

每多学一点知识
就少写一行代码

C指针 编程之道

孔浩 张华杰 陈猛 编著

关注策划微博：
<http://t.sina.com.cn/jamjar>



```
typedef struct {char name[10];
```

```
char num[10];char sex[10];char age[10];char record[10];
```

```
char position[15];char wanges[10];
```

```
char tel[15];char addr[50];people; void Menu();
```

```
void Display();void Search();void Add();
```

```
void SearchForN
```

```
void SearchForPosition();void M
```

```
void Delete();void main()
```

```
u();void Menu();int m;while(1){p
```

```
("\\n\\n\\n\\n\\n");printf("
```

```
\\n");printf("\\n\\t\\t\\t
```

```
");printf("\\n\\t\\t\\t2");printf("\\n\\t\\t\\t
```

```
printf("\\n\\t\\t\\t4");printf("\\n\\t\\t\\t
```

```
scanf("%d",&m);(m>=0&&m<=5)
```

```
witch(m){case 1:Add();break;c
```

```
2: Search ();break;case 3:Modify()
```

```
break;case 4:Delete();break;
```

```
case 5:Display();}
```

```
{peopleone;FILE *fp;int flag=0;/**/if((fp=fopen("workers.ab"))==NULL)
```

```
{printf("\\n");exit();}printf("\\n--\\n");scanf("%s",one.name);printf("\\n");scanf("%s",one.num);**/while(flag){fla
```

```
if(flag){printf("\\n");scanf("%s",one.num);}}printf("\\n");scanf("%s",one.sex);printf("\\n");scanf("%s",
```

```
s",one.age);printf("\\n");scanf("%s",one
```

```
printf("\\n");scanf("%s",one.position);prin
```

```
printf("\\n");scanf("%s",one.tel);printf("\\n");scanf("%s",one.ad
```

```
printf("\\n");fclose(fp);**/int IsUsed(char *nu){FIL
```

```
if((fp=fopen("workers",))=
```

```
NULL){printf("\\n");exi
```

```
};**/fread(&one,sizeof(people),1,fp);while(*one.num,one
```

```
return 0;fread(&one,sizeof(people),1,fp);fre
```

```
{people one;FILE *fp;/**/if((fp=fopen("workers.ab"))=
```

```
{printf("\\n");exit();}/**/printf("20s\\n",****);
```

```
printf("\\n-10s%-8s%-5s%-5s%-10s%-8s%-8s%-10s%-15s\\n",one.name,one.num,one
```

```
-5s%-5s%-10s%-8s%-8s%-10s%-15s\\n",one.name,one.num,one
```

```
osition,one.wanges,one.tel,one.addr);fre
```

```
}fclose(fp);**/void Se
```

```
{switch(i){case 1:Search ForName();break;ca
```

```
case 3:SearchForPosition();break;}printf("\\n\\n
```

```
);**/fread(&one,sizeof(people),1,fp);w
```

```
p);**/if(strcmp(one.name)==0){k++;if(k==1){printf("\\n\\n%s\\n",na);
```

```
%-5s%-5s%-10s%-8s%-8s%-10s%-15s\\n");printf("\\n-10s%-8s%-5s%-5s%-10s%-8s%-8s%-10s%-15s",one.name,one.num,one.sex,one.age,one.record,one.position,one.wanges,one.tel,one.addr);fread
```

```
(&one,sizeof(people),1,fp);if(k==0)printf("\\n\\n");elseprintf("\\n\\n
```

```
%d%s!\\n",k,na);fclose(fp);**/void Search
```

```
ForNum(){int k=0;char nu[10];FILE *fp;int IsUsed(char *nu){FILE *fp;people
```

```
rkers","rb"))=NU
```

```
&one,sizeof(people
```

人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目 (C I P) 数据

C指针编程之道 / 孔浩, 张华杰, 陈猛编著. -- 北京: 人民邮电出版社, 2011.6
ISBN 978-7-115-25084-1

I. ①C… II. ①孔… ②张… ③陈… III. ①C语言—程序设计 IV. ①TP312

中国版本图书馆CIP数据核字(2011)第055163号

内 容 提 要

本书是一本帮助程序员提高编程素养的图书, 书中结合开发人员多年的编程经验和感悟, 介绍了指针在数组中的应用、在函数中的应用、指向指针的指针、数据结构中指针的应用, 指向文件类型的指针、指针在C语言算法中的应用、典型迷宫算法实例、C语言和汇编语言的接口、Linux C编程技术简介、Linux进程与线程通信实例、C语言管理系统设计案例、C语言游戏设计案例等内容。

本书通过简单生动的语言和经典的开发实例讲解C语言指针应用的方方面面, 帮助读者完全掌握C语言指针的使用细节。

本书适合大中专院校在校生、毕业生、求职者、编程爱好者学习, 同时也可作为想要学习编程的初学者的指导用书。

C 指针编程之道

-
- ◆ 编 著 孔 浩 张华杰 陈 猛
责任编辑 蒋 佳
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
三河市潮河印业有限公司印刷
 - ◆ 开本: 800×1000 1/16
印张: 25.25
字数: 615千字 2011年6月第1版
印数: 1-3500册 2011年6月河北第1次印刷

ISBN 978-7-115-25084-1

定价: 55.00元

读者服务热线: (010)67132692 印装质量热线: (010)67129223
反盗版热线: (010)67171154

前 言

C 语言指针是一个特殊的变量，其里面存储的数值被解释成为内存里的一个地址。读者在学习指针的时候，一定要理解并且区分指针的 4 个概念：指针的类型、指针所指向的类型、指针的值（指针所指向的内存区）、指针本身所占据的内存区。

当然了，读者理解并掌握了上述的 4 个概念，只是学好 C 语言指针的前提。真正要学好指针还要留意很多指针相关的使用细节，细节决定成败，这些开发使用细节才是决定一个 C 语言程序优劣的依据。本书将带领读者走进这些很容易被忽略的细节中，让广大读者一起来感受“细节决定成败”的力量。

本书详细讲解了 C 语言开发的诸多细节，让读者透彻理解 C 语言开发的关键技术点。具体内容如下。

- C 语言指针（3 个细节提示）
- 指针在数组中的应用（6 个细节提示）
- 指针在函数中的应用（3 个细节提示）
- 指向指针的指针（2 个细节提示）
- 数据结构中指针的应用（5 个细节提示）
- 指向文件类型的指针（3 个细节提示）
- 指针在 C 语言算法中的应用（2 个细节提示）
- 典型迷宫算法实例（3 个细节提示）
- C 语言和汇编语言的接口（6 个细节提示）
- Linux C 编程技术简介（4 个细节提示）
- Linux 进程与线程通信实例（1 个细节提示）
- C 语言管理系统设计（2 个细节提示）
- C 语言游戏设计（2 个细节提示）

本书适合以下读者阅读：

C 语言开发初中级爱好者

C 语言开发专业人员

高等院校的学生

本书由孔浩、张华杰、陈猛编写。在编写过程中，严雨、房明浩、梅乐夫、王亮、门店宏、吴洋、石峰、张圣亮、邱文勋、刘鲲、朱飞、岂兴明、刘变红、周建兴、刘会灯、张高煜、邓志宝、赵红波、刘坤、刘明辉、李鹏、白学明、步士建等人参与了资料整理和代码编写，在此，对以上人员致以诚挚的谢意。

由于作者水平有限、时间仓促，书中错误和疏漏之处在所难免，恳请广大读者批评指正。欢迎广大读者访问工作室网站 <http://www.coronabook.com>，提出您的宝贵建议和意见。

编者

2011 年 3 月

目 录

第 1 章 C 语言指针	1
1.1 C 语言的灵魂.....	1
1.1.1 从这里开始.....	1
1.1.2 知识提示.....	1
1.2 相逢在 C 语言.....	1
1.2.1 初识指针——指针变量的定义.....	1
1.2.2 学以致用——指针变量的引用.....	3
1.2.3 知识提示.....	4
1.3 走近指针的日子.....	4
1.3.1 方圆世界——指针运算符.....	4
1.3.2 玩转指针——指针操作.....	5
1.3.3 知识提示.....	8
第 2 章 指针在数组中的应用	9
2.1 由内及外——数组在内存中的样子.....	9
2.1.1 物以类聚——数组.....	9
2.1.2 数组在内存中的样子.....	9
2.1.3 知识提示.....	10
2.2 自从有了你.....	11
2.2.1 数组的左膀——下标.....	11
2.2.2 数组的右臂——指针.....	11
2.2.3 知识提示.....	12
2.3 数组, 不止一面.....	12
2.3.1 引蛇出洞——数组指针的定义.....	12
2.3.2 数组指针的引用.....	15
2.3.3 知识提示.....	18
2.4 数组元素那点事.....	19
2.4.1 这是直观的世界——数组的下标表示法.....	19
2.4.2 数组的另一面——数组的指针表示法.....	23
2.4.3 异曲同工——数组的下标与指针.....	27
2.4.4 知识提示.....	30
2.5 指针集中营——指针数组.....	30
2.5.1 戏说指针数组.....	30
2.5.2 深度对话——指针数组的引用.....	31

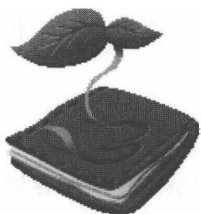
2.5.3	知识提示	34
2.6	聚焦字符串的指针	34
2.6.1	字符串指针的定义	35
2.6.2	字符串指针的引用	35
2.6.3	知识提示	40
第 3 章	指针在函数中的应用	41
3.1	完美的传递	41
3.1.1	指针新用途——指针用做函数参数	41
3.1.2	说来说去——普通参数与指针参数	49
3.1.3	知识提示	50
3.2	函数也有指针	50
3.2.1	何谓函数的指针	50
3.2.2	指点江山——指向函数的指针变量	50
3.2.3	翻来覆去——函数指针与指针函数	53
3.2.4	知识提示	54
3.3	我要的是指针，不是寂寞	54
3.3.1	函数的返回值	54
3.3.2	这个函数很奇怪——返回指针类型的函数	54
3.3.3	main 函数的返回值	55
3.3.4	知识提示	57
第 4 章	指向指针的指针	58
4.1	渐行渐进——多维数组	58
4.1.1	多维数组的定义	58
4.1.2	指针的指针	64
4.1.3	当指针遇上多维数组	67
4.1.4	知识提示	69
4.2	不能不说的秘密——main 函数的秘密	69
4.2.1	main 函数也是函数	69
4.2.2	野百合也有春天——main 函数的参数	70
4.2.3	知识提示	71
第 5 章	数据结构中指针的应用	72
5.1	程序蜗居——内存	72
5.1.1	铁打的营盘流水的兵——动态内存分配	73
5.1.2	可持续发展——内存的释放	74
5.1.3	知识提示	74
5.2	再说队列	74
5.2.1	有头有尾的队列	75

5.2.2	无头无尾的循环队列	77
5.2.3	链式队列	81
5.2.4	知识提示	83
5.3	永恒的话题——堆栈	83
5.3.1	特殊线性表之堆栈	84
5.3.2	堆栈的存储结构	84
5.3.3	知识提示	89
5.4	顺藤摸瓜——链表	89
5.4.1	链表种种	89
5.4.2	寻根问祖——链表的建立	91
5.4.3	链表的操作	95
5.4.4	知识提示	100
5.5	C世界的树	100
5.5.1	C世界的树是这样的	101
5.5.2	“Y”形的二叉树	102
5.5.3	今天，你“栽树”了吗——二叉树的创建	106
5.5.4	一个也不能少——二叉树的遍历	107
5.5.5	知识提示	112
第6章 指向文件类型的指针		113
6.1	动与静——流和文件	113
6.1.1	流动的“流”	113
6.1.2	静悄悄的玫瑰——文件	114
6.1.3	不得不说——文件类型指针	114
6.1.4	知识提示	115
6.2	进进出出的I/O	115
6.2.1	getchar()	116
6.2.2	gets()	116
6.2.3	scanf()	117
6.2.4	putchar()	120
6.2.5	puts()	120
6.2.6	printf()	121
6.2.7	知识提示	125
6.3	文件加工厂	125
6.3.1	fopen()	126
6.3.2	fclose()	127
6.3.3	fgetc()	127
6.3.4	fputc()	127
6.3.5	fgets()	128
6.3.6	fputs()	129
6.3.7	fread()	130
6.3.8	fwrite()	130
6.3.9	fprintf()	133
6.3.10	fscanf()	133
6.3.11	fseek()	133

6.3.12	rewind()	134
6.3.13	ftell()	134
6.3.14	feof()	135
6.3.15	知识提示	135
6.4	学以致用	135
第 7 章	指针在 C 语言算法中的应用	142
7.1	排序杂谈	142
7.1.1	“7”上“8”下——冒泡排序	144
7.1.2	高效快捷——快速排序	150
7.1.3	各回各家——直接选择排序	153
7.1.4	心随我动——直接插入排序	155
7.1.5	好风凭借力——希尔排序	157
7.1.6	树形结合——堆排序	160
7.1.7	分久必合——归并排序	165
7.1.8	排序方法总结	168
7.1.9	知识提示	169
7.2	查找	169
7.2.1	地毯式搜索——顺序查找	170
7.2.2	事半功倍——折半查找	173
7.2.3	集中粉碎——分块查找	177
7.2.4	知识提示	179
第 8 章	典型迷宫算法实例	180
8.1	迷宫问题	180
8.1.1	C 语言的梦幻家园——迷宫	182
8.1.2	知识提示	184
8.2	算法解析	184
8.2.1	摸着石头过河——回溯法	184
8.2.2	用回溯法解迷宫问题	189
8.2.3	原来迷宫可以这样走	197
8.2.4	知识提示	200
8.3	指针的应用	200
8.3.1	“迷宫”里的指针	200
8.3.2	知识提示	207
第 9 章	C 语言和汇编语言的接口	208
9.1	不可或缺的保护者——const 和 volatile	208
9.1.1	忠诚的卫士——const	208
9.1.2	不能没有你——volatile	210
9.1.3	知识提示	211
9.2	寄存器变量——register	211

9.2.1	直接快速的访问者——register	211
9.2.2	知识提示	212
9.3	寄存器的方方面面	212
9.3.1	纵观寄存器	212
9.3.2	寄存器的方方面面	214
9.3.3	寄存器操作——因“寄存器”而异	218
9.3.4	知识提示	221
9.4	C语言与汇编语言混合编程简介	221
9.4.1	混编方法	222
9.4.2	汇编语言与C语言混编的几点说明	222
9.4.3	知识提示	224
9.5	C语言调用汇编语言	224
9.5.1	融为一体——嵌套汇编代码	224
9.5.2	严丝缝合——调用汇编子程序	226
9.5.3	知识提示	229
9.6	汇编语言调用C语言	229
9.6.1	汇编语言主程序调用C语言子程序	229
9.6.2	应用举例	230
9.6.3	知识提示	232
第 10 章	Linux C 编程技术简介	233
10.1	Linux 系统程序设计基础	233
10.1.1	揭开那神秘的面纱——第一个 Linux C 程序	233
10.1.2	得力的助手——gcc	234
10.1.3	纠错高手——gdb 调试器	235
10.1.4	强力黏合剂——makefile	239
10.1.5	知识提示	242
10.2	Linux 下的进程控制和进程间通信	242
10.2.1	Linux 进程简介	242
10.2.2	Linux 进程相关函数	244
10.2.3	Linux 进程间通信	254
10.2.4	知识提示	263
10.3	Linux 下的线程控制	263
10.3.1	Linux 线程的概念	264
10.3.2	Linux 线程的标识	265
10.3.3	Linux 线程的创建	265
10.3.4	Linux 线程的终止	268
10.3.5	Linux 线程的同步	270
10.3.6	知识提示	270
10.4	Linux 网络编程技术简介	270
10.4.1	TCP/IP 协议简介	271

10.4.2	套接字编程简介	273
10.4.3	基本 TCP 套接字编程简介	280
10.4.4	基本 UDP 套接字编程简介	287
10.4.5	知识提示	292
第 11 章	Linux 进程与线程通信实例	293
11.1	生产者—消费者问题简介	293
11.2	用 fork()实现简单的生产者—消费者功能	294
11.3	用 fork()实现较复杂的生产者—消费者功能	296
11.4	用 clone()实现生产者—消费者的功能	302
11.4.1	互斥量	303
11.4.2	信号量	304
11.4.3	clone 系统调用	305
11.4.4	用 clone()、信号量、互斥量实现生产者—消费者功能	305
11.4.5	知识提示	313
11.5	用 pthread_create()实现生产者—消费者的功能	313
11.6	画龙点睛——fork()、clone()、pthread_create() 的综合比较	317
第 12 章	C 语言管理系统设计案例	320
12.1	员工信息管理系统	320
12.1.1	系统的功能描述	320
12.1.2	系统的总体设计	320
12.1.3	详细设计	321
12.1.4	知识提示	333
12.2	停车场管理系统	333
12.2.1	系统的功能描述	333
12.2.2	系统的总体设计	333
12.2.3	系统的详细设计	334
12.2.4	知识提示	348
第 13 章	C 语言游戏设计案例	349
13.1	猜字游戏	349
13.1.1	游戏说明	349
13.1.2	总体设计	350
13.1.3	详细设计	351
13.1.4	知识提示	369
13.2	扑克游戏	369
13.2.1	游戏说明	369
13.2.2	总体设计	370
13.2.3	详细设计	371
13.2.4	知识提示	394



第 1 章 C 语言指针

1.1 C 语言的灵魂

C 语言的入门比较容易，编程灵活，可移植性好，所以自从 C 语言产生以来，在过去的将近 40 年中，C 语言已经成了最普及的一种专业编程语言，甚至有人说 C 是进一步学习 C++ 的必由之路。总之不管如何，C 语言的流行与普及，已经说明了一切。

1.1.1 从这里开始

C 语言的高效性、灵活性、可移植性，是所有使用 C 语言的人所共识的。其中灵活性是众多 C 语言学习者看中的一点，也是众多学习其他语言的朋友想要学习 C 语言的重要原因。那 C 语言的灵活性，具体体现在哪儿呢？指针。对，就是指针。指针的应用把 C 语言的灵活性体现得淋漓尽致。所以，学习 C 语言的朋友，一定要认真学习 C 语言的指针。

从现在开始，从这里开始，让我们开始走近 C 语言的指针。

1.1.2 知识提示

提示：C 语言的重要特性是灵活性，灵活性的体现方式之一就是 C 语言指针。

1.2 相逢在 C 语言

提起指针，了解 C 语言的朋友大概都听说过“指针是 C 语言的灵魂”、“指针是 C 语言的精髓”等类似的话，但到底何谓指针？指针又何以如此备受关注呢？从本节开始，我们将认识它、了解它、掌握它，进而应用它。

1.2.1 初识指针——指针变量的定义

何谓指针？

指针是 C 语言的一种数据类型，类似于之前我们学习的整型、字符型等。既然指针也是一种类型，当然同样可以定义该类型的变量，这类变量就称为指针变量。

何以如此备受关注？

在 C 语言中，之所以如此关注指针，主要是因为指针类型的特殊性。指针类型的变量中存储的是地址，这一点不同于整型、字符型等基本类型。在计算机或单片机中，所有对数据的操作，都是通过其地址进行的，指针让程序效率更高，代码更少。

众所周知，所有的数据都是存储在存储器中，存储器中的一个字节，又称为一个存储单元。不同类型的数据，所需要占用的存储单元数不同。比如一般机器中，整型为 2 个单元、字符型为 1 个单元。在程序中，我们要不断地操作一些数据，要操作这些数据，首先要知道这些数据在哪儿，存放在什么地方。对此，有个非常经典的实例，就是我们在房间里找人的例子。我们知道，每个房间都有个房间号，如果我们知道要找的人所在的房间号，那岂不是很容易就可以找到了。根据什么找的呢？房间号。对了，就是房间号。同理，假如我们知道存放数据的“房间号”，也就很容易地能够找到这些数据。前辈们，把数据的这个“房间号”形象地称为地址，俗称指针，根据这个地址或指针就可以找到相应的数据。

有人会问，用地址是很容易找到相应的数据，但前提是要知道数据的地址啊！没错，这是肯定的。那么这些地址在哪儿了？它就在我们所说的指针变量中，也就是说，指针变量中存放的内容就是地址，或者说指针变量的值就是地址。

C 语言中，引入指针后，同时引入了“指向”的概念，即指针变量指向本身保存的内容（地址或指针）所表示的内存单元。

前面学习可知，C 语言中的所有变量，都是先定义后引用。了解了指针变量，接下来我们看看指针变量是怎么定义的。

指针变量定义的一般形式：

类型说明符*变量名；

类型说明符：表示该指针变量指向的变量的数据类型，并且该指针变量只能指向该数据类型的变量。这是指针变量的特性。

*：表示该变量为指针变量。这也是指针变量的特性。

变量名：表示指针变量的名称。

例如：

```
char *pStu;
```

*表示 pStu 是一个指针变量，char 是指针变量 pStu 指向的变量的数据类型。整个 C 语句表示

定义了一个指向 `char` 类型变量的指针变量，变量名称为 `pStu`。但具体 `pStu` 指向哪个 `char` 类型的变量，这就要看在使用 `pStu` 时，给 `pStu` 赋予的值，值是什么就指向哪儿。

为了更多了解指针变量的定义，再看下面的例子：

```
(1) int *pStu           //定义指针变量 pStu，并且 pStu 指向 int 类型变量
(2) static int *pStu  //定义指针变量 pStu，并且 pStu 指向静态整型变量
```

指针类型的变量，与其他类型的变量有共性，也有其特性，但指针变量的特性更值得我们关注，也是必须掌握的。

1.2.2 学以致用——指针变量的引用

指针让大家总是望而生畏，不到迫不得已时都不会选择使用。物质是有规律的，规律是可以掌握的，其实当我们真正掌握了指针后，我们就可以自如应用，到时望而生畏的感觉就会荡然无存了。

在介绍指针变量的引用之前，我们先来看看指针变量在引用前还有什么特殊的操作。我们都知道，指针变量是只能指向某种指定类型的变量，那么具体指向这种类型的哪个变量呢？或者说具体指向哪个存储单元呢？这就是指针变量的另一个特性，也是在使用指针变量前必须进行的操作，即给指针变量赋值，让其指向指定的内存单元。指针变量在使用前，必须指向相应的地址，否则将会造成不可预计的后果。这也是使用指针时，大家容易出错的地方。

在介绍指针变量的引用前，有必要介绍两个运算符，取地址运算符 `&` 和取值运算符 `*`。通过取地址运算符 `&` 可以获得某个变量的地址，用取值运算符 `*` 可以获得某个地址中存放的数据。

例如：

```
(1) char stuName, name; //定义字符型变量 stuName、name
(2) char *pStu;         //定义指向字符型变量的指针变量 pStu
(3) char *pNew;         //定义指向字符型变量的指针变量 pNew
(4) pStu = &stuName;   //取变量 stuName 的地址，并赋给指针变量 pStu
(5) pNew = pStu;       //把指针变量 pStu 的值赋给 pNew
(6) name = *pNew;      //把 pNew 指向的内存单元的值赋给 name
```

上述第 (4) 句包含两个意思，一是取字符型变量 `stuName` 的地址，即 `&stuName`，这里应用了取地址运算符 `&`；二是把获得的地址赋给指针变量 `pStu`，也就是说让 `pStu` 指向字符型变量 `stuName`。这里的 `pStu` 一定是指向字符型变量的指针变量。

第 (5) 句即为指针变量的引用。意思是把 `pStu` 的值赋给 `pNew`，就是说让指针变量 `pNew` 指向 `pStu` 所指向的变量，即二者此刻同时指向字符型变量 `stuName`。

第 (6) 句是取值运算符的应用，意思是把 `pNew` 指向的内存单元的值赋给变量 `name`。

从上面一段程序中可以看到，我们是先给指针变量 `pStu` 赋值，即第 (4) 句，然后才在后面的第 (5) 句中使用 `pStu` 的，即先让指针变量指向所需的变量后，才可以使用该指针变量，否则后果

无法预计，甚至会使系统瘫痪。

例如把上面程序的第（4）句去掉，改成下面这样，就有问题：

```
(1) char stuName;           //定义字符型变量
(2) char *pStu;            //定义指向字符型变量的指针变量 pStu
(3) char *pNew;           //定义指向字符型变量的指针变量 pNew
(4) pNew = pStu;          //把指针变量 pStu 的值赋给 pNew
(5) name = *pNew;         //把 pNew 指向的内存单元的值赋给 name
```

在本段程序中第（4）句，作者的意图是让指针变量 pNew、pStu 指向同一段内存单元，即把指针变量 pStu 的值赋给指针变量 pNew，该语句本身没错，是合法的 C 语句，但没有实际意义和用途，甚至会带来不可预计的后果，因为指针变量 pStu 本身指向哪儿都不知道，又怎么能引导 pNew 指向所需要的地方呢。在这个基础上，第（5）句当然也是无稽之谈了。

总而言之，指针变量在使用前，必须要让其指向某个具体的变量，即指向某个具体的内存单元，然后才能正确引用。

1.2.3 知识提示

提示 1: 指针也是一种数据类型，是专门存放地址的数据类型。

提示 2: 俗称的指针，是指地址，而不是指针变量。

提示 3: 指针是个常量，指针变量是个变量，二者是不同的概念。

提示 4: 指针变量在使用前，必须指向具体的、有效的内存单元。

1.3 走近指针的日子

学以致用，上一节学习了指针的定义，以及基本的引用方法。那么可以对指针进行哪些操作呢？其实上一节内容已经引出了指针的基本使用。指针的基本操作包括赋值、取值、指针与整数的加减、指针加减等。接下来我们就详细介绍对指针的操作。

1.3.1 方圆世界——指针运算符

学习指针，首先来看看与指针相关的运算符。跟指针相关的运算符，其实在前面章节已经提到，一个是地址运算符&，一个是取值运算符*。

1. 地址运算符&

地址运算符，又名取地址运算符，顾名思义就是取地址用的。那么取谁的地址呢？

地址运算符&是一元运算符，规定地址运算符&只能取内存中的变量的地址，就是说通过使用地址运算符&可以获得相应变量的地址，比如变量、数组元素。言外之意，就是不能通过地址运算符&获得常量、表达式之类的地址。

2. 取值运算符*

取值运算符*也是一元运算符。它通过跟指针结合使用，获取指针所指向的变量的值，这样就可以通过指针间接地访问变量了。

介绍完基本的内容，为了更明确地址运算符&和取值运算符*的应用，下面我们举例说明。

假设有两个变量，分别为 num_Max、num_Min，而且有下面一段程序：

```
(1)  int num_temp;           //定义一个整型变量 num_temp
(2)  int *p_num;           //定义一个指向整型变量的指针变量 p_num
(3)  p_num = &num_temp;   //指针变量 p_num 指向变量 num_temp
(4)  *p_num = 5;          //给 p_num 指向的变量赋值
```

在这段程序中，第（1）、（2）句定义了一个整型变量 num_temp 和一个指向整型变量的指针变量 p_num。我们主要看一下第（3）、（4）句。

第（3）句，通过使用地址运算符&，获得了变量 num_temp 的地址，并把该地址赋给指针变量 p_num。因为我们都知道，指针变量的值是个地址。该句换个说法，就是让指针变量 p_num 指向变量 num_temp。

第（4）句，通过使用取值运算符*，获得了指针变量 p_num 指向的变量，并把 5 赋给该变量。该句其实就是利用取值运算符间接访问了变量 num_temp，因为在第（3）句中，指针变量 p_num 是指向变量 num_temp 的。故第（4）句等效于下面语句：

```
num_temp = 5;
```

上面介绍了地址运算符&和取值运算符*的使用。由此看出，指针运算符的应用比较简单，前提是充分理解了地址运算符&和取值运算符*的含义。

1.3.2 玩转指针——指针操作

指针操作离不开指针运算符，上一节介绍了指针运算符，现在我们来介绍指针的一些常用操作。

1. 赋值

指针的赋值操作，其实就是指针的初始化，指针变量的初始化要用到地址运算符&。我们都知道，C语言的变量，都是先定义后使用，而指针变量却不同，指针变量在使用前，不仅要定义，而且还要赋值，也就是说在使用前还要进行初始化，赋值后才能使用。

而且，还要注意指针变量的另一个特性，给指针变量赋值时，只能赋兼容类型的值。我们都知

道，一个 `int` 类型的指针变量，只能指向 `int` 类型的变量，所以在赋值时，不能把 `float` 类型等非 `int` 类型的变量的地址赋给 `int` 类型的指针变量。

例如：

```
(1)  int x, *p;
(2)  float y;
(3)  p = &x;           //正确
(4)  p = &y;           //错误
```

第 (1) 句，定义了一个整型变量 `x` 和一个指向整型变量的指针变量 `p`。第 (2) 句，定义了一个浮点型变量 `y`。

第 (3) 句正确，把 `int` 类型的变量 `x` 的地址赋给 `p`，而指针变量 `p` 是指向 `int` 类型的指针变量。

第 (4) 句，把 `float` 类型的变量 `y` 的地址赋给 `p`，而 `p` 是指向 `int` 类型的指针变量，所以该句是不正确的。

切记，指针变量在使用前，不但要定义，而且要初始化。

2. 取值

通过指针取值，就要用到取值运算符 `*`。在前面介绍取值运算符 `*` 时，我们已经介绍过，这种操作是通过取值运算符 `*` 和指针变量的结合使用，间接访问指针指向的变量。

3. 指针与整数加法

指针与整数的加法，主要用在数组中，而且随着学习的深入会慢慢发现，指针的好多操作都是针对数组展开的。

为介绍指针与整数的加法，我们先看表 1-1。

表 1-1 指针与整数的加法

数组元素值	...	12	13	14	15	...
数组元素	...	Num[0]	Num[1]	Num[2]	Num[3]	...
内存地址	2A02	2A03-2A04	2A05-2A06	2A07-2A08	2A09-2A0A	2A0B
指针指向		↑	↑		↑	
指针 <code>p_New</code>		<code>p_New</code> 初值	<code>p_New++</code>		<code>p_Old</code>	
程序语句		(3)	(4)		(5)	

有如下程序：

```
(1)  int Num[4] = {12, 13, 14, 15};
(2)  int *p_New, *p_Old;
(3)  p_New = Num;
```