程序设计
Programming

Lecture 7: 指针

1、指针的概念及基本运算

变量与内存地址

- C编译器会根据变量的数据类型，分配相应长度的内存单元。
- 程序执行时通过变量名找到内存地址，直接对内存单元进行操作，即“直接访问”
- 也就是说，变量所在的内存地址直接存放了该变量的值

变量与内存地址

- C编译器会根据变量的数据类型，分配相应长度的内存单元。
- 程序执行时通过变量名找到内存地址，直接对内存单元进行操作，即“直接访问”
- 也就是说，变量所在的内存地址直接存放了该变量的值

int a = 1234;

整型变量a的内存单元

指针与内存地址

- 指针是指向一个内存地址的变量（称为“指针变量”）
  - ✓指针本身是一种数据类型
  - ✓指针类型的变量存放了其他变量的地址，即“指向其他变量”
  - ✓可以通过调用指针来操作其他变量，即所谓“间接访问”

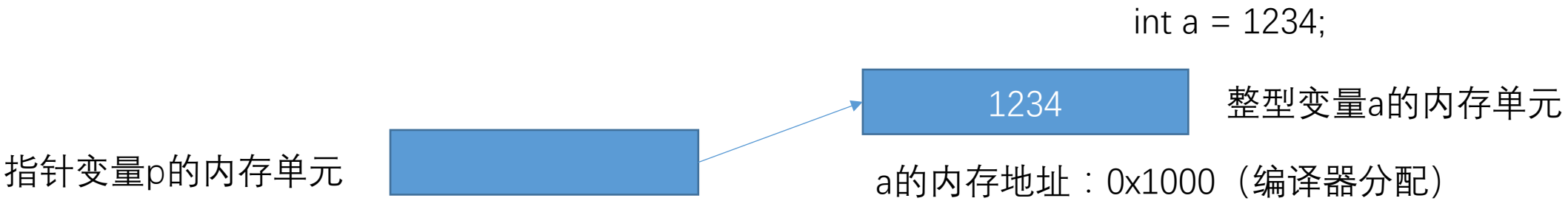
可以简单理解为，普通变量存放了变量的值，而指针变量存放了普通变量的地址

int a = 1234;

整型变量a的内存单元

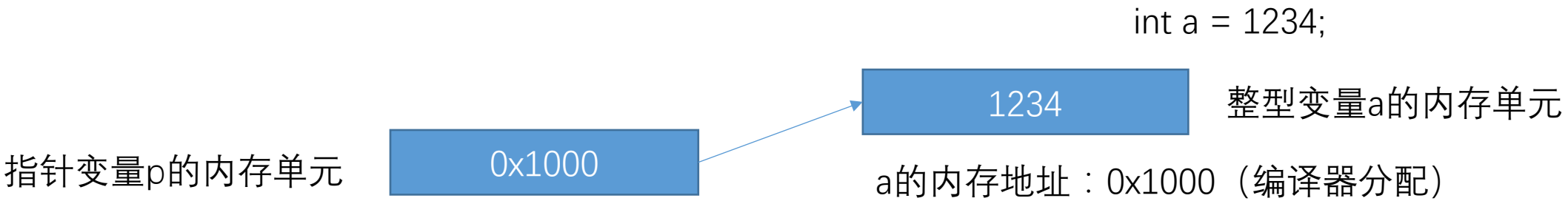
指针与内存地址

- 指针是指向一个内存地址的变量（称为“指针变量”）
  - ✓指针本身是一种数据类型
  - ✓指针类型的变量存放了其他变量的地址，即“指向其他变量”
  - ✓可以通过调用指针来操作其他变量，即所谓“间接访问”
- 例：假设int型变量a内存地址（内存开始地址）为0x1000，如果指针变量p指向a，那么p的内存单元中就存放了a的内存地址



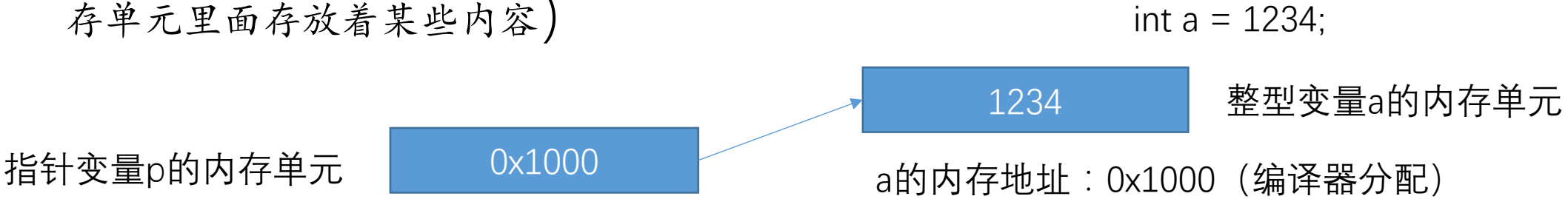
指针与内存地址

- 指针是指向一个内存地址的变量（称为“指针变量”）
  - ✓指针本身是一种数据类型
  - ✓指针类型的变量存放了其他变量的地址，即“指向其他变量”
  - ✓可以通过调用指针来操作其他变量，即所谓“间接访问”
- 例：假设int型变量a内存地址（内存开始地址）为0x1000，如果指针变量p指向a，那么p的内存单元中就存放了a的内存地址，即0x1000。



指针与内存地址

- 指针是指向一个内存地址的变量（称为“指针变量”）
  - ✓指针本身是一种数据类型
  - ✓指针类型的变量存放了其他变量的地址，即“指向其他变量”
  - ✓可以通过调用指针来操作其他变量，即所谓“间接访问”
- 例：假设int型变量a内存地址（内存开始地址）为0x1000，如果指针变量p指向a，那么p的内存单元中就存放了a的内存地址，即0x1000。（注意内存地址和内存内容的区别：每一个内存单元有自己的地址，同时内存单元里面存放着某些内容）



指针变量的定义

类型名 *指针变量名

int *p

此时p不指向任何变量， p是一个空指针，即p==NULL

- p就是一个能够指向int型变量的指针变量
  - ✓*用来表示p是指针类型
  - ✓int用来表示p指向int型变量

指针变量的定义

类型名 *指针变量名

int *p

此时p不指向任何变量， p是一个空指针，即p==NULL

指针变量声明一般须明确指向何种类型，否则无法确定所指向变量的内存长度！

指针变量的定义

类型名 *指针变量名

int *p

此时p不指向任何变量， p是一个空指针，即p==NULL

指针变量声明一般须明确指向何种类型，否则无法确定所指向变量的内存长度！

指针变量本身占据的内存长度和指向变量的类型无关！

指针变量的初始化

类型名 *指针变量名 = 相应类型变量的内存地址或相应类型其他指针变量

```
int a = 10;
int *p = &a;
int *q = p;
```

- &为取地址符号
- &a，即取整型变量a的内存（开始）地址

指针变量的初始化

类型名 *指针变量名 = 相应类型变量的内存地址或相应类型其他指针变量

```
int a = 10;
int *p = &a;
int *q = p;
```

- &为取地址符号
- &a，即取整型变量a的内存（开始）地址
- scanf(“%d” , &a)

指针变量的赋值和运算

- 假设p是一个int型指针变量
  - p=&a;（假设a是一个普通变量）
  - p=q; （假设q也是一个指针变量）
  - p=NULL;

指针变量的赋值和运算

- 假设p是一个int型指针变量
  - p=&a;（假设a是一个普通变量）
  - p=q;（假设q也是一个指针变量）
  - p=NULL;

注意，声明int *p之后
- ✓ p用来表示指针变量
- ✓ *p用来表示p指向的变量的值

- 取地址运算符&和间接访问运算符*
  - ✓&a取变量a的内存地址：int *p=&a;
  - ✓单独使用*p时，表示取指针变量p所指向内存地址的值：*p 等价于 a（注意两个*的不同含义，变量声明的时候*为指针类型符号，使用变量的时候*为间接访问运算符）

指针变量的赋值和运算

- 假设p是一个int型指针变量
  - p=&a;（假设a是一个普通变量）
  - p=q;（假设q也是一个指针变量）
  - p=NULL;

注意，声明int *p之后
- ✓ p用来表示指针变量
- ✓ *p用来表示p指向的变量的值

- 取地址运算符&和间接访问运算符*
  - ✓&a取变量a的内存地址：int *p=&a;
  - ✓单独使用*p时，表示取指针变量p所指向内存地址的值：*p 等价于 a（注意两个*的不同含义，变量声明的时候*为指针类型符号，使用变量的时候*为间接访问运算符）
  - ✓&(*p)？ *(&p)？

指针作为函数的参数

```
int swap1(int a, int b)
{
    int c = a;
    a = b;
    b = c;
    return 0;
}
```

```
int swap2(int *pa, int *pb)
{
    int c = *pa;
    *pa = *pb;
    *pb = c;
    return 0;
}
```

指针作为函数的参数

```
int swap1(i                                    nt *pa, int *pb)
{
    int c =                                     *pa;
    a = b;                                     pb;
    b = c;                                     ;
    return                                     0;
}
```

```
int main()
{
    int a = 10, b = 20;
    int *pa = &a, *pb = &b;
    swap1(a, b);
    printf("\n%d %d\n", a, b);
    swap2(pa, pb);
    printf("\n%d %d\n\n", a, b);
    return 0;

}
```

指针作为函数的参数

```
int swap1(i                                    nt *pa, int *pb)
{
    int c =                                     *pa;
    a = b;                                     pb;
    b = c;                                     ;
}
```

```
int main()
{
    int a = 10, b = 20;
    int *pa = &a, *pb = &b;
    swap1(a, b);
    printf("\n%d %d\n", a, b);
    swap2(pa, pb);
    printf("\n%d %d\n\n", a, b);
}
```

可以通过指针形参，改变指针实参所指向变量的值！

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?

假设p是一个指向char型的指针变量，则p+1？

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）
- ++*p？

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）
- ++*p？
  - ✓将p指向变量的值加1

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）
- ++*p？
  - ✓将p指向变量的值加1
- *p++？

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）
- ++*p？
  - ✓将p指向变量的值加1
- *p++？
  - ✓将p指向下一个int型长度的地址，再间接访问（为何不是先间接访问？）

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）
- ++*p？
  - ✓将p指向变量的值加1
- *p++？
  - ✓将p指向下一个int型长度的地址，再间接访问（为何不是先间接访问？）
  - ✓p值？表达式的值？

指针变量的运算

- 假设p是一个int型的指针变量
- 则p+1指向内存中下一个int型变量“占据长度”的开始地址
  - ✓即，如果int型变量占据4个字节，则p+1指向下一个4字节的第一个字节
  - ✓p++? p--?
- *p+1？
  - ✓将p指向变量的值加1（为何不是指向下一个int型变量，再取值？）
- ++*p？
  - ✓将p指向变量的值加1
- *p++？
  - ✓将p指向下一个int型长度的地址，再间接访问（为何不是先间接访问？）
  - ✓p值？表达式的值？p指向下一个地址，表达式值为p自增前指向内存地址的值

指针变量的运算

- 假设p是一个int型指针变量
  - ✓*++p? ++*p?

- 假设q也是一个int型指针变量
  - ✓p-q表示两个指针所指内存地址的距离（数据类型为long int）
  - ✓该距离以int型变量的长度为单位，即相隔多少个int型变量

指针变量的运算

- 假设p是一个int型指针变量
  - ✓*++p? ++*p?

- 假设q也是一个int型指针变量
  - ✓p-q表示两个指针所指内存地址的距离（数据类型为long int）
  - ✓该距离以int型变量的长度为单位，即相隔多少个int型变量
  - ✓如果p和q都是char型？

指针变量的运算

- 假设p是一个int型指针变量
  - ✓*++p? ++*p?
- 假设q也是一个int型指针变量
  - ✓p-q表示两个指针所指内存地址的距离（数据类型为long int）
  - ✓该距离以int型变量的长度为单位，即相隔多少个int型变量
  - ✓如果p和q都是char型？
- p+q?

2、指针与数组

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0] ）

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0] ）
  - ✓a+1等价于

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0] ）
  - ✓a+1等价于&a[1]，a+k等价于

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0] ）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0] ）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)= ？

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)=a[2]

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)=a[2]，*(a+k)=?

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)=a[2]，*(a+k)=a[k]

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)=a[2]，*(a+k)=a[k]
  - ✓int *p=&a[1], *q = &a[3]; p-q=?

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)=a[2]，*(a+k)=a[k]
  - ✓int *p=&a[1], *q = &a[3]; p-q=-2

指针与数组

- 数组名可以用来表示数组的内存开始地址
  - ✓int a [100]，那么a的值就是数组a的内存开始地址，等价于 &a[0]（a== &a[0]）
  - ✓a+1等价于&a[1]，a+k等价于&a[k]
  - ✓注意a是一个常量（固定地址），不能被赋值，即 a=&b 或 a=a+1 是非法的
- int *p = a, p就指向数组a的内存开始地址，即p等于a或&a[0]
  - ✓p+1指向a[1], p+k指向a[k]
  - ✓注意区别：p是一个指针变量（可以被赋值），a是一个指针常量（不可以被赋值），例如，p=p+1是合法的
  - ✓*(p+2)=a[2]，*(a+k)=a[k]
  - ✓int *p=&a[1], *q = &a[3]; p-q=-2
  - ✓使用指针对数组进行循环操作

```
for (int *p = a; p != a+n; p++)
```

使用指针遍历数组

```c
int main()
{
    int a[100];
    int n = 10;
    for(int i = 0; i < n; i++)
        a[i] = i+1;
    int *p = a;
    int sum = 0;
    for(; p < a+n; p++)
        sum += *p;
    printf("\n%d\n\n", sum);
    return 0;

}
```

3、字符串、字符数组和字符指针

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以‘\0’结束
  - ✓char str[] = “hello”；

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”；

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”；

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

- 同理，也可以用字符指针变量指向一个字符串

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”；

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

- 同理，也可以用字符指针变量指向一个字符串
  - ✓字符串常量的值，即为其首字符的地址（ ‘h’ 的内存地址即为 “hello” 的值）

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”；

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

- 同理，也可以用字符指针变量指向一个字符串
  - ✓字符串常量的值，即为其首字符的地址（ ‘h’ 的内存地址即为 “hello” 的值）
  - ✓char *str_p = “hello”；

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”;

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

- 同理，也可以用字符指针变量指向一个字符串
  - ✓字符串常量的值，即为其首字符的地址（ ‘h’ 的内存地址即为 “hello” 的值）
  - ✓char *str_p = “hello”;
  - ✓str_p是char型指针变量，指向字符 ‘h’ 所在的内存地址（ “hello” 的值或开始地址）

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”；

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

- 同理，也可以用字符指针变量指向一个字符串
  - ✓字符串常量的值，即为其首字符的地址（ ‘h’ 的内存地址即为 “hello” 的值）
  - ✓char *str_p = “hello”；
  - ✓str_p是char型指针变量，指向字符 ‘h’ 所在的内存地址（ “hello” 的值或开始地址）

- 注意str和str_p的区别：str是 “hello” 的开始地址（str的值不能改变），str_p指向 “hello” 的开始地址（str_p可以被重新赋值，指向其他字符串）

字符串、字符数组和字符指针

- 字符串是在内存中连续存放的一串字符，以 ‘\0’ 结束
  - ✓char str[] = “hello”；

  - ✓str是指向char类型的指针常量，str的值就是 ‘h’ 所在内存单元的地址（str == &str[0]）

- 同理，也可以用字符指针变量指向一个字符串
  - ✓字符串常量的值，即为其首字符的地址（ ‘h’ 的内存地址即为 “hello” 的值）
  - ✓char *str_p = “hello”；
  - ✓str_p是char型指针变量，指向字符 ‘h’ 所在的内存地址（ “hello” 的值或开始地址）

- 注意str和str_p的区别：str是 “hello” 的开始地址（str的值不能改变），str_p指向 “hello” 的开始地址（str_p可以被重新赋值，指向其他字符串）
  - ✓str= “hello2”，非法，因为str是常量
  - ✓str_p= “hello2”，合法，因为str_p是变量

字符串、字符数组和字符指针

- 用char str[] = “hello” 和char *str_p = “hello” 定义字符串之后，可以用相似的方式操作其中的字符

```c
int main()
{
    char str[] = "hello";
    char *str_p = "hello";

    for(int i = 0; i < 5; i++)
    {
        printf("%c %c\n", str[i], str_p[i]);
    }
    return 0;
}
```

字符串、字符数组和字符指针

- 用char str[] = “hello” 和char *str_p = “hello” 定义字符串之后，可以用相似的方式操作其中的字符

```c
int main()
{
    char str[] = "hello";
    char *str_p = "hello";

    for(int i = 0; i < 5; i++)
    {
        printf("%c %c\n", str[i], str_p[i]);
    }
    return 0;
}
```

```c
int main()
{
    char str[] = "hello";
    char *str_p = "hello";

    for(int i = 0; i < 5; i++)
    {
        printf("%c %c\n", *(str+i), *(str_p+i));
    }
    return 0;
}
```

字符串输入输出

- printf(), scanf()：以%s作为格式转换说明符

字符串输入输出

- printf(), scanf()：以%s作为格式转换说明符

输出

```
char str[] = "hello";
char *str_p = "hello";
printf("%s %s\n", str, str_p);
```

输入

```
char str[100];
scanf("%s", str);
```

字符串处理函数（#include<string.h>）

- 字符串复制：strcpy(s1, s2)，对字符数组重新赋值，s2的值赋给s1
- s1占据的内存空间要足够容纳s2中的字符串，否则会有内存错误风险

```
int main()
{
    char s1[100];
    char s2[] = "goodbye";
    strcpy(s1, s2);
    printf("%s\n", s1);

    return 0;
}
```

字符串处理函数（#include<string.h>）

- 字符串拼接：strcat(s1, s2)，将s2拼接到字符数组s1尾部
- s1占据的内存空间要足够容纳s1和s2中的字符串，否则会有内存错误风险

```
int main()
{
    char s1[100]="hello ";
    char s2[] = "world!";
    strcat(s1, s2);
    printf("%s\n", s1);

    return 0;
}
```

字符串处理函数（#include<string.h>）

- 字符串比较：strcmp(s1, s2)，比较字符串大小（按字典序）
  - ✓s1比s2小（大），返回负数（正数）；s1和s2相等，返回0

```
int main()
{
    char s1[100]="hello";
    char s2[] = "world!";
    printf("%d\n", strcmp(s1, s2));

    return 0;
}
```

字符串处理函数（#include<string.h>）

- 字符串比较：strcmp(s1, s2)，比较字符串大小（按字典序）
  - ✓s1比s2小（大），返回负数（正数）；s1和s2相等，返回0
  - ✓直接使用运算符：s1 > s2?

```
int main()
{
    char s1[100]="hello";
    char s2[] = "world!";
    printf("%d\n", strcmp(s1, s2));

    return 0;
}
```

字符串处理函数（#include<string.h>）

- 字符串比较：strcmp(s1, s2)，比较字符串大小（按字典序）
  - ✓s1比s2小（大），返回负数（正数）；s1和s2相等，返回0
  - ✓直接使用运算符：s1 > s2? 比较s1和s2两个地址的大小，避免使用！！

```
int main()
{
    char s1[100]="hello";
    char s2[] = "world!";
    printf("%d\n", strcmp(s1, s2));

    return 0;
}
```

字符串处理函数（#include<string.h>）

- 字符串长度：strlen(s)，输出字符串s的长度

```c
int main()
{
    char s[]="hello";
    printf("%ld\n", strlen(s));

    return 0;
}
```

字符串处理函数（#include<string.h>）

- 字符串长度：strlen(s)，输出字符串s的长度
- 注意和字符数组长度的区别！

```c
int main()
{
    char s[]="hello";
    printf("%ld\n", strlen(s));

    return 0;
}
```

```c
int main()
{
    char s[]="hello";
    printf("%ld\n", sizeof(s)/sizeof(char));

    return 0;
}
```

内存动态分配

- 静态分配：int a[100]，编译时分配内存地址

- 动态分配：(int *) malloc (n * sizeof(int))，运行时分配内存地址
  - ✓malloc返回void *，所以要强制转换类型
  - ✓n * sizeof(int) 表示申请能存放n个int型变量的连续内存空间
  - ✓若申请成功则返回起始地址，若申请失败（比如内存不够）则返回NULL

内存动态分配

- 静态分配：int a[100]，编译时分配内存地址

- 动态分配：(int *) malloc (n * sizeof(int))，运行时分配内存地址
  - ✓malloc返回void *，所以要强制转换类型
  - ✓n * sizeof(int) 表示申请能存放n个int型变量的连续内存空间
  - ✓若申请成功则返回起始地址，若申请失败（比如内存不够）则返回NULL

- calloc(n, sizeof(int))：和malloc功能类似，并且将申请到的内存单元全部初始化为0或者NULL
- realloc(p, n * sizeof(int))：重新申请n * sizeof(int)的内存空间，p为原来通过malloc或calloc分配的指针
- free(p)：释放p所指向的动态分配的内存空间（在链表中非常重要!）

内存动态分配

- 动态分配：(int *) malloc (n * sizeof(int))，运行时分配内存地址
  - ✓malloc返回void *，所以要强制转换类型
  - ✓n * sizeof(int) 表示申请能存放n个int型变量的连续内存空间
  - ✓若申请成功则返回起始地址，若申请失败（比如内存不够）则返回NULL

```
int main()
{
    char *s1;
    char s2[] = "goodbye";
    strcpy(s1, s2);
    printf("%s\n", s1);
    return 0;
}
```

内存动态分配

- 动态分配：(int *) malloc (n * sizeof(int))，运行时分配内存地址
  - ✓malloc返回void *，所以要强制转换类型
  - ✓n * sizeof(int) 表示申请能存放n个int型变量的连续内存空间
  - ✓若申请成功则返回起始地址，若申请失败（比如内存不够）则返回NULL

```c
int main()
{
    char *s1;
    char s2[] = "goodbye";
    strcpy(s1, s2);
    printf("%s\n", s1);
    return 0;
}
```

```c
int main()
{
    char *s1 = (char*)malloc(100*sizeof(char));
    char s2[] = "goodbye";
    strcpy(s1, s2);
    printf("%s\n", s1);
    return 0;
}
```

小结

- 指针与内存地址的关系
  - ✓指针的定义、初始化和运算
- 指针与数组的关系
- 字符串与字符指针
  - ✓字符串处理函数
- 内存动态分配和释放

Next：结构