背景

有了比特引用后，可以方便地取出一个位置得比特，但是该如何便利整个迭代器呢?

对于一个 bits 类，可以考虑定义类似于 STL 库的迭代器。

这个迭代器应当是支持随机访问的迭代器，并且所有随机访问操作应当做到 $O(1)$ 时间内完成。

概述

你需要实现一个模版类

你需要实现一个模版类 `bit_iterator` ，其模版参数为指向的比特存储于什么类中。

其需要两个值来描述其指向的位置，分别是一个指针，用于指向数组中的某个位置，和一个值，表示指向的比特距离该数组中的位置的结尾差了多少位。

同时，作为一个随机访问迭代器类，其应当被标注 `std::random_access_iterator_tag` ，否则它不能很好地和 STL 协同工作。

```
template<class origin>
class bit_iterator {
public:
    using iterator_category = std::random_access_iterator_tag;
    using value_type = bool;
    using difference_type = typename origin::difference_type;
    using pointer = bit_iterator;
    using reference = bit_reference<origin>;
private:
    using storage_type = typename origin::storage_type;
    using storage_pointer = typename origin::storage_pointer;
    static const unsigned bits_per_word = origin::bits_per_word;

    storage_pointer seg;
    unsigned ctz;
}
```

基础功能

- 比特迭代器应当具有一些构造函数。思考并实现它的不同类型的构造函数，并且想想每种构造函数是 `public` 还是 `private` 的。
- 比特迭代器最重要的功能之一是访问其指向位置的值，也就是对于 `bit_iterator it` 应当可以访问 `*it` 的值。

  搜索资料并且尝试实现这个功能，你需要思考这个操作的返回值是什么（Hint: 除了访问之外，是否可以通过 `*it` 进行修改）。
- 需要一个等于复制函数，即 `operator =` 。
- 迭代器的最基础的操作： `++it , --it , it++ , it--` 。可以自行查询手册，看看怎么实现这四种重载。
- 同时，由于该迭代器是随机访问迭代器，你应当实现 `+=,-=` 两种操作，用于移动若干位。其参数应当是一个 `difference_type` 。

- 同时，迭代器中你还需要实现 `it` 和整数的加法运算和 `it` 之间做差的运算，即 `+` 和 `-` 两种操作符。
- 由于地址间是有序的，迭代器之间也是有序、可比较的。

  重载并实现 `==,!=,<,>,<=,>=` 这些布尔运算符。

同时，在得到比特迭代器后，你可以尝试通过你实现的比特迭代器，解决 bitset tester 最后的一个部分。

如果你遇到了问题，先尝试阅读实验指北或上网查询资料，如果还是无法解决可以询问助教。

实验指北和提示

- 思考 `bit_iterator` 和之前实现的几种类之间的友元关系，并用 `friend` 关键字声明。
- 同样的，你需要注意 `const` 问题。

  你会发现，在比特迭代器中，常量迭代器和非常量迭代器的**所有**函数是否为常函数是没有区别的。因为比特迭代器不涉及**直接的**修改操作，修改的方式只有通过 `*` 操作得到引用后执行修改。

  因此我们不再需要将比特迭代器分为两个类去写，分别实现常迭代器和非常迭代器。你可以在模版参数中加入 `is_const` 表示该类为常迭代器还是非常迭代器。

  显然，你不应当能直接或间接地修改常迭代器指向的内容，因此任意赋值操作和复制操作构造迭代器时，**只能**从非常迭代器复制构造。

  在这个过程中，`std::conditional_t` 可能会被用到，可以提前查询手册学习它的基本用法。