## 1. 函数对象

举个例子，std::algorithm库中的sort函数有个参数类型是Compare。

我们可以自定义比较函数，这是个**函数指针**。

```
bool comp(int a, int b)
{
    return a > b;//若a在b前，则返回true；若a在b后 或 a等于b，则返回false
}
sort(arr, arr + 5, comp);
```

也可以使用STL中预定义的比较函数，这里是**函数对象**。

```
sort(arr, arr+5, less<int>()) //从小到大
sort(arr, arr+5, greater<int>()) //从大到小
```

为啥比较函数要带括号呢？这是因为它是函数对象。

如何进行自定义类型的排序呢？

- 在类中重载小于运算符
- 自定义比较函数

```
bool compByAge(const People &a, const People &b)
{
    return a.age < b.age;
}

vector<People> vec;
sort(vec.begin(), vec.end(), compByAge);
```

【华为面试真题(2021)】函数指针、函数对象、指针函数都有什么区别？

- 函数指针：指向函数(首地址)的指针变量，即本质是一个指针。在C编译时，每一个函数都有一个入口地址，那么这个指向这个函数的函数指针便指向这个地址。

```
int max(int a, int b) {
    return a > b ? a : b;
}
int min(int a, int b) {
    return a < b ? a : b;
}

int(*f)(int, int); // 声明函数指针，指向返回值类型为int，有两个参数类型都是int的函数
```

```
int main(int argc, _TCHAR* argv[]){
    f = max; // 函数指针f指向函数max(将max函数的首地址赋给指针f)
    int c = (*f)(1, 2);

    f = min; // 函数指针f指向函数min(将min函数的首地址赋给指针f)
    c = (*f)(1, 2);
}
```

- 指针函数：函数返回类型是某一类型的指针，其本质是一个函数。例如函数 `int* f(int a, int b){}` 返回值是int*.
- 函数对象：函数对象是重载了"()"操作符的普通类对象。因此从语法上讲，函数对象与普通的函数行为类似。

## 2. 智能指针和引用计数

来自知乎回答：https://www.zhihu.com/question/20368881/answer/14918675

智能指针和普通指针的区别在于，智能指针实际上是对普通指针加了一层封装机制，这样的一层封装机制的目的是为了使得智能指针可以方便的管理一个对象的生命期。**智能指针实质是一个对象，行为表现的却像一个指针**

在C++中，我们知道，如果使用普通指针来创建一个指向某个对象的指针，那么在使用完这个对象之后我们需要**自己删除它**，例如：

```
ObjectType* temp_ptr= new ObjectType();
temp_ptr->foo();
delete temp_ptr;
```

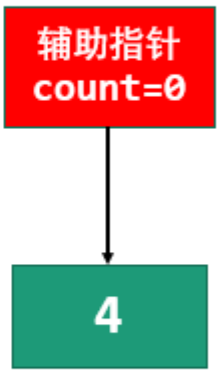
如果程序员**忘记**在调用完temp_ptr之后删除temp_ptr，那么会造成一个悬挂指针(dangling pointer)，也就是说这个指针现在指向的内存区域其内容程序员无法把握和控制，也可能非常容易造成内存泄漏。

可是事实上，不止是"忘记"，在上述的这一段程序中，如果foo()在运行时**抛出异常**，那么temp_ptr所指向的对象仍然不会被安全删除。（try-catch中运行到try，但是没运行完就开始运行catch了）

#####

shared_ptr（智能指针的一种）中所实现的本质是**引用计数**，也就是说shared_ptr是支持复制的，复制一个shared_ptr的本质是对这个智能指针的引用次数加1，而当这个智能指针的引用次数降低到0的时候，该对象自动被析构。

```
shared_ptr<int> p1(new int(4));
cout << p1.use_count() << ' ';
{
    shared_ptr<int> p2 = p1;
    cout << p1.use_count() << ' ';
    cout << p2.use_count() << ' ';
}  //p2出作用域
cout << p1.use_count() << ' ';
//引用计数为0，调用delete，销毁int*
```




输出：1 2 2 1

### 3. Lambda 表达式

lambda表达式是一个匿名函数，是一种**简单的**、**在同一行中定义函数**的方法。

```
f=lambda a,b,c,d:a*b*c*d
print(f(1,2,3,4))  #相当于下面这个函数
def test01(a,b,c,d):
    return a*b*c*d
print(test01(1,2,3,4))
```

部分Python内置函数接受函数作为参数,典型的此类内置函数有这些:

(1). filter

```
list = [1,2,3,4,5,6]
filtered = filter(lambda x:x % 3 == 0, list)
for i in filtered:
    print(i)
```

(2) sort

```
arr = [[1,2],[3,4],[1,3],[2,6]]
arr = sorted(arr,key = lambda x:x[0])
print(arr)
```

(3) map

```
arr = [1,2,3,4,5,6]
arr_ = map(lambda x:x**2, arr)
for i in arr_:
    print(i)
```