

FASTXIO

1

Generated by Doxygen 1.8.11

Contents

1	FASTX-IO	1
2	Namespace Index	5
2.1	Namespace List	5
3	Class Index	7
3.1	Class List	7
4	Namespace Documentation	9
4.1	FASTX Namespace Reference	9
4.1.1	Detailed Description	10
4.1.2	Function Documentation	11
4.1.2.1	get_whitespace(std::string &str)	11
4.1.2.2	is_bzip2(const char *input)	12
4.1.2.3	is_gzip(const char *input)	12
4.1.2.4	is_sequence_char(const char test, char seqtype)	12
4.1.2.5	operator!=(const Record &a, const Record &b)	13
4.1.2.6	operator<(const Record &a, const Record &b)	13
4.1.2.7	operator<<(std::ostream &outstream, const NucFrequency nuc)	13
4.1.2.8	operator<<(std::ostream &outstream, const Record &rec)	13
4.1.2.9	operator<<(std::ostream &out, const SW sw)	14
4.1.2.10	operator<<(std::ostream &outstream, Wrap rec)	14
4.1.2.11	operator<=(const Record &a, const Record &b)	14
4.1.2.12	operator==(const Record &a, const Record &b)	14
4.1.2.13	operator>(const Record &a, const Record &b)	15
4.1.2.14	operator>=(const Record &a, const Record &b)	15
4.1.2.15	recursive_iupac_enum(std::set< Record > &set, Record &rec, const std::map< char, std::vector< char > > &translation_table, std::set< char > &unambiguous_nuc)	15
4.1.2.16	scan_phred(std::istream &instream)	16
4.1.2.17	scan_phred(const char *infile)	16
4.1.2.18	sw_max_neighbour(score_t a, score_t b, score_t c)	16
4.1.3	Variable Documentation	17
4.1.3.1	global	17

5	Class Documentation	19
5.1	FASTX::GData Struct Reference	19
5.1.1	Detailed Description	19
5.1.2	Member Data Documentation	20
5.1.2.1	aa_alphabet	20
5.1.2.2	codon_to_protein_dna	20
5.1.2.3	codon_to_protein_rna	20
5.1.2.4	enum_iupac_dna	21
5.1.2.5	enum_iupac_rna	21
5.1.2.6	nuc_alphabet	21
5.1.2.7	rc	21
5.2	FASTX::NucFrequency Class Reference	22
5.2.1	Detailed Description	22
5.2.2	Member Function Documentation	22
5.2.2.1	add(const std::string &str)	22
5.2.2.2	add(const Record &str)	23
5.2.2.3	letters(void) const	23
5.2.2.4	operator[](char nuc) const	23
5.2.3	Friends And Related Function Documentation	23
5.2.3.1	operator<<	23
5.3	FASTX::Reader Class Reference	24
5.3.1	Detailed Description	24
5.3.2	Constructor & Destructor Documentation	24
5.3.2.1	Reader(const char *file, const char seqtype)	24
5.3.3	Member Function Documentation	24
5.3.3.1	next()	24
5.3.3.2	peek(void)	25
5.4	FASTX::Record Class Reference	25
5.4.1	Detailed Description	27
5.4.2	Constructor & Destructor Documentation	27

5.4.2.1	Record(std::istream &input, char seqtype)	27
5.4.2.2	Record(const std::string &fasta, const std::string &id, char seqtype)	27
5.4.2.3	Record(const std::string &fastq, const std::string &id, const std::string &qual, char seqtype)	27
5.4.3	Member Function Documentation	28
5.4.3.1	count_freq(void)	28
5.4.3.2	enumerate_iupac(void)	28
5.4.3.3	get_id(void) const	28
5.4.3.4	get_numeric_qual(void) const	28
5.4.3.5	get_qual(void) const	29
5.4.3.6	get_qual_ptr(void) const	29
5.4.3.7	get_seq(void) const	29
5.4.3.8	get_seq_ptr(void) const	29
5.4.3.9	get_type(void) const	29
5.4.3.10	kmer(length_t k) const	30
5.4.3.11	rc(void) const	30
5.4.3.12	size(void) const	30
5.4.3.13	subseq(length_t start, length_t stop) const	30
5.4.3.14	translate(unsigned short orf)	31
5.4.3.15	translate(void)	31
5.4.3.16	validate(void) const	31
5.4.3.17	window(length_t width, length_t increment) const	31
5.4.4	Friends And Related Function Documentation	32
5.4.4.1	operator"!="	32
5.4.4.2	operator<	32
5.4.4.3	operator<<	32
5.4.4.4	operator<=	33
5.4.4.5	operator==	33
5.4.4.6	operator>	33
5.4.4.7	operator>=	34
5.5	FASTX::SW Class Reference	34

5.5.1	Detailed Description	35
5.5.2	Constructor & Destructor Documentation	35
5.5.2.1	SW(std::string s1, std::string s2)	35
5.5.3	Member Function Documentation	35
5.5.3.1	print_matrix(std::ostream &out)	35
5.5.3.2	print_path(std::ostream &out)	35
5.5.4	Friends And Related Function Documentation	35
5.5.4.1	operator<<	35
5.6	FASTX::SW_path Struct Reference	36
5.6.1	Detailed Description	36
5.6.2	Member Data Documentation	36
5.6.2.1	i	36
5.6.2.2	j	36
5.7	FASTX::Wrap Class Reference	36
5.7.1	Detailed Description	37
5.7.2	Constructor & Destructor Documentation	37
5.7.2.1	Wrap(const Record &rec)	37
5.7.2.2	Wrap(const Record &rec, unsigned int width)	37
5.7.3	Member Function Documentation	38
5.7.3.1	operator()(std::ostream &outstream) const	38
5.7.4	Friends And Related Function Documentation	38
5.7.4.1	operator<<	38
6	Example Documentation	39
6.1	count_freq.cpp	39
6.2	enumerate_iupac.cpp	39
6.3	generate_kmers.cpp	40
6.4	generate_window.cpp	40
6.5	numeric_qual.cpp	41
6.6	print_wrapped.cpp	41
6.7	reverse_complement.cpp	42
6.8	translate.cpp	42
	Index	43

Chapter 1

FASTX-IO

No BS FASTA/Q input/output and other operations

This library is meant as a simple, easy-to-use and effective means of reading, writing and manipulating [FASTA] and [FASTQ] files. The emphasis of the library lies on a balance of speed and correctness, therefore it aims to be IUPAC compliant, supporting ambiguous nucleotide letters (e.g. `N`) and DNA, RNA, as well as amino acid containing data.

It is meant as building blocks for larger applications.

Features

See also the examples at the end of this document

- Easy I/O from files
- Automatic detecting and reading of `gzip` and `bzip2` compressed files
- Built-in support for many common operations
 - Simple generation of sub-records:
 - * k-mers
 - * sliding windows
 - * sub-sequences
 - Translating DNA/RNA to amino acids
 - Reverse complementing
 - Tabulating counts of nucleotide frequencies
 - Fast enumeration of all possible sequences from ambiguous ones (e.g. primers)
- Automatic validation of records
- Optional compilation without record validation for extra speed

Speed

`fastxio` performs minimal validation of the records and makes use of many speed efficient data structures to perform operations. Record validation can be turned off completely during compilation, though only advisable if the data has been validated already before (e.g. by `fastQValidator`).

Building the library

`cmake` is required to build the library, `doxygen` to build the documentation.

```
1 mkdir build
2 cd build
3 cmake -DCMAKE_BUILD_TYPE=Release ..
4 make
```

In order to build the library without record validation, pass the `NO_CHECKING` flag to `cmake`

```
1 mkdir build
2 cd build
3 cmake cmake -DCMAKE_BUILD_TYPE=Release -DNO_CHECKING=ON ..
4 make
```

“

Examples

Counting the frequencies of all records and calculating GC%

Calculating the GC% of a record can often be useful to guess at the locality of a sequence (e.g. rDNA) or at the origin (fungal contamination). `fastxio` employs a nucleotide counting class:

```
1 // Open a file as a reader
2 FASTX::Reader R("p33.fa", DNA_SEQTYPE);
3
4 // Create a NucFrequency object to keep track
5 FASTX::NucFrequency NF;
6
7 // Loop through records in the file
8 while(R.peek() != EOF)
9 {
10     // Get the next FASTQ record
11     FASTX::Record x = R.next();
12     // Add it to the count
13     NF.add(x);
14 }
15
16 // Print the result
17 std::cout << NF;
18
19 // Calculate GC%
20 std::cout << std::setprecision(2) << static_cast<float>( NF['G'] + NF['C'] ) /
21     static_cast<float>( NF['G'] + NF['C'] + NF['A'] + NF['T'] ) << "%" << std::endl;
```

Output:

```
1 A 10024
2 C 4901
3 G 5106
4 N 110
5 T 9959
6 0.33%
```

Enumerating all possible real sequences from an ambiguous primer

Often, when designing primers, these can contain ambiguous nucleotides, but many detection and trimming programs do not support anything but A,C,G and T. Therefore, it can be useful to have a way of generating all (non-redundant) possibilities.

```
1 // Initialize a Record as DNA with an ambiguous character
2 FASTX::Record R("CCGGACTACHVGGGTWTCTAAT", "Primer", DNA_SEQTYPE);
3
4 // Print all possibilities to stdout
5 for(auto r : R.enumerate_iupac())
6 {
7     std::cout << r << std::endl;
8 }
```

Output:

```
1 >Primer_10A_11A_16A
2 CCGGACTACAAGGGTATCTAAT
3 >Primer_10A_11A_16T
4 CCGGACTACAAGGGTTTCTAAT
5 >Primer_10A_11C_16A
6 CCGGACTACACGGGTATCTAAT
7 >Primer_10A_11C_16T
8 CCGGACTACACGGGTTTCTAAT
9 >Primer_10A_11G_16A
10 CCGGACTACAGGGGTATCTAAT
11 >Primer_10A_11G_16T
12 CCGGACTACAGGGTTTCTAAT
13 >Primer_10C_11A_16A
14 CCGGACTACCAGGGTATCTAAT
15 >Primer_10C_11A_16T
16 CCGGACTACCAGGGTTTCTAAT
17 >Primer_10C_11C_16A
18 CCGGACTACCCGGGTATCTAAT
19 >Primer_10C_11C_16T
20 CCGGACTACCCGGGTTTCTAAT
21 >Primer_10C_11G_16A
22 CCGGACTACCGGGGTATCTAAT
23 >Primer_10C_11G_16T
24 CCGGACTACCGGGTTTCTAAT
25 >Primer_10T_11A_16A
26 CCGGACTACTAGGTATCTAAT
27 >Primer_10T_11A_16T
28 CCGGACTACTAGGTTTCTAAT
29 >Primer_10T_11C_16A
30 CCGGACTACTCGGTATCTAAT
31 >Primer_10T_11C_16T
32 CCGGACTACTCGGTTTCTAAT
33 >Primer_10T_11G_16A
34 CCGGACTACTGGGTATCTAAT
35 >Primer_10T_11G_16T
36 CCGGACTACTGGGTTTCTAAT
```


Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

FASTX	
Auxiliary functions	9

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

FASTX::GData	This global struct holds all translation/complementation tables	19
FASTX::NucFrequency	Helper class to count nucleotide frequencies	22
FASTX::Reader	Class to automatically read records from files	24
FASTX::Record	General purpose FASTA and FASTQ input class	25
FASTX::SW	A Smith-Waterman class to perform local alignments	34
FASTX::SW_path	Helper struct to store the matrix index in the Smith Waterman scoring matrix	36
FASTX::Wrap	Helper class to wrap FASTA when printing	36

Chapter 4

Namespace Documentation

4.1 FASTX Namespace Reference

Auxiliary functions.

Classes

- struct [GData](#)
This global struct holds all translation/complementation tables.
- class [NucFrequency](#)
Helper class to count nucleotide frequencies.
- class [Reader](#)
Class to automatically read records from files.
- class [Record](#)
General purpose FASTA and FASTQ input class.
- class [SW](#)
A Smith-Waterman class to perform local alignments.
- struct [SW_path](#)
Helper struct to store the matrix index in the Smith Waterman scoring matrix.
- class [Wrap](#)
Helper class to wrap FASTA when printing.

Typedefs

- typedef unsigned long [length_t](#)
Type to store length information of sequences.
- typedef long int [score_t](#)
Type to store scores in Smith Waterman matrix.

Functions

- `std::basic_string< char >::const_iterator get_whitespace (std::string &str)`
Return an iterator to the first whitespace character.
- unsigned short `scan_phred (std::istream &instream)`
Detect the PHRED encoding offset.
- unsigned short `scan_phred (const char *infile)`
Detect the PHRED encoding offset.
- bool `is_gzip (const char *input)`
Detect if a file is GZip compressed.
- bool `is_bzip2 (const char *input)`
Detect if a file is BZip2 compressed.
- bool `is_sequence_char (const char test, char seqtype)`
Check if a character is an allowed sequence character in DNA, RNA, or amino acid sequence.
- void `recursive_iupac_enum (std::set< Record > &set, Record &rec, const std::map< char, std::vector< char > > &translation_table, std::set< char > &unambiguous_nuc)`
Recursively enumerate all possibilities from ambiguous IUPAC DNA/RNA characters.
- `std::ostream & operator<< (std::ostream &ostream, const NucFrequency nuc)`
Printing method (prints: Nucleotide< TAB> Count)
- `std::ostream & operator<< (std::ostream &ostream, const Record &rec)`
Print the record to a sink.
- `std::ostream & operator<< (std::ostream &ostream, Wrap rec)`
Print a FASTA record wrapped to a specified column number (internal function)
- bool `operator< (const Record &a, const Record &b)`
Comparison less than operator for [Record](#) classes.
- bool `operator> (const Record &a, const Record &b)`
Comparison greater than operator for [Record](#) classes.
- bool `operator== (const Record &a, const Record &b)`
Comparison equal operator for [Record](#) classes.
- bool `operator!= (const Record &a, const Record &b)`
Comparison not equal operator for [Record](#) classes.
- bool `operator>= (const Record &a, const Record &b)`
Comparison greater than or equal operator for [Record](#) classes.
- bool `operator<= (const Record &a, const Record &b)`
Comparison less than or equal operator for [Record](#) classes.
- `score_t sw_max_neighbour (score_t a, score_t b, score_t c)`
Calculate the highest scoring neighbor in the [SW](#) matrix.
- `std::ostream & operator<< (std::ostream &out, const SW sw)`
Print the local alignment.

Variables

- [GData](#) global

4.1.1 Detailed Description

Auxiliary functions.

The functions here provide auxiliary features to the classes, most notably detecting formats and parsing characters.

4.1.2 Function Documentation

4.1.2.1 `std::basic_string< char >::const_iterator FASTX::get_whitespace (std::string & str)`

Return an iterator to the first whitespace character.

Parameters

<i>str</i>	An input string.
------------	------------------

Returns

Iterator to the first whitespace character in the string.

4.1.2.2 bool FASTX::is_bzip2 (const char * *input*)

Detect if a file is BZip2 compressed.

Specifically, this function tests the first three bytes (magic number) for the bzlib signature. While this method is generally correct, there is a possibility for false positives/negatives.

Parameters

<i>input</i>	Path to the file
--------------	------------------

Returns

True if BZip2 compressed, false otherwise.

4.1.2.3 bool FASTX::is_gzip (const char * *input*)

Detect if a file is GZip compressed.

Specifically, this function tests the first three bytes (magic number) for the zlib signature. While this method is generally correct, there is a possibility for false positives/negatives.

Parameters

<i>input</i>	Path to the file
--------------	------------------

Returns

True if GZip compressed, false otherwise.

4.1.2.4 bool FASTX::is_sequence_char (const char *test*, char *seqtype*)

Check if a character is an allowed sequence character in DNA, RNA, or amino acid sequence.

Parameters

<i>test</i>	The character to be tested
<i>seqtype</i>	The sequence type (macro) DNA_SEQTYPE, RNA_SEQTYPE or AA_SEQTYPE

Returns

True, if the character is allowed, false otherwise.

4.1.2.5 `bool FASTX::operator!=(const Record & a, const Record & b)` `[inline]`

Comparison not equal operator for [Record](#) classes.

The ordering of [Record](#) objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of a is different than b, false otherwise.

4.1.2.6 `bool FASTX::operator<(const Record & a, const Record & b)` `[inline]`

Comparison less than operator for [Record](#) classes.

The ordering of [Record](#) objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of a is alphabetically before b, false otherwise.

4.1.2.7 `std::ostream & FASTX::operator<<(std::ostream & outstream, const NucFrequency nuc)`

Printing method (prints: Nucleotide<TAB>Count)

Parameters

<i>outstream</i>	A reference to an output stream
<i>nuc</i>	A NucFrequency object

4.1.2.8 `std::ostream & FASTX::operator<<(std::ostream & outstream, const Record & rec)`

Print the record to a sink.

The record can be printed to a sink derived from an `std::ostream` class. FASTA/FASTQ formatting is handled automatically via the type of the read.

Parameters

<i>ostream</i>	A sink to print to.
<i>rec</i>	A record object

4.1.2.9 `std::ostream & FASTX::operator<< (std::ostream & out, const SW sw)`

Print the local alignment.

Parameters

<i>out</i>	The sink, derived from <code>std::ostream</code>
<i>sw</i>	The Smith-Waterman class

4.1.2.10 `std::ostream & FASTX::operator<< (std::ostream & ostream, Wrap rec)`

Print a FASTA record wrapped to a specified column number (internal function)

Overloaded operator<< to print to an ostream.

Parameters

<i>ostream</i>	The output stream
<i>rec</i>	The wrapped Record object

4.1.2.11 `bool FASTX::operator<= (const Record & a, const Record & b)` `[inline]`

Comparison less than or equal operator for [Record](#) classes.

The ordering of [Record](#) objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of a is alphabetically before b, or identical to b, false otherwise

4.1.2.12 `bool FASTX::operator== (const Record & a, const Record & b)` `[inline]`

Comparison equal operator for [Record](#) classes.

The ordering of [Record](#) objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is identical to *b*, false otherwise.

4.1.2.13 `bool FASTX::operator> (const Record & a, const Record & b)` `[inline]`

Comparison greater than operator for [Record](#) classes.

The ordering of [Record](#) objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is alphabetically after *b*, false otherwise.

4.1.2.14 `bool FASTX::operator>= (const Record & a, const Record & b)` `[inline]`

Comparison greater than or equal operator for [Record](#) classes.

The ordering of [Record](#) objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is alphabetically after *b*, or identical to *b*, false otherwise

4.1.2.15 `void FASTX::recursive_iupac_enum (std::set< Record > & set, Record & rec, const std::map< char, std::vector< char > > & translation_table, std::set< char > & unambiguous_nuc)`

Recursively enumerate all possibilities from ambiguous IUPAC DNA/RNA characters.

This function not intended for the end user. See rather the `enumerate()` method of the [Record](#) class.

Parameters

<i>set</i>	A set containing fully enumerated sequences
<i>rec</i>	The record to be enumerated
<i>translation_table</i>	A table that translates ambiguous nucleotides to the possibilities
<i>unambiguous_nuc</i>	A set that contains the unambiguous characters

See also

`Record.enumerate()`

4.1.2.16 unsigned short FASTX::scan_phred (std::istream & *instream*)

Detect the PHRED encoding offset.

Parameters

<i>instream</i>	Input stream
-----------------	--------------

Returns

PHRED offset, 33 or 64

Examples:

[numeric_qual.cpp](#).

4.1.2.17 unsigned short FASTX::scan_phred (const char * *infile*)

Detect the PHRED encoding offset.

Parameters

<i>infile</i>	Path to the FASTQ file (can be GZip, BZip2 compressed)
---------------	--

Returns

PHRED offset, 33 or 64

4.1.2.18 score_t FASTX::sw_max_neighbour (score_t *a*, score_t *b*, score_t *c*)

Calculate the highest scoring neighbor in the [SW](#) matrix.

To find the optimal path in reverse from the maximal overall value of the matrix. In case of ties, the function prioritizes the upper left diagonal, then the upper row, and lastly the left column.

Parameters

<i>a</i>	The upper left diagonal value
<i>b</i>	The upper row value
<i>c</i>	The left column value

Returns

The value of the highest scoring neighbor

4.1.3 Variable Documentation

4.1.3.1 GData FASTX::global

Global variable of translation tables

Chapter 5

Class Documentation

5.1 FASTX::GData Struct Reference

This global struct holds all translation/complementation tables.

```
#include <fastxio_common.h>
```

Static Public Attributes

- static const std::set< char > [nuc_alphabet](#)
All nucleotides (IUPAC notation)
- static const std::set< char > [aa_alphabet](#)
All amino acid codes (IUPAC notation)
- static const std::map< char, char > [rc](#)
Reverse complementation table.
- static const std::map< char, std::vector< char > > [enum_iupac_dna](#)
Disambiguation for ambiguous IUPAC DNA/RNA codes.
- static const std::map< char, std::vector< char > > [enum_iupac_rna](#)
Disambiguation for ambiguous IUPAC RNA codes.
- static const std::map< std::string, char > [codon_to_protein_dna](#)
Triplet nucleotide to AA translation (DNA)
- static const std::map< std::string, char > [codon_to_protein_rna](#)
Triplet nucleotide to AA translation (RNA)

5.1.1 Detailed Description

This global struct holds all translation/complementation tables.

5.1.2 Member Data Documentation

5.1.2.1 `const std::set< char > FASTX::GData::aa_alphabet` [static]

Initial value:

```
= {
    'A', 'a', 'B', 'b', 'C', 'c', 'D', 'd', 'E', 'e',
    'F', 'f', 'G', 'g', 'H', 'h', 'I', 'i', 'J', 'j',
    'K', 'k', 'L', 'l', 'M', 'm', 'N', 'n', 'O', 'o',
    'P', 'p', 'Q', 'q', 'R', 'r', 'S', 's', 'T', 't',
    'U', 'u', 'V', 'v', 'W', 'w', 'Y', 'y', 'Z', 'z',
    'X', 'x', '*', '-', '.' }
```

All amino acid codes (IUPAC notation)

5.1.2.2 `const std::map< std::string, char > FASTX::GData::codon_to_protein_dna` [static]

Initial value:

```
= {
    { "TTT", 'F' }, { "TTC", 'F' }, { "TTA", 'L' }, { "TTG", 'L' },
    { "CTT", 'L' }, { "CTC", 'L' }, { "CTA", 'L' }, { "CTG", 'L' },
    { "ATT", 'I' }, { "ATC", 'I' }, { "ATA", 'I' }, { "ATG", 'M' },
    { "GTT", 'V' }, { "GTC", 'V' }, { "GTA", 'V' }, { "GTG", 'V' },
    { "TCT", 'S' }, { "TCC", 'S' }, { "TCA", 'S' }, { "TCG", 'S' },
    { "CCT", 'P' }, { "CCC", 'P' }, { "CCA", 'P' }, { "CCG", 'P' },
    { "ACT", 'T' }, { "ACC", 'T' }, { "ACA", 'T' }, { "ACG", 'T' },
    { "GCT", 'A' }, { "GCC", 'A' }, { "GCA", 'A' }, { "GCG", 'A' },
    { "TAT", 'Y' }, { "TAC", 'Y' }, { "TAA", '.' }, { "TAG", '.' },
    { "CAT", 'H' }, { "CAC", 'H' }, { "CAA", 'Q' }, { "CAG", 'Q' },
    { "AAT", 'N' }, { "AAC", 'N' }, { "AAA", 'K' }, { "AAG", 'N' },
    { "GAT", 'D' }, { "GAC", 'D' }, { "GAA", 'E' }, { "GAG", 'E' },
    { "TGT", 'C' }, { "TGC", 'C' }, { "TGA", '.' }, { "TGG", 'W' },
    { "CGT", 'R' }, { "CGC", 'R' }, { "CGA", 'R' }, { "CGG", 'R' },
    { "AGT", 'S' }, { "AGC", 'S' }, { "AGA", 'R' }, { "AGG", 'R' },
    { "GGT", 'G' }, { "GGC", 'G' }, { "GGA", 'G' }, { "GGG", 'G' }
}
```

Triplett nucleotide to AA translation (DNA)

5.1.2.3 `const std::map< std::string, char > FASTX::GData::codon_to_protein_rna` [static]

Initial value:

```
= {
    { "UUU", 'F' }, { "UUC", 'F' }, { "UUA", 'L' }, { "UUG", 'L' },
    { "CUU", 'L' }, { "CUC", 'L' }, { "CUA", 'L' }, { "CUG", 'L' },
    { "AUU", 'I' }, { "AUC", 'I' }, { "AUA", 'I' }, { "AUG", 'M' },
    { "GUU", 'V' }, { "GUC", 'V' }, { "GUA", 'V' }, { "GUG", 'V' },
    { "UCU", 'S' }, { "UCC", 'S' }, { "UCA", 'S' }, { "UCG", 'S' },
    { "CCU", 'P' }, { "CCC", 'P' }, { "CCA", 'P' }, { "CCG", 'P' },
    { "ACU", 'U' }, { "ACC", 'U' }, { "ACA", 'U' }, { "ACG", 'U' },
    { "GCU", 'A' }, { "GCC", 'A' }, { "GCA", 'A' }, { "GCG", 'A' },
    { "UAU", 'Y' }, { "UAC", 'Y' }, { "UAA", '.' }, { "UAG", '.' },
    { "CAU", 'H' }, { "CAC", 'H' }, { "CAA", 'Q' }, { "CAG", 'Q' },
    { "AAU", 'N' }, { "AAC", 'N' }, { "AAA", 'K' }, { "AAG", 'N' },
    { "GAU", 'D' }, { "GAC", 'D' }, { "GAA", 'E' }, { "GAG", 'E' },
    { "UGU", 'C' }, { "UGC", 'C' }, { "UGA", '.' }, { "UGG", 'W' },
    { "CGU", 'R' }, { "CGC", 'R' }, { "CGA", 'R' }, { "CGG", 'R' },
    { "AGU", 'S' }, { "AGC", 'S' }, { "AGA", 'R' }, { "AGG", 'R' },
    { "GGU", 'G' }, { "GGC", 'G' }, { "GGA", 'G' }, { "GGG", 'G' }
}
```

Triplett nucleotide to AA translation (RNA)

5.1.2.4 `const std::map< char, std::vector< char > > FASTX::GData::enum_iupac_dna` `[static]`

Initial value:

```
= {
    {'R', {'A', 'G'}}, {'Y', {'C', 'T'}}, {'K', {'G', 'T'}},
    {'M', {'A', 'C'}}, {'S', {'C', 'G'}}, {'W', {'A', 'T'}},
    {'B', {'C', 'G', 'T'}}, {'D', {'A', 'G', 'T'}},
    {'H', {'A', 'C', 'T'}}, {'V', {'A', 'C', 'G'}},
    {'N', {'A', 'C', 'T', 'G'}}
}
```

Disambiguation for ambiguous IUPAC DNA/RNA codes.

5.1.2.5 `const std::map< char, std::vector< char > > FASTX::GData::enum_iupac_rna` `[static]`

Initial value:

```
= {
    {'R', {'A', 'G'}}, {'Y', {'C', 'U'}}, {'K', {'G', 'U'}},
    {'M', {'A', 'C'}}, {'S', {'C', 'G'}}, {'W', {'A', 'U'}},
    {'B', {'C', 'G', 'U'}}, {'D', {'A', 'G', 'U'}},
    {'H', {'A', 'C', 'U'}}, {'V', {'A', 'C', 'G'}},
    {'N', {'A', 'C', 'U', 'G'}}
}
```

Disambiguation for ambiguous IUPAC RNA codes.

5.1.2.6 `const std::set< char > FASTX::GData::nuc_alphabet` `[static]`

Initial value:

```
= {
    'A', 'a', 'C', 'c', 'G', 'g', 'T', 't', 'N', 'n',
    'U', 'u', 'R', 'r', 'Y', 'y', 'K', 'k', 'M', 'm',
    'S', 's', 'W', 'w', 'B', 'b', 'D', 'd', 'H', 'h',
    'V', 'v', '-'
}
```

All nucleotides (IUPAC notation)

5.1.2.7 `const std::map< char, char > FASTX::GData::rc` `[static]`

Initial value:

```
= {
    {'A', 'T'}, {'a', 't'}, {'C', 'G'}, {'c', 'g'},
    {'G', 'C'}, {'g', 'c'}, {'T', 'A'}, {'t', 'a'},
    {'N', 'N'}, {'n', 'n'}, {'U', 'A'}, {'u', 'a'},

    {'R', 'Y'}, {'r', 'y'}, {'Y', 'R'}, {'y', 'r'},
    {'K', 'M'}, {'k', 'm'}, {'M', 'K'}, {'m', 'k'},
    {'S', 'S'}, {'s', 's'}, {'W', 'W'}, {'w', 'w'},
    {'B', 'V'}, {'b', 'v'}, {'V', 'B'}, {'v', 'b'},
    {'D', 'H'}, {'d', 'h'}, {'H', 'D'}, {'h', 'd'},
    {'-', '-'}
}
```

Reverse complementation table.

The documentation for this struct was generated from the following files:

- `src/fastxio_common.h`
- `src/fastxio_common.cpp`

5.2 FASTX::NucFrequency Class Reference

Helper class to count nucleotide frequencies.

```
#include <fastxio_nuc_frequency.h>
```

Public Member Functions

- void [add](#) (const std::string &str)
Add all nucleotides in a string to the table.
- void [add](#) (const [Record](#) &str)
Add all nucleotides of a [Record](#) to the table.
- [length_t operator\[\]](#) (char nuc) const
Extract data for a letter.
- std::vector< char > [letters](#) (void) const
Get a vector of the letters in the count table.
- [NucFrequency](#) (void)
NULL constructor.

Friends

- std::ostream & [operator<<](#) (std::ostream &outstream, const [NucFrequency](#) nuc)
Printing method (prints: Nucleotide<TAB>Count)

5.2.1 Detailed Description

Helper class to count nucleotide frequencies.

The [NucFrequency](#) class here is a helper class to compute nucleotide frequency tables. Currently, it is implemented with `std::map` as a red-black BST.

Examples:

[count_freq.cpp](#).

5.2.2 Member Function Documentation

5.2.2.1 void FASTX::NucFrequency::add (const std::string & str)

Add all nucleotides in a string to the table.

Parameters

<i>str</i>	A reference to a string
------------	-------------------------

Examples:

[count_freq.cpp](#).

5.2.2.2 void FASTX::NucFrequency::add (const Record & *str*)

Add all nucleotides of a [Record](#) to the table.

Parameters

<i>str</i>	A reference to a Record object
------------	--

5.2.2.3 std::vector< char > FASTX::NucFrequency::letters (void) const

Get a vector of the letters in the count table.

Returns

A vector of the letters present in the count table

5.2.2.4 length_t FASTX::NucFrequency::operator[] (char *nuc*) const

Extract data for a letter.

Parameters

<i>nuc</i>	The letter of interest
------------	------------------------

Returns

Counts of the letter of interest

5.2.3 Friends And Related Function Documentation

5.2.3.1 std::ostream& operator<< (std::ostream & *ostream*, const NucFrequency *nuc*) [friend]

Printing method (prints: Nucleotide<TAB>Count)

Parameters

<i>ostream</i>	A reference to an output stream
<i>nuc</i>	A NucFrequency object

The documentation for this class was generated from the following files:

- `src/fastxio_nuc_frequency.h`
- `src/fastxio_nuc_frequency.cpp`

5.3 FASTX::Reader Class Reference

Class to automatically read records from files.

```
#include <fastxio_reader.h>
```

Public Member Functions

- [Reader](#) (const char *file, const char seqtype)
File path constructor.
- [Record next](#) ()
Return next record.
- char [peek](#) (void)
Peek the next character.

5.3.1 Detailed Description

Class to automatically read records from files.

This class is mainly a wrapper around the `std::istream&` constructor and is able to detect compression based on the magic number of the file that is passed to it.

Examples:

[count_freq.cpp](#), and [numeric_qual.cpp](#).

5.3.2 Constructor & Destructor Documentation

5.3.2.1 FASTX::Reader::Reader (const char * file, const char seqtype)

File path constructor.

Parameters

<i>file</i>	A path to a file
<i>seqtype</i>	The sequence type: DNA_SEQTYPE, RNA_SEQTYPE, AA_SEQTYPE

5.3.3 Member Function Documentation

5.3.3.1 Record FASTX::Reader::next (void)

Return next record.

Returns

A record object of the next entry in the file

Examples:

[count_freq.cpp](#), and [numeric_qual.cpp](#).

5.3.3.2 char FASTX::Reader::peek (void)

Peek the next character.

This method is a passthrough to the [peek \(\)](#) method of the `std::istream` class and is mainly meant as a way to check for the end of file character when reading the entire file.

Returns

The next character, or EOF.

Examples:

[count_freq.cpp](#), and [numeric_qual.cpp](#).

The documentation for this class was generated from the following files:

- `src/fastxio_reader.h`
- `src/fastxio_reader.cpp`

5.4 FASTX::Record Class Reference

General purpose FASTA and FASTQ input class.

```
#include <fastxio_record.h>
```

Public Member Functions

- `std::string get_seq (void) const`
Get the sequence.
- `std::string get_qual (void) const`
Get the quality as ASCII encoded characters.
- `std::string get_id (void) const`
Get the ID.
- `char get_type (void) const`
Get the type of the record.
- `std::shared_ptr< const std::string > get_seq_ptr (void) const`
Get a shared pointer to sequence.
- `std::shared_ptr< const std::string > get_qual_ptr (void) const`
Get a shared pointer to quality.
- `length_t size (void) const`

- Get the length of the record.*
- [Record](#) (std::istream &input, char seqtype)
 - Constructor from an istream.*
- [Record](#) (const std::string &fasta, const std::string &id, char seqtype)
 - Constructor (FASTA) from sequence and ID.*
- [Record](#) (const std::string &fastq, const std::string &id, const std::string &qual, char seqtype)
 - Constructor (FASTQ) from sequence, ID and qual.*
- [Record translate](#) (unsigned short orf)
 - Translate a record to an AA record, one ORF.*
- std::vector< [Record](#) > [translate](#) (void)
 - Translate a record to an AA record, all ORFs.*
- [NucFrequency count_freq](#) (void)
 - Tabulate all nucleotides and get the frequencies.*
- [Record subseq](#) ([length_t](#) start, [length_t](#) stop) const
 - Get a sub-sequence.*
- [Record rc](#) (void) const
 - Reverse complement a record.*
- std::vector< [Record](#) > [kmer](#) ([length_t](#) k) const
 - Get all k-mers of length k.*
- std::vector< [Record](#) > [window](#) ([length_t](#) width, [length_t](#) increment) const
 - Create a sliding window along the record.*
- std::vector< unsigned short > [get_numeric_qual](#) (void) const
 - Get a numeric representation of the quality values.*
- std::set< [Record](#) > [enumerate_iupac](#) (void)
 - Enumerate all possible sequences from ambiguous IUPAC sequences.*
- bool [validate](#) (void) const
 - Validate the record.*

Friends

- class **Wrap**
- class **NucFrequency**
- std::ostream & [operator<<](#) (std::ostream &ostream, const [Record](#) &rec)
 - Print the record to a sink.*
- bool [operator<](#) (const [Record](#) &a, const [Record](#) &b)
 - Comparison less than operator for [Record](#) classes.*
- bool [operator>](#) (const [Record](#) &a, const [Record](#) &b)
 - Comparison greater than operator for [Record](#) classes.*
- bool [operator==](#) (const [Record](#) &a, const [Record](#) &b)
 - Comparison equal operator for [Record](#) classes.*
- bool [operator!=](#) (const [Record](#) &a, const [Record](#) &b)
 - Comparison not equal operator for [Record](#) classes.*
- bool [operator>=](#) (const [Record](#) &a, const [Record](#) &b)
 - Comparison greater than or equal operator for [Record](#) classes.*
- bool [operator<=](#) (const [Record](#) &a, const [Record](#) &b)
 - Comparison less than or equal operator for [Record](#) classes.*

5.4.1 Detailed Description

General purpose FASTA and FASTQ input class.

This library supports simple I/O, parsing and common operations on FASTQ and FASTA files (abbreviated as [FASTX](#)). Its main emphases lie on *speed* and *correctness*. All IUPAC characters are supported, as are upper- and lowercase notation.

Examples:

[count_freq.cpp](#), [enumerate_iupac.cpp](#), [generate_kmers.cpp](#), [generate_window.cpp](#), [numeric_qual.cpp](#), [print_wrapped.cpp](#), [reverse_complement.cpp](#), and [translate.cpp](#).

5.4.2 Constructor & Destructor Documentation

5.4.2.1 FASTX::Record::Record (std::istream & *input*, char *seqtype* = DNA_SEQTYPE)

Constructor from an istream.

Parameters

<i>input</i>	An std::istream derived source to read from
<i>seqtype</i>	The sequence type (macro), DNA_SEQTYPE, RNA_SEQTYPE, AA_SEQTYPE

5.4.2.2 FASTX::Record::Record (const std::string & *fasta*, const std::string & *id*, char *seqtype* = DNA_SEQTYPE)

Constructor (FASTA) from sequence and ID.

Parameters

<i>fasta</i>	The sequence for the record
<i>id</i>	The ID for the record
<i>seqtype</i>	The sequence type (macro), DNA_SEQTYPE, RNA_SEQTYPE, AA_SEQTYPE

5.4.2.3 FASTX::Record::Record (const std::string & *fastq*, const std::string & *id*, const std::string & *qual*, char *seqtype* = DNA_SEQTYPE)

Constructor (FASTQ) from sequence, ID and qual.

Parameters

<i>fastq</i>	The sequence for the record
<i>id</i>	The ID for the record
<i>qual</i>	The quality string
<i>seqtype</i>	The sequence type (macro), DNA_SEQTYPE, RNA_SEQTYPE, AA_SEQTYPE

5.4.3 Member Function Documentation

5.4.3.1 NucFrequency FASTX::Record::count_freq (void)

Tabulate all nucleotides and get the frequencies.

Returns

A class containing nucleotide frequencies.

5.4.3.2 std::set< Record > FASTX::Record::enumerate_iupac (void)

Enumerate all possible sequences from ambiguous IUPAC sequences.

See also the `enumerate_iupac` example.

Returns

A set of all possible unambiguous sequences

Examples:

[enumerate_iupac.cpp](#).

5.4.3.3 std::string FASTX::Record::get_id (void) const [inline]

Get the ID.

Returns

The ID.

5.4.3.4 std::vector< unsigned short > FASTX::Record::get_numeric_qual (void) const

Get a numeric representation of the quality values.

The function returns a vector of numeric values of the ASCII encoded PHRED scores. The values are still containing the PHRED offset. See the `scan_phred()` function in `fastxio_auxiliary`.

Returns

A vector of numeric quality values

Examples:

[numeric_qual.cpp](#).

5.4.3.5 `std::string FASTX::Record::get_qual (void) const` `[inline]`

Get the quality as ASCII encoded characters.

Returns

The quality values.

5.4.3.6 `std::shared_ptr<const std::string> FASTX::Record::get_qual_ptr (void) const` `[inline]`

Get a shared pointer to quality.

Returns

A shared pointer to the quality string of the object.

5.4.3.7 `std::string FASTX::Record::get_seq (void) const` `[inline]`

Get the sequence.

Returns

The sequence of the object.

5.4.3.8 `std::shared_ptr<const std::string> FASTX::Record::get_seq_ptr (void) const` `[inline]`

Get a shared pointer to sequence.

Returns

A shared pointer to the sequence of the object.

5.4.3.9 `char FASTX::Record::get_type (void) const` `[inline]`

Get the type of the record.

The return of this function is a byte (char), which contains the bit-encoded type of the record. The bits are as follows:

Bit	Meaning
00001	The record is FASTQ formatted
00010	The record is FASTA formatted
00100	The record is a DNA sequence
01000	The record is an RNA sequence
10000	The record is an amino acid sequence

As a default, records are initialized as 00101, i.e. FASTQ/DNA.

Returns

The bit encoded type of the record

5.4.3.10 `std::vector< Record > FASTX::Record::kmer (length_t k) const`

Get all k-mers of length k.

Parameters

<i>k</i>	The k-mer length.
----------	-------------------

Returns

A vector containing all k-mer records.

Examples:

[generate_kmers.cpp](#).

5.4.3.11 `Record FASTX::Record::rc (void) const`

Reverse complement a record.

Returns

A reverse complement of the record.

Examples:

[reverse_complement.cpp](#).

5.4.3.12 `length_t FASTX::Record::size (void) const` `[inline]`

Get the length of the record.

Returns

The length of the record.

Examples:

[numeric_qual.cpp](#).

5.4.3.13 `Record FASTX::Record::subseq (length_t start, length_t stop) const`

Get a sub-sequence.

Parameters

<i>start</i>	The (0-offset) start position of the subsequence.
<i>stop</i>	The stop position.

Returns

The created subsequence.

5.4.3.14 `Record FASTX::Record::translate (unsigned short orf)`

Translate a record to an AA record, one ORF.

Parameters

<i>orf</i>	The open reading frame: 0 for the first base, 1 or 2 for the next
------------	---

Returns

A translated (FASTA) record

Examples:

[translate.cpp](#).

5.4.3.15 `std::vector< Record > FASTX::Record::translate (void)`

Translate a record to an AA record, all ORFs.

See also the example `translate.cpp`.

Returns

A vector of all translated ORFs.

5.4.3.16 `bool FASTX::Record::validate (void) const`

Validate the record.

The sequence and quality lengths need to be the same length, the quality values cannot be outside the allowed range, the sequence is tested for disallowed characters.

Returns

True, if the record is valid, false otherwise

5.4.3.17 `std::vector< Record > FASTX::Record::window (length_t width, length_t increment) const`

Create a sliding window along the record.

Parameters

<i>width</i>	The width of the sliding window.
<i>increment</i>	The increment of the window.

Returns

A vector of the sliding windows.

Examples:

[generate_window.cpp](#).

5.4.4 Friends And Related Function Documentation

5.4.4.1 `bool operator!=(const Record & a, const Record & b)` [friend]

Comparison not equal operator for `Record` classes.

The ordering of `Record` objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of a is different than b, false otherwise.

5.4.4.2 `bool operator<(const Record & a, const Record & b)` [friend]

Comparison less than operator for `Record` classes.

The ordering of `Record` objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of a is alphabetically before b, false otherwise.

5.4.4.3 `std::ostream& operator<<(std::ostream & ostream, const Record & rec)` [friend]

Print the record to a sink.

The record can be printed to a sink derived from an `std::ostream` class. FASTA/FASTQ formatting is handled automatically via the type of the read.

Parameters

<i>ostream</i>	A sink to print to.
<i>rec</i>	A record object

5.4.4.4 `bool operator<= (const Record & a, const Record & b)` [friend]

Comparison less than or equal operator for `Record` classes.

The ordering of `Record` objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is alphabetically before *b*, or identical to *b*, false otherwise

5.4.4.5 `bool operator== (const Record & a, const Record & b)` [friend]

Comparison equal operator for `Record` classes.

The ordering of `Record` objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is identical to *b*, false otherwise.

5.4.4.6 `bool operator> (const Record & a, const Record & b)` [friend]

Comparison greater than operator for `Record` classes.

The ordering of `Record` objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is alphabetically after *b*, false otherwise.

5.4.4.7 `bool operator>= (const Record & a, const Record & b)` [*friend*]

Comparison greater than or equal operator for *Record* classes.

The ordering of *Record* objects is alphabetical by their respective sequences.

Parameters

<i>a</i>	First record.
<i>b</i>	Second record.

Returns

True if the sequence of *a* is alphabetically after *b*, or identical to *b*, false otherwise

The documentation for this class was generated from the following files:

- `src/fastxio_record.h`
- `src/fastxio_record.cpp`

5.5 FASTX::SW Class Reference

A Smith-Waterman class to perform local alignments.

```
#include <fastxio_smith_waterman.h>
```

Public Member Functions

- *SW* (std::string s1, std::string s2)
Perform a Smith-Waterman alignment of two strings.
- void *print_matrix* (std::ostream &out)
Print the scoring matrix.
- void *print_path* (std::ostream &out)
Print the path that was calculated through the matrix.

Friends

- std::ostream & *operator<<* (std::ostream &out, const *SW* sw)
Print the local alignment.

5.5.1 Detailed Description

A Smith-Waterman class to perform local alignments.

This class should be treated as a very simple, early version of an [SW](#) aligner. It performs very basic local alignments of short strings. For long strings, memory efficiency should be a concern as it builds a scoring matrix ($N_a * N_b$), where N_a is the length of string A, and N_b is the length of string B.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 FASTX::SW::SW (std::string *s1*, std::string *s2*)

Perform a Smith-Waterman alignment of two strings.

Parameters

<i>s1</i>	A DNA string
<i>s2</i>	Another DNA string

5.5.3 Member Function Documentation

5.5.3.1 void FASTX::SW::print_matrix (std::ostream & *out*)

Print the scoring matrix.

Parameters

<i>out</i>	The sink, derived from <code>std::ostream</code>
------------	--

5.5.3.2 void FASTX::SW::print_path (std::ostream & *out*)

Print the path that was calculated through the matrix.

Parameters

<i>out</i>	The sink, derived from <code>std::ostream</code>
------------	--

5.5.4 Friends And Related Function Documentation

5.5.4.1 std::ostream& operator<< (std::ostream & *out*, const SW *sw*) `[friend]`

Print the local alignment.

Parameters

<i>out</i>	The sink, derived from <code>std::ostream</code>
<i>sw</i>	The Smith-Waterman class

The documentation for this class was generated from the following files:

- `src/fastxio_smith_waterman.h`
- `src/fastxio_smith_waterman.cpp`

5.6 FASTX::SW_path Struct Reference

Helper struct to store the matrix index in the Smith Waterman scoring matrix.

```
#include <fastxio_smith_waterman.h>
```

Public Attributes

- unsigned long `i` = 0
- unsigned long `j` = 0

5.6.1 Detailed Description

Helper struct to store the matrix index in the Smith Waterman scoring matrix.

5.6.2 Member Data Documentation

5.6.2.1 unsigned long FASTX::SW_path::i = 0

Row index in matrix

5.6.2.2 unsigned long FASTX::SW_path::j = 0

Column index in matrix

The documentation for this struct was generated from the following file:

- `src/fastxio_smith_waterman.h`

5.7 FASTX::Wrap Class Reference

Helper class to wrap FASTA when printing.

```
#include <fastxio_record.h>
```

Public Member Functions

- [Wrap](#) (const [Record](#) &rec)
Constructor from const [Record](#).
- [Wrap](#) (const [Record](#) &rec, unsigned int width)
Constructor from const [Record](#) with specified width.
- std::ostream & [operator\(\)](#) (std::ostream &ostream) const
Overload of operator() to handle formatting and passing to an std::ostream.

Friends

- std::ostream & [operator<<](#) (std::ostream &ostream, [Wrap](#) rec)
Overloaded operator<< to print to an ostream.

5.7.1 Detailed Description

Helper class to wrap FASTA when printing.

Typically, FASTA records are printed to wrap lines at 80 characters per column. This method does just this.

Warning

Note that FASTQ records cannot be printed with this method because they need consistently four lines per record. See the example source `print_wrapped.cpp`.

Examples:

[print_wrapped.cpp](#).

5.7.2 Constructor & Destructor Documentation

5.7.2.1 FASTX::Wrap::Wrap (const Record & rec)

Constructor from const [Record](#).

Parameters

<i>rec</i>	A record object
------------	-----------------

5.7.2.2 FASTX::Wrap::Wrap (const Record & rec, unsigned int width)

Constructor from const [Record](#) with specified width.

Parameters

<i>rec</i>	A record object
<i>width</i>	The column width (Default: 80)

5.7.3 Member Function Documentation

5.7.3.1 `std::ostream & FASTX::Wrap::operator() (std::ostream & ostream) const`

Overload of `operator()` to handle formatting and passing to an `std::ostream`.

This method is used internally to be called from `operator<<`

Parameters

<i>ostream</i>	The output stream
----------------	-------------------

5.7.4 Friends And Related Function Documentation

5.7.4.1 `std::ostream& operator<< (std::ostream & ostream, Wrap rec) [friend]`

Overloaded `operator<<` to print to an `ostream`.

Parameters

<i>ostream</i>	The output stream
<i>rec</i>	The wrapped Record object

The documentation for this class was generated from the following files:

- `src/fastxio_record.h`
- `src/fastxio_record.cpp`

Chapter 6

Example Documentation

6.1 count_freq.cpp

Examples

```
#include <iostream>
#include <iomanip> //setprecision
#include <fastxio_common.h>
#include <fastxio_record.h>
#include <fastxio_reader.h>
#include <fastxio_nuc_frequency.h>

int main(int argc, char ** argv)
{
    // Open a file as a reader
    FASTX::Reader R("p33.fa", DNA_SEQTYPE);

    // Create a NucFrequency object to keep track
    FASTX::NucFrequency NF;

    // Loop through records in the file
    while(R.peek() != EOF)
    {
        // Get the next FASTQ record
        FASTX::Record x = R.next();
        // Add it to the count
        NF.add(x);
    }

    // Print the result
    std::cout << NF;

    // Calculate GC%
    std::cout << std::setprecision(2) << static_cast<float>( NF['G'] + NF['C'] ) /
        static_cast<float>( NF['G'] + NF['C'] + NF['A'] + NF['T'] ) << "%" << std::endl;

    return 0;
}
```

6.2 enumerate_iupac.cpp

Examples

```
#include <iostream>
#include <fastxio_common.h>
#include <fastxio_record.h>

int main(int argc, char ** argv)
{
    // Initialize a Record as DNA with an ambiguous character
    FASTX::Record R("CCGGACTACHVGGGTWCTAAT", "Primer", DNA_SEQTYPE);

    // Print all possibilities to stdout
    for(auto r : R.enumerate_iupac())
    {
        std::cout << r << std::endl;
    }

    // Initialize a Record as RNA with an ambiguous character
    FASTX::Record R2("NUG", "Test", RNA_SEQTYPE);

    // Print all possibilities to stdout
    for(auto r : R2.enumerate_iupac())
    {
        std::cout << r << std::endl;
    }

    return 0;
}
```

6.3 generate_kmers.cpp

```
#include <iostream>
#include <fastxio_common.h>
#include <fastxio_record.h>

int main(int argc, char ** argv)
{
    // Initialize a Record as DNA
    FASTX::Record R("ACGTAGCTAGCTA", "Test", DNA_SEQTYPE);

    // Print all k-mers of length 3
    for(auto k : R.kmer(3) )
    {
        std::cout << k << std::endl;
    }

    return 0;
}
```

6.4 generate_window.cpp

```
#include <iostream>
#include <fastxio_common.h>
#include <fastxio_record.h>

int main(int argc, char ** argv)
{
    // Initialize a Record as DNA
    FASTX::Record R("ACGTAGCTAGCTA", "Test", DNA_SEQTYPE);

    // Print 3 bases every 2 bases
    for(auto k : R.window(3,2) )
    {
        std::cout << k << std::endl;
    }

    return 0;
}
```

6.5 numeric_qual.cpp

```

#include <iostream>
#include <algorithm> //sort
#include <fastxio_common.h>
#include <fastxio_auxiliary.h> //scan_phred
#include <fastxio_record.h>
#include <fastxio_reader.h>

// A quick and dirty function to calculate percentiles, min, max
std::vector<float> fivenum(std::vector<float> input)
{
    float min = input[0];
    float max = input[input.size() - 1];
    float p25 = input[static_cast<int>(
        static_cast<float>(
            input.size() * 0.25 + 1))];
    float p75 = input[static_cast<int>(
        static_cast<float>(
            input.size() * 0.75 + 1))];
    float p50 = input[static_cast<int>(
        static_cast<float>(
            input.size() * 0.5 + 1))];
    return std::vector<float>{min, p25, p50, p75, max};
}

int main(int argc, char ** argv)
{
    // Get the PHRED offset
    unsigned short offset = FASTX::scan_phred("p33.fa");

    // Open a file as a reader
    FASTX::Reader R("p33.fa", DNA_SEQTYPE);

    std::vector<float> quals; // Aggregate quals

    // Loop through records in the file
    while(R.peek() != EOF)
    {
        // Get the next FASTQ record
        FASTX::Record x = R.next();

        unsigned short qsum = 0; // sum of qualities per read
        for(unsigned short q : x.get_numeric_qual() )
        {
            qsum += (q - offset);
        }
        // Save the mean quality of the read
        quals.push_back( static_cast<float>(qsum) /
            static_cast<float>(x.size() ));
    }

    // Sort the data
    std::sort(quals.begin(), quals.end());

    // Calculate stats
    std::vector<float> f = fivenum(quals);

    std::cout <<
        "Min: " << f[0] << std::endl <<
        "25%: " << f[1] << std::endl <<
        "50%: " << f[2] << std::endl <<
        "75%: " << f[3] << std::endl <<
        "Max: " << f[4] << std::endl;

    return 0;
}

```

6.6 print_wrapped.cpp

Examples

```

#include <iostream>
#include <string>
#include <fastxio_common.h>
#include <fastxio_record.h>

```

```

int main(int argc, char ** argv)
{
    // First, create a long string for the sequence (900nt)
    std::string seq;
    for(unsigned int i = 0; i < 300; i++)
    {
        seq += "TAA";
    }

    // Initialize a Record as DNA
    FASTX::Record R(seq, "Test", DNA_SEQTYPE);

    // Print wrapped to std::cout
    std::cout << FASTX::Wrap(R) << std::endl;

    // The column width can be specified
    std::cout << FASTX::Wrap(R, 42) << std::endl;

    return 0;
}

```

6.7 reverse_complement.cpp

```

#include <iostream>
#include <fastxio_common.h>
#include <fastxio_record.h>

int main(int argc, char ** argv)
{
    // Initialize a Record as DNA
    FASTX::Record R("ACGTAGCTAGCTA", "Test", DNA_SEQTYPE);

    // Print the original
    std::cout << R << std::endl;

    // Print the reverse complement
    std::cout << R.rc() << std::endl;

    return 0;
}

```

6.8 translate.cpp

```

#include <iostream>
#include <fastxio_common.h>
#include <fastxio_record.h>

int main(int argc, char ** argv)
{
    // Initialize a Record as DNA
    FASTX::Record R("ACGTAGCTAGCTA", "Test", DNA_SEQTYPE);

    // Print all ORFs
    for(auto k : R.translate() )
    {
        std::cout << k << std::endl;
    }

    // Print only the first ORF
    std::cout << R.translate(0) << std::endl;

    return 0;
}

```


Index

aa_alphabet
 FASTX::GData, 20

add
 FASTX::NucFrequency, 22, 23

codon_to_protein_dna
 FASTX::GData, 20

codon_to_protein_rna
 FASTX::GData, 20

count_freq
 FASTX::Record, 28

enum_iupac_dna
 FASTX::GData, 20

enum_iupac_rna
 FASTX::GData, 21

enumerate_iupac
 FASTX::Record, 28

FASTX::GData, 19
 aa_alphabet, 20
 codon_to_protein_dna, 20
 codon_to_protein_rna, 20
 enum_iupac_dna, 20
 enum_iupac_rna, 21
 nuc_alphabet, 21
 rc, 21

FASTX::NucFrequency, 22
 add, 22, 23
 letters, 23
 operator<<, 23
 operator[], 23

FASTX::Reader, 24
 next, 24
 peek, 25
 Reader, 24

FASTX::Record, 25
 count_freq, 28
 enumerate_iupac, 28
 get_id, 28
 get_numeric_qual, 28
 get_qual, 28
 get_qual_ptr, 29
 get_seq, 29
 get_seq_ptr, 29
 get_type, 29
 kmer, 30
 operator!=, 32
 operator<, 32
 operator<<, 32
 operator<=, 33
 operator>, 33
 operator>=, 34
 operator==, 33
 rc, 30
 Record, 27
 size, 30
 subseq, 30
 translate, 31
 validate, 31
 window, 31

FASTX::SW_path, 36
 i, 36
 j, 36

FASTX::SW, 34
 operator<<, 35
 print_matrix, 35
 print_path, 35
 SW, 35

FASTX::Wrap, 36
 operator<<, 38
 operator(), 38
 Wrap, 37

FASTX, 9
 get_whitespace, 11
 global, 17
 is_bzip2, 12
 is_gzip, 12
 is_sequence_char, 12
 operator!=, 13
 operator<, 13
 operator<<, 13, 14
 operator<=, 14
 operator>, 15
 operator>=, 15
 operator==, 14
 recursive_iupac_enum, 15
 scan_phred, 16
 sw_max_neighbour, 16

get_id
 FASTX::Record, 28

get_numeric_qual
 FASTX::Record, 28

get_qual
 FASTX::Record, 28

get_qual_ptr
 FASTX::Record, 29

get_seq
 FASTX::Record, 29

- get_seq_ptr
 - FASTX::Record, [29](#)
- get_type
 - FASTX::Record, [29](#)
- get_whitespace
 - FASTX, [11](#)
- global
 - FASTX, [17](#)
- i
 - FASTX::SW_path, [36](#)
- is_bzip2
 - FASTX, [12](#)
- is_gzip
 - FASTX, [12](#)
- is_sequence_char
 - FASTX, [12](#)
- j
 - FASTX::SW_path, [36](#)
- kmer
 - FASTX::Record, [30](#)
- letters
 - FASTX::NucFrequency, [23](#)
- next
 - FASTX::Reader, [24](#)
- nuc_alphabet
 - FASTX::GData, [21](#)
- operator!=
 - FASTX::Record, [32](#)
 - FASTX, [13](#)
- operator<
 - FASTX::Record, [32](#)
 - FASTX, [13](#)
- operator<<
 - FASTX::NucFrequency, [23](#)
 - FASTX::Record, [32](#)
 - FASTX::SW, [35](#)
 - FASTX::Wrap, [38](#)
 - FASTX, [13](#), [14](#)
- operator<=
 - FASTX::Record, [33](#)
 - FASTX, [14](#)
- operator>
 - FASTX::Record, [33](#)
 - FASTX, [15](#)
- operator>=
 - FASTX::Record, [34](#)
 - FASTX, [15](#)
- operator()
 - FASTX::Wrap, [38](#)
- operator==
 - FASTX::Record, [33](#)
 - FASTX, [14](#)
- operator[]
 - FASTX::NucFrequency, [23](#)
- peek
 - FASTX::Reader, [25](#)
- print_matrix
 - FASTX::SW, [35](#)
- print_path
 - FASTX::SW, [35](#)
- rc
 - FASTX::GData, [21](#)
 - FASTX::Record, [30](#)
- Reader
 - FASTX::Reader, [24](#)
- Record
 - FASTX::Record, [27](#)
- recursive_iupac_enum
 - FASTX, [15](#)
- scan_phred
 - FASTX, [16](#)
- size
 - FASTX::Record, [30](#)
- subseq
 - FASTX::Record, [30](#)
- SW
 - FASTX::SW, [35](#)
- sw_max_neighbour
 - FASTX, [16](#)
- translate
 - FASTX::Record, [31](#)
- validate
 - FASTX::Record, [31](#)
- window
 - FASTX::Record, [31](#)
- Wrap
 - FASTX::Wrap, [37](#)