指针

指针的作用：可以通过指针间接访问内存

指针本质：简单的讲就是代表一个地址

- 内存的编号都是从 0 开始记录的，一般用 16 进制表示，当然也可以用 int 转成十进制
- 可以利用指针变量保存地址。

指针变量的定义和使用

指针变量的语法：数据类型 * 变量名；

- 普通变量和指针变量的区别是，普通变量存放的是数据，但是指针变量存放的是指针
- 指针变量能通过“*”操作符，操作指针变量指向的内存空间，比如重新对指针指向的值重新赋值，或者输出，这个过程称为解引用、

指针指向的是某个地址,解指针可以对指针指向的某个地址,也就是某个内存空间进行操作,比如改变那个内存空间的值或者输出这内存空间的值。

指针所占用的内存空间

指针本身也是一种数据类型，因此也是能用 sizeof()函数计算其内存空间

结论：所有类型，无论是 int，float，double，char 等等字符类型在 32 位操作系统下是 4 字节，但是在 64 位操作系统下是 8 个字节。

空指针和野指针

- 空指针
    1. 指针变量指向内存中编号为 0 的空间：int * p = NULL;
    2. 可以用于初始化指针变量
    3. 空指针指向的内存是不可以访问的,不能对其进行操作

- 野指针

1. 指针变量指向非法的内存空间
2. 尽量不要使用野指针，容易误用

形式：int * p = (int *)0x1100;

空指针和野指针都不是我们申请的空间，因此不能对其进行访问。

const 修饰指针

三种形式

- const 修饰指针—常量指针：指针的指向可以该，但是指针指向的值不能该
- const 修饰常量—指针常量：指针的指向不能修改，但是能修改指针指向的值
- const 既修饰指针，又修饰常量。

看 const 右侧紧跟着的是指针还是常量， 是指针就是常量指针，是常量就是指针常量

指针和数组

作用：利用指针访问数组中元素

```
int arr2[] = {1,2,3,4,5,6,7,8,9};
int* y = arr2;//指向数组指针

cout << "第一个元素：" << arr2[0] << endl;
cout << "指针访问的第一个元素：  " << *y << endl;
cout << "第一个元素：  " << &arr2[0] << endl;


for (int m = 0; m < 9; m++)
{
    cout << *y << endl;
    y++;
}
```

指针和函数

作用：利用指针作为函数参数，可以修改实参值
值传递不改变实参的值；地址传递会改变实参；

值传递：

```
void swap1(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;

    cout << "交换后的值" << endl;
    cout << "a = " << a << endl;
    cout << "b= " << b << endl;


}
```

值传递的过程中，会改变自定义函数中，形参中的值。但是，对于实参的值并不会产生影响。可以这么理解，实参值传递到形参，相当于是一个复制黏贴的过程，她们在内存空间中是两块不一样的内存空间。因此形参的改变并不会影响到实参。

地址传递：

```
void swap2(int* p1, int* p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
    cout << "交换后的值" << endl;
    cout << "a = " << *p1 << endl;
    cout << "b= " << *p2 << endl;
}
```

地址传递，讲形参作为指针的形式，实际是形参引用了实参的地址，形参在本质上就是一个地址，且该地址即是实参的地址，他们在内存空间上可以理解为同一个空间，因此对形参进行操作相当于就是对实参所在的地址进行了操作。因此，地址传递会改变实参。需要注意的是，在上述两种传递中，一定要注意在主函数中调用函数时，值的形式。地址传递需要对变量进行取地址操作。值传递：数值对数值。 地址传递：（因为指针本身就是一个地址）因此是地址对地址的传递。

当数组名传入到函数作为参数时，被退化为指向首元素的指针。