·第 3 章·
内存操作——指针

常见考点

- 一级指针的用法和一级指针移动时的步长计算。
- 两个指针之间的运算和地址越界问题。
- 常量指针的相关问题。
- 野指针导致程序执行错误的问题。
- 多级指针的用法和多级指针移动时的步长计算。
- void *和 void **的含义。

3.1 指 针 基 础

3.1.1 要点归纳

⋙ 1. 指针变量

变量用于存放数据，指针变量也是如此，只不过指针变量存放的不是普通数据而是其他变量的地址。

在 C 中提供了两种指针运算符。

- *：取指针所指值运算符，也称为解引用。
- &：取地址运算符。

例如，在定义“int x,*px”之后：

```
px=&x
```

将整数变量 x 的地址赋给整数指针 px，即 px 指针变量中存放 x 的地址值，如图 3.1 所示。例如：

```
x=*px          //将 px 所指的值赋给 x
px=&(*px)      //将 px 所指值的地址赋给 px，等同于 px=px
x=*(&x)        //将 x 的地址中的值赋给 x，等同于 x=x
```

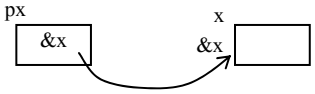


图 3.1　px 指针指向变量 x

⋙ 2. 指针变量的定义和初始化

指针变量也是一个变量，通常简称为指针，所以具有和普通变量一样的属性，在使用之前也需要定义，在指针定义的同时也可以进行初始化。指针定义的一般形式如下：

```
数据类型 *指针名;
```

其中指针变量名前的“*”号仅是一个说明符，表示其后的名称是一个指针变量名。这里的“*”号并没有访问指针目标的含义。

与普通变量不同，指针定义时的数据类型并不是指针变量本身的数据类型，而是其所指目标数据的数据类型，称之为基类型。指针初始化的一般形式如下：

```
数据类型 *指针名=初始地址值;
```

指针初始化的过程是系统按照指定的基类型和存储类别在相应存储区域为该指针分配存储空间，并把初始值置入指针的存储空间中，从而该指针就指向了初始地址值所给定的内存空间。例如：

```
double x;
double *px=&x;
```

把变量 x 的地址（如 0x12ff40）作为初值赋给 double 型指针 px，从而 px 指向了变量 x 的存储空间。“double *px=&x”等同于“double *p,p=&x”。

有关指针的注意事项如下：

❶ 在定义指针变量后系统为该指针变量分配一个地址大小的存储单元，指针变量中存放的是地址值。无论指针变量的基类型是何种数据类型，占用的内存大小都是相同的，例如“char *p1;double*p2;”，在 32 位系统中有 sizeof(p1)=sizeof(p2)=4，即 p1 和 p2 两个指针变量均占用 4 个字节。

❷ 尽管任何指针变量中存放的都是地址值，但一个指针变量只能存放相同基类型的数据地址。

❸ 指针变量的存储单元和它指向的数据空间是两个不同的存储空间。通过指针变量可以对所指数据进行间接操作（相应地通过变量名对其数据的操作看成是直接操作），指针变量所指的空间可以看成是匿名变量空间。

❹ 一个指针变量本身占用的空间很小，但可以指向一整块很大的连续空间。这个一整块的连续空间通常采用第 1 章介绍的动态分配函数分配，存放在堆空间中。

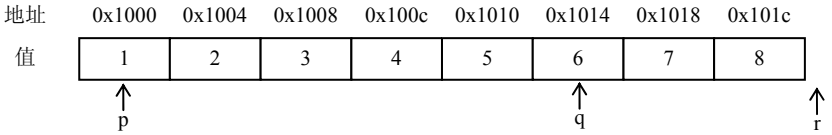
❺ 指针与地址有什么区别？指针意味着已经有一个指针变量存在，其值是一个地址，指针变量本身也有地址；而地址本身并不代表有任何变量存在，地址仅表示内存空间的一个位置。

⋙ 3. 指针运算

指针运算是以指针变量中的地址值为运算量进行的运算，所以指针运算实际上是地址

的计算，如图 3.2 所示。

- **指针与整数的加减**：一个地址量加上或减去一个整数 n 是以该地址量为基点的前方或后方的第 n 个数据的地址。例如对于基类型为 type 的指针 p，p±n 的结果是地址值 p±n*sizeof(type)，其中 sizeof(type)称为步长，显然步长与指针变量的基类型相关。
- **指针自加 1、自减 1**：指针++运算后指向下一个数据的位置，指针--运算后指向上一个数据的位置。运算后指针地址值的变化量取决于指针的基类型。
- **两个指针相减**：两个地址量相减，并非它们的两个地址值之间直接做减法运算，而是两指针所指地址之间的数据个数。
- **两个指针的关系比较**：当两个指针都指向一整块连续空间中的元素时，它们之间的关系运算表示它们指向地址的前后关系，通常指向后方的地址大于指向前方的地址。如果两个指针不是指向同一连续空间中的元素，它们的关系比较是没有意义的。指针与一般整数常量或变量之间的关系运算也是无意义的。但是指针可以和零（指针零用 NULL 表示）之间进行等于或不等于的关系运算，即 p==NULL 或 p!=NULL，以判定指针 p 是否为一空指针。
- **NULL（与 0 等效）**：NULL 表示空指针值，它是一个特殊的指针值。一旦某个指针赋值为 NULL，就将该指针值设置为 0，表示指针变量值没有意义，就不能再在该指针所指向的内存中存储数据，例如“int *p=NULL,*p=10;”是错误的。但可以再修改指针 p 的值为其他有效的地址值，例如“int *p=NULL,n=10; p=&n;”是正确的。



- 共有 n=8 个元素，每个元素占用 m=4 个字节，即步长为 4
- q = p + 5 = 0x1000 + 5*m = 0x1014
- q - p = (0x1014 - 0x1000)/m = 5
- r = p + 9，越界
- p<q，q>p 为真
- p+q、p*q、p/q，没有意义

图 3.2　指针之间的运算

▷▷▷ 4. 野指针

一个指针变量的值（地址值）为垃圾值的指针变量称为野指针。产生野指针的原因及解决办法如下：

❶ 指针变量定义时没有被初始化。解决办法是在定义指针时初始化，可以是具体的地址值，也可以是 NULL。

❷ 指针 p 被 free 或者 delete 之后没有置为 NULL，后面还使用它。解决办法是指针指向的内存空间被释放后指针应该指向 NULL。

❸ 指针操作超越了所指变量的作用域。解决办法是在所指变量的作用域结束前释放掉变量的地址空间，并让指针指向 NULL。

野指针的解决方法是编程规范的基本原则，在编程中使用指针时一定要避免产生野指针，在使用指针前一定要检验指针的合法性。

3.1.2　面试真题解析

【面试题 3-1】字符指针、浮点数指针以及函数指针这 3 种类型的变量哪个占用的内存最大？为什么？

答：指针变量也占用内存单元，而且所有类型的指针变量占用的内存大小都是相同的。也就是说，不管是指向何种对象的指针变量，它们占用的内存空间的字节数都是一样的，通常是一个机器字长。

【面试题 3-2】以下对 C 语言的“指针”的描述不正确的是（　　）。

A．32 位系统下任何类型的指针的长度都是 4 个字节

B．指针的数据类型声明的是指针实际指向内容的数据类型

C．野指针是指向未分配或者已释放的内存地址

D．当使用 free 释放掉一个指针内容后，指针变量的值被置为 NULL

答：当使用 free 释放掉一个指针内容后，只是表示它指向的内容被释放了，但该指针变量的值并没有置为 NULL。答案为 D。

【面试题 3-3】有如下定义，则下列符号中均正确地代表 x 的地址的选项是（　　）。

```
int x,*p;
p=&x;
```

A．&x、p、&*x　　B．*&x、p　　C．*p、&p、x　　D．&x、&*p、p

答：&x 为 x 的地址，p=&x，所以 p 可以表示 x 的地址，&*p 即为 p。答案为 D。

【面试题 3-4】有下列基本类型相同的指针 p1、p2，则下列运算不合理的是（　　）。

A．p1/=5　　B．p1−p2　　C．p1=p2　　D．p1==p2

答：对指针除以 5 是没有意义的。答案为 A。

【面试题 3-5】若有定义“int i,j=2,*p=&I;”，则能完成 i=j 赋值功能的语句是（　　）。

A．i=*p;　　B．*p=*&j;　　C．i=&j;　　D．i=**p;

答：p 指向变量 i，而*&j=j，所以*p=*&j 的功能是执行 i=j。答案为 B。

【面试题 3-6】若有以下定义和赋值语句：

```
int a=4,b=3,*p,*q,*w;
p=&a; q=&b; w=q; q=NULL;
```

则以下选项中错误的语句是（　　）。

A．*q=0;　　B．w=p;　　C．*p=a;　　D．*p=*w;

答：q 是指针变量，已通过 q=NULL 语句将其置为 NULL，不能再设置所指向的数据。答案为 A。

【面试题 3-7】以下程序的输出是（　　）。

```
int *pint=0;
pint +=6;
printf("%d\n",pint);
```

A．12　　B．72　　C．24　　D．0

E．6　　F．任意数

答：int *pint=0（与 int *pint=NULL 等同）是在定义指针变量 p 的同时把 p 的值设置为 0，执行 pint+=6，pint=0+6*sizeof(int)=24。答案为 C。

尽管 pint 中的地址值为 24，但作为地址值是没有意义的，当执行*p=10 时出现程序崩溃。

【面试题 3-8】以下代码执行后 p1+5 和 p2+5 分别是多少？

```
unsigned char *p1;
unsigned long *p2;
p1=(unsigned char *)0x801000;
p2=(unsigned long *)0x810000;
```

答：p1 是基类型为 unsigned char 的指针变量，其步长为 sizeof(unsigned char)=1，p1+5=p1+5*sizeof(unsigned char)=0x801000+5=0x801005。p2 是基类型为 unsigned long 的指针变量，其步长为 sizeof(unsigned long)=4，p2+5=p1+5*sizeof(unsigned long)= 0x810000+20=0x810014。

【面试题 3-9】已有变量定义和函数调用语句：

```
int a=25;
print_value(&a);
```

则下面函数的正确输出结果是（　　）。

```
void print_value(int *x)
{
```

```
    printf("%x\n",++*x);
}
```

A．25　　B．26　　C．19　　D．1a

答：a=25，执行 print_value(&a)时 x 指向 a 变量，++*x 等同于++(*x)，将*x（即 25）加 1 后返回，26 对应的十六进制数为 1a。答案为 D。

【面试题 3-10】以下程序的输出结果是什么？

```
#include <stdio.h>
void main()
{   int a;
    int *p;
    p=&a;                    //让指针 p 指向整型变量 a
    *p=0x500;                //置 p 指向的变量 a 的值为 0x500
    a=(int)(*(&p));          //相当于 a=(int)p，则 a 为 p 中的地址值，如 0x12ff44
    a=(int)(&(*p));          //相当于 a=(int)p，则 a 为 p 中的地址值，如 0x12ff44
    if(a==(int)p)            //a 等于 p 中的地址值，所以为真
        printf("equal!\n");
    else
        printf("not equal!\n");
}
```

答：见程序注释，输出结果为 equal!。

【面试题 3-11】以下程序的输出结果是什么？

```
#include <stdio.h>
void main()
{   int *p1, *p2;
    int value;
    p1=(int *)0x500;
    p2=(int *)0x508;
    value=p2-p1;
    printf("%d\n",value);
}
```

答：p1、p2 指针的基类型为 int，通过强制转换让 p1 的值为 0x500、p2 的值为 0x508，value=p2−p1=(0x508−0x500)/sizeof(int)=8/4=2。所以输出 2。

【面试题 3-12】有以下代码：

```
unsigned char *p1;
unsigned long *p2;
p1=(unsigned char *)0x801000;
p2=(unsigned long *)0x810000;
```

在 32 位系统中执行后 p1+5 和 p2+5 的值分别是（　　）。

A．0x801005、0x810005　　B．0x80100A、0x81000A

C．0x801014、0x810014　　D．0x801005、0x810014

答：p1 的基类型为 unsigned char，该类型数据的长度为 1 个字节，所以 p1+5= 0x801000+5*1=0x801005。p2 的基类型为 unsigned long，该类型数据的长度为 4 个字节，所以 p2+5=0x810000+5*4=0x801014。答案为 D。

【面试题 3-13】以下程序执行时会输出什么？为什么？

```
#include <stdio.h>
int main(void)
{   char *ptr="Linux";
    printf("[%c], ",*ptr++);
    printf("[%c]\n",*ptr);
    return 0;
}
```

答：ptr 指向常量字符串"Linux"，*ptr++返回 ptr 指向的字符 L，并自增 1 指向 i 字符。再执行*ptr 返回 i。输出结果为“[L],[i]”。

【面试题 3-14】请问以下代码有什么问题：

```
#include <stdio.h>
#include <string.h>
int main()
{   char a;
    char *str=&a;
    strcpy(str,"hello");
    printf(str);
    return 0;
}
```

答：程序中没有为字符指针变量 str 分配内存空间就将一个字符串复制到它所指的地址中。虽然可以正确地输出结果，但因为越界进行内在读写而导致程序崩溃。程序应该改为如下：

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
int main()
{   char a;
    char *str=&a;
    str=(char *)malloc(6*sizeof(char));
```

```
    strcpy(str,"hello");
    printf(str);
    free(str);
    return 0;
}
```

【面试题 3-15】以下这段程序中有什么错误：

```
void swap(int *p1,int *p2)
{   int *p;
    *p=*p1;
    *p1=*p2;
    *p2=*p;
}
```

答：在 swap 函数中 p 是一个野指针，并没有分配指向的空间，当执行*p=*p1 时会导致程序执行崩溃。如果要交换 p1、p2 指向的数据，可以改为：

```
void swap(int *p1,int *p2)
{   int p;
    p=*p1;
    *p1=*p2;
    *p2=p;
}
```

【面试题 3-16】请问执行 Test 函数会有什么样的结果？

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
void Test(void)
{   char *str=(char *)malloc(100);
    strcpy(str,"hello");
    free(str);
    if(str!=NULL)
    {   strcpy(str,"world");
        printf(str);
    }
}
```

答：因为执行 free(str)语句后 str 成为野指针，其中地址值是不为 NULL 的垃圾地址，将"world"字符串存放在垃圾地址空间中，后果难以预料，非常危险。

【面试题 3-17】下面这段代码是把中英文混合字符串（汉字用两个字节表示，特点是

第 1 个字节的最高位为 1）中的大写字母转化为小写字母，请找出其中的 bug，注意各种异常情况。

```
for (char *piterator=szWord; *piterator!=0; piterator++)
{   if(*piterator&0x80!=0)
        piterator++;
    else if(*piterator>='A'&&*piterator<='Z')
        piterator+=32;
}
```

答：上述代码中有两处错误，即位运算符&的优先级较!=0 低，应该加括号；piterator += 32 移动指针是错误的。修改代码如下：

```
for (char *piterator=szWord;*piterator!=0;piterator++)
{   if((*piterator&0x80)!=0)    //为汉字
        piterator++;
    else if(*piterator>='A'&&*piterator<='Z')
        *piterator+=32;
}
```

3.2 常量和常量指针

3.2.1 要点归纳

⋙ 1. 常量

在程序执行期间其值不能被改变的量称为常量，常量分为字面常量和符号常量。

1 字面常量

字面常量只能引用不能修改，如 123 等，通常保存在程序符号表中，程序无法读取字面常量的地址，只有一个例外，即字符串常量。例如：

```
char *p="abc";
```

其中"abc"就是一个字符串常量，它存放在静态数据区，由 p 指针指向它，不能通过 p 指针修改该常量。

上述方式定义字符串常量的注意事项如下：

❶ 像 char *p="abc"这种写法在 C 中实在太多了，所以许多 C 编译器不会指出编译错误，但程序员必须理解其含义，其中"abc"为常量，程序员最好采用 const char *p="abc"定义，这样在其后执行 p[0]='x'或者*p='x'时会发生“l-value specifies const object（左值说明的是一个常

量对象）”的编译错误，以便指定错误原因。

❷ 那么 int *p=123 可不可以呢？结论是不允许的，尽管 123 也是常量，这里编译器认为是将 123 作为地址存放在指针变量 p 中，而 123 是整数，正确的做法是 int *p=(int *)123，即将 123 转换为地址值赋给 p。当然这种做法是有危险的。

❸ 由于 p 指向的常量字符串不是通过 malloc 函数分配的，执行 free(p)会导致程序崩溃。

2 符号常量

符号常量主要有两种定义方法，第 1 种是用宏定义实现（即宏常量），例如：

```
#define PI 3.14     //定义一个常量为 3.14 的宏 PI
```

实际上，#define 是定义字面常量。

第 2 种是用 const 定义（即 const 常量），const 的意思是“一个不能被改变的变量”。例如：

```
const int n=123;    //通过 const 定义一个 int 型常量 n
n++;                //错误：不能修改常量
n=10;               //错误：不能修改常量
```

上述两种方法的区别如下：

- 前者是宏替换命令，不是语句，所以不以“;”号结尾，后者是定义，以“;”号结尾。
- 前者在预处理时进行替换，后者定义的常量像变量一样（称为常变量），只是其值不能改变。
- const 常量有数据类型，而宏常量没有数据类型。编译器对前者进行类型安全检查，对后者不进行类型安全检查。

const 常量的注意事项如下：

❶ 实际上 const 修饰的是只读变量，具有变量的某些属性。例如在一个函数内定义“const int n=123;”，n 像局部变量一样是在栈空间中分配的。例如又定义“const in m=13;”，m 和 n 的存储地址是不同的。

❷ 用 const 修饰的常量的值不能修改，所以必须在定义时初始化。

❸ 可以把一个常量赋值给非常量变量，例如 int k=n 是正确的。

❹ 可以把 const 常量的地址赋给指针变量，例如“const int i=10；int *p=&i;”。C 中默认这种隐式转换，但在 C++中出现“cannot convert from 'const int *' to 'int *'”编译错误，需要采用强制转换，即 int *p=(int *)&i。

▷▷▷ 2. const 指针常量

在定义指针时用 const 关键字进行修饰，称为 const 指针常量，有以下几种情况。

1 常量指针

用 const 修饰*时称为常量指针，这样不能通过该指针变量修改指向的内容。例如：

```
const char *p;
```

这样不能通过 p 指针修改指向的内容，否则会出现“l-value specifies const object（左值说明为常对象）”的编译错误。例如：

```
const char *p;  //定义常量指针 p
char *q;
q=(char *)malloc(10*sizeof(char));
strcpy(q,"123");
p=q;            //正确：让 p 也指向"123"
*q='x';         //正确：可以通过非常量指针修改所指内容,p 指向的字符串改变为"x23"
*p='y';         //错误：不能通过常量指针修改所指内容
```

2 常量指针变量

在 C++中还可以用 const 修饰指针变量名，称为常量指针变量，例如：

```
char * const p;
```

表示指针 p 是一个常量指针变量，p 的值不能再发生改变，所以必须初始化。一旦初始化，p 不能指向其他数据，但可以通过指针 p 修改所指的内容。例如：

```
char *q;
q=(char *)malloc(10*sizeof(char));
strcpy(q,"abc");
char * const p=q;           //定义常量指针变量 p
*p='1';                     //正确：可以通过 p 修改指向的数据
p=q;                        //错误：不能再修改 p 的值，即使 p=p 也不行
```

3 指针常量

指针常量既为常量指针，又是常量指针变量。例如：

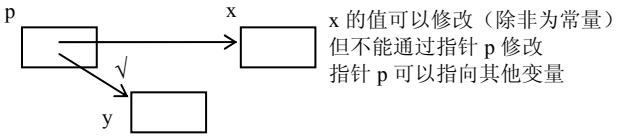
```
const char * const p;
```

表示指针 p 是一个指针常量，前一个 const 修饰指针变量的定义，后一个 const 修饰变量名。指针 p 必须初始化，并且 p 的值和指向的内容都不能修改。例如：

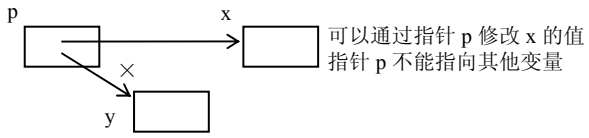
```
char *q;
q=(char *)malloc(10*sizeof(char));
strcpy(q,"abc");
const char * const p=q;     //定义指针常量 p
*p='1';                     //错误：不能修改指向的数据
p=q;                        //错误：不能修改指针值
```

归纳起来，3 种类型的 const 指针常量的差别如图 3.3 所示。

❶ 常量指针：const char *p=&x;（const 修饰*，表示不能修改 p 指向的内容）



❷ 常量指针变量：char * const p=&x;（const 修饰变量名，表示不能修改变量 p）



❸ 指针常量：const char * const p=&x;（前一个 const 修饰*，后一个修饰变量名）

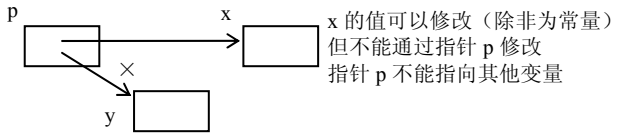


图 3.3　3 种类型的常量指针的差别

3.2.2　面试真题解析

【面试题 3-18】有如下程序：

```
#include <stdio.h>
char *GetStr()
{   char *tmp;
    tmp="123";
    return tmp;
}
void main()
{
    printf("%s\n",GetStr());
}
```

问上述程序会输出"123"吗？"123"创建在堆上还是栈上？"123"的空间是什么时候释放的？

答：执行 tmp="123"时，由于之前没有为指针 p 动态分配空间来存放"123"，所以编译

器把"123"作为常量放到静态数据区中。实际上，char *tmp 和 tmp="123"即为 char *tmp="123"，等同于 const char *tmp="123"。所以"123"既不在堆中，也不在栈中存放，而是放到静态数据区，它在程序执行结束时自动释放。

【面试题 3-19】指出以下程序的错误。

```
#include <stdio.h>
void main()
{   char *p="ABCD";
    p[0]='a';
    printf("%s\n",p);
}
```

答：该程序没有编译问题，但会出现执行错误。字符串"ABCD"是常量，存放在静态数据区，其首地址保存在 p 变量中。p 是一个字符指针，指向这个字符串常量，常量是不能修改的，所以执行 p[0]='a'（或者*p='a'）时出错。

实际上，定义指针变量 p 并初始化时就有问题，应为 cosnt char *p="ABCD"，这样对 p[0]的赋值操作是不合法的，会出现“l-value specifies const object（左值为常数对象）”的编译错误。

如果改为如下程序，编译和执行都没有问题：

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
void main()
{   char *p;
    p=(char *)malloc(5*sizeof(char));
    strcpy(p,"ABCD");
    *p='a';
    printf("%s\n",p);
}
```

【面试题 3-20】以下代码有什么错误？

```
char *s="AAA";
printf("%s",s);
s[0]='B';
printf("%s",s);
```

答："AAA"是字符串常量。s 是指针，指向这个字符串常量，所以在定义 s 的时候就有问题，应该改为：

```
const char* s="AAA";
```

因为是常量，所以对 s[0]的赋值操作是不合法的。

【面试题 3-21】以下程序的输出结果是（　　）。

```
const int i=0;
int *j=(int *)&i;
*j=1;
printf("%d,%d",i,*j);
```

A．0,1　　B．1,1　　C．1,0　　D．0,0

答：i 为整型常量，和变量一样有一个存储空间（如地址是 0x12ff44），j 为非常量整型指针，在执行 int *j=(int *) &i 时为指针变量 j 分配空间（如地址是 0x12ff40），将 i 的地址转换为整数（如 0x12ff44）存放在 j 中，此时*j=i=0；但执行*j=1 时并不是修改 i 的值（其为常量不能修改），而是为 j 分配另外一个指向的整数空间并存放整数 1，即*j=1，i 值仍然为 0，所以答案为 A。

【面试题 3-22】简述以下定义的区别：

```
char *const p;
char const *p;
const char *p;
```

答：在 char * const p 中 const 修饰变量 p，p 指针值不能修改，指针 p 指向的内容可以修改。在 const char *p 中 const 修饰*，表示指针 p 指向的内容不能修改，但 p 指针值可以修改。char const *p 和 const char *p 相同。

【面试题 3-23】若有定义 int b，以下哪两个是等同的？

A．const int* a = &b;

B．int * const a = &b;

C．const int * const a = &b;

D．int const * const a = &b;

答：选项 A 定义常量指针 b（const 修饰*）。选项 B 定义常量指针变量 b（const 修饰变量名）。选项 C 和 D 都是定义指针常量 b（两个 const 分别修饰*和变量名），所以 C 和 D 等同。

【面试题 3-24】下面 3 行程序代码的效果一样吗？

```
int b;
(1) const int *a=&b;
(2) int const *a=&b;
(3) int * const a=&b;
```

A．（2）=（3）　　B．（1）=（3）　　C．（1）=（2）

D．都不一样　　E．都一样

答：（1）和（2）中 const 都是修饰*，即指向的内容不能修改。答案为 C。

【面试题 3-25】给出以下定义，下列（　　）操作是合法的。

```
char a[]="hello";
char b[]="world";
const char *p1=a;
char* const p2=b;
```

A．p1++;　　B．p1[2]='w';　　C．p2[2]='l';　　D．p2++;

答：p1 是常量指针（const 修饰*，不能通过 p1 修改指向的内容，但 p1 的值可以改变），所以选项 A 正确，选项 B 错误。p2 为常量指针变量（const 修饰变量名，p2 值不能修改，但 p2 指向的内容可以修改），所以选项 C 正确，选项 D 错误。答案为 A、C。

【面试题 3-26】有以下定义：

```
int a=248,b=4;
int const c=21;
const int *d=&a;
int *const e=&b;
int const *f=&a;
```

问下列表达式中（　　）是有问题的。

A．d=&b;　　B．*d=43;　　C．*e=34;　　D．e=&a;

E．f=0x321f;

答：在 const int *d=&a 和 int const *f=&a 中，const 修饰*，定义的 d、f 指针值可以修改，而指针 d、f 指向的内容不能修改。在 int * const e=&b 中，const 修饰变量名，e 指针值不能修改，指针 e 指向的内容可以修改。

选项 B 修改了指针 d 指向的数据；选项 D 修改了指针 e 的值；选项 E 修改了指针 f 指向的数据，所以选项 B、D 和 E 是错误的。

3.3 多 级 指 针

3.3.1 要点归纳

⋙ 1. 多级指针定义

在 C 中除了允许指针指向普通数据之外，还允许指针指向另外的指针，这种指向指针的指针称为多级指针。

当一个指针指向普通数据时称为一级指针，指向一级指针的指针称为二级指针，指向二级指针的指针称为三级指针，以此类推。例如有以下定义：

```
int *p,**pp,***ppp;
```

其中，p 为一级指针，pp 为二级指针，ppp 为三级指针。

2. 多级指针的运算

多级指针运算与一级指针运算类似。需要注意的是，在访问一个指针的目标时只有一级指针的目标才是要处理的数据，多级指针的目标仍是一个指针。例如：

```
int x=10;
int *p=&x;
int **pp=&p;
int ***ppp=&pp;
```

其示意图如图 3.4 所示。如同一级指针常用来操作一维数组元素，多级指针也常与多维数组一起使用，将在下一章讨论。

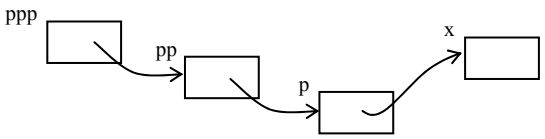


图 3.4　多级指针的示意图

例如，分析以下程序的输出结果。

```
#include <stdio.h>
#include <malloc.h>
void main()
{   int *p,**pp;
    p=(int *)malloc(3*sizeof(int));
    *p=1; *(p+1)=2; *(p+2)=3;
    pp=&p;
    printf("%d,",**pp);
    printf("%d\n",*(*pp+1));
}
```

在上述程序中，p 为一级指针，为它分配了 3 个整数的空间，分别存放整数 1、2、3。pp 为二级指针，其值为 p 的地址。在一次执行时各个变量的地址如图 3.5 所示（每次执行可能地址不同）。

相关表达式的含义如下。

- p：为 p 所指元素的地址，即为数据 1 的地址 562eb8。
- p+1：为 p 所指元素的下一个元素的地址，即 p+sizeof(int)为数据 2 的地址 562ebc，其中 sizeof(int)为一级指针 pp 的步长。

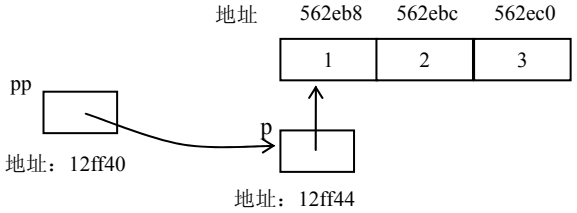


图 3.5　在一次执行时各个变量的地址

- pp：为 pp 所指变量 p 的地址，即 12ff44。
- pp+1：为 pp 所指变量 p 的下一个地址，即 pp+sizeof(p)=12ff48，其中 sizeof(p)为二级指针 pp 的步长，这里超界。
- *pp：为 pp 所指变量 p 的值，即*pp=p，为 p 的值 12ff44。
- *pp+1：pp 所指变量 p 的值加 1，*pp+1=p+1，为数据 2 的地址 562ebc。
- **pp：**pp=*(*pp)=*p=1。
- *(*pp+1)：*(*pp+1)=*(p+1)=2。

所以上述程序的输出为“1,2”。

对于二级指针 pp，**pp 运算是正确的，表示解两次引用，其结果是元素值。但取地址运算符&不能连续多次使用，如有定义 int n=10，而 pp=&&n 或者 pp=&(&n))都是错误的。

3. void *和 void **的含义

void 的字面意思是“无类型”，void *则为“无类型指针”，void *可以指向任何类型的数据。

“void a;”语句编译时会出错，提示“illegal use of type 'void'（非法使用 void 类型）”不过，即使 void a 的编译不会出错，它也没有任何实际意义。void 真正发挥的作用在于对函数返回值的限定和对函数参数的限定。

void *的一个重要应用是在强制类型转换中把赋值运算符右边指针的类型转换为左边指针的类型。例如：

```
double *p1;
int *p2;
p1=p2;
```

其中“p1 = p2;”语句会编译出错，提示“'=' : cannot convert from 'int *' to 'float *'”，必须改为：

```
p1=(double *)p2;
```

而 void *不同，任何类型的指针都可以直接赋值给它，无须进行强制类型转换：

```
void *p1;
int *p2;
p1=p2;                    //正确：可以将任何有类型地址赋给无类型指针变量
```

这并不意味着 void *也可以无须强制类型转换地赋给其他类型的指针。因为“无类型”可以包容“有类型”，而“有类型”不能包容“无类型”。下面的语句会出现“'=' : cannot convert from 'void *' to 'int *'”的编译错误：

```
void *p1;
int *p2;
p2=p1;                    //错误：不能将任何无类型地址赋给有类型指针变量
```

但改为强制类型转换就正确了：

```
p2=(int *)p1;             //正确：将无类型地址强制转换为有类型地址
```

这就是 malloc 等库函数原型的返回值为 void *的原因。

那么 void **又是什么呢？例如*((void **)&f)，其中(void **)可以被看成(void * *)，首先做前面的(void *)，即(void *)&f，将变量 f 的地址转换为一个无类型的地址，再做后面的*转换，根据&f 的基类型将这个无类型的地址转换为基类型地址。也就是说，(void **)&f 相当于&f。最后再做最外面的*运算，它不同于(void **)中的*（仅作为说明符，就像 int *p 定义中的*），而是取值运算符，所以*((void **)&f)相当于*&f，即 f。

例如：

```
int n=10;                 //n 的地址为 12ff44
int *p=&n;                //p 的值为 12ff44
printf("%d\n",*(void *)p); //错误：不能对无类型地址取值
```

结果出现编译错误，这里将指向整型变量 n 的地址转换为 void *地址后再取它的值是错误的，也就是说，实际上是取无类型地址的值。改为以下语句就正确了：

```
printf("%d\n",*(void **)p); //正确：(void **)p 相当于(int *)(void *)p
```

因为*(void **)p 相当于*p，所以输出 p 指向变量 n 的值，即 10。

3.3.2　面试真题解析

【面试题 3-27】以下程序的输出结果是（　　）。

```
#include <stdio.h>
void main()
{   int **k,*j,i=100;
```

```
    j=&i;
    k=&j;
    printf("%d\n",**k);
}
```

A．运行错误　　B．100　　C．i 的地址　　D．j 的地址

答：j=&i，j 的值就是 i 的地址，即*j=i=100。执行 k=&j 将 j 的地址赋给 k，即*k=j，所以**k=*j=i=100。答案为 B。

【面试题 3-28】有以下程序段：

```
void main()
{   int a=5,*b,**c;
    c=&b;b=&a;
}
```

程序在执行“c=&b;b=&a;”语句后表达式**c 的值是（　　）。

A．变量 a 的地址　　B．变量 b 中的值

C．变量 a 中的值　　D．变量 b 的地址

答：c 为二级指针，在执行“c=&b,b=&a;”后**c=*b=a=5。答案为 C。

【面试题 3-29】(void *)ptr 和*(void **)ptr 的结果是否相同？其中 ptr 为同一个指针。

答：不相同。(void *)ptr 返回的是将 ptr 转换成的无类型的地址。在*(void **)ptr 中，(void **)ptr 相当于 ptr，所以该表达式返回 ptr 指向变量的值。

【面试题 3-30】在 32 位编译器下，sizeof(void)的值是（　　）。

A．0　　B．4　　C．这取决于主机的字的大小　　D．8

E．编译错误或者为 1

答：void 表示“无类型”，在 VC++中会出现“illegal sizeof operand（非法 sizeof 运算量）”的编译错误。在 GCC 中为 1。答案为 E。

3.4 自测题和参考答案

3.4.1 自测题

3-1　若有定义“int *p1,*p2,m=5,n;”，以下均是正确赋值语句的选项是（　　）。

A．p1=&m;p2=&p1;　　B．p1=&m;p2=&n;*p1=*p2;

C．p1=&m;*p1=*p2;　　D．p1=&m;*p2=*p1;

3-2　以下程序段完全正确的是（　　）。

A．int *p; scanf("%d",&p);

B．int *p; scanf("%d",p);

C．int k,*p=&k; scanf("%d",p);

D．int k,*p; *p=&k;　scanf("%d",p);

3-3　以下程序中 p 和"hello,world"分别存储在内存中的哪个区域？

```
int main()
{   char *p="hello,world";
    return 0;
}
```

A．栈、堆　　B．栈、栈

C．堆、只读存储区（静态数据区）　　D．栈、只读存储区（静态数据区）

3-4　执行以下程序有什么问题？

```
#include <stdio.h>
#include <malloc.h>
void main()
{   int i,*p;
    p=(int *)malloc(3*sizeof(int));
    for(i=0;i<3;i++)
        *(p+i)=i+1;
    p++;
    free(p);
}
```

3-5　以下程序执行后的输出结果是（　　）。

```
#include <stdio.h>
void main()
{   int a=1;
    int b=3;
    int c=5;
    int *p1=&a;
    int *p2=&b;
    int *p=&c;
    *p=*p1*(*p2);
    printf("%d\n",c);
}
```

A．1　　B．2　　C．3　　D．4

3-6　以下程序有错，错误的原因是（　　）。

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

```
void main()
{   int *p,i;
    char *q,ch;
    p=&i;
    q=&ch;
    p=40;
    *p=*q;
}
```

A．p 和 q 的类型不一致，不能执行*p=*q

B．*p 中存放的是地址值，因此不能执行 p=40

C．q 没有指向具体的存储单元，所以*q 没有实际意义

D．以上都不对

3-7　给出以下程序的执行结果。

```
#include <stdio.h>
void main()
{   int k=2,m=4,n=6;
    int *pk=&k,*pm=&m,*p;
    *(p=&n)=*pk*(*pm);
    printf("%d\n",n);
}
```

3-8　给出以下程序的执行结果。

```
#include <stdio.h>
void main()
{   char a[]="Language",b[]="programe";
    char *p1,*p2;
    int k;
    p1=a;p2=b;
    for(k=0;k<=7;k++)
        if(*(p1+k)==*(p2+k))
            printf("%c",*(p1+k));
    printf("\n");
}
```

3-9　对于下面的代码，说法正确的是（　　）。

```
char *s1="Hello world";
char s2[]="Hello world";
```

```
s1[2]='E';      // 1
s2[2]='E';      // 2
*(s1+2)='E';    // 3
*(s2+2)='E';    // 4
```

A．语句 2、4 是非法的　　B．语句 3、4 是非法的
C．语句 1、3 是非法的　　D．仅语句 1 是非法的
E．仅语句 2 是非法的　　F．语句 1～4 都是合法的

3-10　以下程序的编译和执行结果是（　　）。

```
void main()
{   int i=11;
    int const *p=&i;
    p++;
    printf("%d",*p);
}
```

A．11　　B．12　　C．Garbage value（垃圾值）
D．Compile error（编译错误）　　E．None of above（以上都不对）

3-11　执行以下程序有什么问题？

```
#include <stdio.h>
#include <malloc.h>
void main()
{   int i,*p,**pp=&p;
    p=(int *)malloc(3*sizeof(int));
    for(i=0;i<3;i++)
        *(p+i)=i+1;
    printf("%d\n",**(++pp));
    free(p);
}
```

3-12　给出以下程序的输出结果。

```
#include <stdio.h>
#include <malloc.h>
void main()
{   int i,*p,*q,**pp;
    p=(int *)malloc(3*sizeof(int));
    for(i=0;i<3;i++)
```

```
        *(p+i)=i+1;
    q=(int *)malloc(4*sizeof(int));
    for(i=0;i<3;i++)
        *(q+i)=i+5;
    pp=(int **)malloc(2*sizeof(int *));
    *pp++=p;
    *pp=q;
    printf("%d,%d\n",*(*(pp+1)+1),**(--pp));
    free(p);
    free(q);
    free(pp);
}
```

3.4.2 参考答案

3-1 **答**：在选项 A 中，p2=&p1 错误，因为&p1 是二级地址。在选项 C 中，*p1=*p2 错误，因为*p2 是垃圾值。在选项 D 中，*p2=*p1 错误，因为 p2 没有分配指向的空间，为野指针。答案为 B。

3-2 **答**：scanf 函数的输入项需要给出输入变量的地址。选项 A 用于输入一个地址值，错误。选项 B 没有分配指针 p 指向的空间，错误。选项 D 没有为指针 p 分配指向的空间，而将变量 k 的地址放在指向的空间中，错误。答案为 C。

3-3 **答**：p 指针变量是在栈空间分配的，而它指向的常量字符串"hello,world"是在静态数据区分配的。答案为 D。

3-4 **答**：p 为一级指针，指向整数 1、2、3。首先 p 指向整数 1，p++让 p 指向整数 2。当执行 free(p)时，由于 free 释放的不是分配的起始地址，导致程序崩溃。

3-5 **答**： *p=*p1*(*p2)=1*3=3，p 指向 c 变量，即 c=3。答案为 C。

3-6 **答**：不能将常量作为地址值。答案为 B。

3-7 **答**：pk 指向 2，pm 指向 4。对于*(p=&n)=*pk*(*pm)，先执行*pk*(*pm)=2*4=8，再执行 p=&n，使 p 指向 n，即*p=8，间接使 n=8。程序输出为 8。

3-8 **答**：程序的功能是同步顺序扫描字符数组 a 和 b，输出对应位置相同的字符。程序的输出为 gae。

3-9 **答**：指针 s1 指向常量字符串，不能通过 s1 修改它，所以 s1[2]='E'和*(s1+2)='E'错误。答案为 C。

3-10 **答**：在 int const *p=&i 中，const 修饰*，不能修改 p 指向的内容，但 p 可以修改，而执行 p++使 p 指向一个垃圾值。答案为 C。

3-11　**答**：一级指针 p 指向 3 个整数，二级指针 pp 指向 p。对于表达式**(++pp)，先执行++pp 让 pp 指向 p 的下一个地址并返回它，该地址不是有效数据的地址，在执行**pp 时出现取垃圾地址内容的情况，导致程序崩溃。

3-12　**答**：p 为一级指针，指向整数 1、2、3。q 为一级指针，指向整数 5、6、7。pp 为二级指针，其中 pp 指向 p，pp+1 指向 q。当执行*pp=q 时，pp 指向 q。在 printf 语句中先执行表达式**(--pp) ⇨ pp 自减 1 指向 p，返回**pp，即 p[0]=1；再执行表达式*(*(pp+1)+1)，相当于*(q+1) ⇨返回 q[1]，即 6。程序的输出为“6,1”。