Lab 4 Pseudo Random

1.0

Generated by Doxygen 1.9.5

Chapter 1

Bug List

File generator.cpp

Currently, there are no any known bugs.

File generator.hpp

Currently, there are no any known bugs.

File model.cpp

Currently, there are no any known bugs.

File model.hpp

Currently, there are no any known bugs.

File pseudo_random.hpp

Currently, there are no any known bugs.

Currently, there are no any known bugs.

File search.cpp

Currently, there are no any known bugs.

File sorting.cpp

Currently, there are no any known bugs.

File sorting.hpp

Currently, there are no any known bugs.

Currently, there are no any known bugs.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 Generator Class Reference

A class that generates models.

`#include <generator.hpp>`

Public Member Functions

- Generator ()
  Constructor for the Generator class.
- ~Generator ()
  Destructor for the Generator class.
- Model model_generator ()
  Generates a model.

4.1.1 Detailed Description

A class that generates models.

The Generator class is responsible for generating models with random data. It uses a set of predefined lists for first names, last names, departments, and job titles to create realistic models. The generated models include information such as full name, department, job title, and a random date of birth within a certain range.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Generator()

```
Generator::Generator ( )
```

Constructor for the Generator class.

Initializes the lists of first names, last names, departments, and job titles to empty lists. It also initializes the random number generator and the distributions for generating random indices and dates.

4.1.2.2 ~Generator()

```
Generator::~Generator ( )
```

Destructor for the Generator class.

Cleans up any resources used by the Generator class.

4.1.3 Member Function Documentation

4.1.3.1 model_generator()

```
Model Generator::model_generator ( )
```

Generates a model.

Returns

The generated model.

Generates a random model by selecting a random combination of first name, last name, department, job title, and date of birth. The date of birth is within a certain range of years, and the other attributes are randomly selected from the predefined lists.

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

- src/generator/generator.hpp
- src/generator/generator.cpp

4.2 Hashing Class Reference

Contains static methods for hashing and related operations.

```
#include <model.hpp>
```

Static Public Member Functions

- static std::uint32_t basic_hashing_function (const std::string &value)

  Basic hashing function for strings.
- static std::uint32_t djb2_hashing_function (const std::string &value)

  djb2 hashing function for strings.
- static std::uint32_t advanced_hashing_function (const std::string &value)

  Advanced hashing function for strings.
- static std::uint32_t count_collisions (const std::vector< std::list< Model > > &hash_table)

  Count the number of collisions in a hash table.
- static std::vector< std::list< Model > > hash_model (std::vector< Model > &model_vector, std::function< std::size_t(const std::string &value)> hash_function)

  Hash the models into a hash table.
- static std::optional< Model > find_in_hash_table (const std::vector< std::list< Model > > &hash_table, std::uint32_t hash, std::size_t size)

  Find a model in a hash table.

4.2.1 Detailed Description

Contains static methods for hashing and related operations.

4.2.2 Member Function Documentation

4.2.2.1 advanced_hashing_function()

```
std::uint32_t Hashing::advanced_hashing_function (
            const std::string & value )  [static]
```

Advanced hashing function for strings.

Parameters

in	*value*	The value to hash

Returns

The hash value

4.2.2.2 basic_hashing_function()

```
std::uint32_t Hashing::basic_hashing_function (
            const std::string & value )  [static]
```

Basic hashing function for strings.

Parameters

in	*value*	The value to hash

Returns

The hash value

4.2.2.3 count_collisions()

```
std::uint32_t Hashing::count_collisions (
            const std::vector< std::list< Model > > & hash_table )  [static]
```

Count the number of collisions in a hash table.

Parameters

in	*hash_table*	The hash table

Returns

The number of collisions

4.2.2.4 djb2_hashing_function()

```
std::uint32_t Hashing::djb2_hashing_function (
            const std::string & value )  [static]
```

djb2 hashing function for strings.

Parameters

in	*value*	The value to hash

Returns

The hash value

4.2.2.5 find_in_hash_table()

```
static std::optional< Model > Hashing::find_in_hash_table (
            const std::vector< std::list< Model > > & hash_table,
            std::uint32_t hash,
            std::size_t size )  [static]
```

Find a model in a hash table.

Parameters

in	*hash_table*	The hash table
in	*hash*	The hash value
in	*size*	The size of the hash table

Returns

The model if found, otherwise an empty optional

4.2.2.6 hash_model()

```
std::vector< std::list< Model > > Hashing::hash_model (
            std::vector< Model > & model_vector,
            std::function< std::size_t(const std::string &value)> hash_function ) [static]
```

Hash the models into a hash table.

Parameters

in	*model_vector*	The model vector
in	*hash_function*	The hash function

Returns

The hash table

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

- src/model/model.hpp
- src/model/model.cpp

4.3 Model Class Reference

Represents an employee model.

```
#include <model.hpp>
```

Public Member Functions

- Model (std::string full_name, std::string department, std::string job_title, std::chrono::year_month_day employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)

  Constructor for the Model class.
- Model (std::string full_name, std::string department, std::string job_title, std::string employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)

  Constructor for the Model class.
- Model (std::uint8_t decor_type)

  Constructor for the Model class.
- Model (const Model &other)

  Copy constructor for the Model class.
- ~**Model** ()

  Destructor for the Model class.
- void set_model (std::string full_name, std::string department, std::string job_title, std::chrono::year_month_day employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)

  Sets the properties of the model.

- void set_model (std::string full_name, std::string department, std::string job_title, std::string employment↩ _date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std↩ ::size_t(const std::string &value)> > &optional_func=std::nullopt)

  Sets the properties of the model.
- void set_decor (std::uint8_t decor_type)

  Sets the decoration type of the model.
- void set_hash_func (std::function< std::size_t(const std::string &value)> hash_fucntion)

  Sets the hash function for the model.
- void set_hash (std::uint32_t model_hash)

  Sets the hash value for the model.
- void set_hash_field (std::uint8_t hash_field)

  Sets the hash field for the model.
- ModelComp compare_type (const Model &r_model, std::uint8_t mode)

  Compares the model with another model using specified comparison mode.
- template<typename T >
  ModelComp compare_type (std::uint8_t mode, T r_value)

  Compares the model with a value of specified type using specified comparison mode.
- template<typename T >
  T get_field (std::uint8_t field) const

  Gets the value of a specific field of the model.
- template<typename T >
  void set_field (std::uint8_t field, T value)

  Sets the value of a specific field of the model.
- std::uint32_t get_hash () const

  Gets the hash value of the model.
- std::uint8_t get_hash_field () const

  Gets the hash field of the model.
- template<typename T >
  T get_hash_field () const

  Gets the hash field of the model.

Static Public Member Functions

- static void save_model (const std::vector< Model > &model_vector, std::filesystem::path file_path)

  Saves the model vector to a file.
- static void load_model (std::vector< Model > &model_vector, std::filesystem::path file_path)

  Loads models from a file into a model vector.
- static void print_model (const std::vector< Model > &model_vector)

  Prints the models in the model vector.

Friends

- std::ostream & operator<< (std::ostream &stream, const Model &model)

  Overloading the << operator for Model class.
- ModelComp operator< (const Model &l_model, const Model &r_model)

  Overloaded less than operator for Model objects.
- ModelComp operator> (const Model &l_model, const Model &r_model)

  Overloaded greater than operator for Model objects.
- ModelComp operator<= (const Model &l_model, const Model &r_model)

  Overloaded less than or equal to operator for Model objects.

- ModelComp operator>= (const Model &l_model, const Model &r_model)

  Overloaded greater than or equal to operator for Model objects.
- ModelComp operator== (const Model &l_model, const Model &r_model)

  Overloaded equality operator for Model objects.
- ModelComp operator!= (const Model &l_model, const Model &r_model)

  Overloaded inequality operator for Model objects.

4.3.1 Detailed Description

Represents an employee model.

The Model class represents an employee with attributes such as full name, department, job title, employment date, model hash, and hash field.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Model() [1/4]

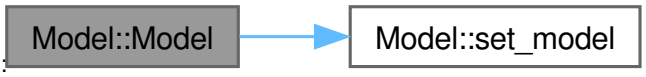
```
Model::Model (
            std::string full_name,
            std::string department,
            std::string job_title,
            std::chrono::year_month_day employment_date,
            std::uint32_t model_hash = 0,
            std::uint8_t hash_field = 255,
            const std::optional< std::function< std::size_t(const std::string &value)> > &
optional_func = std::nullopt )
```

Constructor for the Model class.

Parameters

full_name	The full name of the model.
department	The department of the model.
job_title	The job title of the model.
employment_date	The employment date of the model.
model_hash	The hash value of the model.
hash_field	The hash field of the model.
optional_func	An optional hash function for the model.

Here is the call graph for this function:



4.3.2.2 Model() [2/4]

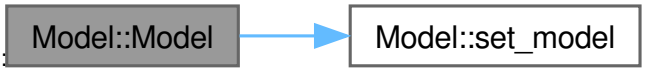
```
Model::Model (
            std::string full_name,
            std::string department,
            std::string job_title,
            std::string employment_date,
            std::uint32_t model_hash = 0,
            std::uint8_t hash_field = 255,
            const std::optional< std::function< std::size_t(const std::string &value)> > &
optional_func = std::nullopt )
```

Constructor for the Model class.

Parameters

full_name	The full name of the model.
department	The department of the model.
job_title	The job title of the model.
employment_date	The employment date of the model.
model_hash	The hash value of the model.
hash_field	The hash field of the model.
optional_func	An optional hash function for the model.

Here is the call graph for this function:



4.3.2.3 Model() [3/4]

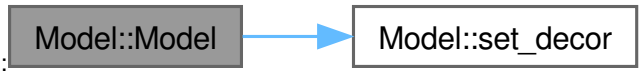
```
Model::Model (
            std::uint8_t decor_type )
```

Constructor for the Model class.

Parameters

decor_type	The decoration type of the model.

Here is the call graph for this function:



4.3.2.4 Model() [4/4]

```
Model::Model (
            const Model & other )  [inline]
```

Copy constructor for the Model class.

Parameters

other	The other model to be copied.

4.3.3 Member Function Documentation

4.3.3.1 compare_type() [1/2]

```
ModelComp Model::compare_type (
            const Model & r_model,
            std::uint8_t mode )
```

Compares the model with another model using specified comparison mode.

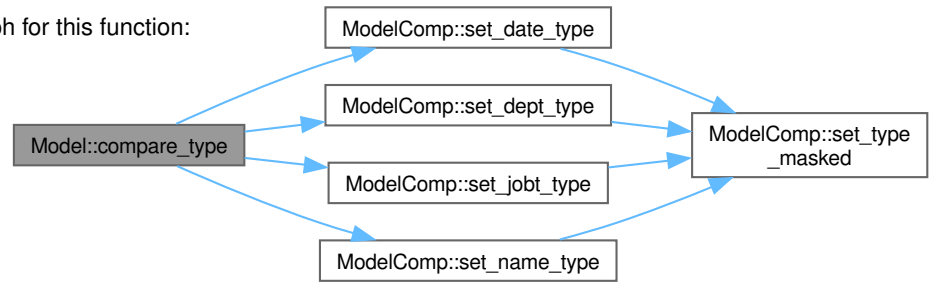
Parameters

r_model	The other model to compare with.
mode	The comparison mode.

Returns

The comparison result as a ModelComp object.

Here is the call graph for this function:



4.3.3.2 compare_type() [2/2]

```
template<typename T >
ModelComp Model::compare_type (
            std::uint8_t mode,
            T r_value )
```

Compares the model with a value of specified type using specified comparison mode.

Template Parameters

T	The type of the value to compare with.

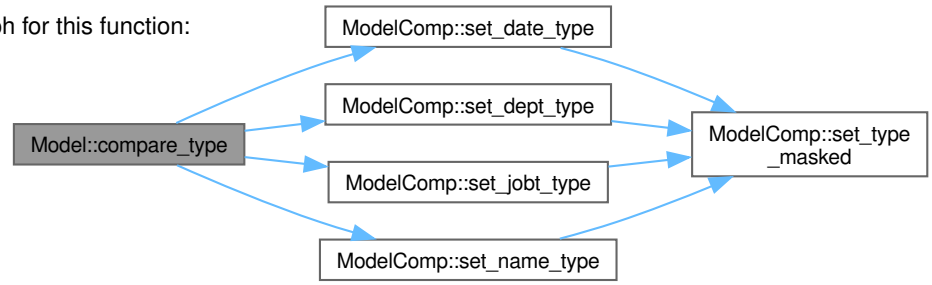
Parameters

mode	The comparison mode.
r_value	The value to compare with.

Returns

The comparison result as a ModelComp object.

Here is the call graph for this function:



4.3.3.3 get_field()

```
template<typename T >
T Model::get_field (
            std::uint8_t field ) const
```

Gets the value of a specific field of the model.

Template Parameters

T	The type of the field.

Parameters

field	The field identifier.

Returns

The value of the field.

4.3.3.4 get_hash()

```
std::uint32_t Model::get_hash ( ) const
```

Gets the hash value of the model.

Returns

The hash value.

4.3.3.5 get_hash_field() [1/2]

```
std::uint8_t Model::get_hash_field ( ) const
```

Gets the hash field of the model.

Returns

The hash field.

4.3.3.6 get_hash_field() [2/2]

```
template<typename T >
T Model::get_hash_field
```

Gets the hash field of the model.

Template Parameters

T	The type of the hash field.

Returns

The hash field.

4.3.3.7 load_model()

```
void Model::load_model (
            std::vector< Model > & model_vector,
            std::filesystem::path file_path )  [static]
```

Loads models from a file into a model vector.

Parameters

model_vector	The vector to load the models into.
file_path	The path of the file to load from.

4.3.3.8 print_model()

```
void Model::print_model (
            const std::vector< Model > & model_vector )  [static]
```

Prints the models in the model vector.

Parameters

model_vector	The vector of models to print.

Here is the call graph for this functi

4.3.3.9 save_model()

```
void Model::save_model (
            const std::vector< Model > & model_vector,
            std::filesystem::path file_path )  [static]
```

Saves the model vector to a file.

Parameters

model_vector	The vector of models to be saved.
file_path	The path of the file to save to.

4.3.3.10 set_decor()

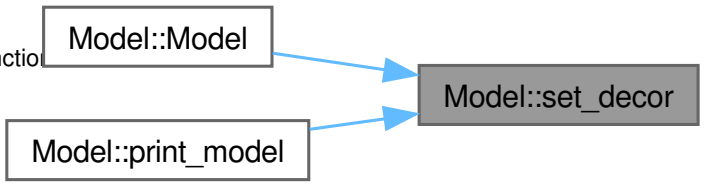
```
void Model::set_decor (
            std::uint8_t decor_type )
```

Sets the decoration type of the model.

Parameters

decor_type	The decoration type.

Here is the caller graph for this functio



4.3.3.11 set_field()

```
template<typename T >
void Model::set_field (
            std::uint8_t field,
            T value )
```

Sets the value of a specific field of the model.

Template Parameters

T	The type of the field.

Parameters

field	The field identifier.
value	The value to set.

4.3.3.12 set_hash()

```
void Model::set_hash (
            std::uint32_t model_hash )
```

Sets the hash value for the model.

Parameters

model_hash	The hash value to be set.

4.3.3.13 set_hash_field()

```
void Model::set_hash_field (
            std::uint8_t hash_field )
```

Sets the hash field for the model.

Parameters

hash_field	The hash field to be set.

4.3.3.14 set_hash_func()

```
void Model::set_hash_func (
            std::function< std::size_t(const std::string &value)> hash_fucntion )
```

Sets the hash function for the model.

Parameters

hash_function	The hash function to be set.

4.3.3.15 set_model() [1/2]

```
void Model::set_model (
            std::string full_name,
            std::string department,
            std::string job_title,
```

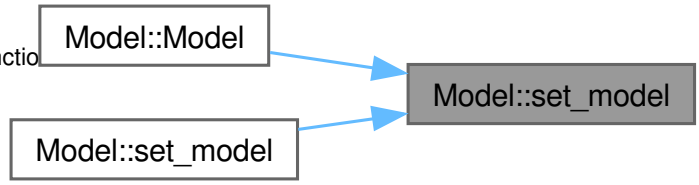
```
            std::chrono::year_month_day employment_date,
            std::uint32_t model_hash = 0,
            std::uint8_t hash_field = 255,
            const std::optional< std::function< std::size_t(const std::string &value)> > &
optional_func = std::nullopt )
```

Sets the properties of the model.

Parameters

full_name	The full name of the model.
department	The department of the model.
job_title	The job title of the model.
employment_date	The employment date of the model.
model_hash	The hash value of the model.
hash_field	The hash field of the model.
optional_func	An optional hash function for the model.

Here is the caller graph for this functio



4.3.3.16 set_model() [2/2]

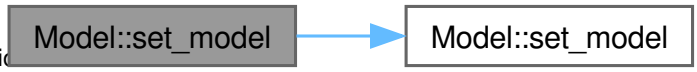
```
void Model::set_model (
            std::string full_name,
            std::string department,
            std::string job_title,
            std::string employment_date,
            std::uint32_t model_hash = 0,
            std::uint8_t hash_field = 255,
            const std::optional< std::function< std::size_t(const std::string &value)> > &
optional_func = std::nullopt )
```

Sets the properties of the model.

Parameters

full_name	The full name of the model.
department	The department of the model.
job_title	The job title of the model.
employment_date	The employment date of the model.
model_hash	The hash value of the model.
hash_field	The hash field of the model.
optional_func	An optional hash function for the model.

Here is the call graph for this functi



4.3.4 Friends And Related Function Documentation

4.3.4.1 operator"!=

```
ModelComp operator!= (
            const Model & l_model,
            const Model & r_model )  [friend]
```

Overloaded inequality operator for Model objects.

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

4.3.4.2 operator<

```
ModelComp operator< (
            const Model & l_model,
            const Model & r_model )  [friend]
```

Overloaded less than operator for Model objects.

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

4.3.4.3 operator<<

```
std::ostream & operator<< (
            std::ostream & stream,
            const Model & model )  [friend]
```

Overloading the << operator for Model class.

Parameters

stream	The output stream.
model	The model to be output.

Returns

The output stream after printing the model.

4.3.4.4 operator<=

```
ModelComp operator<= (
            const Model & l_model,
            const Model & r_model )  [friend]
```

Overloaded less than or equal to operator for Model objects.

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

4.3.4.5 operator==

```
ModelComp operator== (
            const Model & l_model,
            const Model & r_model )  [friend]
```

Overloaded equality operator for Model objects.

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

4.3.4.6 operator>

```
ModelComp operator> (
            const Model & l_model,
            const Model & r_model )  [friend]
```

Overloaded greater than operator for Model objects.

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

4.3.4.7 operator>=

```
ModelComp operator>= (
            const Model & l_model,
            const Model & r_model )  [friend]
```

Overloaded greater than or equal to operator for Model objects.

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

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

- src/model/model.hpp
- src/model/model.cpp

4.4 ModelComp Class Reference

Represents the comparison result of two models.

`#include <model.hpp>`

Public Member Functions

- **ModelComp** ()
  Constructor for ModelComp class.
- ModelComp (std::uint8_t value)
  Constructor for ModelComp class.
- ~**ModelComp** ()
  Destructor for ModelComp class.
- void set_type_masked (std::uint8_t value, std::uint8_t offset)
  Sets the type masked value at the given offset.
- void set_name_type (std::uint8_t value)
  Sets the name type value.
- void set_dept_type (std::uint8_t value)
  Sets the department type value.
- void set_jobt_type (std::uint8_t value)
  Sets the job type value.
- void set_date_type (std::uint8_t value)
  Sets the date type value.
- std::uint8_t get_type_masked (std::uint8_t offset) const
  Gets the type masked value at the given offset.
- std::uint8_t get_name_type () const
  Gets the name type value.
- std::uint8_t get_dept_type () const
  Gets the department type value.
- std::uint8_t get_jobt_type () const
  Gets the job type value.
- std::uint8_t get_date_type () const
  Gets the date type value.
- ModelComp operator! () const
  Overloaded logical NOT operator for ModelComp objects.

Friends

- ModelComp operator== (const ModelComp &l_bool, const ModelComp &r_bool)
  Overloaded equality operator for ModelComp objects.
- ModelComp operator!= (const ModelComp &l_bool, const ModelComp &r_bool)
  Overloaded inequality operator for ModelComp objects.
- std::ostream & operator<< (std::ostream &stream, const ModelComp &model)
  Overloaded stream insertion operator for ModelComp objects.

4.4.1 Detailed Description

Represents the comparison result of two models.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ModelComp()

```
ModelComp::ModelComp (
            std::uint8_t value )
```

Constructor for ModelComp class.

Parameters

value	The value to initialize the ModelComp object.

4.4.3 Member Function Documentation

4.4.3.1 get_date_type()

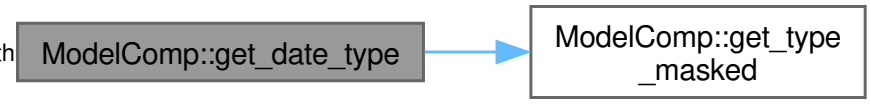
```
std::uint8_t ModelComp::get_date_type ( ) const
```

Gets the date type value.

Returns

The date type value.

Here is the call graph for th



4.4.3.2 get_dept_type()

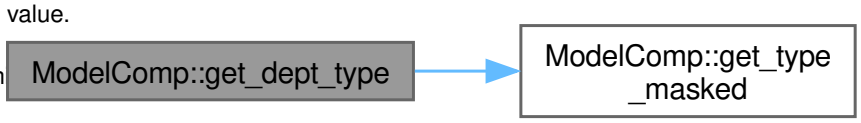
```
std::uint8_t ModelComp::get_dept_type ( ) const
```

Gets the department type value.

Returns

The department type value.

Here is the call graph for th



4.4.3.3 get_jobt_type()

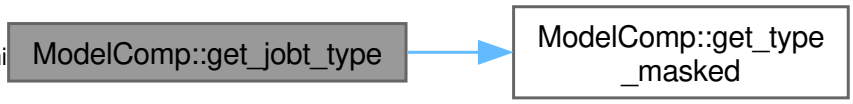
```
std::uint8_t ModelComp::get_jobt_type ( ) const
```

Gets the job type value.

Returns

The job type value.

Here is the call graph for thi



4.4.3.4 get_name_type()

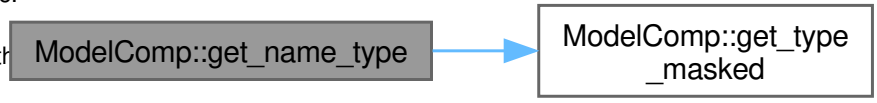
```
std::uint8_t ModelComp::get_name_type ( ) const
```

Gets the name type value.

Returns

The name type value.

Here is the call graph for th



4.4.3.5 get_type_masked()

```
std::uint8_t ModelComp::get_type_masked (
            std::uint8_t offset ) const
```

Gets the type masked value at the given offset.

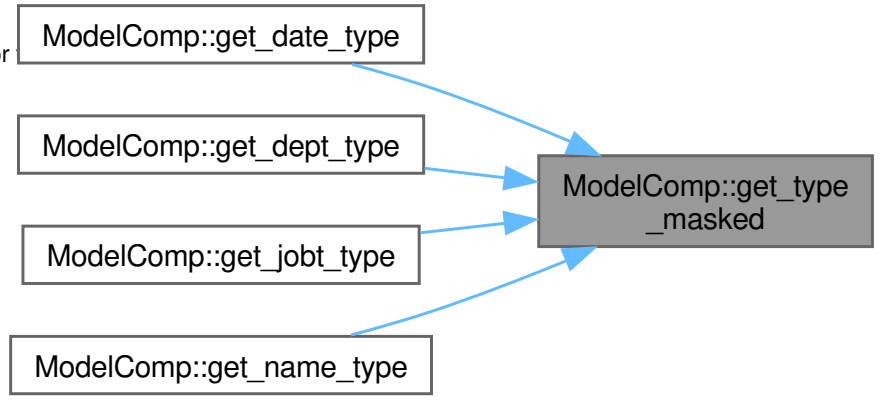
Parameters

offset	The offset from which to retrieve the value.

Returns

The type masked value.

Here is the caller graph for



4.4.3.6 operator"!()

```
ModelComp ModelComp::operator!  ( ) const
```

Overloaded logical NOT operator for ModelComp objects.

Returns

The logical NOT of the ModelComp object. The internal value of the ModelComp object.

4.4.3.7 set_date_type()

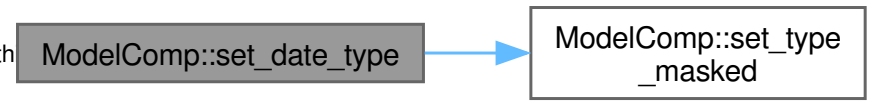
```
void ModelComp::set_date_type (
            std::uint8_t value )
```

Sets the date type value.

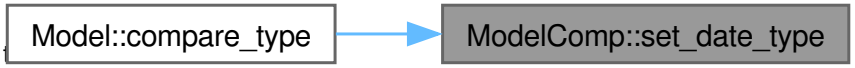
Parameters

value	The value to set.

Here is the call graph for th



Here is the caller graph for t



4.4.3.8 set_dept_type()

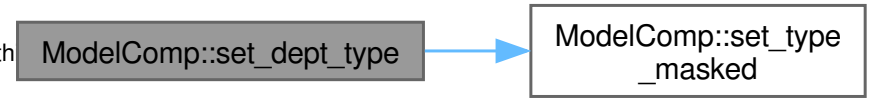
```
void ModelComp::set_dept_type (
            std::uint8_t value )
```

Sets the department type value.

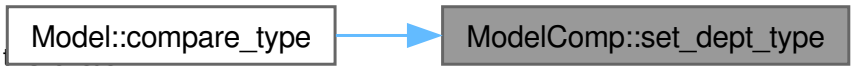
Parameters

value	The value to set.

Here is the call graph for th



Here is the caller graph for t



4.4.3.9 set_jobt_type()

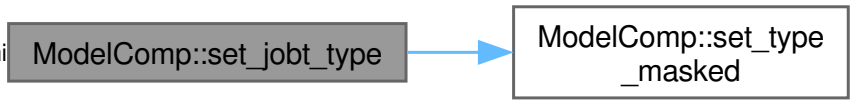
```
void ModelComp::set_jobt_type (
            std::uint8_t value )
```

Sets the job type value.

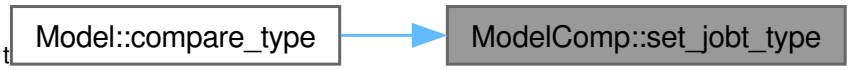
Parameters

value	The value to set.

Here is the call graph for thi



Here is the caller graph for t



4.4.3.10 set_name_type()

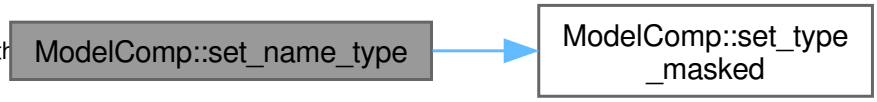
```
void ModelComp::set_name_type (
            std::uint8_t value )
```

Sets the name type value.

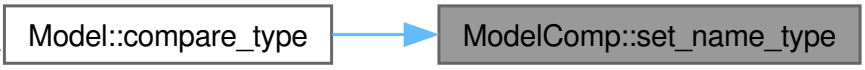
Parameters

value	The value to set.

Here is the call graph for th



Here is the caller graph for



4.4.3.11 set_type_masked()

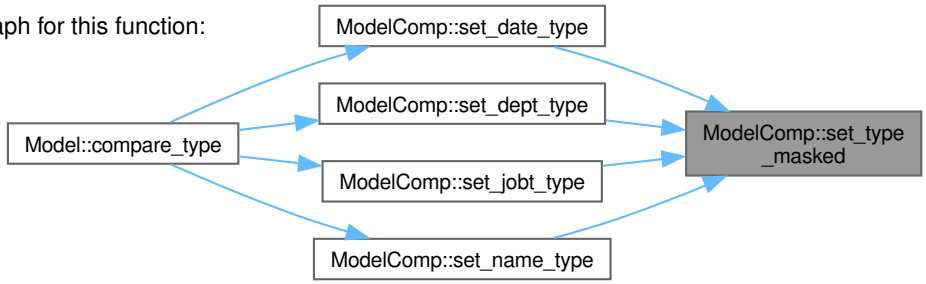
```
void ModelComp::set_type_masked (
            std::uint8_t value,
            std::uint8_t offset )
```

Sets the type masked value at the given offset.

Parameters

value	The value to set.
offset	The offset at which to set the value.

Here is the caller graph for this function:



4.4.4 Friends And Related Function Documentation

4.4.4.1 operator"!=

```
ModelComp operator!= (
            const ModelComp & l_bool,
            const ModelComp & r_bool )  [friend]
```

Overloaded inequality operator for ModelComp objects.

Parameters

l_bool	The left ModelComp object.
r_bool	The right ModelComp object.

Returns

The comparison result.

4.4.4.2 operator<<

```
std::ostream & operator<< (
            std::ostream & stream,
            const ModelComp & model )  [friend]
```

Overloaded stream insertion operator for ModelComp objects.

Parameters

stream	The output stream.
model	The ModelComp object to insert.

Returns

The modified output stream.

4.4.4.3 operator==

```
ModelComp operator== (
            const ModelComp & l_bool,
            const ModelComp & r_bool )  [friend]
```

Overloaded equality operator for ModelComp objects.

Parameters

l_bool	The left ModelComp object.
r_bool	The right ModelComp object.

Returns

The comparison result.

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

- src/model/model.hpp
- src/model/model.cpp

4.5 PseusoRandom Class Reference

Static Public Member Functions

- template<typename T , std::size_t LINE>
  static std::vector< T > generate_uniform_n (std::size_t n)

Generation of random line with length of n by using uniform distribution.

- template<typename T , std::size_t LINE>
  static std::vector< T > generate_normal_n (std::size_t n)

  Generation of random line with length of n by using normal distribution.
- template<typename T >
  static std::vector< T > uniform_distribution (T min, T max, std::size_t line, std::size_t n)

  uniform distribuion as an array
- template<typename T , std::size_t IRWIN_NUM = 12>
  static std::vector< T > normal_distribution (std::size_t n)

  normal distribuion as an array

Static Protected Attributes

- static const std::uint32_t **lce_a** = 4096
- static const std::uint32_t **lce_c** = 150889
- static const std::uint32_t **lce_m** = 714025

4.5.1 Member Function Documentation

4.5.1.1 generate_normal_n()

```
template<typename T , std::size_t LINE>
static std::vector< T > PseusoRandom::generate_normal_n (
            std::size_t n )  [inline], [static]
```

Generation of random line with length of n by using normal distribution.

Template Parameters

N	Length of generated line

Returns

Pseuorandom line with the lenght of N

4.5.1.2 generate_uniform_n()

```
template<typename T , std::size_t LINE>
static std::vector< T > PseusoRandom::generate_uniform_n (
            std::size_t n )  [inline], [static]
```

Generation of random line with length of n by using uniform distribution.

Template Parameters

N	Length of generated line

Returns

Pseuorandom line with the lenght of N

4.5.1.3 normal_distribution()

```
template<typename T , std::size_t IRWIN_NUM = 12>
static std::vector< T > PseusoRandom::normal_distribution (
            std::size_t n )  [inline], [static]
```

normal distribuion as an array

```
Normal distribution is being approximated by the Irwin-Hall distribution
```

Parameters

in	*mean*	Irwin-Hall mean
in	*sigma*	Irwin-Hall sigma

Template Parameters

T	Typename of min/max value
SIZE	Size of normalized uniform distribution
IRWIN_NUM	Irwin distribution numbers

Returns

{ description_of_the_return_value }

4.5.1.4 uniform_distribution()

```
template<typename T >
static std::vector< T > PseusoRandom::uniform_distribution (
            T min,
            T max,
            std::size_t line,
            std::size_t n )  [inline], [static]
```

uniform distribuion as an array

Parameters

in	*min*	Minimum distribution value
in	*max*	Maximum distribution value

Template Parameters

T	Typename of min/max value
SIZE	Size of normalized uniform distribution

Returns

std::array of uniformely distributed values in the range of [min, max]

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

- src/pseudo_random/pseudo_random.hpp

4.6 Search Class Reference

A class that provides static searching methods for sorting a vector of Model objects based on a specific field.

`#include <search.hpp>`

Static Public Member Functions

- template<typename T >
  static int binary_search (std::vector< Model > &model_vector, T search_value, std::uint8_t field)
  Performs binary search on a vector of Model objects based on a specific field.
- template<typename T >
  static int straight_search (std::vector< Model > &model_vector, T search_value, std::uint8_t field)
  Performs straight search on a vector of Model objects based on a specific field.

4.6.1 Detailed Description

A class that provides static searching methods for sorting a vector of Model objects based on a specific field.

Note

Currently provides implementations for binary search.

4.6.2 Member Function Documentation

4.6.2.1 binary_search()

```
template<typename T >
int Search::binary_search (
            std::vector< Model > & model_vector,
            T search_value,
            std::uint8_t field ) [static]
```

Performs binary search on a vector of Model objects based on a specific field.

Template Parameters

T	The type of the search value.

Parameters

model_vector	The vector of Model objects to search in.
search_value	The value to search for.
field	The field on which to perform the search.

Returns

The index of the found element, or -1 if not found.

4.6.2.2 straight_search()

```
template<typename T >
int Search::straight_search (
            std::vector< Model > & model_vector,
            T search_value,
            std::uint8_t field )  [static]
```

Performs straight search on a vector of Model objects based on a specific field.

Template Parameters

T	The type of the search value.

Parameters

model_vector	The vector of Model objects to search in.
search_value	The value to search for.
field	The field on which to perform the search.

Returns

The index of the found element, or -1 if not found.

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

- src/search/search.hpp

4.7 Sorting Class Reference

A class that provides static sorting methods for sorting a vector of Model objects based on a specific field.

`#include <sorting.hpp>`

Static Public Member Functions

- static void bubble_sort (std::vector< Model > &model_vector, uint8_t field)

  Sorts the given vector of Model objects using bubble sort algorithm.
- static void heap_sort (std::vector< Model > &model_vector, uint8_t field)

  Performs heap sort on a vector of Model objects.
- static void merge_sort (std::vector< Model > &model_vector, uint8_t field, std::size_t left=0, std::size_t right=0, bool initial=true)

  Sorts a vector of Model objects using merge sort algorithm.

4.7.1 Detailed Description

A class that provides static sorting methods for sorting a vector of Model objects based on a specific field.

Note

Currently provides implementations for bubble sort, heap sort, and merge sort.

4.7.2 Member Function Documentation

4.7.2.1 bubble_sort()

```
void Sorting::bubble_sort (
            std::vector< Model > & model_vector,
            uint8_t field ) [static]
```

Sorts the given vector of Model objects using bubble sort algorithm.

This function uses bubble sort algorithm to sort the given vector of Model objects based on the field specified by the `field` parameter. The objects are compared using the `compare_type()` method of the Model class.

Parameters

model_vector	The vector of Model objects to be sorted.
field	The index of the field to be used for sorting the objects.

Returns

void.

Note

This function modifies the original vector passed to it.

The `compare_type()` method of the Model class must return a Type object.

The Type object must have a method `get_type_masked()` that takes an offset and returns a boolean indicating whether the specified bit is set or not.

This function prints the number of iterations taken to sort the vector.

4.7.2.2 heap_sort()

```
void Sorting::heap_sort (
            std::vector< Model > & model_vector,
            uint8_t field )  [static]
```

Performs heap sort on a vector of Model objects.

This function sorts a vector of Model objects using the heap sort algorithm. The function uses the make_heap function to create a heap from the input vector, then sorts the heap by repeatedly extracting the maximum element from the heap and placing it at the end of the vector.

Parameters

model_vector	The vector of Model objects to be sorted.
field	The field of the Model object to sort by.

Returns

void.

4.7.2.3 merge_sort()

```
void Sorting::merge_sort (
            std::vector< Model > & model_vector,
            uint8_t field,
            std::size_t left = 0,
            std::size_t right = 0,
            bool initial = true )  [static]
```

Sorts a vector of Model objects using merge sort algorithm.

This function sorts a given vector of Model objects using merge sort algorithm. It takes the field to be sorted as input, along with left and right indices of the sub-vector to be sorted. If left and right indices are not provided, it sorts the entire vector by setting left and right indices accordingly.

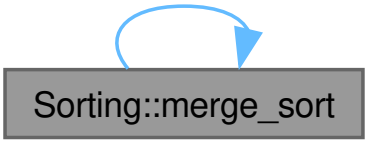
Parameters

model_vector	The vector of Model objects to be sorted.
field	The field to be sorted.
left	The left index of the sub-vector to be sorted (default is 0).
right	The right index of the sub-vector to be sorted (default is size-1).
initial	A boolean flag indicating whether this is the initial call to the function (default is true).

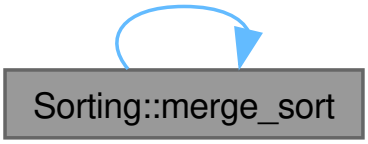
Returns

void.

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:

- src/sorting/sorting.hpp
- src/sorting/sorting.cpp

Chapter 5

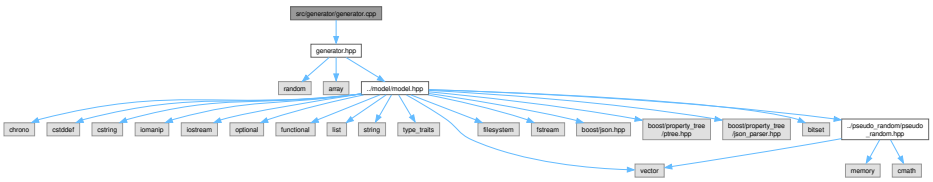
File Documentation

5.1 src/generator/generator.cpp File Reference

This source file holds implementation of Generator class.

```
#include "generator.hpp"
```

Include dependency graph for generator.cpp:



5.1.1 Detailed Description

This source file holds implementation of Generator class.

>

```
This calss implements model generator needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

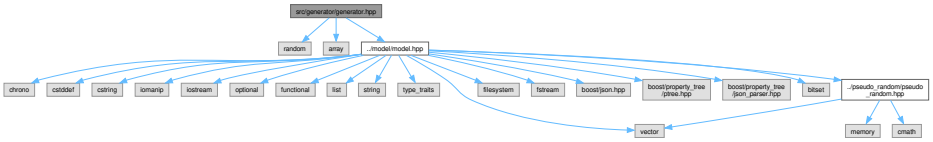
Copyright

5.2 src/generator/generator.hpp File Reference

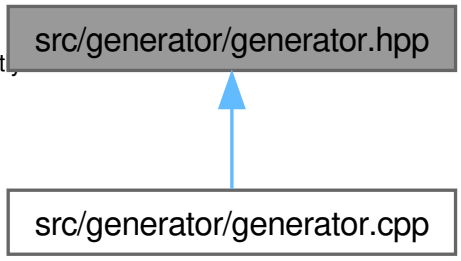
This header file holds implementation of Generator class.

```
#include <random>
#include <array>
#include "../model/model.hpp"
```

Include dependency graph for generator.hpp:



This graph shows which files directly or indirectly



Classes

- class Generator

  A class that generates models.

5.2.1 Detailed Description

This header file holds implementation of Generator class.

>

```
This calss implements model generator needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.3 generator.hpp

Go to the documentation of this file.

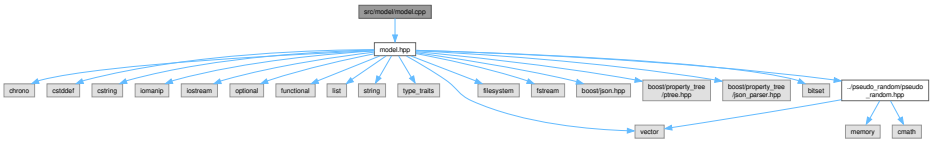
```
1
20 #ifndef GENERATOR_HPP
21 #define GENERATOR_HPP
22
23 #include <random>
24
25 #include <array>
26
27
28 #ifndef MODEL_HPP
29 #include "../model/model.hpp"
30 #endif // MODEL_HPP
31
42 class Generator
43 {
44 public:
52     Generator();
53
```

```
59     ~Generator();
60
70     Model model_generator();
71
72 private:
73     std::array<std::string, 2738> _first_name_list;
74     std::array<std::string, 1000> _last_name_list;
75     std::array<std::string, 449> _department_list;
76     std::array<std::string, 357> _job_title_list;
78     std::random_device       _random_device;
79     std::mt19937             _generator;
81     std::uniform_int_distribution<uint32_t> _first_name_distribution;
82     std::uniform_int_distribution<uint32_t> _last_name_distribution;
83     std::uniform_int_distribution<uint32_t> _department_distribution;
84     std::uniform_int_distribution<uint32_t> _job_title_distribution;
85     std::uniform_int_distribution<uint16_t> _year_distribution;
86     std::uniform_int_distribution<uint16_t> _month_distribution;
87     std::uniform_int_distribution<uint16_t> _sex_distribution;
88     std::uniform_int_distribution<uint16_t> _day_distribution;
89 };
90
91 #endif // GENERATOR_HPP
```

5.4 src/model/model.cpp File Reference

This source file holds implementation of Model class.

`#include "model.hpp"`
Include dependency graph for model.cpp:



Functions

- std::ostream & operator<< (std::ostream &stream, const Model &model)
- ModelComp operator< (const Model &l_model, const Model &r_model)
- ModelComp operator>= (const Model &l_model, const Model &r_model)
- ModelComp operator<= (const Model &l_model, const Model &r_model)
- ModelComp operator> (const Model &l_model, const Model &r_model)
- ModelComp operator== (const Model &l_model, const Model &r_model)
- ModelComp operator!= (const Model &l_model, const Model &r_model)
- ModelComp operator== (const ModelComp &l_bool, const ModelComp &r_bool)
- ModelComp operator!= (const ModelComp &l_bool, const ModelComp &r_bool)
- std::ostream & operator<< (std::ostream &stream, const ModelComp &r_bool)

5.4.1 Detailed Description

This source file holds implementation of Model class.

>

```
This calss implements model needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.4.2 Function Documentation

5.4.2.1 operator"!=() [1/2]

```
ModelComp operator!= (
            const Model & l_model,
            const Model & r_model )
```

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

5.4.2.2 operator"!=() [2/2]

```
ModelComp operator!= (
            const ModelComp & l_bool,
            const ModelComp & r_bool )
```

Parameters

l_bool	The left ModelComp object.
r_bool	The right ModelComp object.

Returns

The comparison result.

5.4.2.3 operator<()

```
ModelComp operator< (
            const Model & l_model,
            const Model & r_model )
```

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

5.4.2.4 operator<<() [1/2]

```
std::ostream & operator<< (
            std::ostream & stream,
            const Model & model )
```

Parameters

stream	The output stream.
model	The model to be output.

Returns

The output stream after printing the model.

5.4.2.5 operator<<() [2/2]

```
std::ostream & operator<< (
            std::ostream & stream,
            const ModelComp & r_bool )
```

Parameters

stream	The output stream.
model	The ModelComp object to insert.

Returns

The modified output stream.

5.4.2.6 operator<=()

```
ModelComp operator<= (
            const Model & l_model,
            const Model & r_model )
```

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

5.4.2.7 operator==() [1/2]

```
ModelComp operator== (
            const Model & l_model,
            const Model & r_model )
```

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

5.4.2.8 operator==() [2/2]

```
ModelComp operator== (
            const ModelComp & l_bool,
            const ModelComp & r_bool )
```

Parameters

l_bool	The left ModelComp object.
r_bool	The right ModelComp object.

Returns

The comparison result.

5.4.2.9 operator>()

```
ModelComp operator> (
            const Model & l_model,
            const Model & r_model )
```

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

The comparison result.

5.4.2.10 operator>=()

```
ModelComp operator>= (
            const Model & l_model,
            const Model & r_model )
```

Parameters

l_model	The left Model object.
r_model	The right Model object.

Returns

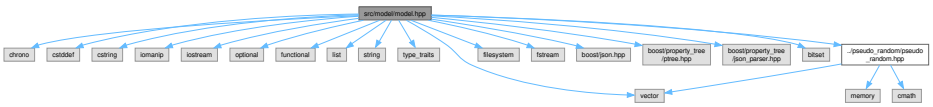
The comparison result.

5.5 src/model/model.hpp File Reference

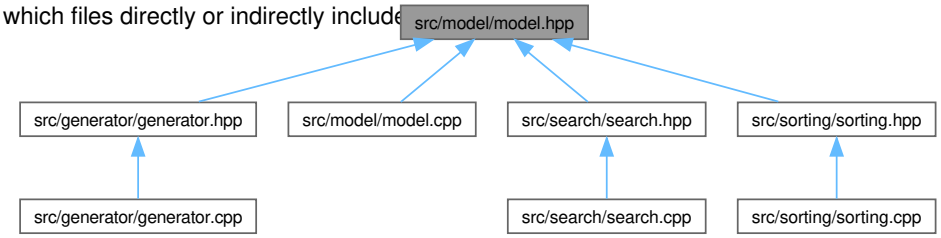
This header file holds implementation of Model class.

```
#include <chrono>
#include <cstddef>
#include <cstring>
#include <iomanip>
#include <iostream>
#include <optional>
#include <functional>
#include <list>
#include <string>
#include <type_traits>
#include <vector>
#include <filesystem>
#include <fstream>
#include <boost/json.hpp>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/json_parser.hpp>
#include <bitset>
#include "../pseudo_random/pseudo_random.hpp"
```

Include dependency graph for model.hpp:



This graph shows which files directly or indirectly include



Classes

- class Hashing

  Contains static methods for hashing and related operations.
- class Model

  Represents an employee model.
- class ModelComp

  Represents the comparison result of two models.

5.5.1 Detailed Description

This header file holds implementation of Model class.

>

```
This calss implements model needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.6 model.hpp

Go to the documentation of this file.

```
1
20 #ifndef MODEL_HPP
21 #define MODEL_HPP
22
23
24 #include <chrono>
25 #include <cstddef>
26 #include <cstring> // strcmp has better performance
27 #include <iomanip>
28 #include <iostream>
29 #include <optional>
30 #include <functional>
31 #include <list>
32 #include <optional>
33 #include <string>
34 #include <type_traits>
35 #include <vector>
36
37
38 #include <filesystem>
39 #include <fstream>
40 #include <boost/json.hpp>
41 #include <boost/property_tree/ptree.hpp>
42 #include <boost/property_tree/json_parser.hpp>
43
44 #include <bitset>
45
46 #include "../pseudo_random/pseudo_random.hpp"
47
48 class Model;
49
50
55 class Hashing
56 {
57 public:
65     static std::uint32_t basic_hashing_function(const std::string& value);
66
74     static std::uint32_t djb2_hashing_function(const std::string& value);
75
83     static std::uint32_t advanced_hashing_function(const std::string& value);
84
92     static std::uint32_t count_collisions(const std::vector<std::list<Model>& hash_table);
93
102     static std::vector<std::list<Model> hash_model(std::vector<Model>& model_vector,
      std::function<std::size_t(const std::string& value)> hash_function);
103
113     static std::optional<Model> find_in_hash_table(const std::vector<std::list<Model>& hash_table,
      std::uint32_t hash, std::size_t size);
114 };
115
116 class ModelComp;
117
125 class Model
126 {
127 public:
139     Model(std::string full_name, std::string department, std::string job_title,
      std::chrono::year_month_day employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field =
      255, const std::optional<std::function<std::size_t(const std::string& value)>& optional_func =
      std::nullopt);
140
152     Model(std::string full_name, std::string department, std::string job_title, std::string
      employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field = 255, const
      std::optional<std::function<std::size_t(const std::string& value)>& optional_func = std::nullopt);
153
159     Model(std::uint8_t decor_type);
160
166     Model(const Model& other) {
167         try
168         {
169             this->_full_name = other._full_name;
170             this->_department = other._department;
171             this->_job_title = other._job_title;
172             this->_employment_date = other._employment_date;
173             this->_decor_type = other._decor_type;
174             this->_hash_field = other._hash_field;
175             this->_model_hash = other._model_hash;
176         }
177         catch (const std::exception& e)
178         {
179             std::cout << e.what() << std::endl;
180         }
181     }
```

```
182
186     ~Model();
187
199     void set_model(std::string full_name, std::string department, std::string job_title,
        std::chrono::year_month_day employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field =
        255, const std::optional<std::function<std::size_t(const std::string& value)»& optional_func =
        std::nullopt);
200
212     void set_model(std::string full_name, std::string department, std::string job_title, std::string
        employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field = 255, const
        std::optional<std::function<std::size_t(const std::string& value)»& optional_func = std::nullopt);
213
219     void set_decor(std::uint8_t decor_type);
220
226     void set_hash_func(std::function<std::size_t(const std::string& value)> hash_fucntion);
227
233     void set_hash(std::uint32_t model_hash);
234
240     void set_hash_field(std::uint8_t hash_field);
241
250     ModelComp compare_type(const Model& r_model, std::uint8_t mode);
251
261     template<typename T>
262         ModelComp compare_type(std::uint8_t mode, T r_value);
263
272     template<typename T>
273         T get_field(std::uint8_t field) const;
274
283     template<typename T>
284         void set_field(std::uint8_t field, T value);
285
291     std::uint32_t get_hash() const;
292
298     std::uint8_t get_hash_field() const;
299
307     template<typename T>
308         T get_hash_field() const;
309
316     static void save_model(const std::vector<Model>& model_vector, std::filesystem::path file_path);
317
324     static void load_model(std::vector<Model>& model_vector, std::filesystem::path file_path);
325
331     static void print_model(const std::vector<Model>& model_vector);
332
341     friend std::ostream& operator« (std::ostream& stream, const Model& model);
342
351     friend ModelComp     operator<  (const Model& l_model, const Model& r_model);
352
361     friend ModelComp     operator>  (const Model& l_model, const Model& r_model);
362
371     friend ModelComp     operator<= (const Model& l_model, const Model& r_model);
372
381     friend ModelComp     operator>= (const Model& l_model, const Model& r_model);
382
391     friend ModelComp     operator== (const Model& l_model, const Model& r_model);
392
401     friend ModelComp     operator!= (const Model& l_model, const Model& r_model);
402
403 private:
404     std::string                    _full_name;
405     std::string                    _department;
406     std::string                    _job_title;
407     std::chrono::year_month_day    _employment_date;
408
409     std::uint8_t                   _decor_type;
410
411     //HASHING
412     std::uint8_t                   _hash_field;
413
414     std::uint32_t                  _model_hash;
415 };
416
421 class ModelComp
422 {
423 public:
427     ModelComp();
428
434     ModelComp(std::uint8_t value);
435
439     ~ModelComp();
440
447     void set_type_masked(std::uint8_t value, std::uint8_t offset);
448
454     void set_name_type(std::uint8_t value);
455
461     void set_dept_type(std::uint8_t value);
462
```

```
468     void set_jobt_type(std::uint8_t value);
469
475     void set_date_type(std::uint8_t value);
476
484     std::uint8_t get_type_masked(std::uint8_t offset) const;
485
491     std::uint8_t get_name_type() const;
492
498     std::uint8_t get_dept_type() const;
499
505     std::uint8_t get_jobt_type() const;
506
512     std::uint8_t get_date_type() const;
513
522     friend ModelComp operator== (const ModelComp& l_bool, const ModelComp& r_bool);
523
532     friend ModelComp operator!= (const ModelComp& l_bool, const ModelComp& r_bool);
533
542     friend std::ostream& operator« (std::ostream& stream, const ModelComp& model);
543
549     ModelComp operator! () const;
550
551 private:
552     std::uint8_t _value;
553 };
554
555
556 template<typename T>
557 ModelComp Model::compare_type(std::uint8_t mode, T r_value)
558 {
559     int8_t comp_result = 4;
560     ModelComp model_comp;
561
562     if (mode > 3)
563     {
564         mode = 0;
565     }
566
567     switch (mode)
568     {
569     case 0:
570         {
571             if (std::is_same<T, decltype(this->_full_name)>::value)
572             {
573                 comp_result = this->_full_name.compare(r_value);
574                 if (comp_result < 0)
575                 {
576                     comp_result = 1;
577                 }
578                 else if(comp_result > 0)
579                 {
580                     comp_result = 2;
581                 }
582                 else
583                 {
584                     comp_result = 0;
585                 }
586             }
587             else
588             {
589                 throw std::invalid_argument("r_value should be _full_name");
590             }
591             model_comp.set_name_type(comp_result);
592             break;
593         }
594     case 1:
595         {
596             if (std::is_same<T, decltype(this->_department)>::value)
597             {
598                 comp_result = this->_department.compare(r_value);
599                 if (comp_result < 0)
600                 {
601                     comp_result = 1;
602                 }
603                 else if(comp_result > 0)
604                 {
605                     comp_result = 2;
606                 }
607                 else
608                 {
609                     comp_result = 0;
610                 }
611             }
612             else
613             {
614                 throw std::invalid_argument("r_value should be _department");
615             }
```

```
616             model_comp.set_dept_type(comp_result);
617             break;
618         }
619     case 2:
620         {
621             if (std::is_same<T, decltype(this->_job_title)>::value)
622             {
623                 comp_result = this->_job_title.compare(r_value);
624                 if (comp_result < 0)
625                 {
626                     comp_result = 1;
627                 }
628                 else if(comp_result > 0)
629                 {
630                     comp_result = 2;
631                 }
632                 else
633                 {
634                     comp_result = 0;
635                 }
636             }
637             else
638             {
639                 throw std::invalid_argument("r_value should be _job_title");
640             }
641             model_comp.set_jobt_type(comp_result);
642             break;
643         }
644     case 3:
645         {
646 
647             throw std::invalid_argument("r_value should be _employment_date");
648             model_comp.set_date_type(comp_result);
649             break;
650         }
651     default:
652         {
653             break;
654         }
655     }
656 
657     return model_comp;
658 }
659 
660 template<typename T>
661 T Model::get_field(std::uint8_t field) const
662 {
663     switch (field)
664     {
665 
666         case 1:
667         {   if (std::is_same<T, decltype(this->_department)>::value)
668             {
669                 return this->_department;
670             }
671             else
672             {
673                 throw std::invalid_argument("T should be _department");
674             }
675             break;
676         }
677 
678         case 2:
679         {   if (std::is_same<T, decltype(this->_job_title)>::value)
680             {
681                 return this->_job_title;
682             }
683             else
684             {
685                 throw std::invalid_argument("T should be _job_title");
686             }
687             break;
688         }
689 
690         case 3:
691         {
692             throw std::invalid_argument("T should be _employment_date");
693             break;
694         }
695 
696         default:
697         {   if (std::is_same<T, decltype(this->_full_name)>::value)
698             {
699                 return this->_full_name;
700             }
701             else
702             {
```

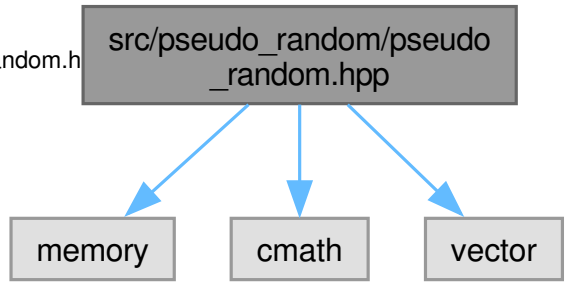
```
703                 throw std::invalid_argument("T should be _full_name");
704             }
705             break;
706         }
707 
708     }
709 }
710 
711 
712 template<typename T>
713 void Model::set_field(std::uint8_t field, T value)
714 {
715     switch (field)
716     {
717 
718         case 1:
719         {   if (std::is_same<T, decltype(this->_department)>::value)
720             {
721                 this->_department = value;
722             }
723             else
724             {
725                 throw std::invalid_argument("T should be _department");
726             }
727             break;
728         }
729 
730         case 2:
731         {   if (std::is_same<T, decltype(this->_job_title)>::value)
732             {
733                 this->_job_title = value;
734             }
735             else
736             {
737                 throw std::invalid_argument("T should be _job_title");
738             }
739             break;
740         }
741 
742         case 3:
743         {
744             throw std::invalid_argument("T should be _employment_date");
745             break;
746         }
747 
748         default:
749         {   if (std::is_same<T, decltype(this->_full_name)>::value)
750             {
751                 this->_full_name = value;
752             }
753             else
754             {
755                 throw std::invalid_argument("T should be _full_name");
756             }
757             break;
758         }
759 
760     }
761 }
762 
763 template<typename T>
764 T Model::get_hash_field() const
765 {
766     return this->get_field<T>(this->_hash_field);
767 }
768 
769 #endif // MODEL_HPP
```

5.7 src/pseudo_random/pseudo_random.hpp File Reference

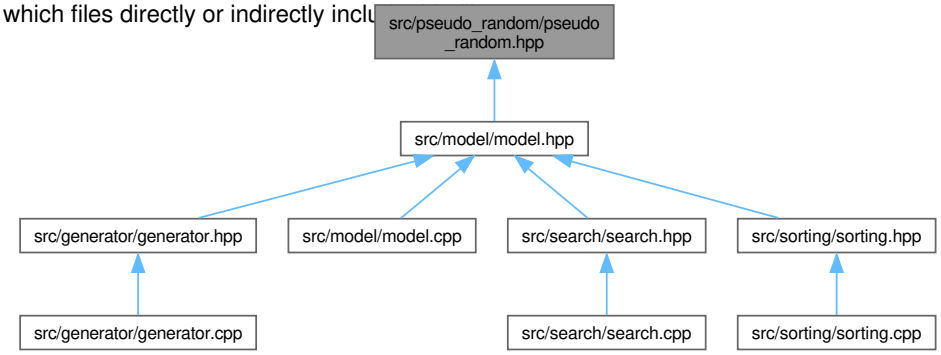
This source file holds implementation of PseudoRandomGenerator class.

```
#include <memory>
#include <cmath>
#include <vector>
```

Include dependency graph for pseudo_random.h



This graph shows which files directly or indirectly inclu



Classes

- class PseusoRandom

5.7.1 Detailed Description

This source file holds implementation of PseudoRandomGenerator class.

This header file holds implementation of PseudoRandomGenerator class.

>

```
This calss implements search algorithms needed for successfull completion of laboratory work 4.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

>

```
This calss implements search algorithms needed for successfull completion of laboratory work 4.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.8 pseudo_random.hpp

Go to the documentation of this file.

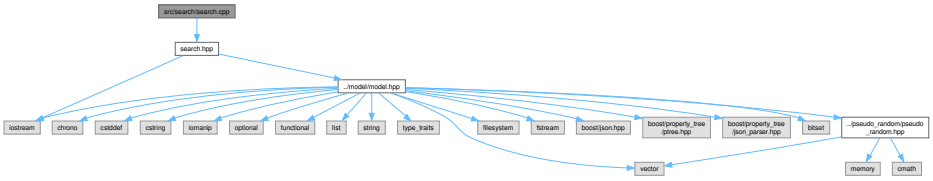
```
1
20 #ifndef PSEUDO_RANDOM_HPP
21 #define PSEUDO_RANDOM_HPP
22
23 #include <memory>
24 #include <cmath>
25 #include <vector>
26
27
28 class PseusoRandom
29 {
30 public:
31
40     template<typename T, std::size_t LINE>
41         static std::vector<T> generate_uniform_n(std::size_t n)
42         {
43             std::vector<T> key;
44             key.resize(n, 0);
45
46             std::vector<T> distribution = uniform_distribution<int>(0, 2147483647, LINE, n);
47
48             for (std::size_t i = 0; i < n; ++i)
49             {
50                 key[i] = distribution.at(i);
51             }
52
53             return key;
54         }
55
64     template<typename T, std::size_t LINE>
65         static std::vector<T> generate_normal_n(std::size_t n)
66         {
67             std::vector<T> key;
68             key.resize(n, 0);
69
70             std::vector<T> distribution = normal_distribution<int>(n);
71
72             for (std::size_t i = 0; i < n; ++i)
73             {
74                 key[i] = distribution.at(i);
75             }
76
77             return key;
78         }
79
80
92     template <typename T>
93         static std::vector<T> uniform_distribution(T min, T max, std::size_t line, std::size_t n)
94         {
95             std::vector<T> distribution;
96             distribution.resize(n, 0);
97
98             auto previous = int_seed(line);
99             for (auto &element : distribution)
100             {
101                 element = static_cast<T>(uniform_distribution_n(previous) * (max - min) + min);
102             }
103
104             return distribution;
105         }
106
121     template <typename T, std::size_t IRWIN_NUM = 12>
122         static std::vector<T> normal_distribution(std::size_t n)
123         {
124             std::vector<T> distribution;
125             distribution.resize(n, 0);
126
127             auto previous = int_seed(0);
128             for (auto &element : distribution)
129             {
130                 double value = 0;
131                 for (std::size_t i = 0; i < IRWIN_NUM; ++i)
132                 {
133                     value += uniform_distribution_n(previous);
134                 }
135
136                 element = value / std::sqrt(IRWIN_NUM / 12.0f) - IRWIN_NUM / 2.0f;
137             }
138
139             return distribution;
140         }
141
```

```
142 private:
151     static unsigned int_ctoi(const char* str, int offset)
152     {
153         return static_cast<std::uint32_t>(str[offset] - '0') * 10 +
      static_cast<std::uint32_t>(str[offset + 1] - '0');
154     }
155
161     static unsigned int_seed(std::size_t line)
162     {
163         auto compile_time = __TIME__;
164         auto compile_date = __DATE__;
165         return int_ctoi(compile_time, 0) * 60 * 60 + int_ctoi(compile_time, 3) * 60 +
      int_ctoi(compile_time, 6) + int_ctoi(compile_date, 5) * 24 * 60 * 60 - line;
166     }
167
177     static std::uint32_t uniform_distribution(std::uint32_t &previous)
178     {
179         previous = ((lce_a * previous + lce_c) % lce_m);
180         return previous;
181     }
182
192     static double uniform_distribution_n(std::uint32_t &previous)
193     {
194         auto distribution = uniform_distribution(previous);
195         return static_cast<double>(distribution) / lce_m;
196     }
197
198 protected:
199     static const std::uint32_t lce_a = 4096;
200     static const std::uint32_t lce_c = 150889;
201     static const std::uint32_t lce_m = 714025;
202 };
203
204 #endif // PSEUDO_RANDOM_HPP
```

5.9 src/search/search.cpp File Reference

This source file holds implementation of Search class.

`#include "search.hpp"`
Include dependency graph for search.cpp:



5.9.1 Detailed Description

This source file holds implementation of Search class.

>

```
This calss implements search algorithms needed for successfull completion of laboratory work 2.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
        (Not really)
```

5.10 search.hpp

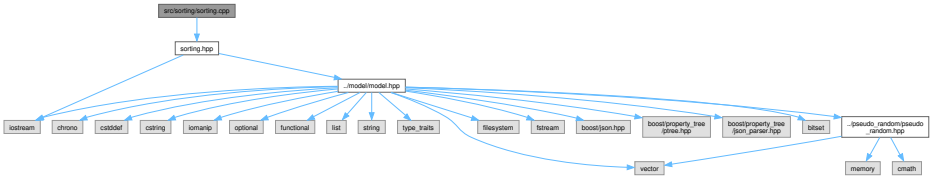
```
1
20 #ifndef SEARCH_HPP
21 #define SEARCH_HPP
22
23 #ifndef MODEL_HPP
24 #include "../model/model.hpp"
25 #endif // MODEL_HPP
26
27 #include <iostream>
28
36 class Search
37 {
38 public:
49     template<typename T>
50         static int binary_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field);
51
62     template<typename T>
63         static int straight_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field);

64 };
65
66
67 template<typename T>
68 int Search::binary_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field)
69 {
70     int dummy = 0;
71     int left = 0;
72     int right = model_vector.size() - 1;
73     uint8_t offset = field * 2;
74
75     while (left <= right)
76     {
77         int mid = (left + right) / 2;
78
79         if (((int)model_vector.at(mid).compare_type<T>(field, search_value).get_type_masked(offset) ==
    0))
80         {
81             return mid;
82         }
83
84         else if (((int)model_vector.at(mid).compare_type<T>(field, search_value).get_type_masked(offset)
    == 1))
85         {
86             left = mid + 1;
```

```
87          }
88
89          else if (((int)model_vector.at(mid).compare_type<T>(field, search_value).get_type_masked(offset)
        == 2))
90          {
91              right = mid - 1;
92          }
93      }
94      return -1;
95  }
96
97  template<typename T>
98  int Search::straight_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field)
99  {
100      uint8_t offset = field * 2;
101
102      for (std::size_t index = 0; index < model_vector.size(); ++index)
103      {
104          if (((int)model_vector.at(index).compare_type<T>(field, search_value).get_type_masked(offset) ==
        0))
105          {
106              return index;
107          }
108      }
109      return -1;
110  }
111
112 #endif // SEARCH_HPP
```

5.11 src/sorting/sorting.cpp File Reference

This source file holds implementation of Sorting class.

`#include "sorting.hpp"`
Include dependency graph for sorting.cpp:



5.11.1 Detailed Description

This source file holds implementation of Sorting class.

>

```
This calss implements sorting algorithms needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

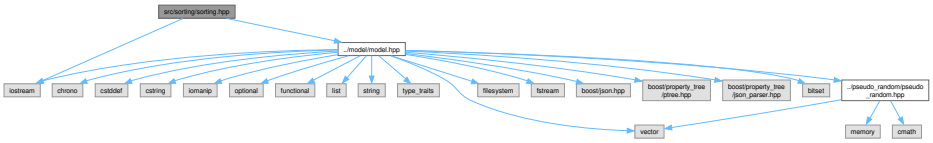
Copyright

5.12 src/sorting/sorting.hpp File Reference

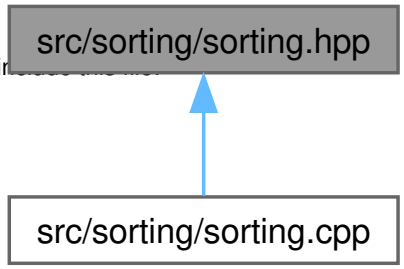
This header file holds implementation of Search class.

```
#include "../model/model.hpp"
#include <iostream>
```

Include dependency graph for sorting.hpp:



This graph shows which files directly or indirectly in



Classes

- class Sorting

  A class that provides static sorting methods for sorting a vector of Model objects based on a specific field.

5.12.1 Detailed Description

This header file holds implementation of Search class.

This header file holds implementation of Sorting class.

>

```
This calss implements search algorithms needed for successfull completion of laboratory work 2.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

>

```
This calss implements sorting algorithms needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.13 sorting.hpp

Go to the documentation of this file.

```
1
20 #ifndef SORTING_HPP
21 #define SORTING_HPP
22
23 #ifndef MODEL_HPP
24 #include "../model/model.hpp"
25 #endif // MODEL_HPP
26
27 #include <iostream>
28
36 class Sorting
37 {
38 public:
39     static void bubble_sort(std::vector<Model>& model_vector, uint8_t field);
40     static void heap_sort(std::vector<Model>& model_vector, uint8_t field);
41     static void merge_sort(std::vector<Model>& model_vector, uint8_t field, std::size_t left = 0,
      std::size_t right = 0, bool initial = true);
42
43 private:
44     static void make_heap(std::vector<Model>& model_vector, std::size_t index, uint8_t field, std::size_t
      last_index = 0);
45     static void make_merge(std::vector<Model>& model_vector, uint8_t field, std::size_t left, std::size_t
      right, std::size_t middle);
46 };
47
48 #endif // SORTING_HPP
```

Chapter 6

PRNG statistics

6.0.1 Definition for PRNG

FULL NAME	DEPARTMENT	JOB TITLE	DATE
MODE 0	**MODE 1**	**MODE 2**	**MODE 3**
Alexzander Oliver Baxterovna	Data Entry	Architectural Technologist	2015/09/09
...	...	...	...
Billy Barrett Okeogheneovich	Audio Engineering	Production Manager	1999/01/15

Table 6.1 Employee information

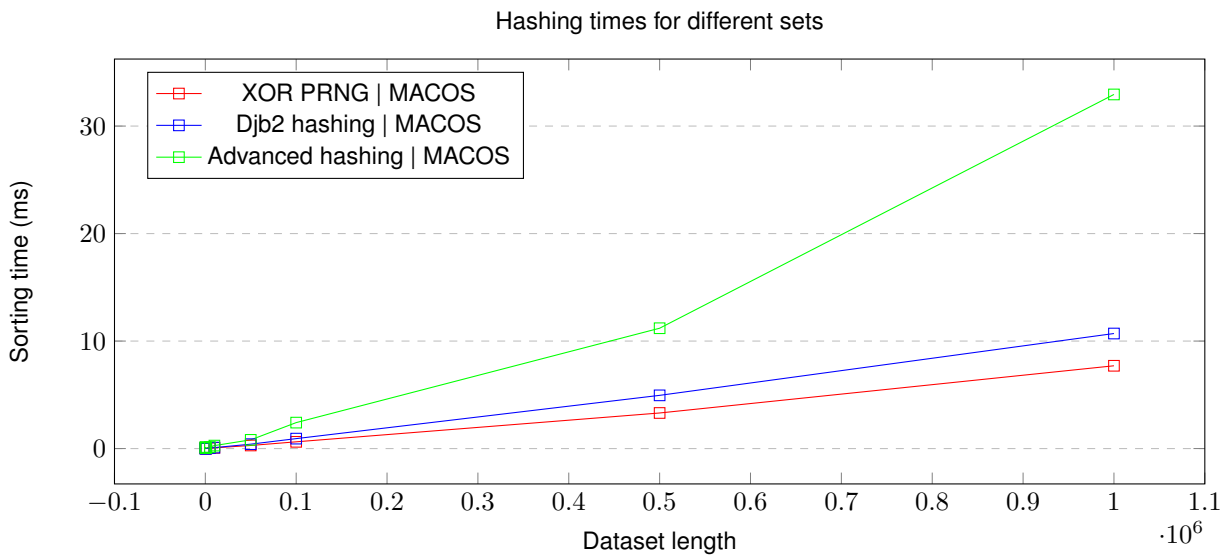
Table 6.2 MACOS specs

Model	MacBook Air (11-inch, Early 2015)
OS	MacOS Montery 12.6
Processor	1.6 GHz 2-core Intel Core i5
Memory	4 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 6000 1536 MB

Table 6.3 WIN specs

Model	-
OS	Windows 10 Pro 20H2
Processor	Intel Core i7-8086K @ 4.50 GHz
Memory	80,0 GB 2333 MHz DDR4
Graphics	NVIDIA GeForce GTX 1080

6.0.2 Hashing comparison



Index