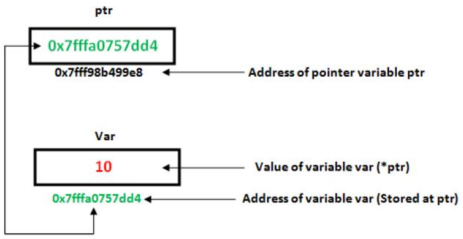
一、指针与引用

（一）指针（Pointer）



☑什么是指针?

指针（Pointer）是一种“变量”，它不直接存放数据，而是存放另一个变量的**“地址”**。

类比一下：

- 普通变量像是“一个房子里住着一个数字”。
- 指针就是“这栋房子的地址”，你拿着地址就可以找到住着那个数字的房子！

C++ 示例：简单指针

```
#include <iostream>
using namespace std;
int main() {
    int x = 10;
    int* p = &x; // p 是一个指向整数的指针，存的是 x 的地址
    cout << "x 的值是：" << x << endl;
    cout << "x 的地址是：" << &x << endl;
    cout << "p 存的是地址：" << p << endl;
    cout << "p 指向的值是：" << *p << endl; // *p 代表取出地址里的值
}
```

程序解释：

- **int* p;** 就像是“地图”，指向某个变量的地址。
- **&x** 是获取变量 x 的地址（“房子的位置”）。
- ***p** 是打开这个地址，取出里面的值（“去那房子拿东西”）。

考试注意事项：

易错点	正确做法说明

忘记加 * 定义指针	定义时要写：int* p;
用错地址和取值符号	&变量名 是取地址，*指针名 是取值
忘记初始化指针	指针未初始化不能随便用，可能指向未知内存

（二）基于指针的数组访问（Pointer + Array）

☑为什么数组和指针有关?

数组（Array）在内存中是“连续的一排房子”，而数组名其实就是**第一个元素的地址**，这和指针非常像。

示例：数组与指针访问方式

```
#include <iostream>
using namespace std;
int main() {
    int arr[3] = {10, 20, 30};
    int* p = arr;   // p 是 &arr[0]
    cout << "第一项：" << *(p + 0) << endl; // arr[0],输出 10
    cout << "第二项：" << *(p + 1) << endl; // arr[1],输出 20
    cout << "第三项：" << *(p + 2) << endl; // arr[2],输出 30
}
```

数组就是“排队的同学”，p + 1 就是往后走一步，* 是点名叫人。

☑总结三种访问方式:

方式	含义
arr[1]	第二个元素
*(arr + 1)	第二个元素
p[1]	第二个元素（p=arr）

考试易错点：

易错点	正确说明
*p[1] 写法错误	应该是 *(p + 1) 或 p[1]
没理解数组名是地址	arr 就是 &arr[0]
指针越界	不要访问 arr[3]（不存在）

（三）字符指针（Character Pointer）

☑什么是字符指针?

字符指针（**char***）是用来**指向字符串的地址**的指针。

示例：指向字符串的指针

```
#include <iostream>
using namespace std;
int main() {
    char* str = "Hello";
    cout << "字符串是：" << str << endl;
    cout << "第一个字符是：" << *str << endl;
    cout << "第二个字符是：" << *(str + 1) << endl;
}
```

- C++里，字符串其实就是一个字符数组，如 "Hello" 被看成：['H', 'e', 'l', 'l', 'o', '\0']
- 如果字符串是糖果串，字符指针就是拿着**糖串第一颗的位置**，往后慢慢吃每一颗。

☑使用 char 数组 vs char 指针：

```
char name1[] = "Tom";      // 字符数组，内容存在数组里
```

```
char* name2 = "Jerry";      // 字符指针，指向字符串常量
```

注意事项（非常重要）：

问题	原因说明
尝试修改 char* str = "abc"	字符串常量不能被修改，修改会导致程序崩溃
str[5] 越界访问	字符串以 '\0' 结束，不要访问超过范围

（四）指向结构体的指针（Pointer to Struct）

指向结构体的指针是一个指针变量，专门用来存储**某个结构体变量在内存中的地址**。

- 通过指针，可以访问或修改该结构体中的成员。
- 结构体就像一个“学生信息表”，里面有姓名、年龄、成绩等；
- 指针是学生信息表的“地址牌”，通过地址可以找到并修改学生的信息。

C++ 示例：

```
#include <iostream>
using namespace std;
// 定义结构体 Student
struct Student {
    string name;
    int age;
    float score;
};
int main() {
    Student s1 = {"小明", 10, 95.5};
    // 定义指向结构体的指针，指向 s1 的地址
    Student* p = &s1;
    // 通过指针访问成员，注意用箭头操作符 "->"
```

```
    cout << "姓名：" << p->name << endl; // s1.name
    cout << "年龄：" << p->age << endl;
    cout << "成绩：" << p->score << endl;
    // 通过指针修改成员
    p->score = 98.0;
    cout << "更新后成绩：" << s1.score << endl;
    return 0;
}
```

重点说明：

- 定义指向结构体的指针：Student* p = &s1;
- 访问结构体成员：**p->成员名**，不能用点号（.）
- 修改结构体成员：通过指针修改，改变的是原变量的值

考试注意事项 & 易错点：

易错点	正确写法说明
用错访问符号	结构体指针访问成员要用 ->，不是 .
指针未初始化	指针必须指向有效结构体地址
误用指针	指针是地址，不能直接赋值结构体变量

（五）引用（Reference）

引用（Reference）是**给变量起的另一个名字**，它不是新变量，而是原变量的别名。

- 通过引用访问或修改变量，直接作用于原变量本身。
- 就像你有个昵称“阿明”，这个昵称就是你的“引用”，说阿明和说你，都是同一个人。
- 给变量取个“小名”，你用小名和用真名是一样的。

C++ 示例：

```
#include <iostream>
using namespace std;
void addOne(int& x) {   // x 是引用，指向调用时传入的变量
    x = x + 1;
}
int main() {
    int a = 10;
    addOne(a);    // 传入变量 a 的引用
    cout << "a 的值是：" << a << endl;   // 输出 11，a 被函数修改了
    return 0;
}
```

重点说明：

- 引用定义时加 &，如 int& x，表示 x 是引用
- 引用必须在定义时初始化
- 引用本质是原变量的别名，操作引用就是操作原变量

考试注意事项 & 易错点：

易错点	正确写法说明
忘记引用符号 &	引用必须写 类型 &名
引用未初始化	定义引用时必须立即赋值
误用指针与引用混淆	指针是变量地址，引用是别名