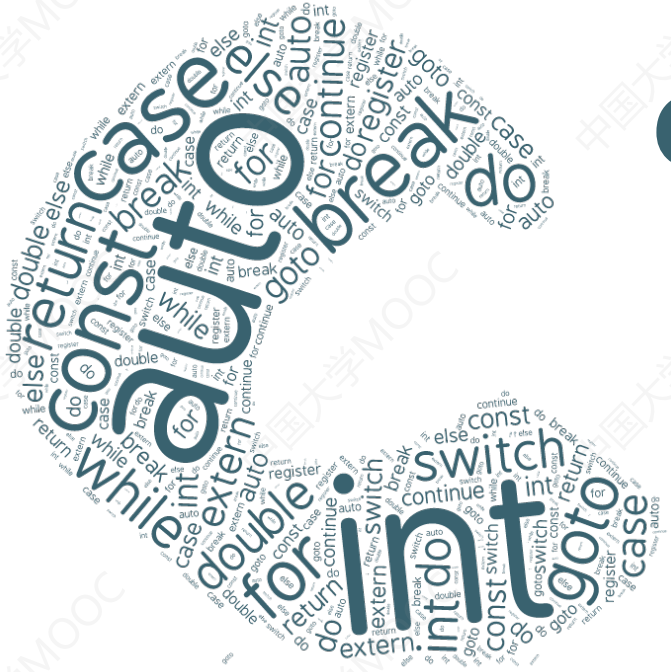

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while

chapter 6

指针

目录
content

1 指针的基本概念

2 指针运算

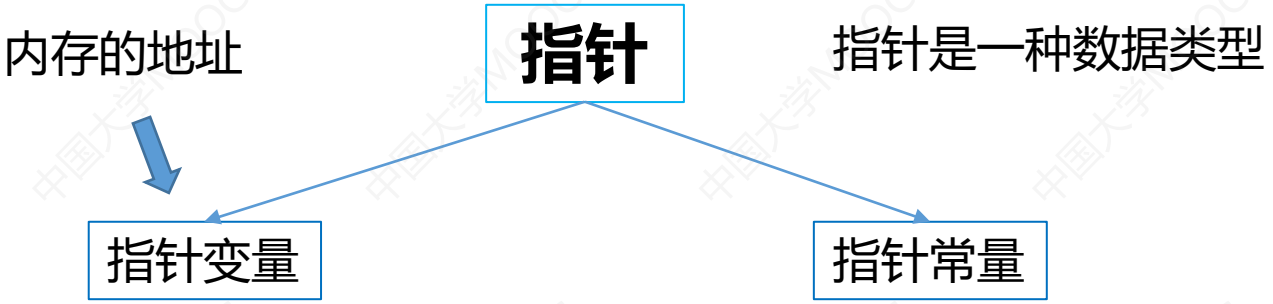
3 指针与数组

4 指针数组与多级指针

5 指针与函数

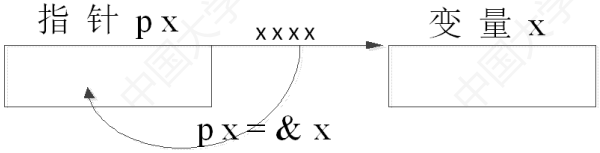
6 综合应用

6.1.1 指针的基本概念

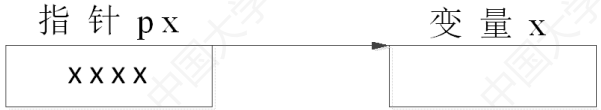


例：int *px, x;

px = &x;



变量的地址装入指针



指针指向变量

指针的目标变量 x是px的目标变量

变量的访问方式 直接访问方式 x =10;

间接访问方式 *px =10;

6.1.2 指针变量的定义与初始化

指针变量的定义

<存储类型> <数据类型> *指针名;

存储类型是指针变量本身的存储类型

数据类型说明指针所指向目标的数据类型

例如：

```
int  *px;
char  *name;
static  int  *pa;
int  *px,  *py,  *pz;
char  cc,  *name;
int *pointer=1000;
```

具有相同存储类型和数据类型的指针可在一行中说明，也可和普通变量在一起说明

不要将一个整型量赋给一个指针变量

指针的存储类型和指针说明的程序位置决定了指针的寿命和可见性。即指针变量也分为内部的和外部的，全局的和局部的。

6.1.2 指针变量的定义与初始化

void指针: **<存储类型> void *指针名;**

例:

```
int  x,*px=&x;
void *pp;
pp = px;
```

可以将已定向的各种类型指针直接赋给void型指针

```
int *px;
void *pp = calloc(100);
px = (int *)pp;
```

必须采用强制类型转换

指针变量的初始化 <存储类型> <数据类型> *指针名[=初始地址值];

例如:

```
char cc;
char  *pc = &cc;

int   n;
int   *p=&n;
int   *q=p;

int *pointer=1000;
```

```
#include <stdio.h>
void main()
{
   int a;
   int  *pa = &a;   //指针pa指向a所在内存地址
   a = 10;
   printf("a:%d\n",a);
   printf("*pa:%d\n",*pa);
   printf("&a:0x%lx\n",&a);
   printf("pa:0x%lx\n",pa);
  printf("&pa:%lx\n",&pa);
}
```

运行结果:

```
a:10
*pa:10
&a:0x1000fff4
pa:0x1000fff4
&pa:0x1000fff8
```

指针的使用

取地址运算符 &

& 操作对象

操作对象必须是左值表达式

单目&与操作对象组成地址表达式，运算结果为操作对象变量的地址，结果类型为操作对象类型的指针

例如

```
int  x;
char  y;
double  z;
int  a [4];
register int k;
```

```
&x
&y
&z
&a[2]
&a
&k
```

```
int *
char *
double *
int *
错误
错误
```

6.1.3 指针的使用

取内容运算符 *

* 操作对象

操作对象必须是地址表达式，即指针（地址常量或指针变量）

运算结果为指针所指的对象，即变量本身（间接访问表达式是左值表达式），结果类型为指针所指对象的类型。

例如

```
char  c,  *pc = &c;
*(&c) = 'a';
*pc = 'a';
c = 'a';
```

注意区分*的不同含义：

```
char  *pc;
*pc = 10;
a = a*10;
```

抽象指针说明符

间访运算符（单目*）

乘运算符

单目*和 &的运算关系：

单目*和 &互为逆运算

*(& 左值表达式) = 左值表达式

&(*地址表达式) = 地址表达式

6.1.3 指针的使用

指针的正确用法

（1） 必须按被引用变量的类型说明指针变量；

（2） 必须用被引用变量的地址给指针变量赋值（或用指针变量初始化方式），使指针指向确定的目标对象，然后才能使用指针来引用变量。

例:
```
int *p，a;
 p=&a;
*p = 5;
```

NULL指针

ANSI C++标准定义NULL指针，它作为指针变量的一个特殊状态，表示不指向任何东西

使一个指针变量为NULL，可以给它赋一个零值

测试一个指针变量是否为NULL，可以将它和零进行比较

对一个NULL指针进行解引用操作是非法的

6.1.3 指针的使用

例6.2 输入a和b两个整数，按先大后小的顺序来输出a和b

```
#include <stdio.h>
void main()
{
   int  *p1,*p2,*p,a,b
   a = 10;
   printf(“please enter tow numbers:”);
   scanf(“%d,%d”,&a,&b);
   p1=&a;p2=&b;
   if(a<b)
   {
        p=p1;p1=p2;p2=p;
   }
   printf(“a=%d,b=%d\n”,a,b);

printf(“max=%d,min=%d\n”,*p1,*p2);
   return 0;
}
```

交换两个指针变量的值实现a和b的交换输出

运行结果：

```
please enter two numbers:
5  9
a=5, b=9
max=9,min=5
```

指针的运算种类

算术运算

关系运算

赋值运算

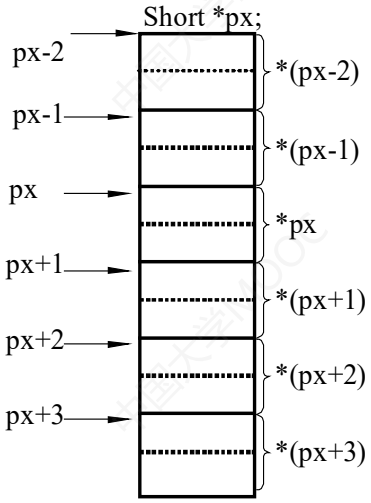
指针的算术运算

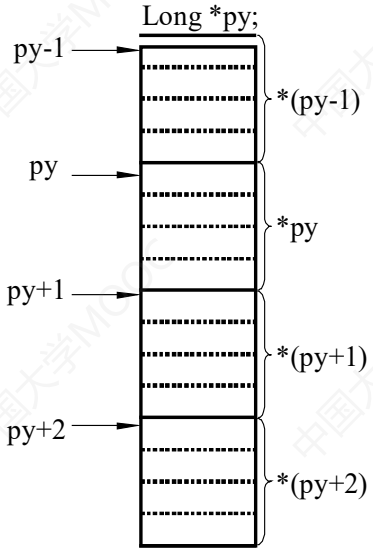
指针可以进行的算术运算：

前提：指针是指向一片存储单元（如数组）

- p1+n
- p1-n
- p1++
- ++p1
- p1--
- --p1
- p1-p2

指针当前指向位置的前方或后方第n个数据的位置，位置的地址值是：(p)±n×sizeof(数据类型)（字节）





指针的算术运算

指针可以进行的算术运算：

前提：指针是指向一片存储单元（如数组）

p1+n

p1-n

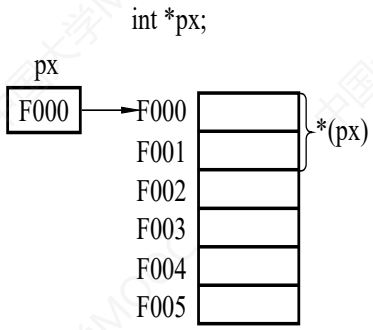
p1++

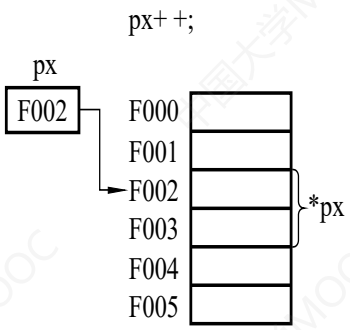
++p1

p1—

--p1

p1-p2





指向下一个/上一个数据的位置，位置的地址值是：(p1)±sizeof(数据类型)（字节）

y= * px++;　　y= * (px++);　　px的当前目标变量的值赋予y后，px加1指向下一个目标

y=++(*px);　　px的目标变量的值加一后赋予y

6.2.1 指针的算术运算

指针可以进行的算术运算有如下几种

前提：指针是指向一片存储单元（如数组）

- p1+n
- p1-n
- p1++
- ++p1
- p1—
- --p1
- p1-p2

```
int a[100],n;
int *p1=a;
*p2=&a[20];
n = p2-p1;
```

```
int a[100],b,n;
int *p1=a; *p2=&b;
n = p2-p1;
```

两指针指向的地址位置之间的数据个数：((p1)-(p2))/sizeof(数据类型)

p1和p2指向同一组数据类型一致的数据

指针的算术运算

例6.2 通过指针变量输出a数组的5个元素

```
int main()
{
    int*p,i,a[5];
    p=a;
    printf(“please input 5 numbers:\n”);
    for (i=0;i<5;i++)
        scanf(“%d”,p++);
    printf (“/n”);
    printf(“the input array is:\n”);
    for(p=a,i=0;i<5;i++)
        printf(“%d”,*p++);
    return 0;
}
```

通过p获得数组a各元素的地址

通过p访问数组a各元素的值

运行结果：

please input 5 numbers:
12 34 56 78 90
the input array is:
12 34 56 78 90

6.2.2 指针的关系运算

两指针之间的关系运算表示它们指向的地址位置之间的关系。
指向后方的指针大于指向前方的指针[存储器的编号从小到大]

前提：两个指针指向同一组类型相同的数据

p1<p2	成立即p1指向位置在p2指向位置的前面
p1>p2	成立即p1指向位置在p2指向位置的后面
p1==p2	成立即p1和p2指向同一位置

关系表达式的值为 1：表达式成立
0：表达式不成立

6.2.3 指针的赋值运算

向指针变量赋值时，赋的值必须是地址常量或变量，不能是普通整数

变量的地址赋予一个指向相同数据类型的指针	char c, *pc; pc=&c;　地址常量
一个指针的值赋予指向相同数据类型的另一个指针	int *p, *q; p=q;　指针变量
数组的地址赋予指向相同数据类型的指针	char name[20], *pname; pname=name;

动态内存分配	int *p, n=20;
	p=(int *)malloc(n*sizeof(int));
	if(p!=NULL)
	{ … }

6.2.3 指针的赋值运算

动态内存分配: 在程序运行期间动态地分配存储空间,分配的内存空间放在数据区的堆（Heap）中

动态内存分配函数: `void * malloc(unsigned size);`

指定所分配内存空间的大小

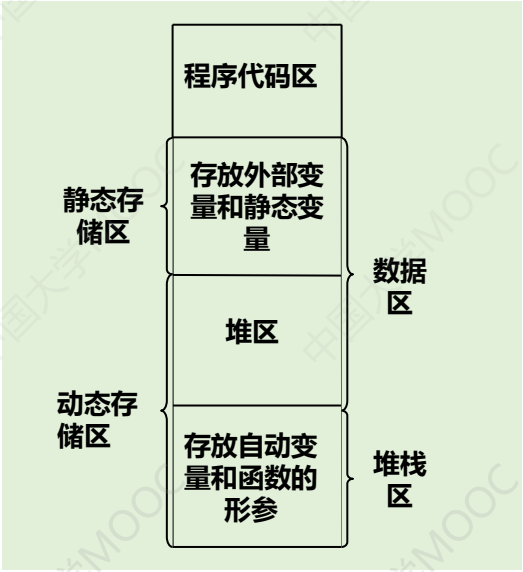
函数运行成功，则返回值是大小为size的内存空间首地址，否则，返回空指针。采用指针赋值操作把它的返回值通过类型强制转换赋给一个指向相同数据类型的指针变量

malloc函数调用一般形式

```
if((指针名 = (类型 * )malloc(空间大小)) == NULL)
{
    出错处理操作
}
```

原型函数在stdlib.h中，在使用它们的程序开头处，必须写有:

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



6.2.3 指针的赋值运算

动态内存释放函数: void free(void * ptr);

接受malloc( )函数在堆区中所分配的内存空间首地址

free( )是malloc( )的配对物，即在整个源程序内，它们是成对出现的

例6.4 使用malloc和free函数的示例程序

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
int main()
{
    int i = 0, *a, N;
    printf( "Input array length:" );
    scanf( "%d" ,&N);
    a = (int*)malloc(N*sizeof(int));
    for(i = 0;i<N;i++)
    {
        a[i] = i+1;
        printf( "%-5d" , a[i]);
        if((i+1)%10 == 0)
            printf( "\n" );
    }
    free(a);
    printf( "\n" );
    return 0;
}
```

运行结果:

```
Input array length:15
1   2   3   4   5   6   7   8   9   10
11  12  13  14  15
```

数组与指针在访问内存时采用统一的地址计算方法，都可以处理内存中连续存放的一系列数据

6.3.1 一维数组与指针

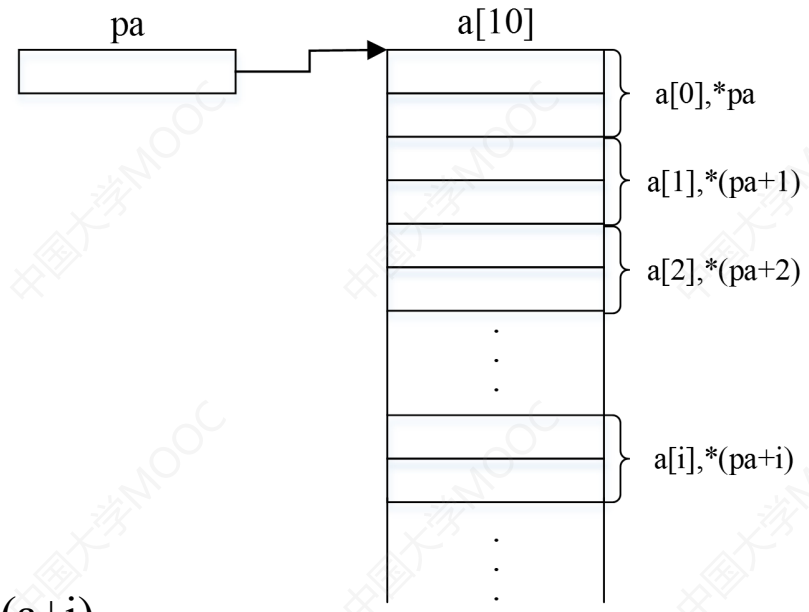
一维数组与指针的关系

```
int  a[10];
int  *pa;
pa=a;   或pa=&a[0];
```

*pa ⟷ a[0]

*(pa+1) ⟷ a[1]

*(pa+i) ⟷ a[i] ⟷ pa[i] ⟷ *(a+i)



6.3.1 一维数组与指针

例6.6 指针和数组表现形式互换性的程序

```
#include <stdio.h>
void main( )
{
   int i,*pa;
   int a[]={2,4,6,8,10};

   pa=a;

   for (i=0 ; i<5 ; i++)
      printf ( "a[%d]:%d " , i ,pa[i]);

   printf (" \n" );

   for(i=0 ; i<5 ; i++)
      printf ( "*(pa+%d):%-4d" , i,*(a+i));

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

指针指向数组的首地址

指针采用数组的形式使用

数组采用指针的形式使用

指针是地址变量，而数组名是地址常量

```
pa = a;
pa++,p--;
pa += n;
```

```
a = pa;
a++, a--;
a += n;
```

运行结果：

```
a[0]:2    a[1]:4    a[2]:6    a[3]:8    a[4]:10
*(pa+0):2  *(pa+1):4  *(pa+2):6  *(pa+3):8  *(pa+4):10
```

6.3.1 一维数组与指针

例6.8 将数组a中n个整数按相反的顺序存放

```
#include <stdio.h>
void invert(int  *pdata,int n);  //函数原型声明
void main( )
{
    int i;
    int data[10]={1,80,2,5,8,12,45,56,9,6};
    int *p=data;      //定义指针变量p
    printf("original array:\n");
    for(i=0;i<10;i++)
        printf("%-4d",*(p+i));
    printf("\n");

    invert(p,10);
    printf("inverted array:\n");
    for(i=0;i<10;i++)
        printf("%-4d",p[i]);
}
void invert(int *pdata, int n)
{
    int i,j,temp;
    for(i=0,j=n-1;i<j;i++,j--)
   {
        temp=pdata[i];
        pdata[i]=pdata[j];
        pdata[j]=temp;
    }
}
```

调用函数(以指针作为实参)

一维数组的信息一般要包含数组地址和元素个数两个参数

可替代为：void invert(int pdata[],int n)

分析：将a[0]与a[n-1]对换，再将a[1]与a[n-2]对换，……直到将a[(n-1)/2]与a[n-int((n-1)/2)]对换

运行结果：

```
original array:
1   80  2   5   8   12  45  56  9   6
inverted array:
6   9   56  45  12  8   5   2   80  1
```

6.3.1 一维数组与指针

例6.9 编写三个函数分别完成指定一维数组元素的数据输入、求一维数组的平均值、求一维数组的最大值和最小值。由主函数完成这些函数的调用。

分析：采用地址传递的方式，把一维数组的存储首地址作为实参数调用函数。在被调用的函数中，以一般（一级）指针变量作为形式参数接收数组的地址。该指针被赋予数组的地址之后，使用这个指针就可以对数组中的所有数据进行处理。

函数原型:

float []

```
void input(float *,int);
float average(float *,int);
void maxmin(float *,int, float *,float *);
```

最大值和最小值需要“返回”，采用地址传递方式

6.3.1 一维数组与指针

例6.9 编写三个函数分别完成指定一维数组元素的数据输入、求一维数组的平均值、求一维数组的最大值和最小值。由主函数完成这些函数的调用。

函数原型声明

```
#include <stdio.h>
void input(float *, int);
float average(float *, int);
void maxmin(float *, int , float *, float *);
void main( )
{     float data[10]; //一维数组定义
      float aver,max,min;
      float *num=data;
      input(data,10);
      aver=average(data,10);
      maxmin(num,10,&max,&min);
      printf("aver=%f\n",aver);     //输出平均值
      printf("max=%f,min=%f\n",max,min);
}
float average(float *pdata, int n)    //数组平均值函数
{      int i;
       float  avg;
       for(avg=0,i=0;i<n;i++)       avg+= pdata[i];
       avg /=n;                    //求平均值
       return(avg);        //将平均值返回给被调用函数
}
```

求最大值和最小值，以指针num以及max与min的地址作为函数实参

```
void input(float *pdata,int n)   //输入数据函数
{
      int i;
      printf("please input array data: ");
      for(i=0;i<n;i++)   //逐个输入数组的数据
            scanf("%f",pdata+i);
     // scanf("%f",&pdata[i]);
}
void maxmin(float *pdata,int n, float *pmax,
float *pmin)
{     int i;
      *pmax=*pmin=pdata[0];
      for(i=1;i<n;i++)
      {
            if(*pmax<pdata[i])//求最大值
                  *pmax=pdata[i];
            if(*pmin>pdata[i])//求最小值
                  *pmin=pdata[i];
      }
}
```

指针形式

数组形式

6.3.2 多维数组与指针

多维数组的地址:

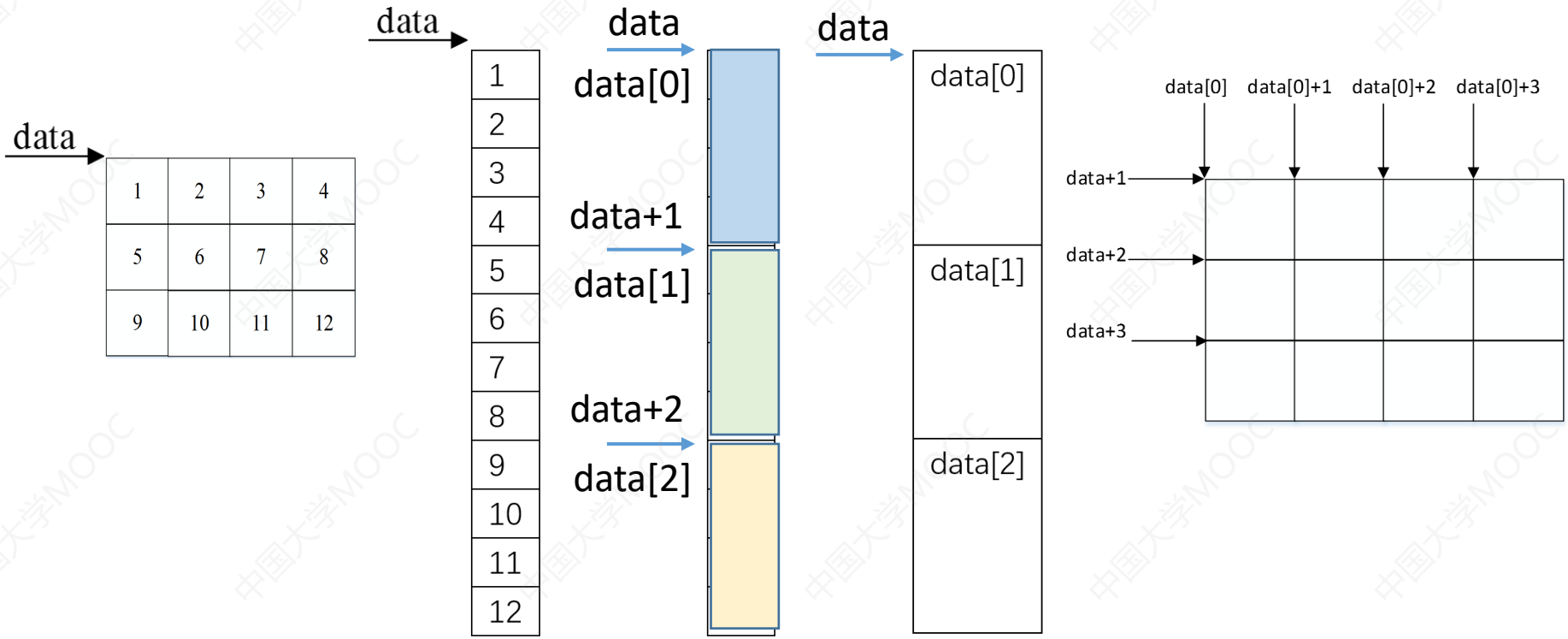
C语言中，数组在实现方法上只有一维的概念，多维数组被看成以下一级数组为元素的一维数组

（1）n维数组（n≥2）可以逐级分解为n-1维数组为元素的一维数组；

（2）n维数组的数组名是指向n-1维数组的指针，其值为n维数组的首地址，类型为n-1维数组类型的指针。

（3）n维数组的元素是指向n-1维数组的元素的指针，其值为n-1维数组的首地址，类型为n-1维数组的元素类型的指针；

多维数组的地址：　int data[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};



第一级分解，数组data被看成长度为3的一维数组

第二级分解，data[0]，data[1]，data[2]又是三个长度为4的一维数组

6.3.2 多维数组与指针

数组指针:

如何用单个指针变量处理多维数组?

单个指针变量可处理一维数组 ⟷ N维数组是以N-1维数组为元素的一维数组

→ 以N-1维数组为数据类型定义指针变量，则可处理以N-1维数组为元素的一维数组

数组指针的一般定义格式为:

<存储类型> <数据类型> (*数组指针名) [元素个数];

数组指针的(物理)地址增量值以N-1维数组的长度为单位

6.3.2 多维数组与指针

数组指针:

(1)不能把一个一维数组的首地址，即一维数组名直接赋给指向相同数据类型的数组指针，如：

```
int  a[10], (*ap)[10];
ap = a;          //错误
ap = (int (*)[10])a;    //正确
```

类型不一致，a的类型为int *, ap的类型为int (*)[10] 。

强制类型转换

或者用初始化操作写成：

```
int a[10], (*ap)[10] = (int ( * )[10])a;
```

(2)数组指针指向了一维数组a，也不能用“ap[i]”或“*ap[i]”等形式的表达式访问一维数组a的第i号元素。

```
(*ap) = a
(*ap)+i = a+i
(*ap)[i] = *((*ap)+i) = a[i] = *(a+i)
```

6.3.2 多维数组与指针

例6.10 用数组指针指向一维数组

```
#include <stdio.h>
 void main( )
{
        int  i;
        int a[4], (*ap)[4];        // 定义数组指针ap
        ap = (int (*)[4])a;        // 给数组指针ap定向，指向一维数组a
         //借助数组指针ap的地址计算公式，用键盘给一维数组a的每个元 素赋值
        for(i = 0; i < 4; i++)
         {
             printf("第[%d]号元素 : ", i);
             scanf("%d", *ap + i);
         }
        //由数组指针ap的地址计算公式读取一维数组a的元素值显示在CRT上
        for(i = 0; i < 4; i++)
             printf("%d\t", *(*ap + i));
        printf("\n");
}
```

例6.11 输出二维数组任意元素

```
#include <stdio.h>
void main( )
{
    int  a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12}; //定义二维数组a并且初始化
    int  (*ap)[4], i, j;        // 定义包含4个元素的一维数组指针ap
    ap = a;            // ap指向二维数组a的第0行
    printf("please enter row and column number:");
    scanf("%d%d",&i,&j);  //键入元素行列号
    printf("a(%d,%d) =%d:\n",i,j,*(*(ap+i)+j));
}
```

6.3.2 多维数组与指针

多维数组与指针的应用：多维数组可以看成是特殊的一维数组，当需要在函数之间进行多维数组的传递时，可以将多维数组进行降维处理，使之变为一维数组然后对降维后的一维数组进行相应的处理。

例6.12 有一个班，3个学生，各学4门功课，计算总平均成绩和第二个学生的平均成绩

```
#include <stdio.h>
float aver(float *,int n);  //原型声明
void main( )
{
   float score[3][4]={{63,65,75,61},
                      {83,87,90,85},
                      {90,95,100,93}};
   printf("total average
score=%f\n",aver(*score,12));
   printf("second student average score
=%f",aver(score[1],4));
}
```

行首地址作为实参

```
float aver(float *pdata,int n)
{
    int i;
    float average=0;   //初始化为零

    for(i=0;i<n;i++)
       average+=pdata[i];  //累加和
    average=average/n;    //求平均值
    return average;       //将平均值返回
 }
```

例6.13 求4×3的矩阵中元素的最大值

```
#include<stdio.h>
int max(int array[][3], int n);   //函数声明
int main( )
{
    int a[4][3]={ {1,2,3},{4,5,6},{3,6,8},{7,12,11} };
    printf（”Max value is %d\n”,max(a, 4)）;

    return 0;
}

int max（int array[][3], int n）
{
    int i，j,max;
    max=array[0][0];
    for(i=0;i<3;i++)
        for(j=0;j<n;j++)
            if(array[i][j]>max) max=array[i][j];
    return(max);
}
```

二维数组名a作为实参

形参实质为数组指针形式

6.3.2 多维数组与指针

例6.14 一个班有3个学生，他们各学4门功课，编程实现分别显示每个学生有几门课程是优秀的（90分以上为优秀）以及每个学生的成绩

```
#include <stdio.h>
void search(int (*p)[4],int n); //原型声明
void main()
{
    //二维数组定义及初始化
    int score[3][4]={{93,96,44,61}, \
                     {83,87,90,45}, \
                     {58,95,26,59}};
    search(score,3); //寻找满足要求的学生
}
```

```
void search( int(*p)[4],int n)
{
   int i,j,k, flag_f;
   for(i=0;i<n;i++)
   {
     //k=0;
     flag_f=0;        //flag_f用来存放优秀的成绩数
     //do
     //{
        for(j=0;j<4;j++)
        {
           if(p[i][j]>90)
           {   flag_f++;
              //break;
           }
        }
       // k=j+1; //k存储上次查询完之后的一个数的列数
     //}  while(j<4);
    if(flag_f >0)
    {printf("No.%d has %d grade excellently, his score \
          are:\n",i+1,flag_f);
     for(j=0;j<4;j++)      printf("%-4d",p[i][j]);
     printf("\n");}
   }
```

void search(int p[ ][4],int n)

数组形式

有优秀成绩时，计数器加1并退出里层循环

运行结果：

```
No.1 has 2 grade excellently ,his score are:93  96  44  61
No.2 has 1 grade excellently, his score are:83  87  90  45
No.3 has 1 grade excellently, his score are:58  95  26  59
```

指针数组

6.4.1 指针数组：指针的集合，它的每一个元素都是一个指针变量，并且它们具有相同的存储类型和指向相同的数据类型。

指针数组的定义：

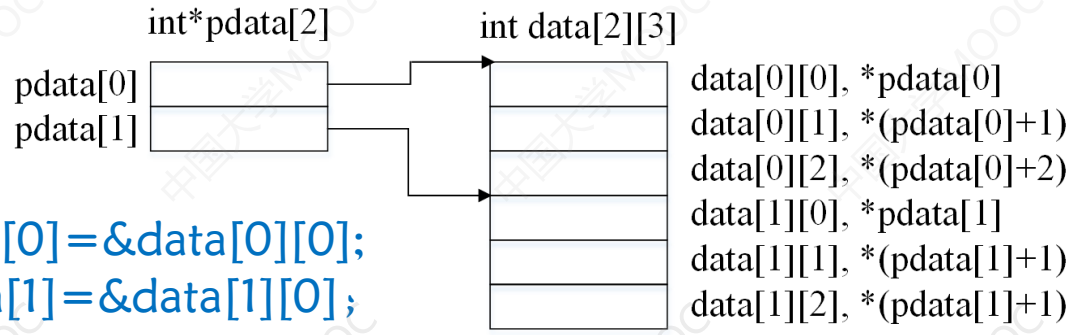
<存储类型> <数据类型> *指针数组名[元素个数];

int *p[2];

指针数组可以用来处理多维数组

int data[2][3];

int *pdata[2];



pdata[0]=data[0]; 或pdata[0]=&data[0][0];

pdata[1]=data[1]; 或pdata[1]=&data[1][0];

指针数组

例6.15 用指针数组处理二维数组的数据

```
#include <stdio.h>
int main( )
{
   int data[2][3], *pdata[2];
   int i,j;

   for(i=0;i<2;i++)      //二维数组赋值
   for(j=0;j<3;j++)
      data[i][j]=(i+1) *(j+1);
   pdata[0]=data[0];     //将指针数组的各个元素指向降维后的
一维数组
   pdata[1]=data[1];
   for(i=0;i<2;i++)
   for(j=0;j<3;j++,pdata[i]++)    //采用指针数组输出数组内容
      printf("data[%d][%d]:%-2d\n",i,j, *pdata[i]);
  return 0;
}
```

运行结果：

```
data[0][0]:1
data[0][1]:2
data[0][2]:3
data[1][0]:2
data[1][1]:4
data[1][2]:6
```

data[i][j]和*(data[i]+j)，*（pdata[i]+j）和pdata[i][j]是意义相同的表示方法

例6.16 指针数组和数组指针分别处理二维数组

```
#include<stdio.h>
void output1(int *app[ ],int n);
void output2(int (*bpp)[3],int n);
void main( )
{
    int *ap[5];
    int (*bp)[3];
    int i,j;
    int    arr[5][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12},{13,14,15}};
    for(i=0;i<5;i++)    ap[i]=arr[i];
    bp=arr;
    output1(ap, 5);        //以指针数组名作为实参
    output2(bp, 5);        //以数组指针名作为实参
}
void output1(int *app[ ], int n)
{
    int i,j;
    printf("the array is:\n");
    for(i=0;i<n;i++)
    {   for(j=0;j<3;j++)
        printf("%-5d", *(*(app+i)+j));
        printf("\n");
    }
}
```

数组指针指向二维数组

二级指针作为形参接收指针数组

```
void output2(int (*bpp)[3], int n)
{
    int i,j;
    printf("the array is:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<3;j++)
        printf("%-5d", *(*(bpp+i)+j));
        printf("\n");
    }
}
```

数组指针作为形参处理二维数组

运行结果：

```
the array is:
1   2   3
4   5   6
7   8   9
10  11  12
13  14  15
the array is:
1   2   3
4   5   6
7   8   9
10  11  12
13  14  15
```

6.4.2 多级指针

一个指针变量指向的变量仍然是一个指针变量，就构成指向指针变量的指针变量，简称指向指针的指针或二级指针

二级指针的声明形式：

存储类型 数据类型 **指针名；

目标变量的数据类型

```
int **ap;
int  *s;
int num;
ap=&s;
s=&num;
num=100;
```

*ap是取ap中的内容，得到一个指针值即s中的内容

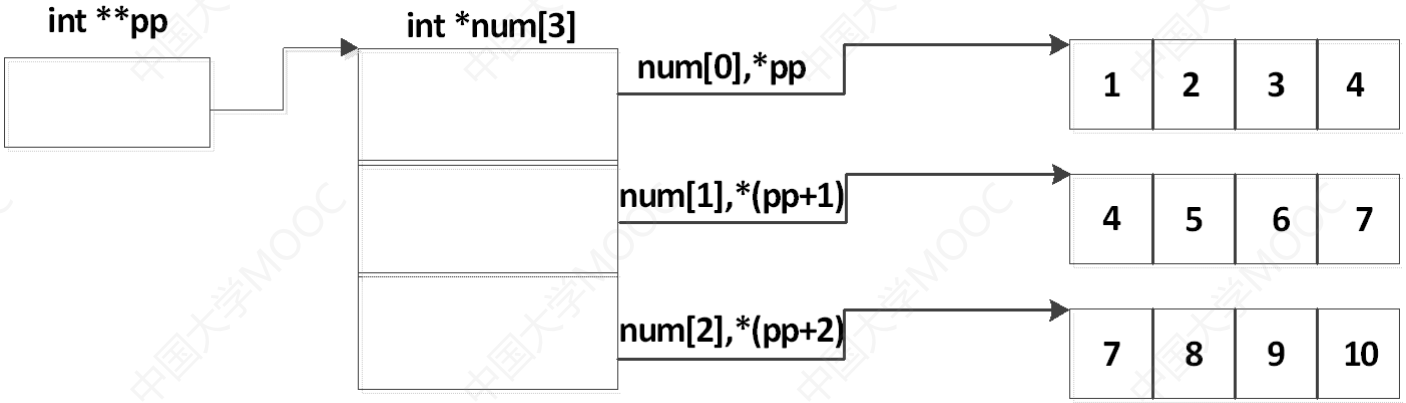
**ap即“*（*ap）”，再取上述“*ap”的内容，得到num的值

num=100; ⇔ *s=100; ⇔ **ap=100;

6.4.2 多级指针

程序中经常使用二级指针来处理指针数组

```
int  arr[3][4]=
{{1,2,3,4},{4,5,6,7},{7,8,9,10}};
int *num[3];
for  (i=0;  i<3;i++)
       num[i]=arr[i];
```



pp指向指针数组num[ ]。pp的目标变量*pp就是num[0], *(pp+1)就是num[1],*(pp+2)就是num[2]。pp就是指向指针型数据的指针变量

多级指针

例6.17 多级指针处理二维数组

```
#include <stdio.h>
int main( )
{
    int  num[3][4]={{0,1,2,3},{1,2,3,4},{2,3,4,5}};
    int  *p[3], **pp;
    int  i, j;
    for (i=0; i<3; i++)
        p[i]=num[i];
   pp=p;     //使二级指针p指向指针数组p的首地址

    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        printf("%-3d",*(*(pp+i)+j));
        printf("\n");
     }
    return 0;
}
```

指针数组p存放数组num每行的首地址

借助二级指针输出数组元素

运行结果：

```
0  1  2  3
1  2  3  4
2  3  4  5
```

表达形式等价

((pp+i)+j)， *(p[i]+j)， p[i][j]，
((num+i)+j)，*(num[i]+j)，num[i][j]

多级指针

例6.17 多级指针处理二维数组

```
#include <stdio.h>
void output(int **p, int n1, int n2);
int main( )
{   int  num[3][4]={{0,1,2,3},{1,2,3,4},{2,3,4,5}};
    int  *p[3], **pp;
    for (i=0; i<3; i++)
        p[i]=num[i];
    pp=p;
    output(pp, 3, 4);  //调用函数
    return 0;
}
void output(int **p, int n1, int n2);  //n1,n2为二级指针指向的数组行列数
{
    int i, j;
    for(i=0;i<n1;i++)
    {
        for(j=0;j<n2;j++)   printf("%-3d",*(*(pp+i)+j));
         printf("\n");
    }
}
```

二级指针变量作为函数形参

6.5.1 返回指针的函数：当函数的返回值是地址时，该函数就是指针型函数。

<存储类型> <数据类型> *函数名(函数的形式参数及说明);

该函数本身的存储特性，分外部型或static型

返回值地址所在的内存空间中存储数据的数据类型

函数体内必须有return语句，其后跟随的表达式结果值可以是变量的地址、数组首地址、已经定向的指针变量、结构变量地址、结构数组的首地址等

这些变量和数组须是全局或静态型，不能把被调用函数内的自动变量的地址和自动型数组的首地址作为指针函数的返回值

6.5.1 返回指针的函数

例6.20 求平方（错误例）

```
#include <stdio.h>
double *square(double);
int main()
{
    double num = 5.0;
    double *ptr = 0;
    ptr = square(num);
    printf("Num's square = %f \n", num*num);
    printf("Result = %f \n", *ptr);
    return 0;
}

double *square(double data)
{
   double result = 0.0;
   result = data*data;
   return &result;
}
```

永远不要从函数中返回局部自动变量的地址

运行结果：

```
Num's square = 25.000000
Result = 6.122651851057717610000000000000000000000e+212
```

6.5.1 返回指针的函数

例6.21 模拟成绩检索系统，对学生成绩进行搜索，找出其中有不及格分数的学生学号和不及格课程号及分数。学生的成绩按照百分制进行记录，学生的学号为（1-10），课程编号为（1-4）。

问题分析与设计：

数据结构：建立学生成绩10*4二维数组，行代表不同学号，列代表不同课程。

函数原型： 完成一行数据的搜索，搜索到不及格分数时，终止搜索，返回不及格分数的存储地址，以及其在数组中的列号

```
int *search( int (*pointer)[4], int *pm);
```

返回不及格分数的存储地址

该不及格分数在数组中的列号

6.5.1 返回指针的函数

```
#include <stdio.h>
/*声明查找函数*/
int *search( int (*pointer)[4], int *pm);
int main( )
{   int score[10][4]
={   {67,68,78,88},{90,69,66,79},{67,70,89,85},{65,76,69,70
},{78,87,83,79},{88,70,48,57},{80,63,90,84},{67,48,70,84},
{92,90,77,70},{87,88,69,84}      };
   int *pt_1;
   int row,column=0;
   int *pm_1=&column;
   for(row=0;row<10;row++)
   {
      pt_1=search(score+row,pm_1);
      if(pt_1!=NULL) //出现不及格
      {
          printf("Stu[%d]'s No.%d course's score
is:",row+1,column);
          printf("%d\n",*pt_1);
          row--;     //该行未扫描完，应继续扫描剩下列
       }
       else *pm_1=0;            //表示该行已扫描完
   }
}
```

逐行扫描的基础上，当发现有不及格现象的时候，转为逐个扫描

```
/*定义查找函数*/
/*入口参数中：(*pointer)[4]是所查找的一行，*pm用于存放列地址*/
int *search( int (*pointer)[4], int *pm)
{
     int i;      int *pt;
     pt=NULL;
 /*采用*pm间接传递了扫描列初址，当该生有多科不及格时为上次扫描之后一列*/
     for(i=*pm;i<4;i++)
     {
         if(*(*pointer+i)<60)
         {
             pt=*pointer+i;
             *pm=(i+1);
             break;
         }
     }
     return pt;
}
```

数组指针传递二维数组

运行结果：

```
Stu[6]'s No.3 course's score is:48
Stu[6]'s No.4 course's score is:57
Stu[8]'s No.2 course's score is:48
```

6.5.2 指向函数的指针

在C语言中，函数整体不能作为参数直接传递给另一个函数。尽管函数不是变量，但它却具有内存物理地址。函数的函数名和数组相似，函数名表示该函数有存储首地址，即函数的执行入口地址。

函数指针：存储函数的入口地址，可以通过该指针变量来调用它所指的函数。

其定义格式 <存储类型> <数据类型> (* 函数指针名)(参数表) [= 函数名] ;

指针所指函数返回值的数据类型

函数指针本身的存储类型，有auto型、static型和extern型

```
float add(float a, float b);

float (*pfun)( float ,float);
pfun = add;
      或
float (*pfun)(float,float) = add;
```

函数指针进行定向操作的两种方式

先定义后用地址赋值语句

初始化操作

函数和指向它的函数指针应具有相同的数据类型，否则，应该采用强制类型转换，两者的形参表必须完全相同

定义一个函数指针时，至少应指明参数表内的参数个数和各参数的数据类型

6.5.2 指向函数的指针

函数指针与数据指针的不同

- 数据指针指向数据区，而函数指针指向程序代码区
- 数据指针的取内容运算表达式“* 数据指针名”，是访问该指针所指的数据，而函数指针的取内容运算表达式“* 函数指针名”，是使程序控制转移到函数指针所指的函数目标代码模块首地址，执行该函数的函数体目标代码。

函数指针代替函数名的调用语句一般格式：

```
(*函数指针名)(参数表)；      ①
   函数指针名(参数表)；      ②
```

```
float c;
c = pfun(3.5, 6.2);
c = (*pfun)(3.5, 6.2);
```

在实际应用中，在一个执行过程中可以调用不同函数时，函数的传递能体现出较大的优越性。

指向函数的指针

例6.22 编写函数实现积分

```
#include <stdio.h>
#include <math.h>
float integrate(float a , float b, int n, float (*f)(float));
float sin2(float x);
float f1(float x);
/*积分函数定义*/
/*a,b为积分边界，n为积分区间分割数，f指向被积分函数*/
float integrate(float a , float b, int n, float (*f)(float))
 {
      float s,d;
      int i;
      d=(b-a)/n;            //微元精度d
      s=(*f)(a)*d;          //微元面积
      for( i=1;i<=n-1;i++)
            s=s+(*f)(a+i*d)*d;   //微元面积相加
      return s;
}
```

```
/定义被积函数*/
float sin2(float x){  return sin(x)*sin(x);  }
float f1(float x){  return x*x+x/2;  }

int main( )
{
    //定义指针指向函数sin2
    float (*p)( float ) = sin2;
    printf("%f",integrate(0,1,100,p));
    printf("%f",integrate(-1,2,100,f1));
    return 0;
 }
```

函数指针变量作为实参

函数名作为实参

运行结果：
0.269143
3.682950

指向函数的指针

例6.24 计算从2015年开始某年母亲节（5月第2个周日）的具体日期

问题分析与设计：

设计思路：首先通过输入获得母亲节的周数（2）和星期数（7），以及需要查询的年份，然后调用日期获取函数（指针函数）得到基准年（2015年）的母亲节是5月10日，然后由指针指向闰年判断函数（函数指针）得到逻辑值来进行日期演算，从而得到当年母亲节的日期

数据结构：写出2015年5月的日期编号，建立基准二维数组。

函数原型：

```
int *GetDate(int (*p)[7],int wk, int dy);
int  Leap_Year_Judge(int y);
```

6.5.2 指向函数的指针

```
#include <stdio.h>
int *GetDate(int (*p)[7],int wk, int dy);
int  Leap_Year_Judge(int y);
void main()
{   int May_2015[6][7]={  {-1,-1,-1,-1,1 ,2 ,3 }, { 4, 5, 6, 7, 8,
9, 10 }, {11,12,13,14,15,16,17},{18,19,20,21,22,23,24},
{25,26,27,28,29,30,31}, {-1, -1,-1,-1,-1,-1,-1}    };
    int  yr, wk, dy, i, date, temp;
    int  (*ad)( int );
    do{
        printf("Enter week(1-6)and day(1-7): ");
        scanf("%d%d\n", &wk, &dy);
        printf("Enter year(>=2015): ");
        scanf("%d\n", &yr);
     } while(wk<1||wk>6||dy<1||dy>7||      \
              (*GetDate(May_2015,wk,dy)==-1));
    date = *GetDate(May_2015,wk,dy);
    ad = Leap_Year_Judge(yr);
```

运行结果：

```
Enter week(1-6)and day(1-7): 2  7
Enter year(>=2015): 2017
the 2017 mother's day is May 14st
```

```
 for( i=2016; i<=yr; i++ )
    {
        if( (*ad)(i) == 0 ) temp=1;
        if( (*ad)(i) == 1 ) temp=2;
        date - = temp;
        if(date <= 7) date += 7;  //取得第i年母亲
节的date
    }
    printf("the %d mother's day is May %dst\n",\
          yr , date);
}
/*日期地址获取函数*/
int *GetDate(int (*p)[7],int wk,int dy)
{
        return &p[wk-1][dy-1];
}

int Leap_Year_Judge(int y)
{
    if( (y%4==0&&y%100!=0)||(y%400==0) )
        return 1;
    else return 0;
}
```

例6.25 编写程序，用指针形式访问数组元素，实现矩阵转置功能

问题分析与设计：

模块划分：矩阵转置功能块：完成矩阵的转置。

数据结构：采用数组指针，以地址传递方式实现函数间二维数组（矩阵）传递。

函数原型：

void convert(int (*a)[], int (*at)[], int row , int col);

功能：矩阵的转置

形参：数组指针a传递转置前的矩阵，数组指针at返回转置后的矩阵数据

row和col分别为矩阵的行列数

6.6 综合举例

```
#include <stdio.h>
#define  N1  3
#define  N2  4
void convert(int (*)[], int (*)[], int , int );

int main( )
{
   int  arr1[N1][N2]={1,2,3,4,5,6,7,8,9,10,11,12};
   int  arr2[N2][N1];
   int  (*p)[N2],      (*pt)[N1];
   int  i, j;
   p=arr1;        pt=arr2;
   convert( p, pt, N1, N2);
   printf("the new array is:\n");
   for(i=0; i<N2; i++)
   {
      for(j=0; j<N1; j++)   printf("%-4d",*(*(pt+i)+j));
      printf("\n");
    }
    printf("the new array is:\n");
    for(i=0; i<N2; i++)
    {
       for(j=0; j<N1; j++)    printf("%-4d",arr2[i][j]);
       printf("\n");
     }
}
```

宏定义行列大小

函数声明

转置后的目标矩阵

定义数组指针

函数调用

采用指针方法输出

采用数组方法输出

```
/**转置函数 convert()
**指针数组a和at指向转置前后的矩阵
**row和col为转置前矩阵的行列值**/
void convert(int (*a)[N2], int (*at)[N1], int
row, int col)
{
   int  i,  j;
   for( i=0; i<row; i++)
   for( j=0; j<col; j++)
       *(*(at+j)+i)=*(*(a+i)+j);
}
```

运行结果：

```
the new array is:
1   5   9
2   6   10
3   7   11
4   8   12
```

例6.26 编写程序，对整型数组进行排序（冒泡法）

问题分析与设计：

设计思路：定义函数指针以方便调用升降序逻辑函数，从而实现升序或降序的排序功能。

数据结构：采用一维数组存储待排序的数据，以地址传递方式实现函数间数组传递。

函数原型：

（1） int ascending(int a, int b);

功能：升序逻辑判断

形参：a,b为待比较的两个整数

返回：若a>b, 值为1, 否则，值为0

（2） int descending(int a, int b);

功能：降序逻辑判断

形参：a,b为待比较的两个整数

返回：若a<b, 值为1, 否则，值为0

（3） void exchange(int *a, int *b);

功能：两个整数的交换

形参：a,b为待交换的两个整数存储地址，通过地址传递方式，实现实参的交换

返回：无

（4）void sort(int a[], int n, int flag);

功能：按给定的升序或降序完成一维数组的排序

形参：a[]为指向待排序数组的指针，采用地址传递方式

n为数组的长度

flag为升降序标志

返回：无

6.6 综合举例

```
#include <stdio.h>
#define  FALSE  0
#define  TRUE   !0
/*******************************/
int ascending( int, int );
int descending( int, int );
void exchange( int *, int * );
void sort(int [], int, int );
int main( )
 {
    int num[10]={1,5,9,2,6,10,3,4,7,8};
    int flag1;          //升降序标志符
    int i;

    scanf("%d", &flag1);
    sort(num,10,flag1);
    printf("sorted array is:");
    for( i=0; i<10; i++ )
       printf( "%-4d", num[i] );
    return 0;
}
```

声明排序函数

待排序数列

排序结果输出

```
void sort(int a[], int n, int flag )
{
    bool ( *ad )( int , int );
    int t, c;
    if( flag==TURE )
       ad=ascending;
    else ad=descending;

    for( t=0; t<n; t++ )
       for( c=0; c<n-1; c++ )
          if(( *ad )( a[c],a[c+1] ))
             exchange( &( a[c] ),&( a[c+1] ) );
}
bool ascending( int a, int b ){ return a>b;}
bool descending( int a, int b ){ return a<b;};

void exchange( int *a, int *b )
 {
    int temp;
    temp= *a;
    *a= *b;
    *b= temp;
```

定义函数指针快速指向升降序逻辑函数以方便调用

通过标志符给定指针指向的函数

冒泡法排序

升序逻辑判断函数

降序逻辑判断函数

交换函数

运行结果：

```
please input num 0 or 1, 0=descending ,1=ascending:  1
sorted array is:1   2   3   4   5   6   7   8   9   10
```

综合应用

例6.26　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一下操作：
A）调用C库函数的随机函数给数组元素赋0-99之间的随机值；
B）输出显示数组元素；
C）按顺序对每隔5个数求一个和数，并输出显示结果。

问题分析与设计：

设计思路：（1）通过调用包含在头文件stdlib.h之中的随机函数生成随机数，生成0-(x-1)之间的随机数n的方法是：n=rand()%x;

（2）两次输出显示可以采用调用一个函数的方法来实现。

模块划分：随机数产生模块、求和模块、显示模块

数据结构：采用一维数组存储生成的随机数序列和和数序列。

综合应用

函数原型： **（1） void getrand(int *a, int n);**

功能：随机数序列生成

形参：a为指向数组存储空间的指针，n为数组长度

返回：无

（2） void printarr(int *a, int n);

功能：一维数据的输出

形参：a为指向数组的指针，n为数组长度

返回：无

（3）void getsum(int *a,int *b,int n);

功能：完成数据序列的分块求和

形参：a为指向原数组的指针，采用地址传递方式

b为指向求和结果数组的指针

n为原数组的长度

返回：无

```c
#include <stdio.h>
#include <stdlib.h>
/*宏定义以方便修改*/
#define N1 15
#define N2 5
void getrand(int *, int);
void getsum(int *,int *,int);
void printarr(int *,int);
int main( )
{  /*x[N1]保存最初数组数据，w[N1/N2]保存求和后数组数据*/
   int x[N1],w[N1/N2]={0};

   getrand(x,N1);

   printf("the number is:\n");
   printarr(x,N1);

   getsum(x,w,N1);

   printf("the sum number is:\n");
   printarr(w,N1/N2);
   return 0;
}
```

定义数组长度

求和间隔

函数声明

生成随机数组存入x[]

输出x[]显示

对x求和存入w

输出w[]显示

随机数组生成函数

```c
void getrand(int *a,int n)
{
   int i;
   for(i=0;i<n;i++)
      *(a+i)=rand()%100;
}
```

获得0-99随机数

数组显示函数

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

宽度为4、右对齐显示数组

数组分块求和函数

```c
void getsum(int *a,int *b,int n)
{
   int i,j=0,sum=0;
   for(i=0;i<n;i++)
   {
      sum+=*(a+i);
      if((i+1)%5==0)
      {
         b[j]=sum;  sum=0;    j++;
      }
   }
}
```

统计结果初始化为0

以每5个元素为单位进行求和

求和并存入新的数组

本章小结

本章要掌握的内容有：

- 各种类型指针的定义形式与初始化方法和使用指针的方法；
- 指针允许运算的种类和运算方法；
- 如何用指针表示一维数组及二维数组，尤其注意指针和数组表现形式的互换性；
- 多级指针的概念、多级指针处理指针数组的方法；
- 区分数组指针与指针数组以及指针函数与函数指针并掌握其用法。