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.

```
19 {
20
21     if (k <= 0){    // Precondition violation.
22         throw std::invalid_argument(
23             std::string("Kmer(k): Number of nucleotides in a given") +
24                         " Kmer cannot be less than or equal to zero");
25     } // Throws error message when precondition is violated
26
27     else{ // k > 0
28         for(int i = 0; i < k; i++) {
29             _text += MISSING_NUCLEOTIDE;
30         } // Fills a "k" sized Kmer with the default character '_'
31     }
32
33 }
```

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.

```
41 {
42
43     if (text.size()==0){    // text is empty
44         throw std::invalid_argument( // composed string
45             std::string("Kmer(const std::string&text): ") +
46                         "text is an empty string");
47     }
48     _text = text; // if the text is not empty, create a Kmer with it.
49 }
```

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.

```
88 {
89
90     if (index < 0 || index >= getK()){ // Precondition violation
91         throw std::out_of_range(       // composed string
92             std::string("char& Kmer::at(int index) const: ") +
93                         "invalid position " + std::to_string(index));
94     }
95     else{
96         return (_text.at(index)); // Returns a reference to the character
```

```
97                                  //  in a given position.
98     }
99 }
```

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.

```
74 {
75
76     if (index < 0 || index >= getK()){ // Precondition violation
77         throw std::out_of_range(       // composed string
78             std::string("const char& Kmer::at(int index) const: ") +
79                         "invalid position " + std::to_string(index));
80     }
81     else{
82         return (_text.at(index)); // Returns a constant reference to
83                                   // the character in a given position.
84     }
85 }
```

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 139 of file Kmer.cpp.

```
141 {
142     if (nucleotides.size() != complementaryNucleotides.size()){ // Different sizes
143         throw std::invalid_argument( // composed string
144             std::string("Kmer Kmer::complementary(const std::string& nucleotides,") +
145                         "const std::string& complementaryNucleotides) const:" +
146                         " nucleotides and complementary nucleotides cannot be" +
147                         " differently sized");
148     }
149     else {
150         int text_size = _text.size();                    // In order to prevent signed problems
151         int nucleotide_size = nucleotides.size();        // In order to prevent signed problems
152         Kmer complementary_kmer(text_size);              // We create the object we're gonna return
153         complementary_kmer._text = _text;
154 
155         // We will use two for loops, one to run through our Kmer and another one
156         // to, for each individual character, convert it into its complementary.
157 
158         for (int i = 0; i < text_size; i++) {
159             // We move through our Kmer
160 
161             bool complemented = false;
162             int pos = 0;
163 
164             while (pos < nucleotide_size && !complemented) {
165 
166                 // We check which Nucleotide in specific we're studying in our Kmer
167                 // depending on which one it is we match it with the corresponding
168                 // complementary nucleotide. If it doesn't match any of the valid
169                 // ones, then we will do nothing and it will stay the same.
170                 // As soon as we have interchanged them, the process is done
171                 // and we exit the while loop
172 
173                 if (complementary_kmer.at(i) == nucleotides.at(pos)){
174                     complementary_kmer._text.at(i) = complementaryNucleotides.at(pos);
175                     complemented = true;
176                 }
177                 else pos ++;
178             }
179 
180         }
181 
182         return (complementary_kmer);
183     }
184 
185 }
```

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.

```
52 {
53     return(_text.size()); // Returns the size of the string that is
54                           // representing the Kmer.
55 }
```

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.

```
102 {
103     // This method performs its function in two steps, first it converts
104     // the characters to uppercase then it replaces any invalid character.
105
106     // 1. Converting:
107     int size = _text.size(); // In order to prevent signed problems
108
109     for (int i = 0; i < size; i ++) {
110         _text.at(i) = std::toupper(_text.at(i));
111     }
112
113     // As a reference is used in the function's parameters, it will modify
114     // the string converting it to uppercase.
115
116     // 2. Formatting:
117
118
119     for (int i = 0; i < size; i++){
120         // First we run through the string
121
122
123         // Then we check if the character we're looking at matches any of the
124         // valid ones, if it doesn't we switch it with our constant
125         // MISSING_NUCLEOTIDE character
126         // We can efficiently make use of our function.
127
128         if (!IsValidNucleotide(_text.at(i),validNucleotides)){
129             _text.at(i) = MISSING_NUCLEOTIDE;
130         }
131
132         // If it matches one of the valid ones we simply move on to study
133         // the next character.
134
135     }
136
137 }
```

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.

```
60 {
61     return(_text.size()); // Returns the size of the string which is
62                           // representing the Kmer.
63 }
```

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.

```
66 {
67     return(_text); // Simply returns the string as it's
68                    // already representing a Kmer.
69 }
```

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:

- /home/adolfo/Desktop/ugr_mp/NetBeansProjects/Kmer0/include/Kmer.h
- /home/adolfo/Desktop/ugr_mp/NetBeansProjects/Kmer0/src/Kmer.cpp

Chapter 4

File Documentation

4.1 /home/adolfo/Desktop/ugr_mp/NetBeansProjects/Kmer0/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 187 of file Kmer.cpp.

```
188 {
189     bool value = false; // First initialize the value as false to go in the loop
190
191     int size = validNucleotides.size(); // In order to avoid signed/unsigned integers
192
193     // We will use a while loop in order to minimize the amount of comparisons
194     // so we can exit it if we find the nucleotide to be valid.
195
196     int pos = 0;
197
198     while (pos < size && !value) {
199
200         if (nucleotide == validNucleotides.at(pos)) value = true; // if it belongs to the valid ones,
       we're done.
201
202         else pos ++; // if it doesn't, we move to the next valid nucleotide.
203     }
204
205     return (value);
206 }
```

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 208 of file Kmer.cpp.

```
209 {
210     int size = kmer.size(); // To prevent problems with signed/unsigned ints
211
212     for (int i = 0; i < size; i++) { // We run through our Kmer, converting to lowercase any letter
        which
213                                      // isn't already
214
215         kmer.at(i) = std::tolower(kmer.at(i));
216     }
217
218 }
```

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 220 of file Kmer.cpp.

```
221 {
222     int size = kmer.size(); // To prevent problems with signed/unsigned ints
223
224     for (int i = 0; i < size; i++) { // We run through our Kmer, converting to uppercase any letter
        which
225
226         kmer.at(i) = std::toupper(kmer.at(i));
227     }
228
229 }
```

4.2 /home/adolfo/Desktop/ugr_mp/NetBeansProjects/Kmer0/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 187 of file Kmer.cpp.

```
188 {
189     bool value = false; // First initialize the value as false to go in the loop
190
191     int size = validNucleotides.size(); // In order to avoid signed/unsigned integers
192
193     // We will use a while loop in order to minimize the amount of comparisons
194     // so we can exit it if we find the nucleotide to be valid.
195
196     int pos = 0;
197
198     while (pos < size && !value) {
199
200         if (nucleotide == validNucleotides.at(pos)) value = true; // if it belongs to the valid ones,
        we're done.
201
202         else pos ++; // if it doesn't, we move to the next valid nucleotide.
203     }
204
205     return (value);
206 }
```

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 208 of file Kmer.cpp.

```
209 {
210     int size = kmer.size(); // To prevent problems with signed/unsigned ints
211
212     for (int i = 0; i < size; i++) { // We run through our Kmer, converting to lowercase any letter
    which
213                                      // isn't already
214
215         kmer.at(i) = std::tolower(kmer.at(i));
216     }
217
218 }
```

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 220 of file Kmer.cpp.

```
221 {
222     int size = kmer.size(); // To prevent problems with signed/unsigned ints
223
224     for (int i = 0; i < size; i++) { // We run through our Kmer, converting to uppercase any letter
    which
225
226         kmer.at(i) = std::toupper(kmer.at(i));
227     }
228
229 }
```