

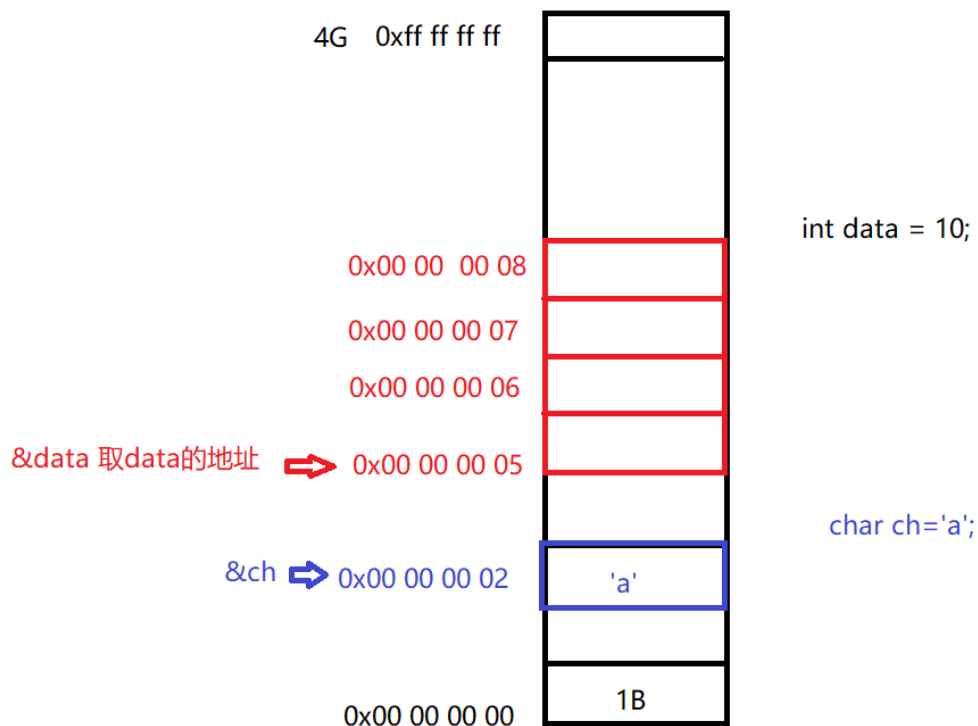
知识点 1【指针的概念】

1、内存的概述

物理内存又叫**内部存储器**，暂时存放数据，掉电数据丢失。

虚拟内存：从物理内存通过操作系统使用 MMU 技术抽象出来的内存的地址。

操作系统 将**内存的每一个字节** 分配一个 **4 字节**或 **8 字节**的编号，而这个**编号**就是**地址编号**。（背）



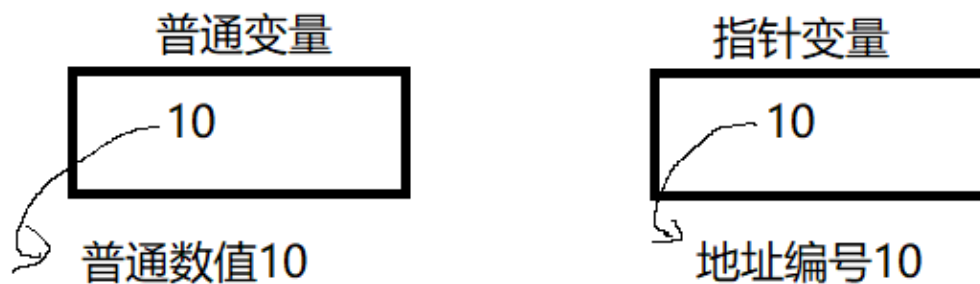
指针：强调的是地址编号的类型。

系统的最小分配单位是**字节**。最小存储单位**二进制位**。（背）

2、指针变量

2.1 指针变量概述

指针变量本质是**变量** 只是该变量 存储的是**地址编号**。



在 32 位系统中 任何指针变量的大小为 4 字节。

在 64 位系统中 任何指针变量的大小为 8 字节。

```
1 #include <stdio.h>
2 void test01()
3 {
4     printf("%ld\n", sizeof(char *));
5     printf("%ld\n", sizeof(short *));
6     printf("%ld\n", sizeof(int *));
7     printf("%ld\n", sizeof(long *));
8     printf("%ld\n", sizeof(float *));
9     printf("%ld\n", sizeof(double *));
10    printf("%ld\n", sizeof(int *****));
11 }
```

```
edu@edu:~/work/c/day05$ ls
00_code.c
edu@edu:~/work/c/day05$ gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
8
8
8
8
8
8
8
edu@edu:~/work/c/day05$
```

2.2 指针变量的定义

- 1、*修饰指针变量名
- 2、保存谁的地址就定义谁
- 3、从上往下 整体 替换

案例 1：定义指针变量 p 保存 int data 的地址

```
int *p;
```

案例 2：定义指针变量 p 保存 int arr[5]的首地址

```
int (*p)[5] 数组指针变量、
```

案例 3：定义指针变量 p 保存 int arr[5]的首元素地址

```
int *p;
```

案例 4: 定义指针变量 p 保存 struct stu lucy 的地址

struct stu *p;结构体指针变量

案例 5: 定义指针变量 p 保存 int func(int,int)的函数入口地址

int (*p)(int,int);//函数指针

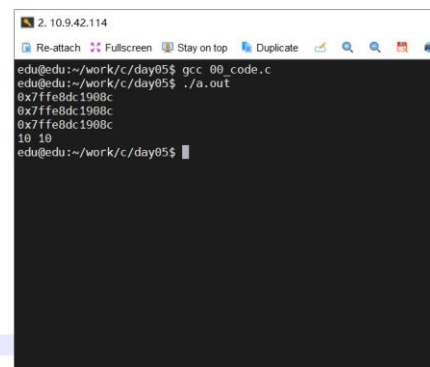
案例 5: 定义指针变量 p 保存数组 arr 的首地址, 该数组有 5 个元素, 每个元素为函数的

入口地址 该函数为 int func(int,int)

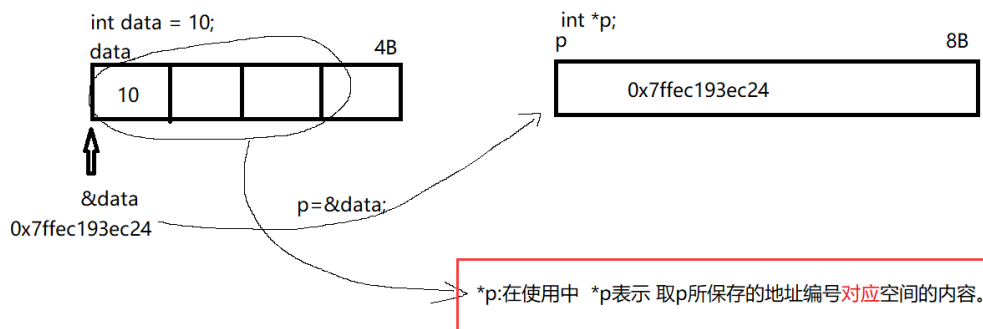
int (*p)[5](int,int)//函数指针数组的指针变量

3、指针变量和普通变量建立关系

```
13 void test02()
14 {
15     int data = 10; //4字节合法地址
16     //取变量的地址
17     printf("%p\n", &data);
18
19     //定义指针变量p 保存&data
20     //不要操作 没有合法指向的指针变量
21     int *p; //定义的时候*仅仅修饰p为指针变量 (重要!!!!)
22
23     //建立p和data地址的关系
24     p = &data;
25     //p保存了data的地址
26     //p指向了data
27     //p保存。。。的地址 就是 p指向了。。。
28
29
30     //p保存的是data的地址 所以 p的值 就是data的地址编号
31     printf("%p\n", p);
32     printf("%p\n", &data);
33
34     printf("%d %d\n", data, *p);
35 }
```



```
2. 10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day05$ gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
0x7ffe8dc1908c
0x7ffe8dc1908c
0x7ffe8dc1908c
10 10
edu@edu:~/work/c/day05$
```



4、指针变量的初始化

```
void test01()
{
    // p1 是局部指针变量 不初始化 指向不确定的空间 一旦操作 容易访问非法内存

    int *p1;

    int data = 10;

    // p2 指向了 data &data 赋值给 p2, 而不是*p2 这是定义语句*描述 p2 为指针变量

    int *p2 = &data;

    int num = 10, *p3 = &num;

    // NULL 本质为 (void *) 0, 如果指针变量初始化为 NULL 不要立即操作

    int *p4 = NULL;

    int data1 = 10;

    p4 = &data1;
```

```

printf("*p4=%d\n", *p4);

int data2 = 20;

p4 = &data2;

printf("*p4=%d\n", *p4);
}

```

知识点 2【指针变量的类型】（重要）

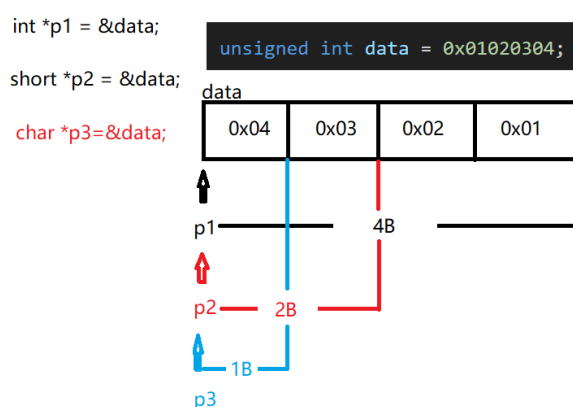
1、指针变量的类型分类

`int *p;` `p` 自身的类型为 `int *` 指向的类型为 `int`

指针变量**自身**的类型：只将**指针变量名**拖黑 剩下啥类型 **自身**为啥类型。（赋值语句左右判断）

指针变量**指向**的类型：只将**指针变量名**和离它最近的一个***** 一起拖黑，剩下啥类型 **指向**啥类型。（决定了指针变量的跨度、取值宽度）

2、指向类型决定**取值宽度**。



```

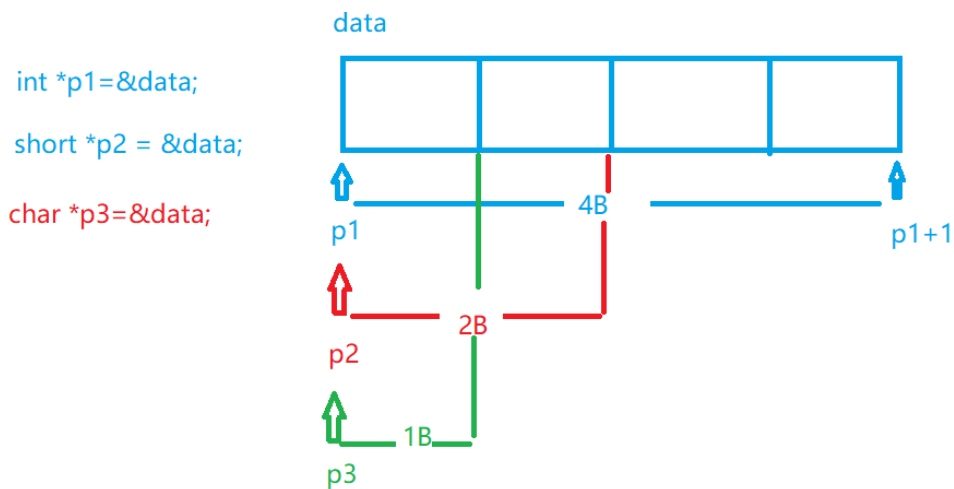
void test02()
{
    unsigned int data = 0x01020304;
    unsigned int *p1 = &data;
    printf("%#x\n", *p1); // 0x1020304

    short *p2 = &data;
    printf("%#x\n", *p2); // 0x304

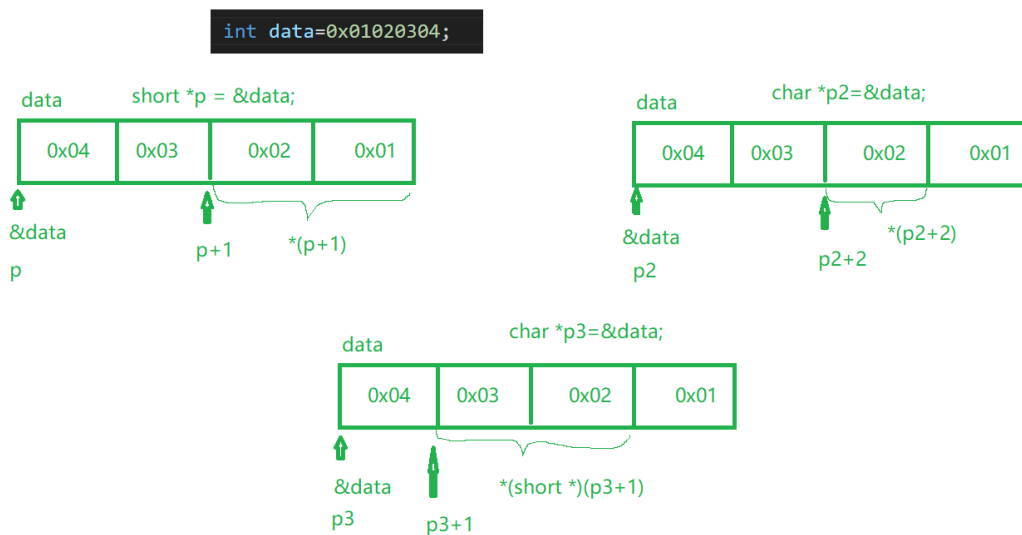
    char *p3 = &data;
    printf("%#x\n", *p3); // 0x4
}

```

3、指向类型决定单位跨度。



综合练习：



知识点 3 【使用中&和*的关系】

```
int data = 10;
```

data 的类型为 int

&data 的类型为 int *

在使用中 &让表达式+*

```
int *p = &data;
```

p 的类型是 int *

*p 的类型是 int

在使用中*让表达式-*

总结：在使用中，&和*同时出现 从右往左 依次抵消

```
*&*p == *p
```

知识点 4【指针变量的注意事项】

1、void 不能直接定义变量

void num;系统无法根据 void 类型 确定 num 的空间大小 所以定义失败

2、void *可以定义指针变量

void *p; p 的类型为 void *, 在 32 位平台 void *为 4 字节，系统能够为 p 开辟 4 字节空间，所以定义成功。

不能直接对 p 取*， *p 报错 不能通过 void 决定取值宽度，所以*p 失败

不能让 p+1， p+1 报错 不能通过 void 决定单位跨度，所以 p+1 失败

void *p 是万能指针变量， 可以保存任何一级地址。一般用于函数形参 让算法通用。

如果要是用 p 取值或跳跃 必须实现对 p 进行 强制类型转换。

3、不要对未初始化的指针变量取值

```
int *p;
```

```
*p;//访问非法内存
```

4、不要对初始化为 NULL 的指针变量取值

```
int *p=NULL;
```

`*p; //访问非法内存`

5、操作指针变量不要越界

```
int data=10;
```

```
int *p=&data;
```

```
p++;
```

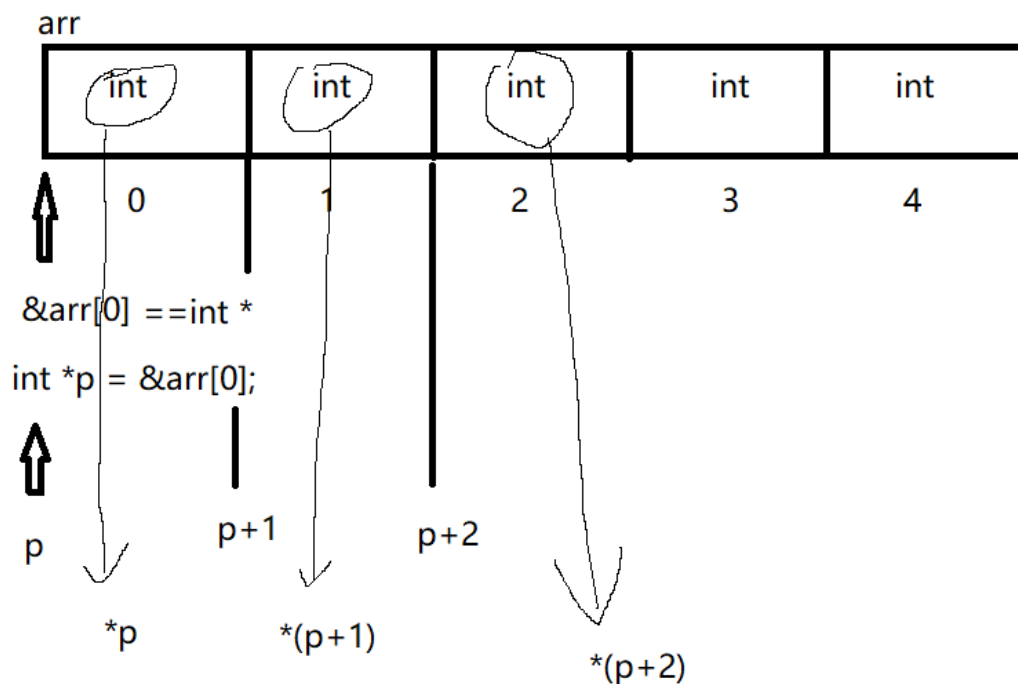
`*p;`访问越界

知识点 5【数组元素的指针变量】

1、数组元素指针变量的概述

数组元素的指针变量+1 跳过一个元素，取*取一个元素的值。

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



注意：操作数组元素的指针变量的时候 一定要确定 `p` 指向的是哪个元素（决定的是 `p` 的

其实位置)

案例 1: `int arr[5]={10,20,30,40,50}, *p = &arr[1];` 请问 `*(p+2)` 的值__40__

案例 2: `int arr[5]={10,20,30,40,50}, *p = &arr[0];`

`p++;`

请问 `*(p+2) == 40`

2、一维数组名 作为地址 代表的是数组的首元素地址。

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

    int n = sizeof(arr) / sizeof(arr[0]);

    // 数组名作为类型 代表的是数组的总大小 == sizeof(arr)

    // 数组名作为地址 代表的是数组的首元素地址 arr == &arr[0]

    printf("&arr[0] = %p\n", &arr[0]); // 0x7ffdf2afb0e0

    printf("arr = %p\n", arr);        // 0x7ffdf2afb0e0

    printf("arr+1 = %p\n", arr + 1);  // 0x7ffdf2afb0e4


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

    int *p = arr;

    int i = 0;

    for (i = 0; i < n; i++)

    {
```

```
    //printf("%d ", *(p + i));

    printf("%d ", *(arr + i));

    //printf("%d ", arr[i]);

}

printf("\n");

}
```

```
86 void test06()
87 {
88     int arr[5] = {10, 20, 30, 40, 50};
89     int *p = arr;
90
91     //p本质是变量 可以被赋值
92     p++;
93     printf("*p=%d\n", *p); //20
94
95     //arr++; //arr=arr+1 arr是数组名是符号常量 不能被赋值
96 }
```

```

98 void test07()
99 {
100     int arr[5] = {10, 20, 30, 40, 50};
101     int n = sizeof(arr) / sizeof(arr[0]);
102     int *p = arr;
103     int i = 0;
104     for (i = 0; i < n; i++)
105     {
106         // scanf("%d", &arr[i]);
107         // scanf("%d", p + i);
108         scanf("%d", arr + i);
109     }
110     for (i = 0; i < n; i++)
111     {
112         printf("%d ", arr[i]);
113     }
114     printf("\n");
115 }
116 int main(int argc, char *argv[])

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

edu@edu:~/work/c/day05\$ gcc 01_code.c

edu@edu:~/work/c/day05\$./a.out

1 2 3 4 5

1 2 3 4 5

edu@edu:~/work/c/day05\$

3、指向同一数组的两个元素指针变量的关系（重要）

```

117 void test08()
118 {
119     int arr[5] = {10, 20, 30, 40, 50};
120     int *p1 = &arr[0];
121     int *p2 = &arr[3];
122
123     printf("%d\n", p2 - p1); //3
124 }

```

- 1、指向同一数组的元素指针变量**相减** 得到的是相差元素的**个数**。
- 2、指向同一数组的元素指针变量**相加** 无意义。
- 3、指向同一数组的元素指针变量**判断相等** `==` 判断是否指向同一个元素。
- 4、指向同一数组的元素指针变量**判断大小** 判断位置关系。
- 5、指向同一数组的元素指针变量**赋值** `p2=p1` 让 `p2` 和 `p1` 指向同一处。

知识点 6 【[]和*()的关系】

```
126 void test09()
127 {
128     int arr[5] = {10, 20, 30, 40, 50};
129     printf("arr[1]=%d\n", arr[1]);
130     printf("*(arr+1)=%d\n", *(arr + 1));
131     printf("-----\n");
132     printf("*(arr+1)=%d\n", *(1 + arr));
133     printf("*(arr+1)=%d\n", 1 [arr]); //千万别这样写
134     //arr[i]的展开 *(arr+i) []是* () 的缩写
135     arr=&arr[0] == &*(arr+0) == arr+0==arr
136 }
```

案例 1:

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

```
int *p = arr+1;
```

p[2]的值为__40__

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

```
int *p = arr+3;
```

p[-2]的值为__20__

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

```
int *p=arr;
```

以下代表数组元素的是__C__

A:arr[5] B:arr+1 C:*p D:p+1

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

```
int *p=arr;
```

以下代表数组元素地址是__D__

A:&arr[5] B:&arr C:&p D:p

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

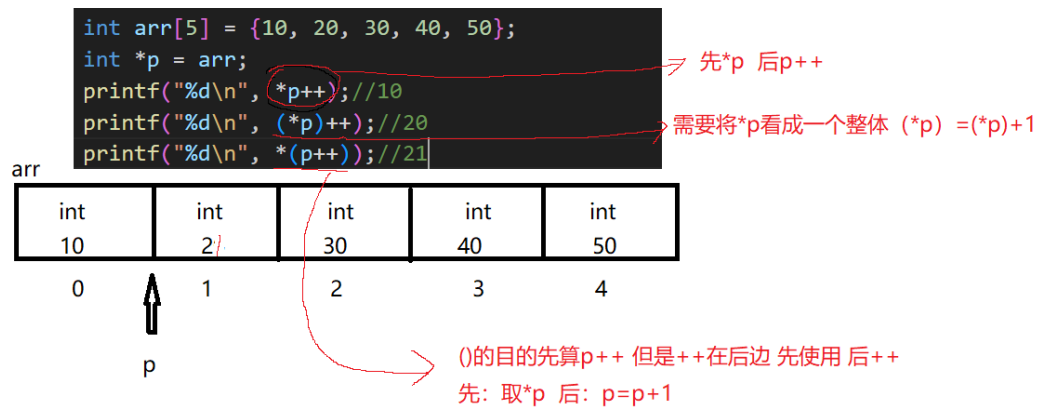
```
int *p=arr;
```

```
printf("%d\n", *p++);
```

```
printf("%d\n", (*p)++);
```

```
printf("%d\n", *(p++));
```

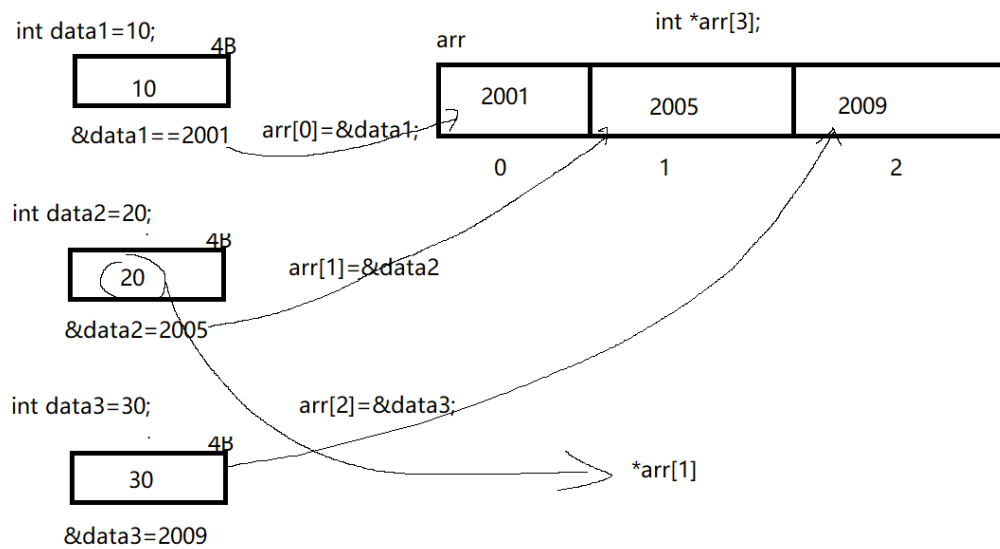
```
138 void test10()  
139 {  
140     int arr[5] = {10, 20, 30, 40, 50};  
141     int *p = arr;  
142     printf("%d\n", *p++); //10  
143     printf("%d\n", (*p)++); //20  
144     printf("%d\n", *(p++)); //21  
145 }
```



知识点 7 【指针数组】 (重要)

1、指针数组 的概述

指针数组：本质是数组 只是该数组的每个元素 存放的是地址编号。（背）

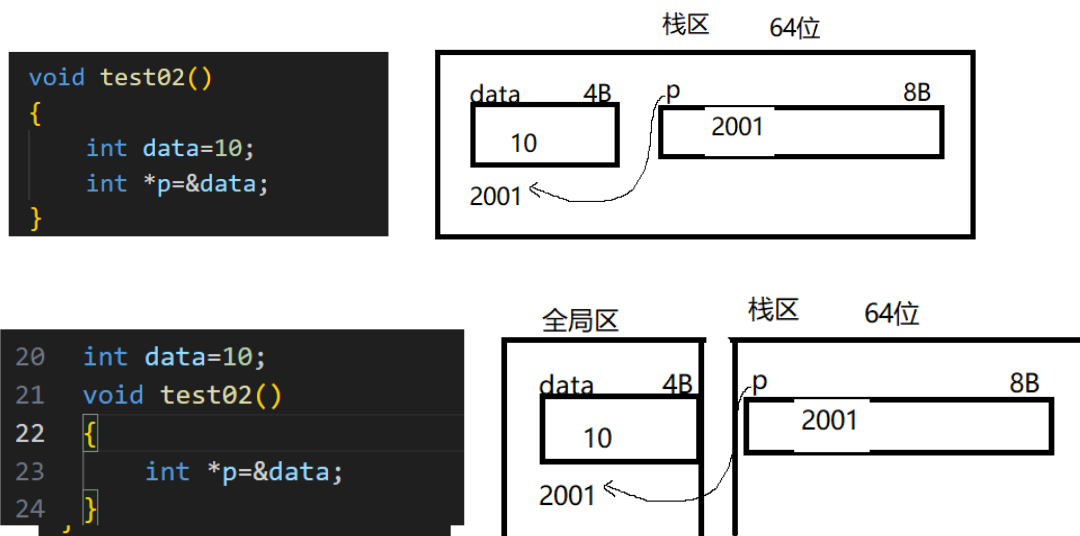


```
1  #include <stdio.h>
2  int main(int argc, char const *argv[])
3  {
4      int data1 = 10;
5      int data2 = 20;
6      int data3 = 30;
7
8      // 定义指针数组 存放变量的地址编号
9      int *arr[3] = {&data1, &data2, &data3};
10     int n = sizeof(arr) / sizeof(arr[0]);
11     int i = 0;
12     for (i = 0; i < n; i++)
13     {
14         printf("%d ", *arr[i]);
15     }
16     printf("\n");
17
18     return 0;
19 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
edu@edu:~/work/c/day05$ gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
10 20 30
edu@edu:~/work/c/day05$
```

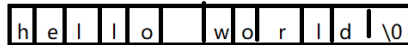
2、指针变量的一个空间分析



3、字符数组和字符指针变量的区别

```
char buf[]="hello world";
```

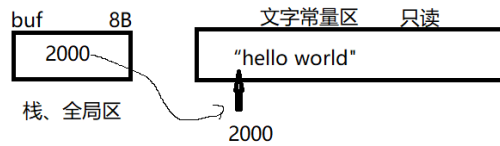
字符数组使用数组的空间存放字符串的每个字符
栈区、全局区



用户可以对数组的元素进行读写

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

buf是一个字符指针变量 保存的是字符串首元素地址
字符串则存放在文字常量



```
buf[1]='E';//error 文字常量只读
```

//字符数组：用数组的空间存放各个字符 可读可写 在栈或全局区

```
char buf[]="hello world";
```

//字符指针变量：只是保存字符串的首元素地址 字符串本身在文字常量区 不同通过指针
变量给字符串赋值

//指针变量 可以在栈或全局区

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

4、字符指针数组

```
30 void test03()
31 {
32     char *arr[5] = {"hehehehe", "hahahaha", "xixixixi", "lalalalala", "henhenhenhen"};
33     int n = sizeof(arr) / sizeof(arr[0]);
34
35     int i = 0;
36     for (i = 0; i < n; i++)
37     {
38         printf("%s\n", arr[i]);
39     }
40
41     printf("%c\n", *(arr[1] + 3));
42     // arr[1]==A
43     printf("%c\n", arr[1][3]);
44
45 }
46 int main(int argc, char const *argv[])
{
    test03();
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
edu@edu:~/work/c/day05$ gcc 02_code.c
edu@edu:~/work/c/day05$ ./a.out
hehehehe
hahahaha
xixixixi
lalalalala
henhenhenhen
a
a
edu@edu:~/work/c/day05$
```

//字符指针数组 存放的是每个字符串首元素地址 而字符串本身存放在文字常量区（只读）

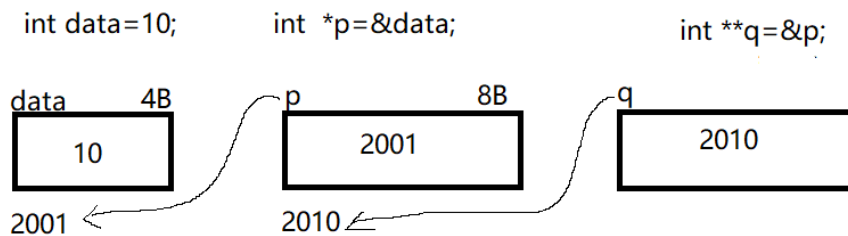

```
char *arr[5] = {"hehehehe", "hahahaha", "xixixixi", "lalalalala", "henhenhenhen"};
```

//二维字符数组：使用每一行存放每个字符串（栈或全局区）

```
char arr[5][128] = {"hehehehe", "hahahaha", "xixixixi", "lalalalala",  
"henhenhenhen"};
```

知识点 8 【指针的指针】

用一个指针变量 保存另一个指针变量的地址编号。

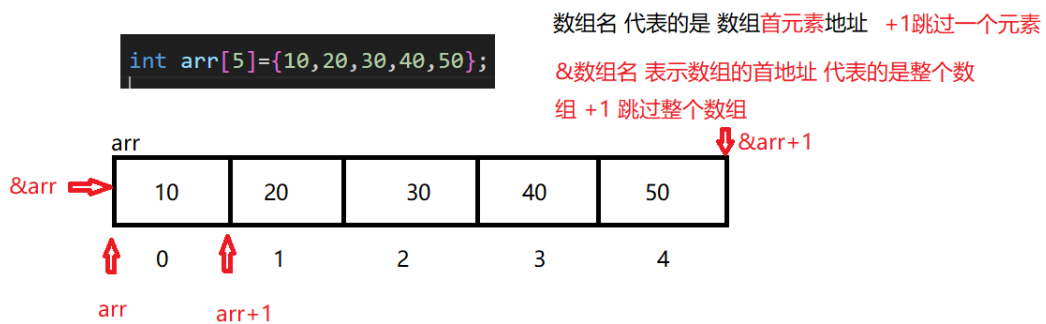


```
q == &p  
*q == p == &data  
**q == *p == &data == data  
  
p == &data  
*p == data
```

知识点 9 【数组指针】

数组指针变量：本质是**指针变量** 保存的是数组的**首地址**。

1、数组的首地址

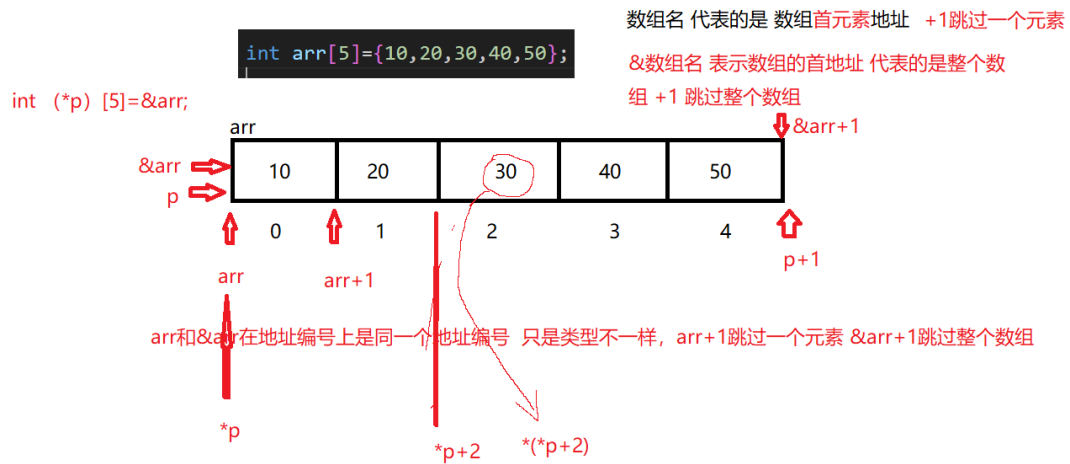


arr和&arr在地址编号上是同一个地址编号 只是类型不一样，arr+1跳过一个元素 &arr+1跳过整个数组

重要：对数组的**首地址取*** 表示的是数组的**首元素地址**。

2、定义一个数组指针变量 保存数组的首地址

int (*p)[5];//数组指针变量



arr和&arr在地址编号上是同一个地址编号 只是类型不一样，arr+1跳过一个元素 &arr+1跳过整个数组

```

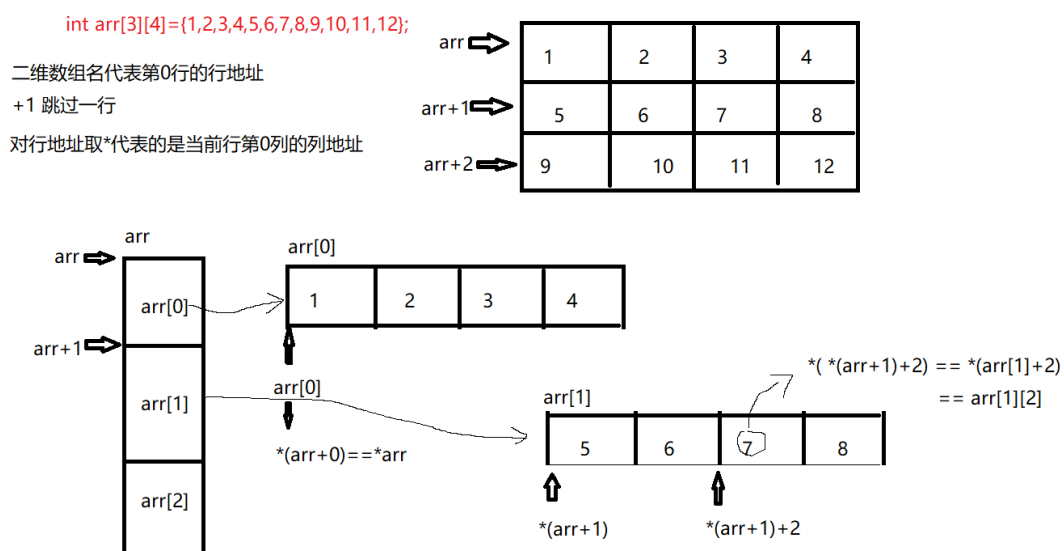
46 void test04()
47 {
48     int arr[5] = {10, 20, 30, 40, 50};
49     int(*p)[5];
50
51     // p自身的类型为:int (*)[5];
52     printf("%lu\n", sizeof(p)); // 8B
53     // p指向的类型为: int [5];
54     printf("%lu\n", p);
55     printf("%lu\n", p + 1);
56
57     // 数组指针变量本质 其实就是保存数组的首地址
58     p = &arr;
59
60     printf("%d\n", *(*p + 2)); // 30
61 }

```

案例 1: `int arr[5] = {10, 20, 30, 40, 50}`, `(*p)[5]=&arr`;则表达式 `* ((int *) (p+1) -2)` 的值为__40__

知识点 10 【数组指针和二维数组的关系】

1、二维数组的详解



`arr[1]+1` 代表的是第 1 行第 1 列的列地址

`*arr+1` 代表的是第 0 行第 1 列的列地址

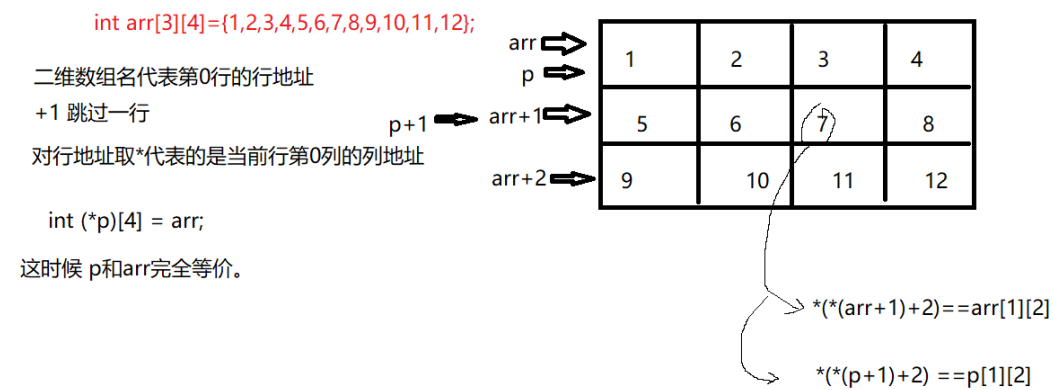
`**arr` 代表的是第 0 行第 0 列的值

$*(&arr[1]+1) == * (arr+2)$ 代表的是第 2 行第 0 列的列地址

$*(arr+1)+1$ 代表的是第 1 行第 1 列的列地址

$*arr[1]+1 == * (* (arr+1)) +1$ 代表的是第 1 行第 0 列的值+1

2、二维数组 和一维数组指针的关系



一维的数组指针 `int (*p)[4]` 和二维的数组名 `int arr[3][4]` 是等价的。（重要）

二维的数值指针 `int (*p)[4][5]` 和三维的数组名 `int arr[3][4][5]` 是等价的（重要）

知识点 11 【指针和函数的关系】

1、指针与函数形参的关系

1、普通变量作为函数的形参 函数内部无法通过形参 修改外部实参 的值。

```
2 void set_data01(int a)//int a=data;
3 {
4     a=1000;
5 }
6 void test01()
7 {
8     int data = 0;
9     set_data01(data);
10    printf("data=%d\n",data); //0
11 }
```

2、指针变量作为函数的参数。

如果函数内部想修改外部实参的值 需要将外部实参的地址传递函数，那么函数的形参就

应该是指针变量。在函数内部使用*指针变量间接的操作外部实参的空间内容。（背）

```
7 void set_data02(int *p) // int *p=&data;
8 {
9     // *p等价data
10    *p = 1000; // data=1000
11 }
12 void test01()
13 {
14     int data = 0;
15     set_data02(&data);
16     printf("data=%d\n", data); // 1000
17 }
```

3、一维数组作为函数的形参 会被优化成指针变量。

```
//void print_int_array(int arr[5], int n)
```

```
void print_int_array(int *arr, int n)
```

```

{

    printf("内部 sizeof(arr)=%ld\n", sizeof(arr)); // 8B

    int i=0;

    for ( i = 0; i < n; i++)

    {

        //printf("%d ", *(arr+i));

        printf("%d ", arr[i]);

    }

    printf("\n");

}

void test02()

{

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

    int n = sizeof(arr) / sizeof(arr[0]);

    printf("外部 sizeof(arr)=%ld\n", sizeof(arr)); // 20B

    print_int_array(arr, n);

}

```

4、数组指针作为函数的参数

```

int arr[4];

int (*p)[4]; //数组指针

p=&arr;

```

```
int arr[3][4];

int (*p)[4]; //数组指针

p=arr;

//void print_int_two_array(int arr[3][4], int row, int col)

void print_int_two_array(int (*arr)[4], int row, int col)
{
    printf("内部 sizeof(arr)=%ld\n", sizeof(arr)); // 8B

    //函数内部使用 arr 等价使用外部的 arr

    int i=0;

    for ( i = 0; i < row; i++)
    {
        int j=0;

        for ( j = 0; j < col; j++)
        {
            printf("%d ", arr[i][j]); // (*(arr+i)+j)

        }

        printf("\n");
    }
}

void test03()
{
```

```

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)=%ld\n", sizeof(arr)); // 48B

print_int_two_array(arr, row, col);

}

```

总结：

如果形参是二维数组 arr[3][4] 优化成 一维数组指针 (*p)[4]

如果形参是三维数组 arr[3][4][5] 优化成 二维数组指针 (*p)[4][5]

如果形参是四维数组 arr[3][4][5][6] 优化成 三维数组指针 (*p)[4][5][6]

• • • • •

如果形参是 N 维数组 arr[3][4][5][6]...[N] 优化成 N-1 维数组指针 (*p)[4][5][6]...[N]

案例 1：需求如下

```

void get_max_min_data(int *arr, int n, int *p_max, int *p_min)
{
    // *p_max 等价 max *p_min=min

    int max = arr[0], min = arr[0];

    int i = 0;

    for (i = 1; i < n; i++)

    {

        max = (max < arr[i] ? arr[i] : max);

        min = (min > arr[i] ? arr[i] : min);
    }
}

```



```
}

// 更新外部变量的值

*p_max = max;

*p_min = min;

return;
}

void input_int_array(int *arr, int n)
{
    printf("请输入%d 个 int 数据:", n);

    int i = 0;

    for (i = 0; i < n; i++)

    {
        scanf("%d", arr + i);
    }

    return;
}

void test04()
{
    int arr[10] = {0};

    int n = sizeof(arr) / sizeof(arr[0]);

    int max = 0, min = 0;
```

```

// 键盘输入 10 个 int 数组 求出最大值和最小值 必须通过函数的形参更新

input_int_array(arr, n);

get_max_min_data(arr, n, &max, &min);

printf("max=%d, min=%d\n", max, min);
}

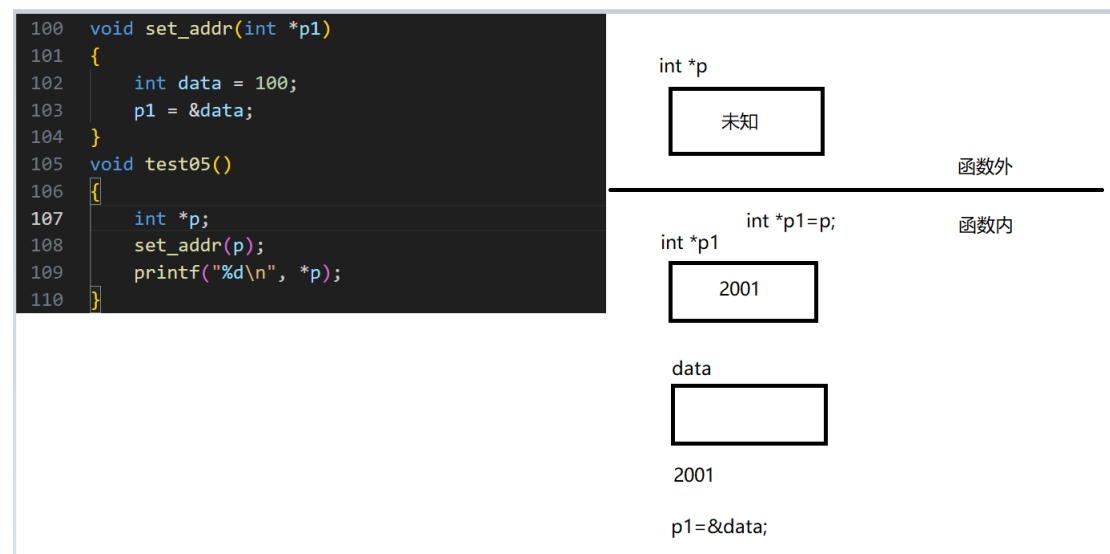
```

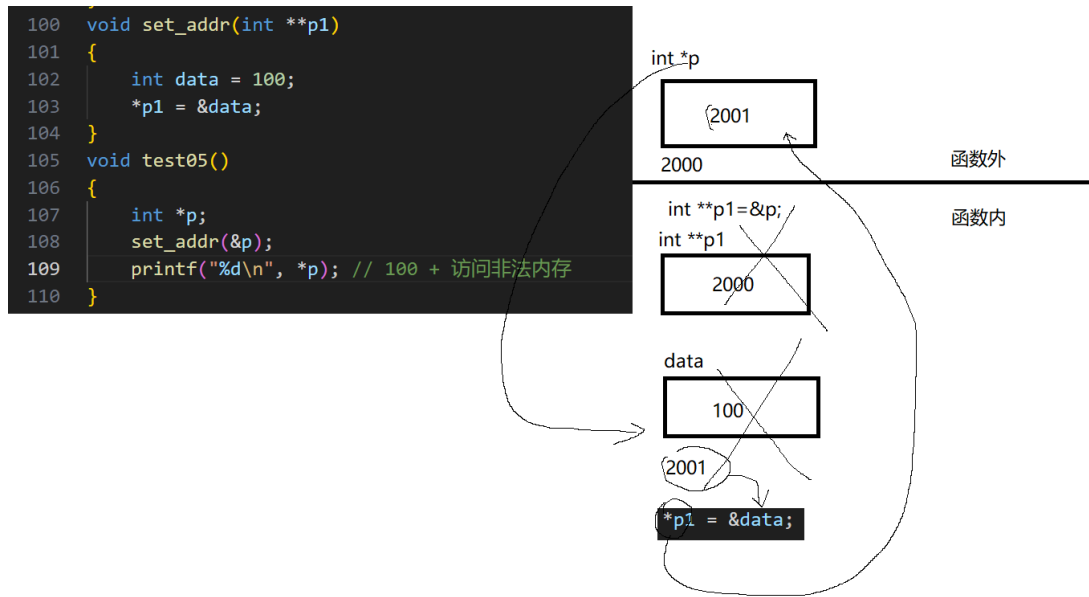
案例 1:

```

100 void set_addr(int *p1)
101 {
102     int data = 100;
103     p1 = &data;
104 }
105 void test05()
106 {
107     int *p;
108     set_addr(p);  函数内部无法修改p的指向
109     printf("%d\n", *p); 访问非法内存出现段错误
110 }

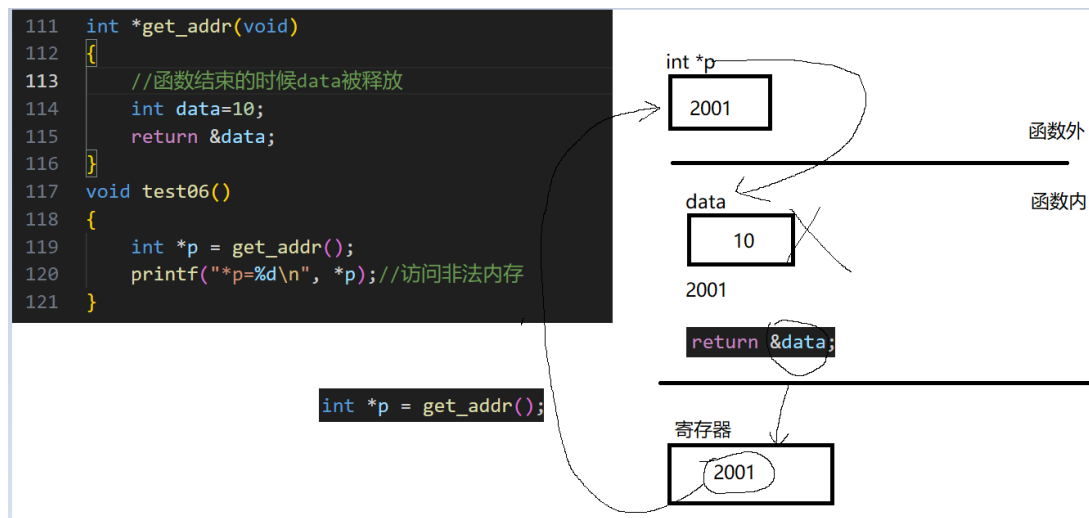
```





2、指针作为函数的返回值。

函数外边需要操作函数内部的地址空间，这时需要将函数内部地址编号通过 返回值返回。



函数不要返回值局部变量或局部数组的地址编号。（函数结束 函数内部的局部变量或局部数组空间将被释放）（重要）

```

111 int *get_addr(void)
112 {
113     // data为静态局部变量 进程结束才释放 函数结束不释放
114     static int data = 10;
115     return &data;
116 }
117 void test06()
118 {
119     int *p = get_addr();
120     printf("*p=%d\n", *p); // 访问非法内存
121 }
122 int main(int argc, char const *argv[])
123 {

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

edu@edu:~/work/c/day05$ gcc 03_code.c
edu@edu:~/work/c/day05$ ./a.out
*p=10
edu@edu:~/work/c/day05$

```

函数返回的地址 一般是堆区的地址编号。（一般函数从堆区申请空间 通过返回值 将地址编号返回 给外部使用堆区空间地址）（重要）

3、函数指针（重要）

1、函数指针的概述

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。

在 c 语言中，函数名 代表的是函数的入口地址。

```
3 int my_add(int x, int y)
4 {
5     return x + y;
6 }
7 void test01()
8 {
9     // 定义一个指针变量p保存my_add的入口地址
10    int (*p)(int x, int y) = NULL;
11    p = my_add;
12    // 如何通过p调用这个my_add函数呢?
13    printf("%d\n", p(100, 200));
14    // 不要对函数指针变量取* 会被编译器优化掉
15    printf("%d\n", (*****p)(100, 200));
16    printf("%d\n", my_add(100, 200));
17 }
18 int main(int argc, char const *argv[])
19 {
20     test01();
21     return 0;
22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● edu@edu:~/work/c/day05$ gcc 04_code.c
● edu@edu:~/work/c/day05$ ./a.out
300
300
300
○ edu@edu:~/work/c/day05$
```

函数指针 一般作用与函数的形参 让你的函数更加通用。(重要)

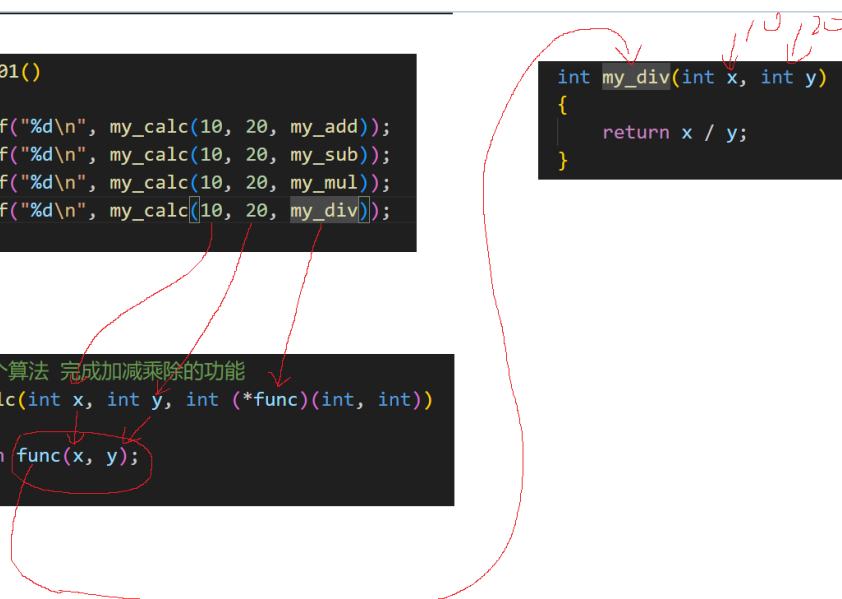
函数指针变量+ - 取* > < != 都无意义, == 有意义。

2、通过函数指针作为函数的形参 实现函数的通用性

```
void test01()
{
    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));
}
```

```
int my_div(int x, int y)
{
    return x / y;
}
```

```
// 设计一个算法 完成加减乘除的功能
int my_calc(int x, int y, int (*func)(int, int))
{
    return func(x, y);
}
```



```
#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;

}


// 设计一个算法 完成加减乘除的功能

int my_calc(int x, int y, int (*func)(int, int))

{

    return func(x, y);

}
```

```

}

void test01()
{
    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));
}

int main(int argc, char const *argv[])
{
    test01();

    return 0;
}

```

案例 1：终端输入 add 100 200 就执行加法

```

#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 my_max(int x, int y)  
{  
    return (x > y ? x : y);  
}  
  
  
// 字符指针数组  
  
char *str_buf[] = {"add", "sub", "mul", "div", "max"};  
  
// 函数指针数组  
  
int (*fun_buf[])(int, int) = {my_add, my_sub, my_mul, my_div, my_max};  
  
int n = sizeof(str_buf) / sizeof(str_buf[0]);
```



```
int main(int argc, char const *argv[])

{

    while (1)

    {

        char cmd[32] = "";

        int data1 = 0, data2 = 0;

        printf("请输入 add 100 200:");

        scanf("%s %d %d", cmd, &data1, &data2);

        int i = 0;

        for (i = 0; i < n; i++)

        {

            if (strcmp(str_buf[i], cmd) == 0)

            {

                printf("%d\n", fun_buf[i](data1, data2));

            }

        }

    }

    return 0;

}
```

```
● edu@edu:~/work/c/day05$ gcc 06_code.c
○ edu@edu:~/work/c/day05$ ./a.out
请输入add 100 200:max 10 20
20
请输入add 100 200:[]
```