综合案例：

知识点1【指针变量的初始化】

总结：指针变量的初始化

知识点2【&取地址符　和 *指针解引用符 区别】（使用中...）

知识点3【指针的注意事项】（重要）

1、void 不能定义变量

2、void *可以定义变量

3、不要对 没有初始化的 指针变量 取*

4、不要对 初始化为NULL的指针变量 取*

5、不要给 指针变量 赋普通的数值。

6、指针变量不要操作越界的空间。

知识点4【数组元素的指针】

案例：通过数组元素的指针变量　遍历　数组的元素

案例提高：

案例：通过数组元素的指针变量给数组的元素 获取键盘输入

知识点5【数组的[]和*()的关系】（重要）

总结：

知识点6【arr 和 &arr的区别】（了解）

知识点7【指向同一数组的两个元素的指针变量 间关系】

综合案例：

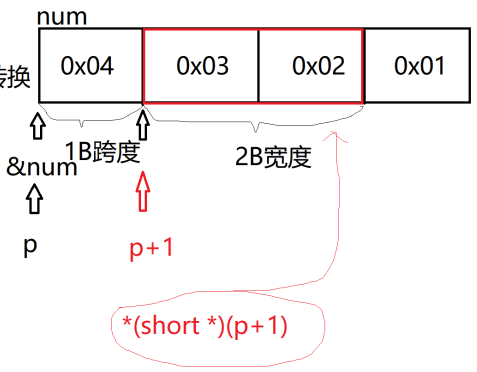
int num = 0x01020304;

1、如果跨度和宽度不等时 需要用到类型转换

2、选择min(跨度，宽度) 定义指针变量

char *p;

p = #




```c
#include<stdio.h>
void test01()
{
 int num = 0x01020304;
 char *p;
 short *p2;

 p = &num;
 p2=&num;

 printf("%#x\n", *(short *)(p+1));//0x203
 printf("%#x\n", *(short *)((char *)p+1));//0x203

 return;
}
int main(int argc,char *argv[])
{
 test01();
 return 0;
}
```

知识点1【指针变量的初始化】

```c
void test02()
{
 int num = 10;
 int data = 200;

 //如果 局部 指针变量 不初始化 保存的是随机的地址编号（千万别取值）
 int *p;

 //不想让指针变量指向任何地方 应该初始化为NULL(千万别取值)
```

```
//#define NULL ((void *)0)
int *p1 = NULL;


//将指针变量初始化为合法的地址(可以取值)
//千万别看成 *p2 = & num;
//*修饰p2为指针变量，p2=& num;
int *p2 = &num;//第一步：int *p2; 第二步：p2=& num;
printf("%d\n", *p2);//num==10


//指针变量 p2本质 是一个变量 可以更改指向
p2 = &data;
printf("%d\n", *p2);//data==200

}
```

运行结果：

```
10
200
Press any key to continue
```

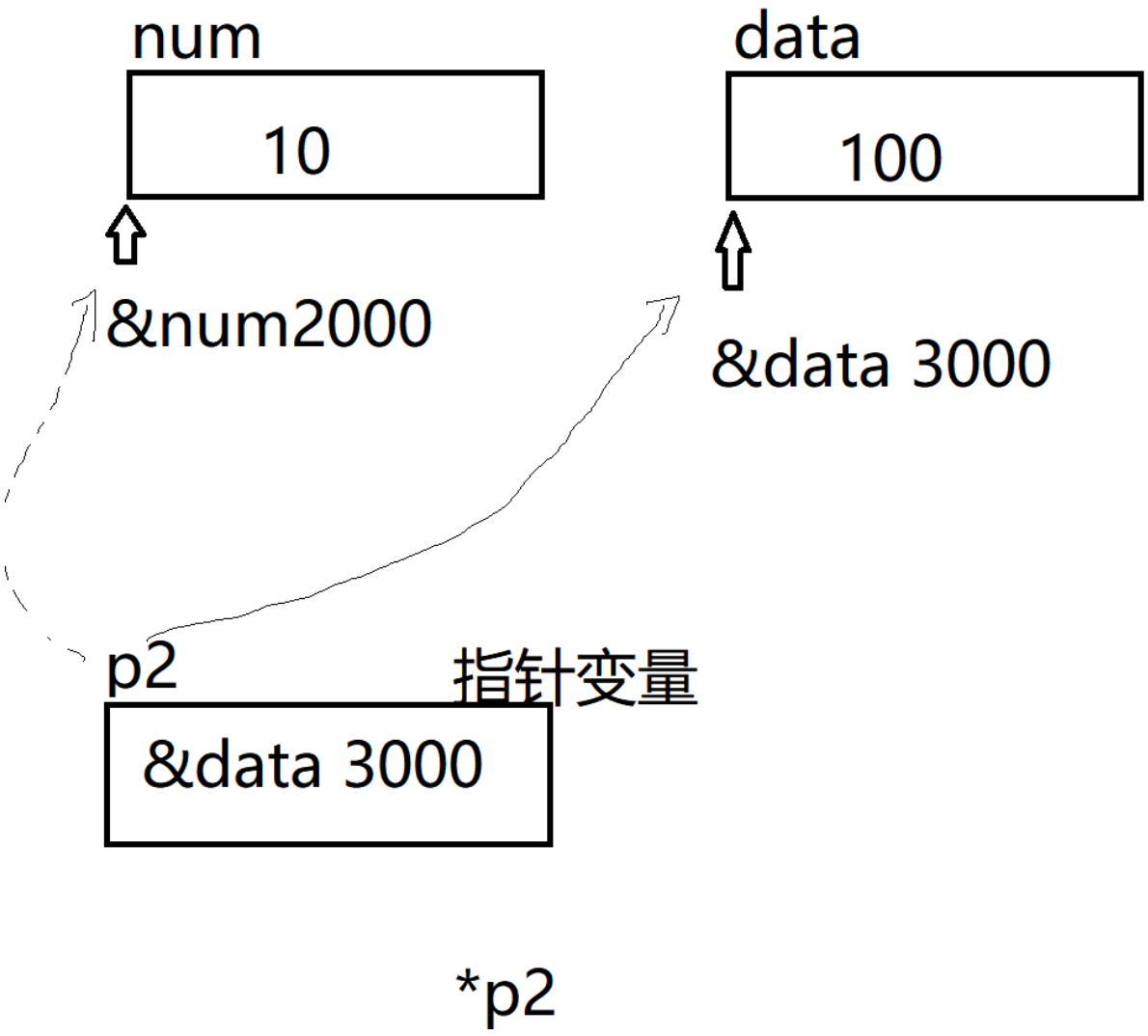
总结：指针变量的初始化

1、指针变量初始化为NULL

```
int *p = NULL;//不要对p进行*p操作 容易出段错误
```

2、指针变量初始化为 合法的空间

```
int num = 10;
int *p = &num;//第一步定义指针变量int *p; 第二步给指针变量赋值：p=&num
```



知识点2【&取地址符　和 *指针解引用符 区别】（使用中...）

```
void test03()
{
 int num = 10;
 int *p;

 //num的类型是 int 类型
 //&num的类型是 int * 类型
 //如果对一个变量取地址 整个表达式的类型 就是变量的类型+*.

  p=&num;

  //p的类型是 int *类型
  //*p的类型是 int 类型
```

```
  //如果对指针变量取* 整个表达式的类型是 指针变量的类型-*.

  //高级总结：如果&和*同时存在 可以相互抵消(从右-->左)。（重要）

  //论证：*p ==num
  //*p=*&num==num;

  //&*&*&num == &num
  printf("p=%p\n",p);
  printf("&*&**&p=%p\n",&*&**&p);
}
int main(int argc,char *argv[])
{
  test03();
  return 0;
}
```

运行结果：

```
p=0019FED8
&*&**&p=0019FED8
Press any key to continue
```

知识点3【指针的注意事项】（重要）

1、void 不能定义变量

```
void num;//错误的 系统不知道 num的大小
```

2、void *可以定义变量

```
void *p;//p的类型为void *, 而void *指针类型,32为平台4字节，系统知道给p开辟4字节
//p叫万能指针 p可以保存 任意一级指针
char ch;
p = &ch;//char *
int num;
p = &num;//int *
float f;
p = &f;//flaot *
```

对于p 不能直接使用*p操作。 必须实现对p进行强制类型转换

```
void test04()
{
```

```
3   int num = 10;
4   void *p;
5   p = &num;
6   //printf("*p = %d\n", *p);//err 因为p的指向类型为void 系统确定不了宽度
7   printf("*p = %d\n", *(int *)p);//ok p临时的指向类型为int 系统确定宽度4B
8   }
```

3、不要对 没有初始化的 指针变量 取*

```
1  int *p;
2  printf("*p=%d\n",*p);
3  //因为p没有初始化 内容随机 也就是p指向了一个未知空间 系统不允许用户 取值*p操作
```

4、不要对 初始化为NULL的指针变量 取*

```
1  //NULL 就是(void *)0 地址，也是内存的起始地址 受系统保护
2  int *p=NULL;
3  printf("*p = %d\n", *p);//也不能 *p
```

5、不要给 指针变量 赋普通的数值。

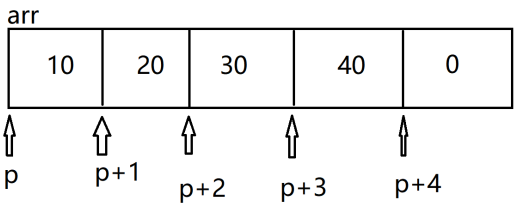
```
1  int *p = 1000;//此时的1000对于p来说 是地址编号 1000
2  //*p表示在地址编号为1000的位置 取值 ，而地址编号1000不是合法的空间 所以不能*p
3  printf("*p = %d\n", *p);//也不能 *p
4
```

6、指针变量不要操作越界的空间。

```
1  char num=10;
2  int *p = &num;
3  //num只占1B空间 而p的指向类型为int 所以*p取值宽度为4B,所以越界3B
4  printf("*p =%d\n",*p);//操作非法空间
```

知识点4【数组元素的指针】

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



需求：定义一个指针变量保存arr数组第0个元素的地址（首元素的地址）

int *p=NULL;

p = &arr[0];

p+i:表示第i个元素的地址

*(p+i):表示第i个元素的值

案例：通过数组元素的指针变量 遍历 数组的元素

```c
void test05()
{
 int arr[5]={10,20,30,40,50};
 int n = sizeof(arr)/sizeof(arr[0]);
 int i=0;

 //p保存了 第0个元素的地址（首元素的地址）
 int *p = &arr[0];
 printf("*p = %d\n", *p);//10

  p++;//p=p+1
  printf("*p = %d\n", *p);//20

  //前提是p保存的是第0个元素的地址
  p=&arr[0];
  for(i=0;i<n;i++)
  {
  //printf("%d ", arr[i]);
  //p+i代表的是第i个元素的地址
  //*(p+i)代表的是第i个元素的值
  printf("%d ", *(p+i) );
  }
  printf("\n");
}
```

运行结果：

```
*p = 10
*p = 20
10 20 30 40 50
Press any key to co
```

案例提高：

```c
void test06()
{
 int arr[5]={10,20,30,40,50};
```

```
 int *p = &arr[1];
 p++;
 p++;
 printf("%d\n", *(p+1));//50
}
```

运行结果：50

案例：通过数组元素的指针变量给数组的元素 获取键盘输入

```
void test07()
{
 int arr[5]={0};
 int n = sizeof(arr)/sizeof(arr[0]);
 int i=0;
 int *p = &arr[0];

 printf("请输入%d个int数据\n",n);
 for(i=0;i<n;i++)
  {
  //scanf("%d", &arr[i]);
  scanf("%d", p+i);//p+i == &arr[i]
  }

  for(i=0;i<n;i++)
  {
  printf("%d ",*(p+i));
  }
  printf("\n");

}
```

运行结果：

```
请输入5个int数据
10 20 30 40 70
10 20 30 40 70
Press any key to continue
```

知识点5【数组的[]和*()的关系】（重要）

```
void test08()
```

```c
{
 //数组名arr 作为类型 代表的是数组的总大小 sizeof(arr)
 //数组名arr 作为地址 代表的是首元素地址（第0个元素的地址）
 int arr[5]={10,20,30,40,50};
 int n = sizeof(arr)/sizeof(arr[0]);
 int *p = NULL;

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

 //在使用中 本质:[] 是*( ) 缩写
 //缩写规则：+左边的值 放在[]左边边 +右边的值 放在[]里面
 printf("arr[1]=%d\n",arr[1]);//20
 printf("arr[1]=%d\n",*(arr+1));//20
 printf("-------------------------\n");
 printf("arr[1]=%d\n",*(1+arr));//20
 printf("arr[1]=%d\n",1[arr] );//20

 //为啥 arr代表的是 第0个元素的地址（&arr[0]）
 //&arr[0] == &*(arr+0) == arr+0 == arr
 //所以：数组名arr代表第0个元素的地址
}
```

总结：

1、【】是*（）的缩写

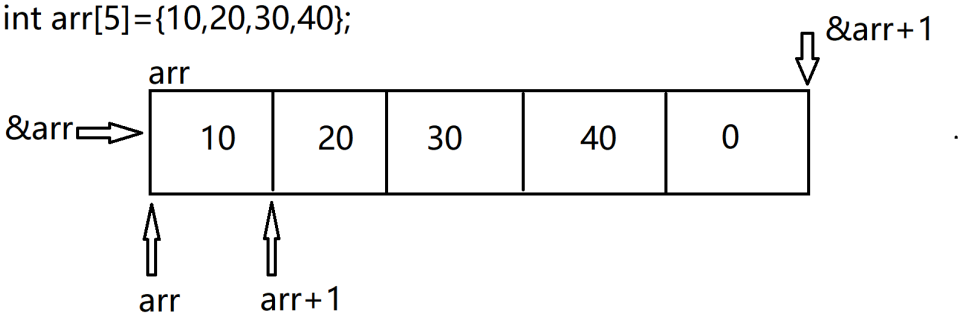
2、数组名arr 代表的是数组 首元素地址（第0个元素的地址）

知识点6【arr 和 &arr的区别】（了解）

arr:数组的首元素地址。

&arr:数组的首地址。

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



arr代表的是首元素地址 +1 跳过一个元素

&arr代表是数组的首地址 +1 跳过整个数组

arr和&arr 在地址编号一样 但是 类型是完全不一样。

```c
void test09()
{
 int arr[5]={10,20,30,40,50};
 printf("arr = %u\n",arr);
 printf("arr+1 = %u\n",arr+1);
 printf("-------------\n");
 printf("&arr = %u\n",&arr);
 printf("&arr+1 = %u\n",&arr+1);
}
```

运行结果：

```
C:\WORK\C\CODE\DAY08\00_test\Debug\00_test.exe
arr = 1703624
arr+1 = 1703628
--------------

&arr = 1703624
&arr+1 = 1703644
Press any key to cont
```

数组名arr 是一个符号常量。不能被赋值。（了解）

```c
void test10()
{
 int arr[5]={10,20,30,40,50};
 //arr=1000;//err arr符号常量 不能被赋值
```

```
5
6   //arr++;//arr=arr+1; err
7   arr+1;//ok
8  }
```

知识点7【指向同一数组的两个元素的指针变量 间关系】

```
1  void test11()
2  {
3   int arr[5]={10,20,30,40,50};
4   int *p1 = arr;
5   int *p2 = arr+3;
6   //1、指向同一数组的两个指针变量相减 返回的是相差元素的个数
7   printf("%d\n",p2-p1);//3
8
9   //2、指向同一数组的两个指针变量 可以比较大小 > < >= <= == !=
10   if(p2>p1)
11   {
12   printf(">\n");
13   }
14   else
15   {
16   printf("<=\n");
17   }
18   //3、指向同一数组的两个指针变量 可以赋值
19   p1=p2;//p1 和 p2指向同一处
20
21   //4、指向同一数组的两个指针变量 尽量不要相加
22   printf("p2=%u\n",p2);
23   //p1+p2;//err 越界很厉害了
24
25   //5、[]里面在不越界的情况下 可以为负数
26   printf("%d\n",p2[-2]);//20
27  }
```

案例：

```
1  void test12()
2  {
3   int arr[5]={10,20,30,40,50};
```

```
 int *p = arr;

 printf("%d\n", *p++);//10
 printf("%d\n", (*p)++);//20
 printf("%d\n", *(p++));//21
}
```