

# 第 11 章 指针进阶

## 11.1 教学要点

本章主要介绍指针数组的概念与应用、指向指针的指针（二级指针）的概念与操作、命令行参数的使用、指针作为函数的返回值、函数指针的使用、单向链表的概念与基本操作（包括建立、增加、删除、遍历）等知识。其中，重点内容是指针数组的使用、指针作为函数返回值、单向链表的基本操作。而指针数组与二级指针的概念与操作、单向链表的概念与基本操作是教学难点内容。

11.1 节通过示例程序“奥运五环色”，引出指针数组的概念，介绍了指向指针的指针（二级指针）的概念、操作以及与指针数组的关系，并进一步介绍和分析了使用指针数组处理多个字符串的情况，另外，还介绍了命令行参数的使用方法。

11.2 节通过示例程序“字符定位”，介绍指针作为函数返回值的相关知识，并进一步讨论了在此时使用动态存储分配的改进方法。另外，还介绍了函数指针的概念，并通过示例程序详细介绍其使用方法和步骤。

11.3 节通过示例程序“用链表构建学生信息库”，介绍了单向链表的基本概念，链表结点的结构定义方法，以及单向链表的建立、遍历，链表结点的增加和删除等基本操作。

讲授学时：4 学时，实验学时同讲授学时。

本章的知识能力结构图见图 11.1。

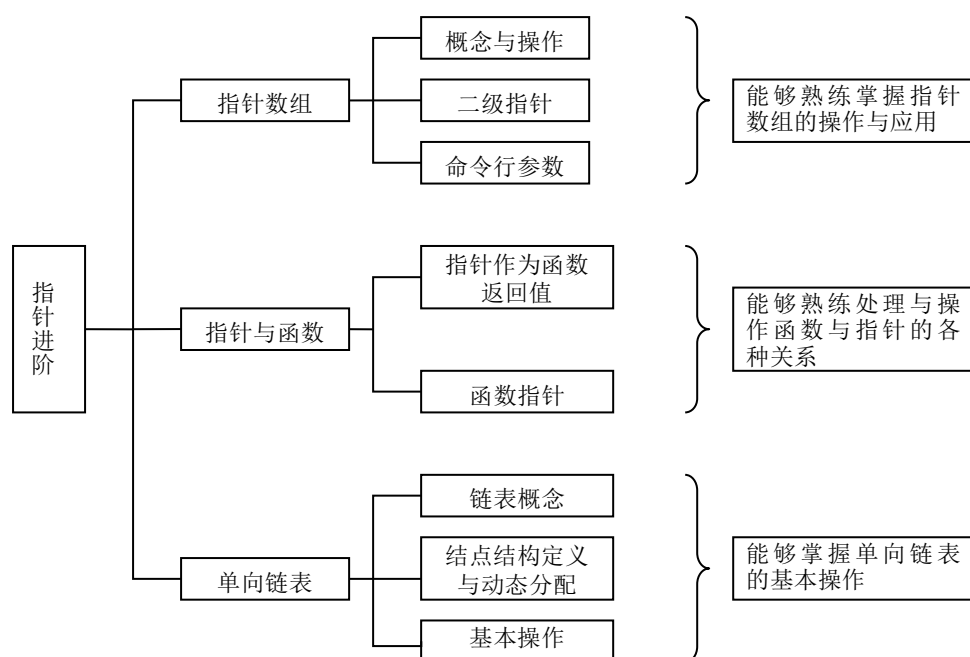






图 11.1 知识能力结构图

## 11.2 讲稿





1	 <b>Chap 11 指针进阶</b>  11.1 奥运五环色 11.2 字符定位 11.3 用链表构建学生信息库	本章分 3 节。
2	 <b>本章要点</b> <ul style="list-style-type: none"><li>■ 指针数组和指向指针的指针是如何被定义和使用的?</li><li>■ 指针如何作为函数的返回值?</li><li>■ 指向函数的指针的意义是什么?</li><li>■ 什么是结构的递归定义, 哪种应用需要这种定义方法?</li><li>■ 对链表这种数据结构, 如何使用动态内存分配操作?</li><li>■ 如何建立单向链表并实现插入、删除以及查找操作?</li></ul>	提出本章的学习要点。
3	 <b>11.1 奥运五环色</b>  11.1.1 程序解析 11.1.2 指针数组的概念 11.1.3 指向指针的指针 11.1.4 用指针数组处理多个字符串 *11.1.5 命令行参数	向学生介绍本节的主要教学内容, 其中 11.1.5 小节为选学内容。
4	 <b>11.1.1 程序解析</b> <ul style="list-style-type: none"><li>■ <b>【例11-1】</b> 已知奥运五环的5种颜色的英文单词按一定顺序排列, 输入任意一个颜色的英文单词, 从已有颜色中查找并输出该颜色的位置值, 若没有找到, 则输出“<b>Not Found</b>”。</li></ul>	介绍奥运五环的 5 种颜色, 每个英文单词是一个字符串。

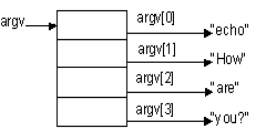
5	<div data-bbox="300 197 614 257"> <h3>11.1.1 程序解析</h3> </div> <div data-bbox="327 271 702 555"> <pre>#include&lt;stdio.h&gt; #include&lt;string.h&gt; int main(void) {     int i;     char *color[5] = {"red", "blue", "yellow", "green", "black"};     char str[20];     printf("Input a color:");     scanf("%s", str);     for(i = 0; i &lt; 5; i++)         if(strcmp(str, color[i]) == 0) /* 比较颜色是否相同 */             break;     if(i &lt; 5)         printf("position:%d\n", i+1);     else         printf("Not Found\n");     return 0; }</pre> <div data-bbox="486 293 560 315">指针数组</div> </div> <div data-bbox="619 241 782 383"> <p>运行结果1 Input a color:yellow position:3</p> <p>运行结果2 Input a color:purple Not Found</p> </div>	<p>运行演示示例程序。</p> <p>分析示例程序代码，指出 color 即指针数组，用于指向 5 个字符串，具体概念及操作后面再介绍。</p> <p>两个英文单词（字符串）进行比较时，需要调用函数 strcmp()。</p>
6	<div data-bbox="300 568 630 645"> <h3>11.1.2 指针数组的概念</h3> </div> <div data-bbox="327 683 782 862"> <ul style="list-style-type: none"> <li>■ C语言中的数组可以是任何类型，如果数组的各个元素都是指针类型，用于存放内存地址，那么这个数组就是<b>指针数组</b>。</li> <li>■ 一维指针数组定义的一般格式为： 类型名 *数组名[数组长度]</li> </ul> </div>	<p>介绍指针数组的概念，定义一维指针数组的语法格式。</p>
7	<div data-bbox="300 987 630 1064"> <h3>11.1.2 指针数组的概念</h3> </div> <div data-bbox="359 1093 686 1301"> <ul style="list-style-type: none"> <li>■ int a[10]; <ul style="list-style-type: none"> <li>□ a是一个数组，它有10个元素</li> <li>□ 每个元素的类型都是整型</li> </ul> </li> <li>■ char *color[5]; <ul style="list-style-type: none"> <li>□ color是一个数组，它有5个元素</li> <li>□ 每个元素的类型都是字符指针</li> </ul> </li> </ul> </div>	<p>通过对比，举例说明一维指针数组的定义方法及其含义。</p>
8	<div data-bbox="300 1406 630 1482"> <h3>11.1.2 指针数组的概念</h3> </div> <div data-bbox="311 1507 798 1637"> <pre>char *color[5] = {"red", "blue", "yellow", "green", "black"};</pre> <ul style="list-style-type: none"> <li>□ color是一个数组，它有5个元素</li> <li>□ 每个元素的类型都是字符指针</li> <li>□ 数组元素可以处理字符串</li> </ul> <p>对指针数组元素的操作：</p> <ul style="list-style-type: none"> <li>□ printf("%s %x\n", color[i], color[i]);</li> </ul> </div> <div data-bbox="603 1541 798 1619"> <p>对指针数组元素的操作和对同类型指针变量的操作相同</p> </div> <div data-bbox="422 1653 638 1771"> <pre> graph LR     subgraph color_array [color]         direction TB         color0[color[0]]         color1[color[1]]         color2[color[2]]         color3[color[3]]         color4[color[4]]     end     color0 --&gt; red0[red]     color1 --&gt; blue0[blue]     color2 --&gt; yellow0[yellow]     color3 --&gt; green0[green]     color4 --&gt; black0[black] </pre> </div>	<p>结合图示进一步说明例 11-1 中指针数组 color 的含义，以及指针数组元素的引用方法。</p> <p>特别指出，指针数组的每个元素就是一个指针变量，与同类型的指针变量的操作方法相同。</p>

9	<div data-bbox="300 203 347 241" data-label="Image"></div> <h3>11.1.2 指针数组的概念</h3> <p>■ 继续执行：</p> <pre>char * tmp; tmp = color[0]; color[0] = color[4]; color[4] = tmp;</pre> <div data-bbox="555 297 794 450" data-label="Diagram"> <p>color[0]与color[4]交换后的情况</p> </div> <div data-bbox="347 461 780 568" data-label="Text"> <p>指针数组操作时：</p> <ul style="list-style-type: none"> <li>可以直接对数组元素进行引用操作 tmp=color[0];</li> <li>也可以间接访问操作数组元素所指向的单元内容 printf ("%c", *(color[0]+1) );</li> </ul> </div>	<p>结合图示分析 color[0]与 color[4]交换前后的情况，说明指针数组元素的值是地址值，相互交换后实际上是交换了指向关系。</p> <p>进一步说明对指针数组元素操作的两个层面：指针数组元素本身、指针数组元素指向的内容。</p>
10	<div data-bbox="300 660 347 698" data-label="Image"></div> <h3>11.1.3 指向指针的指针</h3> <p>■ C语言中，指向指针的指针(二级指针)一般定义为：</p> <p>类型名 **变量名</p> <pre>int a = 10; int *p = &amp;a; int **pp = &amp;p;</pre> <div data-bbox="459 936 794 1021" data-label="Diagram"> </div>	<p>介绍二级指针变量的定义格式、赋值方法及其含义。</p>
11	<div data-bbox="300 1077 347 1115" data-label="Image"></div> <p>int a = 10, b = 20, t; int *pa = &amp;a, *pb = &amp;b, *pt; int **ppa = &amp;pa, **ppb = &amp;pb, **ppt;</p> <div data-bbox="323 1193 703 1317" data-label="Diagram"> </div> <p>操作(1): ppt = ppb; ppb = ppa; ppa = ppt; 操作(2): pt = pb; pb = pa; pa = pt; 操作(3): t = b; b = a; a = t;</p>	<p>通过示例程序，结合图示，详细介绍和分析在使用二级指针变量过程中，三个不同级别（二级指针、一级指针、基本数据变量）的操作产生的影响。</p>
12	<div data-bbox="300 1494 347 1532" data-label="Image"></div> <h3>11.1.3 指向指针的指针</h3> <p>■ 二维数组的指针形式</p> <p>■ a[3][4]: 看成是由a[0]、a[1]、a[2]组成的一维数组，而a[0]、a[1]、a[2]各自又是一个一维数组。也即二维数组是数组元素为一维数组的一维数组。</p> <p>■ a: 第0行地址（行地址）</p> <p>■ a+i: 第i行的地址</p> <p>■ *(a+i) / a[i]: 第i行首元素的地址</p> <p>■ *(a+i)+j / a[i]+j: 第i行第j个元素的地址</p> <p>■ ***(a+i) / a[i][0]: 第i行首元素的值</p> <p>■ *((a+i)+j) / a[i][j]: 第i行第j个元素的值</p>	<p>介绍二维数组的指针内涵：</p> <p>二维数组名的指针意义</p> <p>二维数组中的二级指针</p> <p>二维数组中的一级指针</p> <p>二维数组元素的指针形式</p>

13	<div><h3>11.1.3 指向指针的指针</h3><ul style="list-style-type: none"><li>【例11-3】改写例11-1，用指向指针的指针实现。<pre>#include&lt;stdio.h&gt; #include&lt;string.h&gt; int main(void) { int i;   char *color[5] = {"red", "blue", "yellow", "green", "black"}; /   char **pc; /* 定义二级指针变量 */   char str[20];   pc = color; /* 二级指针赋值 */   printf("Input a color:");   scanf("%s", str);   for(i = 0; i &lt; 5; i++)     if(strcmp(str, *(pc+i)) == 0) /* 比较颜色是否相同 */       break;   if(i &lt; 5)     printf("position:%d\n", i+1);   else     printf("Not Found\n");   return 0; }</pre></li></ul></div>	使用指向指针的指针改写例 11-1，指出程序中的 pc 就是指向指针的指针变量，即二级指针变量，使用 pc 指向指针数组元素后，后续代码中也改为使用 pc 操作数据。																														
14	<div><h3>11.1.3 指向指针的指针</h3><ul style="list-style-type: none"><li>【例11-3】改写例11-1，用指向指针的指针实现。</li></ul><p>pc ⇔ color ⇔ &amp;color[0] *pc ⇔ color[0] *(pc+i) ⇔ color[i] **pc ⇔ (*(pc) ⇔ *color[0] : 'r'</p></div>	结合图示，通过分析各种表达式含义来介绍二级指针与指针数组的关系。																														
15	<div><h3>11.1.4 用指针数组处理多个字符串</h3><ul style="list-style-type: none"><li>1. 指针数组与二维数组<ul style="list-style-type: none"><li>□ 二维字符数组<pre>char ccolor[ ][7] = {"red", "blue", "yellow", "green", "black"};</pre></li><li>□ 指针数组<pre>char *pcolor[ ] = {"red", "blue", "yellow", "green", "black"};</pre></li></ul></li></ul><p>使用指针数组更节省内存空间</p><p>ccolor</p><table><tr><td>r</td><td>e</td><td>d</td><td>\0</td><td></td><td></td></tr><tr><td>b</td><td>l</td><td>u</td><td>e</td><td>\0</td><td></td></tr><tr><td>y</td><td>e</td><td>l</td><td>l</td><td>o</td><td>w</td></tr><tr><td>g</td><td>r</td><td>e</td><td>e</td><td>n</td><td>\0</td></tr><tr><td>b</td><td>l</td><td>a</td><td>c</td><td>k</td><td>\0</td></tr></table><p>pcolor</p></div>	r	e	d	\0			b	l	u	e	\0		y	e	l	l	o	w	g	r	e	e	n	\0	b	l	a	c	k	\0	处理多个字符串时通常有两种方式：二维字符数组和指针数组。通过分析其内存情况，可以发现，使用指针数组更节省内存空间。
r	e	d	\0																													
b	l	u	e	\0																												
y	e	l	l	o	w																											
g	r	e	e	n	\0																											
b	l	a	c	k	\0																											
16	<div><h3>11.1.4 用指针数组处理多个字符串</h3><ul style="list-style-type: none"><li>2. 用指针数组操作多个字符串<ul style="list-style-type: none"><li>【例11-4】将5个字符串从小到大排序后输出。</li></ul></li></ul><pre>void main( ) { int i;   int a[5] = {6, 5, 2, 8, 1};    void fsort(int a[ ], int n);   fsort(a, 5);   for(i = 0; i &lt; 5; i++)     printf("%d ", a[i]); }  #include &lt;string.h&gt; void main( ) { int i;   char *pcolor[ ] = {"red", "blue", "yellow", "green", "black"};   void fsort(char *color[ ], int n);   fsort(pcolor, 5);   for(i = 0; i &lt; 5; i++)     printf("%s ", pcolor[i]); }</pre></div>	在介绍程序时，采用对比分析的方法，将本示例程序与整型数组排序的程序做对比，从中可以看出普通数组与指针数组在操作上的差别。																														

17	<div data-bbox="300 203 775 271" data-label="Section-Header"> <h4>11.1.4 用指针数组处理多个字符串</h4> </div> <div data-bbox="300 293 815 562" data-label="Text"> <pre> void fsort(int a[], int n) {     int k, j;     int temp;     for(k = 1; k &lt; n; k++)         for(j = 0; j &lt; n-k; j++)             if(a[j] &gt; a[j+1]){                 temp = a[j];                 a[j] = a[j+1];                 a[j+1] = temp;             } }  void fsort(char *color[], int n) {     int k, j;     char *temp;     for(k = 1; k &lt; n; k++)         for(j = 0; j &lt; n-k; j++)             if(strcmp(color[j], color[j+1]) &gt; 0){                 temp = color[j];                 color[j] = color[j+1];                 color[j+1] = temp;             } } </pre> </div>	<p>运行演示该程序。</p> <p>特别指出，在排序过程中，字符串间的大小比较不能直接使用比较运算符，而是要使用函数 strcmp()。</p>
18	<div data-bbox="300 620 775 687" data-label="Section-Header"> <h4>11.1.4 用指针数组处理多个字符串</h4> </div> <div data-bbox="331 741 767 920" data-label="Diagram"> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <span>排序前</span> <span>排序后</span> </div> </div>	<p>用图示比较排序前后的差别。可以看出，排序后，字符串本身没有发生变化，而是指针数组元素的内容发生了变化，即指向关系发生了变化。</p>
19	<div data-bbox="300 1037 775 1104" data-label="Section-Header"> <h4>11.1.4 用指针数组处理多个字符串</h4> </div> <div data-bbox="312 1126 788 1216" data-label="Text"> <p>■ 例11-5 解密藏头诗。所谓藏头诗，就是将一首诗每一句的第一个字连起来，所组成的内容就是该诗的真正含义。编写程序，输出一首藏头诗的真实含义。</p> </div> <div data-bbox="363 1238 523 1350" data-label="Text"> <p>             一 叶轻舟向东流，              帆 梢轻握杨柳手，              风 纤碧波微微起舞，              顺 水任从雅客悠。         </p> </div>	<p>介绍什么是藏头诗。</p> <p>本例用指针数组组织一首藏头诗。</p>
20	<div data-bbox="300 1453 775 1520" data-label="Section-Header"> <h4>11.1.4 用指针数组处理多个字符串</h4> </div> <div data-bbox="331 1520 775 1834" data-label="Text"> <pre> #include &lt;stdio.h&gt; char *change(char s[][20]); int main(void) {     int i;     char *poem[4] = {"一叶轻舟向东流，", "帆梢轻握杨柳手，", "风纤碧波微微起舞，", "顺水任从雅客悠。"}; /* 指针数组初始化 */     char mean[10];     for(i = 0; i &lt; 4; i++){ /* 每行取第1个汉字存入mean */         mean[2 * i] = *(poem[i]);         mean[2 * i + 1] = *(poem[i] + 1);     }     mean[2 * i] = '\0';     printf("%s\n", mean); /* 输出结果 */     return 0; } </pre> </div>	<p>运行演示示例程序。</p> <p>一个汉字占 2 个字节。</p> <p>详细分析将每行第一个汉字取出生成字符串的代码。</p>

21	<div>  <h3>11.1.4 用指针数组处理多个字符串</h3> <h4>3. 动态输入多个字符串</h4> <p><b>例11-6</b> 输入一些有关颜色的单词，每行一个，以#作为输入结束标志，再按输入的反序输出这些单词。其中单词数小于20，每个单词不超过15个字母（用动态分配内存的方法处理多个字符串的输入）。</p> </div>	使用指针数组处理多个字符串时，字符串的输入通常采用动态分配内存的方法。
22	<div>  <pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; int main(void) {     int i, n = 0;     char *color[20], str[15];     scanf("%s", str);     while(str[0] != '#') {         color[n] = (char *)malloc(sizeof(char)*(strlen(str)+1)); /* 动态分配 */         strcpy(color[n], str); /* 将输入的字符串赋值给动态内存单元 */         n++;         scanf("%s", str);     }     for(i = n-1; i &gt;= 0; i--)         printf("%s ", color[i]);     return 0; }</pre> <div> <p>优点：能够根据实际输入数据的多少来申请和分配内存空间，从而提高了内存的使用率。</p> <p>运行结果： red blue yellow # yellow blue red</p> </div> </div>	<p>在输入多个字符串时，先将字符串暂存于字符数组 str 中，再根据字符串长度动态申请内存单元，其地址赋给指针数组相应元素，最后再将 str 中的字符串复制至此内存单元。</p> <p>结合本例说明采用动态分配内存方法的优点。</p>
23	<div>  <h3>*11.1.5 命令行参数</h3> <ul style="list-style-type: none"> <li>■ C语言源程序经编译和连接处理，生成可执行程序（例如test.exe）后，才能运行。</li> <li>■ 在DOS环境的命令窗口中，输入可执行文件名，就以命令方式运行该程序。</li> <li>■ 输入命令时，在可执行文件（命令）名的后面可以跟一些参数，这些参数被称为命令行参数</li> </ul> <p><b>test world</b></p> <p>(test是命令名，world是命令行参数)</p> </div>	介绍命令行参数的作用和使用环境。
24	<div>  <h3>*11.1.5 命令行参数</h3> <ul style="list-style-type: none"> <li>■ 命令行的一般形式为： <b>命令名 参数1 参数2 ... 参数n</b></li> </ul> <div> <p>命令名和各个参数之间用空格分隔，也可以没有参数</p> </div> <ul style="list-style-type: none"> <li>■ 使用命令行的程序不能在编译器中执行，需要将源程序经编译、链接为相应的命令文件（一般以.exe为后缀），然后回到命令行状态，再在该状态下直接输入命令文件名。</li> </ul> </div>	<p>命令行的格式要求。</p> <p>命令行必须在完成程序的编译、链接，生成可执行文件（.exe）后，在命令窗口中使用。</p>

25	<p><b>*11.1.5 命令行参数</b></p> <ul style="list-style-type: none"> <li>带参数的main()函数格式：  <pre>int main(int argc, char *argv[]) {     ..... }</pre> </li> <li>第1个参数argc接收命令行参数（包括命令名）的个数；第2个参数argv接收以字符串常量形式存放的命令行参数（包括命令名本身也作为一个参数）。</li> </ul>	带参数的主函数的参数格式，其中，要重点介绍参数 argc 和 argv 的数据类型和作用。
26	<p><b>*11.1.5 命令行参数</b></p> <ul style="list-style-type: none"> <li>【例11-7】编写C程序，令行参数在同一行上输入</li> </ul> <pre>#include &lt;stdio.h&gt; int main(int argc, char *argv) {     int k;     for(k = 1; k &lt; argc; k++)         printf("%s ", argv[k]);     printf("\n");     return 0; }</pre>  <pre>/* 从第1个命令行参数开始 */ /* 打印命令行参数 */</pre> <p><b>运行结果</b> 在命令行状态下输入： <b>echo How are you?</b> 输出： <b>How are you?</b></p>	<p>结合示例 11-7，介绍命令行参数的编程和使用方法，并演示完整的程序编程、运行过程。</p> <p>结合图示进一步分析 argv 与输入的各命令行参数的关系，即指针数组 argv 中的各元素指向相应的各参数字符串。</p>
27	<p><b>11.2 字符定位</b></p> <p>11.2.1 程序解析</p> <p>11.2.2 指针作为函数的返回值</p> <p><b>*11.2.3 指向函数的指针</b></p>	11.2.3 指向函数的指针（函数指针）为选学内容。
28	<p><b>11.2.1 程序解析</b></p> <ul style="list-style-type: none"> <li>例11-8 输入一个字符串和一个字符，如果该字符在字符串中，就从该字符首次出现的位置开始输出字符串中的字符。例如，输入字符r和字符串program后，输出rogram。</li> <li>要求定义函数match(s, ch)，在字符串s中查找字符ch，如果找到，返回第一次找到的该字符在字符串中的位置（地址）；否则，返回空指针NULL。</li> </ul>	<p>介绍程序要求。</p> <p>该示例程序的知识要点是指针作为函数的返回值。</p>



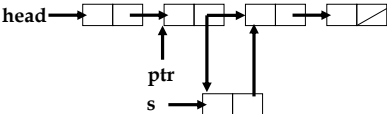
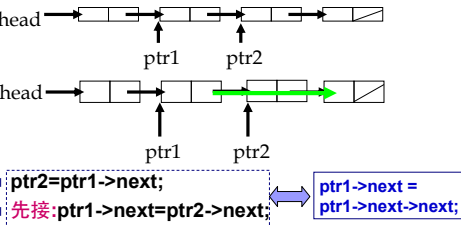
29	<div><div><h3>11.2.1 程序解析</h3><pre>#include &lt;stdio.h&gt; char *match(char *s, char ch) /* 函数 */ {     while(*s != '\0')     {         if(*s == ch) return(s); /* 若找到 */         else s++; /* 没有找到ch */     }     return(NULL); }  int main(void) {     char ch, str[80], *p = NULL;     printf("Please Input the string:\n");     scanf("%s", str);     getchar(); /* 跳过输入字符串和换行符 */     ch = getchar();     if((p = match(str, ch)) != NULL) /* 若找到 */         printf("%s\n", p);     else         printf("Not Found\n");     return 0; }</pre></div><div><p>运行结果1 Please Input the string: University 运行结果2 Please Input the string: School Not Found</p><p>•函数match()返回一个地址(指针)。 •在主函数中,用字符指针p接收match()返回的地址,从p指向的存储单元开始,连续输出其中的内容,直至'\0'为止。</p></div></div>	<p>在此程序中,函数 match()将查询到的字符位置(地址值)作为返回值。</p> <p>主函数中,字符指针变量 p 用于接收 match()的返回值,并从 p 指向的存储单元开始,输出字符串内容。</p>					
30	<div><div><h3>11.2.2 指针作为函数的返回值</h3><ul style="list-style-type: none"><li>■ 函数返回值的类型<ul style="list-style-type: none"><li>□ 整型</li><li>□ 浮点型</li><li>□ 字符型</li><li>□ 结构类型</li><li>□ 指针 (返回一个地址)</li></ul></li><li>■ 在C语言中,函数返回值也可以是指针,定义和调用这类函数的方法与其他函数是一样的。</li></ul></div></div>	<p>函数的返回值可以是 C 语言中的任何类型,甚至指针。</p>					
31	<div><div><h3>11.2.2 指针作为函数的返回值</h3><ul style="list-style-type: none"><li>■ 思考:在例11-8中,如果将str的定义及相应的数据输入都放在函数match()中,结果会如何?</li></ul><pre>char * match() {     char ch, str[80], *s = str; /* 定义局部变量 */     printf("Please Input the string:\n");     scanf("%s", str);     getchar ();     ch = getchar ();     while (*s != '\0')     {         if (*s == ch)         {             return s; /* 返回局部字符串地址 */         }         else         {             s++;         }     }     return (NULL); }</pre></div><div><p>不能返回在函数内部定义的局部数据对象的地址,这是因为所有的局部数据对象在函数返回时就会消亡,其值不再有效</p><p>返回指针的函数一般都返回全局数据对象或主调函数中数据对象的地址</p></div></div>	<p>通过程序演示说明:</p> <p>在函数中不能返回在内部定义的局部数据对象的地址,而一般需返回全局数据对象或主调函数中数据对象的地址。</p>					
32	<div><div><h3>*11.2.3 指向函数的指针</h3><ul style="list-style-type: none"><li>■ 每个函数都占用一段内存单元,它们有一个入口地址(起始地址)</li><li>■ 在C语言中,函数名代表函数的入口地址。</li><li>■ 我们可以定义一个指针变量,接收函数的入口地址,让它指向函数,这就是指向函数的指针,也称为函数指针。</li><li>■ 通过函数指针可以调用函数,它也可以作为函数的参数。</li></ul></div><div><div>入口地址</div><table><tr><td>指令1</td></tr><tr><td>指令2</td></tr><tr><td>指令3</td></tr><tr><td>...</td></tr><tr><td>指令n</td></tr></table></div></div>	指令1	指令2	指令3	...	指令n	<p>本小节为选学内容。</p> <p>对于每个函数来说,其指令起始地址就是函数入口地址,函数名即代表此入口地址。</p> <p>如果将函数入口地址存入某指针变量,则此指针变量就是指向函数的指针(函数指针)。</p>
指令1							
指令2							
指令3							
...							
指令n							

33	<p><b>*11.2.3 指向函数的指针</b></p> <p><b>1.函数指针的定义</b></p> <ul style="list-style-type: none"> <li>函数指针定义的一般格式为：  <b>类型名 (*变量名)( 参数类型表);</b>            类型名指定函数返回值的类型，变量名是指向函数的指针变量的名称。</li> <li>例如：  <b>int (*funptr)( int, int);</b>            定义一个函数指针<b>funptr</b>，它可以指向有两个整型参数且返回值类型为<b>int</b>的函数。</li> </ul>	介绍函数指针的定义格式。
34	<p><b>*11.2.3 指向函数的指针</b></p> <p><b>2.通过函数指针调用函数</b></p> <ul style="list-style-type: none"> <li>通过函数指针调用函数的一般格式为：  <b>(*函数指针名)(参数表)</b></li> <li>例如：  <b>int fun(int x, int y);</b>  <b>int (*funptr)(int, int);</b>  <b>funptr = fun;</b>  <b>(*funptr)(3 , 5);</b></li> </ul>	通过函数指针调用所指向的函数的方法。 要注意：定义函数指针时，其参数表要与指向的函数参数表相同。
35	<p><b>*11.2.3 指向函数的指针</b></p> <p><b>3.函数指针作为函数的参数</b></p> <ul style="list-style-type: none"> <li>C语言的函数调用中，函数名或已赋值的函数指针也能作为实参，此时，形参就是函数指针，它指向实参所代表函数的入口地址。</li> <li>例如：  <b>f(int (*funptr)(int, int))</b>  <b>{...}</b>  <b>void main()</b>  <b>{ ...</b>  <b>int (*funptr)(int, int);</b>  <b>funptr = fun;</b>  <b>f( funptr );</b>  <b>...}</b></li> </ul>	在前面已经学习过指针作为函数参数的情况，函数指针是一种特殊的指针，当然也可以作为函数参数。 当函数指针作为函数参数时，可以实现通过函数指针参数调用不同函数功能的效果。
36	<p><b>*11.2.3 指向函数的指针</b></p> <p>例11-9 编写一个函数<b>calc(f, a, b)</b>，用梯形公式求函数<b>f(x)</b>在<b>[a, b]</b>上的数值积分。</p> $\int_a^b f(x)dx \approx (b-a)/2 * (f(a)+f(b))$ <p>然后调用该函数计算下列数值积分。（函数指针作为函数参数示例）</p> $\int_0^1 x^2 dx \quad \int_1^2 \sin x/x dx$ <p><b>分析：</b><b>calc()</b>是一个通用函数，用梯形公式求解数值积分。它和被积函数<b>f(x)</b>以及积分区间<b>[a, b]</b>有关，相应的形参包括函数指针、积分区间上下限参数。在函数调用时，把被积函数的名称（或函数指针）和积分区间的上下限作为实参。</p>	例 11-9 要求通过一个函数 <b>calc()</b> 实现对不同积分公式的调用。 其方法就是在函数 <b>calc()</b> 中设置一个函数指针参数，可以指向不同的积分公式函数。

37	<div data-bbox="304 203 344 237" data-label="Image"></div> <h3>*11.2.3 指向函数的指针</h3> <pre data-bbox="312 271 788 584">/* 计算数值积分 (函数指针作为函数参数示例) */ int main(void) {     double result;     double (*fnp)(double);     result = calc(f1, 0.0, 1.0); /* 函数名f1作为函数calc的实参 */     printf("1: resule=%.4fn", result);     fnp = f2; /* 对函数指针fnp赋值 */     result = calc(fnp, 1.0, 2.0); /* 函数指针fnp作为函数calc的实参 */     printf("2: resule=%.4fn", result);     return 0; } /* 函数指针fnp作为函数的形参 */ double calc(double (*fnp)(double), double a, double b) {     double z;     z = (b-a)/2 * ((*fnp)(a) + (*fnp)(b)); /* 调用fnp指向的函数 */     return(z); }</pre> <div data-bbox="624 427 788 501" data-label="Text"> <p>运行结果</p> <p>1: resule=0.5000</p> <p>2: resule=0.6481</p> </div>	<p>具体介绍和分析 main()、calc()、f1()、f2() 各函数之间的关系，以及函数指针变量 fnp 如何发挥作用。</p>
38	<div data-bbox="304 620 344 654" data-label="Image"></div> <h2>11.3 用链表构建学生信息库</h2> <h3>11.3.1 程序解析</h3> <h3>11.3.2 链表的概念</h3> <h3>11.3.3 单向链表的常用操作</h3>	
39	<div data-bbox="304 1037 344 1070" data-label="Image"></div> <h3>11.3.1 程序解析</h3> <p>■ 例11-10 建立一个学生成绩信息（包括学号、姓名、成绩）的单向链表，学生记录按学号由小到大顺序排列，要求实现对成绩信息的插入、修改、删除和遍历操作。</p> <div data-bbox="328 1245 788 1379" data-label="Diagram"> <pre> graph TD     main --&gt; Create_Stu_Doc     main --&gt; InsertDoc     main --&gt; DeleteDoc     main --&gt; Print_Stu_Doc     Create_Stu_Doc --&gt; InsertDoc     </pre> </div>	<p>简要分析例 11-10 的程序功能和结构。</p>
40	<div data-bbox="304 1453 344 1487" data-label="Image"></div> <h3>11.3.1 程序解析</h3> <pre data-bbox="320 1525 775 1827">struct stud_node{ /*链表结点类型*/     int num;     char name[20];     int score;     struct stud_node *next; }; struct stud_node * Create_Stu_Doc(); /* 新建链表 */  struct stud_node * InsertDoc(struct stud_node * head,     struct stud_node *stud); /* 插入 */  struct stud_node * DeleteDoc(struct stud_node * head,     int num); /* 删除 */  void Print_Stu_Doc(struct stud_node * head); /* 遍历 */</pre>	<p>介绍结点结构、各功能函数的功能和参数。</p>

41	<div data-bbox="300 203 820 241" data-label="Section-Header"> <h3>11.3.2 链表的概念</h3> </div> <div data-bbox="323 288 798 360" data-label="List-Group"> <ul style="list-style-type: none"> <li>■ 链表是一种常见而重要的<b>动态存储</b>分布的数据结构。它由若干个同一结构类型的“<b>结点</b>”依次串接而成。链表分<b>单向链表</b>和双向链表。</li> </ul> </div> <div data-bbox="323 412 785 517" data-label="Diagram"> </div>	<p>介绍链表的基本概念，包括结点内容的构成，结点是动态分配的，什么是头结点、尾结点、头指针。</p>
42	<div data-bbox="300 620 820 658" data-label="Section-Header"> <h3>11.3.2 链表的概念</h3> </div> <div data-bbox="323 705 758 752" data-label="List-Group"> <ul style="list-style-type: none"> <li>■ 通常使用结构来定义单向链表结点的数据类型：</li> </ul> </div> <div data-bbox="343 752 604 898" data-label="Text"> <pre>struct stud_node{     int  num;     char name[20];     int  score;     struct stud_node *next; };</pre> </div> <div data-bbox="604 792 751 824" data-label="Text"> <p>结构的递归定义</p> </div>	<p>链表的结点通常使用结构类型，其中必须包含一个指针成员，其指向本结构类型，这样就形成结构的递归定义。</p>
43	<div data-bbox="300 1037 820 1075" data-label="Section-Header"> <h3>11.3.2 链表的概念</h3> </div> <div data-bbox="323 1124 461 1153" data-label="Section-Header"> <h4>■ 与数组比较</h4> </div> <div data-bbox="343 1155 772 1357" data-label="List-Group"> <ul style="list-style-type: none"> <li>□ 在用数组存放数据时，一般需要事先定义好固定长度的数组，在数组元素个数不确定时，可能会发生浪费内存空间的情况。</li> <li>□ 链表是动态存储分布的数据结构。根据需要动态地开辟内存空间，可以比较自由方便地插入新元素（结点），故使用链表可以<b>节省内存，操作效率高</b>。</li> </ul> </div>	<p>链表与数组相比较，最大的特点是每个数据单元都是动态分配的，节省内存空间，且操作灵活。</p>
44	<div data-bbox="300 1453 820 1491" data-label="Section-Header"> <h3>11.3.2 链表的概念</h3> </div> <div data-bbox="323 1538 529 1565" data-label="Section-Header"> <h4>■ 动态分配相关函数</h4> </div> <div data-bbox="343 1563 791 1825" data-label="List-Group"> <ul style="list-style-type: none"> <li>□ <b>void *malloc(unsigned size)</b> 功能：在内存的动态存储区中分配一块长度为size的连续空间。 返回值：指针，存放被分配内存的起始地址。若未申请到空间，则返回 <b>NULL(0)</b>。 例如：(int *) malloc(sizeof(int)) (struct student *) malloc(sizeof(struct student))</li> <li>□ <b>void free(void *ptr)</b> 功能：释放由malloc()申请的动态内存空间，ptr存放该空间的首地址。 返回值：无。 例如：free(p);</li> </ul> </div>	<p>介绍相关动态分配函数 malloc()和 free()。说明 malloc()的参数含义，并特别要注意，在使用 malloc()时，其返回值（指针）要根据实际情况进行适当的强制类型转换。</p>

45	<div><div><div></div><div></div></div><div><h3>11.3.3 单向链表的常用操作</h3><ul style="list-style-type: none"><li>1. 链表的建立</li><li>2. 链表的遍历</li><li>3. 插入结点</li><li>4. 删除结点</li></ul></div></div>	单向链表的基本操作
46	<div><div><div></div><div></div></div><div><h3>11.3.3 单向链表的常用操作</h3><ul style="list-style-type: none"><li>1. 链表的建立</li></ul></div></div> <div><pre>graph TD     Start([head=tail=NULL]) --&gt; Input[输入: num, name, score]     Input --&gt; Num0{num=0}     Num0 -- 真 --&gt; Exit([退出])     Num0 -- 假 --&gt; Alloc[p = (struct stud_node *) malloc(size)]     Alloc --&gt; SetData[输入: 学号, 姓名, 成绩]     SetData --&gt; NextNull[p-&gt;next=NULL]     NextNull --&gt; HeadNull{head=NULL}     HeadNull -- 真 --&gt; HeadP[head=p]     HeadNull -- 假 --&gt; TailNext[tail-&gt;next=p]     TailNext --&gt; TailP[tail=p]     TailP --&gt; Input</pre></div>	详细介绍链表建立的算法流程，在建立过程中采用结点的尾部插入法。 例 11-10 中是有序表插入，与此不同，可以让学生比较一下。
47	<div><div><div></div><div></div></div><div><h3>11.3.3 单向链表的常用操作</h3><ul style="list-style-type: none"><li>1. 链表的建立</li></ul></div></div> <div><pre>head = tail = NULL; scanf("%d%s%d", &amp;num, name, &amp;score); while(num != 0){     p = (struct stud_node *) malloc(size);     p-&gt;num = num;     strcpy(p-&gt;name, name);     p-&gt;score = score;     p-&gt;next = NULL;     if(head == NULL)         head = p;     else         tail-&gt;next = p;     tail = p;     scanf("%d%s%d", &amp;num, name, &amp;score); }</pre></div> <div><div>尾部插入</div><div>头部插入法: p-&gt;next=head; head=p;</div></div>	根据算法思路，详细介绍实现语句。 在将结点插入链表时，有尾部插入和头部插入两种方法。
48	<div><div><div></div><div></div></div><div><h3>11.3.3 单向链表的常用操作</h3><ul style="list-style-type: none"><li>2. 链表的遍历</li></ul></div></div> <div><pre>for(ptr = head; ptr!=NULL; ptr = ptr-&gt;next)     printf("%d\t%s\t%d\n", ptr-&gt;num, ptr-&gt;name, ptr-&gt;score);</pre></div> <div><p>(a) 执行 ptr = ptr-&gt;next 前</p><p>(b) 执行 ptr = ptr-&gt;next 后</p></div>	结合图示，介绍链表的遍历方法。

49	<div data-bbox="300 203 820 591">  <p><b>11.3.3 单向链表的常用操作</b></p> <p>■ 3. 插入结点</p> <p>■ 先连: <code>s-&gt;next = ptr-&gt;next;</code>          ■ 后断: <code>ptr-&gt;next = s;</code></p> </div>	<p>结合图示说明：要在链表的结点 <code>ptr</code> 后面插入结点 <code>s</code>，必须“先连后断”。</p>
50	<div data-bbox="300 620 820 1008">  <p><b>11.3.3 单向链表的常用操作</b></p> <p>■ 4. 删除结点</p> <p>■ <code>ptr2=ptr1-&gt;next;</code>          ■ 先接: <code>ptr1-&gt;next=ptr2-&gt;next;</code>          ■ 后删: <code>free(ptr2);</code></p> </div>	<p>结合图示说明：要删除链表结点 <code>ptr1</code> 后面的结点，必须“先接后删”。</p>
51	<div data-bbox="300 1037 820 1424"> <p><b>本章总结</b></p> <ul style="list-style-type: none"> <li>■ 指针数组             <ul style="list-style-type: none"> <li>□ 指针数组概念与应用</li> <li>□ 指向指针的指针（二级指针）</li> <li>□ 命令行参数</li> </ul> </li> <li>■ 指针与函数             <ul style="list-style-type: none"> <li>□ 指针作为函数的返回值</li> <li>□ 函数指针</li> </ul> </li> <li>■ 单向链表             <ul style="list-style-type: none"> <li>□ 链表的概念</li> <li>□ 结点的结构定义与动态分配</li> <li>□ 链表的基本操作（建立、遍历、插入、删除）</li> </ul> </li> </ul> <div data-bbox="587 1070 815 1227"> <p>•能够熟练掌握指针数组的操作与应用</p> <p>•能够熟练处理与操作函数与指针的各种关系</p> <p>•能够掌握单向链表的基本操作</p> </div> </div>	<p>回顾和总结本章的教学要点，对学生提出能力要求。</p>

## 11.3 练习与习题参考答案

### 11.3.1 练习参考答案

11-1 如何理解指针数组，它与指针、数组有何关系？为何可以用二级指针对指针数组进行操作？

解答：

首先，指针数组是数组，其次，指针数组中各元素的数据类型都是指针，即每个数组元素都是指针。

一维普通数组可以用一级指针进行操作，而一维指针数组中每个数组元素都是一个指针，因此，相当于二级指针。

11-2 用指针数组处理多个字符串有何优势？可以直接输入多个字符串给未初始化的指针数组吗？为什么？

解答：

用指针数组处理多个字符串时内存使用效率更高。

最好不要直接输入多个字符串给未初始化的指针数组。因为如果指针数组未初始化，各数组元素将指向不确定的内存单元，此时将字符串写入这些单元，可能会引起系统错误。

11-3 参考例 11-3，使用二级指针操作改写例 11-4 中的程序 A。

解答：

```
#include <stdio.h>
#include <string.h>
int main(void )
{   int i;
    char *pcolor[ ] = {"red", "blue", "yellow", "green", "black"};
    void fsort(char **color[ ], int n);
    fsort(pcolor, 5);    /* 调用函数 */
    for(i = 0; i < 5; i++)
        printf("%s ", pcolor[i]);
    return 0;
}
void fsort(char **pc, int n)
{   int k, j;
    char *temp;
    for(k = 1; k < n; k++)
        for(j = 0; j < n-k; j++)
            if(strcmp(*(pc+j), *(pc+j+1)) > 0){
                temp = *(pc+j);
                *(pc+j) = *(pc+j+1);
                *(pc+j+1) = temp;
            }
}
```

11-4 改写例 11-8 中的函数 match()，要求返回字符串 s 中最后一个字符 ch 的位置(地址)。

解答：

```
char *match(char *s, char ch)
{   char *p=NULL;
    for( ; *s != '\0' ; s++)
        if(*s == ch)
            p=s;
    return(p);
}
```

11-5 前面章节中介绍的指针变量都可以进行算术运算，请思考：指向函数的指针变量可以进行算术运算吗？

解答:

指向函数的指针变量不能进行算术运算。指针变量如果要加减, 必须知道该指针指向类型的内存占用大小, 由于函数指针无法确定该大小, 所以不能进行加减算术运算。

11-6 运行例 11-10, 试执行程序中各函数的功能, 观察结果。

解答: (略)

11-7 改写例 11-10 中的函数 DeleteDoc(), 要求删除链表中成绩小于 60 分的学生结点。

解答:

```
struct stud_node * DeleteDoc(struct stud_node * head)
{
    struct stud_node *ptr1, *ptr2;
    /* 如果要被删除结点为表头结点 */
    while(head!=NULL && head->num <60){
        ptr2 = head;
        head = head->next;
        free(ptr2);
    }
    if(head == NULL) /*链表空 */
        return NULL;
    /* 如果要被删除结点为非表头结点 */
    ptr1 = head;
    ptr2 = head->next; /*从表头的下一个结点搜索所有符合删除要求的结点 */
    while(ptr2!=NULL){
        if(ptr2->num <60){ /* ptr2 所指结点符合删除要求 */
            ptr1->next = ptr2->next;
            free(ptr2);
        }
        else
            ptr1 = ptr2; /* ptr1 后移一个结点 */
        ptr2 = ptr1->next; /* ptr2 指向 ptr1 的后一个结点 */
    }
    return head;
}
```

11-8 在例 11-10 的基础上, 再编写一个函数 UpdateDoc(), 实现对链表中某结点信息(成绩)的修改。函数原型为: void UpdateDoc(struct stud\_node \* head, int num, int score), 其中, num 为需要修改信息的学生学号, score 为需要修改的成绩值。

解答:

```
void UpdateDoc(struct stud_node * head, int num, int score)
{
    struct stud_node *p;
    for(p=head; p!=NULL; p=p->next){
        if(p->num==num)
            p->score = score;
    }
}
```



```

    }
}

```

### 11.3.2 习题参考答案

#### 一、选择题

1	2	3	4	5
B	D	D	C	D

#### 二、填空题

1. \*\*q                  language + k
2. GetMax(score, 10)          score + pos
3.        1 0 0 1  
          0 1 1 0  
          0 1 1 0  
          1 0 0 1
4. p = p->next                  p!=NULL
5. p = head, x = 0              p->score < 60

#### 三、程序设计题

1. 输出月份英文名：输入月份，输出对应的英文名称。要求用指针数组表示 12 个月的英文名称。例如，输入 5，输出 May。试编写相应程序。

解答：

```

#include<stdio.h>
int main(void)
{   char *months[]={"January", "February", "March", "April", "May", "June", "July",
    "August", "September", "October", "November", "December"};
    int n;
    scanf("%d", &n);
    if(n<1 || n>12) printf("wrong input!\n");
    else
        printf("%s\n", months[n-1]);
    return 0;
}

```

2. 查找星期：定义一个指针数组，将下表的星期信息组织起来，输入一个字符串，在表中查找，若存在，输出该字符串在表中的序号，否则输出-1。试编写相应程序。

0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

解答:

```
#include<stdio.h>
#include<string.h>
int main(void)
{   char *weeks[]={ "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
    "Saturday"}, str[80];
    int i, index;
    scanf("%s", str);
    index=-1;
    for(i=0; i<7; i++)
        if(strcmp(weeks[i], str)==0)
            index=i;
    printf("%d \n", index);
    return 0;
}
```

3. 计算最长的字符串长度: 输入 n(n<10)个字符串, 输出其中最长字符串的有效长度。要求自定义函数 int max\_len(char \*s[ ], int n), 用于计算有 n 个元素的指针数组 s 中最长的字符串的长度。试编写相应程序。

解答:

```
#include <stdio.h>
#include <string.h>
int max_len(char *s[],int n);
int main(void)
{
    int i,n;
    char s[10][80],*p[10];

    scanf("%d",&n);
    for(i=0;i<n;i++){
        scanf("%s",s[i]);
        p[i]=s[i];
    }
    printf("%d\n",max_len(p,n));
    return 0;
}
int max_len(char *s[],int n)
{
    int i,res=strlen(s[0]);
    for(i=1;i<n;i++)
        if(strlen(s[i])>res) res=strlen(s[i]);
    return res;
}
```

4. 字符串的连接：输入两个字符串，输出连接后的字符串。要求自定义函数 `char *str_cat(char *s, char *t)`，将字符串 `t` 复制到字符串 `s` 的末端，并且返回字符串 `s` 的首地址。试编写相应程序。

解答：

用字符数组实现：

```
char *str_cat(char s[], char t[])
{ int i, j;
  i=0;
  while(s[i]!='\0') i++;
  j=0;
  while(t[j]!='\0')
  { s[i]=t[j];
    i++; j++;
  }
  s[i]='\0';
  return s;
}
```

用字符指针实现：

```
char *str_cat(char *s, char *t)
{ char *p;
  p=s;
  while(*s!='\0') s++;
  while(*t!='\0')
  { *s=*t;
    s++; t++;
  }
  *s='\0';
  return p;
}
```

5. 指定位置输出字符串：输入一个字符串后再输入两个字符，输出此字符串中从与第 1 个字符匹配的位置开始到与第 2 个字符匹配的位置结束的所有字符。例如，输入字符串“program”与 2 个字符‘r’和‘g’后，输出“rog”。要求自定义函数 `char *match(char *s, char ch1, char ch2)` 返回结果字符串的首地址。试编写相应程序。

解答：

```
#include <stdio.h>
char newstr[80];
int main()
{
  char str[80],ch1,ch2;
  char *match(char *s, char ch1, char ch2);
  scanf("%s",str);
  getchar();
  ch1=getchar();
```

```

        getchar();
        ch2=getchar();
        puts(match(str,ch1,ch2));
        return 0;
    }
    char *match(char *s, char ch1, char ch2)
    {
        char *p=s;
        int i=0;
        while(*s!=ch1&&*s)
            s++;
        p=s;
        while(*s!=ch2&&*s){
            s++;
        }
        s[1]='\0';
        return p;
    }

```

6. 查找子串：输入两个字符串 s 和 t，在字符串 s 中查找子串 t，输出起始位置，若不存在，则输出-1。要求自定义函数 char \*search(char \*s, char \*t)返回子串 t 的首地址，若未找到，则返回 NULL。试编写相应程序。

解答：

```

char *search(char *s, char *t)
{
    int i,j, len;
    for(i=0; i<=strlen(s) - strlen(t); i++){
        for(j=0; j<strlen(t); j++)
            if( *(s+i+j) != *(t+j))    break;
        if(j == strlen(t))
            return s+i;
    }
    return NULL;
}

```

7. 奇数值结点链表：输入若干个正整数（输入-1 为结束标志）建立一个单向链表，头指针为 L，将链表 L 中奇数值的结点重新组成一个新的链表 NEW，并输出新建链表的信息。试编写相应程序。

解答：

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct stud_node{
    int num;
    struct stud_node *next;
}

```

```

};
void Print_Stu_Doc(struct stud_node *head);
int main(void)
{
    struct stud_node *L,*tail1,*tail2,*p1,*p2,*NEW;
    int num;
    int size=sizeof(struct stud_node);
    L=tail1=NULL;
    scanf("%d",&num);
    while(num!=-1){
        p1=(struct stud_node *)malloc(size);
        p1->num=num;
        p1->next=NULL;
        if(L==NULL)
            L=p1;
        else
            tail1->next=p1;
        tail1=p1;
        scanf("%d",&num);
    }
    NEW=tail2=NULL;
    p1=L;
    while(p1){
        if(p1->num%2==0&& p1!=NULL)
        {
            if(p1->next!=NULL)
            {
                p1=p1->next;
                continue;
            }
            else
                break;
        }
        if(p1==NULL) break;
        p2=(struct stud_node *)malloc(size);
        p2->num=p1->num;
        p2->next=NULL;
        if(NEW==NULL)
            NEW=p2;
        else
            tail2->next=p2;
        tail2=p2;
        p1=p1->next;
    }
}

```

```

        tail2->next=NULL;
        Print_Stu_Doc(NEW);
        return 0;
    }
void Print_Stu_Doc(struct stud_node *head)
{
    struct stud_node *ptr;
    if(head==NULL){
        printf("No Records\n");
        return;
    }
    for(ptr=head;ptr!=NULL;ptr=ptr->next)
        printf("%d ",ptr->num);
    printf("\n");
}

```

8. 删除结点：输入若干个正整数（输入-1 为结束标志）建立一个单向链表，再输入一个整数 m，删除链表中值为 m 的所有结点。试编写相应程序。

解答：

```

#include <stdio.h>
#include <stdlib.h>
struct node {
    struct node *next;
    int x;
} *head;
void InsertDoc(int x) {
    struct node *doc = (struct node*) malloc(sizeof(struct node));
    doc->next = NULL;
    doc->x = x;
    if (head == NULL) {
        head = doc;
    } else {
        struct node *p = head;
        while (p->next != NULL) {
            p = p->next;
        }
        p->next = doc;
    }
}
void DeleteDoc(int x) {
    struct node *prev = NULL, *cur = head, *t;
    while (cur != NULL) {
        if (cur->x == x) {
            if (prev == NULL) {

```

```

        head = cur->next;
    } else {
        prev->next = cur->next;
    }
    t = cur->next;
    free(cur);
    cur = t;
} else {
    prev = cur;
    cur = cur->next;
}
}
}
void PrintDoc() {
    struct node *p = head;
    while (p != NULL) {
        printf("%d\n", p->x);
        p = p->next;
    }
}
int main() {
    int x, m;
    head = NULL;
    while(1) {
        scanf("%d", &x);
        if (x == -1) break;
        InsertDoc(x);
    }
    scanf("%d", &m);
    DeleteDoc(m);
    PrintDoc();
    return 0;
}

```

## 11.4 实验指导教材参考答案

### 实验 11.1 指针数组、指针与函数

#### 一、调试示例

英文单词排序：输入若干有关颜色的英文单词（单词数小于 20，每个单词不超过 10 个字母），每行一个，以#作为输入结束标志，对这些单词按长度从小到大排序后输出。（源程序 error11\_1.cpp）

错误行号: 7      正确语句: char \*color[20], str[10], \*temp;  
错误行号: 20      正确语句: if(strlen(color[j]) > strlen(color[j + 1])){

## 二、基础编程题

(1) 输出月份英文名: 输入一个月份, 输出对应的英文名称, 要求用指针数组表示 12 个月的英文名称。试编写相应程序。

解答: 参见习题程序设计第 1 题

(2) 查找星期: 定义一个指针数组将下表的星期信息组织起来, 输入一个字符串, 在表中查找, 若存在, 输出该字符串在表中的序号, 否则输出-1。试编写相应程序。

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

解答: 参见习题程序设计第 2 题

(3) 计算最长的字符串长度: 输入  $n(n < 10)$  个字符串, 输出其中最长字符串的有效长度。要求自定义函数 `int max_len(char *s[], int n)`, 用于计算有  $n$  个元素的指针数组  $s$  中最长的字符串的长度。试编写相应程序。

解答: 参见习题程序设计第 3 题

(4) 字符串的连接: 输入两个字符串, 输出连接后的字符串。要求自定义函数 `char *strcat(char *s, char *t)`, 将字符串  $t$  复制到字符串  $s$  的末端, 并且返回字符串  $s$  的首地址。试编写相应程序。

解答: 参见习题程序设计第 4 题

(5) 指定位置输出字符串: 输入一个字符串后再输入两个字符, 输出此字符串中从与第 1 个字符匹配的位置开始到与第 2 个字符匹配的位置结束的所有字符。要求自定义函数 `char *match(char *s, char ch1, char ch2)` 返回结果字符串的首地址。试编写相应程序。

解答: 参见习题程序设计第 5 题

## 三、改错题

藏头词: 输入一组英文单词 (不超过 8 个), 要求按输入顺序取出每个单词的第一个字母并连接在一起形成一个字符串并输出。(源程序 `error11_2.cpp`)

错误行号: 14      正确语句: printf("%s\n", change(p, n));  
错误行号: 24      正确语句: t[i]='\0';

## 四、拓展编程题

(1) 查找子串: 输入两个字符串  $s$  和  $t$ , 在字符串  $s$  中查找子串  $t$ , 输出起始位置, 若不存在, 则输出-1。要求自定义函数 `char *search(char *s, char *t)` 返回子串  $t$  的首地址, 若未找到, 则返回 NULL。试编写相应程序。



解答：参见习题程序设计第 6 题

(2) 藏尾词：输入一组英文单词（不超过 8 个），按输入顺序将每个单词的最后一个字母连起来形成一个新单词。用返回字符指针的函数实现。试编写相应程序。

解答：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char * change(char *s[ ], int n);
int main(void)
{
    int i,n;
    char poem[8][20], *p[8];
    scanf("%d",&n);
    for(i = 0; i < n; i++){
        scanf("%s",poem[i]);
        p[i] = poem[i];
    }
    printf("%s\n", change(p, n));
    return 0;
}
char * change(char *s[ ], int n)
{
    int i;
    char *t = (char *)malloc(9 * sizeof(char));
    for(i = 0; i < n; i++)
        t[ i ] = s[i][strlen(s[i])-1];
    t[i]='\0';
    return t;
}
```

## 实验 11.2 单向链表

### 一、调试示例

建立学生信息链表：输入若干个学生的信息（学号、姓名、成绩），当输入学号为 0 时结束，用单向链表组织这些学生信息后，再按顺序输出。（源程序 error11\_3.cpp）

错误行号： 21      正确语句： p = (struct stud\_node\*)malloc(size);  
错误行号： 27      正确语句： if(head==NULL) head=p; else tail->next = p;  
错误行号： 32      正确语句： for(p = head; p!= NULL; p = p->next)

### 二、基础编程题

(1) 单向链表建立：输入若干个学生信息（包括学号、姓名和成绩），输入学号为 0 时输入结束，建立一个单向链表，再输入一个成绩值，将成绩大于等于该值的学生信息输出。试编写相应程序。

解答:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct stud_node{
    int num;
    char name[20];
    int score;
    struct stud_node *next;
};
struct stud_node *Create_Stu_Doc();
struct stud_node *DeleteDoc(struct stud_node *head,int min_score);
void Print_Stu_Doc(struct stud_node *head);
void main()
{
    struct stud_node *head;
    int min_score;
    head=Create_Stu_Doc();
    scanf("%d",&min_score);
    head=DeleteDoc(head,min_score);
    Print_Stu_Doc(head);
}
struct stud_node *Create_Stu_Doc()
{
    struct stud_node *head,*tail,*p;
    int num,score;
    char name[20];
    int size=sizeof(struct stud_node);
    head=tail=NULL;
    scanf("%d",&num);
    if(num==0)
        return head;
    scanf("%s%d",name,&score);
    while(num!=0){
        p=(struct stud_node *)malloc(size);
        p->num=num;
        strcpy(p->name,name);
        p->score=score;
        p->next=NULL;
        if(head==NULL)
            head=p;
        else
            tail->next=p;
        tail=p;
        scanf("%d",&num);
    }
```

```

        scanf("%d",&num);
        if(num==0)
            break;
        scanf("%s%d",name,&score);
    }
    return head;
}
void Print_Stu_Doc(struct stud_node *head)
{
    struct stud_node *ptr;
    if(head==NULL){
        printf("No Records\n");
        return;
    }
    for(ptr=head;ptr;ptr=ptr->next)
        printf("%d %s %d\n",ptr->num,ptr->name,ptr->score);
}
struct stud_node *DeleteDoc(struct stud_node *head,int min_score)
{
    struct stud_node *ptr1,*ptr2;
    while(head&&head->score<min_score){
        ptr2=head;
        head=head->next;
        free(ptr2);
    }
    if(head==NULL)
        return NULL;
    ptr1=head;
    ptr2=head->next;
    while(ptr2)
    {
        if(ptr2->score<min_score){
            ptr1->next=ptr2->next;
            free(ptr2);
        }
        else
            ptr1=ptr2;
        ptr2=ptr1->next;
    }
    return head;
}

```

(2) 逆序数据建立链表：输入若干个正整数（输入-1 为结束标志），要求按输入数据的逆序建立一个链表，并输出。试编写相应程序。

解答:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct stud_node{
    int num;
    struct stud_node *next;
};
struct stud_node *Create_Stu_Doc();
void Print_Stu_Doc(struct stud_node *head);
void main()
{
    struct stud_node *head;
    head=Create_Stu_Doc();
    Print_Stu_Doc(head);
}
struct stud_node *Create_Stu_Doc()
{
    struct stud_node *head,*p;
    int num;
    int size=sizeof(struct stud_node);
    head=NULL;
    scanf("%d",&num);
    while(num!=-1){
        p=(struct stud_node *)malloc(size);
        p->num=num;
        p->next=head;
        head=p;
        scanf("%d",&num);
    }
    return head;
}
void Print_Stu_Doc(struct stud_node *head)
{
    struct stud_node *ptr;
    if(head==NULL){
        printf("No Records\n");
        return;
    }
    for(ptr=head;ptr;ptr=ptr->next)
        printf("%d ",ptr->num);
    printf("\n");
}
```

(3) 删除单向链表偶数节点：输入若干个正整数（输入-1 为结束标志），并建立一个单向链表，将其中的偶数值结点删除后输出。试编写相应程序。

解答：

```
#include<stdio.h>
#include<stdlib.h>
struct stud_node{
    int num;
    struct stud_node *next;
};
struct stud_node *Create_Stu_Doc();
struct stud_node *DeleteDoc(struct stud_node *head);
void Print_Stu_Doc(struct stud_node *head);
void main()
{
    struct stud_node *head;

    head=Create_Stu_Doc();
    head=DeleteDoc(head);
    Print_Stu_Doc(head);
}
struct stud_node *Create_Stu_Doc()
{
    struct stud_node *head,*tail,*p;
    int num;
    int size=sizeof(struct stud_node);

    head=tail=NULL;
    scanf("%d",&num);
    while(num!=-1){
        p=(struct stud_node *)malloc(size);
        p->num=num;
        p->next=NULL;
        if(head==NULL)
            head=p;
        else
            tail->next=p;
        tail=p;
        scanf("%d",&num);
    }
    return head;
}
void Print_Stu_Doc(struct stud_node *head)
{
    struct stud_node *ptr;
```

```

        if(head==NULL){
            printf("No Records\n");
            return;
        }
        for(ptr=head;ptr;ptr=ptr->next)
            printf("%d ",ptr->num);
        printf("\n");
    }
}

struct stud_node *DeleteDoc(struct stud_node *head)
{
    struct stud_node *ptr1,*ptr2;
    while(head&&head->num%2==0){
        ptr2=head;
        head=head->next;
        free(ptr2);
    }
    if(head==NULL)
        return NULL;
    ptr1=head;
    ptr2=head->next;
    while(ptr2)
    {
        if(ptr2->num%2==0){
            ptr1->next=ptr2->next;
            free(ptr2);
        }
        else
            ptr1=ptr2;
        ptr2=ptr1->next;
    }
    return head;
}

```

(4) 链表拼接：输入若干个正整数（输入-1 为结束标志）建立两个已按升序排序的单向链表，头指针分别为 list1、list2，把两个链表拼成一个链表，并输出新链表信息。要求自定义函数，实现将两个链表拼成一个链表，并返回拼组后的新链表。试编写相应程序。

解答：

```

#include<stdio.h>
#include<stdlib.h>
struct stud_node{
    int num;
    struct stud_node *next;
};

struct stud_node *Create_Stu_Doc();

```

```

void Print_Stu_Doc(struct stud_node *head);
struct stud_node *InsertDoc(struct stud_node *list1,struct stud_node *list2);
void main()
{
    struct stud_node  *list1,*list2;

    list1=Create_Stu_Doc();
    list2=Create_Stu_Doc();
    list1=InsertDoc(list1,list2);
    Print_Stu_Doc(list1);
}
struct stud_node *Create_Stu_Doc()
{
    struct stud_node *head,*tail,*p;
    int num;
    int size=sizeof(struct stud_node);

    head=tail=NULL;
    scanf("%d",&num);
    while(num!=-1){
        p=(struct stud_node *)malloc(size);
        p->num=num;
        p->next=NULL;
        if(head==NULL)
            head=p;
        else
            tail->next=p;
        tail=p;
        scanf("%d",&num);
    }
    return head;
}
void Print_Stu_Doc(struct stud_node *head)
{
    struct stud_node *ptr;

    if(head==NULL){
        printf("No Records\n");
        return;
    }
    for(ptr=head;ptr;ptr=ptr->next)
        printf("%d ",ptr->num);
    printf("\n");
}

```

```

struct stud_node *InsertDoc(struct stud_node *list1,struct stud_node *list2)
{
    struct stud_node *ptr,*ptr1,*ptr2;
    ptr2=list1;
    ptr=list2;
    while(list2){
        if(list1==NULL){
            list1=ptr;
            list1->next=NULL;
        }
        else{
            while((ptr->num>ptr2->num)&&(ptr2->next!=NULL)){
                ptr1=ptr2;
                ptr2=ptr2->next;
            }
            if(ptr->num<=ptr2->num){
                list2=list2->next;
                if(list1==ptr2){
                    list1=ptr;
                    ptr1=ptr2;
                }
                else
                    ptr1->next=ptr;
                ptr->next=ptr2;
                ptr=list2;
                ptr1=ptr1->next;
            }
            else{
                ptr2->next=ptr;
                break;
            }
        }
    }
    return list1;
}

```

(5) 奇数值结点链表：输入若干个正整数（输入-1 为结束标志）建立一个单向链表，头指针为 L，将链表 L 中奇数值的结点重新组成一个新的链表 NEW，并输出新建链表的信息。试编写相应程序。

解答：参见习题程序设计第 7 题

### 三、改错题



统计专业人数：输入若干个学生的学号（共 7 位，其中第 2、3 位是专业编号），以#作为输入结束标志，将其生成一个链表，统计链表中专业为计算机（编号为 02）的学生人数。

（源程序 error11\_4.cpp）

错误行号： 20	正确语句： <u>p-&gt;next = head;</u>
错误行号： 25	正确语句： <u>for(p = head; p!= NULL; p = p-&gt;next)</u>
错误行号： 26	正确语句： <u>if(p-&gt;code[1] == '0' &amp;&amp; p-&gt;code[2] == '2')</u>

#### 四、拓展编程题

（1）删除结点：输入若干个正整数（输入-1 为结束标志）建立一个单向链表，再输入一个整数 m，删除链表中值为 m 的所有结点。试编写相应程序。

解答：参见习题程序设计第 8 题

（2）链表逆置：输入若干个正整数（输入-1 为结束标志）建立一个单向链表，再将链表逆置后输出，即表头置为表尾，表尾置为表头。试编写相应程序。

解答：

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    struct node *next;
    int x;
} *head;
void InsertDoc(int x) {
    struct node *doc = (struct node*) malloc(sizeof(struct node));
    doc->next = NULL;
    doc->x = x;
    if (head == NULL) {
        head = doc;
    } else {
        struct node *p = head;
        while (p->next != NULL) {
            p = p->next;
        }
        p->next = doc;
    }
}
void ReverseDoc() {
    struct node *p = head;
    head = NULL;
    while (p != NULL) {
        struct node *t = p->next;
        p->next = head;
        head = p;
        p = t;
    }
}
```

```
}  
void PrintDoc() {  
    struct node *p = head;  
    while (p != NULL) {  
        printf("%d ", p->x);  
        p = p->next;  
    }  
    printf("\n");  
}  
int main() {  
    int x, m;  
    head = NULL;  
    while(1) {  
        scanf("%d", &x);  
        if (x == -1) break;  
        InsertDoc(x);  
    }  
    ReverseDoc();  
    PrintDoc();  
    return 0;  
}
```