

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 节通过有序表的操作，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。

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。本节另一个重点是介绍当多个函数位于不同文件模块上的解决方法——文件包含，以及变量与多文件模块的关系。本节内容看上去比较散，但教师在教学时要抓住复杂问题需要多个函数解决这条主线。

10.4 节通过学生信息库系统实例介绍如何使用多文件模块构建较大规模程序，以及如何通过外部变量和静态全局变量实现模块之间的通信。

讲授学时：6 学时，实验：4 学时。

本章的知识能力结构图见图 10.1。

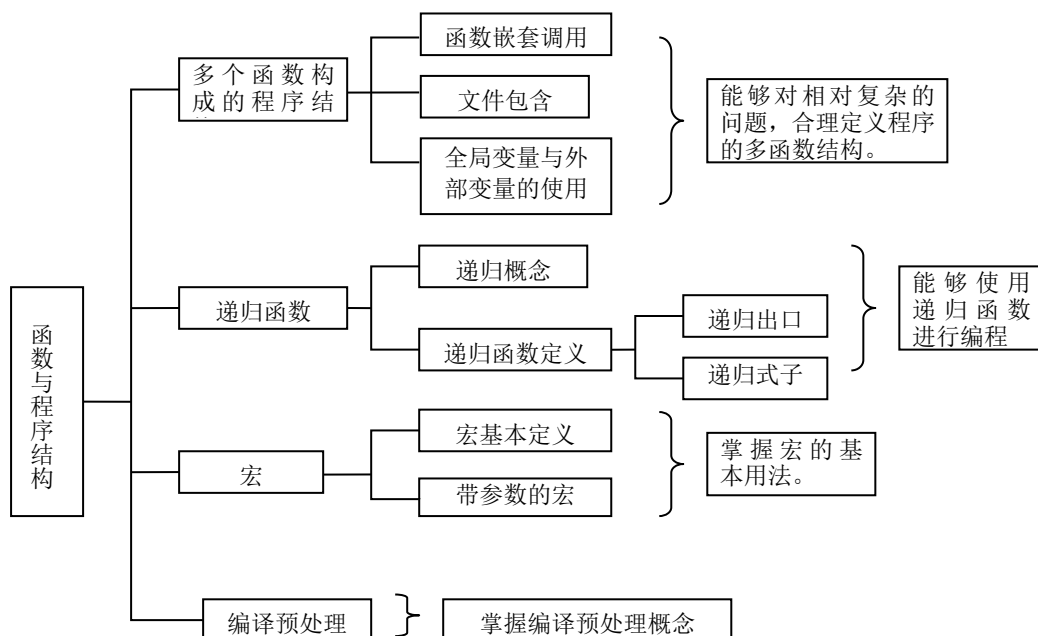




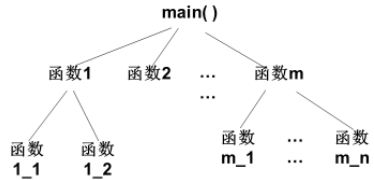
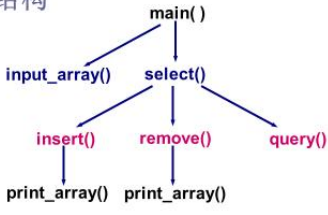






图 10.1 知识能力结构图





10.2 讲稿

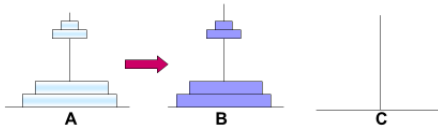
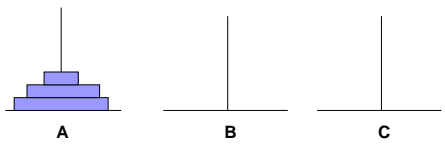
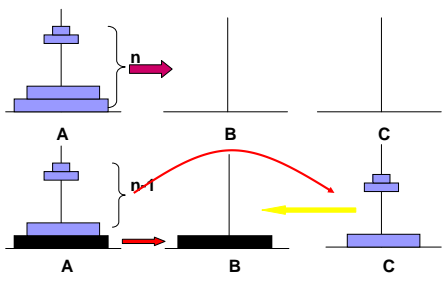
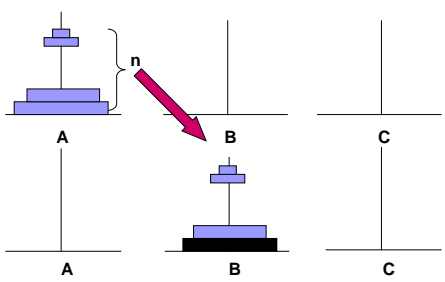
1	 Chap 10 函数与程序结构 10.1 有序表的操作 10.2 汉诺塔问题 10.3 长度单位转换 10.4 大程序构成	本章分 4 节。
2	 本章要点 <ul style="list-style-type: none">■ 怎样把多个函数组织起来？■ 怎样用结构化程序设计的思想解决问题？■ 怎样用函数嵌套求解复杂的问题？■ 怎样用函数递归解决问题？■ 如何使用宏？	提出本章的学习要点。
3	 10.1 有序表操作 使用结构化程序设计方法解决复杂的问题 <ul style="list-style-type: none">□ 把大问题分解成若干小问题，小问题再进一步分解成若干更小的问题□ 写程序时，用main()解决整个问题，它调用解决小问题的函数□ 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	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。
4	 程序结构 	层次（树形）结构描述结构化思想。

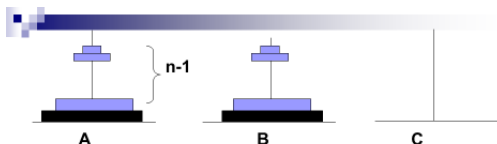
5	<p>10.1.1 程序解析-有序表操作</p> <ul style="list-style-type: none"> ■ 例10-1 首先输入一个无重复元素的、从小到大排列的有序表，并在屏幕上显示以下菜单（编号和选项），用户可以反复对该有序表进行插入、删除和查找操作，也可以选择结束。当用户输入编号1~3和相关参数时，将分别对该有序表进行插入、删除和查找操作，输入其他编号，则结束操作。 <p>[1] Insert [2] Delete [3] Query [Other option] End</p>	介绍例 10-1 的功能需求
6	<p>例10-1 分析</p> <ul style="list-style-type: none"> ■ 输入有序表； ■ 输入1、2、3选择插入、删除、查找操作，其他输入结束； <ul style="list-style-type: none"> □ 设计一个控制函数select()，经它辨别用户输入的编号后，调用相应的插入、删除和查找函数，再调用有序表输出函数显示结果 ■ 分别设计函数实现有序表的插入、删除、查找和输入、输出等常规操作 	分析需求，介绍设计思路
7	<p>例10-1程序结构</p>  <pre> graph TD main["main()"] --> input["input_array()"] main --> select["select()"] select --> insert["insert()"] select --> remove["remove()"] select --> query["query()"] insert --> print1["print_array()"] remove --> print2["print_array()"] </pre> <p>4层结构，7个函数 降低程序的构思、编写、调试的复杂度，可读性好</p>	详细介绍程序的函数结构
8	<p>例10-1源程序</p> <pre> int Count = 0; void select(int a[], int option, int value); void input_array(int a[]); void print_array(int a[]); void insert(int a[], int value); void remove(int a[], int value); void query(int a[], int value); int main(void) { int option, value, a[100]; input_array(a); /* 调用函数输入有序数组 a */ printf("[1] Insert\n"); printf("[2] Delete\n"); printf("[3] Query\n"); printf("[Other option] End\n"); while (1) { printf("Input option: "); scanf("%d", &option); if (option < 1 option > 3) {break;} printf("Input an element: "); scanf("%d", &value); select(a, option, value); /* 调用控制函数 */ printf("\n"); } printf("Thanks."); 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。</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>

9	 <h3>select()</h3> <pre> /* 控制函数 */ void select(int a[], int option, int value) { switch (option) { case 1: insert(a, value); /* 调用插入函数在有序数组 a 中插入元素 value */ break; case 2: remove(a, value); /* 调用删除函数在有序数组 a 中删除元素 value */ break; case 3: query(a, value); /* 调用查询函数在有序数组 a 中查找元素 value */ break; } } </pre>	函数 select()通过对 option 值的判断,分别调用函数 insert()、remove()、query()
10	 <h3>input_array() print_array()</h3> <pre> /* 有序表输入函数 */ void input_array(int a[]) { printf("Input the number of array elements: "); scanf("%d", &Count); printf("Input an ordered array element: "); for (int i = 0; i < Count; i++) { scanf("%d", &a[i]); } } /* 有序表输出函数 */ void print_array(int a[]) { printf("The ordered array a is: "); for (int i = 0; i < Count; i++) { /* 相邻数字间空格分隔, 行末无空格 */ if (i == 0) { printf("%d", a[i]); } else { printf(" %d", a[i]); } } } </pre>	有序表输入、输出操作
11	 <h3>insert()</h3> <pre> /* 有序表插入函数 */ void insert(int a[], int value) { int i, j; for (i = 0; i < Count; i++) { /* 定位: 待插入的位置即退出循环时 i 的值 */ if (value < a[i]) { break; } } for (j = Count - 1; j >= i; j--) { /* 腾位: 将 a[i]~a[Count-1] 向后顺移一位 */ a[j+1] = a[j]; } a[i] = value; /* 插入: 将 value 的值赋给 a[i] */ Count++; /* 增1: 数组 a 中待处理的元素数量增1 */ print_array(a); /* 调用输出函数, 输出插入后的有序数组 a */ } </pre>	在有序表中插入一个元素: 先找插入点, 再移位, 最后插入
12	 <h3>remove()</h3> <pre> /* 有序表删除函数 */ void remove(int a[], int value) { int i, index = -1; /* index 的值为 -1 表示没找到, 否则表示找到 */ for (i = 0; i < Count; i++) { /* 定位: 找到则用 index 记录其下标 */ if (value == a[i]) { index = i; break; } } if (index == -1) { /* 没找到, 则输出相应的信息 */ printf("Failed to find the data, deletion failed."); } else { /* 找到, 则删除 a[index] */ for (i = index; i < Count - 1; i++) { a[i] = a[i+1]; /* 将 a[Count-1]~a[index+1] 向前顺移一位 */ } Count--; /* 减1: 数组 a 中待处理的元素数量减1 */ print_array(a); /* 调用输出函数, 输出删除后的有序数组 a */ } } </pre>	在有序表中删除一个元素: 先查找删除元素位置, 再通过移位实现删除

13	<div data-bbox="304 203 783 584" data-label="Text"> <pre> /* 有序表二分法查询函数 */ void query(int a[], int value) { int mid, left = 0, right = Count - 1; /* 开始时查找区间为整个数组 */ while (left <= right) { mid = (left + right) / 2; /* 得到中间位置 */ if (value == a[mid]) { /* 查找成功, 输出下标, 函数返回 */ printf("The index is: %d", mid); return; } else if (value < a[mid]) { /* 缩小查找区间为前半段, right前移 */ right = mid - 1; } else { /* 缩小查找区间为后半段, left后移 */ left = mid + 1; } } printf("This element does not exist."); /* value不在数组a中 */ } </pre> </div> <div data-bbox="671 226 767 259" data-label="Text"> <p>query()</p> </div>
----	--

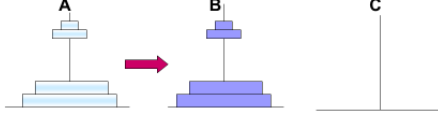

17	 <h3>10.1.2 函数的嵌套调用</h3> <ul style="list-style-type: none"> ■ 在一个函数中再调用其它函数的情况称为函数的嵌套调用。 <ul style="list-style-type: none"> □ 如果函数A调用函数B，函数B再调用函数C，一个调用一个地嵌套下去，构成了函数的嵌套调用。 □ 具有嵌套调用函数的程序，需要分别定义多个不同的函数体，每个函数体完成不同的功能，它们合起来解决复杂的问题。 	结构化的程序设计方法一般都存在函数的嵌套调用结构，用以解决复杂的问题。
18	 <h3>结构化程序设计方法</h3> <ul style="list-style-type: none"> ■ 自顶向下，逐步求精，函数实现 <ul style="list-style-type: none"> □ 自顶向下 <ul style="list-style-type: none"> ■ 先考虑全局目标，后考虑局部目标； ■ 先考虑总体步骤，后考虑步骤的细节； ■ 先从最上层总目标开始设计，逐步使问题具体化。 □ 逐步求精 <ul style="list-style-type: none"> ■ 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。 □ 函数实现 <ul style="list-style-type: none"> ■ 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。 	介绍结构化程序设计的基本思想
19	 <h3>函数设计时应注意的问题</h3> <ul style="list-style-type: none"> ■ 限制函数的长度 <ul style="list-style-type: none"> □ 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。 ■ 避免函数功能间的重复 <ul style="list-style-type: none"> □ 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。 ■ 减少全局变量的使用 <ul style="list-style-type: none"> □ 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。 	
20	 <h2>10.2 汉诺塔问题</h2> <h3>10.2.1 程序解析</h3> <h3>10.2.2 递归函数基本概念</h3> <h3>10.2.3 递归程序设计</h3>	结合示例介绍递归函数的使用方法。

21	<p>10.2.1 汉诺(Hanoi)塔问题解析</p>  <p>将 64 个盘从座A搬到座B</p> <p>(1) 一次只能搬一个盘子 (2) 盘子只能插在A、B、C三个杆中 (3) 大盘不能压在小盘上</p>	<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"> 1、 能实现非递归很难实现的问题 2、 程序简洁 <p>难点：</p> <p>递归思想不容易建立。</p>
22	<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>
23	<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>
24	<p>分析</p> 	<p>演示</p>




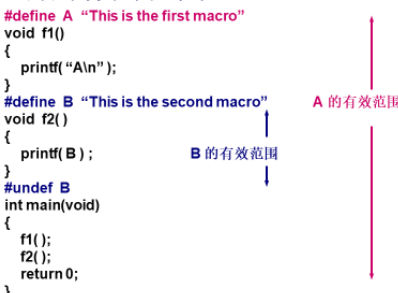

25	<div><div></div><h3>10.2.1 汉诺(Hanoi)塔问题解析</h3><ul style="list-style-type: none">■ 递归方法的两个要点<ul style="list-style-type: none">□ 递归出口：一个盘子的解决方法；□ 递归式子：如何把搬动64个盘子的的问题简化成搬动63个盘子的的问题。■ 把汉诺塔的递归解法归纳成三个步骤：<ul style="list-style-type: none">□ n-1个盘子从座A搬到座C□ 第n号盘子从座A搬到座B□ n-1个盘子从座C搬到座B</div>	分析递归出口和总结归纳递归式是使用递归方法解决问题的两个要点						
26	<div><div></div><p>算法：</p><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) 把第n号盘 A→B hanio(n-1个盘, C→B, A为过渡) } }</pre></div>	<p>分析算法：</p> <ul style="list-style-type: none">● n 个盘子的的问题 → n-1 个盘子的的问题● n=1 时问题直接解决（出口） <p>这就是递归 2 要素。</p>						
27	<div><div></div><h3>10.2.2 递归函数基本概念</h3><table border="1"><thead><tr><th>函数直接递归调用^①</th><th colspan="2">函数间接递归调用^②</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^①</p></div>	函数直接递归调用 ^①	函数间接递归调用 ^②		<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">■ 直接调用自己■ 间接调用自己
函数直接递归调用 ^①	函数间接递归调用 ^②							
<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>						
28	<div><div><div></div><h3>10.2.2递归函数基本概念</h3><ul style="list-style-type: none">■ 例10-2 用递归函数实现求n！<ul style="list-style-type: none">□ 递推法<ul style="list-style-type: none">■ 在学习循环时，计算n! 采用的就是递推法： n! = 1×2×3×...×n■ 用循环语句实现： result = 1; for(i = 1; i <= n; i++){ result = result * i; }□ 递归法<ul style="list-style-type: none">■ n! = n × (n-1)! 当n>1 递归式子■ = 1 当n=1或n=0 递归出口■ 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)</div></div>	<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">■ 自己调用自己■ 有 n=0 n=1 的特殊情况考虑						

29	<div data-bbox="304 203 767 584"> <div>例10-2 源程序</div> <pre> #include <stdio.h> double fact(int n); int main(void) { int n; scanf("%d", &n); printf("%f", 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> </div>	运行演示程序
30	<div data-bbox="304 620 807 1001"> <div>例10-2分析</div> <pre> #include <stdio.h> double fact(int n); int main(void) { int n; scanf("%d", &n); printf("%f", fact(n)); return 0; } double fact(int n) { double result; if (n==1 n==0){ result = 1; }else{ result = n * fact(n-1); } return result; } </pre> <div> <div>求n! 递归定义</div> <div> $n! = n * (n-1)!$ ($n > 1$) $n! = 1$ ($n = 0, 1$) </div> </div> <div> <div>递归出口</div> <div>递归式</div> <div>$fact(n) = n * fact(n-1);$</div> </div> </div>	<p>递归函数的 2 要素：</p> <ul style="list-style-type: none"> ■ 递归出口 ■ 递归式子（表示） <p>fact(n)=n*fact(n-1)不符合 C 语言的语法规则，赋值号=左面只能是变量，表示保存单元。而 fact(n)是函数形式，表示函数计算，最终是数值。</p>
31	<div data-bbox="304 1037 807 1417"> <div>递归函数 fact(n)的实现过程</div> <div> $fact(3) = 3 * fact(2) = 3 * 2 = 6$ </div> <div>同时有4个函数在运行，且都未完成</div> <div> $2 * fact(1) = 2 * 1 = 2$ </div> <div> $fact(1) = 1$ </div> <div> <pre> main() { ... printf(fact(3)) ... } fact(3) { ... f=3*fact(2) ... return(f) } fact(2) { ... f=2*fact(1) ... return(f) } fact(1) { ... f=1 ... return(f) } </pre> </div> </div>	<p>动态演示递归函数的调用与返回过程，从而了解递归函数的计算过程。</p> <p>递归函数的运算远比递推法的循环来的复杂，像例子中，main 调用 fact（3），最后有 4 个函数同时打开，然后再一层层返回，关闭函数，最终回到 main。</p>
32	<div data-bbox="304 1453 807 1848"> <div>10.2.3 递归程序设计</div> <p>用递归实现的问题，满足两个条件：</p> <ul style="list-style-type: none"> ■ 问题可以逐步简化成自身较简单的形式（递归式） <div> $n! = n * (n-1)!$ $\sum_{i=1}^n i = n + \sum_{i=1}^{n-1} i$ </div> <ul style="list-style-type: none"> ■ 递归最终能结束(递归出口) <p>两个条件缺一不可 解决递归问题的两个着眼点</p> </div>	<p>小结递归函数的 2 要素：递归出口和递归式子。</p> <p>举几个例子讨论递归出口和递归式子分别是什么。</p> <p>$n! = n * (n-1)!$</p> <p>$\sum i = n + \sum i$</p> <p>等等。</p>

33	<div> <div></div> <div>10.2.3 递归程序设计</div> </div> <ul style="list-style-type: none"> ■ 例10-3 定义函数gcd(m, n)，用递归求m和n的最大公约数。 <ul style="list-style-type: none"> □ 辗转相除法（欧几里得算法） <ol style="list-style-type: none"> (1) $r = m \% n$ (2) 若r为0，则返回n的值；否则转第(3)步 (3) $m = n, n = r$, 返回第(1)步 □ 递归实现的两个关键点： <ul style="list-style-type: none"> ■ 递归出口：n 当 $m \% n = 0$ ■ 递归式子：递归调用gcd(n, m%n) 当 $m \% n != 0$ 	用递归方法求最大公约数：重点介绍递归出口与递归式
34	<div> <div></div> <div>例10-3 源程序</div> </div> <pre>int gcd(int m, int n) { if(m % n == 0){ /* 递归出口 */ return n; }else{ return gcd(n, m % n); /* 递归调用 */ } }</pre>	函数实现
35	<div> <div></div> <div>例10-4 递归实现将整数逆序输出</div> </div> <ul style="list-style-type: none"> ■ 编写递归函数reverse(int n)实现将整数n逆序输出。 <p>分析：</p> <ul style="list-style-type: none"> □ 将整数n逆序输出可以用循环实现，且循环次数与n的位数有关。 □ 递归实现整数逆序输出也需要用位数作为控制点。归纳递归实现的两个关键点如下： <ul style="list-style-type: none"> ■ 递归出口：直接输出n，如果$n \leq 9$，即n为1位数 ■ 递归式子：输出个位数$n \% 10$，再递归调用reverse(n/10) 输出前n-1位，如果n为多位数 	重点介绍将 n 拆分成两部分： $n \% 10$ 、 $n / 10$ 的思路，对 $n / 10$ 继续递归
36	<div> <div></div> <div>例10-4 源程序</div> </div> <ul style="list-style-type: none"> ■ 由于直接输出结果，因此函数返回类型为void <pre>void reverse(int num) { if (num <= 9){ printf ("%d", num); /* 递归出口 */ }else { printf ("%d", num % 10); reverse (num / 10); /* 递归调用 */ } }</pre>	运行演示程序

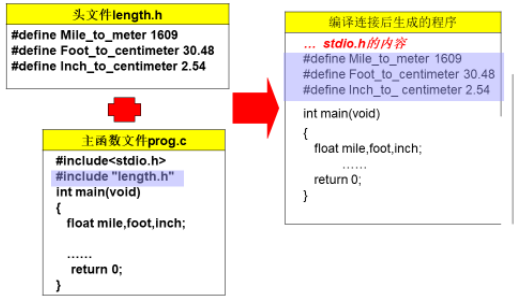
37	<p>例10-5 汉诺(Hanoi)塔问题</p>  <pre> hanio(n个盘, A→B, C为过渡) { if (n == 1) 直接把盘子A→B else{ hanio(n-1个盘, A→C, B为过渡) 把n号盘A→B hanio(n-1个盘, C→B, A为过渡) } } </pre>	回到汉诺塔问题，回顾其相对复杂的二路递归思路
38	<p>源程序</p> <pre> /* 搬动n个盘，从a到b，c为中间过渡 */ void hanio(int n, char a, char b, char c) { if (n == 1){ printf("%c-->%c\n", a, b); }else{ hanio(n-1, a, c, b); printf("%c-->%c\n", a, b); hanio(n-1, c, b, a); } } int main(void) { int n; printf("input the number of disk: "); scanf("%d", &n); printf("the steps for %d disk are:\n", n); hanio(n, 'a', 'b', 'c'); return 0; } </pre> <p>input the number of disk: 3 the steps for 3 disk are: a-->b a-->c b-->c a-->b c-->a c-->b a-->b</p>	运行演示程序
39	<p>input the number of disk: 3 the steps for 3 disk are: a-->b a-->c b-->c a-->b c-->a c-->b a-->b</p> 	判断其结果是否正确
40	<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>	让学生现场编程练习

41	<div> <div></div> <div> <h3>例10-6 分治法求解金块问题</h3> <ul style="list-style-type: none"> 老板有一袋金块，两名最优秀的雇员每人可以得到其中的一块，排名第一的得到最重的金块，排名第二的则得到袋子中最轻的金块。 输入n（共n块，$2 \leq n \leq 100$）及n个整数，用分治法求出最重金块和最轻金块。 <ul style="list-style-type: none"> 定义递归函数max(int a[], int m, int n)，在a[m]~a[n]中找出最大值 <ul style="list-style-type: none"> 递归出口：a[m]，当m == n，即a中只有1个元素 递归式：将数组a分割为两部分，分别递归求最大值 <ul style="list-style-type: none"> k = (m + n) / 2; u = max(a, m, k); v = max(a, k+1, n); </div> </div>	分治法与递归方法相结合的示例 介绍分治法的基本思想
42	<div> <div></div> <div> <h3>例10-6 源程序</h3> <pre> /* 分治法求a[m]~a[n]中最大值的递归函数 */ int max(int a[], int m, int n) { int k, u, v; if (m == n) { return a[m]; } k = (m + n) / 2; /* 计算中间元素的下标k */ u = max(a, m, k); /* 在a[m]~a[k]中找出最大值 */ v = max(a, k+1, n); /* 在a[k+1]~a[n]中找出最大值 */ return (u > v) ? u : v; /* 返回u和v中较大的值 */ } </pre> </div> </div>	运行演示程序
43	<div> <div></div> <div> <h2>10.3 长度单位转换</h2> <ul style="list-style-type: none"> 10.3.1 程序解析 10.3.2 宏基本定义 10.3.3 带参数的宏定义 10.3.4 文件包含 10.3.5 编译预处理 </div> </div>	
44	<div> <div> <div></div> <div> <h3>10.3.1 程序解析</h3> <p>例10-7 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<stdio.h> #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",&mile,&foot,&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> </div> </div> </div>	程序中，定义了三个宏，也叫符号常量，主函数中出现这些宏的位置，都用它们对应的值替换

45	<div>  <h3>10.3.2 宏基本定义</h3> <p>#define 宏名标识符 宏定义字符串 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"> □ 宏名一般用大写字母，以与变量名区别 □ 宏定义不是C语句，后面不得跟分号 □ 宏定义可以嵌套使用 <pre>#define PI 3.14 #define S 2*PI*PI</pre> <p>多用于符号常量</p> </div>	<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，后面不用加；</p>
46	<div>  <h3>10.3.2 宏基本定义</h3> <ul style="list-style-type: none"> ■ 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。 ■ 可以通过 “#undef”强制指定宏的结束范围。 </div>	与变量相同，宏定义也有作用范围
47	<div>  <h3>宏的作用范围</h3> <pre>#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>  </div>	<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。</p>
48	<div>  <h3>10.3.3 带参数的宏定义</h3> <p>例10-8 简单的带参数的宏定义。</p> <pre>#include <stdio.h> #define MAX(a,b) a > b ? a : b #define SQR(x) x * x int main(void) { int x, y; scanf ("%d%d", &x, &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> </div>	通过例子说明带参数的宏定义的使用方法。





49	<p>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$1^3+5^3+3^3=153$</p> <h3>10.3.3 带参数的宏</h3> <p>例: <code>#define f(a) (a)*(a)*(a)</code></p> <pre>int main (void) /* 水仙花数 */ { int i, x, y, z; for (i = 1; i < 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>$f(x+y) = (x+y)^3 ?$ $= x+y*x+y*x+y$</p>	使用带参数的宏时，必须先替换，再计算
50	<p>示例 用宏实现两个变量值的交换</p> <pre># define f(a,b,t) t=a; a=b; b=t; int main () { int x, y, t; scanf ("%d%d", &x, &y); t=x ; x=y ; 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"> 带参数的宏定义不是函数，宏与函数是两种不同的概念 宏可以实现简单的函数功能 	<p>讨论宏与函数的异同点：</p> <p>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>
51	<p>宏定义应用示例</p> <ul style="list-style-type: none"> 定义宏LOWCASE，判断字符c是否为小写字母。 <code>#define LOWCASE(c) (((c) >= 'a') && ((c) <= 'z'))</code> 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。 <code>#define CTOD(c) (((c) >= '0') && ((c) <= '9') ? c - '0' : -1)</code> 	注意（）的使用
52	<p>练习——带宏定义的程序输出</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>	课堂练习




53	<p>结果分析</p> <ul style="list-style-type: none"> ■ 阅读带宏定义的程序，先全部替换好，最后再统一计算 ■ 不可一边替换一边计算，更不可以人为添加括号 <p>$D(3) = x * F(x)$ 先用 x 替换展开 $= x * x - 2$ 进一步对 $F(x)$ 展开，这里不能加括号 $= 3 * 3 - 2 = 7$ 最后把 $x=3$ 代进去计算</p> <p>$D(D(3)) = D(x * x - 2)$ 先对 $D(3)$ 用 x 替换展开， $= x * x - 2 * F(x * x - 2)$ 拿展开后的参数对 D 进一步进行宏替换 $= x * x - 2 * x * x - 2 - 2$ 拿展开后的参数对 F 进一步进行宏替换 $= 3 * 3 - 2 * 3 * 3 - 2 - 2 = -13$ 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>
54	<p>10.3.4 文件包含</p> <ul style="list-style-type: none"> ■ 系统文件以 stdio.h、math.h 等形式供编程者调用 ■ 实用系统往往有自己诸多的宏定义，也以 .h 的形式组织、调用 ■ 问题：如何把若干 .h 头文件连接成一个完整的可执行程序？ <ul style="list-style-type: none"> □ 文件包含 include 	<p>文件包含是解决多个文件模块程序连接的有效方法。</p>
55	<p>文件包含</p> <ul style="list-style-type: none"> ■ 格式 <ul style="list-style-type: none"> □ #include <需包含的文件名> 系统文件夹 □ #include "需包含的文件名" 当前文件夹+系统文件夹 ■ 作用 <p>把指定的文件模块内容插入到 #include 所在的位置，当程序编译连接时，系统会把所有 #include 指定的文件拼接生成可执行代码。</p> ■ 注意 <ul style="list-style-type: none"> □ 编译预处理命令，以 # 开头。 □ 在程序编译时起作用，不是真正的 C 语句，行尾没有分号。 	<p>区分 include 的两种写法，一般：</p> <ul style="list-style-type: none"> ● <...> 适用于系统头文件 ● "..." 适用于多文件模块连接
56	<p>例10-9 将例10-7中长度转换的宏，定义成头文件 length.h，并写出主函数文件。</p> <p>头文件length.h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prog.c源程序</p> <pre>#include <stdio.h> #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", &mile, &foot, &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>	<p>文件包含示例</p>

57	 <p>头文件length.h</p> <pre>#define Mile_to_meter 1609 #define Foot_to_centimeter 30.48 #define Inch_to_centimeter 2.54</pre> <p>主函数文件prog.c</p> <pre>#include<stdio.h> #include "length.h" int main(void) { float mile,foot,inch; return 0; }</pre> <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>	<p>文件包含的作用：</p> <p>在编译时，把需包含的文件插入到 include 位置，插入后的程序规模将扩大。</p> <p>可进一步引申到系统头文件，例如每个程序都会用到的 include <stdio.h>，其作用将把系统头文件 stdio.h 的所有程序插入到我们编写的程序前，来确保 printf 等功能的实现。</p>
58	<h3>常用标准头文件</h3> <ul style="list-style-type: none"> ■ ctype.h 字符处理 ■ math.h 与数学处理函数有关的说明与定义 ■ stdio.h 输入输出函数中使用的有关说明和定义 ■ string.h 字符串函数的有关说明和定义 ■ stddef.h 定义某些常用内容 ■ stdlib.h 杂项说明 ■ time.h 支持系统时间函数 	
59	<h3>10.3.5 编译预处理</h3> <ul style="list-style-type: none"> ■ 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。 ■ 文件包含(#include)和宏定义(#define)都是编译预处理指令 <ul style="list-style-type: none"> □ 在形式上都以“#”开头，不属于 C 语言中真正的语句 □ 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 	总结编译预处理概念与使用方式。
60	<h3>编译预处理</h3> <ul style="list-style-type: none"> ■ 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。 ■ 由于 #define 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。 	解释什么是编译，它与编译预处理的关系。

61	<div data-bbox="300 203 820 241" data-label="Image"></div> <h3 data-bbox="331 253 555 286">编译预处理功能</h3> <ul style="list-style-type: none"> ■ 编译预处理的主要功能： <ul style="list-style-type: none"> □ 文件包含（#include） □ 宏定义（#define） □ 条件编译 	
62	<div data-bbox="300 622 820 660" data-label="Image"></div> <h3 data-bbox="331 672 555 705">编译预处理功能</h3> <ul style="list-style-type: none"> ■ 条件编译 <pre data-bbox="355 757 512 929"> #define FLAG 1 #if FLAG 程序段1 #else 程序段2 #endif </pre> 	该内容了解即可。
63	<div data-bbox="300 1041 820 1079" data-label="Image"></div> <h3 data-bbox="312 1070 770 1137">10.4 大程序构成 ——多文件模块的学生信息库系统</h3> <ul style="list-style-type: none"> ■ 10.4.1 分模块设计学生信息库系统 ■ 10.4.2 C程序文件模块 ■ 10.4.3 文件模块间的通信 	<p>引入文件模块概念。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"> 1、便于分模块编译、连接与调试，降低上机调试复杂度 2、可实现多人合作开发程序
64	<div data-bbox="300 1460 820 1498" data-label="Image"></div> <h3 data-bbox="331 1496 775 1529">10.4.1 分模块设计学生信息库系统</h3> <p data-bbox="328 1543 794 1639">例10-10 请综合例9-1、例9-2、例9-3，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> <div data-bbox="316 1659 796 1771" data-label="Diagram"> <pre> graph TD main["主函数 main()"] --- new["新建 new_student()"] main --- output["输出 output_student()"] main --- average["计算平均分 average()"] main --- sort["平均成绩排 序 sort()"] main --- modify["修改 modify()"] main --- search["查询 search_student()"] </pre> </div>	多文件模块示例

65	<p>例10-10 程序文件模块</p> <ul style="list-style-type: none"> ■ 分成5个程序文件模块，所有文件存放在同一个文件夹下，采用文件包含的形式进行连接和调用。 □ 主函数程序文件student_system.c <pre> int Count = 0; /* 全局变量，记录当前学生总数 */ #include "input_output.c" #include "computing.c" #include "update.c" #include "search.c" int main(void) { struct student students[MaxSize]; /* 定义学生信息结构数组 */ new_student(students); /* 输入学生信息结构数组 */ average(students); /* 计算每一个学生的平均成绩 */ sort(students); /* 按学生的平均成绩排序 */ output_student(students); /* 显示排序后的结构数组 */ modify(students); /* 修改指定输入的学生信息 */ output_student(students); /* 显示修改后的结构数组 */ return 0; } </pre>	用文件包含方式将各文件模块组织在一起
66	<p>例10-10 程序文件模块</p> <ul style="list-style-type: none"> □ 输入输出程序文件input_output.c <pre> extern Count; /* 外部变量声明 */ void new_student(struct student students[]) /* 新建学生信息 */ { } void output_student(struct student students[]) /* 输出学生信息 */ { } </pre> <ul style="list-style-type: none"> □ 计算平均成绩程序文件computing.c <pre> extern Count; /* 外部变量声明 */ void average(struct student students[]) /* 计算个人平均成绩 */ { } </pre>	在文件模块中声明使用了主文件模块student_system.c 中的外部变量 Count
67	<p>例10-10 程序文件模块</p> <ul style="list-style-type: none"> □ 修改排序程序文件update.c <pre> extern Count; /* 外部变量声明 */ void modify(struct student *p) /* 修改学生成绩 */ { } void sort(struct student students[]) /* 平均成绩排序 */ { } </pre> <ul style="list-style-type: none"> □ 查询程序文件search.c <pre> extern Count; /* 外部变量声明 */ void search_student(struct student students[], int num) /* 查询学生信息 */ { } </pre>	
68	<p>10.4.2 C程序文件模块</p> <ul style="list-style-type: none"> ■ 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 ■ 一个大程序最好由一组小函数构成 ■ 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 ■ 为了避免一个文件过长，也会把程序分别保存为几个文件。 ■ 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。 	对为什么使用程序文件模块做进一步说明

69	 <h3>10.4.2 C程序文件模块</h3> <ul style="list-style-type: none"> ■ 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。 ■ 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。 ■ 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"> □ 文件包含 □ 工程文件（由具体语言系统提供） 	将程序文件模块合成一个整体的方法：工程方式、文件包含
70	 <h3>10.4.2 C程序文件模块</h3> <ul style="list-style-type: none"> ■ 程序—文件—函数关系 <ul style="list-style-type: none"> □ 小程序：主函数+若干函数 → 一个文件 □ 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 □ 整个程序只允许有一个main()函数 	无论是只有一个文件的小程序还是包含多个文件模块的大程序，都只能有一个主函数
71	 <h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> ■ 文件模块与变量 <ul style="list-style-type: none"> □ 外部变量 □ 静态全局变量 ■ 文件模块与函数 <ul style="list-style-type: none"> □ 外部函数 □ 静态的函数 	文件模块之间可以有条件地使用全局变量和函数
72	 <h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> ■ 外部变量 <ul style="list-style-type: none"> □ 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： extern 变量名表; □ 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 □ 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。 	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。

73	 <h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> ■ 静态全局变量 <ul style="list-style-type: none"> □ 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 □ 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，静态全局变量可以把变量的作用范围仅局限于当前的文件模块中 □ 即使其他文件模块使用外部变量声明，也不能使用该变量。 	<p>静态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>静态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>
74	 <h3>10.4.3 文件模块间的通信</h3> <ul style="list-style-type: none"> ■ 文件模块与函数 <ul style="list-style-type: none"> □ 外部函数 <ul style="list-style-type: none"> ■ 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： extern 函数类型 函数名(参数表说明); □ 静态的函数 <ul style="list-style-type: none"> ■ 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。 ■ 增加模块的独立性。 	<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，采用外部函数声明。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>
75	 <h3>本章小结</h3> <ul style="list-style-type: none"> ■ 多函数程序的组织结构 <ul style="list-style-type: none"> □ 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 <ul style="list-style-type: none"> □ 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 <ul style="list-style-type: none"> □ 文件包含 □ 宏实质：编译预处理的替代 □ 带参的宏——不是函数 	<p>回顾和总结本章的教学要点</p>

10.3 练习与习题参考答案

10.3.1 练习参考答案

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;
    }

```

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)

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
A	C	B	B	A	D

二. 填空题

1	5	2	720
	<hr/>		<hr/>
	g = 4		文件包含
3	g = 3	4	宏定义
	k = 6		条件编译
	<hr/>		<hr/>
5	7, -13		
	<hr/>		<hr/>

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 $\text{fact}(n)$ 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≥0 且 n≥0), 输出函数 Ack(m,n)的值。
Ack(m,n)定义为(m≥0 且 n≥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 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;
}
```

11. 有序表的增删改查操作。首先输入一个无重复元素的、从小到大排列的有序表，并在屏幕上显示以下菜单，用户可以反复对该有序表进行插入、删除、修改和查找操作，也可以选择结束。当用户输入编号 1~4 和相关参数时，将分别对该有序表进行插入、删除、修改和查找操作，输入其他编号，则结束操作。

```
[1] Insert
[2] Delete
[3] Modify
[4] Query
[Other option] End
```

解答：参见例 10-1

10.4 实验指导教材参考答案

一、调试示例

圆形体体积计算器：设计一个常用圆形体体积的计算器，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，并输入计算所需的相应参数。该计算器可支持多次反复计算，只要输入 1、2、3，即计算相应圆形体的体积；如果输入其他数字，将结束计算。

本例一共包含 5 个函数，调用结构如图 10.1 所示。采用 3 个文件模块实现：test10_1_main.cpp、test10_1_cal.cpp、test10_1_vol.cpp，其中 test10_1_vol.cpp 包含 3 个函数 vol_ball()、vol_cylind()、vol_cone()。

解答：略

二、基础编程题

(1) 递归实现指数函数：输入双精度浮点数 x 和整数 n ($n \geq 1$)，求 x^n 。要求定义和调用函数 calc_pow(x, n) 计算 x 的 n 次幂的值，用递归实现。试编写相应程序。

解答：参见习题程序设计第 3 题

(2) 递归计算 Ackermann 函数：输入两个整数 m 和 n ($m \geq 0$ 且 $n \geq 0$)，输出 Ackermann 函数的值，其函数定义如下。要求定义和调用函数 ack(m, n) 计算 Ackermann 函数的值。试编写相应程序。

解答：参见习题程序设计第 5 题

(3) 递归实现顺序输出整数：输入一个正整数 n ，对其进行按位顺序输出。要求定义和调用函数 printdigits(n) 将 n 的每一位数字从高位到低位顺序打印出来，每位数字占一行，用递归实现。试编写相应程序。

解答：参见习题程序设计第 8 题

(4) 递归求阶乘和：输入一个正整数 n ($0 < n \leq 10$)，求 $1!+2!+3!+\dots+n!$ 。要求定义和调用函数 fact(n) 计算 $n!$ 的值，定义和调用函数 factsum(n) 计算 $1!+2!+\dots+n!$ 的值，这 2 个函数都用递归

实现。试编写相应程序。

解答：参见习题程序设计第 2 题

(5) 递归求简单交错幂级数的部分和：输入双精度浮点数 x 和整数 n ($n \geq 1$)，计算下列简单交错幂级数的部分和，计算结果保留两位小数。要求定义并调用函数 $fn(x, n)$ 计算以下级数的部分和，用递归实现。试编写相应程序。

解答：参见习题程序设计第 4 题

(6) 统计素数（分别使用文件包含和工程文件的方式）：输入一个正整数 n ($0 < n < 10$) 和 n 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，另一个文件中编写判断素数的函数，分别使用文件包含和工程文件的方式实现。

解答：参见习题程序设计第 9 题

三、改错题

递归求 Fabonacci 数列：输入正整数 n ($1 \leq n \leq 46$)，输出斐波那契 (Fibonacci) 数列的第 n 项。Fabonacci 数列的定义如下。要求定义和调用函数 `fib(n)` 计算第 n 个 Fabonacci 数，用递归实现。试编写相应程序。

解答：

错误行号： 5	正确语句： <code>int fib(int n);</code>
错误行号： 15	正确语句： <code>return n;</code>
错误行号： 17	正确语句： <code>return fib(n-1) + fib(n-2);</code>

四、拓展编程题

(1) 递归求逆序数：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，用递归实现。例如 `reverse (123)` 的返回值是 321。试编写相应程序。

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;
}

```

(2) 十进制转换二进制：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)`，它的功能是在一行中打印出二进制的 n ，用递归实现。例如，调用 `dectobin(10)`，输出 1010。试编写相应程序。

解答：参见习题程序设计第 7 题。

(3) 简单加减法计算器（分别使用文件包含和工程文件的方式）：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。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、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);  
}
```

(4) 三角形面积公式如下，其中 **a**、**b**、**c** 分别是三角形的 3 条边。

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

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：参见习题程序设计第 10 题。