第5章

指　　针

学习目标：

(1) 掌握地址、指针、指针变量的概念。

(2) 掌握指针变量的定义，理解指针指向数据类型的意义。

(3) 掌握指针变量作为函数参数的用法。

(4) 掌握指针函数与函数指针的不同及用法。

在C语言中，指针的使用非常广泛。作为C语言区别于其他程序设计语言的主要特性之一，指针可以有效地表示和访问复杂的数据结构，动态分配内存，直接对内存地址进行操作，提高程序的执行效率。由于指针的使用比较复杂，较难掌握，而且指针的误用还会导致严重后果，甚至系统崩溃。因此，学习时要注意领会指针的本质和特点，只有谨慎地使用指针，才可以利用它写出简单、清晰、高效的程序。

5.1　指针、指针变量的概念

5.1.1　地址与指针

计算机内存中的各个存储单元都是有序的，按字节编码。字节(byte)是最小的存储单位。在计算机中，所有的数据都是存放在存储器中的。为了能够正确访问内存单元，内存都是有编址的，根据编址可以快速准确地找到需要的内存单元。内存单元的编址也叫地址，通常也把这个地址称为指针。内存单元的地址(指针)和内存单元的内容是两个不同的概念，如同银行账号和存款数额不是一个概念一样。对一个内存单元来说，单元的地址即为指针，其中存放的数据是该单元的内容。C语言中，允许用一个变量来存放地址(指针)，这种变量称为指针变量。因此一个指针变量的值就是某个内存单元的地址，或称为某内存单元的指针。

严格地说，一个指针是一个地址，是一个常量，而一个指针变量是可以被赋予不同的指针值(地址)，是变量。但通常把指针变量简称为“指针”。为了避免混淆，约定“指针”是指地址，是常量；“指针变量”是指取值为地址的变量。定义指针变量的目的是为了通过指针变量去访问内存单元，从而获得内存单元里存放的数据。

5.1.2 指针变量

在C语言中，一种数据类型或数据结构往往都占有一组连续的内存单元。用“地址”这个概念并不能很好地描述一种数据类型或数据结构，只是所占用存储单元的起始地址在哪，而“指针”虽然实际上也是一个地址，但它是“指向”一个数据结构的，它不仅可以表达出起始地址在哪，而且还能反映出这种数据结构的存储空间。因而概念更为清楚，表示更为明确。这也是引入“指针”概念的一个重要原因。例如，数组或函数都是连续存放的，如果在一个指针变量中存放数组或一个函数的首地址，则通过这个指针变量不仅能够找到该数组或函数的入口，还能反映出这个入口地址是什么数据结构的地址。

CPU访问内存时需要的是地址，而不是变量名和函数名。变量名和函数名只是地址的一种助记符，当源文件被编译和连接成可执行程序之后，它们都会被替换成地址。编译和连接过程的一项重要任务就是找到这些名称所对应的地址。变量名在声明时会有一个自己独特的地址，而程序在编译时也会把声明的变量名转换为指针，CPU访问内存时需要的就是转换之后的地址指针。编译就是负责这个转换的过程。变量名和函数名为我们提供了方便，让我们在编写代码的过程中，可以使用易于阅读和理解的英文字符串，不用直接面对二进制地址。需要注意的是，虽然变量名、函数名、字符串名、数组名等在本质上是一样的，它们都是地址的助记符，但在编写代码的过程中，我们认为变量名表示的是数据本身，而函数名和数组名表示的是代码块或数据块的首地址。原因是因为变量名指向的是一个某种数据类型的内存空间地址，而函数名、字符串名、数组名，指向的是一块内存中连续的数据类型空间的第一个字节的地址。

5.2 指 针 变 量

5.2.1 指针变量的定义

指针变量首先是一个变量，在使用之前必须先定义。定义时不仅要标识出该变量是指针类型的变量，而且还要标识出是可以指向什么类型数据的指针变量。

一般形式：

```
类型说明符   * 指针变量名;
```

其中，*表示定义的是一个指针变量，类型说明符指出该指针变量能够指向什么类型的数据，即该指针变量可以赋予什么类型数据的地址。定义指针变量时，类型标识符一旦确定就不能改变，如果定义了一个指向整型变量的指针变量，那么它就不能再指向其他类型的变量了。也就是说，一个指针变量只能指向同一种类型的变量。

例如：

```
int * p1;              /* 定义一个能够指向整型变量的指针变量 */
```

```
float * p2;          /* 定义一个能够指向单精度变量的指针变量 */
char * p3;           /* 定义一个能够指向字符变量的指针变量 */
```

int *表示整型指针类型,char *表示字符指针类型。数据类型和*不能分开,如果分开则不能表示是指针类型。

指针可以定义指向各种数据类型或结构的指针变量,包括基本数据类型、数组、函数等,甚至还可以指向指针类型(另一个指针变量)。

1) &:取地址运算符

例如:

```
int a;
int * p=&a;          /* 定义一个能够指向整型变量的指针变量,并将指向整型变量 a */
```

2) *:取内容运算符

单目运算符,其结合性为自右至左,用来表示指针变量所指向变量的值(内容)。

例如:

```
int a=3;
int * p;
p=&a;                    /* 整型指针变量 p 指向整型变量 a */
printf("%d,%d",a,*p);    /*  *p 表示取得 p 所指向变量的值 */
*p=5;                    /* 将指针变量 p 指向的存储单元内容赋值为 5,即 a=5 */
```

【例 5-1】 使用交换指针的方式,将两个整数按由大到小的顺序输出。

程序代码如下:

```
/* e5_1.c */
#include<stdio.h>
void main(){
    int * p1,* p2,* p,a,b;
    scanf("%d,%d",&a,&b);
    p1=&a;p2=&b;                /* p1 指向 a,p2 指向 b */
    if(a<b){
        p=p1;
        p1=p2;                  /* 通过指针变量 p,交换 p1 和 p2 的指向 */
        p2=p;
    }
    printf("a=%d,b=%d\n",a,b);
    printf("max=%d,min=%d\n",*p1,*p2);     /* 输出 *p1 和 *p2 的值 */
}
```

程序运行结果:(输入 5,9)

```
a=5,b=9
max=9,min=5
```

交换前后的情况如图 5.1 所示。

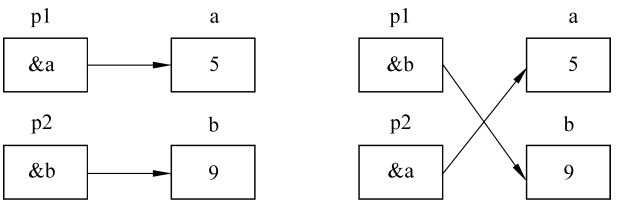

图 5.1　例 5-1 交换前后的情况

程序说明：

(1) a 和 b 并未交换，它们的值保持不变，但是 p1 和 p2 的值改变了。 * p1 和 * p2 是取出指针 p1 和 p2 所指向的内存空间里的值。

(2) 如要得到 a 和 b 的存储单元地址，可用语句“printf("%#0X,%#0X",&a,&b);”输出。

若例 5-1 改写成下列代码：

```
#include <stdio.h>
void main(){
    int * p1, * p2,t,a,b;
    scanf("%d,%d",&a,&b);
    p1=&a;p2=&b;
    if(a<b){
        t= * p1;
        * p1= * p2;          /* 通过临时整型变量 t,交换 * p1 和 * p2 的值 */
        * p2=t;
    }
    printf("a=%d,b=%d\n",a,b);   /* 输出 a 和 b 的值 */
    printf("max=%d,min=%d\n", * p1, * p2);   /* 输出 * p1 和 * p2 的值 */
}
```

程序运行结果：(输入 5,9)

```
a=9,b=5
max=9,min=5
```

程序说明：

修改前的程序，通过 if 语句，确保 p1 指向 a、b 中的大者，如输入 5 和 9，p1 原指向 a，则改成指向 b，而 a、b 值没有变；修改后的程序，通过 if 语句，确保 p1 指向的单元(即 a)的数值大，如输入 5 和 9，则 p1 指向的单元值为 9，p2 指向的单元值为 5，p1、p2 的指向没有改变，但内部存储的值改变了(也就是通过指针变量 p1、p2 进行了变量 a、b 值的交换)。

不允许直接把一个数值赋予指针变量，下面的赋值是错误的：

```
int * p;
p=2000;                          /* 将一个十进制数赋予指针变量 p,类型不匹配 */
```

可以在定义一个指针变量的同时进行初始化。如：

```
int a=25;
int * p=&a;  /* 正确。定义整型指针变量 p 的同时,将 a 的地址赋给 p,即使 p 指向变量 a */
```

不能写成下列语句：

```
int a=25, * p;
* p=&a;                    /* 错误。左边是值,右边是地址,类型不一致。应改为 p=&a; */
```

未经赋值的指针变量不能使用,否则将造成未知的错误,给系统正常运行带来隐患。如下面的使用是错误的：

```
int * p;
* p=5;
```

由于指针变量 p 在定义后指向的位置不确定,因此直接对其赋值是危险的。

5.2.2 多级指针

如果在一个指针变量中存放一个目标变量的地址,是“单级间址”;如果在一个指针变量中存放另一个指针变量,则为指向指针的指针,是“二级间址”。理论上说,间址方法可以延伸到更多的级,但实际上在程序中很少有超过二级间址的情况。级数越多,越难理解,程序产生混乱和错误的机会也会增多。

指向指针的指针即指向指针变量的指针变量。例如,指针变量 q 指向指针变量 p,而 p 又指向另一个数据变量 i,则变量 q 就是指向指针的指针,如图 5.2 所示。

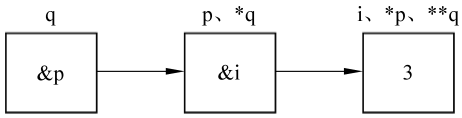

图 5.2　指向指针的指针

定义的一般形式：

```
类型说明符 **变量名
```

例如：

```
int **q;
```

定义了一个指针变量 q,它指向另一个整型指针变量。变量 q 前面的 * 表示 q 是一个指针变量,那 q 是一个能够存放什么类型的指针变量呢？是一个能够存放(int *)类型的,而(int *)类型是整型指针类型。因此,q 是一个能够指向整型指针变量的指针变量。

使用时,由于 * 是按自右至左结合,因此**q 相当于 *(*q)。如图 5.2 所示,q 是一个指向整型指针变量 p 的指针变量,*q 就找到了 p,**q 就找到了 i。

【例 5-2】 多级指针的应用。

程序代码如下：

```
/* e5_2.c */
```

```
#include<stdio.h>
void main(){
    int **p1, * p2,n;
    n=3;
    p1=&p2;
    p2=&n;
    printf("%d,%d,%d\n",n, * p2,**p1);
    *p2=5;
    printf("%d,%d,%d\n",n, * p2,**p1);
    **p1=7;
    printf("%d,%d,%d\n",n, * p2,**p1);
}
```

程序运行结果：

```
3,3,3
5,5,5
7,7,7
```

程序说明：

p2 是指向变量 n，p1 是指向指针变量 p2，因此 * p2 就是 p1，而**p1 就是 n。

5.2.3 指向 void 类型的指针

可以定义一个指针变量，但不指定指向哪一种类型数据。void 的字面意思是“无类型”，void * 则为“无类型指针”，void * 可以指向任何类型的数据。

假设有指针变量 p1 和 p2，如果指针 p1 和 p2 的类型相同，那么可以直接在 p1 和 p2 间互相赋值；如果 p1 和 p2 指向不同的数据类型，则必须使用强制类型转换运算符把赋值运算符右边的指针类型转换为左边指针的类型。例如：

```
float * p1;
int * p2;
p1=p2;                                  /* 编译出错,应改成 p1=(float *)p2; */
```

无类型指针不同，任何类型的指针都可以直接赋值给它，无须进行强制类型转换。如：

```
void * p1;
int * p2;
p1=p2;                                  /* 正确 */
```

但这并不意味着，void * 也可以无须强制类型转换地赋给其他类型的指针。因为“无类型”可以包容“有类型”，而“有类型”则不能包容“无类型”。下面的语句编译错误：

```
void * p1;
int * p2;
```

```
p2=p1;
```

必须改为

```
p2=(int *)p1;
```

5.3 指针变量作为函数参数

函数参数可以是整型、实型、字符型等基本数据类型，也可以是指针类型。使用指针变量作为函数参数，实际上向函数传递的是变量的地址。可以将被调用函数外部的地址传递到函数内部，使得在函数内部可以操作函数外部的数据，并且这些数据不会随着函数的调用结束而被销毁。像数组、字符串、动态分配的内存等都是一系列数据的集合，没有办法通过一个参数全部传入函数内部，只能传递它们的指针，在函数内部通过指针来影响这些数据集合。

【例 5-3】 将两个整数按由大到小的顺序输出。用指针变量作为函数参数实现。

程序代码如下：

```
/* e5_3.c */
#include<stdio.h>
void swap(int * p1,int * p2){
    int * p;
    p=p1;                          /* 通过指针变量 p 交换形参 p1 和 p2 的指向 */
    p1=p2;
    p2=p;
}
void main(){
    int a,b;
    int * pointer1, * pointer2;
    printf("input a,b: ");
    scanf("%d,%d",&a,&b);
    pointer1=&a;
    pointer2=&b;
    if(a<b)
        swap(pointer1,pointer2);
    printf("%d,%d\n",a,b);
    }
```

程序运行结果：

```
input a,b: 5,9
5,9
```

程序说明：很显然，程序的目标未能实现。

(1) 程序运行时，将 a 和 b 的地址分别赋给指针变量

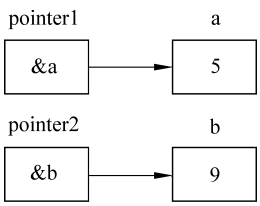

图 5.3 调用函数前的情况

pointer1 和 pointer2，使 pointer1 指向 a，pointer2 指向 b，如图 5.3 所示。

(2) 由于 a<b，因此调用 swap()，实参是两个指针变量，因此在定义 swap()时形参为两个指针变量 p1 和 p2，用于接收实参。发生调用后，pointer1 的值传给 p1，pointer2 的值传给 p2。此时 p1 和 pointer1 都指向变量 a，p2 和 pointer2 都指向变量 b，如图 5.4 所示。

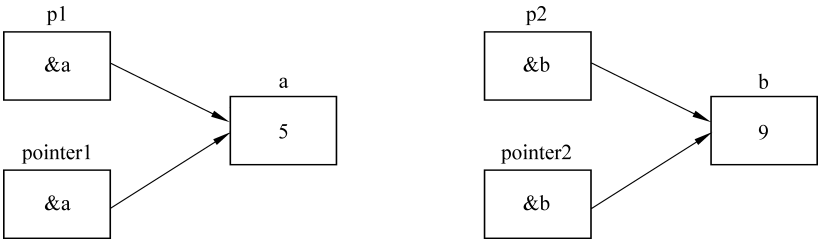

图 5.4　调用函数并进行参数传递

(3) 执行 swap()过程中，通过指针变量 p 使 p1 和 p2 的值互换(即指向互换)。此时，p2 指向变量 a，p1 指向变量 b，如图 5.5 所示。

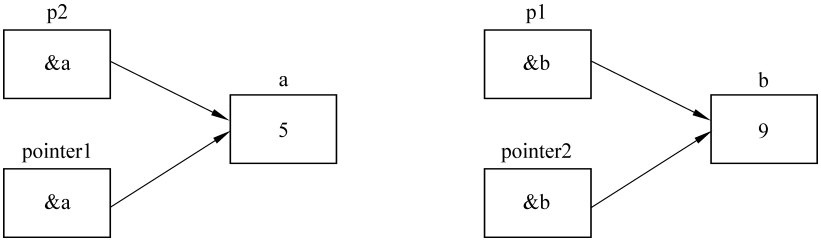

图 5.5　执行 swap()过程中指针发生变化

(4) swap()调用结束后，p1 和 p2 不复存在(已释放)。可以看出，程序运行过程中，仅交换 swap()的形参 p1 和 p2 的值(即交换了指向)，而实参 pointer1、pointer2 的指向及所指对象(a、b)的值始终没有改变，因此，未能实现程序目标。

那么，如何才能实现程序目标呢？可以用下面的代码实现。

```
#include<stdio.h>
void swap(int * p1,int * p2){
    int p;
    p= * p1;
    * p1= * p2;          /* 通过整型变量 p 交换形参 p1 和 p2 所指变量的值 */
    * p2=p;
}
void main(){
    int a,b;
    int * pointer1,* pointer2;
    printf("input a,b: ");
    scanf("%d,%d",&a,&b);
```

```
    pointer1=&a;pointer2=&b;
    if(a<b)
        swap(pointer1,pointer2);
    printf("%d,%d\n",a,b);
}
```

程序运行结果：

```
input a,b: 5,9
9,5
```

程序说明：很显然，程序的目标实现。

由于 a＜b，因此调用 swap()。执行 swap()过程中，通过整型变量 p 使 p1 和 p2 所指变量的值互换。与 p1 和 p2 指向互换不同，* p1 和 * p2 互换意味着 p1 和 p2 指向的存储单元的值进行了互换，即 a＝9，b＝5，p1 和 p2 的指向并没有改变，还是 p1 指向 a，p2 指向 b，如图 5.6 所示。

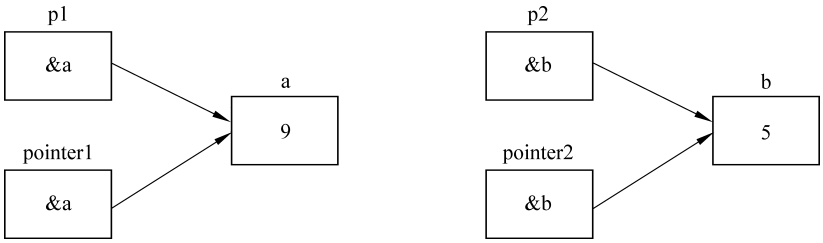

图 5.6　例 5-3 交换前后的情况

【例 5-4】 一个自然数是素数，且它的数字位置经过任意对换后仍为素数，则称为绝对素数，如 13 和 31 都是素数，所以 13 和 31 就是绝对素数。试求所有两位绝对素数。

程序代码如下：

```
/* e5_4.c */
#include<stdio.h>
void main(){
    int m,m1,flag1,flag2;
    void prime(int n,int * f);      /* 函数的声明 */
    for(m=10;m<100;m++){
        m1=(m%10)*10+m/10;          /* m1 为 m 数字位置对换后的数 */
        prime(m,&flag1);            /* 判断 m 是否为素数 */
        prime(m1,&flag2);           /* 判断 m1 是否为素数 */
        if(flag1&&flag2)            /* 只有 m 和 m1 同时为素数才输出 */
            printf("%5d",m);
    }
}
void prime(int n,int * f){
    int k;
```

```
    *f=1;                          /* 将*f预先初始化为1 */
    for(k=2;k<=n/2;k++)
        if(!(n%k)) *f=0;           /* 如果n不是素数,则置*f为0 */
}
```

程序运行结果：

```
11    13    17    31    37    71    73    79    97
```

程序说明：

(1) 函数 prime()用于判断 n 是否为素数，若 *f 最后取值为 1 则表示 n 是素数，*f 最后取值为 0 表示 n 不是素数。

(2) 形参 f 指向实参 flag1 和 flag2，在被调函数 prime()中改变 f 所指变量的值，也就是改变主调函数中 flag1 和 flag2 的值。

5.4 指针函数与函数指针

5.4.1 指针函数

学习函数时介绍过，所谓函数类型是指函数返回值的类型。在 C 语言中允许一个函数的返回值是一个指针(即地址)，这种返回指针值的函数称为指针函数。指针函数，首先是一个函数，指针是说明函数的返回值类型，即一个返回值为指针的函数。

一般定义形式：

```
类型说明符 * 函数名(形参表){
    函数体
}
```

其中，“类型说明符 *”是函数返回值的类型，很显然是一个指针类型。

函数返回值必须用同类型的指针变量来接收。也就是说，指针函数一定有函数返回值，返回一个地址给主调函数，因此，在主调函数中，函数返回值必须赋给同类型的指针变量。

【例 5-5】 指针函数的使用。

```
/* e5_5.c */
#include<stdio.h>
int * add(int a, int b, int * pc){          /定义一个指针函数*/
    *pc=a+b;
    return pc;
}
int main(){
    int a, b;
    a=1;
```