```
ostream& operator<<(ostream& os, const Data& da){
    return os << "(" << da.x << ", " << da.y;
}
```

这里为什么参数不能用const ostream& os?

在 C++ 中，`ostream` 对象（如 `cout`）在输出操作时会被修改，因为它的内部状态会随着写入操作而改变（比如流的位置指针、错误状态等）。因此，不能将 `ostream` 参数声明为 `const`。

具体来说：

流对象的状态会改变：每次向 `ostream` 写入数据（如 `os << ...`）都会更新流的状态（如 `failbit`、`eofbit`、当前写入位置等）。如果参数是 `const ostream&`，编译器会阻止这种修改，导致编译错误。

运算符 `<<` 通常返回非 const 引用：`operator<<` 通常返回 `ostream&`（非 const），以便支持链式调用（如 `cout << a << b`）。如果参数是 `const ostream&`，返回类型也必须是 `const ostream&`，这会破坏链式调用的约定。

正确的写法：

cpp 复制 下载

```cpp
ostream& operator<<(ostream& os, const Data& da) {
    return os << "(" << da.x << ", " << da.y << ")";
}
```

`os` 是非 const 引用，允许修改流的状态。

返回 `ostream&` 以支持链式调用（如 `cout << data1 << data2`）。

如果强行用 `const ostream&` 会怎样？

cpp 复制 下载

```cpp
ostream& operator<<(const ostream& os, const Data& da); // 错误！
```

编译会失败，因为 `os << da.x` 会尝试调用非 const 的 `operator<<`，而 `os` 是 const 的。

即使能编译，也无法修改流的状态（如错误标志、写入位置等），导致行为异常。

总结：

`ostream` 必须是 **非 const 引用**，因为输出操作会修改流的状态。这是 C++ 标准库的设计约定，所有流操作都遵循这一规则。