The CodingTool Library

0.9

by Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria

Contents

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 CodeMatrix Class Reference

This class represents a binary code matrix.

`#include <CodeMatrix.h>`

Public Member Functions

- CodeMatrix (void)
  Constructor.
- virtual ~CodeMatrix (void)
  Destructor.
- void Build (CodeWord(*pBuildFunction)(uint64_t &), uint64_t dDim)
  Creates a code matrix.
- void AddRow (CodeWord oRow)
  Adds a row to the matrix.
- CodeMatrix GetSubMatrix (std::vector< uint64_t > &vRows, std::vector< uint64_t > &vCols)
  Returns a submatrix of the current matrix.
- bool AtBool (uint64_t dRow, uint64_t dCol)
  Returns the bit at the given position.
- uint32_t At32 (uint64_t dRow, uint64_t dCol)
  Returns the 32-bit value at the given position.
- uint64_t At64 (uint64_t dRow, uint64_t dCol)
  Returns the 64-bit value at the given position.
- void SetBool (uint64_t dRow, uint64_t dCol, bool dData)
  Changes the bit at the given position.

- void Set32 (uint64_t dRow, uint64_t dCol, uint32_t dData)

  Changes the element at the given position.

- void Set64 (uint64_t dRow, uint64_t dCol, uint64_t dData)

  Changes the element at the given position.

- void DeleteRow (uint64_t dRow)

  Deletes a row of the matrix.

- void DeleteColumn (uint64_t dColumn)

  Deletes a column of the matrix.

- uint64_t GetRows () const

  Returns the number of rows.

- uint64_t GetColumns () const

  Returns the number of columns.

- uint64_t GetColumns64 () const

  Returns the number of columns.

- void PrintMatrix (const std::string &sFileName="")

  Outputs the matrix.

- void ReadFromFile (const std::string &sFileName)

  Creates matrix from the given data.

- bool IsSystematic ()

  Returns true if the matrix is a systematic generator matrix.

- CodeMatrix Transpose ()

  Returns the transposed matrix.

- CodeMatrix & operator= (const CodeMatrix &oM)

  Overloads the assignment operator.

- CodeWord & operator[ ] (uint64_t dIndex)

  Returns one row of the matrix.

3.1.1 Detailed Description

This class represents a binary code matrix. The data structure consists of a STL vector containing instances of the class CodeWord. Each row of the matrix is one code word. Therefore, the data is stored in 64-bit or 32-bit words, depending on the target architecture. Data can be added and accessed in different ways (see CodeWord). The matrix can be considered as a bit, 32-bit word or 64-bit word matrix (little endian). The number of rows or columns refers to the bit representation of the matrix, since it represents code dimension and length.

See also

CodeWord

Examples:

allinone.cpp, search.cpp, sha1me.cpp, and shortening.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CodeMatrix::CodeMatrix (void)

Constructor.

Does nothing special.

3.1.2.2 CodeMatrix::~CodeMatrix (void) `[virtual]`

Destructor.

Does nothing special.

3.1.3 Member Function Documentation

3.1.3.1 void CodeMatrix::AddRow (CodeWord *oRow*)

Adds a row to the matrix.

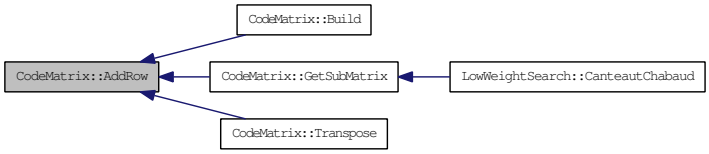
If the matrix is not empty the length of the added code word has to be the same as the current length of the matrix. If it is not an error message is printed to the console and the code word is not added.

Parameters

oRow The code word which should be added.

Here is the call graph for this function:

Here is the caller graph for this function:



3.1.3.2 uint32_t CodeMatrix::At32 (uint64_t *dRow*, uint64_t *dCol*)

Returns the 32-bit value at the given position.

This method considers the matrix consisting of 32-bit words. It returns the element at the given position. The indices should be given with respect to the 32-bit representation of the matrix.

Parameters

dRow The row index of the matrix.

dCol The column index of the matrix with respect to the 32-bit representation.

Returns

The value at (dRow,dCol).

3.1.3.3 uint64_t CodeMatrix::At64 (uint64_t *dRow*, uint64_t *dCol*)

Returns the 64-bit value at the given position.

This method considers the matrix consisting of 64-bit words. It returns the element at the given position. The indices should be given with respect to the 64-bit representation of the matrix.

Parameters

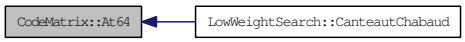
dRow The row index of the matrix.

dCol The column index of the matrix with respect to the 64-bit representation.

Returns

The value at (dRow,dCol).

Here is the caller graph for this function:



3.1.3.4 bool CodeMatrix::AtBool (uint64_t *dRow*, uint64_t *dCol*)

Returns the bit at the given position.

This method returns the element at the given position with respect to the binary version of the matrix.

Parameters

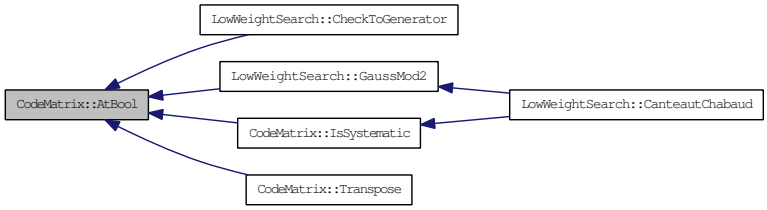
dRow The row index of the matrix.

dCol The column index of the matrix with respect to the binary representation.

Returns

The value at (dRow,dCol).

Here is the caller graph for this function:



3.1.3.5 void CodeMatrix::Build (CodeWord(∗)(uint64_t &) *pBuildFunction*, uint64_t *dDim*)

Creates a code matrix.

This method creates a matrix with dimension dDim, by calling the function pBuildFunction dDim times. The provided function should take as argument an integer and return a code word, which represents one row the code matrix.

Parameters

pBuildFunction Pointer to the function which returns a code word.

dDim Dimension of the code (=number of rows).

Examples:

allinone.cpp, and sha1me.cpp.

Here is the call graph for this function:

3.1.3.6 void CodeMatrix::DeleteColumn (uint64_t *dColumn*)

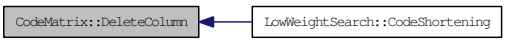
Deletes a column of the matrix.

This method deletes the specified column. The matrix is considered as binary.

Parameters

dColumn The index of the row with respect to the binary representation of the matrix.

Here is the caller graph for this function:



3.1.3.7 void CodeMatrix::DeleteRow (uint64_t *dRow*)

Deletes a row of the matrix.

This method deletes the specified row.

Parameters

dRow The index of the row.

Here is the caller graph for this function:

3.1.3.8 uint64_t CodeMatrix::GetColumns () const

Returns the number of columns.

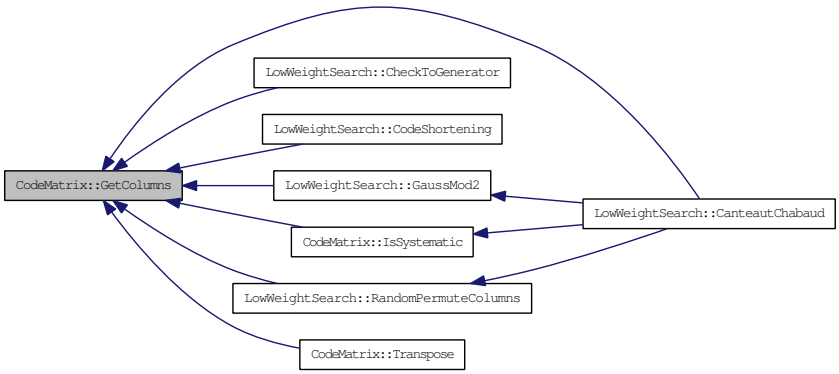
Returns

The number of columns.

Examples:

allinone.cpp, and shortening.cpp.

Here is the caller graph for this function:



3.1.3.9 uint64_t CodeMatrix::GetColumns64 () const

Returns the number of columns.

This method considers the matrix as a 64-bit word matrix.

Returns

The number of columns.

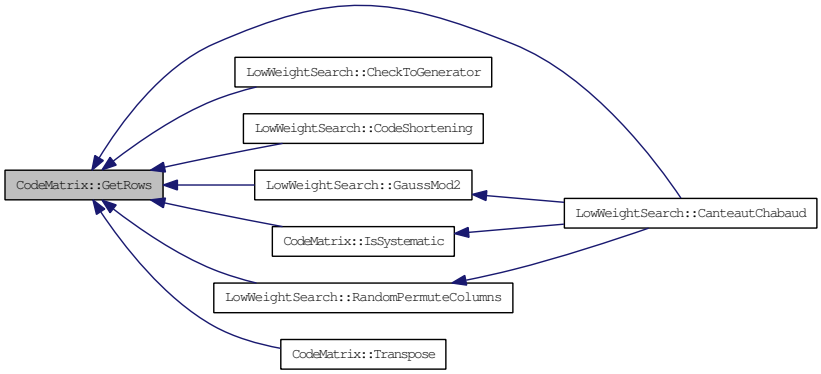
3.1.3.10 uint64_t CodeMatrix::GetRows () const

Returns the number of rows.

Returns

The number of rows.

Here is the caller graph for this function:



3.1.3.11 CodeMatrix CodeMatrix::GetSubMatrix (std::vector< uint64_t > & *vRows*, std::vector< uint64_t > & *vCols*)

Returns a submatrix of the current matrix.

This method returns a new code matrix defined by the inputs vRows and vCols.

Parameters

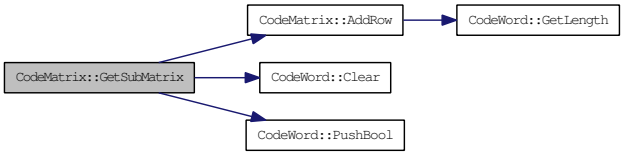
vRows The indices of the rows which should be included in the new matrix.

vCols The indices of the columns which should be included in the new matrix.

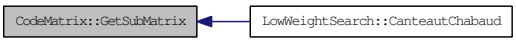
Returns

The submatrix fo the current matrix.

Here is the call graph for this function:



Here is the caller graph for this function:



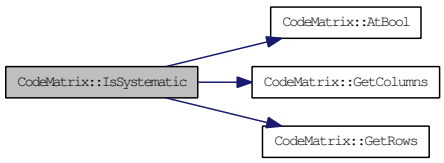
3.1.3.12 bool CodeMatrix::IsSystematic ()

Returns true if the matrix is a systematic generator matrix.

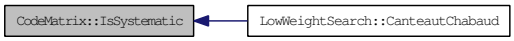
Returns

Returns true if matrix is systematic, otherwise false.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.13 CodeMatrix & CodeMatrix::operator= (const CodeMatrix & *oM*)

Overloads the assignment operator.

The overloaded operator does a deep copy of the object.

Parameters

oM The right hand side of the assignment.

Returns

The left hand side of the assignment.

3.1.3.14 CodeWord & CodeMatrix::operator[ ] (uint64_t *dIndex*)

Returns one row of the matrix.

This method returns a reference of the specified code word, which represents a row of the matrix.

Parameters

dIndex The row index.

Returns

The reference to the code word.

3.1.3.15 void CodeMatrix::PrintMatrix (const std::string & *sFileName* = "")

Outputs the matrix.

If no filename is given the matrix is written to the console. Otherwise it is written to the given file. The given file will be overwritten. This method considers the matrix as a binary matrix.

Parameters

sFileName The filename.

Examples:

sha1me.cpp.

3.1.3.16 void CodeMatrix::ReadFromFile (const std::string & *sFileName*)

Creates matrix from the given data.

This method reads data from the given file and creates the matrix. The matrix has to be in the same format as specified by PrintMatrix.

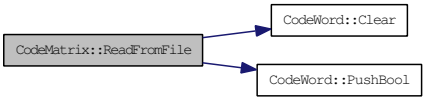
Parameters

sFileName The filename containing data.

Examples:

search.cpp, and shortening.cpp.

Here is the call graph for this function:



3.1.3.17 void CodeMatrix::Set32 (uint64_t *dRow*, uint64_t *dCol*, uint32_t *dData*)

Changes the element at the given position.

This method considers the matrix consisting of 32-bit words. It sets the element at the given position. The indices should be given with respect to the 32-bit representation of the matrix.

Parameters

dRow The row index of the matrix.

dCol The column index of the matrix with respect to the 32-bit representation.

dData The new 32-bit value.

3.1.3.18 void CodeMatrix::Set64 (uint64_t *dRow*, uint64_t *dCol*, uint64_t *dData*)

Changes the element at the given position.

This method considers the matrix consisting of 64-bit words. It sets the element at the given position. The indices should be given with respect to the 64-bit representation of the matrix.

Parameters

dRow The row index of the matrix.

dCol The column index of the matrix with respect to the 64-bit representation.

dData The new 64-bit value.

3.1.3.19 void CodeMatrix::SetBool (uint64_t *dRow*, uint64_t *dCol*, bool *dData*)

Changes the bit at the given position.

This method sets the element at the given position with respect to the binary version of the matrix.

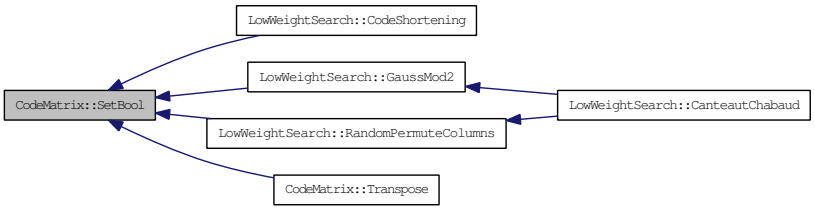
Parameters

dRow The row index of the matrix.

dCol The column index of the matrix with respect to the binary representation.

dData The new value for the element.

Here is the caller graph for this function:



3.1.3.20 CodeMatrix CodeMatrix::Transpose ()

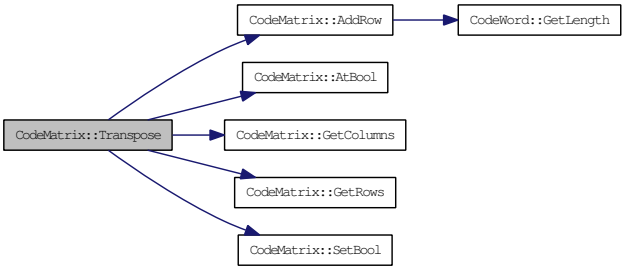
Returns the transposed matrix.

This method considers the matrix as binary.

Returns

The transposed matrix.

Here is the call graph for this function:



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

- includes/CodeMatrix.h
- src/CodeMatrix.cpp

3.2 CodeWord Class Reference

This class represents a binary code word.

`#include <CodeWord.h>`

Public Member Functions

- CodeWord (void)
  Constructor.
- virtual ~CodeWord (void)
  Destructor.
- uint64_t GetHammingWeight () const
  Returns the Hamming weight of the code word.
- uint64_t GetHammingWeight (std::vector< uint64_t > &vWeights) const
  Returns the Hamming weight where bits are weighted differently.
- uint64_t GetLength () const
  Returns the length of the code word.
- uint64_t GetLength64 () const
  Returns the length of the code word.
- std::vector< bool > GetDataBool () const
  Returns the code word as vector of bits.
- std::vector< uint32_t > GetDataUInt32 () const
  Returns the code word as a vector of 32-bit words.
- std::vector< uint64_t > GetDataUInt64 () const
  Returns the code word as a vector of 64-bit words.
- bool AtBool (uint64_t dIndex) const
  Returns the bit from the given position.
- uint32_t At32 (uint64_t dIndex) const
  Returns the 32-bit word from the given position.
- uint64_t At64 (uint64_t dIndex) const
  Returns the 64-bit word from the given position.
- void SetBool (uint64_t dIndex, bool dData)
  Sets the bit at the given position.
- void Set32 (uint64_t dIndex, uint32_t dData)
  Sets the 32-bit word at the given position.

- void Set64 (uint64_t dIndex, uint64_t dData)
  Sets the 64-bit word at the given position.
- void EraseBool (uint64_t dIndex)
  Deletes a bit at the given position.
- void Erase32 (uint64_t dIndex)
  Deletes the 32-bit word at the given position.
- void Erase64 (uint64_t dIndex)
  Deletes the 64-bit word at the given position.
- void PushBool (bool dData)
  Adds a bit at the end of the code word.
- void Push32 (uint32_t dData)
  Adds 32 bits at the end of the code word.
- void Push64 (uint64_t dData)
  Adds 64 bits at the end of the code word.
- void PopBool ()
- void Pop32 ()
- void Pop64 ()
- void Clear ()
- void PrintBool (const std::string sOutput="") const
  Outputs the code word bit-wise.
- void Print32 (const std::string sOutput="") const
  Outputs the code word as 32-bit words.
- void Print64 (const std::string sOutput="") const
  Outputs the code word as 64-bit words.
- CodeWord & operator= (const CodeWord &oCodeWord)
  Overloads the assignment operator.
- CodeWord & operator^= (const CodeWord &oCodeWord)
  Overloads the XOR operator.
- const CodeWord operator^ (const CodeWord &oCodeWord) const
  Overloads the XOR operator.

3.2.1 Detailed Description

This class represents a binary code word. The data structure consists of a vector from STL containing either 32-bit or 64-bit words, depending on the target architecture. Data can be added and accessed in different ways. A code word can be considered as a bit, 32-bit word or 64-bit word (little endian). The length of a code word refers to the bit representation.

Examples:

allinone.cpp, search.cpp, sha1me.cpp, and shortening.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CodeWord::CodeWord (void)

Constructor.

Does nothing special.

Here is the caller graph for this function:

3.2.2.2 CodeWord::~CodeWord (void) `[virtual]`

Destructor.

Does nothing special.

3.2.3 Member Function Documentation

3.2.3.1 uint32_t CodeWord::At32 (uint64_t *dIndex*) const

Returns the 32-bit word from the given position.

This method returns a 32-bit word from the given position with respect to the 32-bit word representation of the code word.

Parameters

dIndex The position of the element.

Returns

The value at dIndex.

Examples:

allinone.cpp, and shortening.cpp.

3.2.3.2 uint64_t CodeWord::At64 (uint64_t *dIndex*) const

Returns the 64-bit word from the given position.

This method returns a 64-bit word from the given position with respect to the 64-bit word representation of the code word.

Parameters

dIndex The position of the element.

Returns

The value at dIndex.

3.2.3.3 bool CodeWord::AtBool (uint64_t *dIndex*) const

Returns the bit from the given position.

This method returns the bit from the given position with respect to the bit representation of the code word.

Parameters

dIndex The position of the bit.

Returns

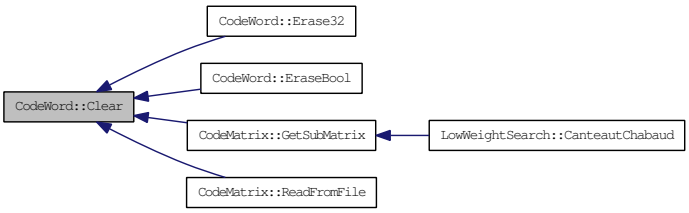
The value at dIndex.

Here is the caller graph for this function:

3.2.3.4 void CodeWord::Clear ()

Deletes all data.

Here is the caller graph for this function:



3.2.3.5 void CodeWord::Erase32 (uint64_t *dIndex*)

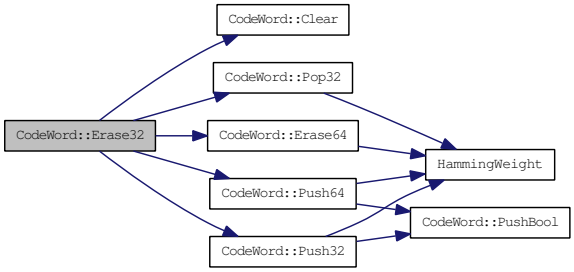
Deletes the 32-bit word at the given position.

This method deletes the 32-bit word at the given position with respect to the 32-bit word representation of the code word.

Parameters

dIndex The position of the element.

Here is the call graph for this function:



3.2.3.6 void CodeWord::Erase64 (uint64_t *dIndex*)

Deletes the 64-bit word at the given position.

This method deletes the 64-bit word at the given position with respect to the 64-bit word representation of the code word.

Parameters

dIndex The position of the element.

Here is the call graph for this function:

Here is the caller graph for this function:

3.2.3.7 void CodeWord::EraseBool (uint64_t *dIndex*)

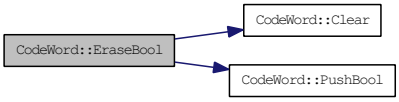
Deletes a bit at the given position.

This method deletes the bit at the given position with respect to the bit representation of the code word.

Parameters

dIndex The position of the bit.

Here is the call graph for this function:



3.2.3.8 std::vector< bool > CodeWord::GetDataBool () const

Returns the code word as vector of bits.

Returns

The vector containing the bits of the code word.

Here is the caller graph for this function:

3.2.3.9 std::vector< uint32_t > CodeWord::GetDataUInt32 () const

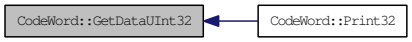
Returns the code word as a vector of 32-bit words.

The method returns the code word considered as 32-bit words in little endian format.

Returns

The 32-bit words of the code word.

Here is the caller graph for this function:



3.2.3.10 std::vector< uint64_t > CodeWord::GetDataUInt64 () const

Returns the code word as a vector of 64-bit words.

The method returns the code word considered as 64-bit words in little endian format.

Returns

The 64-bit words of the code word.

Here is the caller graph for this function:

3.2.3.11 uint64_t CodeWord::GetHammingWeight (std::vector< uint64_t > & *vWeights*) const

Returns the Hamming weight where bits are weighted differently.

Parameters

vWeights Contains the weight for each bit of the code word.

Returns

The weighted Hamming weight.

3.2.3.12 uint64_t CodeWord::GetHammingWeight () const

Returns the Hamming weight of the code word.

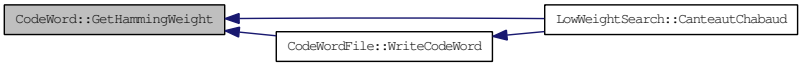
Returns

The Hamming weight of the code word.

Examples:

allinone.cpp, search.cpp, and shortening.cpp.

Here is the caller graph for this function:



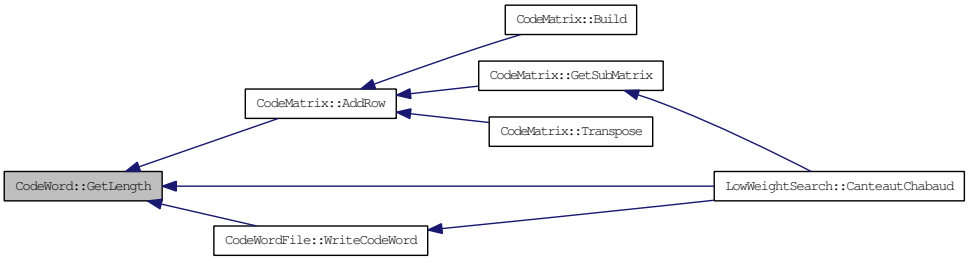
3.2.3.13 uint64_t CodeWord::GetLength () const

Returns the length of the code word.

Returns

The length of the code word with respect to its binary representation.

Here is the caller graph for this function:



3.2.3.14 uint64_t CodeWord::GetLength64 () const

Returns the length of the code word.

Returns

The length of the code word with respect to its 64-bit words representation.

3.2.3.15 CodeWord & CodeWord::operator= (const CodeWord & *oCodeWord*)

Overloads the assignment operator.

The overloaded operator does a deep copy of the object.

Parameters

oCodeWord The right hand side of the assignment.

Returns

The left hand side of the assignment.

3.2.3.16 const CodeWord CodeWord::operator^ (const CodeWord & *oCodeWord*) const

Overloads the XOR operator.

The overloaded operator xors each elements of the code words.

Parameters

oCodeWord The right hand side of the assignment.

Returns

The left hand side of the assignment.

Here is the call graph for this function:

3.2.3.17 CodeWord & CodeWord::operator^= (const CodeWord & *oCodeWord*)

Overloads the XOR operator.

The overloaded operator xors each elements of the code words.

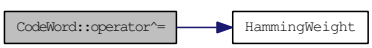
Parameters

oCodeWord The right hand side of the assignment.

Returns

The left hand side of the assignment.

Here is the call graph for this function:



3.2.3.18 void CodeWord::Pop32 ()

Removes the last 32 bits of the code word.

Here is the call graph for this function:

Here is the caller graph for this function:

3.2.3.19 void CodeWord::Pop64 ()

Removes the last 64 bits of the code word.

Here is the call graph for this function:

3.2.3.20 void CodeWord::PopBool ()

Removes the last bit of the code word.

Here is the call graph for this function:

3.2.3.21 void CodeWord::Print32 (const std::string *sOutput* = "") const

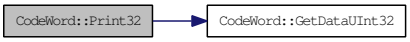
Outputs the code word as 32-bit words.

This method writes the code word to the given file or if no filename is given to the console. The data is written as hexadecimal numbers.

Parameters

sOutput The file name.

Here is the call graph for this function:



3.2.3.22 void CodeWord::Print64 (const std::string *sOutput* = "") const

Outputs the code word as 64-bit words.

This method writes the code word to the given file or if no filename is given to the console. The data is written as hexadecimal numbers.

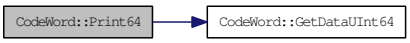
Parameters

sOutput The file name.

Examples:

allinone.cpp, search.cpp, and shortening.cpp.

Here is the call graph for this function:



3.2.3.23 void CodeWord::PrintBool (const std::string *sOutput* = "") const

Outputs the code word bit-wise.

This method writes the code word to the given file or if no filename is given to the console.

Parameters

sOutput The filename.

Here is the call graph for this function:

3.2.3.24 void CodeWord::Push32 (uint32_t *dData*)

Adds 32 bits at the end of the code word.

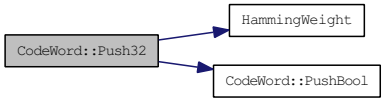
Parameters

dData The new bits.

Examples:

allinone.cpp, and sha1me.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

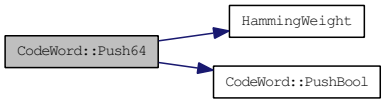
3.2.3.25 void CodeWord::Push64 (uint64_t *dData*)

Adds 64 bits at the end of the code word.

Parameters

dData The new bits.

Here is the call graph for this function:



Here is the caller graph for this function:

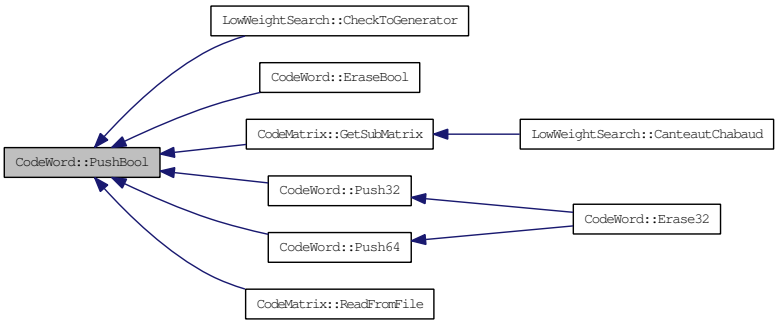
3.2.3.26 void CodeWord::PushBool (bool *dData*)

Adds a bit at the end of the code word.

Parameters

dData The new bit.

Here is the caller graph for this function:



3.2.3.27 void CodeWord::Set32 (uint64_t *dIndex*, uint32_t *dData*)

Sets the 32-bit word at the given position.

This method sets the 32-bit word at the given position with respect to the 32-bit word representation of the code word.

Parameters

dIndex The position of the element.

dData The new value.

Here is the call graph for this function:

3.2.3.28 void CodeWord::Set64 (uint64_t *dIndex*, uint64_t *dData*)

Sets the 64-bit word at the given position.

This method sets the 64-bit word at the given position with respect to the 64-bit word representation of the code word.

Parameters

dIndex The position of the element.

dData The new value.

Here is the call graph for this function:

3.2.3.29 void CodeWord::SetBool (uint64_t *dIndex*, bool *dData*)

Sets the bit at the given position.

This method sets the bit at the given position with respect to the bit representation of the code word.

Parameters

dIndex The position of the bit.

dData The new value.

Here is the call graph for this function:

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

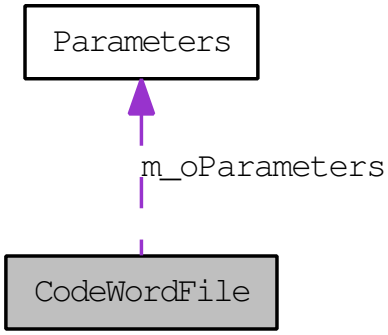
- includes/CodeWord.h
- src/CodeWord.cpp

3.3 CodeWordFile Class Reference

This class reads and writes files containing code words.

`#include <CodeWordFile.h>`

Collaboration diagram for CodeWordFile:



Public Types

- typedef std::vector< CodeWord > CodeWordList
  This is a list holding code words.

Public Member Functions

- CodeWordFile ()
  Constructor.
- CodeWordFile (std::string &sFileName)
  Constructor.
- virtual ~CodeWordFile (void)
  Destructor.
- bool Read (const std::string &sFileName)
  Reads a code word file.
- bool Write (const std::string &sFileName)
  Writes the header to the file.
- bool WriteCodeWord (const CodeWord &oCodeWord)
  Writes the code word to the file.
- void SetFileName (const std::string &sName)
  Sets the filename.
- void SetParameters (const Parameters &oParameters)

Sets the parameter object.

- const Parameters & GetParameters ()

  Returns the parameters.

- const std::string & GetFileName ()

  Return the filename.

- const CodeWordList & GetCodeWords ()

  Returns the list of code words from a file.

3.3.1 Detailed Description

This class reads and writes files containing code words. This class handles files containing code words generated by LowWeightSearch::CanteautChabaud. The parameters used for the search are also written to the file.

The format of such files is:

%BEGIN

list of integer parameters used to find code words

%STRING

list of string parameters used to find code words

%END

hamming_weight2 code_word2

hamming_weight2 code_word2

.

.

.

hamming_weightn code_wordn

If the class is used to read a file then the parameters and code words are read from file. If the class is used to write to a file then first the parameters are written. Afterwards CodeWordFile::WriteCodeWord has to be used to write a code word directly into the file after the header.

See also

Parameters
LowWeightSearch

3.3.2 Member Typedef Documentation

3.3.2.1 typedef std::vector<CodeWord> CodeWordFile::CodeWordList

This is a list holding code words.

This type definition is only used in context with a CodeWordFile object.

3.3.3 Constructor & Destructor Documentation

3.3.3.1 CodeWordFile::CodeWordFile ()

Constructor.

Does nothing special.

3.3.3.2 CodeWordFile::CodeWordFile (std::string & *sFileName*)

Constructor.

Parameters

sFileName Sets the filename.

3.3.3.3 CodeWordFile::~CodeWordFile (void) `[virtual]`

Destructor.

Does nothing special.

3.3.4 Member Function Documentation

3.3.4.1 const CodeWordFile::CodeWordList & CodeWordFile::GetCodeWords ()

Returns the list of code words from a file.

Returns

The list of code words.

3.3.4.2 const std::string & CodeWordFile::GetFileName ()

Return the filename.

Returns

The filename.

3.3.4.3 const Parameters & CodeWordFile::GetParameters ()

Returns the parameters.

Returns

The parameters used for the generated code words.

3.3.4.4 bool CodeWordFile::Read (const std::string & *sFileName*)

Reads a code word file.

Reads the header and the code words from the given file.

Parameters

sFileName The filename.

Returns

True if the file was successfully read.

3.3.4.5 void CodeWordFile::SetFileName (const std::string & *sName*)

Sets the filename.

Parameters

sName The filename.

3.3.4.6 void CodeWordFile::SetParameters (const Parameters & *oParameters*)

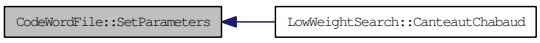
Sets the parameter object.

This method sets the object with the parameters used by the library.

Parameters

oParameters The parameter object.

Here is the caller graph for this function:



3.3.4.7 bool CodeWordFile::Write (const std::string & *sFileName*)

Writes the header to the file.

After CodeWord::Write is called, one can add code words to the file using CodeWord::WriteCodeWord. If the filename does not exist an error is printed.

Parameters

sFileName The filename.

Returns

True if header was successfully written.

Here is the caller graph for this function:

3.3.4.8 bool CodeWordFile::WriteCodeWord (const CodeWord & *oCodeWord*)

Writes the code word to the file.

The file has to be specified either by CodeWord::Write or by CodeWord::SetFileName. The Hamming weight is written first. The code word is written bit-wise, where bits are separated by a whitespace.

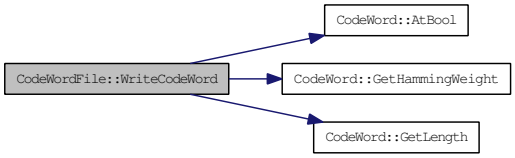
Parameters

oCodeWord the code word.

Returns

True if data was successfully written.

Here is the call graph for this function:



Here is the caller graph for this function:

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

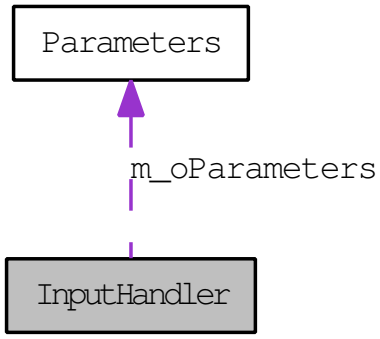
- includes/CodeWordFile.h
- src/CodeWordFile.cpp

3.4 InputHandler Class Reference

This class handles the arguments from the command line interface.

`#include <InputHandler.h>`

Collaboration diagram for InputHandler:



Public Member Functions

- InputHandler (Parameters &oParameters)

  Constructor.

- virtual ~InputHandler ()

  Destructor.

- bool ParseSettings (int argc, const char *argv[ ])

  Parses command line arguments and stores them in a Parameters object.

- void PrintUsage ()

  Prints the usage of a program.

3.4.1 Detailed Description

This class handles the arguments from the command line interface. This class provides the functionality of parsing command line arguments. The valid arguments are defined in the Parameters object. Several default parameters with respect to the low weight search algorithm are already included.

The default parameters are:

- Parameters::SIGMA sigma
- Parameters::ITER number of iteration
- Parameters::MINIMUM minimum weight
- Parameters::OUTPUT code word file for the output
- Parameters::DOUTPUT disable output

- Parameters::PERMUTE permute the columns of a generator matrix
- Parameters::CWFILE code word file for the input
- Parameters::CMFILE file containing a code matrix

See also

Parameters

Examples:

allinone.cpp, search.cpp, and shortening.cpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 InputHandler::InputHandler (Parameters & *oParameters*)

Constructor.

The constructor needs a reference of a Parameters object. Only parameters already included in this object are considered as valid arguments. The default values are overwritten if an argument was found by InputHandler::ParseSettings. Several default parameters are added in the constructor.

Parameters

oParameters A reference to the Parameters object.

Here is the call graph for this function:

3.4.2.2 InputHandler::~InputHandler () `[virtual]`

Destructor.

Does nothing special.

3.4.3 Member Function Documentation

3.4.3.1 bool InputHandler::ParseSettings (int *argc*, const char * *argv*[ ])

Parses command line arguments and stores them in a Parameters object.

Parameters

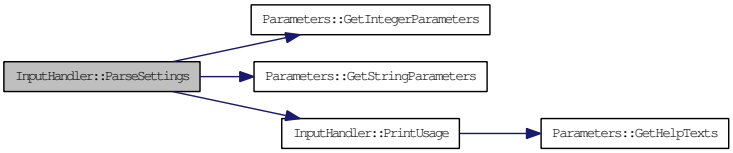
argc Number of arguments.

argv List of arguments.

Examples:

search.cpp.

Here is the call graph for this function:



3.4.3.2 void InputHandler::PrintUsage ()

Prints the usage of a program.

Prints the help text of each parameter.

Here is the call graph for this function:

Here is the caller graph for this function:

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

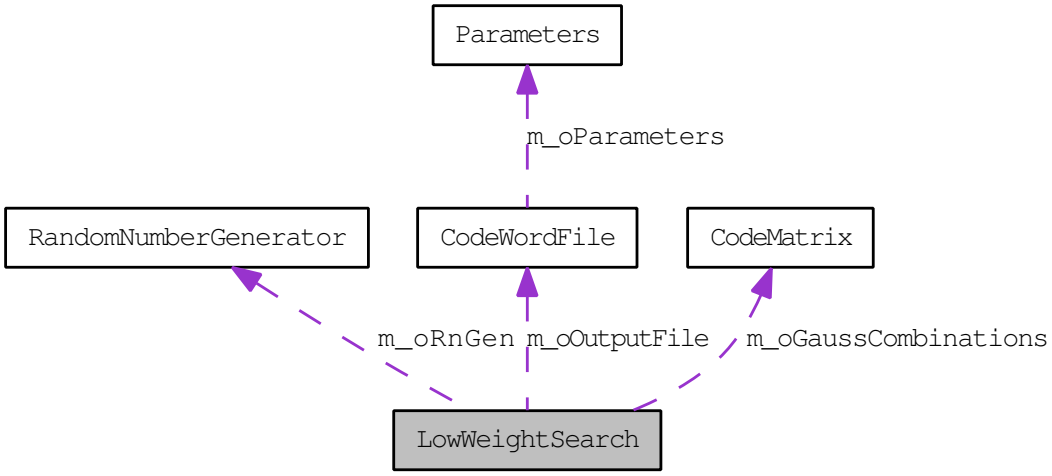
- includes/InputHandler.h
- src/InputHandler.cpp

3.5 LowWeightSearch Class Reference

The main part of the CodingTool library.

`#include <LowWeightSearch.h>`

Collaboration diagram for LowWeightSearch:



Classes

- struct **HashTableRecord**

  An entry for the hash table used in LowWeightSearch::CanteautChabaud.

Public Member Functions

- LowWeightSearch ()

  Constructor.

- virtual ~LowWeightSearch ()

  Destructor.

- CodeWord CanteautChabaud (CodeMatrix oGenerator, Parameters &oParameters)

  An algorithm to find code words with low Hamming weight.

- std::vector< uint64_t > GaussMod2 (CodeMatrix &oMatrix)

  Transforms the matrix to reduced row echelon form.

- std::vector< uint64_t > RandomPermuteColumns (CodeMatrix &oMatrix)

  Randomly permutes the columns of the given matrix.

- void SetCheckFunction (bool(*pCheckFunction)(CodeWord &))

  Sets the check function for the code words.

- std::vector< uint64_t > GetCombinedRows ()

  Returns the indices of the combined rows.

- CodeMatrix & GetGaussCombinations ()

  Returns the performed Gaussian combinations during the search.

- void SetWeightVector (std::vector< uint64_t > &vWeights)

  Sets the weight vector.

Static Public Member Functions

- static CodeMatrix CheckToGenerator (CodeMatrix &oCheckMatrix)

  Transforms a check matrix to a generator matrix.

- static CodeMatrix CodeShortening (CodeMatrix &oMatrix, std::vector< uint64_t > &vColumns)

  Forces specific columns of the given matrix to zero.

3.5.1 Detailed Description

The main part of the CodingTool library. Finding code words in a linear code with low Hamming weights is very useful in cryptanalysis. This class provides several functionality for cryptanalysts. Combined with the other classes it simplifies the usage of coding theory in cryptanalysis.

The main function is LowWeightSearch::CanteautChabaud, which implements an algorithm for finding low Hamming word code words. The user has only to provide a generator matrix (CodeMatrix). The algorithm returns a code word with the lowest Hamming weight found.

Other functionalities are:

- LowWeightSearch::GaussMod2 : takes as input a code matrix and returns the matrix in reduced row echelon form.
- LowWeightSearch::RandomPermuteColumns : randomly permutes columns of a code matrix.
- LowWeightSearch::SetCheckFunction : sets a user defined function which checks the validity of a code word during the search.
- LowWeightSearch::CheckToGenerator : transforms a check matrix to a generator matrix.
- LowWeightSearch::SetWeightVector : sets weights for specific bits of the code, if not each bit should be weighted equally.
- LowWeightSearch::CodeShortening :shortens the linear code to eliminate specific columns. This can be useful for linearized Hash functions to find only code words which produce a collision.

Examples:

allinone.cpp, search.cpp, and shortening.cpp.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 LowWeightSearch::LowWeightSearch ()

Constructor.

Does nothing special.

3.5.2.2 LowWeightSearch::~LowWeightSearch () [virtual]

Destructor.

Does nothing special.

3.5.3 Member Function Documentation

3.5.3.1 CodeWord LowWeightSearch::CanteautChabaud (CodeMatrix *oGenerator*, Parameters & *oParameters*)

An algorithm to find code words with low Hamming weight.

This method implements the low weight search algorithm by Canteaut and Chabaud. The user has to provide a code matrix, which represents the generator matrix. If the matrix is not systematic, it is transformed into a systematic generator matrix. The Parameters object contains the search parameters like sigma, number of iterations or minimum weight (see default parameters in InputHandler). The algorithm returns the code word with lowest Hamming weight found. The algorithm stops after the given number of iterations or the given minimum weight is reached.

The user can specify a check function using LowWeightSearch::SetCheckFunction. This functions is applied to each code word found and determines if the code word is stored or discarded.

By using LowWeightSearch::SetWeightVector one can define a weight vector if different bits of the code word should be weighted differently.

If Parameters::PERMUTE is set the columns of the generator matrix are permuted using LowWeightSearch::RandomPermuteColumns.

This class offers a lot of possible improvements. Additional (faster) search algorithms or faster implementations can easily be added.

Parameters

oGenerator The generator matrix.

oParameters The parameters for the search.

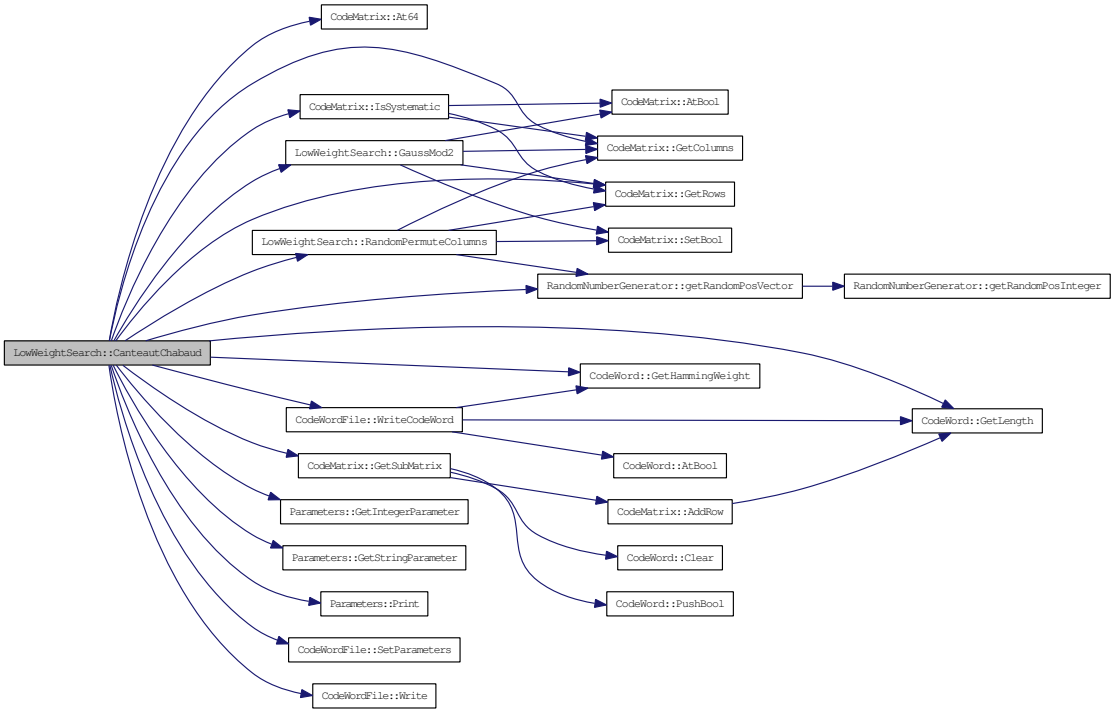
Returns

The code word with the lowest Hamming weight found.

Examples:

allinone.cpp, search.cpp, and shortening.cpp.

Here is the call graph for this function:



3.5.3.2 CodeMatrix LowWeightSearch::CheckToGenerator (CodeMatrix & *oCheckMatrix*) [static]

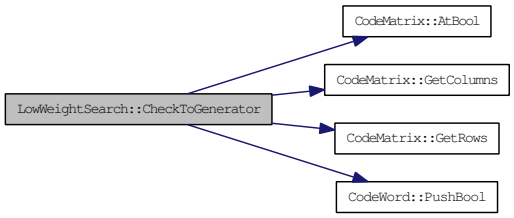
Transforms a check matrix to a generator matrix.

A cryptanalyst may face scenarios where she/he has rather a check matrix than a generator matrix. This method transforms the given check matrix to a generator matrix. May be not necessary in the future if search algorithms are implemented which work directly with check matrices.

Parameters

oCheckMatrix The check matrix.

Here is the call graph for this function:



3.5.3.3 CodeMatrix LowWeightSearch::CodeShortening (CodeMatrix & *oMatrix*, std::vector< uint64_t > & *vColumns*) [static]

Forces specific columns of the given matrix to zero.

This method may have different purposes. One purpose is to force collisions for a linearized Hash function. If the given generator matrix was created from a linearized hash function one can use to force the last n bits of the matrix to zero by code shortening, where n is the output size of the hash function. In that way, the code words found by the search algorithm produce a collision for the linearized hash function.

This method uses Gaussian elimination to fulfill its task. The code dimension and code length is reduced. If the dimension is reduced to zero, i.e. forcing to much columns to zero may not have a solution, an error is printed.

Parameters

oMatrix The generator matrix.

vColumns Indices of the columns which should be forced to zero.

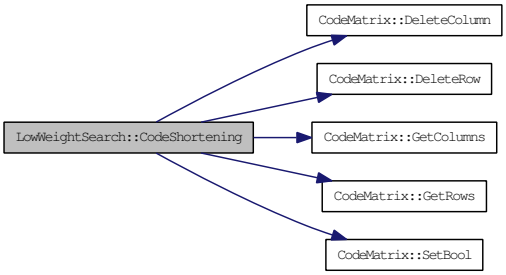
Returns

The reduce matrix.

Examples:

allinone.cpp, and shortening.cpp.

Here is the call graph for this function:



3.5.3.4 std::vector< uint64_t > LowWeightSearch::GaussMod2 (CodeMatrix & *oMatrix*)

Transforms the matrix to reduced row echelon form.

During the operations the columns maybe permuted. The method returns the permutation of the columns.

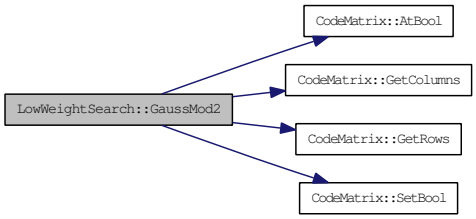
Parameters

oMatrix The code matrix.

Returns

The permutation of the columns.

Here is the call graph for this function:



Here is the caller graph for this function:

3.5.3.5 std::vector< uint64_t > LowWeightSearch::GetCombinedRows ()

Returns the indices of the combined rows.

The final code word is usually a linear combination of up to 4 rows of the generator matrix. This method returns the indices of the combined rows.

Returns

The indices of the combined rows.

3.5.3.6 CodeMatrix & LowWeightSearch::GetGaussCombinations ()

Returns the performed Gaussian combinations during the search.

After each iteration Delta Gauss is applied on the generator matrix. In parallel the same operations are applied on an identity matrix. This matrix is returned by this method.

Returns

A reference to the matrix representing the performed Gaussian operations.

3.5.3.7 std::vector< uint64_t > LowWeightSearch::RandomPermuteColumns (CodeMatrix & *oMatrix*)

Randomly permutes the columns of the given matrix.

This method can be used to increase the randomness of the search algorithm.

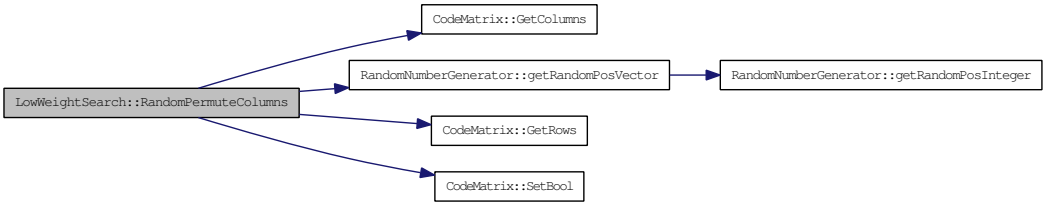
Parameters

oMatrix The code matrix.

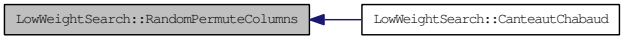
Returns

The permutation of the columns.

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.3.8 void LowWeightSearch::SetCheckFunction (bool(∗)(CodeWord &) *pCheckFunction*)

Sets the check function for the code words.

The specified check function is applied on each found code word and determines if the code word is discarded or stored.

Parameters

pCheckFunction Pointer to the check function.

3.5.3.9 void LowWeightSearch::SetWeightVector (std::vector< uint64_t > & *vWeights*)

Sets the weight vector.

Sets the weight vector if bits of the code words should be weighted differently. The length of the vector has to be the same as the length of the code.

Parameters

vWeights The weights.

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

- includes/LowWeightSearch.h
- src/LowWeightSearch.cpp

3.6 Parameters Class Reference

This is a handler for parameters used in the CodingTool.

`#include <Parameters.h>`

Public Member Functions

- Parameters (void)
  Constructor.
- virtual ~Parameters (void)
  Destructor.
- void AddParameter (const std::string &sName, const uint64_t &dValue, std::string sHelpText="")
  Adds an integer parameter.
- void AddParameter (const std::string &sName, const std::string &sValue, std::string sHelpText="")
  Adds a string parameter.
- std::string GetStringParameter (const std::string &sName)
  Returns the value of a string parameter.
- uint64_t GetIntegerParameter (const std::string &sName)
  Returns the value of an integer parameter.
- std::string GetHelpText (const std::string &sName)
  Returns the help text of a parameter.
- void SetParameter (const std::string &sName, const uint64_t &dValue)
  Changes the value of an integer parameter.
- void SetParameter (const std::string &sName, const std::string &sValue)
  Changes the value of a string parameter.
- void SetHelpText (const std::string &sName, const std::string &sValue)
  Changes the help text of a parameter.
- std::map< std::string, uint64_t > & GetIntegerParameters ()
  Returns all integer parameters.
- std::map< std::string, std::string > & GetStringParameters ()
  Returns all string parameters.
- std::map< std::string, std::string > & GetHelpTexts ()
  Returns all help texts.
- void Print ()
  Outputs parameters to the console.

Static Public Attributes

- static const std::string SIGMA = "-s"

  sigma parameter of LowWeightSearch::CanteautChabaud.

- static const std::string ITER = "-i"

  Number of iterations for LowWeightSearch::CanteautChabaud.

- static const std::string MINIMUM = "-m"

  Minimum weight for LowWeightSearch::CanteautChabaud.

- static const std::string OUTPUT = "-o"

  Code word file for LowWeightSearch::CanteautChabaud.

- static const std::string PERMUTE = "-pc"

  Flag for random permutation in LowWeightSearch::CanteautChabaud.

- static const std::string CWFILE = "-cw"

  Code word file for input.

- static const std::string CMFILE = "-cm"

  File containing a code matrix.

- static const std::string DOUTPUT = "-d"

  Flag to disable the output.

3.6.1 Detailed Description

This is a handler for parameters used in the CodingTool. This class handles parameters for the CodingTool. There are default parameters which are mainly used by LowWeightSearch::CanteautChabaud. The user has the possibility to add custom parameters, including default values, command line argument and help text. In combination with the InputHandler a custom parameter is automatically parsed from the command line.

The default parameters are:

- Parameters::SIGMA sigma
- Parameters::ITER number of iteration
- Parameters::MINIMUM minimum weight
- Parameters::OUTPUT code word file for the output
- Parameters::DOUTPUT disable output
- Parameters::PERMUTE permute the columns of a generator matrix
- Parameters::CWFILE code word file for the input
- Parameters::CMFILE file containing a code matrix

See also

InputHandler
LowWeightSearch

Examples:

allinone.cpp, search.cpp, and shortening.cpp.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Parameters::Parameters (void)

Constructor.

Does nothing special.

3.6.2.2 Parameters::~Parameters (void) [virtual]

Destructor.

Does nothing special.

3.6.3 Member Function Documentation

3.6.3.1 void Parameters::AddParameter (const std::string & *sName*, const std::string & *sValue*, std::string *sHelpText* = "")

Adds a string parameter.

This method adds a new string parameter with a default value and a optional help text. The help text is printed by InputHandler::PrintUsage.

Parameters

sName Name of the parameter.

sValue Default value.

sHelpText Help text.

3.6.3.2 void Parameters::AddParameter (const std::string & *sName*, const uint64_t & *dValue*, std::string *sHelpText* = "")

Adds an integer parameter.

This method adds a new integer parameter with a default value and a optional help text. The help text is printed by InputHandler::PrintUsage.

Parameters

sName Parameter name.

dValue Default value.

sHelpText Help text.

Examples:

allinone.cpp.

Here is the caller graph for this function:

3.6.3.3 std::string Parameters::GetHelpText (const std::string & *sName*)

Returns the help text of a parameter.

This method returns the help text for the given parameter.

Parameters

sName Name of the parameter.

Returns

Help text of the parameter.

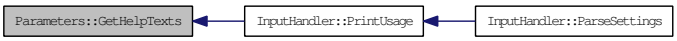
3.6.3.4 std::map< std::string, std::string > & Parameters::GetHelpTexts ()

Returns all help texts.

Returns

All help texts.

Here is the caller graph for this function:



3.6.3.5 uint64_t Parameters::GetIntegerParameter (const std::string & *sName*)

Returns the value of an integer parameter.

This method returns the value for the given parameter.

Parameters

sName Name of the parameter.

Returns

Value of the parameter.

Examples:

allinone.cpp.

Here is the caller graph for this function:

3.6.3.6 std::map< std::string, uint64_t > & Parameters::GetIntegerParameters ()

Returns all integer parameters.

Returns

All integer parameters.

Here is the caller graph for this function:

3.6.3.7 std::string Parameters::GetStringParameter (const std::string & *sName*)

Returns the value of a string parameter.

This method returns the value for the given parameter.

Parameters

sName Name of the parameter.

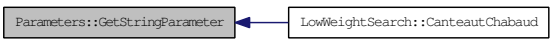
Returns

Value of the parameter.

Examples:

search.cpp, and shortening.cpp.

Here is the caller graph for this function:



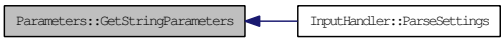
3.6.3.8 std::map< std::string, std::string > & Parameters::GetStringParameters ()

Returns all string parameters.

Returns

All string parameters.

Here is the caller graph for this function:



3.6.3.9 void Parameters::Print ()

Outputs parameters to the console.

This method writes all parameter names and values to the console.

Here is the caller graph for this function:

3.6.3.10 void Parameters::SetHelpText (const std::string & *sName*, const std::string & *sValue*)

Changes the help text of a parameter.

Parameters

sName Name of the parameter.

sValue New help text of the parameter.

3.6.3.11 void Parameters::SetParameter (const std::string & *sName*, const std::string & *sValue*)

Changes the value of a string parameter.

Parameters

sName Name of the parameter.

sValue New value of the parameter.

3.6.3.12 void Parameters::SetParameter (const std::string & *sName*, const uint64_t & *dValue*)

Changes the value of an integer parameter.

Parameters

sName Name of the parameter.

dValue New value of the parameter.

3.6.4 Member Data Documentation

3.6.4.1 const std::string Parameters::CMFILE = "-cm" [static]

File containing a code matrix.

Examples:

search.cpp, and shortening.cpp.

3.6.4.2 const std::string Parameters::CWFILE = "-cw" [static]

Code word file for input.

3.6.4.3 const std::string Parameters::DOUTPUT = "-d" [static]

Flag to disable the output.

3.6.4.4 const std::string Parameters::ITER = "-i" [static]

Number of iterations for LowWeightSearch::CanteautChabaud.

3.6.4.5 const std::string Parameters::MINIMUM = "-m" [static]

Minimum weight for LowWeightSearch::CanteautChabaud.

3.6.4.6 const std::string Parameters::OUTPUT = "-o" [static]

Code word file for LowWeightSearch::CanteautChabaud.

3.6.4.7 const std::string Parameters::PERMUTE = "-pc" [static]

Flag for random permutation in LowWeightSearch::CanteautChabaud.

3.6.4.8 const std::string Parameters::SIGMA = "-s" [static]

sigma parameter of LowWeightSearch::CanteautChabaud.

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

- includes/Parameters.h
- src/Parameters.cpp

3.7 RandomNumberGenerator Class Reference

This class provides access to a random number generator.

`#include <RandomNumberGenerator.h>`

Public Member Functions

- RandomNumberGenerator ()

  Constructor.

- virtual ~RandomNumberGenerator ()

  Destructor.

- uint64_t getRandomPosInteger (uint64_t n)

  Returns a positive integer.

- std::vector< uint64_t > getRandomPosVector (uint64_t lb, uint64_t ub, uint64_t ln)

  Returns a vector of positive integers.

- uint32_t getSeed ()

  Returns the seed of the generator.

3.7.1 Detailed Description

This class provides access to a random number generator. The CodingTool does not implement its own random number generator. Instead it is using the generator from Makoto Matsumoto and Takuji Nishimura which was ported to C++ by Jasper Bedaux (see `http://www.bedaux.net/mtrand/`).

The purpose of this interface is to make the type of random number generator easy exchangeable.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 RandomNumberGenerator::RandomNumberGenerator ()

Constructor.

The constructor initializes the random number generator using /dev/urandom on ∗nix machines and time(NULL) on Windows machines.

3.7.2.2 RandomNumberGenerator::~RandomNumberGenerator () `[virtual]`

Destructor.

Does nothing special.

3.7.3 Member Function Documentation

3.7.3.1 uint64_t RandomNumberGenerator::getRandomPosInteger (uint64_t *n*)

Returns a positive integer.

This method returns a random integer between 0 and n.

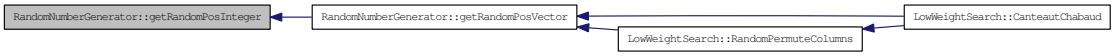
Parameters

n Maximum random number.

Returns

A random number.

Here is the caller graph for this function:



3.7.3.2 std::vector< uint64_t > RandomNumberGenerator::getRandomPosVector (uint64_t *lb*, uint64_t *ub*, uint64_t *ln*)

Returns a vector of positive integers.

This method returns a vector of random integers from the closed interval [lb,ub]. The vector does not contain duplicates.

Parameters

lb Lower bound for the random numbers.

ub Upper bound for the random numbers.

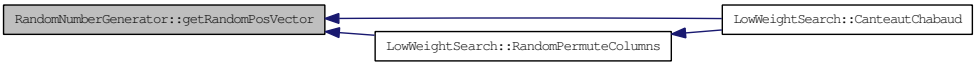
ln Number of random numbers.

Returns

A vector with random numbers.

Here is the call graph for this function:

Here is the caller graph for this function:



3.7.3.3 uint32_t RandomNumberGenerator::getSeed ()

Returns the seed of the generator.

Returns

The 32-bit word seed.

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

- includes/RandomNumberGenerator.h
- src/RandomNumberGenerator.cpp

Chapter 4

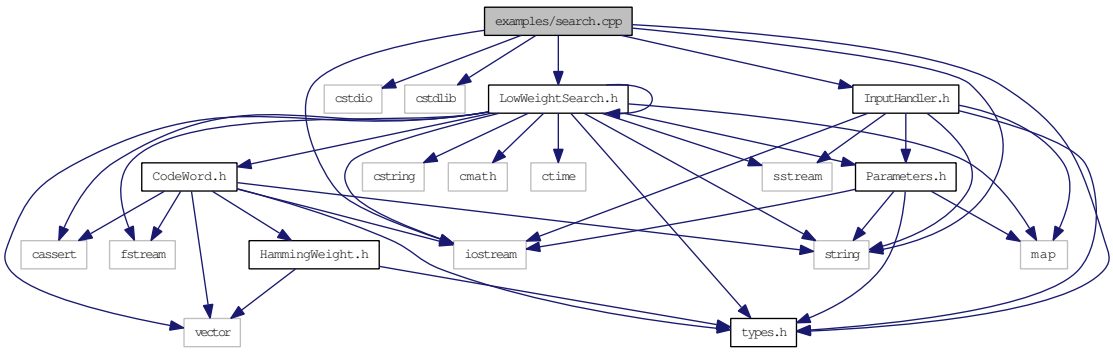
File Documentation

4.1 examples/search.cpp File Reference

Example for finding low Hamming weight words.

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>
#include "LowWeightSearch.h"
#include "InputHandler.h"
#include "types.h"
```

Include dependency graph for search.cpp:



Functions

- int **main** (int argc, const char *argv[ ])

4.1.1 Detailed Description

Example for finding low Hamming weight words.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

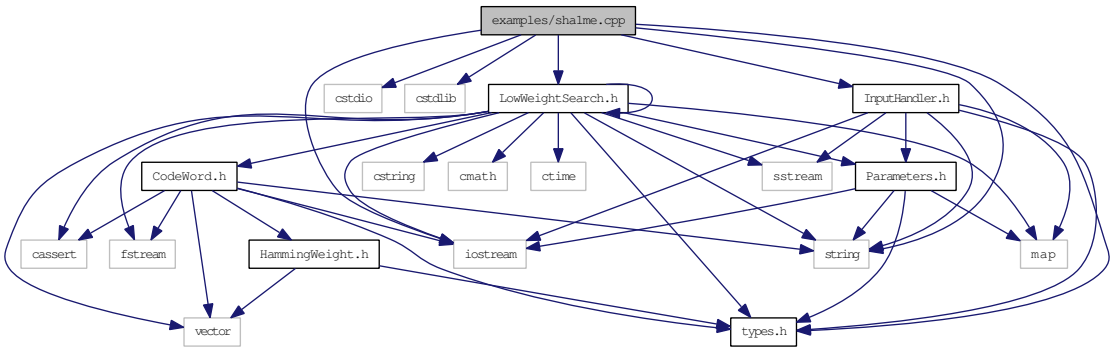
0.9

4.2 examples/sha1me.cpp File Reference

Example for creating a code matrix.

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>
#include "LowWeightSearch.h"
#include "InputHandler.h"
#include "types.h"
```

Include dependency graph for sha1me.cpp:



Defines

- #define ROTR(w, x) ((((w) & 0xFFFFFFFF) >> (x))|((w) << (32 - (x))))

  Rotate a 32-bit word to the right.

- #define ROTL(w, x) (((w) << (x))|(((w) & 0xFFFFFFFF) >> (32 - (x))))

  Rotate a 32-bit word to the left.

Functions

- CodeWord **BuildFunction** (uint64_t &dIter)
- void **SHA1ME** (uint32_t *m)
- int **main** (int argc, const char *argv[ ])

4.2.1 Detailed Description

Example for creating a code matrix.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

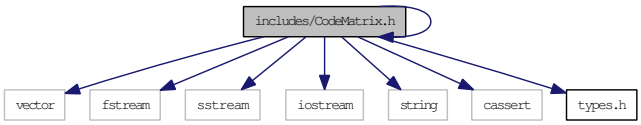
0.9

4.3 includes/CodeMatrix.h File Reference

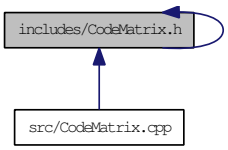
This is the header file of the class CodeMatrix.

```
#include <vector>
#include <fstream>
#include <sstream>
#include <iostream>
#include <string>
#include <cassert>
#include "types.h"
#include "types.h"
```

Include dependency graph for CodeMatrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CodeMatrix

  This class represents a binary code matrix.

4.3.1 Detailed Description

This is the header file of the class CodeMatrix.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

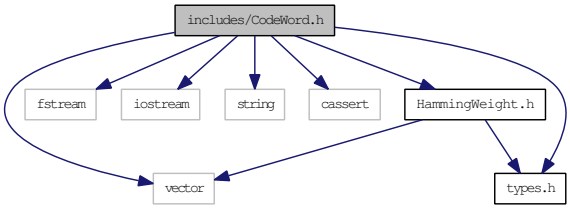
0.9

4.4 includes/CodeWord.h File Reference

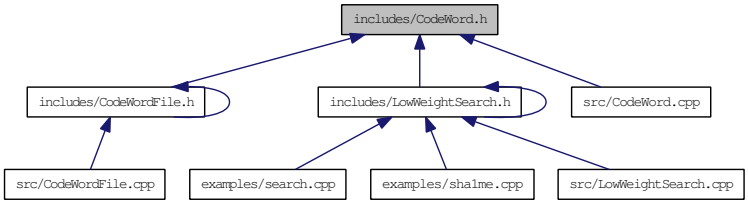
This is the header file of the class CodeWord.

```
#include <vector>
#include <fstream>
#include <iostream>
#include <string>
#include <cassert>
#include "types.h"
#include "HammingWeight.h"
```

Include dependency graph for CodeWord.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CodeWord

  This class represents a binary code word.

4.4.1 Detailed Description

This is the header file of the class CodeWord.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

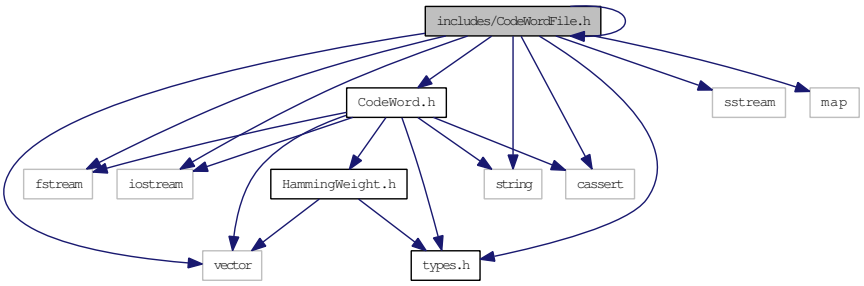
0.9

4.5 includes/CodeWordFile.h File Reference

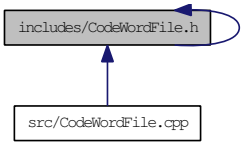
This is the header file of the class CodeWordFile.

```
#include <vector>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <cassert>
#include "types.h"
#include "Parameters.h"
#include <map>
#include "CodeWord.h"
```

Include dependency graph for CodeWordFile.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CodeWordFile

  This class reads and writes files containing code words.

4.5.1 Detailed Description

This is the header file of the class CodeWordFile.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

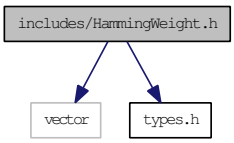
0.9

4.6 includes/HammingWeight.h File Reference

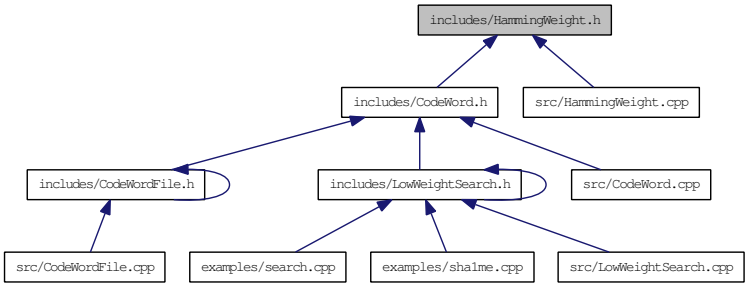
This is the header file defining functions to compute the Hamming weight of words.

```
#include <vector>
#include "types.h"
```

Include dependency graph for HammingWeight.h:



This graph shows which files directly or indirectly include this file:



Functions

- uint32_t HammingWeight (uint64_t dWord)

  Computes the Hamming weight of one word.

- uint64_t HammingWeight (uint64_t dWord, std::vector< uint64_t > &vWeights)

  Computes the Hamming weight of one word where bits are weighted differently.

4.6.1 Detailed Description

This is the header file defining functions to compute the Hamming weight of words.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

0.9

4.6.2 Function Documentation

4.6.2.1 uint64_t HammingWeight (uint64_t *dWord*, std::vector< uint64_t > & *vWeights*)

Computes the Hamming weight of one word where bits are weighted differently.

The maximum word size is 64-bit.

Parameters

dWord Value for which the Hamming weight should be computed.

vWeights The weights for the bits.

Returns

The Hamming weight.

4.6.2.2 uint32_t HammingWeight (uint64_t *dWord*)

Computes the Hamming weight of one word.

The maximum word size is 64-bit.

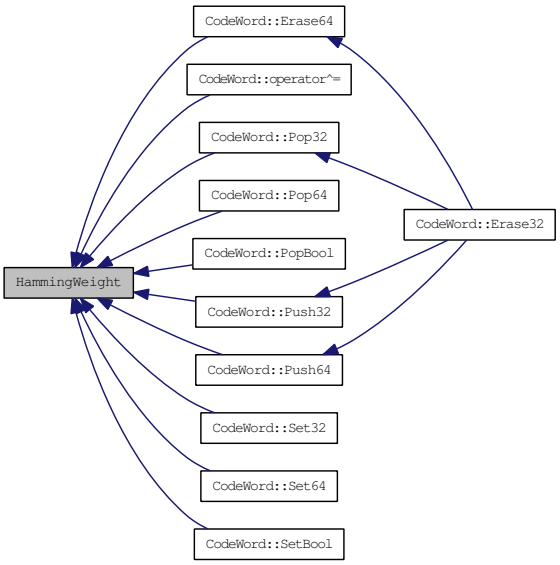
Parameters

dWord Value for which the Hamming weight should be computed.

Returns

The Hamming weight.

Here is the caller graph for this function:

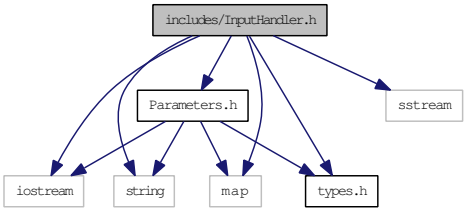


4.7 includes/InputHandler.h File Reference

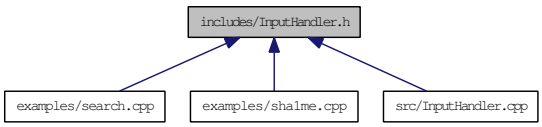
This is the header file of the class InputHandler.

```
#include <iostream>
#include <string>
#include <sstream>
#include <map>
#include "types.h"
#include "Parameters.h"
```

Include dependency graph for InputHandler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class InputHandler

  This class handles the arguments from the command line interface.

4.7.1 Detailed Description

This is the header file of the class InputHandler.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

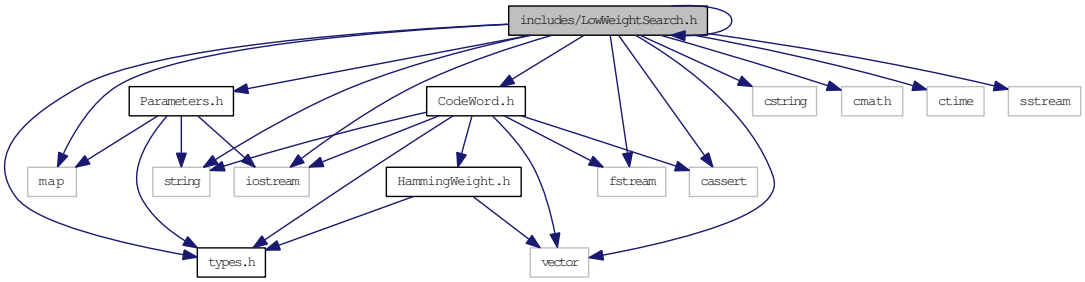
0.9

4.8 includes/LowWeightSearch.h File Reference

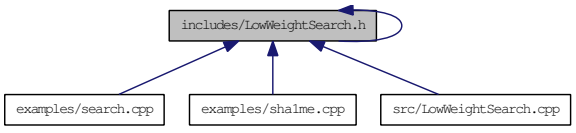
This is the header file of the class LowWeightSearch.

```
#include <vector>
#include <map>
#include <string>
#include <fstream>
#include <cstring>
#include <cmath>
#include "types.h"
#include "RandomNumberGenerator.h"
#include <ctime>
#include "CodeWord.h"
#include <sstream>
#include <iostream>
#include <cassert>
#include "Parameters.h"
```

Include dependency graph for LowWeightSearch.h:



This graph shows which files directly or indirectly include this file:



Classes

- class LowWeightSearch

  The main part of the CodingTool library.

- struct **LowWeightSearch::HashTableRecord**

  An entry for the hash table used in LowWeightSearch::CanteautChabaud.

4.8.1 Detailed Description

This is the header file of the class LowWeightSearch.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

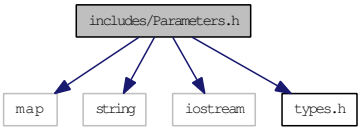
0.9

4.9 includes/Parameters.h File Reference

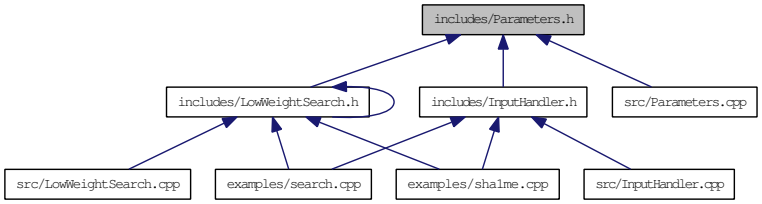
This is the header file of the class Parameters.

```
#include <map>
#include <string>
#include <iostream>
#include "types.h"
```

Include dependency graph for Parameters.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Parameters

  This is a handler for parameters used in the CodingTool.

4.9.1 Detailed Description

This is the header file of the class Parameters.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

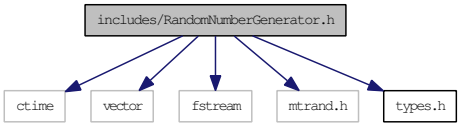
0.9

4.10 includes/RandomNumberGenerator.h File Reference

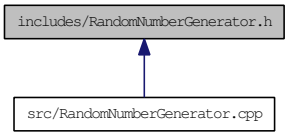
This is the header file of the class RandomNumberGenerator.

```
#include <ctime>
#include <vector>
#include <fstream>
#include "mtrand.h"
#include "types.h"
```

Include dependency graph for RandomNumberGenerator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class RandomNumberGenerator

  This class provides access to a random number generator.

4.10.1 Detailed Description

This is the header file of the class RandomNumberGenerator.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

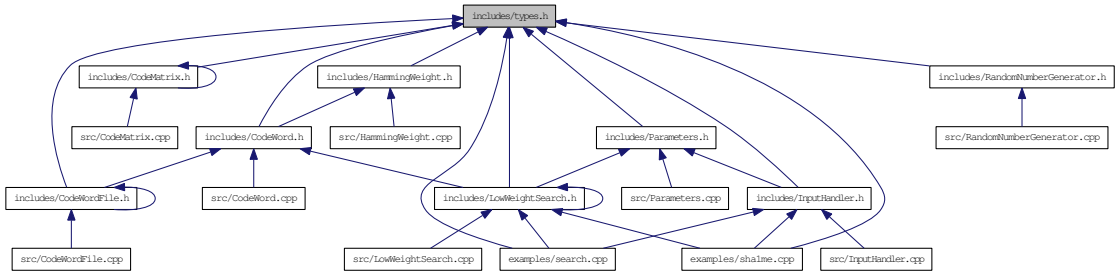
Version

0.9

4.11 includes/types.h File Reference

This file contains type definitions used through the library.

This graph shows which files directly or indirectly include this file:



4.11.1 Detailed Description

This file contains type definitions used through the library.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

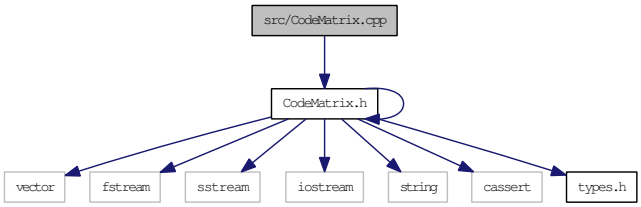
0.9

4.12 src/CodeMatrix.cpp File Reference

This file contains the implementation of the class CodeMatrix.

```
#include "CodeMatrix.h"
```

Include dependency graph for CodeMatrix.cpp:



4.12.1 Detailed Description

This file contains the implementation of the class CodeMatrix.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

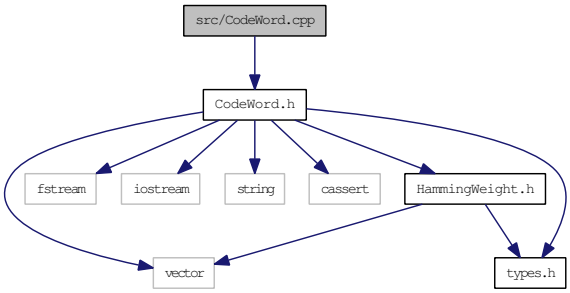
0.9

4.13 src/CodeWord.cpp File Reference

This file contains the implementation of the class CodeWord.

```
#include "CodeWord.h"
```

Include dependency graph for CodeWord.cpp:



4.13.1 Detailed Description

This file contains the implementation of the class CodeWord.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

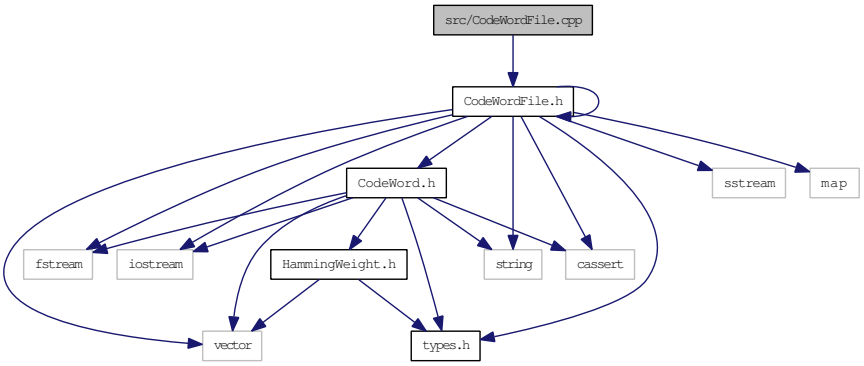
0.9

4.14 src/CodeWordFile.cpp File Reference

This file contains the implementation of the class CodeWordFile.

```
#include "CodeWordFile.h"
```

Include dependency graph for CodeWordFile.cpp:



4.14.1 Detailed Description

This file contains the implementation of the class CodeWordFile.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

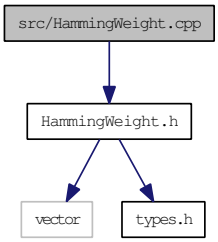
0.9

4.15 src/HammingWeight.cpp File Reference

This file implements the Hamming weight functions.

`#include "HammingWeight.h"`

Include dependency graph for HammingWeight.cpp:



Functions

- uint32_t HammingWeight (uint64_t dWord)
  Computes the Hamming weight of one word.
- uint64_t HammingWeight (uint64_t dWord, std::vector< uint64_t > &vWeights)
  Computes the Hamming weight of one word where bits are weighted differently.

4.15.1 Detailed Description

This file implements the Hamming weight functions.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

0.9

4.15.2 Function Documentation

4.15.2.1 uint64_t HammingWeight (uint64_t *dWord*, std::vector< uint64_t > & *vWeights*)

Computes the Hamming weight of one word where bits are weighted differently.

The maximum word size is 64-bit.

Parameters

dWord Value for which the Hamming weight should be computed.

vWeights The weights for the bits.

Returns

The Hamming weight.

4.15.2.2 uint32_t HammingWeight (uint64_t *dWord*)

Computes the Hamming weight of one word.

The maximum word size is 64-bit.

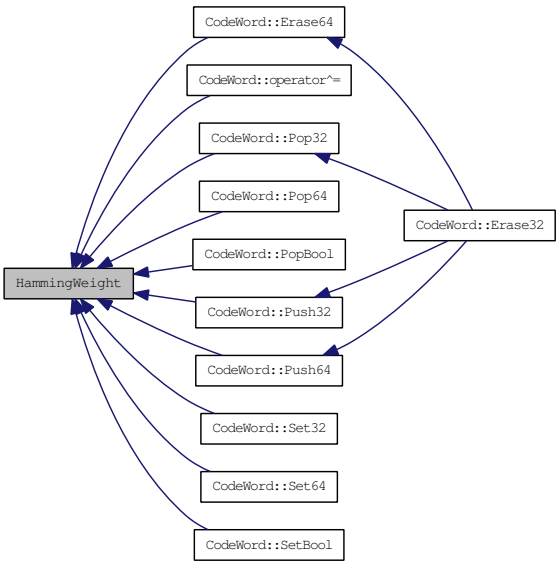
Parameters

dWord Value for which the Hamming weight should be computed.

Returns

The Hamming weight.

Here is the caller graph for this function:

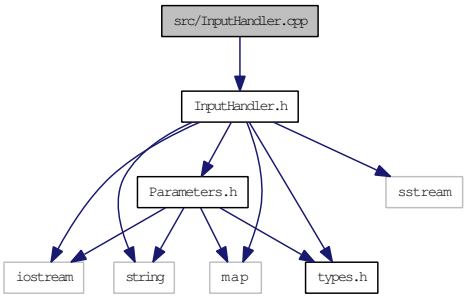


4.16 src/InputHandler.cpp File Reference

This file contains the implementation of the class InputHandler.

```
#include "InputHandler.h"
```

Include dependency graph for InputHandler.cpp:



4.16.1 Detailed Description

This file contains the implementation of the class InputHandler.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

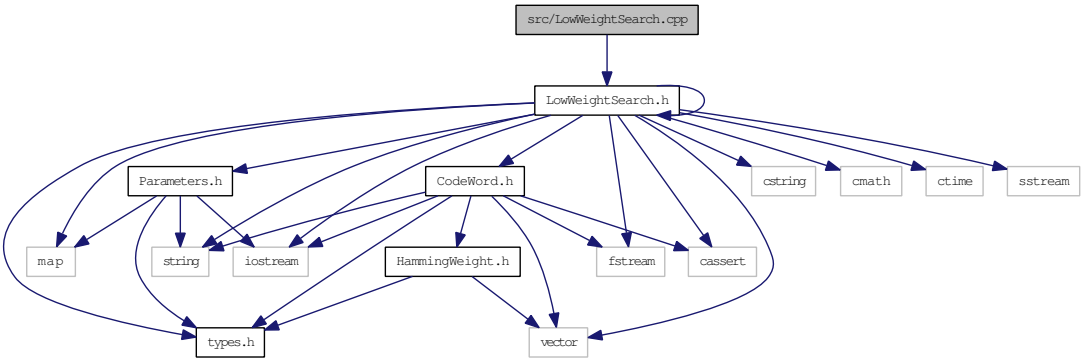
0.9

4.17 src/LowWeightSearch.cpp File Reference

This file contains the implementation of the class LowWeightSearch.

```
#include "LowWeightSearch.h"
```

Include dependency graph for LowWeightSearch.cpp:



4.17.1 Detailed Description

This file contains the implementation of the class LowWeightSearch.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

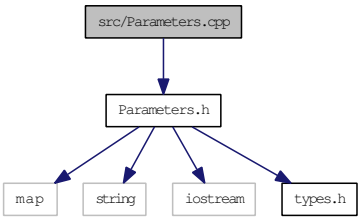
0.9

4.18 src/Parameters.cpp File Reference

This file contains the implementation of the class Parameters.

```
#include "Parameters.h"
```

Include dependency graph for Parameters.cpp:



4.18.1 Detailed Description

This file contains the implementation of the class Parameters.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

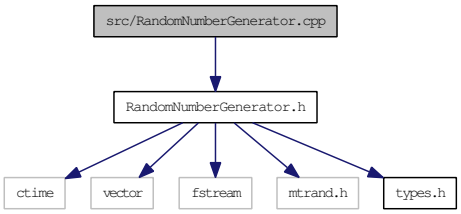
0.9

4.19 src/RandomNumberGenerator.cpp File Reference

This file contains the implementation of the class RandomNumberGenerator.

```
#include "RandomNumberGenerator.h"
```

Include dependency graph for RandomNumberGenerator.cpp:



4.19.1 Detailed Description

This file contains the implementation of the class RandomNumberGenerator.

Author

Tomislav Nad, Tomislav.Nad@iaik.tugraz.at

Version

0.9

Chapter 5

Example Documentation

5.1 allinone.cpp

This is an example which shows how one can do all the stuff in one program, instead of splitting it up like in the other examples. Also it is shown how one can add his own parameters.

Again the SHA1 message expansion is used for demonstration.

```
// Copyright (c) 2010 Graz University of Technology (IAIK) <http://www.iaik.tugra
      z.at>
//
// This file is part of the CodingTool.
//
// The CodingTool is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// CodingTool is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with CodingTool.  If not, see <http://www.gnu.org/licenses/>.

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>


#include "LowWeightSearch.h"
#include "InputHandler.h"
#include "types.h"

using namespace std;

# define ROTR(w,x) ((((w) & 0xFFFFFFFF) >> (x))|((w) << (32 - (x))))

# define ROTL(w,x) (((w) << (x))|(((w) & 0xFFFFFFFF) >> (32 - (x))))

CodeWord BuildFunction(uint64_t & dIter);
void SHA1ME(uint32_t * m);
```

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

        // create an empty generator matrix
        CodeMatrix oGenerator;
        // create an empty code word
        CodeWord oCodeWord;
        // create parameters
        Parameters oParameters;
        // create an input handler
        InputHandler oInputHandler(oParameters);
        // create the low weight search object
        LowWeightSearch oLowWS;

        // add a custom parameter
        bool bShortening = false;
        oParameters.AddParameter("-f",0,"enable code shortening");

        // parse the command line arguments
        // example: ./allinone -i 100 -o sha1me.cw -f 1
        if(oInputHandler.ParseSettings(argc, argv))
                exit(-1);

        bShortening = oParameters.GetIntegerParameter("-f");

        // use the build function to create the generator matrix
        // for the last 60 words of the SHA1 m.e.
        oGenerator.Build(&BuildFunction,512);

        // if shortening is enabled...
        if(bShortening) {
                vector<uint64_t> vForceZero;
                for(uint32_t i = 0; i < 32; i++)
                        vForceZero.push_back(oGenerator.GetColumns()-32+i);
                oGenerator = LowWeightSearch::CodeShortening(oGenerator,vForceZer
      o);
        }

        oCodeWord = oLowWS.CanteautChabaud(oGenerator,oParameters);
        oCodeWord.Print64();
        cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;

        // the last word should only be zero with "-f 1"
        uint32_t m[60];
        for(uint32_t j = 0; j<16; j++)
                m[j] = oCodeWord.At32(j);
        SHA1ME(m);

        cout << "last word = " << m[59] << endl;

        exit(1);
}

CodeWord BuildFunction(uint64_t & i) {

        CodeWord oCodeWord;
        uint32_t m[60];
        uint32_t unitv = 1;

        // 512 bit message block
        for(uint32_t j = 0; j<16; j++)
                m[j] = 0;

        // create i-th unit vector for the input
        unitv = ROTR(unitv , i+1);

        // set input to i-th unit vector
        m[i/32] = unitv;
```

```
        // call the message expansion
        SHA1ME(m);

        // add message to the code
        for(uint32_t j = 0; j<60; j++)
                oCodeWord.Push32(m[j]);

        return oCodeWord;
}


void SHA1ME(uint32_t * m) {

        // SHA-1 message expansion for the last 60 words
        for(uint32_t j = 16; j<60; j++)
                m[j] = ROTL((m[j-3] ^ m[j-8] ^ m[j-14] ^ m[j-16]),1);
}
```

5.2 search.cpp

This is an example how to read a code matrix from a file and applies a low Hamming weight search. If the matrix was previously created from the SHA1 message expansion example, the algorithm should find the Hamming weight of 25 after few iterations. Using the argument "-pc 1" which enables random permutaion of the columns, results in feewer needed iterations.

See also

sha1me.cpp

```
// Copyright (c) 2010 Graz University of Technology (IAIK) <http://www.iaik.tugra
      z.at>
//
// This file is part of the CodingTool.
//
// The CodingTool is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// CodingTool is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with CodingTool.  If not, see <http://www.gnu.org/licenses/>.

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>


#include "LowWeightSearch.h"
#include "InputHandler.h"
#include "types.h"

using namespace std;

int main(int argc, const char* argv[]) {

        // create an empty generator matrix
        CodeMatrix oGenerator;
        // create an empty code word
        CodeWord oCodeWord;
        // create parameters
        Parameters oParameters;
        // create an input handler
        InputHandler oInputHandler(oParameters);
        // create the low weight search object
        LowWeightSearch oLowWS;

        string sCMFile = "";

        // parse the command line arguments
        // example: ./search -i 100 -cm matrix.cm -pc 1
        if(oInputHandler.ParseSettings(argc, argv))
                exit(-1);

        // get the file name of the code matrix
        sCMFile = oParameters.GetStringParameter(Parameters::CMFILE);
```

```
        // read data from the file
        oGenerator.ReadFromFile(sCMFile);
        // start the search
        oCodeWord = oLowWS.CanteautChabaud(oGenerator,oParameters);

        // print the code word and the Hamming weight
        oCodeWord.Print64();
        cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;

        exit(1);
}
```

5.3 sha1me.cpp

This is an example for creating a code matrix. This example uses the SHA-1 message expansion to create a linear code. It is known that the SHA-1 message expansion code has minimum weight 25 in the last 60 words. Therefore, we create such a linear code and will use it in the other examples. The size of the code is 512x1920.

See also

search.cpp
shortening.cpp

```
// Copyright (c) 2010 Graz University of Technology (IAIK) <http://www.iaik.tugra
      z.at>
//
// This file is part of the CodingTool.
//
// The CodingTool is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// CodingTool is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with CodingTool.  If not, see <http://www.gnu.org/licenses/>.

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>


#include "LowWeightSearch.h"
#include "InputHandler.h"
#include "types.h"

using namespace std;

# define ROTR(w,x) ((((w) & 0xFFFFFFFF) >> (x))|((w) << (32 - (x))))

# define ROTL(w,x) (((w) << (x))|(((w) & 0xFFFFFFFF) >> (32 - (x))))

CodeWord BuildFunction(uint64_t & dIter);
void SHA1ME(uint32_t * m);

int main(int argc, const char* argv[]) {

        // create an empty generator matrix
        CodeMatrix oGenerator;

        // use the build function to create the generator matrix
        // with dimenstion 512
        oGenerator.Build(&BuildFunction,512);
        // save to file
        oGenerator.PrintMatrix("sha1me.cm");

        exit(1);
}

CodeWord BuildFunction(uint64_t & i) {
```

```
        CodeWord oCodeWord;
        uint32_t m[60];
        uint32_t unitv = 1;

        // 512 bit message block
        for(uint32_t j = 0; j<16; j++)
                m[j] = 0;

        // create i-th unit vector for the input
        unitv = ROTR(unitv , i+1);

        // set input to i-th unit vector
        m[i/32] = unitv;

        // call the message expansion
        SHA1ME(m);

        // add message to the code
        for(uint32_t j = 0; j<60; j++)
                oCodeWord.Push32(m[j]);

        return oCodeWord;
}


void SHA1ME(uint32_t * m) {

        // SHA-1 message expansion for the last 60 words
        for(uint32_t j = 16; j<60; j++)
                m[j] = ROTL((m[j-3] ^ m[j-8] ^ m[j-14] ^ m[j-16]),1);
}
```

5.4 shortening.cpp

This is an example how to use code shortening to force specific bits of a code word to be zero. It reads a matrix from a file and forces the last 32 bits to zero. The last word should be zero if the input matrix was previously generated by sha1me.cpp.

See also

search.cpp
sha1me.cpp

```
// Copyright (c) 2010 Graz University of Technology (IAIK) <http://www.iaik.tugra
      z.at>
//
// This file is part of the CodingTool.
//
// The CodingTool is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// CodingTool is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with CodingTool.  If not, see <http://www.gnu.org/licenses/>.

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>


#include "LowWeightSearch.h"
#include "InputHandler.h"
#include "types.h"

using namespace std;

# define ROTR(w,x) ((((w) & 0xFFFFFFFF) >> (x))|((w) << (32 - (x))))

# define ROTL(w,x) (((w) << (x))|(((w) & 0xFFFFFFFF) >> (32 - (x))))

void SHA1ME(uint32_t * m);

int main(int argc, const char* argv[]) {

        // create an empty generator matrix
        CodeMatrix oGenerator;
        // create an empty code word
        CodeWord oCodeWord;
        // create parameters
        Parameters oParameters;
        // create an input handler
        InputHandler oInputHandler(oParameters);
        // create the low weight search object
        LowWeightSearch oLowWS;

        string sCMFile = "";

        // parse the command line arguments
        // example: ./shortening -i 100 -cm sha1me.cm
```

```
        if(oInputHandler.ParseSettings(argc, argv))
                exit(-1);

        // get the file name of the code matrix
        sCMFile = oParameters.GetStringParameter(Parameters::CMFILE);

        // read data from the file
        oGenerator.ReadFromFile(sCMFile);

        // force the last 32 bits to zero;
        vector<uint64_t> vForceZero;
        for(uint32_t i = 0; i < 32; i++)
                vForceZero.push_back(oGenerator.GetColumns()-32+i);
        oGenerator = LowWeightSearch::CodeShortening(oGenerator,vForceZero);

        // start the search
        oCodeWord = oLowWS.CanteautChabaud(oGenerator,oParameters);

        // print the code word and the Hamming weight
        oCodeWord.Print64();
        cout << "Hamming weight is " << oCodeWord.GetHammingWeight() << endl;

        // if input matrix was SHA-1 message expansion:
        uint32_t m[60];
        // first 512 bits of the code word are input
        for(uint32_t j = 0; j<16; j++)
                m[j] = oCodeWord.At32(j);

        SHA1ME(m);

        // this should output 0
        cout << "last word = " << m[59] << endl;
        exit(1);
}

void SHA1ME(uint32_t * m) {

        // SHA-1 message expansion for the last 60 words
        for(uint32_t j = 16; j<60; j++)
                m[j] = ROTL((m[j-3] ^ m[j-8] ^ m[j-14] ^ m[j-16]),1);
}
```

Index