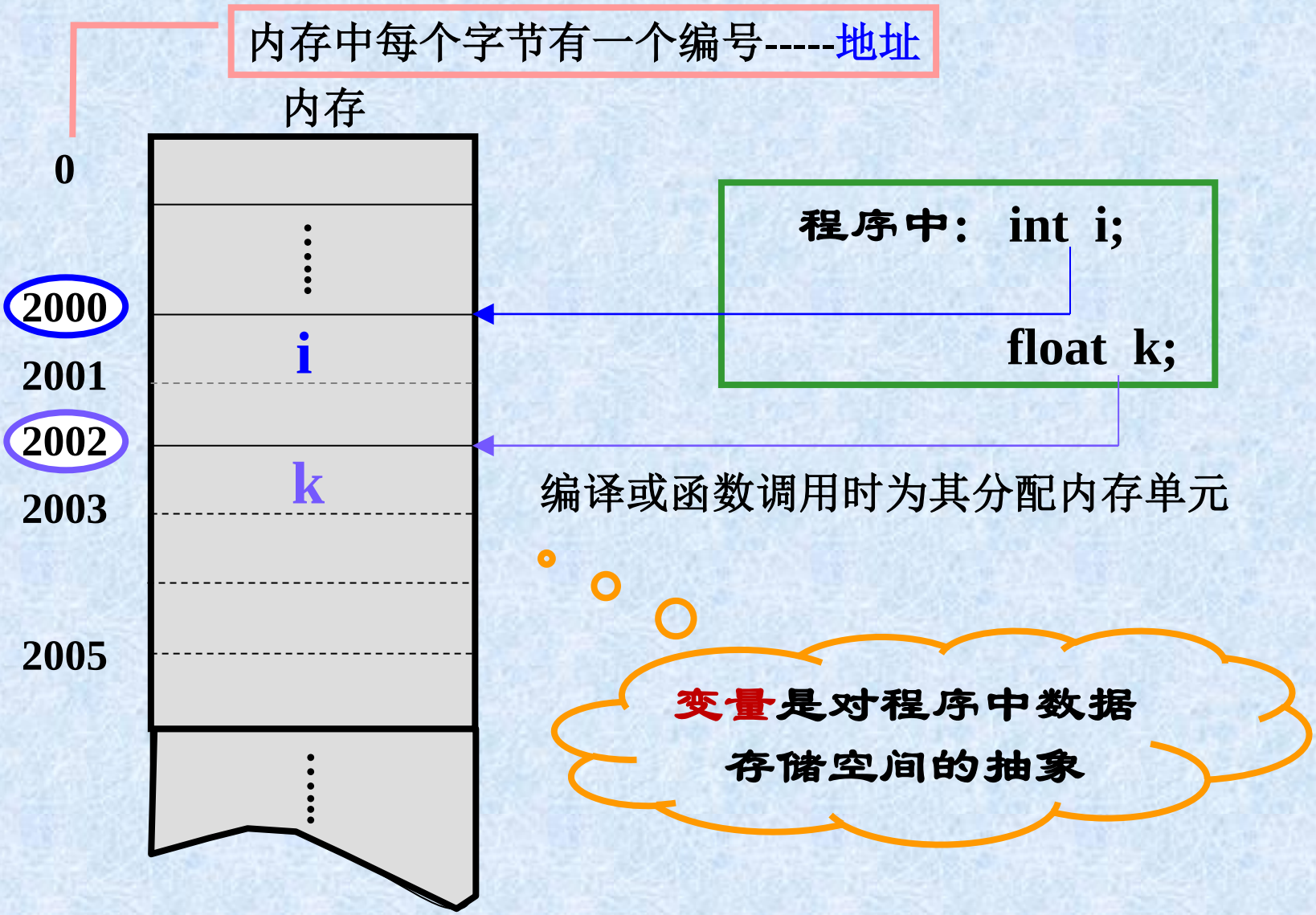
复习一： 指针的概念

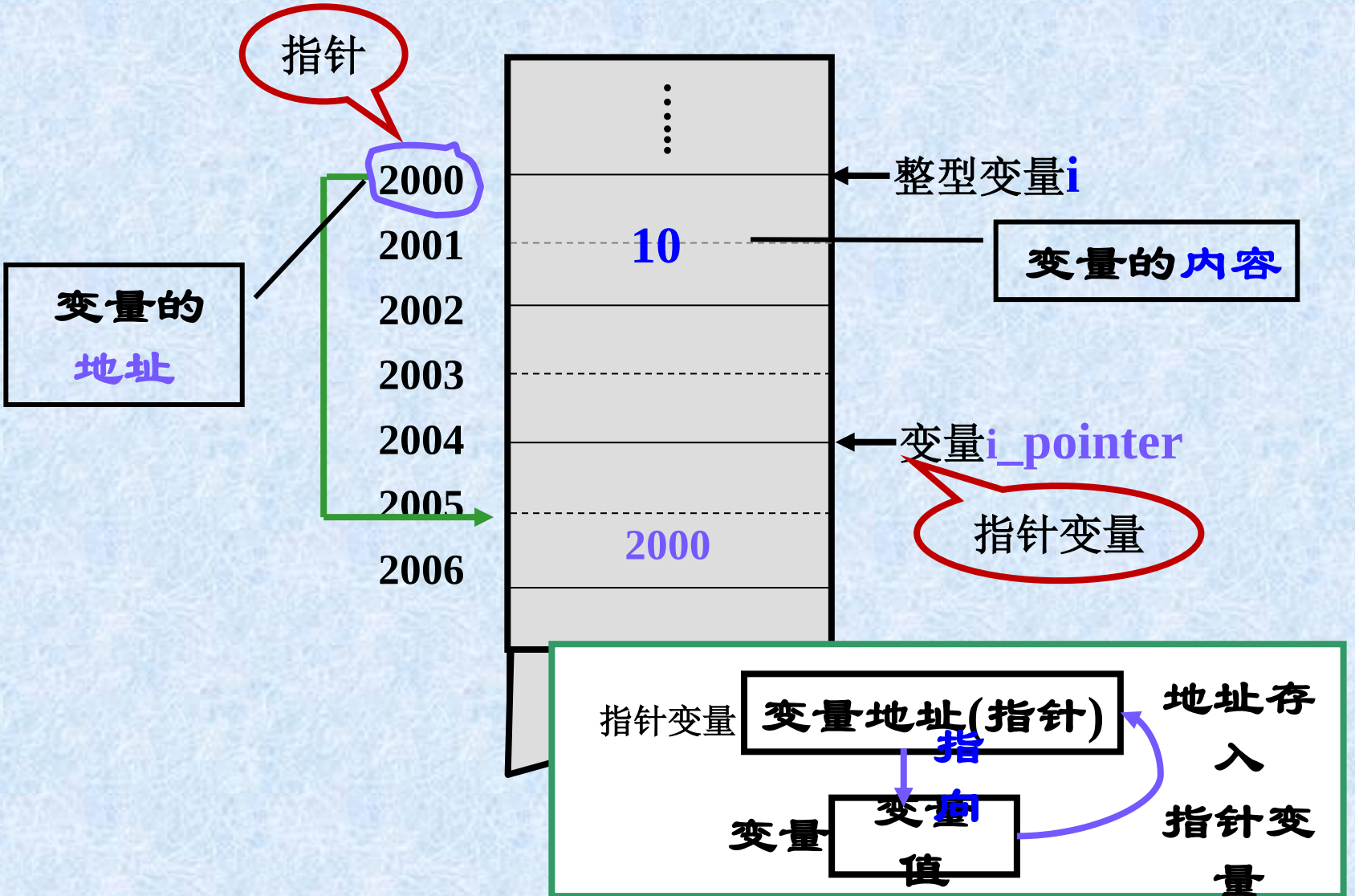
- 变量与地址



指针与指针变量

指针：一个变量的地址.

指针变量：专门存放变量地址的变量叫~.



&与*运算符

两者关系：互为逆运算

- 含义

含义: 取变量
单目运算符
优先级: 2
结合性:自

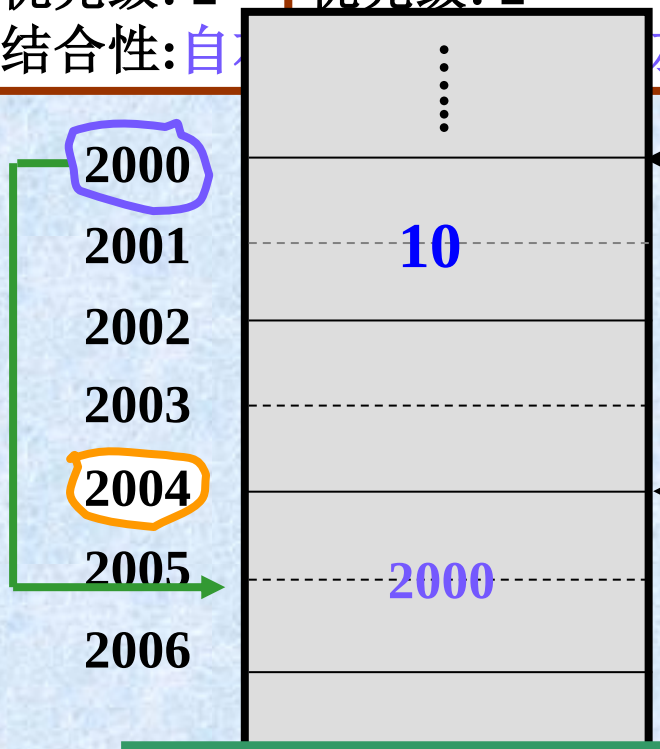
含义: 取指针所指向变量的内容
单目运算符
优先级: 2
左

&i_pointer

i_pointer　　*i_pointer

2000 → 10

i



i_pointer = &i = &(*i_pointer)

i = *i_pointer = *(&i)

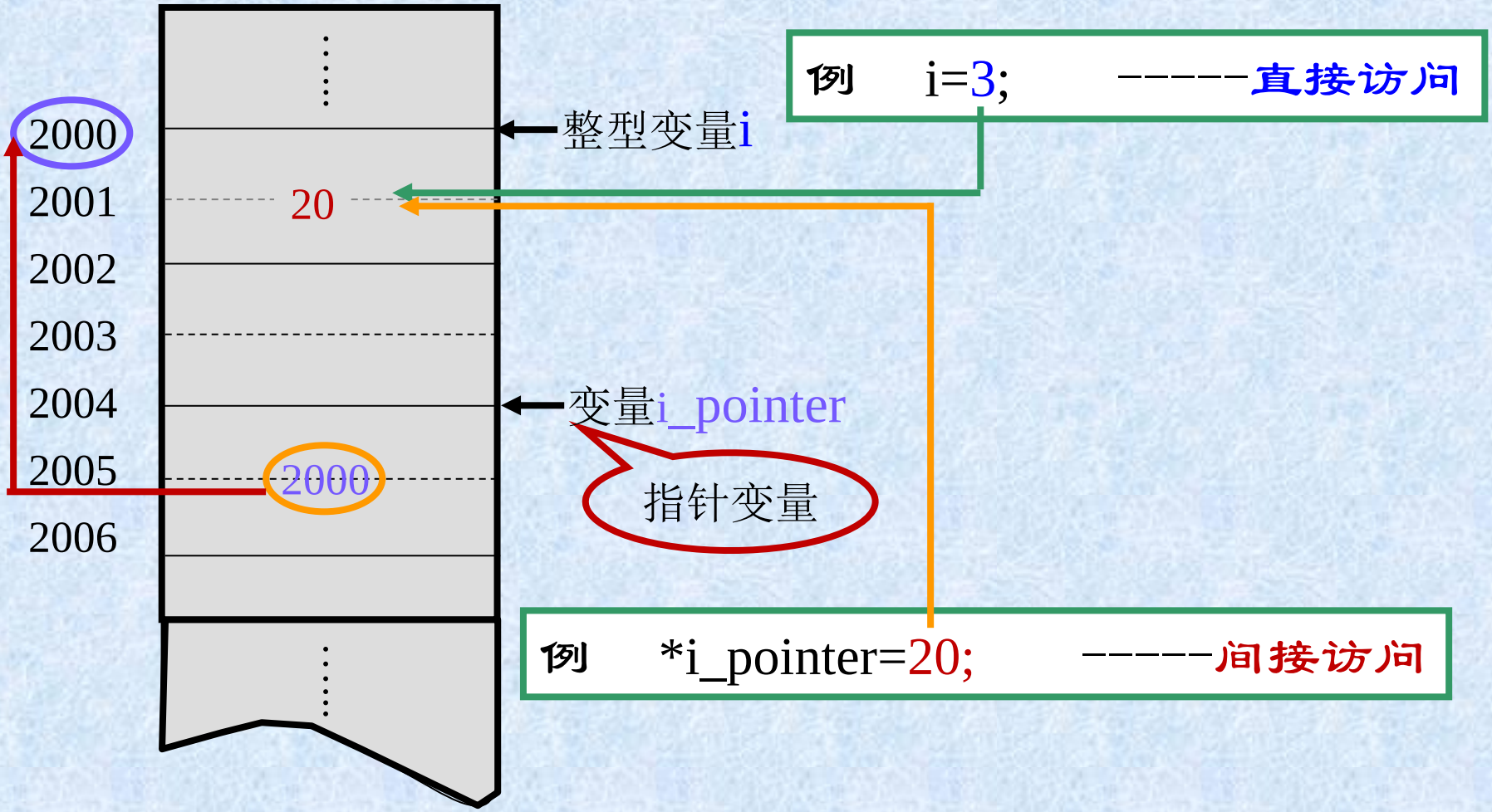
i_pointer------指针变量，它的内容是地址量

*i_pointer-----指针的目标变量，它的内容是数据

&i_pointer---指针变量占用内存的地址

直接访问与间接访问

- 直接访问：按变量地址存取变量值
- 间接访问：通过存放变量地址的变量去访问变量



复习二：　结构体

- 结构体是一种构造数据类型
- 用途：把不同类型的数据组合成一个整体-------自定义数据类型
- 结构体类型定义

```
struct    [结构体名]
{
      类型标识符    成员名;
      类型标识符    成员名;
          ……
};
```

合法标识符

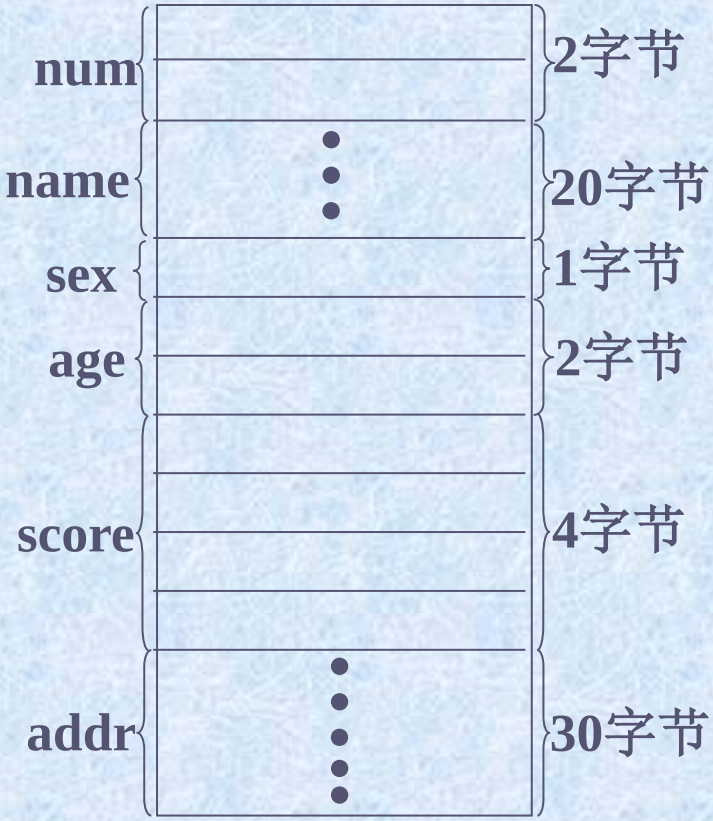
可省:无名结构体

struct是关键字，不能省略

成员类型可以是基本型或构造型

例子图解

```
例    struct   student
      {       int num;
              char  name[20];
              char sex;
              int age;
              float score;
              char addr[30];
      };
```



结构体类型定义的作用域

结构体类型定义描述结构的组织形式,不分配内存

结构体变量的定义

- 先定义结构体类型，再定义结构体变量
  - 一般形式：

```
struct    结构体名
{
      类型标识符    成员名；
      类型标识符    成员名；
          ……
};
struct  结构体名  变量名表列；
```

定义结构体类型的同时定义结构体变量

一般形式：

```
struct     结构体名
{
      类型标识符    成员名；
      类型标识符    成员名；
          ……
}变量名表列；
```

例

```
struct   student
{       int num;
        char  name[20];
        char sex;
        int age;
        float score;
        char addr[30];
}stu1,stu2;
```

说明：

- 结构体类型与结构体变量概念不同

  类型:不分配内存；变量:分配内存
  类型:不能赋值、存取、运算；　变量:可以
- 结构体可嵌套

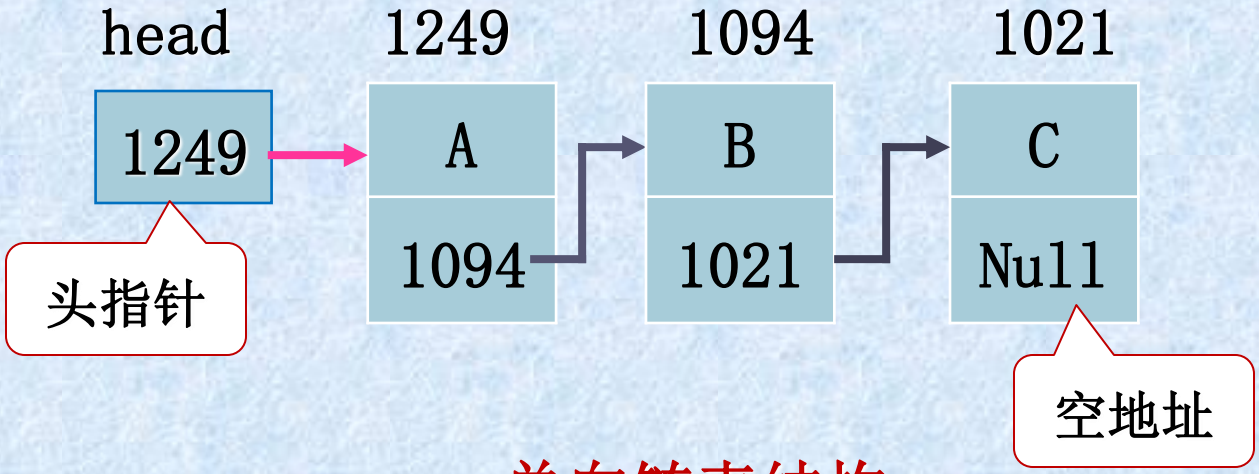
结构体成员名与程序中变量名可相同，不会混淆；

复习三： 用指针处理链表

一、链表概述

链表是指将若干个数据项按一定的原则连接起来的表。链表中每一个数据称为节点。链表连接的原则是：前一个节点指向下一个节点;而且只有通过前一个节点才能找到下一个节点。

链表是一种常见的重要的数据结构。利用它可以实现动态地进行存储分配。

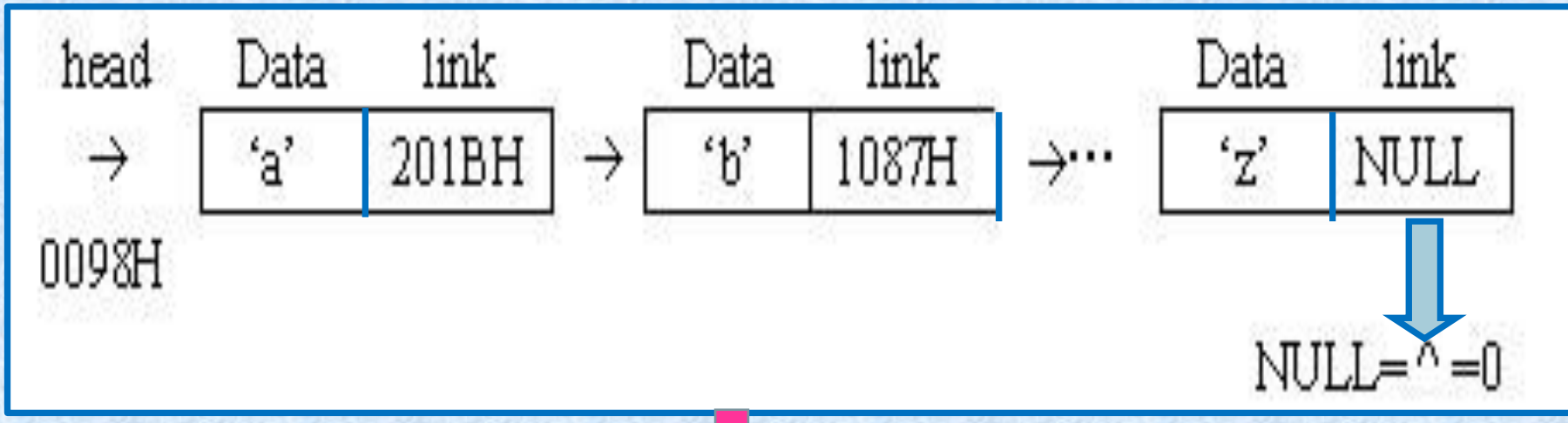


------ 单向链表结构

例： 画出26 个英文字母表的链式存储结构

逻辑结构： （ a, b, … ,y, z)

链式存储结构：



复习四：函数参数及其传递方式

- 形参与实参
  - 形式参数：定义函数时函数名后面括号中的变量名
  - 实际参数：调用函数时函数名后面括号中的表达式

例: 比较两个数并输出大者

```
main()
{   int a,b,c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("Max is %d",c);
}
max(int  x, int  y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```

实参

形参

```
    c=max(a,b);
max(int x, int  y)
{   int z;
    z=x>y?x:y;
    return(z);
}              //（max 函数）
```

函数参数及其传递方式

形参与实参

形式参数：定义函数时函数名后面括号中的变量名

实际参数：调用函数时函数名后面括号中的表达式

说明：

实参必须有确定的值

形参必须指定类型

形参与实参类型一致，个数相同

若形参与实参类型不一致，自动按形参类型转换——函数调用转换

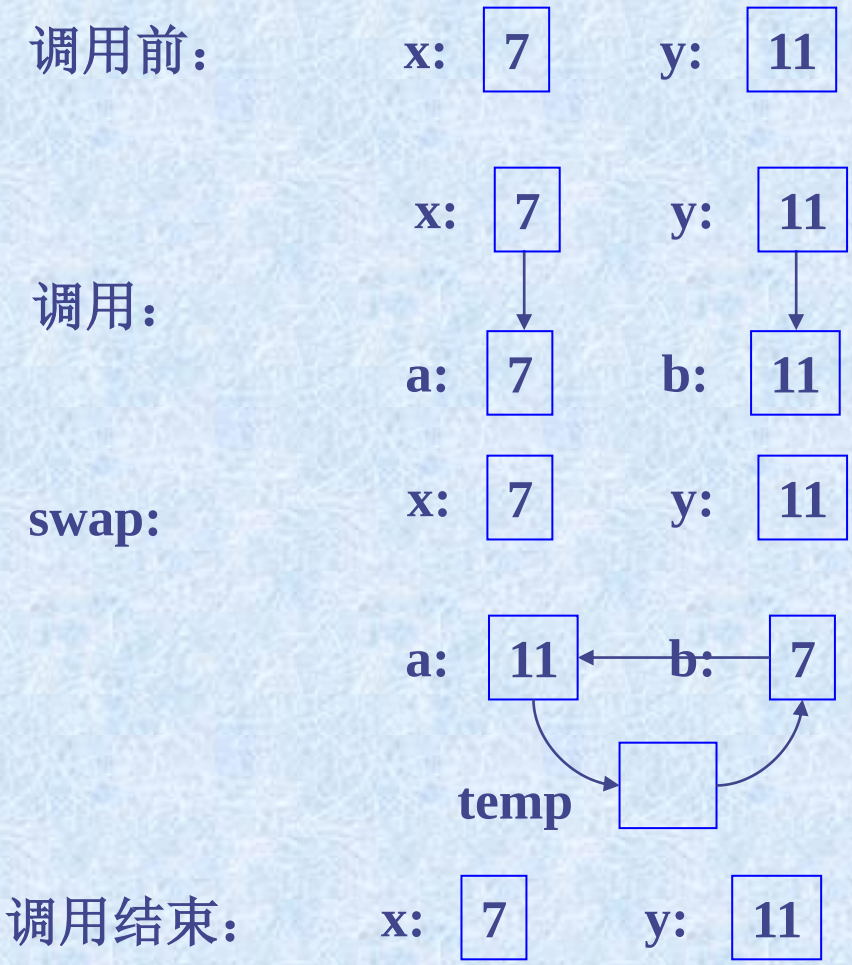
形参在函数被调用前不占内存;函数调用时为形参分配内存；调用结束，内存释放

参数传递方式

- 值传递方式
  - 方式：函数调用时,为形参分配单元,并将实参的值复制到形参中；调用结束，形参单元被释放，实参单元仍保留并维持原值
  - 特点：
    - 形参与实参占用不同的内存单元
    - 单向传递

例：交换两个数

```
/*ch7_2.c*/
#include <stdio.h>
main()
{   int x=7,y=11;
    printf("x=%d,\ty=%d\n",x,y);
    printf("swapped:\n");
    swap(x,y);
    printf("x=%d,\ty=%d\n",x,y);
}
swap(int a,int b)
{   int temp;
    temp=a; a=b; b=temp;
}
```



函数的地址传递

- 方式：函数调用时，将数据的存储地址作为参数传递给形参
- 特点：
  - 形参与实参占用同样的存储单元
  - “双向”传递
  - 实参和形参必须是地址常量或变量

例：交换两个数

```
swap(p1,p2)
int *p1,*p2;
{ int p;
  p=*p1;
  *p1=*p2;
  *p2=p;
}
main()
{int a,b;
 scanf("%d,%d",&a,&b);
 printf("a=%d,b=%d\n",a,b);
 printf("swapped:\n");
 swap(&a,&b);
 printf("a=%d,b=%d\n",a,b);
}
```

例子图解