数传递、函数声明上,思路完全与以前相同,只是函数类型变为 `void`。重点介绍它适用的场合。

最后,强调函数的优点,即结构化程序设计的思路,包括按自顶向下的方法对问题进行分析、模块化设计和结构化编码 3 个步骤,并引入一些生活中的实例展开说明。

5.3 节以复数运算为例,介绍变量与函数的关系,主要掌握局部变量、全局变量和静态变量三种形式。需要分清三种变量的作用与定义的位置,区分局部变量与全局变量的异同点,局部变量与静态局部变量的异同点。最后引入两个实例,财务现金记账和求  $n$  的阶乘,分别介绍了全局变量和静态局部变量的典型应用。要求学生观察如何通过全局变量返回多个计算结果,以及如何利用静态局部变量的生存周期保存函数多次调用的结果。

讲授学时: 6 学时, 实验学时: 4 学时。

本章的知识能力结构图见图 5.1。

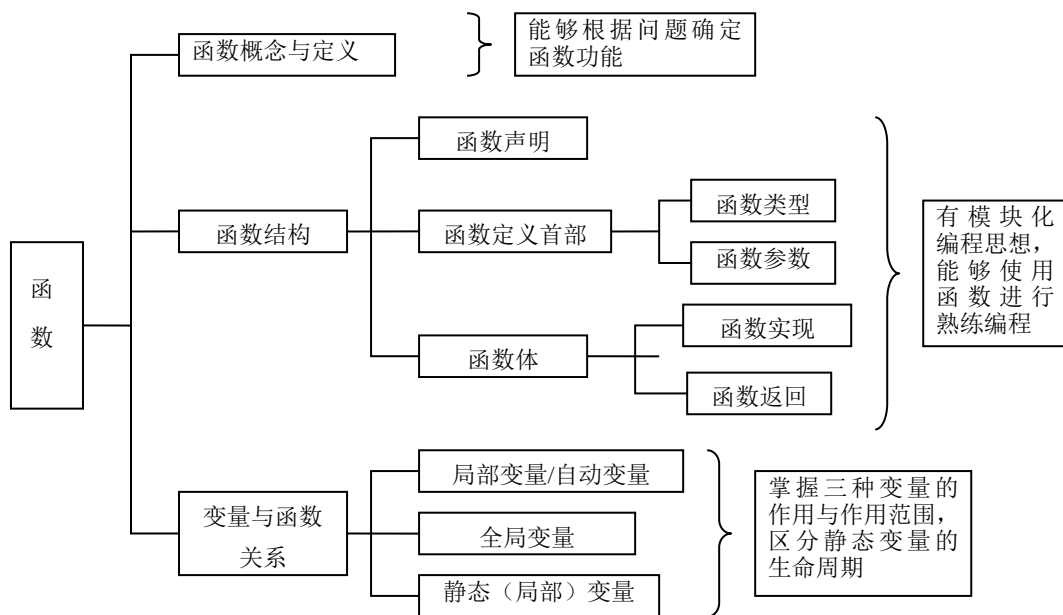








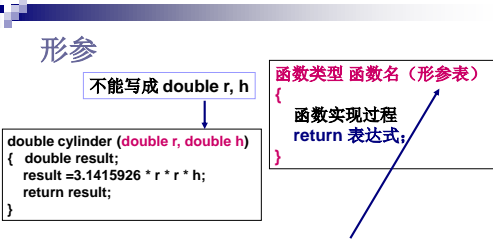
图 5.1 知识能力结构图

## 5.2 讲稿

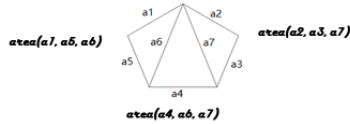
|   |  |            |
|---|--|------------|
| 1 |  <p><b>Chap 5 函数</b></p> <p>5.1 计算圆柱体积</p> <p>5.2 数字金字塔</p> <p>5.3 复数运算</p>   | 本章分 3 节。   |
| 2 |  <p><b>本章要点</b></p> <ul style="list-style-type: none"> <li>■ 函数的作用？如何确定函数功能？</li> <li>■ 怎样定义函数？如何调用函数？定义函数与声明函数有何区别？</li> <li>■ 什么是函数的参数？怎样确定函数的参数？</li> <li>■ 在函数调用时，参数是如何传递数据的？</li> <li>■ 变量与函数有什么关系？如何使用局部变量和全局变量？</li> <li>■ 什么是静态变量？</li> </ul> | 提出本章的学习要点。 |

|   |  |   |
|---|--|---|
| 3 |  <h2>5.1 计算圆柱体积</h2> <ul style="list-style-type: none"> <li>5.1.1 程序解析</li> <li>5.1.2 函数的定义</li> <li>5.1.3 函数的调用</li> <li>5.1.4 函数程序设计</li> </ul>   | 本章共分为 4 小节。                                       |
| 4 |  <h3>5.1.1 程序解析—计算圆柱体积</h3> <p><b>例5-1</b> 输入圆柱体的高和半径，求圆柱体积，<math>\text{volume}=\pi*r^2*h</math>。</p> <p>要求定义和调用函数 <b>cylinder(r, h)</b> 计算圆柱体的体积。</p>  | 以简单例子入门，整体说明函数结构。<br>可以先请学生回顾第二章中学习过的函数结构。        |
| 5 |  <h4>例5-1源程序</h4> <pre> /* 计算圆柱体积 */ #include &lt;stdio.h&gt; double cylinder(double r, double h); /* 函数声明 */ int main( void ) {     double height, radius, volume;      printf("Enter radius and height: ");     scanf("%lf%lf", &amp;radius, &amp;height);     /* 调用函数，返回值赋给volume */     volume = cylinder(radius, height);     printf("Volume = %.3f\n", volume);      return 0; } </pre> | 主函数结构，重点关注函数声明与函数调用形式                             |
| 6 |  <h4>例5-1源程序</h4> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;">         Enter radius and height: 3.0 10<br/>         Volume = 282.743     </div> <pre> /* 定义求圆柱体积的函数 */ double cylinder(double r, double h) {     double result;      result = 3.1415926 * r * r * h; /* 计算体积 */      return result; /* 返回结果 */ } </pre>  | <b>Cylinder</b> 函数的定义，重点关注函数首部与函数声明的区别、函数结果返回的形式。 |

|    |   |   |
|----|---|---|
| 7  | <div> <div>例5-1源程序</div> <pre> #include &lt;stdio.h&gt; double cylinder (double r, int main( void ) {     double height, radius, volume;      printf ("Enter radius and height: ");     scanf ("%lf%lf", &amp;radius, &amp;height);     volume = cylinder (radius, height);     printf ("Volume = %.3fn", volume);     return 0; }  double cylinder (double r, double h) {     double result;     result = 3.1415926 * r * r * h;     return result; } </pre> <div>Enter radius and height: 3.0 10<br/>Volume = 282.743</div> <div>问题:<br/>函数是如何运行的?</div> </div> | <p>重点介绍函数的调用过程:</p> <ul style="list-style-type: none"> <li>main 函数→cylinder 函数</li> <li>实参→形参</li> <li>return 的结果返回。</li> </ul> <p>再对比一下函数首部与函数声明。</p>  |
| 8  | <div> <div>5.1.2 函数的定义</div> <ul style="list-style-type: none"> <li>函数是指完成一个特定工作的独立程序模块。 <ul style="list-style-type: none"> <li>库函数: 由C语言系统提供定义<br/>如scanf()、printf()等函数</li> <li>自定义函数: 需要用户自己定义<br/>如计算圆柱体体积函数cylinder()</li> </ul> </li> <li>main ()也是一个函数, C程序由一个main ()或多个函数构成。</li> <li>程序中一旦调用了某个函数, 该函数就会完成特定的计算, 然后返回到调用它的地方。 <ul style="list-style-type: none"> <li>函数经过运算, 得到一个明确的运算结果, 并需要回送该结果。例如, 函数cylinder ()返回圆柱的体积。</li> </ul> </li> </ul> </div>   | <p>概括一下到目前用过的函数:</p> <p>main---主函数</p> <p>printf、scanf---系统函数</p> <p>cylinder---自定义函数</p> <p>问题: 函数到底有什么作用?</p> <p>根据作用 (是否有运算结果值) 分成2类, 下面根据这两类分别讨论函数定义</p>                                  |
| 9  | <div> <div>5.1.2 函数定义</div> <pre> double cylinder (double r, double h) {     double result;     result = 3.1415926 * r * r * h;     return result; } </pre> <div>函数返回值的类型</div> <div>函数类型 函数名 (形参表)</div> <div>没有分号</div> <div>/* 函数首部 */</div> <div>/* 函数体 */</div> <div>函数实现过程</div> <div>return 表达式;</div> <div>只能返回一个值</div> <div>把函数运算的结果回送给主函数</div> </div>   | <p>先介绍有结果值的函数, 主要体现在:</p> <ol style="list-style-type: none"> <li>函数有确切类型---函数首部</li> <li>return 语句</li> </ol> <p>函数类型与 return 返回的表达式类型应一致。</p>  |
| 10 | <div> <div>分析函数的定义</div> <div> <div>函数类型</div> <div>函数名</div> <div>形参表</div> </div> <pre> double cylinder (double r, double h) {     double result;      result = 3.1415926 * r * r * h;      return result; } </pre> <div>/* 函数首部 */</div> <div>/* 函数体, 写在一大括号内 */</div> <div>/* 计算圆柱体积 */</div> <div>/* 返回运算结果 */</div> <div>与函数类型一致</div> </div>   | <p>仍关注函数首部---形参表:</p> <p>建议和学生讨论一下什么是形参 (函数计算必须提供的已知条件), 可以举一个例子 (比如 <math>1! + 2! + \dots + n!</math>), 形参是什么? 再问: 计算过程中要用到的循环变量 i、求和值 sum 是否也是形参?</p> <p><b>提醒:</b> 形参表定义时每个参数名都需要类型指定。</p> |

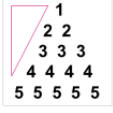
|    |   |  |
|----|---|--|
| 11 |  <p>形参</p> <p>不能写成 double r, h</p> <pre>double cylinder (double r, double h) {     double result;     result = 3.1415926 * r * r * h;     return result; }</pre> <p>函数类型 函数名 (形参表)</p> <p>函数实现过程<br/>return 表达式;</p> <p>类型1 参数1, 类型2 参数2, ....., 类型n 参数n<br/>参数之间用逗号分隔, 每个参数前面的类型都必须分别写明</p> | <p>小结一下函数结构组成:</p> <p>函数首部 (函数类型、函数名、形参表) ----无分号, 它不是一个完整的语句。</p> <p>函数体 (函数实现、函数返回---注意类型)。</p>  |
| 12 | <h3>5.1.3 函数的调用</h3> <ul style="list-style-type: none"> <li>定义一个函数后, 就可以通过程序来调用这个函数。</li> <li>调用标准库函数时, 在程序的最前面用 <b>#include</b> 命令包含相应的头文件。</li> <li>调用自定义函数时, 程序中必须有与调用函数相对应的函数定义。</li> </ul>   | <p>本小节主线: 函数调用 (函数参数传递) + 返回</p> <p>什么是函数调用?</p> <p>——可以从系统函数 <code>scanf</code> 和 <code>printf</code> 解释; 任何函数调用都要有来源:</p> <ul style="list-style-type: none"> <li><code>include</code> (系统函数), 调用标准库函数<br/>所需包含的头文件可参阅附录 A</li> <li>自定义函数定义</li> </ul>  |
| 13 | <h3>1. 函数调用的形式</h3> <ul style="list-style-type: none"> <li>函数调用的一般形式为:<br/><b>函数名 (实际参数表)</b></li> <li>对于实现计算功能的函数, 函数调用通常出现在两种情况: <ul style="list-style-type: none"> <li>赋值语句<br/><code>volume = cylinder (radius, height);</code></li> <li>输出函数的实参<br/><code>printf ("%f", cylinder (radius, height));</code></li> </ul> </li> </ul>  | <p>函数调用形式:</p> <ul style="list-style-type: none"> <li>有返回值 <ul style="list-style-type: none"> <li>表达式调用形式</li> <li>函数参数形式</li> </ul> </li> <li>无返回值 <ul style="list-style-type: none"> <li>调用语句</li> </ul> </li> </ul> <ol style="list-style-type: none"> <li>在函数调用时, 实参前不要加数据类型;</li> <li>函数调用可以直接作为输出参数</li> </ol> |
| 14 | <h3>2. 函数调用的过程</h3> <ul style="list-style-type: none"> <li>计算机在执行程序时, 从主函数 <code>main</code> 开始执行, 如果遇到某个函数调用, 主函数被暂停执行, 转而执行相应的函数, 该函数执行完后, 将返回主函数, 然后再从原先暂停的位置继续执行。</li> <li>函数遇 <code>return</code> 返回主函数</li> </ul>   | <p>结合函数调用的单步调试工具, 用实例演示程序流程</p>  |

|    |  |  |
|----|--|--|
| 15 | <p><b>分析函数调用的过程</b></p> <pre> #include &lt;stdio.h&gt; double cylinder (double r, double h); int main( void ) {     double height, radius, volume;     printf ("Enter radius and height: ");     scanf ("%lf%lf", &amp;radius, &amp;height);     volume = cylinder (radius, height);     printf ("Volume = %.3f\n", volume);     return 0; } double cylinder (double r, double h) {     double result;     result = 3.1415926 * r * r * h;     return result; } </pre> <p>调用函数<br/>实参→形参<br/>执行函数中的语句<br/>返回调用它的地方</p> | 穿插介绍单步调试工具的使用。   |
| 16 | <p><b>3. 参数传递</b></p> <ul style="list-style-type: none"> <li>函数定义时的参数被称为<b>形式参数</b>（简称<b>形参</b>）<br/>double cylinder (double r, double h);</li> <li>函数调用时的参数被称为<b>实际参数</b>（简称<b>实参</b>）<br/>volume = cylinder (radius, height);</li> <li>参数传递：<b>实参→形参</b> <b>单向传递</b> <ul style="list-style-type: none"> <li>在参数传递过程中，实参把值复制给形参。</li> <li>形参和实参<b>一一对应</b>：数量一致，类型一致，顺序一致</li> <li><b>形参</b>：变量，用于接受实参传递过来的值</li> <li><b>实参</b>：常量、变量或表达式</li> </ul> </li> </ul>  | <p>参数传递是函数内容的难点：</p> <ol style="list-style-type: none"> <li>1、分清形参与实参的概念</li> <li>2、形参的形式——只能是变量，在函数首部定义</li> <li>3、实参的形式可以是：变量、常量或表达式</li> <li>4、参数传递：实参把结果<b>值</b>传给形参</li> <li>5、形参改变不影响实参变量（单向传递）</li> </ol> |
| 17 | <p><b>4. 函数结果返回</b></p> <ul style="list-style-type: none"> <li>完成确定的运算，将运算结果返回给主调函数。</li> <li>函数结果返回的形式： <ul style="list-style-type: none"> <li>return 表达式；</li> <li>return (表达式)；</li> </ul> </li> </ul>  | <p>本例子的函数调用是针对有返回值函数——采用赋值表达式形式；sss</p> <p>对上一页讲到的执行流程在本例子中给予具体解释，加深学生理解：当调用 cylinder 时，main 函数并未结束，只是暂停，等 cylinder 结束后，main 再继续</p> <p>提醒学生：函数声明别忘</p>  |
| 18 | <p><b>5. 函数原型声明</b></p> <p>只写函数定义中的第1行（函数首部），并以分号结束。</p> <p>函数类型 函数名(参数表)；<br/>double cylinder (double r, double h);<br/>void pyramid (int n);</p> <ul style="list-style-type: none"> <li>函数必须先定义后调用，将主调函数放在被调函数的后面，就像变量先定义后使用一样。</li> <li>如果自定义函数在主调函数的后面，就需要在函数调用前，加上函数原型声明。</li> <li>函数声明：说明函数的类型和参数的情况，以保证程序编译时能判断对该函数的调用是否正确。</li> </ul>  | <p>强调两点：</p> <ol style="list-style-type: none"> <li>1、函数原型声明的位置与形式（简单记忆：即函数首部加分号）</li> <li>2、函数原型声明的作用（同变量声明）</li> </ol>   |

|    |   |  |
|----|---|--|
| 19 | <p><b>§ 5.1.3 函数调用</b></p> <p>要调用函数，必须先要声明！</p> <p>在执行函数调用时，实参把值计算出来，拷贝给相应位置的形参；函数执行完后，通过return ()，可返回一个结果值。</p> <p>实参与形参个数相同、类型一致</p> <p>有多个实参时后面的先计算</p> <p>形参的改变，不影响实参变量的值</p> <p>只能返回一个结果，类型与函数定义时一致</p>  | <p>强调实参与形参之间的“三个一致”原则：</p> <ol style="list-style-type: none"> <li>1、个数一致</li> <li>2、类型一致</li> <li>3、顺序一致</li> </ol> <p>可借鉴例子演示，当一致原则不符合时，程序会出现什么结果，加深印象。</p> |
| 20 | <p><b>5.1.4 函数程序设计</b></p> <p><b>例5-2 计算五边形的面积</b></p> <p>将一个五边形分割成三个三角形，输入这些三角形的7条边长，计算该五边形的面积。</p> <p>要求定义和调用函数area(x,y,z)计算边长为x、y、z的三角形面积。</p>    | <p>本节通过若干例子学习函数程序设计方法。</p> <p>例 5-2，定义一个计算三角形面积的函数，调用 3 次，计算五边形面积。</p>   |
| 21 | <p><b>例5-2 源程序段</b></p> <pre> Please input 7 side lengths in the order a1 to a7: 2.5 2.5 2.5 2.5 2.5 3.6 3.6 The area of the Pentagon is 10.47  double a1, a2, a3, a4, a5, a6, a7, s; double area(double x, double y, double z); printf("Please input 7 side lengths in the order a1 to a7:\n"); scanf("%lf%lf%lf%lf%lf%lf%lf", &amp;a1, &amp;a2, &amp;a3, &amp;a4, &amp;a5, &amp;a6, &amp;a7); s = area(a1, a5, a6) + area(a4, a6, a7) + area(a2, a3, a7); /* 调用3次area */ printf("The area of the Pentagon is %.2f\n", s);  /* 使用海伦-秦九韶公式计算三角形面积的函数 */ double area(double x, double y, double z) /* 函数首部 */ {     double p = (x + y + z) / 2;     return sqrt(p * (p - x) * (p - y) * (p - z)); } </pre> | <p>分析：</p> <ol style="list-style-type: none"> <li>1、明确函数形参与函数内局部变量。</li> <li>2、函数调用形式（赋值表达式）</li> <li>3、函数结果返回形式：表达式</li> <li>4、函数声明构成</li> </ol>            |
| 22 | <p><b>例5-3 使用函数判断完全平方数</b></p> <p>定义函数IsSquare(n)：当n为完全平方数时返回1，否则返回0。</p> $1+3+5+7+\dots+(2 \times m-1) = m^2 = n$ <pre> /* 判断完全平方数的函数 */ int IsSquare(int n) /* 函数首部 */ {     int i;      for (i = 1; n &gt; 0; i = i + 2) {         n = n - i;     }     if (n == 0) {         return 1; /* 是完全平方数返回1 */     } else {         return 0; /* 不是完全平方数返回0 */     } } </pre>   | <p>例 5-3：用于判断结果的函数示例，是返回 1，不是返回 0</p>  |

|    |  |  |
|----|--|--|
| 23 | <div data-bbox="300 203 790 593" data-label="Complex-Block"> <div> <div>例5-4 使用函数求最大公约数</div> <div> <div>定义函数gcd(int m, int n): 计算m和n的最大公约数</div> <div>           辗转相除法（欧几里得算法）           <ol style="list-style-type: none"> <li>(1)r = m % n</li> <li>(2)若r为0，则返回n的值作为结果并结束；否则转第(3)步；</li> <li>(3)m = n, n = r,返回第(1)步。</li> </ol> </div> </div> <div> <pre>int gcd(int m, int n) {     int r, temp;     if(m &lt; n){         temp = m; m = n; n = temp;     }     r = m % n;     while(r != 0){         m = n;         n = r;         r = m % n;     }     return n; }</pre> </div> </div> </div> | <div data-bbox="842 190 1358 271" data-label="Text"> <p>例 5-4：求最大公约数的函数，使用辗转相除法</p> </div> |
|----|--|--|



|    |   |   |
|----|---|---|
| 27 | <p><b>例5-6</b></p> <pre> /* 输出数字金字塔 */ #include &lt;stdio.h&gt; void pyramid(int n); int main (void) {     pyramid(5);     return 0; }  void pyramid (int n) {     int i, j;     for (i = 1; i &lt;= n; i++){         for (j = 1; j &lt;= n-i; j++){             printf(" ");         }         for (j = 1; j &lt;= i; j++){             printf("%d ", i);             putchar(' ');         }         putchar('\n');     } } </pre> <p>for (i = 1; i &lt;= n; i++) {<br/>for (j = 1; j &lt;= n-i; j++)<br/>printf(" ");<br/>一行中的空格处理;<br/>一行中的数字显示</p> <p>for (i = 1; i &lt;= n; i++) {<br/>一行中的空格处理;<br/>一行中的数字显示</p>  | <p>向学生指出本例子主函数的简洁性---使用函数的优点;</p> <p>二维输出格式一般需要二重循环,但从思路可以先讲一重(外循环),进而对于一行的实现,又需要一层循环,从而构成了二重循环。</p>  |
| 28 | <p><b>5.2.2 不返回运算结果的函数定义</b></p> <p><b>表示不返回结果</b></p> <pre> void 函数名 (参数表) /* 函数首部 */ {     函数实现过程     return; /* 可以省略return */ } </pre> <p>这类函数通常用于屏幕输出等</p> <p><b>不能省略, 否则<br/>函数类型被默认定义为int</b></p>   | <p>起计算作用的函数结果返回,学生比较容易理解,但要注意类型;</p> <p>对于无结果返回的函数,其作用是在函数过程中完成(屏幕或变量的改变),不使用 return 时,以函数体右大括号返回函数调用过程: 执行流程改变需向学生交待清楚。</p> <p>对于无返回值的函数,可以不需要 return 语句,到函数结束时的最后一个大括号也能返回。</p> |
| 29 | <p><b>5.2.2 不返回运算结果的函数定义</b></p> <ul style="list-style-type: none"> <li>由于函数没有返回结果,函数调用不可能出现在表达式中,通常以独立的调用语句方式,如pyramid(5);</li> <li>不返回结果的函数,在定义、调用、参数传递、函数声明上,思路完全与以前相同,只是函数类型变为void。</li> <li>它适用把一些确定的、相对独立的程序功能包装成函数。 <ul style="list-style-type: none"> <li>主函数通过调用不同的函数,体现算法步骤</li> <li>各步骤的实现由相应函数完成</li> <li>简化主函数结构,以体现结构化程序设计思想。</li> </ul> </li> </ul>  | <p>解单介绍不返回结果函数的适用及常用场合,通常不是为了计算得到某个结果,而是要用程序产生某些效果,如打印屏幕等。</p> <p>需强调的是: 不返回结果的函数可以没有 return 语句,但并不意味着不能返回,遇到函数体最后的大括号可以自动返回主调函数,可通过单步演示加深印象。</p>                                 |
| 30 | <p><b>5.2.3 结构化程序设计思想</b></p> <ul style="list-style-type: none"> <li><b>结构化程序设计(Structured Programming)</b> <ul style="list-style-type: none"> <li>程序设计技术</li> <li>C语言是结构化程序设计语言</li> </ul> </li> <li><b>强调程序设计的风格和程序结构的规范化,提倡清晰的结构</b> <ul style="list-style-type: none"> <li>基本思路是将一个复杂问题的求解过程划分为若干阶段,每个阶段要处理的问题都容易被理解和处理。</li> <li>按<b>自顶向下</b>的方法对问题进行分析、模块化设计和结构化编码等3个步骤。</li> </ul> </li> </ul>   | <p>可简单结合“软件工程”的概念介绍模块化程序设计思想,以生活中的实际例子举例说明。</p>   |

|    |   |   |
|----|---|---|
| 31 | <div data-bbox="300 203 347 241" data-label="Image"></div> <h3 data-bbox="341 244 662 280">1. 自顶向下的分析方法</h3> <ul style="list-style-type: none"> <li>■ 把大的复杂的<b>问题分解</b>成小问题后再解决 <ul style="list-style-type: none"> <li>□ 面对一个复杂的问题，首先进行上层（整体）的分析，按组织或功能将问题分解成<b>子问题</b></li> <li>□ 如果子问题仍然十分复杂，再做进一步分解，直到处理对象<b>相对简单</b>，容易处理为止。</li> <li>□ 当所有的子问题都得到了解决，整个问题也就解决了。</li> </ul> </li> <li>■ 每一次分解都是对上一层的问题进行细化和逐步求精，最终形成一种类似<b>树形的层次结构</b>，来描述分析的结果。</li> </ul>                           | <p>可结合“瀑布式”概念简单介绍。</p>                  |
| 32 | <div data-bbox="300 618 347 656" data-label="Image"></div> <h3 data-bbox="331 669 735 703">学生成绩统计程序的层次结构图</h3> <div data-bbox="331 725 798 891" data-label="Diagram"> <pre> graph TD     A[学生成绩统计程序] --&gt; B[成绩输入]     A --&gt; C[数据计算]     A --&gt; D[数据查找]     A --&gt; E[输出成绩]     C --&gt; F[计算学生平均分]     C --&gt; G[计算课程平均分] </pre> </div> <div data-bbox="469 925 633 954" data-label="Text"> <p>模块用函数实现</p> </div>  | <p>要求学生掌握层次结构图的画法，体现在实验报告中。</p>         |
| 33 | <div data-bbox="300 1032 347 1070" data-label="Image"></div> <h3 data-bbox="349 1072 547 1108">2. 模块化设计</h3> <ul style="list-style-type: none"> <li>■ 将模块组织成良好的层次系统 <ul style="list-style-type: none"> <li>□ 顶层模块调用其下层模块以实现程序的完整功能</li> <li>□ 每个下层模块再调用更下层的模块，从而完成程序的一个子功能，</li> <li>□ 最下层的模块完成最具体的功能。</li> </ul> </li> <li>■ 遵循模块独立性的原则，即模块之间的联系应尽量简单。 <ul style="list-style-type: none"> <li>□ 模块用函数实现。</li> <li>□ 一个模块只完成一个指定的功能。</li> <li>□ 模块之间只通过带参数的函数进行调用。</li> </ul> </li> </ul> | <p>模块化设计有利于多人合作的大型软件开发，强调团队精神，共享意识。</p> |
| 34 | <div data-bbox="300 1447 347 1485" data-label="Image"></div> <h3 data-bbox="346 1487 670 1523">3. 结构化编码主要原则</h3> <ul style="list-style-type: none"> <li>■ 经模块化设计后，每一个模块都可以独立编码。编程时应选用顺序、选择和循环三种控制结构</li> <li>■ 对变量、函数、常量等命名时，要见名知意，有助于对变量含义或函数功能的理解。</li> <li>■ 在程序中增加必要的注释，增加程序的可读性。</li> <li>■ 要有良好的程序视觉组织，利用缩进格式</li> <li>■ 程序要清晰易懂，语句构造要简单直接</li> <li>■ 程序有良好的交互性，输入有提示，输出有说明</li> </ul>  | <p>强调代码的规范性、可读性、美观程度等，尤其要重视注释的作用。</p>   |

|    |  |  |
|----|--|--|
| 35 | <p><b>5.3 复数运算</b></p> <p><b>5.3.1 程序解析</b></p> <p><b>5.3.2 局部变量和全局变量</b></p> <p><b>5.3.3 变量生命周期和静态局部变量</b></p>  | <p>本节新的内容：全局变量与静态变量，它们都是与函数相关的。</p>  |
| 36 | <p>例5-7 分别输入2个复数的实部与虚部，用函数实现计算2个复数之和与积。</p> <p>■ 分析</p> <ul style="list-style-type: none"> <li>□ 若2个复数分别为：<br/> <math>c1 = x1 + y1i</math>, <math>c2 = x2 + y2i</math>,</li> <li>□ 则：<br/> <math>c1 + c2 = (x1 + x2) + (y1 + y2)i</math><br/> <math>c1 * c2 = (x1 * x2 - y1 * y2) + (x1 * y2 + x2 * y1)i</math></li> </ul>   | <p>该例的特殊之处在于，需要返回两个计算结果，而通过常规函数调用是无法实现的，需要考虑引入全局变量。</p>  |
| 37 | <p>运行结果</p> <pre>#include &lt;stdio.h&gt; double Enter 1st complex number(real and imaginary):1 1 double Enter 2nd complex number(real and imaginary):-2 3 /* 函数 */ void complex_add(double real1, double imag1, double real2, double imag2, double *result_real, double *result_imag) void complex_prod(double real1, double imag1, double real2, double imag2, double *result_real, double *result_imag) int main(void) {     double imag1, imag2, real1, real2; /* 两个复数的实、虚部变量 */     printf("Enter 1st complex number(real and imaginary):");     scanf("%lf%lf", &amp;real1, &amp;imag1); /* 输入第一个复数 */     printf("Enter 2nd complex number(real and imaginary):");     scanf("%lf%lf", &amp;real2, &amp;imag2); /* 输入第二个复数 */     complex_add(real1, imag1, real2, imag2); /* 求复数之和 */     printf("addition of complex is %f+%fi\n", result_real, result_imag);     complex_prod(real1, imag1, real2, imag2); /* 求复数之积 */     printf("product of complex is %f+%fi\n", result_real, result_imag);     return 0; }</pre> | <p>将复数之积的实部与虚部定义为全局变量，扩大其作用范围。</p>   |
| 38 | <pre>void complex_add (double real1, double imag1, double real2, double imag2) {     result_real = real1 + real2;     result_imag = imag1 + imag2; }  void complex_prod (double real1, double imag1, double real2, double imag2) {     result_real = real1 * real2 - imag1 * imag2;     result_imag = real1 * imag2 + real2 * imag1; }</pre>   | <p>让学生观察，result_real 和 result_imag 两个变量在 complex_add() 和 complex_prod() 中是否有定义？体会全局变量与局部变量定义位置的不同。</p> |

|    |   |   |
|----|---|---|
| 39 | <h3>5.3.2 局部变量和全局变量</h3> <ul style="list-style-type: none"> <li>■ 局部变量 <ul style="list-style-type: none"> <li>□ 在函数内定义的变量（包括形参）<br/>作用范围：本函数内部</li> <li>□ 定义在复合语句内的变量<br/>作用范围：复合语句内部</li> </ul> </li> <li>■ 全局变量 <p>在函数以外定义的变量，不从属于任一函数。</p> <p>作用范围：从定义处到源文件结束（包括各函数）</p> </li> </ul>   | <p>前面使用的变量均为局部变量——在函数内使用，这里再增加一种在复合语句内使用的局部变量。</p> <p>全局变量是相对于局部变量而言的——不属于任意函数的，在函数外定义，包括主函数。</p> |
| 40 | <p>在复合语句中定义局部变量。</p> <pre>#include &lt;stdio.h&gt; int main(void) {     int a;     a = 1;     {         int b = 2;         b = a + b;         a = a + b;     }     printf("%d", a);     return 0; }</pre> <p>输出：<br/>4</p> <p>/* 复合语句开始 */<br/>b:小范围内的临时变量<br/>/* 复合语句结束 */</p> <p>改成b会如何?</p>  | <p>学习复合语句内使用的局部变量的使用，注意变量 b 的有效范围（复合语句内）</p>  |
| 41 | <pre>#include "stdio.h" int x; /* 定义全局变量x */ int f() {     int x = 4; /* x为局部变量 */     return x; } int main(void) {     int a = 1;     x = a; /* 对全局变量x赋值 */     a = f(); /* a的值为4 */     {         int b = 2;         b = a + b; /* b的值为4 */         x = x + b; /* 全局变量运算 */     }     printf("%d %d", a, x);     return 0; }</pre> <p>全局变量定义</p> <p>若局部变量与全局变量同名，局部变量优先</p> <p>输出：<br/>4, 7</p> | <p>注意全局变量的有效范围，一旦局部变量与全局变量同名，局部变量优先</p>   |
| 42 | <h3>变量作用范围示例</h3> <pre>int x=1; void main() {     int a=2;     {         int b=3;         f();     }     int t=4;     void f()     {         int x=5, b=6;     }     int a=7; }</pre> <p>x=? a=? b=?</p> <p>b=?</p> <p>x=5 b=6 t=4 a没定义</p> <p>x=? b=? t=? a=?</p>  | <p>静态变量 x,t,a 所处位置不同，作用范围不同。本例中 a 虽然是全局变量，但因位于程序最后，未起任何作用。</p>                                    |

|    |  |  |
|----|--|--|
| 43 | <p>■ <b>【例5-8】</b></p> <ul style="list-style-type: none"> <li>□ 用函数实现财务现金记账。先输入操作类型(1收入, 2支出, 0结束), 再输入操作金额, 计算现金余额, 经多次操作直到输入操作为0结束。要求定义并调用函数, 其中现金收入与现金支出分别用不同函数实现。</li> <li>■ 分析: <ul style="list-style-type: none"> <li>□ 设变量<b>cash</b>保存现金余额值, 由于它被主函数、现金收入与现金支出函数共用, 任意使用场合其意义与数值都是明确和唯一的, 因此令其为全局变量。</li> </ul> </li> </ul>  | 分析变量 cash 被设置为全局变量的原因  |
| 44 | <pre> /* 定义计算现金收入函数 */ void income(double number) {     cash = cash + number; /* 改变全局变量cash */ }  /* 定义计算现金支出函数 */ void expend(double number) {     cash = cash - number; /* 改变全局变量cash */ }  #include&lt;stdio.h&gt; double cash; void income(double number) void expend(double number) int main (void) {     int choice;     double value;      cash = 0;     printf("Enter operate choice(0--end, 1--income, 2--expend):");     scanf("%d", &amp;choice); /* 输入操作类型 */     while (choice != 0) { /* 若输入类型0, 循环结束 */         if (choice == 1    choice == 2) {             printf("Enter cash value:"); /* 输入操作现金额 */             scanf("%f", &amp;value);             if (choice == 1) income(value); /* 函数调用, 计算现金收入 */             else expend(value); /* 函数调用, 计算现金支出 */             printf("current cash: %.0f\n", cash);         }         printf("Enter operate choice(0--end, 1--income, 2--expend):");         scanf("%d", &amp;choice);     }     return 0; } </pre> | 同时强调全局变量不能滥用, 及有可能带来的副作用。  |
| 45 | <p><b>5.3.2 局部变量和全局变量</b></p> <p>■ 讨论</p> <ul style="list-style-type: none"> <li>□ 全局变量比局部变量自由度大, 更方便?</li> </ul> <p>■ 引起注意</p> <ul style="list-style-type: none"> <li>□ 对于规模较大的程序, 过多使用全局变量会带来副作用, 导致各函数间出现相互干扰。如果整个程序是由多人合作开发, 各人都按自己的想法使用全局变量, 相互的干扰可能会更严重。</li> <li>□ 因此在变量使用中, 应尽量使用局部变量, 从某个角度看使用似乎受到了限制, 但从另一个角度看, 它避免了不同函数间的相互干扰, 提高了程序质量。</li> </ul>  | 注意全局变量的有效范围, 一旦局部变量与全局变量同名, 局部变量优先   |
| 46 | <p><b>5.3.3 变量生命周期和静态局部变量</b></p> <p>■ 变量生命周期</p> <p>变量从定义开始分配存储单元, 到运行结束存储单元被回收的整个过程。</p> <p>■ 自动变量 (auto): 普通的局部变量</p> <pre> int x, y; ← auto int x, y; char c1; ← auto char c1; </pre> <ul style="list-style-type: none"> <li>□ 函数调用时, 定义变量, 分配存储单元。</li> <li>□ 函数调用结束, 收回存储单元。</li> </ul> <p>■ 全局变量: 从程序执行开始, 到程序的结束, 存储单元始终保持。</p>  | 变量生命周期与作用范围是两个不同概念, 它是指变量被系统分配单元到被回收的时间。在生命周期内, 变量可能处于作用范围中, 也可能不在作用范围中。但变量不在生命周期内, 必定不会有作用范围。 |

47

C程序存储分布示意图（例5-7）

系统存储区

操作系统（如 windows）、语言系统（如 Visual C++）

用户存储区

程序区（C 程序代码）  
如主函数、函数 complex\_add()、函数 complex\_prod()等

静态存储区

全局变量  
如 result\_real,result\_imag

静态局部变量

数据区

动态存储区  
(如自动变量)

main()变量区:  
real1,imag1,real2,imag2

complex\_add()变量区:  
real1,imag1,real2,imag2

\*\*\*\*\*

从内存空间分配引入静态变量概念：  
内存分系统区和用户区，用户区又分程序区与数据区，数据区再分静态与动态。  
静态变量的生命期较动态变量长。

48

静态局部变量

static

类型名 变量表

■ 作用范围：局部变量

■ 生命周期：全局变量

静态变量定义及特征。

49

double fact\_s (int n)  
【例】 {  
static double f = 1; /\* 定义静态变量，第一次赋值为1 \*/  
f = f \* n; /\* 在上一次调用时的值上乘n \*/  
return (f);  
}  
  
#include <stdio.h>  
double fact\_s (int n);  
int main (void)  
{  
int i, n;  
printf ("Input n:");  
scanf ("%d", &n);  
for (i = 1; i <= n; i++){  
printf ("%3d!=%.0f\n", i, fact\_s (i)); /\* 输出i和i! \*/  
}  
return 0;  
}

fact\_s()函数中并没有循环语句，它是靠静态变量f保存着上次函数调用时，计算得到的(n-1)!值，再乘上n，实现n! 的计算。

通过这个例子可以较好的理解静态变量的使用。要解释静态变量 f 的初值规定、第二次进入函数时 f 的值  
静态变量通常要多次调用函数才会显示作用。

50

静态局部变量



■ 自动变量如果没有赋初值，其存储单元中将是随机值。

■ 就静态变量而言，如果定义时没有赋初值，系统将自动赋0。

■ 赋初值只在函数第一次调用时起作用，以后调用都按前一次调用保留的值使用。

■ 静态局部变量受变量作用范围限制，不能作用于其他函数（包括主函数）。

特点：  
1、初值自动赋为 0  
2、初值第一次有效

|    |   |  |
|----|---|--|
| 51 |  <h3>静态局部变量</h3> <ul style="list-style-type: none"> <li>■ 静态变量与全局变量均位于静态存储区 <ul style="list-style-type: none"> <li>□ 他们的共同点是生命周期贯穿整个程序执行过程。</li> <li>□ 区别在于作用范围不同，全局变量可作用于所有函数，静态变量只能用于所定义函数，而不能用于其他函数。</li> </ul> </li> </ul>   | 静态局部变量具有全局变量的生命周期、局部变量的作用范围。   |
| 52 |  <h3>本章小结</h3> <ul style="list-style-type: none"> <li>■ 系统介绍函数的定义和函数调用 <ul style="list-style-type: none"> <li>□ 学习如何针对具体问题，确定需要使用函数的功能要求，再将功能用函数程序实现</li> <li>□ 考虑如何调用定义好的函数，实现主调函数与被调函数的连接</li> <li>□ 确定参数功能，掌握参数的传递实现</li> </ul> </li> <li>■ 函数与变量间的关系，不同形式的变量在函数中起的作用不同。 <ul style="list-style-type: none"> <li>□ 局部变量、全局变量和静态变量</li> </ul> </li> </ul> | 回顾和总结本章的教学要点，对学生提出能力要求： <ul style="list-style-type: none"> <li>• 能够根据问题合理确定函数功能</li> <li>• 能够使用函数结构进行熟练编程</li> <li>• 能够重点掌握参数的定义、传递</li> </ul> 能够合理运用全局变量、静态变量 |

## 5.3 练习与习题参考答案

### 5.3.1 练习参考答案

5-1 使用函数求 1 到 n 之和：输入一个正整数 n，输出 1~n 之和。要求定义和调用函数 sum(n) 求 1~n 之和。若要计算 m~n (m<n) 之和，又该如何定义函数？试编写相应程序。

解答：

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

```
    return result;
}
```

5-2 使用函数找最大值：输入 2 个数，输出其中较大的数。要求定义和调用函数 `max(a,b)` 找出并返回 `a`、`b` 中较大的数。试编写相应程序。

解答：

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

    scanf("%d%d",&a, &b);
    maximum = max(a, b);
    printf("max(%d,%d) = %d\n", a, b, maximum);
    return 0;
}
int max(int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

5-3 字符金字塔：输入一个正整数 `n` 和一个字符 `ch`，输出 `n` 行由字符 `ch` 构成的字符金字塔。试编写相应程序。

解答：

```
#include <stdio.h>
void pyramid(int n, char ch);
int main(void)
{
    int n;  char ch;
    scanf("%d%c", &n, &ch);
    pyramid(n, ch);
    return 0;
}
void pyramid(int n, char ch)
{
    int i, j;
    for(i = 1; i <= n; i++){
        for(j = 1; j <= n - i; j++)
            printf(" ");
        for(j = 1; j <= i; j++)
```



```

        printf("%c ", ch);
        putchar('\n');
    }
}

```

5-4 思考：若把例 5-9 中静态变量  $f$  定义成普通局部变量，还能实现计算  $n!$  吗？请上机检验。若把  $f$  换成全局变量又会如何？

解答：如果把静态变量  $f$  定义成普通局部变量，则每次进入函数时变量  $f$  的值都会重新赋值为 1，不能保留上次调用的计算结果，那么就无法实现计算  $n!$ 。如果把  $f$  换成全局变量，也可以实现计算  $n!$ 。

### 5.3.2 习题参考答案

#### 一. 选择题

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

#### 二. 填空题

|   |   |   |   |
|---|---|---|---|
| 1 | 2#3#5#7#  | 2 | <pre> int fun(int m); fun(n) != 0 int m m = m/10 old_digit = cur_digit </pre> |
| 3 | <pre> char ch printf("Yes.\n"); break; printf("No.\n"); YesNo(ch); </pre> |   |   |

#### 三、程序设计题

1. 使用函数计算分段函数的值：输入  $x$ ，计算并输出分段函数  $f(x)$  的值。要求定义和调用函数  $\text{sign}(x)$  实现该分段函数。试编写相应程序。

解答：

```

#include <stdio.h>
int sign(int x);
int main(void)
{
    int x, y;
    scanf("%d",&x);
    y = sign(x);
    printf("f(%d) = %d\n", x, y);
    return 0;
}
int sign(int x)
{
    int y;

```

```

    if(x > 0) y = 1;
    else if(x == 0) y = 0;
    else y = -1;
    return y;
}

```

2. 使用函数求奇数和：输入一批正整数（以零或负数为结束标志），求其中的奇数和。要求定义和调用函数 `even(n)` 判断数的奇偶性，当 `n` 为偶数时返回 1，否则返回 0。试编写相应程序。

解答：

```

#include <stdio.h>
int even(int n);
int main(void)
{
    int n, sum;
    scanf("%d",&n);
    sum = 0;
    while(n > 0){
        if(even(n) == 0) sum = sum + n;
        scanf("%d", &n);
    }
    printf("The sum of the odd numbers is %d.\n", sum);
    return 0;
}
int even(int n)
{
    int y;
    if (n % 2 == 0)
        y = 1;
    else
        y = 0;
    return y;
}

```

3. 使用函数计算两点间的距离：给定平面任意两点坐标(`x1, y1`)和(`x2, y2`)，求这两点之间的距离（保留 2 位小数）。要求定义和调用函数 `dist(x1, y1, x2, y2)` 计算两点间的距离。试编写相应程序。

解答：

```

#include <stdio.h>
#include <math.h>
double dist(double x1, double y1, double x2, double y2);
int main(void)
{
    double distance, x1, y1, x2, y2;

```

```

scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
distance = dist(x1, y1, x2, y2);
printf("Distance = %.2f\n", distance);
return 0;
}
double dist(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

```

4. 利用函数计算素数个数并求和：输入 2 个正整数  $m$  和  $n$  ( $1 \leq m, n \leq 500$ )，统计并输出  $m$  到  $n$  之间的素数的个数以及这些素数的和。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。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。

解答：

```

#include "stdio.h"
#include "math.h"
int main(void)
{
    int count, i, m, n, sum;
    int prime(int m);
    scanf("%d%d", &m, &n);
    count = sum = 0;
    for(i = m; i <= n; i++)
        if(prime(i) != 0){
            count++;
            sum = sum + i;
        }
    printf("Count = %d, sum = %d\n", count, sum);
}
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;
}

```

5. 使用函数统计指定数字的个数：读入一个整数，统计并输出该数中“2”的个数。要求定义并调用函数 `countdigit (number, digit)`，它的功能是统计整数 `number` 中数字 `digit` 的个数。例如，`countdigit(12292, 2)` 的返回值是 3。试编写相应程序。

解答：

```

#include "stdio.h"
int countdigit(int number, int digit);
int main(void)
{
    int count, in;
    scanf("%d", &in);
    count = countdigit(in, 2);
    printf("Number %d of digit 2: %d\n", in, count);
    return 0;
}
int countdigit(int number, int digit)
{
    int count;
    if(number < 0) number = -number;
    count = 0;
    do{
        if(number % 10 == digit) count++;
        number = number / 10;
    }while(number != 0);
    return count;
}

```

6. 使用函数输出水仙花数：输入 2 个正整数  $m$  和  $n$  ( $1 \leq m, n \leq 1000$ )，输出  $m$  到  $n$  之间的所有满足各位数字的立方和等于其自身的数。要求定义并调用函数 `is(number)` 判断 `number` 的各位数字之立方和是否等于其自身，若相等则返回 1，否则返回 0。试编写相应程序。

解答：

```

#include "stdio.h"
int is(int number);
int main(void)
{
    int i, m, n;
    scanf("%d%d", &m, &n);
    printf("result:\n");
    for(i = m; i <= n; i++)
        if(is(i) != 0)
            printf("%d\n", i);
    return 0;
}
int is(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;
}

```

7. 使用函数求余弦函数的近似值：输入精度  $e$  和  $x$ ，用下列公式求  $\cos(x)$  的近似值，精确到最后一项的绝对值小于  $e$ 。

$$\cos(x) = \frac{x^0}{0!} - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

要求定义并调用函数 `funcos(e,x)` 计算  $\cos(x)$  的近似值。试编写相应程序。

解答：

```

#include "stdio.h"
#include "math.h"
double funcos(double e, double x);
int main(void)
{
    double e, sum, x;
    scanf("%le%le", &e, &x);
    sum = funcos(e, x);
    printf("sum = %f\n", sum);
    return 0;
}
double funcos(double e, double x)
{
    int flag, i;
    double item, sum, tmp, power;

    flag = 1;
    power = 1;
    tmp = 1;
    i = 0;
    item = 1;
    sum = 0;
    while(fabs(item) >= e){
        item = flag * power / tmp;
        sum = sum + item;
        power = power * x * x;
        tmp = tmp * (i+1) * (i+2);
        flag = -flag;
        i = i + 2;
    }
}

```

```

    return sum;
}

```

8. 输入一个正整数  $n$ , 输出  $n$  行空心的数字金字塔。要求定义和调用函数 `hollow_pyramid ( n )` 输出  $n$  行空心的数字金字塔。当  $n=5$  时, 5 行空心的数字金字塔如下所示。

```

    1
  2 2
 3  3
4   4
555555555

```

解答:

```

#include <stdio.h>
void hollow_pyramid(int n);
int main(void)
{
    int n;
    scanf("%d", &n);
    hollow_pyramid(n);
    return 0;
}
void hollow_pyramid(int n)
{
    int i, j;
    for(i = 1; i <= n; i++){
        for(j = 1; j <= n - i; j++)
            printf(" ");
        printf("%d", i);
        if(i < n){
            for(j = 1; j <= 2*(i-1)-1; j++)
                printf(" ");
        } else{
            for(j = 1; j <= 2*(i-1)-1; j++)
                printf("%d", i);
        }
        if(i > 1) printf("%d", i);
        putchar('\n');
    }
}

```

## 5.4 实验指导教材参考答案

### 一、调试示例

使用函数计算两点间的距离: 给定平面任意两点坐标 $(x_1, y_1)$ 和 $(x_2, y_2)$ , 求这两点之间的

距离（保留 2 位小数）。要求定义和调用函数 `dist(x1, y1, x2,y2)` 计算两点间的距离。

解答：

```
#include <stdio.h>
#include <math.h>
double dist(double x1, double y1, double x2, double y2 );
int main(void)
{
    double x1, y1, x2, y2;

    printf("Enter x1, y1, x2,y2: ");
    scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
    printf("dist = %.2f\n", dist(x1, y1, x2, y2));

    return 0;
}
double dist(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
```

## 二、基础编程题

（1）符号函数：输入  $x$ ，计算并输出下列分段函数  $\text{sign}(x)$  的值。要求定义和调用函数  $\text{sign}(x)$  实现该分段函数。

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

解答：参见习题程序设计第 1 题。

（2）使用函数求 Fibonacci 数：输入正整数  $n$  ( $1 \leq n \leq 46$ )，输出斐波那契（Fibonacci）数列的第  $n$  项。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。

解答：

```
#include <stdio.h>
int fib(int n);
int main(void)
{
    int n;
    printf("Enter x:");
    scanf("%d", &n);
    printf("fib(%d)=%d\n", n, fib(n));
    return 0;
}
int fib(int n)
```

```

{
    int a, b, c, i;
    a=b=1;
    if(n<3) return 1;
    for(i = 3; i <= n; i++){
        c=a+b;
        a=b;
        b=c;
    }
    return c;
}

```

(3) 输出每个月的天数：输入年 **year**，输出该年 1~12 月每个月的天数。其中 1、3、5、7、8、10、12 月有 31 天，4、6、9、11 月有 30 天，2 月平年有 28 天，闰年有 29 天。判断闰年的条件是：能被 4 整除但不能被 100 整除，或者能被 400 整除。要求定义并调用函数 **month\_days(year, month)**，该函数返回 **year** 年 **month** 月的天数。

解答：

```

#include <stdio.h>
int month_days(int year, int month);
int main(void)
{
    int n, i;
    printf("Enter year:");
    scanf("%d", &n);
    for(i=1; i<=12; i++)
        printf("%d ", month_days(n, i));
    return 0;
}
int month_days(int year, int month)
{
    switch(month){
        case 1: case 3: case 5: case 7: case 8:
        case 10: case 12: return 31;
        case 4: case 6: case 9: case 11: return 30;
    }
    if((year%4==0 && year%100!=0)|| year%400==0)
        return 29;
    else
        return 28;
}

```

(4) 使用函数求最大公约数和最小公倍数：输入两个正整数 **m** 和 **n** ( $0 < m, n \leq 1000$ )，输出最大公约数和最小公倍数。要求定义和调用函数 **gcd(m, n)** 计算 **m** 和 **n** 的最大公约数，定义和调用函数 **lcm(m, n)** 计算 **m** 和 **n** 的最小公倍数。



解答:

```
#include <stdio.h>
int gcd(int m, int n)
{
    int r, temp;

    if(m < n){
        temp = m; m = n; n = temp;
    }
    r = m % n;
    while(r != 0){
        m = n;
        n = r;
        r = m % n;
    }
    return n;
}
int lcm(int m, int n)
{
    return m*n/gcd(m,n);
}
int main(void)
{
    int m,n;
    printf("Enter m, n:");
    scanf("%d%d", &m, &n);
    printf("gcd = %d\n", gcd(m, n));
    printf("lcm = %d\n", lcm(m, n));
    return 0;
}
```

(5) 使用函数求素数和: 输入两个正整数  $m$  和  $n(1 \leq m \leq n \leq 500)$ , 求  $m$  和  $n$  之间的素数和。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。要求定义并调用函数 `prime(p)` 判断  $p$  是否为素数, 当  $p$  为素数时返回 1, 否则返回 0。定义并调用函数 `prime_sum(m, n)`, 该函数返回区间  $[m, n]$  内所有素数的和。

解答: 参见习题程序设计第 4 题。

(6) 使用函数输出字符矩阵: 输入矩形的长度 `length`、宽度 `width` 和字符 `ch`, 输出一个长宽分别为 `length` 和 `width` 的实心字符矩阵。要求定义并调用函数 `matrix(length, width, ch)`, 它的功能时在屏幕上显示长度为 `length`、宽度为 `width`, 由字符 `ch` 组成的实心矩形图案。

解答:

```
#include <stdio.h>
void matrix(int length, int width, char ch)
{
    int i, j;
```

```

        for(i=1; i<=width; i++){
            for(j=1; j<=length; j++){
                printf("%c", ch);
                putchar('\n');
            }
        }
    }
    int main(void)
    {
        int length,width;
        char ch;
        printf("Enter length, width, ch:");
        scanf("%d%d %c", &length, &width, &ch);
        matrix(length, width, ch);
        return 0;
    }

```

### 三、改错题

使用函数求圆台体积：输入圆台的下底半径 `r_lower`、上底半径 `r_upper` 和高度 `h`，计算圆台的体积。要求定义并调用函数 `volume_tc(r_lower, r_upper, h)` 计算下底半径为 `r_lower`、上底半径为 `r_upper`、高度为 `h` 的圆台的体积，函数类型是 `double`。

错误行号： 2 正确语句： `double volume_tc(double r_lower, double r_upper, double h);`

错误行号： 9 正确语句： `v = volume_tc(r_lower, r_upper, h);`

错误行号： 14 正确语句： `double volume_tc(double r_lower, double r_upper, double h)`

### 四、拓展编程题

(1) 使用函数求余弦函数的近似值：输入 2 个实数 `eps`（精度）和 `x`，用下列公式求 `cos(x)` 的近似值，精确到最后一项的绝对值小于 `eps`。要求定义和调用函数 `funcos(eps, x)` 求余弦函数的近似值。

$$\cos(x) = \frac{x^0}{0!} - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

解答：参见习题程序设计第 7 题。

(2) 使用函数统计指定数字的个数：输入一个整数 `number` 和一个整数 `digit` ( $0 \leq \text{digit} \leq 9$ )，统计并输出整数 `number` 中数字 `digit` 的个数。要求定义并调用函数 `countdigit(number, digit)`，它的功能是统计整数 `number` 中数字 `digit` 的个数。例如，`countdigit(10090, 0)` 的返回值是 3。

解答：参见习题程序设计第 5 题。

(3) 空心的数字金字塔：输入一个正整数 `n` ( $1 \leq n \leq 9$ )，输出 `n` 行空心的数字金字塔。要求定义和调用函数 `hollow_pyramid(n)` 打印出 `n` 行空心的数字金字塔。

解答：参见习题程序设计第 8 题。