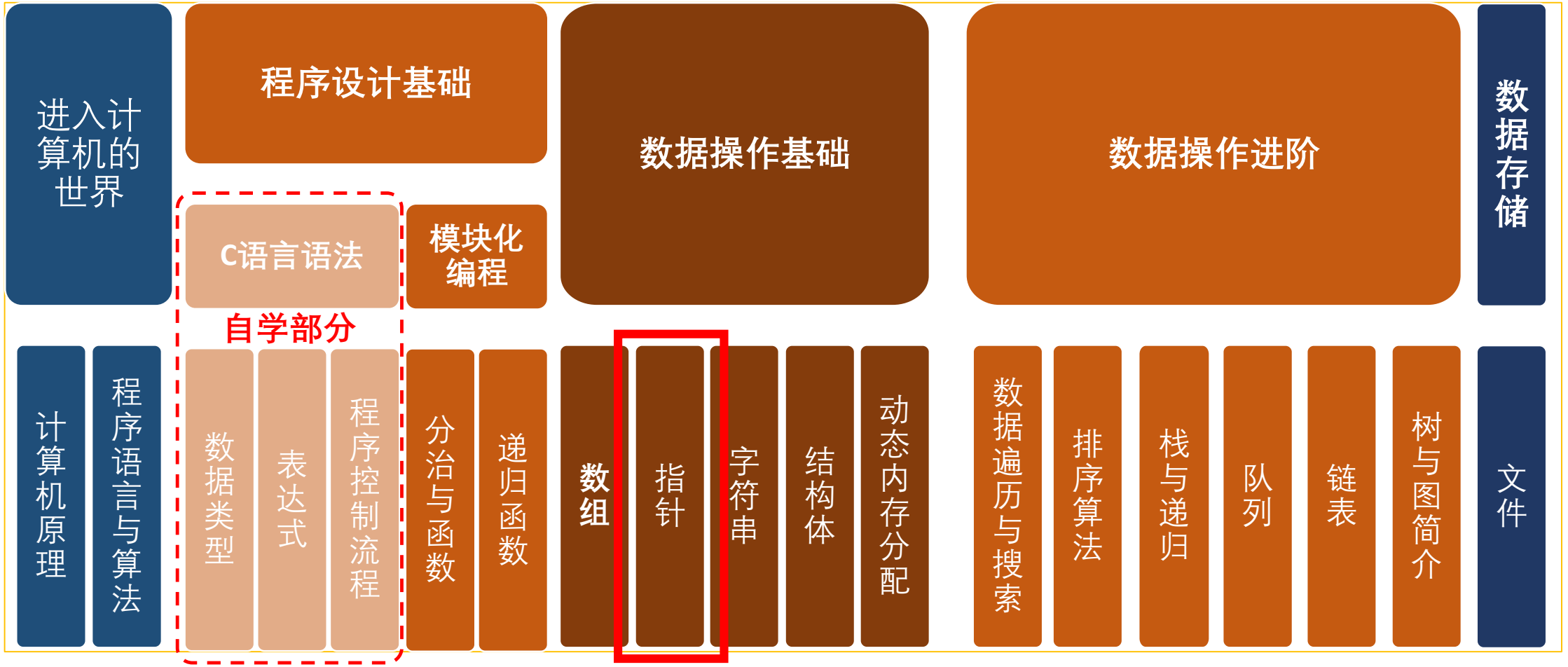
计算思维与实践

哈尔滨工业大学（深圳）
计算机科学与技术学院
大数据技术中心
张保权

课程内容安排



程序设计思想与数据操作方法融会贯通，内容由浅入深

第七讲 指针综合——学习内容

7.1 指针的概念和变量的地址

7.2 指针变量的定义和初始化

7.3 取地址和间接寻址运算符

7.4 指针作函数参数

7.5 指针与一维数组

第七讲 指针综合——学习内容

7.1 指针的概念和变量的地址

7.2 指针变量的定义和初始化

7.3 取地址和间接寻址运算符

7.4 指针作函数参数

7.5 指针与一维数组

7.1.1 指针的概念

- 指针是“稀饭”最挚爱的武器
  - 稀饭 == C Fans
- C的高效、高能主要来自于指针
- 很多“Mission Impossible”由指针完成
  - 大多数语言都有无数的“不可能”
  - 而C语言是

    一切皆有可能” ——

    “Impossible is Nothing” ——

7.1.2 变量的内存地址——基本概念

内存中的每个字节都有唯一的编号（地址）

地址按字节编号，其字长一般与主机相同

32位机使用32位地址，最多支持2^{32}字节内存(4G)

变量的地址

```
int  a=0;
```

按变量名读取变量的值

&a 0x0037b000

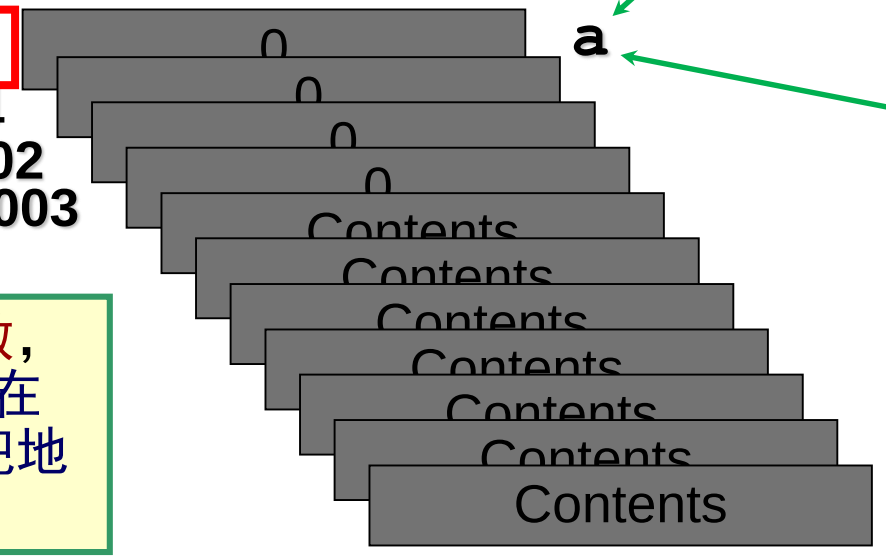
0x0037b001

0x0037b002

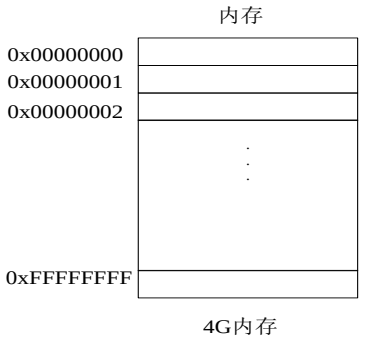
0x0037b003

a

变量名



某存储区域



地址是一个无符号整数，从0开始，依次递增。在表达和交流时，通常把地址写成十六进制数

7.1.3 变量的内存地址——程序示例

【例1】使用取地址运算符&取出变量的地址，然后将其显示在屏幕上。

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0, b = 1;
5       char c = 'A';
6       printf("a is %d, &a is %p\n", a, &a);
7       printf("b is %d, &b is %p\n", b, &b);
8       printf("c is %c, &c is %p\n", c, &c);
9       return 0;
10  }
```

%p表示输出变量a的地址值

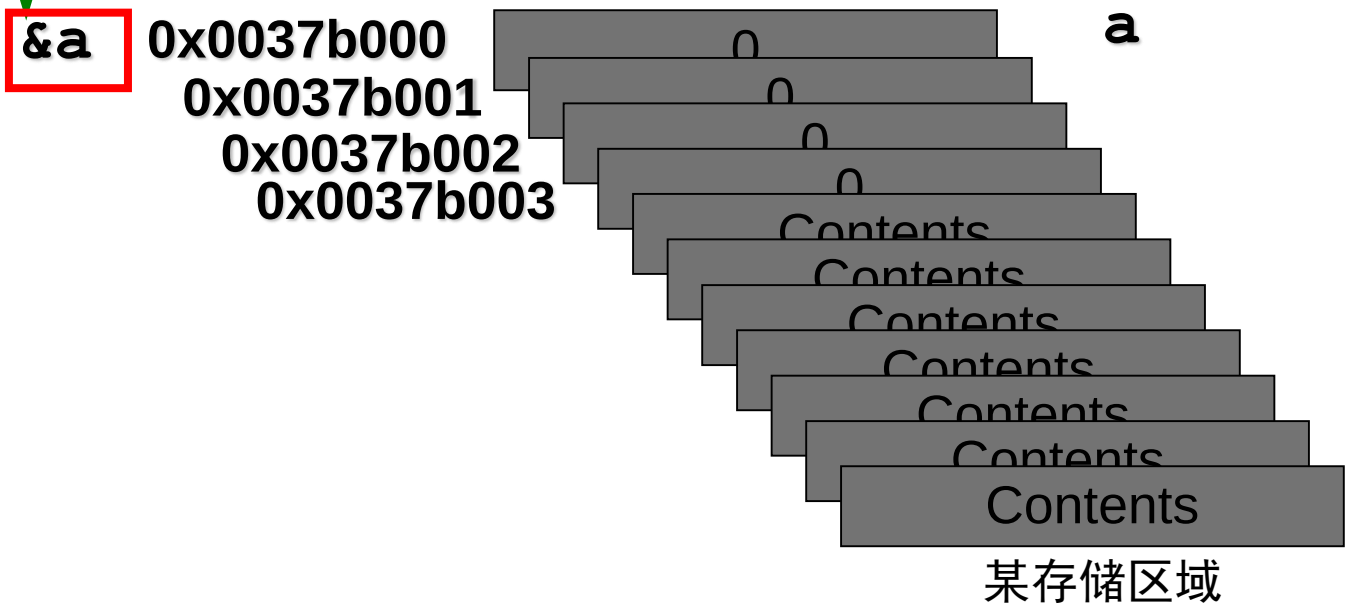
```
a is 0, &a is 0023FF74
b is 1, &b is 0023FF70
c is A, &c is 0023FF6F
```

7.1.4 变量的内存地址——直接寻址

直接寻址：按变量的地址直接访问

```
int  a=0;
```

```
scanf("%d", &a);
```



7.1.4 变量的内存地址——直接寻址

Errors

```
int i;
scanf("%d", i);
/* 这样会如何？*/
```

i的值被当作地址。如i值为100，则输入的整数就会从地址100开始写入内存

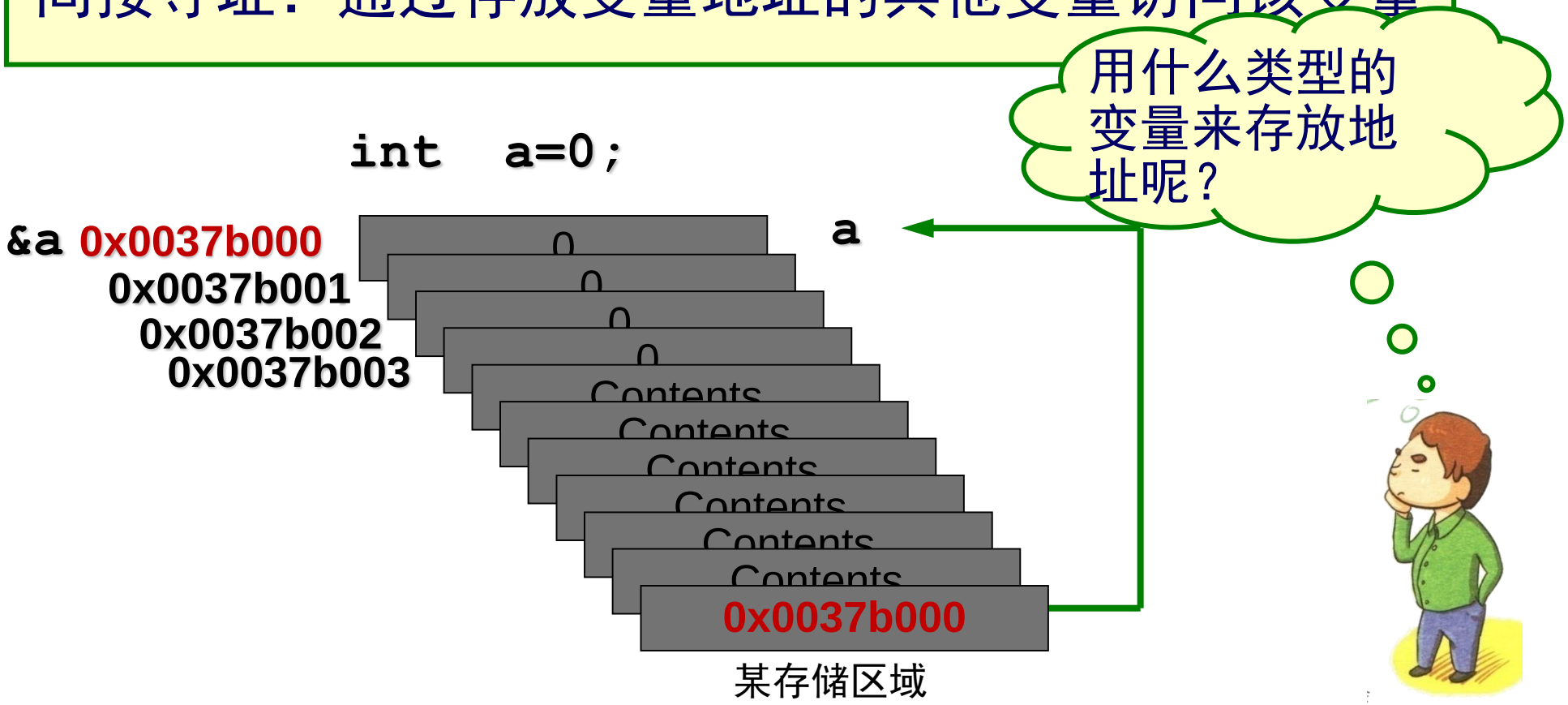
```
char c;
scanf("%d", &c);
/* 这样呢？*/
```

输入数据以int的二进制形式写到c所在的内存空间。

c所占内存不足以放下一个int，其后的空间也被覆盖

7.1.5 变量的内存地址——间接寻址

间接寻址：通过存放变量地址的其他变量访问该变量



第七讲 指针综合——学习内容

7.1 指针的概念和变量的地址

7.2 指针变量的定义和初始化

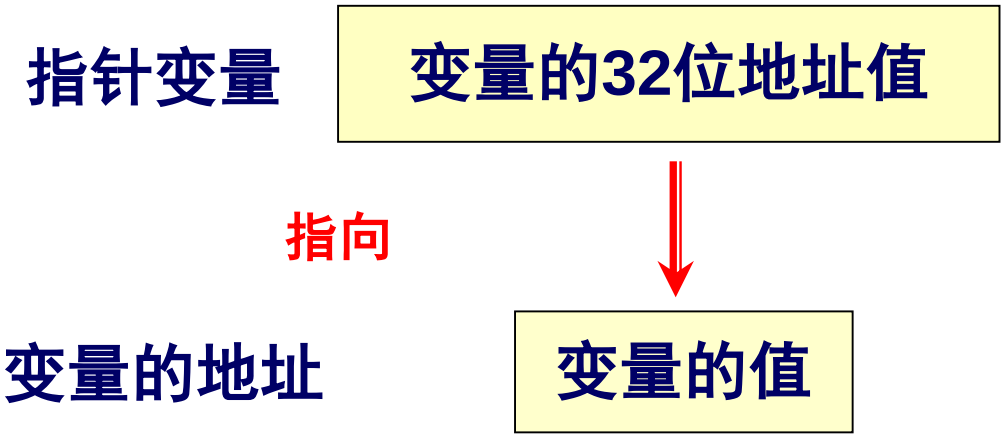
7.3 取地址和间接寻址运算符

7.4 指针作函数参数

7.5 指针与一维数组

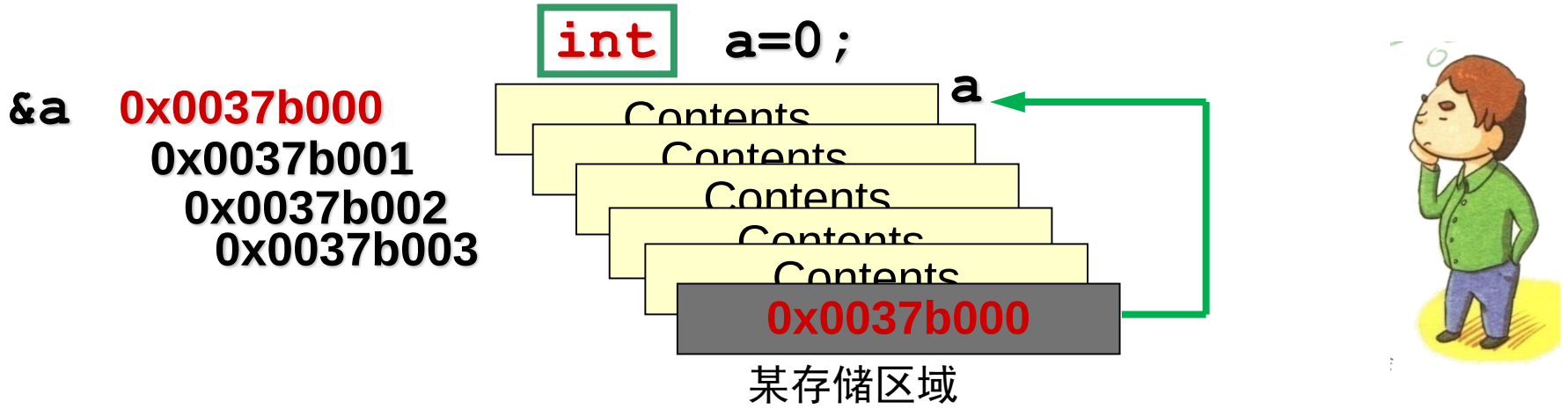
7.2.1 指针变量的定义和初始化——概述

- 用什么类型的变量来存放变量的地址呢？
  - 指针（Pointer）类型
- 指针变量——具有指针类型的变量
- 变量的指针←→变量的地址



7.2.2 指针变量的定义和初始化——指针基类型

- 保存32位地址值的指针变量占4个字节的内存，这4个字节中保存了一个地址
  - 只知道这些是不够的
- 从这个地址开始多少个字节内的数据是有效的呢？
- 用什么数据类型去理解这些数据呢？
  - 指针的基类型就是回答这个问题的



7.2.2 指针变量的定义和初始化——指针基类型

【例2】使用

指针变量指向的数据类型，称为基类型

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0;
5       char c = 'A';
6       int *pa, *pb;          /* 定义了可以指向整型数据的指针变量 pa 和 pb */
7       char *pc;              /* 定义了可以指向字符型数据的指针变量 pc */
8       printf("a is %d, &a is %p, pa is %p\n", a, &a, pa);
9       printf("b is %d, &b is %p, pb is %p \n", b, &b, pb);
10      printf("c is %c, &c is %p, pc is %p \n", c, &c, pc);
11      return 0;
12  }
```

告诉编译器，pa是一个指针变量，占4字节内存，需要用一个int型变量的地址给它赋值，但pa并未具体指向某个int型变量

```
a is 0, &a is 0023FF74, pa is 0023FF78
b is 1, &b is 0023FF70, pb is 00401394
c is A, &c is 0023FF6F, pc is 77C04E42
```

7.2.3 指针变量的定义和初始化——指针初始化

【例3】使用指针变量在屏幕上显示变量的地址值

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0, b = 1;
5       char c = 'A';
6       int *pa, *pb;          /* 定义了可以指向整型数据的指针变量 pa 和 pb */
7       char *pc;              /* 定义了可以指向字符型数据的指针变量 pc */
8       printf("a is %d, &a is %p, pa is %p\n", a, &a, pa);
9       printf("b is %d, &b is %p, pb is %p \n", b, &b, pb);
10      printf("c is %c, &c is %p, pc is %p \n", c, &c, pc);
11      return 0;
12  }
```

指针变量使用之前必须初始化
Never use uninitialized pointers

```
warning: local variable 'pa' used without having been initialized
warning: local variable 'pb' used without having been initialized
warning: local variable 'pc' used without having been initialized
```

7.2.3 指针变量的定义和初始化——指针初始化

【例4】使用指针变量在屏幕上显示变量的地址值

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0
5       char c = 'A';
6       int *pa = NULL, *pb = NULL; /* 定义指针变量并用 NULL 对其初始化 */
7       char *pc = NULL;            /* 定义指针变量并用 NULL 对其初始化 */
8       printf("a is %d, &a is %p, pa is %p\n", a, &a, pa);
9       printf("b is %d, &b is %p, pb is %p \n", b, &b, pb);
10      printf("c is %c, &c is %p, pc is %p \n", c, &c, pc);
11      return 0;
12  }
```

如果你不知把它指向哪里，那就指向**NULL**

```
a is 0, &a is 0013FF7C, pa is 00000000
b is 1, &b is 0013FF78, pb is 00000000
c is A, &c is 0013FF74, pc is 00000000
```

7.2.4 指针变量的定义和初始化——空指针

- 何谓空指针？
  - 值为**NULL**的指针，即无效指针
- 既然0（**NULL**）用来表示空指针，那么空指针就是指向地址为0的单元的指针吗？
- 答案：不一定
  - 每个C编译器都被允许用不同的方式来表示空指针
  - 并非所有编译器都使用0地址
  - 某些编译器为空指针使用不存在的内存地址
  - 硬件会检查出这种试图通过空指针访问内存的方式

第七讲 指针综合——学习内容

7.1 指针的概念和变量的地址

7.2 指针变量的定义和初始化

7.3 取地址和间接寻址运算符

7.4 指针作函数参数

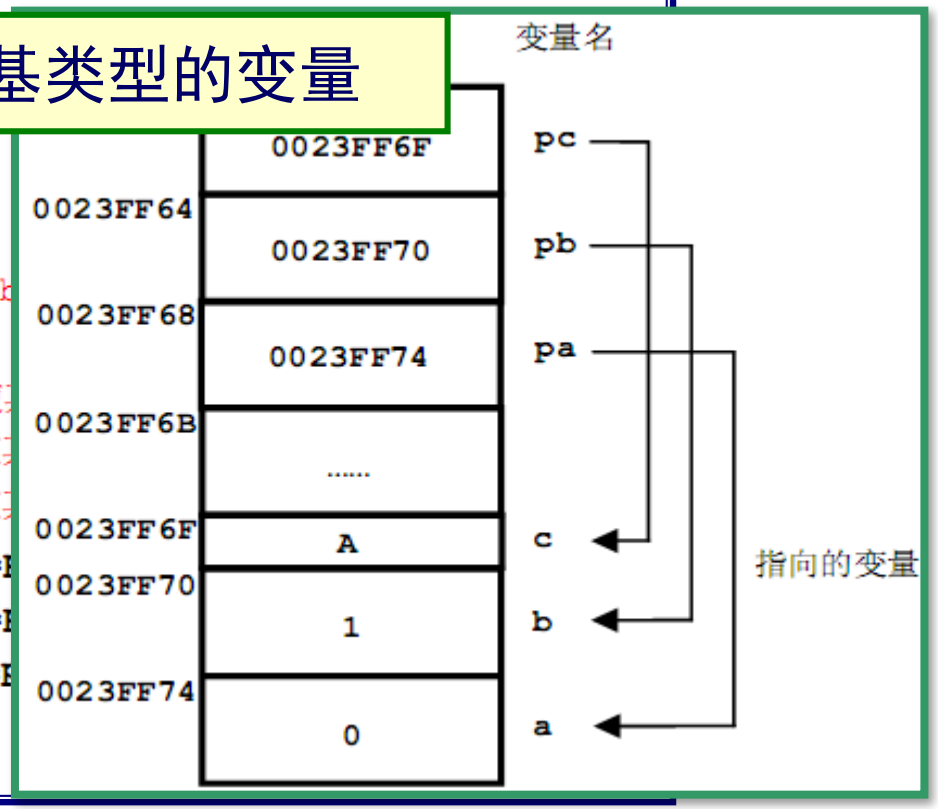
7.5 指针与一维数组

7.3.1 取地址和间接寻址运算符——错误示例

【例5】使用指针变量在屏幕上显示变量的地址值

```
1   #include
2   int main
3   {
4       int a = 0, b = 1;
5       char c = 'A';
6       int *pa, *pb;          /* 定义指针变量 pa 和 pb
7       char *pc;              /* 定义指针变量 pc  */
8       pa = &a;               /* 初始化指针变量 pa 使
9       pb = &b;               /* 初始化指针变量 pb 使
10      pc = &c;               /* 初始化指针变量 pc 使
11      printf("a is %d, &a is %p, pa is %p, &p
12      printf("b is %d, &b is %p, pb is %p, &p
13      printf("c is %c, &c is %p, pc is %p, &p
14      return 0;
15  }
```

指针变量只能指向同一基类型的变量




```
a is 0, &a is 0023FF74, pa is 0023FF74, &pa is 0023FF68
b is 1, &b is 0023FF70, pb is 0023FF70, &pb is 0023FF64
c is A, &c is 0023FF6F, pc is 0023FF6F, &pc is 0023FF60
```

7.3.1 取地址和间接寻址运算符——错误示例

【例6】使用指针变量在屏幕上显示变量的地址值

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0, b = 1;
5       char c = 'A';
6       int *pa, *pb;          /* 定义指针变量 pa 和 pb */
7       char *pc;              /* 定义指针变量 pc */
8       pa = &a;               /* 初始化指针变量 pa 使其指向 a */
9       pb = &b;               /* 初始化指针变量 pb 使其指向 b */
10      pc = &c;               /* 初始化指针变量 pc 使其指向 c */
11      printf("a is %d, &a is %p, pa is %p, &pa is %p\n", a, &a, pa, &pa);
12      printf("b is %d, &b is %p, pb is %p, &pb is %p\n", b, &b, pb, &pb);
13      printf("c is %c, &c is %p, pc is %p, &pc is %p\n", c, &c, pc, &pc);
14      return 0;
15  }
```

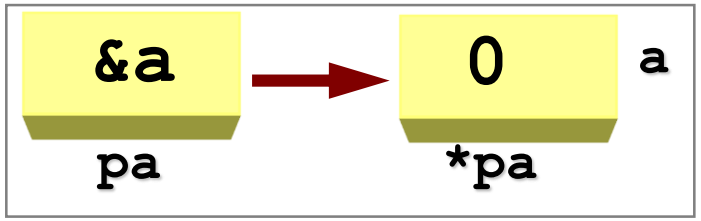
不能写成：int* pa, pb;

```
a is 0, &a is 0023FF74, pa is 0023FF74, &pa is 0023FF68
b is 1, &b is 0023FF70, pb is 0023FF70, &pb is 0023FF64
c is A, &c is 0023FF6F, pc is 0023FF6F, &pc is 0023FF60
```

7.3.2 取地址和间接寻址运算符——间接寻址运算符

【例7】使用指针变量，通过间接寻址输出变量的值

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0, b = 1;
5       char c = 'A';
6       int *pa = &a, *pb = &b; /* 在定义指针变量 pa 和 pb 的同时对其初始化 */
7       char *pc = &c;          /* 在定义指针变量 pc 的同时对其初始化 */
8       printf("a is %d, &a is %p, pa is %p, *pa is %d\n", a, &a, pa, *pa);
9       printf("b is %d, &b is %p, pb is %p, *pb is %d\n", b, &b, pb, *pb);
10      printf("c is %c, &c is %p, pc is %p, *pc is %c\n", c, &c, pc, *pc);
11      return 0;
12  }
```

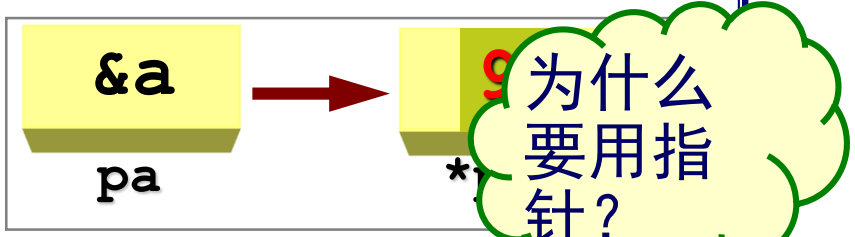



```
a is 0, &a is 0023FF74, pa is 0023FF74, *pa is 0
b is 1, &b is 0023FF70, pb is 0023FF70, *pb is 1
c is A, &c is 0023FF6F, pc is 0023FF6F, *pc is A
```

7.3.3 取地址和间接寻址运算符——解引用

【例8】使用指针变量，通过间接寻址输出变量的值

```
1   #include <stdio.h>
2   int main()
3   {
4       int  a = 0, b = 1;
5       char c = 'A';
6       int *pa = &a, *pb = &b; /* 在定义指针变量 pa 和 pb 的同时对其初始化 */
7       char *pc = &c;          /* 在定义指针变量 pc 的同时对其初始化 */
8       *pa = 9;                /* 修改指针变量 pa 所指向的变量的值 */
9       printf("a is %d, &a is %p, pa is %p, *pa is %d\n", a, &a, pa, *pa);
10      printf("b is %d, &b is %p, pb is %p, *pb is %d\n", b, &b, pb, *pb);
11      printf("c is %c, &c is %p, pc is %p, *pc is %c\n", c, &c, pc, *pc);
12
13  }
```



引用指针指向的变量的值，称为指针的解引用(Pointer Dereference)

```
a is 9, &a is 0023FF74, pa is 0023FF74, *pa is 9
b is 1, &b is 0023FF70, pb is 0023FF70, *pb is 1
c is A, &c is 0023FF6F, pc is 0023FF6F, *pc is A
```

第七讲 指针综合——学习内容

7.1 指针的概念和变量的地址

7.2 指针变量的定义和初始化

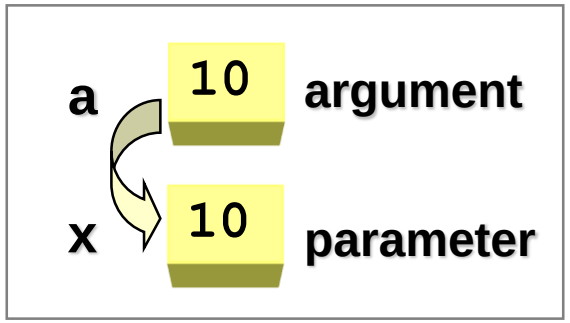
7.3 取地址和间接寻址运算符

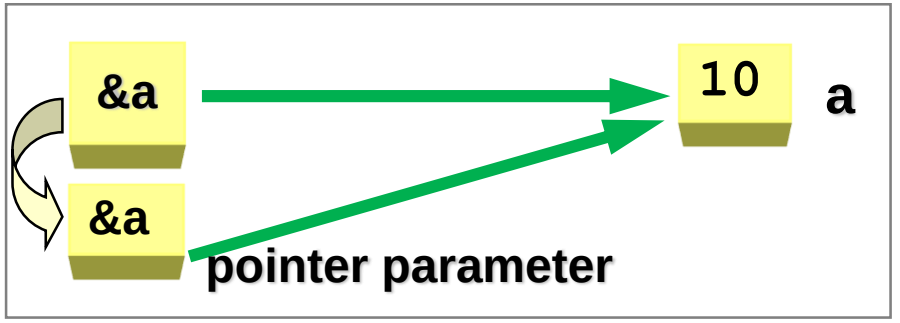
7.4 指针作函数参数

7.5 指针与一维数组

7.4.1 指针作函数形参——基本概念

- 普通变量作函数参数—按值调用
  - 实参的值不随形参值的改变而改变
- 形参（**parameter**）← 实参变量的值
- 指针做函数参数—按地址调用
  - 为了在被调函数中修改其无法直接访问的实参的值
- 指针形参(**pointer parameter**) ← 实参变量的地址





7.4.2 指针作函数形参——简单程序示例

【例9】演示按值调用

```
1    #include <stdio.h>
2    void Fun(int  par);
3    int main()
4    {
5        int arg
6        printf("arg = %d\n", arg);
7        Fun(arg);
8        printf("arg = %d\n", arg);
9        return 0;
10   }
11   void Fun(int par)
12   {
13       printf("par = %d\n", par);
14       par = 2;
15   }
```

传变量的值

```
arg = 1
par = 1
arg = 1
```

形参值的改变
不影响对应的实参

【例10】演示按地址调用

```
1    #include <stdio.h>
2    void Fun(int *par);
3    int main()
4    {
5        int arg
6        printf("arg = %d\n", arg);
7        Fun(&arg);
8        printf("arg = %d\n", arg);
9        return 0;
10   }
11   void Fun(int *par)
12   {
13       printf("par = %d\n", *par);
14       *par = 2;
15   }
```

传变量的地址

指针变量作函数形参可以修改相应实参的值，为函数提供了修改变量值的手段

```
arg = 1
par = 1
arg = 2
```

7.4.4 指针作函数形参——输出最高分和学号

【例12】计算并输出最高分及相应学生的学号

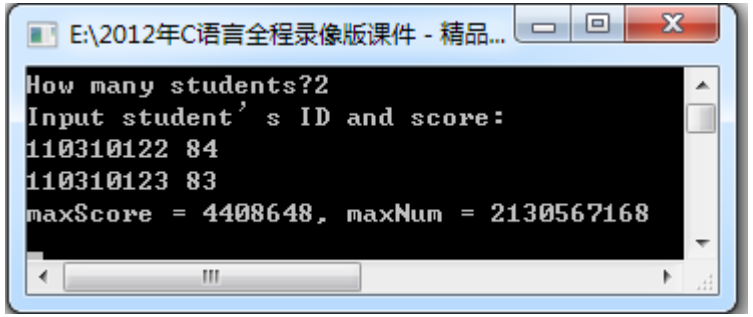
```
4   int main()
5   {
6       int  score[N], maxScore;
7       int    n, i;
8       long   num[N], maxNum;
9       printf("How many students?");
10      scanf("%d", &n);
11      printf("Input student's ID and score:\n");
12      for (i=0; i<n; i++)
13      {
14          scanf("%ld%d", &num[i], &score[i]);   /* 字母d前为字母l */
15      }
16      FindMax(score, num, n, maxScore, maxNum); /* 按值调用函数 */
17      printf("maxScore = %d, maxNum = %ld\n", maxScore, maxNum);
18      return 0;
19  }
```

7.4.4 指针作函数形参——输出最高分和学号


```
void FindMax(int score[],long num[],int n, int pMaxScore, long pMaxNum)
{
    int    i;
    pMaxScore = score[0];
    pMaxNum = num[0];
    for (i=1; i<n; i++)
    {
        if (score[i] > pMaxScore)
        {
            pMaxScore = score[i];
            pMaxNum = num[i];
        }
    }
}
```

真正原因：普通变量作函数参数按值调用
不能在被调函数中改变相应实参的值

Not Work! Why?



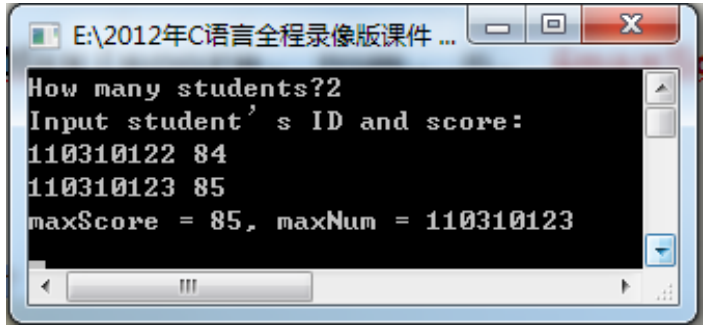
```
FindMax(score, num, n, maxScore, maxNum);
```

VC
```
warning: local variable 'maxNum' used without having been initialized
warning: local variable 'maxScore' used without having been initialized
```

CB
```
warning: 'maxScore' is used uninitialized in this function
warning: 'maxNum' is used uninitialized in this function
```

7.4.4 指针作函数形参——输出最高分和学号

```
void FindMax(int score[],long num[],int n, int *pMaxScore, long *pMaxNum)
{
    int    i;
    *pMaxScore = score[0];
    *pMaxNum = num[0];
    for (i=1; i<n; i++)
    {
        if (score[i] > *pMaxScore)
        {
          *pMaxScore = score[i];
          *pMaxNum = num[i];
        }
    }
}
```




```
FindMax(score, num, n, &maxScore, &maxNum);
```

第七讲 指针综合——学习内容

7.1 指针的概念和变量的地址

7.2 指针变量的定义和初始化

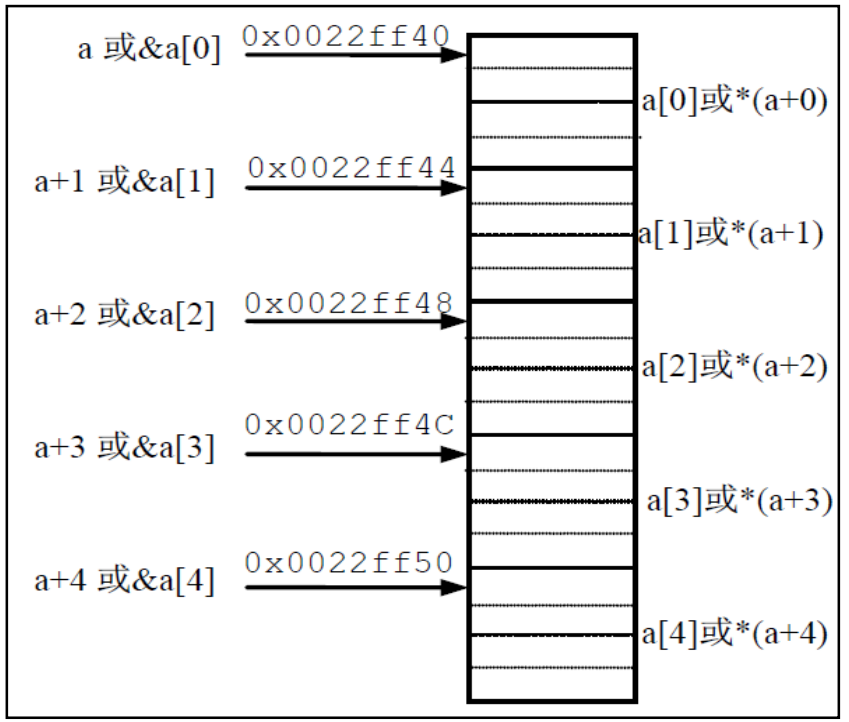
7.3 取地址和间接寻址运算符

7.4 指针作函数参数

7.5 指针与一维数组

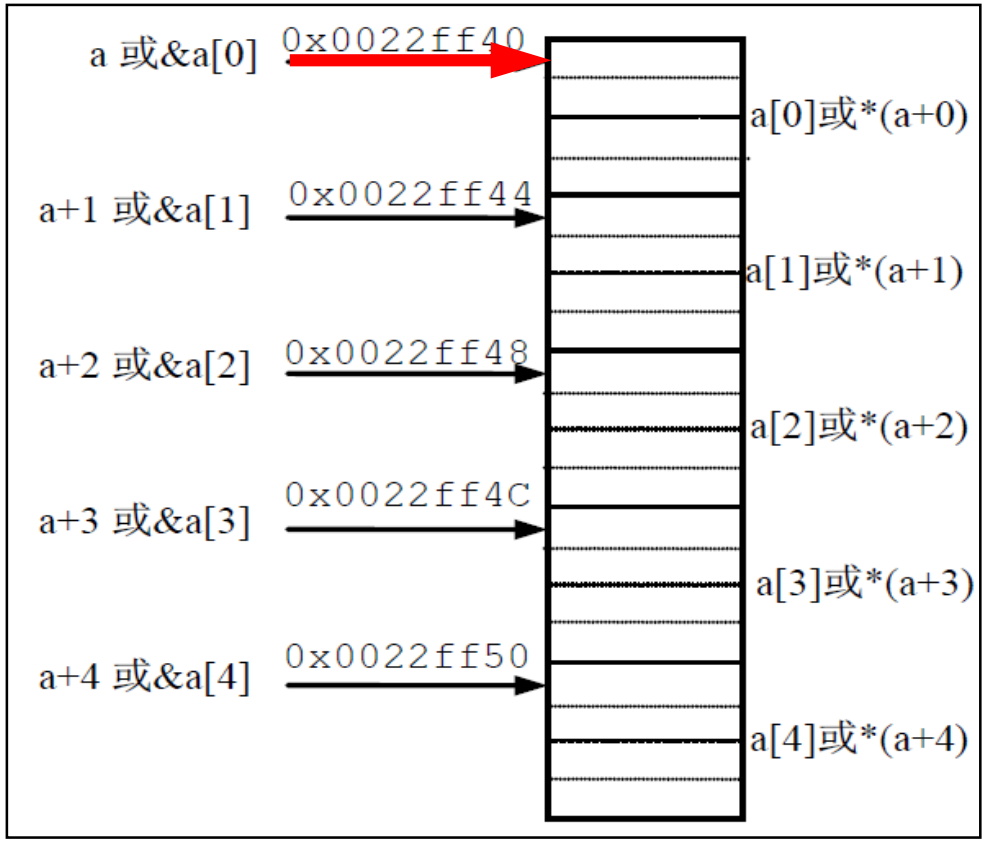
7.5.1 指针和一维数组间的关系——概述

- 指针和数组的关系极为密切
  - 数组元素的等价引用形式：`a[i]` ↔ `*(a+i)`
  - 数组的名称既是标识符又是数组指针
  - 用下标形式访问数组元素，本质是计算该元素在内存中的地址



7.5.2 指针和一维数组间的关系——等价形式

- 为什么一个int型指针能够指向一个整型数组呢？




```
int a[5];
int *p = a;
```

数组名是数组的首地址

```
int *p = &a[0];
```

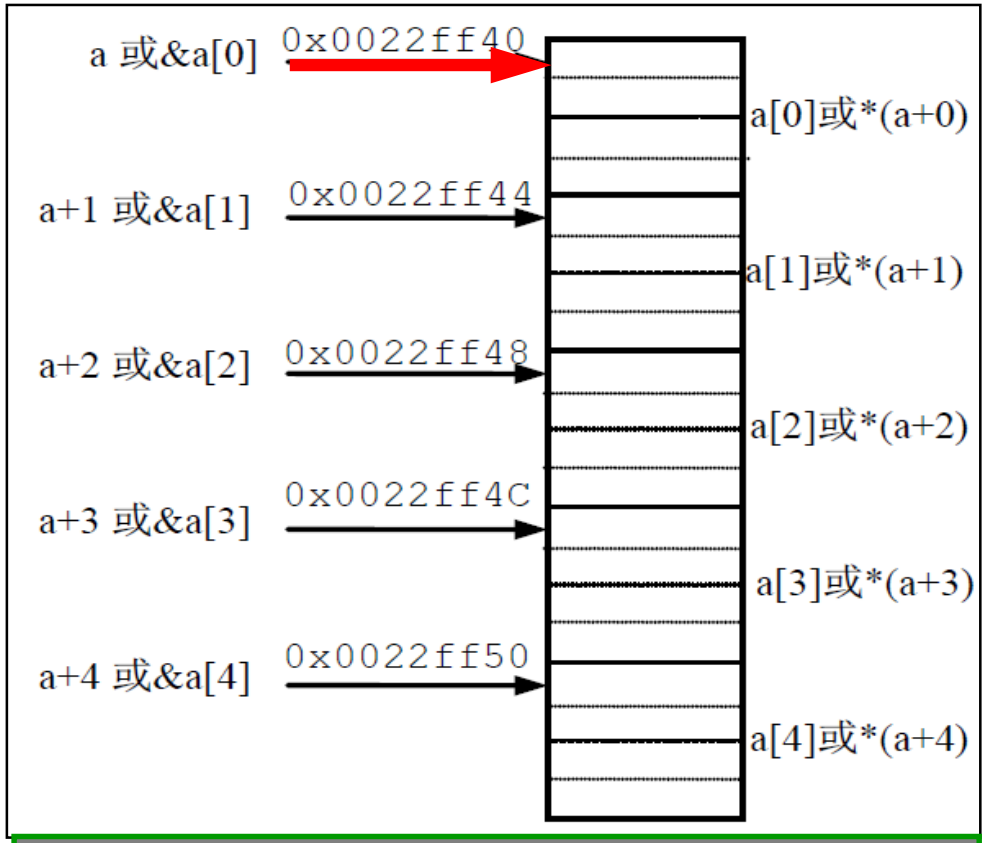
&a[0]是整型元素的地址

p是整型指针

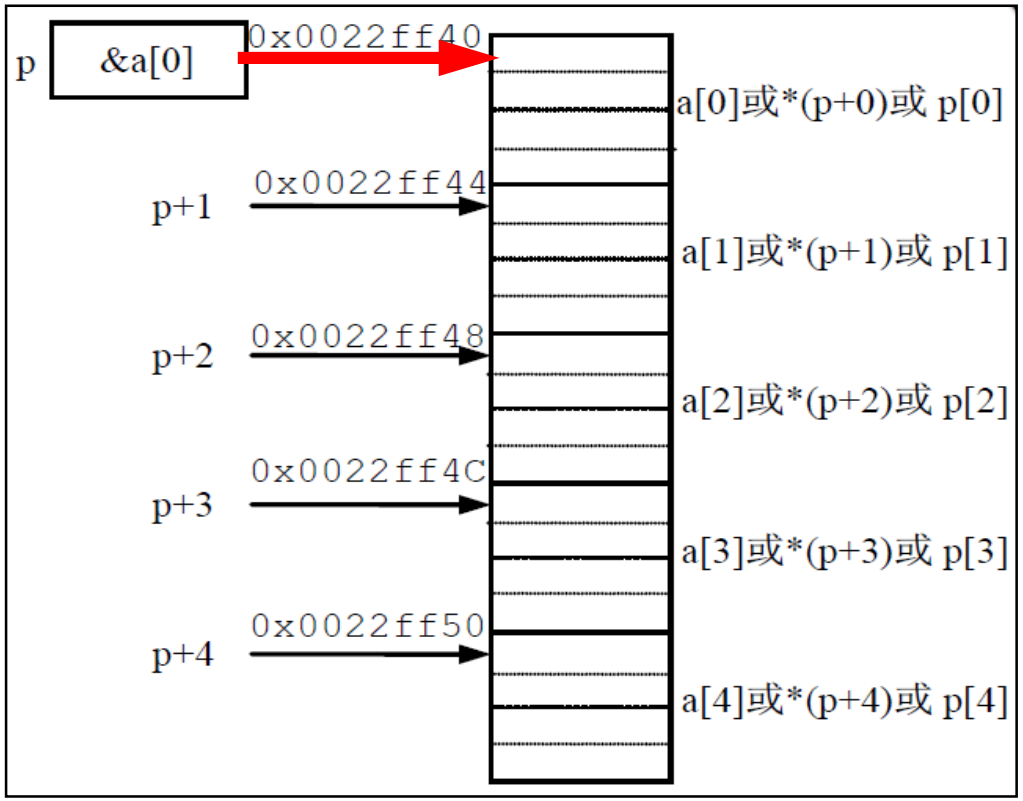
a[0]的类型和p的基类型相同

数组元素的等价引用形式：a[i] ↔ *(a+i)

7.5.2 指针和一维数组间的关系——等价形式



因p保存的是数组的首地址
所以p也可看成是数组名

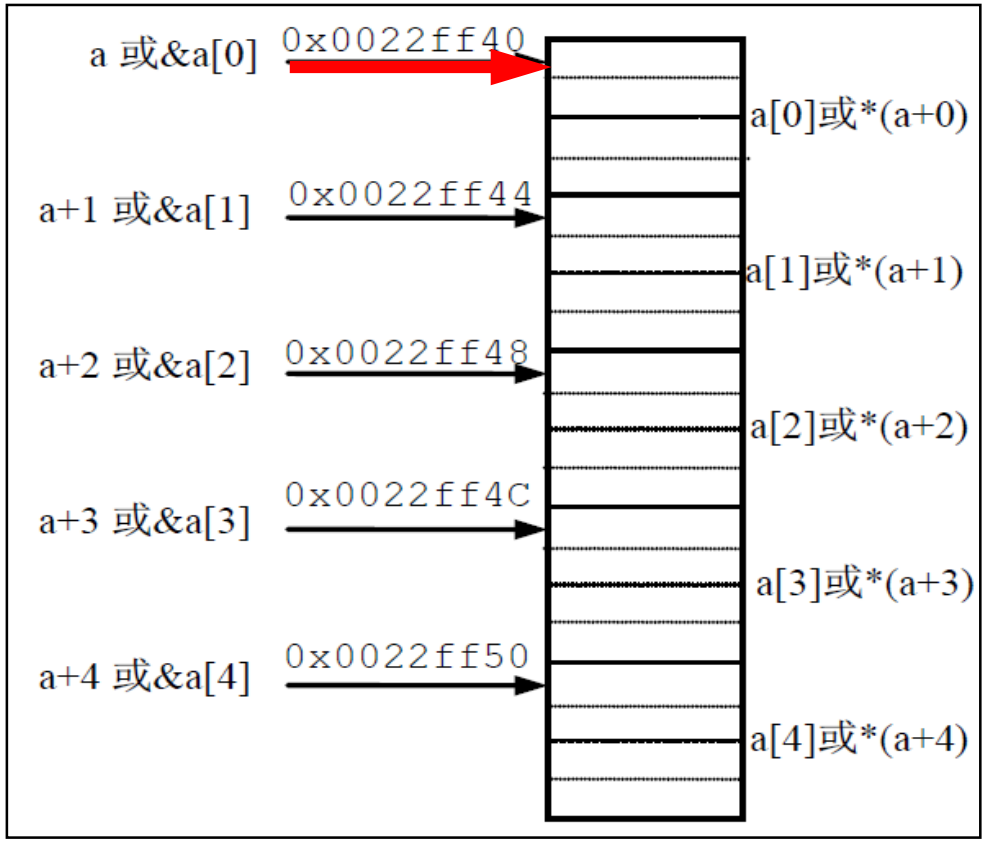


另两种等价的引用形式：

p[i] ↔ *(p+i)

7.5.3 指针和一维数组间的关系——等价示例

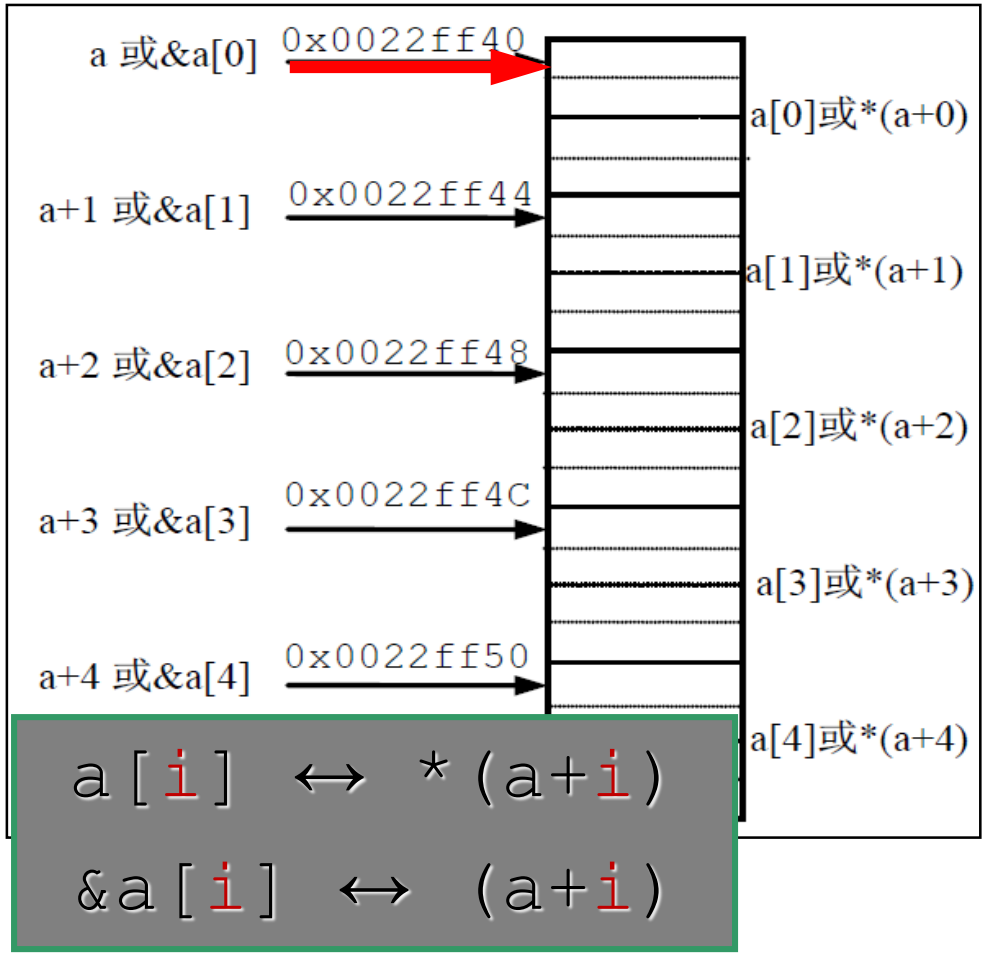
【例13.1】演示数组元素的引用方法




```
1    #include <stdio.h>
2    int  main()
3    {
4        int  a[5], i;
5        printf("Input five numbers:");
6        for (i=0; i<5; i++)
7        {
8            scanf("%d", &a[i]);
9        }
10       for (i=0; i<5; i++)
11       {
12           printf("%4d", a[i]);
13       }
14       printf("\n");
15       return 0;
16   }
```

7.5.3 指针和一维数组间的关系——等价示例

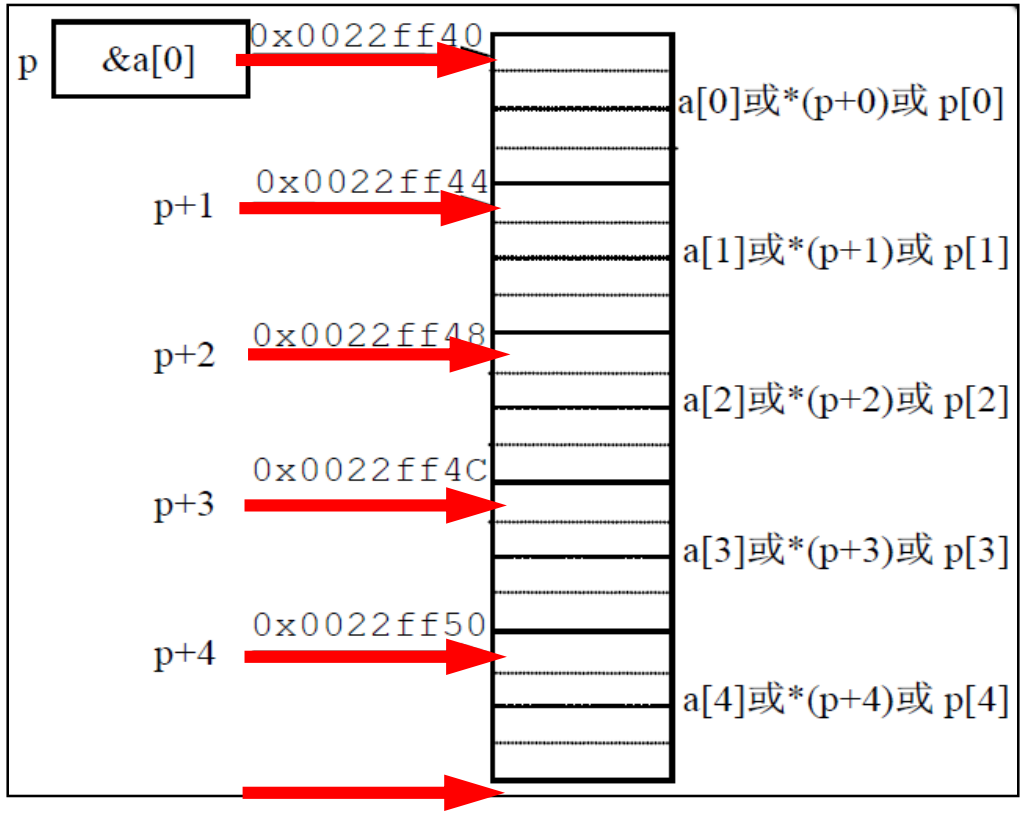
【例13.2】演示数组元素的引用方法




```
1   #include <stdio.h>
2   int  main()
3   {
4       int  a[5], i;
5       printf("Input five numbers:");
6       for (i=0; i<5; i++)
7       {
8           scanf("%d", a+i);
9       }
10      for (i=0; i<5; i++)
11      {
12          printf("%4d", *(a+i));
13      }
14      printf("\n");
15      return 0;
16  }
```

7.5.3 指针和一维数组间的关系——等价示例

【例13.3】演示数组元素的引用方法

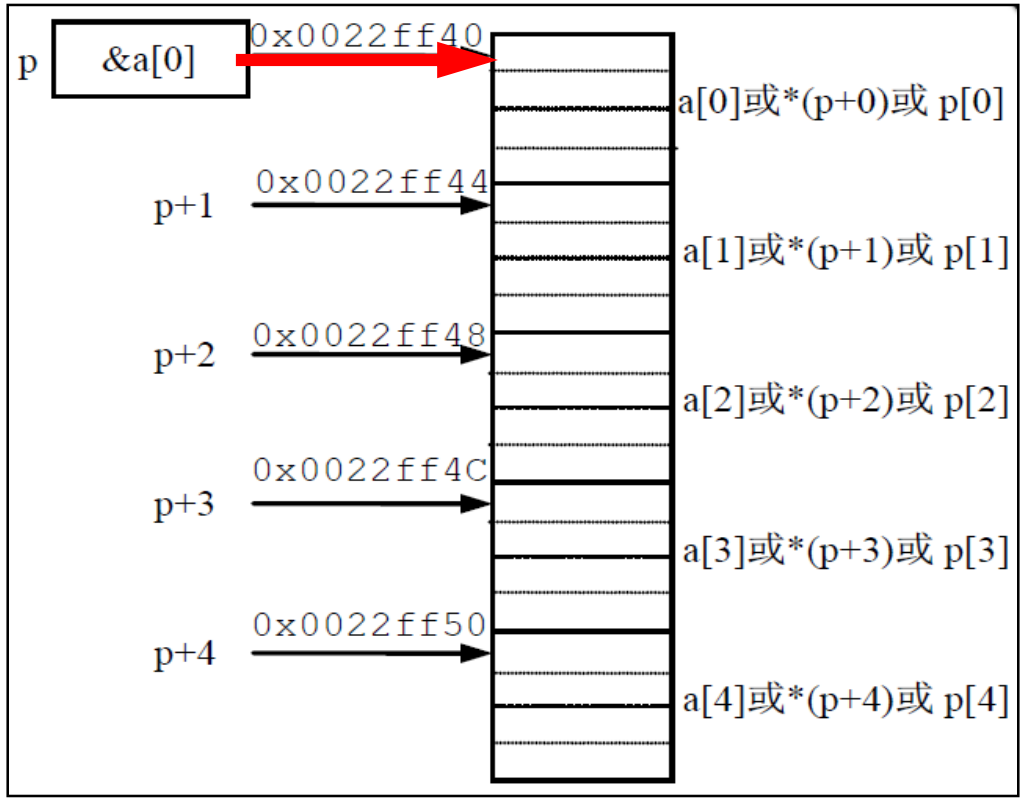



```
1   #include <stdio.h>
2   int  main()
3   {
4       int  a[5], *p;
5       printf("Input five numbers:");
6       for (p = a; p<a+5; p++)
7       {
8           scanf("%d", p);
9       }
10      for (p = a; p<a+5; p++)
11      {
12          printf("%4d", *p);
13      }
14      printf("\n");
        return 0;
```

p++不是增加1字节，取决于p的基类型

7.5.3 指针和一维数组间的关系——等价示例

【例13.4】演示数组元素的引用方法



p[i] ↔ *(p+i)

```
1   #include <stdio.h>
2   int  main()
3   {
4       int  a[5], *p = NULL, i;
5       printf("Input five numbers:");
6       p = a;
7       for (i=0; i<5; i++)
8       {
9           scanf("%d", &p[i]);
10      }
11      p = a;
12      for (i=0; i<5; i++)
13      {
14          printf("%4d", p[i]);
15      }
16      printf("\n");
17      return 0;
18  }
```

7.5.4 指针和一维数组间的关系——等价的函数参数

【例14.1】演示数组和指针变量作函数参数

```
3    void InputArray(int a[], int n)
4    {
5       int  i;
6       for (i=0; i<n; i++)
7       {
8           scanf("%d", &a[i]);
9       }
10   }
```

被调函数的形参声明为数组类型，用下标法访问数组元素

```
11   void OutputArray(int a[], int n)
12   {
13      int  i;
14      for (i=0; i<n; i++)
15      {
16          printf("%4d", a[i]);
17      }
18      printf("\n");
19   }
```

7.5.4 指针和一维数组间的关系——等价的函数参数

【例14.2】演示数组和指针变量作函数参数

```
3    void InputArray(int *pa, int n)
4    {
5        int i;
6        for (i=0; i<n; i++, pa++)
7        {
8            scanf("%d", pa);
9        }
10   }
```

被调函数的形参声明为指针类型，用指针法访问数组元素

```
11   void OutputArray(int *pa, int n)
12   {
13       int i;
14       for (i=0; i<n; i++, pa++)
15       {
16           printf("%4d", *pa);
17       }
18       printf("\n");
19   }
```

7.5.4 指针和一维数组间的关系——等价的函数参数

【例14.3】演示数组和指针变量作函数参数

```
1    #include <stdio.h>
2    int  main()
3    {
4       int  a[5];
5       printf("Input five numbers:");
6       InputArray(a, 5);
7       OutputArray(a, 5);
8       return 0;
9    }
```

在主函数中这样做
没有实际意义

```
1    #include <stdio.h>
2    int  main()
3    {
4       int  a[5];
5       int *p = a;
6       printf("Input five numbers:");
7       InputArray(p, 5);
8       OutputArray(p, 5);
9       return 0;
10   }
```

本讲小结

- 指针的概念和变量的地址
- 指针变量的定义和初始化
- 取地址和间接寻址运算符
- 指针作函数参数
- 指针与一维数组