

指 针+程序设计讲解

Wang Houfeng

EECS, PKU

Wanghf@pku.edu.cn

内容

➤ 指针相关知识

- 指针与二维数组
- 程序设计讲解

指针与地址的基本概念

- 假定有：

```
int num, *iPtr=&num;
```

- `iPtr` //取地址（`num`的地址）
- `num` //取值
- `*num` **//错误**
- `*iPtr` //取地址对应的存储单元中的值（`num`值）
- `&iPtr` //取地址变量的地址（`iPtr` 变量的地址）
- `&num` //取变量的地址（`num`的地址）
- `&*iPtr`与`&num`相同, `&*`的作用相抵消

指出如下程序的问题

```
char s[] = "this is a string";
```

```
for (; *s != '\0'; s++)
```

```
    cout << *s << " ";
```

错误，数组名是常量

因此s不允许变动

```
char *p; p=s; //把for循环中s改为 p
```

指针可以移动

- 或者

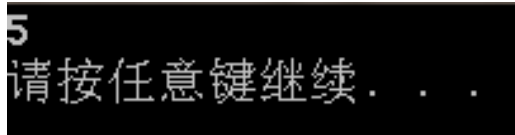
```
char *s = "this is a string";
```

阅读程序

- 输出什么？

```
int f(char *s)
{
    char *p = s;
    while(*p) p++;  相当于返回串的长度
    return p-s;
}

int main()
{
    char *a = "abcde";  字符串就是以0结束
    cout << f(a)<<endl;
    return 0;
}
```



5
请按任意键继续. . .

阅读程序

- 输出什么？

```
976531
请按任意键继续 . . .
```

```
void f(char *a, char c)
{
    while (*(a++) != '\0') ;
    while(*(a - 1) < c)
        *(a--) = *(a - 1);
    *a = c;
}

int main()
{
    char s[10] = "97531";
    f(s, '6');
    cout << s << endl;
    return 0;
}
```

常量类型指针

- 指向常量的指针（不能通过 p 改变它所指向的内容）

`const int *p;` // 注意 `const` 的位置在最前面

`const int a=15;`

`int b;`

`p = &a;` ✓ // 即可以指

`p = &b;` ✓ // 又可以指

`*p = 18;` ✗ // 不能通过

`b=*p;` ✓ // 但可以取

`b=18;` ✓ // b是被指向的变量，可以给它赋值

定义指向常量的指针

只限制指针的间接赋值操作，而不能限制指针指向的量的自身的操作特性。

即仅仅 `*p=...` ✗

常量类型指针

- 指针常量（所指的地址不能变）

```
int a,b
```

```
int * const p = &a; // p 是指针常量，固定指向 a  
// 注意 const 的位置
```

```
*p = 15; ✓ //通过 p间接访问  
p = &b; ✗ //不可以给指针  
a = 8; ✓ //直接访问变量 a  
b=*p; ✓ // 可以取出它所指
```

定义指针常量，
同时必须给该指针赋初值，
不能改变指针的指向。
即 p=... ✗

常量类型指针

- 指向常量的指针常量（地址和值均不能变）

```
int a,b
```

```
const int *const p = &a;    // 注意2个 const
```

```
*p = 15; ✗ //不能改变
```

```
p = &b; ✗ //不可以给
```

```
a = 8; ✓ //直接访问变量
```

```
b=*p; ✓ // 可以取出它用
```

p 为指向常量的指针常量，

即 **p=...** ✗

***p=...** ✗

const 类型指针小结

```
int a, b;
```

(1) 指向常量的指针 (不能改变所指内容)

```
const int *p;
```

*p=... ✗

(2) 指针常量 (不能改变所指地址)

```
int * const p = &a; // p 是指针常量, 固定指向 a, 必须赋初始值
```

p=... ✗

(3) 指向常量的指针常量 (地址和值均不能改变)

```
const int * const p = &a; // 注意有两个 const
```

即 p=... ✗

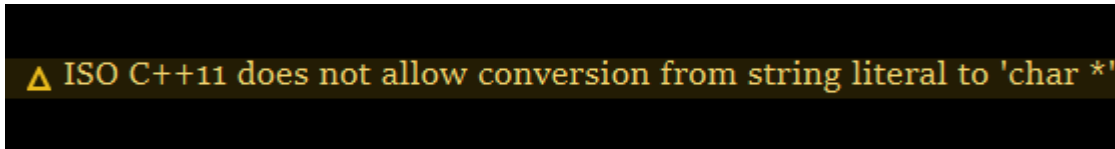
*p=... ✗

再看字符指针赋常量值

```
char *p="Fortran"; // 或char *p; *p="Fortran";
```

基本原理是为字符串常量“Fortran”分配内存空间，然后将其首地址给指针 p

注意：在C语言下没问题。但在有些 C++下会报错（如 VC）：



Δ ISO C++11 does not allow conversion from string literal to 'char *'

如果前面加上 const 则可以：

```
const char *p="Fortran"; // 或char *p; *p="Fortran";
```

其它C++可以查阅相关资料

使用指向常量的指针的意义

```
#include <iostream.h>

void my_strcpy(char * dest, const char *source)
{
    while( *dest++ = *source ++);
}

void main( )
{
    char a[20] = "How are you! ";
    char b[20];
    my_strcpy(b, a);
    cout << b << endl;
}
```

如果误写 *source =
则编译时报错。

在被调函数中，
不能通过指针改变
源字符串的内容。
即保护源串内容。

定义const类型指针的主要目的是提高程序的安全性。

思考题

- 指出下列2个函数各自的功能

```
void mystry1(char *s1, const char *s2)
```

```
{
```

```
    while(*s1 != '\0')
```

```
        ++s1;
```

```
    for (; *s2 != '\0'; s1++, s2++) *s1 = *s2;
```

```
}
```

```
int mystry2(const char *s)
```

```
{
```

```
    for (int i = 0; *s != '\0'; s++)
```

```
        ++i;
```

```
    return i;
```

```
}
```

指向常量的指针地址
可以变，值不能变



指向指针的指针

- 定义方式

- `char c = 'a';`
- `char *q = &c;`
- `char **p = &q;`

c 的值是什么?

q 的值是什么? c的地址

***q 的值是什么?** c的内容a

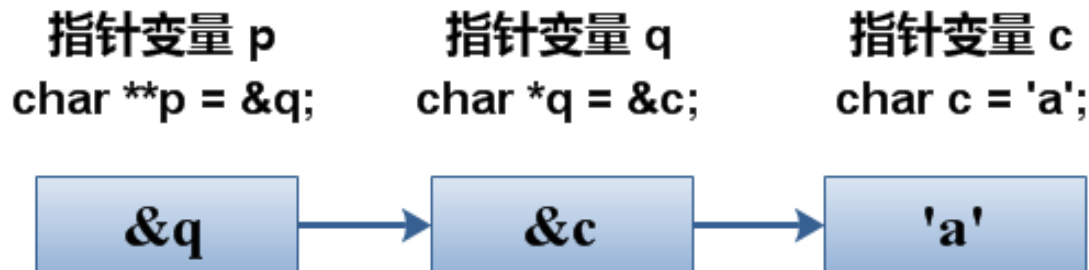
p 的值是什么? q的地址

***p 的值是什么?** c的地址

****p 的值是什么?** c的内容a

定义一个:

指向“指向char型数据的指针变量”的指针变量



指针数组

- 指针数组的特点：
 - 多个元素构成数组
 - 数组中的每个元素为指针变量
 - 常用于处理多个一维数组和多个字符串

定义形式：[**存储类型**] **数据类型** ***数组名**[**数组长度说明**];

例 int *p[4];

指针所指向变量的数据类型

int *p[4]表示有 4 个指针指向整型变量

指针数组的初始化和赋值

赋值:

```
main()
```

```
{  char a[]="Fortran";
```

```
    char b[]="Lisp";
```

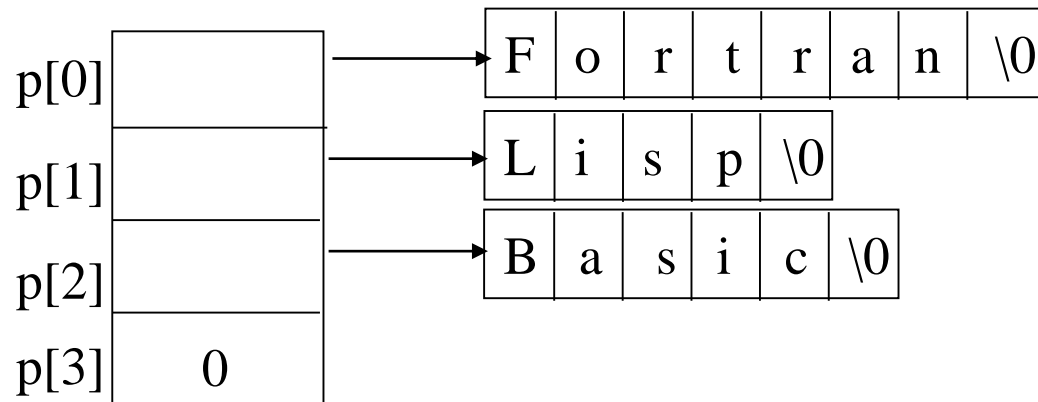
```
    char c[]="Basic";
```

```
    char *p[4];
```

```
    p[0]=a; p[1]=b; p[2]=c; p[3]=NULL;
```

```
    .....
```

```
}
```



内容

- 指针相关知识
 - 指针与二维数组
- 程序设计讲解

指针与二维数组

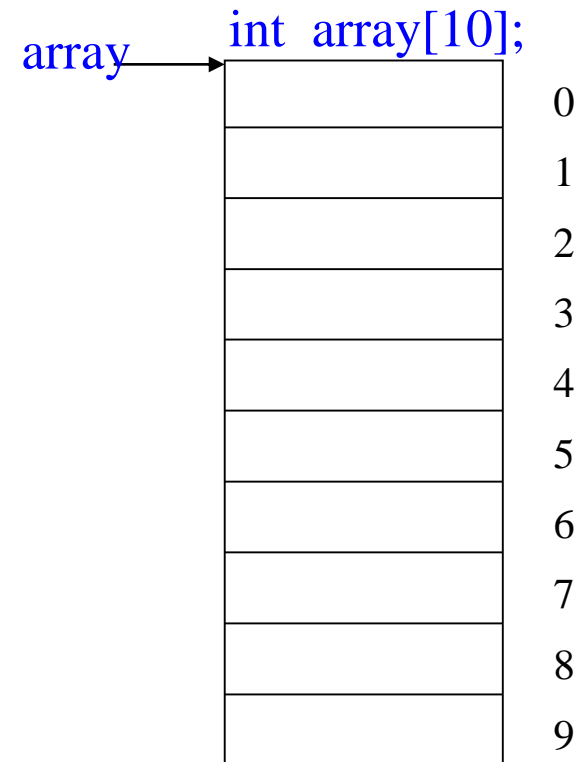
- 数组与地址(一维数组为例)

(1) 数组名array表示数组的首地址，即array[0]的地址；

(2) 数组名array是地址**常量**

(3) **array+i是元素array[i]的地址**

(4) $\text{array}[i] \Leftrightarrow *(\text{array}+i)$

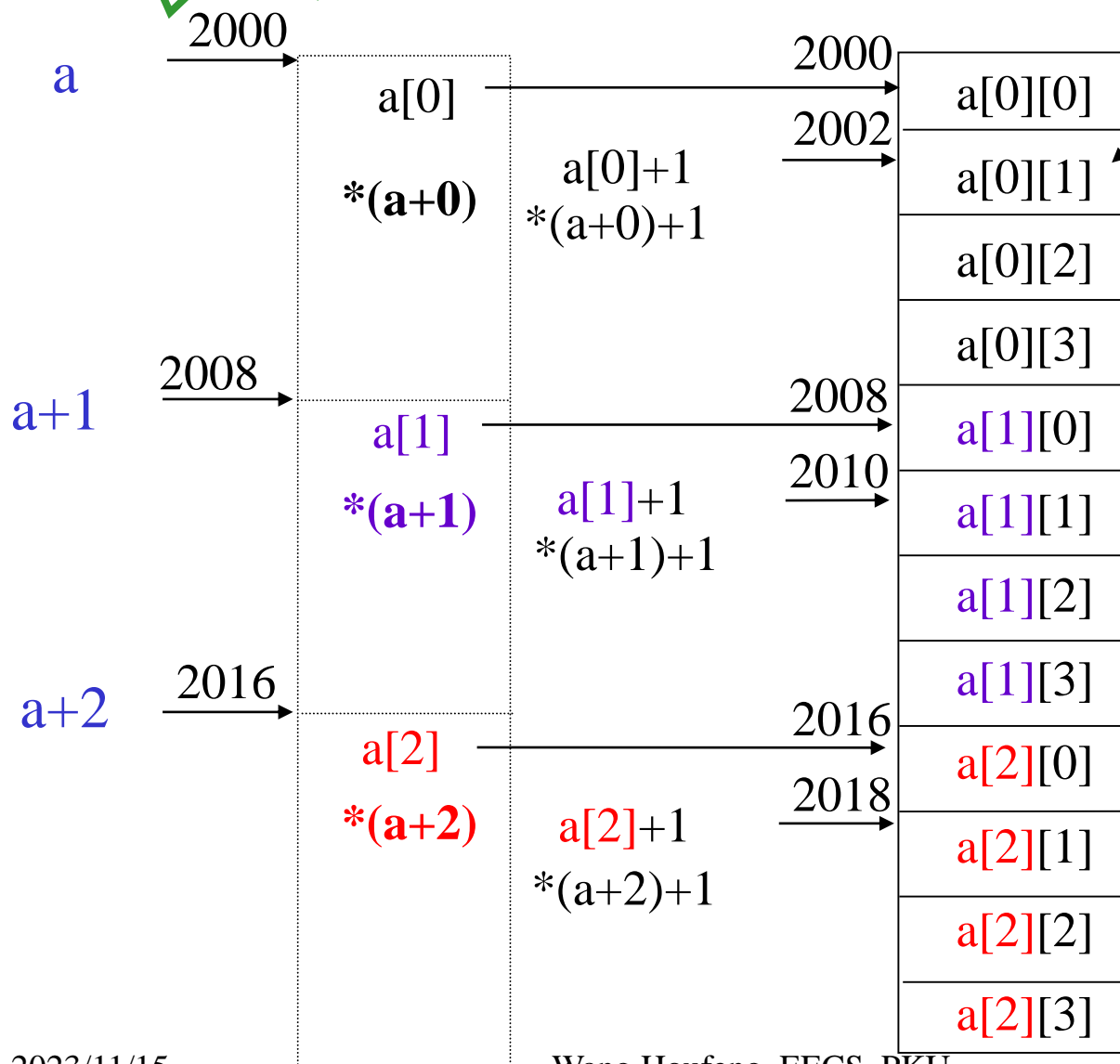


行指针与列指针

int a[3][4];

$*(a[0]+1)$

$*(*(a+0)+1)$



对于二维数组:

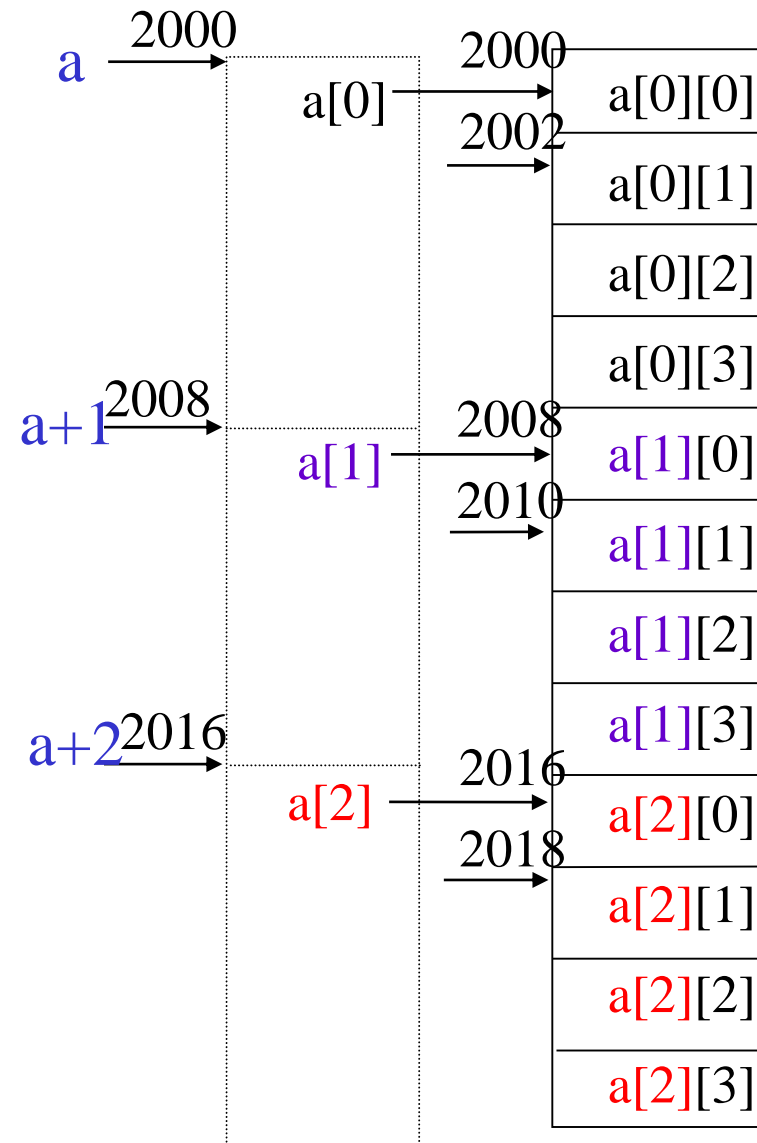
(1) `a`是数组名,
包含三个元素
`a[0]`,`a[1]`,`a[2]`

(2) 每个元素`a[i]`
又是一个一维
数组, 包含4个
元素

$**(\mathbf{a+2})$

$*(*(\mathbf{a+2})+1)$

`int a[3][4];`



– 对二维数组 `int a[3][4]`,有

- » `a`-----二维数组的首地址，即第0行的首地址
- » `a+i`-----第*i*行的首地址
- » `a[i] ⇔ *(a+i)`-----第*i*行第0列的元素地址
- » `a[i]+j ⇔ *(a+i)+j` -----第*i*行第*j*列的元素地址
- » `*(a[i]+j) ⇔ *(*a+i)+j ⇔ a[i][j]`

– `a+i=&a[i]=a[i]=*(a+i) =&a[i][0]`,
值相等，含义不同

- `a+i ⇔ &a[i]`,表示第*i*行首地址，指向行
- `a[i] ⇔ *(a+i) ⇔ &a[i][0]`，表示第*i*行第0列元素地址，指向列

int a[3][4];

a[0][0]
a[0][1]
a[0][2]
a[0][3]
a[1][0]
a[1][1]
a[1][2]
a[1][3]
a[2][0]
a[2][1]
a[2][2]
a[2][3]

地址表示:

- (1) a+1
- (2) &a[1][0]
- (3) a[1]
- (4) *(a+1)
- (5) (int *) (a+1)

← 行指针

} 列指针

地址表示:

- (1) &a[1][2]
- (2) a[1]+2
- (3) *(a+1)+2
- (4) &a[0][0]+1*4+2

二维数组元素表示形式:

- (1) a[1][2]
- (2) *(a[1]+2)
- (3) *(* (a+1)+2)
- (4) *(&a[0][0]+1*4+2)

指向二维数组

```
main()
{
    static int a[3][4]={ 1,3,5,7,9,11,13,15,17,19,21,23};
    int *p;
    for(p=a[0];p<a[0]+12;p++)
    {
        if((p-a[0])%4==0) printf("\n");
        printf("%4d ",*p);
    }
}
```

p →

int a[3][4];
a[0][0]
a[0][1]
a[0][2]
a[0][3]
a[1][0]
a[1][1]
a[1][2]
a[1][3]
a[2][0]
a[2][1]
a[2][2]
a[2][3]

p=*a;
p=&a[0][0];
p=*(a+0);
p=a;

p=*a;
p=&a[0][0];
p=(int *)a;
p=a;

如何跳



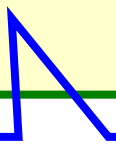
不同表示

int a[3][4]

表示形式	含义	地址
a	二维数组名，数组首地址	2000
a[0],*(a+0),*a	第0行第0列元素地址	2000
a+1	第1行首地址	2008
a[1],*(a+1)	第1行第0列元素地址	2008
a[1]+2,*(a+1)+2,&a[1][2]	第1行第2列元素地址	2012
(a[1]+2),(*(a+1)+2),a[1][2]	第1行第2列元素值	13

指向数组中单个元素的指针

```
main()
{  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int *p;
}
```



```
p=*a;
p=*(a+0);
p=&a[0][0];
```

- 总是指向数组中单个元素的位置
- 如何指向整个数组？

指向数组的指针

❖ 形式：数据类型 (*指针名)[一维数组维数];

例 `int (*p)[4];`

指向的一维数组有多大

() 不能少，表示指向数组
`int (*p)[4]`与`int *p[4]`不同

p的值是某一维数组的首地址，p是行指针

可让p指向二维数组某一行

如 `int a[3][4], (*p)[4]=a;`

一维数组指针变量维数和二维数组列数必须相同.每行包含4列!

举例

```
main()
{  static int a[3][4]={ 1,3,5,7,9,11,13,15,17,19,21,23};
   int i,j,(*p)[4];
   for(p=a,i=0;i<3;i++,p++)
       for(j=0;j<4;j++)
           printf("%d ",*(*p+j));

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

p=a[0];

×

p=*a;

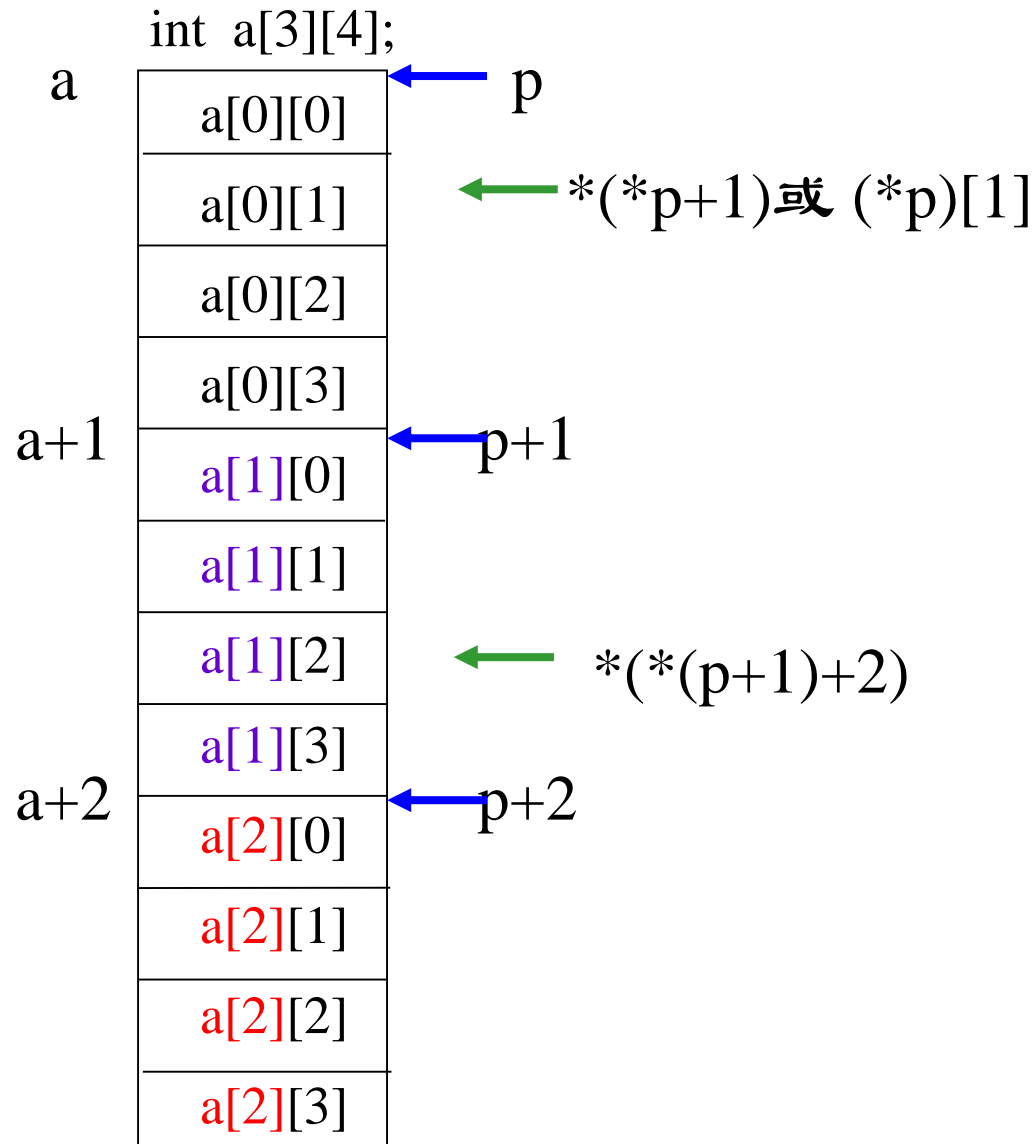
×

p=&a[0][0];

×

p=&a[0];

✓



二维数组与指针运算例子

```
main()
{  int a[3][4]={ {1,2,3,4},{3,4,5,6},{5,6,7,8}};
    int i;
    int (*p)[4]=a,*q=a[0];
    for(i=0;i<3;i++)
    {  if(i==0)
        (*p)[i+i/2]=*q+1;
        else
            {p++,++q;}
    }
    for(i=0;i<3;i++)
        printf("%d,",a[i][i]);
    printf("%d,%d\n",*( *p),*q);
}
```

p →	2	2	3	4
p →	3	4	5	6
p →	5	6	7	8

运行结果： 2， 4， 7， 5， 3

二维数组与一维数组指针

如 `int a[5][10]` 与 `int (*p)[10];`

☆ 二维数组名是一个指向有10个元素的一维数组的指针常量

☆ `p=a+i` 使 `p` 指向二维数组的第 `i` 行

☆ `*(*(p+i)+j) ⇔ a[i][j]`

☆ 二维数组形参实际上是一维数组指针变量,

即 `int x[][10] ⇔ int (*x)[10]`

☆ 变量定义时两者不等价

☆ 系统只给 `p` 分配能保存一个指针值的内存区(一般2字节)；而给 `a` 分配 `2*5*10` 字节的内存区

小 结

(1) 指针变量加（减）一个整数

例如： $p++$ 、 $p--$ 、 $p+i$ 、 $p-i$ 、
 $p+=i$ 、 $p-=i$ 等。

小结

(2) 指针变量赋值

将一个变量地址赋给一个指针变量。如：

$p = \&a$; （将变量 a 的地址赋给 p ）

$p = \text{array}$; （将数组 array 首元素地址赋给 p ）

$p = \&\text{array}[i]$; （将数组 array 第 i 个元素的地址赋给 p ）

$p_1 = p_2$; （ p_1 和 p_2 都是指针变量，将 p_2 的值赋给 p_1 ）

小结

(3) 指针变量可以有空值(**NULL**), 即该指针变量不指向任何变量。

(4) 两个指针变量比较

若两个指针指向同一个数组的元素, 则可以进行比较(**地址位置关系比较**)。指向前面的元素的指针变量“小于”指向后面元素的指针变量。

小 结

(5) ANSIC新标准增加了一种“void”指针类型，即可定义一个指针变量，但不指定它是指向哪一种类型数据的。使用时要注意类型转换，使之适合于被赋值的变量的类型。例如：

```
char *p1;
```

```
void *p2;
```

```
...
```

```
p1 = (char *) p2;
```

内容

- 指针相关知识
- 指针与二维数组
- 程序设计讲解

再看递归

- 循环的两种形式：
 - 迭代（循环语句）
 - 递归（递归函数）
 - 两种形式的循环方向恰好方向相反
- 例子
 - 递归判断回文数
 - 递归判断回文串
 - 递归计算最大公约数
 - 递归判断素数

递归判断回文数


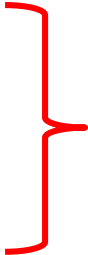
- 判断回文数的基本思想
 - 已知一个数为 m ;
 - 计算 m 的逆序数 n ;
 - 比较是否 m 与 n 相等。
- 关键问题：
 - 如何计算逆序

回忆非递归算法

- 例子说明计算过程（设 $m=12345$ ）
- 令逆序 n 的初始值 $n=0$;
- $k = m \% 10$, 得 $k=5$; $n = n * 10 + k = 0 * 10 + 5 = 5$
 - $m = m / 10$, 得 $m=1234$
- $k = m \% 10$, 得 $k=4$; $n = n * 10 + k = 5 * 10 + 4 = 54$
 - $m = m / 10$, 得 $m=123$
- $k = m \% 10$, 得 $k=3$; $n = n * 10 + k = 54 * 10 + 3 = 543$
 - $m = m / 10$, 得 $m=12$
- $k = m \% 10$, 得 $k=2$; $n = n * 10 + k = 543 * 10 + 2 = 5432$
 - $m = m / 10$, 得 $m=1$
- $k = m \% 10$, 得 $k=1$; $n = n * 10 + k = 5432 * 10 + 1 = 54321$
 - $m = m / 10$, 得 $m=0$
- $m=0$ （结束条件）

实现程序

```
#include <stdio.h>
main()
{ int m,k,temp ,n=0;
  scanf("%d",&m);
  temp=m;
  while(temp>0)
  {   k=temp%10;
      n=n*10+k;
      temp=temp/10;
  }
  if(m==n)
      printf("%d is palindrome number\n",m);
  else   printf("%d is not a palindrome number\n",m);
}
```



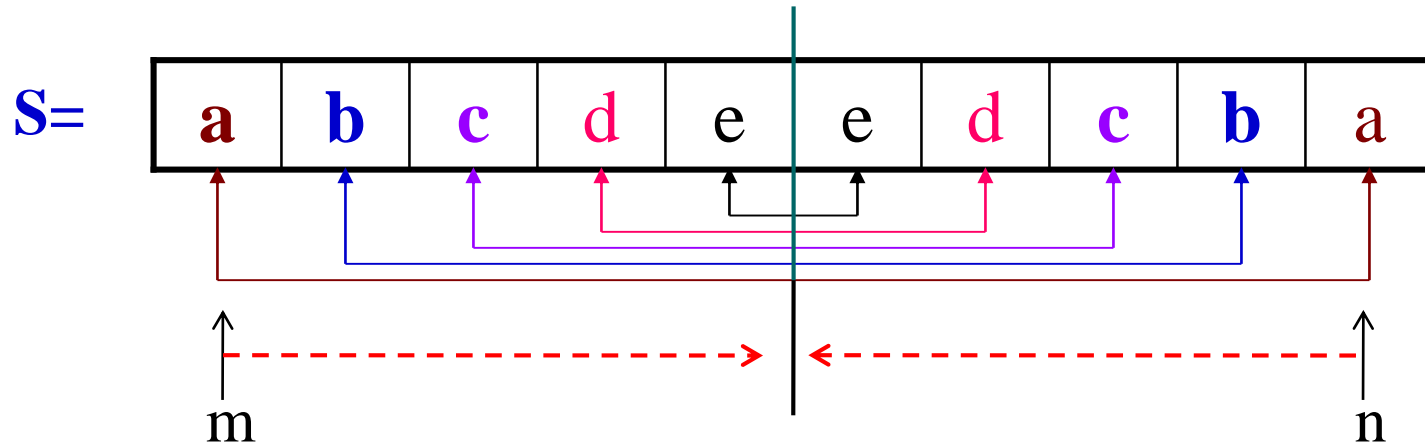
求m的逆序

递归判断回文数程序

```
int huiwen(int m, int n);
main()
{   int m, n=0;
    scanf("%d",&m);
    n=huiwen(m,0);
    if(m==n)
        printf("Yes\n");
    else printf("No\n");
}
int huiwen(int m, int n)
{   if(m==0)
        return(n);
    else return(huiwen(m/10, n*10+m%10));
}
```

求m的逆序

扩展：判断回文串



- 方法：对称点判断
 - 左右对称点定义为： m, n
 - $m=0; n=\text{strlen}(s)-1$
 - $(m < n) \ \&\& \ (s[m] == s[n])$
 - $m++; n--$

程序实现

```
int huiwenstr(char s[], int left, int right)
{
    if(!(left<right)) return 1;
    else if (s[left]!=c[right]) return 0;
        else return huiwenstr(s, left+1, right-1);
}
```

输出回文子串

给定一个字符串，输出所有回文子串。

要查找的子串长度应该大于等于2

子串长度小的优先输出，出现在原始字符串靠左的子串优先输出。

比如对于abcddcbaab

输出：

dd

aa

cddc

baab

bcddcb

abcddcba

问题分析

- **子串判断的两个问题：**
 - **判断一个串 s 的子串是否是回文串（关键问题）**
 - 一个串 S 的子串表示为 `sub(char s[], int start, int end)`
 - S 表示串，`start` 表示子串开始，`end` 表示子串结束；
 - **如何逐次取子串？**
 - 按长度变化，2,3, .. n (S 的总长度)
 - 从左向右移动
- **子串输出**

递归计算最大公约数

- 基本思想：
 - 假设: $m > n$
 - 引入临时变量: p 作为余数
 - step1: $p = m \% n$;
 - step2: if ($p == 0$) 结果为 n
 - step3: 否则, $m = n, n = p$, 转 step1
- 函数实现:
 - `int gcd(int m, int n)`

函数实现

```
int gcd(int m, int n)
{
    if(m%n==0) return (n);
    else return gcd(n, m%n);
}
```

递归判断素数

- 基本思想：
 - 设 $n > 1$
 - 逐个测试 $m \leq \sqrt{n}$
 - 如果能整除，则，不为素数
 - 否则，为素数。
- 函数实现：

```
int prime(int n, int limit, int num)  
// limit: n的平方根, num 当前要判的数
```

函数实现

```
int prime(int n, int limit, int num)
{
    if(num>limit) return 1;
    else if(n%num==0) return 0;
        else return prime(n, limit, num+1);
}
```

8 进制转化为10进制的递归

$$Y = (X_{n-1}X_{n-2}\dots X_1X_0)_8 = (?)_{10}$$

$$= \sum_{i=0}^{n-1} (8^i \square X_i)$$

$$= 8(\dots 8 (8 (8 (X_{n-1}) + X_{n-2}) + X_{n-3}) \dots + X_1) + X_0$$

反过来表示 (最高位: A_0 最低位: A_{n-1})

$$Y = (A_0A_1\dots A_{n-2}A_{n-1})_8 = (?)_{10}$$

$$= \sum_{i=0}^{n-1} (8^{n-1-i} \square A_i)$$

$$= 8(\dots 8 (8 (8 (A_0) + A_1) + A_2) \dots + A_{n-2}) + A_{n-1}$$

$$f(0) = (A_0 - '0');$$

$$f(m) = 8 * f(m-1) + (A_m - '0')$$

假设A 表示的是数字字符


```
int convertor (char A[], int base, int len)
{
    if(len==0)
        return(A[0]-'0');
    else return(A[len]- '0' +base*convertor(A,base,len-1));
} // 这里,base 表示进制
```

思考题:

冒泡排序和选择排序可以用递归实现吗?