

Automatiser dit studievalg

Generated by Doxygen 1.8.13

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
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
3.6	qualification Struct Reference	7
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	8
3.8	vector Struct Reference	8

4	File Documentation	9
4.1	commands.h File Reference	9
4.2	constants.h File Reference	9
4.2.1	Detailed Description	10
4.3	database.h File Reference	10
4.3.1	Detailed Description	10
4.3.2	Function Documentation	10
4.3.2.1	findEducation()	10
4.4	education.h File Reference	11
4.4.1	Detailed Description	11
4.4.2	Function Documentation	11
4.4.2.1	createArrayOfEducations()	11
4.4.2.2	createDefaultEducation()	12
4.5	parser.h File Reference	12
4.5.1	Detailed Description	13
4.5.2	Function Documentation	13
4.5.2.1	createArrayOfStrings()	13
4.5.2.2	findDatabaseLine()	13
4.5.2.3	parseDatabase()	14
4.5.2.4	parseDatabaseLine()	14
4.5.2.5	parseEduDesc()	14
4.5.2.6	parseEduNames()	15
4.5.2.7	parseEduRegion()	15
4.5.2.8	parseEduString()	16
4.5.2.9	parseInterestNames()	16
4.5.2.10	parseInterestValues()	16
4.5.2.11	parseNumOfEdu()	17
4.5.2.12	parseNumOfInterests()	17
4.5.2.13	parseReqGrade()	17
4.5.2.14	parseSubReq()	17

4.5.2.15	readReqString()	18
4.5.2.16	strToReg()	18
4.6	profile.h File Reference	18
4.6.1	Detailed Description	19
4.6.2	Function Documentation	19
4.6.2.1	createProfile()	19
4.6.2.2	freeProfile()	19
4.6.2.3	printProfile()	20
4.7	region.h File Reference	20
4.7.1	Detailed Description	20
4.7.2	Enumeration Type Documentation	20
4.7.2.1	region	21
4.8	serialize.h File Reference	21
4.8.1	Detailed Description	21
4.9	subjects.h File Reference	22
4.9.1	Detailed Description	22
4.9.2	Function Documentation	22
4.9.2.1	charToLevel()	22
4.9.2.2	levelToChar()	23
4.9.2.3	stringToClass()	23
4.10	vector.h File Reference	23
4.10.1	Detailed Description	24
4.10.2	Function Documentation	24
4.10.2.1	addVector()	24
4.10.2.2	copyVector()	25
4.10.2.3	createVector()	25
4.10.2.4	dotProduct()	25
4.10.2.5	freeVector()	25
4.10.2.6	freeVectorM()	26
4.10.2.7	lengthOfVector()	26
4.10.2.8	normalizeVector()	26
4.10.2.9	printVector()	27
4.10.2.10	scaleVector()	27
4.10.2.11	subtractVector()	27

Chapter 1

Class Index

1.1 Class List

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

database	5
Database		
	A structure to store a database	6
education		
	Describes an education and all it requirements	6
location	7
profile	7
qualification	7
subject	8
vector	8

Chapter 2

File Index

2.1 File List

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

commands.h	Contains functions related to command handling	9
constants.h	Contains symbolic constants used throughout the program	9
database.h	Contains elements relating to the database	10
education.h	Contains elements relating to educations	11
parser.h	Contains elements relating to parsing the database	12
profile.h	Contains elements relating to profile	18
region.h	Contains geographical elements	20
serialize.h	Save and load profile data	21
subjects.h	Contains code regarding subjects and qualifications for educations	22
vector.h	Contains elements relating to vectors	23

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

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]

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 current_line

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](#)

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, 24
- amount_of_interests
 - database, 5
- charToLevel
 - subjects.h, 22
- commands.h, 9
- constants.h, 9
- copyVector
 - vector.h, 24
- createArrayOfEducations
 - education.h, 11
- createArrayOfStrings
 - parser.h, 13
- createDefaultEducation
 - education.h, 12
- createProfile
 - profile.h, 19
- createVector
 - vector.h, 25
- Database, 6
- database, 5
 - amount_of_interests, 5
 - educations, 5
 - interest_string, 5
- database.h, 10
 - findEducation, 10
- dotProduct
 - vector.h, 25
- education, 6
- education.h, 11
 - createArrayOfEducations, 11
 - createDefaultEducation, 12
- educations
 - database, 5
- findDatabaseLine
 - parser.h, 13
- findEducation
 - database.h, 10
- freeProfile
 - profile.h, 19
- freeVector
 - vector.h, 25
- freeVectorM
 - vector.h, 26
- interest_string
 - database, 5
- lengthOfVector
 - vector.h, 26
- level
 - subject, 8
- levelToChar
 - subjects.h, 23
- location, 7
- normalizeVector
 - vector.h, 26
- parseDatabase
 - parser.h, 14
- parseDatabaseLine
 - parser.h, 14
- parseEduDesc
 - parser.h, 14
- parseEduNames
 - parser.h, 15
- parseEduRegion
 - parser.h, 15
- parseEduString
 - parser.h, 15
- parseInterestNames
 - parser.h, 16
- parseInterestValues
 - parser.h, 16
- parseNumOfEdu
 - parser.h, 16
- parseNumOfInterests
 - parser.h, 17
- parseReqGrade
 - parser.h, 17
- parseSubReq
 - parser.h, 17
- parser.h, 12
 - createArrayOfStrings, 13
 - findDatabaseLine, 13
 - parseDatabase, 14
 - parseDatabaseLine, 14
 - parseEduDesc, 14
 - parseEduNames, 15
 - parseEduRegion, 15
 - parseEduString, 15
 - parseInterestNames, 16
 - parseInterestValues, 16
 - parseNumOfEdu, 16
 - parseNumOfInterests, 17

- parseReqGrade, [17](#)
 - parseSubReq, [17](#)
 - readReqString, [18](#)
 - strToReg, [18](#)
- printProfile
 - profile.h, [20](#)
- printVector
 - vector.h, [26](#)
- profile, [7](#)
- profile.h, [18](#)
 - createProfile, [19](#)
 - freeProfile, [19](#)
 - printProfile, [20](#)
- qualification, [7](#)
 - subjects, [8](#)
- readReqString
 - parser.h, [18](#)
- region
 - region.h, [20](#)
- region.h, [20](#)
 - region, [20](#)
- scaleVector
 - vector.h, [27](#)
- serialize.h, [21](#)
- strToReg
 - parser.h, [18](#)
- stringToClass
 - subjects.h, [23](#)
- subject, [8](#)
 - level, [8](#)
- subjects
 - qualification, [8](#)
- subjects.h, [22](#)
 - charToLevel, [22](#)
 - levelToChar, [23](#)
 - stringToClass, [23](#)
- subtractVector
 - vector.h, [27](#)
- vector, [8](#)
- vector.h, [23](#)
 - addVector, [24](#)
 - copyVector, [24](#)
 - createVector, [25](#)
 - dotProduct, [25](#)
 - freeVector, [25](#)
 - freeVectorM, [26](#)
 - lengthOfVector, [26](#)
 - normalizeVector, [26](#)
 - printVector, [26](#)
 - scaleVector, [27](#)
 - subtractVector, [27](#)