知识点1【指针数组】(了解)

知识点2【数组指针】（了解）

总结:

知识点3【二维数组的分析】(了解)

知识点4【数组指针 与 二维数组的关系】（了解）

知识点6【任何维度的数组 在物理存储上 都是一维的】（了解）

知识点7【多级指针】（了解）

知识点8【指针变量作为函数的参数】

1、如果想在函数内部 修改 外部变量的值 就需要将外部变量的地址 传递给函数（以指针变量作为函数的参数）(重要！！！！)

知识点引入:

知识点9【一维数组名作为函数的参数】

1、如果函数内部想操作（读、写）外部数组的元素，请将外部数组的数组名传递函数。（重要！！！）

2、一维数组作为函数的形参 会被优化成一级指针变量。

知识点10【二维数组名作为函数的参数】(了解)

1、如果函数内部想操作（读、写）外部数组的元素，请将外部数组的数组名传递函数。（重要！！！）

2、二维数组名 作为函数的形参 会被优化成 数组指针。

知识点11【指针作为函数的返回值】

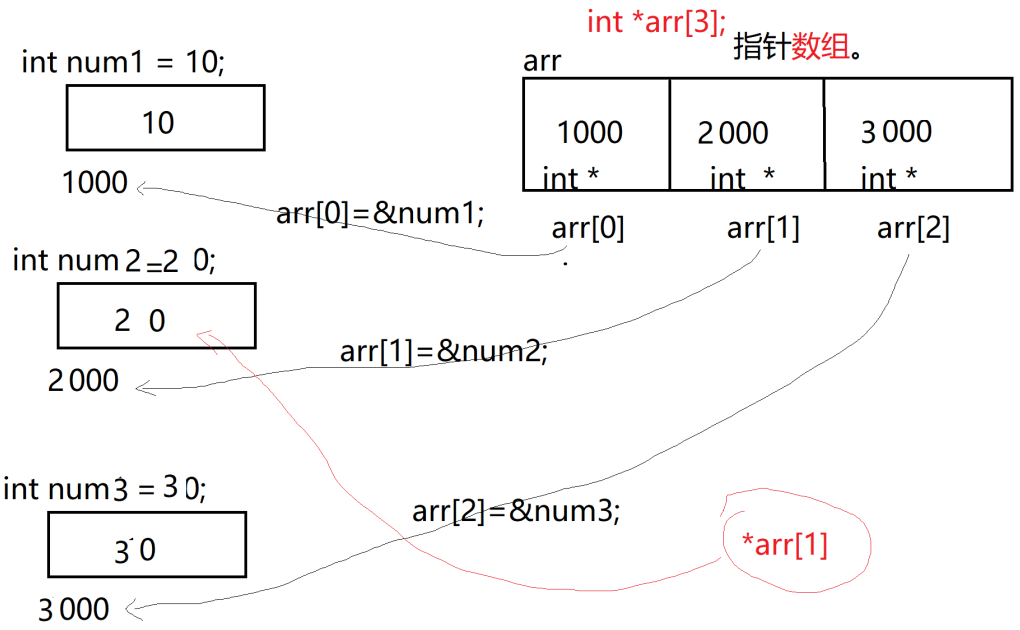
1、函数不要返回普通局部变量的地址。

2、解决上述问题:

知识点12【函数名 代表的是函数的入口地址】

知识点1【指针数组】(了解)

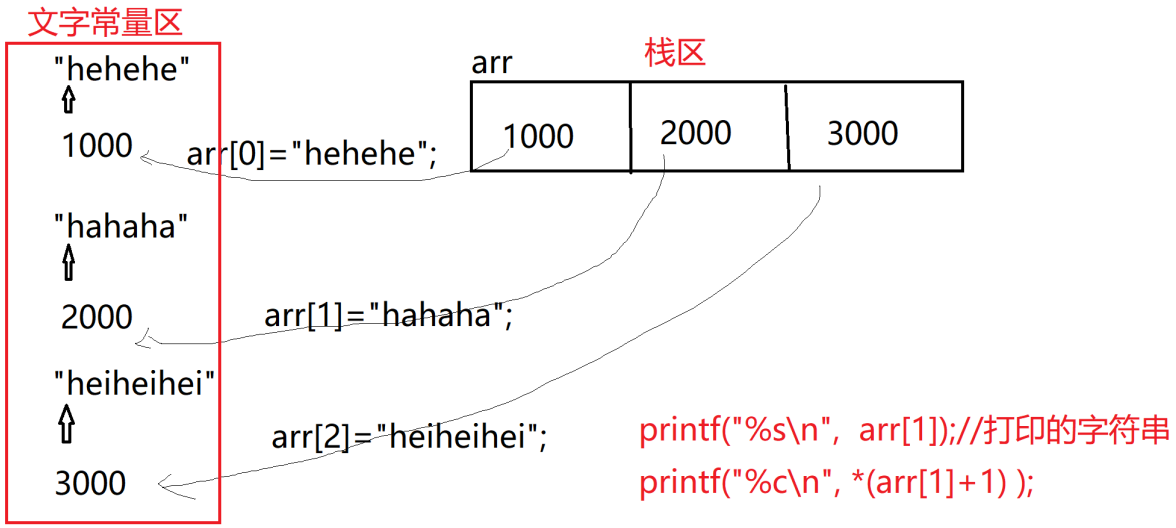
指针数组：本质是数组 只是数组的每个元素 是指针。



案例：

```
1  void test01()
2  {
3   int num1 = 10;
4   int num2 = 20;
5   int num3 = 30;
6   //指针数组
7   int *arr[3]={&num1, &num2, &num3};
8   char *arr2[3];
9   //arr[0] = &num1, arr[1]=&num2, arr[2]=&num3
10
11   printf("%d\n", *arr[1]);//*arr[1]=*&num2=num2
12
13   printf("%d\n",sizeof(arr));//12
14   printf("%d\n",sizeof(arr2));//12
15   return;
16  }
```

案例：

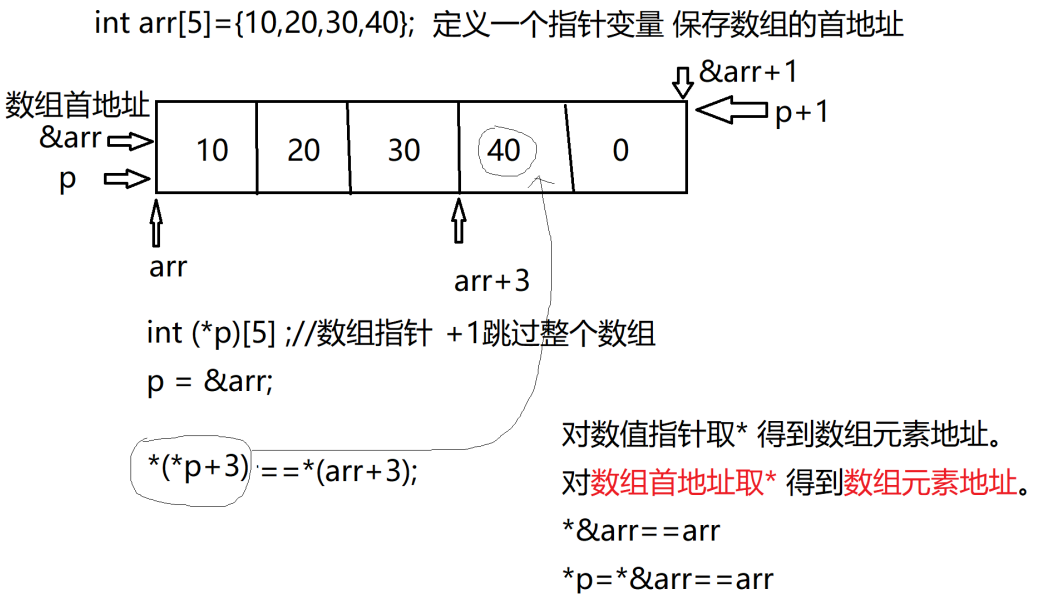



```c
void test02()
{
 char *arr[3]={"hehehe","hahaha","heiheihei"};
 printf("%d\n", sizeof(arr));//12

 printf("%s\n", arr[1]);
 printf("%c\n", *(arr[1]+1));
}
```

运行结果：

```
12
hahaha
a
Press any key
```

知识点2【数组指针】（了解）



案例：

```
1  void test03()
2  {
3   int arr[5]={10,20,30,40};
4   int (*p)[5];//数组指针：本质是一个指针变量 只是该变量 保存的是数组的首地址
5
6   printf("%d\n",sizeof(p));//4
7
8   printf("p=%u\n", p);
9   printf("p+1=%u\n", p+1);
10
11   p = &arr;//&arr 才代表数组的首地址
12
13   printf("%d\n", *(*p+3));//40
14   //*(*p+3) == *(*(p+0)+3)==*(p[0]+3)==p[0][3]
15   printf("%d\n", p[0][3]);//40
16
17  }
```

运行结果：

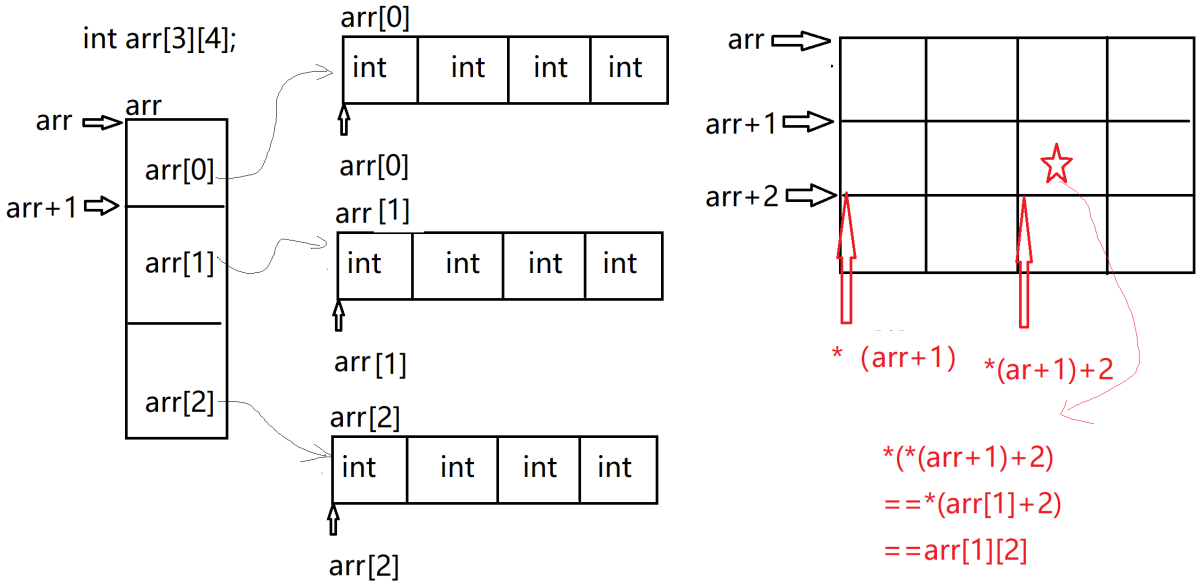
```
4
p=3435973836
p+1=3435973856
40
40
Press any key to continue
```

总结：

指针数组：本质是数组 只是数组的每个元素是指针

数组指针：本质是指针变量 只是保存的是 数组的首地址

知识点3【二维数组的分析】(了解)



二维数组名：代表的是二维数组的首行地址 +1跳过一行。

对行地址取* 将变成 当前行的第0列的列地址。

案例：int arr[3][4];

*arr+2 ://第0行 第2列的列地址

arr[1]://*(arr+1) 第1行第0列的列地址

&arr[0]+2: &*(arr+0)+2==arr+2 表示的是第2行的行地址

arr://arr == * (*(arr+0)+0) ==arr[0][0]

知识点4【数组指针 与 二维数组的关系】（了解）

需求：定义一个指针变量 保存二维数组的行地址

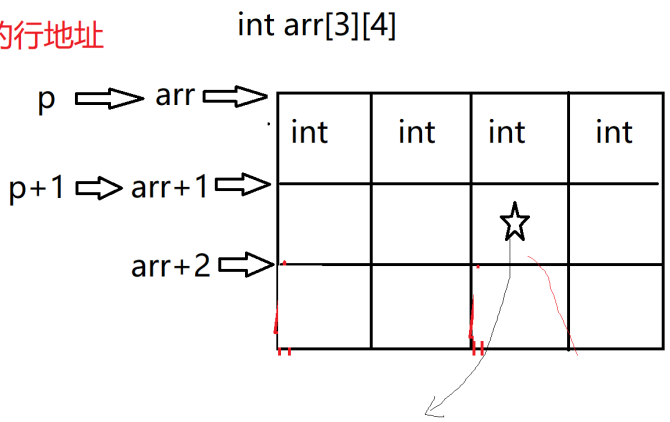
int arr[3][4]

//数组指针

int (*p)[4];

p =arr;

p和arr完全等价



((arr+1)+2)==arr[1][2]

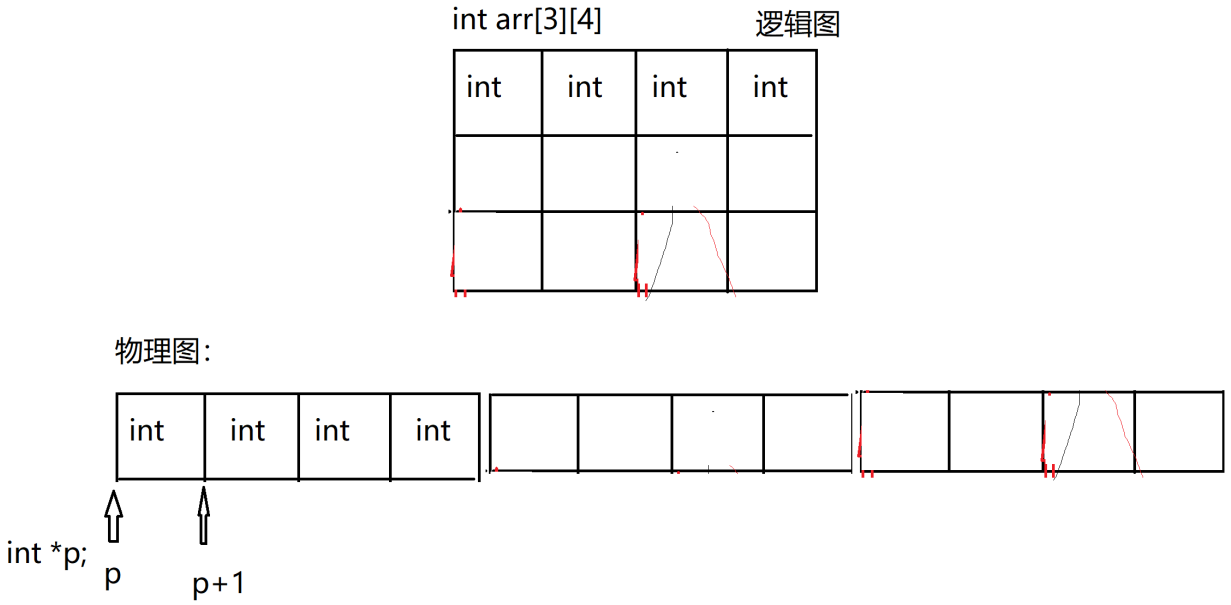
((p+1)+2)==p[1][2]

```c
void test04()
{
 int arr[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
 int i=0,j=0;
 //数组指针 本质是指针变量
 int (*p)[4] =arr;
 printf("%d\n",sizeof(p));

 for(i=0;i<3;i++)
  {
  for(j=0;j<4;j++)
  {
  //printf("%d ", *(*(p+i)+j));
  printf("%d ", p[i][j]);
  }
  printf("\n");
  }
}
```

运行结果：

```
4
1 2 3 4
5 6 7 8
9 10 11 12
Press any key to continue
```

知识点6【任何维度的数组 在物理存储上 都是一维的】（了解）



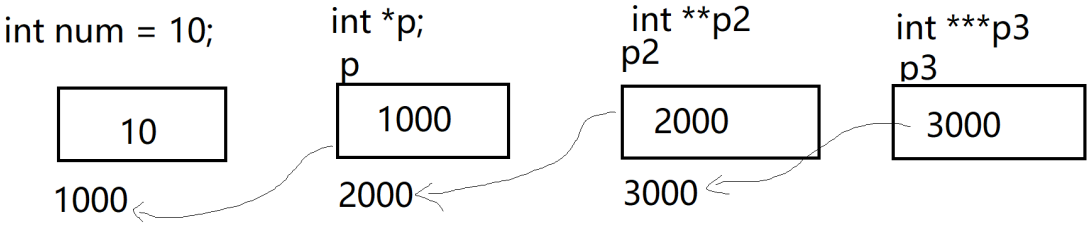

```c
void test05()
{
 int arr[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
 int i=0;
 int *p = &arr[0][0];
 for(i=0;i<3*4; i++)
 {
 //printf("%d ", *(p+i));
 printf("%d ", p[i]);

  }
```

```
12   printf("\n");
13  }
```

运行结果：

```
1 2 3 4 5 6 7 8 9 10 11 12
Press any key to continue
```

知识点7【多级指针】（了解）



1级指针变量 保存 0级指针变量（普通变量）的地址　　***p3==num

2级指针变量 保存 1级指针变量（普通变量）的地址　　**p2 == num

3级指针变量 保存 2级指针变量（普通变量）的地址　　*p　==num

.....

n级指针变量 保存 n-1级指针变量（普通变量）的地址　*p3 ==p2==&p

**p3==*p2==p==&num

知识点8【指针变量作为函数的参数】

1、如果想在函数内部 修改 外部变量的值 就需要将外部变量的地址 传递给函数（以指针变量作为函数的参数）（重要！！！！）

知识点引入：

```
1  void my_swap(int a,int b)//a=data1, b=data2
2  {
3   int tmp=0;
4   tmp = a;
5   a = b;
6   b = tmp;
7  }
8  void test06()
9  {
```

```
 int data1 = 10,data2 = 20;
 printf("data1=%d,data2=%d\n",data1,data2);
 //如果传递的是值 函数内部 是无法修改外部变量值
 my_swap(data1,data2);
 printf("data1=%d,data2=%d\n",data1,data2);
}
int main(int argc,char *argv[])
{
 test06();
 return 0;
}
```

上面的代码 是无法修改data1 data2的值

解决上面代码的问题 就需要将data1 data2的地址 传递到函数。

```
void my_swap2(int *p1,int *p2)//p1=&data1 p2=&data2
{
 int tmp;
 tmp = *p1;
 *p1 = *p2;
 *p2 = tmp;
}
void test07()
{
 int data1 = 10,data2 = 20;
 printf("data1=%d,data2=%d\n",data1,data2);

 //函数内部 修改外部变量的值 需传递外部变量的地址
 my_swap2(&data1,&data2);
 printf("data1=%d,data2=%d\n",data1,data2);
}
int main(int argc,char *argv[])
{
 test07();
 return 0;
}
```

运行结果：

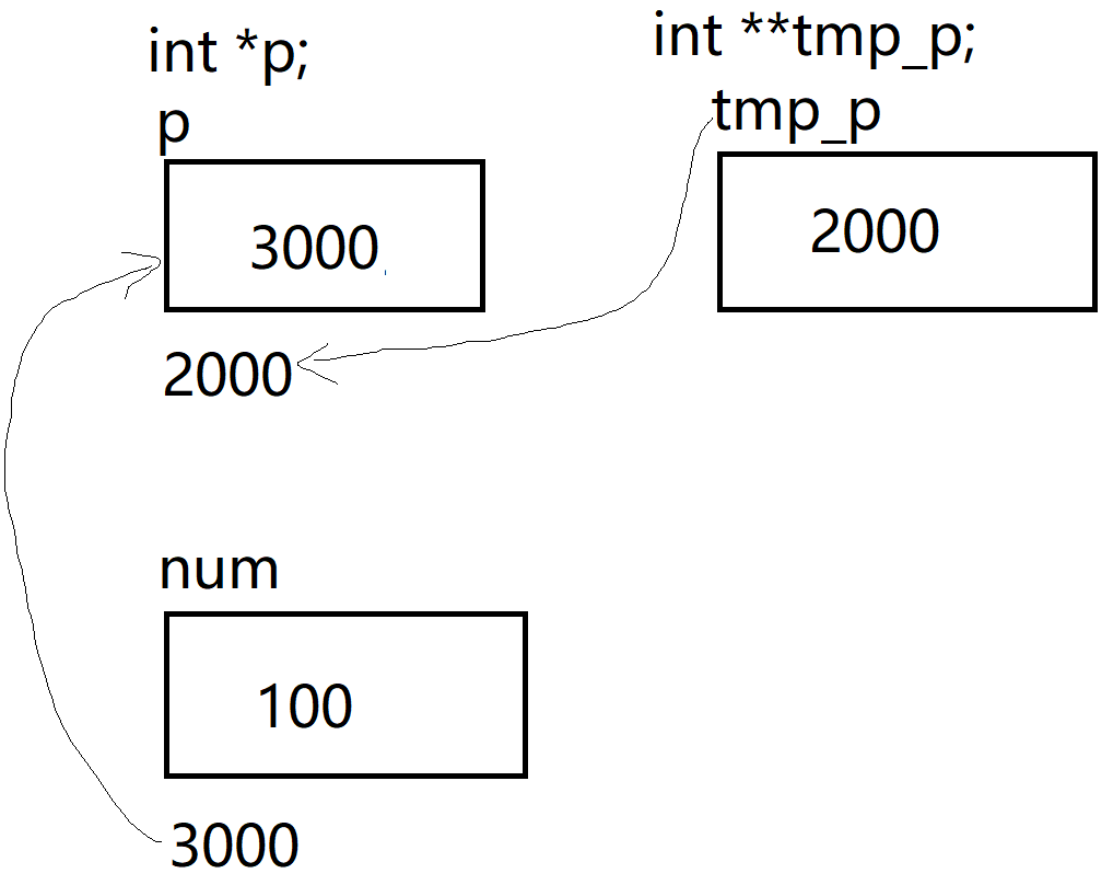
```
data1=10, data2=20
data1=20, data2=10
Press any key to continue
```

案例：

```c
void my_set_p(int **tmp_p)//tmp_p = &p;
{
 static int num = 100;
 //*tmp_p == p
 *tmp_p = &num;//p = & num;

}
void test08()
{
  int *p = NULL;

  //在函数内部 更改p的指向(在函数内部 给p赋值 就必须传递p的地址)
  my_set_p(&p);

  printf("*p = %d\n", *p);//100

 }
int main(int argc,char *argv[])
{
 test08();
 return 0;
}
```

运行结果：

```
*p = 100
Press any key to continue
```



总结：函数内部修改外部变量的值 请传外部变量的地址.

外部变量为0级指针　　函数的形参为1级指针

外部变量为1级指针　　函数的形参为2级指针

外部变量为2级指针　　函数的形参为3级指针

。。。。。。

外部变量为n-1级指针　　函数的形参为n级指针

知识点9【一维数组名作为函数的参数】

1、如果函数内部想操作（读、写）外部数组的元素，请将外部数组的数组名传递函数。（重要！！！）

2、一维数组作为函数的形参　会被优化成一级指针变量。

案例：

```
1  //void my_input_array(int arr[5], int n)
2  //void my_input_array(int arr[], int n)
3  //一维数组 作为函数的形参会被优化成 指针变量
4  void my_input_array(int *arr, int n)
5  {
6   int i=0;
```

```c
 printf("B:%d\n",sizeof(arr));//4
 printf("请输入%d个int数据\n",n);
 for(i=0;i<n;i++)
  {
  scanf("%d", arr+i);
  }
  return;
}
void my_print_array(int *arr, int n)
{
  int i=0;
  for(i=0;i<n; i++)
  {
  //printf("%d ", *(arr+i));
  printf("%d ", arr[i]);

  }
  printf("\n");
}
void test09()
{
  //arr作为类型 代表的是数组的总大小
  //arr作为地址 代表的是数组的首元素地址
  int arr[5]={0};
  int n = sizeof(arr)/sizeof(arr[0]);

  printf("A:%d\n",sizeof(arr));//20

  //定义一个函数 给arr获取键盘输入
  my_input_array(arr, n);

  //定义一个函数 遍历数组元素
  my_print_array(arr, n);
}
int main(int argc,char *argv[])
{
  test09();
  return 0;
}
```

运行结果：

```
A:20
B:4
请输入5个int数据
10 20 30 40 50
10 20 30 40 50
Press any key to cont
```

知识点10【二维数组名作为函数的参数】(了解)

1、如果函数内部想操作（读、写）外部数组的元素，请将外部数组的数组名传递函数。（重要！！！）

2、二维数组名 作为函数的形参 会被优化成 数组指针。

int arr1[5] ------> int *p;

int arr2[3][4]----->int (*p1)[4];

int arr3[3][4][5]---->int (*p2)[4][5];

int arr4[3][4][5][6]--->int (*p3)[4][5][6];

```c
//当二维数组作为函数形参 会被优化成 数组指针
void my_print_two_array(int (*arr)[4], int row, int col)
{
 int i=0,j=0;
 printf("B:%d\n",sizeof(arr));//4
 for(i=0;i<row;i++)
 {
 for(j=0;j<col;j++)
 {
  printf("%d ",arr[i][j]);
  }
  printf("\n");
  }
}
void test10()
```

```
{
 int arr[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
 int row = sizeof(arr)/sizeof(arr[0]);//行数
 int col = sizeof(arr[0])/sizeof(arr[0][0]);//列数

 printf("A:%d\n",sizeof(arr));//48

 my_print_two_array(arr, row, col);

}
int main(int argc,char *argv[])
{
 test10();
 return 0;
}
```

运行结果：

```
A:48
B:4
1 2 3 4
5 6 7 8
9 10 11 12
Press any key to continue
```

知识点11【指针作为函数的返回值】

1、函数不要返回普通局部变量的地址。

```
int* get_addr(void)
{
 int num = 1000;

 return &num;//不要返回普通局部变量地址
}
void test11()
{
```

```
 int *p = NULL;

  p = get_addr();

  printf("*p = %d\n", *p);//不确定
}
int main(int argc,char *argv[])
{
  test11();
  return 0;
}
```

2、解决上述问题：

```
int* get_addr(void)
{
  static int num = 1000;

  return &num;//静态变量 函数结束 不会被释放
}
void test11()
{
  int *p = NULL;

  p = get_addr();

  printf("*p = %d\n", *p);//1000
}
int main(int argc,char *argv[])
{
  test11();
  return 0;
}
```

知识点12【函数名 代表的是函数的入口地址】

案例1：

```
int my_add(int a,int b)
{
  return a+b;
```

```c
}
void test12()
{
 //定一个指针变量 保存该函数的入口地址
 //函数指针 本质是指针变量 保存的是函数的入口地址
 int (*p)(int,int)=NULL;
 printf("%d\n",sizeof(p));

 //my_add代表的是函数的入口地址
 printf("%p\n", my_add);//00401069

 //将函数指针 和函数名 建立关系
 p = my_add;
 printf("%p\n", p);//00401069

 //函数调用:  函数入口地址+()
 printf("%d\n", my_add(10,20));
 printf("%d\n", p(100,200));

 //对函数指针变量 取* 无意义
 (****************printf)("hello fun\n");

 //0x00401069
 printf("%d\n", ((int(*)(int,int))(0x00401069))(1000,2000));
}
int main(int argc,char *argv[])
{
 test12();
 return 0;
}
```

运行结果:

```
4
00401069
00401069
30
300
hello fun
3000
Press any key to continue
```