C 语言课程.QA

11.24——11.30

Q1: 多维数组传递的真相：降维（Decay）

A: 在 C 语言中，当一个一维数组（如 int data[4]）被用作函数参数时，它会立即“降维”或“退化”为一个指向其元素类型的指针。

$$\text{int arr[ ]} \quad \longrightarrow \quad \text{int *}$$

Q2: 当你调用 print-array(data[0], 4) 时

A: data[0] 是什么？data[0] 代表二维数组的第 0 行，它本身就是一个一维数组（int [4]）。传递时发生了什么？当这个一维数组 data[0] 被传递给函数 print-array 时，它立即降维成一个指向其第一个元素（data[0][0]）的普通指针（int *）。

Q3: int nums[3][4] 是怎么储存的

A: C 语言在内存中是连续存储的，所以我们可以将 3x4 视为一个 12 元素的扁平数组

Q4: 指针的本质：地址变量

A: 我们知道变量（如 int a）存储的是数据（值）。而指针变量（如 int *p）存储的不是数据，它存储的是另一个变量的内存地址。

普通变量 (int a) 存储数字 10

指针变量 (int *p) 存储数字 10 所在的内存地址（如 0x7ffc...）

```c
int a = 10;
int *p = &a;    // p 里存的是 a 的地址
printf("%d\n", a);    // 10
printf("%d\n", *p);   // 10

*p = 20;                // 通过指针修改 a
printf("%d\n", a);    // 20
```

Listing 1: 代码

Q5: 指针的内存关系

A:

```
        内存地址      内容
        0x1000 ----> 10   (a)
        0x2000 ----> 0x1000   (p)
        int a = 10;      // 在 0x1000
        int *p = &a;     // 在 0x2000, 存 0x1000
        p   = 0x1000
        *p  = 10
        &a  = 0x1000
```

Listing 2: 代码

Q6: 临时变量——局部变量

A: 局部变量：当你在一个函数内声明并使用一个变量时，它通常是临时的。这个变量的生命周期仅限于函数的执行期间，函数执行完毕后该变量就会消失。

```
    int add(int a, int b) {
        int sum = a + b;  // sum 是一个临时变量, 用于存储结
            果
        return sum;
    }
```

Listing 3: 代码

Q7: 临时变量——表达式中的临时变量

A: 在表达式中，C 编译器会自动创建临时变量来存储中间值

```
    int x = 5;
    int y = 10;
```

```
    int result = (x + y) * 2;  // (x + y) 可能会临时存储结果在一个临时变量中
```

Listing 4: 代码

Q8: 临时变量——中间计算结果存储

A: 用来保存一个计算中的结果，以便后续使用。例如，在数学运算或数组排序中，常使用临时变量来交换两个元素或存储排序过程中的中间结果

```
void swap(int *a, int *b) {
    int temp = *a;  // temp 是临时变量
    *a = *b;
    *b = temp;
}
```

Listing 5: 代码

Q9: 用指针来“操控数组”

A: 数组名本身就是地址

```
int arr[5] = {10, 20, 30, 40, 50};

printf("%d\n", arr[0]);      // 10
printf("%d\n", *arr);        // 10
printf("%d\n", *(arr + 1)); // 20

arr      = 第0个元素地址
arr+1    = 第1个元素地址
*(arr+1) = 第1个元素的值
```

Listing 6: 代码