Спецкурс Си++. Лабораторная работа 3
C++ Stream

Введение:

Во многих языках, например таких как Java или C# поддерживается абстракция потоков для обработки запросов к различным данным. В Java 8 это Stream API, в C# это LINQ. Требуется реализовать схожую функциональность в C++.

Пример использования:

```
auto s = Stream{1, 2, 3, 4, 5};
(s | map([](auto x) { return x * x; }) | print_to(std::cout)) << std::endl;
```

Здесь создается стрим, который инициализируется значениями от 1 до 5. Затем над этим стримом выполняется операция возведения в квадрат и наконец стрим выводится в стандартный поток вывода.

Формальное описание:

Необходимо реализовать библиотеку, которая позволяет использовать абстракции потоковой обработки данных.

Библиотека должна предоставлять класс Stream, параметризированный типом хранимого значения.

API:

```
/**
 * construct Stream from range [begin, end)
 * range must be alive during stream operations
 * @example Stream a(myVector.begin(), myVector.end())
*/
Stream(Iterator begin, Iterator end);

/**
 * construct Stream from the Container
 * container must be alive during stream operations
 * @example Stream a(myVector)
*/
Stream(const Container& cont);

/**
 * construct Stream from the rref Container
 * @example Stream a(std::move(myVector))
*/
Stream(Container&& cont);
```

```
/**
 * construct Stream from the initializer_list
 * @example Stream a = {1, 2, 3, 4, 5}
*/
Stream(std::initializer_list<T> init);

/**
 * constructs Stream whose values the return values
 * of repeated calls to the generate function with no arguments.
 * @example
 *   Stream a(rand)
 *   a | sum() //compile time error with clear message, because stream is infinite
*/
Stream(Generator&& generator);

/**
 * get n elements from Stream
 * @example
 * Stream s(rand) | get(10) | print_to(std::cout)
 * should be 10 random numbers
*/
get(std::size_t n);

/**
 * construct Stream from the pack
 * @example
 *   Stream a(1, 2, 3, 4, 10)
*/
Stream(value1, value2, ...);

/**
 * transform the stream and return a new transformed stream
*/
map(Transform&& transform);

/**
 * @pseudocode
 * U result = identity(stream[0]);
 * for(T element : stream[1...]) {
 *     result = accum(result, element);
 * }
 * return result;
*/
reduce(Accumulator&& accum);
reduce(IdentityFn&& identityFn, Accumulator&& accum);

/**
 * filter the stream and return new one
 * @example
 * Stream s{1, 2, 3, 4, 5} | filter([](int x) {return x % 2 == 0;})
 * | print_to(std::cout);
 * should be 2, 4
*/
```

```
filter(Predicate&& predicate);

/**
 * skip first amount items from the stream and return new one
 * @example
 * Stream s{1, 2, 3, 4, 5} | skip(3) | print_to(std::cout);
 * should be 4, 5
*/
skip(std::size_t amount);

/**
 * group items in stream by N return stream of std::vector<value> with size <=N
 * @example
 * Stream s{1, 2, 3, 4, 5} | group(3);
 * should be (1, 2, 3), (4, 5)
*/
group(std::size_t N);

/**
 * sum all values in the stream
 * @example
 * int s = Stream s{1, 2, 3, 4, 5} | sum();
 * s should be 15
*/
sum();

/**
 * print stream into std::ostream and return this std::ostream
 * @example
 * Stream s{1, 2, 3, 4, 5} | print_to(std::cout) << std::endl;
*/
print_to(std::ostream& os, const char* delimiter = " ");

/**
 * convert to vector
 * @example
 * std::vector<int> vec = Stream s{1, 2, 3, 4, 5} | to_vector();
 */
to_vector();

/**
 * get the nth element from the stream
 * @example
 * int third = Stream s{1, 2, 3, 4, 5} | nth(3);
 * third should be 4
*/
nth(std::size_t index);
```

Нефункциональные требования:

Все операции должны исполняться отложено aka лениво, кроме терминирующих операций требующих непосредственно значений.
Реализация должна быть выполнена посредством композиций функций и полиморфизма времени компиляции.