

C语言与程序设计

The C Programming Language



第11章 复杂类型的指针

毛伏兵

华中科技大学计算机学院



指向数组的指针又称为数组的指针。

Diagram illustrating the components of an array declaration:

- 指针变量名 (Pointer variable name)
- 所指数组元素的类型 (Type of the array element it points to)
- 所指数组的长度 (Length of the array it points to)

11.1.2 用数组名间访二维数组的元素

二维数组被看成以1维数组（行）为元素的一维数组；

`int u[2][3]={1, 3, 5
 {2, 4, 6}};` \xrightarrow{u}

$u[0]$	1	3	5
$u[1]$	2	4	6

u 被看成有两个1维数组（行）元素($u[0], u[1]$)组成的一维数组

$u[0]$ 第0行首地址 即 $u[0] == \&u[0][0]$

$u[1]$ 第1行首地址 即 $u[1] == \&u[1][0]$

i 行 j 列的元素的地址？

(1) 用指向数组元素的指针表示: $u[i]+j$

(2) 用指向一维数组的指针表示: $*(u+i)+j$

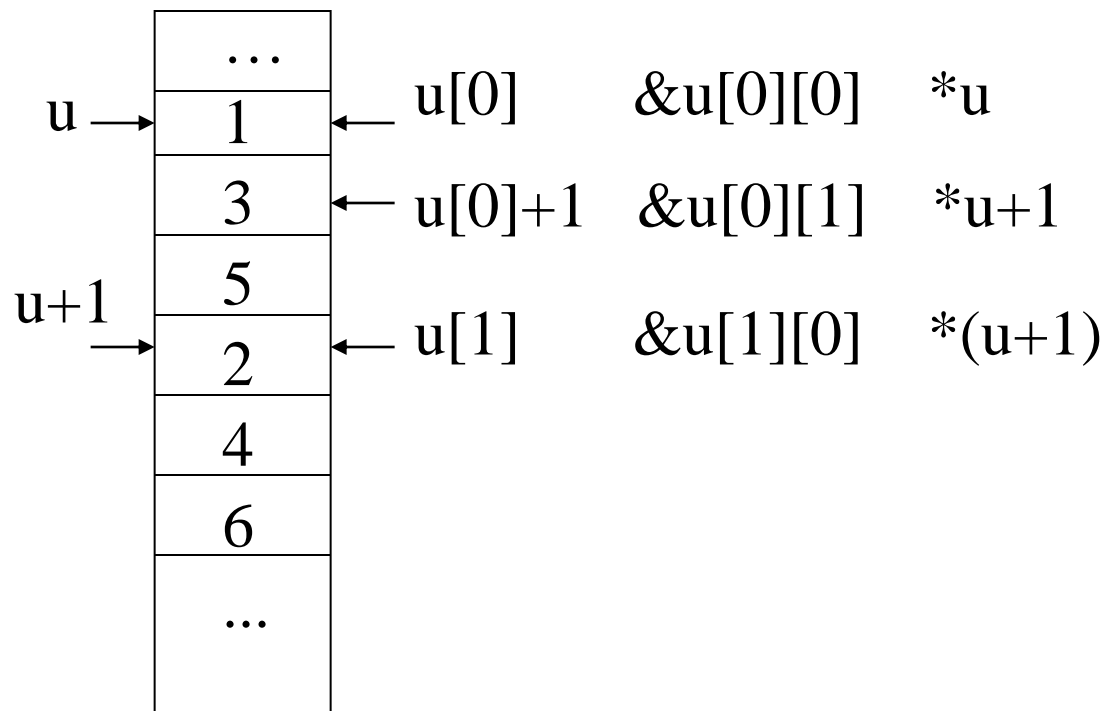
如何间访 i 行 j 列的元素？

(1) $*(u[i]+j)$

(2) $*(*(u+i)+j)$

用指针变量描述二维数组元素

```
int u[2][3]={  
    {1, 3, 5}  
    {2, 4, 6}  
};
```



1、将指针定义为指向数组元素

见8.4.3

2、将指针定义为指向由m个元素组成的数组的指针

```
int a[3][2]={ {1, 3}, {4, 6}, {7, 9} };
```

```
int (*p)[2];
```

p是指向数组的指针，该数组有2个int 型元素

```
p=a;
```

<code>*(p)</code> 或 <code>(p)[0]</code>	<code>*(p+1)</code> 或 <code>(p)[1]</code>
<code>*(p+1)</code> 或 <code>(p+1)[0]</code>	<code>*(p+1+1)</code> 或 <code>(p+1)[1]</code>
<code>*(p+2)</code> 或 <code>(p+2)[0]</code>	<code>*(p+2+1)</code> 或 <code>(p+2)[1]</code>

例 二维数组元素的输入/输出

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

```
#define I 2
```

```
#define J 3
```

```
void main(void)
```

```
{   int u[I][J], (*p)[J]=u;
```

```
    int j;
```

```
    for(j=0;j<J;j++)    /* 用指向数组元素的指针完成第0行元素的输入 */
```

```
        scanf("%d", (u[0]+j));
```

```
    for(p++,j=0;j<J;j++) /* 用指向数组的指针完成第1行元素的输入 */
```

```
        scanf("%d", (*p+j));
```

```
    for(j=0;j<J;j++) /* 用指向数组的指针完成第0行元素的输出 */
```

```
        printf("%6d", *(&u[0][j]));
```

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

```
    for(j=0;j<J;j++)    /* 用指向数组元素的指针完成第1行元素的输出 */
```

```
        printf("%6d", *(&u[1][j]));
```

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

```
}
```

11.1.4 二维数组作函数参数

- 形参说明为数组

```
fun(int x[ ][4])  
{  
    ...  
}
```

- 形参说明为指针

{ 指向数组元素的指针
 指向下一级数组的指针

```
fun(int *x)  
{  
    ...  
}
```

```
fun(int (*x)[4] )  
{  
    ...  
}
```

11.2 用typedef定义类型表达式

11.2.1 类型表达式

C中的表达式可以分成两类。

- (1) **值表达式**，由运算符和操作数组成，可被CPU处理和计算
- (2) **类型表达式**，由类型说明符和数据类型名组成，类型说明符有：()、[]、*

int (*) [5]

2) typedef定义中

typedef是关键字，为一个类型表达式定义一个别名。

typedef 类型区分符 说明符 ;



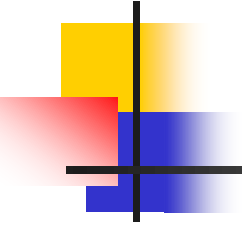
基本类型 结构 联合

也可以是由 **typedef** 定义的类型名

(1) **typedef unsigned int size_t;**

size_t定义为**unsigned int**类型

size_t x, y; /* unsigned int x, y; */

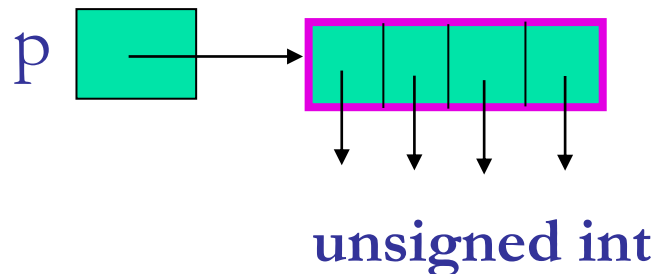


(4) `typedef char * (*p_to_fun)(char *,char *);`
p_to_fun 定义为 `char *(*)(char *,char *)`

`p_to_fun fptr;`

11.3 复杂说明的解释

*unsigned *(*p)[4]*



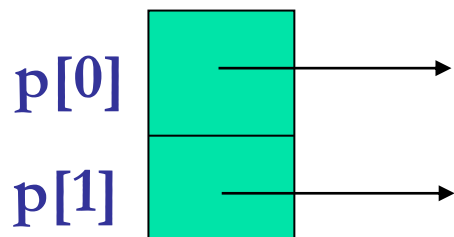
p 是指向数组的指针，
该数组有4个无符号整型指针元素。
或

p 是有4个无符号整型指针元素的数组的指针。



```
char *(*p[2])(char *, int);
```

*p*是含有2个指针元素的数组，每个指针指向有一个字符指针参数和一个整型参数，返回值为字符型指针的函数。



函数：*char* * (***char*** *, ***int***)

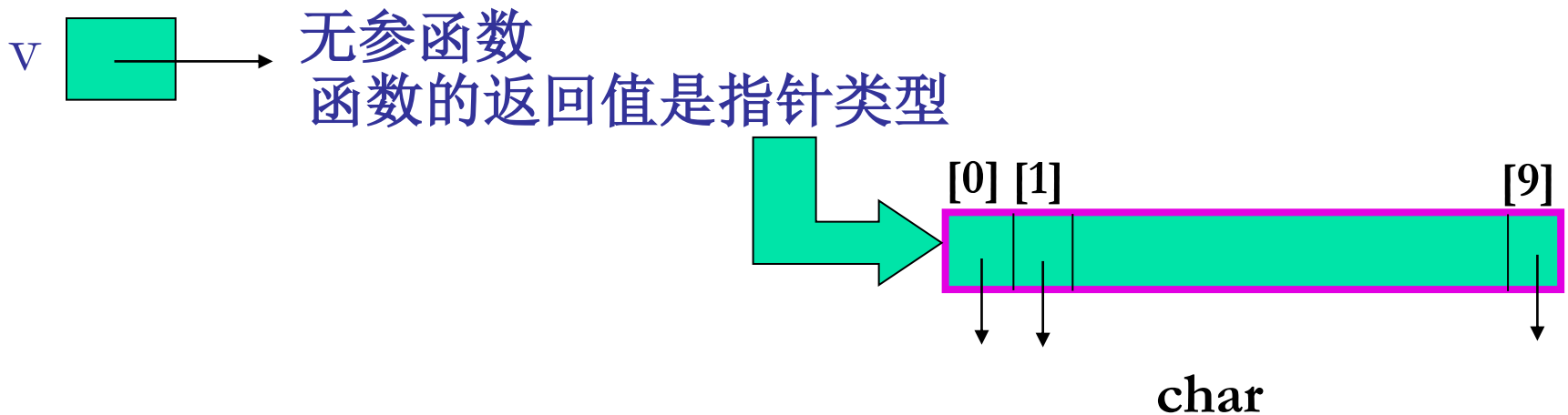
例11.7 有两个元素的函数指针数组的复杂说明的使用举例。

```
#include "stdio.h"
int a1(int);
int a2(int);
int main(void)          /*fpa是有2个元素的函数指针数组，每个元素指向*/
{  int i,(*fpa[2])(int); /*的函数有一个整型形参，返回整型值*/
    fpa[0]=a1;   /*第1个（下标为0）元素指向a1函数*/
    fpa[1]=a2;   /*第2个（下标为1）元素指向a2函数*/
    for(i=0;i<2;i++)
        printf("%d\n",fpa[i]((i+1)*5));/*依次调用fpa[0]、fpa[1]所指函数*/
    return 0;
}
int a1(int x){ printf("in function a1,x=%d\n",x); return 2*x; }
int a2(int y){ printf("in function a2,y=%d\n",y); return 2*y; }
```



```
char * (3) (4)
      (6) (5) (1) (2) (*(*v)(void))[10];
```

v 是函数的指针，该函数没有参数，
返回值是指向有10个元素的字符指针数组的指针。





```
int (*f(char *(*)(int)))[10];
```

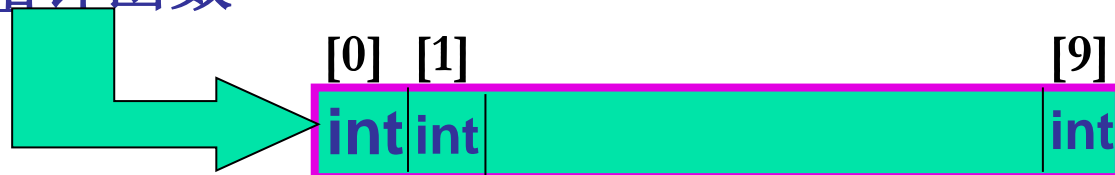
*f*是一个指针函数,

*f*函数的形参为一个指向函数的指针, 所指函数有一个整型形参且返回值类型为char *;

*f*函数的返回值是指向有10个整型元素的数组的指针。

f 的形参: char *(*)(int)

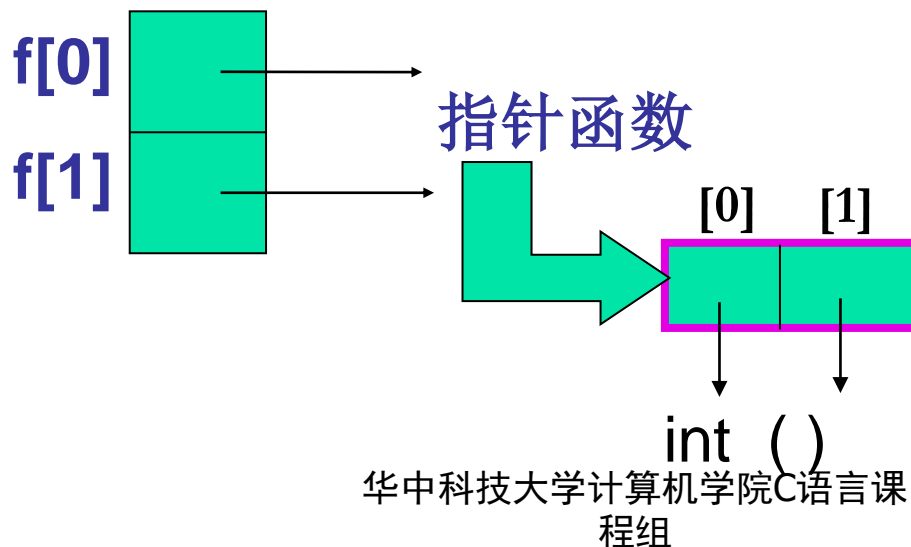
*f*是一个指针函数





```
int ( *(*(*f[2])())[2])();
```

f是一个有两个元素的函数指针数组；所指函数的返回值是指向有两个元素的指针数组的指针，指针数组的每个指针元素指向一个无参整型函数。（无参整型函数指函数没有参数且返回值为整型值）。





****11.4 复杂说明的应用**

- **例11.6** 设p是一个函数指针，所指向的函数无参，且返回值为一个函数指针；该（返回的）函数指针指向的函数有两个字符指针形参，且返回值为字符指针值。请写出该声明语句并编写使用p的应用程序。
- 相应的声明语句为：

char *(*(*p)(void))(char *,char *);

```

#include "stdio.h"
char *(*p)(void)(char *,char *);/*f1是指针函数，返回的指针指向有两个*/
char *(*f1(void))(char *,char *);/*字符指针形参，返回值为字符指针值的函数*/
char *strcpy(char *,char *);/*有两个字符指针形参，返回值为字符指针值的函数*/
int main(void)
{
    char a[80],b[]="aaa,bbb,ccc",*str;
    p=f1;/*p指向f1，下面语句先计算(*p)()，即f1函数*/
    str=(*p)()(a,b);/*返回strcpy；然后调用strcpy(a,b)，结果赋给str*/
    printf("%s\n",str);
    return 0;
}
char *(*f1(void))(char *,char *)
{
    /*f1(void)表示f1是无参函数，其余部分都是描述它的返回值的类型*/
    return strcpy; /*返回指向有两个字符指针形参，返回值为字符指针值的函数*/
}
char *strcpy(char *t,char *s)
{
    char *p=t;
    while((*t++=*s++)!='\0') ;
    return p;
}

```

演示： [源程序\ex11_6.c](#)



复杂类型的使用

- 下面进一步就复杂说明：

`int (*fpa[2])(int);`

和`int ((*(*f[2])())[2])(int,int);`

为例，说明复杂类型的使用。

- 对于复杂说明`int (*fpa[2])(int);`要从理解fpa的含义，对fpa数组的元素进行赋值，以及通过fpa去调用fpa数组元素所指函数三个方面加以把握。
- 由于fpa是一个有两个元素的函数指针数组，因此构造两个函数a1和a2，并通过`fpa[0]=a1;`和`fpa[1]=a2;`使fpa[0]指向a1，fpa[1]指向a2。
- 同时，通过表达式`fpa[i]((i+1)*5)`，当i=0调用函数a1，当i=1调用函数a2，而 $(i+1)*5$ 是实参。

例11.7 有两个元素的函数指针数组的复杂说明的使用举例。

```
#include "stdio.h"
int a1(int);
int a2(int);
int main(void)          /*fpa是有2个元素的函数指针数组，每个元素指向*/
{  int i,(*fpa[2])(int); /*的函数有一个整型形参，返回整型值*/
    fpa[0]=a1;   /*第1个（下标为0）元素指向a1函数*/
    fpa[1]=a2;   /*第2个（下标为1）元素指向a2函数*/
    for(i=0;i<2;i++)
        printf("%d\n",fpa[i]((i+1)*5));/*依次调用fpa[0]、fpa[1]所指函数*/
    return 0;
}
int a1(int x){ printf("in function a1,x=%d\n",x); return 2*x; }
int a2(int y){ printf("in function a2,y=%d\n",y); return 2*y; }
```