实验课 2 操作符重载

背景

一个没有操作符的比特流是没有灵魂的。对于两个不同的比特流，应当存在按位与、或等二元运算和取反这种一位运算。

在这次试验中，你需要为你的比特流实装所有的运算符重载。

实验描述

首先你需要填上上次课的坑，如果需要的话，给出应有的 copy-constructor 和 copy assignment constructor 。

你需要在类里面实现：

```
bits& operator &= ( const bits& v );
bits& operator |= ( const bits& v );
bits& operator ^= ( const bits& v );
bits& operator <<= ( size_t p );
bits& operator >>= ( size_t p );
```

左、右移是直接将这个 bits 进行移动，向左移动，则高位补零；向右移动，则高位舍去。

它们的功能可以参考对任意整数执行这几种操作。

除此之外，你需要在类外实现这些操作符：

```
// 比较操作符
bool operator == ( const bits& _a , const bits& _b );
bool operator != ( const bits& _a , const bits& _b );
bool operator < ( const bits& _a , const bits& _b );
bool operator > ( const bits& _a , const bits& _b );
bool operator <= ( const bits& _a , const bits& _b );
bool operator >= ( const bits& _a , const bits& _b );
```

```
// 友元函数的操作符
bits operator << ( const bits& _a , const size_t b );
bits operator >> ( const bits& _a , const size_t b );
```

```
bits operator | ( const bits& _a , const bits& _b );
bits operator & ( const bits& _a , const bits& _b );
bits operator ^ ( const bits& _a , const bits& _b );
bits operator ~ ( const bits& _a );
```

同时，你需要实现流操作输出符，最终可以使用 `std::cout`, `ofstream fout`, ... 输出。

```
std::ostream& operator<<(std::ostream& out, const bits& _bi
t);
```

实验指北

你可以在实现一些函数或操作符时调用其他已经实现的函数或者操作符从而减少代码量。

部分基本操作符的重载可以调用相关函数。

部分操作（左右移）需要自行写。

对于重载流操作输出符的重载，需要详细思考使用 `std::cout` 输出的过程。使用 `std::cout << a << b` 时，本质上是调用了一个操作符函数 `operator<<(std::cout, a)`，然后 `std::cout` 处理将 `a` 的输出，然后返回 `std::cout`，让下一个内容输出。所以这里我们的输出重载也需要这样操作。传入 `std::ostream& out` 为一个与 `std::cout` 类似的对象，而它可以直接输出一个字符串，然后返回这个 `out`。