Kmer

0

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Kmer Class Reference

It represents a list of k consecutive nucleotides of a DNA or RNA sequence. Each nucleotide is represented with a character like 'A', 'C', 'G', 'T', 'U'.

```
#include <Kmer.h>
```

Public Member Functions

- Kmer (int k=1)

  It builds a Kmer object using a string with `k` characters (nucleotides). Each character will be set to the value `MISSING_NUCLEOTIDE`.
- Kmer (const std::string &text)

  It builds a Kmer object with the characters in the string `text` representing the list of nucleotides of the new Kmer.
- int getK () const

  Returns the number of nucleotides in this Kmer. Query method.
- int size () const

  Returns the number of nucleotides in this Kmer. Query method.
- std::string toString () const

  Returns a string with a list of characters, each one representing a nucleotide of this Kmer. Query method.
- const char & at (int index) const

  Gets a const reference to the character (nucleotide) at the given position. Query method.
- char & at (int index)

  Gets a reference to the character (nucleotide) at the given position. Modifier method.
- void normalize (const std::string &validNucleotides)

  Normalizes this Kmer. That is, it converts all the characters to uppercase. Then, invalid characters are replaced by the MISSING_NUCLEOTIDE value. Modifier method.
- Kmer complementary (const std::string &nucleotides, const std::string &complementaryNucleotides) const

  Returns the complementary of this Kmer. For example, given the Kmer "TAGAC", the complementary is "ATCTG" (assuming that we use. `nucleotides="ATGC"` and `complementaryNucleotides="TACG"`). If a nucleotide in this object is not in `nucleotides`, then that nucleotide remains the same in the returned kmer. Query method.

Static Public Attributes

- static const char MISSING_NUCLEOTIDE = '_'

3.1.1 Detailed Description

It represents a list of k consecutive nucleotides of a DNA or RNA sequence. Each nucleotide is represented with a character like 'A', 'C', 'G', 'T', 'U'.

Definition at line 28 of file Kmer.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Kmer() `[1/2]`

```
Kmer::Kmer (
              int k = 1 )
```

It builds a Kmer object using a string with `k` characters (nucleotides). Each character will be set to the value `MISSING_NUCLEOTIDE`.

Exceptions

std::invalid_argument	Throws an std::invalid_argument exception if `k` is less or equal than zero

Parameters

k	the number of nucleotides in this Kmer. It should be an integer greater than zero. Input parameter

Definition at line 18 of file Kmer.cpp.

```
{

    if (k <= 0){     // Precondition violation.
        throw std::invalid_argument(
            std::string("Kmer(k): Number of nucleotides in a given") +
                        " Kmer cannot be less than or equal to zero");
    } // Throws error message when precondition is violated

    else{ // k > 0
        for(int i = 0; i < k; i++) {
            _text += MISSING_NUCLEOTIDE;
        } // Fills a "k" sized Kmer with the default character '_'
    }

}
```

3.1.2.2 Kmer() `[2/2]`

```
Kmer::Kmer (
              const std::string & text )
```

It builds a Kmer object with the characters in the string `text` representing the list of nucleotides of the new Kmer.

Exceptions

std::invalid_argument	Throws an std::invalid_argument exception if the given text is empty

Parameters

text	a string with the characters representing the nucleotides for the kmer. It should be a string with at least one character. Input parameter

Definition at line 40 of file Kmer.cpp.

```
{

    if (text.size()==0){    // text is empty
        throw std::invalid_argument( // composed string
            std::string("Kmer(const std::string&text): ") +
                        "text is an empty string");
    }
    _text = text; // if the text is not empty, create a Kmer with it.
}
```

3.1.3 Member Function Documentation

3.1.3.1 at() [1/2]

```
char & Kmer::at (
              int index )
```

Gets a reference to the character (nucleotide) at the given position. Modifier method.

Parameters

index	the position to consider. Input parameter

Exceptions

std::out_of_range	Throws an std::out_of_range exception if the index is not in the range from 0 to k-1 (both included).

Returns

A reference to the character at the given position

Definition at line 87 of file Kmer.cpp.

```
{

    if (index < 0 || index >= getK()){ // Precondition violation
        throw std::out_of_range(       // composed string
            std::string("char& Kmer::at(int index) const: ") +
                        "invalid position " + std::to_string(index));
    }
    else{
        return (_text.at(index)); // Returns a reference to the character
```

```
                                 //  in a given position.
    }
}
```

3.1.3.2 at() [2/2]

```
const char & Kmer::at (
              int index ) const
```

Gets a const reference to the character (nucleotide) at the given position. Query method.

Parameters

index	the position to consider. Input parameter

Exceptions

std::out_of_range	Throws an std::out_of_range exception if the index is not in the range from 0 to k-1 (both included).

Returns

A const reference to the character at the given position

Definition at line 73 of file Kmer.cpp.

```
{

    if (index < 0 || index >= getK()){ // Precondition violation
        throw std::out_of_range(       // composed string
            std::string("const char& Kmer::at(int index) const: ") +
                        "invalid position " + std::to_string(index));
    }
    else{
        return (_text.at(index)); // Returns a constant reference to
                                  // the character in a given position.
    }
}
```

3.1.3.3 complementary()

```
Kmer Kmer::complementary (
              const std::string & nucleotides,
              const std::string & complementaryNucleotides ) const
```

Returns the complementary of this Kmer. For example, given the Kmer "TAGAC", the complementary is "ATCTG" (assuming that we use. `nucleotides="ATGC"` and `complementaryNucleotides="TACG"`). If a nucleotide in this object is not in `nucleotides`, then that nucleotide remains the same in the returned kmer. Query method.

Parameters

nucleotides	A string with the list of possible nucleotides. Input parameter
complementaryNucleotides	A string with the list of complementary nucleotides. Input parameter

Exceptions

std::invalid_argument	Throws an std::invalid_argument exception if the sizes of `nucleotides` and `complementaryNucleotides` are not the same

Returns

The complementary of this Kmer

Definition at line 136 of file Kmer.cpp.

```
{
    if (nucleotides.size() != complementaryNucleotides.size()){ // Different sizes
        throw std::invalid_argument( // composed string
            std::string("Kmer Kmer::complementary(const std::string& nucleotides,") +
                        "const std::string& complementaryNucleotides) const:" +
                        " nucleotides and complementary nucleotides cannot be" +
                        " differently sized");
    }
    else {
        int text_size = _text.size();             // In order to prevent signed problems
        int nucleotide_size = nucleotides.size(); // In order to prevent signed problems
        Kmer complementary_kmer(text_size);       // We create the object we're gonna return
        complementary_kmer._text = _text;

        // We will use two for loops, one to run through our Kmer and another one
        // to, for each individual character, convert it into its complementary.

        for (int i = 0; i < text_size; i++) {
            // We move through our Kmer

            bool complemented = false;
            int pos = 0;

            while (pos < nucleotide_size && !complemented) {

                // We check which Nucleotide in specific we're studying in our Kmer.
                // Depending on which one it is we match it with the corresponding
                // complementary nucleotide. If it doesn't match any of the valid
                // ones, then we will do nothing and it will stay the same.
                // As soon as we have interchanged them, the process is done
                // and we exit the while loops

                if (complementary_kmer.at(i) == nucleotides.at(pos)){
                    complementary_kmer._text.at(i) = complementaryNucleotides.at(pos);
                    complemented = true;
                }
                else pos ++;
            }

        }

        return (complementary_kmer);
    }

}
```

3.1.3.4 getK()

`int Kmer::getK ( ) const`

Returns the number of nucleotides in this Kmer. Query method.

Returns

the number of nucleotides in this Kmer

Definition at line 51 of file Kmer.cpp.

```
{
    return(_text.size()); // Returns the size of the string that is
                          // representing the Kmer.
}
```

3.1.3.5 normalize()

```
void Kmer::normalize (
             const std::string & validNucleotides )
```

Normalizes this Kmer. That is, it converts all the characters to uppercase. Then, invalid characters are replaced by the MISSING_NUCLEOTIDE value. Modifier method.

Parameters

validNucleotides	a string with the list of characters (nucleotides) that should be considered as valid. Input parameter

Definition at line 101 of file Kmer.cpp.

```
{
    // This method performs its function in two steps, first it converts
    // the characters to uppercase then it replaces any invalid character.

    // 1. Converting:
    int size = _text.size(); // In order to prevent signed problems

    for (int i = 0; i < size; i ++) {
        _text.at(i) = std::toupper(_text.at(i));
    }

    // 2. Formatting:


    for (int i = 0; i < size; i++) {
        // First we run through the string


        // Then we check if the character we're looking at matches any of the
        // valid ones, if it doesn't we switch it with our constant
        // MISSING_NUCLEOTIDE character
        // We can efficiently make use of our function.

        if (!IsValidNucleotide(_text.at(i),validNucleotides)){
            _text.at(i) = MISSING_NUCLEOTIDE;
        }

        // If it matches one of the valid ones we simply move on to study
        // the next character.

    }

}
```

3.1.3.6 size()

```
int Kmer::size ( ) const
```

Returns the number of nucleotides in this Kmer. Query method.

Returns

the number of nucleotides in this Kmer

Definition at line 59 of file Kmer.cpp.

```
{
    return(_text.size()); // Returns the size of the string which is
                          // representing the Kmer.
}
```

3.1.3.7 toString()

```
std::string Kmer::toString ( ) const
```

Returns a string with a list of characters, each one representing a nucleotide of this Kmer. Query method.

Returns

The text of this Kmer as a string object

Definition at line 65 of file Kmer.cpp.

```
{
    return(_text); // Simply returns the string as it’s
                   // already representing a Kmer.
}
```

3.1.4 Member Data Documentation

3.1.4.1 MISSING_NUCLEOTIDE

```
const char Kmer::MISSING_NUCLEOTIDE = '_'  [static]
```

A static const character representing an unknown nucleotide. It is used when we do not known which nucleotide we have in a given position of a Kmer

Definition at line 35 of file Kmer.h.

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

- include/Kmer.h
- src/Kmer.cpp

Chapter 4

File Documentation

4.1 include/Kmer.h File Reference

```
#include <iostream>
#include <string>
```

Classes

- class Kmer

  It represents a list of k consecutive nucleotides of a DNA or RNA sequence. Each nucleotide is represented with a character like 'A', 'C', 'G', 'T', 'U'.

Functions

- bool IsValidNucleotide (char nucleotide, const std::string &validNucleotides)

  Checks if the given nucleotide is contained in `validNucleotides`*. That is, if the given character can be considered as part of a genetic sequence.*
- void ToLower (Kmer &kmer)

  Converts to lowercase the characters (nucleotides) of the given Kmer.
- void ToUpper (Kmer &kmer)

  Converts to uppercase the characters (nucleotides) of the given Kmer.

4.1.1 Detailed Description

Author

Silvia Acid Carrillo `acid@decsai.ugr.es`

Andrés Cano Utrera `acu@decsai.ugr.es`

Luis Castillo Vidal `L.Castillo@decsai.ugr.es`

Javier Martínez Baena `jbaena@ugr.es`

Created on 24 October 2023, 14:00

4.1.2 Function Documentation

4.1.2.1 IsValidNucleotide()

```
bool IsValidNucleotide (
            char nucleotide,
            const std::string & validNucleotides )
```

Checks if the given nucleotide is contained in `validNucleotides`. That is, if the given character can be considered as part of a genetic sequence.

Parameters

nucleotide	The nucleotide (a character) to check. Input parameter
validNucleotides	The set of characters that we consider as possible characters in a genetic sequence. Input parameter

Returns

true if the given character is contained in `validNucleotides`; false otherwise

Definition at line 184 of file Kmer.cpp.

```
{
    bool value = false; // First initialize the value as false to go in the loop

    int size = validNucleotides.size(); // In order to avoid signed/unsigned integers

    // We will use a while loop in order to minimize the amount of comparisons
    // so we can exit it if we find the nucleotide to be valid.

    int pos = 0;

    while (pos < size && !value) {

        if (nucleotide == validNucleotides.at(pos)) value = true; // if it belongs to the valid ones,
    we're done.

        else pos ++; // if it doesn't, we move to the next valid nucleotide.
    }

    return (value);
}
```

4.1.2.2 ToLower()

```
void ToLower (
            Kmer & kmer )
```

Converts to lowercase the characters (nucleotides) of the given Kmer.

Parameters

kmer	A Kmer object. Output parameter

Definition at line 205 of file Kmer.cpp.

```
{
    int size = kmer.size(); // To prevent problems with signed/unsigned ints

    for (int i = 0; i < size; i++) { // We run through our Kmer, converting to
                                     // lowercase any letter which isn't already

        kmer.at(i) = std::tolower(kmer.at(i));
    }

}
```

4.1.2.3 ToUpper()

```
void ToUpper (
             Kmer & kmer )
```

Converts to uppercase the characters (nucleotides) of the given Kmer.

Parameters

kmer	A Kmer object. Output parameter

Definition at line 217 of file Kmer.cpp.

```
{
    int size = kmer.size(); // To prevent problems with signed/unsigned ints

    for (int i = 0; i < size; i++) { // We run through our Kmer, converting
                                     // to uppercase any letter which isn't
                                     // already

        kmer.at(i) = std::toupper(kmer.at(i));
    }

}
```

4.2 src/Kmer.cpp File Reference

```
#include <iostream>
#include "Kmer.h"
```

Functions

- bool IsValidNucleotide (char nucleotide, const std::string &validNucleotides)

  Checks if the given nucleotide is contained in `validNucleotides`. That is, if the given character can be considered as part of a genetic sequence.
- void ToLower (Kmer &kmer)

  Converts to lowercase the characters (nucleotides) of the given Kmer.
- void ToUpper (Kmer &kmer)

  Converts to uppercase the characters (nucleotides) of the given Kmer.

4.2.1 Detailed Description

Author

Adolfo Martínez Olmedo adolfomarol@correo.ugr.es

Last modified on 7 October 2024, 13:42

4.2.2 Function Documentation

4.2.2.1 IsValidNucleotide()

```
bool IsValidNucleotide (
            char nucleotide,
            const std::string & validNucleotides )
```

Checks if the given nucleotide is contained in `validNucleotides`. That is, if the given character can be considered as part of a genetic sequence.

Parameters

nucleotide	The nucleotide (a character) to check. Input parameter
validNucleotides	The set of characters that we consider as possible characters in a genetic sequence. Input parameter

Returns

true if the given character is contained in `validNucleotides`; false otherwise

Definition at line 184 of file Kmer.cpp.

```
{
    bool value = false; // First initialize the value as false to go in the loop

    int size = validNucleotides.size(); // In order to avoid signed/unsigned integers

    // We will use a while loop in order to minimize the amount of comparisons
    // so we can exit it if we find the nucleotide to be valid.

    int pos = 0;

    while (pos < size && !value) {

        if (nucleotide == validNucleotides.at(pos)) value = true; // if it belongs to the valid ones,
    we're done.

        else pos ++; // if it doesn't, we move to the next valid nucleotide.
    }

    return (value);
}
```

4.2.2.2 ToLower()

```
void ToLower (
            Kmer & kmer )
```

Converts to lowercase the characters (nucleotides) of the given Kmer.

Parameters

kmer	A Kmer object. Output parameter

Definition at line 205 of file Kmer.cpp.

```
{
    int size = kmer.size(); // To prevent problems with signed/unsigned ints

    for (int i = 0; i < size; i++) { // We run through our Kmer, converting to
                                     // lowercase any letter which isn't already

        kmer.at(i) = std::tolower(kmer.at(i));
    }

}
```

4.2.2.3 ToUpper()

```
void ToUpper (
            Kmer & kmer )
```

Converts to uppercase the characters (nucleotides) of the given Kmer.

Parameters

kmer	A Kmer object. Output parameter

Definition at line 217 of file Kmer.cpp.

```
{
    int size = kmer.size(); // To prevent problems with signed/unsigned ints

    for (int i = 0; i < size; i++) { // We run through our Kmer, converting
                                     // to uppercase any letter which isn't
                                     // already

        kmer.at(i) = std::toupper(kmer.at(i));
    }

}
```

4.3 src/main.cpp File Reference

```
#include <iostream>
#include <string>
#include "Kmer.h"
```

Functions

- int main (int argc, char ∗argv[ ])

  Main function.

4.3.1 Detailed Description

Author

Adolfo Martínez Olmedo Created on 05 March 2024, 15:00

4.3.2 Function Documentation

4.3.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main function.

This program first reads from the standard input an integer k (length of Kmer) and a string with a genetic sequence. Then, it obtains from the genetic sequence, the list of kmers (of length k) and saves them in the array kmers. Then, the kmers are normalized. After that, the complementary kmers, converted to lowercase, are saved in the array complementaryKmers. Finally the kmers in the arrays kmers and complementaryKmers are shown in the standard output. See the next example:

Running example:

kmer0 < data/easyDNA5_missing.k0in

6 GCGCC<-->cgcgg CGCCC<-->gcggg GCCC_<-->cggg_ CCC_G<-->ggg_c CC_G_<-->gg_c_ C_G_G<-->g_c_c

Returns

Always 0

Definition at line 43 of file main.cpp.

```
                              {

    int k;
    std::string genetic_sequence;

    // This string contains the list of nucleotides that are considered as
    // valid within a genetic sequence. The rest of characters are considered as
    // unknown nucleotides
    const std::string VALID_NUCLEOTIDES = "ACGT";

    // This string contains the list of complementary nucleotides for each
    // nucleotide in validNucleotides
    const std::string COMPLEMENTARY_NUCLEOTIDES = "TGCA";

    // This is a constant with the dimension of the array kmers
    const int DIM_ARRAY_KMERS = 100;

    // This is the array where the kmers of the input genetic sequence will be
    // saved
    Kmer kmers[DIM_ARRAY_KMERS];

    // This is the array where the complementary kmers will be
    // saved
    Kmer complementaryKmers[DIM_ARRAY_KMERS];

    // Read K (integer) and a string with the input nucleotides list

    std::cin » k » genetic_sequence;

    int sequence_size = genetic_sequence.size(); // To avoid comparing signed a
                                                 // and unsigned ints.
    int kmers_used = 0;

    // We make sure we won’t accept a sequence longer than the dimension of our
    // array.

    if (sequence_size > DIM_ARRAY_KMERS && k < sequence_size)
        sequence_size = DIM_ARRAY_KMERS + k - 1;

    // Obtain the kmers: find the kmers in
    // the input string and put them in an array of Kmers

    // First we will run through the whole string.


    for (int i = 0; i < (sequence_size - k + 1); i++) {

        // We create a k sized implicit string which will be used to contain the
        // nucleotides to initialize the Kmer.

        std::string k_nucleotides;

        // Move from the position we’re at in the string until we cover "k"
        // places, that will be our kmer.

        k_nucleotides = genetic_sequence.substr(i,k);

         // Now that we have all the nucleotides we need in the string we create
         // the k sized Kmer.

         Kmer new_Kmer (k_nucleotides);

         // After having created the kmer, we add it to the array.

         kmers[kmers_used] = new_Kmer;

         // We normalize it.

         kmers[kmers_used].normalize(VALID_NUCLEOTIDES);

         kmers_used ++;

         // Obtain the complementary kmers and turn them into lowercase

         // First let’s create the implicit kmer which will eventually become
         // our complementary.

         Kmer complementary_kmer(k);

         complementary_kmer = kmers[i].complementary(VALID_NUCLEOTIDES,
                                                      COMPLEMENTARY_NUCLEOTIDES);

         ToLower(complementary_kmer);

         complementaryKmers[i] = complementary_kmer;

     }

     // Show the list of kmers and complementary kmers as in the example

     std::cout « kmers_used « std::endl;

     for(int i = 0; i < kmers_used; i++) {
         std::string output;
         output = kmers[i].toString() + "<-->" +
                                 complementaryKmers[i].toString();
         std::cout « output « std::endl;
     }

     return (0);
 }
```