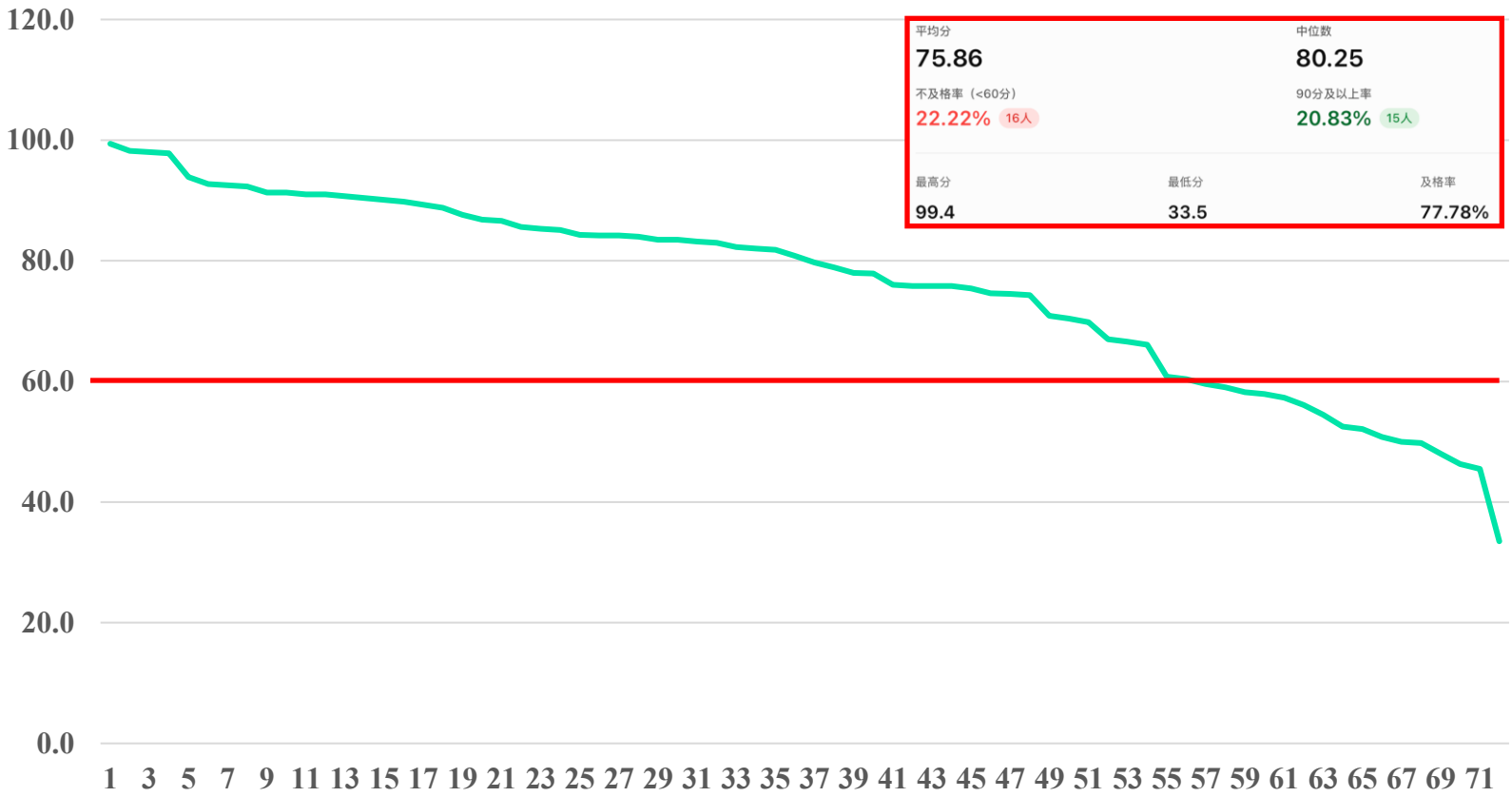


第9讲 指针与引用（Part I）

周水庚

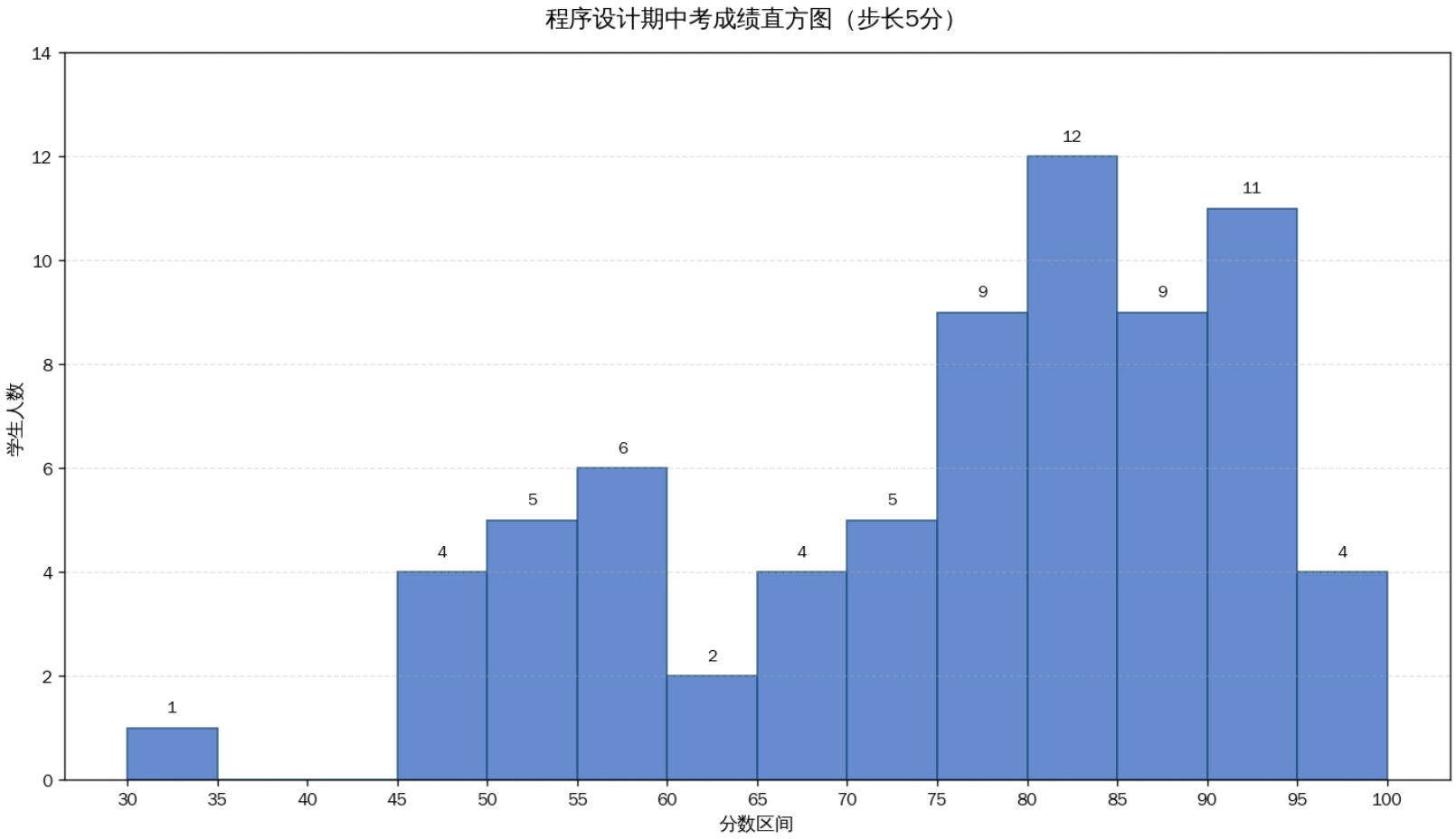
2026年11月19日

期中考试成绩分布图



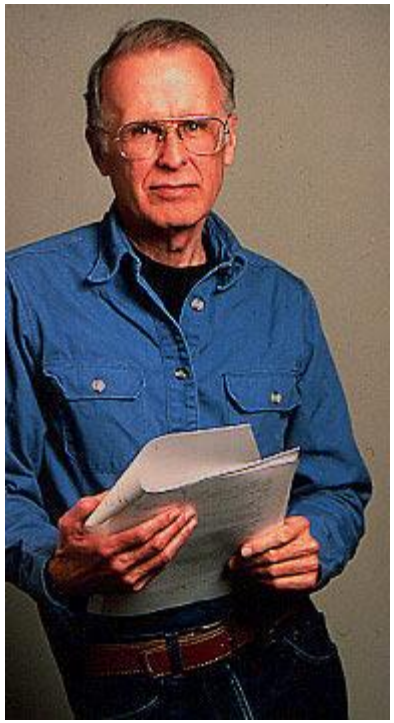
均分：75.9，中位数：80.3。16位同学不及格（22%）

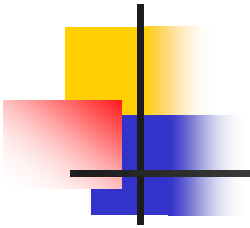
期中考试成绩分布图


程序设计期中考成绩直方图（步长5分）
学生人数
分数区间
0
2
4
6
8
10
12
14
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
1
4
5
6
2
4
5
9
12
9
11
4

John Backus：从“学渣”到图灵奖得主

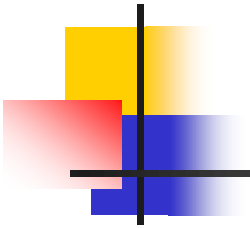
- **约翰·巴克斯**（1924-2007）。出生于美国费城
- 1942年进入弗吉尼亚大学主修化学，**不到一年被开除**；后入伍并学医，1946年退伍；1949、1950年获哥伦比亚大学数学学士、硕士学位，此后一直在IBM工作
  - 1950—1954，担任IBM高级科学部程序员；1954—1959，担任IBM程序研究部经理；1959—1963，担任IBM研究人员；1963年—1991年，担任IBM Fellow；1991年，从IBM退休
- 1974年，当选美国国家科学院院士
- 1977年，获得图灵奖，同年当选美国国家工程院院士
  - “***For profound, influential, and lasting contributions to the design of practical high-level programming systems***”
- 1985年，当选美国艺术与科学院院士
- 主要成就：**FORTRAN语言（第一种高级编程语言）、BNF范式**





提要

- 指针
- 指针和数组



提要

- 指针
- 指针和数组

指针与指针的作用

- 指针是C语言中用于表示程序对象地址的一类数据

- 指针的作用
  - 间接引用它所指的对象
  - 描述数据和数据之间的关系，以便构造复杂的数据结构
  - 利用各种类型指针形参，能使函数增加活力
  - 指针与数组结合，使引用数组元素的形式更加多样、访问数组元素的手段更加灵活
  - 熟练正确应用指针能写特别紧凑高效的程序
  - ……

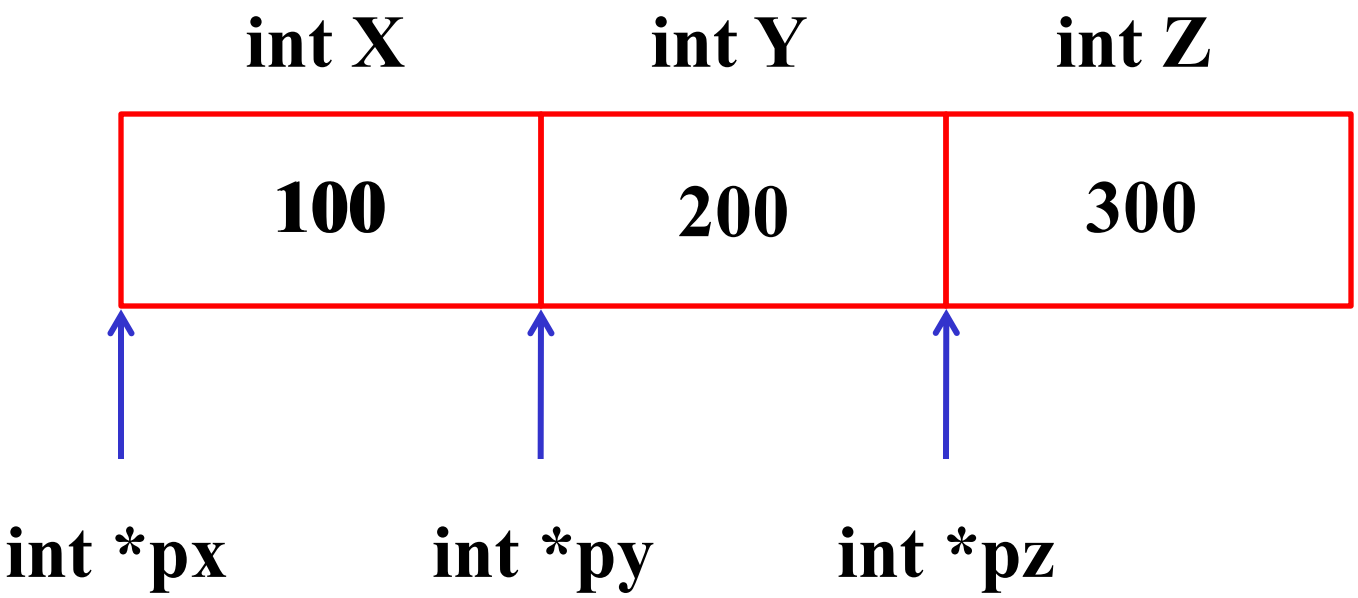
指针类型的历史

- 1940s-1950s：汇编语言直接操作内存地址，是指针思想的源头，但无“类型”概念
- 1960s：CPL/BCPL引入了通过地址间接访问内存的操作符，是 C 指针的直接前身
- 1968，ALGOL 68首次将指针（引用）作为一种正式的、强类型的数据类型引入
- 1970s：C语言继承并完善了指针概念，通过其巨大影响力使其普及和流行
- 同一时期，Pascal也引入了指针类型（用 ^ 符号），进一步推广了这一概念

变量地址与变量内容

- 程序中的变量在内存中占据一定的存储单元，存储单元的开始地址称为变量的地址，在存储单元中存储的数据称为变量的内容
- 数据对象在程序中用变量与其对应，程序用变量定义引入变量、指定变量的类型和名
- 编译系统根据类型确定变量所需的内存空间的字节数量和它的值的表示形式，检查程序对变量操作的合法性，对合法的操作翻译出正确的计算机指令
- 变量的名供程序引用它，程序按名引用变量的内容或变量的地址

变量地址与变量内容



变量地址与变量内容(续)

- 对于代码： int x = 1; x = x + 2
  - “x = x+2;”中的第一个x表示引用变量x的地址;第二个x表示引用变量x的内容
  - 该语句的意义是:完成取x的内容，加上2的计算，并将计算结果存入变量x的地址所对应的单元中
- 在程序执行时，源程序中按名对变量的引用，已被转换成按地址引用，利用变量的地址或取其内容或存储值

指针变量

- 指针变量是指取地址值的变量，用于存放某个变量的地址
- 当指针变量p的值为变量v的地址时，就说指针变量p指向变量v
- 为了便于类型检查，一种指针变量所指向的变量类型在定义指针变量时说明

指针变量定义

- 指针变量定义的一般形式：类型 *标识符;
  - 标识符是指针变量的名
  - 标识符之前的符号“*”:表示该变量是指针变量
  - 最前面的“类型”:该指针变量所指向的程序对象的类型
- 例句：int i, *ip;
  - 定义一个整型变量i和一个指向int型变量的指针变量ip

- 定义指针变量时也可指定初值

  int j; int *intpt = &j;
  - 定义整型指针变量intpt时，初始化它为整型变量j的地址

指针变量及其所指向的变量

- 运算符 &：取变量的地址
  - &x的值就是变量x的地址
- 给定指针变量p和整型变量x，若p=&x，则x是p所指向的变量
- 变量的地址可作为一种值被存储和运算。除能按名引用变量外，也可用变量的地址引用变量
  - 按变量名引用变量称为直接引用
  - 将变量A的地址赋给指针B，借助于指针变量B引用变量A称为对A的间接引用

有关指针的几个概念

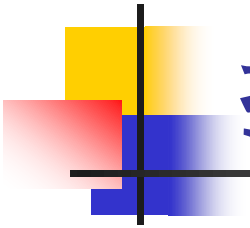
- int i, *ip=&i;
  - 指针类型：int *
  - 指针所指向的类型：int
  - 指针的值或者指针所指向的内存区开始地址
    - &i，即存储i的开始地址
  - 指针所指向的变量的值或者指针所指向的地址空间所存储的值
    - i的值
  - 指针本身存储空间即存储指针所占据的内存区
    - 存储ip所占有的内存空间 ，其地址为&ip

指针类型

- 把指针定义语句里的指针名字去掉，剩下的部分就是这个指针的类型。这是指针本身所具有的类型
- 例如
  - int *ptr; //指针的类型是int *
  - char *ptr; //指针的类型是char *
  - int **ptr; //指针的类型是 int **
  - int (*ptr)[3]; //指针的类型是 int(*)[3]
  - int *(*ptr)[4]; //指针的类型是 int *(*)[4]

指针指向的类型

- 把指针定义语句中的指针名字和名字左边的指针声明符*去掉，剩下的就是指针所指向的类型
- 例如
  - int *ptr; //指针所指向的类型是int
  - char *ptr; //指针所指向的类型是char
  - int **ptr; //指针所指向的类型是int *
  - int (*ptr)[3]; //指针所指向的类型是int()[3]
  - int *(*ptr)[4]; //指针所指向的类型是int *()[4]
- 通过指针来访问指针所指向的内存区时，指针所指向的类型决定了编译器将把那片内存区里的内容当做什么来看待



指针的值

- 指针的值，或者叫指针所指向的内存区地址，是指针本身在内存区的值。这个值将被编译器当作一个地址，而不是一个一般的数值
  - 在32位程序里，所有类型的指针的值都是一个32位整数，因为32位程序里内存地址全都是32位长

指针的值

- 指针所指向的内存区就是从指针的值所代表的那个内存地址开始，长度为sizeof(指针所指向的类型)的一片内存区
  - 我们说一个指针的值是xxxx，就相当于说该指针指向了以xxxx为首地址的一片内存区域
  - 我们说一个指针指向了某块内存区域，就相当于说该指针的值是这块内存区域的首地址
- 指针所指向的内存区和指针所指向的类型是两个完全不同的概念
  - int *p：p所指向的类型已经有了，但由于指针还未初始化，所以它所指向的内存区是不存在的，或者说是无意义的

指针本身所占据的内存区

- 指针本身占了多大的内存，可以用函数sizeof(指针的类型)获得
- 在32位平台里，指针本身占据了4个字节的长度

```
#include <stdio.h>
int main()
{
  char ch,*chp; chp=&ch;
  printf("&ch1=0x%\nchp=0x%x\n",&ch, chp);
}
结果为:
&ch=0x12ff7c
chp=0x12ff7c
```

运算符&和*

- &是取地址运算符；*叫“间接运算符”
- &a的运算结果是一个指针，指针的类型是a的类型加个*，指针所指向的类型是a的类型，指针所指向的地址是a的地址
- *p的结果是p所指向的内容，它的类型是p指向的类型，它所占用空间的开始地址是p所指向的地址

空指针

- 空指针表示 “未分配” 或者“尚未指向任何地方”的指针，用null表示
- 空指针在概念上不同于未初始化的指针。**空指针可以确保不指向任何对象或函数**；而**未初始化指针则可能指向任何地方**
- 每种指针类型都有一个空指针， 而不同类型的空指针的内部表示可能不尽相同。尽管程序员不必知道内部值，但编译器必须时刻明确需要那种空指针，以便在需要的时候加以区分

指针表达式

- 一个表达式的最后结果如果是一个指针，那么这个表达式就叫**指针表达式**
- 下面是一些指针表达式的例子

```
int a,b; int array[10]; int *pa;
pa=&a;//&a是一个指针表达式。
int **ptr=&pa;//&pa也是一个指针表达式。
*ptr=&b;//*ptr和&b都是指针表达式。
pa=array;
pa++;//这也是指针表达式。
```

指针类型转换

- 如果有一个指针p，我们需要把它的类型和所指向的类型改为TYPE *和TYPE，那么语法格式是：(TYPE *)p;
- 这样强制类型转换的结果是一个新指针，该新指针的类型是TYPE *，它指向的类型是TYPE，它指向的地址就是原指针指向的地址。而原来的指针p的一切属性都没有被修改

指针类型转换

- int a=123, b;

  int *ptr=&a;

  char *str;

  b=(int)ptr;//把指针ptr的值当作一个整数取出来

  str=(char *)b;//把这个整数的值当作一个地址赋给指针str

指针的安全问题

- char s='a';

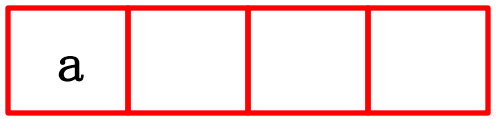
  int *ptr;

  ptr=(int *)&s;

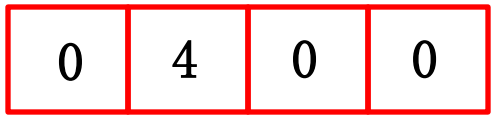
(int *ptr=(int *)&s;)

*ptr=1024;

Before *ptr=1024



After *ptr=1024



指针ptr是一个int*类型的指针，它指向的类型是int，它指向的地址就是s的首地址。在32位程序中，s占一个字节，int类型占四个字节。

最后一条语句不但改变了s所占的一个字节，还把和s相临的高地址方向的三个字节也改变了！

使用指针变量注意事项

- 不能将一个不能指的对象的地址赋给指针变量
  - int i = 100, j, *ip, *intpt; float f, *fp;
  - ip = 100; /*错！指针变量不能赋整数值 */
  - intpt = j; /* 错！指针变量不能赋整型变量的值 */
  - fp = &i; /*错！ float*类型变量，不能指向int型变量 */
  - fp = ip; /*错！不同类型指针变量不能相互赋值 */

使用指针变量注意事项（续）

- 以下都是正确的赋值
  - ip = &i ; /* 使 ip 指向 i */
  - intpt = ip ; /* 使 intpt 指向 ip 所指变量 */
  - fp = &f ; /* 使 fp 指向 f */
  - ip = NULL ; /* 使 ip 不再指向任何变量 */

- 以上例子说明:同类型的指针变量可以相互赋值;只能赋指针变量能指向的程序对象地址给指针变量

使用指针变量注意事项（续）

- 当指针变量明确指向一个对象，它的值不是 NULL 时，可以用运算符“*”，引用指针变量所指向的对象
  - 如 ip = &i; j = *ip;
  - 实现将指针变量ip所指变量的内容(即变量i的内容)赋给变量j。其中，赋值号右边的*ip 表示引用ip所指变量的内容。上述赋值等价于： j = i;
- 语句 *ip = 200;实现向指针变量ip所指变量(即变量i)赋值200。
  - 赋值号左边的*ip表示引用ip所指变量。上述赋值等价于 i = 200;
- int *p, i=10; *p=10; /* 错！P指向未定 */

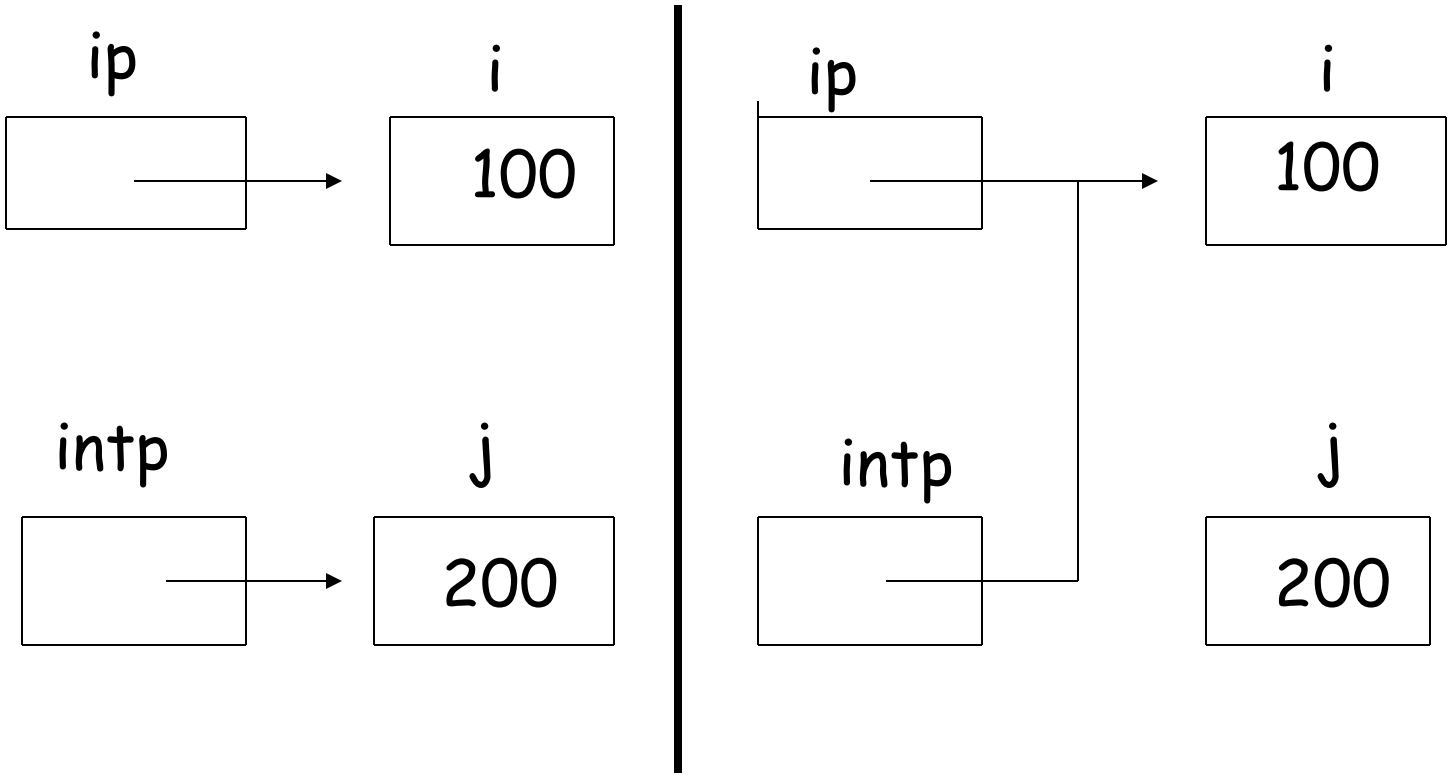
使用指针变量注意事项（续）

- 记号“*指针变量名”与指针变量所指变量的“变量名”等价
  - 如：“ip=&i; intpt = &j; *intpt = *ip+5;”与语句“j = i+5;”等价

- 要特别注意指针变量之间的赋值与指针变量所指向的变量之间的赋值在表示方法上的区别
  - 如 “intpt = ip;”使两个指针变量intpt与ip指向同一个对象，或都不指向任何对象(如果ip的值为NULL)

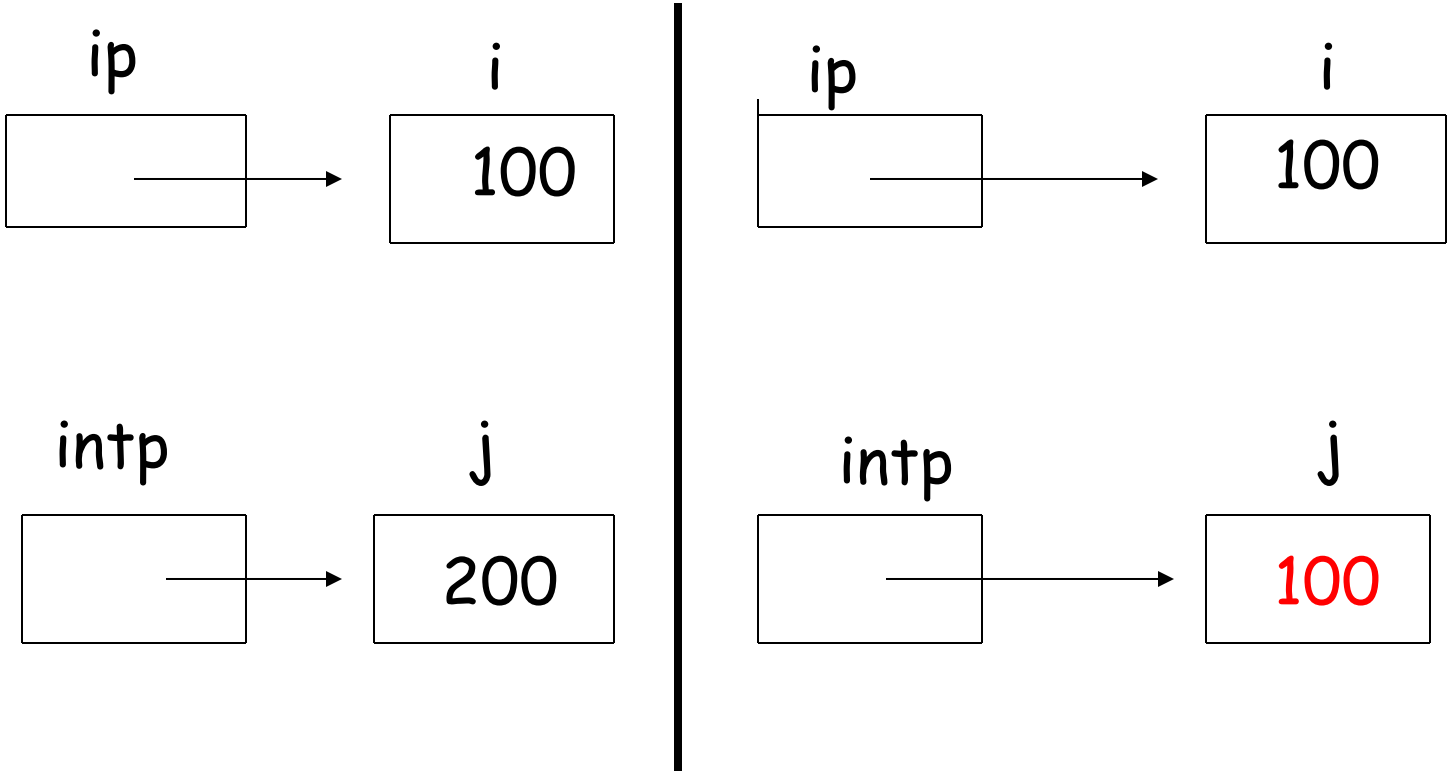
使用指针变量注意事项（续）

- i=100; j=200; ip=&i; intp=&j; intp=ip;



使用指针变量注意事项（续）

- i=100; j=200; ip=&i; intp=&j; *intp=*ip;



使用指针变量注意事项（续）

- 定义指针变量，使它指向某变量，并通过指针变量引用它所的指变量：
  - int a = 100, x = 2, *intp = &a;
  - x += *intp; *intp *= x;
  - 改变指针变量的值，就改变了它的指向，从而实现同样的表示形式能间接引用不同的变量
- 指针变量最主要的应用有两个方面
  - 让指针变量指向数组的元素，以便逐一改变指针变量的指向，遍历数组的全部元素
  - 让函数设置指针形参，让函数体中的代码通过指针形参引用调用环境中的变量

使用指针变量注意事项（续）

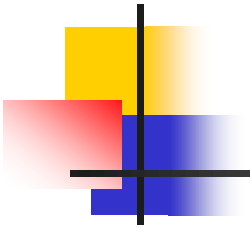
- 指针变量定义与引用指针变量所指对象采用相似的标记形式(* 指针变量名)，但它们的作用与意义是完全不同的
- 在指针变量定义中(如 int *ip;)，指针变量名之前的符号“*”说明其随后的标识符是指针变量名
- 如指针变量定义时带有初始化表达式，如“int i, *ip = &i;”初始化表达式的地址是赋给指针变量本身，而不是指针变量所指对象
  - 在初始化之前，指针变量还未指向任何对象

使用指针变量注意事项（续）

- 通过某个指向变量i的指针变量ip引用变量i与直接按其名i引用变量i，效果是相同的
  - 如有 int i, *ip = &i;则凡变量i能使用的地方，*ip一样能用
- 因单目运算符*、&、++和- -是从右向左结合的。注意分清运算对象是指针变量、还是指针变量所指对象
  - 如有 int i, j, *ip = &i ;
  - 语句 j = ++*ip;相当于*ip=*ip+1; j= *ip;
  - j = *ip++;相当于语句 j = *ip; ip++;因为后缀 ++ 的优先级高于 *

使用指针变量注意事项（续）

- “j = (*ip)++;”不同于 “j = *ip++”
- 前者是先引用ip所指向的对象，取该对象的内容赋给j，并让该对象的内容增加1个单位
- 后者是将原来ip指向的值赋予j，然后ip本身增加了1个单位



提要

- 指针
- 指针与数组

指针与数组的区别

- 指针是与地址相关的类型，它的值是数据存放的位置（地址）；数组则是一系列相同类型的变量
- 数组名对应着（而不是指向）一块内存，其地址与容量在生命期内保持不变，只有数组的内容可以改变。指针可以随时指向任意类型的内存块，它的特征是“可变”，所以常用指针来操作动态内存
- 当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针
- 整数数组a、a[0]和&a的区别
  - 在“地址值”上，a = &a[0]
  - 对数组a的直接引用将产生一个指向数组第一个元素的地址
  - &a的结果则产生一个指向整个数组的地址，相当于一个二级地址的值
  - 单就“地址值”而言，我们有：&a=&a[0]=a

指针与一维数组

- 指向一维数组元素的指针
  - 当指针变量指向一维数组的元素时，就可用指针引用数组的元素
  - 设有以下变量定义
    - int a[100], *p;
    - 赋值运算 p = &a[0]使p指向a[0]
    - 表示&a[0]还有更简洁的方法，即数组名a

指向一维数组元素的指针

- 对指向一维数组元素的指针允许作有限的运算
  - 设有以下代码

```
int *p, *q, a[100];
p = &a[10];
q = &a[50];
```

  - **当两个指针指向同一个数组的元素时，这两个指针可以作关系比较(< ，<=， ==，>， >=， !=)**
    - 若两指针p和q指向同一个数组的元素，则p==q为真表示p，q指向数组的同一个元素；若p<q为真，表示p所指向的数组元素的下标小于q所指向的数组元素的下标。如：对上述代码p<q为真

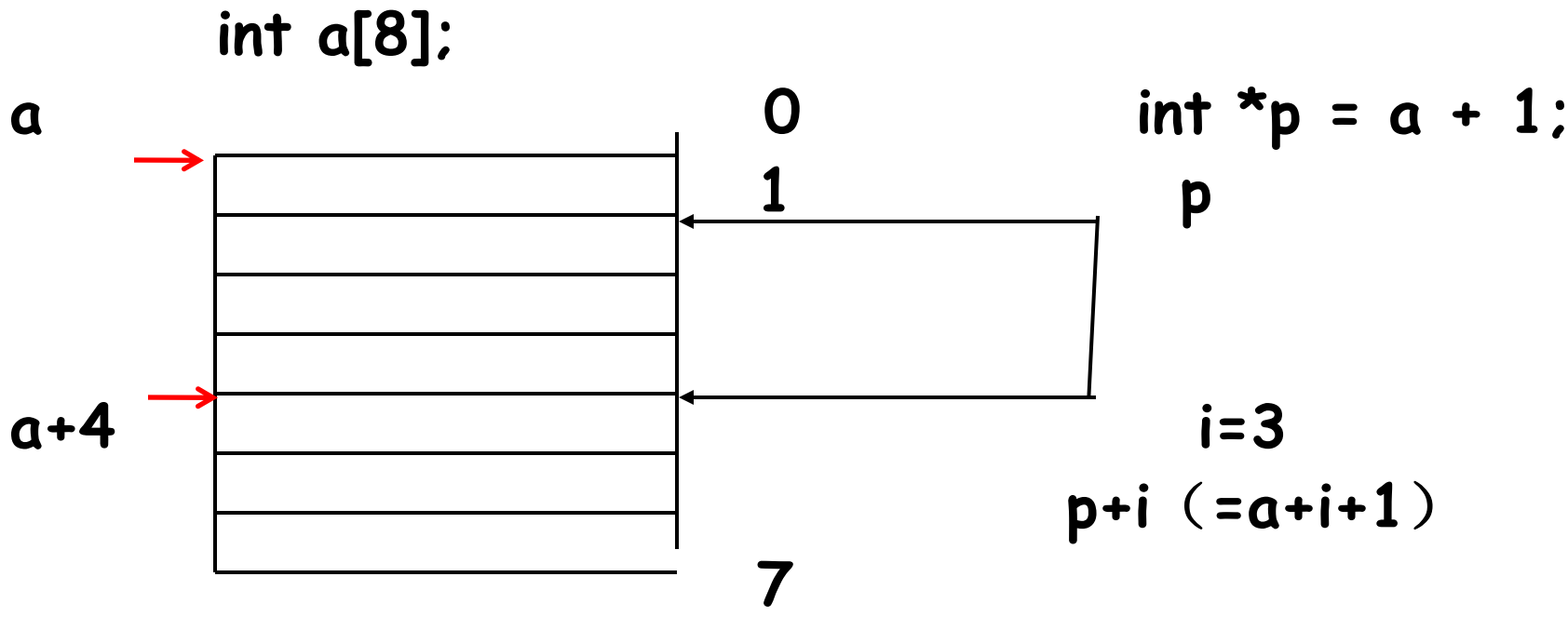
指向一维数组元素的指针(续)

- 对指向一维数组元素的指针允许作有限的运算(续)
  - **指向数组元素的指针可与整数进行加减运算**
    - 由数组元素在内存中顺序连续存放的规定以及地址运算规则，表达式a+1为a[1] 的地址，a+2为a[2]的地址。一般地，表达式a+i为a[i] 的地址。把这个结论应用于指向数组元素的指针，同样成立
    - 若p的值为a[0]的地址，则表达式p+i的值为a[i]的地址。或者说，p+i的值为a[i]的地址值。若p指向数组元素a[10]，则p+n就表示指向数组元素a[10+n]。这里，n是任意的整数表达式
    - **当指针变量指向数组a的元素时，不论数组元素的类型是什么，指针和整数n进行加减运算，指针值向前或向后移动n个元素位置**

指向一维数组元素的指针(续)

- 对指向一维数组元素的指针允许作有限的运算(续)
  - **当两个指针指向同一个数组的元素时，允许两个指针作减法运算，其绝对值等于两指针所指数组元素之间相差的元素个数**
    - 利用运算符 * 可引用指针所指对象，*(a+i)表示引用a+i所指向的数组元素a[i]。这样*(a+i)就是a[i]
    - 对于指向数组元素的指针变量p，若p指向a[10]，*(p+i) 表示引用p+i所指向的数组元素a[10+i]，所以*(p+i)= a[10+i]
    - 与用数组名和下标引用数组元素的标记法相一致，指向数组元素的指针变量也可带下标引用数组的元素，即*(p+i)也可写成p[i]
    - 若p = &a[10]，则p[i]引用的是a[10+i]， p[-2]引用的是a[8]

指向一维数组元素的指针(续)



指向一维数组元素的指针与数组元素位置之间的关系

指向一维数组元素的指针(续)

- **引用一维数组元素有以下三种形式**
  - 用数组元素的下标引用数组元素，如a[5]
  - 利用数组名的值是数组首元素地址的约定，以地址表达式引用表达式所指的元素，如*(a+i)=a[i]
  - 利用指向数组元素的指针变量，用它构成指向数组元素的指针表达式，并用该表达式引用数组元素。如*(p+i)或p[i]
  - **注：** 用数组名a表达数组元素地址与用指向数组元素的指针p来表达数组元素的指针，在实际应用上有区别
    - p是变量，**其值可改变**，如p++；
    - 数组名a只代表数组a的首元素的指针，是不可改变的，**程序只能把它作为常量使用**

指向一维数组元素的指针样例-1

- 利用指针遍历数组，输入生成一个数组和将已知数组复制到另一个数组

```
int a[100], b[100], *p, *q;
for(p = a; p < a + 100;)
    scanf("%d", p++);
for(p = a, q = b; p < a + 100;)
    *q++ = *p++;
```

指向一维数组元素的指针样例-2

- 引用数组元素各种不同方法的示意程序

```
#include <stdio.h>
int a[] = {1, 2, 3, 4};
int main()
{int j, *p;
 for(j = 0; j < 4; j++)
    printf("a[%d]\t=%d\t",j,a[j]);
 printf("\n");
 for(p = a; p <= &a[3]; p++)
    printf("*p\t=%d\t", *p);
 printf("\n");
```

```
for( p=a, j==0; p+j<a+4; j++)
   printf("*(p+%d)\t=%d\t", j, *(p+j));
printf("\n");
for( p=a+3,j=3;j>=0;j--)
   printf("p[-%d]\t=%d\t", j, p[-j]);
printf("\n");
}
```

指向字符串的指针

- 为字符串常量提供存储空间有两种方法
  - 把字符串常量存放在一个字符数组中
    - 例如， char s[] = "I am a string.";
  - 由编译系统将字符串常量与程序中出现的其他常量一起存放在常量存储区中
    - 例如，

```
char *cp1, *cp2 = "I am a string";
cp1 = "Another string";
```

    - 程序为了能访问存于常量存储区中的字符串常量，可用一个字符指针指向它的第一个字符
    - 当字符串常量出现在表达式中时，系统将字符串常量放入常量存储区，而字符串常量转换成指向该字符串常量存储单元的第一个字符的字符指针

指向字符串的指针(续)

- **系统预定义了许多用于字符串处理的库函数，程序可以用字符串常量或指向某字符串的指针调用这些库函数**
  - 如调用库函数strlen() 求一字符串常量的长度

    strlen("I am a string.")
    - 该函数调用的结果是14，表示此字符串常量由14个有效字符组成
  - 如s是一个字符数组，其中已存有字符串；cp是一个字符指针变量，它指向某个字符串的首字符，则
    - 代码：printf("%s\n", s); 输出存于字符数组s中的字符串
    - 代码：printf("%s\n", cp); 输出以字符指针变量cp所指字符为首字符的串字符串

指向字符串的指针示例-1

- 将一个已知字符串复制到一个字符数组，设from为已知字符串的首字符指针，to为存储复制字符串的字符数组首元素的指针
  - 若用下标引用数组元素标记法，完成复制的代码可写成

```
k = 0;
while ((to[k] = from[k]) != '\0')  k++;
```

  - 如采用字符指针描述有

```
while ((*to++ = *from++) != '\0');
```

    - 由于字符串结束符\0的值为0，上述测试当前复制字符不是字符串结束符的代码中，“!='\0' ”是多余的，字符串复制更简洁写法是

```
while (*to++ = *from++);
```

指向字符串的指针示例-2

- **将字符串s中的某种字符去掉，设要去掉的字符与字符变量c中的字符相同**
  - 采用一边考察字符一边复制
  - 引入两个字符指针p和q，p指向当前正考察的字符，q指向下一个用于存储复制字符的位置
  - 若p所指字符与c不相同，则将它复制到新字符串
  - 否则，该字符不被复制
  - 每复制一个字符q才增1，p是每考察一个字符就增1

```
for(p = q = s; *p; p++)
    if (*p != c)  *q++ = *p; /* 复制 */
*q= '\0';    /* 重新构成字符串 */
```

指向二维数组的指针

- 设二维数组为

```
int a[3][4] = {{1,  2,  3,  4},
               {5,  6,  7,  8},
               {9, 10, 11, 12}};
```

  - 这里，数组a有3行4列。按行来看数组a，数组a有三个元素，分别为a[0]，a[1]，a[2]。它们又分别是一个一维数组，各有4个元素
  - 例如，a[0]所代表的一维数组为

    a[0][0]、a[0][1]、a[0][2]、a[0][3]

指向二维数组的指针(续)

- 与一维数组名可看作数组的第一个元素(下标为 0)的地址的约定相一致，二维数组名a可以看作a的首元素a[0]的地址，即表示二维数组第一行的首地址。一般地，a+i可以看作数组a的元素a[i]的地址，即二维数组第i+1行的首地址。所以，a+i=&a[i]；即*(a+i)=a[i]

- 因二维数组a能用a[0]，a[1]，a[2]分别表示其中的一维数组，所以a[0]能表示用a[0]表示的一维数组的首元素a[0][0]的地址；a[1]能表示用a[1]表示的一维数组的首元素a[1][0]的地址。一般地，a[i]能表示用a[i]表示的一维数组的首元素a[i][0]的地址。所以， a[i]=&a[i][0]; a[i]+j=&a[i][j]；即*a[i]=a[i][0]; *(a[i]+j)=a[i][j]

- 综上，a和a+i就是地址的地址。a相当于二级地址
  - *(a+i)=a[i]; *(a[i]+j)=a[i][j] ; *(*(a+i)+j)=a[i][j]; **a=a[0][0]

指向二维数组的指针(续)

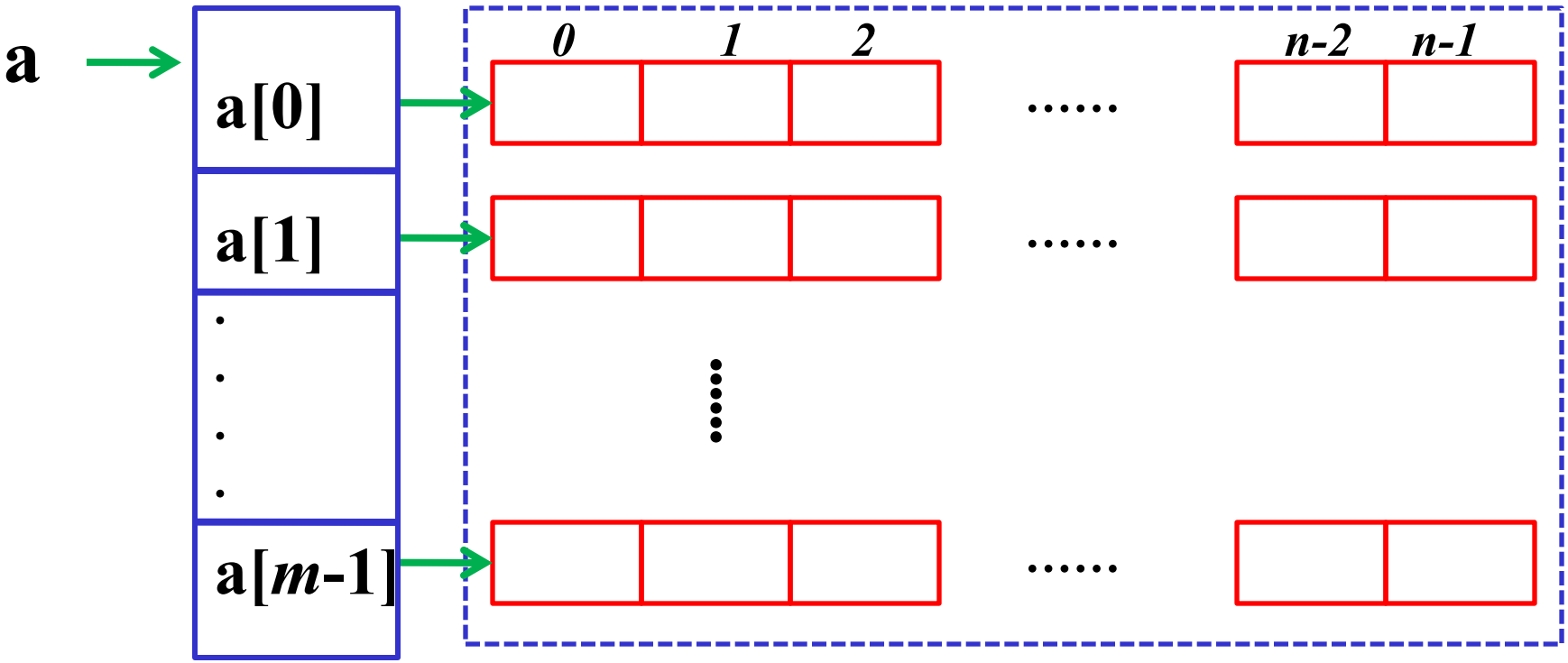
- a[i]表示用a[i]表示的一维数组的首元素a[i][0]的地址
- 因a[i]可写成*(a+i)，a[i]或*(a+i)表示二维数组a的元素a[i][0]的地址，即&a[i][0]。
- 根据地址运算规则，a[i]+j即代表数组a的元素a[i][j]的地址，即&a[i][j]。因a[i]与 *(a+i)等价，所以*(a+i)+j也与&a[i][j]等价

指向二维数组的指针(续)

- **指针变量指向二维数组中一整行（性质续）**
  - 数组元素a[i][j]有以下三种等价表示形式：
    (a[i]+j)、(*(a+i)+j)、(*(a+i))[j]
  - 特别地，a[0][0]等价表示形式有： *a[0] 和 **a
  - 数组元素a[i][j]的地址也有三种等价表示形式
    a[i]+j、*(a+i)+j、&a[i][j]

二维数组与指针图示

int a[*m*][*n*];



数组指针：指向数组的指针

- 定义指向数组的指针变量，如：int (*p)[4];
  - 定义指针变量p能指向一个由四个int型元素组成的数组
  - 以上定义中，圆括号是必需的，否则 int *p[4]; 定义为一个指针数组p，即p有四个元素，每个元素是一个指向整型变量的指针
  - p是一个指向由四个整型元素组成的数组，对p作增减1运算，就表示向前进或向后退四个整型元素
- 如有变量定义： int a[3][4], (*p)[4];
  - p = a ;使p指向二维数组a的第1行
  - p+1的指针值为指向二维数组a的第二行
  - 若p = a，则p+i指向二维数组a的第i+1行，与a+i一样
  - 同二维数组元素的地址计算规则相对应，*p+j指向a[0][j]; *(p+i)+j，或者p[i]+j指向数组a的元素a[i][j], 即 *(p+i)+j=p[i]+j=&a[i][j]
  - 所以，有*(*(p+i)+j)=*(p[i]+j)=p[i][j]=a[i][j]

数组指针：指向数组的指针（示例）

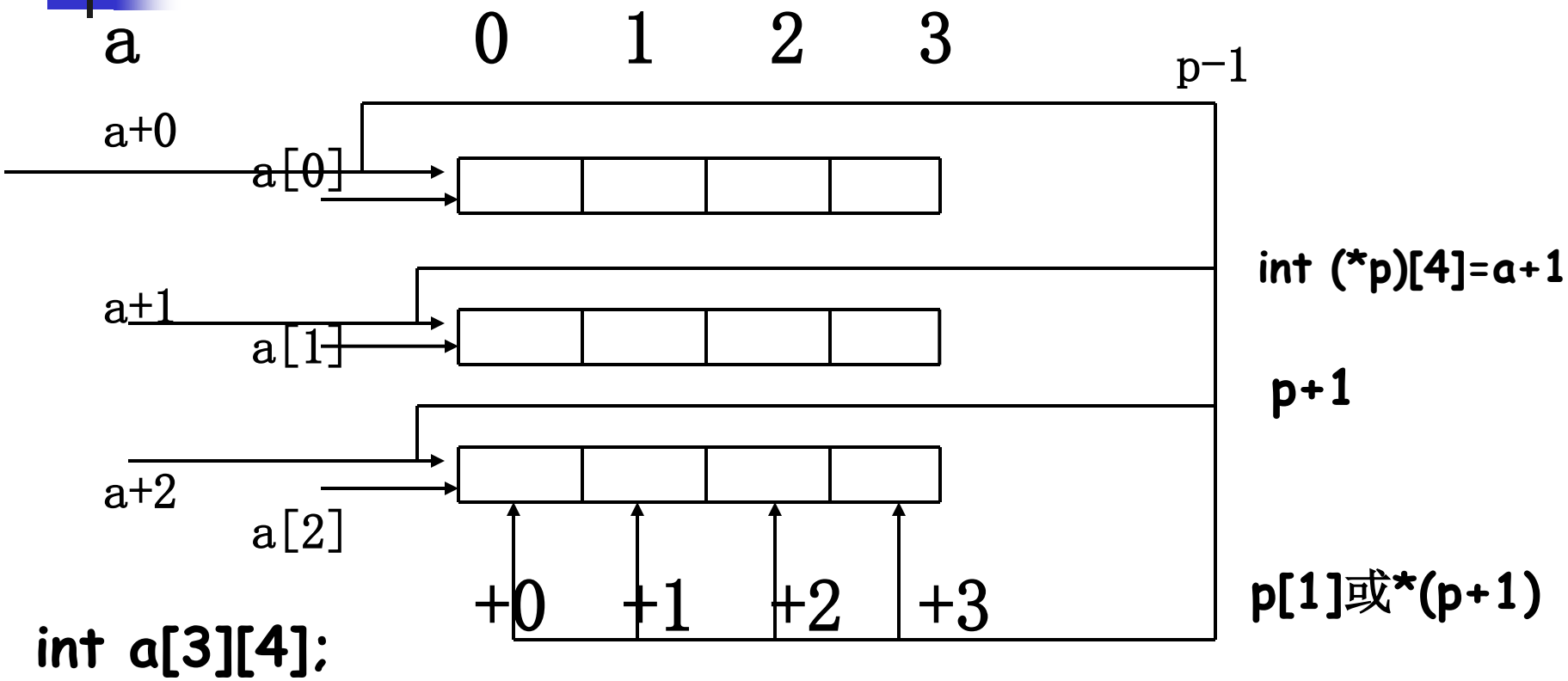
- **指向数组元素的指针和指向数组的指针的区别示意程序**

```
#include <stdio.h>
int main()
{ int a[3][4] = { { 1,  3,  5,  7},
                  { 9, 11, 13, 15},
                  {17, 19, 21, 23}};
  int i, *ip, (*p)[4];
  p  = a+1;    ip = p[0];
   /* *(p+i)=p[i] */
  for(i = 1; i <= 4; ip += 2, i++)
      printf("%d\t", *ip);
  printf("\n");
  p = a;
  for (i = 0; i < 2; p++, i++)
      printf("%d\t", *(*(p+i)+1));
  printf("\n");
}
```

程序运行后，将输出

```
9       13      17      21
3       19
```

二维数组与一维数组指针图示



二维数组名和指向一维数组的指针及数组元素位置之间的关系

指针数组

- 当数组元素类型为某种指针类型时，该数组就称为指针数组
- 指针数组的定义形式为

  类型说明符 * 数组名[常量表达式];

  - 常量表达式是一个整型的，是数组的长度说明，用于指明数组元素的个数；类型说明符指明指针数组的元素指针指向的对象的类型。数组名之前的“*”是必需的，由于它出现在数组名之前，使该数组成为指针数组
  - 例如， int *p[10];
    - 定义指针数组p的每个元素都能指向int型数据的指针变量，有十个元素:p[0]、p[1]、…、p[9]。和一般的数组定义一样，数组名p也可作为p[0]的地址

指针数组(续)

- 在指针数组的定义形式中，由于“[]”比“*”的优先级高，使数组名先与“[]”结合，形成数组，然后再与数组名之前的“*”结合，表示数组元素是指针类型
  - 注意，在“*”与数组名之外不能加上圆括号，否则变成指向数组的指针变量
  - 如，int (*q)[10];是定义指向由10个int型元素组成的数组的指针
- 引入指针数组的主要目的是为了便于统一管理同类的指针
  - 如利用指针数组能实现对一组独立的指针变量以数组的形式对它们作统一处理

指针数组(续)

- 有以下定义

```
int a, b, c, d, e, f;
int *apt[] = {&a, &b, &c, &d, &e, &f};
```

  - 下面循环语句能顺序访问独立变量 a、b、c、d、e、f

```
for(k = 0; k < 6; k++)
    printf("%d\t", *apt[k]);
/* 其中*apt[k]可写成**(apt+k) */
```

- 下面的两个程序实现将一组独立的变量输入它们的值，排序后输出。前一个程序排序时交换变量的值，后一个程序排序时交换它们的指针

指针数组示例-1

- 排序时交换变量值

```
#include <stdio.h>
#define N sizeof ap/sizeof ap[0]
int a, b, c, d, e, f;
int main()
{ int *ap[] = {&a, &b, &c, &d, &e, &f};
  int k, j, t;
  printf("Enter a, b, c, d, e, f.\n");
  for(k = 0; k < N; k++)
     scanf("%d",ap[k]);
     /*scanf("%d",*(ap+k))*/
```

```
 for(k = 1; k < N; k++)
   for(j = 0; j < N-k; j++)
     if (*ap[j] > *ap[j+1]) {
        t =*ap[j]; /*交换变量的值*/
        *ap[j] = *ap[j+1];
        *ap[j+1] = t;
      }
 for(k = 0; k < N; k++)
    printf("%d\t", *ap[k]);
 printf("\n\n");
}
```

指针数组示例-2

- 排序时不交换变量的值，而是交换它们的指针

```
#include <stdio.h>
#define N sizeof ap/sizeof ap[0]
int a, b, c, d, e, f;
void main()
{ int *ap[] = {&a, &b, &c, &d, &e, &f};
   int k, j, *t;
   printf("Enter a, b, c, d, e, f.\n");
   for(k = 0; k < N; k++)
      scanf("%d", ap[k]);
```

```
 for(k = 1; k < N; k++)
    for(j = 0; j < N-k; j++)
       if (*ap[j] > *ap[j+1]) {
          t = ap[j];/*交换变量的指针*/
          ap[j] = ap[j+1];
          ap[j+1] = t;
        }
 for(k = 0; k < N; k++)
    printf("%d\t", *ap[k]);
 printf("\n\n");
}
```

指针数组示例-3

- 从文件输入字符行，对输入的字符行排序后输出到另一文件中。设文件字符行的行数不超过500行，每行字符不超过80个字符

```
#include <stdio.h>
#include <string.h>
#define LINES    500
#define Nline    80
FILE *fp;
char rfname[40], wfname[40];
char lines[LINES][Nline+1];  char *lptr[LINES];
void main()
{ int n, i, j;  char *t;
  printf("输入文件名？ \n ");  scanf(" %s%*c ", rfname);
  if ((fp = fopen(rfname, " r ")) == NULL) {
      printf("不能打开输入文件 %s\n ", rfname);  return ;
   }
```

指针数组示例-3(续)

从文件输入字符行，对输入字符行排序后输出到另一文件(续)

```
n = 0;
while(n<LINES && fgets(lines[n], Nline, fp) != NULL)
    n++; /* fgets从文件输入字符行，存储到数组中 */
fclose(fp);
/* 将输入的诸字符串的首字符指针置入指针数组 */
for(i = 0; i < n; i++)
   lptr[i] = *(lines + i);  /*  &lines[i][0] */
for(j = 1; j < n; j++)
   for(i = 0; i < n-j; i++)
      if (strcmp(lptr[i], lptr[i+1]) > 0)  {
         t = lptr[i]; lptr[i] = lptr[i+1];
         lptr[i+1] = t;
    }
 printf("输出文件名？ \n ");
 scanf(" %s%*c ", wfname);
 fp = fopen(wfname, "w");
 for(i = 0; i < n; i++)
    fprintf(fp, " %s ", lptr[i]);
 fclose(fp);
 printf("\n\n\n");
 for(i = 0; i < n; i++)
   printf("%s ", lptr[i]);
 printf("\n\n\n");
}
```

指针数组(续)

- 当指针数组的元素分别指向二维数组各行首元素时，也可用指针数组引用两维数组的元素
- 以下代码说明指针数组引用两维数组元素的方法

```
int a[10][20], i;
int *b[10];
for(i = 0; i < 10; i++)
    b[i] = &a[i][0]; /* b[i]指向数组元素a[i][0] */
```

  - 则表达式a[i][j]与表达式b[i][j]引用同一个元素，即从指针数组方向来看，因b[i]指向元素a[i][0]，*(b[i]+j)或b[i][j]引用元素a[i][j]
    - b[i][j]=*(b[i]+j)=*(&a[i][0]+j)=*(&a[i][j])=a[i][j]

指针数组(续)

- 当指针数组的元素指向不同的一维数组的元素时，也可通过指针数组，把它们当作两维数组那样来引用
- 如以下代码所示

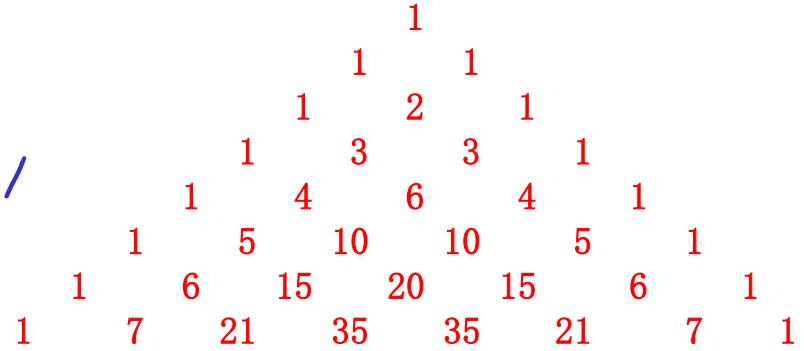
```
char   w0[ ] = "Sunday",
       w1[ ] = "Monday",
       w2[ ] = "Tuesday",
       w3[ ] = "Wednesday",
       w4[ ] = "Thursday",
       w5[ ] = "Friday",
       w6[ ] = "Saturday";
  char *wName[] = {w0,w1, w2, w3, w4, w5, w6};
```

  - 则语句　　for(i = 0; i <= 6; i++)
　　printf("%s\n", wName[i]);
  - 输出星期英文名称。wName[2][4]引用字符w2[4]，其值为' d'

指针数组示例

以下例子程序把一维数组分割成不等长的段，从指针数组方向来看，把它当作两维数组来处理

程序产生如下形式的二项式的系数三角形




```
#include <stdio.h>
#define  N  8
int p[N*(N+1)/2], i, j, *pt[N];
void main()
 {  for(pt[0] = p, i = 1; i < N; i++)
        pt[i] = pt[i-1] + i; /* 确定指针关系 */
    for(i = 0; i < N; i++) {
        pt[i][0] = pt[i][i] = 1;
        /* 左右两边赋值 */
        for(j = 1; j < i; j++)
           pt[i][j] = pt[i-1][j-1] + pt[i-1][j]; /* 确定中间值 */
     }
    for(i = 0; i < N; i++)  { /* 打印值 */
        printf("%*c", 40-2*i, ' ');
        for(j = 0; j <= i; j++)  printf("%4d", pt[i][j]);
         printf("\n");
      }
 }
```

二级指针

- 当指针变量pp所指变量ip是一种指针时，pp是一种指向指针的指针，称指针变量pp是一种2级指针
- **定义指向指针变量的指针变量的一般形式为**

  类型说明符 ** 变量名；

  - 首先定义变量为指针变量，其次是该变量能指向一种指针对象，最后是被指向的指针对象能指向的对象的类型
  - 例如：`int **pp, *ip, i;`

    `ip = &i;  pp = &ip;`
  - 定义说明pp是指向指针的指针变量；它指向的是这样一种指针对象，该指针对象是指向int型变量

二级指针(续)

- 二级指针与指针数组有密切的关系
- 若有指针数组

  char *lines[ ] = {"ADA","ALGOL","C","C++","FORTRAN","PASCAL"};

  - 则lines指针数组的每个元素分别指向以上字符串常量的首字符。在这里数组名lines可以作为它的首元素lines[0]的指针，lines+k是指向元素lines[k]的指针
  - lines[k]也是指针，表达式lines+k的值是一种**指针的指针**
  - 如有必要还可引入指针变量cp，让它指向数组lines的某元素，如cp = &lines[k]。这样，cp就是指向指针型数据的指针变量。在这里，cp是指向字符指针的指针变量，它应被定义成　char **cp;

二级指针(续)

- 对于**指向字符指针的指针变量**的定义 char **cp;
  - 为了定义这样的cp，它的前面有**两个*号**
  - 由于*自右向左结合，首先是“*cp”表示cp是指针变量，再有**cp表示cp指向的是某种指针类型，最后“char **cp”表示指针变量cp指向指向字符型数据的指针
  - 针对前一个例子：如果有赋值cp = &lines[1]，让它指向数组元素lines[1]，则*cp的表示引用lines[1]，它也是一个指针，指向字符串“ALGOL”的首字符。如引用指针*cp所指内容，**cp表示引用lines[1][0]，其值是字符'A'

二级指针示例

- 下面代码实现顺序输出指针数组lines各元素所指字符串

```
for(cp = lines; cp < lines+6; cp++)
    printf("%s\n", *cp);
```

- 下面代码是采用”%c“格式，逐一输出字符串字符，实现顺序输出指针数组lines各元素所指的字符串

```
for(i = 0; i < 6; i++) {  /* 设i 和j是int类型 */
    for(j = 0; lines[i][j] != '\0'; j++)
        printf("%c", lines[i][j]);
    printf("\n");
}
```

二级指针示例(续)

- 设有数组a[]和指针数组pt[]有以下代码所示的关系

```
int a[ ] = { 2, 4, 6, 8, 10};
int *pt[ ] = { &a[3], &a[2], &a[4], &a[0], &a[1]};
int **p;
```

- 下面的代码利用指针数组pt[]和指针的指针p，遍历数组a[]

```
for(p = pt; p < pt + 5; p++)
    printf("%d\t", **p);
```

- 上例说明指针的指针与指针数组有密切关系，**指向指针数组元素的指针即为指针的指针**，如以上程序中的指针变量p
- 上述代码首先让它指向指针数组的首元素，然后循环让它顺序遍历指向指针数组的各元素，标记*p能引用p所指的数组元素，**p能引用p所指数组元素所指的变量。程序中用**p访问数组a[]的元素

本讲小结

- 指针
- 指针和数组
  - 指向数组元素的指针、指向字符串的指针
  - 指向数组的指针、指针数组、二级指针