

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_educations	5
3.1.1.2	amount_of_interests	5
3.1.1.3	educations	6
3.1.1.4	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
3.3.2	Member Data Documentation	7
3.3.2.1	description	7
3.3.2.2	interests	7
3.3.2.3	link	7
3.3.2.4	name	7
3.3.2.5	region	7
3.3.2.6	required_grade	7

3.3.2.7	required_qualifications	8
3.4	location Struct Reference	8
3.4.1	Member Data Documentation	8
3.4.1.1	region	8
3.4.1.2	region_importance	8
3.5	profile Struct Reference	8
3.5.1	Detailed Description	9
3.5.2	Member Data Documentation	9
3.5.2.1	adjustment_vector	9
3.5.2.2	average	9
3.5.2.3	interests	9
3.5.2.4	last_recommended	9
3.5.2.5	location	10
3.5.2.6	name	10
3.5.2.7	qualifications	10
3.5.2.8	recommended_educations	10
3.5.2.9	saved_educations	10
3.6	qualification Struct Reference	10
3.6.1	Member Data Documentation	11
3.6.1.1	amount_of_subjects	11
3.6.1.2	subjects	11
3.7	subject Struct Reference	11
3.7.1	Member Data Documentation	11
3.7.1.1	level	11
3.8	vector Struct Reference	12
3.8.1	Member Data Documentation	12
3.8.1.1	array	12
3.8.1.2	size	12

4 File Documentation	13
4.1 AllTests.c File Reference	13
4.2 commands.c File Reference	13
4.2.1 Function Documentation	15
4.2.1.1 checkProfile()	15
4.2.1.2 chooseFromList()	15
4.2.1.3 classNameStr()	15
4.2.1.4 clearBuffer()	15
4.2.1.5 convertScale()	16
4.2.1.6 deleteCmd()	16
4.2.1.7 evalCmd()	16
4.2.1.8 findCmd()	17
4.2.1.9 getEmptyIndex()	17
4.2.1.10 getIndex()	17
4.2.1.11 getRegionName()	18
4.2.1.12 getValidDouble()	18
4.2.1.13 getValidInteger()	18
4.2.1.14 isQualified()	18
4.2.1.15 levelAsValue()	18
4.2.1.16 listCmd()	19
4.2.1.17 listIsFull()	19
4.2.1.18 loadProfile()	19
4.2.1.19 menuCmd()	19
4.2.1.20 printEducation()	20
4.2.1.21 recommendCmd()	20
4.2.1.22 saveCmd()	20
4.2.1.23 saveProfile()	20
4.2.1.24 searchCmd()	21
4.2.1.25 setImportantSubjects()	21
4.2.1.26 setOtherSubjects()	21

4.2.1.27	setProfileInterests()	22
4.2.1.28	setProfileLocation()	22
4.2.1.29	setProfileQualifications()	22
4.2.1.30	setSubjects()	22
4.2.1.31	surveyCmd()	24
4.2.1.32	validScaleValue()	24
4.3	commands.h File Reference	24
4.3.1	Detailed Description	26
4.3.2	Function Documentation	26
4.3.2.1	checkProfile()	26
4.3.2.2	chooseFromList()	26
4.3.2.3	classNameStr()	26
4.3.2.4	clearBuffer()	27
4.3.2.5	convertScale()	27
4.3.2.6	deleteCmd()	27
4.3.2.7	evalCmd()	28
4.3.2.8	findCmd()	28
4.3.2.9	getEmptyIndex()	28
4.3.2.10	getIndex()	29
4.3.2.11	getRegionName()	29
4.3.2.12	getValidDouble()	29
4.3.2.13	getValidInteger()	29
4.3.2.14	isQualified()	29
4.3.2.15	levelAsValue()	30
4.3.2.16	listCmd()	30
4.3.2.17	listIsFull()	30
4.3.2.18	loadProfile()	31
4.3.2.19	menuCmd()	31
4.3.2.20	printEducation()	31
4.3.2.21	recommendCmd()	31

4.3.2.22	saveCmd()	32
4.3.2.23	saveProfile()	32
4.3.2.24	searchCmd()	32
4.3.2.25	setImportantSubjects()	32
4.3.2.26	setOtherSubjects()	33
4.3.2.27	setProfileInterests()	33
4.3.2.28	setProfileLocation()	33
4.3.2.29	setProfileQualifications()	34
4.3.2.30	setSubjects()	34
4.3.2.31	surveyCmd()	34
4.3.2.32	validScaleValue()	34
4.4	constants.h File Reference	35
4.4.1	Detailed Description	35
4.4.2	Macro Definition Documentation	36
4.4.2.1	ADJUSTMENT_CONSTANT	36
4.4.2.2	DATABASE_PATH	36
4.4.2.3	EDU_MAX_SUBJECTS	36
4.4.2.4	EDUCATION_LIST_LENGTH	36
4.4.2.5	FIELD_SIZE	36
4.4.2.6	IMPORTANT_SUBJECTS	36
4.4.2.7	LANGUAGE_SUBJECTS	36
4.4.2.8	MAX_COMMAND_LENGTH	37
4.4.2.9	MAX_EDU_NAME_LENGTH	37
4.4.2.10	MAX_FILE_NAME_LENGTH	37
4.4.2.11	MAX_INPUT_LENGTH	37
4.4.2.12	MAX_NAME_LENGTH	37
4.4.2.13	NO_EMPTY_INDEX	37
4.4.2.14	NOT_FOUND_STRING	37
4.4.2.15	NOT_IN_LIST	37
4.4.2.16	NUMBER_OF_REGIONS	38

4.4.2.17	OTHER_SUBJECTS	38
4.4.2.18	STRING_MAX_LENGTH	38
4.4.2.19	TABS	38
4.4.2.20	TOTAL_SUBJECTS	38
4.4.2.21	USELESS_SUBJECTS	38
4.4.2.22	VERSION	38
4.5	database.c File Reference	39
4.5.1	Function Documentation	39
4.5.1.1	compareEducations()	39
4.5.1.2	createDatabase()	39
4.5.1.3	freeDatabase()	39
4.6	database.h File Reference	39
4.6.1	Detailed Description	40
4.6.2	Function Documentation	40
4.6.2.1	createDatabase()	40
4.6.2.2	findEducation()	40
4.6.2.3	freeDatabase()	40
4.7	education.c File Reference	41
4.7.1	Function Documentation	41
4.7.1.1	createArrayOfEducations()	41
4.7.1.2	createDefaultEducation()	41
4.7.1.3	freeEducation()	42
4.8	education.h File Reference	42
4.8.1	Detailed Description	42
4.8.2	Function Documentation	42
4.8.2.1	createArrayOfEducations()	42
4.8.2.2	createDefaultEducation()	43
4.8.2.3	freeEducation()	43
4.9	main.c File Reference	43
4.9.1	Typedef Documentation	45

4.9.1.1	command	45
4.9.2	Enumeration Type Documentation	45
4.9.2.1	command	45
4.9.3	Function Documentation	45
4.9.3.1	argType()	46
4.9.3.2	baselsQualified()	46
4.9.3.3	convertCommand()	46
4.9.3.4	createBobo()	46
4.9.3.5	handleCommand()	46
4.9.3.6	introduction()	47
4.9.3.7	main()	47
4.9.3.8	scanCommand()	47
4.9.3.9	testConvertScale0()	48
4.9.3.10	testConvertScale10()	48
4.9.3.11	testConvertScale5()	48
4.9.3.12	testIsQualifiedHH()	48
4.9.3.13	testIsQualifiedHL()	48
4.9.3.14	testIsQualifiedHS()	49
4.9.3.15	testIsQualifiedLH()	49
4.9.3.16	testIsQualifiedLL()	49
4.9.3.17	testIsQualifiedLS()	49
4.9.3.18	testIsQualifiedSH()	49
4.9.3.19	testIsQualifiedSL()	49
4.9.3.20	testIsQualifiedSS()	49
4.9.3.21	testLevelAsValueA()	50
4.9.3.22	testLevelAsValueB()	50
4.9.3.23	testLevelAsValueC()	50
4.9.3.24	testLevelAsValueZ()	50
4.9.3.25	testParseNumOfEdu()	50
4.9.3.26	testParseNumOfInterests()	50

4.9.3.27	testSameAdjustment()	50
4.9.3.28	testSameAverageGrade()	51
4.9.3.29	testSameInterests()	51
4.9.3.30	testSameLastRecommended()	51
4.9.3.31	testSameName()	51
4.9.3.32	testSameRecommendedList()	51
4.9.3.33	testSameRegion()	51
4.9.3.34	testSameRegionImportance()	51
4.9.3.35	testSameSavedList()	52
4.9.3.36	testSuiteConvertScale()	52
4.9.3.37	testSuiteDatabase()	52
4.9.3.38	testSuiteIsQualified()	52
4.9.3.39	testSuiteLevelAsValue()	52
4.9.3.40	testSuiteProfile()	52
4.9.3.41	testSuiteValidScaleValue()	52
4.9.3.42	testValidScaleValue0()	53
4.9.3.43	testValidScaleValue10()	53
4.9.3.44	testValidScaleValue15()	53
4.9.3.45	testValidScaleValue5()	53
4.9.3.46	testValidScaleValueN5()	53
4.10	parser.c File Reference	53
4.10.1	Function Documentation	54
4.10.1.1	createArrayOfStrings()	54
4.10.1.2	findDatabaseLine()	55
4.10.1.3	parseDatabase()	55
4.10.1.4	parseDatabaseLine()	55
4.10.1.5	parseEduDesc()	56
4.10.1.6	parseEduLink()	56
4.10.1.7	parseEduNames()	56
4.10.1.8	parseEduRegion()	57

4.10.1.9	parseEduString()	57
4.10.1.10	parseInterestNames()	57
4.10.1.11	parseInterestValues()	58
4.10.1.12	parseNumOfEdu()	58
4.10.1.13	parseNumOfInterests()	58
4.10.1.14	parseReqGrade()	59
4.10.1.15	parseSubReq()	59
4.10.1.16	readReqString()	59
4.10.1.17	sseek()	60
4.10.1.18	strToReg()	60
4.11	parser.h File Reference	60
4.11.1	Detailed Description	61
4.11.2	Function Documentation	61
4.11.2.1	createArrayOfStrings()	61
4.11.2.2	findDatabaseLine()	62
4.11.2.3	parseDatabase()	62
4.11.2.4	parseDatabaseLine()	62
4.11.2.5	parseEduDesc()	63
4.11.2.6	parseEduLink()	63
4.11.2.7	parseEduNames()	63
4.11.2.8	parseEduRegion()	63
4.11.2.9	parseEduString()	64
4.11.2.10	parseInterestNames()	64
4.11.2.11	parseInterestValues()	64
4.11.2.12	parseNumOfEdu()	65
4.11.2.13	parseNumOfInterests()	65
4.11.2.14	parseReqGrade()	65
4.11.2.15	parseSubReq()	66
4.11.2.16	readReqString()	66
4.11.2.17	sseek()	66

4.11.2.18 strToReg()	67
4.12 profile.c File Reference	67
4.12.1 Function Documentation	67
4.12.1.1 createProfile()	67
4.12.1.2 freeProfile()	68
4.12.1.3 printProfile()	68
4.13 profile.h File Reference	68
4.13.1 Detailed Description	69
4.13.2 Function Documentation	69
4.13.2.1 createProfile()	69
4.13.2.2 freeProfile()	69
4.13.2.3 printProfile()	69
4.14 region.h File Reference	70
4.14.1 Detailed Description	70
4.14.2 Enumeration Type Documentation	70
4.14.2.1 region	70
4.15 subjects.c File Reference	71
4.15.1 Function Documentation	71
4.15.1.1 charToLevel()	71
4.15.1.2 createQualifications()	71
4.15.1.3 freeQualifications()	71
4.15.1.4 levelToChar()	72
4.15.1.5 stringToClass()	72
4.16 subjects.h File Reference	72
4.16.1 Detailed Description	73
4.16.2 Enumeration Type Documentation	73
4.16.2.1 anonymous enum	73
4.16.2.2 level	74
4.16.3 Function Documentation	74
4.16.3.1 charToLevel()	74

4.16.3.2	<code>createQualifications()</code>	74
4.16.3.3	<code>freeQualifications()</code>	75
4.16.3.4	<code>levelToChar()</code>	75
4.16.3.5	<code>stringToClass()</code>	75
4.17	<code>vector.h</code> File Reference	75
4.17.1	Detailed Description	76
4.17.2	Function Documentation	76
4.17.2.1	<code>addVector()</code>	76
4.17.2.2	<code>copyVector()</code>	77
4.17.2.3	<code>createVector()</code>	77
4.17.2.4	<code>dotProduct()</code>	77
4.17.2.5	<code>freeVector()</code>	77
4.17.2.6	<code>freeVectorM()</code>	78
4.17.2.7	<code>lengthOfVector()</code>	78
4.17.2.8	<code>normalizeVector()</code>	78
4.17.2.9	<code>printVector()</code>	79
4.17.2.10	<code>scaleVector()</code>	79
4.17.2.11	<code>subtractVector()</code>	79
4.18	<code>vectors.c</code> File Reference	79
4.18.1	Function Documentation	80
4.18.1.1	<code>addVector()</code>	80
4.18.1.2	<code>copyVector()</code>	81
4.18.1.3	<code>createVector()</code>	81
4.18.1.4	<code>dotProduct()</code>	81
4.18.1.5	<code>freeVector()</code>	81
4.18.1.6	<code>freeVectorM()</code>	82
4.18.1.7	<code>lengthOfVector()</code>	82
4.18.1.8	<code>normalizeVector()</code>	82
4.18.1.9	<code>printVector()</code>	83
4.18.1.10	<code>scaleVector()</code>	83
4.18.1.11	<code>subtractVector()</code>	83

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	8
profile		
	Describes a user	8
qualification	10
subject	11
vector	12

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

AllTests.c	13
commands.c	13
commands.h	
Contains functions related to command handling	24
constants.h	
Contains symbolic constants used throughout the program	35
database.c	39
database.h	
Contains elements relating to the database	39
education.c	41
education.h	
Contains elements relating to educations	42
main.c	43
parser.c	53
parser.h	
Contains elements relating to parsing the database	60
profile.c	67
profile.h	
Contains elements relating to user profiles	68
region.h	
Contains geographical elements	70
subjects.c	71
subjects.h	
Contains code regarding subjects and qualifications for educations	72
vector.h	
Contains elements relating to vectors	75
vectors.c	79

Chapter 3

Class Documentation

3.1 database Struct Reference

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

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_educations

```
int database::amount_of_educations
```

3.1.1.2 amount_of_interests

```
int database::amount_of_interests
```

an array of educations delimited by amount_of_educations

3.1.1.3 educations

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

the amount of educations in the database

3.1.1.4 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.

3.3.2 Member Data Documentation

3.3.2.1 description

```
char* education::description
```

3.3.2.2 interests

```
struct vector education::interests
```

3.3.2.3 link

```
char* education::link
```

3.3.2.4 name

```
char* education::name
```

3.3.2.5 region

```
enum region education::region
```

3.3.2.6 required_grade

```
double education::required_grade
```

3.3.2.7 required_qualifications

```
struct qualification education::required_qualifications
```

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

- [education.h](#)

3.4 location Struct Reference

```
#include <region.h>
```

Public Attributes

- enum [region](#) [region](#)
- double [region_importance](#)

3.4.1 Member Data Documentation

3.4.1.1 region

```
enum region location::region
```

3.4.1.2 region_importance

```
double location::region_importance
```

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

- [region.h](#)

3.5 profile Struct Reference

Describes a user.

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

Collaboration diagram for profile:

Public Attributes

- struct [vector](#) [interests](#)
- struct [vector](#) [adjustment_vector](#)
- char [name](#) [[MAX_NAME_LENGTH](#)]
- struct [qualification](#) [qualifications](#)
- double [average](#)
- struct [location](#) [location](#)
- char [saved_educations](#) [[EDUCATION_LIST_LENGTH](#)][[MAX_EDU_NAME_LENGTH](#)]
- int [last_recommended](#)
- char [recommended_educations](#) [[EDUCATION_LIST_LENGTH](#)][[MAX_EDU_NAME_LENGTH](#)]

3.5.1 Detailed Description

Describes a user.

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

3.5.2 Member Data Documentation

3.5.2.1 [adjustment_vector](#)

```
struct vector profile::adjustment_vector
```

3.5.2.2 [average](#)

```
double profile::average
```

3.5.2.3 [interests](#)

```
struct vector profile::interests
```

3.5.2.4 [last_recommended](#)

```
int profile::last_recommended
```

3.5.2.5 location

```
struct location profile::location
```

3.5.2.6 name

```
char profile::name[MAX\_NAME\_LENGTH]
```

3.5.2.7 qualifications

```
struct qualification profile::qualifications
```

3.5.2.8 recommended_educations

```
char profile::recommended_educations[EDUCATION\_LIST\_LENGTH][MAX\_EDU\_NAME\_LENGTH]
```

3.5.2.9 saved_educations

```
char profile::saved_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

```
#include <subjects.h>
```

Collaboration diagram for qualification:

Public Attributes

- int [amount_of_subjects](#)
- struct [subject](#) * [subjects](#)

3.6.1 Member Data Documentation

3.6.1.1 amount_of_subjects

```
int qualification::amount_of_subjects
```

3.6.1.2 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

```
#include <subjects.h>
```

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

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

Public Attributes

- double * [array](#)
- int [size](#)

3.8.1 Member Data Documentation

3.8.1.1 array

```
double* vector::array
```

3.8.1.2 size

```
int vector::size
```

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

- [vector.h](#)

Chapter 4

File Documentation

4.1 AllTests.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "profile.h"
#include "education.h"
#include "subjects.h"
#include "vector.h"
#include "CuTest.h"
#include "commands.h"
Include dependency graph for AllTests.c:
```

4.2 commands.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include "profile.h"
#include "education.h"
#include "subjects.h"
#include "vector.h"
#include "commands.h"
#include "constants.h"
Include dependency graph for commands.c:
```

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. Saves the interest in studying in this location.
- double `convertScale` (int v)
Returns the converted value.
- 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.
- 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)
- double `getValidDouble` (void)
Returns a valid double entered in the terminal.
- 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.
- 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.
- 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.
- void `printEducation` (struct `education` education)
- const char * `getRegionName` (enum `region` r)
- 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)

4.2.1 Function Documentation

4.2.1.1 `checkProfile()`

```
int checkProfile (
    const char name[ ] )
```

4.2.1.2 `chooseFromList()`

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

4.2.1.3 `classNameStr()`

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

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

Parameters

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

4.2.1.4 `clearBuffer()`

```
void clearBuffer (
    void )
```

Empties the buffer for standard input.

4.2.1.5 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

4.2.1.6 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
-------------	---------------------------------

4.2.1.7 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

4.2.1.8 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

4.2.1.9 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
------------------	---

4.2.1.10 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

4.2.1.11 getRegionName()

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

4.2.1.12 getValidDouble()

```
double getValidDouble (
    void )
```

Returns a valid double entered in the terminal.

4.2.1.13 getValidInteger()

```
int getValidInteger (
    void )
```

Returns a valid integer given through the terminal.

4.2.1.14 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

4.2.1.15 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
----------	--

4.2.1.16 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
-------------	---------------------------------

4.2.1.17 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
----------	---

4.2.1.18 loadProfile()

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

4.2.1.19 menuCmd()

```
void menuCmd (
    void )
```

Prints all the possible commands the user can use.

4.2.1.20 printEducation()

```
void printEducation (
    struct education education )
```

4.2.1.21 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.

4.2.1.22 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

4.2.1.23 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
-------------	---------------------------------

4.2.1.24 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.

4.2.1.25 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
-------------	--

4.2.1.26 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 ends of the subjects to be asked for

4.2.1.27 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

4.2.1.28 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
-------------	---

4.2.1.29 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
-------------	--

4.2.1.30 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
-------------	--

4.2.1.31 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

4.2.1.32 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

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

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. Saves the interest in studying in this location.
- 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. 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.
- 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)
- 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. The distance of the change is determined by the argument.
- 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)
- 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)`

4.3.1 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.

4.3.2 Function Documentation

4.3.2.1 checkProfile()

```
int checkProfile (
    const char name[ ] )
```

4.3.2.2 chooseFromList()

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

4.3.2.3 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
--------------	---

4.3.2.4 clearBuffer()

```
void clearBuffer (  
    void )
```

Empties the buffer for standard input.

4.3.2.5 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

4.3.2.6 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
-------------	---------------------------------

4.3.2.7 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

4.3.2.8 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

4.3.2.9 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
------------------	---

4.3.2.10 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

4.3.2.11 getRegionName()

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

4.3.2.12 getValidDouble()

```
double getValidDouble (
    void )
```

Returns a valid double entered in the terminal.

4.3.2.13 getValidInteger()

```
int getValidInteger (
    void )
```

Returns a valid integer given through the terminal.

4.3.2.14 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

4.3.2.15 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
----------	--

4.3.2.16 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
-------------	---------------------------------

4.3.2.17 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
----------	---

4.3.2.18 loadProfile()

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

4.3.2.19 menuCmd()

```
void menuCmd (
    void )
```

Prints all the possible commands the user can use.

4.3.2.20 printEducation()

```
void printEducation (
    struct education )
```

4.3.2.21 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.

4.3.2.22 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

4.3.2.23 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
-------------	---------------------------------

4.3.2.24 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.

4.3.2.25 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
-------------	--

4.3.2.26 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

4.3.2.27 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

4.3.2.28 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
-------------	---

4.3.2.29 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
-------------	--

4.3.2.30 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
-------------	--

4.3.2.31 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

4.3.2.32 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

4.4 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)`
- `#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"`

4.4.1 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 lenght of strings or constants used for string formatting.

4.4.2 Macro Definition Documentation

4.4.2.1 ADJUSTMENT_CONSTANT

```
#define ADJUSTMENT_CONSTANT 0.1
```

4.4.2.2 DATABASE_PATH

```
#define DATABASE_PATH "./bin/data/database.txt"
```

4.4.2.3 EDU_MAX_SUBJECTS

```
#define EDU_MAX_SUBJECTS 10
```

4.4.2.4 EDUCATION_LIST_LENGTH

```
#define EDUCATION_LIST_LENGTH 10
```

4.4.2.5 FIELD_SIZE

```
#define FIELD_SIZE 25
```

4.4.2.6 IMPORTANT_SUBJECTS

```
#define IMPORTANT_SUBJECTS 5
```

4.4.2.7 LANGUAGE_SUBJECTS

```
#define LANGUAGE_SUBJECTS 11
```

4.4.2.8 MAX_COMMAND_LENGTH

```
#define MAX_COMMAND_LENGTH 10
```

4.4.2.9 MAX_EDU_NAME_LENGTH

```
#define MAX_EDU_NAME_LENGTH 40
```

4.4.2.10 MAX_FILE_NAME_LENGTH

```
#define MAX_FILE_NAME_LENGTH MAX_NAME_LENGTH + 12
```

4.4.2.11 MAX_INPUT_LENGTH

```
#define MAX_INPUT_LENGTH (MAX_COMMAND_LENGTH + 100)
```

4.4.2.12 MAX_NAME_LENGTH

```
#define MAX_NAME_LENGTH 20
```

4.4.2.13 NO_EMPTY_INDEX

```
#define NO_EMPTY_INDEX -1
```

4.4.2.14 NOT_FOUND_STRING

```
#define NOT_FOUND_STRING " "
```

4.4.2.15 NOT_IN_LIST

```
#define NOT_IN_LIST -1
```

4.4.2.16 NUMBER_OF_REGIONS

```
#define NUMBER_OF_REGIONS 5
```

4.4.2.17 OTHER_SUBJECTS

```
#define OTHER_SUBJECTS 11
```

4.4.2.18 STRING_MAX_LENGTH

```
#define STRING_MAX_LENGTH 10000
```

4.4.2.19 TABS

```
#define TABS ' '
```

4.4.2.20 TOTAL_SUBJECTS

```
#define TOTAL_SUBJECTS (IMPORTANT_SUBJECTS + OTHER_SUBJECTS + LANGUAGE_SUBJECTS)
```

4.4.2.21 USELESS_SUBJECTS

```
#define USELESS_SUBJECTS 2
```

4.4.2.22 VERSION

```
#define VERSION "1.0.1"
```

4.5 database.c File Reference

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

Functions

- int [compareEducations](#) (const void *, const void *)
- void [freeDatabase](#) (struct [database](#) *[database](#))
- struct [database](#) * [createDatabase](#) (char *[database_file](#))

4.5.1 Function Documentation

4.5.1.1 [compareEducations\(\)](#)

```
int compareEducations (
    const void * ,
    const void * )
```

4.5.1.2 [createDatabase\(\)](#)

```
struct database* createDatabase (
    char * database\_file )
```

4.5.1.3 [freeDatabase\(\)](#)

```
void freeDatabase (
    struct database * database )
```

4.6 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` *)

4.6.1 Detailed Description

Contains elements relating to the database.

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

4.6.2 Function Documentation

4.6.2.1 `createDatabase()`

```
struct database* createDatabase (  
    char * )
```

4.6.2.2 `findEducation()`

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

4.6.2.3 `freeDatabase()`

```
void freeDatabase (  
    struct database * )
```

4.7 education.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "region.h"
#include "subjects.h"
#include "education.h"
#include "vector.h"
#include "profile.h"
Include dependency graph for education.c:
```

Functions

- struct [education](#) [createDefaultEducation](#) (int amount_of_interests, int amount_of_subjects)
Assigns default values to the fields of the education struct.
- void [freeEducation](#) (struct [education](#) *[education](#))
- struct [education](#) * [createArrayOfEducations](#) (int amount_of_educations)
Allocate memory for an array of educations and return a pointer to it.

4.7.1 Function Documentation

4.7.1.1 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
-----------------------------	--

4.7.1.2 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

4.7.1.3 freeEducation()

```
void freeEducation (
    struct education * education )
```

4.8 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](#) *)

4.8.1 Detailed Description

Contains elements relating to educations.

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

4.8.2 Function Documentation

4.8.2.1 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
-----------------------------	--

4.8.2.2 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

4.8.2.3 freeEducation()

```
void freeEducation (
    struct education * )
```

4.9 main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "commands.h"
#include "constants.h"
#include "CuTest.h"
#include "database.h"
#include "education.h"
#include "profile.h"
#include "region.h"
#include "subjects.h"
#include "vector.h"
#include "parser.h"
Include dependency graph for main.c:
```

Typedefs

- typedef enum [command](#) [command](#)

Enumerations

- enum `command` {
`find`, `search`, `load`, `save`,
`save_prof`, `recommend`, `list`, `eval`,
`survey`, `menu`, `quit`, `delete` }

Functions

- void `introduction` (void)
Prints information about the program.
- void `handleCommand` (`command` c, char arg[`MAX_INPUT_LENGTH`], int arg_num, struct `profile` *user, const struct `database` *database, struct `education` *current_education)
Calls the function associated with the given command.
- `command` `scanCommand` (char arg[`MAX_INPUT_LENGTH`], int *arg_num)
Finds valid command and argument.
- `command` `convertCommand` (char s[`MAX_COMMAND_LENGTH`])
- int `argType` (`command` c)
Determines whether the entered command takes a string argument, a integer argument or no argument.
- struct `profile` `createBobo` (int amount_of_interests)
- int `main` (void)
Takes commands from the user and executes those commands until the quit command is entered.
- void `testLevelAsValueA` (CuTest *tc)
- void `testLevelAsValueB` (CuTest *tc)
- void `testLevelAsValueC` (CuTest *tc)
- void `testLevelAsValueZ` (CuTest *tc)
- CuSuite * `testSuiteLevelAsValue` (void)
- void `testValidScaleValueN5` (CuTest *tc)
- void `testValidScaleValue0` (CuTest *tc)
- void `testValidScaleValue5` (CuTest *tc)
- void `testValidScaleValue10` (CuTest *tc)
- void `testValidScaleValue15` (CuTest *tc)
- CuSuite * `testSuiteValidScaleValue` (void)
- void `testConvertScale0` (CuTest *tc)
- void `testConvertScale5` (CuTest *tc)
- void `testConvertScale10` (CuTest *tc)
- CuSuite * `testSuiteConvertScale` (void)
- int `baselsQualified` (int user1, int user2, int edu1, int edu2)
- void `testIsQualifiedLL` (CuTest *tc)
- void `testIsQualifiedLS` (CuTest *tc)
- void `testIsQualifiedLH` (CuTest *tc)
- void `testIsQualifiedSL` (CuTest *tc)
- void `testIsQualifiedSS` (CuTest *tc)
- void `testIsQualifiedSH` (CuTest *tc)
- void `testIsQualifiedHL` (CuTest *tc)
- void `testIsQualifiedHS` (CuTest *tc)
- void `testIsQualifiedHH` (CuTest *tc)
- CuSuite * `testSuiteIsQualified` (void)
- void `testParseNumOfEdu` (CuTest *tc)
- void `testParseNumOfInterests` (CuTest *tc)
- CuSuite * `testSuiteDatabase` (void)
- void `testSameName` (CuTest *tc)
- void `testSameAverageGrade` (CuTest *tc)

- void [testSameLastRecommended](#) (CuTest *tc)
- void [testSameRegion](#) (CuTest *tc)
- void [testSameRegionImportance](#) (CuTest *tc)
- void [testSameSavedList](#) (CuTest *tc)
- void [testSameRecommendedList](#) (CuTest *tc)
- void [testSameInterests](#) (CuTest *tc)
- void [testSameAdjustment](#) (CuTest *tc)
- CuSuite * [testSuiteProfile](#) (void)

4.9.1 Typedef Documentation

4.9.1.1 command

typedef enum [command](#) [command](#)

4.9.2 Enumeration Type Documentation

4.9.2.1 command

enum [command](#)

Enumerator

find	
search	
load	
save	
save_prof	
recommend	
list	
eval	
survey	
menu	
quit	
delete	

4.9.3 Function Documentation

4.9.3.1 argType()

```
int argType (
    command c )
```

Determines whether the entered command takes a string argument, a integer argument or no argument.

Parameters

<i>c</i>	An Enum command
----------	-----------------

Returns

An integer being 0 if the command takes no argument, 1 if the command takes a string argument and -1 if it takes an integer argument.

4.9.3.2 baselsQualified()

```
int baseIsQualified (
    int user1,
    int user2,
    int edu1,
    int edu2 )
```

4.9.3.3 convertCommand()

```
command convertCommand (
    char s[MAX_COMMAND_LENGTH] )
```

4.9.3.4 createBobo()

```
struct profile createBobo (
    int amount_of_interests )
```

4.9.3.5 handleCommand()

```
void handleCommand (
    command c,
    char arg[MAX_INPUT_LENGTH],
    int arg_num,
    struct profile * user,
    const struct database * database,
    struct education * current_education )
```

Calls the function associated with the given command.

Parameters

<i>c</i>	The enum command which the user have entered
<i>arg</i>	The string argument used by some commands
<i>arg_num</i>	The integer argument used by some commands
<i>user</i>	A pointer to the profile struct associated with the user of the program
<i>database</i>	A pointer to the database
<i>current_education</i>	A pointer to the last shown education

4.9.3.6 introduction()

```
void introduction (
    void )
```

Prints information about the program.

4.9.3.7 main()

```
int main (
    void )
```

Takes commands from the user and executes those commands until the quit command is entered.

4.9.3.8 scanCommand()

```
int scanCommand (
    char arg[MAX_INPUT_LENGTH],
    int * arg_num )
```

Finds valid command and argument.

Compares the entered string to the command words and returns the enum command associated with the string.

Parameters

<i>arg</i>	Output parameter for argument string
<i>arg_num</i>	Output parameter for argument int

Returns

The enum command associated with the parameter string

Parameters

s	A command as a string
----------	-----------------------

Returns

The enum command associated with the parameter string

4.9.3.9 testConvertScale0()

```
void testConvertScale0 (  
    CuTest * tc )
```

4.9.3.10 testConvertScale10()

```
void testConvertScale10 (  
    CuTest * tc )
```

4.9.3.11 testConvertScale5()

```
void testConvertScale5 (  
    CuTest * tc )
```

4.9.3.12 testIsQualifiedHH()

```
void testIsQualifiedHH (  
    CuTest * tc )
```

4.9.3.13 testIsQualifiedHL()

```
void testIsQualifiedHL (  
    CuTest * tc )
```

4.9.3.14 testIsQualifiedHS()

```
void testIsQualifiedHS (  
    CuTest * tc )
```

4.9.3.15 testIsQualifiedLH()

```
void testIsQualifiedLH (  
    CuTest * tc )
```

4.9.3.16 testIsQualifiedLL()

```
void testIsQualifiedLL (  
    CuTest * tc )
```

4.9.3.17 testIsQualifiedLS()

```
void testIsQualifiedLS (  
    CuTest * tc )
```

4.9.3.18 testIsQualifiedSH()

```
void testIsQualifiedSH (  
    CuTest * tc )
```

4.9.3.19 testIsQualifiedSL()

```
void testIsQualifiedSL (  
    CuTest * tc )
```

4.9.3.20 testIsQualifiedSS()

```
void testIsQualifiedSS (  
    CuTest * tc )
```

4.9.3.21 testLevelAsValueA()

```
void testLevelAsValueA (  
    CuTest * tc )
```

4.9.3.22 testLevelAsValueB()

```
void testLevelAsValueB (  
    CuTest * tc )
```

4.9.3.23 testLevelAsValueC()

```
void testLevelAsValueC (  
    CuTest * tc )
```

4.9.3.24 testLevelAsValueZ()

```
void testLevelAsValueZ (  
    CuTest * tc )
```

4.9.3.25 testParseNumOfEdu()

```
void testParseNumOfEdu (  
    CuTest * tc )
```

4.9.3.26 testParseNumOfInterests()

```
void testParseNumOfInterests (  
    CuTest * tc )
```

4.9.3.27 testSameAdjustment()

```
void testSameAdjustment (  
    CuTest * tc )
```


4.9.3.28 testSameAverageGrade()

```
void testSameAverageGrade (
    CuTest * tc )
```

4.9.3.29 testSameInterests()

```
void testSameInterests (
    CuTest * tc )
```

4.9.3.30 testSameLastRecommended()

```
void testSameLastRecommended (
    CuTest * tc )
```

4.9.3.31 testSameName()

```
void testSameName (
    CuTest * tc )
```

4.9.3.32 testSameRecommendedList()

```
void testSameRecommendedList (
    CuTest * tc )
```

4.9.3.33 testSameRegion()

```
void testSameRegion (
    CuTest * tc )
```

4.9.3.34 testSameRegionImportance()

```
void testSameRegionImportance (
    CuTest * tc )
```

4.9.3.35 testSameSavedList()

```
void testSameSavedList (
    CuTest * tc )
```

4.9.3.36 testSuiteConvertScale()

```
CuSuite* testSuiteConvertScale (
    void )
```

4.9.3.37 testSuiteDatabase()

```
CuSuite* testSuiteDatabase (
    void )
```

4.9.3.38 testSuiteIsQualified()

```
CuSuite* testSuiteIsQualified (
    void )
```

4.9.3.39 testSuiteLevelAsValue()

```
CuSuite* testSuiteLevelAsValue (
    void )
```

4.9.3.40 testSuiteProfile()

```
CuSuite* testSuiteProfile (
    void )
```

4.9.3.41 testSuiteValidScaleValue()

```
CuSuite* testSuiteValidScaleValue (
    void )
```

4.9.3.42 testValidScaleValue0()

```
void testValidScaleValue0 (
    CuTest * tc )
```

4.9.3.43 testValidScaleValue10()

```
void testValidScaleValue10 (
    CuTest * tc )
```

4.9.3.44 testValidScaleValue15()

```
void testValidScaleValue15 (
    CuTest * tc )
```

4.9.3.45 testValidScaleValue5()

```
void testValidScaleValue5 (
    CuTest * tc )
```

4.9.3.46 testValidScaleValueN5()

```
void testValidScaleValueN5 (
    CuTest * tc )
```

4.10 parser.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "database.h"
#include "parser.h"
#include "region.h"
#include "vector.h"
#include "constants.h"
#include "education.h"
Include dependency graph for parser.c:
```

Functions

- void `parseDatabase` (struct `database` *`database`, FILE *`filereader`)
Parse the database file and set all values in the database.
- char ** `createArrayOfStrings` (int `amount_of_strings`)
Allocate memory for an array of strings and return a pointer to it.
- 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`.
- void `parseDatabaseLine` (const char `key`[], struct `database` *`database`, FILE *`filereader`)
Parse the line containing key and return into database.
- void `parseInterestNames` (struct `database` *`database`, FILE *`filereader`)
Parse the names of each interest and return to the database.
- int `parseNumOfInterests` (FILE *`filereader`)
Parse/count the number of intersts in the database file and return as int.
- 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.
- 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 `parseEduRegion` (int `amount_of_educations`, struct `education` *`educations`, char `current_line`[])
Parses the region for each education.
- enum `region strToReg` (char *`region_string`)
Converts a string into an enum region and return an enum region.
- int `parseNumOfEdu` (FILE *`filereader`)
Returns the number of educations from database file.
- 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`[])
- 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.
- int `sseek` (char *`string`, char `ch`)
- void `readReqString` (struct `qualification` *`qualification`, char *`string`, int `education_location`)
Read a requiremnt from a string.

4.10.1 Function Documentation

4.10.1.1 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
--------------------------	---

4.10.1.2 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

4.10.1.3 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

4.10.1.4 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

4.10.1.5 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

4.10.1.6 parseEduLink()

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

4.10.1.7 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

4.10.1.8 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

4.10.1.9 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

4.10.1.10 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

4.10.1.11 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

4.10.1.12 parseNumOfEdu()

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

Returns the number of educations from database file.

Parameters

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

4.10.1.13 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
-------------------	-------------------

4.10.1.14 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

4.10.1.15 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

4.10.1.16 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 colomn is the educations requirements in.

4.10.1.17 sseek()

```
int sseek (
    char * string,
    char ch )
```

4.10.1.18 strToReg()

```
enum region 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
----------------------	-----------------------

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

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.*

4.11.1 Detailed Description

Contains elements relating to parsing the database.

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

4.11.2 Function Documentation

4.11.2.1 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
--------------------------	---

4.11.2.2 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

4.11.2.3 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

4.11.2.4 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

4.11.2.5 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

4.11.2.6 parseEduLink()

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

4.11.2.7 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

4.11.2.8 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

4.11.2.9 parseEduString()

```

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

```

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

Parameters

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

4.11.2.10 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

4.11.2.11 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

4.11.2.12 parseNumOfEdu()

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

Returns the number of educations from database file.

Parameters

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

4.11.2.13 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
-------------------	-------------------

4.11.2.14 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

4.11.2.15 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

4.11.2.16 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 colomn is the educations requirements in.

4.11.2.17 sseek()

```
int sseek (
    char * ,
    char )
```


4.11.2.18 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
----------------------	-----------------------

4.12 profile.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "profile.h"
#include "education.h"
#include "subjects.h"
#include "vector.h"
#include "commands.h"
```

Include dependency graph for profile.c:

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.

4.12.1 Function Documentation

4.12.1.1 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
----------------------------	-----------------------------------

4.12.1.2 freeProfile()

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

Frees the allocated memory for the given profile.

Parameters

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

4.12.1.3 printProfile()

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

Prints information stored in the given profil.

Parameters

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

4.13 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"
```

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

Classes

- struct `profile`
Describes a user.

Functions

- struct `profile` `createProfile` (int number_of_interests)
Allocates memory for each of the fields in the profile struct.
- void `freeProfile` (struct `profile` p)
Frees the allocated memory for the given profile.
- void `printProfile` (struct `profile` p)
Prints information stored in the given profil.

4.13.1 Detailed Description

Contains elements relating to user profiles.

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

4.13.2 Function Documentation

4.13.2.1 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
----------------------------	-----------------------------------

4.13.2.2 freeProfile()

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

Frees the allocated memory for the given profile.

Parameters

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

4.13.2.3 printProfile()

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

Prints information stored in the given profil.

Parameters

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

4.14 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.

4.14.1 Detailed Description

Contains geographical elements.

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

4.14.2 Enumeration Type Documentation

4.14.2.1 region

enum [region](#)

Describes a region.

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

Enumerator

NORTH_JUTLAND	
CENTRAL_JUTLAND	
SOUTHERN_DENMARK	
ZEALAND	
CAPITAL_REGION	

4.15 subjects.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "subjects.h"
Include dependency graph for subjects.c:
```

Functions

- struct [qualification](#) [createQualifications](#) (int number_of_qualifications)
- void [freeQualifications](#) (struct [qualification](#) *[qualification](#))
- enum [stringToClass](#) (char *string)
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.

4.15.1 Function Documentation

4.15.1.1 [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
-----------	---

4.15.1.2 [createQualifications\(\)](#)

```
struct qualification createQualifications (  
    int number_of_qualifications )
```

4.15.1.3 [freeQualifications\(\)](#)

```
void freeQualifications (  
    struct qualification * qualification )
```

4.15.1.4 levelToChar()

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

Returns the character associated with the given enum level.

Parameters

/	The enum level which is converted into a character
---	--

4.15.1.5 stringToClass()

```
enum 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
---------------	--

4.16 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 {
[@0::MATHEMATICS](#), [@0::CHEMISTRY](#), [@0::BIOLOGY](#), [@0::PHYSICS](#),
[@0::ENGLISH](#), [@0::BIOTECHNOLOGY](#), [@0::GEOSCIENCE](#), [@0::HISTORY](#),
[@0::IDEA_HISTORY](#), [@0::INFORMATICS](#), [@0::INTERNATIONAL_ECONOMICS](#), [@0::COMMUNICATI↵](#)
[ON_AND_IT](#),
[@0::RELIGION](#), [@0::SOCIALSTUDIES](#), [@0::BUSINESS_ECONOMICS](#), [@0::CONTEMPORARY_HISTO↵](#)
[RY](#),
[@0::FRENCH](#), [@0::SPANISH](#), [@0::GERMAN](#), [@0::CHINESE](#),
[@0::ARABIC](#), [@0::GREEK](#), [@0::ITALIAN](#), [@0::JAPANESE](#),
[@0::LATIN](#), [@0::PORTUGESE](#), [@0::RUSSIAN](#), [@0::NONE](#),
[@0::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.

4.16.1 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

4.16.2 Enumeration Type Documentation

4.16.2.1 anonymous enum

```
anonymous enum [strong]
```

Enumerator

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	

Enumerator

GREEK	
ITALIAN	
JAPANESE	
LATIN	
PORTUGUESE	
RUSSIAN	
NONE	
DANISH	

4.16.2.2 level

enum qualifications [A](#) struct containing an array of subjects where each [subject](#) has a name and and a [level](#)

Enumerator

Z	
C	
B	
A	

4.16.3 Function Documentation**4.16.3.1 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
-----------	---

4.16.3.2 createQualifications()

```
struct qualification createQualifications (  
    int number_of_qualifications )
```


4.16.3.3 freeQualifications()

```
void freeQualifications (
    struct qualification * )
```

4.16.3.4 levelToChar()

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

Returns the character associated with the given enum level.

Parameters

/	The enum level which is converted into a character
---	--

4.16.3.5 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
---------------	--

4.17 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)

4.17.1 Detailed Description

Contains elements relating to vectors.

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

4.17.2 Function Documentation

4.17.2.1 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

4.17.2.2 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
----------	---------------------------------

4.17.2.3 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
-------------	--------------------------------------

4.17.2.4 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

4.17.2.5 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
----------	--

4.17.2.6 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
------------	--

4.17.2.7 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
----------	----------------------------------

4.17.2.8 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
----------	--------------------------------------

4.17.2.9 printVector()

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

Prints a vector.

Parameters

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

4.17.2.10 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

4.17.2.11 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

4.18 vectors.c File Reference

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "vector.h"
```

```
#include <stdarg.h>
Include dependency graph for vectors.c:
```

Functions

- struct [vector](#) [createVector](#) (int size)
creates a vector on the heap and outputs it
- void [freeVectorM](#) (int num,...)
Frees a variable number of struct vectors using free(Vector)
- void [freeVector](#) (struct [vector](#) v)
frees the dynamically allocated array on the heap
- struct [vector](#) [copyVector](#) (struct [vector](#) v)
Copies the the inputted vector into vector copy and returns this.
- void [printVector](#) (struct [vector](#) v)
Prints a vector.
- 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.
- double [lengthOfVector](#) (struct [vector](#) v)
Calculates and returns the length of the given 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 [dotProduct](#) (struct [vector](#) v1, struct [vector](#) v2)
Calculates and returns the dot product of two vectors.

4.18.1 Function Documentation

4.18.1.1 [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

4.18.1.2 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
----------	---------------------------------

4.18.1.3 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
-------------	--------------------------------------

4.18.1.4 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

4.18.1.5 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
----------	--

4.18.1.6 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
------------	--

4.18.1.7 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
----------	----------------------------------

4.18.1.8 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
----------	--------------------------------------

4.18.1.9 printVector()

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

Prints a vector.

Parameters

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

4.18.1.10 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

4.18.1.11 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

ADJUSTMENT_CONSTANT

constants.h, 36

addVector

vector.h, 76

vectors.c, 80

adjustment_vector

profile, 9

AllTests.c, 13

amount_of_educations

database, 5

amount_of_interests

database, 5

amount_of_subjects

qualification, 11

argType

main.c, 45

array

vector, 12

average

profile, 9

baseIsQualified

main.c, 46

charToLevel

subjects.c, 71

subjects.h, 74

checkProfile

commands.c, 15

commands.h, 26

chooseFromList

commands.c, 15

commands.h, 26

classNameStr

commands.c, 15

commands.h, 26

clearBuffer

commands.c, 15

commands.h, 27

command

main.c, 45

commands.c, 13

checkProfile, 15

chooseFromList, 15

classNameStr, 15

clearBuffer, 15

convertScale, 15

deleteCmd, 16

evalCmd, 16

findCmd, 16

getEmptyIndex, 17

getIndex, 17

getRegionName, 17

getValidDouble, 18

getValidInteger, 18

isQualified, 18

levelAsValue, 18

listCmd, 19

listIsFull, 19

loadProfile, 19

menuCmd, 19

printEducation, 19

recommendCmd, 20

saveCmd, 20

saveProfile, 20

searchCmd, 21

setImportantSubjects, 21

setOtherSubjects, 21

setProfileInterests, 22

setProfileLocation, 22

setProfileQualifications, 22

setSubjects, 22

surveyCmd, 24

validScaleValue, 24

commands.h, 24

checkProfile, 26

chooseFromList, 26

classNameStr, 26

clearBuffer, 27

convertScale, 27

deleteCmd, 27

evalCmd, 27

findCmd, 28

getEmptyIndex, 28

getIndex, 28

getRegionName, 29

getValidDouble, 29

getValidInteger, 29

isQualified, 29

levelAsValue, 30

listCmd, 30

listIsFull, 30

loadProfile, 31

menuCmd, 31

printEducation, 31

recommendCmd, 31

saveCmd, 31

saveProfile, 32

searchCmd, 32

- setImportantSubjects, 32
 - setOtherSubjects, 33
 - setProfileInterests, 33
 - setProfileLocation, 33
 - setProfileQualifications, 34
 - setSubjects, 34
 - surveyCmd, 34
 - validScaleValue, 34
- compareEducations
 - database.c, 39
- constants.h, 35
 - ADJUSTMENT_CONSTANT, 36
 - DATABASE_PATH, 36
 - EDU_MAX_SUBJECTS, 36
 - EDUCATION_LIST_LENGTH, 36
 - FIELD_SIZE, 36
 - IMPORTANT_SUBJECTS, 36
 - LANGUAGE_SUBJECTS, 36
 - MAX_COMMAND_LENGTH, 36
 - MAX_EDU_NAME_LENGTH, 37
 - MAX_FILE_NAME_LENGTH, 37
 - MAX_INPUT_LENGTH, 37
 - MAX_NAME_LENGTH, 37
 - NO_EMPTY_INDEX, 37
 - NOT_FOUND_STRING, 37
 - NOT_IN_LIST, 37
 - NUMBER_OF_REGIONS, 37
 - OTHER_SUBJECTS, 38
 - STRING_MAX_LENGTH, 38
 - TABS, 38
 - TOTAL_SUBJECTS, 38
 - USELESS_SUBJECTS, 38
 - VERSION, 38
- convertCommand
 - main.c, 46
- convertScale
 - commands.c, 15
 - commands.h, 27
- copyVector
 - vector.h, 76
 - vectors.c, 80
- createArrayOfEducations
 - education.c, 41
 - education.h, 42
- createArrayOfStrings
 - parser.c, 54
 - parser.h, 61
- createBobo
 - main.c, 46
- createDatabase
 - database.c, 39
 - database.h, 40
- createDefaultEducation
 - education.c, 41
 - education.h, 43
- createProfile
 - profile.c, 67
 - profile.h, 69
- createQualifications
 - subjects.c, 71
 - subjects.h, 74
- createVector
 - vector.h, 77
 - vectors.c, 81
- DATABASE_PATH
 - constants.h, 36
- Database, 6
- database, 5
 - amount_of_educations, 5
 - amount_of_interests, 5
 - educations, 5
 - interest_string, 6
- database.c, 39
 - compareEducations, 39
 - createDatabase, 39
 - freeDatabase, 39
- database.h, 39
 - createDatabase, 40
 - findEducation, 40
 - freeDatabase, 40
- deleteCmd
 - commands.c, 16
 - commands.h, 27
- description
 - education, 7
- dotProduct
 - vector.h, 77
 - vectors.c, 81
- EDU_MAX_SUBJECTS
 - constants.h, 36
- EDUCATION_LIST_LENGTH
 - constants.h, 36
- education, 6
 - description, 7
 - interests, 7
 - link, 7
 - name, 7
 - region, 7
 - required_grade, 7
 - required_qualifications, 7
- education.c, 41
 - createArrayOfEducations, 41
 - createDefaultEducation, 41
 - freeEducation, 42
- education.h, 42
 - createArrayOfEducations, 42
 - createDefaultEducation, 43
 - freeEducation, 43
- educations
 - database, 5
- evalCmd
 - commands.c, 16
 - commands.h, 27
- FIELD_SIZE

- constants.h, 36
- findCmd
 - commands.c, 16
 - commands.h, 28
- findDatabaseLine
 - parser.c, 55
 - parser.h, 61
- findEducation
 - database.h, 40
- freeDatabase
 - database.c, 39
 - database.h, 40
- freeEducation
 - education.c, 42
 - education.h, 43
- freeProfile
 - profile.c, 68
 - profile.h, 69
- freeQualifications
 - subjects.c, 71
 - subjects.h, 74
- freeVector
 - vector.h, 77
 - vectors.c, 81
- freeVectorM
 - vector.h, 78
 - vectors.c, 82
- getEmptyIndex
 - commands.c, 17
 - commands.h, 28
- getIndex
 - commands.c, 17
 - commands.h, 28
- getRegionName
 - commands.c, 17
 - commands.h, 29
- getValidDouble
 - commands.c, 18
 - commands.h, 29
- getValidInteger
 - commands.c, 18
 - commands.h, 29
- handleCommand
 - main.c, 46
- IMPORTANT_SUBJECTS
 - constants.h, 36
- interest_string
 - database, 6
- interests
 - education, 7
 - profile, 9
- introduction
 - main.c, 47
- isQualified
 - commands.c, 18
 - commands.h, 29

- LANGUAGE_SUBJECTS
 - constants.h, 36
- last_recommended
 - profile, 9
- lengthOfVector
 - vector.h, 78
 - vectors.c, 82
- level
 - subject, 11
 - subjects.h, 74
- levelAsValue
 - commands.c, 18
 - commands.h, 30
- levelToChar
 - subjects.c, 71
 - subjects.h, 75
- link
 - education, 7
- listCmd
 - commands.c, 19
 - commands.h, 30
- listIsFull
 - commands.c, 19
 - commands.h, 30
- loadProfile
 - commands.c, 19
 - commands.h, 31
- location, 8
 - profile, 9
 - region, 8
 - region_importance, 8
- MAX_COMMAND_LENGTH
 - constants.h, 36
- MAX_EDU_NAME_LENGTH
 - constants.h, 37
- MAX_FILE_NAME_LENGTH
 - constants.h, 37
- MAX_INPUT_LENGTH
 - constants.h, 37
- MAX_NAME_LENGTH
 - constants.h, 37
- main
 - main.c, 47
- main.c, 43
 - argType, 45
 - baselsQualified, 46
 - command, 45
 - convertCommand, 46
 - createBobo, 46
 - handleCommand, 46
 - introduction, 47
 - main, 47
 - scanCommand, 47
 - testConvertScale0, 48
 - testConvertScale10, 48
 - testConvertScale5, 48
 - testIsQualifiedHH, 48
 - testIsQualifiedHL, 48

- testIsQualifiedHS, 48
- testIsQualifiedLH, 49
- testIsQualifiedLL, 49
- testIsQualifiedLS, 49
- testIsQualifiedSH, 49
- testIsQualifiedSL, 49
- testIsQualifiedSS, 49
- testLevelAsValueA, 49
- testLevelAsValueB, 50
- testLevelAsValueC, 50
- testLevelAsValueZ, 50
- testParseNumOfEdu, 50
- testParseNumOfInterests, 50
- testSameAdjustment, 50
- testSameAverageGrade, 50
- testSameInterests, 51
- testSameLastRecommended, 51
- testSameName, 51
- testSameRecommendedList, 51
- testSameRegion, 51
- testSameRegionImportance, 51
- testSameSavedList, 51
- testSuiteConvertScale, 52
- testSuiteDatabase, 52
- testSuiteIsQualified, 52
- testSuiteLevelAsValue, 52
- testSuiteProfile, 52
- testSuiteValidScaleValue, 52
- testValidScaleValue0, 52
- testValidScaleValue10, 53
- testValidScaleValue15, 53
- testValidScaleValue5, 53
- testValidScaleValueN5, 53
- menuCmd
 - commands.c, 19
 - commands.h, 31
- NO_EMPTY_INDEX
 - constants.h, 37
- NOT_FOUND_STRING
 - constants.h, 37
- NOT_IN_LIST
 - constants.h, 37
- NUMBER_OF_REGIONS
 - constants.h, 37
- name
 - education, 7
 - profile, 10
- normalizeVector
 - vector.h, 78
 - vectors.c, 82
- OTHER_SUBJECTS
 - constants.h, 38
- parseDatabase
 - parser.c, 55
 - parser.h, 62
- parseDatabaseLine
 - parser.c, 55
 - parser.h, 62
- parseEduDesc
 - parser.c, 56
 - parser.h, 62
- parseEduLink
 - parser.c, 56
 - parser.h, 63
- parseEduNames
 - parser.c, 56
 - parser.h, 63
- parseEduRegion
 - parser.c, 57
 - parser.h, 63
- parseEduString
 - parser.c, 57
 - parser.h, 64
- parseInterestNames
 - parser.c, 57
 - parser.h, 64
- parseInterestValues
 - parser.c, 58
 - parser.h, 64
- parseNumOfEdu
 - parser.c, 58
 - parser.h, 65
- parseNumOfInterests
 - parser.c, 58
 - parser.h, 65
- parseReqGrade
 - parser.c, 58
 - parser.h, 65
- parseSubReq
 - parser.c, 59
 - parser.h, 66
- parser.c, 53
 - createArrayOfStrings, 54
 - findDatabaseLine, 55
 - parseDatabase, 55
 - parseDatabaseLine, 55
 - parseEduDesc, 56
 - parseEduLink, 56
 - parseEduNames, 56
 - parseEduRegion, 57
 - parseEduString, 57
 - parseInterestNames, 57
 - parseInterestValues, 58
 - parseNumOfEdu, 58
 - parseNumOfInterests, 58
 - parseReqGrade, 58
 - parseSubReq, 59
 - readReqString, 59
 - sseek, 60
 - strToReg, 60
- parser.h, 60
 - createArrayOfStrings, 61
 - findDatabaseLine, 61
 - parseDatabase, 62

- parseDatabaseLine, 62
- parseEduDesc, 62
- parseEduLink, 63
- parseEduNames, 63
- parseEduRegion, 63
- parseEduString, 64
- parseInterestNames, 64
- parseInterestValues, 64
- parseNumOfEdu, 65
- parseNumOfInterests, 65
- parseReqGrade, 65
- parseSubReq, 66
- readReqString, 66
- sseek, 66
- strToReg, 66
- printEducation
 - commands.c, 19
 - commands.h, 31
- printProfile
 - profile.c, 68
 - profile.h, 69
- printVector
 - vector.h, 78
 - vectors.c, 82
- profile, 8
 - adjustment_vector, 9
 - average, 9
 - interests, 9
 - last_recommended, 9
 - location, 9
 - name, 10
 - qualifications, 10
 - recommended_educations, 10
 - saved_educations, 10
- profile.c, 67
 - createProfile, 67
 - freeProfile, 68
 - printProfile, 68
- profile.h, 68
 - createProfile, 69
 - freeProfile, 69
 - printProfile, 69
- qualification, 10
 - amount_of_subjects, 11
 - subjects, 11
- qualifications
 - profile, 10
- readReqString
 - parser.c, 59
 - parser.h, 66
- recommendCmd
 - commands.c, 20
 - commands.h, 31
- recommended_educations
 - profile, 10
- region
 - education, 7
 - location, 8
 - region.h, 70
- region.h, 70
 - region, 70
- region_importance
 - location, 8
- required_grade
 - education, 7
- required_qualifications
 - education, 7
- STRING_MAX_LENGTH
 - constants.h, 38
- saveCmd
 - commands.c, 20
 - commands.h, 31
- saveProfile
 - commands.c, 20
 - commands.h, 32
- saved_educations
 - profile, 10
- scaleVector
 - vector.h, 79
 - vectors.c, 83
- scanCommand
 - main.c, 47
- searchCmd
 - commands.c, 21
 - commands.h, 32
- setImportantSubjects
 - commands.c, 21
 - commands.h, 32
- setOtherSubjects
 - commands.c, 21
 - commands.h, 33
- setProfileInterests
 - commands.c, 22
 - commands.h, 33
- setProfileLocation
 - commands.c, 22
 - commands.h, 33
- setProfileQualifications
 - commands.c, 22
 - commands.h, 34
- setSubjects
 - commands.c, 22
 - commands.h, 34
- size
 - vector, 12
- sseek
 - parser.c, 60
 - parser.h, 66
- strToReg
 - parser.c, 60
 - parser.h, 66
- stringToClass
 - subjects.c, 72
 - subjects.h, 75
- subject, 11

- level, 11
- subjects
 - qualification, 11
- subjects.c, 71
 - charToLevel, 71
 - createQualifications, 71
 - freeQualifications, 71
 - levelToChar, 71
 - stringToClass, 72
- subjects.h, 72
 - charToLevel, 74
 - createQualifications, 74
 - freeQualifications, 74
 - level, 74
 - levelToChar, 75
 - stringToClass, 75
- subtractVector
 - vector.h, 79
 - vectors.c, 83
- surveyCmd
 - commands.c, 24
 - commands.h, 34
- TABS
 - constants.h, 38
- TOTAL_SUBJECTS
 - constants.h, 38
- testConvertScale0
 - main.c, 48
- testConvertScale10
 - main.c, 48
- testConvertScale5
 - main.c, 48
- testIsQualifiedHH
 - main.c, 48
- testIsQualifiedHL
 - main.c, 48
- testIsQualifiedHS
 - main.c, 48
- testIsQualifiedLH
 - main.c, 49
- testIsQualifiedLL
 - main.c, 49
- testIsQualifiedLS
 - main.c, 49
- testIsQualifiedSH
 - main.c, 49
- testIsQualifiedSL
 - main.c, 49
- testIsQualifiedSS
 - main.c, 49
- testLevelAsValueA
 - main.c, 49
- testLevelAsValueB
 - main.c, 50
- testLevelAsValueC
 - main.c, 50
- testLevelAsValueZ
 - main.c, 50
- testParseNumOfEdu
 - main.c, 50
- testParseNumOfInterests
 - main.c, 50
- testSameAdjustment
 - main.c, 50
- testSameAverageGrade
 - main.c, 50
- testSameInterests
 - main.c, 51
- testSameLastRecommended
 - main.c, 51
- testSameName
 - main.c, 51
- testSameRecommendedList
 - main.c, 51
- testSameRegion
 - main.c, 51
- testSameRegionImportance
 - main.c, 51
- testSameSavedList
 - main.c, 51
- testSuiteConvertScale
 - main.c, 52
- testSuiteDatabase
 - main.c, 52
- testSuiteIsQualified
 - main.c, 52
- testSuiteLevelAsValue
 - main.c, 52
- testSuiteProfile
 - main.c, 52
- testSuiteValidScaleValue
 - main.c, 52
- testValidScaleValue0
 - main.c, 52
- testValidScaleValue10
 - main.c, 53
- testValidScaleValue15
 - main.c, 53
- testValidScaleValue5
 - main.c, 53
- testValidScaleValueN5
 - main.c, 53
- USELESS_SUBJECTS
 - constants.h, 38
- VERSION
 - constants.h, 38
- validScaleValue
 - commands.c, 24
 - commands.h, 34
- vector, 12
 - array, 12
 - size, 12
- vector.h, 75
 - addVector, 76
 - copyVector, 76

- createVector, [77](#)
- dotProduct, [77](#)
- freeVector, [77](#)
- freeVectorM, [78](#)
- lengthOfVector, [78](#)
- normalizeVector, [78](#)
- printVector, [78](#)
- scaleVector, [79](#)
- subtractVector, [79](#)
- vectors.c, [79](#)
 - addVector, [80](#)
 - copyVector, [80](#)
 - createVector, [81](#)
 - dotProduct, [81](#)
 - freeVector, [81](#)
 - freeVectorM, [82](#)
 - lengthOfVector, [82](#)
 - normalizeVector, [82](#)
 - printVector, [82](#)
 - scaleVector, [83](#)
 - subtractVector, [83](#)