# 重载运算符

重载运算符是通过对运算符的重新定义，使得其支持特定数据类型的运算操作。重载运算符是重载函数的特殊情况。

C++ 自带的运算符，最初只定义了一些基本类型的运算规则。当我们要在用户自定义的数据类型上使用这些运算符时，就需要定义运算符在这些特定类型上的运算方式。

## 限制

重载运算符存在如下限制：

- 只能对现有的运算符进行重载，不能自行定义新的运算符。
- 以下运算符不能被重载：`::`（作用域解析），`.`（成员访问），`.*`（通过成员指针的成员访问），`?:`（三目运算符）。
- 重载后的运算符，其运算优先级，运算操作数，结合方向不得改变。
- 对 `&&`（逻辑与）和 `||`（逻辑或）的重载失去短路求值。

## 实现

重载运算符分为两种情况，重载为成员函数或非成员函数。

当重载为成员函数时，因为隐含一个指向当前成员的 `this` 指针作为参数，此时函数的参数个数与运算操作数相比少一个。

而当重载为非成员函数时，函数的参数个数与运算操作数相同。

下面将给出几个重载运算符的示例。

## 函数调用运算符

函数调用运算符 `()` 只能重载为成员函数。通过对一个类重载 `()` 运算符，可以使该类的对象能像函数一样调用。

重载 `()` 运算符的一个常见应用是，将重载了 `()` 运算符的结构体作为自定义比较函数传入优先队列等 STL 容器中。

下面就是一个例子：给出 $n$ 个学生的姓名和分数，按分数降序排序，分数相同者按姓名字典序升序排序，输出排名最靠前的人的姓名和分数。

```
#include <iostream>
#include <queue>
using namespace std;
struct student {
  string name;
  int score;
};
struct cmp {
  bool operator()(const student& a, const student& b) const {
    return a.score < b.score || (a.score == b.score && a.name >
b.name);
  }
};
priority_queue<student, vector<student>, cmp> pq;
int main() {
  int n;
  cin >> n;
  for (int i = 1; i <= n; i++) {
    string name;
    int score;
    cin >> name >> score;
    pq.push({name, score});
  }
  student rk1 = pq.top();
  cout << rk1.name << ' ' << rk1.score << endl;
  return 0;
}
```

## 自增自减运算符

自增自减运算符分为两类，前置和后置。为了能将两类运算符区别开来，对于后置自增自减运算符，重载的时候需要添加一个类型为 `int` 的空置形参。

另外一点是，内置的自增自减运算符中，前置的运算符返回的是引用，而后置的运算符返回的是值。虽然重载后的运算符不必遵循这一限制，不过在语义上，仍然期望重载的运算符与内置的运算符在返回值的类型上保持一致。

因此，对于类型 T，典型的重载自增运算符的定义如下：

| 重载定义（以 `++` 为例） | 成员函数 | 非成员函数 |
| --- | --- | --- |
| 前置 | `T& T::operator++();` | `T& operator++(T& a);` |
| 后置 | `T T::operator++(int);` | `T operator++(T& a, int);` |

## 比较运算符

在 `std::sort` 和一些 STL 容器中，需要用到 `<` 运算符。在使用自定义类型时，我们需要手动重载。

还是以讲函数调用运算符时举的例子说起，如果我们重载比较运算符，实现代码是这样的（主函数因为没有改动就略去了）：

```
struct student {
  string name;
  int score;
  bool operator<(const student& a) const {
    return score < a.score || (score == a.score && name >
a.name);
    // 上面省略了 this 指针，完整表达式如下：
    // this->score<a.score||(this->score==a.score&&this-
>name>a.name);
  }
};
priority_queue<student> pq;
```

上面的代码将小于号重载为了成员函数，当然重载为非成员函数也是可以的。

```
struct student {
  string name;
  int score;
};
bool operator<(const student& a, const student& b) {
  return a.score < b.score || (a.score == b.score && a.name >
b.name);
}
priority_queue<student> pq;
```

事实上，只要有了 `<` 运算符，则其他五个比较运算符的重载也可以很容易实现。

```
/* clang-format off */

// 下面的几种实现均将小于号重载为非成员函数

bool operator<(const T& lhs, const T& rhs) { /* 这里重载小于运算符
*/ }
bool operator>(const T& lhs, const T& rhs) { return rhs < lhs; }
bool operator<=(const T& lhs, const T& rhs) { return !(lhs >
rhs); }
bool operator>=(const T& lhs, const T& rhs) { return !(lhs <
rhs); }
bool operator==(const T& lhs, const T& rhs) { return !(lhs < rhs)
&& !(lhs > rhs); }
bool operator!=(const T& lhs, const T& rhs) { return !(lhs ==
rhs); }
```

参考资料与注释：

- 运算符重载 - cppreference [https://zh.cppreference.com/w/cpp/language/operators]

---

> 本页面最近更新：2021/2/818:02:48，更新历史 [https://github.com/OI-wiki/OI-wiki/commits/master/docs/lang/op-overload.md]
> 发现错误？想一起完善？ 在 GitHub 上编辑此页！ [https://oi-wiki.org/edit-landing/?ref=/lang/op-overload.md]
> 本页面贡献者：StudyingFather [https://github.com/StudyingFather], Enter-tainer [https://github.com/Enter-tainer]