指针和数组

指针

1、复习：**指针变量的定义、类型和赋值**

2、对指针变量的操作

- 通过指针来引用一个存储单元
- 移动指针
- 指针比较

3、函数之间地址值的传递

- 形参为指针变量时实参和形参之间的数据传递
- 通过传送地址值在被调用函数中直接改变调用函数中的变量的值
- 函数返回地址值

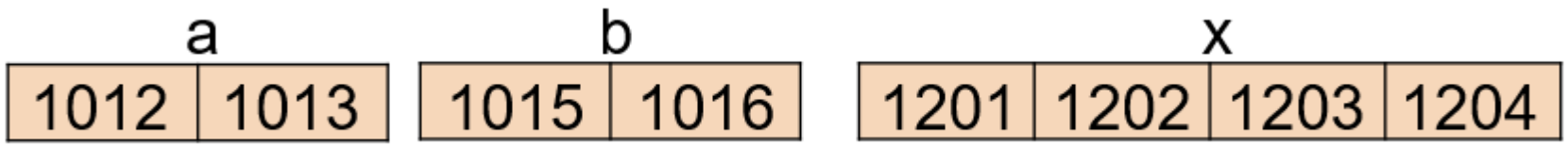
数组

- 一维数组的定义和一维数组元素的引用
- 一维数组和指针
- 函数之间对一维数组和数组元素的引用
- 一维数组应用举例
- 二维数组的定义和二维数组元素的引用
- 二维数组和指针
- 二维数组名和指针数组作为实参
- 二维数组程序举例

一、指针

1、复习：指针变量的**定义**、**类型**和**赋值**

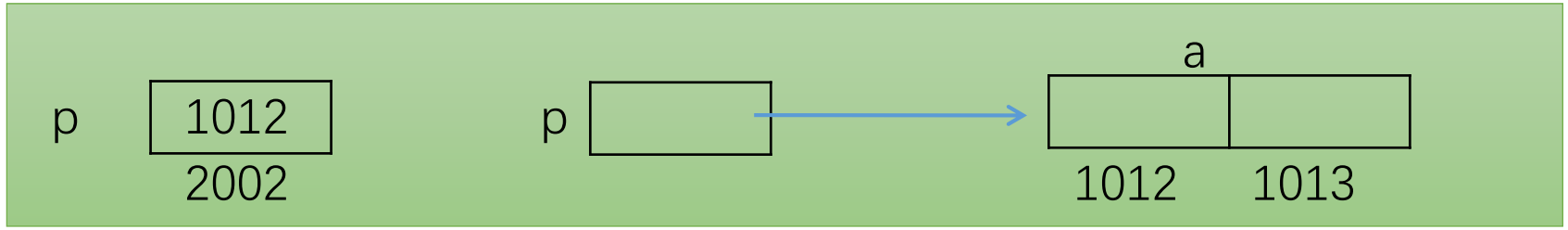
- 一个变量实质上代表“内存中的某个存储单元”。
- 计算机的内存是以字节为单位的一片连续的存储空间,每一个字节都有一个编号,这个编号就称为内存地址。
- 在VC 6.0中, short int型数据占2字节、int型数据和float型数据占4字节、double型数据占8字节、char型数据占1字节、指针变量占4字节。
- 若有定义: short int a, b; float x;



变量在内存中所占字节编号的地址示意图

1、复习：指针变量的**定义、类型**和**赋值**

- 对变量进行存取操作,就是对某个地址的存储单元进行操作。这种直接按变量的地址存取变量值的方式称为“直接存取”方式。
- 用来存放内存地址的变量称为“指针变量”，例如：



存放地址的指针变量示意图

变量p存放的是变量a的内存地址，通过变量p存取变量a的值的方式称为“**间接存取**”方式。变量p就是指针变量。称指针变量p指向了变量a，变量a是指针变量p所指的对象，这种**“指向”关系**是通过地址建立的。

1、复习：指针变量的**定义**、**类型**和**赋值**

定义指针变量的一般形式:

类型名 *指针变量名1, *指针变量名2, ……;

其中, 每个变量前的星号*是一个说明符,用来说明该变量是指针变量, 该星号不可省。

int *pi, *pj;

这里说明了pi和pj是两个指向整型(int类型)变量的指针, 也就是说变量pi和pj中只能存放int类型变量的地址, 称int是指针变量pi和pj的**基类型**。

1、复习：指针变量的**定义**、**类型**和**赋值**

double *pd;

/* pd的基类型为double型，pd中只能存放double型变量的地址 */

char *s1, *s2;

/* s1和s2的基类型为char型, s1和s2中只能存放char型变量的地址*/

一个指针变量中存放的是一个存储单元的地址值。

“一个存储单元”代表的字节数不同: 在VC++中，对short int类型整数，它代表2个字节; 对int类型或float类型, 它代表4个字节,这就是基类型的不同含义。

为什么指针变量要有**基类型**?

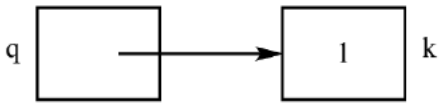
一个指针变量存放的是一个存储单元的地址值，不同的存储单元存储的字节数不一样，比如，short int型2个字节，float型4个字节，如果对指针进行地址移动操作，移动的最小单位即一个存储单元，不同基类型跨越的字节数是不一样的，因此指针变量必须区分基类型。

1、复习：指针变量的**定义、类型和赋值**

一个指针变量可以通过不同的方式获得一个确定的地址值, 从而指向一个具体的对象。

1 .通过求地址运算符(&)获得地址值

- int k =1, * q, * p ;
- q =&k;
- 把变量k的地址赋予了q,下图是指针变量q与变量k的关系示意图:

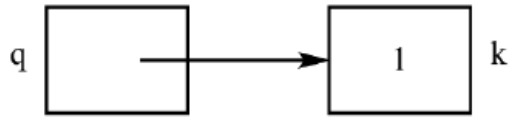


- q指向了变量k。
- 说明： 求地址运算符&只能应用于变量和数组元素,不可以用于表达式、常量。
- 表达式q =&(k +1)是错误的。

1、复习：指针变量的**定义**、**类型**和**赋值**

2 .通过指针变量获得地址值

- 可以通过赋值运算,把一个指针变量中的地址值赋给另一个指针变量,使这两个指针指向同一地址。例如,
- int k =1, * q, * p ;
- q = &k;
- p = q;
- 使指针变量p中也存放了变量k的地址,也就是说指针变量p和q都指向了变量k。图表示了变量q、p和k的关系:



- 注意:当进行赋值运算时,赋值号两边指针变量的基类型必须相同。

1、复习：指针变量的**定义**、**类型**和**赋值**

给指针变量赋NULL值，使指针变量具有一个确定的值—“空”。例如:

- int *p =NULL;

NULL的代码值为0,当执行了以上的赋值语句后，称p为空指针。与其等价的语句:

- p= '\0'　或　 p = 0;

NULL是在stdio.h中定义的预定义符,使用NULL时,应该在程序的前面用预定义行:

- #include <stdio.h >

注意：指针p并不是指向地址为0的存储单元,而是具有一个确定的值—“空”。企图通过一个空指针去访问一个存储单元时，将会得到一个出错信息。

2、对指针变量的操作

——通过指针来引用一个存储单元

“**间接访问运算符**”(也称间址运算符)称为单目运算符:“*”。当指针变量中存放了一个确切的地址值时,就可以用“间接访问运算符”通过指针来引用该地址的存储单元。

假定有以下定义和语句:

```
int *p, i =10, j ;
p =&i;
j =*p;
```

j =*p;是把p所指存储单元(i)的内容(整数10)赋予变量j,
等价于:j =i;

j=*(&i);
首先&i求出变量i的地址,再取地址&i中的内容赋予j。根据优先级原则,可以写成:
j =*&i;

2、对指针变量的操作

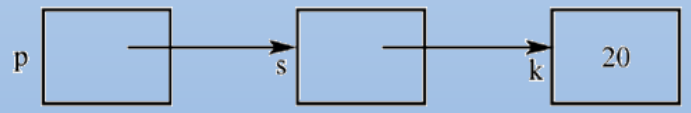
——通过指针来引用一个存储单元

“间接访问运算符”(也称间址运算符)称为单目运算符:“*”。当指针变量中存放了一个确切的地址值时,就可以用“间接访问运算符”通过指针来引用该地址的存储单元。

若有以下定义和语句:

```
int **p, *s, k =20;
s =&k;  p =&s;
```

可以用图形象地表示变量p、s和k的关系:



*s代表存储单元k,*p代表存储单元s,因此**p也代表存储单元k。

其中p是一个指向指针的指针，p的基类型是基类型为整型的指针的**指针类型**。

2、对指针变量的操作——移动指针

移动指针是对指针变量加上或减去一个整数，或通过赋值运算，使指针变量指向相邻的存储单元。

只有当指针指向**一串连续的存储单元**时，指针的移动才有意义。

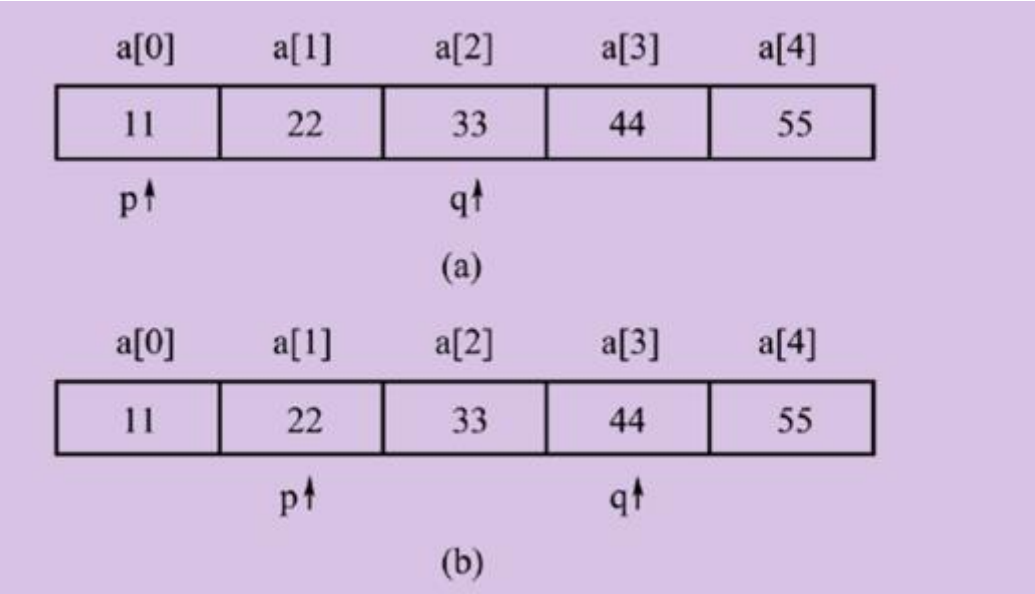
可以对**指向同一串连续存储单元的两个指针进行相减的运算**。

不可以对指针进行任何其他的算术运算。

2、对指针变量的操作——移动指针

假定在内存中开辟了五个连续的、存放int类型整数的存储单元:a[0]、a[1]、a[2]、a[3]、a[4]，其中分别存有:11、22、33、44、55,同时有如下定义:

```
int *p, *q, a[5];
p=a;          /* p指向存储单元a[0] */
```



p=a; 在图(a)中，

(1) q =p+2 ; 问q指向哪儿?

(2) 然后q++指向哪儿?

(3) 然后再q++指向哪儿?

(4) 然后再q--指向哪儿?

p=a; 在图(b)中

(1) p++指向哪儿?

(2) 然后q =p+2 ; 问q指向哪儿?

(3) 然后再q++指向哪儿?

用printf 演示!

2、对指针变量的操作——指针比较

在关系表达式中可以对两个指针进行比较。

例如:

p和q是两个指针变量, 以下语句是正确的:

```
if ( p <q ) printf ( " p points to lower memory than q .\n " ) ;
if ( p == '\0' ) printf ( " p points to Null .\n " ) ;
```

通常**两个或多个指针指向同一目标(如一串连续的存储单元)**时比较才有意义。

3、函数之间地址值的传递

形参为指针变量时实参和形参之间的数据传递

若函数的形参为指针类型，调用该函数时，**对应的实参必须是类型相同的地址值**或者是已指向某个存储单元的指针变量。

例：编写函数myadd (int *a, int *b)，函数中把指针a和b所指的存储单元中的两个值相加，然后将和值作为函数值返回。在主函数中输入两个数给变量，把变量地址作为实参，传送给对应形参。

```
#include <stdio.h>
int myadd ( int *a, int *b )
{  int sum;
   sum =*a+*b;
   return sum;
}
```

```
main( )
{  int x, y, z ;
   printf( " Enter x, y :  " );
   scanf ( " %d%d " , &x, &y );
   z =myadd ( ?, ?);
   printf( " %d+%d =%d\n " , x, y, z );
}
```

这里填什么？

演示

3、函数之间地址值的传递

通过传送地址值在被调用函数中

直接改变调用函数中的变量的值

- 通过return语句返回函数值只能返回一个数据。
- 通过传送地址值,可以在被调用函数中对调用函数中的变量进行引用,使得通过形参改变对应实参的值。

例：调用swap函数, 交换主函数中变量x和y中的数据。

```
#include <stdio.h>
void swap( int *, int * );
main( )
{  int x =30, y =20;
   printf( " (1) x =%d, y =%d\n " , x, y );
   swap ( &x, &y );
   printf( " (4) x =%d, y =%d\n " , x, y );
}
void swap ( int *a, int *b )
{  int t;
   printf( " (2) a =%d, b =%d\n " , * a, * b );
   t =*a;  *a =*b;  *b =t;
   printf( " (3) a =%d, b =%d\n " , * a, * b );
}
```

3、函数之间地址值的传递

函数返回地址值

- 函数值的类型不仅可以是简单的数据类型, 而且**可以是指针类型**。

例：以下函数把主函数中变量 i 和 j 中存放较大数的那个地址作为函数值传回。

```
#include <stdio.h>
int * fun ( int*, int* );    /*函数说明语句*/
main ( )
{  int * p, i , j;
   printf ( " Enter two number:  " );
   scanf (  " %d%d " , &i, &j );
   p =fun ( &i, &j );
   printf ( " i =%d, j =%d, *p =%d\n " , i, j , *p );
}
int *fun ( int *a, int *b )
{  if (* a >*b ) return a;
   return b;
}
```

二、数组和指针

- 一维数组的定义和一维数组元素的引用
- 一维数组和指针
- 函数之间对一维数组和数组元素的引用
- **一维数组和指针应用举例**
- 二维数组的定义和二维数组元素的引用
- 二维数组和指针
- 二维数组名和指针数组作为实参
- 二维数组程序举例

1、一维数组的定义和一维数组元素的引用

数组：包含一组具有同一类型的变量，这些变量在内存中占有连续的存储单元。

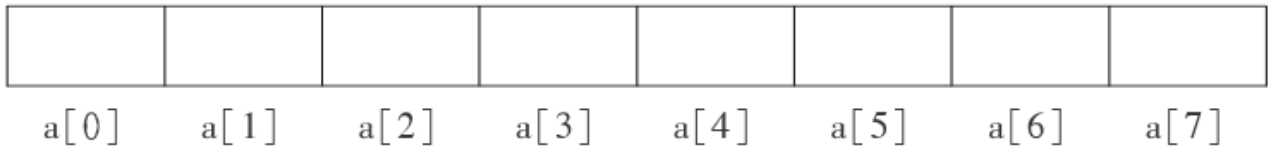
一维数组的定义形式如下:

类型名 数组名[整型常量表达式];

例如: int a[8] ;

int是类型名，a[8]是一维数组说明符。

C编译程序将为a数组在内存中开辟如图所示的8个连续的存储单元,每个存储单元的名字分别为a[0]，a[1]，a[2]，a[3]，a[4]，a[5]，a[6]，a[7],可以用这些名字直接引用各存储单元。



1、一维数组的定义和一维数组元素的引用

一维数组元素的引用

- 引用形式:
  数组名[下标表达式]
- 例如:
  double x[8] ;
  则x[0]、x[j]、x[i+k] 都是对x数组中的元素的合法引用形式。

1) 一个数组元素实质上就是一个变量名, 代表内存中的一个存储单元。一个数组占有一串连续的存储单元。

2) 一个数组不能整体引用。**数组名中存放的是一个地址常量**, 它**代表整个数组的首地址**。

3) 在引用数组元素时, 数组元素中下标表达式的值必须是整数。

1、一维数组的定义和一维数组元素的引用

一维数组的定义和数组元素引用举例

例1：编写程序，定义一个含有30个元素的int类型数组。依次给数组元素赋奇数1、3、5、…，然后按每行10个数顺序输出，最后再按每行10个数逆序输出。

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

#define  M 30
int main ( )
{  int s[M], i, k=1;
    for ( i =0; i <M; i++) { s[i] = k ; k+=2;}
    printf ("\n Sequence Output:\n");
    for ( i =0; i <M; i++)
    {  printf ("%4d", s[i] );
        if ( (i+1)%10==0 ) printf ("\n");
    }
    printf ("\n Invert Output:\n");
    for ( i =M-1; i >=0; i--)
        printf ("%3d%c", s[i], (i%10==0)?'\n':' ');
    printf("\n");
    return 0;
}
```

2、一维数组和指针

一维数组和数组元素的地址

- 在函数体中或在函数外部定义的**数组名是一个存放地址值的指针变量名,其中的地址值是数组第一个元素的地址**，这个指针变量中的地址值不可改变，即不能给数组名重新赋值!

数组名是一个地址常量。
若有定义:
float a[10], *p, x;
则
a =&x; 或a++; 都是非法的。

p =a+2 是合法的。

判断下面语句是否合法:

```
*(a +2) ; p[1];
for (k =0; k <10; k++) scanf ( " %d " ,a+k );
for ( p =a, k =0; k <10; k++)  p++;
for ( p =a, k =0; k <10; k++) { scanf ( " % d " , p ); p++; }
for ( p =a, k =0; k <10; k++) scanf ( " % d " , p++);
for ( p =a; p-a <10; p++) scanf ( " % d " , p );
```

p指向了a数组的首地址。可以使用“**间接访问运算符**”，通过指针变量p来引用a数组中的元素。

```
1）for ( p =a , k =0; k <10; k++) printf ( " %4 d " , *( p+k ) );
2）for ( p =a , k =0; k <10; k++) { printf( " %4 d " , * p ); p++;}
3）for ( p =a , k =0; k <10; k++) printf( " %4 d " , * p++);
4）for ( p =a , p-a <10; p++) printf( " %4 d " , * p );
```

2、一维数组和指针

通过数组的首地址引用数组元素

a是a数组元素的首地址, a的值等于&a[0], 则

a +1的值等于&a[1]、a +2的值等于&a[2]、……、a +9的值等于&a[9]。

对于数组元素a[i],可以用 *&a[i] 或 *(a+i)来引用。

```
for ( k =0; k <10; k++) printf ( " %4d " , *( a+k ) );
```

相当于:

```
for ( k =0; k <10; k++) printf ( " %4d " , a[k] );
```

3、函数之间对一维数组和数组元素的引用

数组元素作实参

- 当调用函数时，数组元素作为实参传送给形参，每个数组元素实际上代表内存中的一个存储单元，对应的形参必须是类型相同的变量。

数组名作实参

- **数组名是一个地址值**，数组名作为实参传送时，**对应的形参应当是一个指针变量**，此指针变量的基类型必须与数组的类型一致。在函数中，可以通过此指针变量来引用调用函数中对应的数组元素，实现对调用函数中对应的数组元素进行操作而改变其中的值。

数组元素地址作为实参

- 当用数组元素地址作为实参时，因为是地址值，所以对应的形参也应当是基类型相同的指针变量。

4、一维数组和指针应用的典型例题

例2：编写函数，对具有10个元素的char类型数组，从下标为4的元素开始，全部设置星号“*”，保持前4个元素中的内容不变。

```
#include <stdio.h>
#define M 10
#define B 4
void setstar ( char  *, int );
int main ( )
{char c[M] ={ 'A','B','C','D','E','F','G','H','I','J' };
  setstar ( &c[4], M-B );

}

void setstar ( char *a, int n )
{ int i;
   for ( i =0; i <n; i++)  *( a+i ) = ? ;
}
```

（2）这里是什么作为实参？

（3）为了查看结果，这里应添加什么语句？

（1）问号位置填什么？

4、一维数组和指针应用的典型例题

例3：编写程序，定义一个含有15个元素的数组，并编写函数分别完成以下操作：
(1)调用C库函数中的随机函数给所有元素赋以0 ~49的随机数；
(2)输出数组元素中的值；
(3)按顺序对每隔三个数求一个和数，并传回主函数；
(4)最后输出所有求出的和值。

调用随机函数的方法如下(在程序开头应该包含头文件stdlib.h)

```
n=rand ()% x;
```

n将得到一个0到x -1的随机整数。因此，可用rand()%50产生0 ~49的随机数。

程序设计思想：1、模块化，单独函数完成单独模块功能。2、然后或者先主函数。

(1)调用C库函数中的随机函数给所有元素赋以0 ~49的随机数；

```
void getrand ( int *a, int n )
{  int i;
   for ( i =0; i <n; i++) a[i] = rand( )%50;
}
```

(2)输出数组元素中的值；

```
void printarr ( int *a, int n )
{  int i;
   for ( i =0; i <n; i++)
   {  printf ("%5d", a[i] );
      if ( ( i+1)%5 ==0 ) printf ("\n");
   }
   printf("\n");
}
```

4、一维数组和指针应用的典型例题

(3)按顺序对每隔三个数求一个和数,并传回主函数;

```
void getave( int *a, int *b, int n )
{  int i, j, sum;
  for ( sum =0, i =0, j =0; i <=n; i ++)
      {  sum+=a[i];
         if ( ( i+1) %3 ==0 )
          {  b[j] = sum;
             sum =0;
             j++;
          }
       }
}
```

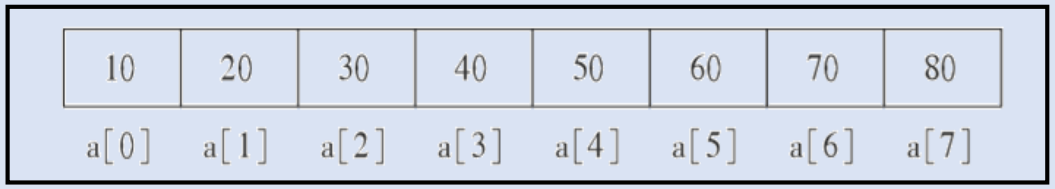
(4) 主函数;

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 15
#define N 3
void getrand( int *, int );
void getave( int *, int *, int );
void printarr( int *, int);
int main ( )
{  int x[SIZE] , w[SIZE/N] ={ 0 };
  getrand( x, SIZE);
  printf("Output %d random numbers:\n", SIZE);
  printarr( x, SIZE );
  getave( x, w, SIZE );
  printf("Output 5 sum numbers:\n");
  printarr( w, SIZE/N );
  return 0;
}
```

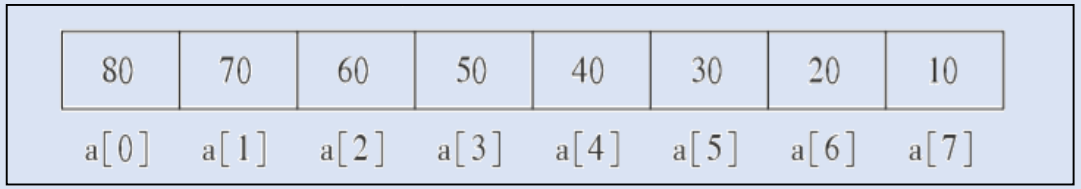
4、一维数组和指针应用的典型例题

例4：将数组中的数按颠倒的顺序重新存放。在操作时,只能借助一个临时存储单元而不得另外开辟数组。

不是要求按颠倒的顺序打印数据, 而是要求按逆序重新放置数组中的内容。假定a数组有8个元素,它们中的原始内容如图所示:



现要求改变成如下图所示的存储内容:



程序设计思想：1、模块化，单独函数完成单独模块功能。2、然后或者先主函数。

4、一维数组和指针应用的典型例题

（1）按逆序重新放置数组中的内容，只能借助一个临时存储单元

```
void invert( int *a, int n )
{  int i, j, t;
   i =0;  j =n-1;
   while( i <j )
        {  t =a[i]; a[i] = a[j]; a[j] = t;
           i++; j--;
        }
}
```

（2）打印数组

```
void printout ( int *s, int n )
{  int i ;
  for( i =0; i <n; i++) printf ("%4d ", s[ i] );
  printf("\n");
}
```

（3）主函数

```
#include <stdio.h>
#include <stdlib.h>
#define NUM 8
void invert(int *, int);
void priout(int *, int);
int main( )
{
  int a[NUM] ={10,20,30,40,50,60,70,80};
  printf ("Output primary data: \n");
  printout(a, NUM);
  invert(a, NUM);
  printf("Output the inverse data: \n ");
  printout(a, NUM);
}
```

4、一维数组和指针应用的典型例题

例5：已知整型数组中的值在0至9的范围内，统计每个整数的个数。

程序设计思想：1、模块化，单独函数完成单独模块功能。2、然后或者先主函数。

（1）主函数

```
#include <stdio.h>
#include <stdlib.h>
#define M 20
#define N 10
void getdata( int *s);
void statistic( int *a, int *c );
void outdata( int *c);

int main( )
{  int a[M], c[N];
   getdata(a);
   statistic(a, c);
   printf("Output the result:\n");
   outdata( c );
   return 0;
}
```

（2）产生0至9的范围内的数

```
void getdata( int *s)
{  int i;
   for(i =0; i<M; i++)
    {s[i] = rand()%10; printf("%d ", s[i]);}
   printf("\n");
}
```

（3）统计每个整数的个数

```
void statistic( int *a, int *c )
{  int i;
  for ( i =0; i <N; i++) c[i] =0;
  for ( i =0; i <M; i++) c[a[i]] ++;
}
```

（4）输出每个整数的个数

```
void outdata( int *c)
{  int i;
  for (i =0; i <N; i++)
      printf("c[%d] = %d\n", i, c[i]);
}
```

4、一维数组和指针应用的典型例题

例6：w数组中存放n个数据，编写函数删除下标为k的元素中的值。

程序设计思想：1、模块化，单独函数完成单独模块功能。2、然后或者先主函数。

（2）删除下标为k的元素中的值

```
int arrdel ( int *w, int n, int k )
{  int i;
   for ( i=k; i<n-1; i++) w[i] = w[ i+1];
   n--;
   return n;
}
```

（3）打印阵列元素中的值

```
void arrout ( int *a, int n )
{  int i ;
   for( i =0; i <n; i++) printf ("%4d ", a[i] );
   printf("\n");
}
```

（1）主函数

```
#include <stdio.h>
#define NUM 10
int arrdel( int *, int , int );
void arrout( int *, int );
int main( )
{  int n, k;
   int a[NUM] ={21,22,23,24,25,26,27,28,29,30};
   n = NUM;
   printf("The original array a is: \n");
   arrout(a, n);
   k = scanf("%d",&k);
   n = arrdel(a, n, k);
   printf("The modified array a is: \n");
   arrout(a, n);
}
```

4、一维数组和指针应用的典型例题

例7：用选择法对数组中的数进行排序(按由小到大顺序)。

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

#define NUM 6
void arrsort( int *, int );
void arrout( int *, int );
int main( )
{  int a[NUM] ={5,7,4,2,8,6};
  arrout(a, NUM );
  arrsort(a, NUM );
  arrout(a, NUM );
}
```

```
void arrsort( int *a, int n )
{  int i, j, t;
   for ( i = 0; i < n-1; i++)
    {
      for ( j = i+1; j < n; j++)
          if ( a[i] > a[j] )
          { t =a[i]; a[i] = a[j]; a[j] = t;}
    }
}

void arrout(int *a, int n )
{  int i;
  for( i =0; i <n; i++) printf ("%d ", a[i]);
  putchar('\n');
}
```

强调：课程统一要求指针作为形参调用数组

变长数组定义：

```
int n;
scanf(“%d”,&n);
int a[n]; //只有C99支持n决定数组范围
```

例：打印数组元素函数

```
void arrout1( int n, int a[n])
{  int i ;
   for( i =0; i <n; i++) printf ("%4d ", a[i] );
   printf("\n");
}
```

```
void arrout( int n, int *a)
{  int i ;
   for( i =0; i <n; i++) printf ("%4d ", a[i] );
   printf("\n");
}
```

```
#include <stdio.h>
#include <stdlib.h>
#define NUM 10
void arrout1( int n, int a[n] );
```

不推荐变长数组作为形参，只得一半！

```
void arrout( int n, int *);
```

课程要求指针作为形参

```
int main( )
{  int n;
   int a[NUM] ={21,22,23,24,25,26,27,28,29,30};
   n = NUM;
   printf("The original array a is: \n");
   // arrout1(n,a);
   arrout(n,a);
}
```