西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

指针的运算规则

1、指针的运算规则

2、指针的限定

7.2.5 指针运算

- 2．指针的关系运算
- 设p1、p2是同一个指向类型的两个指针（常量或变量），则p2和p1可以进行关系运算，用于比较这两个地址的位置关系。
- 例如：

```
int x, y, *p1=&x, *p2=&y;
p2>p1   //如果p2的地址值大于p1的地址值，则表达式为“真”，否则为“假”
p2!=p1  //如果p2的地址值不等于p1的地址值，则表达式为“真”，否则为“假”
p2==NULL   //如果p2为0值，则表达式为“真”，否则为“假”
```

7.2.5 指针运算

- 关系运算对指向不同类型的指针之间是没有意义的。但是，一个指针可以和空指针作相等或不等的关系运算，用来判断该指针是否为0值，以确定是否可以间接引用该指针。
- 例如，通常使用下面的代码避免无效的指针引用。

```
if (p!=NULL) {
   ...... //这里面引用*p
}
```

7.2.5 指针运算

- 3．指针的类型转换
- 设p是一个指针（常量、变量或表达式），可以对p进行显式类型转换，一般形式为：

```
(转换类型*)p
```

- 对指针进行显式类型转换的结果是产生一个临时指针对象，其指向类型为“转换类型”，地址值与p的地址值相同，但p的指向类型和地址值都不变。

7.2.5 指针运算

【例7.7】

输出一个short型数据的高、低字节。

7.2.5 指针运算

例7.7

```
 1 #include <stdio.h>
 2 int main()
 3 {
 4   short x, *p=&x;
 5   unsigned char hi, lo;
 6   scanf("%d",&x);
 7   lo=*( (unsigned char*)p ); //Intel CPU低字节存储在前
 8   hi=*( (unsigned char*)p + 1 ); //Intel CPU高字节存储在后
 9   printf("HI=%x,LO=%x\n",hi,lo);
10   return 0;
11 }
```

7.2.5 指针运算

例7.7

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

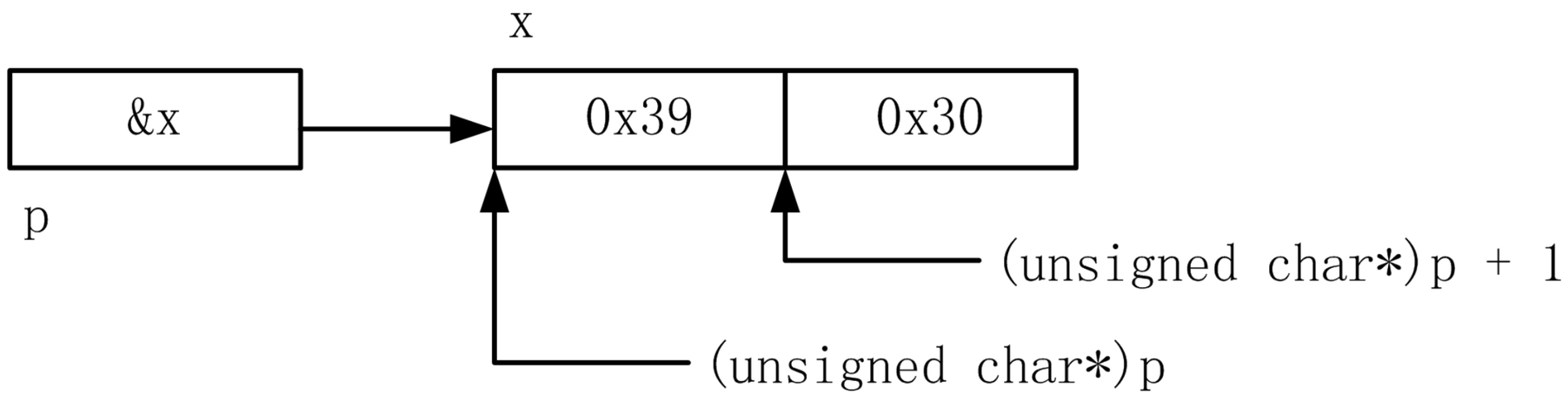
程序运行屏幕

```
HI=30,LO=39
```

```
12345↙
```

7.2.5 指针运算

- 指针变量p指向x（2个字节），为x输入12345后，x数据的十六进制形式为0x3039，其内存形式如图所示。



7.2.5 指针运算

- 4. 指针的赋值运算
- 指针可以进行赋值运算，前提是赋值运算符两边的操作数必须是相同指向类型。例如：

```
int x=10, *p1=&x, *p2;
p2=p1; // p1和p2均是int *
p2=&x; // &x和p2均是int *
```

7.2.5 指针运算

- 还可以进行如下的复合赋值运算：

```
int i=1, *p1=&i;
p1+=i; // p1是指针变量，i是整型（常量、变量或表达式）
p1-=i; // p1是指针变量，i是整型（常量、变量或表达式）
```

7.2.5 指针运算

- 如果操作数不是相同的指向类型则不能进行指针赋值，这时可以先进行显式类型转换再赋值。例如：

```
int a=10, *p;
double b=20, *pf=&b;
p=(int *)pf; // (int *)pf和p均是int*
```

7.2.5 指针运算

- 指针显式类型转换后，并没有改变指向对象的类型。例如这里的b变量仍然是double型，尽管指针变量p2指向了它，但*p2的间接引用得到的并不是b转换成整型的结果，而是b存储在内存中的低4个字节的整数值（b本身为8个字节）。当b是20时，内存中的数据为0x4034000000000000（浮点数格式），因此*p2的结果是0（取低4个字节作为整型）。比较一下double型显式类型转换为int型。

```
a=(int)b; //a=20
```

```
int a=10, *p;
double b=20, *pf=&b;
p=(int *)pf; // (int *)pf和p均是int*
```

7.2.5 指针运算

- 5．void*指针的运算特殊性
- void*指针不能做指针算术运算，例如：

```
void *pv1;
pv1++; //错误，pv1指向类型不明确
```

- 原因是void*指针指向对象的类型不明确，因而也就无法确定指针运算后的指向。

7.2.5 指针运算

- void*指针可以做关系运算，表示两个指针的地址值比较。void*指针可以指向其他任何类型，无需类型转换。假定指针是有效的，可以将void*指针显式类型转换为其他类型，再使用间接引用。例如：

```
int x=10;
double y=20, *pf=&y;
void *pv1;
pv1=&x; //无需指针类型转换
printf("%d\n", *((int*)pv1)); //void指针显示类型转换后再引用
pv1=pf; //无需指针类型转换
printf("%lf\n",*((double*)pv1));//void指针显示类型转换后再引用
```

7.2.6 指针的const限定

- const限定符作用在指针类型上有两种含义：指向const对象的指针和const指针。

7.2.6 指针的const限定

- 1．指向const对象的指针
- 一个指针变量可以指向只读型对象，称为指向const对象的指针，定义形式为：

```
const 指向类型 *指针变量, ......;
```

- 即在指针变量定义前加const限定符，其含义是指针指向的对象是只读的，换言之，不允许通过指针来改变所指向的const对象的值。

7.2.6 指针的const限定

- 例如：

```
const int *p;
```

- 这里的p是一个指向const的int类型对象的指针，const限定了p指针所指向的对象类型，而并非p本身。也就是说，p本身并不是只读的，在定义时不需要对它进行初始化。可以给p重新赋值，使其指向另一个const对象。但不能通过p修改其所指对象的值。

7.2.6 指针的const限定

- 例如：

```
const int a=10, b=20;
const int *p;
p=&a; //正确，p不是只读的
p=&b; //正确，p不是只读的
*p = 42; //错误，*p是只读的
```

7.2.6 指针的const限定

- 把一个const对象的地址赋给一个非const对象的指针变量是错误的，例如：

```
const double pi = 3.14;
double *ptr = &pi; //错误，ptr是非const指针变量
const double *cptr = &pi; //正确，cptr是const指针变量
```

7.2.6 指针的const限定

- 不能使用void*指针保存const对象的地址，而必须使用const void*指针保存const对象的地址，例如：

```
const int x = 42;
const void *cpv = &x; //正确，cpv是const指针变量
void *pv = &x; //错误，x是const
```

7.2.6 指针的const限定

- 允许把非const对象的地址赋给指向const对象的指针，例如：

```
const double pi = 3.14;
const double *cptrf = &pi; //正确，cptrf是const指针变量
double f = 3.14; //f是double型，f是非const
cptrf = &f; //正确，允许将f的地址赋给cptrf
f=1.618; //正确，可以修改f的值
*cptrf = 10.1; //错误，不允许通过引用cptrf修改f的值
```

7.2.6 指针的const限定

- 不能使用指向const对象的指针修改指向对象。例如：

```
double f, *ptr=&f;
const double *cptr=&f;
f = 3.14; //正确，f不是const，允许修改
*cptr = 3.14; //错误，cptr是const指针，不允许修改*cptr
*ptr = 2.72; //正确，ptr不是const指针，允许修改*ptr
```

程序中，指向const的指针cptr实际上指向了一个非const对象；尽管它所指的对象并非const，但仍不能使用cptr修改该对象的值。本质上来说，由于没有方法分辨cptr所指的对象是否为const，系统会把它所指的所有对象都视为const。

7.2.6 指针的const限定

- 如果指向const的指针所指的对象并非const，则可直接给该对象赋值或间接地利用非 const指针修改其值，毕竟这个值不是const的。重要的是要记住：不能保证指向const的指针所指对象的值一定不被其他方式修改。

7.2.6 指针的const限定

- 实际编程中，指向const的指针常用作函数的形参。将形参定义为指向const的指针，以此确保传递给函数的实参对象在函数中不被修改。

7.2.6 指针的const限定

- 2．const指针
- 一个指针变量可以是只读的，称为const指针，定义形式为：

```
指向类型 * const 指针变量, ......;
```

7.2.6 指针的const限定

- 例如：

```
int a=10, b=20;
int *const pc = &a; //pc是const指针
```

- 可以从右向左把上述定义语句理解为“pc是指向int型对象的const指针”。

7.2.6 指针的const限定

- 任何企图给const指针赋值的操作，即使给pc赋回同样的值都会导致编译错误：

```
pc = &b; //错误，pc是只读的
pc = pc; //错误，pc是只读的
pc++; //错误，pc是只读的
```

- 与任何const量一样，const指针必须在定义时初始化。

7.2.6 指针的const限定

- 指针本身是const的并没有说明是否能使用该指针修改它所指向对象的值。指针所指对象的值能否修改完全取决于该对象的类型。

- 例如pc指向一个非const的int型对象a，则可通过pc间接引用修改该对象的值：

```
*pc=100; //正确，a被修改
```

7.2.6 指针的const限定

- 3．指向const对象的const指针
- 可以定义指向const对象的const指针，形式为：

```
const 指向类型 * const 指针变量, ......;
```

7.2.6 指针的const限定

- 例如：

```
const double pi = 3.14159;
const double *const cpc=&pi; //cpc为指向const对象的const指针
```

- 可以从右向左理解上述定义：“cpc首先是一个const指针，指向double类型的const对象”。


程序设计