手把手教你[illegible]针(十四)

◆吕超亚

指针

指针是C语言中的一个重要概念，指针类型数据在C语言程序中的使用十分普遍。C语言区别于其它程序设计语言的主要特点就是处理指针时所表现出的能力和灵活性。

正确地使用指针类型数据，可以有效地表示复杂的数据结构，直接处理内存地址，而且可以更为有效合理地使用数组。

一、指针与地址

计算机程序的指令、常量和变量等都要存放在以字节为单位的内存单元中，内存的每个字节都具有一个唯一的编号，这个编号就是存储单元的地址。

各个存储单元中所存放的数据，称为该单元的内容。计算机在执行任何一个程序时都要涉及到许多的单元访问，就是按照内存单元的地址来访问该单元中的内容，即按地址来读或写该单元中的数据。由于通过地址可以找到所需要的单元，因此这种访问是“直接访问”方式。

另外一种访问是“间接访问”，它首先将欲访问单元的地址存放在另一个单元中，访问时，先找到存放地址的单元，从中取出地址，然后才能找到需访问的单元，再读或写该单元的数据。在这种访问方式中使用了指针。

C语言中引入了指针类型的数据，指针类型数据是专门用来确定其他类型数据地址的，因此一个变量的地址就称为该变量的指针。例如，有一个整型变量i存放在内存单元60H中，则该内存单元地址60H就是变量i的指针。

如果有一个变量专门用来存放另一个变量的地址，则该变量称之为指向变量的指针变量（简称指针变量）。例如，如果用另一个变量pi存放整型变量i的地址60H，则pi即为一个指针变量。

二、指针变量的定义

指针变量与其它变量一样，必须先定义，后使用。

指针变量定义的一般形式：

数据类型[存储器类型]指针变量名；

其中，“指针变量名”是我们定义的指针变量名字。“数据类型”说明了该指针变量所指向的变量的类型。“存储器类型”是可选项，它是C5l编译器的一种扩展，如果带有此选项，指针被定义为基于存储器的指针，无此选项时，被定义为一般指针。这两种指针的区别在于它们的存储字节不同。一般指针在内存中占用3个字节，而基于存储器的指针，则指针的长度可为1个字节（存储器类型选项为idata、data、pdata）或2个字节（存储器类型选项为code、xdata）。

例如：

```
int *pt;
```

定义一个指向对象类型为int的一般指针，指针自身在默认的存储区（由编译模式决定），指针长度为3个字节。

```
char xdata *pa;
```

在xata存储器中定义一个指向的对象类型为char的基于存储器的指针。指针自身在默认的存储器区域（由编译器决定），长度为2字节。

```
float xdata * data pb;
```

在xata存储器中定义一个指向的对象类型为float的基于存储器的指针。指针自身在data区，长度为2字节。

特别要注意，变量的指针和指针变量是两个不同的概念。变量的指针就是该变量的地址，而一个指针变量里面存放的内容是另一个变量在内存中的地址，拥有这个地址的变量则称为该指针变量所指向的变量。每一个变量都有它自己的指针(即地址)，而每一个指针变量都是指向另一个变量的。为了表示指针变量和它所指向的变量之间的关系，C语言中用符号“*”来表示“指向”。例如，整型变量i的地址60H存放在指针变量pi中，则可用*pi来表示指针变量pi所指向的变量，即*pi也表示变量i。

三、指针变量的引用

指针变量是含有一个数据对象地址的特殊变量，指针变量中只能存放地址。在实际的编程和运算过程中，变量的地址和指针变量的地址是不可见的。因此，C语言提供了一个取地址运算符&，使用取地址运算符“&”和赋值运算符“=”就可以使一个指针变量指向一个变量。

例如：

```
int t;
int *pt;
pt=&t;
```

通过取地址运算和赋值运算后，指针变量pt就指向了变量t。

当完成了变量、指针变量的定义以及指针变量的引用后，我们就可以对内存单元进行间接访问了。此时，我们需用到指针运算符（又称间接运算符）“*”。例如，我们需将变量t的值赋给变量x。

```
int x;
int t;
直接访问方式为：x=t;
间接访问方式为：int x;
int t;
int *pt;
pt=&t;
x=*pt;
```

有关的运算符有两个，它们是“&”

和"*"。在不同的场合所代表的含义是不同的,我们一定要搞清楚。

例如: int *pt; 进行指针变量的定义,此时 *pt 的 * 为指针变量说明符。

pt=&t; 此时 &t 的 & 为取 t 的地址并赋给 pt(取地址)。

x=*pt; 此时 *pt 的 * 为指针运算符,即将指针变量 pt 所指向的变量值赋给 x(取内容)。

四、数组指针与指向数组的指针变量

任何变量都占有存储单元,都有地址。数组及其元素同样占有存储单元,都有相应的地址。因此,指针既然可以指向变量,当然也可以指向数组。其中,指向数组的指针是数组的首地址,指向数组元素的指针则是数组元素的地址。

例如:

我们定义一个数组 x[l0]和一个指向数组的指针变量 px;

```
int x[l0];
int *px;
```

当我们未对指针变量 px 进行引用时,px 与 x[l0]毫不相干,即此时指针变量 px 并未

指向数组 a[10]。

当我们将数组的第一个元素的地址 &x[0]赋予 px 时,Px=&x[0];指针变量 px 即指向数组 x[]。这时,可以通过指针变量 px 来操作数组 x 了,即 *px 代表 x[0],*(px+1)代表 x[1],……*(px+i)代表 x[i]。i=1、2、……。

C 语言规定,数组名代表数组的首地址,也是第一个数组元素的地址,因此上面的语句也可改写为:

```
int x[l0];
 int *px;
px=x;
```

形式上更简单一些。

五、指针变量的运算

若先使指针变量 px 指向数组 x[](即 px=x;),则:

1.px++(或 px+=l);将使指针变量 px 指向下一个数组元素,即 x[l]。

2.*px++;因为 ++ 与 * 运算符优先级相同,而结合方向为自右向左,因此,*px++ 等价于 *(px++)。

3.*++px; 先使 px 自加 1,再取 *px 值。

若 p 的初值为乙扎 0],则

执行 x 二,p 干十时,x 值为 a[O]的值,而执行 x 二。十十 p 后,则 x 值等于 a[l]的值。

4.(*px)++; 表示 px 所指向的元素值加 l。要注意的是元素值加 1 而不是指针变量值加 l。要特别注意对 px+i 的含义的理解。C 语言规定:px+l 指向数组首地址的下一个元素,而不是将指针变量 px 的值简单地加 1,例如:若数组的类型是整型(int),每个数组元素占 2 个字节,则对于整型指针变量 px 来说,px+l 意味着使 px 的原值 (地址)加 2 个字节,使它指向下一个元素。

六、指向多维数组的指针和指针变量

指针除了可以指向一维数组外,也可以指向多维数组。下面以二维数组为例进行说明。

假定我们已定义了一个三行四列的二维数组:

```
int x[3][4]={ {1,3,5,7},
{9,11,13,15},
{17,19,21,23}};
```

对这个数组的理解为:x 是数组名,数组包含 3 个元素:x [0]、x[1]、x[2]。

每个元素又是一个一维数组,包含 4 个元素。如 x[0]代表的一维数组包含 x[0][0]={1}, x[0][1]={3}, x[0][2]={5}, x[0][3]={7}。

从二维数组的地址角度看,x 代表整个数组的首地址,也就是第 0 行的首地址。x+1 代表第 1 行的首地址,即数组名为 x[1]的一维数组首地址。

……

根据 C 语言的规定,由于 x[0]、x[1]、x[2]都是一维数组,因此它们分别代表了各个数组的首地址。即 x[0]=&x[0][0],x[1]=&x[1][0],x[2]=&x[2][0]。

我们同时定义一个指针变量 int (*p)[4];其含义是 P 指向一个包含 4 个元素的一维数组。

当 p=x 时,指向数组 x[3][4]的第 0 行首址。

P+1 和 x+1 等价,指向数组 x[3][4]的第 1 行首址。

P+2 和 x+2 等价,指向数组 x[3][4]的第 2 行首址。

*(P+1)+3 和 &x[1][3]等价,指向数组 x[1][3]的地址。

((P+1)+3)和 x[1][3]等价,表示 x[1][3]的值。

……

一般地,对于数组元素 x[i][j]来讲:

*(p+i)+j 就相当于 &x[i][j],表示数组第 i 行第 j 列的元素的地址。

((p+i)+j)就相当于 x[i][j],表示数组第 i 行第 j 列的元素的值。

下面我们做实验,感性认识一下指针。

实验一

在 LED/16*2 字符液晶试验板上做实验,分别采用直接引用变量和间接引用变量的方法,将变量值显示在数码管上。

在我的文档中建立一个文件目录(cs36),然后建立 cs36.uv2 的工程项目,最后建立源程序文件(cs36.c)。

输入下面的程序:

```
#include <REG51.H>// 序号(以下同):1
/**************2************************/
void main(void)                    //3
{                                  //4
 char a,b;                         //5
 char *p;                          //6
 p=&b;                             //7
 a=0x80;                           //8
 *p=0x80;                          //9
 P0=b;                             //10
 P3=a;                             //11
 while(1);                         //12
}                                  //13
```

编译通过后,将生成的 cs36.hex 文件烧录到 89S51 芯片中,将芯片插入到 LED/16*2 字符液晶试验板上,试

验板上接通 9V 电源，可以看到，个位数码管和千位数码管均显示“8”。

我们对程序进行分析解释。

序号 1（程序解释，以下同）：包含头文件 REG51.H。

序号 2：程序分隔。

序号 3：定义函数名为 main 的主函数。

序号 4：main 主函数开始。

序号 5：定义字符型局部变量。

序号 6：定义指向字符型数据类型的指针变量 p。

序号 7：将变量 b 的地址赋给指针变量 p，即指针指向变量 b。

序号 8：采用直接引用变量的方法赋值 0x80 给变量 a。

序号 9：采用间接引用变量的方法赋值 0x80 给指针变量 p 指向的变量（即变量 b）。

序号 10：P0 口显示变量 b。

序号 11：P3 口显示变量 a。

序号 12：动态停机。

序号 13：main 主函数结束。

实验二

在 LED/16*2 字符液晶试验板上，分别用下标法和指针法引用数组元素并在数码管上显示。

在我的文档中建立一个文件目录（cs37），然后建立 cs37.uv2 的工程项目，最后建立源程序文件（cs37.c）。

输入下面的程序：

```
#include <REG51.H>// 序号（以下同）：1
#define uchar unsigned char          //2
#define uint unsigned int            //3
/*************************4**********/
void delay(uint k)                   //5
{                                    //6
uint i,j;                            //7
for(i=0;i<k;i++){                    //8
for(j=0;j<121;j++)                   //9
{;}}                                 //10
}                                    //11
/************************12***********/
void main(void)                      //13
{                                    //14
uchar *pt,i;                         //15
uchar code SEG7[10]={0xc0,0xf9,0xa4,
0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};//16
pt=SEG7;                             //17
    for(i=0;i<10;i++)                //18
    {P0=SEG7[i];                     //19
    delay(500);}                     //20
//=======================21
    P0=0xff;                         //22
    delay(2000);                     //23
//=======================24
    for(i=0;i<10;i++)                //25
    {P0=*(pt+i);                     //26
    delay(500);}                     //27
    P0=0xff;                         //28
//=======================29
while(1);                            //30
}                                    //31
```

编译通过后，将生成的 cs37.hex 文件烧录到 89S51 芯片中，将芯片插入到 LED/16*2 字符液晶试验板上，试验板上接通 9V 电源，可以看到，个位数码管上依次显示“0”…“9”。稍停片刻后，又显示一遍“0”…“9”。第一遍显示时是采用下标法引用数组元素的，而第二遍显示则是是采用指针法引用数组元素。

对程序进行分析解释。

序号 1（程序解释，以下同）：包含头文件 REG51.H。

序号 2、3：数据类型的宏定义。

序号 4：程序分隔。

序号 5~11：延时子函数。

序号 12：程序分隔。

序号 13：定义函数名为 main 的主函数。

序号 14：main 主函数开始。

序号 15：定义指向字符型数据类型的指针变量 pt 及变量 i。

序号 16：数码管 0~9 的字形码。

序号 17：指针变量指向数组 SEG7。

序号 18：for 循环语句。

序号 19：采用下标法引用数组元素并显示于个位数码管上。

序号 20：延时 500ms，便于人们观察。

序号 21：程序分隔。

序号 22：熄灭个位数码管。

序号 23：稍停片刻。

序号 24：程序分隔。

序号 25：for 循环语句。

序号 26：采用指针法引用数组元素并显示于个位数码管上。

序号 27：延时 500ms，便于人们观察。

序号 28：熄灭个位数码管。

序号 29：程序分隔。

序号 30：动态停机。

序号 31：main 主函数结束。

实验三

在《手把手教你学单片机的 C 语言程序设计（十一）》的实验一中，我们已经在 LED/16*2 字符液晶试验板上，实现了参数传递的函数调用。即开机后数码管的低 2 位显示 3 和 8。3 号键按下后（即 P3.0 为低）调用交换子函数 swap，使得数码管的低 2 位交换显示为 8 和 3。

现在，我们要在 LED/16*2 字符液晶试验板上进行传址调用的实验，使指针变量成为函数的参数，将一个变量的地址传送到另一个函数中，运行的结果同上面完全一样。

在我的文档中建立一个文件目录（cs38），然后建立 cs38.uv2 的工程项目，最后建立源程序文件（cs38.c）。

输入下面的程序：

```
#include <REG51.H>// 序号（以下同）：1
#define uchar unsigned char          //2
uchar code SEG7[10]={0xc0,0xf9,0xa4,
0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};//3
sbit P3_0=P3^0;                      //4
//=======================5
void swap(uchar *x,uchar *y);        //6
//=======================7
void main(void)                      //8
{                                    //9
uchar a=3,b=8;                       //10
uchar *pt1,*pt2;                     //11
pt1=&a;                              //12
pt2=&b;                              //13
P1=SEG7[a];                          //14
P0=SEG7[b];                          //15
    while(1)                         //16
    {                                //17
    P3=0x0f;                         //18
        if(!P3_0)                    //19
        {swap(pt1,pt2);              //20
        P1=SEG7[a];                  //21
        P0=SEG7[b];                  //22
        while(1);}                   //23
    }                                //24
}                                    //25
//=======================26
void swap(uchar *x,uchar *y)         //27
{                                    //28
uchar t;                             //29
t=*x;                                //30
```

```
*x=*y;                          //31
*y=t;                           //32
}                               //33
```

编译通过后，将生成的 cs38.hex 文件烧录到 89S51 芯片中，将芯片插入到 LED/16*2 字符液晶试验板上，试验板上接通 9V 电源，右边 2 个 LED 数码管显示“38”。按下 3 号键右边 2 个 LED 数码管显示“83”。显然，结果完全相同。

我们分析程序。

序号 1（程序解释，以下同）：包含头文件 REG51.H。
序号 2：数据类型的宏定义。
序号 3：数码管 0~9 的字形码。
序号 4：定义 P3.0。
序号 5：程序分隔。
序号 6：子函数 swap 声明。
序号 7：程序分隔。
序号 8：定义函数名为 main 的主函数。
序号 9：main 主函数开始。
序号 10：定义无符号字符型局部变量 a、b 并赋初值。
序号 11：定义指向字符型数据类型的指针变量 pt1 及 pt2。
序号 12：pt1 指向变量 a。
序号 13：pt2 指向变量 b。
序号 14：变量 a 的值送 P1 口显示。
序号 15：变量 b 的值送 P0 口显示。
序号 16：无限循环。
序号 17：无限循环语句开始。
序号 18：P3 口置 0x0f，以便读取 P3.0 的输入状态。
序号 19：如果 P3.0 为低电平。
序号 20：调用 swap 子函数，传递的是变量 a、b 的地址。
序号 21：此时变量 a 的值送 P1 口显示。
序号 22：此时变量 b 的值送 P0 口显示。
序号 23：动态停机。
序号 24：无限循环语句结束。
序号 25：main 主函数结束。
序号 26：程序分隔。
序号 27：定义函数名为 swap 的子函数。
序号 28：swap 子函数开始。
序号 29：定义无符号字符型局部变量 t。
序号 30：x 指向的指针变量（即变量 a）值送 t。
序号 31：y 指向的指针变量（即变量 b）值送 x 指向的指针变量（即变量 a）。
序号 32：再将变量 t 赋给 y 指向的指针变量（即变量 b）。这样实现了变量 a、b 的交换。
序号 33：swap 子函数结束。

实验四

在《手把手教你学单片机的 C 语言程序设计（十三）》中，我们曾经介绍过，除了可以用变量作为函数的参数之外，还可以用数组名作为函数的参数。这里我们做一个有关的实验，求出一个一维数组 a[11]中的前 10 个数之和，将其存放在 a[10]中，并在 LED/16*2 字符液晶试验板的数码管上显示出来。在我的文档中建立一个文件目录（cs39），然后建立 cs39.uv2 的工程项目，最后建立源程序文件（cs39.c）。

输入下面的程序：

```
#include <REG51.H>// 序号（以下同）：1
#define uchar unsigned char        //2
#define uint unsigned int          3
uchar code SEG7[10]={0xc0,0xf9,0xa4,
0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};//4
/*********************************5*********/
void sum(uint *q,uint n)           //6
{                                  //7
uint i,s;                          //8
uint *t;                           //9
t=q;                               //10
for(i=0;i<n;i++)s=s+*(t+i);        //11
t=q+10;                            //12
*t=s;                              //13
}                                  //14
/***********************15***********/
void main(void)                    //16
{                                  //17
uint a[11]={0,1,2,3,4,5,6,7,8,9,0}; //18
uint *pt,len=10;                   //19
pt=a;                              //20
sum(a,len);                        //21
P3=SEG7[a[10]/1000];               //22
P2=SEG7[(a[10]/100)%10];           //23
P1=SEG7[(a[10]/10)%10];            //24
P0=SEG7[a[10]%10];                 //25
while(1);                          //26
}                                  //27
```

编译通过后，将生成的 cs39.hex 文件烧录到 89S51 芯片中，将芯片插入到 LED/16*2 字符液晶试验板上，试验板上接通 9V 电源，4 个 LED 数码管显示“0045”。

对程序进行分析。

序号 1（程序解释，以下同）：包含头文件 REG51.H。
序号 2、3：数据类型的宏定义。
序号 4：数码管 0~9 的字形码。
序号 5：程序分隔。
序号 6：定义函数名为 sum 的子函数。
序号 7：sum 子函数开始。
序号 8：定义无符号整型局部变量 i、s。
序号 9：定义指向无符号整型数据类型的指针变量 t。
序号 10：q 传递的数组首地址赋予 t，指针变量 t 指向数组的首单元。
序号 11：for 循环语句，共循环 n 次，将数组 a 的前 n 个元素累加后存入 s。
序号 12：指针变量 t 指向数组 a 的最后一个单元 a[10]。
序号 13：将 s 存入 a[10]。
序号 14：sum 子函数结束。
序号 15：程序分隔。
序号 16：定义函数名为 main 的主函数。
序号 17：main 主函数开始。
序号 18：定义无符号整型数组 a 并赋初值。
序号 19：定义指向无符号整型数据的指针变量 pt 及无符号整型变量 len。
序号 20：指针变量 pt 指向数组 a 的首地址。
序号 21：调用 sum 子函数，求数组 a 的前 10 个数之和并存放在 a[10]中。
序号 22~25：将 a[10]内容显示在数码管上。
序号 26：动态停机。
序号 27：main 主函数结束。