写在最前

这篇文章用来总结一些c++中比较基础但是易错的点，毕竟考场手写代码，有些东西还是比较重要的。

正文

指针与引用

之前在一篇CSDN里看到，引用代表的是一个变量的别名，而指针是一个新的变量，它指向原来的变量。在作为函数参数传递的同时，这二者没啥太大的区别，同时也都能修饰返回值的类型。

```
int x = 10;
int &y = x;
int* p = &x;

cout << ++(y) << endl;
cout << ++p << endl;

int &y = x*6; //wrong
int &y = 6; //wrong
cout << ++(&y) << endl; //wrong
cout << ++(&x) << endl; //wrong
cout << ++(*p) << endl; //okay

const int &y = 10; //okay
double x = 3.1415926;
const int &z = x;//okay
```

指针自增后内存地址向右移了一位，也就是一个int型所占的内存空间，而引用自增后是变量的值增加了一位。

左值与右值

讲义中的原话是，一个变量有双重角色（左右值），这个角色取决于他出现在哪个地方。 左值代表储存这个变量的内存位置，可读可写，右值代表这个变量的值，只读。

```
int x = 10;

//正确，++x是以左值形式返回
cout << ++++++x << endl;
//错误，x++是以右值形式返回
cout << x++++++ << endl;
//正确，x++只读
cout << x++ << endl;
```

一维数组与指针

之前有说法可以把数组理解为一个指针，其实不是特别恰当的。数组名代表了数组首元素的地址，也就是&a[0]，但是&a时要注意，这个操作相当于取出了存放数组首地址的地址，需要用二级指针来接受，同样的，&a+1也就相当于直接进入了数组的下一行，只不过这一行没有被声明。

指针数组与数组指针

声明char型的指针数组

```
char* a[4] = {"my", "name" ,"is" ,"Eric"};
```

可以看到指针数组当成二维数组在用（字符串数组），当然也可以当成指针的集合。要注意的是二者有些小区别。再申请动态内存时，当成二维数组必须要申请两级动态内存：

```
char** a = new char* [10];
for(int i = 0; i < 10; i++)
    a[i] = new char[10];
```

而当真正用于指针数组时，只需要申请一维动态内存，将数组的每一个元素指向需要指向的地方即可。理解这里我们可以类比声明一个一级指针时，我们可以开辟动态内存给他，让他指向一个新元素。也可以让他直接指向一个原有的内容，或是nullptr。 注意的是指针数组其实就是一个二级指针。

声明一个char型数组指针

```
char (*a)[4];
```

此时这个指针指向数组首地址，及首元素地址。

a+i，a[i]，*(a+i)，&a[i][0]是等同的。 由a[i]=*(a+i)得a[i]+j=*(a+i)+j。由于*(a+i)+j是二维数组a的i行j列元素的首地址，所以，该元素的值等于*(*(a+i)+j)。

const修饰的一二级指针

类中需要注意的点

栈与队列复习

内存分配时的问题