知识点1【内存的概述】（了解）

知识点2【指针和指针变量的关系】（了解）

知识点3【指针变量的定义】（了解）

1、定义步骤（定义的时候）

案例1：

2、指针变量的详解

1、在32位平台任何类型的指针变量 都是4字节

2、在64位平台任何类型的指针变量 都是8字节

3、指针变量和普通变量建立关系

知识点4【指针变量的初始化】（了解）

1、指针变量 如果不初始化 立即操作 会出现段错误。

2、指针变量 如果没有指向合法的空间 建议初始化为NULL

3、将指针变量 初始化为 合法的地址（变量的地址、动态申请的地址、函数入口地地址）

知识点5【指针变量的类型】（了解）

1、指针变量自身的类型:

2、指针变量指向的类型(重要)

知识点6【指针变量的指向类型 决定了取值宽度】（重要）

1、知识点的引入

知识点7【指针变量的指向类型 决定了+1跨度】（重要）

综合案例分析：

案例1：取出0x0102的值

案例2：取出0x02的值

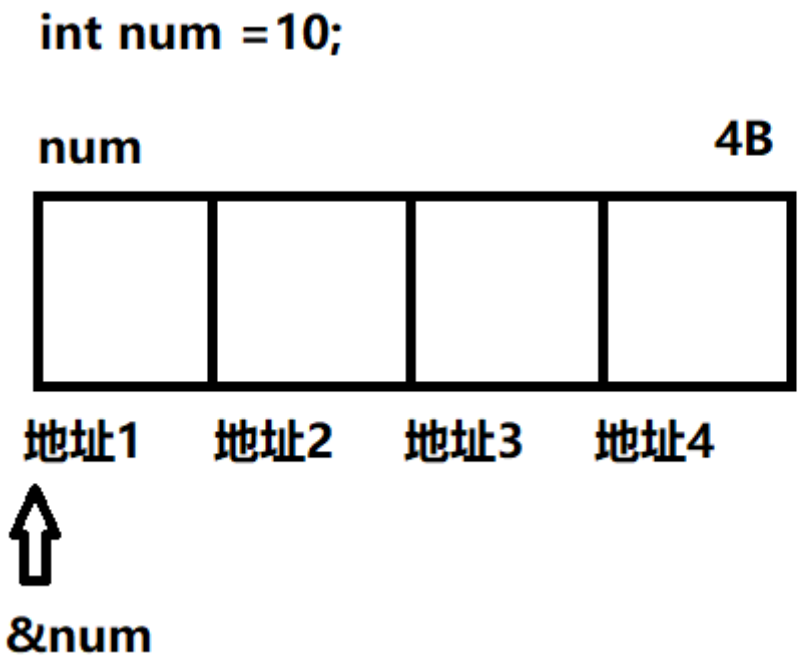
知识点1【内存的概述】（了解）

在32位平台，每一个进程 拥有4G的空间。

系统为内存的每一个字节 分配一个32位的地址编号

地址编号 == 指针

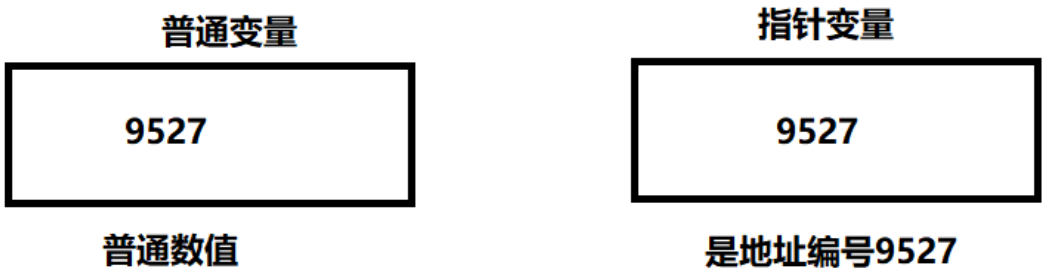
内存的最小分配单位为字节　最小存储单位二进制位



知识点2【指针和指针变量的关系】（了解）

指针 就是内存的地址编号。

指针变量：本质是变量 只是该变量 保存的是 内存的地址编号。（不是普通的数值）



知识点3【指针变量的定义】（了解）

1、定义步骤（定义的时候）

*修饰指针变量p。

保存谁的地址 你先定义谁。

从上往下整体替换。

案例1：

```
定义一个指针变量p 保存 int num的地址；  int *p;
定义一个指针变量p 保存数组int arr[5]首地址；  int (*p)[5]
定义一个指针变量p 保存函数的入口地址 int fun(int,int); int (*p)(int,int);
定义一个指针变量p 保存结构体变量的地址 struct stu lucy; struct stu *p;
定义一个指针变量p 保存指针变量int *p的地址 int **p
```

2、指针变量的详解

1、在32位平台任何类型的指针变量 都是4字节

```
#include <stdio.h>
void test01()
{
    printf("%lu\n", sizeof(char*));
    printf("%lu\n", sizeof(short*));
    printf("%lu\n", sizeof(int*));
    printf("%lu\n", sizeof(long*));
    printf("%lu\n", sizeof(float*));
    printf("%lu\n", sizeof(double*));
    printf("%lu\n", sizeof(int*********));
}
```

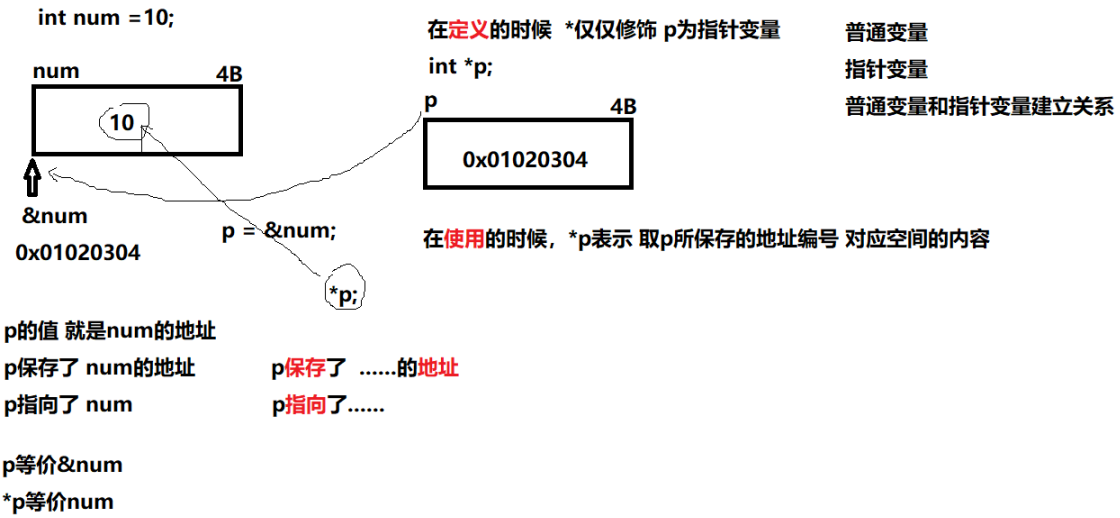
```
Microsoft Visual Studio 调试控制
4
4
4
4
4
4
4
```

2、在64位平台任何类型的指针变量 都是8字节

```c
#include <stdio.h>
void test01()
{
    printf("%lu\n", sizeof(char *));
    printf("%lu\n", sizeof(short *));
    printf("%lu\n", sizeof(int *));
    printf("%lu\n", sizeof(long *));
    printf("%lu\n", sizeof(float *));
    printf("%lu\n", sizeof(double *));
    printf("%lu\n", sizeof(int *********));
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
8
8
8
8
8
8
8
edu@edu:~/work/c/day05$
```

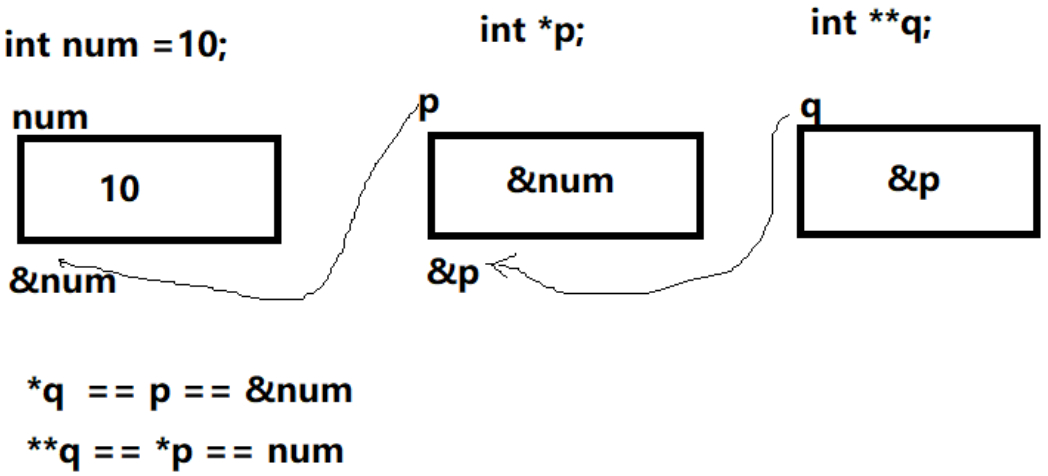
3、指针变量和普通变量建立关系




```c
void test02()
{
    int num = 10;
    int *p;

    //建立关系
    p = &num;

    //%p 打印地址编号
    printf("p = %p\n", p);
    printf("&num = %p\n", &num);

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

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
p = 0x7ffdc64eacec
&num = 0x7ffdc64eacec
num = 10
*p = 10
edu@edu:~/work/c/day05$
```

注意：



*q == p == &num

**q == *p == num

知识点4【指针变量的初始化】（了解）

指针变量 在操作之前 必须指向合法的地址空间。

1、指针变量 如果不初始化 立即操作 会出现段错误。

```
void test03()
{
    //p是局部指针变量 不初始化 指向不确定的内存地址
    int *p;

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

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
段错误 (核心已转储)
edu@edu:~/work/c/day05$
```

2、指针变量 如果没有指向合法的空间 建议初始化为NULL

```
1  int *p = NULL;//NULL是赋值给p int *p; p = NULL;
```

不要操作 指向NULL的指针变量

```
void test03()
{
    //NULL的本质 地址编号0
    int *p = NULL;

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

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
段错误 (核心已转储)
edu@edu:~/work/c/day05$
```

3、将指针变量 初始化为 合法的地址（变量的地址、动态申请的地址、函数入口地地址）

```
1  int num = 10;
2  int *p = &num;//int *p; p = #
```

```
1  int data=10, *p=&data;
```

知识点5【指针变量的类型】（了解）

1、指针变量自身的类型:

将指针变量名拖黑，剩下的类型就是指针变量自身的类型

```
int *p; p自身的类型为int *
```

指针变量自身的类型 一般用于 赋值语句的判断

```
void test04()
{
    int num = 10;
    int *p = &num;

    //在使用中
    //num 为int    &num 为int *    ---->对变量名 取地址 整体类型加一个*
    //p 为int *    *p 为int        ---->对指针变量 取*  整体类型减一个*

     //在使用中  &和*相遇  从右往左 依次抵消
     *&p == p
}
```

案例1：int num=10, *p=&num, **q=&p;以下结果正确的是___ABC___

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

A:*p = 100 B: *q = &num C:p=&num D:q=&num
```

2、指针变量指向的类型(重要)

将指针变量名和离它最近的一个* 一起拖黑，剩下的类型就是指针变量指向的类型

```
int *p; p指向的类型为int
```

知识点6【指针变量的指向类型 决定了取值宽度】(重要)

1、知识点的引入

```
int num = 0x01020304;
int *p = &num;
```

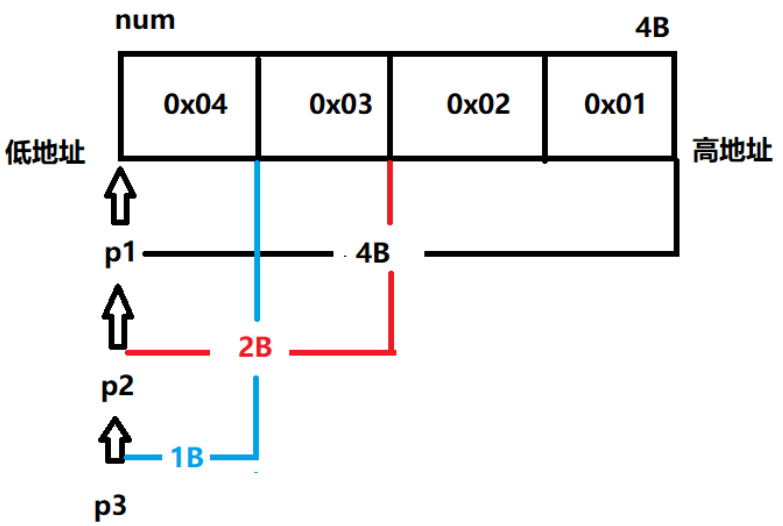
3 为啥 *p == num == 0x01020304?

int num = 0x01020304;

int *p1 = #

short *p2 = #

char *p3 = #




```c
void test05()
{
    int num = 0x01020304;

    int *p1 = &num;
    printf("%#x\n", *p1);

    short *p2 = (short *)&num;
    printf("%#x\n", *p2);

    char *p3 = (char *)&num;
    printf("%#x\n", *p3);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
0x1020304
0x304
0x4
edu@edu:~/work/c/day05$
```

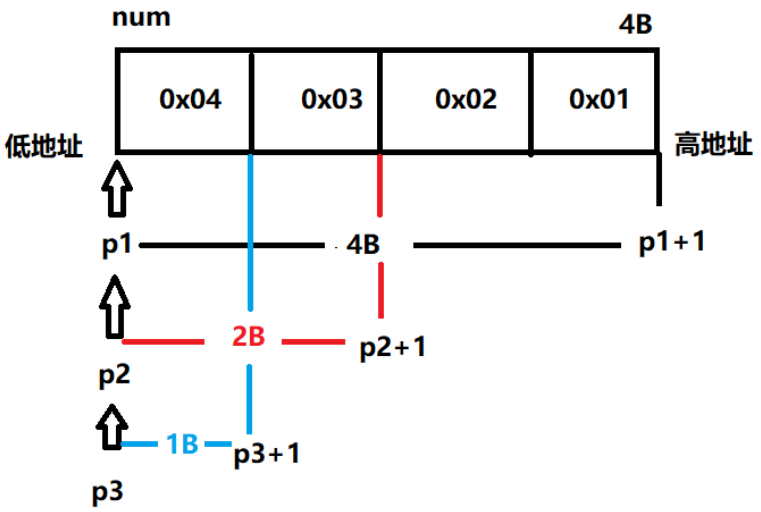
知识点7【指针变量的指向类型 决定了+1跨度】(重要)

int num = 0x01020304;

int *p1 = #

short *p2 = #

char *p3 = #

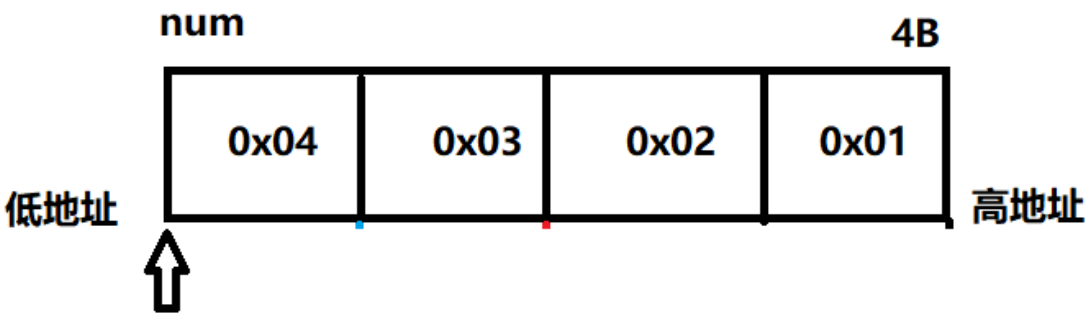



```c
void test06()
{
    int *p1 = NULL;
    printf("p1=%u\n", p1);
    printf("p1=%u\n", p1 + 1);

    short *p2 = NULL;
    printf("p2=%u\n", p2);
    printf("p2=%u\n", p2 + 1);

    char *p3 = NULL;
    printf("p3=%u\n", p3);
    printf("p3=%u\n", p3 + 1);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ ./a.out
p1=0
p1=4
p2=0
p2=2
p3=0
p3=1
edu@edu:~/work/c/day05$
```

综合案例分析：

int num = 0x01020304;



案例1：取出0x0102的值

```c
void test06()
{
    int num = 0x01020304;
    short *p1 = (short *)&num;
    printf("%#x\n", *(p1 + 1));
}
```

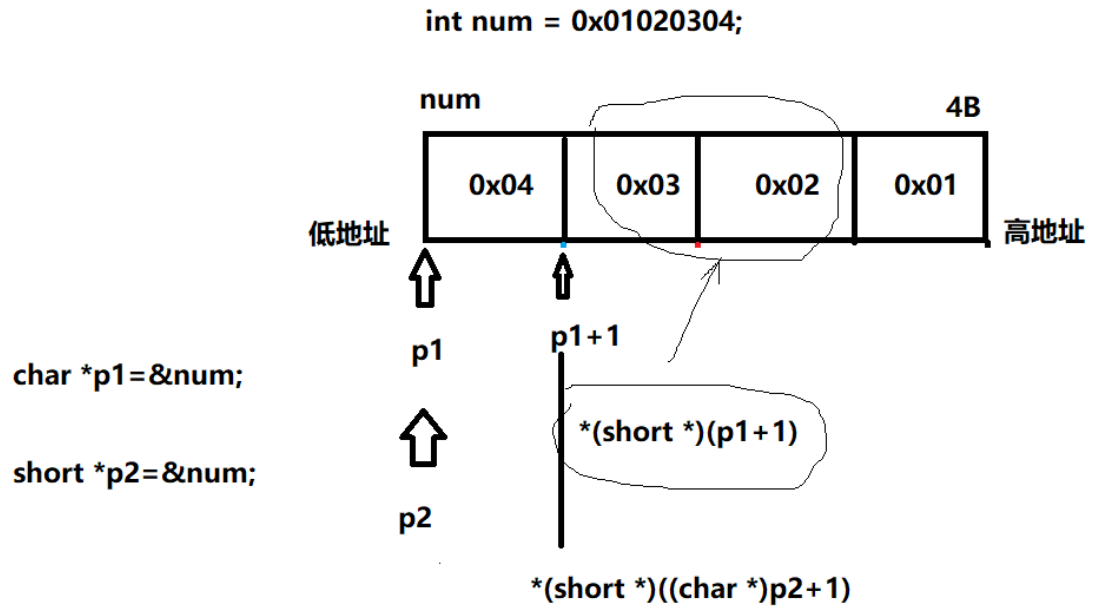
```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
0x102
edu@edu:~/work/c/day05$
```

案例2：取出0x02的值

```c
void test06()
{
    int num = 0x01020304;
    char *p1 = (char *)&num;
    printf("%#x\n", *(p1 + 2));
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
0x2
edu@edu:~/work/c/day05$
```

案例3：取出0x0203的值




```c
void test06()
{
    int num = 0x01020304;
    char *p1 = (char *)&num;
    printf("%#x\n", *(short *)(p1 + 1));
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
0x203
edu@edu:~/work/c/day05$
```

知识点8【*p等价num】(了解)

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

```c
void test07()
{
    int num = 0;
    int *p = &num;

    //p ==&num
    // scanf("%d", &num);
    scanf("%d", p);
    printf("num = %d\n", num);

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

    //num++
    (*p)++; //*p = *p+1
    printf("num = %d\n", num);
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
20
num = 20
num = 100
num = 101
edu@edu:~/work/c/day05$
```

知识点9【指针变量的注意项】(了解)

1、void 不能定义普通变量

```c
void num;//error 不能给num开辟空间
```

2、void * 可以定义指针变量

```c
void *p;//ok p自身类型为void *,在32为平台任意类型的指针 为4B
那么系统知道为p开辟4B空间，所以定义成功
```

p就是万能的一级指针变量， 能保存 任意一级指针的地址编号

```c
void test08()
{
    int num = 10;
    void *p = &num;

    short data = 20;
    p = &data;
}
```

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$
```

万能指针 一般用于 函数的形参 达到算法操作多种数据类型的目的。

记住：不要直接对void *p的指针变量 取*

```c
int num = 10;
void *p = &num;
*p;//err p指向的类型为void 无法确定p的取值宽度 所以不能*p
```

对p取*之前 对p先进行指针类型强转。

```c
void test08()
{
    int num = 10;
    void *p = &num;

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

```
edu@edu: ~/work/c/day05
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
10
edu@edu:~/work/c/day05$
```

3、指针变量 未初始化 不要取*

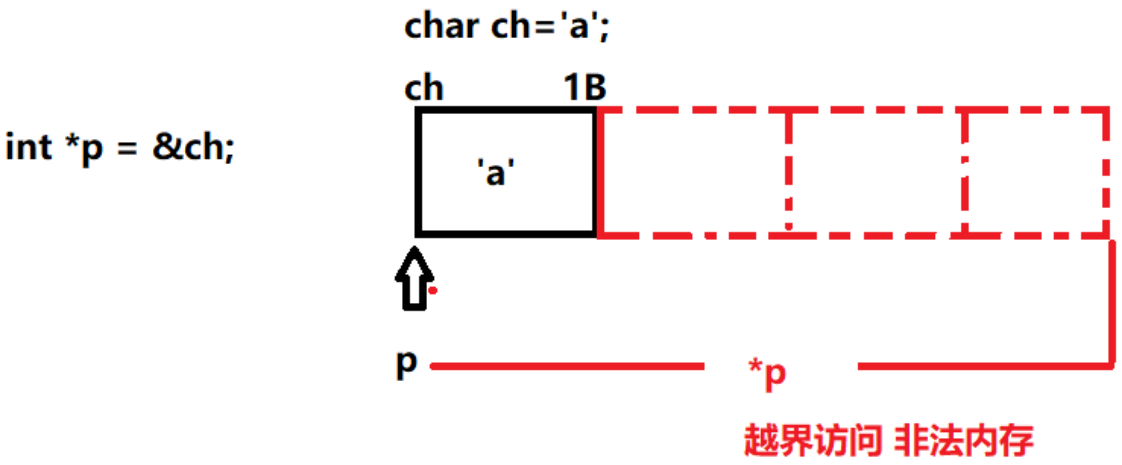
```c
int *p;
*p;//err 段错误
```

4、指针变量 初始化NULL 不要取*

```c
int *p = NULL;
*p;//err 段错误
```

5、指针变量 不要越界访问

```c
char ch = 'a';
int *p = &ch;
*p;//error 越界访问非法内存
```




```c
int num = 10;
int *p = &num;
p++;
*p;//越界访问
```

知识点10【数组元素的指针变量】(了解)

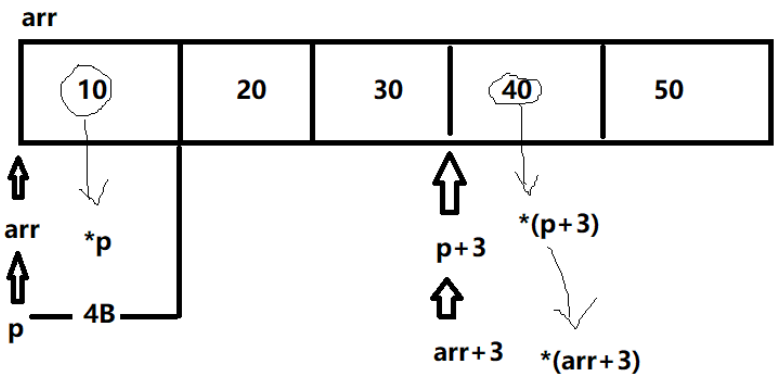
1、概述

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

```
2
3  //需求定义一个指针变量 保存 数组元素的地址
4  int *p;
5  p = &arr[0];
6  p = arr;//arr作为地址 第0个元素的地址 arr==&arr[0]
7  p = &arr[3];
```

2、数组元素的指针变量 和 数组名（作为地址） 等价

int arr[5]={10, 20,30,40,50};　　int *p = arr;　　arr作为地址 代表第0个元素的地址 int *




```
1  void test09()
2  {
3      int arr[5] = {10, 20, 30, 40, 50};
4      int n = sizeof(arr) / sizeof(arr[0]);
5
6      int i = 0;
7      for (i = 0; i < n; i++)
8      {
9          printf("%d ", arr[i]);
10     }
11     printf("\n");
12
13     int *p = arr;
14     for (i = 0; i < n; i++)
15     {
16         printf("%d ", *(p + i));
17     }
18     printf("\n");
19
20     for (i = 0; i < n; i++)
21     {
```

```
        printf("%d ", *(arr + i));
    }
    printf("\n");
}
```

```
edu@edu:~/work/c/day05$ sudo gcc 00_code.c
edu@edu:~/work/c/day05$ ./a.out
10 20 30 40 50
10 20 30 40 50
10 20 30 40 50
edu@edu:~/work/c/day05$
```

3、在使用中 【】就是 *()的缩写

```
void test10()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("arr[1] = %d\n", arr[1]);
    printf("*(arr+1) = %d\n", *(arr + 1));
    printf("--------------------\n");
    printf("*(arr+1) = %d\n", *(1 + arr));
     printf("1[arr] = %d\n", 1 [arr]);

    //[]是*（）的缩写  []左边的值 放在+的左边  []里面的值 放在+右边  整体取*
}
```

为啥arr ==&arr[0]?

```
&arr[0] == &*(arr+0) == arr+0 == arr
```

案例1：p[-1]的值__30__

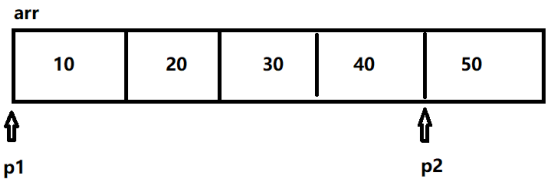
```
int arr[5] = {10, 20, 30, 40, 50};
int *p = arr+3;
```

案例2：p[1]的值__50__

```
int arr[5] = {10, 20, 30, 40, 50};
int *p = arr+3;
```

知识点11【指向同一数组元素的两个指针变量间的关系】(了解)

int arr[5]={10, 20,30,40,50};　　int *p1 = arr; int *p2 = arr+4;



1、两指针变量相减 等于它们间的元素个数

2、两指针变量赋值=　p2=p1 它们指向同一处

3、两指针变量判断相等 ==　p2==p1 它们是否指向同一处

4、两指针变量判断大小 > < >= <= !=

p1> p2　p1!=p2 判断它们的位置关系

5、两指针变量不能相加（重要） p1+p2无意义