Задача №1

Стандартная библиотека содержит std::bitset и специализацию шаблона std::vector<bool> для эффективного хранения и обработки большого объема данных для битов и bool. Задача — реализовать два контейнера. Контейнер для хранения РНК в виде цепочки нуклеотидов, и контейнер для хранения ДНК в виде двух комплиментарных РНК. Нуклеотиды могут принимать 4 значения:
A – аденин
G – гуанин
C – цитозин
T or U – тимин (для РНК) и урацил(для ДНК)

Для упрощения задачи будем считать, что тимин и урацил кодируются одинаковым набором бит. Так как элементы, которые будут храниться в контейнере, имеют всего четыре значения, предлагается хранить их компактно, выделяя на каждый нуклеотид ровно столько памяти, сколько требуется для того, чтобы закодировать значения.
Таблица №1: комплиментарные пары нуклеотидов

A	T\U
G	C
C	G
T\U	A

Для хранения одного нуклеотида достаточно 2 битов. Поэтому контейнеры из std неэффективно расходуют память для хранения тритов. Наш контейнер должен реализовать динамическое управление массивом для хранения тритов. Код должен быть кроссплатформенным (учитывать, что примитивные типы данных могут иметь разные размеры).

```
enum Nucleotide{A, G, C, T};
…….
//резерв памяти для хранения 1000 нуклеотидов, заполнение указанным
нуклеотидо
RNK rnk(A, 1000);
// length of internal array
size_t allocLength = rnk.capacity();
```

```
assert(allocLength >= 1000*2 / 8 / sizeof(uint) );
// 1000*2 - min bits count
// 1000*2 / 8 - min bytes count
// 1000*2 / 8 / sizeof(uint) - min uint[] size

//выделение памяти
set[1000'000'000] = A;
assert(allocLength < set.capacity());
```

Дополнительно реализовать методы:

```
//число заданных нуклеотидов в РНК
//для нуклеотида  - число значений
size_t cardinality( Nucleotide value);
//аналогично но сразу для всех типов тритов
std::unordered_map< Nucleotide, int, std::hash<int> > cardinality();

// забыть содержимое от lastIndex и дальше
void trim(size_t lastIndex);
// logical length - индекс последнего нуклеотида+1
size_t length();
```

Требуемые методы для класса RNK:
RNK operator+ (RNK & r1, RNK & r2);
opertator==
operator!=
operator !
RNK (const & RNK)
operator []
isComplementary (RNK &)
split(size_t index)
Для класса DNK:
DNK (RNK &, RNK &)

Для проверки корректности работы необходимо покрыть unit test-ами все публичные методы и операторы.
В качестве библиотеки для тестирования использовать Google Test Framework
(https://ru.wikipedia.org/wiki/Google_C%2B%2B_Testing_Framework)
http://www.ibm.com/developerworks/aix/library/au-googletestingframework.html