

项目六

用指针优化学生成绩排名



技能目标

具有运用指针优化处理问题的能力。



知识目标

1. 知道指针的概念、指针变量的定义、引用。
2. 会用指针实现数组的输入/输出。
3. 会用指针变量作为函数参数。



项目要求

一个班有 40 名学生参加了期末考试，共考了三门课程，请用指针优化学生成绩排名，即用指针实现 40 名学生三门课的输入/输出以及最高分的输出（在函数中进行）。



项目分析

要用指针优化学生成绩排名，第一必须要了解指针的概念、引用；第二必须会用指针实现数组的输入/输出；第三在函数中用指针实现数组中最高分的挑选，然后调用此函数。为了在介绍的时候条理清晰，所以将这一项目分解成 4 个任务。任务一是了解指针；任务二是用指针优化全班同学一门课成绩的输入/输出；任务三是用指针优化全班同学三门课成绩的输入/输出；任务四是用指针实现输出最高分的记录。

任务一 了解指针

一、问题情境

一个班进行了一次考试，现要将几名学生的成绩输入，用指针的方式输出。

二、具体实现

```
#include<stdio.h>
void main()
{
    int *p1,*p2,a,b;
    printf("输入:");
    scanf("%d,%d",&a,&b);
    p1=&a;p2=&b;
    printf("输出:");
    printf("a=%d,b=%d\n",a,b);
    printf("*p1=%d,*p2=%d\n",*p1,*p2);
}
```

程序运行后显示器上的结果为：

输入:5,6

输出: a=5,b=6

*p1=5,*p2=6

从上述这个例子可分析出要解决这个问题，必须要懂得指针的概念和指针的引用。

三、相关知识

1. 地址和指针的概念

指针是 C 语言中最具特色的内容，也是 C 语言的重要概念和**精华**，我们说 C 语言是既具有低级语言特色又具有高级语言特色的语言，其低级语言特色的主要表现就体现在对地址的直接操作，而对地址的直接操作主要是通过指针来实现，可以这样说，学习 C 语言如果不能正确理解和掌握 C 的指针内容就不算真正掌握 C 语言。

我们知道，如果要住旅馆，办完一定手续后，旅馆会提供住房号，根据住房号可以使用客房。同样计算机内存空间是由顺序排列的以字节为单位的存储单元组成的，将这些存储单元从 0 开始顺序编号，这些编号就构成了每个存储单元的地址，如图 6-1 所示。每个数据都存放于从某个特定的地址开始的一个若干个字节单元中，这个特定的地址就被认定为是该数据的存储地址。计算机对数据的存取都是通过这样的地址才得以实现的。我们知道，数据是分类型的，而不同的类型在内存中所占的空间大小是不同的，以字节为单位，如整型占 2 个字节，单精度浮点型占 4 个字节，现在的问题是每个字节都有地址，那么哪一个地址作为数据的地址呢？C 语言规定地址编号最小的地址作为数据（变量）的地址。例如，int x;两个地址中最小的那一个是变量 x 的地址，如图 6-2 所示；float y;四个地址中最小的那一个是变量 y 的地址，如图 6-3 所示。

前面对数据的存取基本上都是通过变量名进行的，没有直接与地址打交道。但每个变量名都与一个唯一的地址相对应，因此我们对变量的访问实质上还是通过地址来进行数据的存取。由于编译系统所生成的代码能够自动地根据变量名与地址的对应关系完成相应的

地址操作，因而一般情况下我们并不关心一个数据的具体存储地址，也不必为如何进行地址操作而操心。

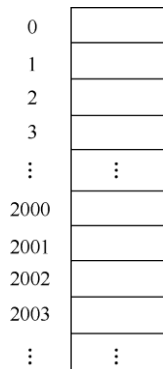


图6-1 内存地址示意图

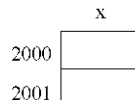


图6-2 整型x的地址是2000

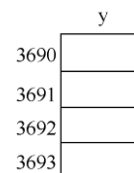


图6-3 单精度浮点型y的地址是3690

但是，在某些类型的应用中，需要先“算出”数据的存储地址，然后再通过该地址间接地访问数据，在这种情况下，地址本身被作为数据处理对象的一部分，成为一种特殊的数据。由于地址指明了数据存储的位置，因此形象地将地址称为指针，该地址处存放的数据也形象地被称为“指针所指向的数据”。

指针与地址虽然有着密切的关系，但它们在概念上是有区别的，指针所标明的地址总是为保存特定的数据类型的数据而准备的，因此指针不但标明了数据的存储位置，而且还标明了该数据的类型，可以说指针是存储特定数据类型的地址。

指针也有类型，指针的类型就是指针所指向的数据的类型。指针的类型限定指针的用途，例如一个 `double` 型指针只能用于指向 `double` 型数据，不限定类型的指针为无类型的指针或者说是 `void` 指针，可用于指向任何类型的数据。

2. 指向变量的指针变量

(1) 指针变量的定义

用来存放数据地址的变量叫指针变量。

定义格式：

类型标识符 *变量名[=地址表达式]

其中，类型标识符是指针变量所指向单元的值的数据类型：“*”是指针变量的定义符；“变量名”命名规则同一般变量，但表示一个地址。

例如：

```
int x,*pointer1; pointer1=&x;
```

`pointer1` 表示 `x` 的内存地址。

(2) 指针变量的初始化

在定义变量的同时给指针变量赋地址值。

例如：

```
int x=3;
```

```
float y;
```

```
int *pointer1=&x;
```

```
float *pointer2=&y;
```

指针的指向过程如图 6-4 所示。

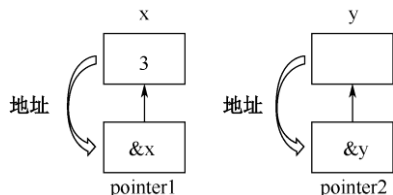


图 6-4 指针的指向过程

`pointer1` 和 `pointer2` 为指针变量，`&x` 和 `&y` 为 `x` 和 `y` 变量的地址。

指针变量使用之前，如果没有给指针变量赋值，即指针变量没有指向一个具体的地址，这样的指针叫空指针，空指针是“危险”的。因此，如果指针变量暂时不指向一个变量地址，请给指针变量赋 `NULL` 值。

(3) 指针变量的引用

①`&`：取地址运算符

用于变量名之前，表示该变量的内存地址。

②`*`：指针运算符（间接访问运算符）

用于指针变量名之前，获取该指针所指向的目标单元的值。

要注意“`*`”号的不同意义，在定义变量时用“`*`”号，表示定义了一个指针变量；在引用时用“`*`”号，表示间接运算。

(4) 指针变量的算术运算

含义：对于地址的运算，只能进行整型数据的加，减运算。

规则：指针变量 `p±n` 表示将指针指向的当前位置向前或向后移动 `n` 个存储单元。

指针变量的算术运算结果是改变指针的指向。

指针变量算术运算的过程：`p=p+n`；`p=p-n`

注意：`p±n` 不是加（减）`n` 个字节，而是加（减）`n` 个数据单元。

【例 6-1】指针与地址的应用

程序如下：

```
#include<stdio.h>
void main()
{
    int a,b,*pointer1,*pointer2;
    a=100,b=200;
    pointer1=&a;
    pointer2=&b;
    printf("%d,%d\n",a,*pointer1);
    printf("%d,%d\n",b,*pointer2);
}
```

程序运行结果：

100,100

200, 200

(5) “&” 和 “*” 两个运算符

“&” 和 “*” 两个运算符的优先级别是相同的，结合规律是右结合性。例如：若 `point1 = &a`，`*point1 = a` 则 `&*point1` 等价于 `&a`、`*&a` 等价于 `a`。

在定义指针变量时，还未规定它指向哪一个变量，此时不能用*运算符访问指针，只有在程序中用赋值语句具体规定后，才能用*运算符访问所指向的变量。

```
int a;  
int *p;           //未规定指向哪个变量  
*p=289;
```

上述表述是错误的，这种错误称为访问悬挂指针。

可以改为：

```
int a;  
int *p=&a;  
*p=289;
```

【例 6-2】 输入两名学生的成绩，按从小到大的顺序输出。

程序如下：

```
#include<stdio.h>  
void main()  
{  
    int *p1,*p2,*p,a,b;  
    printf("输入:");  
    scanf("%d,%d",&a,&b);  
    p1=&a;p2=&b;  
    if(a>b)  
    {  
        p=p1;p1=p2;p2=p;  
    }  
    printf("输出:");  
    printf("a=%d,b=%d\n",a,b);  
    printf("min=%d,max=%d\n",*p1,*p2);  
}
```

程序的运行结果如图 6-5 所示。

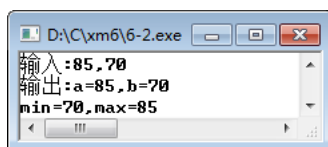


图 6-5 程序的运行结果

程序提示：

`p1=&a; p2=&b;`

使 `p1` 指向 `a` 的地址,`p2` 指向 `b` 的地址

```
if(a>b){p=p1;p1=p2;p2=p;}
```

如果 $a > b$ ，则交换指针，即 $p1$ 指向较小值， $p2$ 指向较大值，如图 6-6 所示。

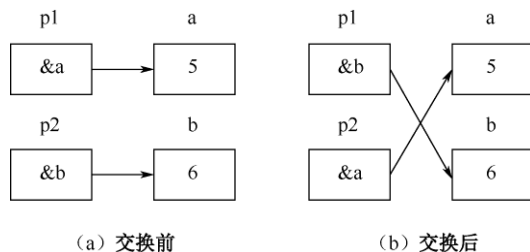


图 6-6 交换指针

3. 指针变量作为函数参数

我们在前面学过，函数可以通过 `return` 返回一个值，如果要函数返回多个值怎么办？显然用 `return` 语句是办不到的。同时“值传递”过程中实参与形参是彼此独立的存储空间，数据的传递本质上是一种数据的复制，如果形参与实参过多，势必造成内存额外开销和引起数据复制量的增大。降低了效率。因此，C 语言采用了一种叫指针传递的方式来改变上述两种不足。

通过“传地址”形参指针成为实参指针的副本，于是通过形参指针也可以访问实参指针所指向的数据，因此指针参数的传递就是把实参指针所指向的数据间接地传递给被调用的函数。

【例 6-3】用指针变量作为函数参数，实现数据的交换。

程序如下：

```
#include<stdio.h>
void swap(int *p1,int *p2)          //用指针变量实现数据的交换
{
    int temp;
    temp=*p1;
    *p1=*p2;
    *p2=temp;
}
void main()
{
    int a,b,*pointer1,*pointer2;
    printf("输入a,b的值:");
    scanf("%d,%d",&a,&b);
    pointer1=&a;pointer2=&b;
    if(a<b)
        swap(pointer1,pointer2);
    printf("调用函数后输出a,b的值为:");
    printf("%d,%d\n",a,b);
}
```

程序运行结果如图 6-7 所示。

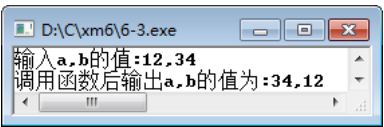


图 6-7 程序运行结果

(1) 指针变量作为参数，从调用函数向被调用函数传递的不是一个变量，而是变量的地址。

(2) 指针变量作为函数的参数，从实参向形参的数据传递仍然遵循“单向值传递”的原则，只不过此时传递的是地址。因此对形参的任何操作都相当于对实参的操作。从这个意义上讲，指针变量作函数参数的传递又具有了“双向性”，可以带回操作后的结果。如图 6-8 所示。

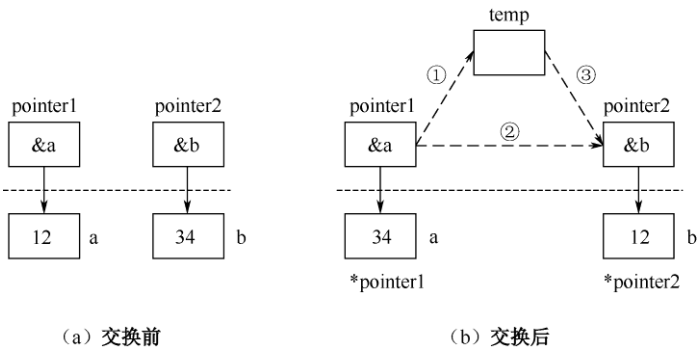


图 6-8 交换示意图

上面例子中的语句 `swap(pointer1,pointer2)`也可以改写为 `swap(&a,&b)`，因为 `pointer1=&a`、`pointer2=b`。

任务二 用指针优化全班同学一门课成绩的输入/输出

一、问题情境

一个班有 40 名同学进行了一次考试，现要用指针实现全班同学成绩的输入/输出。

二、具体实现

以 10 名学生为例，访问数组元素有以下三种方法。

方法一：下标法（直接访问）

```
#include<stdio.h>
```

```

void main()
{
    int score[10],i;
    printf("请输入10名学生的成绩\n");
    for(i=0;i<10;i++)
        scanf("%d",&score[i]);
    printf("输出的10名学生的成绩为\n");
    for(i=0;i<10;i++)
        printf("%3d",score[i]);
    printf("\n");
}

```

方法二：地址法（数名提供数组首地址）

```

#include<stdio.h>
void main()
{
    int score[10],i;
    printf("请输入10名学生的成绩\n");
    for(i=0;i<10;i++)
        scanf("%d",&score[i]);
    printf("请输入10名学生的成绩为\n");
    for(i=0;i<10;i++)
        printf("%3d",*(score+i));
    printf("\n");
}

```

方法三：指针法 1（指针变量指向首个元素）

```

#include<stdio.h>
void main()
{
    int score[10],i,*p;
    p=score;
    printf("请输入10名学生的成绩\n");
    for(i=0;i<10;i++)
        scanf("%d",&score[i]);
    printf("请输入10名学生的成绩为\n");
    for(i=0;i<10;i++)
        printf("%3d",*(p+i));
    printf("\n");
}

```

方法四：指针法 2（指针变量指向任意一个元素）

```

#include<stdio.h>
void main()
{

```



```
int score[10],*p,i;
printf("请输入10名学生的成绩\n");
for(i=0;i<10;i++)
    scanf("%d",&score[i]);
printf("请输入10名学生的成绩为\n");
for(p=score;p<score+10;p++)
    printf("%3d",*p);
printf("\n");
}
```

从上述这个例子可分析出要解决这个问题，必须要懂得指向一维数组元素的指针和一维数组元素的指针访问方式。

三、相关知识

一个变量有地址，一个数组有若干个元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。指针变量既然可以指向变量，当然也可以指向数组和数组元素。

1. 指向数组元素的指针

定义一个指向数组元素的指针变量的方法，与前面介绍的指向变量的指针变量相同。例如：

```
int a[10],*p;
float b[10];
float *p=&b[0];
*p=&a[0];
```

在数组中，数组名表示该数组在内存的起始地址。第一个元素的地址也是数组的起始地址。`&a[0]`表示数组第一个元素的地址，同时 C 语言规定数组名代表数组的首地址，因此，`p=a` 与 `p=&a[0]`等价，都代表数组的首地址，注意不是代表整个数组，如图 6-9 所示。

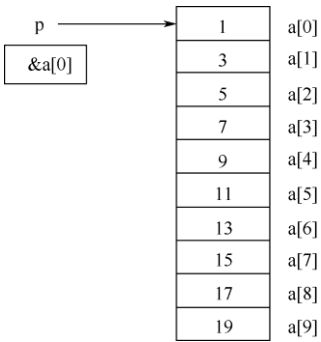


图 6-9 首地址

(1) 计算两地址间数据单元的个数（指针相减）

同类型的两指针相减，其结果是一个整数，表示两地址之间可容纳的相应类型数据的个数。例如：

```
int n,m[13],*p1=&m[5],*p2=&m[10];
```

`n=p2-p1;`
`n=5`, 如图 6-10 所示。

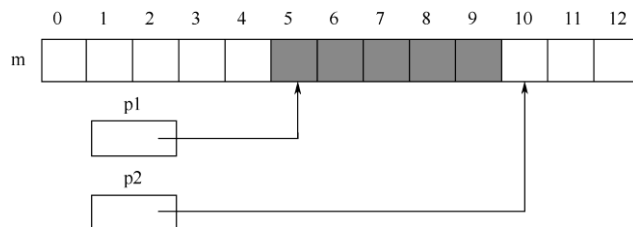


图 6-10 两指针相减

(2) 指针移动

例如，下面的语句进行了指针移动：

```
int m[13],*p1=&m[6],*p2=&m[8],*p3;  
p1=3;           //指针变量 p1 指向数组元素 m[3]  
p3=p2+2;        //指针变量 p3 指向数组元素 m[10]
```

如图 6-11 所示。

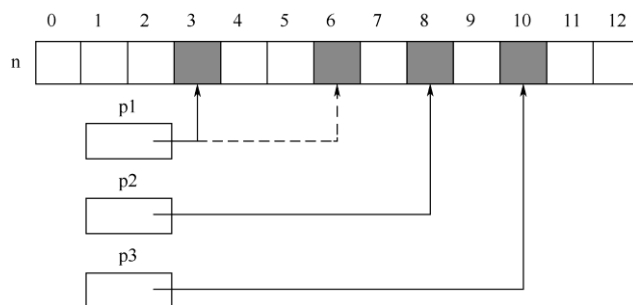


图 6-11 指针移动

思考：若再执行

```
p1++;  
p2--;
```

指针变量 `p1`, `p2` 将分别指向哪个数组元素？

2. 一维数组元素的指针访问方式

一维组的数组名实际上就是指向该数组的第一个单元的指针。一个一维数组若定义为 `int a[10];`

数组名 `a` 的类型是 `int *`（数组名代表数组的首地址，因此是指针类型），并且指向第一个元素。因此 `*a` 和 `a[0]` 访问的是同一个元素，两种表达形式完全等价。这种指针表达形式不仅可以访问第一个元素，结合指针移动还可以访问数组的其他元素。例如：

```
*(a+1)等价于 a[1];  
*(a+2)等价于 a[2];  
.....  
*(a+i)等价于 a[i];
```

因此，访问数组元素的操作可以采用两种方法，一种叫下标法；一种叫指针法。采用指针法比采用下标法更为简洁，执行效率也更高。

指向一维数组第一个元素的指针可以像一维数组名那样使用。例如：

```
int a[10],*p=a;
*(p+1)等价于 a[1];
*(p+2)等价于 a[2];
.....
*(p+i)等价于 a[i];
```

下标法与指针法的对应关系如图 6-12 所示。

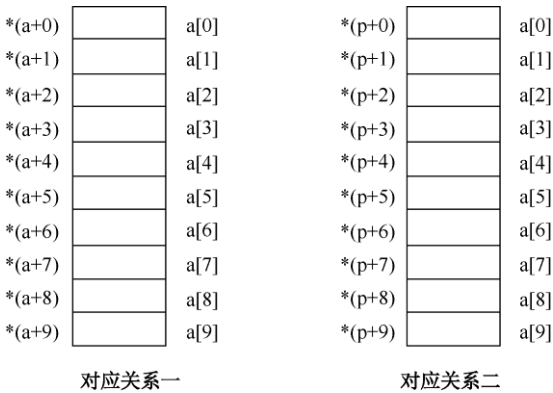


图 6-12 下标法与指针法的对应关系

一维数组的访问方法小结

若有定义：

```
int a[10],*p=a;
数组 a 中任意一个元素 a[i]的地址及值可表示为如表 6-1 所示。
```

表 6-1 一维数组元素的地址及值的表示

	a[i]元素的地址	a[i]元素的值	备 注
下标法	&a[i]	a[i]	
地址法	a+i	*(a+i)	a 为数组名，提供首地址
指针法 1	p+i	*(p+i)	p 指向 a[0]
指针法 2	p	*p	p 指向 a[i]

【例 6-4】数组元素的访问。

程序如下：

```
#include<stdio.h>
void main()
{
    int a[5]={7,9,4,3,8};
    int *p=a;
    //用下标法访问各数组元素
    printf("%d ",a[0]);
```

```

printf("%d\n",a[2]);
//用数组名访问数组元素的地址
printf("%d ",*a);
printf("%d\n",*(a+2));
//用指针变量访问各数组元素
printf("%d ",*p);
printf("%d\n",*(p+2));
}

```

执行结果:

7 4

7 4

7 4

【例 6-5】下面的程序在输入 1 2 3 4 5 6 7 8 9 0 后的输出结果还是 1 2 3 4 5 6 7 8 9 0 吗?

```

#include<stdio.h>
void main()
{
    int a[10],*p,i;
    p=a;
    for(i=0;i<10;i++)
        scanf("%d",p++);           //特别要注意输入时指针的变化
    printf("\n");
    for(i=0;i<10;i++,p++)           //指针p的值已变化
        printf("%d",*p);
}

```

输入: 1 2 3 4 5 6 7 8 9 0

输出: 结果不是预期的值!!

怎样解决? 其实很简单。重新使指针变量指向数组的首地址, 请看改进后的例子。

```

#include<stdio.h>
void main()
{
    int a[10],*p,i;
    p=a;
    for(i=0;i<10;i++)
        scanf("%d",p++);           //特别要注意输入时指针的变化
    printf("\n");
    p=a;                             //指针p的值已变化
    for(i=0;i<10;i++,p++)
        printf("%d ",*p);
}

```

程序运行结果是:

输入: 1 2 3 4 5 6 7 8 9 0

输出: 1 2 3 4 5 6 7 8 9 0

小结：

（1）指针变量可以实现自身值的改变。如：`p++`;而数组名所代表的地址则不能改变，`a++`是错误的用法。

（2）应注意指针变量的当前值，如【例 6-5】所述。

（3）指针变量可以指向数组中各个内存单元。