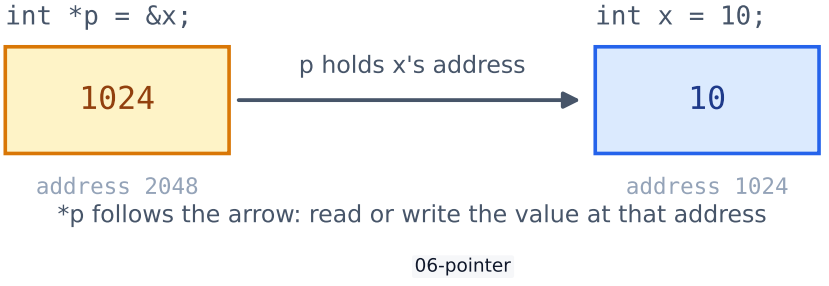
指针

C Reference

&、* 与按指针传递

指针 (pointer) 存储一个变量的地址 (address)。`&x` 给出 `x` 的地址;`*p` 对指针解引用 (dereference)— 读取或写入存放在那里的值。传递指针能让函数修改调用者的变量 (按指针传递,pass-by-pointer)。



指针存储一个地址; 解引用就是沿着箭头找到那个值

```c
#include <stdio.h>

void addOne(int *p) {      // p 保存一个地址
    *p = *p + 1;           // 修改那个地址处的值
}

int main(void) {
    int x = 10;
    int *ptr = &x;         // & 取得 x 的地址
    printf("%d\n", *ptr); // * 读取值:10
    addOne(&x);
    printf("%d\n", x);     // 11 -- 通过指针被修改了
    return 0;
}
```

最经典的用法是交换函数。按值传递只是拷贝 — 交换会丢失。传指针才能让函数够到调用者的变量:

```c
#include <stdio.h>

void swap_values(int a, int b) {      // 拷贝: 调用者看不到变化
    int t = a; a = b; b = t;
}

void swap_pointers(int *a, int *b) {  // 地址: 交换是真的
    int t = *a; *a = *b; *b = t;
}

int main(void) {
    int x = 1, y = 2;
    swap_values(x, y);
    printf("%d %d\n", x, y);   // 1 2  (没变!)
    swap_pointers(&x, &y);
    printf("%d %d\n", x, y);   // 2 1  (交换了)
    return 0;
}
```

数组名的行为就像指向第一个元素的指针, 而指针运算 (pointer arithmetic) 按整个元素来步进:

```c
#include <stdio.h>

int main(void) {
    int a[] = {10, 20, 30};
    int *p = a;                  // 等同于 &a[0]
    printf("%d\n", *p);          // 10
    printf("%d\n", *(p + 2));    // 30 -- 等同于 a[2]
    return 0;
}
```

- NULL 是一个" 什么都不指" 的指针 — 解引用之前先检查它。

常见错误

- &x 是 x 的地址;*p 是 p 指向的值。
- 绝不要对未初始化或 NULL 的指针用 *p— 会崩溃或破坏内存。
- 要改调用者的变量, 就传它的地址, 并通过指针写入。