

第8章 指针

8.1 教学要点

本章主要介绍指针的基本概念、指针变量的定义、指针变量的基本运算、指针作为函数的参数、指针和数组、指针和字符串、动态内存分配和使用等知识。首先要把基本概念讲清，一定要结合图例来说明变量、内存单元和地址之间的关系；本章重点难点较多，讲解时注意趣味性激发学生兴趣，通过对比演示来分析指针和普通变量的区别。

8.1 节通过示例程序“密码开锁”引出指针的概念和主要知识点，教师在讲授时，重点分析密码开锁的过程，通过过程分析来说明变量、内存单元和地址之间的关系，从而引出指针的基本概念，通过指针与所指向变量的示意图，加深学生对指针的理解。接着介绍指针的基本运算以及指针变量的初始化。

8.2 节主要介绍指针作为函数的参数。该小节通过例子“角色互换”引入，演示使用指针作为函数参数的用途和用法。在本小节中要提示学生注意观察函数的形参和实参，区分按值调用和引用调用。在具体讲解时，通过详细分析3个 `swap()` 函数进行对比，讲解过程一定要配合画出指针指向图来说明参数变量值是如何改变的。最后通过一个例子来讲解如何使用指针作为函数参数返回多个值，解决以前一个函数最多只能 `return` 一个值的问题。

8.3 节通过示例程序“冒泡排序”引入数组名作为函数参数。首先重点介绍指针、数组和地址之间的关系，分析数组下标运算的含义和指针偏移的对应关系。通过例子来说明指针和数组可以实现相同的操作。然后重点讲述如何使用数组名作为函数的参数，最后对冒泡排序进行详细分析，总结回顾本节知识点。

8.4 节中通过示例程序“字符串压缩”引入字符指针与字符串处理。首先重点介绍字符指针和字符串的关联和区别。然后对常用字符串处理函数进行讲解，通过对比示例强调字符串运算的特殊性并分析原因。

8.5 节中通过示例程序“任意个整数求和”引入用指针实现内存动态分配。首先要分析为什么要进行内存的动态分配管理，然后介绍动态内存分配的一般步骤和具体的分配函数。最后可以通过和学生一起使用动态内存分配的方法来改写“冒泡排序”，做到融会贯通。

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 8.1。

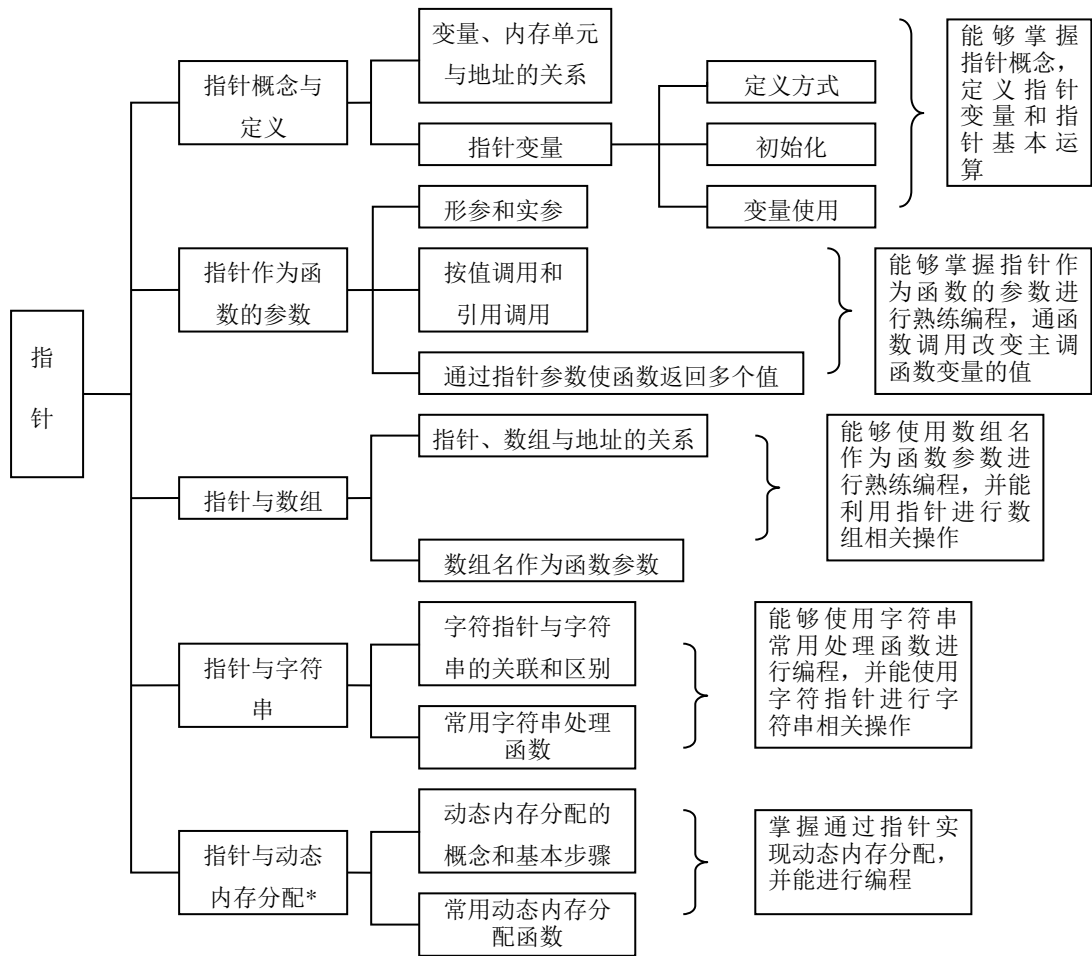


图 8.1 知识能力结构图

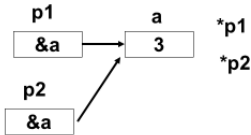
8.2 讲稿

1	<div data-bbox="300 1373 331 1406" style="display: inline-block; width: 100px; height: 10px; background: linear-gradient(to right, blue, white);"></div> <p>Chap 8 指针</p> <p>8.1 密码开锁</p> <p>8.2 角色互换</p> <p>8.3 冒泡排序</p> <p>8.4 字符串压缩</p> <p>8.5 任意个整数求和*</p>	<p>介绍主要章节安排。通过这些小节的名字引起学生对本章学习的兴趣。</p>
---	---	--

2	<div><div></div><h3>本章要点</h3><ul style="list-style-type: none">■ 变量、内存单元和地址之间是什么关系？■ 如何定义指针变量，怎样才能使用指针变量？■ 什么是指针变量的初始化？■ 指针变量的基本运算有哪些？如何使用指针操作所指向的变量？■ 指针作为函数参数的作用是什么？■ 如何使用指针实现函数调用返回多个值？■ 如何利用指针实现内存的动态分配？</div>	要点可快速过一遍，不用展开。可以在本章知识点讲完后再回顾。									
3	<div><div></div><h3>8.1 密码开锁</h3><p>一个密室逃脱游戏中的密码开锁：26个寄存箱，每个寄存箱上按顺序都有一个英文字母和一个编号，字母从A到Z，编号从01到26</p><p>关键点分析</p><ul style="list-style-type: none">■ 得到线索：找到一把钥匙，打开p寄存箱（编号为16）■ 提示地址：里面是一把刻着数字24的钥匙■ 找到目标：打开编号为24的X寄存箱■ 取出内容：“5342”</div>	介绍一下游戏过程，重点放在关键点分析。									
4	<div><div></div><h3>8.1 寻找密码的途径分析</h3><ul style="list-style-type: none">■ 密码存放需要一定的存储空间作为存放地，每个存放地都会有地址■ 如果知道了存放地的名字，当然能够找到密码■ 如果不知道存放地的名字，知道该存放地的地址也能够取出密码■ 如果有另外一个地方存放了该密码存放地的地址，那么就能顺藤摸瓜，间接找到密码</div>	分析寻找密码的两种可能途径									
5	<div><div></div><h3>8.1 密码存放示意图</h3><div><div><div>p</div><div>24</div><div>16</div></div><div><div>x</div><div>5342</div><div>24</div></div><div></div></div><table><tr><td>名字</td><td>p</td><td>x</td></tr><tr><td>地址</td><td>16</td><td>24</td></tr><tr><td>内容</td><td>24</td><td>5342</td></tr></table></div>	名字	p	x	地址	16	24	内容	24	5342	<p>通过示意图说明地址为 16 的 P 变量内容存放的 24 是另外一个变量 x 的地址，通过这个地址可以顺藤摸瓜得到变量 x 的内容：5342</p> <p>强调箭头的指向</p>
名字	p	x									
地址	16	24									
内容	24	5342									

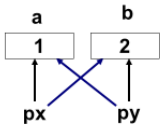
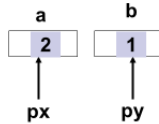
6	<div><div>例8-1 利用指针模拟密码开锁游戏</div><div><div>获取密码的两种方法：</div><div>#include <stdio.h> int main(void) { int x = 5342; /* 变量x用于存放密码值5342 */ int *p = NULL; /* 定义整型指针变量p，NULL值为0，代表空指针 */ p = &x; /* 将变量x的地址存储在p中 */ /* 通过变量名x输出密码值 */ printf("If I know the name of the variable, I can get it's value by name: %d\n", x); /* 通过变量x的地址输出密码值 */ printf("If I know the address of the variable is: %x, then I also can get it's value by address: %d\n", p, *p); return 0; }</div></div><div><div>If I know the name of the variable, I can get it's value by name: 5342</div><div>If I know the address of the variable is: 12ff7c, then I also can get it's value by address: 5342</div></div></div>	<p>简单描述模拟程序并运行</p> <p>让学生留意 <code>int *p</code>，一种新的指针类型变量定义</p>																		
7	<div><div>8.1.2 地址和指针—指针的概念</div><div><div>内存单元</div><table><thead><tr><th>地址</th><th>内容</th><th>变量</th></tr></thead><tbody><tr><td>1000</td><td>20</td><td>x</td></tr><tr><td>1002</td><td>1</td><td>y</td></tr><tr><td>1004</td><td>155</td><td>z</td></tr><tr><td>2000</td><td>1000</td><td>p</td></tr><tr><td>2002</td><td></td><td></td></tr></tbody></table><div>直接访问：通过变量名访问 <code>int x = 20, y = 1, z = 155;</code> <code>printf("%d", x);</code></div><div>间接访问：通过另一个变量访问 把变量的地址放到另一变量中 使用时先找到后者 再从中取出前者的地址</div><div>地址 指针变量</div></div></div>	地址	内容	变量	1000	20	x	1002	1	y	1004	155	z	2000	1000	p	2002			<p>通过对图的分析，详细说明直接访问和间接访问。</p>
地址	内容	变量																		
1000	20	x																		
1002	1	y																		
1004	155	z																		
2000	1000	p																		
2002																				
8	<div><div>指针</div><div><div>内存单元</div><table><thead><tr><th>地址</th><th>内容</th><th>变量</th></tr></thead><tbody><tr><td>1000</td><td>20</td><td>x</td></tr><tr><td>1002</td><td>1</td><td>y</td></tr><tr><td>1004</td><td>155</td><td>z</td></tr><tr><td>2000</td><td>1000</td><td>p</td></tr><tr><td>2002</td><td></td><td></td></tr></tbody></table><div><code>int x = 20, y = 1, z = 155;</code> <code>printf("%d", x);</code></div><div>某个变量的地址</div><div>指向</div><div>指针变量：存放地址的变量</div><div>地址 指针变量</div></div></div>	地址	内容	变量	1000	20	x	1002	1	y	1004	155	z	2000	1000	p	2002			<p>通过例子说明指针变量的概念</p> <p>提出 “指向” 关系</p>
地址	内容	变量																		
1000	20	x																		
1002	1	y																		
1004	155	z																		
2000	1000	p																		
2002																				
9	<div><div>8.1.3 指针变量的定义</div><div><div>类型名 * 指针变量名</div><div>↑</div><div>指针声明符</div><div>指针变量所指向的变量的类型</div></div><div><code>int *p;</code> p 是整型指针，指向整型变量 <code>float *fp;</code> fp 是浮点型指针，指向浮点型变量 <code>char *cp;</code> cp 是字符型指针，指向字符型变量</div></div>	<p>如何定义指针变量</p> <p>强调定义时类型名是指针变量所指向的变量类型</p>																		

10	<p>指针变量的定义</p> <p>类型名 * 指针变量名</p> <p>int *p;</p> <ul style="list-style-type: none"> □ 指针变量名是 p，不是 *p □ * 是指针声明符 <p>int k, *p1, *p2;</p> <p>等价于：</p> <p>int k;</p> <p>int *p1;</p> <p>int *p2;</p> <ul style="list-style-type: none"> □ 定义多个指针变量时，每一个指针变量前面都必须加上* 	<p>注意：指针变量名是 p，不是 *p</p> <p>定义指针变量可以和与它所指变量类型具有相同类型的变量放在同一条语句中定义。</p>
11	<p>8.1.4 指针的基本运算</p> <p>如果指针的值是某个变量的地址，通过指针就能间接访问那个变量。</p> <p>1. 取地址运算和间接访问运算</p> <p>& 取地址运算符</p> <p>int *p, a = 3; 指针变量的类型和它所指向变量的类型相同</p> <p>p = &a; 把 a 的地址赋给 p，即 p 指向 a</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="text-align: center;">p [&a]</div> <div style="margin: 0 10px;">→</div> <div style="text-align: center;">a [3]</div> <div style="margin-left: 10px;">*p</div> </div> <p>* 间接访问运算符，访问指针所指向的变量 printf("%d", *p); 输出 p 指向的变量 a 的值</p>	<p>详细介绍取地址符&</p> <p>一定要在黑板上画出如左图那样的示意图，强调 p 的内容是 a 的地址，然后画出的指向的箭头。</p> <p>建议学生在初学指针时都必须要画出指向关系图</p>
12	<p>例8-2指针取地址运算和间接访问运算</p> <pre> #include <stdio.h> int main (void) { int a = 3, *p; p = &a; printf ("a=%d, *p=%d\n", a, *p); *p = 10; printf ("a=%d, *p=%d\n", a, *p); printf ("Enter a: "); scanf ("%d", &a); printf ("a=%d, *p=%d\n", a, *p); (*p)++; printf ("a=%d, *p=%d\n", a, *p); return 0; }</pre> <div style="display: flex; align-items: center; justify-content: center;"> <div style="text-align: center;">p [&a]</div> <div style="margin: 0 10px;">→</div> <div style="text-align: center;">a [3]</div> <div style="margin-left: 10px;">*p</div> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> a = 3, *p = 3 a = 10, *p = 10 Enter a: 5 a = 5, *p = 5 a = 6, *p = 6 </div>	<p>演示程序的运行</p> <p>提示指针赋值语句 p=&a</p> <p>然后画出指向关系图</p>
13	<p>说明</p> <p>(1) 当 p = &a 后，*p 与 a 相同</p> <p>(2) int *p; 定义指针变量 p</p> <p>*p = 10; 指针 p 所指向的变量，即 a</p> <p>(3) &*p 与 &a 相同，是地址</p> <p>*&a 与 a 相同，是变量</p> <p>(4) (*p)++ 等价于 a++</p> <p>将 p 所指向的变量值加 1</p> <p>*p++ 等价于 *(p++)</p> <p>先取 *p，然后 p 自加，此时 p 不再指向 a</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="text-align: center;">p [&a]</div> <div style="margin: 0 10px;">→</div> <div style="text-align: center;">a [3]</div> <div style="margin-left: 10px;">*p</div> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre> int a = 1, x, *p; p = &a; printf ("%x\n", p); x = *p++; printf ("%d,%x", x, p); </pre> </div>	<p>难点：</p> <p>*p++ 等价于 *(p++)</p> <p>*p++ 的值为 a 的值，因为 p++ 先返回 p 的值作为表达式的值，然后才是 p=p+1，这时 p 就不再指向 a 了</p>

14	<div data-bbox="304 203 794 562" data-label="Complex-Block"> <div>2. 赋值运算</div> <pre>int a = 3, *p1, *p2; p1 = &a; 把 a 的地址赋给 p1, 即 p1 指向 a p2 = p1; p2 也指向 a</pre> <div>相同类型的指针才能相互赋值</div>  </div>	<p>指针赋值会建立指向关系 但只有相同类型的指针才能互相赋值</p>
15	<div data-bbox="304 620 794 981" data-label="Complex-Block"> <div>8.1.5 指针变量的初始化</div> <ol style="list-style-type: none"> 1) 指针变量在定义后也要先赋值再引用 2) 在定义指针变量时, 可以同时对它赋初值 <pre>int a; int *p1 = &a; int *p2 = p1;</pre> 3) 不能用数值作为指针变量的初值, 但可以将一个指针变量初始化为一个空指针 <pre>int *p=1000; p = 0; p = NULL; p = (int*)1732;</pre> <div>使用强制类型转换 (int*) 来避免编译错误, 不提倡</div> </div>	<p>注意:</p> <p>(1) 在指针变量定义或者初始化时变量名前面的“*”只表示该变量是个指针变量, 它既不是乘法运算符也不是间接访问符。</p> <p>(2) 把一个变量的地址作为初始化值赋给指针变量时, 该变量必须在此之前已经定义。因为变量只有在定义后才被分配存储单元, 它的地址才能赋给指针变量。</p> <p>(3) 可以用初始化了的指针变量给另一个指针变量作初始化值。</p> <p>(4) 不能用数值作为指针变量的初值, 但可以将一个指针变量初始化为一个空指针。例如 <pre>int *p=1000;</pre> 是不对的, 而 <pre>int *p=0;</pre> 是将指针变量初始化为空指针。这里 0 是 ASCII 字符 NULL 的值。</p> <p>(5) 指针变量定义时的数据类型和它所指向的目标变量的数据类型必须一致, 因为不同的数据类型所占用的存储单元的字节数不同。</p>
16	<div data-bbox="304 1576 794 1906" data-label="Complex-Block"> <div>8.2 角色互换</div> <p>如何通过函数调用实现代表2个角色的变量互相...</p> <p>三套方案</p> <ul style="list-style-type: none"> ■ swap1() ■ swap2() ■ swap3() <p>哪个方案能成功?</p> </div>	<p>提出问题: 怎样通过函数调用实现 2 个变量的互换?</p> <p>有三套交换方案, 哪套会成功? 可以让学生猜测那个 swap() 可以交换变量 a 和 b 的值</p> <p>讲这节时还可以联系历史故事狸猫换太子, 怎么做狸猫和太子的互换。</p>

17	<p>例8-3 指针作为函数参数模拟角色互换</p> <pre> int main (void) { int a = 1, b = 2; int *pa = &a, *pb = &b; void swap1(int x, int y), swap2(int *px, int *py), swap3 (int *px, int *py); swap1 (a, b); printf ("After calling swap1: a=%d b=%d\n", a, b); a = 1; b = 2; swap2 (pa, pb); printf ("After calling swap2: a=%d b=%d\n", a, b); a = 1; b = 2; swap3 (pa, pb); printf ("After calling swap3: a=%d b=%d\n", a, b); return 0; } </pre> <p>调用哪个函数，可以交换main ()中变量a和b的值？</p>	<p>1、介绍程序框架，注意 3 个 swap()函数</p> <p>2、让同学观察 3 个 swap()函数的不同</p>
18	<p>例8-3 swap1()</p> <pre> swap1 (a, b); void swap1 (int x, int y) { int t; t = x; x = y; y = t; } </pre>	<p>swap1()的形参是普通整型变量</p>
19	<p>例8-3 swap2()</p> <pre> swap2 (&a, &b); void swap2 (int *px, int *py) { int t; t = *px; *px = *py; *py = t; } </pre>	<p>swap2()的形参是指针变量</p>
20	<p>例8-3 swap3()</p> <pre> swap3 (&a, &b); void swap3 (int *px, int *py) { int *pt; pt = px; px = py; py = pt; } </pre> <div style="border: 1px solid black; padding: 2px; width: fit-content;"> <p>After calling swap1: a=1, b=2 After calling swap2: a=2, b=1 After calling swap3: a=1, b=2</p> </div>	<p>swap3()的形参也是指针变量</p> <p>看运行结果，swap2()实现了两个变量的互换，而其他两个没有</p>

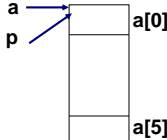
21	<div data-bbox="300 203 820 593" data-label="Complex-Block"> <div> <div></div> <div>8.2.2 指针作为函数参数</div> </div> <ul style="list-style-type: none"> 函数参数包括实参和形参，两者的类型要一致, 可以是指针类型 如果实参是某个变量的地址，相应的形参就是指针 在C语言中实参和形参之间的数据传递是单向的“值传递”方式 </div>	<div data-bbox="842 203 1358 593" data-label="List-Group"> <ol style="list-style-type: none"> 回顾形参和实参 按值调用和引用调用的区别 在函数定义时，将指针做为函数的参数，在函数调用时，要把同类型指针或者变量的地址作为实参 提示学生注意区分 3 个 swap()的形参区别 </div>
22	<div data-bbox="300 620 820 1341" data-label="Complex-Block"> <div> <div></div> <div>例8-3 swap1()</div> </div> <div> <div>swap1 (a, b);</div> <div> <div>a</div> <div>b</div> <div>1</div> <div>2</div> </div> </div> <div> <div>void swap1 (int x, int y)</div> <div>{ int t;</div> <div>t = x;</div> <div>x = y;</div> <div>y = t;</div> <div>}</div> </div> <div> <div> <div>x</div> <div>y</div> <div>2</div> <div>1</div> </div> <div>在swap1()函数中改变了形参x,y的值但不会反过来影响到实参的值</div> <div>swap1()不能改变main()函数中实参a和b的值</div> </div> </div>	<div data-bbox="842 620 1358 1341" data-label="List-Group"> <ol style="list-style-type: none"> 提示实参是 a 和 b，形参是 x 和 y 函数 swap1()使用的是变量调用，也就是值调用 分析函数调用过程： <ol style="list-style-type: none"> 实参 a 和 b 分别把自己的值赋给了可看作函数局部变量的 x 和 y 函数局部变量 x 和 y 在函数中进行了交换 当函数调用结束，x 和 y 也被销毁，不再存在；但原来的变量 a 和 b 仍然保持原值，没有发生任何改变 小结：参数的传递是从实参变量到形参变量的单方向上值的传递，即使在 swap1()函数中改变了形参的值，也不会反过来影响到实参的值。因此，调用 swap1()不能改变 main()函数中实参 a 和 b 的值。 </div>
23	<div data-bbox="300 1368 820 1968" data-label="Complex-Block"> <div> <div></div> <div>例8-3 swap2()</div> </div> <div> <div>swap2 (&a, &b);</div> <div> <div>a</div> <div>b</div> <div>2</div> <div>1</div> </div> </div> <div> <div>void swap2 (int *px, int *py)</div> <div>{ int t;</div> <div>t = *px;</div> <div>*px = *py;</div> <div>*py = t;</div> <div>}</div> </div> <div> <div> <div>px</div> <div>py</div> </div> <div>在swap2()函数中交换*px和*py的值，主调函数中a和b的值也相应交换了</div> <div>值传递，地址未变，但存放的变量值改变了</div> </div> </div>	<div data-bbox="842 1368 1358 1968" data-label="List-Group"> <ol style="list-style-type: none"> 让学生注意观察形参和实参 分析函数调用过程： <ol style="list-style-type: none"> 在函数 swap2()被调用时，将实参 a 的地址和 b 的地址传递给形参 px 和 py。这样，px 和 py 中分别存放了 a 和 b 的地址，px 指向 a，py 指向 b swap2()执行过程中把 px 所指向的单元内容和 py 所指向的单元内容作了互换 函数结束，px 和 py 被销毁，但 a 和 b 的值已经发生改变 小结：在函数 swap2()中交换*px 和*py 的值，主调函数中 a 和 b 的值也相应交换了，即达到了交换数据的目的 </div>

24	<p>例8-3 swap3()</p> <pre> swap3 (&a, &b); void swap3 (int *px, int *py) { int *pt; pt = px; px = py; py = pt; } </pre>  <p>swap3()中直接交换了形参指针px和py的值</p> <p>值传递，形参指针的改变不会影响实参</p>	<ol style="list-style-type: none"> 1、让学生注意观察形参和实参，和前面两个函数比较 2、分析函数调用过程： <ol style="list-style-type: none"> a) 在函数 swap3()被调用时，将实参 a 的地址和 b 的地址传递给形参 px 和 py。这样，px 和 py 中分别存放了 a 和 b 的地址，px 指向 a，py 指向 b b) Swap3()执行过程中把 px 和 py 的值作了互换，但 a 和 b 的值然后保持不变 c) 函数结束，px 和 py 被销毁，但 a 和 b 的值已经发生改变 3、小结：形参 px 和 py 的改变不会影响实参，因此调用该函数并不能改变主调函数 main()中变量 a 和 b 值
25	<p>指针作为函数参数的应用</p> <pre> swap2 (&a, &b); void swap2 (int *px, int *py) { int t; t = *px; *px = *py; *py = t; } </pre>  <p>要通过函数调用来改变主调函数中某个变量的值：</p> <ol style="list-style-type: none"> (1) 在主调函数中，将该变量的地址或者指向该变量的指针作为实参 (2) 在被调函数中，用指针类型形参接受该变量的地址 (3) 在被调函数中，改变形参所指向变量的值 	<p>分析什么情况下可以通过函数调用来改变主调函数中变量的值：</p> <p>可以把指针作为函数的参数。在主调函数中，将该变量的地址或者指向该变量的指针作为实参。在被调函数中，用指针类型形参接受该变量的地址，并改变形参所指向变量的值。</p>
26	<p>通过指针实现函数调用返回多个值</p> <p>例8-4 输入年和天数，输出对应的年、月、日。 例如：输入2000和61，输出2000-3-1。</p> <p>定义函数month_day(year, yearday, *pmonth, *pday) 用2个指针作为函数的参数，带回2个结果</p> <pre> int main(void) { int day, month, year, yearday; void month_day(int year, int yearday, int *pmonth, int *pday); printf("input year and yearday: "); scanf("%d%d", &year, &yearday); month_day(year, yearday, &month, &day); printf("%d-%d-%d\n", year, month, day); return 0; } </pre>	<ol style="list-style-type: none"> 1、先提出背景问题：函数只能通过 return 语句返回 1 个值。如果希望函数调用能将多个计算结果带回主调函数，用 return 语句是无法实现的 2、解决方案：将指针作为函数的参数就能使函数返回多个值 3、演示运行例子

27	<div><div>例8-4</div><pre>void month_day (int year, int yearday, int * pmonth, int * pday) { int k, leap; int tab [2][13] = { {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}, {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}, }; /* 建立闰年判别条件leap */ leap = (year%4 == 0 && year%100 != 0) year%400 == 0; for (k = 1; yearday > tab[leap][k]; k++){ yearday = yearday-tab [leap][k]; } *pmonth = k; *pday = yearday; }</pre><div>input year and yearday: 2000 61 2000-3-1</div><div><div>month</div><div>3</div><div>pmonth</div><div>day</div><div>1</div><div>pday</div></div></div>	<div>1、分析算法设计，特别是 2 维数组的用途</div> <div>2、让学生回顾怎么判断闰年</div> <div>3、分析程序执行：在 main()函数中调用函数 month_day()时，将变量 month 和 day 的地址作为实参，在被调函数中，用形参指针 pmonth 和 pday 分别接收地址，并改变了形参所指向变量的值。因此，main()函数中 month 和 day 的值也随之改变</div>															
28	<div><div>8.3 冒泡排序-程序解析</div><pre>void bubble (int a[], int n) { int i, j; for (i = 1; i < n; i++) for (j = 0; j < n-i; j++) if (a[j] > a[j+1]){ t=a[j]; a[j]=a[j+1]; a[j+1]=t; } }</pre><pre>int main(void) { int n, a[8]; printf("Enter n (n<=8): "); scanf("%d", &n); printf("Enter a[%d] : ", n); for (i=0; i<n; i++) scanf("%d", &a[i]); bubble(a, n); printf("After sorted, a[%d] = ", n); for (i=0; i<n; i++) printf("%3d", a[i]); return 0; }</pre><div>Enter n (n<=8): 8 Enter a[8] : 7 3 66 3 -5 22 -77 2 After sorted, a[8] = -77 -5 2 3 3 7 22 66</div></div>	<div>1、简要回顾选择排序</div> <div>2、运行冒泡排序程序，不用详细讲解</div> <div>3、引入数组作为函数形参</div>															
29	<div><div>8.3.2 数组和地址间的关系</div><pre>int a[100], *p;</pre><div>数组名代表一个地址， 它的值是数组首元素的地址（基地址） a+i 是数组a的基地址的第i个偏移量</div><div><div>sum = 0; for(i = 0; i < 100; i++){ sum = sum + *(a+i); }</div><div>下标运算符[]的含义</div></div><table><thead><tr><th>地址</th><th>内容</th><th>数组元素</th></tr></thead><tbody><tr><td>a</td><td>3000</td><td>a[0]</td></tr><tr><td>a+1</td><td>3002</td><td>a[1]</td></tr><tr><td>a+i</td><td></td><td>a[i]</td></tr><tr><td>a+99</td><td>3198</td><td>a[99]</td></tr></tbody></table><div><div>a+i</div><div>&a[i]</div><div>*(a+i)</div></div></div>	地址	内容	数组元素	a	3000	a[0]	a+1	3002	a[1]	a+i		a[i]	a+99	3198	a[99]	<div>1、在声明数组时，编译器必须分配地址和足够的存储空间，以存储数组的所有元素</div> <div>2、数组的基地址是在内存中存储数组的起始位置，它是数组中第 1 个元素（下标为 0）的地址，因此数组名本身是一个地址即指针值</div> <div>3、指针是以地址作为值的变量，而数组名的值是一个特殊的固定地址，可以把它看作是常量指针</div> <div>4、数组下标运算符的含义和指针偏移的对应关系</div>
地址	内容	数组元素															
a	3000	a[0]															
a+1	3002	a[1]															
a+i		a[i]															
a+99	3198	a[99]															
30	<div><div>指针和数组的关系</div><div>任何由数组下标来实现的操作 都能用指针来完成</div><pre>int a[100], *p; p = a; 或 p = &a[0];</pre><table><thead><tr><th>地址</th><th>内容</th><th>数组元素</th></tr></thead><tbody><tr><td>a</td><td>3000</td><td>a[0]</td></tr><tr><td>a+1</td><td>3002</td><td>a[1]</td></tr><tr><td>a+i</td><td></td><td>a[i]</td></tr><tr><td>a+99</td><td>3198</td><td>a[99]</td></tr></tbody></table><div><div>p = a; sum = 0; for (i = 0; i < 100; i++){ sum = sum + p[i]; }</div><div><div>&a[i]</div><div>a+i</div><div>p+i</div><div>&p[i]</div><div>等价</div><div>等价</div><div>*(a+i)</div><div>*(p+i)</div><div>p[i]</div></div></div></div>	地址	内容	数组元素	a	3000	a[0]	a+1	3002	a[1]	a+i		a[i]	a+99	3198	a[99]	<div>通过示意图来详细说明地址、数组和指针的关系</div>
地址	内容	数组元素															
a	3000	a[0]															
a+1	3002	a[1]															
a+i		a[i]															
a+99	3198	a[99]															

31	<p>用指针完成对数组的操作</p> <pre>int a[100], *p; 移动指针</pre> <table border="1"> <thead> <tr> <th>地址</th> <th>内容</th> <th>数组元素</th> </tr> </thead> <tbody> <tr> <td>p a 3000</td> <td></td> <td>a[0]</td> </tr> <tr> <td>p a+1 3002</td> <td></td> <td>a[1]</td> </tr> <tr> <td>p a+i</td> <td></td> <td>a[i]</td> </tr> <tr> <td>p a+99 3198</td> <td></td> <td>a[99]</td> </tr> </tbody> </table> <pre>sum = 0; for (p = a; p <= &a[99]; p++){ sum = sum + *p; }</pre>	地址	内容	数组元素	p a 3000		a[0]	p a+1 3002		a[1]	p a+i		a[i]	p a+99 3198		a[99]	把上例从数组操作改为对指针操作
地址	内容	数组元素															
p a 3000		a[0]															
p a+1 3002		a[1]															
p a+i		a[i]															
p a+99 3198		a[99]															
32	<p>例8-7 使用指针计算数组元素个数和数组元素的存储单元数</p> <pre>#include <stdio.h> int main (void) { double a[2], *p, *q; p = &a[0]; q = p + 1; printf ("%d\n", q - p); printf ("%d\n", (int) q - (int) p); return 0; }</pre> <table border="1"> <thead> <tr> <th>地址</th> <th>内容</th> <th>数组元素</th> </tr> </thead> <tbody> <tr> <td>p a 3000</td> <td></td> <td>a[0]</td> </tr> <tr> <td>q a+1 3008</td> <td></td> <td>a[1]</td> </tr> </tbody> </table> <p>1 指针p和q之间元素的个数 8 指针p和q之间的字节数 地址值</p>	地址	内容	数组元素	p a 3000		a[0]	q a+1 3008		a[1]	注意 q-p 和(int)q-(int)p 的含义上的差别和结果的不同						
地址	内容	数组元素															
p a 3000		a[0]															
q a+1 3008		a[1]															
33	<p>指针的算术运算和比较运算</p> <pre>double *p, *q; p a 3000 q a+1 3008</pre> <table border="1"> <thead> <tr> <th>地址</th> <th>内容</th> <th>数组元素</th> </tr> </thead> <tbody> <tr> <td>p a 3000</td> <td></td> <td>a[0]</td> </tr> <tr> <td>q a+1 3008</td> <td></td> <td>a[1]</td> </tr> </tbody> </table> <ul style="list-style-type: none"> q - p 两个相同类型的指针相减，表示它们之间相隔的存储单元的数目 p + 1 / p - 1 指向下一个存储单元 / 指向上一个存储单元 其他操作都是非法的 指针相加、相乘和相除，或指针加上和减去一个浮点数 p < q 两个相同类型指针可以用关系运算符比较大小 	地址	内容	数组元素	p a 3000		a[0]	q a+1 3008		a[1]	强度非法的指针操作类型，让学生思考两个相同类型的指针变量能不能相加？为什么？						
地址	内容	数组元素															
p a 3000		a[0]															
q a+1 3008		a[1]															
34	<p>例8-7 分别使用数组和指针计算数组元素之和</p> <pre>int main(void) { int i, a[10], *p; long sum = 0; printf("Enter 10 integers: "); for(i = 0; i < 10; i++) scanf("%d", &a[i]); for (i = 0; i < 10; i++) sum = sum + a[i]; printf("calculated by array, sum=%d\n", sum); sum=0; for(p = a; p <= a+9; p++) sum = sum + *p; printf("calculated by pointer, sum=%d\n", sum); return 0; }</pre> <table border="1"> <thead> <tr> <th>地址</th> <th>内容</th> <th>数组元素</th> </tr> </thead> <tbody> <tr> <td>p a 3000</td> <td></td> <td>a[0]</td> </tr> <tr> <td>p a+1 3002</td> <td></td> <td>a[1]</td> </tr> <tr> <td>p a+i</td> <td></td> <td>a[i]</td> </tr> <tr> <td>p a+9 3018</td> <td></td> <td>a[9]</td> </tr> </tbody> </table> <p>Enter 10 integers: 10 9 8 7 6 5 4 3 2 1 calculated by array, sum=55 calculated by pointer, sum=55</p>	地址	内容	数组元素	p a 3000		a[0]	p a+1 3002		a[1]	p a+i		a[i]	p a+9 3018		a[9]	通过该例子来总结使用数组和指针可以实现相同的操作
地址	内容	数组元素															
p a 3000		a[0]															
p a+1 3002		a[1]															
p a+i		a[i]															
p a+9 3018		a[9]															

35	<h3>8.3.3 数组名作为函数的参数</h3> <p>数组元素作为函数实参时，函数形参为变量 与变量作为函数实参相同，值传递</p> <pre>double fact (int n); int main(void) { int a[5]={1, 4, 5, 7, 9}; int i, n = 5; double sum; sum = 0; for(i = 1; i <= n; i++) sum = sum + fact(a[i-1]); printf("sum = %e\n", sum); return 0; }</pre> <pre>double fact (int n) { int i; double result = 1; for (i = 1; i <= n; i++) result = result * i; return result; }</pre> <p style="text-align: right;">1!+4!+5!+7!+9!</p>	数组元素作为参数实参时，形参为普通变量
36	<h3>数组名作为函数的参数</h3> <ul style="list-style-type: none"> 数组名是指针常量，相当于指针作为函数的参数 数组名做为实参，形参是指针变量（数组） <p>(1) 实参是数组名 (2) 形参是指针变量 可以写成数组形式</p> <pre>int a[]</pre> <pre>int sum (int *a, int n) { int i, s = 0; for(i=0; i<n; i++) s += a[i]; return(s); }</pre> <p>例</p> <pre>int main(void) { int i; int b[5] = {1, 4, 5, 7, 9}; printf("%d\n", sum(b, 5)); return 0; }</pre>	<ol style="list-style-type: none"> 1、数组名作为函数实参时，形参是指针变量（或者数组） 2、在函数定义中，被声明为数组的形参实际上是一个指针 3、当传递数组时，按值调用传递它的基地址，数组元素本身不被复制 4、作为一种表示习惯，编译器允许在作为参数声明的指针中使用数组方括号
37	<pre>int sum (int *a, int n) { int i, s = 0; for(i=0; i<n; i++) s += a[i]; return(s); }</pre> <pre>int main(void) { int i; int b[5] = {1, 4, 5, 7, 9}; printf("%d\n", sum(b, 5)); return 0; }</pre>	结合图来分析 sum 的几个不同调用
38	<h3>示例 二分查找</h3> <pre>int Bsearch(int *p, int n, int x) /* 二分查找函数 */ { int low, high, mid; low = 0; high = n - 1; /* 开始时查找区间为整个数组 */ while (low <= high) { /* 循环条件 */ mid = (low + high) / 2; /* 中间位置 */ if (x == p[mid]) break; /* 查找成功，中止循环 */ else if (x < p[mid]) high = mid - 1; /* 前半段，high前移 */ else low = mid + 1; /* 后半段，low后移 */ } if (low <= high) return mid; /* 找到返回下标 */ else return -1; /* 找不到返回-1 */ }</pre>	以指针作为函数参数形式实现二分查找，演示程序

39	<p>■ 数组名做为函数的参数，在函数调用时，将实参数组首元素的地址传给形参（指针变量），因此，形参也指向实参数组的首元素。如果改变形参所指向单元的值，就是改变实参数组首元素的值。</p> <p>■ 或：形参数组和实参数组共用同一段存储空间，如果形参数组中元素的值发生变化，实参数组中元素的值也同时发生变化。</p> 	<p>强调：形参数组和实参数组共用同一段存储空间，可类别为同一地址的大楼的两个名称。</p>
40	<p>■ 8.3.4 冒泡排序算法分析</p> <p>相邻两个数比较,小的调到前面,大的调到后面</p> <pre> 9 8 8 8 8 8 5 4 4 0 8 9 5 5 5 5 4 5 0 4 5 5 9 4 4 4 6 0 5 4 4 4 9 6 6 0 6 6 6 6 6 9 0 8 0 0 0 0 0 9 </pre>	<p>详细分析冒泡排序的过程</p>
41	<p>■</p> <pre> int main(void) { int i, j, n, t, a[10]; n = 6; for(i = 0; i < n; i++) scanf("%d", &a[i]); for(i = 1; i < n; i++) for(j = 0; j < n-i; j++) if(a[j] > a[j+1]) { t = a[j]; a[j] = a[j+1]; a[j+1] = t; } return 0; } </pre> <pre> 9 8 5 4 6 0 i=1 j=0: 8 9 5 4 6 0 j=1: 8 5 9 4 6 0 j=2: 8 5 4 9 6 0 j=3: 8 5 4 6 9 0 j=4: 8 5 4 6 0 9 </pre>	<p>没调用排序函数的程序写法，主要为验证前面分析的算法思想</p>
42	<p>■</p> <pre> void sort(int *array, int n) { int i, j, t; for(i=1; i<n; i++) for(j=0; j<n-i; j++) if(array[j]>array[j+1]){ t = array[j]; array[j] = array[j+1]; array[j+1] = t; } } </pre> <pre> int main(void) { int i, a[10]; for(i=0; i<10; i++) scanf("%d", &a[i]); sort(a, 10); for(i=0; i<10; i++) printf("%d ", a[i]); printf("\n"); return 0; } </pre>	<p>为了实现算法可重用性，把排序功能提炼为一个通用函数 sort，注意形参</p>

43	<div data-bbox="304 203 807 264" data-label="Section-Header"> <h2>8.4 字符串压缩</h2> </div> <div data-bbox="373 300 564 360" data-label="Text"> <p>字符串：字符数组 字符指针</p> </div> <div data-bbox="373 398 671 495" data-label="List-Group"> <ul style="list-style-type: none"> 8.4.1 程序解析 8.4.2 字符串和字符指针 8.4.3 常用的字符串处理函数 </div>	字符串是一种常用的特殊字符数组
44	<div data-bbox="304 616 807 676" data-label="Section-Header"> <h2>8.4.1 程序解析—字符串压缩</h2> </div> <div data-bbox="323 692 807 748" data-label="Text"> <p>例8-8 输入一个长度小于80的字符串，按规则对字符串进行压缩，输出压缩后的字符串。</p> </div> <div data-bbox="347 748 794 835" data-label="List-Group"> <ul style="list-style-type: none"> 如果某个字符x连续出现n (n>1) 个，则将这n个字符替换为"nx"的形式；否则保持不变。 Mississippi ==> Mi2si2si2pi </div> <div data-bbox="347 864 767 978" data-label="List-Group"> <ul style="list-style-type: none"> 统计连续重复字符个数n 若n>1，则将n转换为字符后复制至新字符串 复制该字符至新字符串 指针移到下一个待处理字符 </div>	讲解本例的解题思路
45	<div data-bbox="304 1028 807 1424" data-label="Complex-Block"> <div> <div>字符串压缩</div> <pre> #include <stdio.h> #define MAXLINE 80 void zip(char *p); int main(void) { char line[MAXLINE]; printf("Input the string: "); gets(line); zip(line); puts(line); return 0; } </pre> <div>Input the string: Mississippi Mi2si2si2pi</div> </div> <div> <pre> void zip(char *p) { char *q = p; int n; while(*p != '\0'){ n = 1; while(*p == *(p+n)) { n++; } if(n >= 10){ *q++ = (n / 10) + '0'; *q++ = (n % 10) + '0'; } else if(n >= 2){ *q++ = n + '0'; } *q++ = *(p + n - 1); p = p + n; } *q = '\0'; } </pre> </div> </div>	运行演示程序
46	<div data-bbox="304 1442 807 1503" data-label="Section-Header"> <h2>8.4.2 字符串和字符指针</h2> </div> <div data-bbox="355 1529 762 1787" data-label="List-Group"> <ul style="list-style-type: none"> 字符串常量 <ul style="list-style-type: none"> "array" "point" 用一对双引号括起来的字符序列 被看做一个特殊的一维字符数组,在内存中连续存放 实质上是一个指向该字符串首字符的指针常量 <pre> char sa[] = "array"; char *sp = "point"; </pre> </div>	<p>字符串常量的特点：</p> <ol style="list-style-type: none"> 1、是一对双引号括起来的字符序列 2、由'\0'标识结束 3、实质上是一个指向该字符串首字符的指针常量 <p>字符数组和字符指针都可以用来处理字符串：</p> <ol style="list-style-type: none"> 1、如果定义一个字符指针接收字符串常量的值，该指针就指向字符串的首字符 2、注意给出的两种初始化方法

47	<div> <div></div> <pre> char sa[] = "array"; printf("%s ", sa+2); char *sp = "point"; printf("%s ", sp+3); printf("%s\n", "string"+1); printf("%s ", sa); printf("%s ", sp); printf("%s\n", "string"); ray nt tring array point string </pre> <p>数组名sa、指针sp和字符串"string"的值都是地址</p> </div>	明确数组名、指针和字符串常量的值都是地址
48	<div> <div></div> <h3>字符数组与字符指针的重要区别</h3> <pre> char sa[] = "This is a string"; char *sp = "This is a string"; </pre> <div> <div>sa</div> <div>T h i s i s a s t r i n g \0</div> </div> <div> <div>sp</div> <div>→ T h i s i s a s t r i n g \0</div> </div> <p>如果要改变数组sa所代表的字符串，只能改变数组元素的内容</p> <p>如果要改变指针sp所代表的字符串，通常直接改变指针的值，让它指向新的字符串</p> </div>	看图讲述字符数组与字符指针的重要区别
49	<div> <div></div> <h3>示例</h3> <pre> char sa[] = "This is a string"; char *sp = "This is a string"; strcpy (sa, "Hello"); sp = "Hello"; sa = "Hello"; 非法 </pre> <p>数组名是常量，不能对它赋值</p> </div>	给数组名直接赋值是非法的
50	<div> <div></div> <h3>字符指针—先赋值，后引用</h3> <p>定义字符指针后，如果没有对它赋值，指针的值不确定。</p> <pre> char *s ; scanf ("%s", s); </pre> <p>不要引用未赋值的指针</p> <pre> char *s, str[20]; s = str; scanf ("%s", s); </pre> <p>定义指针时，先将它的初值置为空</p> <pre> char *s = NULL </pre> </div>	<p>先定义，再使用 先赋值，再使用</p> <p>良好的习惯：把指针初始化为 NULL</p>





51	<div> <div></div> <h3>8.4.3 常用的字符串处理函数</h3> <ul style="list-style-type: none"> 函数原型在 <code>stdio.h</code> 或 <code>string.h</code> 中给出 <h4>1. 字符串的输入和输出</h4> <ul style="list-style-type: none"> 输入字符串: <code>scanf()</code> 或 <code>gets()</code> 输出字符串: <code>printf()</code> 或 <code>puts()</code> <code>stdio.h</code> </div>	使用这些字符串处理函数别忘记包含相应的头文件
52	<div> <div></div> <h3>字符串的输入</h3> <pre>char str[80]; i = 0; while((str[i] = getchar()) != '\n') i++; str[i] = '\0';</pre> <p>(1) <code>scanf("%s", str)</code> 输入参数: 字符数组名, 不加地址符 遇回车或空格输入结束, 并自动将输入的一串字符和 '\0' 送入数组中</p> <p>(2) <code>gets(str)</code> 遇回车输入结束, 自动将输入的一串字符和 '\0' 送入数组中</p> </div>	gets()可以输入含空格的字符串, 而 scanf()则不能
53	<div> <div></div> <h3>字符串的输出</h3> <pre>char str[80]; for (i = 0; str[i] != '\0'; i++) putchar (str[i]);</pre> <p>(3) <code>printf("%s", str)</code> <code>printf("%s", "hello");</code></p> <p>(4) <code>puts(str)</code> <code>puts("hello");</code> 输出字符串后自动换行</p> <p>输出参数可以是字符数组名或字符串常量, 输出遇 '\0' 结束</p> </div>	puts()输出字符串后会自动换行
54	<div> <div></div> <h3>字符串输入输出函数示例</h3> <div> <pre>#include <stdio.h> int main () { char str[80]; scanf ("%s", str); printf ("%s", str); printf ("%s", "Hello"); return 0; }</pre> <div> <div>Programming</div> <div>ProgrammingHello</div> <div>Programming is fun!</div> <div>ProgrammingHello</div> </div> </div> <div> <pre>#include <stdio.h> int main () { char str[80]; gets (str); puts (str); puts ("Hello"); return 0; }</pre> <div> <div>Programming</div> <div>ProgrammingHello</div> <div>Programming is fun!</div> <div>ProgrammingHello</div> </div> </div> </div>	先让学生回答会有什么样的输出结果, 在给出答案

55	<div data-bbox="300 203 820 241" data-label="Section-Header"> <h2>2、字符串的复制、连接、比较、求字符串长度</h2> </div> <div data-bbox="368 338 715 495" data-label="List-Group"> <ul style="list-style-type: none"> □字符串复制: strcpy(str1, str2) □字符串连接: strcat(str1, str2) □字符串比较: strcmp(str1, str2) □求字符串长度: strlen(str) □string.h </div>	几个常用的字符串处理函数
56	<div data-bbox="300 620 820 689" data-label="Section-Header"> <h2>字符串复制函数strcpy()</h2> </div> <div data-bbox="325 703 496 730" data-label="Text"> <p>strcpy(str1, str2);</p> </div> <div data-bbox="347 730 612 757" data-label="Text"> <p>将字符串 str2 复制到 str1 中</p> </div> <div data-bbox="352 786 791 958" data-label="Text"> <pre>static char str1[20]; static char str2[20] = "happy"; strcpy(str1, str2); str1中 strcpy(str1, "world"); str1中:</pre> </div> <div data-bbox="660 763 791 958" data-label="Diagram"> </div>	<p>可以让学生考虑如果自己编写一个 strcpy() 函数, 应该如何实现</p>
57	<div data-bbox="300 1037 820 1106" data-label="Section-Header"> <h2>strcpy() 示例</h2> </div> <div data-bbox="347 1120 585 1400" data-label="Text"> <pre># include "stdio.h" # include "string.h" int main(void) { char str1[20], str2[20]; gets(str2); strcpy(str1, str2); puts(str1); return 0; }</pre> </div> <div data-bbox="660 1189 715 1240" data-label="Diagram"> </div>	<p>演示例子, 可以看出 strcpy() 相当于字符串的赋值</p> <p>提问为什么不能直接写作: str=str2</p>
58	<div data-bbox="300 1453 820 1523" data-label="Section-Header"> <h2>字符串连接函数strcat</h2> </div> <div data-bbox="325 1527 496 1552" data-label="Text"> <p>strcat (str1, str2);</p> </div> <div data-bbox="325 1552 783 1579" data-label="Text"> <p>连接两个字符串 str1 和 str2, 并将结果放入 str1 中</p> </div> <div data-bbox="319 1581 783 1830" data-label="Text"> <pre># include "stdio.h" # include "string.h" int main (void) { char str1[80], str2[20]; gets (str1); gets (str2); strcat (str1, str2); puts (str1); return 0; }</pre> </div> <div data-bbox="564 1599 732 1706" data-label="Text"> <p>str1中: Let us \0 str2中: go.\0 str1中: Let us go.\0 str2中: go.\0</p> </div> <div data-bbox="564 1720 692 1789" data-label="Diagram"> </div> <div data-bbox="604 1798 798 1825" data-label="Text"> <p>str1=str1+str2 非法!</p> </div>	<p>通过例子说明 strcat() 的作用 相当于字符串的加法</p>

59	<div><div></div><div><h3>字符串比较函数strcmp</h3><p>strcmp(str1, str2)</p><p>比较 两个字符串str1和str2的大小。</p><p>规则：按字典序(ASCII码序)</p><ul style="list-style-type: none">□如果 str1 和 str2 相等，返回 0；□如果 str1 大于 str2，返回一个正整数；□如果 str1 小于 str2，返回一个负整数；<pre>static char s1[20] = "sea"; strcmp(s1, "Sea"); 正整数 strcmp("Sea", "Sea "); 负整数 strcmp("Sea", "Sea"); 0</pre></div></div>	字符串比较大小，注意规则	
60	<div><div></div><div><h3>strcmp () 示例</h3><pre># include <stdio.h> # include <string.h> int main (void) { int res; char s1[20], s2[20]; gets (s1); gets (s2); res = strcmp (s1, s2); printf("%d\n", res); return 0; }</pre><div>1234 2 -1</div></div></div> <div>深入学习提示：如何实现 strcmp()？ 关键代码片段： <pre>char *s1 = str1; char *s2 = str2; char c1, c2; do{ c1 = *s1++; c2 = *s2++; if (c1 == '\0') return c1 - c2; } while (c1 == c2); return c1 - c2;</pre></div>		
61	<div><div></div><div><h3>用strcmp ()比较字符串</h3><p>利用字符串比较函数比较字符串的大小</p><pre>strcmp (str1, str2);</pre><p>为什么定义这样的函数？</p><table><tr><td><pre>str1 > str2 str1 < "hello" str1 == str2</pre></td><td><pre>strcmp (str1, str2) > 0 strcmp (str1, "hello") < 0 strcmp (str1, str2) == 0</pre></td></tr></table><p>比较字符串首元素的地址 比较字符串的内容</p></div></div> <div>对比说明含义</div>	<pre>str1 > str2 str1 < "hello" str1 == str2</pre>	<pre>strcmp (str1, str2) > 0 strcmp (str1, "hello") < 0 strcmp (str1, str2) == 0</pre>
<pre>str1 > str2 str1 < "hello" str1 == str2</pre>	<pre>strcmp (str1, str2) > 0 strcmp (str1, "hello") < 0 strcmp (str1, str2) == 0</pre>		
62	<div><div></div><div><h3>字符串长度函数strlen</h3><ul style="list-style-type: none">■ strlen(str) 计算字符串的有效长度，不包括 '\0'。<pre>static char str[20]="How are you?" strlen ("hello") 的值是： 5 strlen(str) 的值是： 12</pre></div></div> <div>注意：长度不包括'\0'，但数组大小要能存放</div>		

63	<h3>字符串处理函数小结</h3> <table border="1"> <thead> <tr> <th>函数</th><th>功能</th><th>头文件</th></tr> </thead> <tbody> <tr> <td>puts (str)</td><td>输出字符串</td><td rowspan="2">stdio.h</td></tr> <tr> <td>gets (str)</td><td>输入字符串（回车间隔）</td></tr> <tr> <td>strcpy (s1,s2)</td><td>s2 ==> s1</td><td rowspan="4">string.h</td></tr> <tr> <td>strcat (s1,s2)</td><td>s1 "+" s2 ==> s1</td></tr> <tr> <td>strcmp(s1,s2)</td><td>若 s1=="s2，函数值为0 若 s1 ">" s2，函数值>0 若 s1 "<" s2，函数值<0</td></tr> <tr> <td>strlen(str)</td><td>计算字符串的有效长度 不包括 '\0'</td></tr> </tbody> </table>	函数	功能	头文件	puts (str)	输出字符串	stdio.h	gets (str)	输入字符串（回车间隔）	strcpy (s1,s2)	s2 ==> s1	string.h	strcat (s1,s2)	s1 "+" s2 ==> s1	strcmp(s1,s2)	若 s1=="s2 ，函数值为 0 若 s1 ">" s2 ，函数值 >0 若 s1 "<" s2 ，函数值 <0	strlen(str)	计算字符串的有效长度 不包括 '\0'	小结
函数	功能	头文件																	
puts (str)	输出字符串	stdio.h																	
gets (str)	输入字符串（回车间隔）																		
strcpy (s1,s2)	s2 ==> s1	string.h																	
strcat (s1,s2)	s1 "+" s2 ==> s1																		
strcmp(s1,s2)	若 s1=="s2 ，函数值为 0 若 s1 ">" s2 ，函数值 >0 若 s1 "<" s2 ，函数值 <0																		
strlen(str)	计算字符串的有效长度 不包括 '\0'																		
64	<h3>例8-9 求最小字符串</h3> <pre> int main () { int i, n; int x, min; scanf ("%d", &n); scanf ("%d", &x); min = x; for (i = 1; i < n; i++){ scanf ("%d", &x); if (x < min) min = x; } printf ("min is %d\n", min); return 0; } </pre> <p>2 8 -1 99 0 min is -1</p> <pre> char sx[80], smin[80]; scanf ("%d", &n); scanf ("%s", sx); strcpy (smin,sx); for (i = 1; i < n; i++){ scanf ("%s", sx); if (strcmp (sx, smin)<0) strcpy (smin,sx); } printf ("min is %s\n", smin); return 0; } </pre> <p>tool key about zoo sea min is about</p>	<p>先回顾求最小值的程序</p> <p>再对比求最小字符串</p> <p>重点描述字符串处理函数的应用</p>																	
65	<h2>8.5 任意个整数求和 *</h2> <p>例8-10 先输入一个正整数n，再输入任意n个整数，计算并输出这n个整数的和。</p> <p>要求使用动态内存分配方法为这n个整数分配空间。</p>	<p>此节为选学部分</p> <p>首先引入背景问题：为什么需要动态内存分配？</p> <p>然后给出例子，说明和以往数组程序的区别，这里没有要求 n 要小于多少</p>																	
66	<h3>8.5.1 程序解析</h3> <pre> int main () { int n, sum, i, *p; printf ("Enter n: "); scanf ("%d", &n); if ((p = (int *) calloc (n, sizeof(int))) == NULL) { printf ("Not able to allocate memory. \n"); exit(1); } printf ("Enter %d integers: ", n); for (i = 0; i < n; i++){ scanf ("%d", p+i); } sum = 0; for (i = 0; i < n; i++){ sum = sum + *(p+i); } printf ("The sum is %d \n", sum); free (p); return 0; } </pre> <p>Enter n: 10 Enter 10 integers: 3 7 12 54 2 -19 8 -1 0 15 The sum is 81</p>	<p>演示程序，简要说明一下，让学生注意新的内存分配相关函数</p>																	

67	<h3>8.5.2 用指针实现内存动态分配</h3> <ul style="list-style-type: none"> ■ 变量在使用前必须被定义且安排好存储空间 ■ 全局变量、静态局部变量的存储是在编译时确定，在程序开始执行前完成。 ■ 自动变量，在执行进入变量定义所在的复合语句时为它们分配存储，变量的大小也是静态确定的。 ■ 一般情况下，运行中的很多存储要求在写程序时无法确定。 	分析不同种类变量的内存分配机制的差别
68	<h3>动态存储管理</h3> <ul style="list-style-type: none"> ■ 不是由编译系统分配的，而是由用户在程序中通过动态分配获取。 ■ 使用动态内存分配能有效地使用内存 <ul style="list-style-type: none"> □ 使用时申请 □ 用完就释放 <p>同一段内存可以有不同的用途</p>	强调动态内存管理的高效性，一定要用完释放，否则将造成内存泄漏
69	<h3>动态内存分配的步骤</h3> <ol style="list-style-type: none"> (1) 了解需要多少内存空间 (2) 利用C语言提供的动态分配函数来分配所需要的存储空间。 (3) 使指针指向获得的内存空间，以便使用指针在该空间内实施运算或操作。 (4) 当使用完毕内存后，释放这一空间。 	内存分配的典型步骤
70	<h3>动态存储分配函数malloc()</h3> <p>void *malloc(unsigned size) 在内存的动态存储区中分配一连续空间，其长度为size</p> <ul style="list-style-type: none"> □ 若申请成功，则返回一个指向所分配内存空间的起始地址的指针 □ 若申请内存空间不成功，则返回NULL（值为0） □ 返回值类型：(void *) <ul style="list-style-type: none"> ■ 通用指针的一个重要用途 ■ 将malloc的返回值转换到特定指针类型，赋给一个指针 	<p>强调要检查内存分配不成功时候的出错处理</p> <p>难点：讲透(void *)通用指针类型</p> <p>malloc 只负责申请一块大小为 size 的连续内存，但该段内存空间如何划分需要通过强制转化实现。比如 100 字节的内存块可以划分为 50 个整形单元（假设每个整形占 2 个字节），也可以划分为 25 个 float 型单元（假设每个 float 型占 4 个字节）</p>

71	<div data-bbox="300 203 347 237"></div> <h3 data-bbox="331 241 507 275">malloc()示例</h3> <pre data-bbox="323 291 751 427">/* 动态分配n个整数类型大小的空间 */ if ((p = (int *)malloc(n*sizeof(int))) == NULL) { printf("Not able to allocate memory. \n"); exit (1); }</pre> <ul data-bbox="323 434 810 539" style="list-style-type: none"> ■ 调用malloc时，用 <code>sizeof</code> 计算存储块大小 ■ 每次动态分配都要检查是否成功，考虑例外情况处理 ■ 虽然存储块是动态分配的，但它的大小在分配后也是确定的，不要越界使用。 	<p data-bbox="847 197 1353 315">强调用 <code>sizeof</code> 来计算存储块的大小，不要直接填字节数，因为操作系统平台不一样可能同一类型占的字节数不同</p> <p data-bbox="847 365 1070 398">强调不要越界使用</p>
72	<div data-bbox="300 620 347 654"></div> <h3 data-bbox="331 669 730 703">计数动态存储分配函数calloc ()</h3> <pre data-bbox="323 728 707 750">void *calloc(unsigned n, unsigned size)</pre> <p data-bbox="347 754 791 788">在内存的动态存储区中分配n个连续空间，每一存储空间的长度为size，并且分配后还把存储块里全部初始化为0</p> <ul data-bbox="347 797 783 938" style="list-style-type: none"> □ 若申请成功，则返回一个指向被分配内存空间的起始地址的指针 □ 若申请内存空间不成功，则返回NULL □ malloc对所分配的存储块不做任何事情 □ calloc对整个区域进行初始化 	<p data-bbox="847 613 1182 692">对比 malloc() calloc()分配后会初始化为 0</p> <p data-bbox="847 739 1353 857">可以类比宾馆房间分配，malloc()是只负责分配，但里面不打扫的，原来怎样就保持原样；但 calloc()会打扫干净，像新的一样</p>
73	<div data-bbox="300 1037 347 1070"></div> <h3 data-bbox="331 1086 616 1120">动态存储释放函数free</h3> <pre data-bbox="331 1158 539 1180">void free(void *ptr)</pre> <p data-bbox="355 1187 762 1234">释放由动态存储分配函数申请到的整块内存空间，ptr为指向要释放空间的首地址。</p> <p data-bbox="331 1272 767 1321">当某个动态分配的存储块不再用时，要及时将它释放</p>	<p data-bbox="847 1030 1353 1108">强调要及时释放，但不能早释放。释放后就不能再使用了</p>
74	<div data-bbox="300 1453 347 1487"></div> <h3 data-bbox="331 1491 600 1525">分配调整函数realloc</h3> <pre data-bbox="323 1534 687 1556">void *realloc(void *ptr, unsigned size)</pre> <p data-bbox="355 1561 512 1583">更改以前的存储分配</p> <ul data-bbox="355 1583 783 1830" style="list-style-type: none"> □ ptr必须是以前通过动态存储分配得到的指针 □ 参数size为现在需要的空间大小 □ 如果调整失败，返回NULL，同时原来ptr指向存储块的内容不变。 □ 如果调整成功，返回一片能存放大小为size的区块，并保证该块的内容与原块的一致。如果size小于原块的大小，则内容为原块前size范围内的数据；如果新块更大，则原有数据存在新块的前一部分。 □ 如果分配成功，原存储块的内容就可能改变了，因此不允许再通过ptr去使用它。 	<p data-bbox="847 1447 1334 1480">分析各种可能性，讲述 realloc 执行策略</p>



本章小结

■ 指针的概念与定义

- 变量、内存单元与地址
- 指针变量的定义与初始化

■ 指针作为函数参数

- 通过指针参数使函数返回

■ 指针与数组

- 指针、数组与地址的关系
- 数组名作为函数参数

■ 指针与字符串

- 常用字符串处理函数

■ *指针实现内存动态分配

•能够掌握指针概念，定义指针变量和指针基本运算
•能够掌握指针作为函数的参数进行熟练编程，通函数调用改变主调函数变量的值
•作为函数参数进行熟练编程，并能利用指针进行数组相关操作
•能够使用字符串常用处理函数进行编程，并能使用字符指针进行字符串相关操作
•了解通过指针实现动态内存分配，并能进行编程

回顾本章，总结要点

8.3 练习与习题答案

8.3.1 练习参考答案

8-1 对于如下变量定义及初始化，与 $m = n$ 等价的表达式是 B。

```
int m, n = 5, *p = &m;
```

- A. $m = *p$ B. $*p = n$ C. $m = \&n$ D. $*p = m$

8-2 执行以下程序段后， $*p$ 的值为 2。

```
int m = 1, *p = &m, *q;
```

```
q = p; *q = 2;
```

8-3 计算两数的和与差。要求自定义函数 `sum_diff(double op1, double op2, double *psum, double *pdiff)`，实现计算两个数的和与差，其中 `op1` 和 `op2` 是需要计算的两个数，`psum` 和 `pdiff` 指向的变量保存计算得出的和与差。

解答：

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

```
int main (void)
```

```
{
```

```
    double op1, op2, sum, diff;
```

```
    void sum_diff(double op1, double op2, double *psum, double *pdiff);
```

```
    printf("input op1 and op2: ");
```

```
    scanf("%lf%lf", &op1, &op2);
```

```
    sum_diff(op1, op2, &sum, &diff);
```

```
    printf("%lf+%lf=%lf; %lf-%lf=%lf\n", op1, op2, sum, op1, op2, diff);
```

```
    return 0;
```

```
}
```

```
void sum_diff(double op1, double op2, double *psum, double *pdiff)
```

```
{
```

```
    *psum = op1 + op2;
```

```
    *pdiff = op1 - op2;
```

```
}
```

8-4 根据表 8.2 所示，这组数据的冒泡排序其实循环到第 6 遍（即 $n-2$ ）时就已经排好序了，说明有时候并不一定需要 $n-1$ 次循环。请思考如何改进冒泡排序算法并编程实现（提示：当发现一遍循环后没有数据发生交换，说明已经排好序了）。

解答：设置一个标志变量 **flag**，进入一轮循环前设置为 0，在循环中有发生数据交换就改写 **flag** 值为 1。当该轮循环结束后检查 **flag** 值，如果变为 1 说明发生了数据交换，还没有排好序，如果为 0 说明没有发生交换，已经排好序。

```
#include <stdio.h>
void bubble (int a[ ], int n);
int main(void)
{
    int n, a[8];
    int i;
    printf("Enter n (n<=8): ");
    scanf("%d", &n);
    printf("Enter a[%d] : ",n);
    for (i=0; i<n;i++)
        scanf("%d",&a[i]);
    bubble(a,n);
    printf("After sorted, a[%d] = ", n);
    for (i=0; i<n; i++)
        printf("%3d",a[i]);
    return 0;
}

void bubble (int a[ ], int n)      /* n 是数组 a 中待排序元素的数量 */
{
    int i, j, t, flag;
    for( i = 1; i < n; i++ ) {      /* 外部循环 */
        flag=0;
        for( j = 0; j < n-i; j++ )  /* 内部循环 */
            if (a[j] > a[j+1]){      /* 比较两个元素的大小 */
                t=a[j]; a[j]=a[j+1]; a[j+1]=t; /* 如果前一个元素大，则交换 */
                flag=1;              /* 发生交换，flag 置为 1 */
            }
        if (flag==0)                /* 如果一轮循环没有发生数据交换，排序结束*/
            break;
    }
}
```

8-5 重做例 8-5，要求使用选择排序算法。

解答：

```
#include <stdio.h>
void bubble (int a[ ], int n);
int main(void)
```

```

{
    int n, a[8];
    int i;
    printf("Enter n (n<=8): ");
    scanf("%d", &n);
    printf("Enter a[%d] : ",n);
    for (i=0; i<n;i++)
        scanf("%d",&a[i]);

    bubble(a,n);

    printf("After sorted, a[%d] = ", n);
    for (i=0; i<n; i++)
        printf("%3d",a[i]);

    return 0;
}

void bubble (int a[ ], int n)      /* n 是数组 a 中待排序元素的数量 */
{
    int i, j, t, index;
    for( i = 0; i < n-1; i++ ) {          /* 外部循环 */
        index=i;
        for (j = i+1; j < n; j++ )        /* 内部循环 */
            if (a[j] < a[index])
                index = j;
        t=a[i]; a[i]=a[index]; a[index]=t;
    }
}

```

8-6 在使用 `scanf()` 函数时，输入参数列表需要使用取地址操作符 `&`，但当参数为字符数组名时并没有使用，为什么？如果在字符数组名前加上取地址操作符 `&`，会发生什么？

解答：

因为字符数组名的值是一个特殊的固定地址，可以看作是常量指针，因此不需要再使用取地址符来获取该数组的地址。

如果在字符数组名 `str` 前加上取地址操作符 `&`，那么对其取地址 `&str` 可以看做是这个数组的第一个元素的地址，由于数组地址和数组第一个元素的地址相同，所以 `&str` 表示地址值和 `str` 表示的地址值是相等的。对 `scanf()` 的变长参数列表的话，编译器只负责参数传递，怎么解释后边的几个地址的含义，是由前边的字符串确定的。所以使用 `scanf("%s",str)` 和 `scanf("%s",&str)` 都能通过编译且正常执行。

8-7 C 语言不允许用赋值表达式直接对数组赋值，为什么？

解答：

数组名可以看作是常量指针，因为不可以对一个常量进行赋值，所以不允许用赋值表达式直接对数组进行赋值。

8-8 输入一个字符串，把该字符串的前 3 个字母移到最后，输出变换后的字符串。比如输入“abcdef”，输出为“defabc”。

解答：

```
#include <stdio.h>
#include <string.h>
#define MAXS 10
void Shift( char s[] );
void GetString( char s[] );
int main()
{
    char s[MAXS];
    GetString(s);
    Shift(s);
    printf("%s\n", s);
    return 0;
}
void Shift( char s[] )
{
    char str[4];
    int i;
    for (i=0;i<3;i++)
        str[i]=s[i];
    str[i]='\0';
    for(i=3; s[i]!='\0';i++)
        s[i-3]=s[i];
    s[i-3]= '\0';
    strcat(s,str);
}
void GetString( char s[] )
{
    gets(s);
}
```

8-9 使用动态内存分配的方法实现例 8-5 的冒泡排序。

解答：

```
#include <stdio.h>
#include <stdlib.h>
void bubble (int a[ ], int n);
int main(void)
{
    int n, *p;
    int i;

    printf("input n : ");
```

```

scanf("%d", &n);

/*为数组 p 动态分配 n 个整数类型大小的空间 */
if ((p=(int *)calloc(n, sizeof(int))) == NULL) {
    printf("Not able to allocate memory. \n");
    exit(1);
}
printf("input %d integers : ",n);
for (i=0; i<n;i++)
    scanf("%d",p+i);
bubble(p,n);
printf("After sorted : ");
for (i=0; i<n; i++)
    printf("%3d",*(p+i));
free(p);
return 0;
}

void bubble (int a[ ], int n)      /* n 是数组 a 中待排序元素的数量 */
{
    int i, j, t;
    for( i = 1; i < n; i++ )      /* 外部循环 */
        for( j = 0; j < n-i; j++ ) /* 内部循环 */
            if (a[j] > a[j+1])    /* 比较两个元素的大小 */
                t=a[j]; a[j]=a[j+1]; a[j+1]=t; /* 如果前一个元素大，则交换 */
}

```

8.3.2 习题参考答案

一. 选择题

1	2	3	4	5	6	7	8
B	C	A	D	C	B	A	B

二. 填空题

- | | |
|--|---|
| <p>1 a, 10, &maxsub, &minsub
 *maxsub = i
 *minsub = i_</p> <hr/> <p>s
 p++
 s+n-1</p> <p>3 q--
 p<q
 q--</p> <hr/> | <p>2 (i< *n)
 j>=i
 a[j+1]
 x</p> <hr/> <p>4 q = p;
 p > str
 *p = *(p - 1);</p> <hr/> |
|--|---|

```
cnt == 3
5  k++;
   cnt = 0;
```

三、程序设计题

1. 拆分实数的整数与小数部分：要求自定义一个函数 `void splitfloat(float x, int *intpart, float *fracpart)`，其中 `x` 是被拆分的实数，`*intpart` 和 `*fracpart` 分别是将实数 `x` 拆分出来的整数部分与小数部分。编写主函数，并在其中调用函数 `splitfloat()`。试编写相应程序。

解答：

```
#include <stdio.h>

void splitfloat(float x, int *intpart, float *fracpart)
{
    *intpart = (int) x;
    *fracpart = x - *intpart;
}

int main()
{
    float x, fracpart;
    int intpart;

    scanf("%f", &x);
    splitfloat(x, &intpart, &fracpart);
    printf("The integer part is %d\n", intpart);
    printf("The fractional part is %g\n", fracpart);
    return 0;
}
```

2. 在数组中查找指定元素：输入一个正整数 `n` ($1 < n \leq 10$)，然后输入 `n` 个整数存入数组 `a` 中，再输入一个整数 `x`，在数组 `a` 中查找 `x`，若找到则输出相应的下标，否则显示“Not found”。要求定义和调用函数 `search(int list[], int n, int x)`，在数组 `list` 中查找元素 `x`，若找到则返回相应下标，否则返回-1，参数 `n` 代表数组 `list` 中元素的数量。试编写相应程序。

解答：

```
#include <stdio.h>

int main(void)
{
    int i, n, res, x;
    int a[10];
    int search(int list[], int n, int x);
    scanf("%d", &n);
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    scanf("%d", &x);
    res = search(a, n, x);
```

```

        if(res != -1)
            printf("index = %d\n", res);
        else
            printf("Not found\n");
    }
}

int search(int list[], int n, int x)
{
    int i;
    for(i = 0; i < n; i++)
        if(list[i] == x) return i;
    return -1;
}

```

3. 循环后移：有 n 个整数，使前面各数顺序向后移 m 个位置，移出的数再从开头移入。编写一个函数实现以上功能，在主函数中输入 n 个整数并输出调整后的 n 个数。试编写相应程序。

解答：

```

#include <stdio.h>
void mov(int *x, int n, int m);
int main(void)
{
    int i, m, n;
    int a[80];
    scanf("%d%d", &n, &m);
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    mov(a, n, m);
    printf("After move: ");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

```

/*函数 mov(x,n,m) 实现循环移动的功能，函数形参 x 的类型是整型指针，形参 n 和 m 的类型是 `int`，函数的类型是 `void`。*/

```

void mov(int *x, int n, int m)
{
    int i, j, k;
    for(i = 1; i <= m; i++){
        k = x[n-1];
        for(j = n - 1; j > 0; j--){
            x[j] = x[j-1];
        }
        x[0] = k;
    }
}

```

4. 报数：有 n 个人围成一圈，按顺序从 1 到 n 编好号。从第一个人开始报数，报到 m ($m < n$) 的人退出圈子，下一个人从 1 开始报数，报到 m 的人退出圈子。如此下去，直到留下最后一个人。输入整数 n 和 m ，并按退出顺序输出退出圈子的人的编号。试编写相应程序。

解答：

```
#include <stdio.h>
#define MAXN 20
void CountOff( int n, int m, int out[] );
int main()
{
    int out[MAXN], n, m;
    int i;
    scanf("%d %d", &n, &m);
    CountOff( n, m, out );
    for ( i = 0; i < n; i++ )
        printf("%d ", out[i]);
    printf("\n");
    return 0;
}

void CountOff( int n, int m, int out[] )
{ int i, count, no, num[MAXN];
  int *p=num;
  count = no = 0;
  for(i = 0; i < n; i++)
      num[i] = i + 1;
  while(no < n-1){
      if(*p != 0)    count++;
      if(count == m){
          no++;
          out[no-1]= *p;
          *p = 0;
          count = 0;
      }
      p++;
      if(p == num + n)
          p = num;
  }
  p = num;
  while(*p == 0)
      p++;
  out[no]= *p;
}
```

5. 使用函数实现字符串复制：输入一个字符串 t 和一个正整数 m ，将字符串 t 中从第 m

个字符开始的全部字符复制到字符串 s 中，再输出字符串 s。要求自定义并调用函数 void strncpy(char *s, char *t, int m)。试编写相应程序。

解答：

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s[80], t[80];
    int m;
    void strncpy(char *s, char *t, int m);
    gets(t);
    scanf("%d", &m);
    getchar();
    if(strlen(t) < m)
        printf("error input");
    else{
        strncpy(s, t, m);
        puts(s);
    }
}
void strncpy(char *s, char *t, int m)
{
    t = t + m - 1;
    while(*t != '\0'){
        *s = *t;
        s++;
        t++;
    }
    *s = '\0';
}
```

6. 删除字符： 输入一个字符串，再输入一个字符 ch，将字符串中所有的 ch 字符删除后输出该字符串。要求定义和调用函数 delchar(s, c)，该函数将字符串 s 中出现的所有 c 字符删除。试编写相应程序。

解答：

```
#include<stdio.h>
void main()
{
    char c;
    char str[80];
    void delchar(char *str, char c);
    gets(str);
    scanf("%c", &c);
    getchar();
}
```

```

        delchar(str, c);
        printf("result: ");
        puts(str);
    }
void delchar(char *str, char c)
{
    int i, j;
    i = j = 0;
    while(str[i] != '\0'){
        if(str[i] != c){
            str[j] = str[i];
            j++;
        }
        i++;
    }
    str[j] = '\0';
}

```

7. 字符串排序：输入 5 个字符串，按由小到大的顺序输出。试编写相应程序。

解答：

```

#include <stdio.h>
#include <string.h>
int main( )
{
    int i, j, index;
    char sx[5][80], stemp[80];
    for(i=0; i<5; i++)
        scanf("%s", sx[i]);
    for(i=0; i<4; i++){
        index=i;
        for(j=i+1; j<5; j++)
            if (strcmp(sx[j], sx[index])<0)
                index=j;
        strcpy(stemp, sx[i]);
        strcpy(sx[i], sx[index]);
        strcpy(sx[index], stemp);
    }
    printf("After sorted:\n");
    for (i=0; i<5; i++)
        puts(sx[i]);
    return 0;
}

```

8. 判断回文：判断输入的一串字符是否为“回文”。所谓“回文”是指顺读和倒读都一样的字符串。如“XYZYX”和“xyzzyx”都是回文。试编写相应程序。

解答：

```
#include <stdio.h>
void main()
{
    char s[80];
    int mirror(char *p);
    gets(s);
    if(mirror(s) != 0)
        printf("YES\n");
    else
        printf("NO\n");
}
int mirror(char *p)
{
    char *q;
    q = p;
    while(*q != '\0')
        q++;
    q--;
    while(p < q){
        if(*p != *q)
            return 0;
        p++;
        q--;
    }
    return 1;
}
```

9. 分类统计字符个数：输入一行文字，统计其中的大写字母、小写字母、空格、数字以及其他字符各有多少。试编写相应程序。

解答：

```
#include<stdio.h>
void main()
{
    char s[80];
    char *p;
    int blank, digit, lower, other, upper;
    gets(s);
    upper = lower = blank = digit = other = 0;
    for(p = s; *p != '\0'; p++)
        if(*p >= 'A' && *p <= 'Z')
            upper++;
}
```



```

        else if(*p >= 'a' && *p <= 'z')
            lower++;
        else if(*p >= '0' && *p <= '9')
            digit++;
        else if(*p == ' ')
            blank++;
        else
            other++;
    printf("upper: %d lower: %d blank: %d digit: %d other: %d\n", upper, lower, blank, digit,
other);
}

```

10. （选做）输出学生成绩（动态分配）：输入学生人数后输入每个学生的成绩，最后输出学生的平均成绩、最高成绩和最低成绩。要求使用动态内存分配来实现。试编写相应程序。

解答：

```

#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int n, i;
    float *p, sum, max, min, avg;
    printf("input students' number n: ");
    scanf("%d", &n);
    /*为数组 p 动态分配 n 个浮点数 float 类型大小的空间 */
    if ((p=(float *)calloc(n, sizeof(float))) == NULL) {
        printf("Not able to allocate memory. \n");
        exit(1);
    }
    sum=0.0;
    max=-1;    /*初始化 */
    min=1000;
    printf("input %d students' scores: ", n);    /* 提示输入 n 个整数 */
    for (i = 0; i < n; i++){
        scanf("%f", p+i);
        sum=sum+*(p+i);
        if (min>*(p+i))    min=*(p+i);
        if (max<*(p+i))    max=*(p+i);
    }
    avg=sum/n;
    printf("The avg is %f, max is %f, min is %f\n",avg,max,min);
    free(p);    /* 释放动态分配的空间 */
    return 0;
}

```

8.4 实验指导教材参考答案

8.1 指针与数组

一、调试示例

利用指针找数组最大值：输入 n ($n \leq 10$) 个整数并存入数组中，利用指针操作数组元素找出最大值，输出到屏幕上。

解答：略

二、基础编程题

(1) 利用指针找最大值：输入 2 个整数 a 和 b ，输出其中的最大值。自定义一个函数 `void findmax(int *px, int *py, int *pmax)`，其中 px 和 py 是用户传入的两个整数的指针，函数 `findmax()` 找出两个指针所指向的整数中的最大值，并存放在 $pmax$ 指向的位置。自定义主函数，并在其中调用函数 `findmax()`，试编写相应程序。

解答：

```
#include <stdio.h>
int main (void)
{
    int x, y, max;
    void findmax( int *px, int *py, int *pmax );
    printf("input x and y: ");
    scanf("%d%d", &x, &y);
    findmax (&x, &y, &max);
    printf("max:%d\n", max);
    return 0;
}
void findmax( int *px, int *py, int *pmax )
{
    if(*px>*py)
        *pmax = *px;
    else
        *pmax = *py;
}
```

(2) 计算两数的和与差：输入 2 个实数 x 和 y ，输出和与差。自定义一个函数 `void sum_diff(float op1, float op2, float *psum, float *pdiff)`，其中 $op1$ 和 $op2$ 是输入的两个实数， $*psum$ 和 $*pdiff$ 是计算得出的和与差。自定义主函数，并在其中调用函数 `sum_diff()`，试编写相应程序。

解答：参见练习 8-3。

(3) 拆分实数的整数与小数部分：输入一个实数 x ($0 \leq x < 10000$)，输出其整数和小数。自定义一个函数 `void splitfloat(float x, int *intpart, float *fracpart)`，其中 x 是被拆分的实数， $*intpart$ 和 $*fracpart$ 分别是将实数 x 拆分出来的整数部分与小数部分。自定义主函数，并在其中调用 `splitfloat()` 函数。试编写相应程序。

解答：参见习题程序设计第 1 题。

(4) 使用函数的选择法排序：输入一个正整数 $n(0 < n \leq 10)$ ，再输入 n 个整数存入数组 a 中，用选择法对数组 a 中的元素升序排序后输出。自定义一个函数 `void sort(int a[], int n)`，用选择法对数组 a 中的元素升序排序。自定义主函数，并在其中调用 `sort()` 函数。试编写相应程序。

解答：参见练习 8-5 题。

(5) 在数组中查找指定元素：输入一个正整数 $n(0 < n \leq 10)$ ，然后输入 n 个整数存入数组 a 中，再输入一个整数 x ，在数组 a 中查找 x ，如果找到则输出相应元素的最小下标（下标从 0 开始），否则输出“Not found”。要求定义并调用函数 `search(list, n, x)`，它的功能是在数组 `list` 中查找元素 x ，若找到则返回相应元素的最小下标（下标从 0 开始），否则返回 -1。试编写相应程序。

解答：参见习题程序设计第 2 题。

三、改错题

数组循环后移：输入 2 个正整数 $n(0 < n \leq 10)$ 和 $m(m \geq 0)$ ，然后输入 n 个整数存入数组 a 中，将每个整数循环向右移 m 个位置，即将最后 m 个数循环移至最前面的 m 个位置，最后输出移位后的 n 个整数。自定义函数 `void mov(int a[], int n, int m)` 实现循环右移。

错误行号： 10	正确语句： <code>scanf("%d", p++);</code>
行号： 26	增加语句： <code>temp = a[n-1];</code>
错误行号： 30	正确语句： <code>a[0] = temp;</code>

四、拓展编程题

(1) 报数：有 n 个人围成一圈，按顺序从 1 到 n 编好号。从第一个人开始报数，报到 3 的人退出圈子，下一个人从 1 开始重新报数，报到 3 的人退出圈子。如此下去，直到留下最后一个人。问留下来的人的编号。试编写相应程序。

解答：参见习题程序设计第 4 题。

(2) 动态内存分配方式计算学生成绩：输入学生人数后输入每个学生的成绩，最后输出学生的平均成绩、最高成绩和最低成绩。要求使用动态内存分配来实现。试编写相应程序。

解答：参见习题程序设计第 10 题。

8.2 指针与字符串

一、调试示例

找最小的字符串：输入 5 个字符串（每个字符串的长度小于 80），输出其中最小的字符串。

解答：略

二、基础编程题

(1) 找最长字符串：输入 n 个字符串，输出其中最长的字符串。调用函数 `scanf()` 输入字符串，试编写相应程序。

解答：

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

```

int main(void)
{
    char sx[80], longest[80];
    int i, n;
    scanf("%d", &n);
    scanf("%s", sx);
    strcpy(longest, sx);
    for(i = 1; i < n; i++){
        scanf("%s", sx);
        if(strlen(longest) < strlen(sx))
            strcpy(longest, sx);
    }
    printf("The longest is: %s\n", longest);
    return 0;
}

```

(2) 删除字符：输入一个字符串 *s*，再输入一个字符 *c*，将字符串 *s* 中出现的所有字符 *c* 删除。要求定义并调用函数 `delchar(s,c)`，它的功能是将字符串 *s* 中出现的所有 *c* 字符删除。试编写相应程序。

解答：参见习题程序设计第 6 题。

(3) 使用函数实现字符串部分复制：输入一个字符串 *t* 和一个正整数 *m*，将字符串 *t* 中从第 *m* 个字符开始的全部字符复制到字符串 *s* 中，再输出字符串 *s*。要求用字符指针定义并调用函数 `strncpy(s,t,m)`，它的功能是将字符串 *t* 中从第 *m* 个字符开始的全部字符复制到字符串 *s* 中。试编写相应程序。

解答：参见习题程序设计第 5 题。

(4) 判断回文字符串：判断输入的一串字符是否为“回文”。所谓“回文”，是指顺读和倒读都一样的字符串。如“XYZYX”和“xyzyx”都是回文。试编写相应程序。

解答：参见习题程序设计第 8 题。

(5) 分类统计字符个数：输入一行字符，统计其中的大写字母、小写字母、空格、数字以及其他字符的个数。试编写相应程序。

解答：参见习题程序设计第 9 题。

三、改错题

连接字符串：输入两个字符串 *s* 和 *t*，将字符串 *s* 连接到字符串 *t* 的尾部，再输出字符串 *t*。要求定义和调用函数 `strcat(s, t)` 完成字符串的连接，将字符串 *s* 连接到字符串 *t* 的尾部。

错误行号： 2 正确语句： void strcat(char *s, char *t);
 错误行号： 16 正确语句： void strcat(char *s, char *t)
 错误行号： 22 正确语句： s++, t++;

四、拓展编程题

(1) 字符串排序：输入 *n* 个字符串，按由小到大的顺序输出。调用函数 `scanf()` 输入字符串，

试编写相应程序。

解答：

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    int i, j, index, n;
    char sx[100][80], stemp[80];
    scanf("%d", &n);
    for(i=0; i<n; i++)
        scanf("%s", sx[i]);
    for(i=0; i<n-1; i++){
        index=i;
        for(j=i+1; j<n; j++)
            if (strcmp(sx[j],sx[index])<0)
                index=j;
        strcpy(stemp,sx[i]);
        strcpy(sx[i],sx[index]);
        strcpy(sx[index],stemp);
    }
    printf("After sorted:\n");
    for (i=0;i<n;i++)
        puts(sx[i]);
    return 0;
}
```

(2) 长整数转化成 16 进制字符串：设计一个函数 `void f(long int x, char *p)`，其中 `x` 是待转化的十进制长整数，`p` 指向某个字符数组的首元素。函数的功能是把转换所得的 16 进制字符串写入 `p` 所指向的数组。设计函数 `main()`，输入一个长整数，调用 `f()`，输出 16 进制结果。试编写相应程序。

解答：

```
#include <stdio.h>
void f(long int x, char *p) {
    int i = 0, j;
    unsigned long int y;
    if (x == 0) {
        p[0] = '0';
        p[1] = '\0';
        return;
    }
    if (x < 0) {
        *(p++) = '-';
        y = -x;
    } else {
```

```

        y = x;
    }
    while (y) {
        int t = y % 16;
        p[i++] = t < 10 ? '0' + t : 'A' + t - 10;
        y /= 16;
    }
    p[i] = '\0';
    for (j = 0, i = i - 1; j < i; j++, i--) {
        char t = p[j]; p[j] = p[i]; p[i] = t;
    }
}

int main() {
    long int x;
    char s[100] = "";
    scanf("%ld", &x);
    f(x, s);
    puts(s);
    return 0;
}

```

(3) **IP 地址转换**：一个 IP 地址是用四个字节（每个字节 8 个位）的二进制码组成。输入 32 位二进制字符串，输出十进制格式的 IP 地址。所输出的十进制 IP 地址由 4 个十进制数组成（分别对应 4 个 8 位的二进制数），中间用圆点分隔开。试编写相应程序。

解答：

```

#include <stdio.h>

int main() {
    int i, j, x;
    for (i = 0; i < 4; i++) {
        x = 0;
        for (j = 0; j < 8; j++) {
            x = (x << 1) | (getchar() - '0');
        }
        if (i) printf(".");
        printf("%d", x);
    }
    printf("\n");
    return 0;
}

```