知识点1【字符串与指针】（了解）

1、字符数组

2、字符串指针变量

知识点2【指针数组】（了解）

1、数值的指针数组

2、字符的指针数组

3、二维字符数组

知识点3【指针的指针】（了解）

知识点4【数组指针】（了解）

1、数组首元素地址 和 数组首地址

2、数组指针 本质是指针变量 保存的是数组的首地址。

案例1：以下结果

知识点5【二维数组和数组指针的关系】（了解）

1、深入了解二维数组

2、二维数组和一维数组指针的关系

知识点6【多维数组的物理存储】（了解）

知识点7【指针变量作为函数的参数】（重要）

案例1：单向传递 之 值传递

案例2：单向传递 之 传递地址

案例3：以下代码能否交换data1 和 data2的值__否___

案例4：以下代码能否交换data1 和 data2的值__能___

知识点8【数组作为函数参数】（重要）

案例：求数组最大值以及最小值

案例：以下案例输出的结果

知识点9【二维数组作为函数参数】（了解）

知识点10【函数的返回值类型 为指针类型】（了解）

知识点11【函数指针】（了解）

1、函数指针的定义

2、函数指针变量注意：

3、函数指针变量 使用typedef定义：（了解）

知识点12【函数指针 作为函数的参数】（了解）

案例1：my_calc完成加减乘除

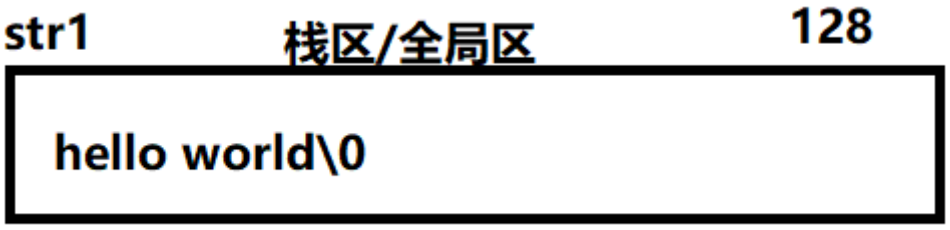
案例2：完成加减乘除

知识点1【字符串与指针】（了解）

1、字符数组

```
char str1[128]="hello world";
```

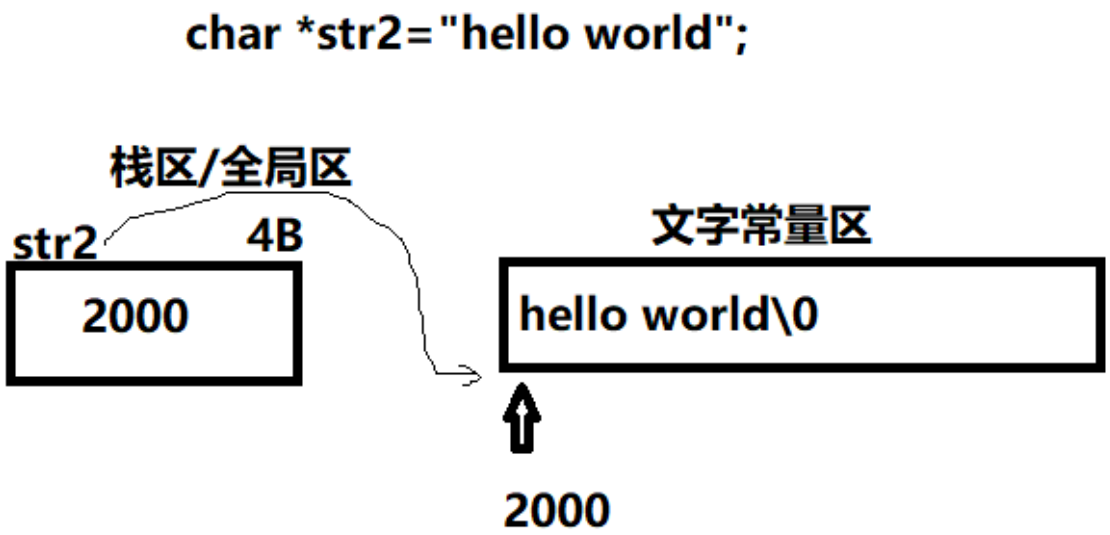
str1是数组 开辟128字节 存放字符串"hello world"




```
sizeof(str1) == 128字节
```

2、字符串指针变量

```
char *str2="hello world";
```




```
sizeof(str2) == 4字节 或 8字节
```

str2是指针变量 在栈区/全局区 保存的是字符串常量"hello world"的首元素地址，而”hello world“在文字常量区

```
#include <stdio.h>
void test01()
{
    char *str2 = "hello world";
    printf("%s\n", str2);     //读
    printf("%c\n", str2[6]); //读

    str2[6] = 'W';//不允许给 文字常量区赋值
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
hello world
w
段错误 (核心已转储)
edu@edu:~/work/c/day05$
```

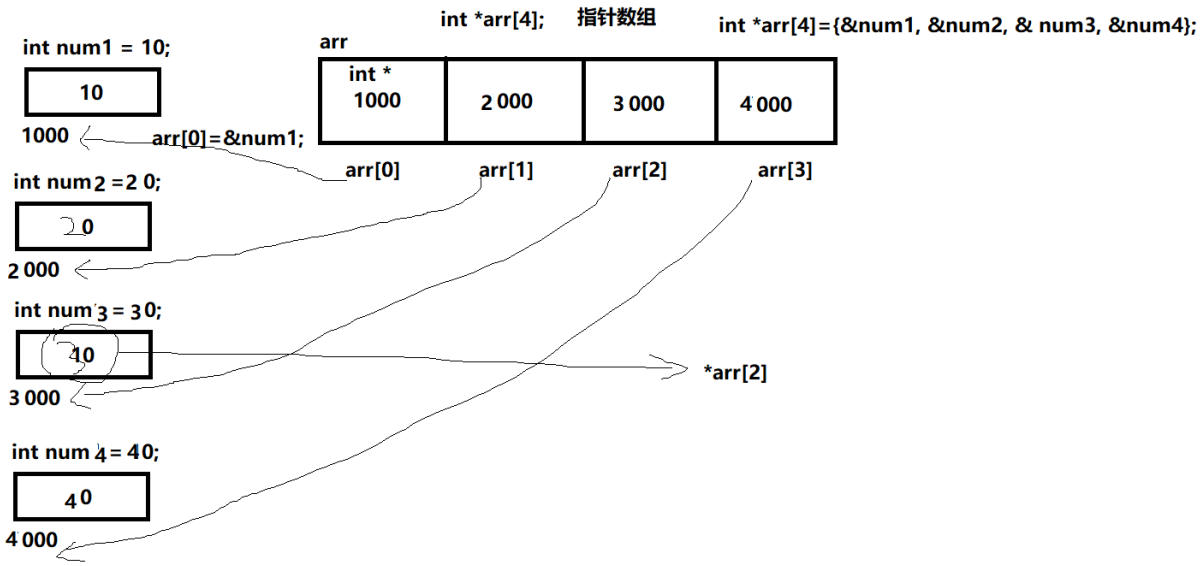
知识点2【指针数组】（了解）

指针数组：本质是数组 只是数组的每个元素为 指针。

32位平台：

```
char *arr1[4];
short *arr2[4];
int *arr3[4];

sizeof(arr1) ==16B
sizeof(arr2) ==16B
sizeof(arr3) ==16B
```

1、数值的指针数组

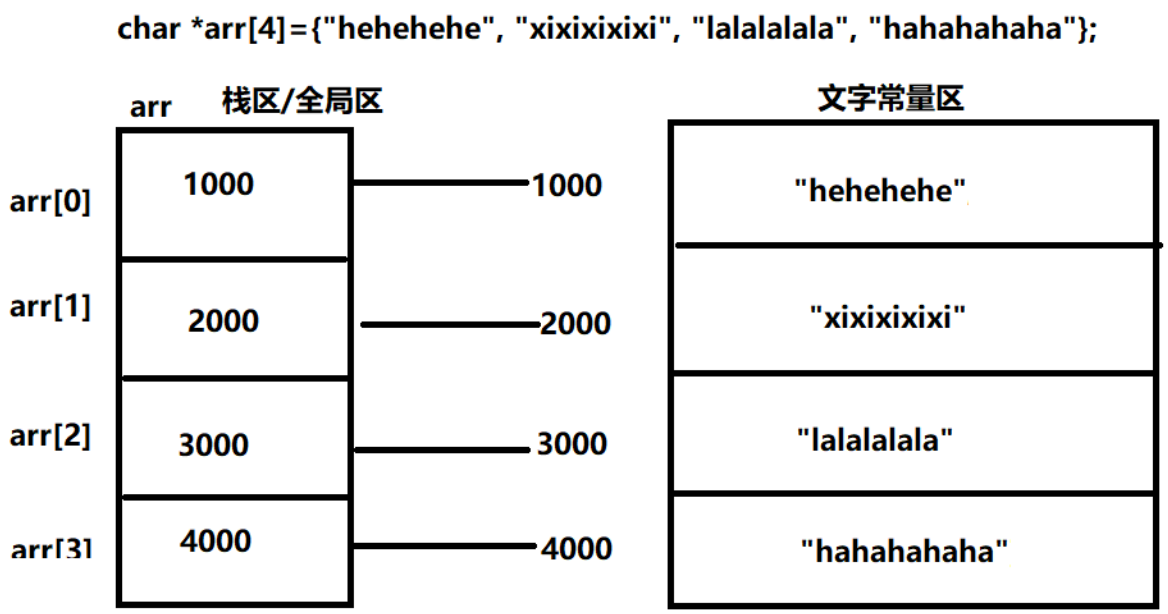



```c
void test02()
{
    int num1 = 10;
    int num2 = 20;
    int num3 = 30;
    int num4 = 40;

    int *arr[4] = {&num1, &num2, &num3, &num4};
    int n = sizeof(arr) / sizeof(arr[0]);

    int i = 0;
    for (i = 0; i < n; i++)
    {
        printf("%d ", *arr[i]);
    }
    printf("\n");
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 01_code.c
[sudo] edu 的密码：
edu@edu:~/work/c/day05$ ./a.out
10 20 30 40
edu@edu:~/work/c/day05$
```

2、字符的指针数组

```c
char *arr[4]={"hehehehe", "xixixixixi", "lalalalala", "hahahahaha"};
```




```c
void test03()
{
    char *arr[4] = {"hehehehe", "xixixixixi", "lalalalala", "hahahahaha"};
    int n = sizeof(arr) / sizeof(arr[0]);

    int i = 0;
    for (i = 0; i < n; i++)
    {
        printf("%s\n", arr[i]);
    }
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 01_code.c
edu@edu:~/work/c/day05$ ./a.out
hehehehe
xixixixixi
lalalalala
hahahahaha
edu@edu:~/work/c/day05$
```

3、二维字符数组

```c
char *arr1[4]={"hehehehe", "xixixixixi", "lalalalala", "hahahahaha"};
char arr2[4][128]={"hehehehe", "xixixixixi", "lalalalala", "hahahahaha"};
```

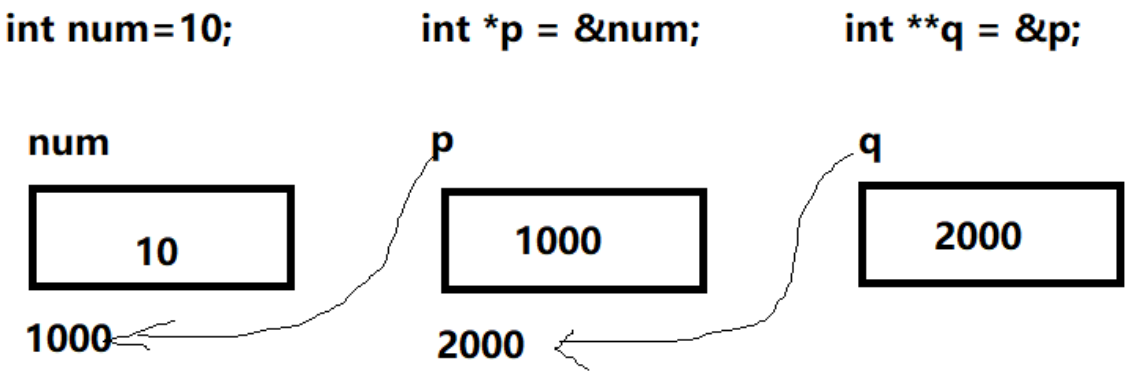
arr1是在指针数组　存放的是每个字符串的首元素的地址

arr2是二维字符数组　存放的是每个字符串

知识点3【指针的指针】（了解）

```c
int num = 10;
int *p = &num;
int **q = &p;
```

n级指针变量 可以保存　n-1级的地址



知识点4【数组指针】（了解）

1、数组首元素地址 和 数组首地址

```
1  int arr[5]={10, 20, 30,40,50};
```

数组首元素地址：&arr[0] == arr　+1跳过一个元素

```
void test04()
{
    int arr[5] = {10, 20, 30, 40, 50};

    //arr首元素地址 +1 跳过一个元素
    printf("arr=%u\n", arr);
    printf("arr+1=%u\n", arr + 1);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
arr=3363584032
arr+1=3363584036
edu@edu:~/work/c/day05$
```

数组的首地址：&arr　+1跳过整个数组

```
void test04()
{
    int arr[5] = {10, 20, 30, 40, 50};

    //&arr首地址 +1 跳过一个数组
    printf("arr=%u\n", arr);
    printf("&arr=%u\n", &arr);
    printf("&arr+1=%u\n", &arr + 1);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
arr=2957050304
&arr=2957050304
&arr+1=2957050324
edu@edu:~/work/c/day05$
```

数组首元素地址编号 和数组的首地址编号 一致。但是他们的类型不同

2、数组指针 本质是指针变量 保存的是数组的首地址。

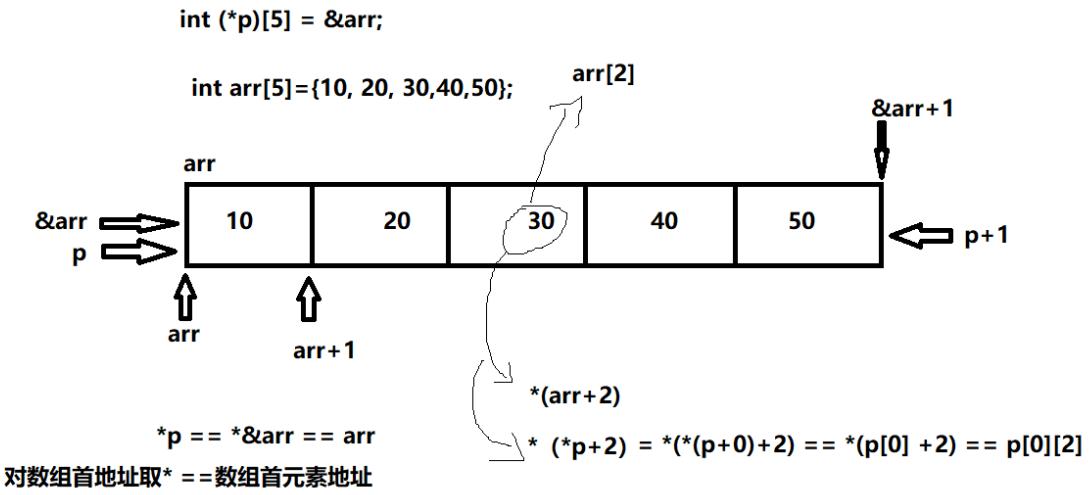
```
1  int arr[5]={10, 20, 30,40,50};
```

```c
int (*p)[5] = NULL;//数组指针
```

```c
void test05()
{
    int arr[5] = {10, 20, 30, 40, 50};

    //数组指针 本质是指针变量
    int(*p)[5] = NULL;
    printf("sizeof(p) =%lu\n", sizeof(p)); //8B

    printf("p = %u\n", p);
    printf("p+1 = %u\n", p + 1);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
sizeof(p) =8
p = 0
p+1 = 20
edu@edu:~/work/c/day05$
```

```c
int arr[5]={10, 20, 30,40,50};
int (*p)[5] = &arr;//数组指针
```




```c
void test05()
{
    int arr[5] = {10, 20, 30, 40, 50};

    //数组指针 本质是指针变量
    int(*p)[5] = NULL;
    printf("sizeof(p) =%lu\n", sizeof(p)); //8B

    p = &arr;
    printf("arr[2] = %d\n", *(*p + 2));
}
int main(int argc, char const *argv[])
{
    test05();
    return 0;
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 01_code.c
edu@edu:~/work/c/day05$ ./a.out
sizeof(p) =8
arr[2] = 30
edu@edu:~/work/c/day05$
```

案例1：以下结果

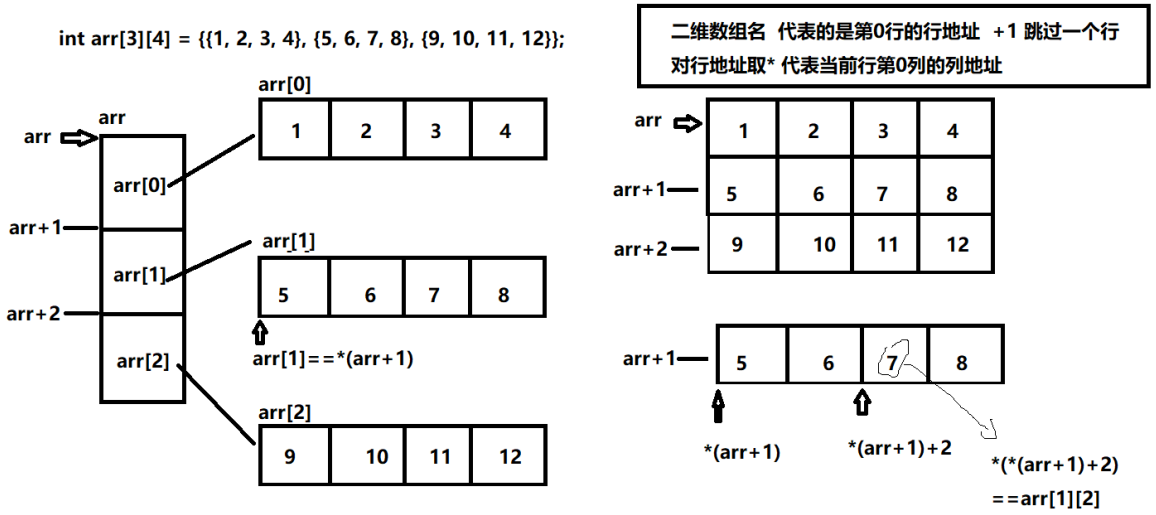
```
int arr[5]={10, 20, 30,40,50};
int (*p)[5] = &arr;//数组指针

printf("%d", *((int *)(p+1)-2) );//40
```

总结：

```
int *arr[5];//指针数组 本质是数组 每个元素为int *
int (*arr)[5];//数组指针 本质是指针变量 保存的是数组的首地址（概数组必须5个元素
每个元素为int）
```

知识点5【二维数组和数组指针的关系】（了解）

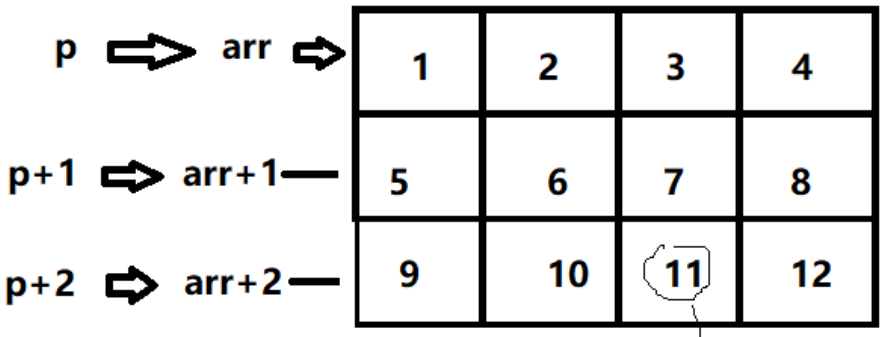
1、深入了解二维数组




```
arr[1] => *(arr+1) 第一行第0列的列地址
&arr[1] => &*（arr+1）=>arr+1 第1行的行地址
*arr+1 => 第0行第1列的列地址
arr[1]+2 =>*(arr+1)+2 =>第1行第2列的列地址
**arr ==*(*(arr+0)+0) == arr[0][0]
```

2、二维数组和一维数组指针的关系

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



int (*p) [4] = arr;

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

一维数组指针 和 二维数组名是 完全等价。

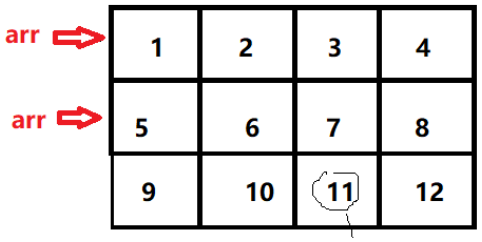
```
int arr[n]; int *p;
int arr[n][m]; int (*p)[m];
int arr[n][m][k]; int (*p)[m][k]
n维数组 和n-1维的数组指针 等价
```

知识点6【多维数组的物理存储】（了解）

不管几维数组在物理上 都是一维存储

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

逻辑图



```c
void test06()
{
    int arr[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
    int row = sizeof(arr) / sizeof(arr[0]);
    int col = sizeof(arr[0]) / sizeof(arr[0][0]);

    int *p = &arr[0][0];

    int i = 0;
    for (i = 0; i < row * col; i++)
    {
        printf("%d ", p[i]);
    }
    printf("\n");
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 01_code.c
[sudo] edu 的密码：
edu@edu:~/work/c/day05$ ./a.out
1 2 3 4 5 6 7 8 9 10 11 12
edu@edu:~/work/c/day05$
```

知识点7【指针变量作为函数的参数】（重要）

如果想在函数内部 修改外部变量的值 需要将外部变量的地址 传递给函数。（重要）

案例1：单向传递 之 值传递

函数内部 不能修改外部变量的值

```c
void set_data01(int d) //int d = num
{
    d = 100;
    return;
}
void test01()
{
    int num = 0;

    set_data01(num); //传值

    printf("num = %d\n", num);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
num = 0
edu@edu:~/work/c/day05$
```

案例2：单向传递 之 传递地址

函数内部 就可以修改 外部变量的值

```c
void set_data02(int *p) //int *p = &num;
{
    //p == &num    --> *p = num
    *p = 100; //num = 100
}
void test01()
{
    int num = 0;

    //set_data01(num); //传值
    set_data02(&num);

    printf("num = %d\n", num);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
num = 100
edu@edu:~/work/c/day05$
```

案例3：以下代码能否交换data1 和 data2的值__否___

```c
void swap(int a, int b)
{
 int tmp= a; a=b;b=tmp;
}
swap(data1, data2);
```

案例4：以下代码能否交换data1 和 data2的值__能___

```c
void swap(int *a, int *b)
{
 //*a == data1 *b == data2
 int tmp= *a; *a=*b;*b=tmp;
}
swap(&data1, &data2);
```

知识点8【数组作为函数参数】（重要）

函数内部 想操作（读或写）外部数组元素，将数组名 传递给函数。

```c
//一维数组作为函数的形参  会被编译器 优化成  指针变量
//void printf_int_array(int p[5], int n)
void printf_int_array(int *p, int n)
{
    printf("sizeof(p)=%lu\n", sizeof(p)); //8B
    int i = 0;
    for (i = 0; i < n; i++)
    {
        //printf("%d ", *(p+i));
         printf("%d ", p[i]);
     }
```

```c
    printf("\n");
}
void test02()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("sizeof(arr)=%lu\n", sizeof(arr)); //20B
    printf_int_array(arr, n);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
sizeof(arr)=20
sizeof(p)=8
10 20 30 40 50
edu@edu:~/work/c/day05$
```

案例：求数组最大值以及最小值

```c
void input_int_array(int *p, int n)
{
    printf("请输入%d个int数据:", n);
    int i = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", p + i);
    }
    return;
}
void get_max_min_data(int *arr, int *p_max, int *p_min, int n)
{
    int max = arr[0], min = arr[0];

    int i = 0;
    for (i = 1; i < n; i++)
    {
        if (max < arr[i])
            max = arr[i];
```

```
        if (min > arr[i])
            min = arr[i];
    }

    //*p_max = 外部max  *p_min==外部的min
    *p_max = max;
    *p_min = min;

    return;
}
void test03()
{
    int arr[5] = {0};
    int n = sizeof(arr) / sizeof(arr[0]);

    //获取键盘输入
    input_int_array(arr, n);

    //求最大最小值
    int max = 0, min = 0;
    get_max_min_data(arr, &max, &min, n);
    printf("max=%d, min=%d\n", max, min);
}
```

```
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
[sudo] edu 的密码:
edu@edu:~/work/c/day05$ ./a.out
请输入5个int数据:1 3 2 4 5
max=5, min=1
edu@edu:~/work/c/day05$
```

案例：以下案例输出的结果

```
void my_fun(int *p)
{
    p[0] = p[1];
    return;
}
```

```c
void test04()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int n = sizeof(arr) / sizeof(arr[0]);

     int i = 0;
     for (i = 0; i < n - 1; i++)
     {
         my_fun(&arr[i]);
     }

     for (i = 0; i < n; i++)
     {
         printf("%d ", arr[i]);
     }
     printf("\n");
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
20 30 40 50 50
edu@edu:~/work/c/day05$
```

知识点9【二维数组作为函数参数】（了解）

函数内部 想操作 函数外部的二维数组 需要将二维数组名 传递给函数。

二维数组 作为函数的形参 会被优化成 一维的数组指针

int arr[3][4][5] 三维数组 作为函数形参 会被优化成 二维的数组指针 int (*arr)[4][5]

n维数组 作为函数形参 会被优化成 n-1维的数组指针

```c
//二维数组 作为函数的形参 会被优化成 一维的数组指针
//void printf_int_two_array(int arr[3][4], int row, int col)
void printf_int_two_array(int (*arr)[4], int row, int col)
{
    printf("内部sizeof(arr)=%lu\n", sizeof(arr)); //8B
    int i = 0, j = 0;
    for (i = 0; i < row; i++)
    {
```

```
        for (j = 0; j < col; j++)
        {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    return;
}
void test05()
{
    int arr[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int row = sizeof(arr) / sizeof(arr[0]);
    int col = sizeof(arr[0]) / sizeof(arr[0][0]);

    printf("外部sizeof(arr)=%lu\n", sizeof(arr)); //48B
    printf_int_two_array(arr, row, col);
}
```

```
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
外部sizeof(arr)=48
内部sizeof(arr)=8
1 2 3 4
5 6 7 8
9 10 11 12
edu@edu:~/work/c/day05$
```

知识点10【函数的返回值类型 为指针类型】（了解）

将函数内部的合法地址 通过返回值 返回给函数外部使用。

注意：函數不要返回值 普通局部变量的地址

```c
int *get_addr(void)
{
    static int data = 10;

    return &data;
}
void test06()
{
    int *p = NULL;
    p = get_addr(); //p =&data

    printf("*p = %d\n", *p); //*p == data
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
*p = 10
edu@edu:~/work/c/day05$
```

提高：

```c
void get_addr(int **q) //int **q = &p;
{
    static int data = 10;
    //q==&p
    //*q == *&p ==p
    //p = &data
    *q = &data;
    return;
}
void test06()
{
    int *p = NULL;
    get_addr(&p); //p=&data

    printf("*p = %d\n", *p); //*p == data
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
*p = 10
edu@edu:~/work/c/day05$
```

知识点11【函数指针】（了解）

1、函数指针的定义

函数名 代表函数的入口地址；

函数指针：本质是一个指针变量 只是该变量 保存的是函数的入口地址

```c
//函数指针 p只能保存 有两int形参以及int返回值 的函数入口地址
int (*p)(int, int) = NULL;
```

```c
int my_add(int x, int y)
{
    return x + y;
}
void test07()
{
    //函数指针 p只能保存 有两int形参以及int返回值 的函数入口地址
    int (*p)(int, int) = NULL;
    printf("%lu\n", sizeof(p));

    p = my_add; //将my_add的入口地址 赋值给了 p

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

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
8
30
edu@edu:~/work/c/day05$
```

2、函数指针变量注意：

函数指针变量 不要+1 无意义

不要对函数指针变量取* 无意义

```c
int (*p)(int, int) = my_add;
*p会被编译器优化成p
```

```c
int my_add(int x, int y)
{
    return x + y;
}
void test07()
{
    //函数指针 p只能保存 有两int形参以及int返回值 的函数入口地址
    int (*p)(int, int) = NULL;
    printf("%lu\n", sizeof(p));

    p = my_add; //将my_add的入口地址 赋值给了 p

    //函数调用: 入口地址+ ()
    //printf("%d\n", my_add(10, 20));
    (**********printf)("%d\n", (**********p)(10, 20));
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_
edu@edu:~/work/c/day05$ ./a.out
8
30
edu@edu:~/work/c/day05$
```

函数指针变量 判断大小 > < 无意义

函数指针变量 可以赋值 p2=p1

函数指针变量 可以判断相等 p2 ==p1

3、函数指针变量 使用typedef定义： （了解）

```c
int my_add(int x, int y)
{
    return x + y;
}
void test07()
{
    //函数指针类型
    typedef int (*FUN_TYPE)(int x, int y);

    FUN_TYPE p = my_add; //将my_add的入口地址 赋值给了 p

    //函数调用: 入口地址+ ()
    //printf("%d\n", my_add(10, 20));
    (**********printf)("%d\n", (**********p)(10, 20));
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_
edu@edu:~/work/c/day05$ ./a.out
30
edu@edu:~/work/c/day05$
```

扩展:

```c
int my_add(int x, int y)
{
    return x + y;
}
void test07()
{
    printf("%p\n", my_add);
    int ret = ((int (*)(int, int))(0x400c0f))(10, 30);
    printf("%d\n", ret);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 02_co
edu@edu:~/work/c/day05$ ./a.out
0x400c0f
40
edu@edu:~/work/c/day05$
```

知识点12【函数指针 作为函数的参数】（了解）

案例1：my_calc完成加减乘除

```c
#include <stdio.h>
int my_add(int x, int y)
{
    return x + y;
}
int my_sub(int x, int y)
{
    return x - y;
}
int my_mul(int x, int y)
{
    return x * y;
}
int my_div(int x, int y)
{
    return x / y;
}

```

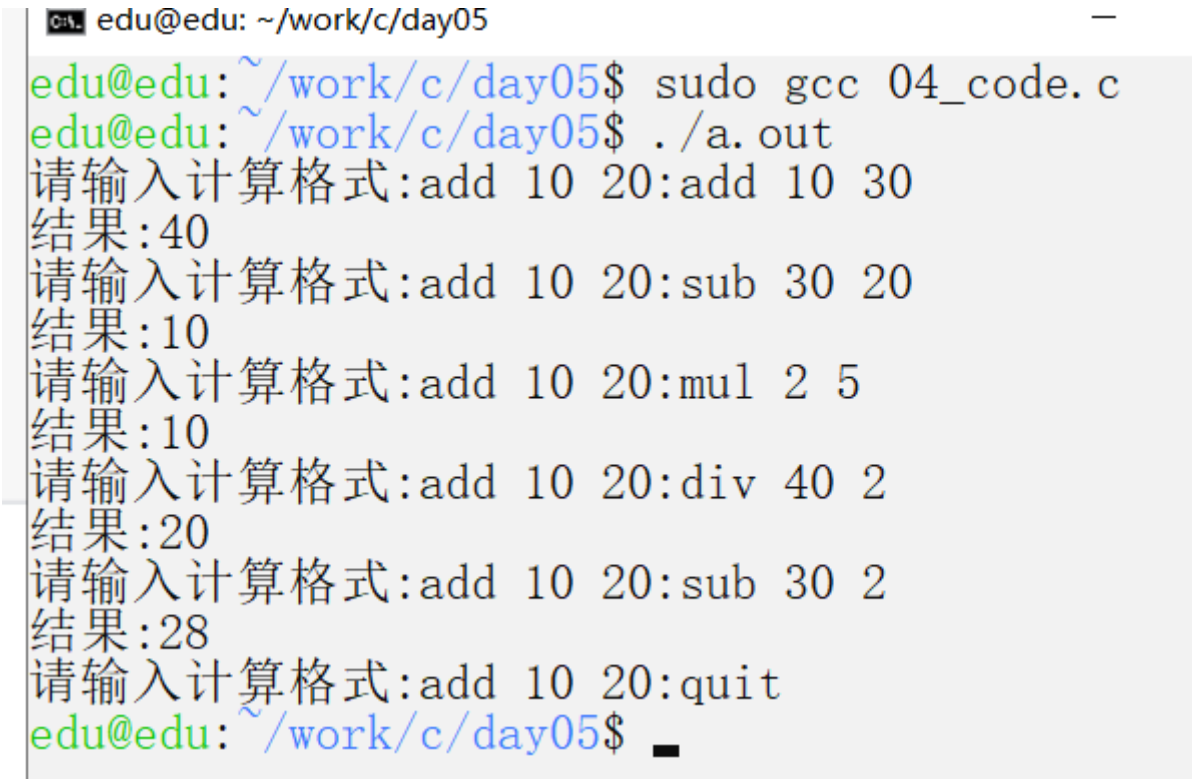
```c
//设计算法 完成加减乘除
int my_calc(int d1, int d2, int (*func)(int, int))
{
    return func(d1, d2);
}
int main(int argc, char const *argv[])
{
    printf("%d\n", my_calc(10, 20, my_add));
    printf("%d\n", my_calc(10, 20, my_sub));
    printf("%d\n", my_calc(10, 20, my_mul));
    printf("%d\n", my_calc(10, 20, my_div));

    return 0;
}
```

```
edu@edu:~/work/c/day05$ sudo gcc 03_code.c
edu@edu:~/work/c/day05$ ./a.out
30
-10
200
0
edu@edu:~/work/c/day05$
```

案例2：完成加减乘除

```c
#include <stdio.h>
#include <string.h>
int my_add(int x, int y)
{
    return x + y;
}
int my_sub(int x, int y)
{
    return x - y;
}
int my_mul(int x, int y)
{
    return x * y;
}
int my_div(int x, int y)
```

```
{
    return x / y;
}

int main(int argc, char const *argv[])
{
    char *cmd_buf[] = {"add", "sub", "mul", "div"};
    int n = sizeof(cmd_buf) / sizeof(cmd_buf[0]);
    //函数指针数组
    int (*fun_arr[])(int, int) = {my_add, my_sub, my_mul, my_div};

    while (1)
    {
        printf("请输入计算格式:add 10 20:");
        char cmd[16] = "";
        scanf("%s", cmd);
        if (strcmp(cmd, "quit") == 0)
            break;

        int data1 = 0, data2 = 0;
        scanf("%d %d", &data1, &data2);

        //算法核心
        int i = 0;
        for (i = 0; i < n; i++)
        {
            //相等为0
            if (strcmp(cmd, cmd_buf[i]) == 0)
            {
                printf("结果:%d\n", fun_arr[i](data1, data2));
                break;
            }
        }
    }

    return 0;
}

```


edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 04_code.c
edu@edu:~/work/c/day05$ ./a.out
请输入计算格式:add 10 20:add 10 30
结果:40
请输入计算格式:add 10 20:sub 30 20
结果:10
请输入计算格式:add 10 20:mul 2 5
结果:10
请输入计算格式:add 10 20:div 40 2
结果:20
请输入计算格式:add 10 20:sub 30 2
结果:28
请输入计算格式:add 10 20:quit
edu@edu:~/work/c/day05$