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

使用指针

1、指针变量的初始化

2、指针变量的赋值

7.2.2 指针的间接访问

- 通过间接引用运算（*）可以访问指针所指向的对象或内存单元。

表7-2 间接引用运算符

运算符	功能	目	结合性	用法
*	间接引用	单目	自右向左	*expr

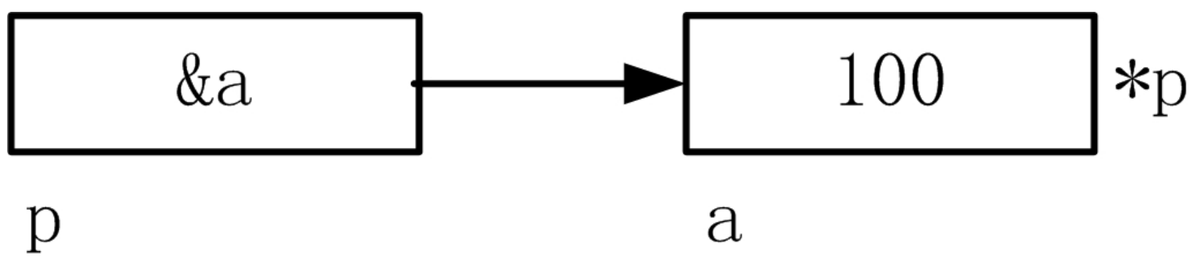
间接引用（又称解引用）运算符在所有运算符中优先级较高，其运算结果是一个左值，即expr所指向的对象或内存单元；expr必须是指针（地址）的含义，可以为地址常量、指针变量、指针运算表达式。

7.2.2 指针的间接访问

- 例如：

```
int a, *p=&a;
a=100; //直接访问a（对象直接访问）
*p=100; //*p就是a，间接访问a（指针间接访问）
*p=*p+1; //等价于a=a+1
```

- 当指针变量p指向整型变量a时，*p的运算结果就是变量a本身，而非a的值



7.2.2 指针的间接访问

【例7.2】

已知：

```
int a, b, *p1=&a, *p2;
```

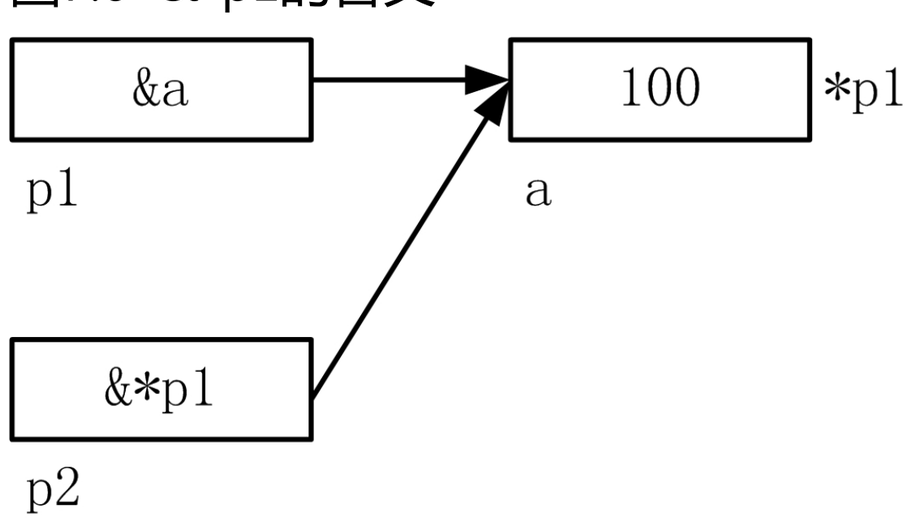
则①&*p1的含义是什么？②*&a的含义是什么？

7.2.2 指针的间接访问

例题分析

①由于“*”和“&”优先级相同，结合性自右向左，所以&*p1先计算*p1得到变量a，再计算&a得到变量a的地址，因此&*p1与&a相同。如图所示。

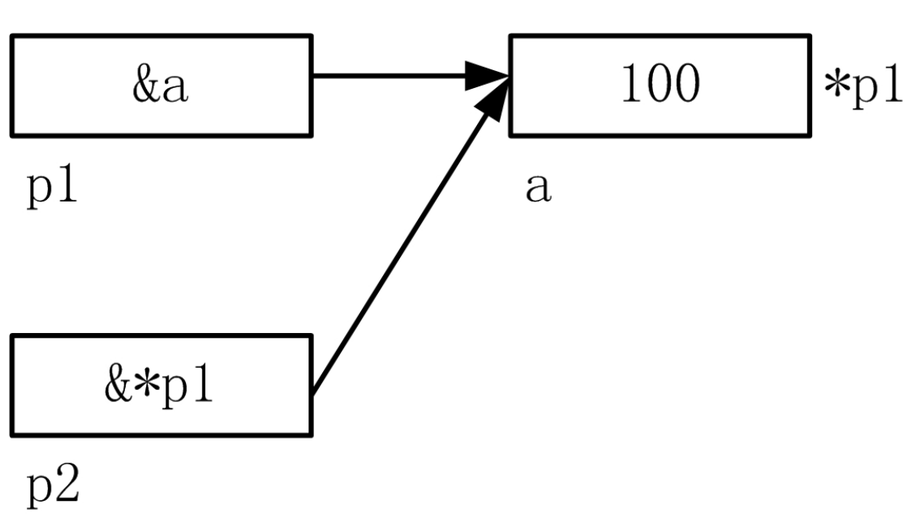
图7.6 &*p1的含义



7.2.2 指针的间接访问

例题分析

②*&a先计算&a得到变量a的地址，再计算“*(变量a的地址)”得到变量a，因此*&a与a等价。



7.2.2 指针的间接访问

【例7.3】

通过指针变量间接访问整型变量。

7.2.2 指针的间接访问

例7.3

```
1  #include <stdio.h>
2  int main()
3  {
4     int i=100, j=200;
5     int *p1, *p2;
6     p1=&i, p2=&j; //p1指向i，p2指向j
7     *p1 = *p1 + 1; //等价于i=i+1
8     p1=p2; //将p2的值赋值给p1，则p1指向j
9     *p1 = *p1 + 1; //等价于j=j+1
10    return 0;
11 }
```

（1）程序第6行的作用是将i和j变量的地址分别赋给p1和p2指针变量，则p1指向i和p2指向j。

7.2.2 指针的间接访问

例7.3

```
1  #include <stdio.h>
2  int main()
3  {
4    int i=100, j=200;
5    int *p1, *p2;
6    p1=&i, p2=&j; //p1指向i，p2指向j
7    *p1 = *p1 + 1; //等价于i=i+1
8    p1=p2; //将p2的值赋值给p1，则p1指向j
9    *p1 = *p1 + 1; //等价于j=j+1
10   return 0;
11 }
```

（2）程序第7行由于此时p1指向变量i，因此“*p1”的结果是i，所以“*p1=*p1+1”等价于“i=i+1”；第8行将p2的值赋给p1，则p1现在指向了变量j（原先指向i），第9行的“*p1=*p1+1”等价于“j=j+1”。

7.2.2 指针的间接访问

例7.3

```
1 #include <stdio.h>
2 int main()
3 {
4    int i=100, j=200;
5    int *p1, *p2;
6    p1=&i, p2=&j; //p1指向i，p2指向j
7    *p1 = *p1 + 1; //等价于i=i+1
8    p1=p2; //将p2的值赋值给p1，则p1指向j
9    *p1 = *p1 + 1; //等价于j=j+1
10   return 0;
11 }
```

（3）这段程序中的两处“*p1=*p1+1”，代码形式一样但实际作用不同，取决于在执行这个语句时p1具体指向了哪个变量。

7.2.2 指针的间接访问

- 从中可以看出，使用指针间接访问，优点是可以简化程序形式和写法，缺点是必须结合上下文分析才能判断*p1究竟是哪个变量。

7.2.2 指针的间接访问

【例7.4】

使用指针方式将两个数按先小后大的顺序输出。

7.2.2 指针的间接访问

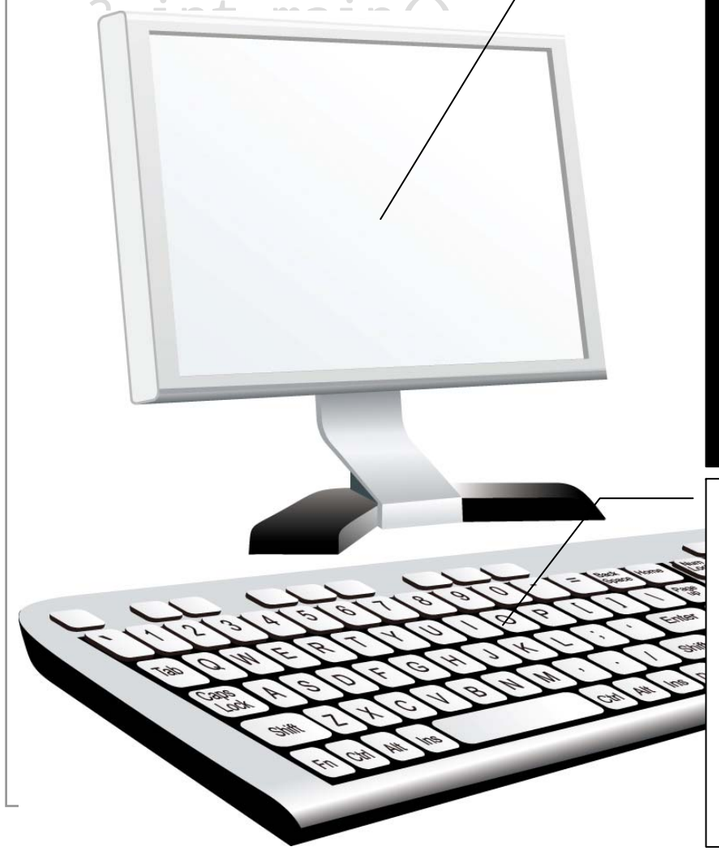
例7.4

```
1  #include <stdio.h>
2  int main()
3  {
4    int a,b,*p,*p1,*p2;
5    p1=&a; //p1指向a
6    p2=&b; //p2指向b
7    scanf("%d%d",p1,p2); //等价于 scanf("%d%d",&a,&b);
8    if(*p1>*p2) {
9      p=p1,p1=p2,p2=p; //是指针p1和p2的值相互交换
10   }
11   printf("a=%d,b=%d\n",a,b); //a和b未变
12   printf("min=%d,max=%d\n",*p1,*p2);
13   return 0;
14 }
```

7.2.2 指针的间接访问

例7.4

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



程序运行屏幕

```
a=34,b=12
min=12,max=34
```

```
34 12↙
```

7.2.2 指针的间接访问

例7.4

```
1  #include <stdio.h>
2  int main()
3  {
4    int a,b,*p,*p1,*p2;
5    p1=&a; //p1指向a
6    p2=&b; //p2指向b
7    scanf("%d%d",p1,p2); //等价于 scanf("%d%d",&a,&b);
8    if(*p1>*p2) {
9      p=p1,p1=p2,p2=p; //是指针p1和p2的值相互交换
10   }
11   printf("a=%d,b=%d\n",a,b); //a和b未变
12   printf("min=%d,max=%d\n",*p1,*p2);
13   return 0;
14 }
```

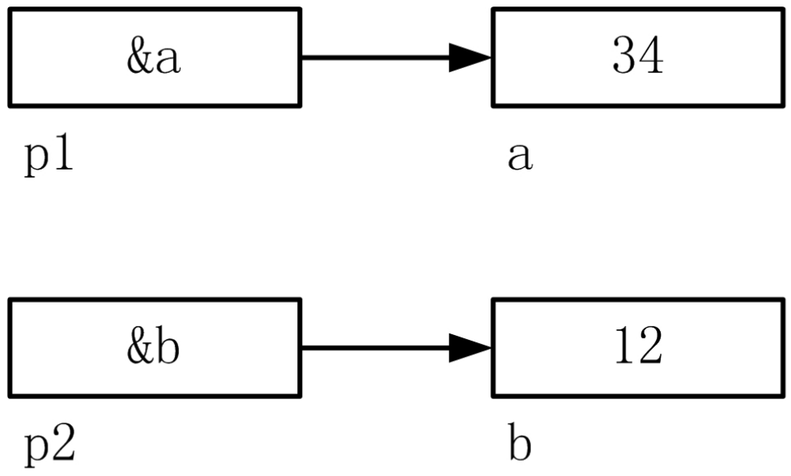
7.2.2 指针的间接访问

例7.4

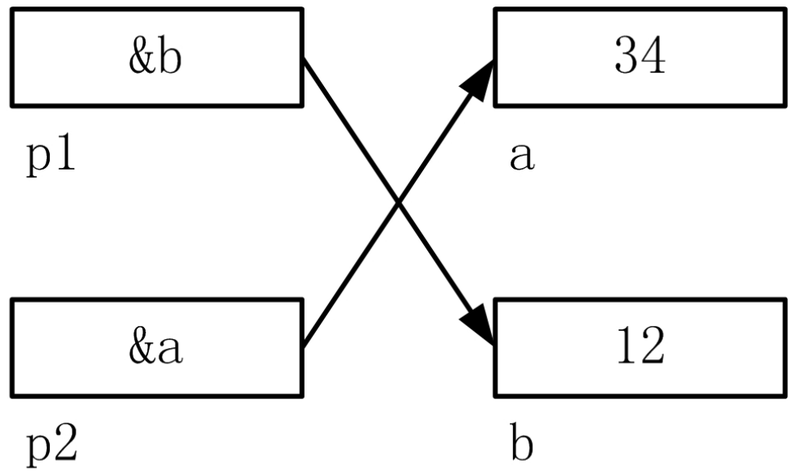
```
1  #include <stdio.h>
2  int main()
3  {
4    int a,b,*p,*p1,*p2;
5    p1=&a; //p1指向a
6    p2=&b; //p2指向b
7    scanf("%d%d",p1,p2); //等价于 scanf("%d%d",&a,&b);
8    if(*p1>*p2) {
9      p=p1,p1=p2,p2=p; //是指针p1和p2的值相互交换
10   }
11   printf("a=%d,b=%d\n",a,b); //a和b未变
12   printf("min=%d,max=%d\n",*p1,*p2);
13   return 0;
14 }
```

7.2.2 指针的间接访问

- 这个程序比较两个数后，没有去交换a和b的值，故变量a和b的内容在交换后仍保持不变，而是交换p1和p2的值。如图所示。



交换前



交换后

7.2.3 指针变量的初始化与赋值

- 可以在定义指针变量时对其初始化，一般形式为：

```
指针类型 *指针变量名=地址初值, ......
```

- 其中地址初值是指该值必须为地址含义。

7.2.3 指针变量的初始化与赋值

- 示例

```
int a;
int *p=&a; //p的初值为变量a的地址
int b, *p1=&b; //正确，p1初始化时变量b已有地址值
```

- 这里的“&”不是取地址运算符，而是取地址的记号。

7.2.3 指针变量的初始化与赋值

- 在定义一个对象时，编译器会自动为其分配内存单元，其存储位置（地址）也就固定下来不再变，即地址为常量值。因此a定义后，a为变量，但a的地址值&a是常量，可以出现在指针变量初值中。

7.2.3 指针变量的初始化与赋值

- 取变量地址一定发生在该变量定义之后（这时才有地址），否则是错误的。例如：

```
int *p2=&c, c; //错误，p2初始化时尚未有c的地址值
```

- 指针变量初始化时，地址初值必须是与指针变量同一指向类型的地址值，例如：

```
int a, *p1=&a; //正确
double f, *p2=&f; //正确
int *p3=&f; //错误，&f的指向类型是double
```

7.2.3 指针变量的初始化与赋值

- 指针变量可以进行赋值运算，例如

```
int a, *p1, *p2;
p1=&a; //将a的地址值赋给p1
p2=p1; //将p1的值赋给p2
```

7.2.3 指针变量的初始化与赋值

- 指针变量赋值时要求左值和右值必须是相同的指向类型，C语言不会对不同指向类型的指针作隐式类型转换。例如：

```
int a,*p1,*p2;
double f,*p3;
p1=&a; //正确
p2=p1; //正确
p3=&a; //错误，&a指向类型为int，p3指向类型为double
p3=&f; //正确
p3=p1; //错误，p1指向类型为int，p3指向类型为double
```

7.2.3 指针变量的初始化与赋值

- 对给指针变量赋值和通过指针进行赋值这两种操作的差别难于分辨。
- 区分的方法是：如果对左值进行间接引用，则修改的是指针所指对象；如果没有使用间接引用，则修改的是指针本身的值。例如：

```
int a, *p1;
p1=&a; //给指针变量赋值，则p1指向a，被赋值的是p1
*p1=100; //通过指针变量p1间接访问a，等价于a=100，被赋值的是a
```

7.2.3 指针变量的初始化与赋值

- 由于指针数据的特殊性，其初始化和赋值运算是有约束条件的，只能使用以下四种值：
- （1）0值常量表达式，即在编译时可获得0值的整型的const对象或常量0，例如：

```
int a, z=0;
const int nul=0;
int *p1=a; //错误，地址初值不能是变量
p1=z; //错误，整型变量不能作为指针，即使此整型变量的值为0
p1=4000; //错误，整型数据不能作为指针
p1=nul; //正确，指针允许0值常量表达式
p1=0; //正确，指针允许0值常量表达式
```

7.2.3 指针变量的初始化与赋值

- 除此之外，还可以使用C语言预定义的符号常量NULL，该常量在多个标准函数库头文件中都有定义，其值为0。如果在程序中使用了这个符号常量，则编译时会自动被数值0替换。因此，把指针初始化为NULL等价于初始化为0值，例如：

```
int *pi=NULL; //正确，等价于int *pi=0;
```

7.2.3 指针变量的初始化与赋值

- （2）相同指向类型的对象的地址。例如：

```
int a, *p1;
double f, *p3;
p1=&a; //正确
p3=&f; //正确
p1=&f; //错误，p1和&f指向类型不相同
```

7.2.3 指针变量的初始化与赋值

- （3）对象存储空间后面下一个有效地址，如数组下一个元素的地址。

- （4）相同指向类型的另一个有效指针。例如：

```
int x, *px=&x; //正确
int *py=px; //正确，相同指向类型的另一个指针
double *pz;
py=px; //正确，相同指向类型的另一个指针
pz=px; //错误，pz和px指向类型不相同
```


程序设计