

Automatiser dit studievalg

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	database Struct Reference . . . . .	5
3.1.1	Member Data Documentation . . . . .	5
3.1.1.1	amount_of_interests . . . . .	5
3.1.1.2	educations . . . . .	5
3.1.1.3	interest_string . . . . .	6
3.2	Database Struct Reference . . . . .	6
3.2.1	Detailed Description . . . . .	6
3.3	education Struct Reference . . . . .	6
3.3.1	Detailed Description . . . . .	7
3.4	location Struct Reference . . . . .	7
3.5	profile Struct Reference . . . . .	7
3.5.1	Detailed Description . . . . .	8
3.6	qualification Struct Reference . . . . .	8
3.6.1	Member Data Documentation . . . . .	8
3.6.1.1	subjects . . . . .	8
3.7	subject Struct Reference . . . . .	8
3.7.1	Member Data Documentation . . . . .	8
3.7.1.1	level . . . . .	9
3.8	vector Struct Reference . . . . .	9

<b>4</b>	<b>File Documentation</b>	<b>11</b>
4.1	commands.h File Reference . . . . .	11
4.2	constants.h File Reference . . . . .	11
4.2.1	Detailed Description . . . . .	12
4.3	database.h File Reference . . . . .	12
4.3.1	Detailed Description . . . . .	12
4.3.2	Function Documentation . . . . .	12
4.3.2.1	findEducation() . . . . .	12
4.4	education.h File Reference . . . . .	13
4.4.1	Detailed Description . . . . .	13
4.4.2	Function Documentation . . . . .	13
4.4.2.1	createArrayOfEducations() . . . . .	13
4.4.2.2	createDefaultEducation() . . . . .	14
4.5	parser.h File Reference . . . . .	14
4.5.1	Detailed Description . . . . .	15
4.5.2	Function Documentation . . . . .	15
4.5.2.1	createArrayOfStrings() . . . . .	15
4.5.2.2	findDatabaseLine() . . . . .	15
4.5.2.3	parseDatabase() . . . . .	16
4.5.2.4	parseDatabaseLine() . . . . .	16
4.5.2.5	parseEduDesc() . . . . .	16
4.5.2.6	parseEduNames() . . . . .	17
4.5.2.7	parseEduRegion() . . . . .	17
4.5.2.8	parseEduString() . . . . .	18
4.5.2.9	parseInterestNames() . . . . .	18
4.5.2.10	parseInterestValues() . . . . .	18
4.5.2.11	parseNumOfEdu() . . . . .	19
4.5.2.12	parseNumOfInterests() . . . . .	19
4.5.2.13	parseReqGrade() . . . . .	19
4.5.2.14	parseSubReq() . . . . .	19

4.5.2.15	<a href="#">readReqString()</a>	20
4.5.2.16	<a href="#">strToReg()</a>	20
4.6	<a href="#">profile.h File Reference</a>	20
4.6.1	<a href="#">Detailed Description</a>	21
4.6.2	<a href="#">Function Documentation</a>	21
4.6.2.1	<a href="#">createProfile()</a>	21
4.6.2.2	<a href="#">freeProfile()</a>	21
4.6.2.3	<a href="#">printProfile()</a>	22
4.7	<a href="#">region.h File Reference</a>	22
4.7.1	<a href="#">Detailed Description</a>	22
4.7.2	<a href="#">Enumeration Type Documentation</a>	22
4.7.2.1	<a href="#">region</a>	23
4.8	<a href="#">serialize.h File Reference</a>	23
4.8.1	<a href="#">Detailed Description</a>	23
4.9	<a href="#">subjects.h File Reference</a>	24
4.9.1	<a href="#">Detailed Description</a>	24
4.9.2	<a href="#">Function Documentation</a>	24
4.9.2.1	<a href="#">charToLevel()</a>	24
4.9.2.2	<a href="#">levelToChar()</a>	25
4.9.2.3	<a href="#">stringToClass()</a>	25
4.10	<a href="#">vector.h File Reference</a>	25
4.10.1	<a href="#">Detailed Description</a>	26
4.10.2	<a href="#">Function Documentation</a>	26
4.10.2.1	<a href="#">addVector()</a>	26
4.10.2.2	<a href="#">copyVector()</a>	27
4.10.2.3	<a href="#">createVector()</a>	27
4.10.2.4	<a href="#">dotProduct()</a>	27
4.10.2.5	<a href="#">freeVector()</a>	27
4.10.2.6	<a href="#">freeVectorM()</a>	28
4.10.2.7	<a href="#">lengthOfVector()</a>	28
4.10.2.8	<a href="#">normalizeVector()</a>	28
4.10.2.9	<a href="#">printVector()</a>	29
4.10.2.10	<a href="#">scaleVector()</a>	29
4.10.2.11	<a href="#">subtractVector()</a>	29



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">database</a>	.....	5
<a href="#">Database</a>		
	A structure to store a database .....	6
<a href="#">education</a>		
	Describes an education and all it requirements .....	6
<a href="#">location</a>	.....	7
<a href="#">profile</a>		
	Describes a user .....	7
<a href="#">qualification</a>	.....	8
<a href="#">subject</a>	.....	8
<a href="#">vector</a>	.....	9





## Chapter 2

# File Index

### 2.1 File List

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

<a href="#">commands.h</a>	Contains functions related to command handling . . . . .	11
<a href="#">constants.h</a>	Contains symbolic constants used throughout the program . . . . .	11
<a href="#">database.h</a>	Contains elements relating to the database . . . . .	12
<a href="#">education.h</a>	Contains elements relating to educations . . . . .	13
<a href="#">parser.h</a>	Contains elements relating to parsing the database . . . . .	14
<a href="#">profile.h</a>	Contains elements relating to profile . . . . .	20
<a href="#">region.h</a>	Contains geographical elements . . . . .	22
<a href="#">serialize.h</a>	Save and load profile data . . . . .	23
<a href="#">subjects.h</a>	Contains code regarding subjects and qualifications for educations . . . . .	24
<a href="#">vector.h</a>	Contains elements relating to vectors . . . . .	25



## Chapter 3

# Class Documentation

### 3.1 database Struct Reference

Collaboration diagram for database:

#### Public Attributes

- int **amount\_of\_educations**
- struct [education](#) \* [educations](#)
- int [amount\\_of\\_interests](#)
- char \*\* [interest\\_string](#)

#### 3.1.1 Member Data Documentation

##### 3.1.1.1 amount\_of\_interests

```
int database::amount_of_interests
```

an array of educations delimited by amount\_of\_educations

##### 3.1.1.2 educations

```
struct education* database::educations
```

the amount of educations in the database

### 3.1.1.3 interest\_string

```
char** database::interest_string
```

the amount of interests in the database

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

- [database.h](#)

## 3.2 Database Struct Reference

A structure to store a database.

```
#include <database.h>
```

### 3.2.1 Detailed Description

A structure to store a database.

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

- [database.h](#)

## 3.3 education Struct Reference

Describes an education and all it requirements.

```
#include <education.h>
```

Collaboration diagram for education:

### Public Attributes

- char \* **name**
- char \* **description**
- char \* **link**
- enum [region](#) **region**
- double **required\_grade**
- struct [vector](#) **interests**
- struct [qualification](#) **required\_qualifications**

### 3.3.1 Detailed Description

Describes an education and all it requirements.

A structure, which contains amount\_of\_educations educations.

This structure defines an education and all the details about the education.

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

- [education.h](#)

## 3.4 location Struct Reference

### Public Attributes

- enum [region](#) **region**
- double **region\_importance**

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

- [region.h](#)

## 3.5 profile Struct Reference

Describes a user.

```
#include <profile.h>
```

Collaboration diagram for profile:

### Public Attributes

- struct [vector](#) **interests**
- struct [vector](#) **adjustment\_vector**
- char **name** [MAX\_NAME\_LENGTH]
- struct [qualification](#) **qualifications**
- double **average**
- struct [location](#) **location**
- char **saved\_educations** [EDUCATION\_LIST\_LENGTH][MAX\_EDU\_NAME\_LENGTH]
- int **last\_recommended**
- char **recommended\_educations** [EDUCATION\_LIST\_LENGTH][MAX\_EDU\_NAME\_LENGTH]

### 3.5.1 Detailed Description

Describes a user.

This structure defines the profile of a user and all the details about the user

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

- [profile.h](#)

## 3.6 qualification Struct Reference

Collaboration diagram for qualification:

### Public Attributes

- int **amount\_of\_subjects**
- struct [subject](#) \* [subjects](#)

### 3.6.1 Member Data Documentation

#### 3.6.1.1 subjects

```
struct subject* qualification::subjects
```

the amount of subjects in qualifications

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

- [subjects.h](#)

## 3.7 subject Struct Reference

### Public Attributes

- enum level [level](#)

### 3.7.1 Member Data Documentation

#### 3.7.1.1 level

```
enum level subject::level
```

the name of the subject

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

- [subjects.h](#)

## 3.8 vector Struct Reference

### Public Attributes

- double \* **array**
- int **size**

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

- [vector.h](#)





## Chapter 4

# File Documentation

### 4.1 commands.h File Reference

Contains functions related to command handling.

```
#include "profile.h"
#include "education.h"
#include "subjects.h"
#include "vector.h"
#include "database.h"
Include dependency graph for commands.h:
```

### 4.2 constants.h File Reference

Contains symbolic constants used throughout the program.

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

#### Macros

- `#define VERSION "1.0.1"`
- `#define NUMBER_OF_REGIONS 5`
- `#define IMPORTANT_SUBJECTS 5`
- `#define OTHER_SUBJECTS 11`
- `#define LANGUAGE_SUBJECTS 11`
- `#define USELESS_SUBJECTS 2`
- `#define TOTAL_SUBJECTS (IMPORTANT_SUBJECTS + OTHER_SUBJECTS + LANGUAGE_SUBJECTS + USELESS_SUBJECTS)`
- `#define MAX_NAME_LENGTH 20`
- `#define MAX_FILE_NAME_LENGTH MAX_NAME_LENGTH + 12`
- `#define EDUCATION_LIST_LENGTH 10`
- `#define MAX_EDU_NAME_LENGTH 40`
- `#define MAX_COMMAND_LENGTH 10`
- `#define MAX_INPUT_LENGTH (MAX_COMMAND_LENGTH + 100)`
- `#define NOT_IN_LIST -1`

- `#define NO_EMPTY_INDEX -1`
- `#define FIELD_SIZE 25`
- `#define ADJUSTMENT_CONSTANT 0.1`
- `#define STRING_MAX_LENGTH 10000`
- `#define TABS ' '`
- `#define NOT_FOUND_STRING " "`
- `#define EDU_MAX_SUBJECTS 10`
- `#define DATABASE_PATH "./bin/data/database.txt"`

#### 4.2.1 Detailed Description

Contains symbolic constants used throughout the program.

<Detailed esription="" here>="">

### 4.3 database.h File Reference

Contains elements relating to the database.

```
#include "education.h"
```

Include dependency graph for database.h: This graph shows which files directly or indirectly include this file:

#### Classes

- struct [database](#)

#### Functions

- void **freeDatabase** (struct [database](#) \*)
  - struct [database](#) \* **createDatabase** (char \*)
  - struct [education](#) \* **findEducation** (char \*, struct [database](#) \*)
- Finds an education in a database and returns a pointer to the education.*

#### 4.3.1 Detailed Description

Contains elements relating to the database.

<Detailed esription="" here>="">

#### 4.3.2 Function Documentation

##### 4.3.2.1 findEducation()

```
struct education* findEducation (
    char * key,
    struct database * database )
```

Finds an education in a database and returns a pointer to the education.

## Parameters

<code>database</code>	is the database, which will be searched
-----------------------	---

## Returns

struct education\* An education which name matches key or NULL if nothing was found

## 4.4 education.h File Reference

Contains elements relating to educations.

```
#include "region.h"
#include "subjects.h"
#include "vector.h"
```

Include dependency graph for education.h: This graph shows which files directly or indirectly include this file:

## Classes

- struct [education](#)  
*Describes an education and all it requirements.*

## Functions

- struct [education](#) [createDefaultEducation](#) (int amount\_of\_interests, int amount\_of\_subjects)  
*Assigns default values to the fields of the education struct.*
- struct [education](#) \* [createArrayOfEducations](#) (int amount\_of\_educations)  
*Allocate memory for an array of educations and return a pointer to it.*
- void [freeEducation](#) (struct [education](#) \*)

### 4.4.1 Detailed Description

Contains elements relating to educations.

This file contains the education struct and the function that creates educations.

### 4.4.2 Function Documentation

#### 4.4.2.1 createArrayOfEducations()

```
struct education * createArrayOfEducations (
    int amount_of_educations )
```

Allocate memory for an array of educations and return a pointer to it.

## Parameters

<code>amount_of_educations</code>	The amount of educations to be stored in the array
-----------------------------------	--

## 4.4.2.2 createDefaultEducation()

```
struct education createDefaultEducation (
    int amount_of_interests,
    int amount_of_subjects )
```

Assigns default values to the fields of the education struct.

## Parameters

<code>amount_of_interests</code>	The number of interests the education should hold
<code>amount_of_subjects</code>	The number of subjects the education should hold

## 4.5 parser.h File Reference

Contains elements relating to parsing the database.

```
#include <stdio.h>
#include <stdlib.h>
#include "database.h"
#include "region.h"
Include dependency graph for parser.h:
```

## Functions

- void `parseDatabase` (struct `database` \*`database`, FILE \*`filereader`)  
*Parse the database file and set all values in the database.*
- void `parseDatabaseLine` (const char key[], struct `database` \*`database`, FILE \*`filereader`)  
*Parse the line containing key and return into database.*
- void `findDatabaseLine` (const char key[], FILE \*`filereader`, char \*`current_line`)  
*Search the database until the first word of a line matches with key. Return the line through current\_line. If line does not exist, return NOT\_FOUND\_STRING.*
- int `parseNumOfEdu` (FILE \*`filereader`)  
*Returns the number of educations from database file.*
- int `parseNumOfInterests` (FILE \*`filereader`)  
*Parse/count the number of intersts in the database file and return as int.*
- void `parseEduNames` (int amount\_of\_educations, struct `education` \*`educations`, char `current_line`[])  
*Parses the name for each education.*
- void `parseEduDesc` (int amount\_of\_educations, struct `education` \*`educations`, char `current_line`[])  
*Parses the description for each education.*
- void `parseEduLink` (int amount\_of\_educations, struct `education` \*`educations`, char `current_line`[])

- void [parseEduRegion](#) (int amount\_of\_educations, struct [education](#) \*educations, char current\_line[])  
*Parses the region for each education.*
- void [parseSubReq](#) (int amount\_of\_educations, struct [education](#) \*educations, char current\_line[])  
*Parses the subject requirements for each education.*
- void [parseReqGrade](#) (int amount\_of\_educations, struct [education](#) \*educations, char current\_line[])  
*Parses the required average grade for each education.*
- void [parseInterestNames](#) (struct [database](#) \*database, FILE \*filereader)  
*Parse the names of each interest and return to the database.*
- void [parseInterestValues](#) (int amount\_of\_interests, int amount\_of\_educations, struct [education](#) \*educations, FILE \*filereader)  
*Parse the values for each interest in all educations and return into educations.*
- char \* [parseEduString](#) (char \*current\_line, int amount\_of\_educations, int offset)  
*Scans the current line + i until TABS or newline. Saves the scanned string and returns a pointer to it.*
- char \*\* [createArrayOfStrings](#) (int amount\_of\_strings)  
*Allocate memory for an array of strings and return a pointer to it.*
- int [sseek](#) (char \*, char)
- void [readReqString](#) (struct [qualification](#) \*, char \*, int)  
*Read a requiremnt from a string.*
- enum [region strToReg](#) (char \*region\_string)  
*Converts a string into an enum region and return an enum region.*

### 4.5.1 Detailed Description

Contains elements relating to parsing the database.

<Detailed esription="" here>="">

### 4.5.2 Function Documentation

#### 4.5.2.1 createArrayOfStrings()

```
char ** createArrayOfStrings (
    int amount_of_strings )
```

Allocate memory for an array of strings and return a pointer to it.

##### Parameters

<i>amount_of_strings</i>	The amount of strings to be stored in the array
--------------------------	---

#### 4.5.2.2 findDatabaseLine()

```
void findDatabaseLine (
    const char key[],
```

```
FILE * filereader,
char * current_line )
```

Search the database until the first word of a line matches with key. Return the line through *current\_line*. If line does not exist, return NOT\_FOUND\_STRING.

#### Parameters

<i>key</i>	The term to search for
<i>filereader</i>	The database file
<i>current_line</i>	Return through this parameter

#### 4.5.2.3 parseDatabase()

```
void parseDatabase (
    struct database * database,
    FILE * filereader )
```

Parse the database file and set all values in the database.

#### Parameters

<i>database</i>	The database to modify
<i>filereader</i>	The database file

#### 4.5.2.4 parseDatabaseLine()

```
void parseDatabaseLine (
    const char key[],
    struct database * database,
    FILE * filereader )
```

Parse the line containing key and return into database.

#### Parameters

<i>key</i>	The relevant line to parse
<i>database</i>	The database
<i>filereader</i>	The database file

#### 4.5.2.5 parseEduDesc()

```
void parseEduDesc (
    int amount_of_educations,
```

```
struct education * educations,  
char current_line[] )
```

Parses the description for each education.

Parses the "read further" link for each education.

#### Parameters

<i>educations</i>	An array of educations
<i>amount_of_educations</i>	The amount of educations
<i>current_line</i>	The line to parse education names

#### 4.5.2.6 parseEduNames()

```
void parseEduNames (  
    int amount_of_educations,  
    struct education * educations,  
    char current_line[] )
```

Parses the name for each education.

#### Parameters

<i>educations</i>	An array of educations
<i>amount_of_educations</i>	The amount of educations
<i>current_line</i>	The line to parse education names

#### 4.5.2.7 parseEduRegion()

```
void parseEduRegion (  
    int amount_of_educations,  
    struct education * educations,  
    char current_line[] )
```

Parses the region for each education.

#### Parameters

<i>educations</i>	An array of educations
<i>amount_of_educations</i>	The amount of educations
<i>current_line</i>	The line to parse regions from

#### 4.5.2.8 parseEduString()

```
char * parseEduString (
    char * current_line,
    int amount_of_educations,
    int offset )
```

Scans the current line + i until TABS or newline. Saves the scanned string and returns a pointer to it.

##### Parameters

<i>current_line</i>	The line to scan
<i>amount_of_educations</i>	The amount of educations in database
<i>offset</i>	The offset to decide how many chars to skip in <i>current_line</i>

#### 4.5.2.9 parseInterestNames()

```
void parseInterestNames (
    struct database * database,
    FILE * filereader )
```

Parse the names of each interest and return to the database.

##### Parameters

<i>database</i>	The database
<i>filereader</i>	The database file

#### 4.5.2.10 parseInterestValues()

```
void parseInterestValues (
    int amount_of_interests,
    int amount_of_educations,
    struct education * educations,
    FILE * filereader )
```

Parse the values for each interest in all educations and return into educations.

##### Parameters

<i>amount_of_interests</i>	The amount of interests
<i>amount_of_educations</i>	The amount of educations
<i>educations</i>	The array of educations
<i>filereader</i>	The database file



#### 4.5.2.11 parseNumOfEdu()

```
int parseNumOfEdu (
    FILE * filereader )
```

Returns the number of educations from database file.

##### Parameters

<i>filereader</i>	The file to read from
-------------------	-----------------------

#### 4.5.2.12 parseNumOfInterests()

```
int parseNumOfInterests (
    FILE * filereader )
```

Parse/count the number of intersts in the database file and return as int.

##### Parameters

<i>filereader</i>	The database file
-------------------	-------------------

#### 4.5.2.13 parseReqGrade()

```
void parseReqGrade (
    int amount_of_educations,
    struct education * educations,
    char current_line[] )
```

Parses the required average grade for each education.

##### Parameters

<i>educations</i>	An array of educations
<i>amount_of_educations</i>	The amount of educations
<i>current_line</i>	The line to parse the required average grade from

#### 4.5.2.14 parseSubReq()

```
void parseSubReq (
    int amount_of_educations,
```

```
struct education * educations,
char current_line[] )
```

Parses the subject requirements for each education.

#### Parameters

<i>educations</i>	An array of educations
<i>amount_of_educations</i>	The amount of educations
<i>current_line</i>	The line to parse subject requirements from

#### 4.5.2.15 readReqString()

```
void readReqString (
    struct qualification * qualification,
    char * string,
    int education_location )
```

Read a requiremnt from a string.

#### Parameters

<i>qualification</i>	The qualification structure, where the read input is stored.
<i>string</i>	The string in which the requirements exists.
<i>education_location</i>	Which column is the educations requirements in.

#### 4.5.2.16 strToReg()

```
int strToReg (
    char * region_string )
```

Converts a string into an enum region and return an enum region.

#### Parameters

<i>region_string</i>	The string to convert
----------------------	-----------------------

## 4.6 profile.h File Reference

Contains elements relating to profile.

```
#include "vector.h"
#include "subjects.h"
```

```
#include "region.h"
#include "education.h"
#include "constants.h"
```

Include dependency graph for profile.h: This graph shows which files directly or indirectly include this file:

## Classes

- struct [profile](#)  
*Describes a user.*

## Functions

- struct [profile](#) [createProfile](#) (int number\_of\_interests)  
*Allocates memory for each of the fields in the profile struct.*
- void [freeProfile](#) (struct [profile](#) p)  
*Frees the allocated memory for the given profile.*
- void [printProfile](#) (struct [profile](#) p)  
*Prints information stored in the given profil.*

### 4.6.1 Detailed Description

Contains elements relating to profile.

<Detailed esription="" here>="">

### 4.6.2 Function Documentation

#### 4.6.2.1 createProfile()

```
struct profile createProfile (
    int number_of_interests )
```

Allocates memory for each of the fields in the profile struct.

#### Parameters

<i>number_of_interests</i>	The number of interests allocated
----------------------------	-----------------------------------

#### 4.6.2.2 freeProfile()

```
void freeProfile (
    struct profile p )
```

Frees the allocated memory for the given profile.

#### Parameters

<i>p</i>	The profile struct which is freed
----------	-----------------------------------

#### 4.6.2.3 printProfile()

```
void printProfile (
    struct profile p )
```

Prints information stored in the given profil.

#### Parameters

<i>p</i>	The profile struct which is printed
----------	-------------------------------------

## 4.7 region.h File Reference

Contains geographical elements.

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

### Classes

- struct [location](#)

### Enumerations

- enum [region](#) {  
    **NORTH\_JUTLAND** = 0, **CENTRAL\_JUTLAND**, **SOUTHERN\_DENMARK**, **ZEALAND**,  
    **CAPITAL\_REGION** }  
    *Describes a region.*

#### 4.7.1 Detailed Description

Contains geographical elements.

This file contains the enums for different regions and the struct that symbolises a location.

#### 4.7.2 Enumeration Type Documentation

#### 4.7.2.1 region

enum `region`

Describes a region.

This enum describes a region AKA it describes a location in denmark.

## 4.8 serialize.h File Reference

Save and load profile data.

```
#include "profile.h"
```

Include dependency graph for serialize.h:

### Macros

- `#define SAVE_FILE "data/save.data"`

### Functions

- `int saveProfile (struct profile *)`
- `struct profile * loadProfile ()`

#### 4.8.1 Detailed Description

Save and load profile data.

#### Author

#### Version

0.1

#### Date

2019-11-27

#### Copyright

Copyright (c) 2019

## 4.9 subjects.h File Reference

Contains code regarding subjects and qualifications for educations.

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

### Classes

- struct [subject](#)
- struct [qualification](#)

### Enumerations

- enum {  
**MATHEMATICS, CHEMISTRY, BIOLOGY, PHYSICS,**  
**ENGLISH, BIOTECHNOLOGY, GEOSCIENCE, HISTORY,**  
**IDEA\_HISTORY, INFORMATICS, INTERNATIONAL\_ECONOMICS, COMMUNICATION\_AND\_IT,**  
**RELIGION, SOCIALSTUDIES, BUSINESS\_ECONOMICS, CONTEMPORARY\_HISTORY,**  
**FRENCH, SPANISH, GERMAN, CHINESE,**  
**ARABIC, GREEK, ITALIAN, JAPANESE,**  
**LATIN, PORTUGUESE, RUSSIAN, NONE,**  
**DANISH }**
- enum **level** { **Z, C, B, A** }

### Functions

- struct [qualification](#) **createQualifications** (int number\_of\_qualifications)
- void **freeQualifications** (struct [qualification](#) \*)
- enum [stringToClass](#) (char \*)  
*Returns the enum class associated with the given string.*
- enum level [charToLevel](#) (char ch)  
*Returns the enum level associated with the given char.*
- char [levelToChar](#) (enum level l)  
*Returns the character associated with the given enum level.*

#### 4.9.1 Detailed Description

Contains code regarding subjects and qualifications for educations.

<Detailed escription="" here>="">

#### 4.9.2 Function Documentation

##### 4.9.2.1 charToLevel()

```
enum level charToLevel (  
    char ch )
```

Returns the enum level associated with the given char.

## Parameters

<i>ch</i>	The character which is converted into an enum level
-----------	---

## 4.9.2.2 levelToChar()

```
enum char levelToChar (  
    enum level l )
```

Returns the character associated with the given enum level.

## Parameters

<i>l</i>	The enum level which is converted into a character
----------	--

## 4.9.2.3 stringToClass()

```
enum class stringToClass (  
    char * string ) [strong]
```

Returns the enum class associated with the given string.

## Parameters

<i>string</i>	The string which is converted into an enum class
---------------	--

## 4.10 vector.h File Reference

Contains elements relating to vectors.

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

### Classes

- struct [vector](#)

### Functions

- struct [vector](#) [createVector](#) (int size)  
*creates a vector on the heap and outputs it*

- struct `vector copyVector` (struct `vector` v)  
*Copies the the inputted vector into vector copy and returns this.*
- struct `vector addVector` (struct `vector` v1, struct `vector` v2)  
*Adds two vectors together and outputs the sum as a vector.*
- struct `vector subtractVector` (struct `vector` v1, struct `vector` v2)  
*Subtracts the second vector from the first vector and returns the result as a vector.*
- struct `vector scaleVector` (struct `vector` v, double scale)  
*Multiplies the given vector's array values by the value inputted as scale, then outputs the result as a vector.*
- struct `vector normalizeVector` (struct `vector` v)  
*Normalises a vector via scaling it by one over it's length, then returns the normalized vector.*
- double `lengthOfVector` (struct `vector` v)  
*Calculates and returns the length of the given vector.*
- double `dotProduct` (struct `vector` v1, struct `vector` v2)  
*Calculates and returns the dot product of two vectors.*
- void `printVector` (struct `vector` v)  
*Prints a vector.*
- void `freeVector` (struct `vector` v)  
*frees the dynamically allocated array on the heap*
- void `freeVectorM` (int num,...)  
*Frees a variable number of struct vectors using free(Vector)*

#### 4.10.1 Detailed Description

Contains elements relating to vectors.

This file contains the vector struct and various functions used to create, manipulate or free vectors.

#### 4.10.2 Function Documentation

##### 4.10.2.1 addVector()

```
struct vector addVector (
    struct vector v1,
    struct vector v2 )
```

Adds two vectors together and outputs the sum as a vector.

##### Parameters

v1	The first vector struct: v1.array[] is a vector, v1.size number of elements in the vector
v2	The second vector struct: v2.array[] is a vector



#### 4.10.2.2 copyVector()

```
struct vector copyVector (
    struct vector v )
```

Copies the the inputted vector into vector copy and returns this.

##### Parameters

<i>v</i>	The input vector that is copied
----------	---------------------------------

#### 4.10.2.3 createVector()

```
struct vector createVector (
    int size )
```

creates a vector on the heap and outputs it

##### Parameters

<i>size</i>	The number of elements in the vector
-------------	--------------------------------------

#### 4.10.2.4 dotProduct()

```
double dotProduct (
    struct vector v1,
    struct vector v2 )
```

Calculates and returns the dot product of two vectors.

##### Parameters

<i>v1</i>	The first vector to be used for dot product calculation
<i>v2</i>	The second vector to be used for dot product calculation

#### 4.10.2.5 freeVector()

```
void freeVector (
    struct vector v )
```

frees the dynamically allocated array on the heap

**Parameters**

<i>v</i>	The vector struct containing the array on the heap
----------	--

**4.10.2.6 freeVectorM()**

```
void freeVectorM (
    int num,
    ... )
```

Frees a variable number of struct vectors using free(Vector)

**Parameters**

<i>num</i>	The number of arguments (vectors) that should be freed
------------	--

**4.10.2.7 lengthOfVector()**

```
double lengthOfVector (
    struct vector v )
```

Calculates and returns the length of the given vector.

**Parameters**

<i>v</i>	The vector whose length is found
----------	----------------------------------

**4.10.2.8 normalizeVector()**

```
struct vector normalizeVector (
    struct vector v )
```

Normalises a vector via scaling it by one over it's length, then returns the normalized vector.

**Parameters**

<i>v</i>	The vector which is to be normalized
----------	--------------------------------------

#### 4.10.2.9 printVector()

```
void printVector (
    struct vector v )
```

Prints a vector.

##### Parameters

<i>v</i>	The vector that is printed
----------	----------------------------

#### 4.10.2.10 scaleVector()

```
struct vector scaleVector (
    struct vector v,
    double scale )
```

Multiplies the given vector's array values by the value inputted as scale, then outputs the result as a vector.

##### Parameters

<i>v</i>	The vector that should be up- or downscaled
<i>scale</i>	The value that the vector should be scaled by

#### 4.10.2.11 subtractVector()

```
struct vector subtractVector (
    struct vector v1,
    struct vector v2 )
```

Subtracts the second vector from the first vector and returns the result as a vector.

##### Parameters

<i>v1</i>	The vector that should be subtracted from
<i>v2</i>	The vector that is used for subtraction



# Index

- addVector
  - vector.h, 26
- amount\_of\_interests
  - database, 5
- charToLevel
  - subjects.h, 24
- commands.h, 11
- constants.h, 11
- copyVector
  - vector.h, 26
- createArrayOfEducations
  - education.h, 13
- createArrayOfStrings
  - parser.h, 15
- createDefaultEducation
  - education.h, 14
- createProfile
  - profile.h, 21
- createVector
  - vector.h, 27
- Database, 6
- database, 5
  - amount\_of\_interests, 5
  - educations, 5
  - interest\_string, 5
- database.h, 12
  - findEducation, 12
- dotProduct
  - vector.h, 27
- education, 6
- education.h, 13
  - createArrayOfEducations, 13
  - createDefaultEducation, 14
- educations
  - database, 5
- findDatabaseLine
  - parser.h, 15
- findEducation
  - database.h, 12
- freeProfile
  - profile.h, 21
- freeVector
  - vector.h, 27
- freeVectorM
  - vector.h, 28
- interest\_string
  - database, 5
- lengthOfVector
  - vector.h, 28
- level
  - subject, 8
- levelToChar
  - subjects.h, 25
- location, 7
- normalizeVector
  - vector.h, 28
- parseDatabase
  - parser.h, 16
- parseDatabaseLine
  - parser.h, 16
- parseEduDesc
  - parser.h, 16
- parseEduNames
  - parser.h, 17
- parseEduRegion
  - parser.h, 17
- parseEduString
  - parser.h, 17
- parseInterestNames
  - parser.h, 18
- parseInterestValues
  - parser.h, 18
- parseNumOfEdu
  - parser.h, 18
- parseNumOfInterests
  - parser.h, 19
- parseReqGrade
  - parser.h, 19
- parseSubReq
  - parser.h, 19
- parser.h, 14
  - createArrayOfStrings, 15
  - findDatabaseLine, 15
  - parseDatabase, 16
  - parseDatabaseLine, 16
  - parseEduDesc, 16
  - parseEduNames, 17
  - parseEduRegion, 17
  - parseEduString, 17
  - parseInterestNames, 18
  - parseInterestValues, 18
  - parseNumOfEdu, 18
  - parseNumOfInterests, 19

- parseReqGrade, [19](#)
  - parseSubReq, [19](#)
  - readReqString, [20](#)
  - strToReg, [20](#)
- printProfile
  - profile.h, [22](#)
- printVector
  - vector.h, [28](#)
- profile, [7](#)
- profile.h, [20](#)
  - createProfile, [21](#)
  - freeProfile, [21](#)
  - printProfile, [22](#)
- qualification, [8](#)
  - subjects, [8](#)
- readReqString
  - parser.h, [20](#)
- region
  - region.h, [22](#)
- region.h, [22](#)
  - region, [22](#)
- scaleVector
  - vector.h, [29](#)
- serialize.h, [23](#)
- strToReg
  - parser.h, [20](#)
- stringToClass
  - subjects.h, [25](#)
- subject, [8](#)
  - level, [8](#)
- subjects
  - qualification, [8](#)
- subjects.h, [24](#)
  - charToLevel, [24](#)
  - levelToChar, [25](#)
  - stringToClass, [25](#)
- subtractVector
  - vector.h, [29](#)
- vector, [9](#)
- vector.h, [25](#)
  - addVector, [26](#)
  - copyVector, [26](#)
  - createVector, [27](#)
  - dotProduct, [27](#)
  - freeVector, [27](#)
  - freeVectorM, [28](#)
  - lengthOfVector, [28](#)
  - normalizeVector, [28](#)
  - printVector, [28](#)
  - scaleVector, [29](#)
  - subtractVector, [29](#)