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

探究指针的本质

1、指针的本质

2、指针变量的定义与引用

第7章 指针

- 计算机系统中，无论是存入或是取出数据都需要与内存单元打交道，物理器件通过地址编码寻找内存单元。

- 地址编码是一种数据，C语言的指针类型正是为了表示这种计算机所特有的地址数据。

- 存取内存单元是任何程序经常性的操作，前面按对象（或变量）名称直接访问内存单元。这里学习通过指针间接访问内存单元，这种近乎机器指令的操作方式大大提高了存取效率。

- 放弃简单直观的直接访问不用，而要用难于理解的指针间接访问，绝不仅仅是为了提高效率。由于两个函数的作用域不同，因而它们的局部变量互不可见，要想让一个函数能访问另一个函数里的变量，只能使用指针的间接访问。在数据和代码都要求封装到函数的结构化程序设计中，指针成为两个函数进行数据交换必不可少的工具。

- 程序运行时申请到的内存空间只有地址没有名称，因此指针成为访问动态内存的唯一工具，指针直接访问内存的形式简化了许多复杂数据结构的表示。

7.1 指针与指针变量

- 首先来理解数据对象（或变量）在内存中是如何存储的，又是如何读取的？

7.1.1 地址和指针的概念

- 程序中的数据对象总是存放在内存中，在生命期内这些对象占据一定的存储空间，有确定的存储位置。C语言将内存单元抽象为对象，就可以按名称来使用对象。

7.1.1 地址和指针的概念

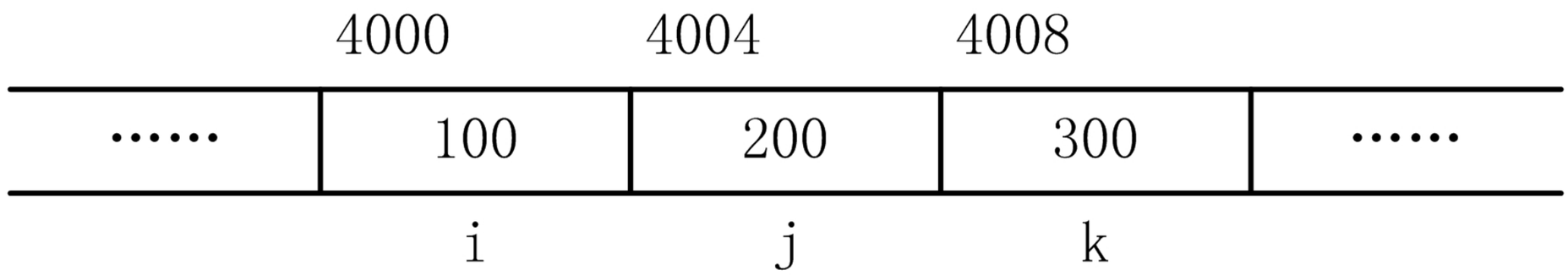
- 定义数据对象时，需要说明对象名称和数据类型。数据类型的作用是告诉编译器要为对象分配多大的存储空间（单位为字节），以及对象中要存储什么类型的值。对象名称的作用是对应分配到的内存单元，允许按名称来访问。

7.1.1 地址和指针的概念

- 例如：

```
int i, j, k; //定义整型变量
double f; //定义双精度浮点型变量
```

- 编译器会为变量i、j、k各分配4个字节（与计算机字长有关）的存储空间，为变量f分配8个字节的存储空间；那么在内存中，会有相应的内存单元对应这些变量。



7.1.1 地址和指针的概念

- 定义变量后，程序可以在变量中存储值和取出值。例如：

```
i=100; //按名称访问，即用i直接访问，存储i值
j=i+100; //按名称访问，即用j直接访问，取出i值，存储j值
```

- 数据值100存储到i对应的内存单元，表达式i+100的值存储到j对应的内存单元。

- 按对象名称存取对象的方式称为对象直接访问。

7.1.1 地址和指针的概念

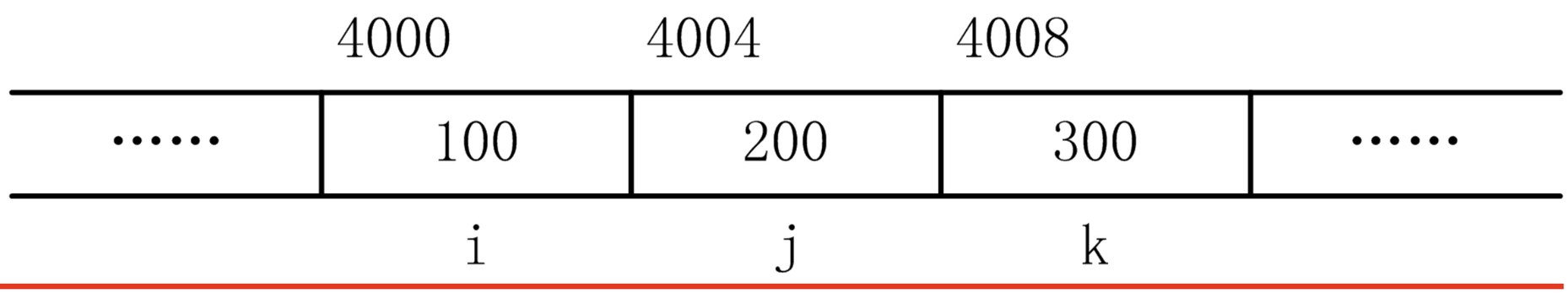
- 在容量可观的存储空间中，计算机硬件实际上是通过地址编码而非名称来寻找内存单元的。

- 地址编码通常按无符号整型数据处理（没有负数），每个内存单元都有一个地址，以字节为单位连续编码。

- 编译器将程序中的对象名转换成机器指令能识别的地址，通过地址来存取对象值。

7.1.1 地址和指针的概念

- 如图所示，变量i的地址为4000，则语句“i=100;”执行时将数据值100存储到地址为4000的内存单元中。变量k的地址为4008，变量j的地址为4004，则语句“k=i+j;”执行时从地址4000的内存单元取出i值，从地址4004的内存单元取出j值，将它们累加后再将结果值300存储到地址4008的内存单元（即变量k）中。



7.1.1 地址和指针的概念

- 内存单元的地址（如4000）和内存单元的内容（如100）尽管都是数据，但它们是两个不同的概念。

7.1.1 地址和指针的概念

- 由于通过地址能寻找到对象的内存单元，因此C语言形象地将地址称为“指针”，即一个对象的地址称为该对象的指针。

- 例如整型变量i的地址是4000，则4000就是整型变量i的指针

7.1.1 地址和指针的概念

- 需要注意，不能简单将指针和地址划等号，指针虽是地址，但它有关联的数据类型。例如内存地址有4000、4001、4002、4003等编码，而由于整型数据类型存储时需要4个字节，所以当整型变量i的指针为4000时，意味着从4000内存地址开始，连续4个字节全都是i的内存单元，其他变量的指针此时是不可能为这4个内存地址的。

7.1.1 地址和指针的概念

- 通过对象地址存取对象的方式称为指针间接访问。

7.1.2 指针变量

- C语言将专门用来存放对象地址（即指针）的变量称为指针变量，其数据类型为指针类型，定义形式如下：

```
指针类型 *指针变量名, ......
```

- 即在变量名前加一个星号（*）表示该变量为指针变量。

7.1.2 指针变量

- 示例

```
int *p1, *p2; //定义p1和p2为指针变量
int *p, k; //定义p为指针变量，k为整型变量
```

7.1.2 指针变量

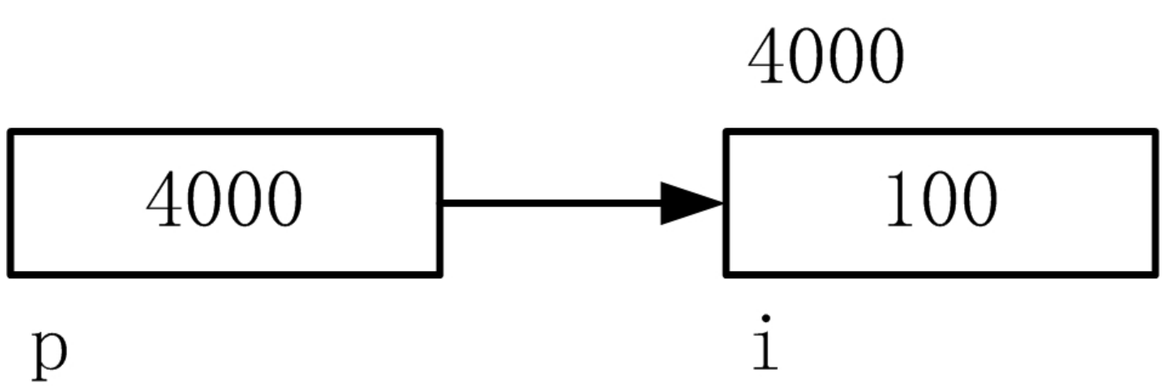
- 需要区分指针和指针变量这两个概念。指针是地址值，指针变量是存储指针的变量，例如可以说变量i的指针是4000，而不能说变量i的指针变量是4000；可以说指针变量p的值是4000，p既可以存储变量i的指针又可以存储变量j的指针。

- 通过指针变量，可以间接访问（或间接存取）对象。

7.1.2 指针变量

- 如图所示，p是指针变量，它存储整型变量i的地址4000，通过p知道i的地址，进而找到变量i的内存单元。

图7.2 通过指针变量间接访问



7.1.2 指针变量

- 每个指针变量都有一个与之关联的指向类型，它决定了指针所指向的对象的数据类型。例如：

```
int *p;
```

- 表示p是指向整型对象的指针变量，p只能用来指向整型对象，而不能指向其他数据类型对象。或者说，p只能存放整型对象的地址，不能存放其他数据类型对象的地址。

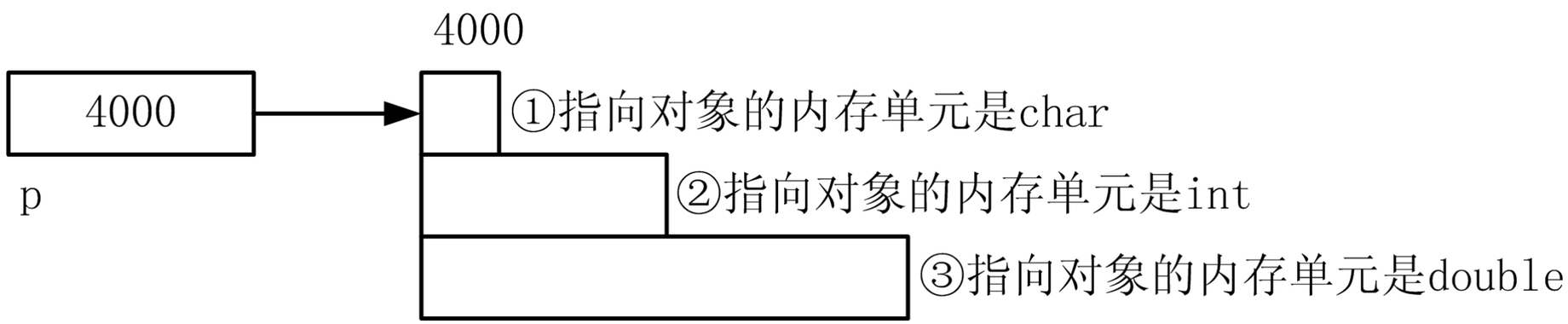
7.1.2 指针变量

- 指向类型可以是C语言任意有效的内置数据类型或自定义类型。由于指针变量的主要用途是通过它间接访问别的对象的内存单元，因此指向类型的说明是很重要的。

7.1.2 指针变量

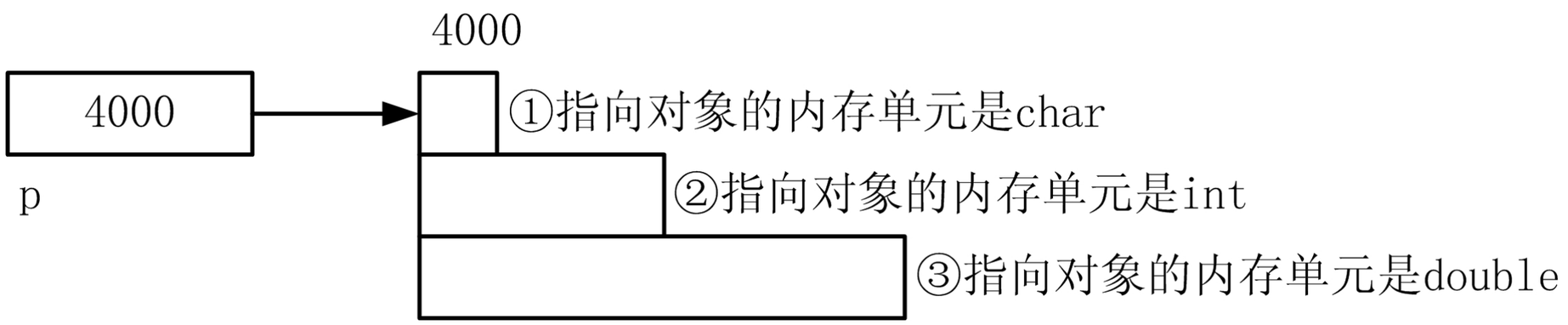
- 例如，假定指针变量p的值是4000，则下面三种写法的实际含义如图所示。

```
①char *p;
②int *p;
③double *p;
```



7.1.2 指针变量

- char *p表示p所指向的内存单元为字符型，该对象占用1个字节，int *p表示p所指向的内存单元为整型，该对象占用4个字节，double *p表示p所指向的内存单元为双精度浮点型，该对象占用8个字节。



7.1.2 指针变量

- 指向类型的不同表示指针变量所指向的对象的类型的不同，而非指针变量的不同。

- 所有指针变量的内存形式均是相同的。通常，编译器为指针变量分配4个字节的存储空间用来存放对象的地址，即指针变量的数据是地址的含义。由于地址值是大于等于零的，因此指针变量的数据习惯上按无符号整型数据对待。

7.1.2 指针变量

- C语言提供一种特殊的指针类型void*，它可以保存任何类型对象的地址，例如：

```
void *p; //纯指针
```

- 表明指针变量p与地址值相关，但不明确存储在此地址上的对象的类型，有时称这样的指针为“纯指针”。

7.1.2 指针变量

- void*指针变量仍然有自己的内存单元，但它的指向对象不明确。通常，void*指针只有几种有限的用途：
- ①与另一个指针进行比较；
- ②向函数传递void*指针或从函数返回void*指针；
- ③给另一个void*指针赋值。

- 需要注意，不允许使用void*指针操纵它所指向的对象，即不对void*指针作间接引用。

7.2 指针的使用及运算

- 指针的使用需要首先获取对象的地址，以此来间接引用对象，就如按对象名直接引用对象一般。
- 通常，当能够按对象名直接引用对象时，程序员理所当然会用直接访问方式，简单而又直观。

7.2 指针的使用及运算

- 只有在“直接引用”不成立时，例如在一个函数中想要引用另一个函数中的局部变量，这时才会（而且只能）按“间接引用”方式引用。换而言之，程序员只是在不得不使用指针的情形才使用指针。应用指针时切记这点。

- 取地址运算和间接引用运算是应用指针工具的基本运算。

7.2.1 获取对象的地址

- 通过取地址运算（&）获取对象的地址。

表7-1 取地址运算符

运算符	功能	目	结合性	用法
&	取地址	单目	自右向左	&expr

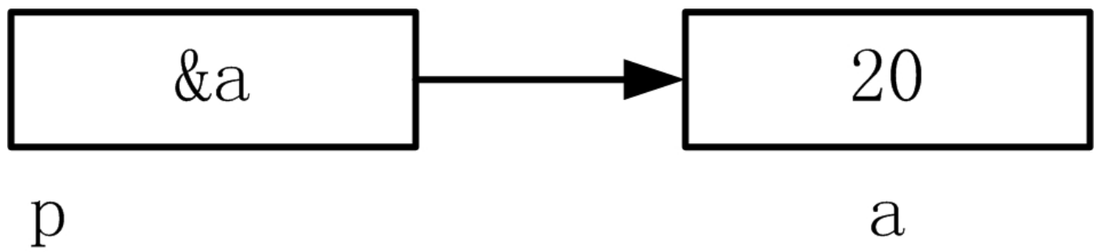
取地址运算符在所有运算符中优先级较高，其结果是得到对象的指针（或地址），expr必须是变量，即有内存单元的数据对象。

7.2.1 获取对象的地址

- 例如：

```
int a=20, *p; //定义指针变量
p = &a ; //指针变量p指向a
```

- 如图所示，指针变量p的值为整型变量a的地址，称指针变量p指向a。



7.2.1 获取对象的地址

- 取地址运算得到的指针不仅值为对象的地址，而且还以对象的数据类型作为指向类型，例如：

```
int a; // &a得到指向int型的指针
double f; // &f得到指向double型的指针
```

7.2.1 获取对象的地址

【例7.1】

输出整型变量的值和它的地址。

7.2.1 获取对象的地址

例7.1

```
1 #include <stdio.h>
2 int main()
3 {
4   int i=400;
5   printf("i=%d, &i=%x\n",i,&i); //输出i的值和i的地址值（指针）
6   return 0;
7 }
```

地址值一般按无符号整型输出，与unsigned int类似，习惯上用十六进制形式表示。

7.2.1 获取对象的地址

例7.1

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

程序运行屏幕

```
i=400, &i=12ff7c
```

7.2.1 获取对象的地址

- 值得注意的是，&i的值并不总是上面输出的结果。

- 对象确切的地址值取决于系统为程序分配的进程空间、编译器对变量分配的数目和顺序等多种因素，是一个复杂的存储分配机制产生的结果。

- 但是，在实际编程中，地址和指针的应用并不需要知道地址值的具体数据，因而我们不用关心这个复杂的存储分配机制的原理和过程。


程序设计