

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“单词索引”，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。

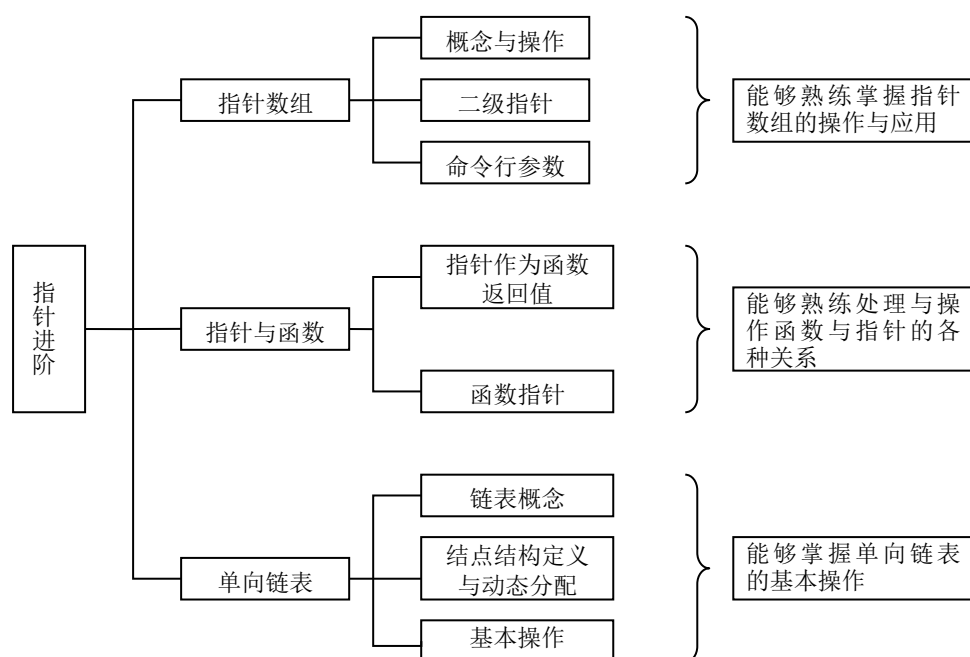






图 11.1 知识能力结构图

11.2 讲稿

1	 Chap 11 指针进阶 11.1 单词索引 11.2 字符定位 11.3 用链表构建学生信息库	本章分 3 节。
2	 本章要点 <ul style="list-style-type: none">■ 指针数组和指向指针的指针是如何被定义和使用的？■ 指针如何作为函数的返回值？■ 指向函数的指针的意义是什么？■ 什么是结构的递归定义，哪种应用需要这种定义方法？■ 对链表这种数据结构，如何使用动态内存分配操作？■ 如何建立单向链表并实现插入、删除以及查找操作？	提出本章的学习要点。
3	 11.1 单词索引 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	 11.1.1 程序解析 <ul style="list-style-type: none">■ 【例11-1】一个单词表存放了五个表示颜色的英文单词，输入一个字母，在单词表中查找并输出所有以此字母开头的单词，若没有找到，输出Not Found。	介绍例 11-1 题目要求，每个英文单词是一个字符串。

5	<div><div>11.1.1 程序解析</div><pre>#include<stdio.h> int main(void) { int i, flag = 0; char ch; const char *color[5] = {"red", "blue", "yellow", "green", "black"}; printf("Input a letter: "); ch = getchar(); for(i = 0; i < 5; i++) { if(*color[i] == ch){ flag = 1; puts(color[i]); } } if(flag == 0) { printf("Not Found\n"); } return 0; }</pre><div>运行结果1 Input a letter: y yellow 运行结果2 Input a letter: a Not Found</div><div>指针数组</div></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>*color[i]表示第 i 个单词首字母。</p>
6	<div><div>11.1.2 指针数组的概念</div><ul style="list-style-type: none">■ 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，那么这个数组就是指针数组。■ 一维指针数组定义的一般格式为： 类型名 *数组名[数组长度]</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><div>11.1.2 指针数组的概念</div><ul style="list-style-type: none">■ int a[10];<ul style="list-style-type: none">□ a是一个数组，它有10个元素□ 每个元素的类型都是整型■ char *color[5];<ul style="list-style-type: none">□ color是一个数组，它有5个元素□ 每个元素的类型都是字符指针</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><div>11.1.2 指针数组的概念</div><pre>const char *color[5] = {"red", "blue", "yellow", "green", "black"};</pre><ul style="list-style-type: none">□ color是一个数组，它有5个元素□ 每个元素的类型都是字符指针□ 数组元素可以处理字符串□ const: 限定变量值不被改变<p>对指针数组元素的操作:</p><ul style="list-style-type: none">□ printf ("%s %x\n", color[i], color[i]);<div>对指针数组元素的操作和对同类型指针变量的操作相同</div><div></div></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="304 203 791 573" data-label="Complex-Block"> <div> <div>11.1.2 指针数组的概念</div> <div> <div>■ 继续执行:</div> <div> char * tmp; tmp = color[0]; color[0] = color[4]; color[4] = tmp; </div> </div> <div> <div>color</div> <div> <div>color[0]</div> <div>color[1]</div> <div>color[2]</div> <div>color[3]</div> <div>color[4]</div> </div> <div> <div>red10</div> <div>blue10</div> <div>yellow10</div> <div>green10</div> <div>black10</div> </div> </div> <div>color[0]与color[4]交换后的情况</div> <div> <div>指针数组操作时:</div> <ul style="list-style-type: none"> 可以直接对数组元素进行引用操作 tmp = color[0]; 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)); </div> </div> </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="304 618 791 999" data-label="Complex-Block"> <div> <div>11.1.3 指向指针的指针（二级指针）</div> <div> <div>■ 指向指针的指针(二级指针)一般定义为:</div> <div>类型名 **变量名</div> </div> <div> <div>int a = 10;</div> <div>int *p = &a;</div> <div>int **pp = &p;</div> </div> <div> <div>pp</div> <div>p</div> <div>a</div> <div>&p</div> <div>&a</div> <div>10</div> <div>*p</div> <div>**pp</div> </div> </div> </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="304 1032 791 1424" data-label="Complex-Block"> <div> <div>int a = 10, b = 20, t;</div> <div>int *pa = &a, *pb = &b, *pt;</div> <div>int **ppa = &pa, **ppb = &pb, **ppt;</div> </div> <div> <div>ppa</div> <div>pa</div> <div>a</div> <div>ppb</div> <div>pb</div> <div>b</div> <div>&pb</div> <div>&a</div> <div>10</div> <div>20</div> <div>**ppa</div> <div>*pb</div> <div>**ppb</div> <div>*pa</div> </div> <div> <div>操作(1): ppt = ppb; ppb = ppa; ppa = ppt;</div> <div>操作(2): pt = pb; pb = pa; pa = pt;</div> <div>操作(3): t = b; b = a; a = t;</div> </div> </div>	<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="304 1447 791 1839" data-label="Complex-Block"> <div> <div>二维数组的指针形式</div> <div>int a[3][4];</div> <div> <div>看成是由a[0]、a[1]、a[2]组成的一维数组</div> <div>a[0]、a[1]、a[2]各自又是一个一维数组</div> <div>二维数组是数组元素为一维数组的一维数组</div> </div> </div> <div> <div>■ a: 第0行地址（行地址）</div> <div>■ a+i: 第i行的地址</div> <div>■ *(a+i) / a[i]: 第i行首元素的地址</div> <div>■ *(a+i)+j / a[i]+j: 第i行第j个元素的地址</div> <div>■ **(a+i) / a[i][0]: 第i行首元素的值</div> <div>■ *(a+i)+j / a[i][j]: 第i行第j个元素的值</div> </div> </div>	<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><div><div>例11-3 单词索引 二级指针实现</div><pre>int main(void) { int i, flag = 0; char ch; const char *color[5] = {"red", "blue", "yellow", "green", "black"}; const char **pc; pc = color; printf("Input a letter:"); ch = getchar(); for(i = 0; i < 5; i++) { if(**(pc + i) == ch) { flag = 1; puts(*(pc + i)); } } if(flag == 0) printf("Not Found\n"); return 0; }</pre></div><div><div>二级指针</div><div>使用二级指针操作数据</div></div></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><div><div>例11-3</div><div><div>pc ⇔ color ⇔ &color[0]</div><div>*pc ⇔ color[0]</div><div>*(pc+i) ⇔ color[i]</div><div>**pc ⇔ *(*pc) ⇔ *color[0] : 'r'</div></div><div><div>pc</div><div>color</div><div>color[0]</div><div>color[1]</div><div>color[2]</div><div>color[3]</div><div>color[4]</div><div>*pc</div><div>red\0</div><div>blue\0</div><div>yellow\0</div><div>green\0</div><div>black\0</div></div></div></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><div><div>11.1.4 用指针数组处理多个字符串</div><div>1. 指针数组与二维数组</div><div><div>□ 二维字符数组</div><div>char ccolor[][7] = {"red", "blue", "yellow", "green", "black"};</div></div><div><div>□ 指针数组</div><div>const char *pcolor[] = {"red", "blue", "yellow", "green", "black"};</div></div></div><div><div>使用指针数组更节省内存空间</div><div><div>ccolor</div><div>r e d \0</div><div>b l u e \0</div><div>y e l l o w \0</div><div>g r e e n \0</div><div>b l a c k \0</div></div><div><div>pcolor</div><div>red\0</div><div>blue\0</div><div>yellow\0</div><div>green\0</div><div>black\0</div></div></div></div>	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。
16	<div><div><div>11.1.4 用指针数组处理多个字符串</div><div>2. 用指针数组操作多个字符串</div><div><div>【例11-4】将5个字符串从小到大排序后输出。</div><div><div>void fsort(int a[], int n);</div><div>void main()</div><div>{ int i;</div><div>int a[5] = {6, 5, 2, 8, 1};</div><div>fsort(a, 5);</div><div>for(i = 0; i < 5; i++)</div><div>printf("%d ", a[i]);</div><div>}</div></div><div><div>void fsort(const char *color[], int n);</div><div>void main()</div><div>{ int i;</div><div>const char *pcolor[5] = {"red", "blue", "yellow", "green", "black"};</div><div>fsort(pcolor, 5);</div><div>for(i = 0; i < 5; i++)</div><div>printf("%s ", pcolor[i]);</div><div>}</div></div></div></div></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><div><div>例11-4 排序函数</div><div><pre>void fsort(int a[], int n) { int k, j; int temp; for(k = 1; k < n; k++) for(j = 0; j < n-k; j++) if(a[j] > a[j+1]){ temp = a[j]; a[j] = a[j+1]; a[j+1] = temp; } }</pre></div><div><pre>void fsort(const char *color[], int n) { int k, j; const char *temp; for(k = 1; k < n; k++) for(j = 0; j < n-k; j++) if(strcmp(color[j], color[j+1]) > 0){ temp = color[j]; color[j] = color[j+1]; color[j+1] = temp; } }</pre></div></div></div> <td><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 <code>strcmp()</code>。</p></td>	<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 <code>strcmp()</code>。</p>
18	<div><div><div>例11-4</div><div><p>排序前</p><p>排序后</p></div></div></div> <td><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></td>	<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><div><div>11.1.4 用指针数组处理多个字符串</div><div>3. 动态输入多个字符串</div><div><div>■ 例11-5 解密英文藏头诗。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。</div><div>□ 编写程序，输出一首英文藏头诗的真实含义。输入的藏头诗小于20行，每行不超过80个字符，以#作为输入结束标志，使用动态内存分配方法处理字符串的输入。</div></div><div><pre>I come into a dream Leaves fall down but spring over a lake birds flying village have its nice morning everywhere can feel happiness Years have never been owners don't need anything until the sun bring another wind # ILoveYou</pre></div></div></div> <td><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></td>	<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><div><div>例11-5 解密英文藏头诗</div><div><pre>#include <stdio.h> #include <stdlib.h> #include <string.h> int main(void) { int i, n = 0; char *poem[20], str[80], mean[20]; gets(str); while(str[0] != '#') { poem[n] = (char *)malloc(sizeof(char)*(strlen(str)+1)); /* 动态分配 */ strcpy(poem[n], str); /* 将输入的字符串赋值给动态内存单元 */ n++; gets(str); } for(i = 0; i < n; i++){ mean[i] = *poem[i]; /* 每行取第一个字符 */ free(poem[i]); /* 释放动态内存单元 */ } mean[i] = '\0'; puts(mean); return 0; }</pre></div><div>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。</div></div></div> <td><p>运行演示示例程序。</p><p>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。</p><p>在输入多个字符串时，先将字符串暂存于字符数组 <code>str</code> 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 <code>str</code> 中的字符串复制至此内存单元。</p><p>*<code>poem[i]</code>是每个字符串第一个字符。</p></td>	<p>运行演示示例程序。</p> <p>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。</p> <p>在输入多个字符串时，先将字符串暂存于字符数组 <code>str</code> 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 <code>str</code> 中的字符串复制至此内存单元。</p> <p>*<code>poem[i]</code>是每个字符串第一个字符。</p>

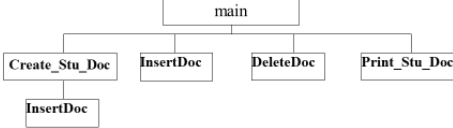
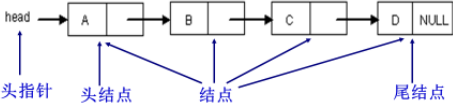
21	<div>例11-6 随机发牌</div> <p>例11-6 一副纸牌有52张，4种花色，每种花色13张。 用程序模拟随机发牌过程，将52张牌按轮转的方式发放给4人，并输出发牌结果。</p> <table><tr><td>Player 1:</td><td>Player 2:</td><td>Player 3:</td><td>Player 4:</td></tr><tr><td>10 of Spade</td><td>6 of Heart</td><td>4 of Heart</td><td>7 of Heart</td></tr><tr><td>6 of Diamond</td><td>6 of Spade</td><td>9 of Diamond</td><td>J of Club</td></tr><tr><td>9 of Spade</td><td>2 of Heart</td><td>3 of Heart</td><td>9 of Heart</td></tr><tr><td>8 of Spade</td><td>2 of Spade</td><td>2 of Diamond</td><td>A of Diamond</td></tr><tr><td>J of Heart</td><td>J of Spade</td><td>A of Spade</td><td>4 of Spade</td></tr><tr><td>5 of Club</td><td>3 of Spade</td><td>10 of Club</td><td>Q of Spade</td></tr><tr><td>K of Spade</td><td>K of Club</td><td>10 of Heart</td><td>4 of Diamond</td></tr><tr><td>7 of Club</td><td>5 of Spade</td><td>5 of Diamond</td><td>A of Heart</td></tr><tr><td>8 of Diamond</td><td>3 of Club</td><td>Q of Heart</td><td>7 of Spade</td></tr><tr><td>9 of Club</td><td>7 of Diamond</td><td>Q of Club</td><td>8 of Heart</td></tr><tr><td>8 of Club</td><td>J of Diamond</td><td>6 of Club</td><td>Q of Diamond</td></tr><tr><td>3 of Diamond</td><td>5 of Heart</td><td>A of Club</td><td>4 of Club</td></tr><tr><td>K of Heart</td><td>2 of Club</td><td>10 of Diamond</td><td>K of Diamond</td></tr></table>	Player 1:	Player 2:	Player 3:	Player 4:	10 of Spade	6 of Heart	4 of Heart	7 of Heart	6 of Diamond	6 of Spade	9 of Diamond	J of Club	9 of Spade	2 of Heart	3 of Heart	9 of Heart	8 of Spade	2 of Spade	2 of Diamond	A of Diamond	J of Heart	J of Spade	A of Spade	4 of Spade	5 of Club	3 of Spade	10 of Club	Q of Spade	K of Spade	K of Club	10 of Heart	4 of Diamond	7 of Club	5 of Spade	5 of Diamond	A of Heart	8 of Diamond	3 of Club	Q of Heart	7 of Spade	9 of Club	7 of Diamond	Q of Club	8 of Heart	8 of Club	J of Diamond	6 of Club	Q of Diamond	3 of Diamond	5 of Heart	A of Club	4 of Club	K of Heart	2 of Club	10 of Diamond	K of Diamond	介绍 11-6 的基本功能要求
Player 1:	Player 2:	Player 3:	Player 4:																																																							
10 of Spade	6 of Heart	4 of Heart	7 of Heart																																																							
6 of Diamond	6 of Spade	9 of Diamond	J of Club																																																							
9 of Spade	2 of Heart	3 of Heart	9 of Heart																																																							
8 of Spade	2 of Spade	2 of Diamond	A of Diamond																																																							
J of Heart	J of Spade	A of Spade	4 of Spade																																																							
5 of Club	3 of Spade	10 of Club	Q of Spade																																																							
K of Spade	K of Club	10 of Heart	4 of Diamond																																																							
7 of Club	5 of Spade	5 of Diamond	A of Heart																																																							
8 of Diamond	3 of Club	Q of Heart	7 of Spade																																																							
9 of Club	7 of Diamond	Q of Club	8 of Heart																																																							
8 of Club	J of Diamond	6 of Club	Q of Diamond																																																							
3 of Diamond	5 of Heart	A of Club	4 of Club																																																							
K of Heart	2 of Club	10 of Diamond	K of Diamond																																																							
22	<div>例11-6 分析</div> <pre>struct card{ /* 用结构表示一张牌 */ int suit; /* 0~4对应花色Heart, Diamond, Club, Spade */ int face; /* 0~12对应点数A, K, Q, J, 10, 9, ... 3, 2 */ }; 4 3 示例中第0张牌的点数和花色的值？ face[4] suit[3] struct card deck[52]; /*按顺序存放4个人的牌 */ player1: deck[0]-deck[12] player2: deck[13]-deck[25] player3: deck[26]-deck[38] player4: deck[39]-deck[51] 以字符串的形式，输出每张牌的点数和花色： const char *suit[] = {"Heart", "Diamond", "Club", "Spade"}; const char *face[] = {"A", "K", "Q", "J", "10", "9", "8", "7", "6", "5", "4", "3", "2"}; 输出第i张牌的点数和花色： printf("%s of %s\n", face[deck[i].face], suit[deck[i].suit]);</pre> <table><tr><td>Player 1:</td></tr><tr><td>10 of Spade</td></tr><tr><td>6 of Diamond</td></tr><tr><td>9 of Spade</td></tr><tr><td>8 of Spade</td></tr><tr><td>J of Heart</td></tr><tr><td>5 of Club</td></tr><tr><td>K of Spade</td></tr><tr><td>7 of Club</td></tr><tr><td>8 of Diamond</td></tr><tr><td>9 of Club</td></tr><tr><td>8 of Club</td></tr><tr><td>3 of Diamond</td></tr><tr><td>K of Heart</td></tr></table>	Player 1:	10 of Spade	6 of Diamond	9 of Spade	8 of Spade	J of Heart	5 of Club	K of Spade	7 of Club	8 of Diamond	9 of Club	8 of Club	3 of Diamond	K of Heart	每张牌用结构类型 card 表示 结构数组 deck 存储 52 张牌 指针数组 suit 指向 4 种花色，指针数组 face 指向 13 个数，前面加 const 表示指针数组指向的内容不可修改																																										
Player 1:																																																										
10 of Spade																																																										
6 of Diamond																																																										
9 of Spade																																																										
8 of Spade																																																										
J of Heart																																																										
5 of Club																																																										
K of Spade																																																										
7 of Club																																																										
8 of Diamond																																																										
9 of Club																																																										
8 of Club																																																										
3 of Diamond																																																										
K of Heart																																																										
23	<div>例11-6 源程序</div> <pre>#include <stdio.h> #include <stdlib.h> #include <time.h> struct card{ int suit; int face; }; void deal(struct card *wdeck); /* 发牌函数声明 */ int main(void) { int i; struct card deck[52]; const char *suit[] = {"Heart", "Diamond", "Club", "Spade"}; const char *face[] = {"A", "K", "Q", "J", "10", "9", "8", "7", "6", "5", "4", "3", "2"}; deal(deck); /* 调用函数，实现发牌 */ for(i = 0; i < 52; i++){ if(i%13 == 0) printf("Player %d:\n", i / 13 + 1); printf("%s of %s\n", face[deck[i].face], suit[deck[i].suit]); } return 0; }</pre>	主函数中调用 deal(deck)实现随机发牌																																																								
24	<div>例11-6 源程序续</div> <pre>void deal(struct card *wdeck) /* 发牌 */ { int i, m, t; static int temp[52] = {0}; /* 发牌标记 0:未发 1:已发 */ srand(time(NULL)); /* 设定随机数的产生与系统时钟关联 */ for(i = 0; i < 52; i++){ while(1){ m = rand() % 52; /* 计算机随机产生一个0~51之间的数 */ if(temp[m] == 0) break; } temp[m] = 1; /* 4人轮转发牌 */ t = (i % 4) * 13 + (i / 4); wdeck[t].suit = m / 13; wdeck[t].face = m % 13; } }</pre>	函数 deal()中，数组 temp 用于表示一副按顺序排放的牌，使用 srand()和 rand()产生随机数 m，在检查第 m 张牌未发出后，按 4 人轮转的方式将牌发给每个人，存放在 wdeck 指向的数组中，t 表示当前存放的位置。																																																								

25	<p>*11.1.5 命令行参数</p> <ul style="list-style-type: none"> ■ 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。 ■ 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。 ■ 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 <p>test world (test是命令名，world是命令行参数)</p>	介绍命令行参数的作用和使用环境。
26	<p>*11.1.5 命令行参数</p> <ul style="list-style-type: none"> ■ 命令行的一般形式为： 命令名 参数1 参数2 ... 参数n <div data-bbox="395 770 641 828"> <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"> ■ 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。 	<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>
27	<p>*11.1.5 命令行参数</p> <ul style="list-style-type: none"> ■ 带参数的main()函数格式： int main(int argc, char *argv[]) { } ■ 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）。 	带参数的main()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。
28	<p>*11.1.5 命令行参数</p> <ul style="list-style-type: none"> ■ 【例11-7】编写C程序e 令行参数在同一行上输入 <pre>#include <stdio.h> int main(int argc, char *argv) { int k; for(k = 1; k < argc; k++) printf("%s ", argv[k]); printf("\n"); return 0; }</pre> <div data-bbox="549 1541 801 1666"> </div> <div data-bbox="568 1684 788 1729"> <p>/* 从第1个命令行参数开始 */ /* 打印命令行参数 */</p> </div> <div data-bbox="577 1729 804 1841"> <p>运行结果 在命令行状态下输入： echo How are you? 输出： How are you?</p> </div>	<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>

29	<h2>11.2 字符定位</h2> <h3>11.2.1 程序解析</h3> <h3>11.2.2 指针作为函数的返回值</h3> <h3>*11.2.3 指向函数的指针</h3>	11.2.3 指向函数的指针（函数指针）为选学内容。
30	<h3>11.2.1 程序解析</h3> <ul style="list-style-type: none"> ■ 【例11-8】字符定位。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。 ■ 要求定义函数<code>match(char *s, char ch)</code>，在字符串<code>s</code>中查找字符<code>ch</code>，如果找到则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<code>NULL</code>。 <pre> program r rogram </pre>	介绍程序要求。 该示例程序的知识要点是指针作为函数的返回值。
31	<h3>例11-8 字符定位 源程序</h3> <pre> #include <stdio.h> char *match(char *s, char ch) /* 函数返回 { while(*s != '\0'){ if(*s == ch) return(s); /* 若找到字符c s++; } return(NULL); /* 没有找到ch. } int main(void) { char ch, str[80], *p = NULL; scanf("%s", str); /* 输入一个字 getchar(); /* 跳过输入字 ch = getchar(); /* 输入一个字 if((p = match(str, ch)) != NULL) /* 0 printf("%s\n", p); else printf("Not Found\n"); return 0; } </pre> <div style="display: flex; justify-content: space-between;"> <div> <p>运行结果1 <u>University</u> v</p> <p>运行结果2 <u>school</u> a</p> <p>Not Found</p> </div> <div> <p>•函数<code>match()</code>返回一个地址（指针）</p> <p>•在主函数中，用字符指针<code>p</code>接收<code>match()</code>返回的地址，从<code>p</code>指向的存储单元开始，连续输出其中的内容，直至'<code>\0</code>'为止。</p> </div> </div>	在此程序中，函数 <code>match()</code> 将查询到的字符位置（地址值）作为返回值。 主函数中，字符指针变量 <code>p</code> 用于接收 <code>match()</code> 的返回值，并从 <code>p</code> 指向的存储单元开始，输出字符串内容。
32	<h3>11.2.2 指针作为函数的返回值</h3> <ul style="list-style-type: none"> ■ 函数返回值的类型 <ul style="list-style-type: none"> □ 整型 □ 浮点型 □ 字符型 □ 结构类型 □ 指针（返回一个地址） ■ 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。 	函数的返回值可以是 C 语言中的任何类型，甚至指针。

33	<div><div><div>11.2.2 指针作为函数的返回值</div><div>思考</div><div><div><div>■ 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？</div><div><pre>char * match() { char ch, str[80], *s = str; /* 定义局部数据对象 */ scanf ("%s", str); getchar (); ch = getchar (); while (*s != '\0'){ if (*s == ch) return s; /* 返回局部字符串地址 */ s++; } return (NULL); }</pre></div><div>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</div><div>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</div></div></div></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>					
34	<div><div><div>*11.2.3 指向函数的指针</div><div><div><div>■ 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）</div><div>入口地址</div><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>■ 在C语言中，函数名代表函数的入口地址。</div><div>■ 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。</div><div>■ 通过函数指针可以调用函数，它也可以作为函数的参数。</div></div></div></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							
35	<div><div><div>*11.2.3 指向函数的指针</div><div>1.函数指针的定义</div><div><div>■ 函数指针定义的一般格式为：</div><div>类型名 (*变量名)(参数类型表);</div><div>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。</div></div><div><div>■ 例如：</div><div><pre>int (*funptr)(int, int);</pre></div><div>定义一个函数指针funptr，它可以指向有两个整型参数且返回值类型为int的函数。</div></div></div></div>	<p>介绍函数指针的定义格式。</p>					
36	<div><div><div>*11.2.3 指向函数的指针</div><div>2.通过函数指针调用函数</div><div><div>■ 通过函数指针调用函数的一般格式为：</div><div>(*函数指针名)(参数表)</div></div><div><div>■ 例如：</div><div><pre>int fun (int x, int y); int (*funptr) (int, int); funptr = fun; (*funptr) (3 , 5);</pre></div></div></div></div>	<p>通过函数指针调用所指向的函数的方法。</p> <p>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。</p>					

37	<p>*11.2.3 指向函数的指针</p> <p>3. 函数指针作为函数的参数</p> <ul style="list-style-type: none"> ■ 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。 ■ 例如： <pre> f(int (*funptr)(int, int)) { ... } void main() { ... int (*funptr)(int, int); funptr = fun; f(funptr); ... }</pre>	<p>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。</p> <p>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。</p>
38	<p>*11.2.3 指向函数的指针</p> <p>例11-9 编写一个函数calc(f, a, b)，用梯形公式求函数f(x)在[a, b]上的数值积分。</p> $\int_a^b f(x)dx = (b-a)/2 * (f(a) + f(b))$ <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>分析：calc()是一个通用函数，用梯形公式求解数值积分。它和被积函数f(x)以及积分区间[a, 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>	<p>例 11-9 要求通过一个函数 calc()实现对不同积分公式的调用。</p> <p>其方法就是在函数 calc()中设置一个函数指针参数，可以指向不同的积分公式函数。</p>
39	<p>*11.2.3 指向函数的指针</p> <p>/* 计算数值积分（函数指针作为函数参数示例） */</p> <pre> int main(void) { double result; double (*funp)(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); funp = f2; /* 对函数指针funp赋值 */ result = calc(funp, 1.0, 2.0); /* 函数指针funp作为函数calc的实参 */ printf("2: resule=%.4fn", result); return 0; } /* 函数指针funp作为函数的形参 */ double calc(double (*funp)(double), double a, double b) { double z; z = (b-a)/2 * ((*funp)(a) + (*funp)(b)); /* 调用funp指向的函数 */ return(z); }</pre> <div data-bbox="619 1249 805 1339" style="border: 1px solid black; padding: 5px; width: fit-content;"> <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 funp 如何发挥作用。</p>
40	<p>11.3 用链表构建学生信息库</p> <p>11.3.1 程序解析</p> <p>11.3.2 链表的概念</p> <p>11.3.3 单向链表的常用操作</p>	

41	<div><div><div></div><div></div><div></div></div><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></div>	简要分析例 11-10 的程序功能和结构。
42	<div><div><div></div><div></div><div></div></div><div><h3>11.3.1 程序解析</h3><pre>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></div></div>	介绍结点结构、各功能函数的功能和参数。
43	<div><div><div></div><div></div><div></div></div><div><h3>11.3.2 链表的概念</h3><p>■ 链表是一种常见而重要的动态存储分布的数据结构。它由若干个同一结构类型的“结点”依次串接而成。链表分单向链表和双向链表。</p></div></div>	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。
44	<div><div><div></div><div></div><div></div></div><div><h3>11.3.2 链表的概念</h3><p>■ 通常使用结构来定义单向链表结点的数据类型：</p><pre>struct stud_node{ int num; char name[20]; int score; struct stud_node *next; };</pre><div>结构的递归定义</div></div></div>	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。

45	<div data-bbox="300 203 794 241" data-label="Section-Header"> <h3>11.3.2 链表的概念</h3> </div> <div data-bbox="320 286 462 313" data-label="Section-Header"> <h4>■ 与数组比较</h4> </div> <div data-bbox="347 315 794 481" data-label="List-Group"> <ul style="list-style-type: none"> □ 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。 □ 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），故使用链表可以节省内存，操作效率高。 </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>
46	<div data-bbox="300 620 794 658" data-label="Section-Header"> <h3>11.3.2 链表的概念</h3> </div> <div data-bbox="320 703 529 730" data-label="Section-Header"> <h4>■ 动态分配相关函数</h4> </div> <div data-bbox="347 732 794 992" data-label="List-Group"> <ul style="list-style-type: none"> □ void *malloc(unsigned size) 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 NULL(0)。 例如：(int *) malloc(sizeof(int)) (struct student *) malloc(sizeof(struct student)) □ void free(void *ptr) 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); </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>
47	<div data-bbox="300 1037 794 1075" data-label="Section-Header"> <h3>11.3.3 单向链表的常用操作</h3> </div> <div data-bbox="336 1128 504 1236" data-label="List-Group"> <ul style="list-style-type: none"> ■ 1. 链表的建立 ■ 2. 链表的遍历 ■ 3. 插入结点 ■ 4. 删除结点 </div>	<p>单向链表的基本操作</p>
48	<div data-bbox="300 1453 794 1491" data-label="Section-Header"> <h3>11.3.3 单向链表的常用操作</h3> </div> <div data-bbox="336 1543 485 1570" data-label="Section-Header"> <h4>■ 1. 链表的建立</h4> </div> <div data-bbox="488 1514 756 1832" data-label="Diagram"> <pre> graph TD Start([head=tail=NULL]) --> Input1[输入: num, name, score] Input1 --> Num0{num=0} Num0 -- 是 --> Exit([退出]) Num0 -- 否 --> Alloc[p = (struct stud_node *) malloc(size)] Alloc --> Input2[输入: 学号, 姓名, 成绩] Input2 --> NextNull[p->next=NULL] NextNull --> HeadNull{head=NULL} HeadNull -- 是 --> HeadP[head=p] HeadNull -- 否 --> TailNext[tail=p->next] TailNext --> TailP[tail=p] TailP --> Input1 </pre> </div>	<p>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。</p>

49	<div data-bbox="300 203 820 591" data-label="Complex-Block"> <div>11.3.3 单向链表的常用操作</div> <div>1. 链表的建立</div> <pre> head = tail = NULL; scanf("%d%s%d", &num, 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%s%d", &num, name, &score); } </pre> <div>尾部插入</div> <div>头部插入法:</div> <pre> p->next = head; head = p; </pre> </div>	<p>根据算法思路，详细介绍实现语句。</p> <p>在将结点插入链表时，有尾部插入和头部插入两种方法。</p>
50	<div data-bbox="300 620 820 1008" data-label="Complex-Block"> <div>11.3.3 单向链表的常用操作</div> <div>2. 链表的遍历</div> <pre> for(ptr = head; ptr != NULL; ptr = ptr->next) printf("%d\t%s\t%d\n", ptr->num, ptr->name, ptr->score); </pre> <div>(a) 执行 ptr = ptr->next 前</div> <div>(b) 执行 ptr = ptr->next 后</div> </div>	<p>结合图示，介绍链表的遍历方法。</p>
51	<div data-bbox="300 1037 820 1424" data-label="Complex-Block"> <div>11.3.3 单向链表的常用操作</div> <div>3. 插入结点</div> <div>■ 先连: s->next = ptr->next;</div> <div>■ 后断: ptr->next = s;</div> </div>	<p>结合图示说明：要在链表的结点 ptr 后面插入结点 s，必须“先连后断”。</p>
52	<div data-bbox="300 1453 820 1841" data-label="Complex-Block"> <div>11.3.3 单向链表的常用操作</div> <div>4. 删除结点</div> <div>■ ptr2 = ptr1->next;</div> <div>■ 先接: ptr1->next = ptr2->next;</div> <div>■ 后删: free(ptr2);</div> <div>ptr1->next = ptr1->next->next;</div> </div>	<p>结合图示说明：要删除链表结点 ptr1 后面的结点，必须“先接后删”。</p>

53	<div><div>本章总结</div><div><ul style="list-style-type: none">■ 指针数组<ul style="list-style-type: none">□ 指针数组概念与应用□ 指向指针的指针（二级指针）□ 命令行参数■ 指针与函数<ul style="list-style-type: none">□ 指针作为函数的返回值□ 函数指针■ 单向链表<ul style="list-style-type: none">□ 链表的概念□ 结点的结构定义与动态分配□ 链表的基本操作（建立、遍历、插入、删除）</div><div><ul style="list-style-type: none">•能够熟练掌握指针数组的操作与应用•能够熟练处理与操作函数与指针的各种关系•能够掌握单向链表的基本操作</div></div>	回顾和总结本章的教学要点，对学生提出能力要求。
----	---	-------------------------

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	A	C	B

二. 填空题

1	continue	2	world-w orld-o-p
3	int oint 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1	4	st[0]; strcmp(st[i], smin) smin
5	GetMax(score, 10) score + pos	6	p = head 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

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。

解答：略

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

中文藏头诗：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。本题要求将一首四行诗每句的第一个汉字（一个汉字占两个字节）连接在一起形成一个字符串并输出。

解答：

错误行号： 14	正确语句： <code>printf("%s\n", change(p, 4));</code>
错误行号： 25	正确语句： <code>t[2*i+1] = *((s+i)+1);</code>
错误行号： 27	正确语句： <code>t[2*i] = '\0';</code>

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;
    }

```

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。

解答：略

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);
            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。解答：

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

```