

# 第六章 指针变量的基础知识

本章学习目标：

- ✓ 掌握指针类型定义
- ✓ 掌握一重指针的声明和使用规则
- ✓ 掌握使用内存示意图的方式分析程序
- ✓ 掌握多重指针的声明和使用
- ✓ 理解 const 指针
- ✓ 了解空指针以及通用指针
- ✓ 了解指针变量的运算。

## 6.1 实践题

### 一、认识指针变量和指针数据类型

#### 实验目的

1. 理解指针数据类型。
2. 掌握指针变量与普通变量的关系。

#### 实验步骤

步骤 1：定义 int 型的变量 a，float 型变量 b，char 型变量 c，观察普通变量所占内存情况。

1. 原始代码

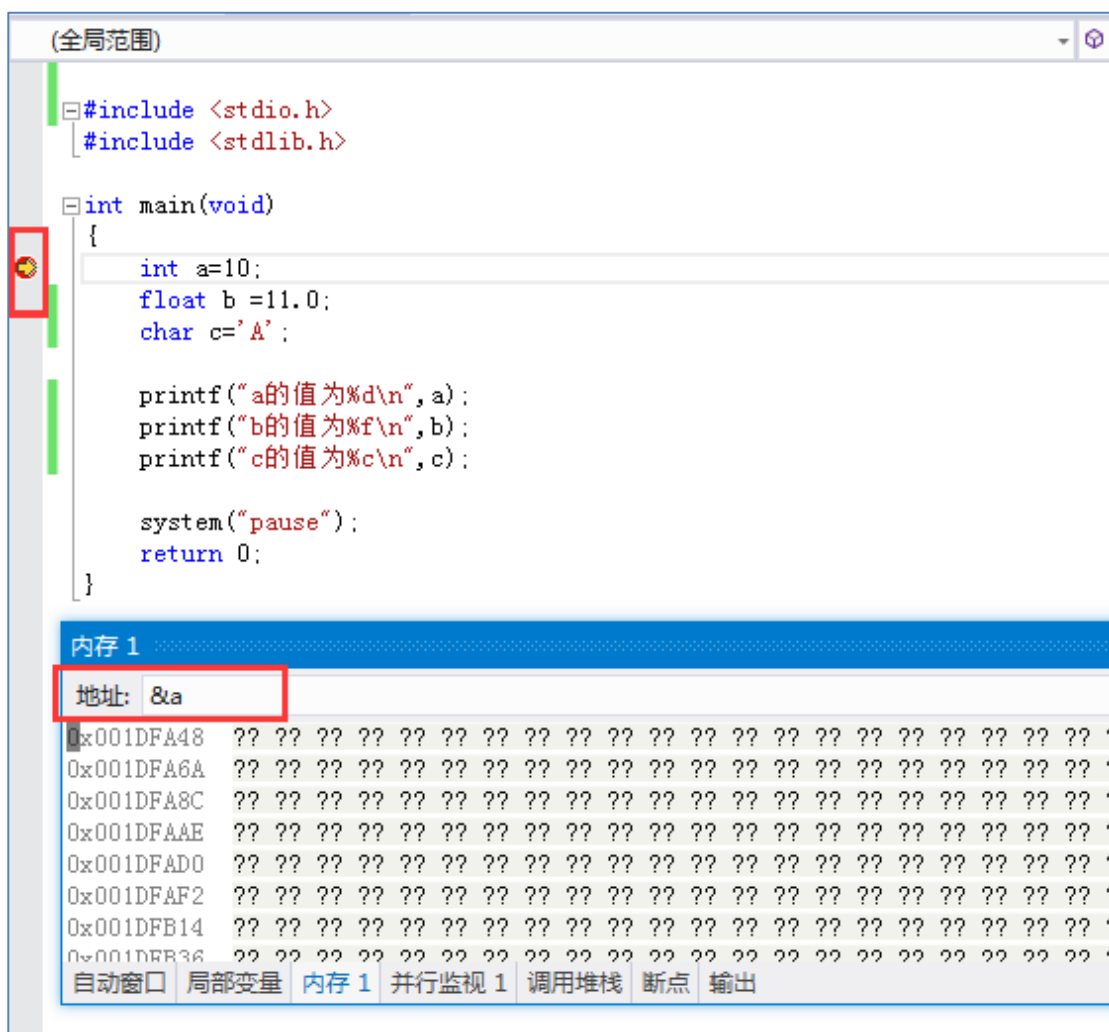
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a=10;
    float b =11.0;
    char c='A' ;

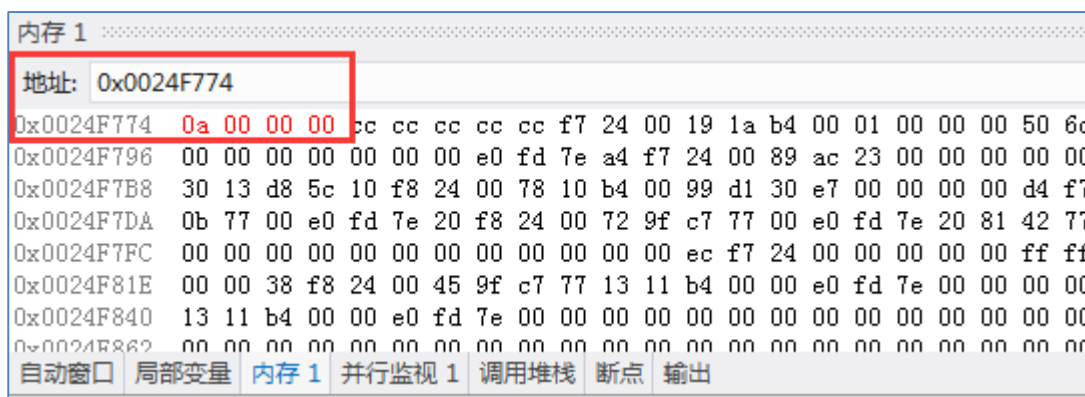
    printf("a的值为%d\n",a);
    printf("b的值为%f\n",b);
    printf("c的值为%c\n",c);

    system("pause");
    return 0;
}
```

2. 加断点，在内存窗口查看各变量的存储



单击单步执行，查看到整型变量 a 的地址为 0x0024F774，从此处开始的 4 个字节存放变量的值 10，且采用“低位在低字节”的形式存储。



同理查看变量 b 和 c 的内存存储情况。

(全局范围)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a=10;
    float b =11.0;
    char c= 'A' ;

    printf("a的值为%d, a的地址为%p\n", a, &a);
    printf("b的值为%f, b的地址为%p\n", b, &b);
    printf("c的值为%c, c的地址为%p\n", c, &c);

    system("pause");
    return 0;
}
```

内存 1

地址: 0x0024F768

|            |             |  |
|------------|-------------|--|
| 0x0024F768 | 00 00 30 41 | cc cc cc cc cc cc cc cc 0a 00 00 00 cc cc cc cc c    |
| 0x0024F78A | 5f 00 08 74 | 5f 00 75 49 a0 e7 00 00 00 00 00 00 00 00 00 00 e0 f |
| 0x0024F7AC | 00 00 25 00 | 00 00 00 90 f7 24 00 30 13 d8 5c 10 f8 24 00 7       |
| 0x0024F7CE | 24 00 0d 1c | b4 00 e0 f7 24 00 8a 33 0b 77 00 e0 fd 7e 20 f8 2    |
| 0x0024F7F0 | 00 00 00 00 | 00 00 00 00 e0 fd 7e 00 00 00 00 00 00 00 00 0       |
| 0x0024F812 | ff ff f5 71 | cb 77 50 bc a0 00 00 00 00 00 38 f8 24 00 45 9f c    |
| 0x0024F834 | 00 00 00 00 | 00 00 00 00 00 00 00 00 13 11 b4 00 00 e0 fd 7e 0    |
| 0x0024F856 | 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0       |

自动窗口 局部变量 内存 1 并行监视 1 调用堆栈 断点 输出

The screenshot shows a debugger window with the following components:

- Code Editor:** Displays C code. The line `char c= 'A' ;` is highlighted with a red box. A red dot is placed on the line `printf("a的值为%d, a的地址为%p\n", a, &a);`.
- Memory Window:** Titled "内存 1", it shows a memory dump starting at address `0x0024F75F`. The first row of data is `41 cc cc cc cc cc cc cc cc 00 00 30 41 cc cc cc cc cc cc cc`, where `41` is highlighted in red.
- Debugger Controls:** At the bottom, there are tabs for "自动窗口", "局部变量", "内存 1", "并行监视 1", "调用堆栈", "断点", and "输出".

步骤 2：内存查看变量的地址并输出，验证每个变量都有地址（多查看几次内存并对比执行结果）。

The screenshot shows a code editor window with the following code:

```
(全局范围)

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a=10;
    float b =11.0;
    char c='A' ;

    printf("a的值为%d, a的地址为%p\n", a, &a);
    printf("b的值为%f, b的地址为%p\n", b, &b);
    printf("c的值为%c, c的地址为%p\n", c, &c);

    system("pause");
    return 0;
}
```

步骤 3：指针变量 p，q 的理解。指针 p，q 的数据类型为指针，理解 p 与&a，q 与&b

等价的原理，\*p，\*q 的意义。

(全局范围)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a=10;
    float b =11.0;
    char c='A';

    int * p = &a;
    float * q = &b;

    printf("a的值为%d, a的地址为%p\n", *p, p);
    printf("b的值为%f, b的地址为%p\n", *q, q);
    printf("c的值为%c, c的地址为%p\n", c, &c);

    system("pause");
}
```

局部变量

| 名称 | 值               |
|----|-----------------|
| c  | 0x41 'A'        |
| b  | 11.0000000      |
| q  | 0xffffffff {??} |
| a  | 0x0000000a      |
| p  | 0xffffffff {??} |

自动窗口 局部变量 内存 1 并行监视 1 调用堆栈 断点 输出

内存查看&a，p，&p 的值，并对比分析其关系。

内存 1

地址: &a

|            |   |
|------------|---|
| 0x003BFBFC | 0a 00 00 00 cc cc cc cc 54 fc 3b 00 19 1a 38 01 01 00 00 00 60 67 |
| 0x003BFC1E | 00 00 00 00 00 00 00 e0 fd 7e 2c fc 3b 00 89 ac 23 00 00 00 00 00 |
| 0x003BFC40 | 60 2a c1 c6 98 fc 3b 00 78 10 38 01 83 a7 07 3e 00 00 00 00 5c fc |
| 0x003BFC62 | f5 76 00 e0 fd 7e a8 fc 3b 00 72 9f bf 77 00 e0 fd 7e 58 63 c6 77 |
| 0x003BFC84 | 00 00 00 00 00 00 00 00 00 00 00 00 00 74 fc 3b 00 00 00 00 ff fd |
| 0x003BFCA6 | 00 00 c0 fc 3b 00 45 9f bf 77 13 11 38 01 00 e0 fd 7e 00 00 00 00 |
| 0x003BFCC8 | 13 11 38 01 00 e0 fd 7e 00 00 00 00 00 00 00 00 00 00 00 00 00    |
| 0x003BFCE4 | 00       |

自动窗口 局部变量 内存 1 并行监视 1 调用堆栈 断点 输出

&a，即变量 a 的地址为 0x003BFBFC。

内存 1

地址: 0x003BFBFC

|            |   |
|------------|---|
| 0x003BFBFC | 0a 00 00 00 cc cc cc cc 54 fc 3b 00 19 1a 38 01 01 00 00 00 60 67 |
| 0x003BFC1E | 00 00 00 00 00 00 00 e0 fd 7e 2c fc 3b 00 89 ac 23 00 00 00 00 00 |
| 0x003BFC40 | 60 2a c1 c6 98 fc 3b 00 78 10 38 01 83 a7 07 3e 00 00 00 00 5c fc |

变量 p 的值为 0x003BFBFC，即变量 a 的地址 &a。

| 内存 1       |  |
|------------|--|
| 地址: p      |  |
| 0x003BFBFC | 0a 00 00 00 cc cc cc cc 54 fc 3b 00 19 1a 38 01 01 00 00 00 60 |
| 0x003BFC1E | 00 00 00 00 00 00 00 e0 fd 7e 2c fc 3b 00 89 ac 23 00 00 00 00 |
| 0x003BFC40 | 60 2a c1 c6 98 fc 3b 00 78 10 38 01 83 a7 07 3e 00 00 00 00 5c |

| 内存 1           |  |
|----------------|--|
| 地址: 0x003BFBFC |  |
| 0x003BFBFC     | 0a 00 00 00 cc cc cc cc 54 fc 3b 00 19 1a 38 01 01 00 00 00 60 |
| 0x003BFC1E     | 00 00 00 00 00 00 00 e0 fd 7e 2c fc 3b 00 89 ac 23 00 00 00 00 |
| 0x003BFC40     | 60 2a c1 c6 98 fc 3b 00 78 10 38 01 83 a7 07 3e 00 00 00 00 5c |

&p 即 p 的地址 0x003BFBFD8，变量 p 占据 4 个字节。

| 内存 1            |  |
|-----------------|--|
| 地址: 0x003BFBFD8 |  |
| 0x003BFBFD8     | fc fb 3b 00 cc cc cc cc cc cc cc cc cc cc cc 41 cc cc cc cc cc |
| 0x003BFBFA      | cc cc 0a 00 00 00 cc cc cc cc 54 fc 3b 00 19 1a 38 01 01 00 00 |
| 0x003BFC1C      | 00 00 00 00 00 00 00 00 00 e0 fd 7e 2c fc 3b 00 89 ac 23 00 00 |

同理自行分析指针变量 q，并设计指向变量 c 的指针。

## 实验结果/结论

### 1. 实验结果

步骤 2 的运行结果

```
D:\软件学院工作\教学资料\C语言\2014\diy\ptr\Debug\ptr.exe
a的值为10,a的地址为0024F774
b的值为11.000000,b的地址为0024F768
c的值为0,c的地址为0024F75F
请按任意键继续. . .
```

注意：每次编译器为变量 a，b，c 分配的内存空间大小确定，受对应数据类型的约束，但是每个变量的地址未必同上次执行过程。

步骤 3 的运行结果。

```
D:\软件学院工作\教学资料\C语言\2014\diy\ptr\Debug\ptr.exe
a的值为10,a的地址为003BFBFC
b的值为11.000000,b的地址为003BFBF0
c的值为0,c的地址为003BFBF7
请按任意键继续. . .
```

注意：通过指针变量取值“\*p”，即通过地址取变量的值，\*运算优先级很高，右结合。地址捆绑空间的大小受所存对象的类型影响。如 p 存储的是整型变量 a 的地址，故而从这个地址起的 4 个字节空间存储 a 的值 10。

## 2. 实验结论

- ✓ C 语言中所有的变量都是要占据内存的，并且其占据内存大小是由变量类型所决定的。
- ✓ 指针变量存储的是地址，指针变量也是要占内存的，所有类型的指针变量在 32 位机上都占 4 个字节。
- ✓ 指针类型为不完整类型，用 \* 表示其非普通类型，指针变量绑定空间大小受存放对象的类型影响。如 `int * p` 表示，`p` 为指针变量，这 4 个字节存放整型变量的地址。
- ✓ 指针变量有时候被简称为指针，注意“指针”具体是变量还是数据类型。

## 二、 一重指针的使用以及程序分析

### 实验目的

1. 掌握一重指针的使用规范
2. 对一重指针程序进行分析

### 实验步骤

步骤 1： 使用一重指针编写程序，源代码如下。指针定义的时候可以赋初值，如 `p1` 和 `p2`，赋值后即可参加运算。

```
(全局范围)
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a = 1, b = 2, t;
    int *p1 = &a, *p2 = &b;

    printf ("a=%d, b=%d\n", a, b);

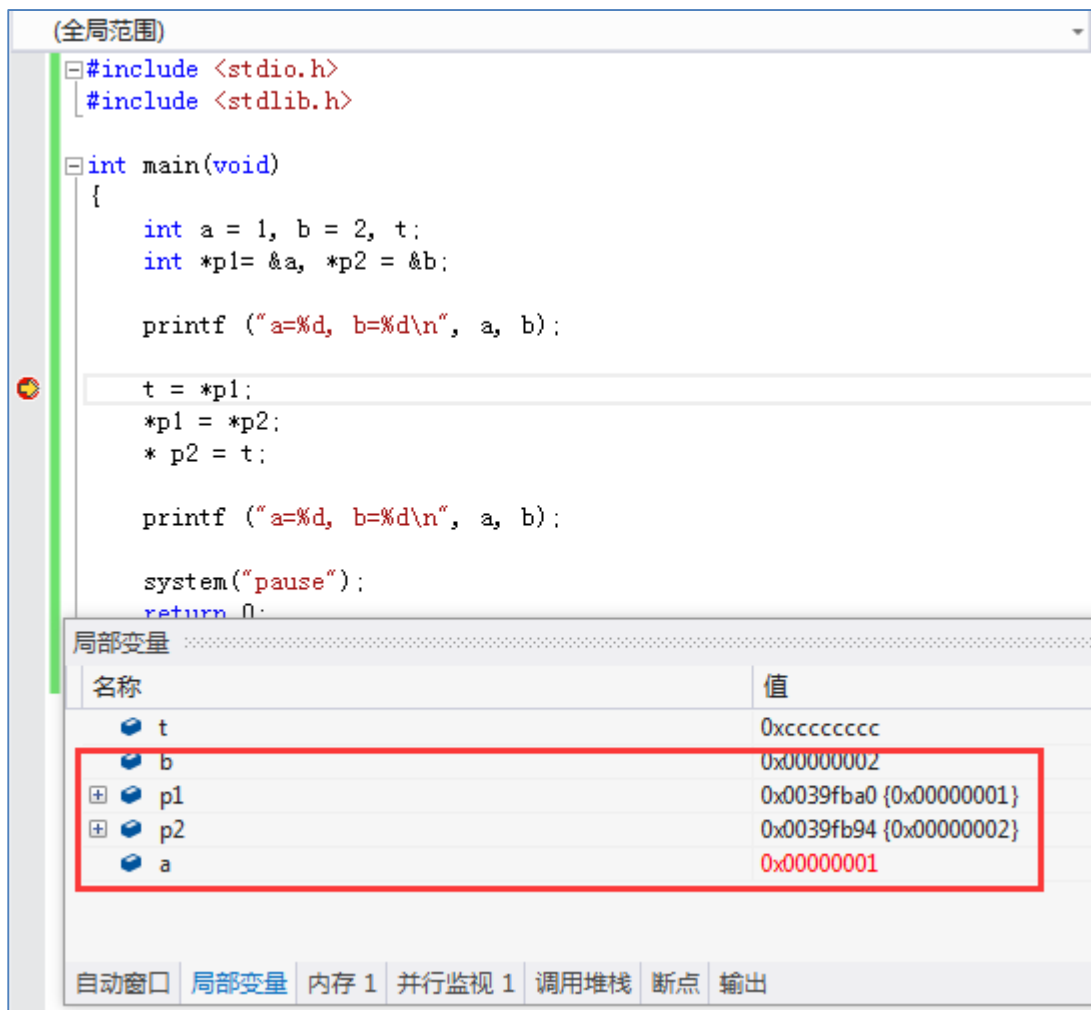
    t = *p1;
    *p1 = *p2;
    *p2 = t;

    printf ("a=%d, b=%d\n", a, b);

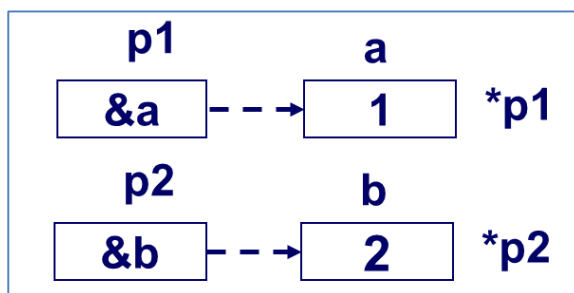
    system("pause");
    return 0;
}
```

步骤 2： 分析该程序

1. `a`, `b`, `p1` 和 `p2` 变量遵循先定义后使用，先赋值后运算的原则。



2. 内存中的存储以及指向情况，可以用如下内存示意图来展示。



3. 执行 3 条赋值语句后的存储情况。



(全局范围)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a = 1, b = 2, t;
    int *p1= &a, *p2 = &b;

    printf ("a=%d, b=%d\n", a, b);

    t = *p1;
    *p1 = *p2;
    * p2 = t;

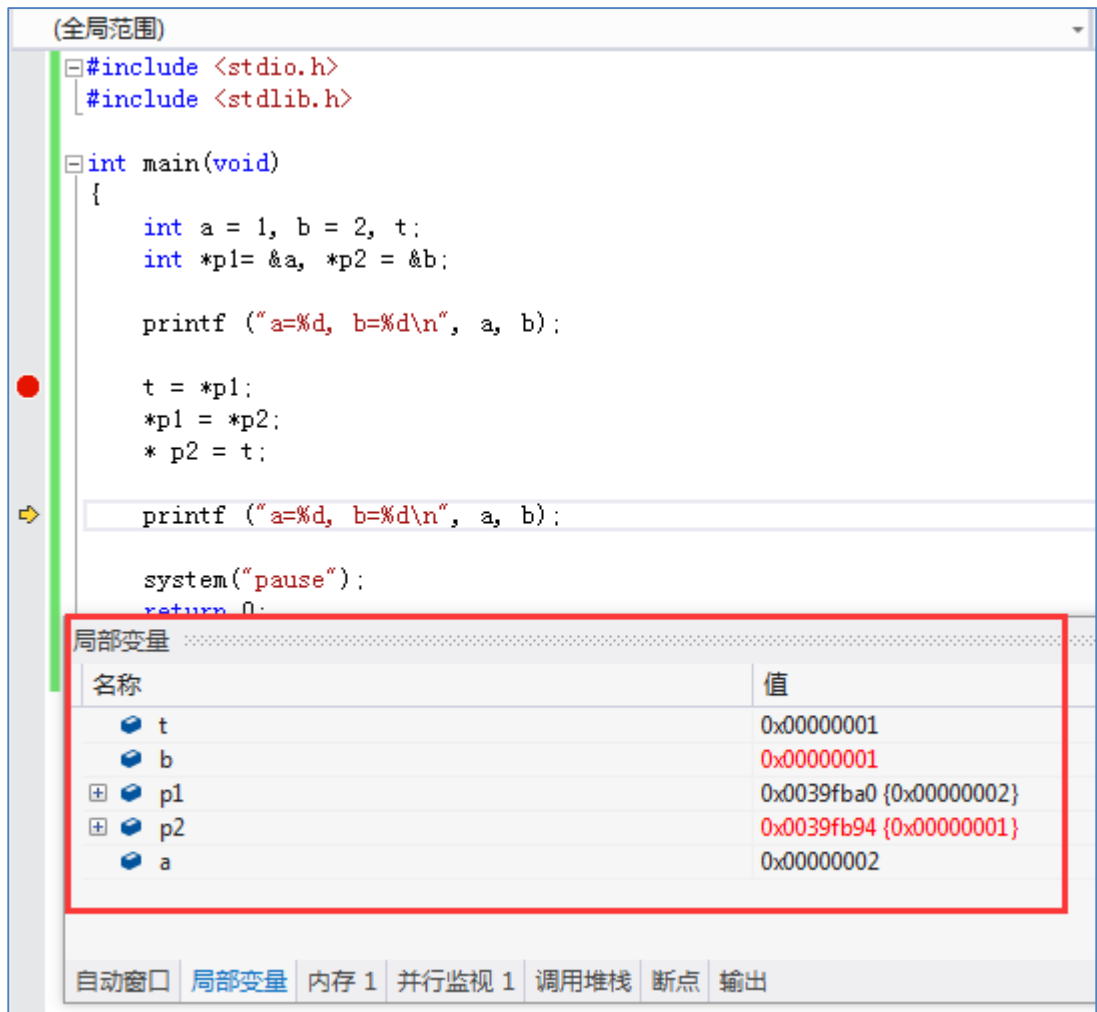
    printf ("a=%d, b=%d\n", a, b);

    system("pause");
    return 0;
}
```

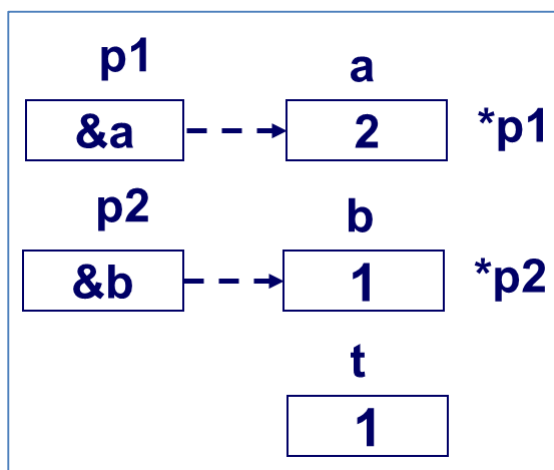
局部变量

| 名称 | 值                       |
|----|-------------------------|
| t  | 0x00000001              |
| b  | 0x00000002              |
| p1 | 0x0039fba0 {0x00000001} |
| p2 | 0x0039fb94 {0x00000002} |
| a  | 0x00000001              |

自动窗口 局部变量 内存 1 并行监视 1 调用堆栈 断点 输出

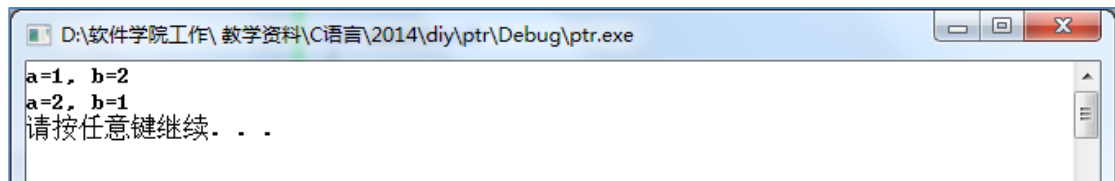


4. 内存中的存储以及指向情况，可以用如下内存示意图来展示。



## 实验结果/结论

### 1. 实验结果



```
D:\软件学院工作\教学资料\C语言\2014\diy\ptr\Debug\ptr.exe
a=1, b=2
a=2, b=1
请按任意键继续...
```

## 2. 实验结论

- ✓ 指针变量 p 的值为变量 a 的地址，称 p 指向 a
- ✓ p 是指针变量，\*p 和 p 指向变量等价（\*p1 即 a，\*p2 即 b）。
- ✓ 涉及指针变量参与运算的程序，使用内存示意图的方式辅助分析程序是一个非常不错的手段。

## 三、 多重指针的使用以及程序分析

### 实验目的

1. 掌握多重指针的使用规范
2. 对多重指针程序进行分析

### 实验步骤

- 步骤 1： 设计一重指针 m，二重指针 f，三重指针 n，分别赋值。执行程序体会各指针的指向，\*运算之后取出的是整型变量的值，还是地址。

```
(全局范围)

int a=123, b=0, c=1, d=2;
int *m, **f, ***n;
m=&a;
f=&m;
n=&f;

b=*m;
c=**f;
d=***n;

printf("a=%d\tb=%d\tc=%d\td=%d\n", a, b, c, d);
printf("m=%p\tf=%p\tn=%p\n", m, f, n);
printf("a的地址%p\tb的地址%p\tc的地址%p\td的地址%p\n", &a, &b, &c, &d);

printf("a的地址%p\tm的地址%p\tf的地址%p\tn的地址%p\n", &a, &m, &f, &n);

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

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

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


system("pause");
return 0;
}
```

D:\软件学院工作\教学资料\C语言\2014\diy\ptr\Debug\ptr.exe

a=123    b=123    c=123    d=123  
m=0030F9E8    f=0030F9B8    n=0030F9AC  
a的地址0030F9E8    b的地址0030F9DC    c的地址0030F9D0    d的地址0030F9C4  
a的地址0030F9E8    m的地址0030F9B8    f的地址0030F9AC    n的地址0030F9A0  
\*m=123  
\*f=0030F9E8  
\*\*f=123  
\*n=0030F9B8  
\*\*n=0030F9E8  
\*\*\*n=123  
请按任意键继续. . .

步骤 2： 分析程序并根据执行结果绘出内存示意图。

| 变量名 | 地址       | 变量值      |
|-----|----------|----------|
| n   | 0030F9A0 | 0030F9AC |
| f   | 0030F9AC | 0030F9B8 |
| m   | 0030F9B8 | 0030F9E8 |
| d   | 0030F9C4 | 2        |
| c   | 0030F9D0 | 1        |
| b   | 0030F9DC | 0        |
| a   | 0030F9E8 | 123      |



## 实验结果/结论

### 1. 实验结果

- ✓ 只要是指针，不管重数多少，存放的都是一个地址，如 m，f，n。
- ✓ 指针变量可以使用\*运算取内容。\*m 等价 a，取出的是整数值，\*f 等价 m，是地址，\*\*f 根据集合性转换为\*(\*f)，即\*m，等价于 a，同理可以分析三重指针 n。
- ✓ 改变 b，c，d 的值，赋值时根据等价分析赋的都是 a 的值。

### 2. 实验结论

- ✓ 多重指针即指向指针的指针，故而存放的是指针变量的地址。
- ✓ 多重指针变量赋值过程中一定要注意类型。随着重数的增加，\*运算之后具体是地址还是基本类型的变量，需要慎重。
- ✓ 注意多重指针的等价形式。

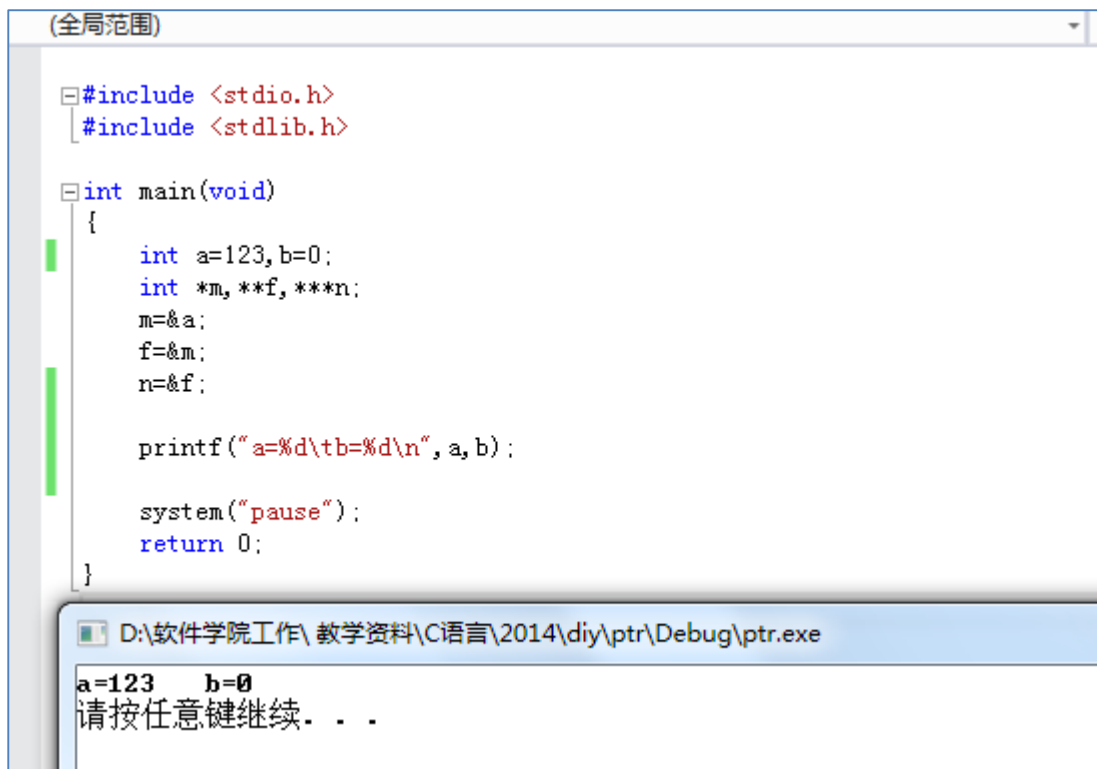
## 四、const 指针

### 实验目的

1. 掌握 const 指针的作用
2. 对 const 指针程序进行分析

### 实验步骤

步骤 1：编写一个程序，含有指针变量，掌握 const 关键字的影响。



```
(全局范围)

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a=123, b=0;
    int *m, **f, ***n;
    m=&a;
    f=&m;
    n=&f;

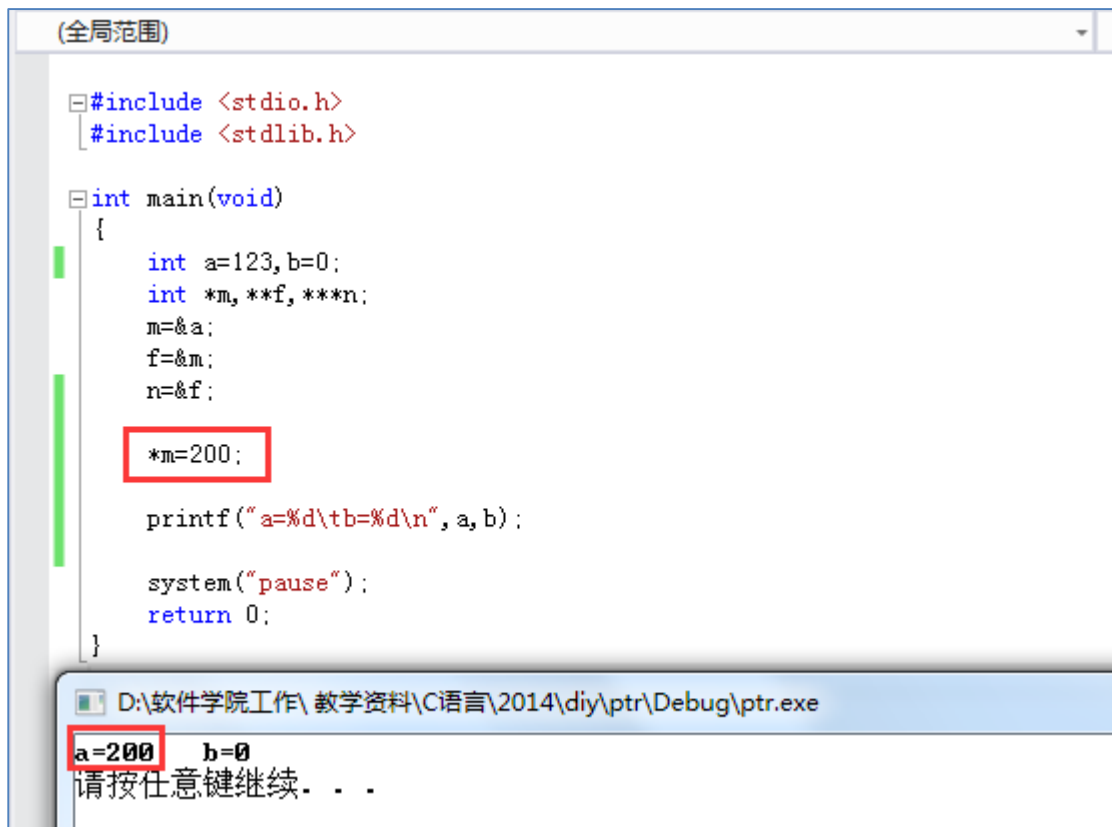
    printf("a=%d\tb=%d\n", a, b);

    system("pause");
    return 0;
}
```

D:\软件学院工作\教学资料\C语言\2014\diy\ptr\Debug\ptr.exe

a=123 b=0  
请按任意键继续. . .

1. 尝试通过一重指针 m 对整型变量 a 的访问操作。



```
(全局范围)

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a=123, b=0;
    int *m, **f, ***n;
    m=&a;
    f=&m;
    n=&f;
    *m=200;

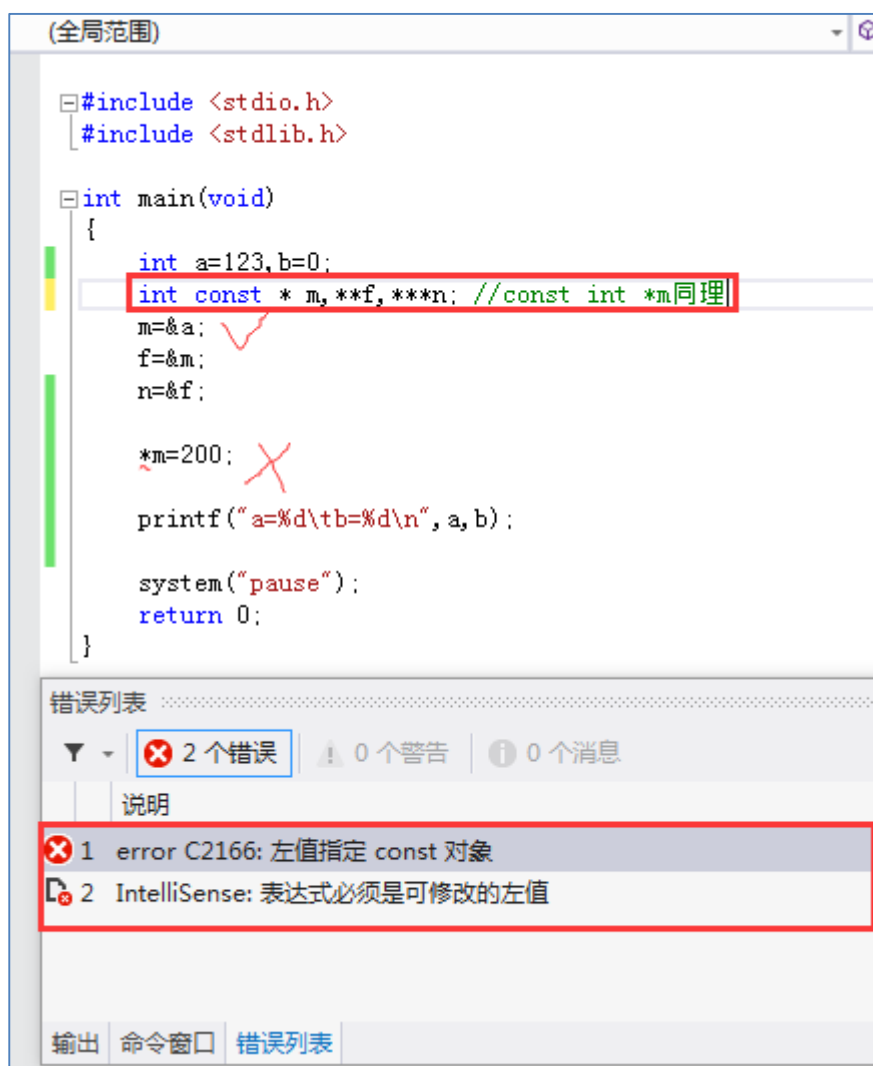
    printf("a=%d\tb=%d\n", a, b);

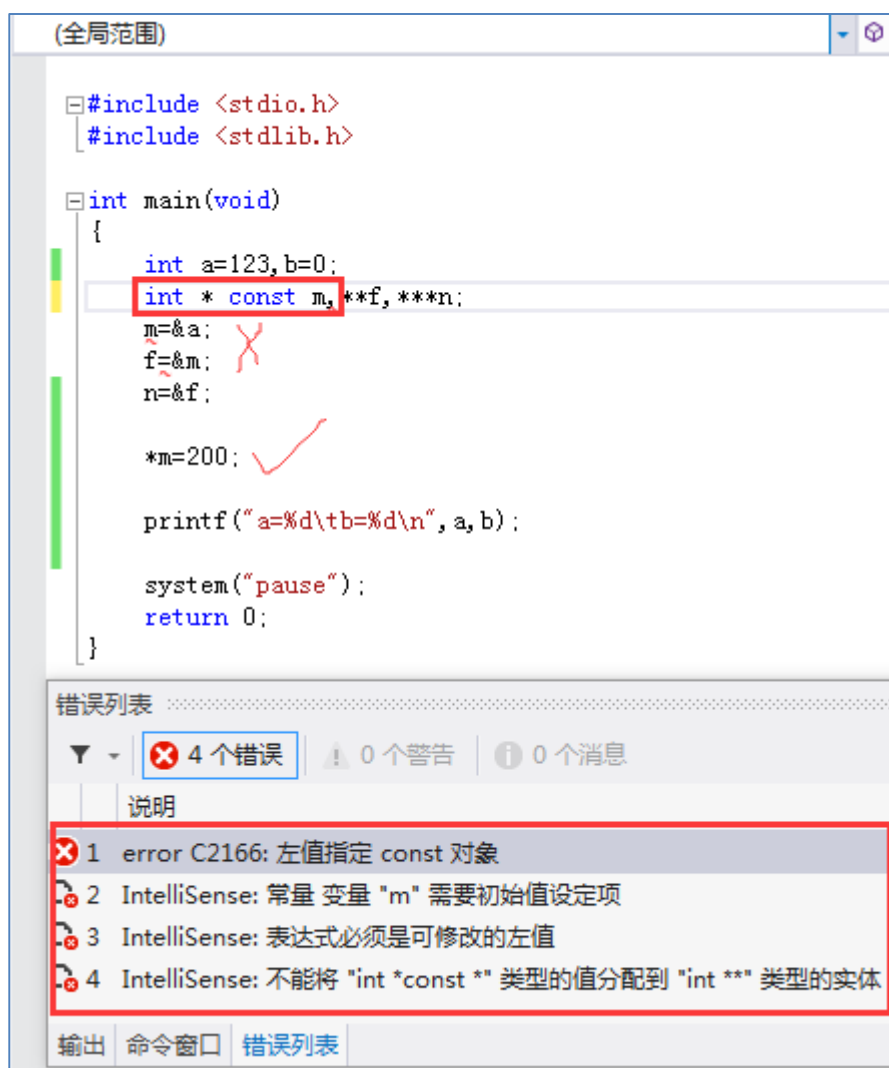
    system("pause");
    return 0;
}
```

D:\软件学院工作\教学资料\C语言\2014\diy\ptr\Debug\ptr.exe

a=200 b=0  
请按任意键继续. . .

2. 在指针 m 的定义过程中加上 const 关键字，根据 const 位置的不同，体会访问的限制。





步骤 2: 用同样的方式分析下二重指针  $f$  的使用过程中 `const` 的位置以及相应的影响。

从编译环境中总结, 从而理解下表。

| const 位置                            | 受限的赋值              | 可以做赋值  |
|-------------------------------------|--------------------|--|
| <code>int * <b>const</b> * f</code> | <code>*f</code>    | <code>f, m</code> ( <code>*f</code> 的等价), <code>**f</code> |
| <code>int ** <b>const</b> f</code>  | <code>f</code>     | <code>*f, **f</code>                                       |
| <code>int <b>const</b> ** f</code>  | <code>**f</code>   | <code>f, *f, a</code> ( <code>**f</code> 的等价)              |
| <code>int *const *const f</code>    | <code>f, *f</code> | <code>**f, m</code>  |

下图为内存示意图, 不同颜色对应着上图 `const` 位置所对应颜色的内存。

| 变量名 | 变量地址 | 变量值 |
|-----|------|-----|
| a   | &a   | 123 |
| b   | &b   | 0   |
| m   | &m   | &a  |
| f   | &f   | &m  |

步骤 3: 自己设计并分析三重指针  $n$  使用过程中, `const` 位置以及相应的影响。

## 实验结果/结论



### 1. 实验结果

- ✓ `int const * m=&a`, 不能使用对\*m 赋值的操作, 但是改变 m, 改变 a 的值都是可以的。
- ✓ `int * const m=&a`, 因为 m 是不可修改, 所以定义的时候要初始化。

### 2. 实验结论

- ✓ `const` 出现的位置不同意义不同, 通过分析以及 vs 中验证可总结出各自的特点。
- ✓ `const` 后面紧跟的变量不可改变, 其他不受影响, 如

## 6.2 理论题

### A 类

#### 一、填空题

1. 若 d 是已定义的双精度变量, 再定义一个指向 d 的指针变量 p 的代码是\_\_\_\_\_.
2. “\*” 称为\_\_\_\_\_ 运算符, “&” 称为\_\_\_\_\_运算符。
3. &后跟变量名, 表示该变量的\_\_\_\_\_; \*后跟指针变量名, 表示该指针变量\_\_\_\_\_. &后跟的是指针变量名, 表示该指针变量的\_\_\_\_\_.

#### 二、选择题

1. 设有定义“`int a=3, b, *p=&a;`”, 则下列语句中使 b 不为3 的语句是 ( )
  - A. `b=* &a`
  - B. `b=* p`
  - C. `b=a`
  - D. `b=*a`
2. 设指针 x 指向的整型变量值为25, 则“`printf ("%d\n", ++ *x);`”的输出是 ( )
  - A. 23
  - B. 24
  - C. 25
  - D. 26
3. 若有说明: “`int i, j= 7, *p=&i;`”, 则与 “`i=j;`” 等价的语句是 ( )
  - A. `i=*p;`
  - B. `*p=* &j;`
  - C. `i=&j;`
  - D. `i=* *p;`
4. 若有如下定义和语句, 则输出结果是 ( )

```
int **pp, *p, a=10, b=20; pp=&p; p=&a; p=&b;
printf ( “%d%d\n”, *p, **pp);
```

  - A. 10, 20
  - B. 10, 10
  - C. 20, 10

D. 20, 20

5. 有四组对指针变量进行操作的语句，以下判断正确的选项是（ ）。

```
(1)int *p,*q; q=p;
    int a,*p,*q; p=q=&a;
(2)int a,*p,*q; q=&a; p=*q;
    int a=20,*p;*p=a;
(3)int a=b=0,*p; p=&a; b=*p;
    int a=20,*p,*q=&a; *p=*q;
(4)int a=20,*p,*q=&a; p=q;
    int p,*q; q=&p;
```

- A. 正确：(1) 不正确：(2),(3),(4)  
B. 正确：(1),(4) 不正确：(2),(3)  
C. 正确：(3) 不正确：(1),(2),(4)  
D. 以上结论都不正确

6. 以下程序中调用 scanf 函数给变量 a 输入数值的方法是错误的，其错误的原因是 。

```
main()
{int *p,*q,a,b;
  p=&a;
  printf("input a:");
  scanf("%d",*p);
  ...
}
```

- A. \*p 表示的是指针变量 p 的地址  
B. \*p 表示的是变量 a 的值，而不是变量 a 的地址  
C. \*p 表示的是指针变量 p 的值  
D. \*p 只能用来说明 p 是一个指针变量

### 三、综合题

1. 读程序题，写出程序运行的结果。

```
#include <stdio.h>

void main()
{ int *p1,*p2,*p;
  int a=5,b=8;
  p1=&a; p2=&b;
  if(a<b) { p=p1; p1=p2; p2=p;}
  printf("%d,%d\n",*p1,*p2);
  printf("%d,%d\n",a,b);
}
```

2. 写程序：输入 3 个整数，按从大到小顺序输出。要求不改变存储这 3 个整数的变量的值。  
3. 设 x 是一个整型变量，并设有 `int * const p1=&x; const int *p2=&x; const int * const p3=&x;`；分别分析 const 对 p1、p2、p3 的限定。

## B 类 (指针结合数组、函数, 不作为基础知识, 这没有展开)

### 一、填空题

1. 若两个指针变量指向同一个数组的不同元素, 可以进行减法运算和\_\_\_\_\_运算。
2. 设有以下定义和语句, 则 $*(*(p+2)+1)$ 的值为\_\_\_\_\_。  
`int a[3][2]={10, 20, 30, 40, 50, 60}, (*p)[2]; p= a;`

### 二、选择题

1. 若有说明语句“`int a[10], *p=a;`”, 对数组元素的正确引用是 ( )  
A. `a[p]`  
B. `p[a]`  
C. `*(p+2)`  
D. `p+2`
2. 下面各语句中, 能正确进行赋字符串操作的语句是 ( )  
A. `char s[5]={ "ABCDE" };`  
B. `char s[5]={ 'A', 'B', 'C', 'D', 'E' };`  
C. `char *s; s= "ABCDE";`  
D. `char *s; scanf ( "%s", &s );`
3. 若有以下定义, 则数值为 4 的表达式是 ( )

`static int w[3][4]={ {0, 1}, {2, 4}, {5, 8} }; int (*p)[4]=w;`

- A. `*w[1]+1`
- B. `p++, *(p+1)`
- C. `w[2][2]`
- D. `p[1][1]`

### 二、综合题

1. 以下程序的运行结果是\_\_\_\_\_。

```
void sub (int x,int y,int *z)
{
    *z=y-x;
}
```

```
int main(void)
{
    int a,b,c;
    sub(10,5,&a);
    sub(7,a,&b);
    sub(a,b,&c);
    printf("%4d,%4d,%4d",a,b,c);
}
```

# 本章答案

## A 类

### 一、 填空题

1. `double *p=&d.`
2. “\*”称为取内容运算符，“&”称为取地址运算符。
3. &后跟变量名，就表示该变量的地址。\*后跟指针变量名，表示该指针变量所指变量的内容。若&后跟的是指针变量名，就表示该指针变量的地址。

### 二、选择题

1. D
2. D
3. B
4. D
5. D
6. B

### 三、综合题

1. 8, 5  
5, 8
2. 设用三个简单变量x, y, z 存储输入的三个整数，如通过比较交换指针变量，当比较后发现要交换时，就交换变量的指针，而不交换变量的值，则在比较结束后，变量的值没有改变，但从指针方向来看，它们的值是从大到小排列的。

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int x,y,z;
```

```
    int *big=&x,*mid=&y,*sma=&z; /*置三个指变量分别指向x, y, z*/
```

```
    int *temp;
```

```
    printf("Enter x, y, z.\n");
```

```
    scanf("%d,%d,%d", big,mid,sma);/*顺序为变量x, y, z 输入值*/
```

```
    if(*big< *mid) { temp=big;big=mid; mid=temp; }/*使*big>=*mid*/
```

```
    if(*big<*sma) { temp=big;big=sma;sma=temp; }/*使*big>=*sma*/
```

```
    if(*mid<*sma ) { temp = mid ; mid=sma ;sma=temp; }/*使*mid>=*sma*/
```

```
    printf("%d\t%d\t%d\n", x,y,z);/*按输入顺序输出x, y, z*/
```

```
    printf("%d\t%d\t%d\n",*big,*mid,*sma);/*按从大到小的顺序输出*/
```

```
    system("pause");
```

```
    return 0;
```

}

3. p1不可改变, 即p1只能指向x变量; \*p2不可改变, 即不能通过\*p2来改变x变量的值, 但是对x变量直接赋值修改是可以的; p3和\*p3都不可改变, 即p3只能指向x变量, 也不能通过\*p3来修改x的值。

## B 类

### 一、 填空题

1. 若两个指针变量指向同一个数组的不同元素, 可以进行减法运算求它们所指元素相差多少元素。进行关系运算, 判定它们所指元素的前后, 或是否指向同一个元素等
2. 60

### 二、 选择题

1. C
2. C
3. D

### 四、 综合题

1. -5, -12, -7