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

指针的有效性和风险控制

1、指针的有效性

2、指针的风险控制

7.2.4 指针的有效性

- 指针是特殊的数据，因此指针的运算和操作要注意有效性问题。

- 程序中的一个指针必然是以下三种状态之一：
- ①指向一个已知对象；
- ②0值；
- ③未初始化的、或未赋值的、或指向未知对象。

7.2.4 指针的有效性

- 无论指针作了什么运算和处理，只要操作后指针指向程序中某个确切的对象，即指向一个有确定存储空间的对象（称为已知对象），则该指针是有效的；如果对该指针使用间接引用运算，总能够得到这个已知对象。

- 指针理论上可以为任意的地址值，若一个指针不指向程序中任何已知对象，称其指向未知对象。未知对象的指针是无效的，无效的指针使用间接引用运算几乎总会导致崩溃性的异常错误。

7.2.4 指针的有效性

- （1）如果指针的值为0，称为0值指针，又称空指针（null pointer），空指针是无效的。例如：

```
int *p=0;
*p=2; //空指针间接引用将导致程序产生严重的异常错误
```

- 多数情况下，应该在指针间接引用之前检测它是否为空指针，从而避免异常错误。

7.2.4 指针的有效性

- （2）如果指针未经初始化，或者没有赋值，或者指针运算后指向未知对象，那么该指针是无效的。
- 一个指针还没有初始化，称为“野指针”（wild pointer）。严格的说，每个指针还没有初始化之前都是“野指针”，大多数的编译器都对此产生警告。例如：

```
int *p; //p是野指针
*p=2; //几乎总会导致程序产生严重的异常错误
```

7.2.4 指针的有效性

- 一个指针曾经指向一个已知对象，在对象的内存空间释放后，虽然该指针仍是原来的内存地址，但指针所指已是未知对象，称为“迷途指针”（dangling pointer）。

7.2.4 指针的有效性

```
1  char *p=NULL; //p是空指针，全局变量
2  void fun()
3  {
4    char c; //局部变量
5    p=&c;
6    //指向局部变量c，函数调用结束后，c的空间释放，p就成了迷途指针
7  }
8  void caller()
9  {
10   fun();
11   *p=2; //p现在是迷途指针，几乎总会导致程序产生严重的异常错误
12 }
```

7.2.4 指针的有效性

- 这两种情况比起空指针更难发现，因为程序无法检测这个非0值的指针p究竟是有效的还是无效的，也无法区分这个指针所指向的对象的地址是已知对象的还是未知对象的。

7.2.4 指针的有效性

- 示例

```
int a,b;
scanf("%d%d" ,a ,b); //错误，几乎总会导致程序产生严重的异常错误
```

- scanf函数的实参要求输入变量的地址（即&a和&b），以便它将输入数据按地址送到变量中，但第2行实际给出的是变量a和b的值，而a和b尚未初始化，于是实参就成了未初始化的指针，是无效指针。

7.2.4 指针的有效性

- 实际编程中，程序员要始终确保引用的指针是有效的，对尚未初始化或未赋值的指针一般先将其初始化为0值，引用指针之前检测它是否为0值。

7.2.5 指针运算

- 指针运算主要是给定范围内指针的算术运算、比较运算、类型转换等，由于指针数据的特殊性，因此需要特别注意指针运算的地址意义。

7.2.5 指针运算

- 1．指针的算术运算
- 指针的算术运算有指针加减整数运算，指针变量自增自减运算，两个指针相减运算。

7.2.5 指针运算

- （1）指针加减整数运算
- 设p是一个指针（常量或变量），n是一个整型（常量或变量），则p+n的结果是一个指针，指向p所指向对象的后面的第n个对象；而p-n的结果是一个指针，指向p所指向对象的前面的第n个对象。

7.2.5 指针运算

- 例如：

```
int x, n=3 , *p=&x;
p+1   //指向存储空间中变量x后面的第1个int型存储单元
p+n   //指向存储空间中变量x后面的第n(此时为3)个int型存储单元
p-1   //指向存储空间中变量x前面的第1个int型存储单元
p-n   //指向存储空间中变量x前面的第n(此时为3)个int型存储单元
```

- 特别地，p+0、p-0与p均指向同一个对象。

7.2.5 指针运算

【例7.5】

指针加减整数运算后的输出。

7.2.5 指针运算

例7.5

```
1 #include <stdio.h>
2 int main()
3 {
4   int x, n=3, *p=&x;
5   printf("p=%x, p+1=%x, ",p,p+1); //地址输出用十六进制形式
6   printf("p+n=%x, p-n=%x\n",p+n,p-n); //地址输出用十六进制形式
7   return 0;
8 }
```

7.2.5 指针运算

例7.5

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

程序运行屏幕

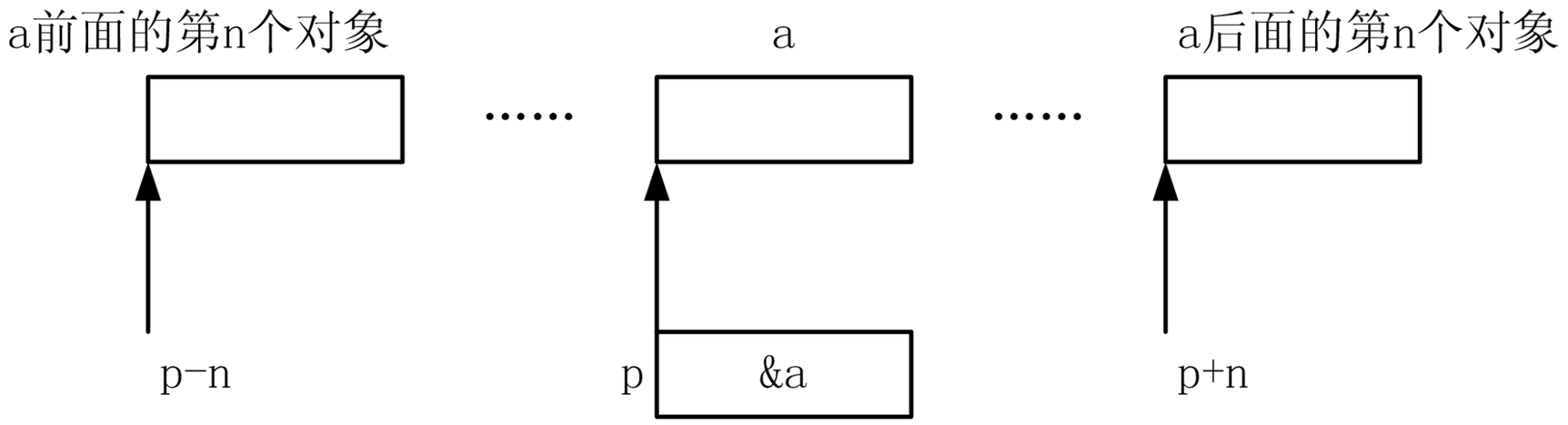
```
p=12ff7c, p+1=12ff80, p+n=12ff88, p-n=12ff70
```

7.2.5 指针运算

- 可以看出，p+1的地址值与p的地址值相差了4，p+n的地址值与p的地址值相差了12。即p+1不是按数学意义来计算的，而是按指针的地址意义来计算的。
- p+1就是p所指向的int型后面的那个int型对象的地址，由于int型对象在内存中占用4个字节，因此p+1的值与p相差4。
- 显然，p+1的值究竟是多少，与p所指向对象的类型有关。

7.2.5 指针运算

- 一般地，如果指针p所指向对象的类型为TYPE，那么p±n的值为：
- p的地址值 ± n*sizeof(TYPE)
- 如图所示。



7.2.5 指针运算

- （2）指针变量自增自减运算
- 设p是一个指针变量，其自增自减运算包括p++、++p、p--、--p形式。

7.2.5 指针运算

```
int x, *p=&x;
p++
//运算后表达式的值（临时指针对象）指向变量x，p指向变量x后面的第1个int型内存单元
++p
//运算后表达式的值（临时指针对象）和p均指向变量x后面的第1个int型内存单元
p--
//运算后表达式的值（临时指针对象）指向变量x，p指向变量x前面的第1个int型内存单元
--p
//运算后表达式的值（临时指针对象）和p均指向存储空间中变量x前面的第1个int型内存单元
```

7.2.5 指针运算

【例7.6】

指针变量自增自减运算后的输出。

7.2.5 指针运算

例7.6

```
 1 #include <stdio.h>
 2 int main()
 3 {
 4    int x, *p1, *p;
 5    p=&x , p1=p++;
 6    printf("p++: &x=%x, p=%x, p++=%x\n",&x,p,p1); //地址输出用十六进制形式
 7    p=&x , p1=++p;
 8    printf("++p: &x=%x, p=%x, ++p=%x\n",&x,p,p1); //地址输出用十六进制形式
 9    return 0;
10 }
```

7.2.5 指针运算

例7.6

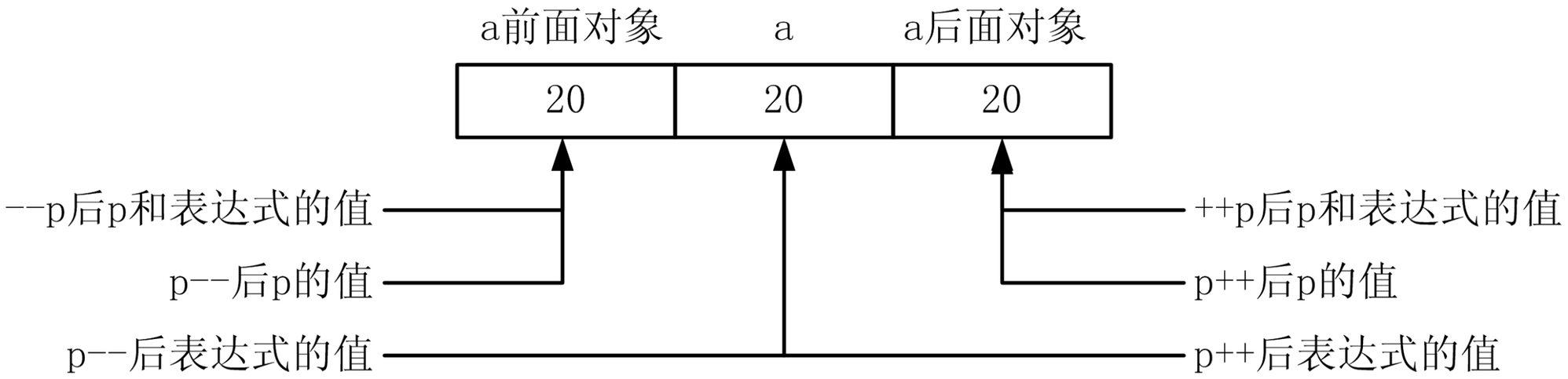
```
1 #include <stdio.h>
2 int main()
```

程序运行屏幕

```
p++: &x=12ff7c, p=12ff80, p++=12ff7c
++p: &x=12ff7c, p=12ff80, ++p=12ff80
```

7.2.5 指针运算

图7.9 指针变量自增自减运算示意



7.2.5 指针运算

- 另外，设有定义“int a＝100,*p=&a;”，需要注意以下形式的运算含义。
- ①(*p)++：等价于a++，运算执行后p值不变；
- ②*p++：按照运算符优先级，等价于*(p++)，运算后表达式的值为a，p指向下一个int型内存单元；
- ③*++p：按照运算符优先级，等价于*(++p)，p先指向下一个int型内存单元，表达式再引用这个内存单元的值。

7.2.5 指针运算

- （3）两个指针相减运算
- 设p1、p2是同一个指向类型的两个指针（常量或变量），则p2-p1的结果为两个指针之间对象的个数，如果p2的地址值大于p1结果为正，否则为负。

7.2.5 指针运算

- 指针算术运算后通常会引起地址的变化，实际编程中要考虑此时指针的有效性。例如：

```
int x, *p=&x;
p++; //迷途指针，指向未知对象
*p=100; //几乎总会导致程序产生严重的异常错误
```

- p原先指向x，是有效的；p++运算后p指向x的“下一个”，但这里“下一个”是未知对象，故自增运算后的p是无效的。

7.2.5 指针运算

- 指针算术运算经常用于数组、字符串或内存数据块，因为这些对象拥有连续的有效地址空间，只要在其存储空间范围内，运算后的指针都是有效的。


程序设计