cawlign

0.0.1

Generated by Doxygen 1.9.8

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 argparse::args_t Class Reference

Public Member Functions

- args_t (int, const char ∗∗)
- ∼args_t ()

Public Attributes

- FILE ∗ **output**
- FILE ∗ **reference**
- FILE ∗ **input**
- ConfigParser ∗ **scores**
- data_t **data_type**
- local_t **local_option**
- space_t **space_type**
- out_format_t **out_format**
- rc_t **reverse_complement**
- bool **quiet**
- bool **affine**
- bool **include_reference**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 args_t()

```
argparse::args_t::args_t (
             int ,
             const char **  )
```

Constructor for args_t, which parses command-line arguments and sets up configuration options.

This constructor processes the command-line arguments, setting up the program's input, output, reference files, and various other configuration options such as data type, space type, and output format. If necessary, default values are assigned to some options.

Parameters

argc	The number of command-line arguments.
argv	The array of command-line arguments.

4.1.1.2 ∼args_t()

```
argparse::args_t::∼args_t ( )
```

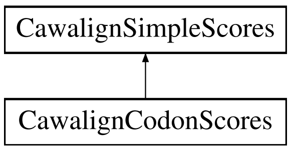
Destructor for args_t, responsible for cleaning up any resources used (files or memory). It closes the input/output/reference files and deletes the scores object, if applicable.

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

- src/argparse.hpp
- src/argparse.cpp

4.2 CawalignCodonScores Class Reference

Inheritance diagram for CawalignCodonScores:



Public Member Functions

- CawalignCodonScores (ConfigParser ∗)

Public Member Functions inherited from CawalignSimpleScores

- CawalignSimpleScores (const char ∗_alphabet, const cawlign_fp ∗_scoring_matrix, const cawlign_fp _↩ open_gap_reference, const cawlign_fp _open_gap_query, const cawlign_fp _extend_gap_reference, const cawlign_fp _extend_gap_query)
- CawalignSimpleScores (ConfigParser ∗)
- void _init_alphabet (long not_found=-1)

Static Public Member Functions

- static int nucleotide_diff (long, long)

Public Attributes

- Vector **translation_table**
- VectorFP **s3x1**
- VectorFP **s3x2**
- VectorFP **s3x4**
- VectorFP **s3x5**
- cawlign_fp **frameshift_cost**
- cawlign_fp **synonymous_penalty**
- StringBuffer **amino_acids**
- int **stop_codon_index**
- int **mismatch_index**

Public Attributes inherited from CawalignSimpleScores

- StringBuffer **alphabet**
- unsigned int **D**
- long **char_map** [255]
- VectorFP **scoring_matrix**
- cawlign_fp **open_gap_reference**
- cawlign_fp **open_gap_query**
- cawlign_fp **extend_gap_query**
- cawlign_fp **extend_gap_reference**
- char **gap_char**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 CawalignCodonScores()

```
CawalignCodonScores::CawalignCodonScores (
             ConfigParser * settings )
```

Constructs a `CawalignCodonScores` object using configuration settings.

This constructor initializes the codon scoring system using values from a `ConfigParser`. It sets up the codon translation table, stop codon index, mismatch index, and scoring matrices for codon alignments. Throws errors if the amino acid alphabet is incomplete or the translation table is invalid.

Parameters

settings	A pointer to a `ConfigParser` object containing configuration settings.

4.2.2 Member Function Documentation

4.2.2.1 nucleotide_diff()

```
int CawalignCodonScores::nucleotide_diff (
             long c1,
             long c2 )  [static]
```

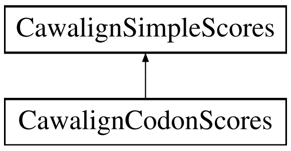
Calculates the number of nucleotide differences between two codons.

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

- src/scoring.hpp
- src/scoring.cpp

4.3 CawalignSimpleScores Class Reference

Inheritance diagram for CawalignSimpleScores:



Public Member Functions

- CawalignSimpleScores (const char ∗_alphabet, const cawlign_fp ∗_scoring_matrix, const cawlign_fp _↩ open_gap_reference, const cawlign_fp _open_gap_query, const cawlign_fp _extend_gap_reference, const cawlign_fp _extend_gap_query)
- CawalignSimpleScores (ConfigParser ∗)
- void _init_alphabet (long not_found=-1)

Public Attributes

- StringBuffer **alphabet**
- unsigned int **D**
- long **char_map** [255]
- VectorFP **scoring_matrix**
- cawlign_fp **open_gap_reference**
- cawlign_fp **open_gap_query**
- cawlign_fp **extend_gap_query**
- cawlign_fp **extend_gap_reference**
- char **gap_char**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 CawalignSimpleScores() [1/2]

```
CawalignSimpleScores::CawalignSimpleScores (
             const char * _alphabet,
             const cawlign_fp * _scoring_matrix,
             const cawlign_fp _open_gap_reference,
             const cawlign_fp _open_gap_query,
             const cawlign_fp _extend_gap_reference,
             const cawlign_fp _extend_gap_query )
```

Constructs a `CawalignSimpleScores` object with a custom alphabet and scoring matrix.

This constructor initializes the scoring system with a user-specified alphabet and scoring matrix, as well as gap penalties for both reference and query sequences. Throws an error if the alphabet is empty.

Parameters

_alphabet	The alphabet for scoring (e.g., nucleotides or amino acids).
_scoring_matrix	A pointer to the scoring matrix values.
_open_gap_reference	Gap opening penalty for reference sequences.
_open_gap_query	Gap opening penalty for query sequences.
_extend_gap_reference	Gap extension penalty for reference sequences.
_extend_gap_query	Gap extension penalty for query sequences.

4.3.1.2 CawalignSimpleScores() [2/2]

```
CawalignSimpleScores::CawalignSimpleScores (
             ConfigParser * settings )
```

Constructs a `CawalignSimpleScores` object using configuration settings.

This constructor reads configuration values from a `ConfigParser` to initialize the alphabet, scoring matrix, and gap penalties. Throws errors if the alphabet is missing or the scoring matrix dimensions are incorrect.

Parameters

settings	A pointer to a `ConfigParser` object containing configuration settings.

4.3.2 Member Function Documentation

4.3.2.1 _init_alphabet()

```
void CawalignSimpleScores::_init_alphabet (
             long not_found = -1 )
```

Initializes the character map for the scoring matrix.

This function populates the `char_map` array, mapping each character in the alphabet to its index in the scoring matrix. Characters not in the alphabet are assigned a value of `not_found`.

Parameters

not_found	The value to assign for characters not found in the alphabet.

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

- src/scoring.hpp
- src/scoring.cpp

4.4 ConfigParser Class Reference

Public Member Functions

- ConfigParser (std::ifstream &configFile)
- template<typename T >
  T **aConfig** (std::string section, std::string name, size_t pos=0)
- template<typename T >
  std::vector< T > **aConfigVec** (std::string section, std::string name)
- template<> bool aConfig (std::string section, std::string configName, size_t pos)
- template<> std::vector< bool > aConfigVec (std::string section, std::string configName)
- template<> bool **aConfig** (std::string section, std::string name, size_t pos)
- template<> std::vector< bool > **aConfigVec** (std::string section, std::string name)

4.4.1 Constructor & Destructor Documentation

4.4.1.1 ConfigParser()

```
ConfigParser::ConfigParser (
            std::ifstream & configFile )
```

Constructor for `ConfigParser`.

This function reads and parses a configuration file. It processes lines to strip whitespace, ignores comments, and splits key-value pairs using the = symbol. If the key is within a section (indicated by `[...]`), the key is prefixed with the section name. Multiple values for a key are stored in a vector. Parsed configurations are stored in a map with section and key names combined as the key.

Parameters

configFile	A reference to an `ifstream` representing the open configuration file.

Exceptions

std::runtime_error	if a parsing error occurs in the configuration file.

4.4.2 Member Function Documentation

4.4.2.1 aConfig()

```
template<>
bool ConfigParser::aConfig (
            std::string section,
            std::string configName,
            size_t pos )
```

Template specialization for retrieving boolean configuration values.

This function retrieves a specific boolean configuration value from a section and configuration name. The value is parsed as either `true` (for "true", "TRUE", or "1") or `false` (for "false", "FALSE", or "0"). If the value does not match any of these, the function defaults to `false`.

Parameters

section	The section name in the configuration file.
configName	The key within the section.
pos	The position of the value in the vector (if there are multiple values).

Returns

`true` if the configuration value is recognized as true, otherwise `false`.

4.4.2.2 aConfigVec()

```
template<>
std::vector< bool > ConfigParser::aConfigVec (
            std::string section,
            std::string configName )
```

Template specialization for retrieving a vector of boolean configuration values.

This function retrieves a vector of boolean values associated with a configuration key in a section. Each value is parsed as either `true` (for "true", "TRUE", or "1") or `false` (for "false", "FALSE", or "0"). If a value does not match any of these, it defaults to `false`.

Parameters

section	The section name in the configuration file.
configName	The key within the section.

Returns

A vector of boolean values parsed from the configuration.

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

- src/configparser.hpp
- src/configparser.cpp

4.5 sequence_gap_structure Struct Reference

Public Attributes

- long **first_nongap**
- long **last_nongap**
- long **resolved_start**
- long **resolved_end**

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

- src/tn93_shared.h

4.6 StringBuffer Class Reference

Public Member Functions

- StringBuffer (void)
- ∼StringBuffer (void)
- char ∗ **getString** (void) const
- void appendChar (const char)
- void appendBuffer (const char ∗, const long=-1)
- void resetString (void)
- void swap (StringBuffer &)
- unsigned long **length** (void) const
- void **reset_length** (unsigned long newL)
- char **setChar** (const long i, const char c)
- char **getChar** (const long i) const
- void flip (void)
- void **detach** (void)

Static Public Attributes

- static long **sbDefaultLength** = 16
- static long **sbDefaultBoost** = 16

4.6.1 Constructor & Destructor Documentation

4.6.1.1 StringBuffer()

```
StringBuffer::StringBuffer (
            void )
```

Initializes an empty `StringBuffer` with a default initial capacity. This buffer dynamically grows as new characters are appended.

4.6.1.2 ∼StringBuffer()

```
StringBuffer::∼StringBuffer (
            void )
```

Frees the memory allocated for the string buffer.

4.6.2 Member Function Documentation

4.6.2.1 appendBuffer()

```
void StringBuffer::appendBuffer (
            const char * buffer,
            const long length = -1 )
```

Appends a string or a buffer of specified length to the `StringBuffer`.

Parameters

buffer	The string or character buffer to append.
length	The length of the buffer, if known. If not, the length is inferred using `strlen`.

4.6.2.2 appendChar()

```
void StringBuffer::appendChar (
             const char c )
```

Appends a single character to the end of the buffer, growing the buffer if needed.

Parameters

c	The character to append.

4.6.2.3 flip()

```
void StringBuffer::flip (
             void  )
```

Reverses the content of the `StringBuffer`.

4.6.2.4 resetString()

```
void StringBuffer::resetString (
             void  )
```

Resets the `StringBuffer` to an empty state.

Clears the buffer content by resetting its length, but keeps the allocated memory.

4.6.2.5 swap()

```
void StringBuffer::swap (
             StringBuffer & src )
```

Swaps the contents of this `StringBuffer` with another `StringBuffer`.

This function exchanges the data, length, and capacity of two `StringBuffer` objects.

Parameters

src	The `StringBuffer` object to swap with.

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

- src/stringBuffer.h
- src/stringBuffer.cc

4.7 Vector Class Reference

Public Member Functions

- Vector (void)
- ∼Vector (void)
- void appendValue (const long)
- void appendVector (const Vector &)
- long extractMin (VectorFP &)
- void resetVector (void)
- void remove (const unsigned long)
- void storeValue (const long, const unsigned long)
- void storeVector (const Vector &, const unsigned long)
- void sort (void)
- void swap (Vector &)
- long value (const long idx) const
- unsigned long **length** (void) const

Static Public Attributes

- static long **vDefaultLength** = 16
- static long **vDefaultBoost** = 16

4.7.1 Constructor & Destructor Documentation

4.7.1.1 Vector()

```
Vector::Vector (
            void  )
```

Initializes an empty vector of `long` values with a default initial capacity.

4.7.1.2 ∼Vector()

```
Vector::∼Vector (
            void  )
```

Frees the memory allocated for the vector data.

4.7.2 Member Function Documentation

4.7.2.1 appendValue()

```
void Vector::appendValue (
            const long l )
```

Appends a value to the `Vector`.

Adds a `long` value to the end of the vector, growing the vector if needed.

Parameters

l	The `long` value to append.

4.7.2.2 appendVector()

```
void Vector::appendVector (
             const Vector & v )
```

Appends the contents of another `Vector` to this `Vector`.

Parameters

v	The source `Vector` whose contents are to be appended.

4.7.2.3 extractMin()

```
long Vector::extractMin (
             VectorFP & values )
```

Extracts the minimum value from the vector based on a `VectorFP` of floating-point values.

Parameters

values	The `VectorFP` of floating-point values to compare.

Returns

The index of the minimum value, or `-1` if the vector is empty.

4.7.2.4 remove()

```
void Vector::remove (
             const unsigned long l )
```

Removes the element at the given index and shifts the remaining elements to fill the gap.

Parameters

l	The index of the element to remove.

4.7.2.5 resetVector()

```
void Vector::resetVector (
             void )
```

Resets the `Vector` to an empty state.

Clears the vector by resetting its length, but keeps the allocated memory.

4.7.2.6 sort()

```
void Vector::sort (
             void  )
```

Sorts the `Vector` in ascending order.

4.7.2.7 storeValue()

```
void Vector::storeValue (
             const long v,
             const unsigned long l )
```

Stores a value at a specific index in the `Vector`.

If the index is beyond the current capacity, the vector grows to accommodate the value.

Parameters

v	The `long` value to store.
l	The index at which to store the value.

4.7.2.8 storeVector()

```
void Vector::storeVector (
             const Vector & v,
             const unsigned long l )
```

Stores a pointer to a `Vector` at a specified index, growing the vector if necessary.

Parameters

v	The `Vector` object to store.
l	The index at which to store the vector.

4.7.2.9 swap()

```
void Vector::swap (
             Vector & src )
```

Swaps the contents of this `Vector` with another `Vector`.

Exchanges the data, length, and capacity of two `Vector` objects.

Parameters

src	The `Vector` object to swap with.

4.7.2.10 value()

```
long Vector::value (
             const long idx ) const
```

Retrieves a value from the `Vector` at the specified index.

Parameters

idx	The index from which to retrieve the value.

Returns

The value stored at the specified index.

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

- src/stringBuffer.h
- src/stringBuffer.cc

4.8 VectorFP Class Reference

Public Member Functions

- VectorFP (void)
- ∼VectorFP (void)
- void appendValue (const cawlign_fp)
- void appendValues (const cawlign_fp ∗, long)
- void storeValue (const cawlign_fp, const unsigned long)
- cawlign_fp **value** (const long idx)
- unsigned long **length** (void) const
- const cawlign_fp ∗ **values** (void)
- cawlign_fp ∗ **rvalues** (void)

Static Public Attributes

- static long **vDefaultLength** = 16
- static long **vDefaultBoost** = 16

4.8.1 Constructor & Destructor Documentation

4.8.1.1 VectorFP()

```
VectorFP::VectorFP (
             void   )
```

Initializes an empty vector of floating-point values with a default initial capacity.

4.8.1.2 ∼VectorFP()

```
VectorFP::∼VectorFP (
             void   )
```

Frees the memory allocated for the floating-point vector data.

4.8.2 Member Function Documentation

4.8.2.1 appendValue()

```
void VectorFP::appendValue (
             const cawlign_fp l )
```

Appends a floating-point value to the `VectorFP`.

Adds a floating-point value to the end of the vector, growing the vector if needed.

Parameters

l	The floating-point value to append.

4.8.2.2 appendValues()

```
void VectorFP::appendValues (
             const cawlign_fp * l,
             long N )
```

Appends multiple floating-point values to the `VectorFP`.

Parameters

l	The array of floating-point values to append.
N	The number of values to append.

4.8.2.3 storeValue()

```
void VectorFP::storeValue (
             const cawlign_fp v,
             const unsigned long l )
```

Stores a floating-point value at a specific index in the `VectorFP`.

If the index is beyond the current capacity, the vector grows to accommodate the value.

Parameters

v	The floating-point value to store.
l	The index at which to store the value.

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

- src/stringBuffer.h
- src/stringBuffer.cc

Chapter 5

File Documentation

5.1 alignment.h

```
/*

 HyPhy - Hypothesis Testing Using Phylogenies.

 Copyright (C) 1997-now
 Core Developers:
 Sergei L Kosakovsky Pond (spond@ucsd.edu)
 Art FY Poon    (apoon42@uwo.ca)
 Steven Weaver (sweaver@ucsd.edu)

 Module Developers:
 Lance Hepler (nlhepler@gmail.com)
 Martin Smith (martin.audacis@gmail.com)

 Significant contributions from:
 Spencer V Muse (muse@stat.ncsu.edu)
 Simon DW Frost (sdf22@cam.ac.uk)

 Permission is hereby granted, free of charge, to any person obtaining a
 copy of this software and associated documentation files (the
 "Software"), to deal in the Software without restriction, including
 without limitation the rights to use, copy, modify, merge, publish,
 distribute, sublicense, and/or sell copies of the Software, and to
 permit persons to whom the Software is furnished to do so, subject to
 the following conditions:

 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.

 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

 */

#ifndef __ALIGNMENT_HEADER_FILE__

#define __ALIGNMENT_HEADER_FILE__

typedef   float     cawlign_fp;

cawlign_fp AlignStrings( char const * r_str
                   , char const * q_str
                   , const long _r_len
                   , const long _q_len
                   , char * & r_res
                   , char * & q_res
                   , long * char_map
                   , const cawlign_fp * cost_matrix
                   , const long cost_stride
                   , const char gap
                   , cawlign_fp open_insertion
                   , cawlign_fp extend_insertion
                   , cawlign_fp open_deletion
                   , cawlign_fp extend_deletion
                   , cawlign_fp miscall_cost
                   , const bool do_local
                   , const bool do_affine
                   , const bool do_codon
                   , const long char_count
                   , const cawlign_fp * codon3x5
                   , const cawlign_fp * codon3x4
                   , const cawlign_fp * codon3x2
                   , const cawlign_fp * codon3x1
                   , const bool do_true_local = false
                   , const bool report_ref_insertions = true
                   , cawlign_fp* score_matrix_cache = nullptr
                   , cawlign_fp* insertion_matrix_cache = nullptr
                   , cawlign_fp* deletion_matrix_cache = nullptr
                   );

cawlign_fp LinearSpaceAlign(    const char * s1           // first string
                           , const char * s2           // second string
                           , const long s1L
                           , const long s2L
                           , long*  cmap     // char -> position in scoring matrix mapper
                           , const cawlign_fp * ccost        // NxN matrix of edit distances on
characters
                           , const long costD
                           , cawlign_fp gopen       // the cost of opening a gap in sequence 1
                           , cawlign_fp gextend     // the cost of extending a gap in sequence 1
(ignored unless doAffine == true)
                           , cawlign_fp gopen2      // the cost of opening a gap in sequence 2
                           , cawlign_fp gextend2    // the cost of opening a gap in sequence 2
(ignored unless doAffine == true)
                           , bool doLocal           // ignore prefix and suffix gaps
                           , bool doAffine          // use affine gap penalties
                           , long * ops      // edit operations for the optimal alignment
                           , cawlign_fp scoreCheck  // check the score of the alignment
                           , long from1
                           , long to1
                           , long from2
                           , long to2
                           , cawlign_fp ** buffer      // matrix storage,
                           , char parentGapLink
                           , char * ha
                           );

#endif
```

5.2 argparse.hpp

```

#ifndef ARGPARSE_H
#define ARGPARSE_H

#include <stdio.h>
#include <configparser.hpp>
// argument defaults

#define PROGNAME                 "cawlign"
#define DEFAULT_DATA_TYPE        nucleotide
#define DEFAULT_REFERENCE        "HXB2_pol"
#define DEFAULT_SCORING          "Nucleotide-BLAST"
#define DEFAULT_SPACE            quadratic
#define DEFAULT_LOCAL_TYPE       trim
#define DEFAULT_OUTPUT_FORMAT    refmap
#define DEFAULT_RC_TYPE          none

#ifndef VERSION_NUMBER
    #define VERSION_NUMBER            "0.0.1"
#endif

#ifndef LIBRARY_PATH
    #define LIBRARY_PATH            "/usr/local/shares/cawlign/"
#endif

#define SCORES_SUBPATH "scoring"
#define REF_SUBPATH    "references"

namespace argparse
{

    enum data_t {
      nucleotide,
      codon,
      protein
    };

    enum local_t {
      trim,
      global,
      local
    };

    enum space_t {
        quadratic,
        linear
    };

    enum out_format_t {
        refmap,
        refalign,
        pairwise
    };

    enum rc_t {
        none,
        silent,
        annotated
    };

   class args_t {
    public:

        FILE            * output,
                        * reference,
                        * input;

        ConfigParser    * scores;

        data_t          data_type;
        local_t         local_option;
        space_t         space_type;
        out_format_t    out_format;
        rc_t            reverse_complement;

        bool            quiet;
        bool            affine;
        bool            include_reference;


        args_t( int, const char ** );
        ~args_t();

    private:
        void parse_input        ( const char * );
        void parse_reference    ( const char * );
        void parse_output       ( const char * );
        void parse_scores       ( const char * );
        void parse_quiet        ( void );
        void parse_affine       ( void );
        void parse_include_ref  ( void );
        void parse_rc           ( const char * );
        void parse_space_t      ( const char * );
        void parse_data_t       ( const char * );
        void parse_local_t      ( const char * );
        void parse_out_format_t ( const char * );

    };

    void ERROR_NO_USAGE ( const char * msg, ... );
}

#endif // ARGPARSE_H
```

5.3 configparser.hpp

```
// Copyright (c) 2018 Daniel Zilles
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
```

```
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef CONFIGPARSER_HPP
#define CONFIGPARSER_HPP

#include <string>
#include <vector>
#include <map>
#include <iostream>
#include <sstream>

typedef std::map<std::string, std::vector<std::string» configList;

class ConfigParser {

  public:
  ConfigParser(std::ifstream& configFile);

    template<typename T>
    T aConfig(std::string section, std::string name, size_t pos = 0);
    template<typename T>
    std::vector<T> aConfigVec(std::string section, std::string name);


  private:
    static void       handleMissingKey (std::string);
    configList mConfigurations;

};

template <>
bool ConfigParser::aConfig<bool>(std::string section, std::string name, size_t pos);

template <typename T>
T ConfigParser::aConfig(std::string section, std::string configName, size_t pos) {

  T tmp;

  const auto& mConfigRef = mConfigurations;
  auto search = mConfigRef.find(section + " - " + configName);

  if (search == mConfigRef.end()) {
      handleMissingKey (std::string("Could not find required configuration section ") + section +
std::string(" key ") + configName);
  }

  std::istringstream iss(search->second[0]);

  if (search->second[0].find( "0x" ) != std::string::npos)
    iss » std::hex » tmp;
  else
    iss » std::dec » tmp;

  return tmp;
}

template <>
std::vector<bool> ConfigParser::aConfigVec<bool>(std::string section, std::string name);

template <typename T>
std::vector<T> ConfigParser::aConfigVec(std::string section, std::string configName) {


  const auto& mConfigRef = mConfigurations;
  auto search = mConfigRef.find(section + " - " + configName);

  if (search == mConfigRef.end()) {
    handleMissingKey (std::string("Could not find required configuration section ") + section +
std::string(" key ") + configName);
  }


  std::vector<T>  tmp(search->second.size());
  for (unsigned i = 0; i < search->second.size(); i++) {

    std::istringstream iss(search->second[i]);

    if (search->second[i].find( "0x" ) != std::string::npos)
      iss » std::hex » tmp[i];
    else
      iss » std::dec » tmp[i];
  }
  return tmp;
}
#endif
```

5.4 scoring.hpp

```
#ifndef SCORING_H
#define SCORING_H

#include <iostream>
#include "alignment.h"
#include "argparse.hpp"
#include "tn93_shared.h"

using namespace std;
using namespace argparse;


class CawalignSimpleScores {
    public:
        CawalignSimpleScores  (
                                const char * _alphabet,
                                const cawlign_fp * _scoring_matrix,
                                const cawlign_fp _open_gap_reference,
                                const cawlign_fp _open_gap_query,
                                const cawlign_fp _extend_gap_reference,
                                const cawlign_fp _extend_gap_query
                              );

        CawalignSimpleScores  (ConfigParser*);
        CawalignSimpleScores  (void) {D=0;};
        virtual ~CawalignSimpleScores (void) {};

        StringBuffer           alphabet;
        /*
            ordered characters that are included in the scoring matrix
        */
        unsigned int          D;
        // the number of characters in the string

        long                  char_map [255];
        /*
            for each ASCII character, this will map the character to the corresping entry the scoring matrix
            all characters NOT in `alphabet` get mapped to index D (the 'not defined' character)
        */

        VectorFP         scoring_matrix;
        /*
                A (D+1)x(D+1) scoring matrix where element (i,j) gives the score of matching (or mis-matching)
                the D-th row/column is for matchign a character NOT in the alphabet
                While generally symmetric, an asymmetric matrix can also be meaningful if there is some reason to have
                substitutions in reference/query weigted differently
        */

        cawlign_fp              open_gap_reference,
                            open_gap_query,
                            extend_gap_query,
                            extend_gap_reference;

        char                gap_char;
        /* gap open and extend character*/

        void                _init_alphabet (long not_found = -1);

};

class CawalignCodonScores : public CawalignSimpleScores {
    public:

        CawalignCodonScores  (ConfigParser*);
        virtual ~CawalignCodonScores (void) {};

        // compute how many nucleotides are different between the two codons encoded as 0-63 integers
        static int nucleotide_diff (long, long);

        Vector                translation_table;
        // codon (0-63 index) to single letter amino-acid code translation table

        // partial score tables
        VectorFP          s3x1,
                                s3x2,
                                s3x4,
                                s3x5;

        // the cost of introducing frameshits
        cawlign_fp                frameshift_cost,
        // the penalty for synonymous substitutions, per nucleotide change
                                synonymous_penalty;

        // ordered amino-acid scoring tables
        StringBuffer          amino_acids;

        int                   stop_codon_index;
        int                   mismatch_index;



};


extern const char   kNucleotideAlphabet[];
extern const cawlign_fp kNucScoring[];

#endif
```

5.5 stringBuffer.h

```
#ifndef __STRINGBUFFER__
#define __STRINGBUFFER__

#include "alignment.h"

//_________________________________________________________________________________________

class StringBuffer {

  char *sData;
  unsigned long sLength, saLength;

public:
  StringBuffer(void);
  ~StringBuffer(void);

  char *getString(void) const { return sData; }
  void appendChar(const char);
  void appendBuffer(const char *, const long = -1);
  void resetString(void);
  void swap(StringBuffer &);
  unsigned long length(void) const { return sLength; }
  void reset_length(unsigned long newL) {
    if (newL < sLength) {
      sLength = newL;
    }
  }

  char setChar(const long i, const char c) {
    char oc = sData[i];
    sData[i] = c;
    return oc;
  }

  char getChar(const long i) const { return sData[i]; }
  void flip (void);
  void detach (void) { sData = nullptr;}

  static long sbDefaultLength, sbDefaultBoost;
};

//_________________________________________________________________________________________

class VectorFP {

  cawlign_fp *vData;

  unsigned long vLength, vaLength;

public:
  VectorFP(void);
  ~VectorFP(void);

  void appendValue(const cawlign_fp);
  void appendValues(const cawlign_fp*, long);
  void storeValue(const cawlign_fp, const unsigned long);
  cawlign_fp value(const long idx) { return vData[idx]; }
  unsigned long length(void) const { return vLength; }
  const cawlign_fp * values (void) {return vData;}
  cawlign_fp * rvalues (void) {return vData;}

  static long vDefaultLength, vDefaultBoost;
};

//_____________________________________________________________________________

class Vector {

  long *vData;

  unsigned long vLength, vaLength;

public:
  Vector(void);
  ~Vector(void);

  void appendValue(const long);
  void appendVector(const Vector &);
  long extractMin(VectorFP &);
  void resetVector(void);
  void remove(const unsigned long);
  void storeValue(const long, const unsigned long);
  void storeVector(const Vector &, const unsigned long);
  void sort(void);
  void swap(Vector &);
  long value(const long idx) const;
  unsigned long length(void) const { return vLength; }

  static long vDefaultLength, vDefaultBoost;
};

#endif
```

5.6 tn93_shared.h

```
#ifndef     __TN93SHARED__
#define     __TN93SHARED__

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <iomanip>
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <climits>
#include "stringBuffer.h"

using namespace std;

#define  RESOLVE_A      0x01
#define  RESOLVE_C      0x02
#define  RESOLVE_G      0x04
#define  RESOLVE_T      0x08


#define  RESOLVE        0
#define  AVERAGE        1
#define  SKIP           2
#define  GAPMM          3
#define  SUBSET         4
#define  MISMATCH       5
#define  INFORMATIVE    6

#define RAND_RANGE 0xffffffffUL /* Maximum value returned by genrand_int32 */

#define MIN(a,b) (a) < (b) ? (a) : (b)
#define MAX(a,b) (a) > (b) ? (a) : (b)

struct sequence_gap_structure {

  long first_nongap,
       last_nongap,
       resolved_start,
       resolved_end;

  sequence_gap_structure (void) {
    first_nongap     = LONG_MAX;
    last_nongap     = 0L;
    resolved_start = 0L;
    resolved_end   = 0L;
  }

};

void init_genrand(unsigned long s);
unsigned long genrand_int32(void);
double      computeTN93 (const char * s1, const char *s2,  const unsigned long L, const char matchMode, const long * randomize, const long min_overlap, unsigned long* = NULL, const double = 0.0, const unsigned long cnt = 0, const long count1 = 1, const long count2 = 1, const sequence_gap_structure * = NULL, const sequence_gap_structure * = NULL);

long   computeDifferences (const char * s1,
                           const char *s2,
                           const unsigned long L,
                           const char matchMode,
                           Vector& storage,
                           const sequence_gap_structure * = NULL,
                           const sequence_gap_structure * = NULL);


long stringLength (Vector& lengths, unsigned long index);
char* stringText (const StringBuffer& strings, const Vector& lengths, unsigned long index);
void addASequenceToList (StringBuffer& sequences, Vector& seqLengths, long &firstSequenceLength, StringBuffer& names, Vector& nameLengths);
int readFASTA (FILE* F, char& automatonState,  StringBuffer &names, StringBuffer& sequences, Vector &nameLengths, Vector &seqLengths, long& firstSequenceLength, bool oneByOne = false,  Vector* sequenceInstances = NULL, char sep = ':', double include_prob = 1.0, bool show_progress = false);
void dump_sequence_fasta (unsigned long index, FILE* output, long firstSequenceLength, double * d = NULL, bool = false, unsigned long from = 0L, unsigned long to = 0L);
void initAlphabets(bool = false, char * = NULL, bool id_map = false);
void merge_two_sequences (const char* source, char* target, const long sequence_length);
long perfect_match (const char* source, char* target, const long sequence_length);
void dump_fasta (const char* source, const long sequence_length, FILE* output, bool newln = true, bool = false, unsigned long from = 0L, unsigned long to = 0L);

int    reverseComplement (StringBuffer& sequence, unsigned long from, unsigned long to);
struct sequence_gap_structure describe_sequence (const char* source, const unsigned long sequence_length, const unsigned long char_count = 4UL);

const long * resolve_char (unsigned char, bool = false, bool = true);
const double resolution_count (unsigned char, bool = false);
const char unmap_char (unsigned char, bool = false);
inline void unpack_difference (long diff, long& location, unsigned& alt) {
    location = diff » 8;
    alt = diff & 0xff;
}


extern StringBuffer names,
       sequences;

extern unsigned char * resolveTheseAmbigs;

extern double   resolve_fraction;

extern Vector       nameLengths,
       seqLengths,
       workingNodes,
       nodeParents;

extern VectorFP distanceEstimates;
extern const  double  resolutionsCount [];
extern char validFlags[];

#endif
```

Index