

# C语言——面向过程编程

## 单元二、指针与数组

### 第一次课 指针基础概念

#### 一. 内存地址

##### 1. 什么是内存地址

①先了解一下操作系统（例如 win10）的位数问题：



##### 1: 支持的内存不同

32位的操作系统，最多支持4G的内存，实际内存为3.25G；64位系统支持4G 8G 16G 32G 64G 128G 256G内存，理论上可以无限支持，只要你主板上有足够的内存条。

系统	
制造商:	DADI
型号:	大地 Ghost Win7 装机版 V 9.1
分级:	4.2 Windows 体验指数
处理器:	Intel(R) Core(TM)2 Duo CPU T8100 @ 2.10GHz 2.10 GHz
安装内存(RAM):	3.00 GB
系统类型:	32 位操作系统
笔和触控:	没有可用于此显示器的笔或触控输入

##### 小米笔记本 15.6"

设备名称	DESKTOP-P9B87BO
处理器	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
机带 RAM	8.00 GB
设备 ID	781299D8-3D23-41A0-8455-C745119B7D31
产品 ID	00330-80000-00000-AA655
系统类型	64 位操作系统, 基于 x64 的处理器
笔和触控	没有可用于此显示器的笔或触控输入

##### 2: 支持的处理器不同

64位的操作系统支持基于64位的处理器，而32位的系统却不能完全支持64位的处理器。

##### 3: 支持的软件不同

32位的操作系统，支持基于32位的软件，不能运行64位的软件；而64位的系统一般这两种类型的都支持，基本上与各种软件都兼容，特别是adobe公司的软件，现在的新版本的都只支持64位。

##### 4: 处理数据的能力

32和64表示CPU可以处理最大位数，一次性的运算量不一样，理论上64位的会比32位快1倍，内存寻址也不一样。

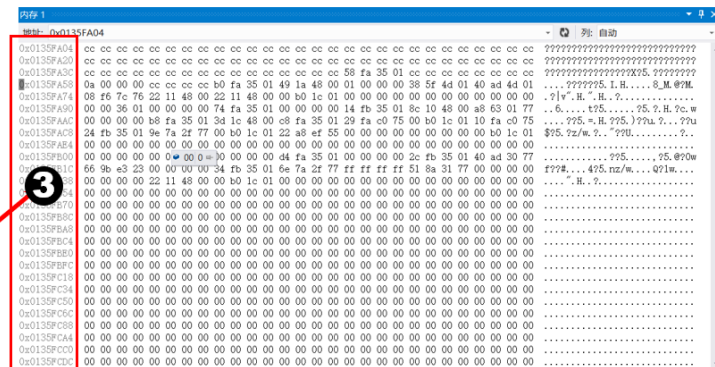
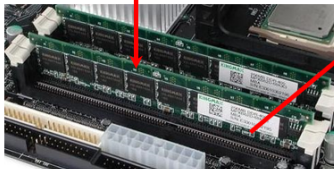
## 5：系统体积大小

64位系统都比32位系统大的多,比如win7 64位比win7 32位系统大700M左右。

### ②理解一下内存地址的基本意义。

我们定义的变量就存在内存硬件中

① `int age = 23;`



② 内存空间的每个字节是有唯一编号的。

在调试器中以十六进制表示内存字节的编号

`printf("%d", age);`

④

代码执行时，会根据age变量找到对应的存储地址，再根据其类型取出里面的数据，再打印出来

## 2. 内存地址运算之：& 取地址

```
// & 取地址运算符
int a = 10;
printf("%d,%p\n",&a,&a);//用十进制打印a的地址， %p以十六进制打印a的内存地址
printf("地址编号的字节长度: %d \n",sizeof(&a) );//根据设置（32位或64位） 可能打印出：4或8
//格式化输入函数—scanf， 在从键盘缓存接收数据时，需要把数据存入内存地址中。
scanf("%d",&a);//需要对变量进行取地址操作，得到真正的内存空间。
printf("你输入的是: %d",a);
```

## 3. 内存地址运算之：\*地址指向内存空间的操作

```
// * 地址空间的操作
//向地址空间赋值的底层写法。
// 向地址空间赋值（存数据）
*(&a)=100;//&a为取地址，* 内存地址 就代表： 地址空间
printf("a= %d\n",a);
printf("*(&a)= %d\n",*(&a));// *(&a)从地址空间取值（去打印）

*(&a)+=55;//对地址空间 进行运算。
printf("a= %d\n",a);
```

## 二. 指针变量与地址操作

### 1. 什么是指针变量

可以存储内存地址的变量为指针变量。定义指针变量的方式：

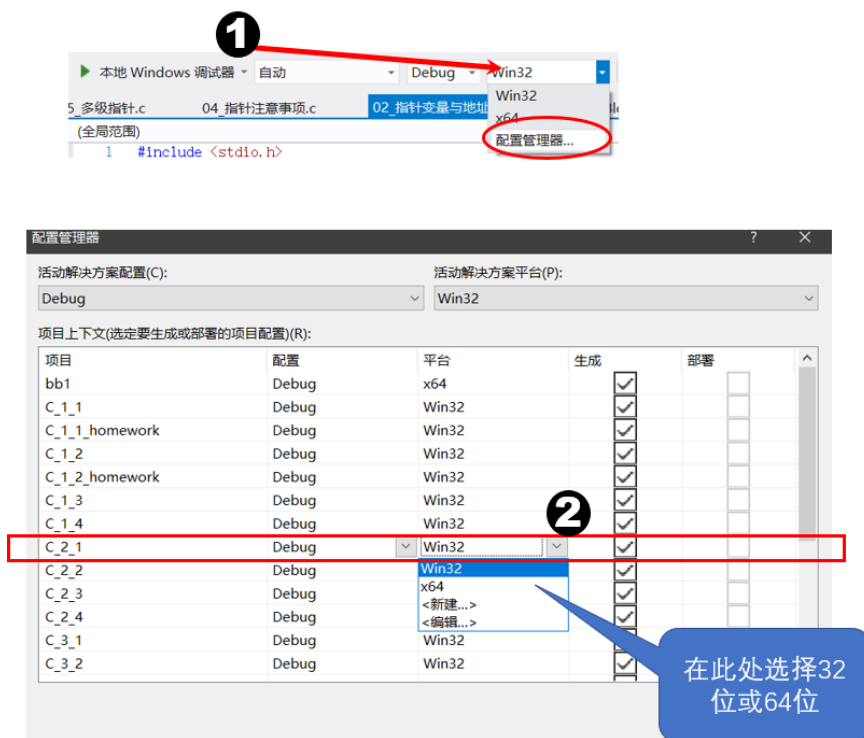
**现有类型 \* 指针变量名；**

```
int a = 10;
int * p=&a; //定义指针变量p, p指向a的地址
printf("%p",p); //打印地址编号
```

### 2. 指针变量的字节数

```
int *p;
double *pd;
long long *pll;
char * pc;
printf("%d\n", sizeof(p));
printf("%d\n", sizeof(pd));
printf("%d\n", sizeof(pll));
printf("%d\n", sizeof(pc));
//无论什么类型的指针变量，所占字节长度是固定的,因为指针变量保留的是内存地址的编号，它只能随着32位系统或64位系统而不同。32位就是用4个字节空间保留地址编号，64位就用8个字节空间保留地址编号。
```

如何调整所开发软件系统位数：



### 3. 指针的移动

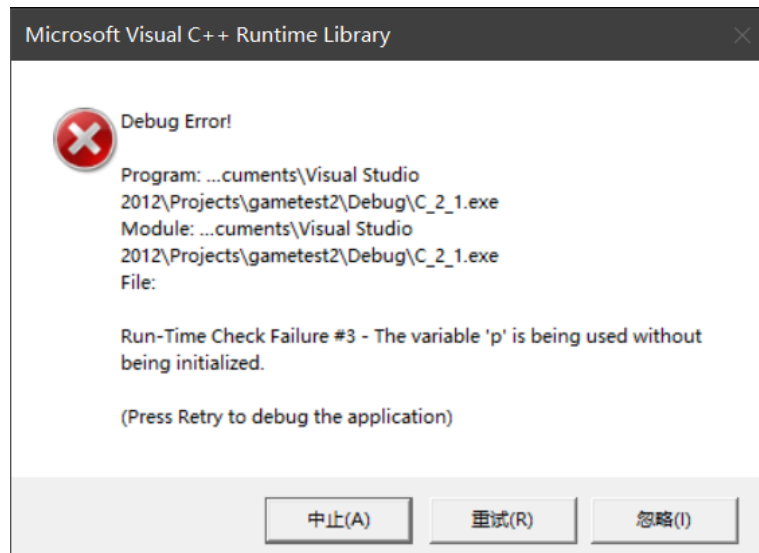
根据指针变量的数据类型不同，指标+或-移动的地址步伐大小也不同

```

int *p;
double *pd;
long long *p11;
char * pc;
printf("现地址: %d 下一个地址: %d\n", p,p+1);
printf("现地址: %d 下一个地址: %d\n", pd,pd+1);
printf("现地址: %d 下一个地址: %d\n", p11 ,p11+1);
printf("现地址: %d 下一个地址: %d\n", pc,pc+1);

```

出于上述代码可能会改动属于它的地址空间。所以操作系统会通报一个终止的提示。



由于我们不会对地址空间进行任何改变，仅仅是查看。连续点击：“忽略...”

```

int *p;
double *pd;
long long *p11;
char * pc;

```

由于没有赋初始值  
默认的地址都是一样的

现地址: -858993460	下一个地址: -858993456	4个字节
现地址: -858993460	下一个地址: -858993452	8个字节
现地址: -858993460	下一个地址: -858993452	8个字节
现地址: -858993460	下一个地址: -858993459	1个字节

请按任意键继续...

但是由于指针变量的类型不同，指针变量+1所计算出的新地址也不同

#### 4. 利用指针的算术运算

```

int a = 10;
int *p = &a;
*p = 100; //利用指针赋值
printf("a= %d\n", *p);
*p += 55; //利用指针自增5
printf("a= %d\n", a);
// *p++; //++ 与 * 一个优先级，且从右向左计算。因此 ++与p结合，会使p指向一个新地址，然后p与*
结合取出新地址里的值。

(*p)++; //让p所指向的内存空间自增1
printf("a= %d\n", a);

```

### 三. 大端与小端

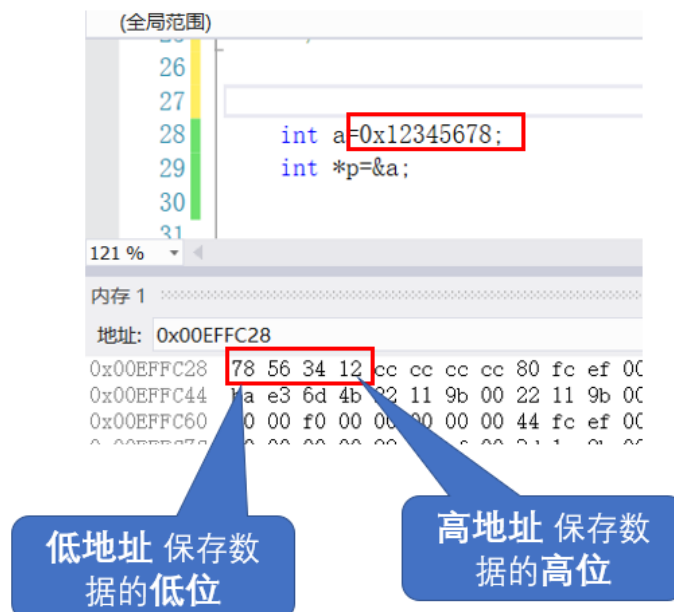
大小端 (Endian) 是指数据存储或者传输时的字节序，具体分为：大端和小端。大端 (Big-Endian) 模式，是指数据的低位（就是权值较小的后面那几位）保存在内存的高地址中，而数据的高位，保存在内存的低地址中；地址由小向大增加，而数据从高位往低位放。而小端 (Little-Endian) 模式，是指数据的低位保存在内存的低地址中，而数据的高位保存在内存的高地址中。

总结起来就是：

小端存储：低位存放在低地址（高位存放在高地址）

大端存储：低位存放在高地址（高位存放在低地址）

## 小端模式的存储特点

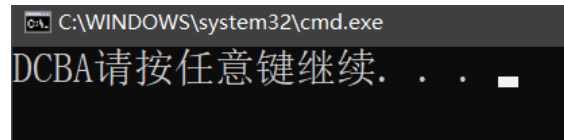


怎样判断大小端

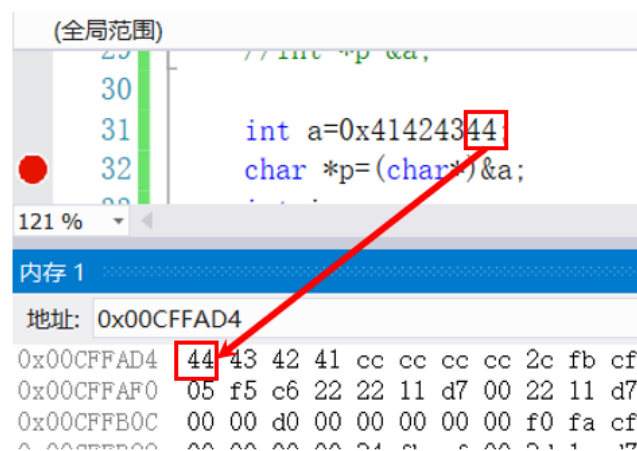
```

int a=0x41424344;//用int空间的四个字节存储4个字符的ASCII码值，0x41是十进制的65即A。依次代表BCD
char *p=(char*)&a;//让字符指针p指向a变量首地址，因为a的类型与p不是一个类型，所以需要强转成char型指针。
int i;
for( i=0; i<4; i++ )
{
    printf("%c",*p);
    p++;
}

```



打印结果并不是：ABCD而是DCBA。



据此可以看出是：小端模式

改造一下代码：

```

int a=0x41424344;
char *p=(char*)&a;
if (*p==0x44)
{
    printf("小端\n");
}
else
{
    printf("大端\n");
}

```

## 为什么会有大小端之分

1. 一开始是由于不同架构的CPU处理多个字节数据的顺序不一样，比如x86的是小端模式，KEIL C51是大端模式。但是后来互联网流行，TCP/IP协议规定为大端模式，为了跨平台通信，还专门出了网络字节序和主机字节序之间的转换接口（ntohs、htons、ntohl、htonl）
2. 大小端模式各有优势：小端模式强制转换类型时不需要调整字节内容，直接截取低字节即可；大端模式由于符号位为第一个字节，很方便判断正负。

## 四. 指针注意事项

1. 移动不越界

```

/*
 注意1：指针不要位移到不属于本程序的内存空间
 也不要利用指针改变不属于本程序的内存空间的数据
*/
int a=10, *p;
p=&a;
p+=5;//p指向了一个新的地址
*p=100; //向新的空间里存入100

```

## 2. 定义指针不省\*

```

/*
 注意2: 声明多个指针变量类型时，*不能省略。
*/
int a,b,c;//创建了3个int型变量
int* p,q,k;//创建了一个指针变量 2个int型变量
int *x,*y,*z;//创建了三个指针变量



```

## 3. 初始空值可用NULL

```

/*
 注意3：指针变量的初始值如果没有明确指向目标用NULL赋值
*/
int *p=NULL;//NULL是空的意思，按F12可以查看其定义：#define NULL 0
//NULL就是0。代表p不指向任何的地址。

```

局部变量	
名称	值
 p	0x00000000 {???
	<无法读取内存>

# 五. 多级指针

多级指针又称为：指向指针的指针（级数过多，引起不适，通常二级居多）。

```

int a=10;//整型变量
int *p=&a;//指向整型的指针变量
int **pp=&p;//指向整型指针的指针变量
int ***ppp=&pp;//指向整型指针的指针的指针变量

printf("%d\n",a);
printf("%d\n",*p);
printf("%d\n",**pp);
printf("%d\n",***ppp);

```

局部变量	
名称	值
pp	0x00adf9ac {0x00adf9b8 {10}}
	0x00adf9b8 {10}
	10
a	10
ppp	0x00adf9a0 {0x00adf9ac {0x00adf9b8 {10}}}
	0x00adf9ac {0x00adf9b8 {10}}
	0x00adf9b8 {10}
	10
p	0x00adf9b8 {10}
	10

二级指针

三级指针

一级指针