南京大学 计算机科学与技术系
Department of Computer Science & Technology, NJU

Chapter 6

指针及其运用

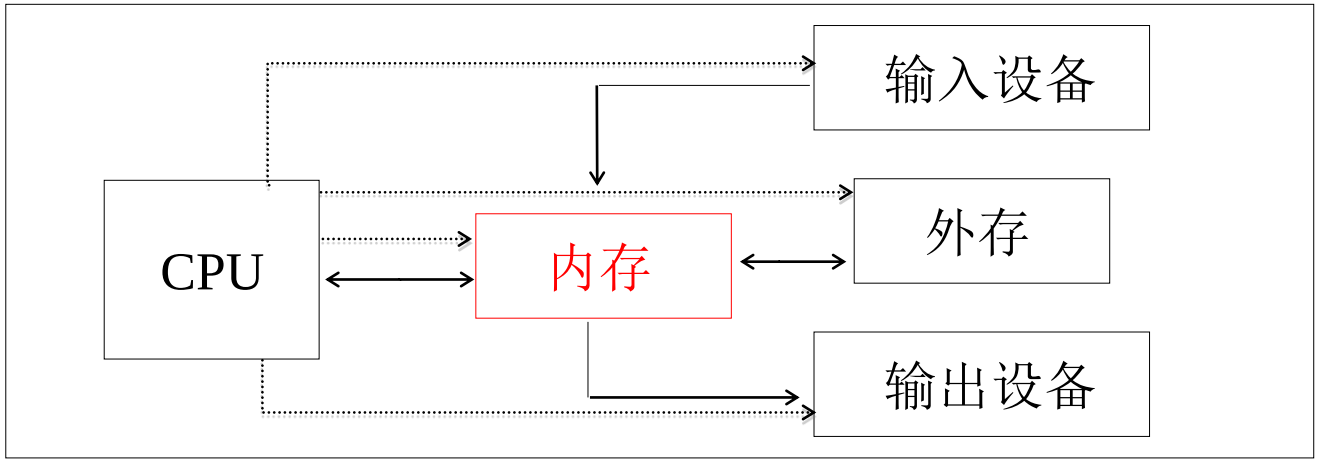
刘奇志

指针及其运用

- 指针的基本概念
  - 指针类型的构造
  - 指针变量的定义与初始化
  - 指针的基本操作
- 用指针操纵数组
- **用指针在函数间传递数据**
- **用指针访问动态变量**
- 用指针操纵函数*

回顾

- **冯诺依曼体系结构示意图**

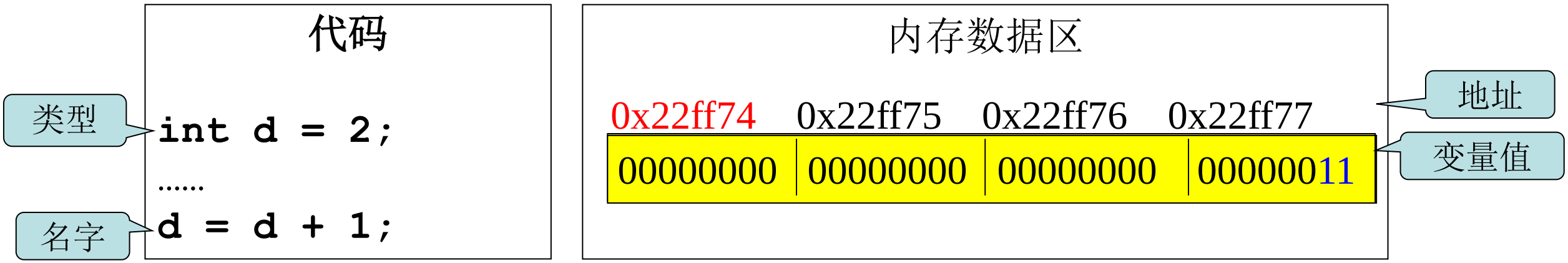


虚线箭头表示控制流

实线箭头表示数据流

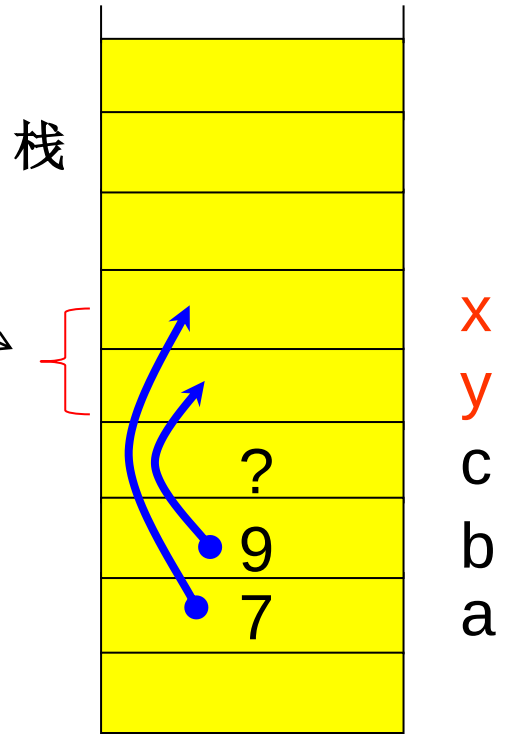
变量的属性

- 变量名
- 变量的类型：决定变量所占用内存单元的个数
- 变量的地址：每个变量实际占用的若干个内存单元中第一个内存单元的地址（即首地址）。
- 变量的值
- …



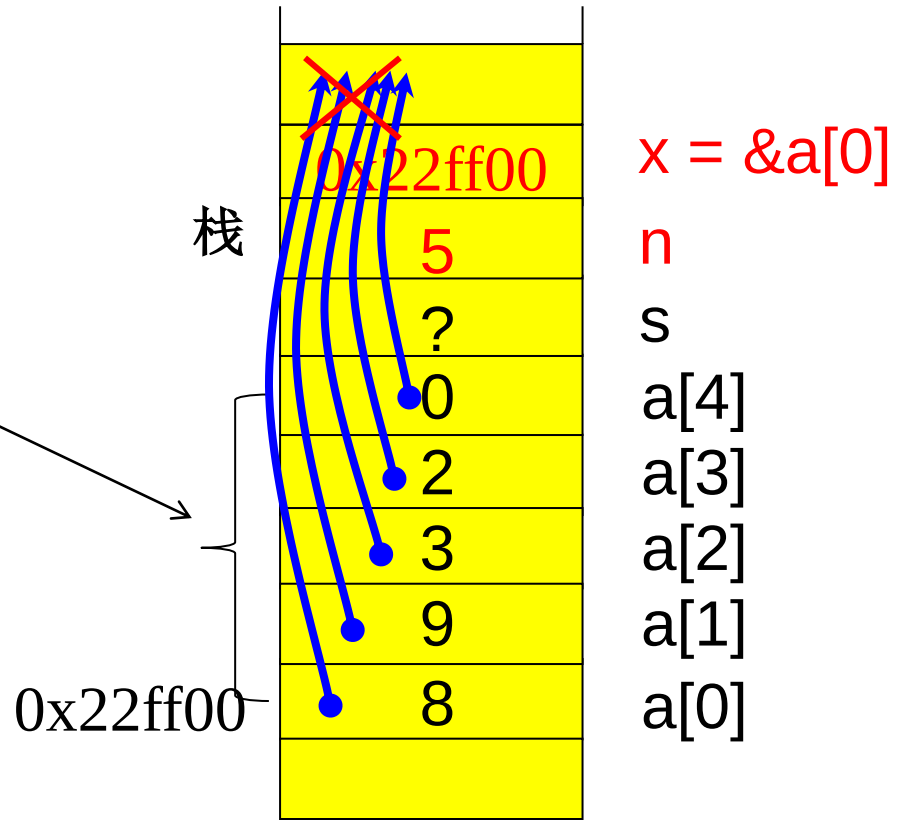
C函数调用的过程

```
int Sum(int x, int y)
{
    return x + y;
}

int main()
{   int a, b, c;
    scanf("%d%d",&a, &b);
    c = Sum(a, b);
    printf("%d", c);
    return 0;
}
```



C函数调用的过程

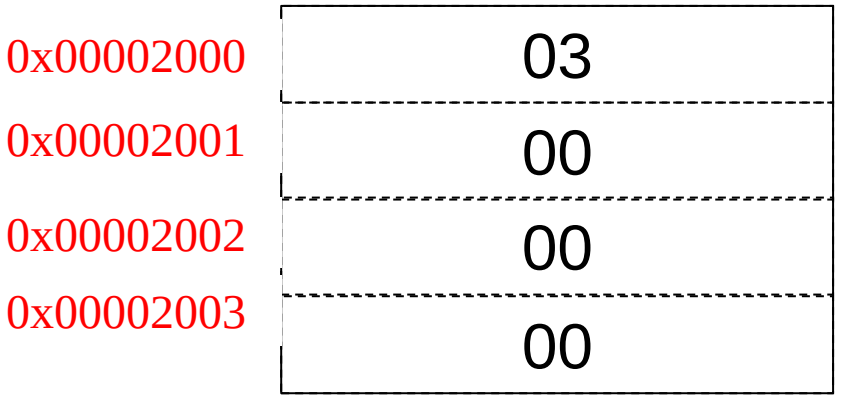
```
int Sum(int x[ ], int n)
{   int s = 0;
    for(int i = 0; i < n; ++i)
        s += x[i];
    return s;
}

int main()
{   int a[5] = {8,9,3,2,0};
    int s = Sum(a, 5);
    printf("%d", s);
    return 0;
}
```



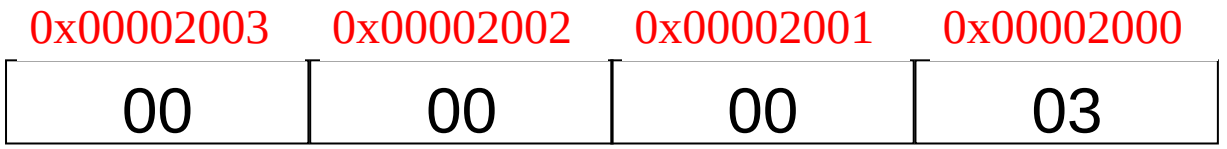
关于内存单元示意图的说明

1个字节（1byte，8bit)

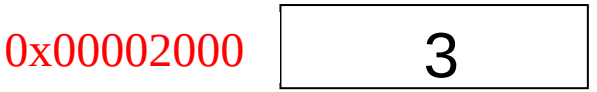
4个字节（4byte， 32bit)



课件里有时压缩成：



或：



每一个内存单元都有一个地址，程序或人在描述数据时，一般用十六进制或十进制一般用十六进制数描述，开头的00可以省略，写作0x3000

地址

- **特殊的整数**

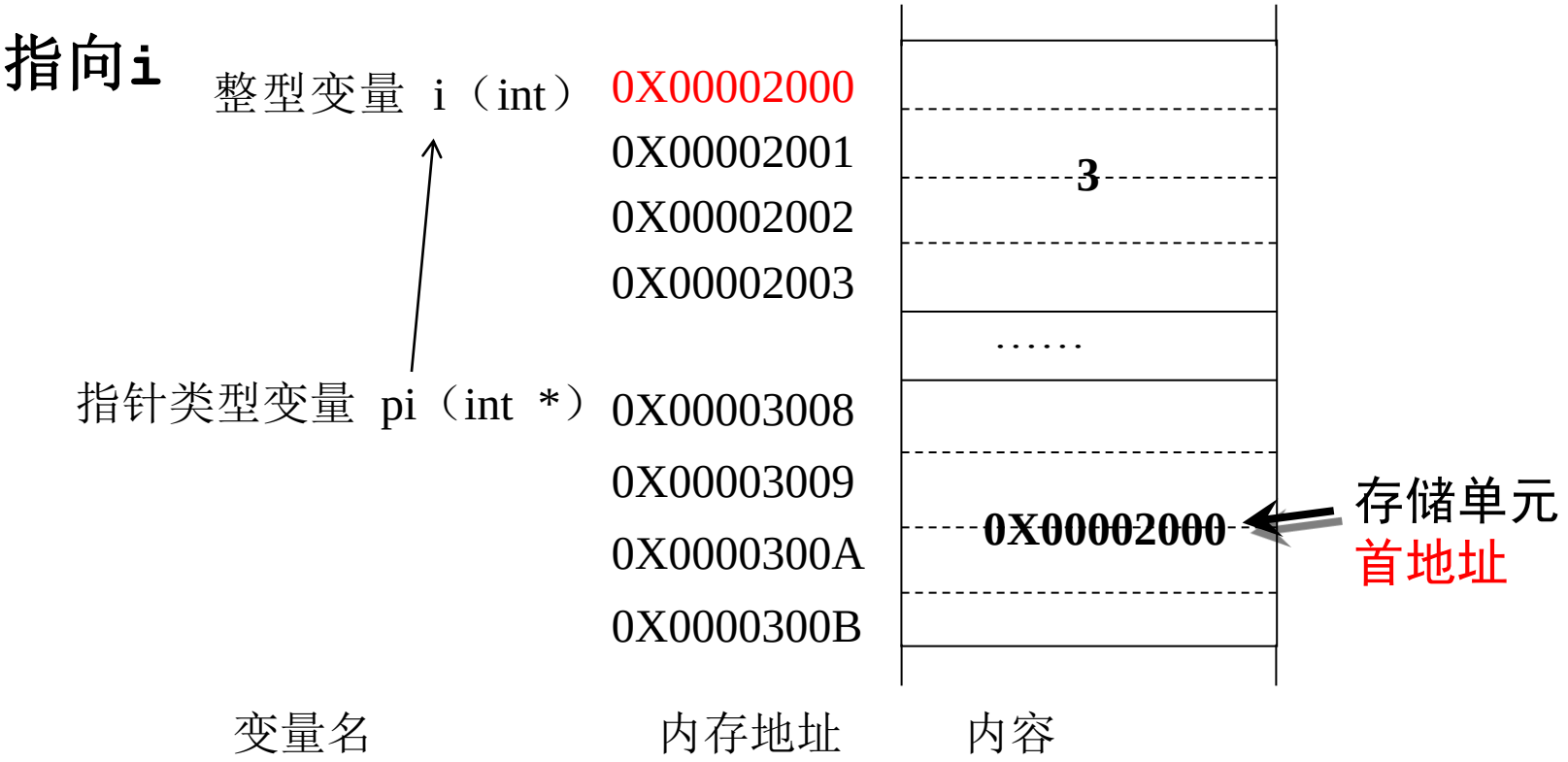
  - 计算机的内存可看作由一系列内存单元组成，内存单元中可以存储不同的内容。每个内存单元（容量为1个字节）由一个特殊的整数（即地址）进行标识。

  - C语言用指针类型描述地址，通过对指针类型数据的相关操作，可以实现与地址有关的数据访问功能。
    - **指针类型数据通常占用1个字空间（与int型数据占用相等大小的空间）**
    - **值集、操作集与 int 不同**

- **C语言的指针类型变量（简称指针变量或指针，pointer）可以表示和存储另一变量的地址。**
  - “指针”的含义未被严格界定，可能是指指针变量，也可能是指地址或指针类型

```
int  i = 3;
int  *pi = &i;  //pi指向i
```

基类型



- 变量的地址与变量名之间存在映射关系，一般情况下，高级语言程序通过这种映射关系用变量名访问数据。
- 如果在程序中知道其他函数中变量的地址，则可以省略数据传输环节，从而提高数据的访问效率。
- 在没有变量名的情况下，这种访问方式更具意义。
- 不过，不是所有内存单元的地址都能在任一程序中使用。一般地，一个程序只能使用执行环境在编译和执行期间分配给该程序的内存单元的地址。而且地址能参与的操作也很有限。所以，引入指针类型。

指针类型的构造

- 指针类型由一个表示变量类型（在这里又叫基类型）的关键字和一个*构造而成。
- 可以给构造好的指针类型取一个别名，作为指针类型标识符。
- 比如，

```
typedef int * Pointer;
// Pointer是类型标识符
//其值集为本程序中 int 型变量的地址
```

指针变量的定义与初始化

- 可以用构造好的指针类型来定义指针变量。
  - 比如，

```
typedef int * Pointer;
Pointer p1, p2;  //定义了两个同类型的指针变量p1、p2
```

- 也可以在构造指针类型的同时直接定义指针变量。
  - 比如，

```
int *p;
//int和*构造了一个指针类型，同时定义了一个指针变量p
//指针变量的变量名不包括*
```

- 多个定义有时可以合并写，但不主张合并写。
  - 比如，

```
int m = 3, n = 5, *px;     或     int *px, m = 3, n = 5;
```

- **指针变量实际存储的是哪一个内存单元的地址，可以通过初始化来指定。**
- **用来初始化指针变量的变量要预先定义，且类型与指针变量的基类型要一致，初始化后称指针变量指向该变量。**
- **指针变量的初始化通常是在函数调用过程中完成的，即实际参数的地址值传递给形式参数的指针变量。**下面先用简单的直接初始化例子示意这一过程中的注意事项。
  - 比如，

```
int i = 0;
int *pi = &i;     //取出变量i的地址，用来初始化指针变量pi，即pi指向i
```

  - 而

```
double f = 3.2;
double *pf = &f;        //不可以 double *pf = &i;
```

- **也可以用另一个指针变量或指针常量来初始化一个指针变量。**
  - 比如，

```
int i = 0;
int *pi = &i;
int *pj = pi;    //pj也指向变量i
```

  - 又比如，

```
int a[10];
int *pa = a;     //pa指向数组a
```

- **还可以用0来初始化一个指针变量，表示该指针变量暂时不指向任何变量。**
  - 比如，

```
int *pv = 0;     //0是一个空地址
```

- **不可以将一个非0整数赋给一个指针变量，因为一个事先确定的地址不一定是系统分配给该程序的内存单元地址，不一定能在该程序中访问。**
  - 比如，

```
int *pn = 2000;
//编译器不一定会报错，但执行时可能会引起系统故障
```

- 初始化之后的指针变量，可以用来访问数据。指针变量的值一般是某个数据的地址，用指针变量访问数据时，地址决定访问的起点，基类型决定访问的终点。可见，指针变量是通过指定起点和终点来访问某个数据的，这是指针类型与基本类型的不同之处。

指针类型相关的基本操作

- **取值操作**
- 赋值操作
- 关系/逻辑操作
- 加/减一个整数
- 减法操作
- 下标操作

- **这些操作只在一定的约束条件下才有意义。不能参与的操作是没有意义的操作，不一定会出语法错误，但可能会造成难以预料的结果（例如，将两个指针类型数据相加或乘/除法运算，结果不一定是有效的内存地址，或者即使地址有效，其对应的内存单元未必能被该程序访问）。**

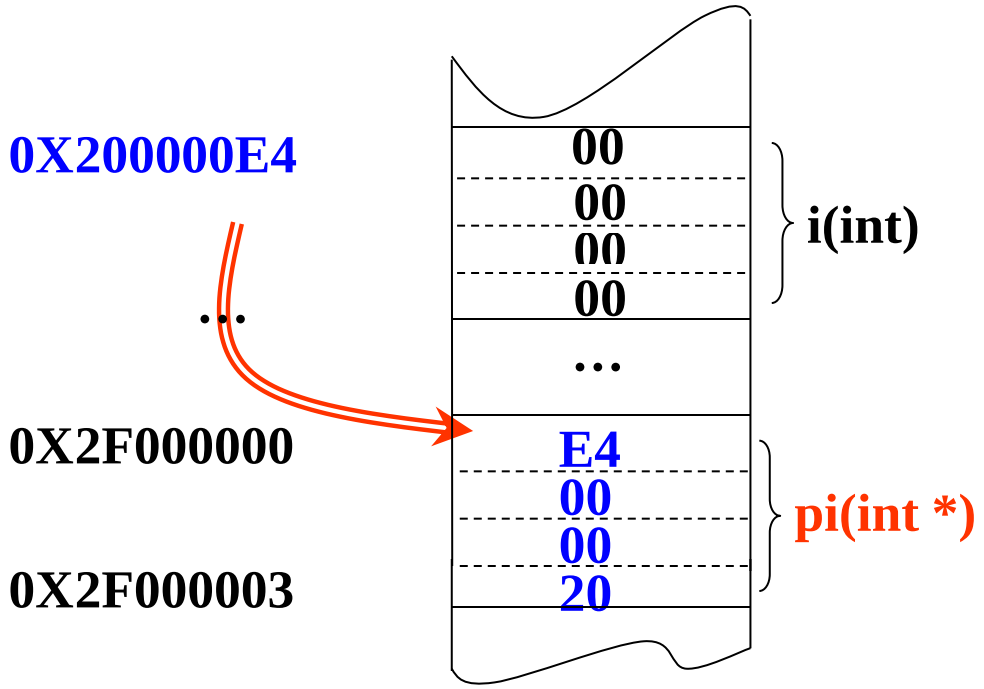
- 取地址操作
  - 用地址操作符（Address Operator）& 获取操作数的地址。
  - & 是一个单目操作符，优先级较高（2级），结合性为自右向左。
  - 其操作数应为各种类型的内存变量、数组元素等，不能是表达式、字面常量、寄存器变量。

- **取值操作**
  - 用指针操作符（Indirection Operator）* 获得指针变量指向的内存数据。
  - * 是一个单目操作符，优先级较高（2级），结合性为自右向左。
  - 其操作数应为地址类型的数据。
  - 注意，构造指针类型或定义指针变量时的 * 不是指针操作符。

- **取值操作与取地址操作是一对逆操作。**

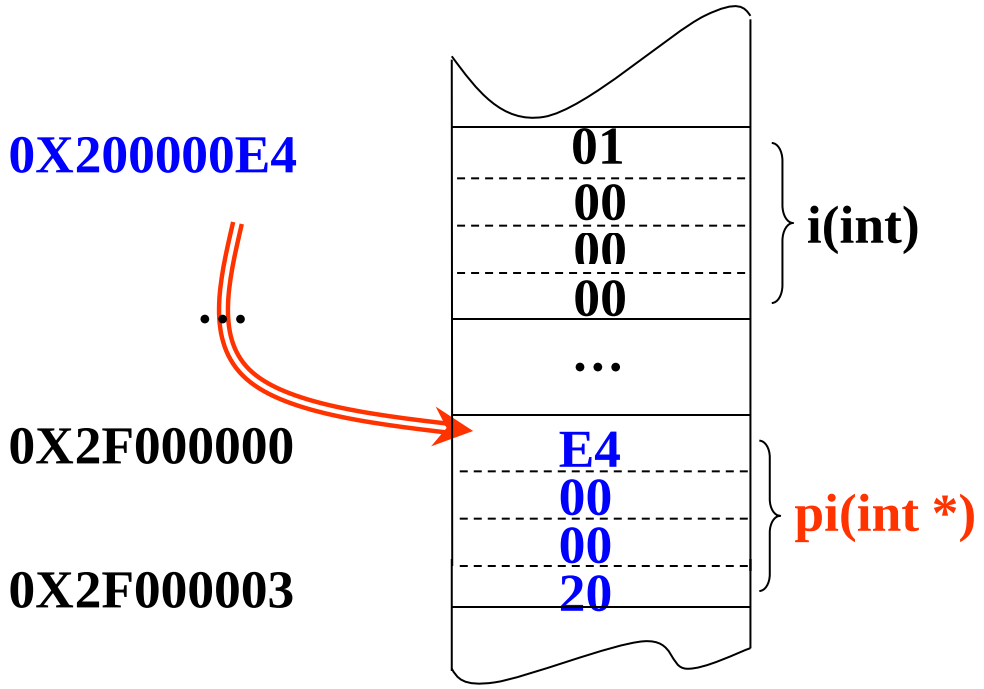
- 比如，

```
int i = 0;
int *pi = &i;     //pi指向i，这里的 * 不是指针操作符
```



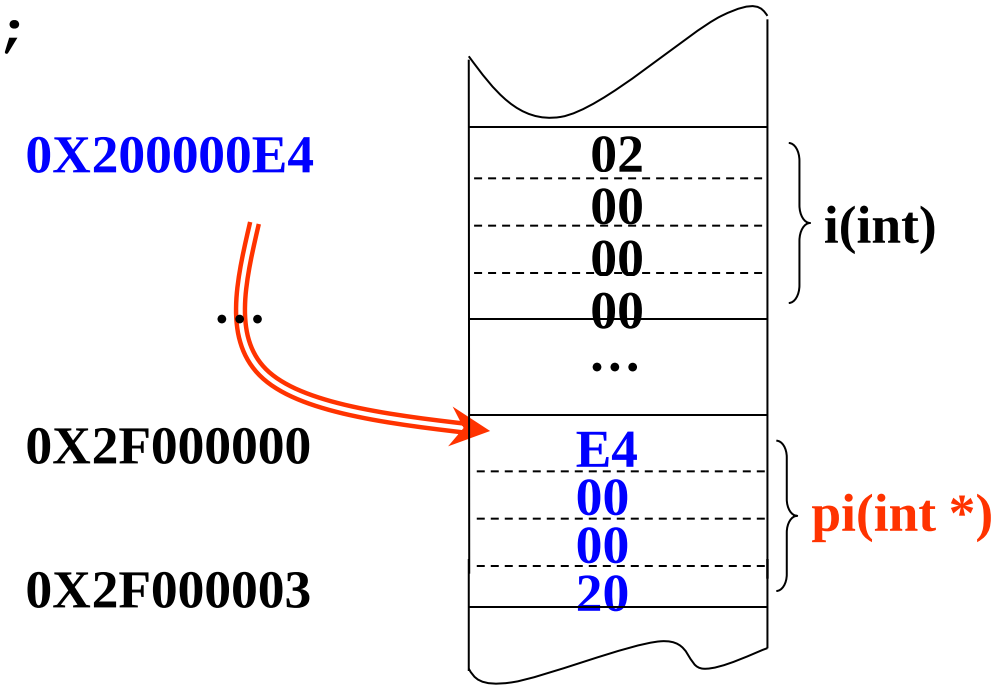
- 比如，

```
int i = 0;
int *pi = &i;     //pi指向i，这里的 * 不是指针操作符

*pi = 1;          //对pi进行取值操作并对其赋值，相当于i = 1;
```



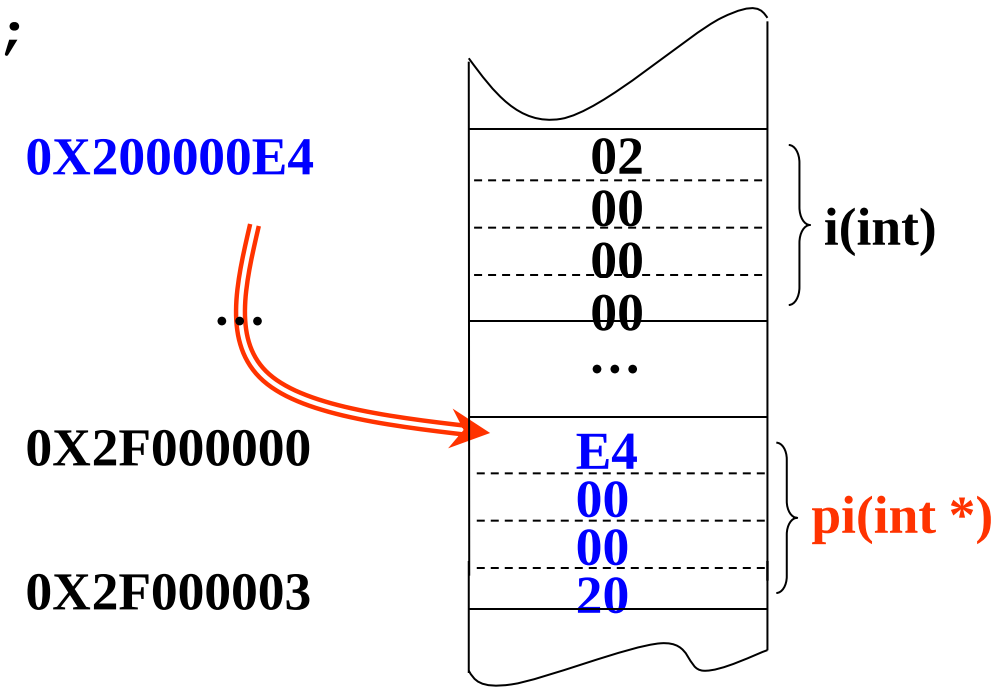
- 比如，

```
int i = 0;
int *pi = &i;      //pi指向i，这里的 * 不是指针操作符

*pi = 1;           //对pi进行取值操作并对其赋值，相当于i = 1;
*(&i) = 2;         //相当于i = 2;
```



- 比如，

```
int i = 0;
int *pi = &i;      //pi指向i，这里的 * 不是指针操作符

*pi = 1;           //对pi进行取值操作并对其赋值，相当于i = 1;
*(&i) = 2;         //相当于i = 2;
pi = &(*pi);       //pi仍然指向i
```



指针类型数据的赋值操作

- 指针变量除初始化外，还可以通过赋值操作指定一个地址，使之指向某一个变量。
  - 比如，

```
int i = 0;
int *pi = &i; //初始化，pi指向i
……
int x = 3;
……
pi = &x; //赋值，pi指向x
*pi = 2; //改变x的值
```

- 对指针变量进行赋值操作的右操作数

  也应为地址类型的数据，
  还可以为0，
  不可以是非0整数，

  右操作数中的变量要预先定义，其类型要与左边指针变量的基类型一致，
  若是指针变量，其基类型要与左边指针变量的基类型一致，
  必要时可小心使用强制类型转换。

- **指针变量必须先初始化或赋值，然后才能进行其他操作，否则其所存储的地址是不确定的，对它的操作会引起不确定的错误。**

```
int *p;
p++;   ?
```

p没有初始化或赋值就使用，警告

```
int i;
int b[3];
int *pi,*pb,*pj,*pv;
pi = &i;
pb = b;
pj = pi;
pv = 0;
```

将首地址赋给指针变量

指针类型数据的加/减一个整数操作

- 一个指针类型的数据加/减一个整数，可以使其成为另一个地址，前提是操作结果仍然是一个有效的内存单元地址。比如，操作前指针变量指向某数组的一个元素，操作后的结果指向该数组的另一个元素（不能超出数组范围）。

- 注意，加/减一个整数n后的结果并非在原来地址值的基础上加/减n，而是n的若干倍，具体倍数由基类型决定，即加/减n*sizeof(基类型)。
  - 比如，

```
int i = 0;
int *pi = &i;
pi++;      //设int型数据占4个字节空间，
           //则pi的地址值实际增加了4，而不是1
```

eg.
0x22ff40
0x22ff41?

两个相同类型的指针类型数据的相减操作

- 两个相同类型的指针类型数据的相减操作，结果为两个地址之间可存储基类型数据的个数。通常用来计算某数组两个元素之间的偏移量。
  - 比如，

```
int *pi, *pj;
……
int offset = pj - pi;
//offset为pj与pi之间可存储int型数据的个数
```

eg.
0x22ff40
0x22ff44
4?

指针类型数据的关系/逻辑操作

- 两个指针类型的数据可以进行比较，以判断在内存的位置前后关系，前提是它们表示同一组数据的地址。
- 比如，
  - 两个指针变量都指向某数组中的元素，用关系操作比较这两个地址在数组中的前后位置关系。
  - 也可以判断一个地址是否为0，以明确该地址是否为某个实际内存单元的地址。

指向一维数组元素的指针变量

- 由于一维数组名表示第一个元素的地址，所以可将一维数组名赋给一个一级指针变量，此时称该指针变量指向这个数组的元素。
  - 比如，

```
int a[5];
int *pa = a;      //相当于int *pa = &a[0];
```

  - 当一个指针变量指向一个数组的元素时，该指针变量可以存储数组的任何一个元素的地址，即pa先存储a[0]的地址，不妨设为0x00002000(简作2000)，然后，pa的值可以变化为2004，2008，200C，2010，于是可以通过pa来访问数组a的各个元素。

- **访问方法有三种：**
  - 下标法：通过下标操作指定元素；
  - 指针移动法：通过自加/减一个整数操作指定元素地址；
  - 偏移量法：通过取值操作和加/减一个整数操作指定元素。

- **通过指针变量操纵数组时，要注意防止下标越界。**

- 例6.1 用指针变量操纵数组。

…

```
#define N 10
int main( )
{ int i;
  int a[N] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int *pa;
  for(pa = a, i = 0; i < N; ++i)
      printf("%d ", pa[i]);    //下标法，用a[i]也行
  for( ; pa < (a + N); ++pa)   //指针移动法，用++a不行（思考原因）
      printf("%d ", *pa);
  for(pa = a, i = 0; i < N; ++i)//这里没有“pa = a, ”，则会出现越界!
      printf("%d ", *(pa + i));//偏移量法，用*(a + i)也行
  return 0; }
```

例6.2 指针变量作为函数形参。

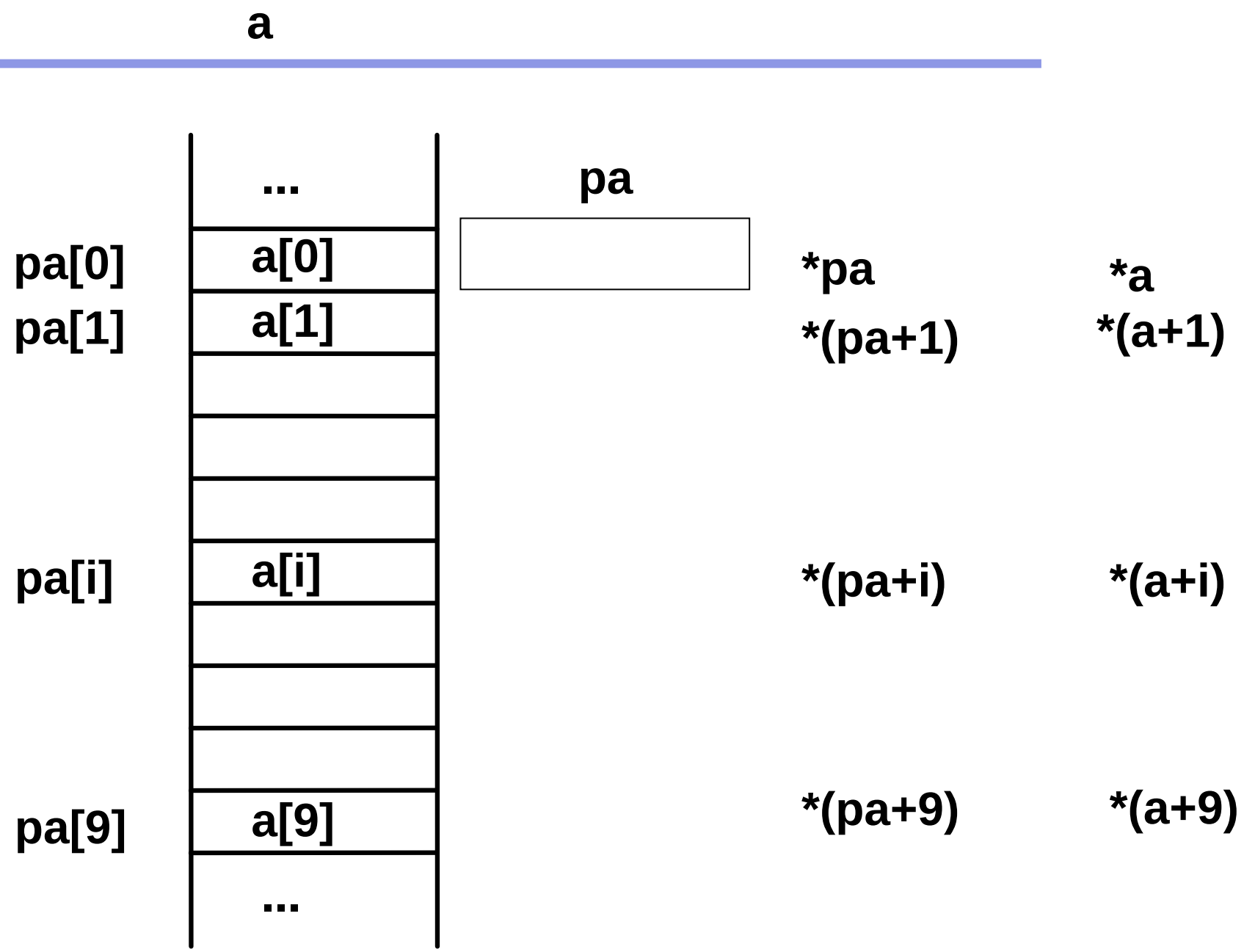
…

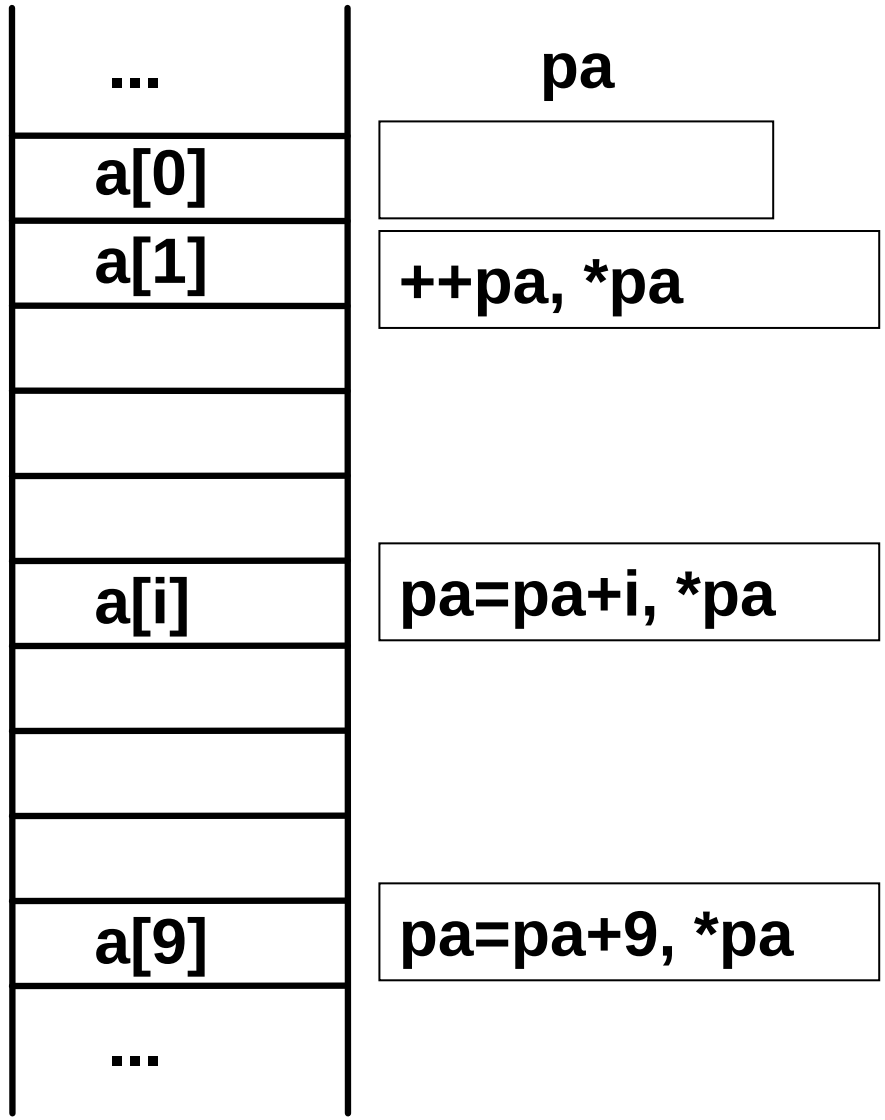
```
#define N 10
int Fun(int *pa, int n);
int main( )
{ int a[N] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
  int sum = Fun(a, N);    // int *pa = a;
  printf("%d \n", sum);
  return 0;
}
```

```
int Fun(int pa[], int n
```

```
 int Fun(int *pa, int n)
{    int s = 0;
     for(int i = 0; i < n; ++i)
          s += *(pa + i);
     return s;
}
```

```
s += pa[i];
```


a
...
pa
pa[0]
a[0]
*pa
*a
pa[1]
a[1]
*(pa+1)
*(a+1)
pa[i]
a[i]
*(pa+i)
*(a+i)
pa[9]
a[9]
*(pa+9)
*(a+9)
...


a
...
pa
a[0]
a[1]
++pa, *pa
a[i]
pa=pa+i, *pa
a[9]
pa=pa+9, *pa
...

- 指向数组元素的指针变量 vs. 数组名
  - 数组名可以代表第一个元素的地址，是地址常量
    - 数组名的值不能被修改
  - 指向数组元素的指针变量
    - 用来存放一个数组各个元素的地址
    - 一般情况下，这个变量的初始值为数组名代表的地址
    - 经过操作，其值可能会是其他元素的地址

Thanks !