

## 第 10 章 函数与程序结构

### 10.1 教学要点

本章主要介绍函数的组织、函数的嵌套调用和递归函数的概念与编程，其中递归函数是本章的重点。本章还介绍了宏——能起简单函数功能的作用，但并不是函数。通过本章的学习，应帮助学生在函数运用上更上一层楼。

10.1 节通过 5 个简单函数构成的一个示例程序——设计一个常用圆形体体积计算器，引出了多函数组成程序的方式。教师在讲授时，先介绍和分析该示例程序的总体框架结构，函数调用的三层结构，并要打消学生对长程序的恐惧心理，让学生理解多函数结构可以有效降低程序复杂度。教师在教学中要抓住复杂问题需要多个函数解决这条主线。

10.2 节主要介绍递归函数。在介绍本节内容时，应抓住递归函数的两个要点：递归式子和递归出口。学生对递归函数的掌握有一定难度，主要问题在递归式子上。像求类似阶乘的例子，递归式子很明显，学生容易理解。但像汉诺塔、整数逆序输出等例子，如何归纳出递归式子是教学重点。要培养学生递归思维，为后续其它编程方面课程打下基础。

10.3 节介绍编译预处理概念，包括文件包含、宏和条件编译的内容。宏是本节重点：宏的概念、定义和使用。要结合示例向学生讲清楚宏在编译预处理时起作用，其实质是替换，并不能像函数那样进行计算。宏能起到简单函数的作用，但它不是函数。像带参数的宏，学生容易与函数混淆。

10.4 节通过多文件模块的学生信息库系统的例子，向学生展示大程序构成，为学生以后编写规模较大程序建立初步概念。本节主要通过文件包含把多个文件模块连接起来，上机实验时，对照实验指导书掌握 VC++ 工程文件的实现方式。外部变量、静态全局变量、外部函数、静态函数等为多文件模块间的通信提供了手段。

讲授学时：4 学时，实验学时同讲授学时。

本章的知识能力结构图见图 10.1。

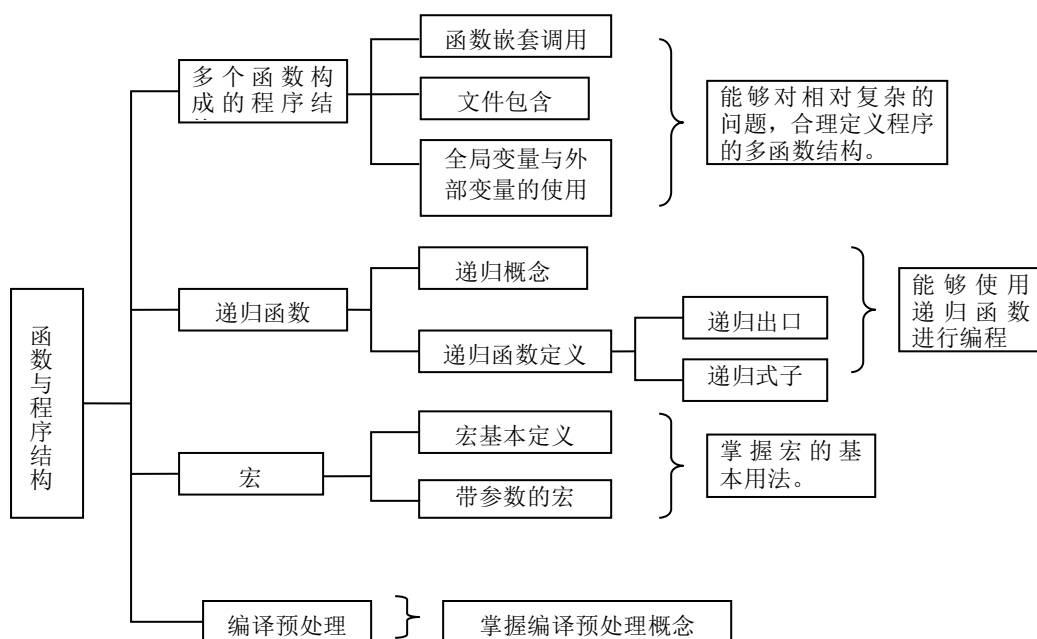




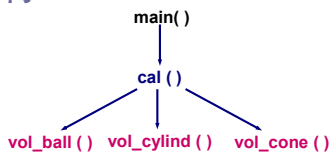


图 10.1 知识能力结构图

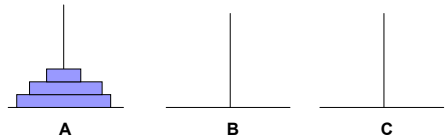
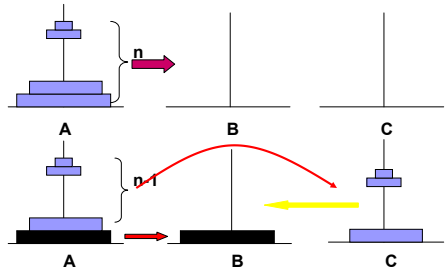
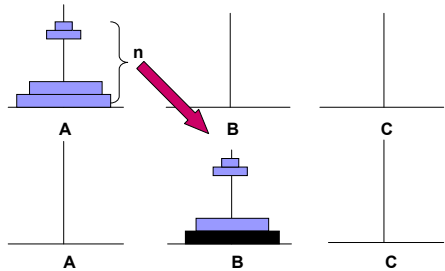
## 10.2 讲稿

1	 <h3>Chap 10 函数与程序结构</h3> <p>10.1 圆形体积计算器</p> <p>10.2 汉诺塔问题</p> <p>10.3 长度单位转换</p> <p>10.4 大程序构成</p>	<p>中心：函数与程序结构关系</p> <p>本章分 4 节：</p> <ol style="list-style-type: none"> <li>1、单文件模块内的函数嵌套调用</li> <li>2、递归函数</li> <li>3、宏为主要内容的编译预处理</li> <li>4、通过文件包含来实现稍大规模的多文件模块程序</li> </ol>
2	 <h3>本章要点</h3> <ul style="list-style-type: none"> <li>■ 怎样把多个函数组织起来？</li> <li>■ 怎样用结构化程序设计的思想解决问题？</li> <li>■ 怎样用函数嵌套求解复杂的问题？</li> <li>■ 怎样用函数递归解决问题？</li> <li>■ 如何使用宏？</li> <li>■ 如何使用多文件模块构建较大规模程序</li> </ul>	<p>提出本章的学习要点。</p> <p>简单函数→多函数构建→递归函数→函数组织——程序文件模块</p> <p>编译预处理</p>
3	 <h3>10.1 圆形体积计算器</h3> <p>使用结构化程序设计方法解决复杂的问题</p> <ul style="list-style-type: none"> <li>□ 把大问题分解成若干小问题，小问题再进一步分解成若干更小的问题</li> <li>□ 写程序时，用main()解决整个问题，它调用解决小问题的函数</li> <li>□ 这些函数又进一步调用解决更小问题的函数，从而形成函数的嵌套调用</li> </ul>	<p>对于复杂问题——本节的讨论核心，以结构化的方法进行分解是一个有效解决办法，虽然学生现在编写的都是小程序，但是结构化思想一定要让学生建立起来。</p>
4	 <h3>程序结构</h3> <pre> graph TD     main["main()"] --&gt; f1["函数1"]     main --&gt; f2["函数2"]     main --&gt; dots1["..."]     main --&gt; fm["函数m"]     f1 --&gt; f1_1["函数1_1"]     f1 --&gt; f1_2["函数1_2"]     fm --&gt; fm_1["函数m_1"]     fm --&gt; dots2["..."]     fm --&gt; fm_n["函数m_n"]     </pre>	<p>层次（树形）结构描述结构化思想。</p>

5	<p><b>10.1.1 程序解析-计算常用圆形体体积</b></p> <p>例10-1 设计一个常用圆形体体积计算器，采用命令方式输入1、2、3，分别选择计算球体、圆柱体、圆锥体的体积，并输入计算所需相应参数。</p> <p>分析：</p> <ul style="list-style-type: none"> <li>□ 输入1、2、3选择计算3种体积，其他输入结束计算</li> <li>□ 设计一个控制函数cal()，经它辨别圆形体的类型再调用计算球体、圆柱体、圆锥体体积的函数</li> <li>□ 设计单独的函数计算不同圆形体的体积</li> </ul>	提醒学生不要匆忙考虑具体程序实现，首先应进行整体分析：功能分解与确定。
6	<p><b>程序结构</b></p>  <pre> graph TD     main(main()) --&gt; cal(cal())     cal --&gt; vol_ball(vol_ball())     cal --&gt; vol_cylind(vol_cylind())     cal --&gt; vol_cone(vol_cone())   </pre> <p>3层结构，5个函数 降低程序的构思、编写、调试的复杂度 可读性好</p>	<p>直观考虑：</p> <p>球体、圆柱体、圆锥体体积的计算分别定义3个函数，再定义主函数</p> <p>主函数功能确定：</p> <p>若把根据输入选择确定3个体积函数放在主函数中，会造成主函数规模偏大，因此通过定义 cal 函数，使主函数功能简洁明了。</p>
7	<p><b>例10-1源程序</b></p> <pre> #define PI 3.141592654 void cal ( int sel ); int main(void) { int sel;   while( 1 ){     printf(" 1-计算球体体积\n");     printf(" 2-计算圆柱体积\n");     printf(" 3-计算圆锥体积\n");     printf(" 其他-退出程序运行\n");     printf(" 请输入计算命令: ");     scanf("%d",&amp;sel);     if (sel &lt; 1    sel &gt; 3)       break; /* 输入非1~3, 循环结束 */     else       cal (sel); /* 输入1~3, 调用cal() */   }   return 0; }   </pre>	<p>while(1)为永真循环，循环结束靠循环体中的 break 语句实现。除去5个 printf 语句，主函数非常简洁。</p> <p>讨论：如果用 while（条件），则条件是什么？循环体需要修改吗？</p>
8	<pre> /* 常用圆形体体积计算器的主控函数 */ void cal ( int sel ) { double vol_ball(void);   double vol_cylind(void);   double vol_cone(void);   switch (sel) {     case 1: printf("球体体积为: %.2f\n", vol_ball());             break;     case 2: printf("圆柱体积为: %.2f\n", vol_cylind());             break;     case 3: printf("圆锥体积为: %.2f\n", vol_cone());             break;   } /* 计算圆锥体积 V=h/3*PI*r*r */   double vol_cone()   { double r, h;     printf("请输入圆锥的底面半径和高: ");     scanf("%lf%lf",&amp;r,&amp;h);     return(PI*r*r*h/3.0);   } }   </pre>	<p>注意3个体积计算的函数声明。函数调用是以 printf 的实参形式出现的。</p> <p>问题：3个体积计算的函数声明可以放到别的地方吗？</p>

9	<div data-bbox="316 208 791 568"> <div> <div></div> <div>顺序调用</div> </div> <div>10.1.2 函数的嵌套调用</div> <pre> int main(void) {     .....     y = fact(3);     .....     z = mypow(3.5, 2);     ..... }  double fact(int n) {     ..... }  double mypow(double x, in n) {     ..... } </pre> <div> </div> </div>	先讨论函数的顺序调用
10	<div data-bbox="316 627 791 985"> <div> <div></div> <div>嵌套调用</div> </div> <div>函数的嵌套调用</div> <pre> int main(void) {     .....     cal (sel);     ..... }  void cal (int sel) {     .....     vol_ball()     ..... }  double vol_ball( ) {     ..... } </pre> <div> </div> </div>	对比顺序调用，嵌套调用的概念就比较清楚。C 语言允许多层的嵌套调用。
11	<div data-bbox="316 1041 805 1411"> <div> <div></div> <div>例10-1 分析</div> </div> <pre> int main(void) {     .....     cal (sel); }  void cal (int sel) {     .....     vol_ball();     vol_cylind();     vol_cone(); }  double vol_ball( ) {     ..... }  double vol_cylind( ) {     ..... }  double vol_cone( ) {     ..... } </pre> <div> </div> </div>	例 10-1 属于嵌套调用的结构。
12	<div data-bbox="316 1458 798 1783"> <div> <div></div> <div>函数的嵌套调用</div> </div> <p>             在一个函数中再调用其它函数的情况称为函数的<b>嵌套调用</b>。              如果函数<b>A</b>调用函数<b>B</b>，函数<b>B</b>再调用函数<b>C</b>，一个调用一个地嵌套下去，构成了函数的嵌套调用。              具有嵌套调用函数的程序，需要分别定义多个不同的函数体，每个函数体完成不同的功能，它们合起来解决复杂的问题。           </p> </div>	结构化的程序设计方法一般都存在函数的嵌套调用结构，用以解决复杂的问题。

13	<div data-bbox="300 203 820 241" data-label="Image"></div> <h2 data-bbox="349 248 635 282">结构化程序设计方法</h2> <ul style="list-style-type: none"> <li>■ 自顶向下，逐步求精，函数实现           <ul style="list-style-type: none"> <li>□ 自顶向下：程序设计时，应先考虑总体步骤，后考虑步骤的细节；先考虑全局目标，后考虑局部目标。先从最上层总目标开始设计，逐步使问题具体化。不要一开始就追求众多的细节。</li> <li>□ 逐步求精：对于复杂的问题，其中大的操作步骤应该再将其分解为一些子步骤的序列，逐步明晰实现过程。</li> <li>□ 函数实现：通过逐步求精，把程序要解决的全局目标分解为局部目标，再进一步分解为具体的小目标，把最终的小目标用函数来实现。问题的逐步分解关系，构成了函数间的调用关系。</li> </ul> </li> </ul>	<p>多个函数有效组织的目的之一——结构化程序设计方法实现：</p> <ul style="list-style-type: none"> <li>● 自顶向下</li> <li>● 逐步求精</li> <li>● 函数实现</li> </ul> <p>虽然第五章介绍过结构化程序设计方法的思想，现在重提，让学生有进一步的认识</p>
14	<div data-bbox="300 620 820 658" data-label="Image"></div> <h2 data-bbox="331 663 681 696">函数设计时应注意的问题</h2> <ul style="list-style-type: none"> <li>■ 限制函数的长度。一个函数语句数不宜过多，既便于阅读、理解，也方便程序调试。若函数太长，可以考虑把函数进一步分解实现。</li> <li>■ 避免函数功能间的重复。对于在多处使用的同一个计算或操作过程，应当将其封装成一个独立的函数，以达到一处定义、多处使用的目的，以避免功能模块间的重复。</li> <li>■ 减少全局变量的使用。应采用定义局部变量作为函数的临时工作单元，使用参数和返回值作为函数与外部进行数据交换的方式。只有当确实需要多个函数共享的数据时，才定义其为全局变量。</li> </ul>	<p>提出<b>函数设计时应注意的问题</b></p> <p>使学生在今后的函数编写中能遵循这些原则</p>
15	<div data-bbox="300 1037 820 1075" data-label="Image"></div> <h2 data-bbox="331 1084 569 1120">10.2 汉诺塔问题</h2> <div data-bbox="360 1153 617 1263" data-label="List-Group"> <ul style="list-style-type: none"> <li>10.2.1 程序解析</li> <li>10.2.2 递归函数基本概念</li> <li>10.2.3 递归程序设计</li> </ul> </div>	<p>本节介绍递归函数，这是函数中一块不易掌握的内容。先从有趣的例子入手，再讲递归函数的基本定义与基本使用</p>
16	<div data-bbox="300 1453 820 1491" data-label="Image"></div> <h3 data-bbox="336 1489 738 1523">10.2.1 汉诺(Hanoi)塔问题解析</h3> <div data-bbox="359 1529 799 1655" data-label="Image"></div> <p data-bbox="376 1680 604 1704">将 64 个盘从座A搬到座B</p> <ol style="list-style-type: none"> <li>(1) 一次只能搬一个盘子</li> <li>(2) 盘子只能插在A、B、C三个杆中</li> <li>(3) 大盘不能压在小盘上</li> </ol>	<p>汉诺塔是递归函数的经典例子，它很难用非递归实现。由此也可以向学生说明递归方法的重要性。</p> <p>递归方法的优越性：</p> <ol style="list-style-type: none"> <li>1、 能实现非递归很难实现的问题</li> <li>2、 程序简洁</li> </ol> <p>难点：</p> <p>递归思想不容易建立。</p>

17	<p>分析</p> 	<p>请学生说明搬动 3 个盘子的过程，教师可以 ppt 非播放状态下移动图形。</p>
18	<p>分析</p> 	<p>演示 n 个盘子的实现过程，强调如果能把 n-1 个盘子搬开，就能解决 n 个盘子的问題——这是递归的根本（递归式子）。</p>
19	<p>分析</p> 	<p>演示</p>
20	<p><b>10.2.1 汉诺(Hanoi)塔问题解析</b></p> <ul style="list-style-type: none"> <li>■ 递归方法的两个要点 <ul style="list-style-type: none"> <li>□ (1) 递归出口：一个盘子的解决方法；</li> <li>□ (2) 递归式子：如何把搬动64个盘子的问題简化成搬动63个盘子的问題。</li> </ul> </li> <li>■ 把汉诺塔的递归解法归纳成三个步骤： <ul style="list-style-type: none"> <li>□ n-1个盘子从座A搬到座C</li> <li>□ 第n号盘子从座A搬到座B</li> <li>□ n-1个盘子从座C搬到座B</li> </ul> </li> </ul>	<p>从汉诺塔例子归纳出递归方法的 2 个要点：递归出口与递归式子 后面的例子都从这两点着手分析</p>

21	<div data-bbox="300 208 798 347"></div> <p>算法：hanio(n个盘, A→B, C为过渡)</p> <pre>{ if (n == 1)     直接把盘子A→B else{     hanio(n-1个盘, A→C, B为过渡)     把n号盘 A→B     hanio(n-1个盘, C→B, A为过渡) }</pre>	<p>以算法形式给出汉诺塔的解法，这里注重分析，不急于写出程序</p>						
22	<div data-bbox="300 622 798 689"></div> <p><b>10.2.2递归函数基本概念</b></p> <p>■ 例10-2 用递归函数实现求n!</p> <p>□ 递推法</p> <ul style="list-style-type: none"><li>在学习循环时，计算n! 采用的就是递推法： <math>n! = 1 \times 2 \times 3 \times \dots \times n</math></li><li>用循环语句实现： result = 1; for(i = 1; i &lt;= n; i++)     result = result * i;</li></ul> <p>□ 递归法</p> <ul style="list-style-type: none"><li><math>n! = n \times (n-1)!</math>      当 <math>n &gt; 1</math>      递归式子     = 1                      当 <math>n=1</math>或<math>n=0</math>      递归出口</li><li>即求n! 可以在(n-1)! 的基础上再乘上n。如果把求n! 写成函数fact(n)，则fact (n)的实现依赖于fact(n-1)。</li></ul>	<p>相同问题采用不同的表示方式：递推法与递归法。它将导致不同程序方法的实现：</p> <ul style="list-style-type: none"><li>■ 第4章通过循环实现递推法</li><li>■ 例10-2用递归法实现</li></ul>						
23	<div data-bbox="300 1039 798 1106"></div> <p>例10-2 用递归函数求n!。 <b>10.2.2递归函数基本概念</b></p> <pre>#include &lt;stdio.h&gt; double fact(int n); int main(void) { int n;   scanf ("%d", &amp;n);   printf ("%d", fact (n) );   return 0; } double fact(int n) /* 函数定义 */ { double result;   if (n==1    n == 0) /* 递归出口 */     result = 1;   else     result = n * fact(n-1);   return result; }</pre>	<p>阶乘函数实现已经学习过，引导学生找出递归函数实现与以前实现的不同：</p> <ul style="list-style-type: none"><li>■ 自己调用自己</li><li>■ 有 <math>n=0  n=1</math> 的特殊情况考虑</li></ul>						
24	<div data-bbox="300 1456 798 1523"></div> <p><b>10.2.2 递归函数基本概念</b></p> <table border="1" data-bbox="355 1574 762 1776"><thead><tr><th>函数直接递归调用</th><th colspan="2">函数间接递归调用</th></tr></thead><tbody><tr><td><pre>int f(int x) {     int y;     ....     y = f (x);     ....     return y; }</pre></td><td><pre>int f(int x) {     int y;     ....     y = g (x);     ....     return y; }</pre></td><td><pre>int g(int x) {     int z;     ....     z = f (x);     ....     return z; }</pre></td></tr></tbody></table> <p>图 10.4 函数递归调用的两种形式</p>	函数直接递归调用	函数间接递归调用		<pre>int f(int x) {     int y;     ....     y = f (x);     ....     return y; }</pre>	<pre>int f(int x) {     int y;     ....     y = g (x);     ....     return y; }</pre>	<pre>int g(int x) {     int z;     ....     z = f (x);     ....     return z; }</pre>	<p>递归调用的2种方式：</p> <ul style="list-style-type: none"><li>■ 直接调用自己</li><li>■ 间接调用自己</li></ul>
函数直接递归调用	函数间接递归调用							
<pre>int f(int x) {     int y;     ....     y = f (x);     ....     return y; }</pre>	<pre>int f(int x) {     int y;     ....     y = g (x);     ....     return y; }</pre>	<pre>int g(int x) {     int z;     ....     z = f (x);     ....     return z; }</pre>						









29	<div data-bbox="300 203 820 237" data-label="Section-Header"> <h3>10.2.3 递归程序设计</h3> </div> <div data-bbox="331 280 775 327" data-label="List-Group"> <ul style="list-style-type: none"> <li>由于结果是在屏幕上输出，因此函数返回类型为 void</li> </ul> </div> <div data-bbox="331 347 700 535" data-label="Text"> <pre>void reverse(int num) {     if (num &lt;= 9)         printf ("%d", num);    /* 递归出口 */     else {         printf ("%d", num % 10);         reverse (num / 10);    /* 递归调用 */     } }</pre> </div>	<div data-bbox="847 197 1212 271" data-label="Text"> <p>展现程序简洁的形式 可以与以前循环实现进行对比</p> </div>
30	<div data-bbox="300 620 820 689" data-label="Section-Header"> <h3>例10-4 汉诺(Hanoi)塔问题</h3> </div> <div data-bbox="359 689 799 806" data-label="Diagram"> </div> <div data-bbox="419 813 732 983" data-label="Text"> <pre>hanio(n个盘, A→B, C为过渡) {     if (n == 1)         直接把盘子A→B     else{         hanio(n-1个盘, A→C, B为过渡)         把n号盘A→B         hanio(n-1个盘, C→B, A为过渡)     } }</pre> </div>	<div data-bbox="847 613 1241 687" data-label="Text"> <p>回到汉诺塔例子，进行程序实现 先回顾算法</p> </div>
31	<div data-bbox="300 1037 820 1426" data-label="Text"> <pre>/* 搬动n个盘，从a到b，c为中间过渡 */ void hanio(int n, char a, char b, char c) {     if (n == 1)         printf ("%c--&gt;%c\n", a, b);     else{         hanio(n-1, a, c, b);         printf ("%c--&gt;%c\n", a, b);         hanio(n-1, c, b, a);     } }  int main(void) {     int n;     printf ("input the number of disk: ");     scanf ("%d", &amp;n);     printf ("the steps for %d disk are:\n", n);     hanio(n, 'a', 'b', 'c');     return 0; }</pre> <div data-bbox="692 1055 788 1090" data-label="Section-Header"> <h4>源程序</h4> </div> <div data-bbox="628 1099 810 1290" data-label="Text"> <pre>input the number of disk: 3 the steps for 3 disk are: a--&gt;b a--&gt;c b--&gt;c a--&gt;b c--&gt;a c--&gt;b a--&gt;b</pre> </div> </div>	<div data-bbox="847 1028 1161 1061" data-label="Text"> <p>3个盘子搬动的程序结果。</p> </div>
32	<div data-bbox="300 1453 820 1848" data-label="Complex-Block"> <div data-bbox="628 1458 810 1664" data-label="Text"> <pre>input the number of disk: 3 the steps for 3 disk are: a--&gt;b a--&gt;c b--&gt;c a--&gt;b c--&gt;a c--&gt;b a--&gt;b</pre> </div> <div data-bbox="368 1686 810 1825" data-label="Diagram"> </div> </div>	<div data-bbox="847 1444 1353 1518" data-label="Text"> <p>针对结果，教师可以 ppt 非播放状态下移动图形来说明结果。</p> </div>

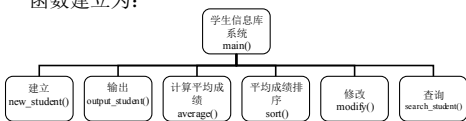
33	<p>课堂练习：利用递归函数计算x的n次幂</p> <pre> int mi(int x, int n) {     if (n==1)         return x;     else         return x*mi(x,n-1); } </pre>	<p>课堂练习</p> <p>还是从递归的 2 个要点检查学生掌握的情况</p>
34	<p>10.3 长度单位转换</p> <ul style="list-style-type: none"> <li>10.3.1 程序解析</li> <li>10.3.2 宏基本定义</li> <li>10.3.3 带参数的宏定义</li> <li>10.3.4 文件包含</li> <li>10.3.5 编译预处理</li> </ul>	<p>宏是本节主要内容，文件包含再系统展现一下，编译预处理了解一下即可</p>
35	<p>10.3.1 程序解析</p> <p>例10-5 欧美国家长度使用英制单位，1英里=1609米，1英尺=30.48厘米，1英寸=2.54厘米。请编写程序转换。</p> <pre> #include&lt;stdio.h&gt; #define Mile_to_meter 1609 /* 1英里=1609米 */ #define Foot_to_centimeter 30.48 /* 1英尺=30.48厘米 */ #define Inch_to_centimeter 2.54 /* 1英寸=2.54厘米 */ int main(void) {     float foot, inch, mile; /* 定义英里、英尺、英寸变量 */     printf("Input mile,foot and inch:");     scanf("%f%f%f", &amp;mile, &amp;foot, &amp;inch);     printf("%f miles=%f meters\n", mile, mile * Mile_to_meter);     /* 计算英里的米数 */     printf("%f feet=%f centimeters\n", foot, foot *         Foot_to_centimeter); /* 计算英尺的厘米数 */     printf("%f inches=%f centimeters\n", inch, inch *         Inch_to_centimeter); /* 计算英寸的厘米数 */     return 0; } </pre>	<p>本例中虽然宏名定义得较长，但可读性非常清晰</p>
36	<p>10.3 宏定义</p> <p><b>#define 宏名标识符 宏定义字符串</b></p> <p>编译时，把程序中所有与宏名相同的字符串，用宏定义字符串替代</p> <pre> #define PI 3.14 #define arr_size 4 </pre> <p>说明：</p> <ul style="list-style-type: none"> <li>宏名一般用大写字母，以与变量名区别</li> <li>宏定义不是C语句，后面不得跟分号</li> <li>宏定义可以嵌套使用</li> </ul> <pre> #define PI 3.14 #define S 2*PI*PI </pre> <p>多用于符号常量</p>	<p>在符号常量使用的基础上，引入宏概念。</p> <p>宏的定义、使用方法</p> <p>宏定义不是 C 语句</p>

37	<div data-bbox="300 203 820 241" data-label="Image"></div> <h3 data-bbox="360 259 624 293">10.3.1 宏基本定义</h3> <ul style="list-style-type: none"> <li>■ 宏定义可以写在程序中任何位置，它的作用范围从定义书写处到文件尾。</li> <li>■ 可以通过“<b>#undef</b>”强制指定宏的结束范围。</li> </ul>	与变量相同，宏定义也有作用范围
38	<div data-bbox="300 620 820 658" data-label="Image"></div> <h3 data-bbox="349 654 539 683">宏的作用范围</h3> <pre data-bbox="331 687 735 981"> #define A "This is the first macro" void f1() {     printf("A\n"); } #define B "This is the second macro" void f2() {     printf(B); } #undef B int main(void) {     f1();     f2();     return 0; } </pre> <p data-bbox="644 772 735 790">A 的有效范围</p> <p data-bbox="523 824 614 842">B 的有效范围</p>	通过例子演示宏定义的作用范围。本例子仅作说明之用，例中 A 与 B 的用法不具有实用性。
39	<div data-bbox="300 1037 820 1075" data-label="Image"></div> <h3 data-bbox="331 1077 699 1104">带参数的宏定义实现简单的函数功能</h3> <p data-bbox="323 1106 563 1126">例10-7 简单的带参数的宏定义。</p> <pre data-bbox="323 1128 703 1373"> #include &lt;stdio.h&gt; #define MAX(a, b) (a) &gt; (b) ? (a): (b) #define SQR(x) (x) * (x) int main (void) {     int x, y;     scanf ("%d %d", &amp;x, &amp;y);     x = MAX (x, y); /* 引用宏定义 */     y = SQR (x);    /* 引用宏定义 */     printf ("%d %d\n", x, y);     return 0; } </pre>	本例说明宏能解决简单函数功能，但参数引用上必须带括号。
40	<div data-bbox="300 1453 820 1491" data-label="Image"></div> <h3 data-bbox="331 1494 552 1525">10.3.3 带参数的宏</h3> <p data-bbox="555 1467 810 1541">各位数字的立方和等于它本身的数。例如153的各位数字的立方和是<math>1^3+5^3+3^3=153</math></p> <p data-bbox="331 1547 592 1574">例: <code>#define f(a) (a)*(a)*(a)</code></p> <pre data-bbox="331 1576 751 1827"> int main (void) /* 水仙花数 */ {     int i, x, y, z;     for (i = 1; i &lt; 1 000; i++) {         x=i%10; y=i/10%10; z=i/100;         if (f(x)+f(y)+f(z) == i)             printf ("%d\n", i);     }     return 0; } </pre> <p data-bbox="523 1765 676 1789"><math>f(x+y) = (x+y)^3 ?</math></p> <p data-bbox="608 1800 751 1827"><math>= x+y*x+y*x+y</math></p>	<p>通过例子说明带参数的宏定义的使用方法。</p> <p>带参数的宏定义不是函数，不能解决 <math>f(x+y)</math> 的问题——除非加上括号。其根本原因是宏的替换作用，这点务必要让学生理解。做习题时必须先替换，再计算。</p>

41	<p><b>示例 用宏实现两个变量值的交换</b></p> <pre> #define f(a,b,t) t=a; a=b; b=t; int main() {     int x,y,t;     scanf("%d%d",&amp;x,&amp;y);     f(x,y,t);     printf("%d %d\n", x, y);     return 0; } </pre> <p>编译时被替换</p> <p>与函数的区别在哪里？</p> <ul style="list-style-type: none"> <li>带参数的宏定义不是函数，宏与函数是两种不同的概念</li> <li>宏可以实现简单的函数功能</li> </ul>	<p>通过本例和学生讨论宏与函数的异同点： 宏能解决简单函数功能，但计算机运行本质截然不同，宏是编译预处理时进行替换，不存在类似函数的调用过程。</p>
42	<p><b>宏定义应用示例</b></p> <ul style="list-style-type: none"> <li>定义宏LOWCASE，判断字符c是否为小写字母。  <pre>#define LOWCASE(c) (((c) &gt;= 'a') &amp;&amp; ((c) &lt;= 'z'))</pre> </li> <li>定义宏CTOD将数字字符（'0'~'9'）转换为相应的十进制整数，-1表示出错。  <pre>#define CTOD(c) (((c) &gt;= '0') &amp;&amp; ((c) &lt;= '9') ? c - '0' : -1)</pre> </li> </ul>	<p>宏有广泛应用方式——简单函数功能。</p>
43	<p><b>带宏定义的程序输出</b></p> <pre> #define F(x) x - 2 #define D(x) x * F(x) int main() {     printf("%d,%d", D(3), D(D(3)));     return 0; } </pre>	<p>本例学生较容易出错，可以先让学生计算，然后通过计算机运行，教师再解释原因。</p>
44	<p><b>结果分析</b></p> <ul style="list-style-type: none"> <li>阅读带宏定义的程序，先全部替换好，最后再统一计算</li> <li>不可一边替换一边计算，更不可以人为添加括号</li> </ul> <p> <math>D(3) = x * F(x)</math>      先用x替换展开  <math>= x * x - 2</math>      进一步对F(x)展开，这里不能加括号  <math>= 3 * 3 - 2 = 7</math>      最后把x=3代进去计算 </p> <p> <math>D(D(3)) = D(x * x - 2)</math>      先对D(3)用x替换展开，  <math>= x * x - 2 * F(x * x - 2)</math>      拿展开后的参数对D进一步进行宏替换  <math>= x * x - 2 * x * x - 2 - 2</math>      拿展开后的参数对F进一步进行宏替换  <math>= 3 * 3 - 2 * 3 * 3 - 2 - 2 = -13</math>      最后把x=3代进去计算 </p> <p>运行结果：7 -13</p>	<p>分析出错原因：</p> <p>其根本原因是宏的替换作用，而不是函数计算，这点务必要让学生理解。做习题时必须先替换，再计算。</p>

45	<div data-bbox="300 203 347 237" data-label="Image"></div> <h3 data-bbox="347 248 572 282">10.3.4 文件包含</h3> <ul style="list-style-type: none"> <li>■ 系统文件以 <b>stdio.h</b>、<b>math.h</b> 等形式供编程者调用</li> <li>■ 实用系统往往有自己诸多的宏定义，也以 <b>.h</b> 的形式组织、调用</li> <li>■ 问题：如何把若干 <b>.h</b> 头文件连接成一个完整的可执行程序？ <ul style="list-style-type: none"> <li>□ 文件包含 <b>include</b></li> </ul> </li> </ul>	文件包含是解决多个文件程序连接的有效方法。
46	<div data-bbox="300 620 347 654" data-label="Image"></div> <h3 data-bbox="651 647 764 680">文件包含</h3> <ul style="list-style-type: none"> <li>■ 格式 <ul style="list-style-type: none"> <li>□ <b>#include &lt;需包含的文件名&gt;</b></li> <li>□ <b>#include "需包含的文件名"</b></li> </ul> </li> <li>■ 作用 <div data-bbox="539 757 742 779" data-label="Text">当前文件夹+系统文件夹</div> <p>把指定的文件模块内容插入到 <b>#include</b> 所在的位置，当程序编译连接时，系统会把所有 <b>#include</b> 指定的文件拼接生成可执行代码。</p> </li> <li>■ 注意 <ul style="list-style-type: none"> <li>□ 编译预处理命令，以 <b>#</b> 开头。</li> <li>□ 在程序编译时起作用，不是真正的 C 语句，行尾没有分号。</li> </ul> </li> </ul>	区分 <b>include</b> 的两种写法，一般： <ul style="list-style-type: none"> <li>● <b>&lt;...&gt;</b> 适用于系统头文件</li> <li>1) <b>"..."</b> 适用于多文件模块连接</li> </ul>
47	<div data-bbox="300 1037 347 1070" data-label="Image"></div> <p>例 10-7 将例 10-5 中长度转换的宏，定义成头文件 <b>length.h</b>，并写出主函数文件。</p> <p>头文件 <b>length.h</b> 源程序</p> <pre>#define Mile_to_meter 1609      /* 1英里=1609米 */ #define Foot_to_centimeter 30.48 /* 1英尺=30.48厘米 */ #define Inch_to_centimeter 2.54 /* 1英寸=2.54厘米 */</pre> <p>主函数文件 <b>prog.c</b> 源程序</p> <pre>#include&lt;stdio.h&gt; #include "length.h"           /* 包含自定义头文件 */ int main(void) {     float foot, inch, mile;    /* 定义英里，英尺，英寸变量 */     printf("Input mile, foot and inch:");     scanf("%f%f%f", &amp;mile, &amp;foot, &amp;inch);     printf("%f miles=%f meters\n", mile, mile * Mile_to_meter);     printf("%f feet=%f centimeters\n", foot, foot * Foot_to_centimeter);     printf("%f inches=%f centimeters\n", inch, inch * Inch_to_centimeter);     return 0; }</pre>	本例把 3 个宏定义单独形成一个 <b>.h</b> 文件，以作自定义头文件的示例
48	<div data-bbox="300 1453 347 1487" data-label="Image"></div> <p>将例 10-1 的 5 个函数分别存储在 2 个 <b>.C</b> 文件上，要求通过文件包含把它们联结起来。</p> <div data-bbox="300 1532 815 1823" data-label="Diagram"> <div data-bbox="300 1532 528 1621" data-label="Text"> <p>头文件 <b>length.h</b></p> <pre>#define Mile_to_meter 1609 #define Foot_to_centimeter 30.48 #define Inch_to_centimeter 2.54</pre> </div> <div data-bbox="336 1655 523 1823" data-label="Text"> <p>主函数文件 <b>prog.c</b></p> <pre>#include&lt;stdio.h&gt; #include "length.h" int main(void) {     float mile, foot, inch;     ...     return 0; }</pre> </div> <div data-bbox="580 1532 815 1756" data-label="Text"> <p>编译连接后生成的程序</p> <pre>... stdio.h的内容 #define Mile_to_meter 1609 #define Foot_to_centimeter 30.48 #define Inch_to_centimeter 2.54 int main(void) {     float mile, foot, inch;     ...     return 0; }</pre> </div> </div>	文件包含的作用： 在编译时，把需包含的文件插入到 <b>include</b> 位置，插入后的程序规模将扩大。 可进一步引申到系统头文件，例如每个程序都会用到的 <b>include &lt;stdio.h&gt;</b> ，其作用将把系统头文件 <b>stdio.h</b> 的所有程序插入到我们编写的程序前，来确保 <b>printf</b> 等功能的实现。

49	 <h3>常用标准头文件</h3> <ul style="list-style-type: none"> <li>■ <b>ctype.h</b> 字符处理</li> <li>■ <b>math.h</b> 与数学处理函数有关的说明与定义</li> <li>■ <b>stdio.h</b> 输入输出函数中使用的有关说明和定义</li> <li>■ <b>string.h</b> 字符串函数的有关说明和定义</li> <li>■ <b>stddef.h</b> 定义某些常用内容</li> <li>■ <b>stdlib.h</b> 杂项说明</li> <li>■ <b>time.h</b> 支持系统时间函数</li> </ul>	可以结合教材后面附录说明。
50	 <h3>10.3.5 编译预处理</h3> <ul style="list-style-type: none"> <li>■ 编译预处理是C语言编译程序的组成部分，它用于解释处理C语言源程序中的各种预处理指令。</li> <li>■ 文件包含(<b>#include</b>)和宏定义(<b>#define</b>)都是编译预处理指令 <ul style="list-style-type: none"> <li>□ 在形式上都以“#”开头，不属于C语言中真正的语句</li> <li>□ 增强了C语言的编程功能，改进C语言程序设计环境，提高编程效率</li> </ul> </li> </ul>	结合宏的运行，再强调编译预处理概念与使用方式。
51	 <h3>编译预处理</h3> <ul style="list-style-type: none"> <li>■ C程序的编译处理，目的是把每一条C语句用若干条机器指令来实现，生成目标程序。</li> <li>■ 由于<b>#define</b>等编译预处理指令不是C语句，不能被编译程序翻译，需要在真正编译之前作一个预处理，解释完成编译预处理指令，从而把预处理指令转换成相应的C程序段，最终成为由纯粹C语句构成的程序，经编译最后得到目标代码。</li> </ul>	解释什么是编译，它与编译预处理的关系。
52	 <h3>编译预处理功能</h3> <ul style="list-style-type: none"> <li>■ 编译预处理的主要功能： <ul style="list-style-type: none"> <li>□ 文件包含 (<b>#include</b>)</li> <li>□ 宏定义 (<b>#define</b>)</li> <li>□ 条件编译</li> </ul> </li> </ul>	

53	<p><b>编译预处理功能</b></p> <ul style="list-style-type: none"> <li>■ 条件编译 <pre> #define FLAG 1 #if FLAG     程序段1 #else     程序段2 #endif </pre> </li> </ul>	该内容了解即可。
54	<p><b>10.4 大程序构成</b> ——多文件模块的学生信息库系统</p> <ul style="list-style-type: none"> <li>■ 10.4.1 分模块设计学生信息库系统</li> <li>■ 10.4.2 C程序文件模块</li> <li>■ 10.4.3 文件模块间的通信</li> </ul>	<p>本节将展示给学生处理规模稍大程序（多函数）的方法，由于教材所限，示例程序规模仍不够大，但思想需要掌握。</p> <p>也希望学生最后能完整实现学生信息库系统。</p>
55	<p><b>10.4.1 分模块设计学生信息库系统</b></p> <p>例10-8 请综合例9-1、例9-2、例9-3和例9-4，分模块设计一个学生信息库系统。该系统包含学生基本信息的建立和输出、计算学生平均成绩、按照学生的平均成绩排序以及查询、修改学生的成绩等功能。</p> <p>函数建立为：</p> 	<p>本例来源于第九章多个例题，由于题目本身已比较熟悉，学生可以把注意力放在集成实现上。</p>
56	<p><b>10.4.1 分模块设计学生信息库系统</b></p> <ul style="list-style-type: none"> <li>■ 由于整个程序规模较大，按照功能图，分成三个程序文件模块，并把结构体定义也写成一个头文件。 <ul style="list-style-type: none"> <li>□ 头文件student.h</li> <li>□ 输入输出程序文件input_output.c <ul style="list-style-type: none"> <li>■ void new_student(struct student students[])</li> <li>■ void output_student(struct student students[])</li> </ul> </li> <li>□ 计算平均成绩与平均成绩排序程序文件aver_sort.c <ul style="list-style-type: none"> <li>■ void average(struct student students[])</li> <li>■ void sort(struct student students[])</li> </ul> </li> <li>□ 查询修改程序文件modify.c <ul style="list-style-type: none"> <li>■ void modify(struct student students[])</li> <li>■ void search_student(struct student students[], int num)</li> </ul> </li> </ul> </li> </ul>	<p>程序分成三个程序文件模块+一个头文件（含结构体定义）</p>

57	<h3>10.4.1 分模块设计学生信息库系统</h3> <ul style="list-style-type: none"> <li>■ 一共定义了三个.c程序文件和一个.h头文件，它们各自独立，再通过主函数main()调用。主函数放在student_system.c文件中，各文件存放在同一个文件夹下，相互间的连接采用文件包含的形式。 <ul style="list-style-type: none"> <li>□ 主函数程序文件student_system.c <pre>#include "student.h" #include "input_output.c" #include "aver_sort.c" #include "modify.c" int Count = 0; /* 全局变量，记录当前学生总数 */ int main(void) { ..... } /* 主函数调用各函数 */</pre> </li> </ul> </li> </ul>	主函数构成
58	<h3>10.4.2 C程序文件模块</h3> <ul style="list-style-type: none"> <li>■ 结构化程序设计：我们把保存有一部分程序的文件称为程序文件模块</li> <li>■ 一个大程序最好由一组小函数构成</li> <li>■ 如果程序规模很大，需要几个人合作完成的话，每个人所编写的程序会保存在自己的.c文件中</li> <li>■ 为了避免一个文件过长，也会把程序分别保存为几个文件。</li> <li>■ 一个大程序会由几个文件组成，每一个文件又可能包含若干个函数。</li> </ul>	从结构化程序设计思想出发，解释例 10-8 设计的依据，并引出程序文件模块的概念
59	<h3>10.4.2 C程序文件模块</h3> <ul style="list-style-type: none"> <li>■ 一个大程序可由几个程序文件模块组成，每一个程序文件模块又可能包含若干个函数。程序文件模块只是函数书写的载体。</li> <li>■ 当大程序分成若干文件模块后，可以对各文件模块分别编译，然后通过连接，把编译好的文件模块再合起来，连接生成可执行程序。</li> <li>■ 问题：如何把若干程序文件模块连接成一个完整的可执行程序？ <ul style="list-style-type: none"> <li>□ 文件包含</li> <li>□ 工程文件（由具体语言系统提供）</li> </ul> </li> </ul>	<p>解释把一个程序用几个文件模块实现的原因——对复杂程序：</p> <ol style="list-style-type: none"> <li>1、便于分模块编译、连接与调试，降低上机调试复杂度</li> <li>2、可实现多人合作开发程序——这样对程序分头编译查错，十分方便</li> </ol> <p>多文件模块通过文件包含连接，注意各模块中不能有各自的#include stdio.h 等内容，统一在 main()文件中</p>
60	<h3>10.4.2 C程序文件模块</h3> <ul style="list-style-type: none"> <li>■ 程序－文件－函数关系 <ul style="list-style-type: none"> <li>□ 小程序：主函数+若干函数 → 一个文件</li> <li>□ 大程序：若干程序文件模块（多个文件） → 每个程序文件模块可包含若干个函数 → 各程序文件模块分别编译，再连接</li> <li>□ 整个程序只允许有一个main()函数</li> </ul> </li> </ul>	<p>帮学生理清关系：</p> <p>程序－文件－函数关系</p>



61	<h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> <li>■ 文件模块与变量 <ul style="list-style-type: none"> <li>□ 外部变量</li> <li>□ 静态全局变量</li> </ul> </li> <li>■ 文件模块与函数 <ul style="list-style-type: none"> <li>□ 外部函数</li> <li>□ 静态的函数</li> </ul> </li> </ul>	<p>局部变量、静态局部变量均与具体函数相关，与文件模块无关。</p> <p>这里介绍函数模块与变量、函数间的关系</p>
62	<h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> <li>■ 外部变量 <ul style="list-style-type: none"> <li>□ 全局变量只能在某个模块中定义一次，如果其他模块要使用该全局变量，需要通过外部变量的声明 外部变量声明格式为： <code>extern 变量名;</code></li> <li>□ 如果在每一个文件模块中都定义一次全局变量，模块单独编译时不会发生错误，一旦把各模块连接在一起时，就会产生对同一个全局变量名多次定义的错误</li> <li>□ 反之，不经声明而直接使用全局变量，程序编译时会出现“变量未定义”的错误。</li> </ul> </li> </ul>	<p>在某一文件模块中定义的全局变量，想在其他模块中使用，相关文件模块必须通过外部变量声明后，方可使用。变量实体只有一个，在定义处，外部变量声明只是告诉编译系统，并没有分配单元。</p>
63	<h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> <li>■ 静态全局变量 <ul style="list-style-type: none"> <li>□ 当一个大的程序由多人合作完成时，每个程序员可能都会定义一些自己使用的全局变量</li> <li>□ 为避免自己定义的全局变量影响其他人编写的模块，即所谓的全局变量副作用，<b>静态全局变量可以把变量的作用范围仅限于当前的文件模块中</b></li> <li>□ 即使其他文件模块使用外部变量声明，也不能使用该变量。</li> </ul> </li> </ul>	<p><b>静态</b>全局变量可以使全局变量只限于本文件模块中使用，而不能被其他文件引用，即使其它文件模块声明了外部变量也无法使用。</p> <p><b>静态</b>全局变量的作用是避免多文件模块间的变量干扰——副作用。</p>
64	<h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> <li>■ 文件模块与函数 <ul style="list-style-type: none"> <li>□ 外部函数 <ul style="list-style-type: none"> <li>■ 如果要实现在一个模块中调用另一模块中的函数时，就需要对函数进行外部声明。声明格式为： <code>extern 函数类型 函数名(参数表说明);</code></li> </ul> </li> <li>□ 静态的函数 <ul style="list-style-type: none"> <li>■ 把函数的使用范围限制在文件模块内，不使某程序员编写的自用函数影响其他程序员的程序，即使其他文件模块有同名的函数定义，相互间也没有任何关联。</li> <li>■ 增加模块的独立性。</li> </ul> </li> </ul> </li> </ul>	<p>类似外部变量的概念，多文件模块上的函数相互间的调用，采用外部函数<b>声明</b>。声明仅对编译起作用，说明要调用的函数在外部文件模块，本地未定义。</p> <p>类似静态全局变量的概念，静态函数可以阻断多文件模块上的函数相互间的调用，避免多文件模块间的函数干扰——副作用。</p>

## 本章小结

- 多函数程序的组织结构
  - 函数调用的层次结构
  - 多文件模块实现：文件包含
  - 合理运用变量在多文件模块、多函数间的关联
  - 程序文件模块：变量与文件模块、函数与文件模块的关系
- 递归函数
  - 构成要素：递归式子（重点）与递归出口
  - 运用递归函数解决特殊问题（如汉诺塔）
- 编译预处理
  - 文件包含
  - 宏实质：编译预处理的替代
  - 带参的宏——不是函数

回顾和总结本章的教学要点，对学生提出能力要求：

- 能够对相对复杂的问题，合理定义程序的多函数结构，能合理运用外部变量、静态全局变量和静态函数。
- 建立递归思想，具备递归函数的编程能力，并解决一些特殊问题。
- 掌握宏的基本用法，特别是带参数的宏。

## 10.3 练习与习题参考答案

### 10.3.1 练习参考答案

【练习 10-1】使用递归函数计算 1 到 n 之和：若要用递归函数计算  $\text{sum}=1+2+3+\dots+n$  (n 为正整数)，请写出该递归函数的递归式子及递归出口。试编写相应程序。

解答：

```
#include <stdio.h>
int sum(int n);
int main(void)
{
    int n;
    int result;

    scanf("%d",&n);
    result=sum(n);
    printf ("%d\n", result);

    return 0;
}
int sum(int n)
{
    int result;

    if (n == 0)
        result = 0;
    else
        result = sum(n-1)+n;

    return result;
}
```

【练习 10-2】请完成以下宏定义：

1) MIN(a, b): 求 a, b 的最小值。

解答: #define MIN(a, b) (a) < (b) ? (a) : (b)

2) ISLOWER(c): 判断 c 是否为小写字母。

解答: #define ISLOWER(c) (((c) >= 'a') && ((c) <= 'z'))

3) ISLEAP(y): 判断 y 是否为闰年。

解答: #define ISLEAP(y) ((y) % 4 == 0 && (y) % 100 != 0) || ((y) % 400 == 0)

4) CIRFER(r): 计算半径为 r 的圆周长。

解答: #define PI 3.14159  
#define CIRFER(r) 2\*PI\*(r)

【练习 10-3】分别用函数和带参数的宏实现从 3 个数中找出最大数，请比较两者在形式上和使用上的区别。

解答:

(1) 使用宏实现

```
#include<stdio.h>
#define f(x,y,z) x>(y>z?y:z)?x:(y>z?y:z)
int main(void)
{
    int a,b,c;

    printf("input a, b, c: ");
    scanf("%d%d%d",&a,&b,&c);
    printf("max=%d\n",f(a,b,c));

    return 0;
}
```

(2) 使用函数实现

```
#include <stdio.h>
int max(int a, int b, int c)
{
    return ((a>b?a:b)>c? (a>b?a:b):c);
}
int main(void)
{
    int a, b, c, s;
    printf("input a, b, c: ");
    scanf("%d%d%d", &a, &b, &c);
    s = max(a,b,c);
    printf("max = %d\n", max(a,b,c));

    return 0;
}
```

## 10.3.2 习题参考答案

### 一、选择题

1	2	3	4	5	6	7
C	A	B	A	D	A	A

### 二、填空题

1、文件包含 宏定义 条件编译

2、5

3、int t0,t1,t; n>1 t0+t1 t

4、int p=1 p=p\*m int s=0 s=s+power(i,k)

5、g=4,g=3,k=6

6、7,-13

### 三、程序设计题

1. 判断满足条件的三位数：编写一个函数，利用参数传入一个3位数n，找出101~n间所有满足下列两个条件的数：它是完全平方数，又有两位数字相同，如144、676等，函数返回找出这样的数据的个数。试编写相应程序。

解答：

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

int fun(int n)
{
    int i, d=0;
    for(i=101; i<=n; i++)
        if(((int)sqrt(i) * (int)sqrt(i)) == i) {
            if (i/100==(i/10)%10|| i/100==i%10||(i/10)%10==i%10)
                d++;
        }
    return d;
}
```

2. 递归求阶乘和：输入一个整数n（ $n > 0$  且  $n \leq 10$ ），求  $1! + 2! + 3! + \dots + n!$ 。定义并调用函数 fact(n) 计算  $n!$ ，函数类型是 double。试编写相应程序。

解答:

```
double fact(int n);
int main(void)
{
    int i,n;
    double sum;

    scanf("%d",&n);
    sum=0;
    for (i=1; i<=n; i++)
        sum += fact (i);
    printf ("%0.0f\n", sum);

    return 0;
}
double fact(int n)
{
    double result;
    if (n==1 || n == 0)
        result = 1;
    else
        result = n * fact(n-1);
    return result;
}
```

3. 递归实现计算  $x^n$  : 输入实数  $x$  和正整数  $n$  , 用递归函数计算  $x^n$  的值。试编写相应程序。

解答:

```
#include <stdio.h>
double fun(double x,int n);
int main(void)
{
    int n;
    double x, root;

    scanf("%lf%d", &x,&n);
    root = fun(x, n);
    printf("Root = %0.2f\n", root);

    return 0;
}
double fun(double x, int n)
{
    if(n == 1)
        return x;
```

```

else if(n == 0)
    return 1;
else
    return x * fun(x, n-1);
}

```

4. 递归求式子和：输入实数  $x$  和正整数  $n$ ，用递归的方法对下列计算式子编写一个函数。

$$f(x, n) = x - x^2 + x^3 - x^4 + \dots + (-1)^{n-1} \times x^n \quad (n > 0)$$

试编写相应程序。

解答：

```

#include <stdio.h>
#include <math.h>
double f(double x, int n)
{
    if(n==1) return x;
    return pow(-1,n-1)*pow(x,n)+f(x, n-1);
}

int main(void)
{
    int n;
    double x;

    scanf("%lf%d", &x, &n);
    printf("%lf\n", f(x,n));
    return 0;
}

```

5. 递归计算函数 Ack(m,n)：输入两个整数  $m$  和  $n$  ( $m \geq 0$  且  $n \geq 0$ )，输出函数 Ack(m,n) 的值。  
Ack(m,n) 定义为 ( $m \geq 0$  且  $n \geq 0$ )：

$$\text{Ack}(m, n) = \begin{cases} n + 1 & m=0 \\ \text{Ack}(m - 1, 1) & n=0 \ \& \ m>0 \\ \text{Ack}(m - 1, \text{Ack}(m, n - 1)) & m>0 \ \& \ n>0 \end{cases}$$

试编写相应程序。

解答：

```

#include <stdio.h>
int Ack(int m, int n);
int main(void)
{
    int m,n;
    int result;

```

```

scanf("%d%d", &m, &n);
result = Ack(m,n);
printf("Ackerman(%d,%d)=%d\n", m, n, result);

return 0;
}
int Ack(int m, int n)
{
    if (m==0) return n+1;
    else if (n==0) return Ack(m-1, 1);
    else return Ack(m-1, Ack(m, n-1));
}

```

6. 递归实现求Fabonacci数列:用递归方法编写求斐波那契数列的函数,返回值为整型,并写出相应的主函数。斐波那契数列的定义为:

$$f(0) = 0, f(1) = 1$$

$$f(n) = f(n-2) + f(n-1) \quad (n > 1)$$

解答:

```

#include <stdio.h>
int fib(int n);
int main(void)
{
    scanf("%d",&n);
    printf("fib(%d)=%d\n",n,fib(n));

    return 0;
}
int fib(int n)
{
    int res;
    if(n==0)
        res=0;
    else if(n==1)
        res=1;
    else
        res=fib(n-2)+fib(n-1);
    return res;
}

```

7. 递归实现十进制转二进制:输入一个正整数 n,将其转换为二进制后输出。要求定义并调用函数 dectobin(n),它的功能是输出 n 的二进制。试编写相应程序。

解答:

```

#include "stdio.h"

```

```

int main(void)
{
    int i,n;
    void dectobin(int n);

    scanf("%d",&n);
    dectobin(n);

    return 0;
}
void dectobin(int n)
{
    if(n<2)
        printf("%d", n);
    else{
        dectobin(n/2);
        printf("%d", n%2);
    }
}

```

8. 递归实现顺序输出整数：输入一个正整数  $n$ ，编写递归函数实现对其进行按位顺序输出。试编写相应程序。

解答：

```

#include <stdio.h>
void inorder(int n);
int main(void)
{
    int n;
    scanf("%d",&n);
    inorder(n);

    return 0;
}
void inorder(int num)
{
    if(num < 10)
        printf("%d\n",num);
    else
    {
        inorder(num/10);
        printf("%d\n",num%10);
    }
}

```



9. 使用文件包含统计素数：输入  $n(n < 10)$  个整数,统计其中素数的个数。要求程序由两个文件组成，一个文件中编写 `main` 函数，另一个文件中编写素数判断的函数。使用文件包含的方式实现。

解答：

文件 `prog.cpp`

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

```
#include "prime.h"
```

```
int main(void)
```

```
{
```

```
    int i,n,count,x;
```

```
    count=0;
```

```
    scanf("%d",&n);
```

```
    for(i=1;i<=n;i++)
```

```
    {
```

```
        scanf("%d",&x);
```

```
        if(IsPrime(x)) count++;
```

```
    }
```

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

```
    return 0;
```

```
}
```

文件 `prime.h`

```
int IsPrime(int m)
```

```
{
```

```
    int i,n;
```

```
    if(m==1) return 0;
```

```
    n=m/2;
```

```
    for(i=2;i<=n;i++)
```

```
        if(m%i==0)
```

```
            return 0;
```

```
    return 1;
```

```
}
```

10. 三角形面积为

$$\text{area} = \sqrt{s \times (s - a) \times (s - b) \times (s - c)} \quad \text{其中, } s = (a + b + c) / 2$$

$a$ 、 $b$ 、 $c$  分别为三角形的 3 条边。请分别定义计算  $s$  和  $\text{area}$  的宏。再使用函数实现，比较两者在形式上和使用上的区别。

解答：

(1) 使用宏实现

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

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

```
#define S(a ,b, c) ((a)+(b)+(c))/2
```

```

#define AREA(s,a,b,c) sqrt((s)*((s)-(a))*((s)-(b))*((s)-(c)))
int main(void)
{
    double a, b, c, s;
    printf("input a, b, c: ");
    scanf("%lf%lf%lf", &a, &b, &c);
    s = S(a,b,c);
    printf("s = %lf, area = %lf\n", s, AREA(s,a,b,c));
    return 0;
}

```

## (2) 使用函数实现

```

#include <stdio.h>
#include <math.h>
double f1(double a, double b, double c)
{
    return (a+b+c)/2;
}

double f2(double s, double a, double b, double c)
{
    return sqrt(s*(s-a)*(s-b)*(s-c));
}

int main(void){
    double a, b, c, s;
    printf("input a, b, c: ");
    scanf("%lf%lf%lf", &a, &b, &c);
    s = f1(a,b,c);
    printf("s = %lf, area = %lf\n", s, f2(s,a,b,c));
    return 0;
}

```

## 10.4 实验指导教材参考答案

### 一、调试示例（略）

### 二、基础编程题

(1) 判断满足条件的三位数：编写一个函数，利用参数传入一个 3 位数 `number`，找出 `101~number` 之间所有满足下列两个条件的数：它是完全平方数，又有两位数字相同，如 144、676 等，函数返回找出这样的数据的个数，并编写主函数。

输入输出示例（括号内为说明文字）

150                      (number=150)

count=2

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 1 题。

(2) 递归求阶乘和：输入一个整数  $n(n>0$  且  $n\leq 10)$ ，求  $1!+2!+3!+\dots+n!$ 。定义并调用函数  $\text{fact}(n)$  计算  $n!$ ，函数类型是 `double`。试编写相应程序。

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 2 题。

(3) 递归实现计算  $x^n$ ：输入双精度浮点数  $x$  和整数  $n(n\geq 1)$ ，调用函数求  $x$  的  $n$  次方，并保留两位小数。

输入输出示例（括号内为说明文字）

2 (x=2)

3 (n=3)

Root = 8.00

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 3 题。

(4) 递归求式子和：用递归的方法对下列计算式子编写一个函数，并写出相应主函数，计算结果保留两位小数。

$$f(x, n) = x - x^2 + x^3 - x^4 + \dots + (-1)^{n-1} \times x^n \quad (n > 0)$$

输入输出示例（括号内为说明文字）

2 (x=2)

4 (n=4)

f(2,4) = -10

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 4 题。

(5) 递归计算函数  $\text{Ack}(m, n)$ ：输入两个整数  $m$  和  $n(m\geq 0$  且  $n\geq 0)$ ，输出函数  $\text{Ack}(m, n)$  的值。 $\text{Ack}(m, n)$  定义为( $m\geq 0$  且  $n\geq 0$ )：

$$\text{Ack}(m, n) = \begin{cases} n + 1 & m = 0 \\ \text{Ack}(m - 1, 1) & n = 0 \ \& \ m > 0 \\ \text{Ack}(m - 1, \text{Ack}(m, n - 1)) & m > 0 \ \& \ n > 0 \end{cases}$$

输入输出示例（括号内为说明文字）

2 (m=2)

3 (n=3)

Ack(2, 3)=9

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 5 题。

(6) 递归实现求 **Fabonacci** 数列：用递归方法编写求斐波那契数列的函数，返回值为整型，并写出相应的主函数。斐波那契数列的定义为：

$$f(0) = 0, f(1) = 1$$

$$f(n) = f(n - 2) + f(n - 1) \quad (n > 1)$$

输入输出示例（括号内为说明文字）

6 (n=6)

fib(6)=8

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 6 题。

### 三、改错题

改正下列程序中的错误，输入一个整数  $n$  ( $n \geq 0$ ) 和一个双精度浮点数  $x$ ，输出函数  $P(n,x)$  的值（保留 2 位小数）。（源程序 error10\_2.cpp）

$$P(n, x) = \begin{cases} 1 & (n=0) \\ x & (n=1) \\ ((2*n-1)*P(n-1,x)-(n-1)*P(n-2,x))/n & (n>1) \end{cases}$$

输入输出示例

Enter n, x: 10 1.7

P(10,1.70) = 3.05

源程序（有错误的程序）

```
1    #include <stdio.h>
2    int main(void)
3    {
4        int n;
5        double x, result;
6
7        printf("Enter n, x: ");
8        scanf("%d%lf", &n, &x);
9        result = p(n,x);
10       printf("P(%d,%.2lf) = %.2lf\n", n, x, result);
11
12       return 0;
13   }
14
15   double p(int n, double x)
16   {
17       p(n,x) = ((2 * n - 1) * p(n - 1,x) - (n - 1) * p(n - 2,x)) / n;
18
19       return p(n,x);
20   }
```

(1) 初次编译后共有 2 个[Error]，请填写出错信息并分析原因。

① 'p' undeclared (first use this function)，函数 **p** 未声明

② non-lvalue in assignment 赋值语句中左边缺少变量

(2) 请填写修改后的正确语句。

改错汇总：

错误行号： 6 正确语句： double p(int n, double x);

错误行号： 16 正确语句： double result;

错误行号： 17 正确语句： result = ((2 \* n - 1) \* p(n - 1,x) - (n - 1) \* p(n - 2,x)) / n;

错误行号： 19 正确语句： return result;

(3) 改正上述编译错误后，再次编译无错误出现，运行程序。

运行输入测试数据为 10 1.7，运行结果为： 程序错误，无输出，是否正确： 否

(4) 请仔细分析错误产生的原因，模仿以前调试示例中的方法进行调试改错。并简要说明你的方法并给出正确语句。

方法： 略

改错汇总：（以下为正确程序段）

```
double p(int n, double x)
{
    double result;

    if(n==0)
        result=1;
    else if(n==1)
        result=x;
    else
        result = ((2 * n - 1) * p(n - 1,x) - (n - 1) * p(n - 2,x)) / n;

    return result;
}
```

#### 四、拓展编程题

(1) 十进制转二进制：输入一个正整数  $n$ ，将其转换为二进制后输出。要求定义并调用函数 `dectobin(n)`，它的功能是输出  $n$  的二进制。例如，调用 `dectobin(10)`，输出 1010。

输入输出示例（括号内为说明文字）

100                    (n=100)  
1100100

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 7 题。

(2) 递归实现顺序输出整数：输入一个正整数  $n$ ，编写递归函数实现对其进行按位顺序输出。试编写相应程序。

输入输出示例（括号内为说明文字）

900                    (n=900)  
9  
0  
0

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 8 题。

(3) 递归实现逆序输出整数：编写实现对一个整数进行逆序处理的递归函数，函数需要有返回值，其值为逆序后的数据。

输入输出示例（括号内为说明文字）

第 1 次运行：（总共运行 2 次）

567                    (n=567)  
reverse=765

第 2 次运行：

800                    (n=800)  
reverse=8

解答:

```
#include <stdio.h>
int reverse(int n);
int main(void)
{
    int n;

    scanf("%d", &n);
    printf("reverse = %d\n", reverse(n));

    return 0;
}
int reverse(int n)
{
    static int p;
    int y;

    if (n<10){
        p = 1; y = n;
    }
    else {
        y = reverse(n / 10);
        p *= 10;
        y = y + (n % 10) * p;
    }

    return y;
}
```

(4) 使用文件包含统计素数: 输入  $n(n<10)$ 个整数,统计其中素数的个数。要求程序由两个文件组成,一个文件中编写 `main` 函数,另一个文件中编写素数判断的函数。使用文件包含的方式实现。

输入输出示例 (括号内为说明文字)

```
5          (n=5)
3  6  7  9  11
count=3
```

解答: 参见《C 语言程序设计》习题 10 中的三、程序设计题,第 9 题。

(5) 使用文件包含编制简单加减法计算器: 编制一个简单加减运算的计算器,输入计算式子的格式为: 整数常量+运算符+整数常量。

输入输出示例

```
5+10
5+10=15
```

要求程序由两个文件组成,把加减运算写成函数: `int Add(int a, int b), int Sub(int a, int b),`

并单独写成一个源程序文件 `cal.c`，分别使用文件包含和工程文件与主函数的源程序进行连接。

解答：

文件 `cal.c`

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

    scanf("%d%c%d", &operand1, &op, &operand2);
    switch(op){
        case '+':
            printf("%d+%d=%d\n", operand1, operand2, Add(operand1, operand2)); break;
        case '-':
            printf("%d-%d=%d\n", operand1, operand2, Sub(operand1, operand2)); break;
        default: printf("运算符错误\n"); break;
    }

    return 0;
}
```

文件 `add.h`

```
int Add(int a, int b)
{
```

```
    return (a+b);
```

```
}
```

文件 `sub.h`

```
int Sub(int a, int b)
```

```
{
```

```
    return (a-b);
```

```
}
```

(6) 三角形面积为：

$$\text{area} = \sqrt{s \times (s - a) \times (s - b) \times (s - c)} \quad s = (a + b + c) / 2$$

其中 `a`、`b`、`c` 分别是三角形的 3 条边。请分别定义计算 `s` 和 `area` 的宏。再使用函数实现，比较两者在形式上和使用上的区别。

解答：参见《C 语言程序设计》习题 10 中的三、程序设计题，第 10 题。