3.指针 结构体

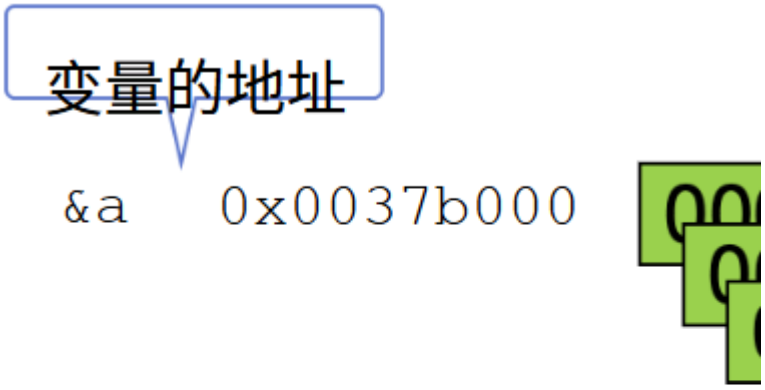
指针

指针意义：

为什么要引入指针

- 指针存储变量的内存地址，允许程序直接访问和修改指定内存单元的数据，这对底层硬件交互、内存资源管理至关重要
- 当传递大型数据（如结构体）时，传递指针（仅占 4/8 字节）比拷贝整个数据更节省内存和时间，尤其在函数参数传递中显著提升效率。
- 链表、树、图等动态数据结构依赖指针实现节点间的关联，没有指针难以灵活构建这类结构。
- C 函数默认只能返回一个值，通过指针作为参数，可在函数内部修改外部变量，间接实现“返回多个结果”
- 通过`malloc/free`等函数分配的堆内存，必须通过指针访问和释放，指针是动态内存操作的唯一接口

内存地址：



地址是一个无符号整数，从0开始，依次递增。在表达和交流时，通常把地址写成十六进制数

指针使用：

- 指针只存储变量的地址（下标），不存储变量的解释方式（大小和格式）
- 因此，在C语言中引入了指针的类型，用于在编译时帮助确定指针的特性
  - 编译时：在程序开始运行之前，对指针变量的操作就已经确定了
  - 假如有一个int型指针，则对其解引用永远都会按照int来读取

- 指针类型由“基类型 + 指针标识*”构成，核心作用是型

- `int *ptr; // ptr 为 int*类型`

```
<TYPE> *<VAR>;
```

取地址：

- &（取地址运算符）
  - 作用：获取变量的内存地址
  - 操作对象：只能对变量使用（不能对常量、表达式使用）
  - 结果：返回变量的地址（指针类型的值）
    - `&<VAR>`

解引用：

*（解引用运算符）

- 作用：通过指针访问其指向的变量（即 “根据地址找到变量”）
- 操作对象：只能对指针变量使用（不能对普通变量或非指针类型使用）
- 结果：返回指针指向的变量本身（可读取或修改其值）
  - `*<PTR>`

动态内存分配：

- 存储位置：堆内存（手动管理）
- 生命周期：从malloc/calloc分配开始，到free释放或程序结束时销毁
- 特性：需手动管理，即使创建它的函数返回，内存仍保留
- 只能由指针访问

```
int *ptr = malloc(sizeof(int));
...
free(ptr);
```

结构体