整数流

通过重载 `++`（包含前置和后置）和 `*` 运算符实现一个**惰性**求值的 `int` 流。

> 本题灵感来自《计算机程序的结构和解释》第 3.5 节“流”

可通过如下三种方式创建一个 `int` 流：

- 仅指定起始值。表示从该值开始、每次加一的流，例如 `IntStream(1)` 能够生成序列 1、2、3、…
- 指定起始值和结束值。表示从指定的起始值开始、每次加一、不包含指定的结束值在内的流。例如，`IntStream(1, 11)` 能够生成序列 1、2、…、**10**
- 指定起始值、结束值和步长。表示从指定的起始值开始、每次增加指定的步长、不包含指定的结束值在内的流。例如，`IntStream(1, 11, 3)` 能够生成序列 1、4、7、10，`IntStream(1, 14, 3)` 能够生成序列 1、4、7、10、13

其中：

- 前置 `++` 和后置 `++` 用于推进 `int` 流
- `*` 用于获取当前 `int` 流最前面的值
- `operator bool` 用于将当前 `int` 流转换为 `bool` 值

输入

本题不需要处理输入。

输入格式参见代码框架的 `main` 函数。

输出

本题不需要处理输出。

输出格式参见代码框架的 `test_*` 系列函数。

提示

算术运算可能存在溢出问题！

代码框架

```
#include <functional>
#include <iostream>
#include <limits>
#include <string>
```

```cpp
#include <unordered_map>

class IntStream {
 public:
  explicit IntStream(int first);
  IntStream(int first, int last);
  IntStream(int first, int last, int stride);

  IntStream &operator++();
  IntStream operator++(int);
  int operator*() const;

  operator bool() const;

  // TODO: your code
};

void print_answer(const IntStream &s, int expect) {
  std::cout << std::boolalpha;
  if (s) {
    std::cout << (*s == expect) << ' ' << *s << std::endl;
  } else {
    std::cout << false << std::endl;
  }
}

/**
 * @brief 测试 IntStream(int)
 */
void test_1() {
  IntStream s(0);
  for (size_t i = 0; i < 10; i++) {
    ++s;
  }
  print_answer(s, 10);
}

/**
 * @brief 测试 IntStream(int, int)
 */
void test_2() {
  IntStream s(0, 10);
  for (size_t i = 0; i < 9; i++) {
    s++;
  }
  print_answer(s, 9);
}

/**
```

```
 * @brief 测试 IntStream(int, int, int) - 不考虑溢出
 */
void test_3() {
  IntStream s(0, 10, 2);
  for (size_t i = 0; i < 4; i++) {
    ++s;
  }
  print_answer(s, 8);
}

/**
 * @brief 测试 IntStream(int, int, int) - 步长为负数
 */
void test_4() {
  IntStream s(10, 0, -1);
  for (size_t i = 0; i < 10; i++) {
    s++;
  }
  print_answer(s, 0);
}

/**
 * @brief 测试 IntStream(int, int, int) - 考虑溢出，大于最大值
 */
void test_5() {
  IntStream s(std::numeric_limits<int>::max() - 10000,
              std::numeric_limits<int>::max(), 123);
  for (size_t i = 0; i < 50; i++) {
    ++s;
  }
  print_answer(s, 2147479797);
}

/**
 * @brief 测试 IntStream(int, int, int) - 考虑溢出，小于最小值
 */
void test_6() {
  IntStream s(std::numeric_limits<int>::min() + 10000,
              std::numeric_limits<int>::min(), -123);
  for (size_t i = 0; i < 50; i++) {
    s++;
  }
  print_answer(s, -2147479798);
}

/**
 * @brief 测试步长为 0 的情况
 */
void test_7() {
```

```
  IntStream s(std::numeric_limits<int>::min(),
std::numeric_limits<int>::max(),
              0);
  for (size_t i = 0; i < 10000; i++) {
    s++;
  }
  print_answer(s, std::numeric_limits<int>::min());
}

/**
 * @brief 测试范围 [first, last) 非常大的情况
 */
void test_8() {
  IntStream s(std::numeric_limits<int>::min(),
std::numeric_limits<int>::max());
  for (size_t i = 0; i < 10000; i++) {
    s++;
  }
  print_answer(s, std::numeric_limits<int>::min() + 10000);
}

int main() {
  std::unordered_map<std::string, std::function<void()>>
test_cases_by_name = {
      {"test_1", test_1}, {"test_2", test_2}, {"test_3", test_3},
      {"test_4", test_4}, {"test_5", test_5}, {"test_6", test_6},
      {"test_7", test_7}, {"test_8", test_8},
  };
  std::string tname;
  std::cin >> tname;
  auto it = test_cases_by_name.find(tname);
  if (it == test_cases_by_name.end()) {
    std::cout << "输入只能是 test_<N>, 其中 <N> 可取整数 1 到 8." <<
std::endl;
    return 1;
  }
  (it->second)();
}
```