

P1: Automatiser dit studievalg

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Coding Style</b>	<b>1</b>
<b>2</b>	<b>Main Program</b>	<b>3</b>
<b>3</b>	<b>Navngivning</b>	<b>5</b>
<b>4</b>	<b>P1</b>	<b>7</b>
<b>5</b>	<b>Class Index</b>	<b>9</b>
5.1	Class List . . . . .	9
<b>6</b>	<b>File Index</b>	<b>11</b>
6.1	File List . . . . .	11
<b>7</b>	<b>Class Documentation</b>	<b>13</b>
7.1	CuString Struct Reference . . . . .	13
7.1.1	Detailed Description . . . . .	13
7.2	CuSuite Struct Reference . . . . .	13
7.2.1	Detailed Description . . . . .	13
7.3	CuTest Struct Reference . . . . .	14
7.3.1	Detailed Description . . . . .	14
7.4	database Struct Reference . . . . .	14
7.4.1	Detailed Description . . . . .	14
7.5	Database Struct Reference . . . . .	14
7.5.1	Detailed Description . . . . .	15
7.6	education Struct Reference . . . . .	15

7.6.1	Detailed Description	15
7.7	location Struct Reference	15
7.7.1	Detailed Description	16
7.8	profile Struct Reference	16
7.8.1	Detailed Description	16
7.9	qualification Struct Reference	16
7.9.1	Detailed Description	17
7.10	subject Struct Reference	17
7.10.1	Detailed Description	17
7.11	vector Struct Reference	17
7.11.1	Detailed Description	17
<b>8</b>	<b>File Documentation</b>	<b>19</b>
8.1	include/commands.h File Reference	19
8.1.1	Detailed Description	20
8.1.2	Function Documentation	21
8.1.2.1	chooseFromList()	21
8.1.2.2	classNameStr()	21
8.1.2.3	convertScale()	22
8.1.2.4	deleteCmd()	22
8.1.2.5	evalCmd()	23
8.1.2.6	findCmd()	23
8.1.2.7	getEmptyIndex()	24
8.1.2.8	getIndex()	24
8.1.2.9	getRegionName()	25
8.1.2.10	isQualified()	25
8.1.2.11	levelAsValue()	26
8.1.2.12	listCmd()	27
8.1.2.13	listIsFull()	27
8.1.2.14	loadProfile()	27
8.1.2.15	recommendCmd()	29

8.1.2.16	<a href="#">saveCmd()</a>	29
8.1.2.17	<a href="#">saveProfile()</a>	30
8.1.2.18	<a href="#">searchCmd()</a>	31
8.1.2.19	<a href="#">setImportantSubjects()</a>	31
8.1.2.20	<a href="#">setOtherSubjects()</a>	32
8.1.2.21	<a href="#">setProfileInterests()</a>	32
8.1.2.22	<a href="#">setProfileLocation()</a>	33
8.1.2.23	<a href="#">setProfileQualifications()</a>	33
8.1.2.24	<a href="#">setSubjects()</a>	34
8.1.2.25	<a href="#">surveyCmd()</a>	34
8.1.2.26	<a href="#">validScaleValue()</a>	35
8.2	<a href="#">include/constants.h File Reference</a>	36
8.2.1	<a href="#">Detailed Description</a>	36
8.3	<a href="#">include/database.h File Reference</a>	36
8.3.1	<a href="#">Detailed Description</a>	37
8.4	<a href="#">include/education.h File Reference</a>	37
8.4.1	<a href="#">Detailed Description</a>	37
8.4.2	<a href="#">Function Documentation</a>	37
8.4.2.1	<a href="#">createArrayOfEducations()</a>	37
8.4.2.2	<a href="#">createDefaultEducation()</a>	38
8.5	<a href="#">include/parser.h File Reference</a>	38
8.5.1	<a href="#">Detailed Description</a>	39
8.5.2	<a href="#">Function Documentation</a>	40
8.5.2.1	<a href="#">createArrayOfStrings()</a>	40
8.5.2.2	<a href="#">findDatabaseLine()</a>	40
8.5.2.3	<a href="#">parseDatabase()</a>	41
8.5.2.4	<a href="#">parseDatabaseLine()</a>	42
8.5.2.5	<a href="#">parseEduDesc()</a>	43
8.5.2.6	<a href="#">parseEduNames()</a>	43
8.5.2.7	<a href="#">parseEduRegion()</a>	44

8.5.2.8	<a href="#">parseEduString()</a>	44
8.5.2.9	<a href="#">parseInterestNames()</a>	45
8.5.2.10	<a href="#">parseInterestValues()</a>	45
8.5.2.11	<a href="#">parseNumOfEdu()</a>	46
8.5.2.12	<a href="#">parseNumOfInterests()</a>	47
8.5.2.13	<a href="#">parseReqGrade()</a>	47
8.5.2.14	<a href="#">parseSubReq()</a>	48
8.5.2.15	<a href="#">readReqString()</a>	48
8.5.2.16	<a href="#">strToReg()</a>	49
8.6	<a href="#">include/profile.h File Reference</a>	50
8.6.1	<a href="#">Detailed Description</a>	50
8.6.2	<a href="#">Function Documentation</a>	50
8.6.2.1	<a href="#">createProfile()</a>	50
8.6.2.2	<a href="#">freeProfile()</a>	51
8.6.2.3	<a href="#">printProfile()</a>	51
8.7	<a href="#">include/region.h File Reference</a>	52
8.7.1	<a href="#">Detailed Description</a>	52
8.7.2	<a href="#">Enumeration Type Documentation</a>	52
8.7.2.1	<a href="#">region</a>	53
8.8	<a href="#">include/subjects.h File Reference</a>	53
8.8.1	<a href="#">Detailed Description</a>	54
8.8.2	<a href="#">Function Documentation</a>	54
8.8.2.1	<a href="#">charToLevel()</a>	54
8.8.2.2	<a href="#">levelToChar()</a>	54
8.8.2.3	<a href="#">stringToClass()</a>	55
8.9	<a href="#">include/vector.h File Reference</a>	56
8.9.1	<a href="#">Detailed Description</a>	57
8.9.2	<a href="#">Function Documentation</a>	57
8.9.2.1	<a href="#">addVector()</a>	57
8.9.2.2	<a href="#">copyVector()</a>	57
8.9.2.3	<a href="#">createVector()</a>	58
8.9.2.4	<a href="#">dotProduct()</a>	58
8.9.2.5	<a href="#">freeVector()</a>	59
8.9.2.6	<a href="#">freeVectorM()</a>	59
8.9.2.7	<a href="#">lengthOfVector()</a>	60
8.9.2.8	<a href="#">normalizeVector()</a>	60
8.9.2.9	<a href="#">printVector()</a>	60
8.9.2.10	<a href="#">scaleVector()</a>	61
8.9.2.11	<a href="#">subtractVector()</a>	61

# Chapter 1

## Coding Style

### General

- Indentation: 4 times whitespace per indent
- Language: English (both code and comments)
- Brackets: None if only one argument

### ### Functions

```
void func(param1, param2){  
    function();  
}
```

### Structures

```
int i;  
for(i = 0; i < 9; i++){  
    function();  
}
```

```
int i;  
for(i = 0; i < 9; i++){  
    function();  
    function1();  
}
```

```
if(this && that)  
    function();  
else if(just this)  
    function7();  
else  
    function1();
```

```
if(this && that){  
    function();  
    function1();  
} else if(just this){  
    function7();  
    function();  
} else{  
    function1();  
    function2();  
}
```

```
while(this is true)
    function();
```

```
while(this is true){
    function();
    function1();
}
```

```
do
    function();
while(that);
```

```
do{
    function();
    function1();
} while(that);
```

```
switch(expression){
    case 1:
        function1();
        break;
    case 2:
        function2();
        break;
    default:
        return shit;
}
```

### ### Operators

```
i = 3 + 4 * (7 / 5 + 5 % 2);
i += 2;
i -= 2;
```

### ### Structures

```
char array[SYMBOLIC_CONSTANT] = "Some text";
char *array = "Some other text";
char array[] = "Some text again";
```

variablenames are written with underscores:

```
int this_integer = 2;
char some_character = 'a';
```

### Calls

Functionnames are written in camelcase:

```
callFunction(a, 2, 3, a, bdw, w);
parseSomething(a, b);
```



## Chapter 2

# Main Program

### Basic functions

#### Input af datasæt

Funktionen skal finde en liste af alle forskellige interesser, der kommer fra datasæt (separat fil). Funktionen skal finde en liste over alle uddannelser, fra filen Funktionen skal gemme vægtningerne for alle uddannelser og interesser Gemme via outputparametre

#### Brugerinput

Læser data fra terminalen, som brugeren indtaster Holder dialogen med brugeren i gang Får interesser fra "Input af datasæt", som den stiller spørgsmål om

#### Output brugervektor

Funktionen gemmer brugerens interesse-vektor til en fil separat fra data-filen

### Vektorregning

Vektorer skal gemmes som arrays af doubles funktioner returnere gemmen outputvektorer Skal kunne udføre følgende udregninger:

- Add
  - Addition af to vektorer
  - Skal også kunne bruges til substraktion
- Scale
  - Skal kunne skalere en vektor
- Length
  - Skal kunne beregne længden af en vektor ved brug af Pythagoras
- Normalize
  - Skal kunne omdanne en vektor til den tilsvarende enhedsvektor (længde = 1)
- Dot Product
  - Skal kunne bestemme prikprodukt hvis vektorne er enhedsvektorer
  - Udregnes som:  $x_1x_2 + y_1y_2 + z_1z_2$

### Beregning af output

Funktionen skal kunne sammenligne brugerens vektor med uddannelsernes vektorerne og bestemme den, der passer bedst. Bruger prikprodukt og en adjustment vektor (som er den, der justerer brugerens vektor efter tidligere evalueringer)

### Formatering af output

Funktionen skal skrive navnet samt information fra datasættet.

### Input evaluering

Funktion skal beregne adjustment vektoren ud fra brugerens evaluering af den viste uddannelse.

$0.1 < k < 0.5$   $\text{adjust\_vector} = \text{adjust\_vector} + \text{normalize}(\text{study\_vector} - \text{user\_vector}) * k * \text{eval}$

### Command handling

Funktionen skal finde ud af hvilken command brugeren har indtastet samt om der hører argumenter med, hvilket også gemmes. Følgende commands skal håndteres:

- Find uddannelse
- Save uddannelse
- Recommend uddannelse
- Load saved
- Evaluate uddannelse

### Interactions

Funktionen skal fungere som kerne-funktionen der kalder på de andre funktioner og holder på de variabler som programmet gemmer undervejs.

## Chapter 3

# Navngivning

En uddannelse kalder vi `bachelor` En vektor af interesser kalder vi `interest[]`

```
struct bachelor{  
    double interest[];  
}
```

### Filstrukturen af databasen

```
Interesse:      .... \t .... \t .... \t .... \t  
Fag:            mat \t dan \t .... \t  
    Biologi     0.3 \t    \t    \t  
    Datalogi    1.0 \t 0.0 \t    \t      \t beskrivelse \t region \t 'z' \t 'A' (krav)  
    Dansk       0.0 \t 1.0  
    ...  
    ..  
    ...
```

Til at beskrive fagenes niveau bruge ASCII code points for bogstaverne 'A', 'B' og 'C'. Bogstavet 'Z' .. .



## Chapter 4

### P1

Automatiser dit studievalg A302



## Chapter 5

# Class Index

### 5.1 Class List

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

<a href="#">CuString</a>	.....	13
<a href="#">CuSuite</a>	.....	13
<a href="#">CuTest</a>	.....	14
<a href="#">database</a>	.....	14
<a href="#">Database</a>		
	A structure to store a database .....	14
<a href="#">education</a>		
	Describes an education and all it requirements .....	15
<a href="#">location</a>	.....	15
<a href="#">profile</a>		
	Describes a user .....	16
<a href="#">qualification</a>	.....	16
<a href="#">subject</a>	.....	17
<a href="#">vector</a>	.....	17





## Chapter 6

# File Index

### 6.1 File List

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

include/ <a href="#">commands.h</a>	Contains functions related to command handling . . . . .	19
include/ <a href="#">constants.h</a>	Contains symbolic constants used throughout the program . . . . .	36
include/ <b>CuTest.h</b>		??
include/ <a href="#">database.h</a>	Contains elements relating to the database . . . . .	36
include/ <a href="#">education.h</a>	Contains elements relating to educations . . . . .	37
include/ <a href="#">parser.h</a>	Contains elements relating to parsing the database . . . . .	38
include/ <a href="#">profile.h</a>	Contains elements relating to user profiles . . . . .	50
include/ <a href="#">region.h</a>	Contains geographical elements . . . . .	52
include/ <a href="#">subjects.h</a>	Contains code regarding subjects and qualifications for educations . . . . .	53
include/ <a href="#">vector.h</a>	Contains elements relating to vectors . . . . .	56
src/ <b>commands.c</b>		??
src/ <b>database.c</b>		??
src/ <b>education.c</b>		??
src/ <b>main.c</b>		??
src/ <b>parser.c</b>		??
src/ <b>profile.c</b>		??
src/ <b>subjects.c</b>		??
src/ <b>vector.c</b>		??
test/ <b>AllTests.c</b>		??
test/ <b>CuTest.c</b>		??
test/ <b>CuTest.h</b>		??
tools/ <b>createDatabase.py</b>		??



## Chapter 7

# Class Documentation

### 7.1 CuString Struct Reference

#### Public Attributes

- int **length**
- int **size**
- char \* **buffer**

#### 7.1.1 Detailed Description

Definition at line 20 of file CuTest.h.

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

- include/CuTest.h

### 7.2 CuSuite Struct Reference

#### Public Attributes

- int **count**
- [CuTest](#) \* **list** [MAX\_TEST\_CASES]
- int **failCount**

#### 7.2.1 Detailed Description

Definition at line 98 of file CuTest.h.

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

- include/CuTest.h

## 7.3 CuTest Struct Reference

### Public Attributes

- char \* **name**
- TestFunction **function**
- int **failed**
- int **ran**
- const char \* **message**
- jmp\_buf \* **jumpBuf**

### 7.3.1 Detailed Description

Definition at line 43 of file CuTest.h.

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

- include/CuTest.h

## 7.4 database Struct Reference

### Public Attributes

- int **amount\_of\_educations**
- struct [education](#) \* **educations**  
*the amount of educations in the database*
- int [amount\\_of\\_interests](#)  
*an array of educations delimited by amount\_of\_educations*
- char \*\* [interest\\_string](#)  
*the amount of interests in the database*

### 7.4.1 Detailed Description

Definition at line 16 of file database.h.

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

- include/[database.h](#)

## 7.5 Database Struct Reference

A structure to store a database.

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

### 7.5.1 Detailed Description

A structure to store a database.

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

- [include/database.h](#)

## 7.6 education Struct Reference

Describes an education and all it requirements.

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

### Public Attributes

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

### 7.6.1 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.

Definition at line 27 of file education.h.

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

- [include/education.h](#)

## 7.7 location Struct Reference

### Public Attributes

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

### 7.7.1 Detailed Description

Definition at line 24 of file region.h.

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

- include/[region.h](#)

## 7.8 profile Struct Reference

Describes a user.

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

### 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]

### 7.8.1 Detailed Description

Describes a user.

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

Definition at line 31 of file profile.h.

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

- include/[profile.h](#)

## 7.9 qualification Struct Reference

### Public Attributes

- int **amount\_of\_subjects**
- struct [subject](#) \* **subjects**  
*the amount of subjects in qualifications*

### 7.9.1 Detailed Description

Definition at line 75 of file subjects.h.

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

- include/[subjects.h](#)

## 7.10 subject Struct Reference

### Public Attributes

- enum level [level](#)  
*the name of the subject*

### 7.10.1 Detailed Description

Definition at line 66 of file subjects.h.

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

- include/[subjects.h](#)

## 7.11 vector Struct Reference

### Public Attributes

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

### 7.11.1 Detailed Description

Definition at line 14 of file vector.h.

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

- include/[vector.h](#)





## Chapter 8

# File Documentation

### 8.1 include/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"
```

#### Functions

- void `menuCmd` (void)  
*Prints all the possible commands the user can use.*
- void `surveyCmd` (struct `profile` \*user, const struct `database` \*db)  
*Tests the current user for name, location, interests, qualifications and average grade.*
- void `setProfileLocation` (struct `profile` \*user)  
*Sets the region of choice in user.*
- double `convertScale` (int initial\_value)  
*Returns the converted value.*
- int `validScaleValue` (int value, int interval\_start, int interval\_end)  
*Returns a value between interval\_start and interval\_end.*
- int `getValidInteger` (void)  
*Returns a valid integer given through the terminal.*
- void `setProfileInterests` (struct `profile` \*user, const struct `database` \*db)  
*Saves all interests to user as a converted value (see convertScale)*
- void `setProfileQualifications` (struct `profile` \*user)  
*Saves all the users qualifications as given by the terminal.*
- void `setSubjects` (struct `profile` \*user)  
*Sets all qualifications in user to match the enum class.*
- void `setImportantSubjects` (struct `profile` \*user)  
*Saves all the qualifications for the important subjects.*
- const char \* `classNameStr` (enum class class)  
*Returns the name as a string of a class given as an enum class.*

- enum level `levelAsValue` (char c)  
*Returns the enum value of a level given as a character.*
- void `setOtherSubjects` (struct `profile` \*user, int start, int end)  
*Saves all the levels of the other subjects (not the important ones)*
- void `chooseFromList` (struct `profile` \*user, int interval\_start, int interval\_end)  
*Saves the levels of chosen subjects to user.*
- double `getValidDouble` (void)  
*Returns a valid double entered in the terminal.*
- void `evalCmd` (struct `profile` \*user, struct `education` \*current\_education, int arg)  
*Changes the adjustment vector for the user to approach the current education.*
- struct `education` `findCmd` (char \*arg, const struct `database` \*db)  
*Finds and prints out the education with the exact name given as and argument.*
- void `searchCmd` (char \*arg, const struct `database` \*db)  
*Finds and prints out the educations whose name contains the given argument.*
- struct `education` `recommendCmd` (struct `profile` \*user, const struct `database` \*database)  
*Goes trough the available educations and compares them to the user: Both their interests, qualifications and location are considered.*
- int `isQualified` (struct `profile` user, struct `education` education)  
*Checks if the user has the subject levels required by the education.*
- const char \* `getRegionName` (enum `region` r)  
*Returns the name of the region as a string.*
- void `printEducation` (struct `education`)
- void `saveCmd` (struct `profile` \*user, struct `education` \*current\_education)  
*Saves the given education to a list in the profile struct.*
- int `getIndex` (char edu\_array[EDUCATION\_LIST\_LENGTH][MAX\_EDU\_NAME\_LENGTH], struct `education` target)  
*Returns the index of the given target in the array.*
- int `getEmptyIndex` (char edu\_array[EDUCATION\_LIST\_LENGTH][MAX\_EDU\_NAME\_LENGTH])  
*Returns an index with an empty string in the given array.*
- int `listIsFull` (int i)  
*A logical statement that returns a boolean value.*
- void `clearBuffer` (void)  
*Empties the buffer for standard input.*
- void `listCmd` (const struct `profile` \*user)  
*Prints out the names of all the saved educations.*
- void `deleteCmd` (struct `profile` \*user, int deleted\_entry)  
*Removes the name of the education at the given index.*
- void `saveProfile` (struct `profile` user)  
*Saves a file with the information collected about the user.*
- int `checkProfile` (const char name[ ])
- struct `profile` `loadProfile` (char \*name, int number\_of\_interests)  
*Loads a user profile from a generated <name>\_profile.txt file.*

### 8.1.1 Detailed Description

Contains functions related to command handling.

Contains all of the functions used for handling commands, such as those relating to verifying input and the functions that act on receiving a command.

## 8.1.2 Function Documentation

### 8.1.2.1 chooseFromList()

```
void chooseFromList (
    struct profile * user,
    int interval_start,
    int interval_end )
```

Saves the levels of chosen subjects to user.

#### Parameters

<i>user</i>	The profile struct where the qualifications should be saved
<i>interval_start</i>	The start of the interval for the qualifications in the list
<i>interval_end</i>	The end of the interval for the qualifications in the list

Definition at line 257 of file commands.c.

```
257                                     {
258     int temp_subject, i = 0, scan_res, length_string;
259     char temp_char;
260     char temp_string[MAX_INPUT_LENGTH];
261
262     fgets(temp_string, MAX_INPUT_LENGTH - 1, stdin);
263     length_string = strlen(temp_string);
264
265     if(length_string < 2)
266         return;
267
268
269     do{
270         scan_res = sscanf(temp_string + i, " %d%c", &temp_subject, &temp_char);
271         if(temp_subject >= 0 && temp_subject < (interval_end - interval_start + 1) &&
levelAsValue(temp_char) != -1 && scan_res == 2){
272             user->qualifications.subjects[temp_subject + interval_start].
level = levelAsValue(temp_char);
273             i += 1;
274             while(isalnum(*(temp_string + ++i)) == 0);
275         }
276     } while(i < length_string && scan_res != 0);
277 }
```

### 8.1.2.2 classNameStr()

```
const char * classNameStr (
    enum class class )
```

Returns the name as a string of a class given as an enum class.

#### Parameters

<i>class</i>	The enum value the name should return for
--------------	---

Definition at line 201 of file commands.c.

```

201                                     {
202     char *classes[TOTAL_SUBJECTS + USELESS_SUBJECTS] = {"MATHEMATICS", "CHEMISTRY", "BIOLOGY", "PHYSICS", "
    ENGLISH",
203                                     "BIOTECHNOLOGY", "GEOSCIENCE", "HISTORY", "IDEA_HISTORY",
204                                     "INFORMATICS", "INTERNATIONAL_ECONOMICS", "COMMUNICATION_AND_IT",
205                                     "RELIGION", "SOCIALSTUDIES", "BUSINESS_ECONOMICS", "
    CONTEMPORAY_HISTORY",
206                                     "FRENCH", "SPANISH", "GERMAN", "CHINESE", "ARABIC", "GREEK", "ITALIAN"
207     ,
208     return classes[class];
209 }
```

### 8.1.2.3 convertScale()

```
double convertScale (
    int v )
```

Returns the converted value.

#### Parameters

<i>v</i>	The value to be converted
----------	---------------------------

#### Returns

A double value between -1 and 1 given that the input is between 0 and 10

Definition at line 99 of file commands.c.

```

99                                     {
100     return (((double) v - 5.0) / 5.0);
101 }
```

### 8.1.2.4 deleteCmd()

```
void deleteCmd (
    struct profile * user,
    int deleted_entry )
```

Removes the name of the education at the given index.

#### Parameters

<i>user</i>	The profile struct for the user
-------------	---------------------------------

Definition at line 550 of file commands.c.

```

550                                     {
551     strcpy(user->saved_educations[validScaleValue(deleted_entry, 0, EDUCATION_LIST_LENGTH)],
552     "");

```

#### 8.1.2.5 evalCmd()

```

void evalCmd (
    struct profile * user,
    struct education * current_education,
    int arg )

```

Changes the adjustment vector for the user to approach the current education.

The distance of the change is determined by the argument

##### Parameters

<i>current_education</i>	The education currently being displayed
<i>user</i>	The profile struct whose adjustment vector is changed
<i>arg</i>	The user input argument for how much to change the adjustment vector

Definition at line 348 of file commands.c.

```

348                                     {
349     struct vector user_vector = addVector(user->interests, user->adjustment_vector);
350     struct vector distance_vector = subtractVector(current_education->interests,
351     user_vector);
352     struct vector scale_vector = scaleVector(distance_vector, ADJUSTMENT_CONSTANT *
353     convertScale(arg));
354     user->adjustment_vector = addVector(user->adjustment_vector, scale_vector);
355     freeVectorM(3, user_vector, distance_vector, scale_vector);
356 }

```

#### 8.1.2.6 findCmd()

```

struct education findCmd (
    char * arg,
    const struct database * db )

```

Finds and prints out the education with the exact name given as and argument.

##### Parameters

<i>arg</i>	The argument string which should be the name of an education
<i>database</i>	The database in which all educations are stored

**Returns**

A struct for the education found

Definition at line 303 of file commands.c.

```

303                                     {
304     int i, edu_found = 0;
305     struct education edu;
306     for(i = 0; i < db->amount_of_educations; i++){
307         if(strcmp(arg, db->educations[i].name) == 0){
308             edu = db->educations[i];
309             edu_found = 1;
310         }
311     }
312     if(edu_found)
313         printEducation(edu);
314     else
315         printf("No education exists by that name\n");
316     return edu;
317 }
```

**8.1.2.7 getEmptyIndex()**

```

int getEmptyIndex (
    char edu_array[EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH] )
```

Returns an index with an empty string in the given array.

**Parameters**

<i>edu_array</i>	An array of strings in which the empty string should be found
------------------	---

Definition at line 495 of file commands.c.

```

495                                     {
496     int i = 0;
497     int index = NO_EMPTY_INDEX;
498
499     for(i = 0; index == NO_EMPTY_INDEX && i < EDUCATION_LIST_LENGTH; i++){
500         if(strcmp(edu_array[i], "") == 0){
501             index = i;
502         }
503     }
504
505     return index;
506 }
```

**8.1.2.8 getIndex()**

```

int getIndex (
    char edu_array[EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH],
    struct education target )
```

Returns the index of the given target in the array.

## Parameters

<i>edu_array</i>	An array of strings
<i>target</i>	An education whose name is to be found in the array

Definition at line 478 of file commands.c.

```

478                                     {
479     int i = 0;
480     int index = NOT_IN_LIST;
481
482     for(i = 0; index == NOT_IN_LIST && i < EDUCATION_LIST_LENGTH; i++){
483         if(strcmp(edu_array[i], target.name) == 0){
484             index = i;
485         }
486     }
487
488     return index;
489 }
```

## 8.1.2.9 getRegionName()

```

const char * getRegionName (
    enum region r )
```

Returns the name of the region as a string.

## Parameters

<i>r</i>	The enum region value of the region to be returned as a string
----------	--

Definition at line 438 of file commands.c.

```

438                                     {
439     switch(r){
440         case NORTH_JUTLAND:
441             return "North Jutland";
442         case CENTRAL_JUTLAND:
443             return "Central Jutland";
444         case SOUTHERN_DENMARK:
445             return "Southern Denmark";
446         case ZEALAND:
447             return "Zealand";
448         case CAPITAL_REGION:
449             return "Capital Region";
450         default:
451             return "Region Not Found";
452     }
453 }
```

## 8.1.2.10 isQualified()

```

int isQualified (
    struct profile user,
    struct education education )
```

Checks if the user has the subject levels required by the education.

**Parameters**

<i>user</i>	The profile struct whose qualification is checked
<i>education</i>	The education struct with the requirements

**Returns**

0 if the user does not have the required levels and 1 if the user does

Definition at line 400 of file commands.c.

```

400                                     {
401     int i;
402     struct subject subject;
403     for(i = 0; i < education.required_qualifications.amount_of_subjects; i++){
404         subject = education.required_qualifications.subjects[i];
405         if(user.qualifications.subjects[subject.name].level <
subject.level)
406             return 0;
407     }
408     return 1;
409 }
```

**8.1.2.11 levelAsValue()**

```

enum level levelAsValue (
    char c )
```

Returns the enum value of a level given as a character.

**Parameters**

<i>c</i>	The level as a character to be converted to enum level
----------	--

Definition at line 215 of file commands.c.

```

215                                     {
216     enum level return_value = -1;
217
218     switch(c){
219         case 'A': case 'a':
220             return_value = A;
221             break;
222         case 'B': case 'b':
223             return_value = B;
224             break;
225         case 'C': case 'c':
226             return_value = C;
227             break;
228         case 'Z': case 'z':
229             return_value = Z;
230             break;
231         default:
232             return_value = -1;
233     }
234     return return_value;
235 }
```



## 8.1.2.12 listCmd()

```
void listCmd (
    const struct profile * user )
```

Prints out the names of all the saved educations.

## Parameters

<i>user</i>	The profile struct for the user
-------------	---------------------------------

Definition at line 531 of file commands.c.

```
531                                     {
532     int i, counter = 0;
533
534     printf("\nList of saved educations:\n");
535     for(i = 0; i < EDUCATION_LIST_LENGTH; i++){
536         if(strcmp(user->sav_educations[i], "") != 0){
537             printf("%2d: %s\n", i, user->sav_educations[i]);
538             counter++;
539         }
540     }
541
542     if(counter == 0)
543         printf("No entries yet\n\n");
544 }
```

## 8.1.2.13 listIsFull()

```
int listIsFull (
    int i )
```

A logical statement that returns a boolean value.

## Parameters

<i>i</i>	The index of an array of education structs 1 if the index is -1 and 0 otherwise
----------	---

Definition at line 513 of file commands.c.

```
513                                     {
514     return i == NO_EMPTY_INDEX;
515 }
```

## 8.1.2.14 loadProfile()

```
struct profile loadProfile (
    char * name,
    int number_of_interests )
```

Loads a user profile from a generated <name>\_profile.txt file.

## Parameters

<i>char</i>	*name The name of the user
<i>int</i>	number_of_interests The number of interests which is a member of the database struct

## Returns

struct profile Returns a user profile

Definition at line 615 of file commands.c.

```

615                                     {
616     int i;
617     FILE *file_pointer;
618     struct profile user;
619     char file_name[MAX_FILE_NAME_LENGTH];
620     char version[MAX_FILE_NAME_LENGTH];
621     char buffer[MAX_INPUT_LENGTH] = "Ingenting";
622
623     sprintf(file_name, "%s_profil.txt", name);
624
625     file_pointer = fopen(file_name, "r");
626
627     if(file_pointer == NULL){
628         printf("File could not be opened");
629         exit(EXIT_FAILURE);
630     }
631
632     user = createProfile(number_of_interests);
633
634     fscanf(file_pointer, "%s %s %f %d %f\n", version, user.name, &user.average, &user.location.region, &
user.location.region_importance);
635
636     fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
637     for (i = 0; i < EDUCATION_LIST_LENGTH; i++){
638         fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
639         sscanf(buffer, "%[^\\n]s", user.saved_educations[i]);
640     }
641
642     fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
643     for (i = 0; i < EDUCATION_LIST_LENGTH; i++){
644         fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
645         sscanf(buffer, "%[^\\n]s", user.recommended_educations[i]);
646     }
647
648     fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
649     for (i = 0; i < user.interests.size; i++){
650         fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
651         sscanf(buffer, "%lf", &user.interests.array[i]);
652     }
653
654     fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
655     for (i = 0; i < user.adjustment_vector.size; i++){
656         fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
657         sscanf(buffer, "%lf", &user.adjustment_vector.array[i]);
658     }
659
660     fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
661     for (i = 0; i < user.qualifications.amount_of_subjects; i++){
662         fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
663         sscanf(buffer, "%d", &user.qualifications.subjects[i].level);
664     }
665
666     fclose(file_pointer);
667
668     printf("Profile successfully loaded.\\n\\n");
669
670     return user;
671 }
```

## 8.1.2.15 recommendCmd()

```
struct education recommendCmd (
    struct profile * user,
    const struct database * database )
```

Goes through the available educations and compares them to the user: Both their interests, qualifications and location are considered.

## Parameters

<i>user</i>	The profile struct which is compared
<i>database</i>	The database containing the educations

## Returns

A struct for the recommended education.

Definition at line 365 of file commands.c.

```
365                                     {
366     int i;
367     struct vector normalized_vector;
368     double highest_result = -3.0, result = 0.0;
369     struct education best_fit;
370     normalized_vector = normalizeVector(addVector(user->interests, user->
adjustment_vector));
371
372     for(i = 0; i < database->amount_of_educations; i++){
373         result = dotProduct(database->educations[i].interests, normalized_vector) +
374             (1.0 - (double) abs(user->location.region - database->
educations[i].region)) *
375             user->location.region_importance;
376         if(result > highest_result && isQualified(*user, database->
educations[i]) &&
377             getIndex(user->recommended_educations, database->educations[i]) == NOT_IN_LIST
378         ){
379             highest_result = result;
380             best_fit = database->educations[i];
381         }
382     }
383     freeVector(normalized_vector);
384
385     strcpy(user->recommended_educations[user->last_recommended], best_fit.name);
386     user->last_recommended = (user->last_recommended + 1) % EDUCATION_LIST_LENGTH;
387
388     printf("\nThe recommended education is:");
389     printEducation(best_fit);
390
391     return best_fit;
392 }
```

## 8.1.2.16 saveCmd()

```
void saveCmd (
    struct profile * user,
    struct education * current_education )
```

Saves the given education to a list in the profile struct.

## Parameters

<i>current_education</i>	A pointer to an education
<i>user</i>	The profile struct of the user in which the education is saved

Definition at line 460 of file commands.c.

```

460                                     {
461     int i;
462
463     i = getEmptyIndex(user->savd_educations);
464
465     if(getIndex(user->savd_educations, *current_education) != -1)
466         printf("Already in list\n");
467     else if(listIsFull(i))
468         printf("List empty. Use delete to delete entries\n");
469     else
470         strcpy(user->savd_educations[i], current_education->name);
471 }
```

## 8.1.2.17 saveProfile()

```

void saveProfile (
    struct profile user )
```

Saves a file with the information collected about the user.

## Parameters

<i>user</i>	The profile struct for the user
-------------	---------------------------------

Definition at line 558 of file commands.c.

```

558                                     {
559     FILE *file_pointer;
560     int i;
561     char file_name[MAX_FILE_NAME_LENGTH];
562     sprintf(file_name, "%s_profil.txt", user.name);
563
564     file_pointer = fopen(file_name, "w");
565
566     if(file_pointer != NULL){
567         fprintf(file_pointer, "%s %s %f %d %f\n", VERSION, user.name, user.average, user.location.region,
568             user.location.region_importance);
569
570         fprintf(file_pointer, "Saved\n");
571         for (i = 0; i < EDUCATION_LIST_LENGTH; i++)
572             fprintf(file_pointer, "%s\n", user.saved_educations[i]);
573
574         fprintf(file_pointer, "Recommend\n");
575         for (i = 0; i < EDUCATION_LIST_LENGTH; i++)
576             fprintf(file_pointer, "%s\n", user.recommended_educations[i]);
577
578         fprintf(file_pointer, "Interests\n");
579         for (i = 0; i < user.interests.size; i++)
580             fprintf(file_pointer, "%f\n", user.interests.array[i]);
581
582         fprintf(file_pointer, "Adjustment\n");
583         for (i = 0; i < user.adjustment_vector.size; i++)
584             fprintf(file_pointer, "%f\n", user.adjustment_vector.array[i]);
585
586         fprintf(file_pointer, "Qualifications\n");
587         for(i = 0; i < TOTAL_SUBJECTS; i++)
```

```

587         fprintf(file_pointer, "%d\n", user.qualifications.subjects[i].
    level);
588     } else{
589         printf("File could not be opened");
590         exit(EXIT_FAILURE);
591     }
592     fclose(file_pointer);
593
594     printf("File saved successfully\n\n");
595 }

```

#### 8.1.2.18 searchCmd()

```

void searchCmd (
    char * arg,
    const struct database * db )

```

Finds and prints out the educations whose name contains the given argument.

##### Parameters

<i>arg</i>	The argument string which should be contained in the name of an education
<i>database</i>	The database in which all educations are stored.

Definition at line 325 of file commands.c.

```

325                                     {
326     int i, edu_found = 0;
327     struct education edu;
328
329     for(i = 0; i < db->amount_of_educations; i++){
330         if(strstr(db->educations[i].name, arg) != NULL) {
331             printf("    %s\n", db->educations[i].name);
332             edu_found = 1;
333         }
334     }
335     if(edu_found)
336         printf("Use the \"find\" command to look up your desired education\n");
337     else
338         printf("No education exists by that name\n");
339 }

```

#### 8.1.2.19 setImportantSubjects()

```

void setImportantSubjects (
    struct profile * user )

```

Saves all the qualifications for the important subjects.

##### Parameters

<i>user</i>	The profile struct where the subjects are saved to
-------------	--

Definition at line 183 of file commands.c.

```

183                                     {
184     char temp_char;
185     int i;
186
187     for(i = 0; i < IMPORTANT_SUBJECTS; i++){
188         printf("%s:%s ", classNameStr(i), (int) (FIELD_SIZE - strlen(
189             classNameStr(i))), "");
189         do{
190             scanf(" %c", &temp_char);
191             while(levelAsValue(temp_char) == -1);
192             user->qualifications.subjects[i].level = levelAsValue(temp_char);
193             clearBuffer();
194         }
195     }

```

#### 8.1.2.20 setOtherSubjects()

```

void setOtherSubjects (
    struct profile * user,
    int start,
    int end )

```

Saves all the levels of the other subjects (not the important ones)

##### Parameters

<i>user</i>	The profile struct where the qualifications are to be saved
<i>start</i>	The start of the subjects to be asked for
<i>end</i>	The ens of the subjects to be asked for

Definition at line 243 of file commands.c.

```

243                                     {
244     int i;
245
246     for(i = 0; i < end - start; i++)
247         printf("%d: %s\n", i, classNameStr(i + start));
248     chooseFromList(user, start, end);
249 }

```

#### 8.1.2.21 setProfileInterests()

```

void setProfileInterests (
    struct profile * user,
    const struct database * db )

```

Saves all interests to user as a converted value (see convertScale)

##### Parameters

<i>user</i>	The profile struct where the interests are saved to
<i>db</i>	The database struct where information about all interests are saved as a pointer

Definition at line 136 of file commands.c.

```

136                                     {
137     int i;
138
139     printf("Next, a series of interests will be shown\n"
140           "You are to give a value between 0 and 10, "
141           "where 0 is negative and 10 is positive towards the interest\n");
142
143     for(i = 0; i < db->amount_of_interests; i++){
144         printf("%s:%s ", db->interest_string[i], (int) (FIELD_SIZE - strlen(db->
145         interest_string[i])), "");
146         user->interests.array[i] = convertScale(validScaleValue(
147         getValidInteger(), 0, 10));
148     }
149     printf("\n\n");
150 }
```

#### 8.1.2.22 setProfileLocation()

```

void setProfileLocation (
    struct profile * user )
```

Sets the region of choice in user.

Saves the interest in studying in this location

##### Parameters

<i>user</i>	The profile struct where the information about location should be saved
-------------	---

Definition at line 81 of file commands.c.

```

81                                     {
82     int i;
83
84     printf("Where do you want to study?\n");
85     for(i = 0; i < NUMBER_OF_REGIONS; i++){
86         printf("%d: %s ", i, getRegionName(i));
87         printf("\n");
88         user->location.region = validScaleValue(getValidInteger(), 0,
89         NUMBER_OF_REGIONS - 1);
90
91         printf("How important is this region to you\n");
92         user->location.region_importance = (convertScale(validScaleValue(
93         getValidInteger(), 0, 10)) + 1.0) / 2.0;
94     }
95 }
```

#### 8.1.2.23 setProfileQualifications()

```

void setProfileQualifications (
    struct profile * user )
```

Saves all the users qualifications as given by the terminal.

**Parameters**

<i>user</i>	The profile struct where the qualifications are saved to
-------------	--

Definition at line 154 of file commands.c.

```

154                                     {
155     setSubjects(user);
156
157     printf("Your qualifications regarding subjects from high school will now be tested\n"
158           "Give a level from A, B, C or Z if you have not had the subject\n");
159
160     setImportantSubjects(user);
161
162     printf("Now some less relevant subjects will be \n");
163
164     setOtherSubjects(user, IMPORTANT_SUBJECTS, IMPORTANT_SUBJECTS + OTHER_SUBJECTS);
165     setOtherSubjects(user, IMPORTANT_SUBJECTS + OTHER_SUBJECTS, TOTAL_SUBJECTS);
166 }
```

**8.1.2.24 setSubjects()**

```

void setSubjects (
    struct profile * user )
```

Sets all qualifications in user to match the enum class.

**Parameters**

<i>user</i>	The profile struct where the subjects are saved to
-------------	--

Definition at line 172 of file commands.c.

```

172                                     {
173     int i;
174
175     for(i = 0; i < TOTAL_SUBJECTS; i++)
176         user->qualifications.subjects[i].name = i;
177 }
```

**8.1.2.25 surveyCmd()**

```

void surveyCmd (
    struct profile * user,
    const struct database * db )
```

Tests the current user for name, location, interests, qualifications and average grade.

**Parameters**

<i>user</i>	The profile struct where all survey results are saved
<i>db</i>	The database where information of interests and subjects are as a pointer



Definition at line 40 of file commands.c.

```

40                                     {
41     char name[MAX_NAME_LENGTH], choice;
42
43     /* Introduction */
44     printf("This survey will ask you several questions about interests, qualifications and grades\n"
45           "The survey requires answers in numbers (integers), and where scale is part, a value between 1
46           and 100\n\n");
47
48     /* Scan for profile name */
49     printf("Profile name: ");
50     scanf("%s", name);
51     if(checkProfile(name) == 1){
52         printf("Profile name is in use. Stop survey? (Y/N)\n");
53         scanf(" %c", &choice);
54         if(choice == 'Y' || choice == 'y')
55             return;
56     }
57     strcpy(user->name, name);
58
59     /* Get location and assesment */
60     setProfileLocation(user);
61
62     /* Get all interests */
63     setProfileInterests(user, db);
64
65     /* Get all qualifications */
66     setProfileQualifications(user);
67
68     /* Get average grade */
69     printf("What is your average grade? ");
70     user->average = getValidDouble();
71
72     /* Ending the survey */
73     printf("The survey has now concluded. Returning to menu...\n\n");
74 }

```

#### 8.1.2.26 validScaleValue()

```

int validScaleValue (
    int value,
    int interval_start,
    int interval_end )

```

Returns a value between `interval_start` and `interval_end`.

If the given value outside the interval it will return the value inside the interval closest the value. The `interval_start` must be less than the `interval_end`

##### Parameters

<i>value</i>	The value to check within the scale
<i>interval_start</i>	The start value of the scale
<i>interval_end</i>	The end value the scale

Definition at line 111 of file commands.c.

```

111                                     {
112     return (value > interval_end ? interval_end : (value < interval_start ? interval_start : value));
113 }

```

## 8.2 include/constants.h File Reference

Contains symbolic constants used throughout the program.

### 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)`
- `#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"`

### 8.2.1 Detailed Description

Contains symbolic constants used throughout the program.

This header-file contains all of the symbolic constants used throughout the entire program, such as those relating to the number of regions, the max length of strings or constants used for string formatting.

## 8.3 include/database.h File Reference

Contains elements relating to the database.

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

### Classes

- struct [database](#)

## Functions

- void **freeDatabase** (struct [database](#) \*)
- struct [database](#) \* **createDatabase** (char \*)
- struct [education](#) \* **findEducation** (char \*, struct [database](#) \*)

### 8.3.1 Detailed Description

Contains elements relating to the database.

Contains the database struct and functions for creating, freeing and finding educations.

## 8.4 include/education.h File Reference

Contains elements relating to educations.

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

## 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](#) \*)

### 8.4.1 Detailed Description

Contains elements relating to educations.

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

### 8.4.2 Function Documentation

#### 8.4.2.1 createArrayOfEducations()

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

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

## Parameters

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

Definition at line 54 of file education.c.

```

54                                     {
55     struct education* educations;
56     educations = (struct education*) calloc(amount_of_educations, sizeof(struct
education));
57
58     if(educations == NULL){
59         printf("Failed to allocate memory for educations.\n");
60         exit(EXIT_FAILURE);
61     }
62
63     return educations;
64 }
```

## 8.4.2.2 createDefaultEducation()

```

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

Assigns default values to the fields of the education struct.

## Parameters

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

Definition at line 16 of file education.c.

```

16                                     {
17     struct education education;
18     char *temp_name = "Nothing";
19     char *temp_desc = "No education selected";
20     char *temp_link = "No education link";
21
22     education.name = (char *) calloc(strlen(temp_name) + 1, sizeof(char));
23     education.description = (char *) calloc(strlen(temp_desc) + 1, sizeof(char));
24     education.link = (char *) calloc(strlen(temp_link) + 1, sizeof(char));
25     education.region = NORTH_JUTLAND;
26     education.required_grade = 0.0;
27     education.interests = createVector(amount_of_interests);
28     education.required_qualifications = createQualifications(amount_of_subjects);
29
30     strcpy(education.name, temp_name);
31     strcpy(education.description, temp_desc);
32     strcpy(education.link, temp_link);
33
34     return education;
35 }
```

## 8.5 include/parser.h File Reference

Contains elements relating to parsing the database.

```
#include <stdio.h>
#include <stdlib.h>
#include "database.h"
#include "region.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.*
- 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.*
- 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.*

### 8.5.1 Detailed Description

Contains elements relating to parsing the database.

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

## 8.5.2 Function Documentation

### 8.5.2.1 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
--------------------------	---

Definition at line 56 of file parser.c.

```
56                                     {
57     char **strings;
58     strings = (char **) calloc(amount_of_strings, sizeof(char*));
59
60     if(strings == NULL){
61         printf("Failed to allocate memory for array of strings.\n");
62         exit(EXIT_FAILURE);
63     }
64
65     return strings;
66 }
```

### 8.5.2.2 findDatabaseLine()

```
void findDatabaseLine (
    const char key[],
    FILE * filereader,
    char * current_line )
```

Search the database until the first word of a line matches with key.

#### Parameters

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

Definition at line 74 of file parser.c.

```
74                                     {
75     int found = 0;
76     char temp_string[STRING_MAX_LENGTH];
77
78     while(!found && fgets(current_line, STRING_MAX_LENGTH, filereader) != NULL){
```

```

79         sscanf(current_line, "%[^\n ]s", temp_string);
80         if(strcmp(temp_string, key) == 0){
81             /*printf("%s WAS FOUND.....\n", key);*/
82             found = 1;
83         }
84     }
85
86     /* Return default string if a line with key does not exist */
87     if(found == 0)
88         strcpy(current_line, NOT_FOUND_STRING);
89
90 }

```

### 8.5.2.3 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

Definition at line 16 of file parser.c.

```

16
17     /* This will contain the first line where the type of database and encoding is read. */
18     char database_format[STRING_MAX_LENGTH];
19     char lines_to_read[STRING_MAX_LENGTH] = {"NAME", "DESC", "LINK",
20                                             "LOCATION", "REQUIREMENTS",
21                                             "REQUIRED GRADE", "INTERESTS"};
22
23     char amount_of_lines_to_read = sizeof(lines_to_read) / sizeof(lines_to_read[0]);
24     int i;
25
26     /* This line holds the type of database and its character encoding. */
27     findDatabaseLine("EDU", filereader, database_format);
28
29     /* Guard to make sure the file is an EDU file */
30     if(strcmp(database_format, NOT_FOUND_STRING) == 0){
31         printf("Error in parseDatabase: The file is not a file of format EDU.");
32         return;
33     }
34
35     database->amount_of_educations = parseNumOfEdu(filereader);
36     database->educations = createArrayOfEducations(database->
amount_of_educations);
37
38     database->amount_of_interests = parseNumOfInterests(filereader);
39     database->interest_string = createArrayOfStrings(database->
amount_of_interests);
40
41     /* Allocate memory for interest vectors in all educations */
42     for(i = 0; i < database->amount_of_educations; i++){
43         database->educations[i].interests = createVector(database->
amount_of_interests);
44     }
45
46     /* Parse all lines from lines_to_read */
47     for(i = 0; i < amount_of_lines_to_read; i++){
48         parseDatabaseLine(lines_to_read[i], database, filereader);
49     }
50 }

```

### 8.5.2.4 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

Definition at line 98 of file parser.c.

```

98                                     {
99     char current_line[STRING_MAX_LENGTH];
100
101     findDatabaseLine(key, filereader, current_line);
102
103     /* Guard to make sure a line with key exists. If it does not, reset pointer in file and return */
104     if(strcmp(current_line, NOT_FOUND_STRING) == 0){
105         printf("An error has occured: Tried to parse line with %s, but entry does not exist in database.\n
106 \n", key);
107         rewind(filereader);
108         return;
109     }
110
111     if(strcmp(key, "NAME") == 0){
112         parseEduNames(database->amount_of_educations, database->
113 educations, current_line);
114     } else if(strcmp(key, "DESC") == 0){
115         parseEduDesc(database->amount_of_educations, database->
116 educations, current_line);
117     } else if(strcmp(key, "LINK") == 0){
118         parseEduLink(database->amount_of_educations, database->educations, current_line);
119     } else if(strcmp(key, "LOCATION") == 0){
120         parseEduRegion(database->amount_of_educations, database->
121 educations, current_line);
122     } else if(strcmp(key, "REQUIREMENTS") == 0){
123         parseSubReq(database->amount_of_educations, database->
124 educations, current_line);
125     } else if(strcmp(key, "REQUIRED GRADE") == 0){
126         parseReqGrade(database->amount_of_educations, database->
127 educations, current_line);
128     } else if(strcmp(key, "INTERESTS") == 0){
129         /* First parse all the names of the interests */
130         parseInterestNames(database, filereader);
131
132         /* Rewind the file pointer and find the correct line again */
133         rewind(filereader);
134         findDatabaseLine(key, filereader, current_line);
135
136         parseInterestValues(database->amount_of_interests,
137                             database->amount_of_educations,
138                             database->educations,
139                             filereader);
140     } else{
141         printf("An error has occured: Attempting to parse %s, but no parsing functions exist.\n\n", key);
142     }
143
144     rewind(filereader);
145 }
```



### 8.5.2.5 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

Definition at line 341 of file parser.c.

```
341                                     {
342     int i;
343     int offset = 0;
344
345     /* Iterate through all educations */
346     for(i = 0; i < amount_of_educations; i++){
347         educations[i].description = parseEduString(current_line, amount_of_educations, offset
348     );
349         offset += strlen(educations[i].description) + 1;
350     }
```

### 8.5.2.6 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

Definition at line 324 of file parser.c.

```
324                                     {
325     int i;
326     int offset = 0;
327
328     /* Iterate through all educations and assign names */
```

```

329     for(i = 0; i < amount_of_educations; i++){
330         educations[i].name = parseEduString(current_line, amount_of_educations, offset);
331         offset += strlen(educations[i].name) + 1;
332     }
333 }

```

### 8.5.2.7 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

Definition at line 256 of file parser.c.

```

256                                     {
257     char *temp_region_string;
258     int i;
259     int offset = 0;
260
261     /* Iterate through all educations */
262     for(i = 0; i < amount_of_educations; i++){
263         temp_region_string = parseEduString(current_line, amount_of_educations, offset);
264         offset += strlen(temp_region_string) + 1;
265         educations[i].region = strToReg(temp_region_string);
266         free(temp_region_string);
267     }
268 }

```

### 8.5.2.8 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

Definition at line 375 of file parser.c.

```

375                                     {
376     char tmp_education_string[STRING_MAX_LENGTH];
377     char *education_string;
378
379     int tmp_education_string_length;
380     int i;
381
382     /* Calculate how many chars to skip. Will always skip the first word */
383     i = strchr(current_line, TABS) - current_line + sizeof(char) + offset;
384
385     sscanf(current_line + i, "%[^\n ]s", tmp_education_string);
386     tmp_education_string_length = strlen(tmp_education_string);
387     education_string = (char *) malloc((tmp_education_string_length + 1) * sizeof(char));
388     strcpy(education_string, tmp_education_string);
389
390     return education_string;
391 }
```

### 8.5.2.9 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

Definition at line 146 of file parser.c.

```

146                                     {
147     int i;
148     char current_line[STRING_MAX_LENGTH];
149     char temp_string[STRING_MAX_LENGTH];
150
151     for(i = 0; i < database->amount_of_interests; i++){
152         fgets(current_line, STRING_MAX_LENGTH, filereader);
153         sscanf(current_line, "%[^\n ]s", temp_string);
154         database->interest_string[i] = calloc(strlen(temp_string) + 1, sizeof(char));
155         strcpy(database->interest_string[i], temp_string);
156     }
157 }
```

### 8.5.2.10 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

Definition at line 184 of file parser.c.

```

184
185     {
186     char *temp_interest_value_string;
187     char current_line[STRING_MAX_LENGTH];
188     int offset;
189     int i;
190     int j;
191     struct vector temp_vector;
192
193     /* Iterate through all interests */
194     for(i = 0; i < amount_of_interests; i++){
195         offset = 0;
196         fgets(current_line, STRING_MAX_LENGTH, filereader);
197
198         /* Assign the interest values to each education */
199         for(j = 0; j < amount_of_educations; j++){
200             temp_interest_value_string = parseEduString(current_line, amount_of_educations,
201             offset);
202             educations[j].interests.array[i] = strtod(temp_interest_value_string, NULL);
203             offset += strlen(temp_interest_value_string) + 1;
204             free(temp_interest_value_string);
205         }
206     }
207
208     /* Normalize Values */
209     for(i = 0; i < amount_of_educations; i++){
210         temp_vector = normalizeVector(educations[i].interests);
211         freeVector(educations[i].interests);
212         educations[i].interests = temp_vector;
213     }
214 }

```

### 8.5.2.11 parseNumOfEdu()

```

int parseNumOfEdu (
    FILE * filereader )

```

Returns the number of educations from database file.

## Parameters

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

Definition at line 296 of file parser.c.

```

296
297     {
298     int number_of_educations = 0;
299     char current_line[STRING_MAX_LENGTH];
300     int line_length;
301     int i;
302
303     findDatabaseLine("NAME", filereader, current_line);
304     line_length = strlen(current_line);
305 }

```

```

304
305     /* Iterate through all educations */
306     for(i = 0; i < line_length; i++){
307         if(current_line[i] == TABS) {
308             number_of_educations++;
309         }
310     }
311
312     /* Reset file pointer */
313     rewind(filereader);
314
315     return number_of_educations;
316 }

```

#### 8.5.2.12 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
-------------------	-------------------

Definition at line 163 of file parser.c.

```

163
164     char current_line[STRING_MAX_LENGTH];
165     int number_of_interests = 0;
166
167     findDatabaseLine("INTERESTS", filereader, current_line);
168
169     while (fgets(current_line, STRING_MAX_LENGTH, filereader) != NULL){
170         number_of_interests++;
171     }
172
173     rewind(filereader);
174     return number_of_interests;
175 }

```

#### 8.5.2.13 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

Definition at line 236 of file parser.c.

```

236                                     {
237     char *temp_grade_string;
238     int i;
239     int offset = 0;
240
241     /* Iterate through all educations */
242     for(i = 0; i < amount_of_educations; i++){
243         temp_grade_string = parseEduString(current_line, amount_of_educations, offset);
244         offset += strlen(temp_grade_string) + 1;
245         educations[i].required_grade = strtod(temp_grade_string, NULL);
246         free(temp_grade_string);
247     }
248 }
```

#### 8.5.2.14 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

Definition at line 220 of file parser.c.

```

220                                     {
221     int i;
222
223     for(i = 0; i < amount_of_educations; i++){
224         educations[i].required_qualifications.subjects = (struct subject *) calloc(
225             EDU_MAX_SUBJECTS, sizeof(struct subject));
226         educations[i].required_qualifications.amount_of_subjects = 0;
227         readReqString(&(educations[i].required_qualifications), current_line, i + 1);
228     }
```

#### 8.5.2.15 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.

Definition at line 420 of file parser.c.

```

420
421     int i, subject_index=0, offset = 0, moreReqs = 1;
422     char reqClass[30];
423
424     /*Find the offset for the current education*/
425     for(i = 0; i < education_location; i++)
426         offset += sseek(string + offset, '\t') + 1;
427
428     do{
429         fflush(stdout);
430         qualification->amount_of_subjects += 1;
431
432         /*read the first requirement name*/
433         for(i = 0; string[offset + i] != ' ' && string[offset + i] != '\n' && string[offset + i] != '=' &&
string[offset + i] != '\t' && string[offset + i] != '_'; ++i)
434             reqClass[i] = string[offset + i];
435
436         reqClass[i] = '\0';
437
438         qualification->subjects[subject_index].name = stringToClass(reqClass);
439         if(qualification->subjects[subject_index].name == NONE){
440             qualification->subjects[subject_index].name = DANISH;
441             qualification->subjects[subject_index].level = Z;
442         }
443
444         if(string[offset + i] == '_'){
445             ++i;
446             qualification->subjects[subject_index].level =
charToLevel(string[offset + i]);
447             ++i;
448         } else{
449             qualification->subjects[subject_index].level = Z;
450         }
451
452         /*Check if there is more req to read*/
453         if(string[offset + i] == '\t' || string[offset + i] == '\n'){
454             moreReqs = 0;
455         }
456
457         ++subject_index;
458         offset += sseek(&string[offset], ' ') + 1;
459     } while(moreReqs);
460 }
461 }
```

## 8.5.2.16 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
----------------------	-----------------------

Definition at line 274 of file parser.c.

```

274                                     {
275     enum region region;
276
277     if(strcmp(region_string, "NORTH_JUTLAND") == 0){
278         region = NORTH_JUTLAND;
279     } else if(strcmp(region_string, "CENTRAL_JUTLAND") == 0){
280         region = CENTRAL_JUTLAND;
281     } else if(strcmp(region_string, "SOUTHERN_DENMARK") == 0){
282         region = SOUTHERN_DENMARK;
283     } else if(strcmp(region_string, "ZEALAND") == 0){
284         region = ZEALAND;
285     } else if(strcmp(region_string, "CAPITAL_REGION") == 0) {
286         region = CAPITAL_REGION;
287     }
288
289     return region;
290 }

```

## 8.6 include/profile.h File Reference

Contains elements relating to user profiles.

```

#include "vector.h"
#include "subjects.h"
#include "region.h"
#include "education.h"
#include "constants.h"

```

### 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.*

#### 8.6.1 Detailed Description

Contains elements relating to user profiles.

Contains the profile struct and the functions for creating, printing and deallocating user profiles.

#### 8.6.2 Function Documentation

##### 8.6.2.1 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
----------------------------	-----------------------------------

Definition at line 15 of file profile.c.

```

15                                     {
16     struct profile profile;
17     int i;
18
19     profile.interests = createVector(number_of_interests);
20     profile.adjustment_vector = createVector(number_of_interests);
21     profile.qualifications = createQualifications(TOTAL_SUBJECTS);
22     profile.average = 0.0;
23     profile.location.region = 0;
24     profile.location.region_importance = 0;
25     profile.last_recommended = 0;
26
27     profile.adjustment_vector.array[0] = 0.0001;
28
29     for(i = 0; i < EDUCATION_LIST_LENGTH; i++){
30         strcpy(profile.saved_educations[i], "");
31         strcpy(profile.recommended_educations[i], "");
32     }
33
34     for(i = 0; i < profile.qualifications.amount_of_subjects; i++)
35         profile.qualifications.subjects[i].name = i;
36
37     return profile;
38 }
```

## 8.6.2.2 freeProfile()

```

void freeProfile (
    struct profile p )
```

Frees the allocated memory for the given profile.

## Parameters

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

Definition at line 44 of file profile.c.

```

44                                     {
45     freeQualifications(&p.qualifications);
46     freeVector(p.interests);
47     freeVector(p.adjustment_vector);
48 }
```

## 8.6.2.3 printProfile()

```

void printProfile (
    struct profile p )
```

Prints information stored in the given profil.

## Parameters

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

Definition at line 54 of file profile.c.

```

54                                     {
55     int i;
56
57     printf("Name: %s\n", p.name);
58     printVector(p.interests);
59
60     for(i = 0; i < p.qualifications.amount_of_subjects; i++){
61         printf("%s %d\n", classNameStr(p.qualifications.subjects[i].name),
62                p.qualifications.subjects[i].level);
63     }
64     printf("Everything is fine\n");
65
66 }
```

## 8.7 include/region.h File Reference

Contains geographical elements.

### Classes

- struct [location](#)

### Enumerations

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

*Describes a region.*

#### 8.7.1 Detailed Description

Contains geographical elements.

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

#### 8.7.2 Enumeration Type Documentation

## 8.7.2.1 region

enum [region](#)

Describes a region.

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

Definition at line 16 of file region.h.

```

16         {
17     NORTH_JUTLAND = 0,
18     CENTRAL_JUTLAND,
19     SOUTHERN_DENMARK,
20     ZEALAND,
21     CAPITAL_REGION
22 };

```

## 8.8 include/subjects.h File Reference

Contains code regarding subjects and qualifications for educations.

## 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.*

### 8.8.1 Detailed Description

Contains code regarding subjects and qualifications for educations.

Contains the enums for different classes and their levels. Also includes the subject and qualification structs and some related functions

### 8.8.2 Function Documentation

#### 8.8.2.1 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
-----------	---

Definition at line 99 of file subjects.c.

```
99  
100     enum level level = Z;    {  
101  
102     if(ch == 'A')  
103         level = A;  
104     else if(ch == 'B')  
105         level = B;  
106     else if(ch == 'C')  
107         level = C;  
108  
109     return level;  
110 }
```

#### 8.8.2.2 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
----------	--

Definition at line 116 of file subjects.c.

```

116                                     {
117     switch(1){
118     case 1:
119         return 'C';
120     case 2:
121         return 'B';
122     case 3:
123         return 'A';
124     default:
125         return 'Z';
126     }
127 }

```

### 8.8.2.3 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
---------------	--

Definition at line 32 of file subjects.c.

```

32                                     {
33     if (strcmp(string, "MATHEMATICS") == 0){
34         return MATHEMATICS;
35     }else if(strcmp(string, "CHEMISTRY") == 0){
36         return CHEMISTRY;
37     }else if(strcmp(string, "BIOLOGY") == 0){
38         return BIOLOGY;
39     }else if(strcmp(string, "PHYSICS") == 0){
40         return PHYSICS;
41     }else if(strcmp(string, "ENGLISH") == 0){
42         return ENGLISH;
43     }else if(strcmp(string, "DANISH") == 0){
44         return DANISH;
45     }else if(strcmp(string, "BIOTECHNOLOGY") == 0){
46         return BIOTECHNOLOGY;
47     }else if(strcmp(string, "GEOSCIENCE") == 0){
48         return GEOSCIENCE;
49     }else if(strcmp(string, "HISTORY") == 0){
50         return HISTORY;
51     }else if(strcmp(string, "IDEAHISTORY") == 0){
52         return IDEA_HISTORY;
53     }else if(strcmp(string, "INFORMATICS") == 0){
54         return INFORMATICS;
55     }else if(strcmp(string, "INTERNATIONALECONOMICS") == 0){
56         return INTERNATIONAL_ECONOMICS;
57     }else if(strcmp(string, "COMMUNICATIONANDIT") == 0){
58         return COMMUNICATION_AND_IT;
59     }else if(strcmp(string, "RELIGION") == 0){
60         return RELIGION;
61     }else if(strcmp(string, "SOCIALSTUDIES") == 0){
62         return SOCIALSTUDIES;
63     }else if(strcmp(string, "BUSINESSECONOMICS") == 0){
64         return BUSINESS_ECONOMICS;
65     }else if(strcmp(string, "CONTEMPORARYHISTORY") == 0){
66         return CONTEMPORARY_HISTORY;
67     }else if(strcmp(string, "FRENCH") == 0){
68         return FRENCH;
69     }else if(strcmp(string, "SPANISH") == 0){
70         return SPANISH;
71     }else if(strcmp(string, "GERMAN") == 0){
72         return GERMAN;
73     }else if(strcmp(string, "CHINESE") == 0){
74         return CHINESE;
75     }else if(strcmp(string, "ARABIC") == 0){

```

```

76     return ARABIC;
77 }else if(strcmp(string, "GREEK") == 0){
78     return GREEK;
79 }else if(strcmp(string, "ITALIAN") == 0){
80     return ITALIAN;
81 }else if(strcmp(string, "JAPANESE") == 0){
82     return JAPANESE;
83 }else if(strcmp(string, "LATIN") == 0){
84     return LATIN;
85 }else if(strcmp(string, "PORTUGESE") == 0){
86     return PORTUGESE;
87 }else if(strcmp(string, "RUSSIAN") == 0){
88     return RUSSIAN;
89 }
90
91 /*default*/
92 return NONE;
93 }

```

## 8.9 include/vector.h File Reference

Contains elements relating to vectors.

### 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)*

### 8.9.1 Detailed Description

Contains elements relating to vectors.

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

### 8.9.2 Function Documentation

#### 8.9.2.1 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

Definition at line 83 of file vector.c.

```
83                                     {
84     struct vector sum = createVector(v1.size);
85     int i;
86
87     for(i = 0; i < v1.size; i++)
88         sum.array[i] = v1.array[i] + v2.array[i];
89
90     return sum;
91 }
```

#### 8.9.2.2 copyVector()

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

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

##### Parameters

v	The input vector that is copied
---	---------------------------------

Definition at line 56 of file vector.c.

```

56                                     {
57     struct vector copy = createVector(v.size);
58     int i;
59
60     for(i = 0; i < v.size; i++)
61         copy.array[i] = v.array[i];
62
63     return copy;
64 }

```

### 8.9.2.3 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
-------------	--------------------------------------

Definition at line 12 of file vector.c.

```

12                                     {
13     struct vector vector;
14     vector.array = (double*)calloc(size, sizeof(double));
15
16     if(vector.array == NULL){
17         printf("Failed to allocate memory. Bye bye.\n");
18         exit(EXIT_FAILURE);
19     }
20     vector.size = size;
21
22     return vector;
23 }

```

### 8.9.2.4 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

Definition at line 150 of file vector.c.

```

150                                     {

```



```

151     double dot_product = 0;
152     int i;
153
154     for(i = 0; i < v1.size; i++)
155         dot_product += v1.array[i] * v2.array[i];
156
157     return dot_product;
158 }

```

### 8.9.2.5 freeVector()

```

void freeVector (
    struct vector v )

```

frees the dynamically allocated array on the heap

#### Parameters

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

Definition at line 47 of file vector.c.

```

47                                     {
48     free(v.array);
49 }

```

### 8.9.2.6 freeVectorM()

```

void freeVectorM (
    int num,
    ... )

```

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

#### Parameters

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

Definition at line 29 of file vector.c.

```

29                                     {
30     int i;
31     va_list list;
32
33     va_start(list, num);
34
35     for(i = 0; i < num; i++){
36         struct vector v = va_arg(list, struct vector);
37         freeVector(v);
38     }
39
40     va_end(list);
41 }

```

### 8.9.2.7 lengthOfVector()

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

Calculates and returns the length of the given vector.

#### Parameters

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

Definition at line 127 of file vector.c.

```
127                                     {
128     double sum = 0.0;
129     int i;
130
131     for(i = 0; i < v.size; i++)
132         sum += pow(v.array[i], 2);
133
134     return sqrt(sum);
135 }
```

### 8.9.2.8 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

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

Definition at line 141 of file vector.c.

```
141                                     {
142     return scaleVector(v, 1 / lengthOfVector(v));
143 }
```

### 8.9.2.9 printVector()

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

Prints a vector.

## Parameters

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

Definition at line 71 of file vector.c.

```

71                                     {
72     int i;
73
74     for(i = 0; i < v.size; i++)
75         printf("%f\n", v.array[i]);
76 }
```

8.9.2.10 `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

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

Definition at line 113 of file vector.c.

```

113                                     {
114     struct vector result = createVector(v.size);
115     int i;
116
117     for(i = 0; i < v.size; i++)
118         result.array[i] = v.array[i] * scale;
119
120     return result;
121 }
```

8.9.2.11 `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

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

Definition at line 98 of file vector.c.

```
98                                     {
99     struct vector sum = createVector(v1.size);
100     int i;
101
102     for(i = 0; i < v1.size; i++)
103         sum.array[i] = v1.array[i] - v2.array[i];
104
105     return sum;
106 }
```

# Index

addVector  
    vector.h, 57

charToLevel  
    subjects.h, 54

chooseFromList  
    commands.h, 21

classNameStr  
    commands.h, 21

commands.h  
    chooseFromList, 21  
    classNameStr, 21  
    convertScale, 22  
    deleteCmd, 22  
    evalCmd, 23  
    findCmd, 23  
    getEmptyIndex, 24  
    getIndex, 24  
    getRegionName, 25  
    isQualified, 25  
    levelAsValue, 26  
    listCmd, 26  
    listIsFull, 27  
    loadProfile, 27  
    recommendCmd, 28  
    saveCmd, 29  
    saveProfile, 30  
    searchCmd, 31  
    setImportantSubjects, 31  
    setOtherSubjects, 32  
    setProfileInterests, 32  
    setProfileLocation, 33  
    setProfileQualifications, 33  
    setSubjects, 34  
    surveyCmd, 34  
    validScaleValue, 35

convertScale  
    commands.h, 22

copyVector  
    vector.h, 57

createArrayOfEducations  
    education.h, 37

createArrayOfStrings  
    parser.h, 40

createDefaultEducation  
    education.h, 38

createProfile  
    profile.h, 50

createVector  
    vector.h, 58

CuString, 13

CuSuite, 13

CuTest, 14

Database, 14

database, 14

deleteCmd  
    commands.h, 22

dotProduct  
    vector.h, 58

education, 15

education.h  
    createArrayOfEducations, 37  
    createDefaultEducation, 38

evalCmd  
    commands.h, 23

findCmd  
    commands.h, 23

findDatabaseLine  
    parser.h, 40

freeProfile  
    profile.h, 51

freeVector  
    vector.h, 59

freeVectorM  
    vector.h, 59

getEmptyIndex  
    commands.h, 24

getIndex  
    commands.h, 24

getRegionName  
    commands.h, 25

include/commands.h, 19

include/constants.h, 36

include/database.h, 36

include/education.h, 37

include/parser.h, 38

include/profile.h, 50

include/region.h, 52

include/subjects.h, 53

include/vector.h, 56

isQualified  
    commands.h, 25

lengthOfVector  
    vector.h, 60

levelAsValue

- commands.h, 26
- levelToChar
  - subjects.h, 54
- listCmd
  - commands.h, 26
- listIsFull
  - commands.h, 27
- loadProfile
  - commands.h, 27
- location, 15
- normalizeVector
  - vector.h, 60
- parseDatabase
  - parser.h, 41
- parseDatabaseLine
  - parser.h, 41
- parseEduDesc
  - parser.h, 42
- parseEduNames
  - parser.h, 43
- parseEduRegion
  - parser.h, 44
- parseEduString
  - parser.h, 44
- parseInterestNames
  - parser.h, 45
- parseInterestValues
  - parser.h, 45
- parseNumOfEdu
  - parser.h, 46
- parseNumOfInterests
  - parser.h, 47
- parseReqGrade
  - parser.h, 47
- parseSubReq
  - parser.h, 48
- parser.h
  - createArrayOfStrings, 40
  - findDatabaseLine, 40
  - parseDatabase, 41
  - parseDatabaseLine, 41
  - parseEduDesc, 42
  - parseEduNames, 43
  - parseEduRegion, 44
  - parseEduString, 44
  - parseInterestNames, 45
  - parseInterestValues, 45
  - parseNumOfEdu, 46
  - parseNumOfInterests, 47
  - parseReqGrade, 47
  - parseSubReq, 48
  - readReqString, 48
  - strToReg, 49
- printProfile
  - profile.h, 51
- printVector
  - vector.h, 60
- profile, 16
- profile.h
  - createProfile, 50
  - freeProfile, 51
  - printProfile, 51
- qualification, 16
- readReqString
  - parser.h, 48
- recommendCmd
  - commands.h, 28
- region
  - region.h, 52
- region.h
  - region, 52
- saveCmd
  - commands.h, 29
- saveProfile
  - commands.h, 30
- scaleVector
  - vector.h, 61
- searchCmd
  - commands.h, 31
- setImportantSubjects
  - commands.h, 31
- setOtherSubjects
  - commands.h, 32
- setProfileInterests
  - commands.h, 32
- setProfileLocation
  - commands.h, 33
- setProfileQualifications
  - commands.h, 33
- setSubjects
  - commands.h, 34
- strToReg
  - parser.h, 49
- stringToClass
  - subjects.h, 55
- subject, 17
- subjects.h
  - charToLevel, 54
  - levelToChar, 54
  - stringToClass, 55
- subtractVector
  - vector.h, 61
- surveyCmd
  - commands.h, 34
- validScaleValue
  - commands.h, 35
- vector, 17
- vector.h
  - addVector, 57
  - copyVector, 57
  - createVector, 58
  - dotProduct, 58

freeVector, [59](#)  
freeVectorM, [59](#)  
lengthOfVector, [60](#)  
normalizeVector, [60](#)  
printVector, [60](#)  
scaleVector, [61](#)  
subtractVector, [61](#)