Pointer

王慧妍

why@nju.edu.cn

南京大学 软件学院 计算机软件研究所

指针

- `scanf("%d", &a);`
  - &a: 取地址
  - 指针就是一个存储内存地址的变量，代表指向某个具体的内存地址

- 本学期最费解的内容来了

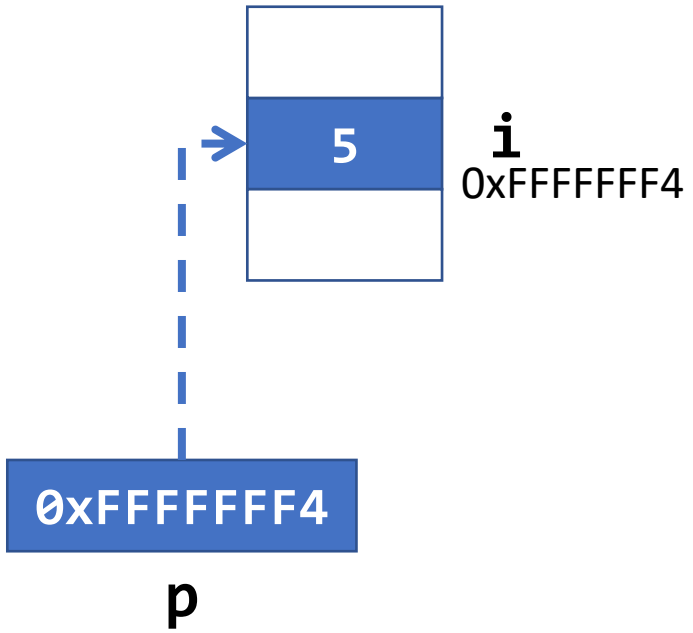
回顾一下

- sizeof
- scanf
- printf(“%p”, &n);

指针

- “A *pointer* is a *variable* that contains the *address* of a variable.”
  - 一个保存内存地址的变量，代表指向某个具体的内存地址
  - `int i = 5;`
  - `int* p;`
  - `p = &i;`

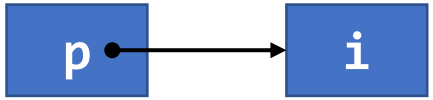
  “p是指向i的指针”
  → “p存储了i的内存地址”



- 指针变量的值
  - 是具有实际值的变量的地址
  - 而普通变量的值是实际的值

指针也是一个变量

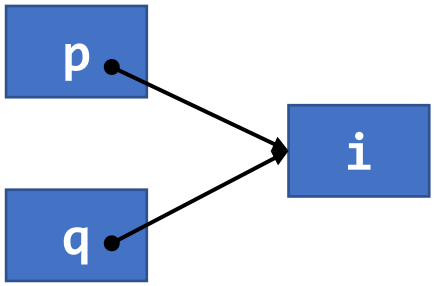
- 指针的声明与定义
  - `int *p = &n;`
    - “pointer to int”
  - `int *p, q; //?`
  - `int* p, q; //?`



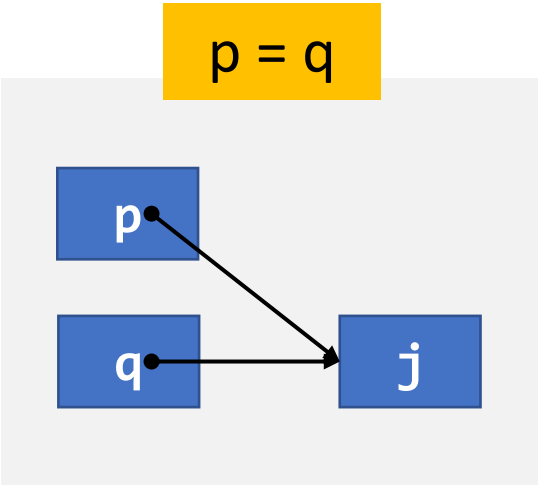
- 指针使用
  - *p：可以看作i的别名，代表使用*运算符访问存储在指向对象中的内容
  - `*p = 2` 等价于 `i = 2`

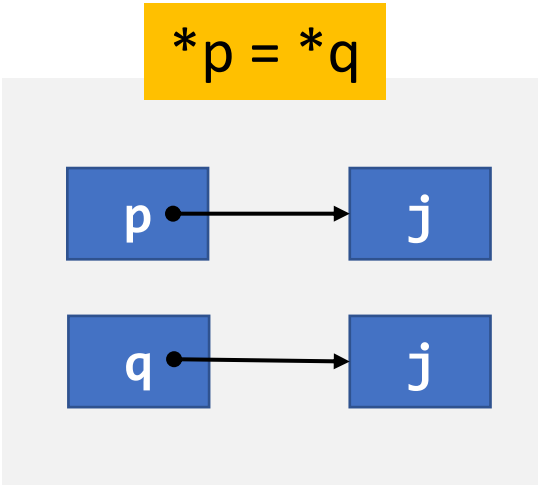
指针的赋值

- 多个指针可指向同一个对象
  - `int *p = &i;`
  - `int *q = &i;`



- 指针赋值操作
  - `int *p = &i, *q = &j;`





*是一个单目运算符

- 左值和右值
- 作用：访问指针的值所表达的内存地址上的变量
- 可以做左值也可以做右值
  - `int k = *p`
  - `*p = k+1`

指针运算符*&

- 互为反作用
  - *：取内存地址所表示的变量
  - &：取变量所在的内存地址

  - *&yptr → *(&yptr) → *(取yptr的内存地址) → yptr
  - &*yptr → &(取yptr表示的内存地址的变量) → &(y) → y的地址，即yptr

指针作为函数参数

- 指针可以作为函数参数，实现函数对其指向内存的操作
  - `void f(int *p);`
  - `int *p = &n; scanf(“%d”, p);`
    - `scanf(“%d”, &n)`
  - `swap(a, b)` → `swap(&a, &b)`
    - `swap(int *, int *)`
  - `void min_max(int a[], int len, int *max, int *min);`

指针作为函数返回值

- `int *max(int a[], int len);`

- 注意：不要返回指向自动局部变量的指针！

```
warning: function returns address of local variable [-Wreturn-local-addr]
```

指针常见错误

- 指针使用前未指向任何变量
  - 野指针：gcc –Wall
  - 未指向任何变量，可能有任何初始值
- *和&含义混乱
- 正确认识：任何类型指针都只是存储地址的变量，区别在于不同类型指针的解读内存方式不同
  - 如何理解？

特殊的地址

- 0地址
  - 虚拟内存，进程以为的0地址
  - 内存有0地址，但往往是不能随便碰的地址（甚至可能不能读取）
  - 你的指针不应该具有0值

  - 可以用指针为0表示特殊含义
    - 无效
    - 未初始化
    - NULL预定义

```
1  #undef NULL
2  #if defined(__cplusplus)
3  #define NULL 0
4  #else
5  #define NULL ((void *)0)
6  #endif
```

指针与数组

- 函数参数中的数组参数，其实就是指针
  - 回顾：sizeof(a)为什么等于4或者8？//a为函数的数组参数

- 可看作等价的函数原型
  - `int sum(int a[], int len);`
  - `int sum(int [], int);`
  - `int sum(int *a, int len);`
  - `int sum(int *, int);`

- 数组变量是一种特殊的指针（const指针）
  - `int a[10]; int *p = a;`
  - `a == &a[0]`
  - ~~`int b[] = a;//wrong!`~~

指针的运算

- 指针可以和数组同等取下标运算
  - `int a[10];`
  - `int *p = a;`
  - `p[0] == a[0]`

- 指针的算术运算
  - 利用指针的算术运算来代替数组下标进行处理
  - `int a[10], *p;`
  - `p+1`：加其指向类型的`sizeof`大小
    - 如果指针指向的不是连续内存，没有意义
    - 一般和数组关系密切

* 与 ++

- *p++
- *(p++)
- (*p)++

- ++*p
- ++(*p)

- *++p
- *(++p)

表 2-1　C 语言运算符优先级表（由上至下，优先级依次递减）

运 算 符	结 合 性
() [] -> .	自左向右
! ~ ++ -- - (type) * & sizeof	自右向左
* / %	自左向右
+ -	自左向右
<< >>	自左向右
< <= > >=	自左向右
== !=	自左向右
&	自左向右
^	自左向右
|	自左向右
&&	自左向右
||	自左向右
?:	自右向左
assignments	自右向左
,	自左向右

*p++

- 等同于*(p++)
  - *虽然优先级高，但是没有++高
  - 取出p所指的数据来，然后顺便把p移到下一个位置
  - 常用于数组遍历这样的连续空间操作
  - 在某些CPU中，可以被翻译成一个单独的指令

表 2-1　C 语言运算符优先级表（由上至下，优先级依次递减）

运 算 符	结 合 性
() [] -> .	自左向右
! ~ ++ -- - (type) * & sizeof	自右向左
* / %	自左向右
+ -	自左向右
<< >>	自左向右
< <= > >=	自左向右
== !=	自左向右
&	自左向右
^	自左向右
|	自左向右
&&	自左向右
||	自左向右
?:	自右向左
assignments	自右向左
,	自左向右

指针也可以比较

- <, <=, ==, >, >=, !=
- 比较表示的内存地址
- 数组中单元的地址肯定是线性递增的

根据C标准（C17 6.5.8）：

> When two pointers are compared, the result depends on the relative locations in the address space of the objects pointed to. If two pointers to object types both point to the same object, or **both point one past the last element of the same array object**, they compare equal.

指针的赋值

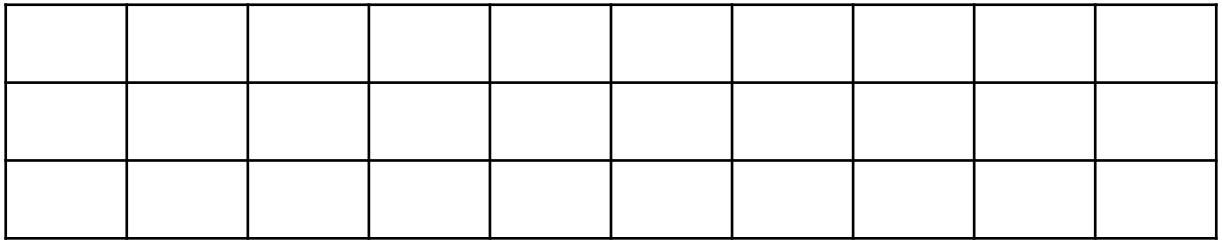
- 不同类型不可以相互赋值
  - `int *p; char *q;`
  - `q = p;`
- `void *`:表示不知道指向什么类型空间的指针
  - 与char*解读类似
- 指针也可以做类型转换
  - `int *p = &i; void *q = (void *)p;`
  - 通过q解读内存的视角变了

指针summary

- 表示函数参数
- 连续空间遍历
- 存储函数返回

多维数组与指针

- int matrix[3][10];
  - matrix // int (*)[4]
  - matrix+1 //&(matrix[1])
  - *(matrix+1)+5 //&(matrix[1][5])
  - &matrix +1 //out of bound
  - *(*(*(matrix+1)+5) //SEGSEV



多维数组作为参数

- `void func(int (*mat)[10])`
- `void func(int mat[][10])`

- `void func(int **mat)//Different`

指针和const

- 指针是const
  - `int * const p = &a;`
  - `*p = 100;//ok`
  - `p = &b; //ERROR`
  - `p++; //ERROR`

p指针指向关系一旦确定不可再变

- 指针所指是const
  - `const int *p = &a;`
  - `*p = 100; //ERROR`
  - `a = 100; //ok`
  - `p = &b; //ok`

表示不能通过该指针修改此变量
（并不能使变量变成const）

- 数组名称天然是const，不可改变其值，常量地址

const与数组

- const数组
  - `const int a[] = {1,2,3,4,5};`
  - 数组变量已经是const指针，再加const代表每个数组单元是const
  - 因此，必须通过初始化赋值

- 参数中用const修饰数组参数
  - `int sum(const int a[], int len);`

  - 可以要求函数内部不应该修改原始数组

End.

表 2-1　　C 语言运算符优先级表（由上至下，优先级依次递减）

运 算 符	结 合 性
() [] -> .	自左向右
! ~ ++ -- - (type) * & sizeof	自右向左
* / %	自左向右
+ -	自左向右
<< >>	自左向右
< <= > >=	自左向右
== !=	自左向右
&	自左向右
^	自左向右
|	自左向右
&&	自左向右
||	自左向右
?:	自右向左
assignments	自右向左
,	自左向右

小quiz

- 这些是啥意思?
  - `int i;`
  - `int * const p = &i;`
  - `const int *p = &i;`
  - `int const *p = &i;`

- 主要看const在*前还是后？
  - 前：所指不能修改（不能通过该指针修改此变量）
  - 后：指针不能修改（该指针指向此变量的关系不能变）