C 语言指针复习

* 的优先级别 低于 []

带着问题学习本章内容：

```
int *p[4];       //指针数组，数组元素师指针的数组
int (*p)[4];     //数组指针，一般用于操作二维数组的指针变量
int *fun();      //指针函数，用于返回指针值的函数
int (*fun)();    //函数指针，指向函数的指针变量
```

1 一维数组与指针

1.1 一维数组元素及其地址表示

int a[5] = {1, 2, 3, 4, 5};

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

1.2 指向一维数组元素的指针

int a[5] = {1, 2, 3, 4, 5}, *p;

p = a; // p = &a[0];

元素	地址	地址	元素
*p	p	a	a[0]
*(p+1)	p+1	a+1	a[1]
*(p+2)	p+2	a+2	a[2]
*(p+3)	p+3	a+3	a[3]
*(p+4)	p+4	a+4	a[4]

✧ 指针变量 p 与数组名 a 都表示数组的首地址，但数组名 a 是指针常量，而 p 是指针变量。

1.3 注意

(1) p++合法，但 a++不合法，因为 a 是地址常量，不能改变，而 p 是一个指针变量，可以改变；

(2) 要注意指针变量的当前值，保证它指向数组中有效的元素；

(3) 注意指针变量的运算：

① *p++ <=> *(p++)

先取 p 所指向的变量，再使 p 自加指向下一个变量。

② *(p++) 与*(++p)不同

*(++p)是先使 p 自加指向下一个变量，再取其值。

③ (*p)++ 先取 p 所指向的变量，再使变量的值加 1

(4) 指针变量也可以加下标 p[i] = a[i] = *(p+i) 。

2 二维数组和指针

2.1 二维数组可以看成一维数组的一维数组

每行数组看成一个元素。

2.2 二维数组名是一个二级指针

定义：

int a[3][4];

✧ 二维数组名 a 表示二维数组的首地址,也是第 0 行的首地址, a <=> &a[0]

则：

a <=> &a[0] <=> &(&(a[0][0]));

*a <=> a[0] <=> &(a[0][0]);

==========

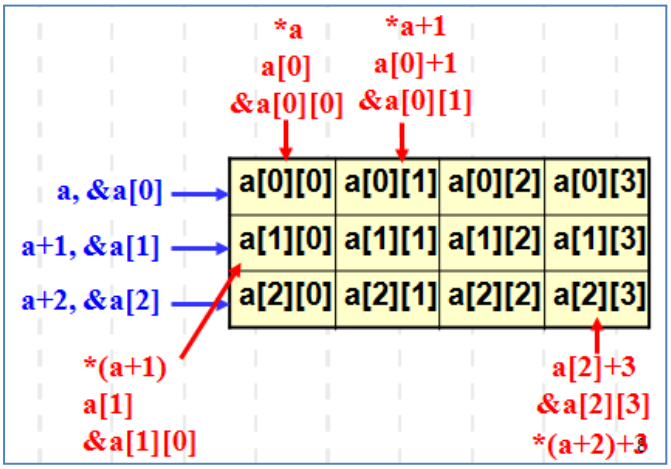
*a 表示元素 a[0][0]的地址

*a+1 表示元素 a[0][1]的地址

*(a+1) 表示元素 a[1][0]的地址

*(a+2)+3 表示元素 a[2][3]的地址

2.3 形象表示

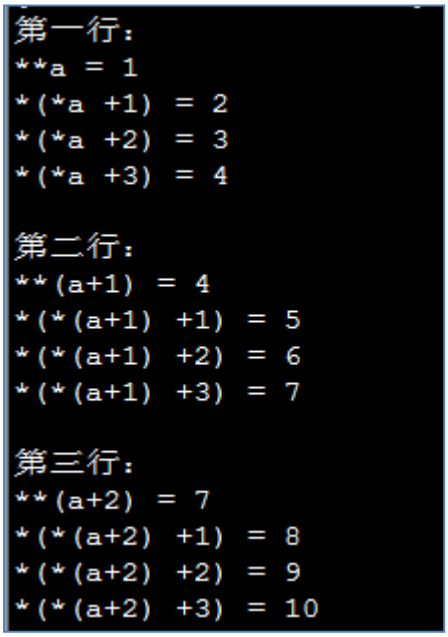


2.4 看个例子

```
//********************double_dimensional_array.c    begin
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
    //输出每一个值
    //第一行
    printf("第一行：\n");
    printf("**a = %d\n", **a);
    printf("*(*a +1) = %d\n", *(*a + 1));
    printf("*(*a +2) = %d\n", *(*a + 2));
    printf("*(*a +3) = %d\n\n", *(*a + 3));
    //第二行
    printf("第二行：\n");
    printf("**(a+1) = %d\n", **(a+1));
    printf("*(*(a+1) +1) = %d\n", *(*(a +1) + 1));
    printf("*(*(a+1) +2) = %d\n", *(*(a +1) + 2));
    printf("*(*(a+1) +3) = %d\n\n", *(*(a +1) + 3));
    //第三行
    printf("第三行：\n");
    printf("**(a+2) = %d\n", **(a+2));
    printf("*(*(a+2) +1) = %d\n", *(*(a +2) + 1));
    printf("*(*(a+2) +2) = %d\n", *(*(a +2) + 2));
    printf("*(*(a+2) +3) = %d\n", *(*(a +2) + 3));
    return 0;
}
```

```
//********************double_dimensional_array.c    end
```

编译运行：



2.5 易混淆讲解

2.5.1 int(*p)[4]和 int*p[4]

- ✧ int (*p)[4];------ptr 为指向含 4 个元素的一维整形数组的指针变量（是指针）
- ✧ int *p[4];-------定义指针数组 p，它由 4 个指向整型数据的指针元素组成（是数组）
- ✧ int (*)[4];--------实际上可以看作是一种数据类型。

也就是第一个（int(*p)[4];）中定义的 p 的数据类型其实你要看这种到底是什么，就是要看他最先和谁结合。 比如 1 中 p 先与*结合，那就说明 p 本质是一个指针；而 2 中 p 先与后面的[4]结合，说明他本质是一个数组。

```
int   p[3][4];
int   (*p)[4]; // //注意，数组指针的大小是 列数 4
p = a;
```

然后可以用 p 操作二维数组，把例子修改一下即可：

```
//********************double_dimensional_array.c    begin
#include <stdio.h>
int main(int argc, char *argv[])
{
    int p[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
    //输出每一个值
```

```
    //第一行
    int (*a)[4]; //注意，数组指针的大小是 列数 4
    a = p;
    printf("第一行：\n");
    printf("**a = %d\n", **a);
    printf("*(*a +1) = %d\n", *(*a + 1));
    printf("*(*a +2) = %d\n", *(*a + 2));
    printf("*(*a +3) = %d\n\n", *(*a + 3));
    //第二行
    printf("第二行：\n");
    printf("**(a+1) = %d\n", **(a+1));
    printf("*(*(a+1) +1) = %d\n", *(*(a +1) + 1));
    printf("*(*(a+1) +2) = %d\n", *(*(a +1) + 2));
    printf("*(*(a+1) +3) = %d\n\n", *(*(a +1) + 3));
    //第三行
    printf("第三行：\n");
    printf("**(a+2) = %d\n", **(a+2));
    printf("*(*(a+2) +1) = %d\n", *(*(a +2) + 1));
    printf("*(*(a+2) +2) = %d\n", *(*(a +2) + 2));
    printf("*(*(a+2) +3) = %d\n", *(*(a +2) + 3));
    return 0;
}
//********************double_dimensional_array.c    end
```

2.5.2 使用指针操作二维数组易错点

正确使用指针操作二维数组方法：

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

错误使用指针操作二维数组方法：

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

解析：

指针指向的地址空间长度不一样。

(*a)[4]指针，每++一次，移动 4 个单位。

[]优先级比*大

**a 是*a[]。p[3][4] 是（*p）[4]

修正：

改一下就可以了。

int *a;

a = *p ;

2.6 小结

小结：二维数组的地址分为行地址和列地址

二维数组名 a 代表二维数组的首地址,即:

第 0 行的地址为 a <=> &a[0]

第 1 行的地址为 a+1 <=> &a[1]

第 2 行的地址为 a+2 <=> &a[2]

二维数组的列地址即元素的地址

元素	地址	地址	地址	元素	元素
a[0][0]	&a[0][0]	a[0]	*a	*a[0]	**a
a[0][1]	&a[0][1]	a[0]+1	*a+1	*(a[0]+1)	*(*a+1)
a[1][0]	&a[1][0]	a[1]	*(a+1)	*a[1]	*(*(a+1))
a[1][2]	&a[1][2]	a[1]+2	*(a+1)+2	*(a[1]+2)	*(*(a+1)+2)
a[2][3]	&a[2][3]	a[2]+3	*(a+2)+3	*(a[2]+3)	*(*(a+2)+3)

char a[3][4];

int (*p)[4];

p = a;

这个是正确的，想想为什么？

地址肯定是 32 位的无符号数!!!

3 动态分配内存之动态数组

函数：

```
//头文件
#include <stdlib.h>
#include <malloc.h>
//相关函数
void *calloc(size_t size);
void *callo(size_t num, size_t size);
void realloc(void *ptr, size_t size);
void *memset(void *s, int c, size_t n);
void free(void* p);
```

详细看我另外的文档。

3.1 一维动态数组

```
int    *p = NULL;
printf("Please enter array size:");
scanf("%d", &n);
p = (int *) malloc(n * sizeof (int));
//…
p[i];            //像使用一维数组一样使用
//…
```

3.2 二维动态数组

```
printf("Please enter array size m,n:");
scanf("%d,%d", &m, &n);
p = (int *) calloc(m * n, sizeof (int));
//…
p[i*n+j]); //像使用一维数组一样使用
//…
```

3.3 注意事项

1. calloc 申请内存时候把指向的内存单元初始化为 0，而 malloc 不会，所以在 malloc 申请内存后一般要用 memset 一下：

```
int p;
p = (int *)malloc(100*sizeof(int));
memset(p, 0, 100*sizeof(int));
```

等价于：

```
p = (int *)calloc(100, sizeof(int));
```

2. free释放指针成功之后，把指针prt置为NULL。

4 函数和指针

4.1 指针函数

返回指针值的函数称为指针函数；

函数的返回值可以是一个指针类型的数据(即地址)；

返回指针值函数的定义格式:

函数类型 *函数名(形参列表)

{

//函数体;

}

说明:定义一个返回指针值的函数与定义普通函数的格式基本类似,只是在函数名前加*，表明该函数返回一个指针值。

例子：

```
int * fun( int a, int b)
{
    //函数体;
}
```

4.2 函数指针

函数的指针：函数的入口地址，函数的入口地址是用函数名来表示的。

因此我们可以定义一个指针变量，让它的值等于函数的入口地址， 然后可以通过这个指针变量来调用函数，该指针变量称为指向函数的指针变量。

指向函数的指针变量

函数：

```
int  fun( int arr[], int n);
```

1. 定义格式:

数据类型 (*指针变量名)(形参列表);

```
int  (*pt)(int arr[], int n);
```

说明:

① 数据类型: 指针变量所指向的函数的返回值类型

② 形参表列: 即指针变量所指向的函数的形参表列

③ 格式中的小括号不能省略

2. 应用

(1) 让指针变量指向函数 pt = fun ;

因为函数名为函数的入口地址，所以直接将函数名赋给指针变量即可

(2) 使用指针变量调用函数

格式: (*指针变量名)(实参表列)

例：求一维数组中全部元素的和

```
#include  <stdio.h>
int  add( int  b[ ] ,  int  n);     //定义函数
void  main( )
{
    int  a[6] = {1,  3,  5,  7,  9,  11},  total ;
```

```
    int    (*pt) ( int    b[ ] , int n ) ; //定义函数指针变量
    pt = add ;                          //让指针变量指向函数
    total = (*pt) ( a , 6 ) ;           //使用函数指针变量
    printf( “total = % d \n”, total ) ;
}
int   add( int   b[ ] , int    n )
{
    int    i , sum = 0 ;
    for ( i = 0; i < n ; i++)
    {
        sum = sum + b[i];
    }
    return(sum);
}
```

5 总结

5.1 几种指针的定义格式及含义

```
char   *p;              /*一般指针*/
char   (*pa)[N];        /*指向数组的指针*/
char   *pa[N];           /*指针数组*/
char   **pp;             /*二级指针*/
char   *fp( );            /*指针函数，即返回值是指针类型的函数*/
char   (*fp)( );          /*函数指针，用来存放函数的入口地址的指针变量*/
```

5.2 通过指针在函数间传递参数

5.2.1 传递某变量的地址，或一维数组（字符串）首地址

类型 myfun(char *p） /*一般指针*/

调用时实参可用指针变量、变量的地址常量如&a、或数组名。

5.2.2 传递二维数组首地址

类型 myfun(int (*pa)[N]） /*指向数组的指针*/

调用时实参可用数组指针变量或二维数组名。

5.2.3 传递指针数组（多个字符串）首地址

类型 myfun（ char *pa[N]; ） /*指针数组*/

或

类型 myfun（char **pp） /*二级指针*/

5.2.4 调用时实参可用二级指针变量、指针数组名、二维字符数组名

传递函数的入口地址

类型 myfun（char (*pf)(形参说明)) /*函数指针*/

调用时实参可用函数名。

5.3 指针与数组

x[i] *(x+i) /*数组第 i 个元素*/

&x[i] x+i /*数组第 i 个元素的地址*/

用指针变量访问数组：

for (; p<a+N;p++) sum+=*p;

while (p<a+5) printf(“%d”,*p++);

知识在不断的使用中能够深入理解、在不断的学习和工作中，深化你自己吧！