知识点1【防止头文件重复包含】

方式一：#pragma once 编译器决定 晚

#pragma once放在头文件的最前方

main.c

```
#include<stdio.h>
#include"a.h"
#include "b.h"
```

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

a.h

```c
#pragma once //防止头文件重复包含
#include "b.h"
```

b.h

```c
#pragma once//防止头文件重复包含
int num = 10;
```

方式二：c/c++的标准制定 早

#ifndef 宏

#define 宏

头文件具体的内容

#endif

案例：

main.c

```c
#include<stdio.h>
#include"a.h"
#include "b.h"

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

a.h

```c
#ifndef __A_H__
#define __A_H__
#include "b.h"
#endif
```

b.h

```
#ifndef __B_H__
#define __B_H__

int num = 10;

#endif
```

总结：

#pragma once 编译器决定 强调的文件名

#ifndef c/c++标准制定 强调的宏 而不是文件

知识点2【原码 反码 补码】

正数			负数			备注
	概念	10		概念	-10	以1字节为例
原码	数据的二进制形式	0000 1010	原码	数据的二进制形式	1000 1010	
反码	就是原码	0000 1010	反码	原码的符号位不变，其他为取反	1111 0101	
补码	就是原码	0000 1010	补码	反码+1	1111 0110	

注意：无符号数，正数，他们的 原码==反码==补码

负数：反码=原码的符号位不变 其他位取反

补码=反+1.

（重要）：负数在计算机中存储的是补码。

计算机 为啥 要补码?

以1字节分析：

如果没有补码：

6-10== -4

6+（-10） == -4

0000 0110

1000 1010

1001 0000 == -16（错误）

如果有补码：

0000 0110

1111 0110

1111 1100----->1000 0011--->1000 0100==>-4

总结：补码的意义-将减法运算 变加法运算

以1字节分析：

有符号符：1111 1111~1000 0000~0000 0000~0111 1111

-127 ~ -0 ~ +0 ~ +127

计算机为了扩数据的表示范围：故意将-0看成-128

-128~127

无符号数：0000 0000 ~ 1111 1111 == 0~255

总结：补码统一 0 的编码。

+0 == 0000 0000==0000 0000（反码）==0000 0000（补码）

-0 == 1000 0000 ==1111 1111（反码）==0000 0000（补码）

总结：补码意义：将减法运算 变加法运算 同时统一了0的编码。

知识点3【计算机对数据的存储 与 读取】

存储：

```
#include<stdio.h>

void test01()//存储
{
 //负数 以补码 存储
 char data = -10;
```

```
 //正数 以原码 存储
 char data2 = 10;
 //十六进制 以 原码存储
  char data3 = 0xae;//0xae==1010 1110
  //八进制 以 原码存储
  char data4 = 0256;//0256==1010 1110
  //每3位二进制 代表 一位八进制
  //如果数据越界 以原码 存储
  char data5 = 129;//1000 0001

  unsigned char data6 = -10;//0000 1010


  //取 %x %u %o 都是输出内存的原样数据
  //每4位二进制 代表 一位十六进制（记住）
  printf("%x\n", data);//0xf6==1111 0110

  printf("%x\n",data2);//0x0a ==0000 1010

  printf("%x\n",data3);//0xae

  printf("%x\n",data4);//0xae

  printf("%x\n",data5);//0x81
  printf("%x\n",data6);//0xfb

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

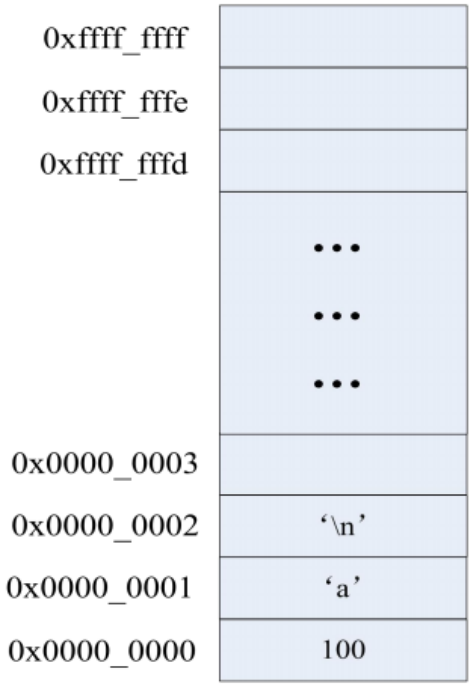
取：

```
void test02()
{
  char data1 = -10;
  char data2 = 10;

  //取:%d %hd %ld有符号取 %u %x %o %lu都是无符号取
  //有符号取:%d %hd %ld
```

```
8   //首先看内存的最高位如果为1 将内存数据符号位不变取反+1到原码
9   //最高位如果为0 将数据原样输出。
10   //无符号取：将内存数据原样输出
11   printf("%d\n",data1);//-10
12
13   //data1&0x000000ff 只取低8位
14   printf("%u\n",data1&0x000000ff);//246==1111 0110
15
16   printf("%d\n",data2);//10
17
18  }
```

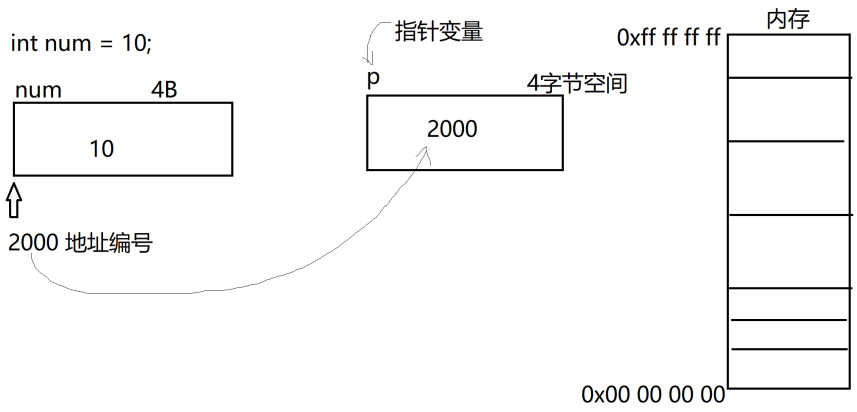
知识点4【内存地址的概述】

系统给内存的每一个字节 分配一个编号 而这个编号 就是内存地址。



内存地址 也叫 指针。 指针 就是 地址 地址 就是 指针。

知识点5【指针变量概念】（重要）



指针变量：本质就是一个变量　只是这个变量 存放 是内存的地址编号（地址/指针）。

在32位平台 任何类型的地址编号 都是4字节。

```c
#include<stdio.h>

void test01()
{
    printf("%d\n",sizeof(char *));
    printf("%d\n",sizeof(short *));
    printf("%d\n",sizeof(int *));
    printf("%d\n",sizeof(long *));
    printf("%d\n",sizeof(float *));
    printf("%d\n",sizeof(double *));
    printf("%d\n",sizeof(double *******));
    return;
}
int main(int argc,char *argv[])
{
    test01();
    return 0;
}
```

```
"C:\work\c\code\day07\03_test\Debug\03_test.
4
4
4
4
4
4
4
Press any
```

知识点6【定义指针变量】（重要）

定义指针变量的步骤：

1、*修饰 指针变量名

2、保存啥类型变量的地址　就用该类型定义一个普通变量。

3、从上 往下 整体 替换

定义指针变量（前提）

1、明确保存 啥类型变量的地址。

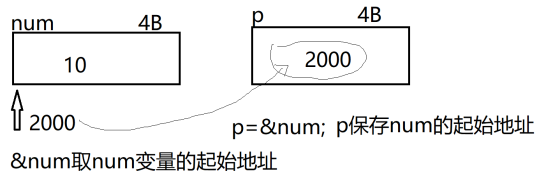
案例：

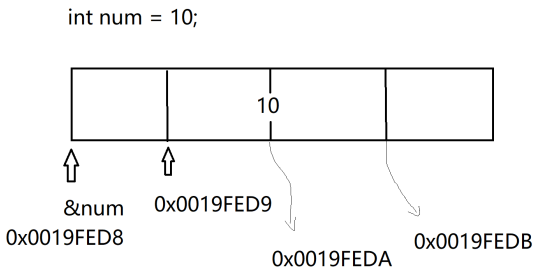
```c
void test02()
{
 //num拥有一个合法的空间
 int num = 10;
```

```
5
6   //需求：请定义一个指针变量 保存num的地址
7   //p就是指针变量 变量名为p 不是*p
8   //在定义的时候:*修饰p 表示p为指针变量
9   int *p;
10
11   //建立p和num的关系：p保存num的地址
12   printf("&num = %p\n",&num);
13   p = &num;//&num 代表的是num变量起始地址（首地址）
14   printf("p = %p\n", p);
15  }
```

运行结果：

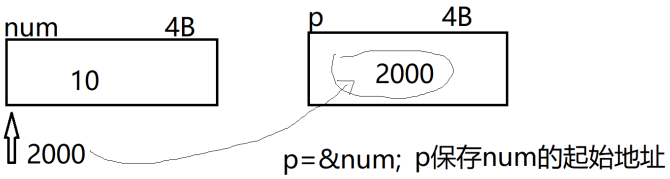
```
"C:\work\c\code\day07\03_test\Debug\03_test.exe"
&num = 0019FED8
p = 0019FED8
Press any key to continue
```





知识点7【指针变量的使用】通过过p 对所保存的地址空间 进行读写操作（重要）

int num = 10; int *p;



p=# p保存num的起始地址

&num取num变量的起始地址

在使用中:

*p:表示取p所保存的地址编号 对应的空间内容。 指针变量p的解引用

*p 等价 num

案例:

```c
void test02()
{
 //num拥有一个合法的空间
 int num = 10;

 //需求: 请定义一个指针变量 保存num的地址
 //p就是指针变量 变量名为p 不是*p
 //在定义的时候:*修饰p 表示p为指针变量
 int *p;


 //建立p和num的关系: p保存num的地址
 printf("&num = %p\n",&num);
 p = &num;//&num 代表的是num变量起始地址（首地址）
 printf("p = %p\n", p);

 //*p 等价 num
 printf("*p = %d\n", *p);//10==num

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

 //scanf("%d", &num);
 scanf("%d", p);//如果此处为 &p表示键盘给p赋值 而不是给num赋值
 printf("num = %d\n",num);
}
```

运行结果：

```
&num = 0019FED8
p = 0019FED8
*p = 10
num = 100
2000
num = 2000
Press any key to continue
```

知识点8【指针变量的类型】（重要）

int *p;

在定义中：

指针变量 自身类型。只将指针变量名拖黑 剩下啥类型 指针变量自身就是啥类型。

p自身的类型是int *。

指针变量 所指向的类型。将指针变量名和离它最近的一个*一起拖黑 剩下啥类型 就指向啥类型。

p指向的类型为int ==== p保存int类型变量的地址。

指向.....类型 ==保存.....类型变量的地址

int num = 10;

int *p;

p = #//p指向num ==p保存了num的地址

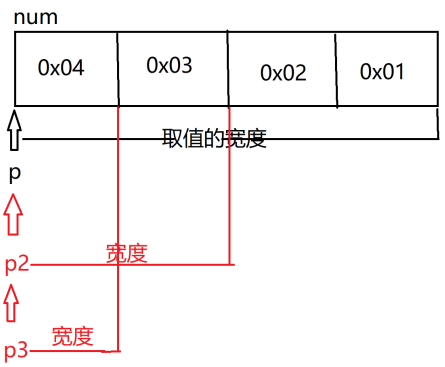
案例：指针变量取值宽度。（重要）

int num = 0x01020304;
int *p;
p=#

char *p3;
p3 = #

short *p2;
p2 = #

宽度：指针变量指向的类型长度决定。




```
1 void test03()
2 {
```

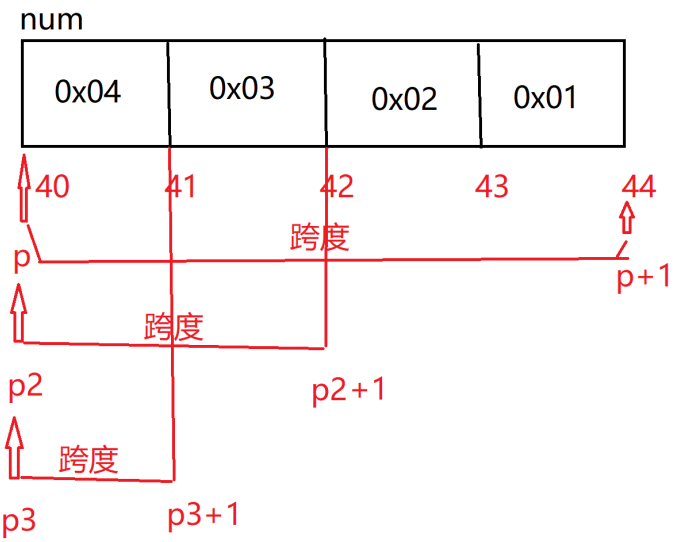
```c
    int num = 0x01020304;
    int *p;
    short *p2;
    char *p3;

    p = &num;
    p2 = &num;
     p3 = &num;

     printf("*p = %#x\n", *p);
     printf("*p2 = %#x\n", *p2);
     printf("*p3 = %#x\n", *p3);
}
```

案例：指针变量的跨度（重要）

int num = 0x01020304;
int *p;
p=#

char *p3;
p3=#

short *p2;
p2 = #



跨度：指针变量指向的类型长度决定

```c
void test04()
{
    int num = 0x01020304;
    int *p;
    short *p2;
    char *p3;

    p = &num;
    p2 = &num;
     p3 = &num;

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

```
13    printf("p+1=%u\n",p+1);
14    printf("-----------------\n");
15    printf("p2=%u\n",p2);
16    printf("p2+1=%u\n",p2+1);
17    printf("-----------------\n");
18    printf("p3=%u\n",p3);
19    printf("p3+1=%u\n",p3+1);
20  }
```

运行结果：

```
p=1703640
p+1=1703644
------------------

p2=1703640
p2+1=1703642
------------------

p3=1703640
p3+1=1703641
Press any key to continue
```

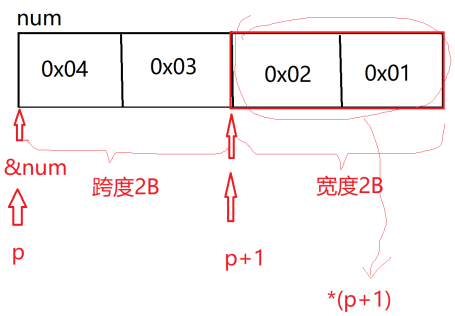
综合案例：

int num = 0x01020304;

需求：自定义指针变量p 取出0x0102

short *p ;

p = #



```c
void test05()
{
    int num = 0x01020304;
    short *p;
    p = &num;

    printf("%#x\n",*(p+1));
}
int main(int argc,char *argv[])
{
    test05();
    return 0;
}
```