重载运算和类型转换

重载运算符

1. 除了重载的函数调用运算符operator()之外，**其他重载运算符都不能含有默认实参**。
2. 重载“<<”：第一个形参为非常量ostream对象的引用，第二个形参为一个常量的引用，是要输出的对象。输出运算符要尽量减少格式化操作，且必须是非成员函数。

```
ostream& operator<<(ostream &os,const 要输出的对象){
    os<<要输出的对象;
    return os;
}
```

3. 重载“>>”与重载“<<”相似，只是第二个形参应该为非常量。且**输入运算符必须处理可能失败的情况**

```
istream& operator>>(istream &is,要输出的对象){
    is>>要输入的对象
    return is;
}
```

4. 重载的赋值运算符必须定义为成员函数。但是复合赋值运算符（如+=，-=）则不一定为类的成员函数。
5. 下标运算符必须是成员函数。如果一个类包含下标运算符，则通常会定义两个版本：一个返回普通引用，另一个是类的常量成员并且返回常量引用。这是为了当类对象是常量时，对对象取下标也不能为其赋值。
6. 递增和递减运算符：为了区分前置和后置运算符，后置版本接受一个额外的（不被使用）的int类型的形参。当我们使用后置运算符时，编译器为这个形参提供一个值为0的实参。此外，一般情况下与内置版本保持一致，前置运算符然后的是引用，后置运算符返回的是一个值。

```
int& operator++(){}     //前置运算符   ++i
int operator++(int){}   //后置运算符   i++
```

7. 成员访问运算符：解引用运算符（*）首先检查对象要解引用的成员是否有效，若有效则返回该成员所指向的元素的**引用**。而箭头运算符（->）只是调用解引用运算符并返回解引用结果元素的**地址**。（必须为成员函数）
8. 对箭头运算符返回值的限定：箭头运算符永远不能丢掉**成员访问**这个最基本的含义。**当我们重载箭头时，可以改变的是箭头是从哪个对象中获取成员，而箭头获取成员这一事实是不变的**。对于形如point->mem的表达式来说，point必须是指向类对象的指针或者是一个重载了operator->的

类的uix。根据point类型的不同，point->mem分别等价于(*point).mem（point是一个内置指针类型）或者point.operator()->mem（point是类的一个对象）。point->mem的执行过程如下：

- 若point是指针，则应用内置的箭头运算符，等价于(*point).mem。首先解引用该指针，然后从所得到的对象中获取指定的成员。
- 若point是定义了operator->的类的一个对象，则我们**使用point.operator->()的结果来获取mem**。若该结果是一个指针，则执行上面所说的第一种情况；若该结果本身含有重载的operator->()则重复调用当前步骤。最终当这一过程结束时，程序或者返回了所需的内容，或者返回一些表示程序错误的信息。

因此**重载的箭头运算符必须返回类的指针或者自定义了箭头运算符的某个类的对象**。

9. 一个类可以定义多个不同版本的调用运算符“()”，它们相互之间应该在参数数量或类型上有所区别。
10. 当我们编写了一个lambda表达式后，**编译器将该表达式翻译成一个未命名类的未命名对象**。在lambda表达式产生的类中含有一个重载的函数调用运算符。

```
sort(v.begin(),v.end(),[](const int &a,const int &b){ return a<b; });
//上面代码的行为类似于下
class cmp{
    public:
        bool operator()(const int &a,const int &b){ return a<b; }
};
sort(v.begin(),v.end(),cmp())
```

默认情况下lambda不能改变它捕获的变量，所以lambda产生的类中的函数调用运算符是一个const成员函数。

11. 当lambda表达式通过引用捕获变量时，编译器可以直接使用这些引用而无须在lambda产生的类中将其存储为数据成员。而当lambda表达式通过值捕获变量时，lambda产生的类必须为每个值捕获的变量建立对应的数据成员，同时创建构造函数。

```
find(v.begin(),v.end(),[arg](const int &a){return a>arg;});
//等价于下面代码
class cmp{
    public:
        cmp(int i):arg(i){}
        bool operator()(const int &a) const{
            return a>arg;
        }
    private:
        int arg;    //需与 return中的数据同名
};
find(v.begin(),v.end(),cmp(arg));
```

12. 使用标准库定义的函数对象：比较两个无关指针将产生未定义的行为，即我们无法直接用比较运算符比较两个指针以比较他们的内存地址。但我们可以使用标准库函数对象来实现

```
vector<string*> pstring;
//这是错误的，指针之间没有关系 这是未定义的行为
sort(pstring.begin(),pstring.end(),[](string *a,string *b){return a<b;});
//可以借助标准库的函数对象
sort(pstring.begin(),pstring.end(),less<string*>());
```

关联容器如map,set使用less<key_type>对元素排序。

类型转换

1. 类型转换运算符是类的一种特殊成员函数，负责将一个类类型的值转换成其他类型，形式如：operator type() const。其中**type表示某种类型**。类型转换运算符可以面向任意类型（除了void）进行定义，只要该类型能作为函数的返回类型。因此**不允许转换成数组或者函数类型**。
2. 类型转换运算符必须是成员函数，且**不能声明返回类型，形参列表也必须为空（因为类型转换运算符是隐式执行的）**，通常是const。
3. 编译器一次只能执行一个**用户定义**的类型转换，但隐式的用户定义类型转换可以置于一个标准类型转换之前或者之后。
4. 类型转换运算符也可以定义为显示的（用explicit声明），当要执行类型转换时必须通过显示的强制类型转换才可以（static_cast<type>），其实也可以直接用：(type)对象 进行转换。若表达式出现在下列位置时，显示的类型转换将被隐式执行：
   - if while 及 do语句的条件部分
   - for语句头的条件表达式
   - 逻辑运算对象
   - 条件运算符的条件表达式

   总结起来就是参与条件的表示以及参与逻辑运算时，显示的类型转换运算符会被隐式执行。
5. 设计类型转换运算符时，要注意避免二义性。